ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

«Ανάπτυξη εφαρμογών για φορητές συσκευές με χρήση της J2ME γλώσσας προγραμματισμού και δημιουργία εκπαιδευτικού περιβάλλοντος για eLearning μέσω της πλατφόρμας Moodle»

Των φοιτητριών
Μανώλα Ευαγγελία, Πετρίδου Άννα
Αρ.Μητρώου: 03/2278, 03/2230

Επιβλέπων καθηγητής
Σιάκα Κέρστιν

Θεσσαλονίκη 2010

Πρόλογος

Η τεχνολογία φορητών συσκευών και ειδικότερα της κινητής τηλεφωνίας έχει εισχωρήσει πλέον στην καθημερινότητα όλων μας. Η συνεχής εξέλιξη και ανάπτυξη αυτού του τομέα ενδιαφέρει όλο και μεγαλύτερο ποσοστό νέων ανθρώπων. Οι ενδιαφερόμενοι για την ανάπτυξη εφαρμογών σε φορητές συσκευές δεν έχουν την δυνατότητα να βρούν εύκολα υλικό λόγω της έλλειψης βιβλιογραφικού περιεχομένου στα ελληνικά και λόγω της απουσίας του θέματος αυτού ως αντικείμενο διδασκαλίας.

Οι παραπάνω λόγοι και κυρίως το προσωπικό μας ενδιαφέρον όσων αφορά την ανάπτυξη εφαρμογών σε φορητές συσκευές με τη χρήση της Java 2 Micro Edition (J2ME) μας οδήγησαν στην επιλογή του συγκεκριμένου θέματος για την εργασία αυτή, η οποία πραγματοποιήθηκε μετά από εκτενή αναζήτηση σε ξένη βιβλιογραφία καθώς και σε ηλεκτρονικές πηγές.

Έως τώρα, οι περισσότεροι άνθρωποι είναι εξοικειωμένοι με την πλατφόρμα Java 2 και το πώς η Sun έχει ομαδοποιήσει τις Java τεχνολογίες σε τρεις εκδόσεις (Standard, Enterprise και Micro Edition) για διαφορετικούς σκοπούς η κάθε μία.

Η Sun παρουσίασε την J2ME, τον Ιούνιο του 1999 [7], για να χρησιμοποιηθεί για προγράμματα ηλεκτρονικών καταναλωτικών προϊόντων με λιγότερες υπολογιστικές δυνατότητες από έναν προσωπικό υπολογιστή, αλλά περισσότερες δυνατότητες από μία smart card.

Σήμερα, η J2ME παρέχει μια Java πλατφόρμα για μια απίστευτα μεγάλη ποικιλία συσκευών, δίνοντας τη δυνατότητα επιπλέον λειτουργικότητας στη συσκευή, τόση όση κι η ίδια έχει τη δυνατότητα να υποστηρίξει, σε συνδυασμό με τις συνήθεις παροχές της Java, όπως π.χ. η δυνατότητα μεταφοράς μεταξύ πλατφορμών, η ασφάλεια, και ο αντικειμενοστραφής χαρακτήρας. Για το λόγο αυτό, η Java είναι η πλέον προτιμώμενη γλώσσα για τις σύγχρονες υπολογιστικές συσκευές.

Περίληψη

Με την πτυχιακή αυτή εργασία δημιουργείται ένα εγχειρίδιο που εξετάζει λεπτομερώς την Java 2 Micro Edition (J2ME) και ένα εκπαιδευτικό περιβάλλον για eLearning μέσω της πλατφόρμας Moodle (sites.it.teithe.gr/netis/moodle). Ο τελικός στόχος είναι να χρησιμοποιηθεί ως βοήθημα διδασκαλίας για δια βίου εκπαίδευση από έναν διδάσκοντα. Επιπλέον, θα μπορούσε να αποτελέσει μια κύρια πηγή εκμάθησης της J2ME για έναν μέσο προγραμματιστή, ο οποίος έχει ένα υπόβαθρο σε προγραμματισμό Java και σε έννοιες του αντικειμενοστραφούς σχεδιασμού και ανάπτυξης.

Αρχικά εξετάζεται το ιστορικό και η αρχιτεκτονική της J2ME. Θα ακολουθήσει η διερεύνηση των J2ME διαμορφώσεων (configurations) και προφίλ (profiles) και θα γίνει εισαγωγή σε θέματα όπως η K Virtual Machine (KVM) και το Application Programming Interface (API) διεπαφής χρήστη.

Στη συνέχεια περιγράφεται ο διαχωρισμός του API αυτού σε χαμηλού και υψηλού επιπέδου, το οποίο χρησιμοποιείται σε συνδυασμό με το Connected Limited Device Configuration (CLDC) API, και το Mobile Information Device Profile (MIDP), το οποίο χρησιμοποιεί CLDC.

Έπειτα θα ξεκινήσει η διαδικασία δημιουργίας του περιβάλλοντος ανάπτυξης για την κατασκευή J2ME εφαρμογών. Αναλυτικότερα θα δείξουμε βήμα βήμα την εγκατάσταση και τον τρόπο λειτουργίας του Wireless Toolkit, που σε συνδυασμό με το JCreator αποτελούν τα βασικά εργαλεία για την ανάπτυξη, εκτέλεση και αναπαράσταση J2ME εφαρμογών.

Τέλος, περιλαμβάνονται παραδείγματα διαφορετικού βαθμού δυσκολίας, ξεκινώντας από ένα απλό παράδειγμα και καταλήγοντας σε μια πιο σύνθετη και ολοκληρωμένη εφαρμογή. Έτσι ο αναγνώστης θα κατανοήσει και θα εμβαθύνει στην εκμάθηση εννοιών της J2ME καθώς και στην υλοποίηση εφαρμογών. Τα αποτελέσματα των εφαρμογών αυτών απεικονίζονται μέσω σχημάτων και εικόνων.

Οι απαιτήσεις συστήματος σε λογισμικό και υλικό για τον χρήστη του εγχειριδίου αυτού περιγράφονται παρακάτω:

Απαιτήσεις Συστήματος – Λογισμικό

- Microsoft Windows XP ή Linux-x86.
- JCreator (για ανάπτυξη κώδικα).
- Java™ 2 SDK, Standard Edition (J2SE SDK), έκδοση 1.4.2 - εάν πρόκειται να κάνετε πραγματική ανάπτυξη, ή JavaTM 2, Standard Edition Runtime Environment (JRE), έκδοση 1.4.2 - μόνο αν σκοπεύετε να εκτελέσετε τα παραδείγματα για επίδειξη. Για να κατεβάσετε το SDK ή JRE, ακολουθείστε τον παρακάτω σύνδεσμο: http://java.sun.com/j2se/downloads.html [8].
- Το Wireless Toolkit (για μεταγλώττιση και προσομοίωση της εκτέλεσης στη συσκευή), περιέχει την CLDC, KVM και MIDP.
- Την Connected Limited Device Configuration (CLDC) αναφορά υλοποίησης. Την K virtual machine (KVM), η οποία περιλαμβάνεται με την CLDC αναφορά υλοποίησης. Το Mobile Information Device Profile (MIDP). (Δεν απαιτείται από σας να τα κατεβάσετε ξεχωριστά αν έχετε εγκαταστήσει το Wireless Toolkit).

Απαιτήσεις Συστήματος - Υλικό

Οι ελάχιστες απαιτήσεις υλικού είναι:

- 50 MB hard disc
- 128 MB RAM
- 800 MHz Pentium III CPU [8]

Abstract

This thesis creates a manual which examines in detail the Java 2 Micro Edition (J2ME) and a learning environment for eLearning in the Moodle platform. The final goal is this thesis to be used as a teaching tool for lifelong learning by an instructor. Moreover, a developer who has a background in Java programming and in concepts of object-oriented design and development, could use it as a source for J2ME learning.

To begin with, the history and architecture of J2ME are examined. After, the investigation of the J2ME configurations and profiles follows and there will be an introduction in issues, such as the K Virtual Machine (KVM) and the Application Programming Interface (API) user interface.

Then, the separation of the API in low and high level is described, which is used in conjunction with the Connected Limited Device Configuration (CLDC) API, and the Mobile Information Device Profile (MIDP), which uses CLDC. Furthermore, the process of creating the development environment for J2ME applications' construction begins. More specifically, we will show step by step the installation and operation of the Wireless Toolkit, which along with JCreator are the basic tools for the development, execution and representation of J2ME applications.

Finally, examples of varying degrees of difficulty are included, starting with a simple example and resulting in a more complex and integrated application. Thus the reader will understand and deepen in learning concepts of J2ME and in applications' implementation. The results of these applications are displayed through figures and images.

The system requirements for a potential user in software and hardware are described below:

System Requirements – Software

- Microsoft Windows XP or Linux-x86.
- JCreator (for code development).
- Java TM 2 SDK, Standard Edition (J2SE SDK), version 1.4.2 - if you are going to have a real development or Java TM 2, Standard Edition Runtime Environment (JRE), version 1.4.2 - only if you intend to run the examples for demonstration . To download the SDK or JRE, follow the link below: http://java.sun.com/j2se/downloads.html [8].
- The Wireless Toolkit (for translation and simulation of device performance), containing the CLDC, KVM and MIDP.
- The Connected Limited Device Configuration (CLDC) reference implementation. The K virtual machine (KVM), which is included in the CLDC reference implementation. The Mobile Information Device Profile (MIDP). (It is not required to download them separately if you have installed the Wireless Toolkit).

System Requirements - Hardware
The minimum hardware requirements are:

- 50 MB hard disc
- 128 MB RAM
- 800 MHz Pentium III CPU [8]

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε ιδιαιτέρως την κ. Σιάκα Κέρστιν για την πολύτιμη βοήθεια, υποστήριξη και εμπιστοσύνη της.

Θέτοντας τη βάση για την επίτευξη του στόχου μας, ελπίζουμε να καταφέρουμε κάποια στιγμή να τον δούμε να υλοποιείται.

Περιεχόμενα

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ ΚΑΙ ΠΙΝΑΚΩΝ

ΕΙΚΟΝΕΣ

ΠΙΝΑΚΕΣ

Εισαγωγή

Τι είναι η J2ME πλατφόρμα;

Το ιδανικό μοντέλο "compile once, run everywhere"
(Sun Microsystems)

Η Sun Microsystems ορίζει την J2ME ως «ένα εξαιρετικά βελτιστοποιημένο, run-time περιβάλλον Java, στοχεύοντας σε ένα ευρύ φάσμα καταναλωτικών προϊόντων, συμπεριλαμβάνοντας κινητά τηλέφωνα, ψηφιακούς τηλεοπτικούς δέκτες, συστήματα πλοήγησης αυτοκινήτων κ.α." [7].

Η Java ME είναι η πλέον χρησιμοποιούμενη πλατφόρμα εφαρμογών για φορητές συσκευές σε όλο τον κόσμο. Παρέχει ένα εύρωστο, ευέλικτο περιβάλλον για εφαρμογές που εκτελούνται σε ένα ευρύ φάσμα από φορητές συσκευές, όπως κινητά τηλέφωνα, PDAs, τηλεοπτικούς αποκωδικοποιητές, εκτυπωτές, smart phones κλπ. Οι συσκευές αυτές έχουν κοινό το γεγονός ότι δεν έχουν την μνήμη ή/και την επεξεργαστική δύναμη ή δεν χρειάζεται να υποστηρίζουν J2SE (η standard JAVA πλατφόρμα που χρησιμοποιούν οι desktop υπολογιστές και οι servers). Η Java ME περιλαμβάνει ευέλικτες διεπαφές χρήστη, ένα ισχυρό μοντέλο ασφαλείας, ένα ευρύ φάσμα ενσωματωμένων πρωτοκόλλων δικτύου, καθώς και εκτεταμένη υποστήριξη δικτυωμένων και offline εφαρμογών που μπορείτε να κατεβάσετε δυναμικά [1] [3].

Προσφέρει μια ιδανική πλατφόρμα για την ανάπτυξη ισχυρών, διαδραστικών και φορητών εφαρμογών καθώς επίσης και ισχυρή υποστήριξη ασύρματης δικτύωσης καθώς και τη δυνατότητα προγραμματισμού εφαρμογών που είναι σε θέση να έχουν πρόσβαση σε ένα ευρύ φάσμα μορφοποιήσεων περιεχομένου ιστού. Ως εκ τούτου, η J2ME μπορεί επίσης να θεωρηθεί ως μια (εναλλακτική λύση για WAP και i-mode) ασύρματη τεχνολογία Internet [2].

Αν και δεν είναι ένα λειτουργικό σύστημα, μέχρι πρόσφατα, ήταν το μόνο μέσο ανάπτυξης εφαρμογών πολλαπλής πλατφόρμας για κινητά τηλέφωνα. Οι εφαρμογές αυτές δεν εκμεταλλεύονται μόνο το πλεονέκτημα της "φύσης" της ασύρματης δικτύωσης (δίκτυο κινητών τηλεφώνων), εκτός από τις υπάρχουσες

υπηρεσίες εκτός Διαδικτύου, αλλά βελτιώνει ριζικά τις προσφάτως διαθέσιμες υπηρεσίες από άποψη κόστους και αποτελεσματικότητας. Υπογραμμίζει, επίσης, το γεγονός ότι οι περιορισμένοι πόροι σε ένα κινητό τηλέφωνο δεν είναι εμπόδιο για τη δημιουργία ελκυστικού περιεχομένου [4].

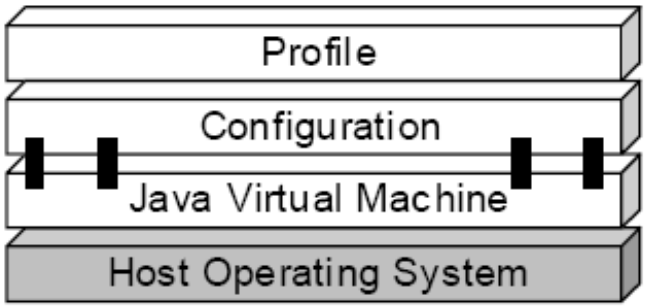
ΚΕΦΑΛΑΙΟ 1

1. Βασικές έννοιες

1.1 Αρχιτεκτονική

Προκειμένου να αντιμετωπίσει τις διαφορετικές ανάγκες ενός ευρέος φάσματος καταναλωτικών προϊόντων, η τεχνολογία J2ME καθορίζεται από ένα σύνολο προδιαγραφών σε επίπεδα. Όταν εφαρμοστεί, κάθε προδιαγραφή απευθύνεται σε γενικές ή / και ειδικές ανάγκες συσκευής. Η J2ME χρησιμοποιεί διαμορφώσεις και προφίλ για να προσαρμόσει το Java Runtime Environment (JRE).

Ως ένα πλήρες JRE, η J2ME αποτελείται από μια διαμόρφωση, η οποία προσδιορίζει την JVM που χρησιμοποιείται, καθώς και ένα προφίλ, το οποίο ορίζει την εφαρμογή με την προσθήκη ειδικών κλάσεων. Η διαμόρφωση ορίζει το βασικό περιβάλλον εκτέλεσης ως ένα σύνολο βασικών κλάσεων και μια ειδική JVM που εκτελείται σε συγκεκριμένους τύπους συσκευών.

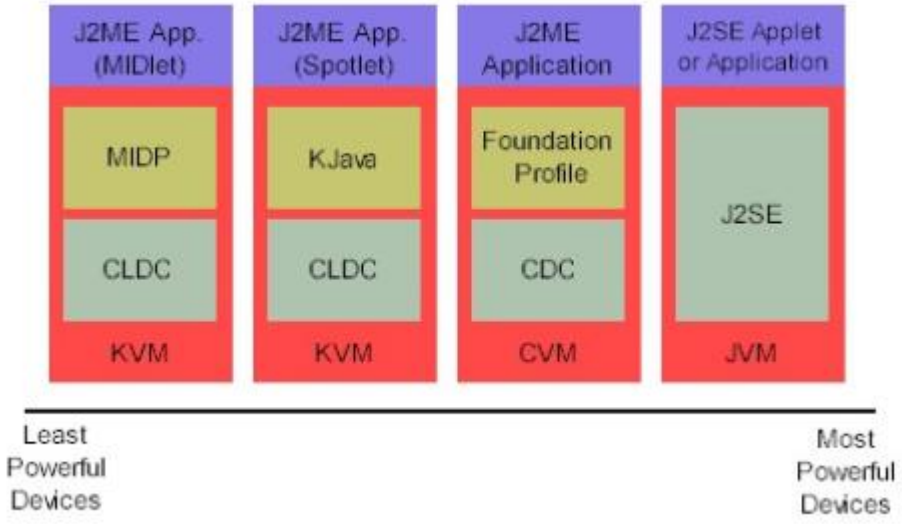


Εικόνα 1: Επίπεδα της J2ME [5]

Το προφίλ ορίζει το Java API για να αντιμετωπιστούν ειδικές ανάγκες ενός συγκεκριμένου υποσυνόλου συσκευών. Συγκεκριμένα, προσθέτει ειδικές (για τον τομέα εφαρμογής) κλάσεις στην J2ME διαμόρφωση για να καθοριστούν ορισμένες χρήσεις των συσκευών.

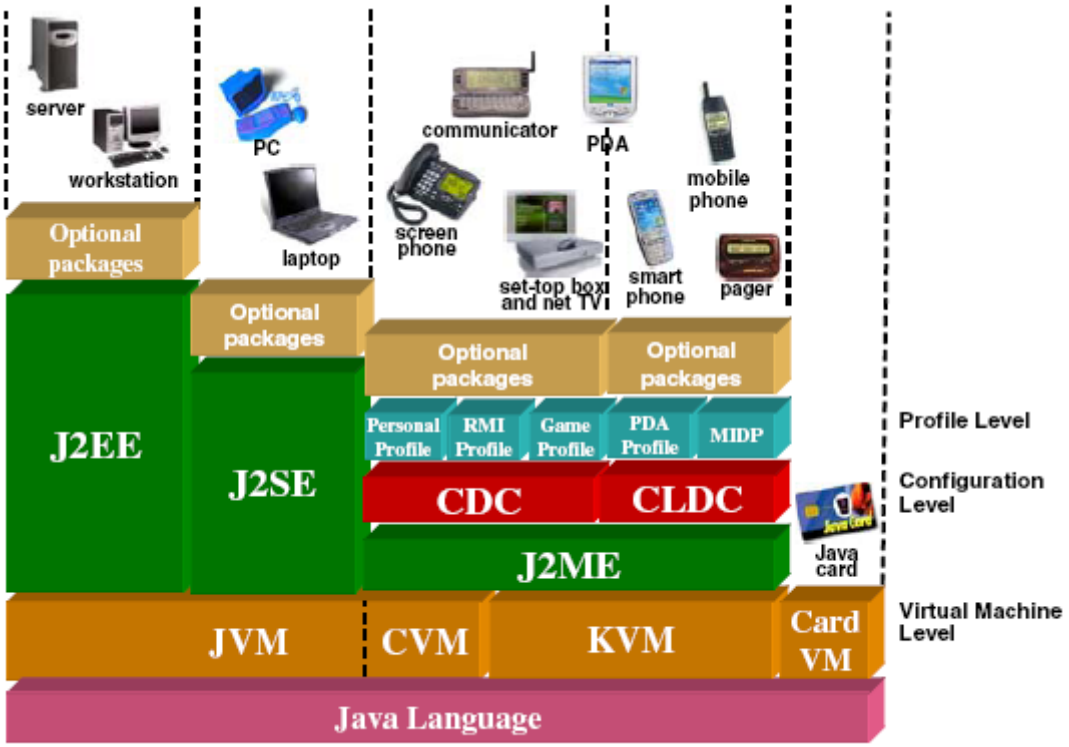
Οι προδιαγραφές διαμόρφωσης και προφίλ αναπτύσσονται, εγκαθίστανται και συντηρούνται από κατασκευαστές και χρήστες διαφόρων συσκευών, μέσω της Sun Java Community Process (JCP).

Το ακόλουθο γράφημα απεικονίζει τη σχέση μεταξύ των διαφόρων εικονικών μηχανών, διαμορφώσεων και προφίλ. Επίσης τα παραλληλίζει με το J2SE API και την Java Virtual Machine της. Αν και η J2SE εικονική μηχανή γενικά αναφέρεται ως JVM, οι J2ME εικονικές μηχανές, KVM και CVM, είναι υποσύνολα της JVM. Και οι δύο, KVM και CVM, μπορούν να θεωρηθούν ως ένα είδος της Java Virtual Machine - είναι απλά «συμπιεσμένες» εκδόσεις της J2SE JVM και είναι ειδικά για J2ME [5] [7].



Εικόνα 2: Σχέση μεταξύ εικονικών μηχανών, διαμορφώσεων και προφίλ [7]

Το ακόλουθο σχήμα απεικονίζει επιπλέον και τις συσκευές που αντιστοιχούν στις java τεχνολογίες:



Εικόνα 3: Αντιστοιχία συσκευών με java τεχνολογίες [2]

1.2 ΔΙΑΜΟΡΦΩΣΕΙΣ (configurations)

Μια διαμόρφωση ορίζει τρία βασικά στοιχεία [8]:

- ένα σύνολο από χαρακτηριστικά της Java γλώσσας προγραμματισμού
- ένα σύνολο από χαρακτηριστικά της Java Virtual Machine
- ένα σύνολο από υποστηριζόμενες Java βιβλιοθήκες και διεπαφές προγραμματισμού εφαρμογών (API).

Μια διαμόρφωση περιλαμβάνει επίσης ένα βασικό σετ από κλάσεις της Java γλώσσας προγραμματισμού. Η βασική κλάση βιβλιοθηκών που καθορίζεται για μια διαμόρφωση (και για προφίλ), πρέπει να βασίζεται σε εκείνες της Java 2 πλατφόρμας. Αυτό προωθεί την, όσο το δυνατόν, μεγαλύτερη συμβατότητα μεταξύ των εφαρμογών που είναι γραμμένες για διάφορες J2ME πλατφόρμες και εκείνων που είναι γραμμένες με J2SE, και μειώνει επίσης την καμπύλη μάθησης για τους J2ME προγραμματιστές.

Σε γενικές γραμμές, αυτό σημαίνει ότι οι προγραμματιστές μπορούν να βασίζονται στα εξής:

Όπου είναι δυνατόν, πρέπει η J2ME να επαναχρησιμοποιεί τις κλάσεις και τα πακέτα της J2SE. Αυτό σημαίνει ότι, για παράδειγμα, δεν θα ήταν αποδεκτό για μια J2ME διαμόρφωση ή προφίλ, να αποφεύγει την java.util.Date κλάση και να εισάγει μια δική της. (Θα μπορούσε να υποστηριχθεί ότι η CLDC σπάει αυτόν τον κανόνα με τη κλάσεις δικτύωσης, γιατί δεν υπάρχει χρήσιμο υποσύνολο του πακέτου java.net που να ταιριάζει στην περιορισμένη διαθέσιμη μνήμη μιας συσκευής βασισμένης στη CLDC. Το πρόβλημα αυτό έχει λυθεί με τη δημιουργία ενός νέου πακέτου που περιέχει ένα πιο ελαφρύ σετ κλάσεων δικτύωσης). Ως αποτέλεσμα, όλα όσα γνωρίζετε σχετικά με την J2SE μπορεί να μεταφερθεί στην J2ME, εφόσον γνωρίζετε τις εξαιρέσεις που ισχύουν για τη διαμόρφωση και τα προφίλ με τα οποία ασχολείστε.

Όταν μια J2SE κλάση, έχει ενσωματωθεί στη J2ME, νέες μέθοδοι και πεδία δεν μπορούν να προστεθούν σε αυτή. Ομοίως, νέες κλάσεις δεν μπορούν να προστεθούν σε ένα J2SE πακέτο. Αυτοί οι κανόνες εξασφαλίζουν ότι κώδικας γραμμένος για J2ME πλατφόρμα που χρησιμοποιεί μόνο εκείνες τις κλάσεις που μοιράζεται με την J2SE θα μεταγλωττιστεί και θα δουλέψει σε πλατφόρμα J2SE, καθιστώντας έτσι δυνατό τον διαμοιρασμό κώδικα μεταξύ αυτών των πλατφορμών [1].

Οι διαμορφώσεις αποτελούν τη βάση της πλατφόρμας J2ME. Παρέχουν τη βασική ελάχιστη λειτουργικότητα για μια συγκεκριμένη κατηγορία συσκευών. Αυτή η λειτουργικότητα είναι κοινή σε όλες τις συσκευές που ανήκουν στην ίδια κατηγορία, ανεξάρτητα από άλλους παράγοντες. Ο κύριος λόγος για τον διαχωρισμό της πλατφόρμας σε διαμορφώσεις και προφίλ είναι να εξυπηρετήσει το σκοπό αυτό. Οι συσκευές στις οποίες απευθύνεται η J2ME, ακόμα και αυτές της ίδιας κατηγορίας, είναι τόσο διαφορετικές από την άποψη των δυνατοτήτων τους που θα ήταν δύσκολο, αν όχι αδύνατο, να εφαρμοστεί το J2SE μοντέλο για αυτές [6].

Μια διαμόρφωση είναι μια προδιαγραφή που ορίζει το περιβάλλον λογισμικού για μια ποικιλία συσκευών, που προσδιορίζονται από ένα σύνολο χαρακτηριστικών στα οποία βασίζεται η προδιαγραφή και είναι συνήθως τα εξής:

- Το είδος και η ποσότητα της διαθέσιμης μνήμης
- Ο τύπος και η ταχύτητα του επεξεργαστή
- Ο τύπος της σύνδεσης δικτύου που διαθέτει η συσκευή

Μια διαμόρφωση υποτίθεται ότι αποτελεί την ελάχιστη βάση για την συσκευή που στοχεύει και δεν επιτρέπεται να ορίζει προαιρετικά χαρακτηριστικά [1].

Περιλαμβάνει την εικονική μηχανή και τις ελάχιστες απαιτούμενες βασικές βιβλιοθήκες. Επειδή οι συσκευές μπορούν γενικά να διαιρεθούν σε δύο κατηγορίες, δύο διαμορφώσεις είναι απαραίτητες. Αυτές είναι η Connected Limited Device Configuration (CLDC) και η Connected Device Configuration (CDC). Οι τομείς τους μπορεί να επικαλύπτονται, σε ορισμένες περιπτώσεις, διότι δεν υπάρχουν συγκεκριμένα όρια μεταξύ των δύο. Με ορισμένες συσκευές, θα ήταν δύσκολο να αποφασιστεί αν η CLDC ή η CDC είναι η πιο κατάλληλη διαμόρφωση. Για παράδειγμα, μερικά τηλέφωνα μπορεί να έχουν περισσότερους πόρους που θα μπορούσαν να εκτελέσουν εφαρμογές που βασίζονται σε CDC ενώ μερικά μπορεί να μην έχουν αρκετούς πόρους για να εκτελέσουν πιο «βαριές» CDC-εφαρμογές.

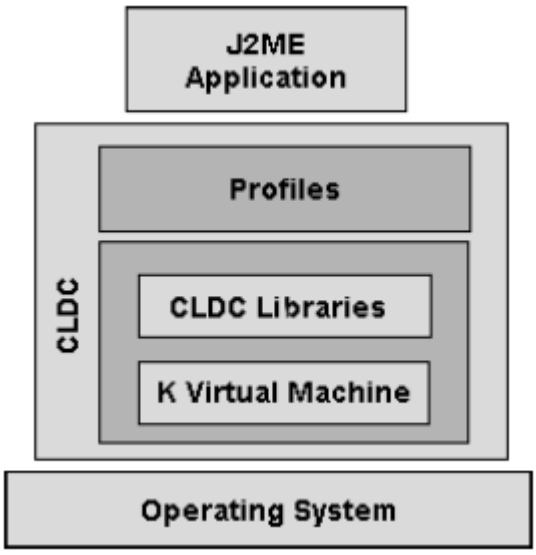
Οι διαμορφώσεις προσδιορίζουν ποια τμήματα της Java γλώσσας, ποια χαρακτηριστικά της Java Virtual Machine, και ποιες βασικές βιβλιοθήκες υποστηρίζονται. Όπως αναφέρθηκε προηγουμένως, μπορεί ορισμένες φορές να υπάρχει η δυνατότητα να χρησιμοποιηθεί είτε η μία είτε η άλλη διαμόρφωση, αλλά δεν μπορούν να χρησιμοποιηθούν και οι δύο ταυτόχρονα. Θα πρέπει να επιλεχθεί μία, πριν να αρχίσει η εκτέλεση Java εφαρμογών στη συσκευή [6].

Η J2ME σήμερα καθορίζει δύο διαμορφώσεις:

1.2.1 Connected Limited Device Configuration (CLDC)

Η CLDC έχει σχεδιαστεί για καταναλωτικά ηλεκτρονικά προϊόντα με περιορισμένη μνήμη, ισχύ (συχνά μπαταρία), σύνδεση με το δίκτυο (σε 9600 bps ή λιγότερο), και με διαφορετικούς βαθμούς πολυπλοκότητας διεπαφής χρήστη ή, ενδεχομένως, χωρίς διεπαφή χρήστη. Σε γενικές γραμμές, αυτή η διαμόρφωση είναι σχεδιασμένη για φορητές συσκευές περιορισμένης συνδεσιμότητας, όπως pagers, κινητά τηλέφωνα και PDAs [5].

Η εικονική μηχανή (KVM ή οποιαδήποτε άλλη παρόμοια υλοποίηση που τηρεί τις προδιαγραφές) είναι μικρή, και μερικά από τα χαρακτηριστικά της Java γλώσσας δεν υποστηρίζονται. Αλλά η πραγματική διαφορά είναι ότι οι διαθέσιμες βιβλιοθήκες είναι πολύ λίγες σε αριθμό. Αυτό σημαίνει ότι δεν χρειάζεται να μάθετε τα βασικά ξανά - ότι έχετε μάθει για την standard έκδοση αρκεί ως επί το πλείστον - αλλά οι βιβλιοθήκες πρέπει να χρησιμοποιούνται με μεγάλη προσοχή (ακόλουθο σχήμα).



Εικόνα 4: CLDC [6]

Μερικά από τα κύρια χαρακτηριστικά της CLDC είναι τα ακόλουθα:

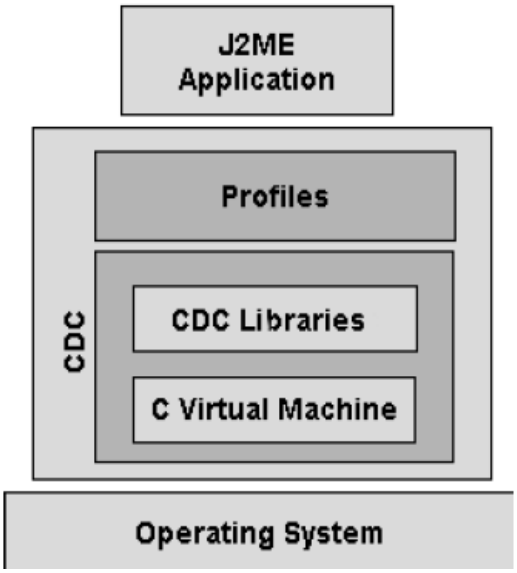
- Οι τύποι δεδομένων long και float δεν υποστηρίζονται. Όλες οι μέθοδοι των J2SE κλάσεων που κληρονομούνται που χρησιμοποιούν αυτούς τους τύπους έχουν αφαιρεθεί.
- Υπάρχουν αρκετές runtime εξαιρέσεις, αλλά ο αριθμός των runtime σφαλμάτων έχει μειωθεί σημαντικά για τις κλάσεις που συμπεριλαμβάνονται στην CLDC. Στην πραγματικότητα, μόνο τα τρία ακόλουθα σφάλματα είναι διαθέσιμα:
- Java.lang.Error
- Java.lang.OutOfMemoryError
- Java.lang.VirtualMachineError
- Άλλα σφάλματα χειρίζονται με τρόπο ανάλογο της εφαρμογής.
- Για να γίνει απλή η συλλογή από τον garbage collector, δεν παρέχεται υποστήριξη για τερματισμό. Δεν υπάρχει μέθοδος finalize στην κλάση java.lang.Object
- Η Java Native Interface (JNI) δεν υποστηρίζεται στην CLDC.
- Μπορείτε να χρησιμοποιήσετε threads, αλλά όχι thread groups ή daemon threads.
- Στην standard έκδοση, μπορείτε να μαρκάρετε αντικείμενα για πιθανή συλλογή τους από τον garbage collector. Αυτό δεν μπορεί να γίνει με την CLDC. Με άλλα λόγια, δεν υπάρχει υποστήριξη για αδύναμες αναφορές.
- Η προεπαλήθευση των κλάσεων για να ελεγχθεί αν ο κώδικας είναι καλογραμμένος πρέπει να γίνετε στο σύστημα στο οποίο η εφαρμογή αναπτύσσεται. Αυτό γίνεται από ένα εργαλείο που ονομάζεται preverifier. Πρέπει να κάνετε την προεπαλήθευση αμέσως αφού μεταγλωττίσετε τον κώδικα.
- Ένα διαφορετικό μοντέλο ασφαλείας χρησιμοποιείται στην CLDC, το οποίο είναι κάπως παρόμοιο με εκείνο που χρησιμοποιείται σε προγράμματα περιήγησης (browsers) για τη λήψη applets. Ο λόγος είναι ότι το μοντέλο που χρησιμοποιείται στην standard έκδοση είναι πάρα πολύ βαρύ για τις μικρές συσκευές, και οι ανάγκες ασφαλείας των συσκευών αυτών είναι παρόμοιες με εκείνες των browsers.

- Υπάρχουν μόνο τέσσερα διαθέσιμα πακέτα στην CLDC. Τα περισσότερα από τα J2SE πακέτα όπως τα java.lang.awt, java.lang.beans, και άλλα έχουν απορριφθεί. Η CLDC περιέχει μόνο τα ακόλουθα πακέτα:
- java.io: Περιέχει τις κλάσεις που απαιτούνται για είσοδο και έξοδο δεδομένων χρησιμοποιώντας streams (ρεύματα δεδομένων).
- java.lang: Περιέχει τις κλάσεις που είναι βασικές στην java γλώσσα όπως οι wrapper κλάσεις για τύπους δεδομένων.
- java.util: Περιέχει κλάσεις όπως τις Calender, Date, Vector, και Random.
- javax.microedition.io: Περιέχει τις κλάσεις για το χειρισμό όλων των τύπων των συνδέσεων χρησιμοποιώντας το ίδιο πλαίσιο (framework) [6].

1.2.2 Connected Device Configuration (CDC)

Οι συσκευές στις οποίες στοχεύει η CDC σε σχέση με την CLDC, έχουν περισσότερη μνήμη (2MB +) και πιο ικανούς επεξεργαστές (32-bit επεξεργαστές) και συνεπώς, η διαμόρφωση αυτή έχει την πολυτέλεια να παρέχει πολύ περισσότερα χαρακτηριστικά της standard έκδοσης της java, τόσο σε υποστήριξη της ίδιας της γλώσσας όσο και της εικονικής μηχανής.

Επίσης σημαντικό είναι πως ο αριθμός των APIs που περιλαμβάνονται στη CDC είναι σημαντικά μεγαλύτερος από ό, τι στη CLDC. Αυτό σημαίνει ότι μπορείτε να έχετε πολύ περισσότερη λειτουργικότητα στην εφαρμογή σας, αν χρησιμοποιείτε τη CDC. Και αυτό φυσικά εφόσον η συσκευή το επιτρέπει (ακόλουθο σχήμα). Η CDC θα μπορούσε να βρεθεί σε ακριβά PDAs και σε «έξυπνα τηλέφωνα» (smart phones), τηλεοπτικούς αποκωδικοποιητές, συστήματα πλοήγησης κ.α. [1] [6].



Εικόνα 5: CDC [6]

Η CDC δίνει καλύτερη υποστήριξη δικτύωσης και ένα πιο ευέλικτο μηχανισμό ασφαλείας. Η CDC πρέπει να χρησιμοποιείται μαζί με ένα προφίλ που ονομάζεται Foundation Profile. Από μόνο του αυτό δεν μπορεί να είναι χρήσιμο. Μερικά από τα σημαντικά χαρακτηριστικά αυτής της διαμόρφωσης είναι τα εξής:

Πλήρης υποστήριξη της Java γλώσσας και της εικονικής μηχανής της, σύμφωνα με τις προδιαγραφές τους.

Οι διεπαφές μεταξύ τμημάτων του περιβάλλοντος εκτέλεσης, όπως ο garbage collector,o interpreter, και ούτω καθεξής, ορίζονται με σαφήνεια, ώστε να είναι εύκολο να προστεθούν νέα χαρακτηριστικά στην εικονική μηχανή.

Η CDC περιλαμβάνει τα ακόλουθα πακέτα, τα οποία είναι σχεδόν ίδια με αυτά με τα ίδια ονόματα στη standard έκδοση:

Πίνακας 1: CDC packages [8]

CDC Package Name	Description

Java.io	Standard IO classes and interfaces
Java.lang	VM classes
Java.lang.ref	Reference classes
Java.lang.reflect	Reflection classes and interfaces
Java.math	Math package
Java.net	Networking classes and interfaces
Java.security	Security classes and interfaces
Java.security.cert	Security certificate classes
Java.text	Text package
Java.util	Standard utility classes
Java.util.jar	Java Archive (JAR) utility classes
Java.util.zip	ZIP utility classes
Javax.microedition.io	CDC generic connection framework classes and interfaces

1.3 KVM – CVM

Η Sun προσφέρει υλοποιήσεις και των δύο διαμορφώσεων, κάθε μία από τις οποίες περιλαμβάνει μια σύμφωνη εικονική μηχανή:

Η αναφορά υλοποίησης της CLDC περιλαμβάνει την Kilobyte Virtual Machine (KVM), μια μειωμένης λειτουργικότητας Virtual Machine με πολύ μικρό μέγεθος μνήμης (περίπου 40-80Kb), κατάλληλη για κινητά τηλέφωνα, PDAs κλπ και όπως και κάθε άλλη εικονική μηχανή Java μας επιτρέπει να κατεβάσουμε και να εκτελέσουμε εφαρμογές. Η KVM μπορεί να τρέξει σε οποιοδήποτε σύστημα που έχει έναν 16-bit/32-bit επεξεργαστή και 160-512Kb συνολική μνήμη ενώ δεν παρέχει υποστήριξη για ορισμένα χαρακτηριστικά, όπως για τους long και float τύπους δεδομένων. Ο σχεδιασμός της KVM βασίζεται σε ορισμένα σημαντικά ζητήματα, συμπεριλαμβανομένων του μικρού της μεγέθους, για τη διατήρηση όσο το δυνατόν περισσότερου χώρου μνήμης της συσκευής, και τη δυνατότητά της να τρέξει σε επεξεργαστές χαμηλής ισχύος, να παρέχει φορητότητα κ.α. Υπάρχουν και άλλες VMs που θα μπορούσαν να χρησιμοποιηθούν αντί αυτής, όπως η J9 VM από την IBM. Εφαρμογές γραμμένες για την JVM ή ακόμη και για την CVM κατά πάσα πιθανότητα δεν μπορούν να τρέξουν σε KVM χωρίς κάποιες αλλαγές. Αλλά το αντίστροφο δεν ισχύει - εφαρμογές γραμμένες για KVM μπορούν εύκολα να τρέξουν σε CVM ή JVM [1] [6].

Η αναφορά υλοποίησης της CDC περιλαμβάνει την CVM και θέτει σε εφαρμογή το πλήρες φάσμα των χαρακτηριστικών της J2SE VM όπως απαιτείται από τις CDC προδιαγραφές. Η CVM παρέχει πρόσθετη λειτουργικότητα και μπορεί να υποστηρίξει ότι δεν μπορεί η KVM. Οι δυνατότητές της είναι πολύ κοντά με εκείνες της JVM. Μπορεί να αναρωτηθείτε γιατί είναι απαραίτητη τελικά, αν είναι σχεδόν τόσο ισχυρή όσο η JVM. Ίσως το πιο σημαντικό, είναι ότι οι εν λόγω συσκευές προορίζονται (τουλάχιστον έως τώρα) για πολύ συγκεκριμένες εργασίες. Υποτίθεται ότι δεν αποτελούν υπολογιστικές μηχανές ικανές να κάνουν κάθε είδους εργασία, όπως τα PCs.

Πολλά χαρακτηριστικά που δεν υποστηρίζονται απ' την KVM υποστηρίζονται απ' την CVM. Αυτό είναι αναμενόμενο, διότι είναι μια πλήρως εξοπλισμένη εικονική μηχανή και έχει τα εξής πλεονεκτήματα σε σχέση με την KVM:

Παροχή διεπαφών και υποστήριξης για υπηρεσίες λειτουργικού συστήματος πραγματικού χρόνου (RTOS) (Αν η CVM χρησιμοποιείται με ένα real time λειτουργικό σύστημα, θα ξέρει πώς να συνεργαστεί χρησιμοποιώντας τις real-time δυνατότητες της).

Προηγμένο σύστημα μνήμης (Ο τρόπος χρήσης της μνήμης στην περίπτωση της CVM είναι πιο αποτελεσματικός. Αυτό επειδή καθίσταται πιο ακριβής, μειώνοντας το χρόνο παύσης του garbage collector, διαχωρίζοντας πλήρως την VM από το σύστημα μνήμης, και ούτω καθεξής).

Καθορισμός Java νημάτων κατευθείαν σε εγγενή (native) νήματα (Αν χρησιμοποιείτε CVM, μπορείτε άμεσα να καθορίσετε Java νήματα σε native νήματα και να τρέξετε Java κλάσεις από την read-only μνήμη).

Φορητότητα (Είναι πιο εύκολο να εκχωρήσετε τη CVM σε νεότερες πλατφόρμες, διότι, με τη CVM, μπορείτε να χρησιμοποιήσετε περισσότερες από μια επιλογές εκχώρησης για τις διαδικασίες που κανονικά κάνουν την εκχώρηση δύσκολη).

Γρήγορο συγχρονισμό (Ο συγχρονισμός μπορεί να γίνει με ένα μικρό αριθμό εντολών μηχανής, αυξάνοντας έτσι την ταχύτητα συγχρονισμού).

Εκτέλεση Java κλάσεων από μνήμη μόνο για ανάγνωση (ROM) (Εκτός από τις δυναμικά φορτωμένες κλάσεις, η CVM μπορεί να χρησιμοποιηθεί με τις λεγόμενες ROMable κλάσεις. Ως αποτέλεσμα, η εικονική μηχανή κάνει λιγότερο χρόνο να ξεκινήσει, ο κατακερματισμός είναι μειωμένος, και η κοινή χρήση

δεδομένων αυξάνεται. Αυτό επίσης σημαίνει ότι μπορείτε να εκτελέσετε κώδικες byte από τη ROM).

Υποστήριξη για όλα τα Java 2, v1.3 VM χαρακτηριστικά και βιβλιοθήκες: ασφάλεια, Java Native Interface (JNI), Java Virtual Machine Debugging Interface (JVMDI), αδύναμες αναφορές, serialization, RMI, και ούτω καθεξής.

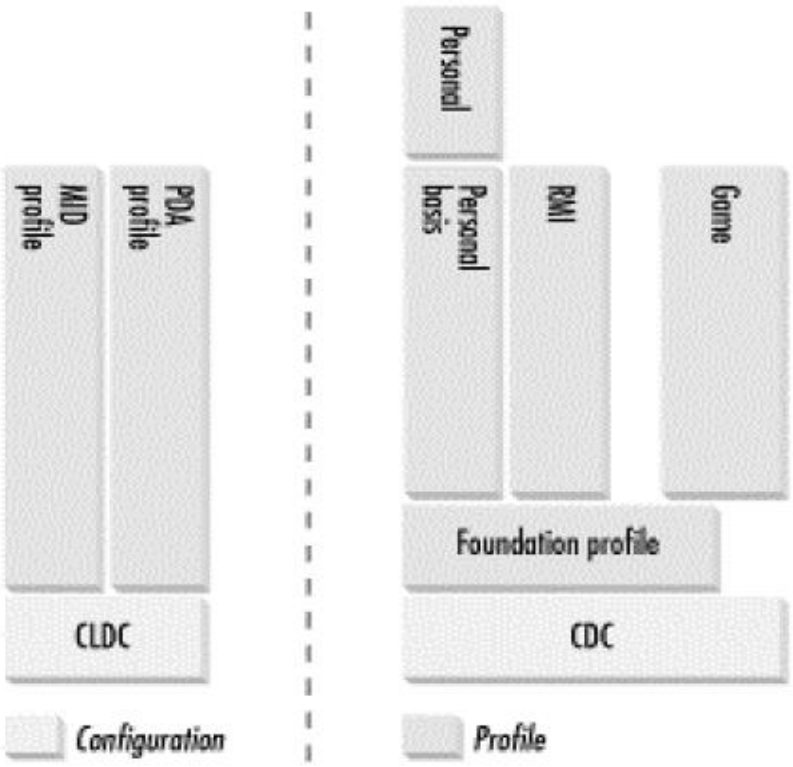
Είναι πιο εύκολο να προσθέσετε λειτουργικότητα σε διεπαφές στην περίπτωση της CVM [6] [8].

1.4 ΠΡΟΦΙΛ

Ένα προφίλ συμπληρώνει μια διαμόρφωση, προσθέτοντας επιπλέον κλάσεις που παρέχουν χαρακτηριστικά κατάλληλα για μια συγκεκριμένη ομάδα συσκευών ή ένα συγκεκριμένο τμήμα της αγοράς. Για παράδειγμα, μια διαμόρφωση παρέχει ένα γενικευμένο Java API για συστήματα περιορισμένης ισχύος, απεικόνισης και ικανότητας επεξεργασίας, ενώ ένα προφίλ παρέχει ένα συμπληρωματικό Java API ειδικό για κινητά τηλέφωνα, το οποίο αποτελεί μέλος μιας μεγαλύτερης ομάδας συσκευών. Είναι σημαντικό να σημειωθεί ότι ένα προφίλ πρέπει να συμπληρώνει μια διαμόρφωση. Ένα προφίλ δεν θα λειτουργήσει χωρίς τη διαμόρφωση και την εικονική μηχανή που παρέχει τις βασικές κλάσεις / API και το περιβάλλον εκτέλεσης.

Στις περισσότερες περιπτώσεις, ένα προφίλ θεωρείται ότι παρέχει την διεπαφή χρήστη, μεθόδους εισαγωγής και μηχανισμούς για ένα συγκεκριμένο τύπο συσκευών. Ωστόσο, ορισμένα J2ME προφίλ παρέχουν ένα πολύ συγκεκριμένο τύπο λειτουργικότητας ή εφαρμογές κατάλληλες για ένα ευρύ φάσμα συσκευών. Η Remote Method Invocation (RMI) είναι ένα παράδειγμα της λειτουργικότητας που παρέχεται από αυτό το είδος προφίλ [1] [5].

Και οι δύο J2ME διαμορφώσεις έχουν ένα ή περισσότερα προφίλ με τα οποία συνδέονται, ορισμένα εκ των οποίων μπορούν και τα ίδια να βασίζονται σε άλλα προφίλ. Το σχήμα που ακολουθεί παρακάτω, δείχνει τα προφίλ που ορίζονται σήμερα ή βρίσκονται στην διαδικασία ορισμού και οι διαμορφώσεις από τις οποίες εξαρτώνται.



Εικόνα 6: Διαμορφώσεις και Προφίλ με τα οποία συνδέονται [1]

Οι διαδικασίες αυτές περιγράφονται παρακάτω [1] :

KJava

Το KJava είναι προφίλ της Sun και περιέχει το KJava API. Βρίσκεται στην κορυφή της CLDC διαμόρφωσης. Η εικονική μηχανή KJava, η KVM, δέχεται τους ίδιους κώδικες byte και το format αρχείων κλάσεων, όπως και η κλασική J2SE εικονική μηχανή.

Το KJava περιέχει ένα API της Sun που εκτελείται για Palm OS, το οποίο έχει πολλά κοινά με το J2SE Abstract Windowing Toolkit (AWT). Ωστόσο, επειδή δεν είναι ένα στάνταρ J2ME πακέτο, το κύριο πακέτο του είναι το com.sun.kjava.

Mobile Information Device Profile (MIDP)

Αυτό το προφίλ προσθέτει δικτύωση, συστατικά διεπαφής χρήστη και τοπική αποθήκευση στην CLDC. Απευθύνεται κυρίως σε κινητά τηλέφωνα περιορισμένης απεικόνισης και δυνατότητας αποθήκευσης, και συνεπώς, παρέχει μια σχετικά απλή διεπαφή χρήστη και βασική δικτύωση που βασίζεται στο HTTP 1.1. Το MIDP

είναι το πιο γνωστό από τα J2ME προφίλ διότι αποτελεί τη βάση για Wireless java και αποτελεί σήμερα το μοναδικό διαθέσιμο προφίλ για συσκευές βασισμένες σε PalmOS.

PDA Profile (PDAP)

Το PDA profile είναι παρόμοιο με το MIDP, αλλά έχει ως στόχο τα PDAs που έχουν καλύτερες οθόνες και περισσότερη μνήμη από τα κινητά τηλέφωνα. Το PDA προφίλ, θα προσφέρει μια πιο εξειδικευμένη βιβλιοθήκη διεπαφής χρήστη και ένα API βασισμένο σε Java για πρόσβαση σε χρήσιμα χαρακτηριστικά του λειτουργικού συστήματος (host).

Foundation Profile

Το Foundation profile επεκτείνει την CDC ώστε να συμπεριλάβει το σύνολο σχεδόν των βασικών βιβλιοθηκών της Java 2 v.1.3. Όπως υποδηλώνει η ονομασία του, προορίζεται να χρησιμοποιηθεί ως βάση για τα περισσότερα από τα άλλα CDC προφίλ.

Personal Basis and Personal Profiles

Το Personal Basis profile προσθέτει βασική λειτουργικότητα διεπαφής χρήστη στο Foundation profile. Προορίζονται να χρησιμοποιηθούν σε συσκευές που έχουν δυνατότητα μη εξειδικευμένης διεπαφής χρήστη και, ως εκ τούτου δεν επιτρέπει περισσότερα από ένα παράθυρο να είναι ενεργό σε οποιαδήποτε χρονική στιγμή. Πλατφόρμες που μπορούν να υποστηρίξουν μια πιο σύνθετη διεπαφή χρήστη θα χρησιμοποιήσουν το Personal προφίλ αντί αυτού.

RMI Profile

Το RMI profile προσθέτει τις J2SE Remote Method Invocation βιβλιοθήκες του Foundation προφίλ. Μόνο η πλευρά του client αυτού του API υποστηρίζεται.

Game Profile

Το Game profile, παρέχει μια πλατφόρμα σύνταξης λογισμικού παιχνιδιών για CDC συσκευές [1].

1.5 J2ME Προδιαγραφές

Όλες οι J2ME διαμορφώσεις και τα προφίλ έχουν αναπτυχθεί ως μέρος της Java Community Process (JCP). Η JCP συγκεντρώνει «κυρίαρχες» βιομηχανίες, με στόχο την επίτευξη συμφωνίας για μια κοινή προδιαγραφή με την οποία θα μπορούν όλοι να σχεδιάζουν τα προϊόντα τους.

Κάθε διαμόρφωση ή προφίλ ξεκίνησε ως ένα Java Specification Request (JSR), το οποίο περιγράφει το πεδίο της εργασίας που πρέπει να γίνει καθώς και μια συνοπτική περιγραφή των περιοχών που πρέπει να καλυφθούν. Μια ομάδα ειδικών, συγκαλείται για να δημιουργήσει την προδιαγραφή, η οποία στη συνέχεια υπόκειται σε εσωτερική ψηφοφορία και αναθεώρηση, πριν διατεθούν για δημόσια κριτική. Μετά τη δημόσια ανασκόπηση και μια πιθανή τελευταία αναθεώρηση, το τελικό σχέδιο παράγεται, και το JSR έχει ολοκληρωθεί.

Ένας κατάλογος από JSRs, συμπεριλαμβανομένων εκείνων που έχουν ολοκληρωθεί, μπορεί να βρεθεί στην ιστοσελίδα http://jcp.org/jsr/all/.

Πίνακας 2: Οι JSRs που καθορίζουν τις τρέχουσες J2ME διαμορφώσεις και προφίλ [1]

Number	Scope
JSR 30	J2ME™ Connected, Limited Device Configuration
JSR 37	Mobile Information Device Profile for the J2ME™ Platform
JSR 75	PDA Optional Packages for the J2ME™ Platform
JSR 36	Connected Device Configuration
JSR 46	Foundation Profile
JSR 129	Personal Basis Profile Specification

JSR 62	Personal Profile Specification
JSR 66	RMI Optional Package Specification Version 1.0
JSR 134	Java™ Game Profile

Πίνακας 3: JSRs που δεν σχετίζονται άμεσα με την κάθε διαμόρφωση ή προφίλ [1]

Number	Scope
JSR 82	Java™ APIs for Bluetooth
JSR 120	Wireless Messaging API
JSR 135	Mobile Media API

Τέλος, ακόμη και αν ορισμένα από τα τρέχοντα προφίλ, δεν έχουν ακόμη καθοριστεί πλήρως, έχει ήδη αρχίσει να καθορίζεται η επόμενη γενιά της J2ME πλατφόρμας. Αξίζει λοιπόν να δούμε τα επόμενα:

Πίνακας 4: JSRs επόμενης γενιάς [1]

Number	Scope
JSR 68	J2ME™ Platform Specification
JSR 118	Mobile Information Device Profile 2.0
JSR 139	Connected Limited Device Configuration 1.1

Σύνοψη Κεφαλαίου

Σ’ αυτό το κεφάλαιο μάθατε τί είναι η J2ME και ποια είναι τα προαπαιτούμενα της τόσο σε υλικό όσο και σε λογισμικό. Γνωρίσατε την αρχιτεκτονική της, τις δύο διαμορφώσεις της, CLDC και CDC, ,καθώς και τις εικονικές μηχανές της καθεμίας. Τέλος είδατε τα προφίλ με τα οποία συνδέονται οι διαμορφώσεις αυτές, όπως και αρκετές από τις J2ME προδιαγραφές.

Ερωτήσεις

1.Τί είναι η J2ME;

2.Τί κοινό έχουν οι συσκευές στις οποίες στοχεύει η J2ME;

3.Αναφέρετε μερικές από τις παραπάνω συσκευές.

4.Τί ορίζει μια διαμόρφωση;

5.Σε τί διαφέρουν οι CLDC και CDC ;

6.Ποιά είναι τα πλεονεκτήματα της CVM σε σχέση με την KVM;

7.Τί ορίζει ένα προφίλ;

8.Αναφέρετε τα προφίλ που έχουν οριστεί μέχρι σήμερα.

ΚΕΦΑΛΑΙΟ 2

2. Προγραμματίζοντας με J2ME

Ποιες είναι οι προϋποθέσεις για προγραμματισμό με J2ME; Από πού να ξεκινήσουμε; Απαντάμε σε αυτές τις ερωτήσεις παρακάτω. Το πρώτο πράγμα που χρειάζεται είναι η γνώση της γλώσσας Java. Πρέπει να είστε ήδη εξοικειωμένοι με τον προγραμματισμό σε Java. Οι βιβλιοθήκες μπορεί να είναι διαφορετικές ανάλογα με τη διαμόρφωση και το προφίλ που χρησιμοποιείτε και αν προχωρήσετε σε CDC, θα βρείτε πολλά από τα τρέχοντα API της J2SE.

Αν γνωρίζετε Java, μπορείτε να αρχίσετε αμέσως την εκμάθηση της J2ME. Για να ξεκινήσετε τον προγραμματισμό σε αυτή την πλατφόρμα, θα χρειαστεί να εγκαταστήσετε το Java Development Kit (JDK).

Μπορείτε να το κατεβάσετε από την διεύθυνση www.java.sun.com ή από την http://www.brothersoft.com/common-jdk-144375.html. Μπορεί να το έχετε ήδη εάν έχετε εργαστεί προηγουμένως με Java.

Το επόμενο πράγμα που χρειάζεστε είναι μια διαμόρφωση. Υπάρχουν δύο επιλογές - CLDC ή CDC. Αυτά μπορείτε επίσης να τα κατεβάσετε από την ίδια διεύθυνση. Για να εργαστείτε με CLDC, το λειτουργικό σας σύστημα πρέπει να είναι τα Windows, Solaris, ή Linux. Για CDC, χρειάζεστε είτε VxWorks ή Linux. Διαφορετικά, θα πρέπει να μεταφέρετε τη διαμόρφωση στην πλατφόρμα σας, κάτι που μπορεί να αποδειχθεί λίγο περίπλοκο. Αν θέλετε να εργαστείτε με προφίλ, πρέπει επίσης να τα κατεβάσετε.

Για την ανάπτυξη των εφαρμογών που θα ακολουθήσουν είναι απαραίτητο να έχετε κατεβάσει το Java Wireless Toolkit (έτσι δεν θα χρειαστεί να κατεβάσετε διαμορφώσεις και προφίλ, καθώς οτι απαιτείται είναι ενσωματωμένο στο JWT) και το JCreator απ’ το site της Sun [6].

2.1 Η βασική κλάση MIDlet

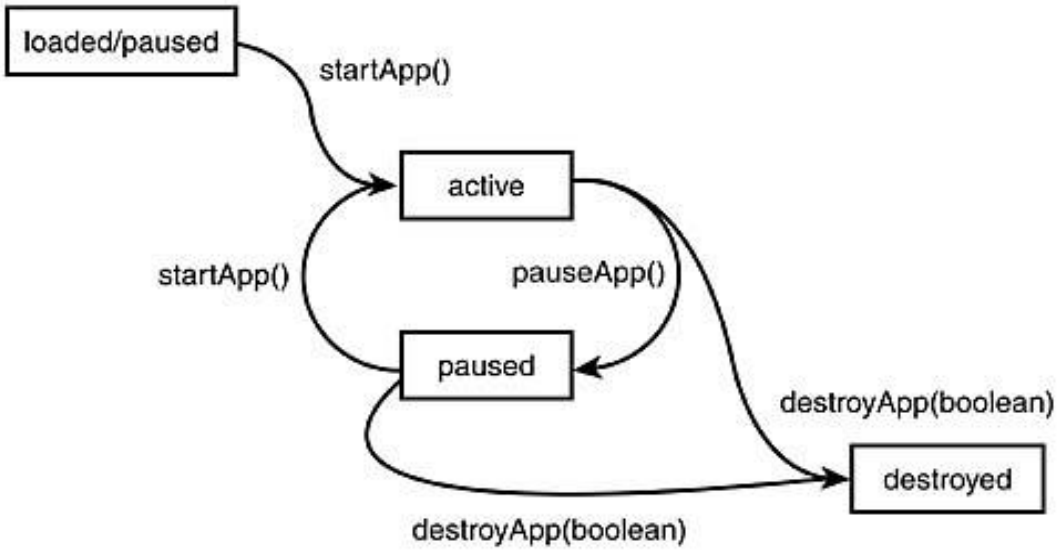
JAM

Ο κύκλος ζωής του MIDlet ελέγχεται από το Java Application Manager (JAM). Το JAM είναι λογισμικό διαχείρισης ενός MIDlet που ελέγχει την διαδικασία εγκατάστασης, εκτέλεσης και διαγραφής των MIDlets. Αυτό είναι που δημιουργεί ένα στιγμιότυπο της MIDlet κλάσης, όταν ο χρήστης επιλέγει να εκτελέσει ένα MIDlet. Καλεί διάφορες μεθόδους στο MIDlet για να εκφράσει αλλαγές από την μια κατάσταση στην άλλη. Σε αυτές τις μεθόδους περιλαμβάνονται οι ακόλουθες : startApp(), pauseApp(), destroyApp() και οι δομητές της MIDlet υποκλάσης [5].

MIDlet

Όλες οι εφαρμογές για το MID(Mobile Information Device) Profile πρέπει να προέρχονται από μια ειδική κλάση, την MIDlet. Η κλάση MIDlet διαχειρίζεται τον κύκλο ζωής της εφαρμογής. Βρίσκεται στο πακέτο javax.microedition.midlet.

Τα MIDlets μπορούν να συγκριθούν με τα J2SE applets, εκτός από το ότι η κατάστασή τους είναι πιο ανεξάρτητη από την κατάσταση απεικόνισης. Ένα MIDlet μπορεί να υπάρχει σε τέσσερις διαφορετικές καταστάσεις: loaded, active, paused, και destroyed. Το σχήμα που ακολουθεί παρουσιάζει μια επισκόπηση του κύκλου ζωής του MIDlet. Όταν ένα MIDlet φορτώνεται στη συσκευή και ο δομητής καλείται, βρίσκεται στην κατάσταση loaded. Αυτό μπορεί να συμβεί ανά πάσα στιγμή πριν ο διαχειριστής του προγράμματος ξεκινήσει την εφαρμογή καλώντας την startApp () μέθοδο. Αφού κληθεί η startApp (), το MIDlet είναι στην κατάσταση active μέχρι ο διαχειριστής του προγράμματος καλέσει την pauseApp () ή την destroyApp (). Η pauseApp () διακόπτει το MIDlet, και η desroyApp () το τερματίζει. Όλες οι κλήσεις των μεθόδων αλλαγής κατάστασης θα πρέπει να τερματίζονται γρήγορα, διότι η κατάσταση δεν μεταβάλλεται πλήρως πριν την επιστροφή της μεθόδου.



Εικόνα 7: Ο κύκλος ζωής ενός MIDlet [13]

Στην pauseApp () μέθοδο, οι εφαρμογές θα πρέπει να σταματήσουν τα animations και να απελευθερώσουν πόρους που δεν χρειάζονται ενώ η εφαρμογή είναι σε παύση. Αυτή η συμπεριφορά αποτρέπει συγκρούσεις πόρων με την εφαρμογή που τρέχει στο προσκήνιο και την άσκοπη κατανάλωση μπαταρίας. Η destroyApp () μέθοδος παρέχει μια παράμετρο, η οποία αν οριστεί ως false, επιτρέπει στο MIDlet να αρνηθεί τον τερματισμό του με το να προκαλέσει μια MIDletStateChangeException. Τα MIDlets μπορούν να ζητήσουν να επανακτήσουν εκ νέου δραστηριότητα καλώντας την resumeRequest (). Για να βρεθεί ένα MIDlet στην κατάσταση παύσης, θα πρέπει να ενημερώσει τον διαχειριστή εφαρμογής καλώντας την μέθοδο notifyPaused (). Προκειμένου να τερματιστεί, μπορεί να καλέσει την notifyDestroyed (). Σημειώστε ότι η System.exit () δεν υποστηρίζεται στο MIDP και θα προκαλέσει μια εξαίρεση αντί τον τερματισμό της εφαρμογής.

Σημείωση : Μερικές συσκευές μπορούν να τερματίσουν το MIDlet κάτω από ορισμένες συνθήκες χωρίς να καλέσουν την destroyApp(), όπως για παράδειγμα σε εισερχόμενες κλήσεις ή όταν η μπαταρία εξαντληθεί. Επομένως, μπορεί να είναι επικίνδυνο να βασιστείτε στην destroyApp() για την αποθήκευση δεδομένων που εισάγονται ή τροποποιούνται από τον χρήστη.

Display και Displayable

Τα MIDlets μπορούν να είναι εφαρμογές που είτε αλληλεπιδρούν με το χρήστη είτε όχι. Ένα MIDlet μπορεί να έχει ένα στιγμιότυπο ενός Display αντικειμένου. Αυτό το αντικείμενο χρησιμοποιείται για να αποκτηθούν πληροφορίες σχετικά με την τρέχουσα απεικόνιση και περιλαμβάνει μεθόδους για την διαχείριση των αντικειμένων που θα απεικονιστούν.Η κλάση Display και όλες οι άλλες κλάσεις διεπαφής χρήστη του MIDP βρίσκονται στο πακέτο javax.microedition.lcdui. Ένα MIDlet μπορεί να πάρει το Display στιγμιότυπό του καλώντας την getDisplay (midlet MIDlet), όπου το ίδιο το MIDlet δίνεται ως παράμετρος. Η μορφή της getDisplay είναι οι εξής:

```
public Display getDisplay(MIDlet Midlet);
```

Η κλάση Display παρέχει την μέθοδο setCurrent () που καθορίζει το τρέχον περιεχόμενο απεικόνισης του MIDlet. Οι δυο μορφές τις setCurrent () είναι οι εξής:

```
setCurrent(Displayable dispRef);
setCurrent(Alert aRef, Displayable dispRef);
```

Υπάρχει μόνο ένα Display αντικείμενο ανά MIDlet, αλλά πολλά αντικείμενα μέσα στο MIDlet μπορούν να είναι displayable. Ένα displayable αντικείμενο είναι ένα συστατικό που είναι ορατό στη συσκευή. Το MIDP περιέχει δύο υποκλάσεις της κλάσης Displayable: Screen και Canvas

```
abstract public class Displayable
public abstract class Canvas extends Displayable
public abstract class Screen extends Displayable
```

Το MIDlet μπορεί να καλέσει την isShown () μέθοδο του Displayable, προκειμένου να καθορίσει αν το περιεχόμενο εμφανίζεται στην οθόνη ή όχι.

Επίσης μπορούμε να βρούμε την απεικόνιση της συσκευής καθώς και τους τύπους των χρωμάτων που το αντικείμενο απεικόνισης υποστηρίζει.

```
public Displayable getCurrent ();

public void boolean isColor (); //επιστρέφει TRUE αν υποστηρίζει Color,
                                FALSE αν υποστηρίζει grayscale
public int numColors (); //Επιστρέφει τον αριθμό των χρωμάτων που
                         υποστηρίζει
```

2.2 J2ME, η πρώτη εφαρμογή

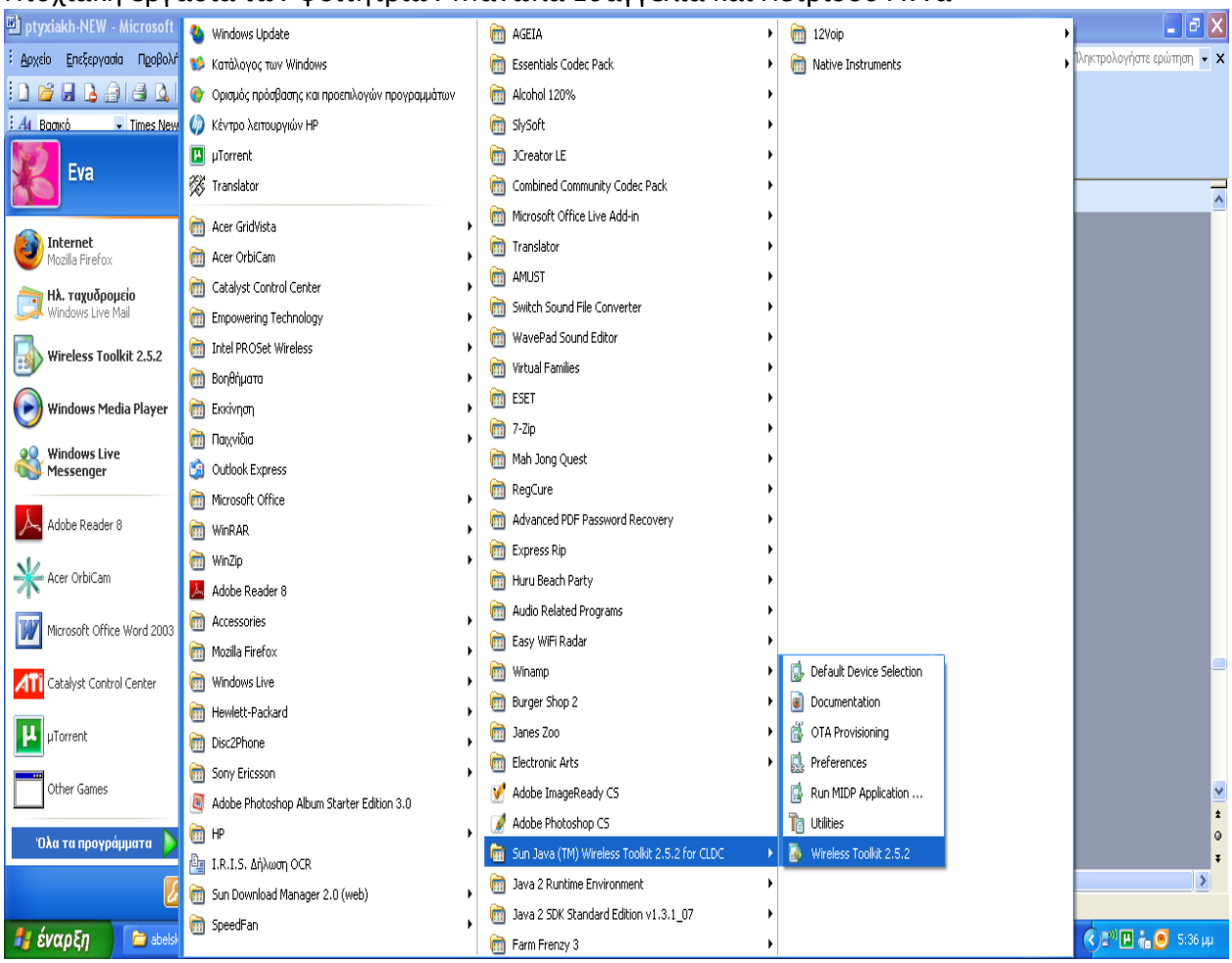
Αρχικά, θα μάθετε να χρησιμοποιείτε το Java Wireless Toolkit για την μεταγλώττιση του κώδικά σας και τη χρήση ενός εξομοιωτή για να τον δοκιμάσετε. Στη συνέχεια, θα είστε σε θέση να φορτώσετε και να δοκιμάσετε τα τελικά σας projects σε ένα κινητό τηλέφωνο. Ενώ ως επί το πλείστον, απλά ακολουθούν οδηγίες, είναι σημαντικό να προσέξετε και να σκεφτείτε κάθε βήμα, προκειμένου να κατανοήσετε τη διαδικασία ανάπτυξης σε J2ME.

Η J2ME είναι παρόμοια με τη J2SE, αλλά υπάρχουν βασικές διαφορές στη διαδικασία μεταγλώττισης και τις συμπεριλαμβανόμενες βιβλιοθήκες [11].

Ακολουθεί ένα απλό παράδειγμα, βήμα-βήμα.

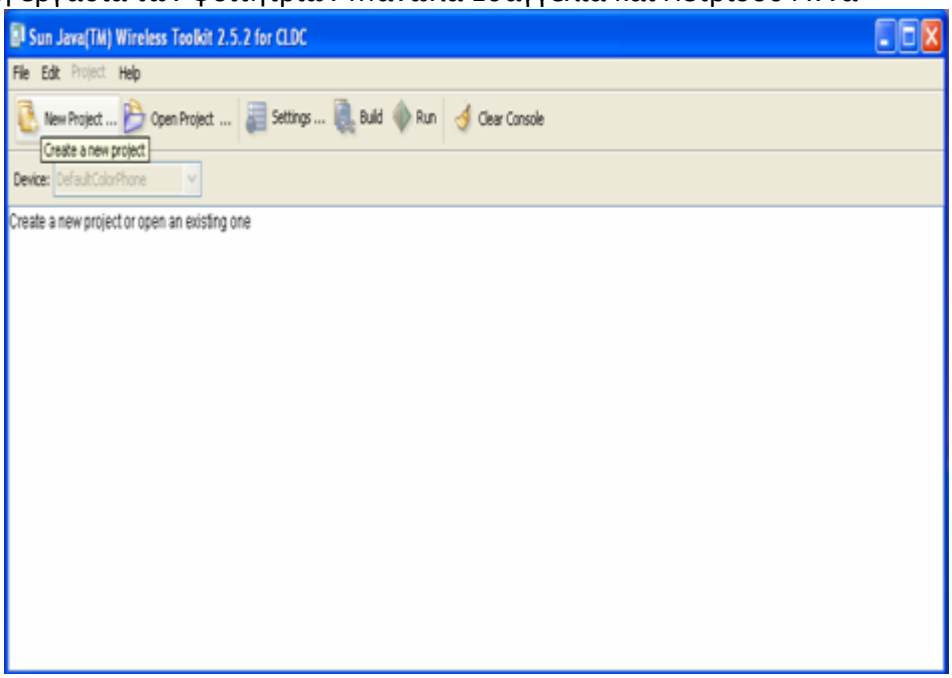
HelloWorld στο κινητό σας

Ανοίξτε το Wireless Toolkit 2.5.2



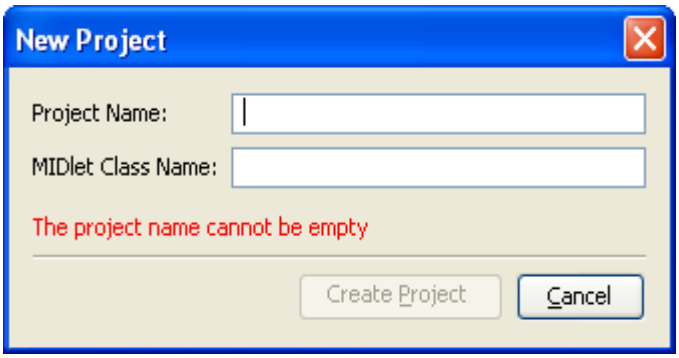
Εικόνα 8: Άνοιγμα του Wireless Toolkit

Κάντε κλικ στο New Project



Εικόνα 9: Δημιουργία νέου project

Ορίστε τόσο το Project Name όσο και το MIDlet Class Name ως HelloWorld και επιλέξτε Create Project.



Εικόνα 10: Εισαγωγή του Project Name και του MIDlet Class Name

Το MIDlet class name πρέπει να είναι το όνομα της κλάσης που επεκτείνει την κλάση MIDlet (βλ. παρακάτω). Αλλάξτε την Target Platform σε JTWI αφήνοντας default τις άλλες ρυθμίσεις και επιλέξτε OK.

Εικόνα 11: Αλλαγή της Target Platform σε JTWI

Τώρα που το project έχει δημιουργηθεί, πρέπει να προσθέσετε πηγαίο κώδικα, μέσω του JCreator.

Πρώτον, εισάγετε τα απαραίτητα midlet και lcdui πακέτα:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

Όπως όλες οι MIDP εφαρμογές, το HelloWorld παράδειγμα πρέπει να επεκτείνει την MIDlet κλαση:

```
public class HelloWorld extends MIDlet {
```

Στον δομητή, αποκτάτε την Display αναφορά και δημιουργείτε μια Form:

```
Display display;
Form mainForm;

public HelloWorld () {
 mainForm = new Form ("HelloWorld");
}
```

Μία Form είναι μια εξειδικευμένη Displayable κλάση. Η Form έχει έναν τίτλο που δίνεται στον δομητή. Δεν χρειάζεται να προσθέσετε περιεχόμενο στη φόρμα ακόμη, κι έτσι μόνο ο τίτλος θα εμφανιστεί. (Μια λεπτομερής περιγραφή της κλάσης Form γίνεται παρακάτω.)

Όταν το MIDlet σας ξεκινήσει την πρώτη φορά, ή όταν το MIDlet συνεχιστεί από μια κατάσταση παύσης, η startApp () μέθοδος καλείται από τον διαχειριστή του προγράμματος. Εδώ, ρυθμίζετε την απεικόνιση της φόρμας σας, και ζητείται από την φόρμα να εμφανιστεί:

```
public void startApp() {
  display = Display.getDisplay (this);
  display.setCurrent (mainForm);
 }
```

Όταν η εφαρμογή είναι σε παύση, εσείς δεν κάνετε τίποτα γιατί δεν έχετε καθόλου διαθέσιμους πόρους να ελευθερωθούν. Ωστόσο, θα πρέπει να παρέχετε μια κενή υλοποίηση, επειδή η υλοποίηση της pauseApp () μεθόδου είναι υποχρεωτική:

```
public void pauseApp() {
 }
```

Όπως με την pauseApp(), η υλοποίηση της destroyApp() είναι υποχρεωτική. Και πάλι, δεν χρειάζεται να κάνετε τίποτα για αυτή την απλή εφαρμογή:

```
public void destroyApp(boolean unconditional) {
}
```

Ανοίξτε το JCreator και επιλέξτε File > New File. Αφού επιλέξετε κενό project και ονομάσετε το αρχείο HelloWorld, αλλάξτε τη θέση αποθήκευσης έτσι ώστε το αρχείο να αποθηκευτεί στο φάκελο src του project HelloWorld που δημιουργήσατε στο Wireless Toolkit. Κάντε το αυτό με πλοήγηση στον κατάλογο αρχείων WTK2.5.1EA, που βρίσκεται στη μονάδα δίσκου C, και προχωρήστε στον φάκελο apps που βρίσκεται στον φάκελο HelloWorld και, τέλος, επιλέξτε το φάκελο src (ή παρατηρείστε το Path που ακολουθεί το “Place java source files in” στο wireless Toolkit του project που μόλις δημιουργήσατε) [5].

Σημείωση: Αναλυτικότερα, ο φάκελος κάθε project περιέχει τους ακόλουθους υποφακέλους:

bin ...Αυτός ο φάκελος περιέχει τα JAR/JAD δημιουργημένα αρχεία
src …Αυτός ο φάκελος πρέπει να περιέχει όλο τον πηγαίο κώδικα του project
lib …Αυτός ο φάκελος θα πρέπει να περιέχει όλες τις επιπρόσθετες κλάσεις
res …Αυτός ο φάκελος θα πρέπει να περιέχει όλα τα επιπρόσθετα αρχεία (εικόνες, ήχο κλπ)

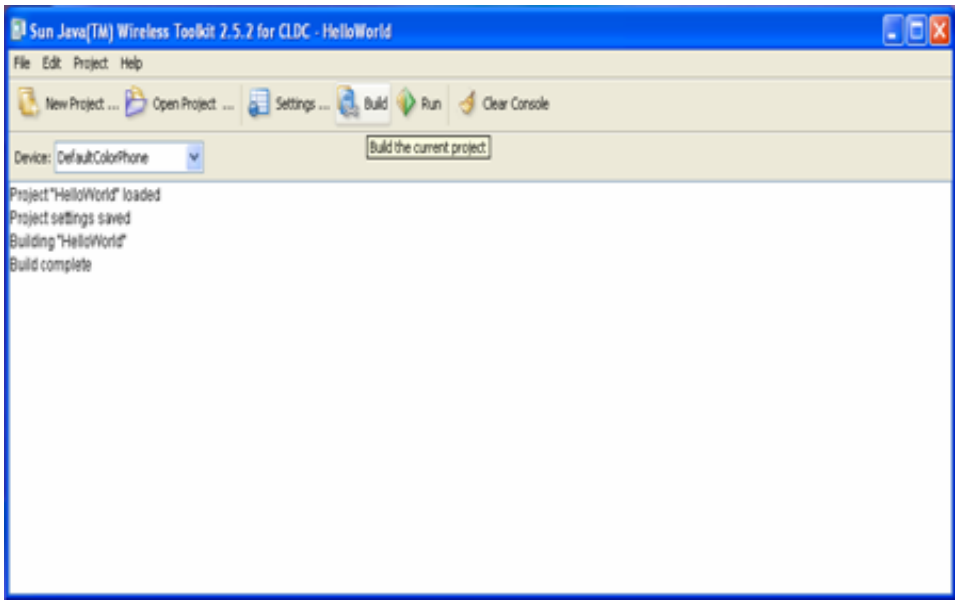
Εικόνα 12: Αποθήκευση του java αρχείου στον src φάκελο του project

Αντιγράψτε τον παρακάτω κώδικα στο HelloWorld.java:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet{
Display display;
Form mainForm;

public HelloWorld() {
mainForm = new Form ("HelloWorld");
}

public void startApp(){
display = Display.getDisplay(this);
display.setCurrent (mainForm);
}

public void pauseApp(){
```

```
}

public void destroyApp(boolean unconditional){
}
}
```

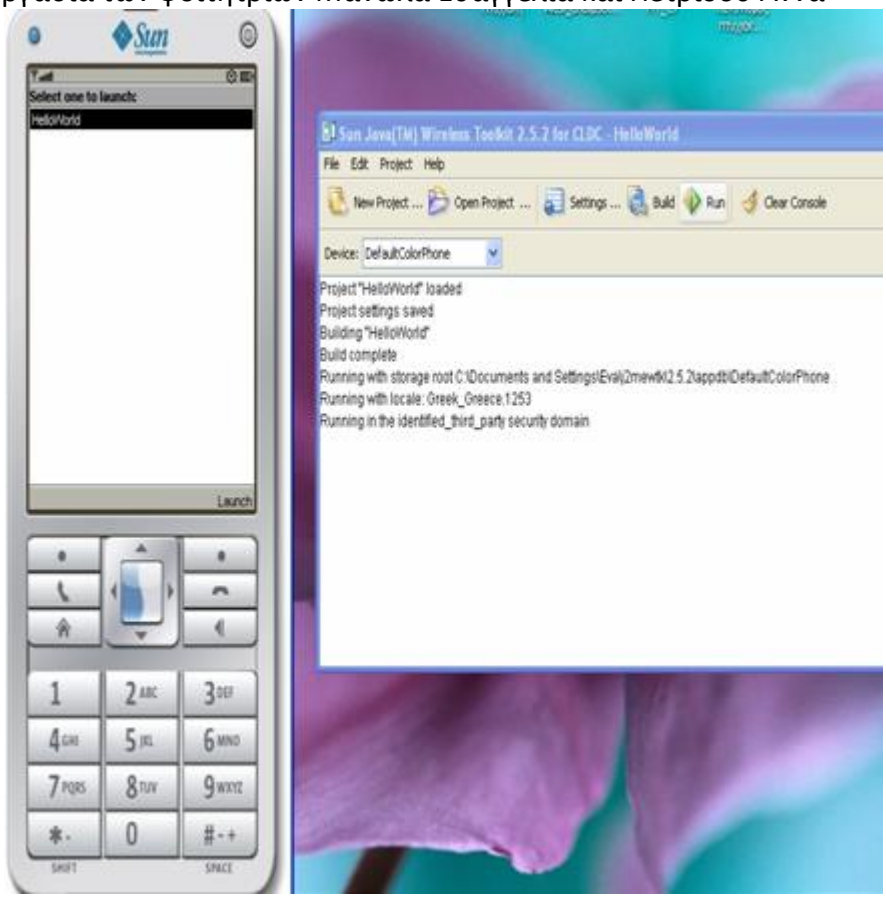
Αποθηκεύστε το αρχείο HelloWorld.java (δεν θα μπορέσετε να το μεταγλωττίσετε στο JCreator). Έχοντας το Wireless Toolkit με το project HelloWorld ακόμη ανοικτό, επιλέξτε Build.



Εικόνα 13: Build του τρέχοντος project

Αν ο κώδικας έχει αντιγραφεί σωστά, θα λάβετε ένα "Build complete" μήνυμα. Σε αντίθετη περίπτωση, βεβαιωθείτε ότι έχετε αντιγράψει σωστά τον κωδικό και προσπαθήστε ξανά. Επιλέξτε Run.

Θα εμφανιστεί ο εξομοιωτής κινητού τηλεφώνου, απεικονίζοντας το αναμενόμενο αποτέλεσμα.



Εικόνα 14: Run του τρέχοντος project

Μετά την εκτέλεση του εξομοιωτή κινητού τηλεφώνου, εξετάσετε τον κώδικα. Παρατηρείστε ότι αυτός ο κώδικας έχει τις ελάχιστες απαιτήσεις για μια εφαρμογή κινητού τηλεφώνου: εισάγει τα midlet και lcdui πακέτα, επεκτείνει την MIDlet κλάση, περιέχει έναν δομητή, μια startApp() μέθοδο, μία pauseApp() μέθοδο, και μία destroyApp() μέθοδο. Τα εισαγόμενα πακέτα περιέχουν βασικές κλάσεις βιβλιοθήκης που θα χρησιμοποιηθούν σε μια εφαρμογή κινητού τηλεφώνου. Η MIDlet είναι μια αφηρημένη κλάση στο πακέτο midlet που απαιτεί την υλοποίηση των startApp (), pauseApp () και destroyApp () μεθόδων σε μια μη αφηρημένη υποκλάση. Ο δομητής καλείται όταν δημιουργηθεί η εφαρμογή και η MIDP υλοποίηση ξεκινά την εφαρμογή μέσω της startApp () μεθόδου. Η pauseApp () και destroyApp () μέθοδοι κάνουν ότι δηλώνουν και τα ονόματά τους, αλλά σε αυτό το σημείο έχουν δοθεί κενές υλοποιήσεις.

Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό μάθατε για τον κύκλο ζωής του MIDLet, τις διάφορες καταστάσεις που μπορεί να βρεθεί και τις μεθόδους που καλεί ο διαχειριστής εφαρμογής μεταξύ αυτών των καταστάσεων. Έχετε δει βήμα – βήμα πώς ν' αναπτύξετε την πρώτη σας εφαρμογή χρησιμοποιώντας το JavaWireless Toolkit και το Jcreator.

Ερωτήσεις

1.Ποιές είναι οι ελάχιστες απαιτήσεις λογισμικού για την εκτέλεση μιας J2ME εφαρμογής;

2.Τι είναι και ποιός ο ρόλος του JAM(Java Application Manager);

3.Ποιός ο ρόλος και τί διαχειρίζεται η κλάση MIDLet;

4.Σε ποιές καταστάσεις μπορεί να βρεθεί ένα MIDLet και πότε;

5.Πότε καλείται η startApp() και τί κάνουν οι pauseApp() και destroyApp() μέθοδοι;

6.Ποιές μεθόδους πρέπει να καλέσει ένα MIDLet προκειμένου να ενημερώσει τον διαχειριστή εφαρμογής ότι θέλει να βρεθεί σε κατάσταση παύσης ή να τερματιστεί;

7.Σε ποιό πακέτο βρίσκονται όλες οι κλάσεις διεπαφής χρήστη του MIDP;

8.Ποιά είναι η διαφορά ανάμεσα σε ένα Display και σε ένα Displayable αντικείμενο;

9.Σε ποιόν φάκελο του project σας πρέπει να τοποθετούνται επιπρόσθετα αρχεία (εικόνα, ήχος κλπ) και σε ποιόν ο πηγαίος κώδικας;

ΚΕΦΑΛΑΙΟ 3

3. High - Level API

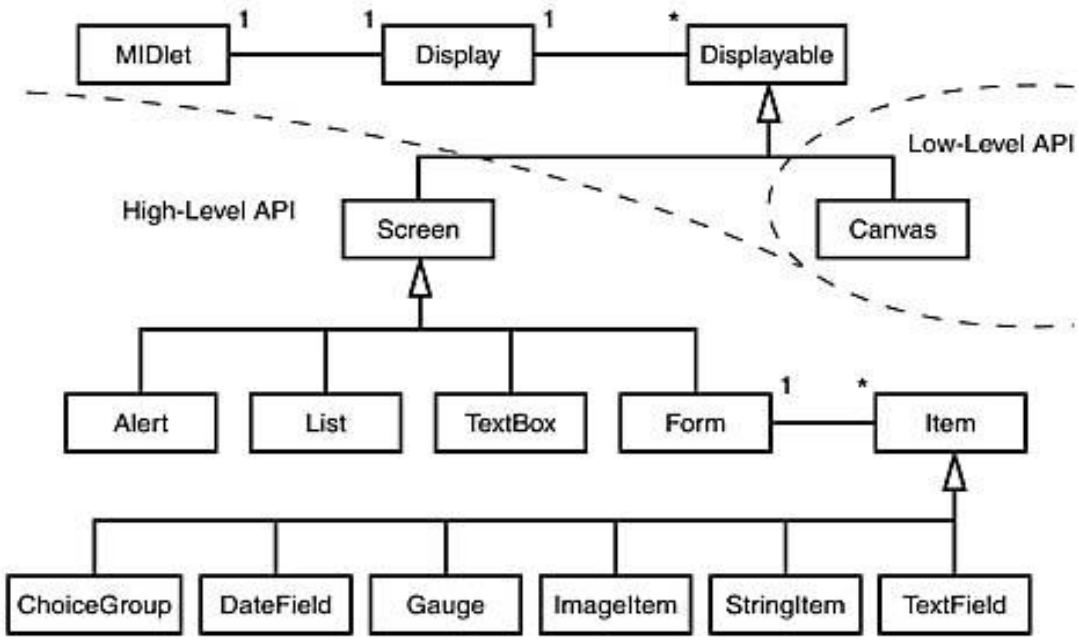
MIDP User Interface APIs

Το API διεπαφής χρήστη του MIDP χωρίζεται σε υψηλού και χαμηλού επιπέδου API. Το υψηλού επιπέδου API παρέχει στοιχεία εισόδου όπως text fields, choices, και gauges. Σε αντίθεση με το Abstract Windows Toolkit (AWT), τα υψηλού επιπέδου στοιχεία δεν μπορούν να τοποθετηθούν (ή να φωλιαστούν) ελεύθερα. Υπάρχουν μόνο δύο σταθερά επίπεδα: Screen και Item. Τα ItemS μπορούν να τοποθετηθούν μέσα σε μια Form, που είναι ένα εξειδικευμένο Screen.

Οι υψηλού επιπέδου Screen και χαμηλού επιπέδου Canvas κλάσεις έχουν ως κοινή βάση την κλάση Displayable. Όλες οι υποκλάσεις της Displayable διαμορφώνουν ολόκληρη την οθόνη της συσκευής. Οι υποκλάσεις της Displayable εμφανίζονται στη συσκευή χρησιμοποιώντας την μέθοδο setCurrent () του αντικειμένου Display. Η απεικόνιση της οθόνης του MIDlet μπορεί να προσπελαστεί καλώντας την στατική μέθοδο getDisplay (), όπου το ίδιο το MIDlet δίνεται ως παράμετρος. Στο παράδειγμα HelloWorld, αυτό το βήμα εκτελείται στις ακόλουθες δύο γραμμές:

```
Display display = Display.getDisplay (this);
  ...
  display.setCurrent (mainForm);
```

Το ακόλουθο σχήμα παρουσιάζει μια επισκόπηση των MIDP GUI κλάσεων και τη δομή της κληρονομικότητας τους.



Εικόνα 15: Οι MIDP GUI κλάσεις και η δομή της κληρονομικότητας τους [13]

Τα ακόλουθα τμήματα αρχικά περιγράφουν το υψηλού επιπέδου API και στη συνέχεια, το χαμηλού επιπέδου API.

Τώρα που γνωρίζετε τα βασικά στοιχεία του κύκλου ζωής του MIDlet και το γενικό μοντέλο απεικόνισης, μπορούμε να αρχίσουμε να εξετάζουμε βαθύτερα το πακέτο lcdui. Θα ξεκινήσουμε με μια άλλη υποκλάση του Screen: την Alert. Μετά θα συζητήσουμε για κάποια απλά ItemS, όπως τα StringItem και ImageItem. Θα εξηγήσουμε τη χρήση πιο προηγμένων ItemS όπως τα TextField και ChoiceGroup δημιουργώντας ένα απλό παράδειγμα, το TeleTransfer. Καθώς προχωρούμε σε νέες δυνατότητες του υψηλού επιπέδου UI του MIDP όπως άλλες Screen υποκλάσεις, θα επεκτείνουμε το παράδειγμα TeleTransfer βήμα προς βήμα.

3.1 Alerts

Γνωρίζετε ήδη την κλάση Form από το πρώτο παράδειγμα. Η απλούστερη υποκλάση του Screen είναι η Alert. Η Alert παρέχει ένα μηχανισμό για να δείχνει ένα παράθυρο διαλόγου για ένα περιορισμένο χρονικό διάστημα. Αποτελείται από μια ετικέτα, ένα κείμενο και ένα προαιρετικό Image. Επιπλέον, είναι δυνατό να οριστεί ένα χρονικό διάστημα όπου το Alert θα εμφανίζεται πριν οποιοδήποτε άλλο

αντικείμενο Screen εμφανιστεί. Εναλλακτικά, ένα Alert μπορεί να εμφανίζεται μέχρι ο χρήστης να το επιβεβαιώνει. Εάν το Alert δεν χωράει στην οθόνη και η κύλιση είναι απαραίτητη για να δείτε ολόκληρο το περιεχόμενο, ο περιορισμός χρόνου απενεργοποιείται αυτόματα.

Το ακόλουθο απόσπασμα κώδικα δημιουργεί ένα Alert με τον τίτλο "HelloAlert" και το εμφανίζει μέχρι να επιβεβαιωθεί από το χρήστη:

```
Alert alert = new Alert ("HelloAlert");
alert.setTimeout (Alert.FOREVER);
display.setCurrent (alert);
```

3.2 Forms και Items

Η πιο σημαντική υποκλάση της Screen είναι η κλάση Form. Η Form μπορεί να κρατήσει οποιοδήποτε αριθμό από Items, όπως StringItems, TextFields, και ChoiceGroups. Χειρίζεται το layout και το scrolling των Items αυτόματα. Τα Items μπορούν να προστεθούν στη Form χρησιμοποιώντας την μέθοδο append ().

Η Item είναι μια αφηρημένη κλάση, η οποία παρέχει μια ετικέτα που αποτελεί κοινή ιδιότητα όλων των υποκλάσεων. Η ετικέτα μπορεί να προσπελαστεί χρησιμοποιώντας τις μεθόδους setLabel () και getLabel (), αντίστοιχα. Η ετικέτα είναι προαιρετική, και μια null τιμή δηλώνει ότι το Item δεν έχει ετικέτα. Για να επιτρέψετε στον χρήστη να παρακολουθεί την κατάσταση του προγράμματος, συνιστάται να δίνετε μια ετικέτα τουλάχιστον για τα αλληλεπιδραστικά αντικείμενα.

Ο παρακάτω πίνακας παρέχει μια επισκόπηση όλων των Items που είναι διαθέσιμα στο MIDP.

Πίνακας 5: Όλες οι υποκλάσεις της κλάσης Item [13]

Item	Περιγραφή
ChoiceGroup	Επιτρέπει την επιλογή στοιχείων ενός group

DateField	Χρησιμοποιείται για την επεξεργασία ημερομηνίας και ώρας
Gauge	Απεικονίζει ένα ραβδόγραμμα για ακέραιες τιμές
ImageItem	Χρησιμοποιείται για να ελέγχει το layout ενός Image
StringItem	Χρησιμοποιείται για read-only στοιχεία κειμένου
TextField	Κρατάει ένα μιας γραμμής πεδίο εισαγωγής

3.2.1 StringItem

Τα StringItems είναι απλά, read-only στοιχεία κειμένου που αρχικοποιούνται μόνο με την ετικέτα και μια παράμετρο τύπου String. Το ακόλουθο απόσπασμα κώδικα δείχνει τη δημιουργία μιας απλής ετικέτας έκδοσης. Μετά τη δημιουργία, η ετικέτα προστίθεται στον δομητή της εφαρμογής HelloWorld:

```
public HelloWorld () {
 mainForm = new Form ("HelloWorld");
 StringItem versionItem = new StringItem ("Version: ", "1.0");
 mainForm.append (versionItem);
}
```

Η ετικέτα ενός StringItem μπορεί να προσπελαστεί χρησιμοποιώντας τις μεθόδους setLabel () και getLabel () που κληρονομούνται από την Item. Για την πρόσβαση στο κείμενο, μπορείτε να χρησιμοποιήσετε τις μεθόδους setText () και getText ().

3.2.2 ImageItem

Όπως και το StringItem, το ImageItem είναι ένα απλό μη διαδραστικό Item. Εκτός από την ετικέτα, ο δομητής του ImageItem παίρνει ένα Image αντικείμενο, μια παράμετρο διάταξης, και ένα εναλλακτικό κείμενο που θα εμφανίζεται όταν η συσκευή δεν είναι σε θέση να εμφανίζει την εικόνα. Η εικόνα που δίνεται στον δομητή πρέπει να είναι μη-μεταβλητή.

Η διαφορά μεταξύ μεταβλητών και μη μεταβλητών Images περιγράφεται με περισσότερες λεπτομέρειες παρακάτω, στο "Low Level API". Για τώρα, θα αντιμετωπίζουμε την κλάση Image ως ένα «μαύρο κουτί» που έχει έναν δομητή που υποδηλώνει τη θέση της εικόνας στο JAR αρχείο.

Η εικόνα πρέπει να τοποθετηθεί στον φάκελο res/ του project μας. Ο φάκελος res περιέχει όλα τα resource files συμπεριλαμβανομένων και των εικόνων. Αν η εικόνα τοποθετηθεί οπουδήποτε αλλού ή δεν μπορεί να φορτωθεί για οποιοδήποτε άλλο λόγο, θα προκληθεί IOException.

Όλες οι εικόνες πρέπει να αποθηκεύονται σε Portable Network Graphics (PNG) μορφή.

Οι παράμετροι διάταξης παρατίθενται στον παρακάτω πίνακα.

Πίνακας 6: Σταθερές διάταξης του ImageItem [13]

Σταθερά	**Τιμή**
LAYOUT_CENTER	Η εικόνα κεντράρεται οριζοντίως
LAYOUT_DEFAULT	Μια default μορφοποίηση εφαρμόζεται στην εικόνα
LAYOUT_LEFT	Η εικόνα είναι στοιχισμένη αριστερά
LAYOUT_NEWLINE_AFTER	Μια νέα γραμμή θα αρχίσει μετά την εικόνα
LAYOUT_NEWLINE_BEFORE	Μια νέα γραμμή θα αρχίσει πριν την εικόνα
LAYOUT_RIGHT	Η εικόνα είναι στοιχισμένη δεξιά

Το ακόλουθο απόσπασμα κώδικα δείχνει πώς ένα κεντραρισμένο ImageItem προστίθεται στο HelloWorld παράδειγμα:

```
public HelloWorld() {
 display = Display.getDisplay (this);
 mainForm = new Form ("HelloWorld");
  try {
  ImageItem logo = new ImageItem
   ("Copyright: ", Image.createImage ("/mcp.png"),
    ImageItem.LAYOUT_CENTER |
```

```
        ImageItem.LAYOUT_NEWLINE_BEFORE
        | ImageItem.LAYOUT_NEWLINE_AFTER, "Macmillian USA");

      mainForm.append (logo);
    }
    catch (IOException e) {
     mainForm.append (new StringItem
      ("Copyright", "Sams Publishing; Image not available:" + e));
    }
  }
```

Αρχίζοντας μια νέα γραμμή πρίν και μετά την εικόνα, εξασφαλίζετε ότι η εικόνα είναι κεντραρισμένη στην δικιά της γραμμή. Η παρακάτω εικόνα δείχνει την οθόνη της συσκευής.

Εικόνα 16: Έξοδος του HelloWorld

3.2.3 Χειρισμός εισαγωγής κειμένου στα TextFields

Όπως φαίνεται στον παρακάτω πίνακα, ο χειρισμός της εισαγωγής κειμένου γίνεται από την κλάση TextField. Ο δομητής της TextField παίρνει τέσσερις τιμές: μια ετικέτα, αρχικό κείμενο, μέγιστο μέγεθος του κειμένου, και τους περιορισμούς

που δείχνουν τον τύπο που επιτρέπεται να εισαχθεί. Προκειμένου να αποφευχθεί η εισαγωγή μη επιτρεπτών χαρακτήρων, οι περιορισμοί μπορεί επίσης να επηρεάσουν τη λειτουργία του πληκτρολογίου. Πολλές MIDP συσκευές έχουν αριθμητικό πληκτρολόγιο μόνο, και οι περιορισμοί επιτρέπουν στον διαχειριστή εφαρμογής να αλλάξει τις εκχωρήσεις των πλήκτρων αναλόγως. Οι σταθερές που παρατίθενται στον παρακάτω πίνακα, δηλωμένες στην κλάση TextField, είναι οι έγκυρες τιμές περιορισμών.

Πίνακας 7: Έγκυρες τιμές περιορισμών του TextField [13]

Σταθερά	**Τιμή**
ANY	Επιτρέπει την προσθήκη οποιουδήποτε κειμένου
EMAILADDR	Προσθέτει μια έγκυρη ηλεκτρονική διεύθυνση
NUMERIC	Επιτρέπει ακέραιες τιμές
PASSWORD	Επιτρέπει στον χρήστη να εισάγει έναν κωδικό, όπου το εισαγόμενο κείμενο έχει την συνηθισμένη μορφή password.
PHONENUMBER	Επιτρέπει στον χρήστη να εισάγει έναν αριθμό τηλεφώνου
URL	Επιτρέπει ένα έγκυρο URL

Θα δείτε τώρα την χρήση των TextFields με τη δημιουργία ενός απλού παραδείγματος για τραπεζικές μεταφορές. Μία φόρμα τραπεζικών μεταφορών περιέχει τουλάχιστον το ποσό των χρημάτων που πρόκειται να μεταφερθεί και το όνομα του παραλήπτη.

Για να ξεκινήσει η υλοποίηση του TeleTransfer MIDlet, πρέπει πρώτα να εισάγετε τα δύο πακέτα που περιέχουν τις midlet και lcdui κλάσεις:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

Κάθε MID εφαρμογή προέρχεται από ένα MIDlet, έτσι επίσης πρέπει να επεκτείνετε την MIDlet κλάση:

```
public class TeleTransfer extends MIDlet {
```

Δημιουργήστε ένα νέο project με Project Name και MIDlet Class Name TeleTransfer. Δημιουργήστε επίσης ένα νέο αρχείο Java στον κατάλληλο φάκελο (src, του παραπάνω project).

Επειδή θέλετε να δημιουργήσετε μια Form που περιέχει Items για να εισάγετε τις πληροφορίες μεταφοράς, χρειάζεστε μία αντίστοιχη μεταβλητή mainForm. Μπορείτε ακόμη να αρχικοποιήσετε την μεταβλητή κατά την δήλωσή της επειδή δεν εξαρτάται από τις παραμέτρους του δομητή.

```
Form mainForm = new Form ("TeleTransfer");
```

Προκειμένου να επιτρέψετε τον χρήστη να εισάγει τις πληροφορίες μεταφοράς, προσθέστε TextFields για το όνομα του παραλήπτη και για την ποσότητα χρημάτων που θα μεταφερθεί. Λόγω της έλλειψης floating-point τιμών στην CLDC, τα αριθμητικά TextFields στο MIDP μπορούν να κρατήσουν μόνο ακέραιες τιμές. Έτσι πρέπει να διαχωρίσετε το ποσό σε ξεχωριστά πεδία για δολάρια και σεντς. (Μια εναλλακτική λύση θα ήταν να χρησιμοποιήσετε ένα αλφαριθμητικό πεδίο και να αναλύσετε το string σε δύο ξεχωριστές τιμές. Σε αυτή την περίπτωση, θα περιορίσετε το μέγεθος των πιθανών τιμών σε πέντε ψηφία για το σύνολο του τμήματος των δολαρίων και δύο ψηφία για το κλασματικό μέρος.) Και πάλι, θα αρχικοποιήσετε τις μεταβλητές εκεί όπου δηλώνονται:

```
TextField receiverName = new TextField ("Receiver Name", "", 20,
                                        TextField.ANY);

TextField receiverAccount = new TextField ("Receiver Account#", "", 12,
                                           TextField.NUMERIC);

TextField amountWhole = new TextField ("Dollar", "", 6,TextField.NUMERIC);

TextField amountFraction = new TextField("cent", "", 2,TextField.NUMERIC);
```

Τέλος, προσθέστε ένα πεδίο αποθήκευσης του Display στιγμιότυπου για την εφαρμογή σας:

```
Display display = Display.getDisplay (this);
```

Τώρα μπορείτε να προσθέσετε τον δομητή στην εφαρμογή σας προσθέτοντας τα προηγούμενα TextFields στην κυρίως φόρμα:

```
public TeleTransfer() {
mainForm.append (receiverName);
mainForm.append (receiverAccount);
mainForm.append (amountWhole);
mainForm.append (amountFraction);
}
```

Όταν ξεκινήσει η εφαρμογή, ζητάτε απ' το display να δείξει την φόρμα μεταφοράς χρημάτων καλώντας την setCurrent (). Ο διαχειριστής εφαρμογής σας ειδοποιεί για την έναρξη της εφαρμογής καλώντας την startApp () μέθοδο. Έτσι υλοποιείτε αυτή τη μέθοδο ανάλογα:

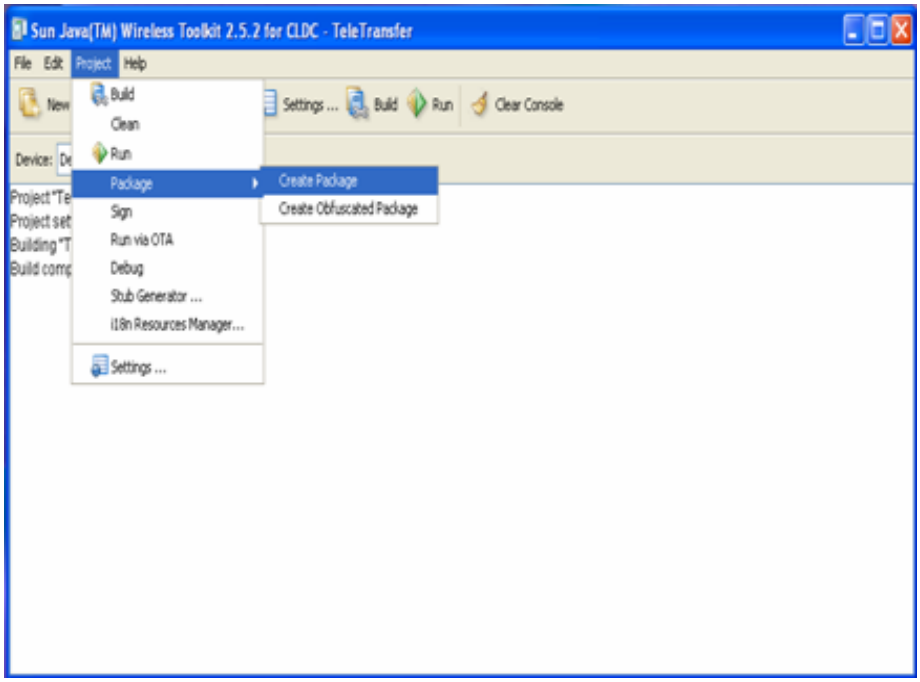
```
public void startApp() {
display.setCurrent (mainForm);
}
```

Σημειώστε ότι η startApp () καλείται επίσης όταν το MIDlet ξαναρχίζει από κατάσταση παύσης, έτσι δεν μπορείτε να μετακινήσετε τον κώδικα αρχικοποίησης από τον δομητή σε αυτήν την μέθοδο. Τόσο η pauseApp () όσο και η destroyApp () δηλώνονται ως abstract στην MIDlet κλάση, έτσι πρέπει να υλοποιήσετε αυτές τις μεθόδους στην εφαρμογή σας, ακόμα κι αν δεν έχετε πραγματικό περιεχόμενο γι αυτές. Απλώς παρέχετε κενές υλοποιήσεις, όπως στο HelloWorld παράδειγμα που είδαμε προηγουμένως:

```
public void pauseApp() {
}
```

```
public void destroyApp (boolean unconditional) {

}
```

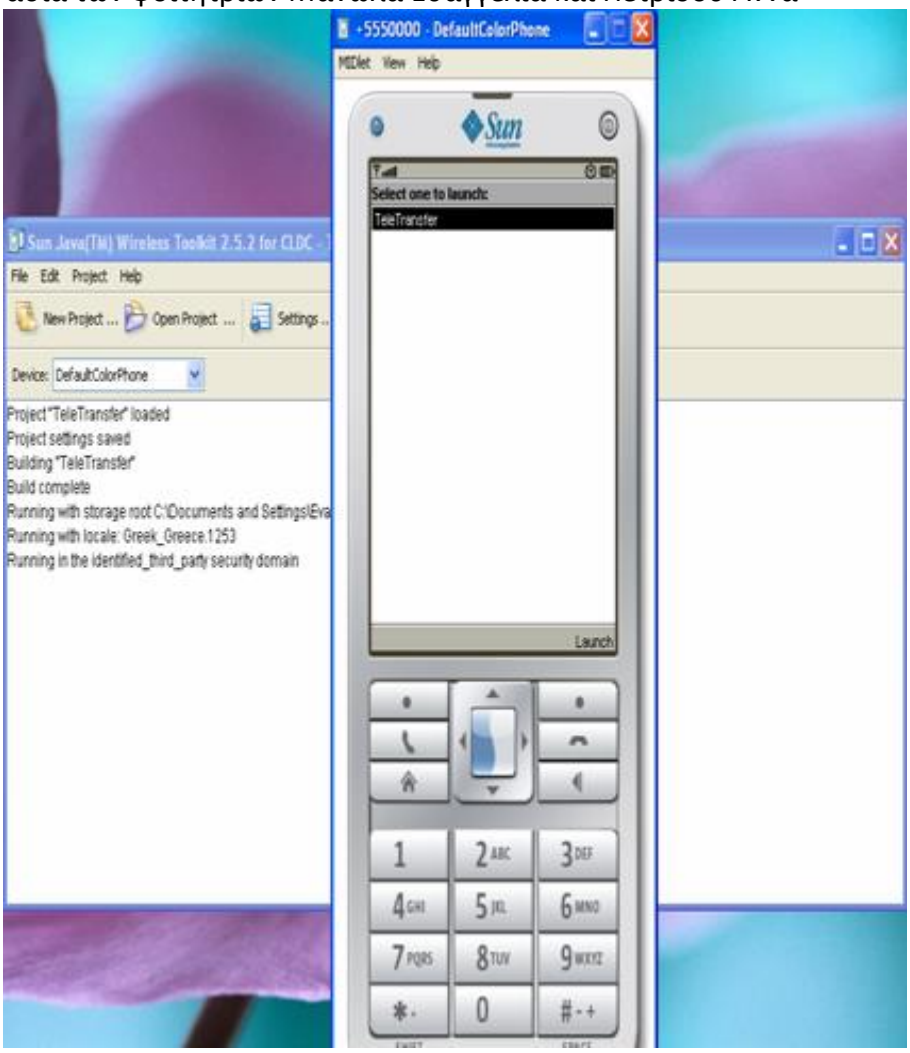
Αφού δημιουργήσετε το TeleTransfer project και το TeleTransfer.java αρχείο (αντιγράφοντας τον παραπάνω κώδικα), το αποθηκεύσετε στον κατάλληλο φάκελο, όπως προαναφέρθηκαν στο προηγούμενο παράδειγμα, και κάνετε build από το wireless toolkit, επιλέξτε Project -> Package -> Create Package ώστε να δημιουργήσετε τα JAR/JAD αρχεία.



Εικόνα 17: Δημιουργία Package

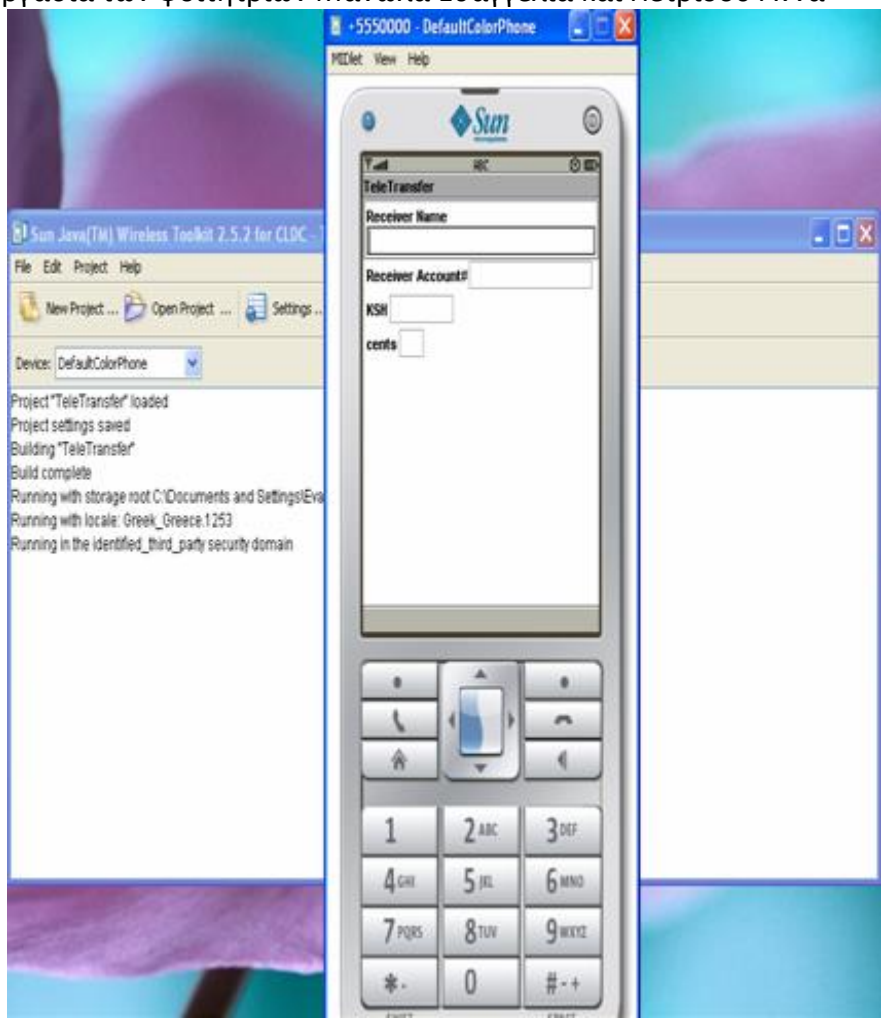
Μπορείτε τώρα να βρείτε τα JAR/JAD αρχεία μέσα στον φάκελο bin του project σας.

Μπορείτε να εγκαταστήσετε τα JAR και JAD αρχεία στο κινητό σας τηλέφωνο [10] [13].



Εικόνα 18: Προσομοίωση φόρτωσης του TeleTransfer

Εναλλακτικά, μπορείτε να επιλέξετε Run και να τρέξετε τον κώδικα σας μέσω του wireless toolkit σε έναν από τους προσομοιωτές του (όπως δείξαμε στο προηγούμενο παράδειγμα).



Εικόνα 19: Προσομοίωση εκτέλεσης του TeleTransfer

3.2.4 Επιλογή στοιχείων χρησιμοποιώντας ChoiceGroups

Προηγουμένως, δημιουργήσατε μια απλή φόρμα για να εισάγετε πληροφορίες για τη μεταφορά χρημάτων μεταξύ δύο λογαριασμών. Τώρα θα επεκτείνετε την εφαρμογή για να επιτρέψετε στο χρήστη να επιλέξει διαφορετικά νομίσματα. Για το σκοπό αυτό, τώρα θα προσθέσετε ένα ChoiceGroup στην εφαρμογή σας.

Το ChoiceGroup είναι ένα MIDP UI widget που επιτρέπει στο χρήστη να επιλέξει μεταξύ διαφορετικών στοιχείων σε μια φόρμα. Τα στοιχεία αυτά αποτελούν απλά Strings, αλλά μπορεί να εμφανιστεί μια προαιρετική εικόνα ανά στοιχείο. Τα ChoiceGroups μπορεί να είναι δύο διαφορετικών ειδών. Οι αντίστοιχες σταθερές των ειδών αυτών ορίζονται στην Choice διεπαφή. Αυτές οι σταθερές χρησιμοποιούνται στην κλάση List επίσης. Η κλάση List επιτρέπει ένα

επιπλέον τρίτο είδος. Οι τρεις σταθερές των ειδών αυτών αναφέρονται στον παρακάτω πίνακα.

Πίνακας 8: Σταθερές των τύπων της Choice [13]

Σταθερά	**Τιμή**
EXCLUSIVE	Καθορίζει ένα ChoiceGroup ή ένα List έχοντας ένα μόνο στοιχείο επιλεγμένο την ίδια στιγμή
IMPLICIT	Ισχύει μόνο για ListS. Επιτρέπει στην List να στείλει Commands που υποδεικνύουν αλλαγές κατάστασης
MULTIPLE	Σε αντίθεση με το EXCLUSIVE, το MULTIPLE επιτρέπει την επιλογή πολλαπλών στοιχείων.

Ο δομητής της ChoiceGroup απαιτεί τουλάχιστον μια ετικέτα και μια σταθερά του είδους. Επιπλέον, ένας πίνακας String και ένας πίνακας Image που περιέχουν τα στοιχεία μπορούν να περαστούν στον δομητή. Τα στοιχεία μπορούν επίσης να προστεθούν δυναμικά χρησιμοποιώντας την μέθοδο append (). Η μέθοδος append () έχει δύο παραμέτρους, ένα String για την ετικέτα και ένα Image. Και στις δύο περιπτώσεις, η παράμετρος Image μπορεί να είναι null, εφόσον δεν θέλουμε εικόνες.

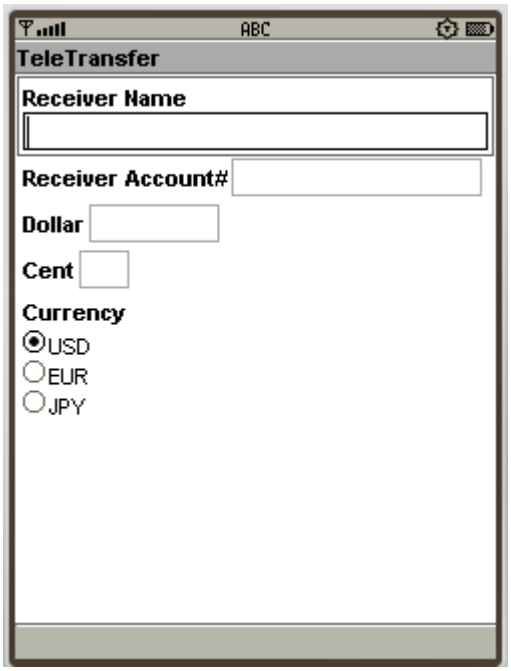
Για να προσθέσετε ένα ChoiceGroup στην εφαρμογή TeleTransfer, εισάγετε μια νέα μεταβλητή currency τύπου ChoiceGroup. Καθορίζοντας τον τύπο ως EXCLUSIVE, παίρνετε ένα ChoiceGroup όπου μόνο ένα στοιχείο μπορεί να επιλεγεί κάθε φορά. Προσθέτετε στοιχεία για τις ΗΠΑ (USD), την Ευρωπαϊκή Ένωση (EUR), και την Ιαπωνία (JPY), περνώντας έναν String πίνακα. Το ChoiceGroup επιτρέπει στο χρήστη να επιλέξει μεταξύ τριών νομισμάτων, που εκπροσωπούνται με λέξεις από τις συντομογραφίες που αναφέρονται στον πίνακα String. Η τελευταία παράμετρος του δομητή ορίζεται ως null, επειδή δεν θέλετε Images να εμφανίζονται αυτή τη στιγμή:

```
ChoiceGroup currency = new ChoiceGroup
("Currency", Choice.EXCLUSIVE,
new String[] {"USD", "EUR", "JPY"}, null);
```

Πρέπει να προσθέσετε το currency ChoiceGroup στην κύρια φόρμα σας. Όπως και για τα πεδία κειμένου, αυτό γίνεται μέσω της μεθόδου append () της Form:

```
mainForm.append (currency);
```

Το παρακάτω σχήμα δείχνει την εφαρμογή TeleTransfer που έχει επεκταθεί για να επιλέξετε ένα νόμισμα με τη χρήση ενός ChoiceGroup.



Εικόνα 20: Έξοδος του TeleTransfer μετά από χρήση ενός ChoiceGroup

3.2.5 Περαιτέρω Item κλάσεις: Gauge και DateField

Τώρα έχετε χρησιμοποιήσει όλες τις Item υποκλάσεις εκτός από τις Gauge και DateField. Και οι δύο κλάσεις είναι εξειδικευμένα στοιχεία εισόδου, όπου η Gauge μπορεί επίσης να έχει νόημα ως ένα καθαρά read-only στοιχείο πληροφοριών.

Το Gauge στοιχείο απεικονίζει μια οριζόντια μπάρα και αρχικοποιείται με μια ετικέτα, ένα flag που δείχνει αν είναι διαδραστικό, ένα ανώτατο όριο και μια αρχική τιμή.

```
Gauge(String label, boolean interactive, int maxValue, int initialValue)
```

Εάν ένα Gauge είναι διαδραστικό, ο χρήστης έχει τη δυνατότητα να αλλάξει την τιμή χρησιμοποιώντας μια, ανάλογα με την συσκευή, μέθοδο εισόδου. Αλλαγές στην τιμή του Gauge θα προκαλέσουν ItemEvents εάν ένας αντίστοιχος ακροατής έχει καταχωρηθεί στη φόρμα.

Το ακόλουθο απόσπασμα κώδικα δείχνει την κατασκευή και προσθήκη ενός μη διαδραστικού Gauge με ετικέτα Progress που έχει αρχικοποιηθεί με την τιμή 0 και μέγιστο όριο το 100:

```
Gauge gauge = new Gauge ("Progress", false, 100, 0);
…
mainForm.append(gauge);
```

Εάν ένα Gauge χρησιμοποιείται για την απεικόνιση της προόδου μιας διαδικασίας που διαρκεί μεγαλύτερο χρονικό διάστημα, θα πρέπει επίσης να προσθέσετε μια αντίστοιχη εντολή Stop στη φόρμα για να διακόψετε την πρόοδο.

Η τρέχουσα τιμή του Gauge μπορεί να οριστεί χρησιμοποιώντας τη μέθοδο setValue () και να αναγνωστεί χρησιμοποιώντας τη μέθοδο getValue (). Ανάλογα, οι setMaxValue () και getMaxValue () μέθοδοι σας επιτρέπουν να έχετε πρόσβαση στη μέγιστη τιμή του Gauge. Ακόμη, ελέγχετε αν είναι διαδραστικό με την isInteractive() μέθοδο.

Το DateField είναι ένα εξειδικευμένο widget για την εισαγωγή πληροφοριών ημερομηνίας και ώρας με έναν απλό τρόπο. Μπορεί να χρησιμοποιηθεί για να εισάγετε μια ημερομηνία, μια ώρα, ή και τα δύο είδη πληροφοριών ταυτόχρονα. Η εμφάνιση του DateField προσδιορίζεται χρησιμοποιώντας τρείς δυνατές σταθερές. Οι πιθανές αυτές σταθερές αναφέρονται στον παρακάτω πίνακα.

Πίνακας 9: Σταθερές του DateField [13]

Σταθερά	**Τιμή**
DATE	Περνιέται αν το DateField χρησιμοποιείται για εισαγωγή

	ημερομηνίας μόνο.
DATE_TIME	Χρησιμοποιείται για την δημιουργία ενός DateField για εισαγωγή πληροφοριών ημερομηνίας και ώρας.
TIME	Χρησιμοποιείται για εισαγωγή πληροφοριών ώρας μόνο.

Η κλάση DateField έχει δύο δομητές στους οποίους η ετικέτα και η μορφή μπορούν να προσδιοριστούν. Χρησιμοποιώντας τον δεύτερο δομητή, ένα πρόσθετο TimeZone μπορεί να περαστεί. Το ακόλουθο απόσπασμα κώδικα δείχνει πώς ένα DateField για την εισαγωγή της ημερομηνίας γέννησης μπορεί να αρχικοποιηθεί:

```
DateField dateOfBirth = new DateField("Date of birth:", DateField.DATE);
```

Αφού εισάγετε την ημερομηνία στο DateField, μπορεί να προσπελαστεί χρησιμοποιώντας την getdate () μέθοδο. Η DateField προσφέρει μερικές πρόσθετες μεθόδους για να πάρει πληροφορίες σχετικά με την μορφή και μεθόδους για τον καθορισμό της μορφής εισαγωγής.

3.3 Λήψη αλλαγών από διαδραστικά υψηλού επιπέδου UI Items

Εάν εκτελέσετε την νέα έκδοση του TeleTransfer MIDlet, μπορείτε να αλλάξετε το νόμισμα χρησιμοποιώντας το ChoiceGroup, αλλά οι ετικέτες των TextFields για το Dollar και το Cent δεν αλλάζουν αναλόγως. Χρειάζεστε έναν τρόπο για να γνωστοποιήσετε στην εφαρμογή μια επιλογή που γίνεται στο νόμισμα του ChoiceGroup.

Η λήψη αλλαγών από διαδραστικά υψηλού επιπέδου UI items στο MIDP βασίζεται σε ένα πρότυπο ακροατή (listener) παρόμοιο με αυτό του AWT. Οι κλάσεις που υλοποιούν την ItemStateListener διεπαφή είναι σε θέση να λαμβάνουν τις ειδοποιήσεις για τα ακόλουθα γεγονότα:

- Αλλαγή κατάστασης ενός ChoiceGroup
- Ρύθμιση τιμής ενός διαδραστικού Gauge

- Η τιμή του TextField αλλάζει
- Αλλαγή του DateField

Τα γεγονότα στέλνονται στη μέθοδο itemStateChanged () του ItemStateListener, όπου το στοιχείο που έχει αλλάξει δίνεται ως παράμετρος. Προκειμένου να λάβει αυτά τα γεγονότα, το ItemStateChangeListener πρέπει να καταχωρηθεί χρησιμοποιώντας την μέθοδο setItemStateListener () στην αντίστοιχη Form.

Τώρα που ξέρετε για τα γεγονότα αλλαγής κατάστασης ενός item, μπορείτε να προσθέσετε την λειτουργικότητα που θέλετε στο TeleTransfer MIDlet σας. Αρχικά, πρέπει να προσθέσετε την ItemStateListener διεπαφή με τη δήλωση της κλάσης:

```
public class TeleTransfer extends MIDlet implements ItemStateListener {
```

Πρέπει επίσης να υλοποιήσετε μια αντίστοιχη itemStateChanged () μέθοδο. Δεδομένου ότι η itemStateChanged () μέθοδος καλείται για αλλαγές από όλα τα ItemS στη Form, θα πρέπει να ελέγξετε την παράμετρο που δείχνει την πηγή του γεγονότος πρώτα. Εάν η πηγή του γεγονότος είναι το νόμισμα ChoiceGroup, ορίζετε τις ετικέτες του ποσού και του κλασματικού μέρους των TextFields αντίστοιχα:

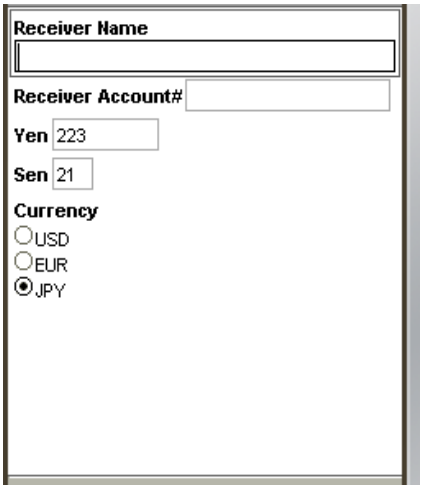
```
public void itemStateChanged (Item item) {
 if (item == currency) {
  int index = currency.getSelectedIndex ();
  switch (index) {
  case 0: amountWhole.setLabel ("Dollar");
     amountFraction.setLabel ("Cent");
     break;
  case 1: amountWhole.setLabel ("Euro");
    amountFraction.setLabel ("Cent");
     break;
  case 2: amountWhole.setLabel ("Yen");
    amountFraction.setLabel ("Sen");
```

```
        } } }
```

Απλά προσθέτοντας την διεπαφή και υλοποιώντας τις αντίστοιχες μεθόδους δεν αρκεί για να επιτρέψετε στο MIDlet να λάβει αλλαγές κατάστασης. Θα πρέπει επιπλέον να καταχωρήσετε το ItemStateListener στη Form που περιέχει το στοιχείο currency. Αυτό το κάνετε, καλώντας την μέθοδο setItemStateListener () στον δομητή της κλάσης TeleTransfer:

```
public TeleTransfer () {
        ...
 mainForm.append (currency);
 mainForm.setItemStateListener (this);
}
```

Η παρακάτω εικόνα δείχνει την νέα έκδοση του παραδείγματος TeleTransfer, όπου οι ετικέτες έχουν αλλάξει ανάλογα με την επιλογή του νομίσματος στο ChoiceGroup.



Εικόνα 21: Έξοδος του TeleTransfer με αλλαγή ετικετών ανάλογα με την επιλογή του νομίσματος

3.4 Χρησιμοποιώντας Εντολές για αλληλεπίδραση με τον χρήστη

Τώρα μπορείτε να εισάγετε όλες τις πληροφορίες που απαιτούνται για μια μεταφορά, αλλά δεν έχετε κανένα μέσο για να ξεκινήσετε την πραγματική μεταβίβαση.

Σε αντίθεση με τους desktop υπολογιστές, που έχουν πολύ χώρο στην οθόνη για την εμφάνιση κουμπιών ή μενού, μια διαφορετική προσέγγιση είναι απαραίτητη για φορητές συσκευές. Ορισμένες συσκευές παρέχουν τα λεγόμενα soft buttons, τα οποία είναι κουμπιά χωρίς καθορισμένη λειτουργικότητα που έχουν εκχωρηθεί δυναμικά ανάλογα με το περιεχόμενο της εφαρμογής. Ο αριθμός των soft buttons μπορεί να ποικίλει, εφόσον αυτά είναι διαθέσιμα. Άλλες φορητές συσκευές δεν έχουν καν χώρο για soft buttons, αλλά παρέχουν scrolling menus. Το MIDP πρέπει να είναι ανεξάρτητο από τη συγκεκριμένη συσκευή και να παρέχει έναν μηχανισμό που να είναι κατάλληλος για όλες τις συσκευές, ανεξάρτητα από την διαθεσιμότητα και τον αριθμό των soft buttons. Επομένως, το πακέτο lcdui δεν παρέχει κουμπιά ή μενού, αλλά μια αφηρημένη έννοια (abstraction) που ονομάζεται Command.

Τα Commands μπορούν να προστεθούν σε όλες τις κλάσεις που προέρχονται από την κλάση Displayable. Αυτές οι κλάσεις είναι η Screen και οι υποκλάσεις της, όπως οι Form, List, και TextBox για το υψηλού επιπέδου API και η Canvas για το χαμηλού επιπέδου API.

Καμία πληροφορία για την θέση ή την διάταξη δεν μεταβιβάζεται στο Command. Η ίδια η κλάση Displayable είναι πλήρως υπεύθυνη για να διευθετεί τα ορατά στοιχεία που αντιστοιχούν σε CommandS σε μια συγκεκριμένη συσκευή. Η μόνη πληροφορία διάταξης και απεικόνισης που μπορεί να ανατεθεί σε ένα Command, εκτός από την εντολή ετικέτας, αποτελείται από έναν τύπο και μια προτεραιότητα. Η προτεραιότητα επιτρέπει στη συσκευή να αποφασίσει ποιες εντολές θα εμφανίζονται ως soft buttons, εάν ο αριθμός των εντολών υπερβαίνει τον αριθμό των διαθέσιμων soft buttons. Για τα επιπλέον Commands που δεν εμφανίζονται ως soft buttons, ένα ξεχωριστό μενού δημιουργείται αυτόματα. Η πληροφορία για τον τύπο είναι μια πρόσθετη ένδειξη για τη συσκευή σχετικά με το πώς θα εμφανιστεί η εντολή. Για παράδειγμα, εάν η εντολή Exit αποδίδεται πάντα στο αριστερό soft button στις μητρικές εφαρμογές ενός συγκεκριμένου τύπου συσκευής, η MIDP υλοποίηση είναι σε θέση να κάνει την ίδια εκχώρηση (της εντολής Exit στο αριστερό soft button). Επομένως, μπορεί να επιτευχθεί ένα σταθερό “look and feel” για την συσκευή.

Οι διαθέσιμες σταθερές του τύπου της εντολής αναφέρονται στον παρακάτω πίνακα:

Πίνακας 10: Σταθερές του τύπου του Command [13]

Σταθερά	**Τιμή**
Command.BACK	Χρησιμοποιείται για εντολές πλοήγησης που χρησιμοποιούνται για να επιστρέψουν τον χρήστη στο προηγούμενο Screen
Command.CANCEL	Απαιτείται για να γνωστοποιήσει στην οθόνη ότι μια ακύρωση συνέβη
Command.EXIT	Χρησιμοποιείται για να ορίσει ένα Command για έξοδο από την εφαρμογή
Command.HELP	Περνιέται όταν η εφαρμογή ζητά να προβληθεί μια βοήθεια επί της οθόνης
Command.ITEM	Ένας τύπος εντολής που λέει στην εφαρμογή ότι προσαρτάται σε ένα συγκεκριμένο αντικείμενο στην οθόνη
Command.OK	Απαιτείται για να γνωστοποιήσει στην οθόνη ότι μια επιβεβαίωση συνέβη
Command.SCREEN	Καθορίζει ένα Command της εφαρμογής που αφορά την απεικόνιση της οθόνης
Command.STOP	Διακόπτει μια διαδικασία που εκτελείται εκείνη τη στιγμή

Ο δομητής της Command λαμβάνει την ετικέτα, τον τύπο της εντολής και την προτεραιότητα ως είσοδο. Η Command κλάση παρέχει read () μεθόδους για όλα αυτά τα πεδία, αλλά δεν είναι δυνατή η αλλαγή των παραμέτρων μετά την δημιουργία. Χρησιμοποιώντας την μέθοδο addCommand (), οι εντολές μπορούν να προστεθούν σε μια Form ή οποιαδήποτε άλλη υποκλάση της Displayable.

Όπως και στην περίπτωση λήψης αλλαγών κατάστασης των UI widgets, το MIDP χρησιμοποιεί ένα μοντέλο ακροατή για την ανίχνευση ενεργειών εντολών. Για το σκοπό αυτό, το lcdui πακέτο περιλαμβάνει τη διεπαφή CommandListener. Η CommandListener μπορεί να καταχωρηθεί σε οποιαδήποτε Displayable κλάση χρησιμοποιώντας τη μέθοδο setCommandListener. Μετά την καταχώρηση, η commandAction () μέθοδος του CommandListener καλείται κάθε φορά που ο

χρήστης θέτει μια Command. Σε αντίθεση με το AWT, μόνο ένας ακροατής επιτρέπεται για κάθε Displayable κλάση. Για την επανάκληση της commandAction () μεθόδου παρέχονται ως παραμέτροι η Displayable κλάση, όπου τέθηκε η εντολή, και το αντίστοιχο Command αντικείμενο.

Με αυτές τις πληροφορίες, μπορείτε να επεκτείνετε την TeleTransfer εφαρμογή σας με τις επιθυμητές CommandS. Αλλά πριν πάμε σε πραγματική υλοποίηση εντολών, θα πρέπει να προσθέσετε κάποια αντίστοιχη λειτουργικότητα. Θα προσθέσετε τρεις εντολές: Μια εντολή Send, μια εντολή Clear, και μια εντολή Exit. Για την Clear, απλά προσθέστε μια μέθοδο καθορίζοντας το περιεχόμενο των πεδίων της φόρμας σας σε κενά Strings:

```
public void clear () {
receiverName.setString ("");
receiverAccount.setString ("");
amountWhole.setString ("");
amountFraction.setString ("");
 }
```

Η εντολή Send είναι λίγο πιο δύσκολη επειδή δεν έχετε ακόμη το υπόβαθρο για να υποβάλετε στοιχεία μέσω του δικτύου. Έτσι απλά απεικονίζετε το περιεχόμενο που είναι να διαβιβαστεί σε μια οθόνη alert ως προσωρινή αντικατάσταση:

```
public void send () {
 Alert alert = new Alert ("Send");
 alert.setString ("transfer " + amountWhole.getString ()
     + "." + amountFraction.getString () + " "
     + amountWhole.getLabel ()
     + "\nto Acc#" + receiverAccount.getString ()
     + "\nof " + receiverName.getString ());
 alert.setTimeout (2000);
 display.setCurrent (alert);
 clear ();
}
```

Για την έξοδο από την εφαρμογή, το MIDlet παρέχει ήδη την notifyDestroyed() μέθοδο, ώστε να μην χρειάζεται να προσθέσετε τίποτα.

Τώρα που έχετε υλοποιήσατε την αντίστοιχη λειτουργικότητα, το επόμενο βήμα είναι να προσθέσετε τα πραγματικά Command αντικείμενα στην κλάση της εφαρμογής σας:

```
static final Command sendCommand =
        new Command ("Send", Command.SCREEN, 1);
static final Command clearCommand =
        new Command ("Clear", Command.SCREEN, 2);
static final Command exitCommand =
        new Command ("Exit", Command.EXIT, 2);
```

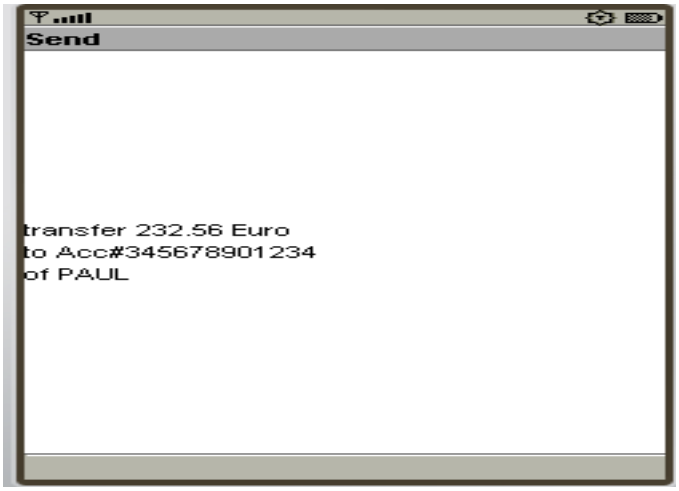
Για να επιτρέψετε στο MIDlet να λάβει ενέργειες εντολών, θα πρέπει να υλοποιήσετε την διεπαφή CommandListener, και την αντίστοιχη commandAction () μέθοδο. Ανάλογα με την εντολή που λαμβάνετε, καλείτε την Send (), Clear (), ή notifyDestroyed ():

```
public class TeleTransfer extends MIDlet implements ItemStateListener,
CommandListener {
public void commandAction (Command c, Displayable d) {
 if (c == exitCommand) {
  notifyDestroyed(); }
 else if (c == sendCommand) {
  send ();  }
 else if (c == clearCommand) {
  clear (); }
  }
```

Με αυτές τις τροποποιήσεις, το TeleTransfer MIDlet σας είναι σε θέση να χειριστεί τις επιθυμητές εντολές. Πρέπει ακόμα να προσθέσετε τα CommandS στην Form, καθώς και να καταχωρήσετε στο TeleTransfer MIDlet σας τον CommandListener προκειμένου να λαμβάνει πράγματι τις εντολές:

```
public TeleTransfer () {
 ...
 mainForm.addCommand (sendCommand);
 mainForm.addCommand (clearCommand);
 mainForm.addCommand (exitCommand);
 mainForm.setCommandListener (this);
}
```

Το παρακάτω σχήμα δείχνει το Send Alert της νέας έκδοσης της TeleTransfer εφαρμογής σας.



Εικόνα 22: Το TeleTransfer MIDlet που δείχνει μια ειδοποίηση απεικονίζοντας συνοπτικά πληροφορίες μεταφοράς πριν την αποστολή.

3.5 Περαιτέρω Screen κλάσεις: List και TextBox

Η τρέχουσα έκδοση του TeleTransfer MIDlet δείχνει τον τρόπο χρήσης της Form και των αντίστοιχων στοιχείων που διατίθενται στο lcdui πακέτο. Η εφαρμογή αποτελείται από μια κύρια φόρμα που κρατά όλα τα widgets της εφαρμογής. Ωστόσο, η κύρια φόρμα σας είναι αρκετά μεγάλη τώρα, κι έτσι προκύπτει το ερώτημα πώς μπορεί να βελτιωθεί η χρηστικότητα της εφαρμογής. Παρακάτω λοιπόν δείχνουμε τον τρόπο δόμησης της διεπαφής χρήστη, χρησιμοποιώντας πολλαπλά screens και εισάγοντας τις κλάσεις List και TextBox.

3.5.1 Η κλάση List

Ένας τρόπος να «καθαρίσει» η διεπαφή χρήστη είναι να απομακρύνετε την επιλογή νομίσματος σε μια ξεχωριστή οθόνη. Χρειάζεται πολύς χώρος και μπορεί να χρειαστεί ακόμη περισσότερος αν προστεθούν επιπλέον επιλογές. Επίσης, μπορείτε να υποθέσετε ότι το νόμισμα δεν αλλάζει πολύ συχνά.

Μπορείτε να δημιουργήσετε μια νέα Form και απλά να μετακινήσετε εκεί το ChoiceGroup. Ωστόσο, το lcdui πακέτο παρέχει μια ειδική List κλάση που κληρονομείται από την κλάση Screen για το σκοπό αυτό. Το πλεονέκτημα της κλάσης List είναι ότι παρέχει τον τύπο IMPLICIT που αναφέρθηκε ήδη στο τμήμα "Επιλέγοντας Στοιχεία Χρησιμοποιώντας ChoiceGroupS". Χρησιμοποιώντας τον τύπο IMPLICIT, η εφαρμογή λαμβάνει άμεση ειδοποίηση όταν ένα στοιχείο επιλέγεται. Κάθε φορά που ένα στοιχείο στη List είναι επιλεγμένο, ένα Command του τύπου List.SELECT_COMMAND έχει τεθεί. Όπως και στην ChoiceGroup, τα στοιχεία αποτελούνται από Strings και προαιρετικά από Images.

Για την αρχικοποίηση του List, τα lcdui πακέτα προσφέρουν δομητές, οι οποίοι λειτουργούν όπως οι ChoiceGroup δομητές. Ο πρώτος δημιουργεί μια κενή List με δοσμένα μόνο τίτλο και είδος. Ο δεύτερος παίρνει τον τίτλο, το είδος, έναν πίνακα από Strings των αρχικών στοιχείων της List, και έναν προαιρετικό πίνακα από Images για κάθε στοιχείο της List. Κατά την υλοποίηση της TeleTransfer εφαρμογής, υλοποιείτε μια νέα κλάση CurrencyList που επεκτείνει την List και που θα χρησιμοποιηθεί για την επιλογή νομίσματος. Δεδομένου ότι θα χρησιμοποιήσετε τον τύπο IMPLICIT, θα πρέπει να υλοποιήσετε έναν command listener. Έτσι μπορείτε να προσθέσετε την αντίστοιχη δήλωση:

```
public class CurrencyList extends List implements CommandListener {
```

Για να ρυθμίσετε τις ετικέτες των TextFields της κύριας φόρμας σύμφωνα με το δείκτη του επιλεγμένου στοιχείου στην CurrencyList, θα δημιουργήσετε δύο String πίνακες, τους CURRENCY_NAMES και CURRENCY_FRACTIONS:

```
static final String [] CURRENCY_NAMES = {"Dollar", "Euro", "Yen"};
static final String [] CURRENCY_FRACTIONS = {"Cent", "Cent", "Sen"};
```

Για να ρυθμίσετε τις ετικέτες των TextFields της κύριας φόρμας για το ακέραιο και το κλασματικό μέρος του ποσού ανάλογα με το επιλεγμένο νόμισμα στην CurrencyList, χρειάζεστε μια αναφορά πίσω στο κύριο TeleTransfer MIDlet. Για το λόγο αυτό, αποθηκεύετε την αναφορά TeleTransfer σε μια μεταβλητή που θα ονομάζεται teleTransfer. Η αναφορά θα βρίσκεται στον δομητή της CurrencyList σας:

```
TeleTransfer teleTransfer;
```

Στον δομητή, μπορείτε επίσης να προσθέσετε εικόνες των συμβόλων των νομισμάτων στη λίστα. Πρέπει να τις φορτώσετε, αλλά η κλήση του super δομητή πρέπει να είναι η πρώτη δήλωση στον δομητή. Έτσι καλείτε τον δομητή της super κλάσης προσδιορίζοντας μόνο τον τίτλο και το είδος. Στη συνέχεια, δημιουργείτε τα Images που απαιτούνται για κάθε στοιχείο της λίστας, τα οποία αποθηκεύονται στο JAR αρχείο του MIDlet. Επίσης καλείτε την setCommandListener () για να καταχωρήσετε την λίστα νομισμάτων για το χειρισμό των εντολών που τίθενται:

```
public CurrencyList (TeleTransfer teletransfer) {
    super ("Select Currency", Choice.IMPLICIT);
    this.teleTransfer = teletransfer;

    try {
     append ("USD", Image.createImage ("/Dollar.png"));
     append ("EUR", Image.createImage ("/Euro.png"));
     append ("JPY", Image.createImage ("/Yen.png"));
    }
    catch (java.io.IOException x) {
     throw new RuntimeException ("Images not found");
    }
    setCommandListener (this);
}
```

Το τελικό βήμα για τη δημιουργία της CurrencyList είναι η υλοποίηση της commandAction () μεθόδου της CommandListener διεπαφής. Όπως ήδη

γνωρίζετε, μια List τύπου IMPLICIT θέτει μια List.SELECT_COMMAND στον καταχωρημένο CommandListener κάθε φορά που ένα νέο στοιχείο επιλέγεται για να δείξει την αλλαγή επιλογής. Σε περίπτωση αλλαγής επιλογής, αλλάζετε τις ετικέτες από τα TextFields της κύριας φόρμας. Οι πραγματικές ετικέτες λαμβάνονται από τους CURRENCY_NAMES και CURRENCY_FRACTIONS String πίνακες. Χρησιμοποιώντας την αναφορά teleTransfer, μπορείτε να αποκτήσετε πρόσβαση στα TextFields. Τέλος καλείτε την teleTransfer.back() μέθοδο, η οποία θέτει την οθόνη πίσω στην κύρια φόρμα (η back () μέθοδος θα δοθεί στο τέλος του παρόντος τμήματος):

```
public void commandAction (Command c, Displayable d) {
      if (c == List.SELECT_COMMAND) {
       teleTransfer.amountWhole.setLabel
        (CURRENCY_NAMES [getSelectedIndex ()]);
       teleTransfer.amountFraction.setLabel
        (CURRENCY_FRACTIONS [getSelectedIndex ()]);
       teleTransfer.back ();
      }
     }
    }
```

Το παρακάτω σχήμα δείχνει τα Images των νομισμάτων και τις συντομογραφίες τους στην CurrencyList.

Εικόνα 23: Έξοδος του TeleTransfer για επιλογή νομίσματος σε νέο Screen με χρήση Images

3.5.2 Η TextBox κλάση

Εκτός από τις Alert, List, και Form, υπάρχει μια μόνο ακόμη υποκλάση της Screen: η TextBox. Η TextBox επιτρέπει στο χρήστη να εισάγει κείμενο πολλαπλών γραμμών σε μια ξεχωριστή οθόνη. Οι παράμετροι του δομητή και οι σταθερές περιορισμών είναι πανομοιότυπες με εκείνες του TextField.

Μπορείτε λοιπόν να προσθέσετε μια νέα οθόνη που θα επιτρέπει στο χρήστη να εισάγει έναν λόγο μεταφοράς, αν θέλετε. Παρόμοια με την CurrencyList, θα υλοποιήσετε μια νέα κλάση για τον χειρισμό των εντολών που σχετίζονται με τη νέα οθόνη. Ωστόσο, αυτή τη φορά προέρχεται από το TextBox. Και πάλι, θα υλοποιήσετε το CommandListener interface:

```
public class TransferReason extends TextBox implements
CommandListener {
```

Στο TextBox, παρέχετε δύο εντολές, την okCommand για επιβεβαίωση του εισαγόμενου κειμένου και την clearCommand για την εκκαθάριση του:

```
static final Command okCommand = new Command
("OK", Command.BACK, 1);

static final Command clearCommand = new Command
 ("Clear",Command.SCREEN,  2);
```

Και πάλι, αποθηκεύετε μια αναφορά στο TeleTransfer MIDlet μέσα στο TransferReason TextBox:

```
TeleTransfer teleTransfer;
```

Ο δομητής παίρνει την αναφορά στο TeleTransfer MIDlet και την αποθηκεύει στη μεταβλητή που δηλώθηκε προηγουμένως. Επίσης, προσθέτετε τις εντολές και καλείτε την setCommandListener():

```
public TransferReason (TeleTransfer teleTransfer) {
    super ("Transfer Reason", "", 50, TextField.ANY);
    this.teleTransfer = teleTransfer;

    addCommand (okCommand);
    addCommand (clearCommand);
    setCommandListener (this);
   }
```

Η CommandAction () υλοποίησή σας καθαρίζει το κείμενο ή επιστρέφει στην κύρια οθόνη, ανάλογα με την εντολή που επιλέχτηκε:

```
public void commandAction (Command c, Displayable d) {
    if (c == clearCommand) {
     setString ("");
    }
    else if (c == okCommand) {
     teleTransfer.back ();       } } }
```

Το παρακάτω σχήμα δείχνει το TransferReason TextBox.

Εικόνα 24: Το TransferReasonTextBox του TeleTransfer με δείγμα κειμένου για τον λόγο μεταφοράς

3.6 Το TeleTransfer με Πολλαπλές Οθόνες

Τώρα έχετε δημιουργήσει δύο επιπλέον οθόνες, αλλά θα πρέπει να τις εντάξετε στην κύρια εφαρμογή σας. Για να γίνει αυτό, πρέπει να αλλάξετε την TeleTransfer εφαρμογή σας με κάποιο τρόπο. Αφού το ChoiceGroup της TeleTransfer για την επιλογή του νομίσματος έχει αντικατασταθεί από την CurrencyList, δεν χρειάζεστε πλέον το ItemStateListener για την ανίχνευση αλλαγών στα items. Έτσι καταργείτε τον ακροατή, αλλά και την αντίστοιχη μέθοδο επανάκλησης itemStateChanged (). Για να εμφανίσετε τις δύο νέες οθόνες CurrencyList και TransferReason, θα υλοποιήσετε τις δύο εντολές currencyCommand και reasonCommand. Οι νέες εντολές προστίθενται στον δομητή του MIDlet χρησιμοποιώντας την addCommand () μέθοδο. Τέλος, προσθέτετε την back () μέθοδο στην εφαρμογή TeleTransfer. Η μέθοδος αυτή καλείται από τις νέες οθόνες για να επιστρέψετε στην κύρια φόρμα. Η commandAction () μέθοδος επεκτείνεται για να χειριστεί τις νέες εντολές, κατά την απεικόνιση των νέων Screens. Παρακάτω παρουσιάζεται ο πλήρης πηγαίος κώδικας της τελικής έκδοσης της εφαρμογής TeleTransfer.

TeleTransfer.java

Ο ολοκληρωμένος πηγαίος κώδικας του TeeTransfer Παραδείγματος

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

class CurrencyList extends List implements CommandListener {

    TeleTransfer teleTransfer;
    static final String [] CURRENCY_NAMES = {"Dollar", "Euro", "Yen"};
    static final String [] CURRENCY_FRACTIONS = {"Cent", "Cent", "Sen"};

    public CurrencyList (TeleTransfer teletransfer) {
     super ("Select Currency", Choice.IMPLICIT);

     this.teleTransfer = teletransfer;
```

```
        try {
         append ("USD", Image.createImage ("/Dollar.png"));
         append ("EUR", Image.createImage ("/Euro.png"));
         append ("JPY", Image.createImage ("/Yen.png"));
        }
        catch (java.io.IOException x) {
         throw new RuntimeException ("Images not found");
        }
        setCommandListener (this);
       }


      public void commandAction (Command c, Displayable d) {
      if (c == List.SELECT_COMMAND) {
      teleTransfer.amountWhole.setLabel
    (CURRENCY_NAMES[getSelectedIndex ()]);
        teleTransfer.amountFraction.setLabel
        (CURRENCY_FRACTIONS [getSelectedIndex ()]);
        teleTransfer.back ();
        } } }

class TransferReason extends TextBox implements CommandListener {

    static final Command okCommand = new Command ("OK",
    Command.BACK, 1);
     static final Command clearCommand = new Command ("Clear",
    Command.SCREEN, 2);

      TeleTransfer teleTransfer;

      public TransferReason (TeleTransfer teleTransfer) {
       super ("Transfer Reason", "", 50, TextField.ANY);
       this.teleTransfer = teleTransfer;
```

```
        addCommand (okCommand);
        addCommand (clearCommand);
        setCommandListener (this);
       }

       public void commandAction (Command c, Displayable d) {
        if (c == clearCommand) {
         setString ("");
        }
        else if (c == okCommand) {
         teleTransfer.back ();
        }
       }
      }
```

```
public class TeleTransfer extends MIDlet implements CommandListener {

      static final Command sendCommand =
            new Command ("Send", Command.SCREEN, 1);
     static final Command clearCommand =
            new Command ("Clear", Command.SCREEN, 2);
     static final Command exitCommand =
            new Command ("Exit", Command.EXIT, 2);

      static final Command currencyCommand = new Command
       ("Currency", Command.SCREEN, 3);
      static final Command reasonCommand = new Command
       ("Reason", Command.SCREEN, 4);

      Form mainForm = new Form ("TeleTransfer");

      TextField receiverName = new TextField
```

```
("Receiver Name", "", 20, TextField.ANY);
TextField receiverAccount = new TextField
("Receiver Account#", "", 8, TextField.NUMERIC);
TextField amountWhole = new TextField ("Dollar", "", 6,
        TextField.NUMERIC);
TextField amountFraction = new TextField
       ("Cent", "", 2, TextField.NUMERIC);

CurrencyList currencyList = new CurrencyList (this);
TransferReason transferReason = new TransferReason (this);
Display display;

public TeleTransfer () {
 mainForm.append (receiverName);
 mainForm.append (receiverAccount);
 mainForm.append (amountWhole);
 mainForm.append (amountFraction);

 mainForm.addCommand (currencyCommand);
 mainForm.addCommand (reasonCommand);
 mainForm.addCommand (clearCommand);
 mainForm.addCommand (sendCommand);
 mainForm.addCommand (exitCommand);
 mainForm.setCommandListener (this);
}

public void startApp () {
 display = Display.getDisplay (this);
 display.setCurrent (mainForm);
}

public void clear () {
 receiverName.setString ("");
 receiverAccount.setString ("");
 amountWhole.setString ("");
```

```
 amountFraction.setString ("");
 transferReason.setString ("");
}

public void send () {
 Alert alert = new Alert ("Send");
 alert.setString ("transfer " + amountWhole.getString ()
     + "." + amountFraction.getString ()
     + " " + amountWhole.getLabel ()
     + "\nto Acc#" + receiverAccount.getString ()
     + "\nof " + receiverName.getString ());
 alert.setTimeout (2000);
 display.setCurrent (alert);
 clear ();
}

public void pauseApp () {
}

public void destroyApp (boolean unconditional) {
}

public void back () {
 display.setCurrent (mainForm);
}

public void commandAction (Command c, Displayable d) {
 if (c == exitCommand) {
  notifyDestroyed();
 }
 else if (c == sendCommand) {
  send ();
 }
 else if (c == clearCommand) {
  clear ();
```

```
        }
        else if (c == currencyCommand) {
         display.setCurrent (currencyList);
        }
        else if (c == reasonCommand) {
         display.setCurrent (transferReason);
        }
       }
      }
```

Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό αναπτύχθηκε διεξοδικά το υψηλού επιπέδου API ,που είναι ένα API διεπαφής χρήστη το οποίο παρέχει στοιχεία εισόδου όπως Textfields, Choices και άλλα. Περιγράφηκαν οι MIDP GUI κλάσεις του και τα βασικότερα χαρακτηριστικά τους. Τέλος , είδατε τον χειρισμό γεγονότων αλλαγής κατάστασης ενός Item μέσω του ItemStateListener και την ανίχνευση ενεργειών εντολών μέσω του CommandListener.

Ερωτήσεις

1.Γράψτε τις 2 εντολές που είναι απαραίτητες για την προσπέλαση και εμφάνιση της επιθυμητής απεικόνισης της οθόνης.

2.Ποιά η λειτουργία της Alert;

3.Αναφέρετε τι κάνει η Form.

4.Ποιές είναι οι υποκλάσεις της κλάσης Item και τι αυτή παρέχει ως κοινή ιδιότητα όλων των υποκλάσεων της;

5.Περιγράψτε τη χρήση των παραπάνω υποκλάσεων.

6.Ποιά κομμάτια κώδικα πρέπει να προσθέσετε στον κώδικά σας, ώστε να μπορεί να χειριστεί γεγονότα αλλαγής κατάστασης ενός Item;

7.Τι παραμέτρους λαμβάνει ο δομητής της Command;

8.Ποιές είναι οι σταθερές του τύπου ενός Command;

9. Ποιά κομμάτια κώδικα πρέπει να προσθέσετε στον κώδικά σας, ώστε να μπορεί να λάβει και να χειριστεί ενέργειες εντολών;

10.Ποιό το πλεονέκτημα της κλάσης List;

11.Ποιά η λειτουργία ενός TextBox;

ΚΕΦΑΛΑΙΟ 4

4. Low-Level API

Σε αντίθεση με το υψηλού επιπέδου API, το χαμηλού επιπέδου API επιτρέπει τον πλήρη έλεγχο της απεικόνισης του MID σε επίπεδο pixel. Για το σκοπό αυτό, το lcdui πακέτο περιέχει μια κλάση που ονομάζεται Canvas. Η Canvas από μόνη της δεν παρέχει καμία μέθοδο σχεδίασης, αλλά παρέχει μια μέθοδο επανάκλησης paint () παρόμοια με την paint () μέθοδο των AWT components. Κάθε φορά που ο διαχειριστής του προγράμματος κρίνει ότι είναι απαραίτητο να σχεδιάσει το περιεχόμενο της οθόνης, καλείται η μέθοδος paint () του Canvas. Η μόνη παράμετρος της μεθόδου paint () είναι ένα αντικείμενο τύπου Graphics. Σε αντίθεση με τις lcdui υψηλού επιπέδου κλάσεις, υπάρχουν πολλές ομοιότητες στο AWT με το χαμηλού επιπέδου API.

Το αντικείμενο Graphics παρέχει όλες τις μεθόδους που απαιτούνται για την σχεδίαση του περιεχομένου της οθόνης, όπως η drawLine () για την σχεδίαση γραμμών, η fillRect () για την σχεδίαση μιας γεμάτης ορθογώνιας περιοχής ή η drawstring () για την σχεδίαση συμβολοσειρών.

Σε αντίθεση με το AWT, το lcdui δεν σας επιτρέπει να αναμείξετε υψηλού επιπέδου και χαμηλού επιπέδου γραφικά. Δεν είναι δυνατόν να απεικονίσετε υψηλού επιπέδου και χαμηλού επιπέδου στοιχεία στην οθόνη ταυτόχρονα.

Ο διαχειριστής προγράμματος γνωρίζει ότι πρέπει να καλέσει την paint () μέθοδο του Canvas, όταν το στιγμιότυπο του Canvas εμφανίζεται στην οθόνη. Ωστόσο, ένα repaint μπορεί επίσης να ενεργοποιηθεί από την εφαρμογή ανά πάσα στιγμή. Καλώντας την repaint () μέθοδο του Canvas, το σύστημα ειδοποιείται ότι ένα repaint είναι απαραίτητο, και θα καλέσει την paint () μέθοδο. Η κλήση της paint () μεθόδου δεν εκτελείται αμέσως. Μπορεί να καθυστερήσει μέχρι ο έλεγχος ροής να επιστρέψει από την τρέχουσα μέθοδο χειρισμού γεγονότος. Το σύστημα μπορεί επίσης να συλλέξει διάφορες repaint αιτήσεις πριν η paint () κληθεί στην πραγματικότητα. Αυτή η καθυστέρηση κανονικά δεν αποτελεί πρόβλημα, αλλά όταν κάνετε animation, ο ασφαλέστερος τρόπος για να

προκαλέσετε repaints είναι η χρήση της Display.callSerially () μεθόδου ή να ζητήσετε ένα repaint από ένα ξεχωριστό thread ή TimerTask. Εναλλακτικά, η εφαρμογή μπορεί να προκαλέσει ένα άμεσο repaint καλώντας την serviceRepaints ().

Η κλάση Canvas επίσης παρέχει ορισμένες input μεθόδους που καλούνται όταν ο χρήστης πατήσει ή απελευθερώνει ένα πλήκτρο ή αγγίζει την οθόνη με τη γραφίδα (αν υποστηρίζεται από τη συσκευή).

4.1 Βασική Σχεδίαση

Πριν προχωρήσουμε σε λεπτομέρειες σχετικά με την εισαγωγή από τον χρήστη ή το animation, θα ξεκινήσουμε με ένα μικρό παράδειγμα σχεδίασης που δείχνει τη συγκεκριμένη χρήση των κλάσεων Canvas και Graphics.

Στο παρακάτω παράδειγμα καθαρίζεται η οθόνη με τον καθορισμό του χρώματος σε λευκό και γεμίζοντας ένα ορθογώνιο στο μέγεθος της οθόνης, το οποίο προσδιορίζεται καλώντας τις getWidth () και getHeight (). Στη συνέχεια σχεδιάζεται μια γραμμή από τις συντεταγμένες (0,0) έως (100,200). Τέλος, σχεδιάζεται ένα ορθογώνιο ξεκινώντας από το (20,30), των 30 pixels πλάτος και 20 pixels ύψος:

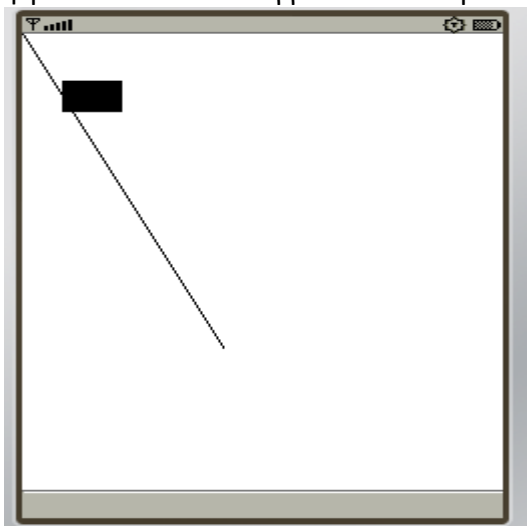
```
import javax.microedition.lcdui.*;

    class DrawingDemoCanvas extends Canvas {

     public void paint (Graphics g) {
      g.setGrayScale (255);
      g.fillRect (0, 0, getWidth (), getHeight ());

      g.setGrayScale (0);
      g.drawLine (0, 0, 100, 200);
      g.fillRect (20, 30, 30, 20);            } }
```

Όπως μπορείτε να δείτε στον κώδικα του παραδείγματος, δημιουργείτε μια DrawingDemoCanvas κλάση, προκειμένου να υλοποιήσετε την μέθοδο paint (). Στην πραγματικότητα, δεν είναι δυνατό να σχεδιάσετε γραφικά χωρίς να δημιουργηθεί μια νέα κλάση και να υλοποιηθεί η μέθοδος paint ().

Προκειμένου να δείτε την Canvas υλοποίησή σας να λειτουργεί, χρειάζεστε ένα αντίστοιχο MIDlet. Εδώ είναι ο κώδικας που λείπει:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DrawingDemo extends MIDlet {

 public void startApp () {
  Display.getDisplay (this).setCurrent (new DrawingDemoCanvas ());
     }

 public void pauseApp () {}

 public void destroyApp (boolean forced) {}
    }
```

Τώρα μπορείτε να ξεκινήσετε το DrawingDemo MIDlet σας. Ανάλογα με το μέγεθος της οθόνης της συσκευής, θα δημιουργήσει έξοδο παρόμοια με το σχήμα που φαίνεται παρακάτω. Στα περισσότερα από τα επόμενα παραδείγματα, θα παραλείψετε το MIDlet δεδομένου ότι είναι ουσιαστικά το ίδιο με το παραπάνω, εκτός από το ότι το όνομα της Canvas κλάσης σας θα είναι διαφορετικό.



Εικόνα 25: Έξοδος του DrawingDemo παραδείγματος

Στο παράδειγμα, η οθόνη καθαρίζεται πριν από την σχεδίαση, διότι το σύστημα βασίζεται στην μέθοδο paint () για το γέμισμα κάθε pixel της περιοχής σχεδίασης με μια έγκυρη τιμή. Δεν διαγράφεται το προηγούμενο περιεχόμενο της οθόνης αυτόματα, διότι αυτό μπορεί να προκαλέσει «τρεμούλιασμα» των animations. Η εφαρμογή δεν μπορεί να κάνει υποθέσεις σχετικά με το περιεχόμενο της οθόνης πριν η paint () να κληθεί. Η οθόνη μπορεί να είναι γεμάτη με το περιεχόμενο που σχεδιάστηκε κατά την τελευταία κλήση της paint (), αλλά μπορεί επίσης να περιέχει κάποιο μήνυμα ειδοποίησης που απέμεινε από μια εισερχόμενη κλήση, για παράδειγμα.

Σχεδίαση Style και Color

Κατά την υλοποίηση της DrawingDemoCanvas, μπορείτε να βρείτε δύο κλήσεις της μεθόδου setGrayScale (). Η setGrayScale () μέθοδος καθορίζει την τιμή της grayscale για τις μετέπειτα εργασίες σχεδίασης. Οι έγκυρες αποχρώσεις της grayscale κυμαίνονται από 0 έως 255, όπου 0 σημαίνει μαύρο και 255 λευκό. Δεν μπορούν όλες οι πιθανές τιμές να καταστήσουν πράγματι διαφορετικές γκρι αποχρώσεις στην οθόνη. Αν η συσκευή παρέχει λιγότερες από 256 αποχρώσεις του γκρι, θα επιλεχθεί η καλύτερη αρμόζουσα τιμή που υποστηρίζεται από τη συσκευή. Στο παράδειγμα, η τιμή ορίζεται αρχικά στη λευκή απόχρωση, και η οθόνη καθαρίζεται από την κλήση της drawRect (). Στη συνέχεια, το χρώμα ορίζεται σε μαύρο για τις επόμενες εργασίες σχεδίασης.

Η μέθοδος setGrayScale () δεν είναι ο μόνος τρόπος για να επηρεαστεί το χρώμα της σχεδίασης. Το MIDP παρέχει επίσης μια μέθοδο setColor (). Η setColor () μέθοδος έχει τρεις παραμέτρους που «κρατούν» το κόκκινο, το πράσινο και το μπλε ως συστατικά του επιθυμητού χρώματος. Και πάλι, οι τιμές κυμαίνονται από 0 έως 255, όπου 255 σημαίνει πιο ανοιχτή και 0 πιο σκούρη απόχρωση . Αν και οι τρεις παράμετροι έχουν οριστεί με την ίδια τιμή, η κλήση της setColor () είναι ισοδύναμη με την αντίστοιχη κλήση της setGrayScale (). Αν η συσκευή δεν είναι σε θέση να εμφανίσει το επιθυμητό χρώμα, επιλέγει το καλύτερο χρώμα ή απόχρωση του γκρι που υποστηρίζονται από τη συσκευή αυτόματα. Μερικά παραδείγματα παρατίθενται στον παρακάτω πίνακα.

Πίνακας 11: Παράδειγμα Ρυθμίσεων Παραμέτρων Χρώματος [13]

Ρυθμίσεις Παραμέτρων	Τελικό χρώμα
setColor (255, 0, 0)	Κόκκινο
setColor (0, 255, 0)	Πράσινο
setColor (0, 0, 255)	Μπλέ
setColor (128, 0, 0)	Σκούρο κόκκινο
setColor (255, 255, 0)	Κίτρινο
setColor (0, 0, 0)	Μαύρο
setColor (255, 255, 255)	Άσπρο
setColor (128, 128, 128)	50% γκρί

Η μόνη άλλη μέθοδος που επηρεάζει το τρέχον στυλ σχεδίασης είναι η μέθοδος setStrokeStyle (). Η setStrokeStyle () ορίζει το στυλ σχεδίασης γραμμών σε DOTTED ή SOLID. Καθορίζετε το στυλ ορίζοντας την παράμετρο σε μία από τις σταθερές DOTTED ή SOLID, που ορίζονται στην κλάση Graphics.

Όταν εισάγεται η μέθοδος paint (), το αρχικό χρώμα είναι πάντα το μαύρο και το στυλ γραμμής είναι SOLID.

4.2 Απλές Μέθοδοι Σχεδίασης

Στο παράδειγμα έχετε ήδη δει τις μεθόδους fillRect () και drawLine (). Ο παρακάτω πίνακας δείχνει όλα τα πρωταρχικά σχέδια που περιέχονται στην

κλάση Graphics. Όλες οι ενέργειες σχεδίασης όπου το όνομα της μεθόδου αρχίζει με το draw, εκτός από τις drawstring () και drawImage (), επηρεάζονται από το τρέχον χρώμα και στυλ γραμμής. Σχεδιάζουν το περίγραμμα ενός σχήματος, ενώ οι fill μέθοδοι γεμίζουν την αντίστοιχη περιοχή με το τρέχον χρώμα και δεν εξαρτώνται από το στυλ γραμμής.

Πίνακας 12: Σχεδιαστικές Μέθοδοι της Κλάσης Graphics [13]

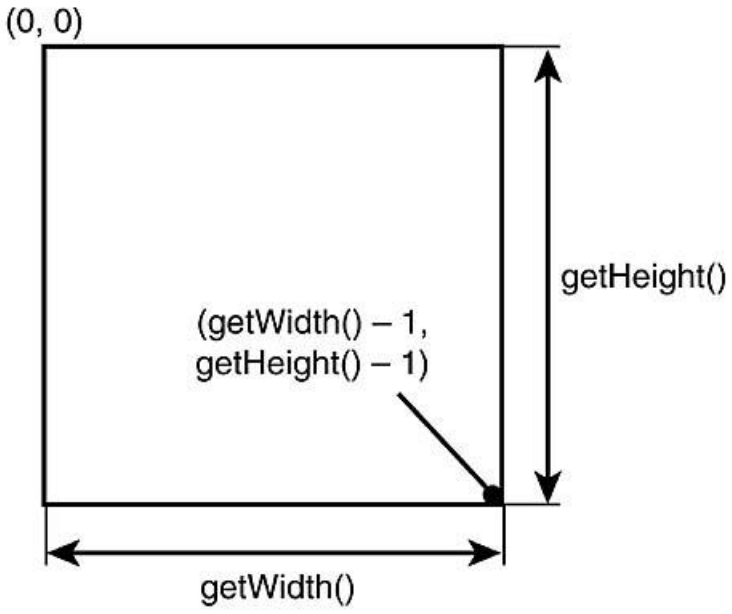
Μέθοδοι	**Στόχος**
drawImage (Image image, int x, int y, int align)	Σχεδιάζει ένα Image.
drawString (String text, int x, int y, int align)	Σχεδιάζει ένα κείμενο συμβολοσειρών στη δοσμένη θέση στο τρέχον χρώμα. (Βλέπε “Κείμενο και γραμματοσειρές”)
drawRect (int x, int y, int w, int h)	Σχεδιάζει ένα άδειο ορθογώνιο με την άνω-αριστερή γωνία στις δοσμένες (x, y) συντεταγμένες, με το δοσμένο πλάτος και ύψος.
drawRoundRect (int x, int y, int w, int h, int r)	Όπως η drawRect (), εκτός από το ότι δίνεται μια επιπλέον ακτίνα για στρογγυλεμένες γωνίες του ορθογωνίου.
drawLine (int x0, int y0, int x1, int y1)	Σχεδιάζει μια γραμμή από το (x0,y0) εως το (x1,y1).
drawArc (int x, int y, int w, int h, int startAng, int arcArc)	Σχεδιάζει το περίγραμμα ενός κυκλικού ή ελλειπτικού τόξου που καλύπτει το συγκεκριμένο ορθογώνιο, χρησιμοποιώντας το τρέχον χρώμα και στυλ. Το τελικό τόξο ξεκινά από την startAng και επεκτείνεται για arcAng μοίρες. Μια θετική τιμή δείχνει αριστερόστροφη περιστροφή, ενώ μια αρνητική τιμή δηλώνει μια δεξιόστροφη περιστροφή.
fillRect (int x, int y, int w, int h)	Λειτουργεί όπως η drawRect (), αλλά γεμίζει την δοσμένη περιοχή με το τρέχον χρώμα.
fillRoundRect (int x, int y, int startAng, int endAng)	Σχετίζεται με την fillRect () όπως η drawRoundRect () με την drawRect ().
fillArc (int x, int y, int w, int h, int startAng, int endAng)	Λειτουργεί όπως η drawArc (), αλλά γεμίζει την αντίστοιχη περιοχή.

Σύστημα συντεταγμένων και Clipping

Στο προηγούμενο παράδειγμα, έχουμε ήδη χρησιμοποιήσει συντεταγμένες οθόνης χωρίς να εξηγήσουμε τι πραγματικά σημαίνουν. Ίσως γνωρίζετε ότι η απεικόνιση στην οθόνη αποτελείται από μικρά στοιχεία εικόνας (pixels). Καθένα

από αυτά τα pixels προσδιορίζεται από τη θέση του στην οθόνη, η οποία μετράται από την επάνω αριστερή γωνία της συσκευής, που είναι η αρχή του συστήματος συντεταγμένων. Το παρακάτω σχήμα δείχνει το lcdui σύστημα συντεταγμένων.

Στην πραγματικότητα, οι συντεταγμένες στην Java δεν προσδιορίζουν το ίδιο το pixel, αλλά τον χώρο μεταξύ δύο pixels, όπου η "πένα σχεδίασης" βρίσκεται στο κάτω δεξί pixel. Για την σχεδίαση γραμμών, αυτό δεν κάνει καμία διαφορά, αλλά για τα ορθογώνια και τα γεμάτα ορθογώνια αυτό έχει ως αποτέλεσμα μία διαφορά ενός pixel σε πλάτος και ύψος: Τα ορθογώνια γίνονται ένα pixel πλατύτερα και ψηλότερα ενώ τα γεμάτα ορθογώνια ένα pixel στενότερα και κοντύτερα, σε σχέση με αυτό που ίσως περιμένετε.



Εικόνα 26: Σύστημα συντεταγμένων [13]

Σε όλες τις μεθόδους σχεδίασης, η πρώτη συντεταγμένη (x) είναι η οριζόντια απόσταση από την αρχή και η δεύτερη συντεταγμένη (y) είναι η κατακόρυφη απόσταση. Θετικές συντεταγμένες σημαίνει μια μετακίνηση προς τα κάτω και προς τα δεξιά. Πολλές μέθοδοι σχεδίασης απαιτούν πρόσθετες παραμέτρους πλάτους και ύψους. Εξαίρεση αποτελεί η drawLine () μέθοδος, η οποία απαιτεί τις απόλυτες συντεταγμένες του σημείου προορισμού.

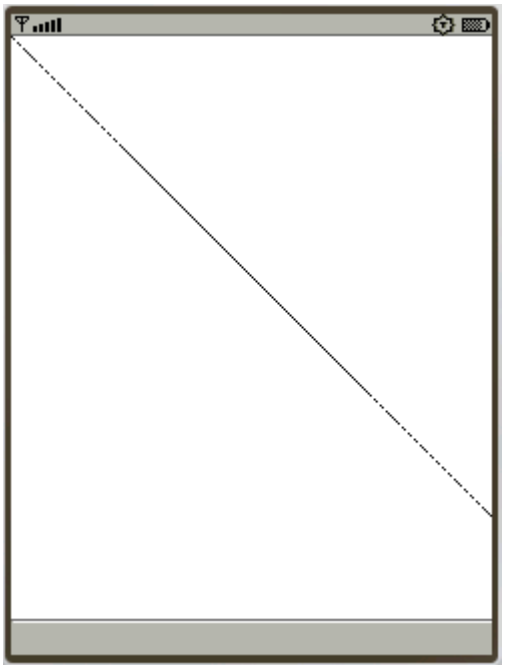
Η αρχή του συστήματος συντεταγμένων μπορεί να αλλάξει χρησιμοποιώντας την μέθοδο translate (). Οι δοσμένες συντεταγμένες προστίθενται αυτόματα σε όλες τις μεταγενέστερες ενέργειες σχεδίασης.

Το πραγματικό μέγεθος της προσβάσιμης περιοχής απεικόνισης μπορεί να τεθεί ως ερώτημα χρησιμοποιώντας τις getWidth () και getHeight () μεθόδους, όπως είδαμε στο πρώτο παράδειγμα. Η περιοχή της οθόνης όπου η σχεδίαση μπορεί να απεικονιστεί, μπορεί να περιοριστεί ακόμη περισσότερο σε μια ορθογώνια περιοχή μέσω της μεθόδου clipRect (). Η σχεδίαση εκτός της περιοχής αυτής (clipping area) δεν θα έχει κανένα αποτέλεσμα.

Το ακόλουθο παράδειγμα δείχνει τα αποτελέσματα της clipRect () μεθόδου. Πρώτον, μια διακεκομμένη γραμμή σχεδιάζεται διαγωνίως στην οθόνη. Μετά, ορίζεται μια clipping area. Τέλος, σχεδιάζεται η ίδια γραμμή με την προηγούμενη χρησιμοποιώντας το στυλ SOLID:

```
import javax.microedition.lcdui.*;

    class ClipDemoCanvas extends Canvas {

     public void paint (Graphics g) {
      g.setGrayScale (255);
      g.fillRect (0, 0, getWidth (), getHeight ());

      int m = Math.min (getWidth (), getHeight ());
      g.setGrayScale (0);

      g.setStrokeStyle (Graphics.DOTTED);
      g.drawLine (0, 0, m, m);

      g.setClip (m / 4, m / 4, m / 2, m / 2);

      g.setStrokeStyle (Graphics.SOLID);
      g.drawLine (0, 0, m, m);
     }
    }
```

Το παρακάτω σχήμα απεικονίζει την τελική εικόνα. Παρόλο που και οι δύο γραμμές έχουν ίδια σημεία έναρξης και λήξης, μόνο το μέρος που καλύπτει την clipping area αντικαθίσταται από μια SOLID γραμμή.



Εικόνα 27: Έξοδος του παραδείγματος ClipRect(): Μόνο το κομμάτι της γραμμής που καλύπτει η clipping area είναι ξανασχεδιασμένο ως SOLID.

Όταν η paint () μέθοδος καλείται από το σύστημα, μια clipping area μπορεί ήδη να έχει οριστεί. Αυτό μπορεί να συμβεί αν η εφαρμογή μόλις ζήτησε τον ανασχεδιασμό σε μια περιορισμένη περιοχή χρησιμοποιώντας την παραμετροποιημένη κλήση της repaint, ή αν η συσκευή μόλις ακύρωσε μια περιορισμένη περιοχή της οθόνης. Για παράδειγμα, αν ένα pop-up παράθυρο διαλόγου που δείχνει μια εισερχόμενη κλήση εμφανίστηκε αλλά δεν κάλυψε ολόκληρη την περιοχή απεικόνισης.

Στην πραγματικότητα, η clipRect () δεν ορίζει μια νέα clipping area, αλλά αντιθέτως συρρικνώνει την τρέχουσα clipping area σε διασταύρωση με το δοσμένο ορθογώνιο. Προκειμένου να μεγενθύνετε την clipping area, χρησιμοποιήστε την μέθοδο setClip ().

Η τρέχουσα clipping area μπορεί να τεθεί ως ερώτημα χρησιμοποιώντας τις getClipX (), getClipY (), getClipWidth (), και getClipHeight () μεθόδους. Όταν η σχεδίαση είναι υπολογιστικά δαπανηρή, οι πληροφορίες αυτές μπορούν να

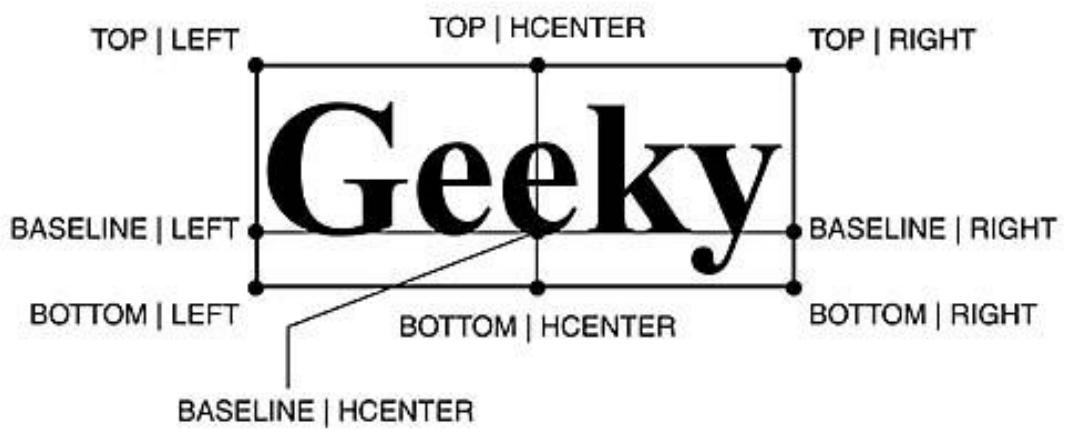
ληφθούν υπόψη προκειμένου να ανασχεδιαστούν μόνο οι περιοχές της οθόνης που χρειάζονται ενημέρωση.

4.3 Κείμενο και γραμματοσειρές

Μέθοδος drawString ()

Για τη σύνταξη κειμένου, το lcdui παρέχει τη μέθοδο drawString (). Εκτός από τη βασική μέθοδο drawString (), αρκετές παραλλαγές σας επιτρέπουν την σύνταξη τμημάτων κειμένου ή μεμονωμένων χαρακτήρων. (Λεπτομέρειες σχετικά με τις επιπρόσθετες μεθόδους μπορούν να βρεθούν στο lcdui API documentation.) Η απλή drawstring () μέθοδος λαμβάνει τέσσερις παραμέτρους: Την συμβολοσειρά που θα εμφανιστεί, την συντεταγμένη x, την συντεταγμένη y και έναν ακέραιο που καθορίζει την οριζόντια και κατακόρυφη στοίχιση του κείμενο. Η παράμετρος στοίχισης σας επιτρέπει να τοποθετήσετε το κείμενο σε σχέση με οποιαδήποτε από τις τέσσερις γωνίες ενός αόρατου πλαισίου γύρω του. Επιπλέον, το κείμενο μπορεί να στοιχηθεί σε μια βάση κειμένου (baseline) και σε οριζόντιο κεντράρισμα. Το sum (+) ή το λογικό or (|) μιας σταθεράς για οριζόντια στοίχιση (LEFT, RIGHT, και HCENTER) και οι σταθερές για την κατακόρυφη στοίχιση (TOP, BOTTOM, και BASELINE) καθορίζουν την πραγματική στοίχιση.

Η παρακάτω εικόνα δείχνει σημεία των έγκυρων συνδυασμών σταθερών.



Εικόνα 28: Έγκυροι συνδυασμοί των σταθερών στοίχισης και τα αντίστοιχα σημεία [13]

Το ακόλουθο παράδειγμα παρουσιάζει τη χρήση της μεθόδου drawString (). Με την επιλογή του αντίστοιχου σημείου, το κείμενο εμφανίζεται σε σχέση με την επάνω αριστερή και κάτω δεξιά γωνία της οθόνης χωρίς να επικαλύπτει τα όρια της:

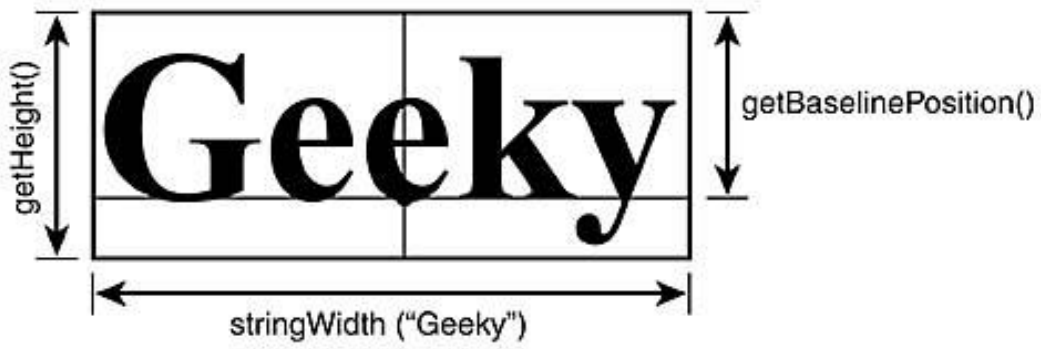
```
import javax.microedition.lcdui.*;

    class TextDemoCanvas extends Canvas {

     public void paint (Graphics g) {
      g.setGrayScale (255);
      g.fillRect (0, 0, getWidth (), getHeight ());

      g.setGrayScale (0);
      g.drawString ("Top/Left", 0, 0, Graphics.TOP | Graphics.LEFT);
      g.drawString ("Baseline/Center", getWidth () / 2, getHeight () / 2,
          Graphics.HCENTER | Graphics.BASELINE);
      g.drawString ("Bottom/Right", getWidth (), getHeight (),
          Graphics.BOTTOM | Graphics.RIGHT);
     }
    }
```

Εικόνα 29: Έξοδος του TextDemo παραδείγματος

Εκτός από το τρέχον χρώμα, το αποτέλεσμα της μεθόδου drawString () επηρεάζεται από την τρέχουσα γραμματοσειρά. Το MIDP παρέχει υποστήριξη για τρεις διαφορετικές γραμματοσειρές σε τρία διαφορετικά μεγέθη και με τα τρία διαφορετικά χαρακτηριστικά: έντονη, πλάγια, και υπογραμμισμένη.

Μέθοδος setFont ()

Μια γραμματοσειρά δεν επιλέγεται άμεσα, αλλά η setFont () μέθοδος λαμβάνει ένα ξεχωριστό αντικείμενο τύπου Font, περιγράφοντας την επιθυμητή γραμματοσειρά, ως παράμετρο. Η κλάση Font παρέχει πρόσθετες πληροφορίες σχετικά με τη γραμματοσειρά, όπως το πλάτος και το ύψος σε pixels, τη θέση της baseline και ούτω καθεξής. Το σχήμα που ακολουθεί, παρουσιάζει την έννοια των αντίστοιχων τιμών. Οι πληροφορίες αυτές είναι σημαντικές για ενέργειες όπως η σχεδίαση πλαισίου γύρω από το κείμενο. Επιπλέον, όταν παρέχονται αλγόριθμοι για word-wrapping (αναδίπλωση κειμένου), βασίζονται στο πραγματικό πλάτος σε pixel των συμβολοσειρών.



Εικόνα 30: Ιδιότητες γραμματοσειράς και οι αντίστοιχες μέθοδοι [13]

Μέθοδος createFont()

Ένα αντικείμενο Font δημιουργείται με την κλήση της στατικής μεθόδου createFont () της κλάσης Font του πακέτου lcdui. Η createFont () μέθοδος λαμβάνει τρεις παραμέτρους: τον τύπο της γραμματοσειράς, το στυλ, και το μέγεθος της γραμματοσειράς. Παρόμοια με τη διάταξη του κειμένου, υπάρχουν προκαθορισμένες σταθερές για τον ορισμό της αντίστοιχης τιμής των παραπάνω παραμέτρων. Οι σταθερές αυτές αναφέρονται στον παρακάτω πίνακα.

Πίνακας 13: Σταθερές παραμέτρων της createFont () [13]

Ιδιότητα	**Σταθερές**
Μέγεθος	SIZE_SMALL, SIZE_MEDIUM, SIZE_LARGE
Στυλ	STYLE_PLAIN,STYLE_ITALICS,STYLE_BOLD, STYLE_UNDERLINED
Τύπος	FACE_SYSTEM,FACE_MONOSPACE, FACE_PROPORTIONAL

Οι σταθερές για το στυλ μπορούν να συνδυαστούν, όπως για παράδειγμα, STYLE_ITALICS | STYLE_BOLD που θα οδηγήσει σε έντονο πλάγιο στυλ γραμματοσειράς.

Το ακόλουθο παράδειγμα δείχνει μια λίστα με όλες τις διαθέσιμες γραμματοσειρές:

```
import javax.microedition.lcdui.*;

    class FontDemoCanvas extends Canvas {

     static final int [] styles = {Font.STYLE_PLAIN,
           Font.STYLE_BOLD,
           Font.STYLE_ITALIC};
     static final int [] sizes = {Font.SIZE_SMALL,
           Font.SIZE_MEDIUM,
           Font.SIZE_LARGE};
     static final int [] faces = {Font.FACE_SYSTEM,
           Font.FACE_MONOSPACE,
           Font.FACE_PROPORTIONAL};

     public void paint (Graphics g) {
      Font font = null;
      int y = 0;
      g.setGrayScale (255);
      g.fillRect (0, 0, getWidth (), getHeight ());
```

```
g.setGrayScale (0);

for (int size = 0; size < sizes.length; size++) {
 for (int face = 0; face < faces.length; face++) {
  int x = 0;
  for (int style = 0; style < styles.length; style++) {
   font = Font.getFont
     (faces [face], styles [style], sizes [size]);
   g.setFont (font);
   g.drawString
    ("Test", x+1, y+1, Graphics.TOP | Graphics.LEFT);
   g.drawRect
    (x, y, font.stringWidth ("Test")+1,
     font.getHeight () + 1);
   x += font.stringWidth ("Test")+1;
  }
  y += font.getHeight () + 1;
 }
 }
}
```

Εικόνα 31: Έξοδος του παραδείγματος FontDemo

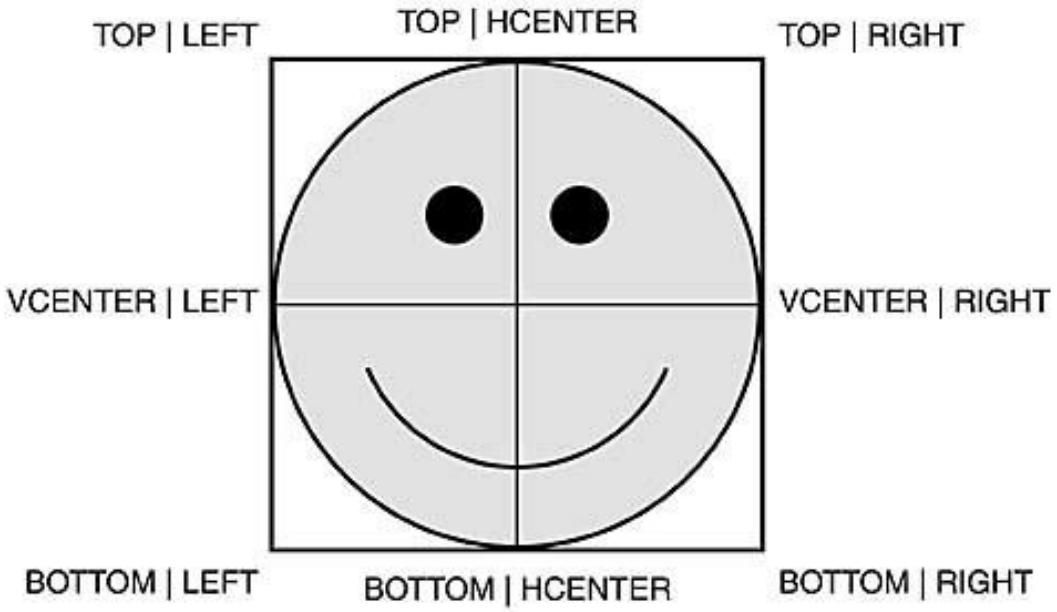
4.4 Images

Η κλάση Graphics παρέχει επίσης μια μέθοδο για την επεξεργασία εικόνων. Όπως φαίνεται στην τελική έκδοση της εφαρμογής TeleTransfer, τα ImageS μπορούν να προκαθοριστούν και να περιέχονται στο αρχείο JAR του MIDlet. Η μόνη μορφοποίηση αρχείου που είναι υποχρεωτική για το MIDP είναι η Portable Network Graphics (PNG). Η μορφοποίηση PNG έχει πολλά πλεονεκτήματα σε σχέση με άλλες μορφοποιήσεις γραφικών. Για παράδειγμα, είναι ελεύθερη για χρήση και υποστηρίζει έγχρωμες εικόνες. Οι PNG εικόνες είναι πάντα συμπιεσμένες με έναν αλγόριθμο λιγότερων απωλειών. Ο αλγόριθμος είναι ταυτόσημος με τον αλγόριθμο που χρησιμοποιείται για τα αρχεία JAR, έτσι ώστε η MIDP εφαρμογή να μπορεί να εξοικονομήσει χώρο, χρησιμοποιώντας τον ίδιο αλγόριθμο και για τους δύο σκοπούς.

Μια εικόνα μπορεί να φορτωθεί από το αρχείο JAR χρησιμοποιώντας τη στατική μέθοδο Image.create (String name). Η παράμετρος name δηλώνει το όνομα του αρχείου της εικόνας στο αρχείο JAR. Σημειώστε ότι η μέθοδος create () μπορεί να προκαλέσει ένα IOException.

Μέθοδος drawImage ()

Η drawImage () μέθοδος της Graphics απαιτεί ένα αντικείμενο τύπου Image, τις συντεταγμένες, και έναν ακέραιο αριθμός που δηλώνει την στοίχιση ως παραμέτρο. Η παράμετρος στοίχισης είναι παρόμοια με την στοίχιση της drawString (), εκτός από το ότι η BASELINE σταθερά δεν υποστηρίζεται. Μια πρόσθετη σταθερά στοίχισης διαθέσιμη μόνο για εικόνες είναι η VCENTER, που οδηγεί την εικόνα να κεντραριστεί κάθετα σε σχέση με τις δοσμένες συντεταγμένες. Το σχήμα που ακολουθεί παρουσιάζει τους έγκυρους συνδυασμούς σταθερών και τα αντίστοιχα σημεία. (Ο HCENTER **|** VCENTER είναι επίσης ένας έγκυρος συνδυασμός)

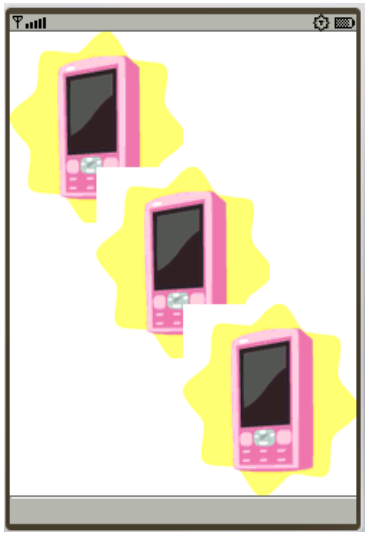


Εικόνα 32: Οι συνδυασμοί σταθερών στοίχισης έγκυροι για εικόνες και τα αντίστοιχα σημεία [13]

Το ακόλουθο παράδειγμα πρώτα φορτώνει την εικόνα logo.png από το MIDlet JAR αρχείο στον κατασκευαστή, και στη συνέχεια, εμφανίζει την εικόνα τρεις φορές. Μία εικόνα θα βρίσκεται στην επάνω αριστερή γωνία, μία στην κάτω δεξιά γωνία, και μία στο κέντρο της οθόνης, όπως φαίνεται στο επόμενο σχήμα:

```
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

    class ImageDemoCanvas extends Canvas {

     Image image;

     public ImageDemoCanvas () {
      try {
       image = Image.createImage ("/logo.png");
      }
```

```
        catch (IOException e) {
         throw new RuntimeException ("Unable to load Image: "+e);
        }
       }

       public void paint (Graphics g) {
        g.setGrayScale (255);
        g.fillRect (0, 0, getWidth (), getHeight ());

        g.drawImage (image, 0, 0, Graphics.TOP | Graphics.LEFT);
        g.drawImage (image, getWidth () / 2, getHeight () / 2,
            Graphics.HCENTER | Graphics.VCENTER);
        g.drawImage (image, getWidth (), getHeight (),
            Graphics.BOTTOM | Graphics.RIGHT);
       }
      }
```



Εικόνα 33: Έξοδος του ImageDemo παραδείγματος

Οι εικόνες μπορούν επίσης να δημιουργηθούν από το μηδέν κατά τον χρόνο εκτέλεσης. Η στατική μέθοδος Image.create (int width, int height) δημιουργεί μια νέα δυναμική εικόνα του συγκεκριμένου μεγέθους. Σε αντίθεση με τις εικόνες που φορτώνονται από ένα JAR αρχείο, οι εικόνες αυτές είναι ευμετάβλητες. Οι ευμετάβλητες εικόνες μπορούν να τροποποιηθούν με την κλήση της μεθόδου getGraphics (). Το Graphics αντικείμενο που επιστρέφεται μπορεί να

χρησιμοποιηθεί για την τροποποίηση της εικόνας με όλες τις μεθόδους που προβλέπονται από την κλάση Graphics. Σημειώστε ότι οι εικόνες που φορτώνονται από ένα αρχείο JAR δεν μπορούν να τροποποιηθούν. Ωστόσο, είναι δυνατόν να δημιουργηθεί μια ευμετάβλητη εικόνα, και στη συνέχεια να σχεδιαστεί πάνω σ' αυτήν μια άλλη εικόνα.

Με την τροποποίηση του δομητή του προηγούμενου παραδείγματος, η σχεδιασμένη εικόνα που δημιουργείται στην paint () μέθοδο και συμπληρώνεται κατά το χρόνο εκτέλεσης, αντί να φορτωθεί μια εικόνα από το αρχείο JAR:

```
public ImageDemoCanvas () {
     image = Image.createImage (10,10);
     image.getGraphics ().fillArc (0,0,10,10,0, 360);
    }
```

Το μειονέκτημα των ευμετάβλητων εικόνων είναι ότι δεν μπορούν να χρησιμοποιηθούν σε υψηλού επιπέδου GUI στοιχεία, δεδομένου ότι είναι δυνατή η τροποποίηση τους ανά πάσα στιγμή, που πιθανώς να οδηγήσει σε άνιση απεικόνιση των widgets. Για το λόγο αυτό, μια άλλη στατική create μέθοδος, η createImage(Image image) παρέχεται, η οποία δημιουργεί μια αμετάβλητη εικόνα από μια άλλη εικόνα.

4.5 Αλληλεπίδραση

Επειδή η κλάση Canvas είναι μια υποκλάση της Displayable, παρέχει την ίδια υποστήριξη για εντολές όπως οι κλάσεις οθόνης υψηλού επιπέδου. Εδώ, θα επικεντρωθείτε στις πρόσθετες δυνατότητες αλληλεπίδρασης που προσφέρει η κλάση Canvas: άμεσο key input και υποστήριξη pointer.

Σημειώστε ότι όλα τα γεγονότα εισαγωγής, οι ειδοποιήσεις εντολών και η μέθοδος paint() καλούνται σειριακά. Αυτό σημαίνει ότι ο διαχειριστής εφαρμογής δε θα καλέσει καμία από τις μεθόδους μέχρι η προηγούμενη μέθοδος χειρισμού γεγονότος να επιστρέψει. Έτσι, όλες αυτές οι μέθοδοι θα πρέπει να επιστρέψουν γρήγορα, αλλιώς ο χρήστης δε θα είναι σε θέση να αλληλεπιδράσει με την εφαρμογή. Για περισσότερες εργασίες, μπορεί να ξεκινήσει ένα ξεχωριστό νήμα.

4.5.1 Key Input

Μέθοδοι keyPressed (), keyReleased () και keyRepeated ()

Για την εισαγωγή απ' το πληκτρολόγιο, η κλάση Canvas παρέχει τρεις μεθόδους επανάκλησης: τις keyPressed (), keyReleased () και keyRepeated (). Όπως δείχνουν και τα ονόματα, η keyPressed () καλείται όταν πιεστεί ένα πλήκτρο, η keyRepeated () καλείται όταν ο χρήστης κρατά πατημένο το πλήκτρο για μεγαλύτερο χρονικό διάστημα, και η keyReleased () καλείται όταν ο χρήστης απελευθερώνει ένα πλήκτρο.

Και οι τρεις μέθοδοι επανάκλησης παρέχουν έναν ακέραιο ως παράμετρο, που δηλώνει έναν κωδικό ενός χαρακτήρα στο Unicode, ο οποίος έχει εκχωρηθεί σε ένα αντίστοιχο πλήκτρο. Εάν ένα πλήκτρο δεν έχει αντιστοίχηση στο Unicode, ο δοσμένος ακέραιος είναι αρνητικός. Το MIDP ορίζει τις ακόλουθες σταθερές για τα πλήκτρα ενός standard ITU-T πληκτρολογίου: KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND και KEY_STAR. Οι εφαρμογές δεν πρέπει να βασίζονται στην παρουσία τυχόν πρόσθετων κωδικών πλήκτρων. Συγκεκριμένα, οι κεφαλαίοι και πεζοί χαρακτήρες ή οι χαρακτήρες που παράγονται πατώντας ένα πλήκτρο πολλές φορές δεν υποστηρίζονται από γεγονότα πληκτρολογίου χαμηλού επιπέδου. Ένα "name" που ανατέθηκε σε ένα πλήκτρο μπορεί να ζητηθεί χρησιμοποιώντας την GetKeyName () μέθοδο.

Κάποια πλήκτρα μπορεί να έχουν μια πρόσθετη σημασία για παιχνίδια. Για το σκοπό αυτό, το MIDP παρέχει τις σταθερές UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C και GAME_D. (Σημείωση: οι GAME_ σταθερές αναφέρονται σε ενέργειες που ποικίλουν και προσδιορίζονται από παιχνίδι σε παιχνίδι). Η σημασία του πατήματος ενός πλήκτρου για ένα παιχνίδι μπορεί να προσδιοριστεί καλώντας την getGameAction () μέθοδο. Η αντιστοίχηση των κωδικών των πλήκτρων σε ενέργειες του παιχνιδιού εξαρτάται από την συσκευή, οπότε διαφορετικά πλήκτρα μπορούν να αντιστοιχίζονται στην ίδια ενέργεια παιχνιδιού σε διαφορετικές συσκευές. Για παράδειγμα, μερικές συσκευές μπορεί

να έχουν διαχωρισμένα πλήκτρα για τον κέρσορα ενώ άλλες μπορεί να αντιστοιχούν το number pad σε μετακίνηση τεσσάρων κατευθύνσεων. Επίσης, διάφορα πλήκτρα μπορεί να αντιστοιχιστούν στον ίδιο κωδικό ενέργειας ενός παιχνιδιού. Ο κωδικός ενέργειας ενός παιχνιδιού μπορεί να «μεταφραστεί» σε ένα κωδικό πλήκτρου χρησιμοποιώντας την getKeyCode () μέθοδο. Αυτό προσφέρει επίσης έναν τρόπο για να πάρουμε το όνομα του πλήκτρου που ανατέθηκε σε μια ενέργεια του παιχνιδιού. Για παράδειγμα, η οθόνη βοήθειας μιας εφαρμογής μπορεί να εμφανίσει

"Πατήστε " + getKeyName (getKeyCode (GAME_A))

αντί του "Πατήστε GAME_A".

Η ακόλουθη υλοποίηση της Canvas δείχνει τη χρήση των μεθόδων γεγονότων πλήκτρων. Για κάθε πλήκτρο που πιέζεται, κρατιέται πατημένο ή απελευθερώνεται δείχνει τον τύπο του γεγονότος, τον χαρακτήρα και τον κωδικό του, το όνομα του πλήκτρου και την ενέργεια του παιχνιδιού.

Το πρώτο μέρος της υλοποίησης αποθηκεύει τον τύπο του γεγονότος και τον κωδικό σε δύο μεταβλητές και προγραμματίζει ένα repaint κάθε φορά που συμβαίνει ένα γεγονός πλήκτρου:

```
import javax.microedition.lcdui.*;

class KeyDemoCanvas extends Canvas {

 String eventType = "- Press any!";
 int keyCode;

 public void keyPressed (int keyCode) {
  eventType = "pressed";
  this.keyCode = keyCode;
  repaint ();
 }

 public void keyReleased (int keyCode) {
  eventType = "released";
```

```
 this.keyCode = keyCode;
 repaint ();
}

public void keyRepeated (int keyCode) {
 eventType = "repeated";
 this.keyCode = keyCode;
 repaint ();
}
```

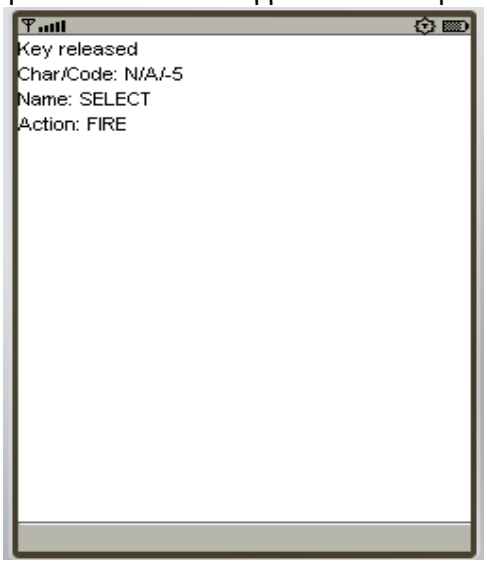
Το δεύτερο μέρος εμφανίζει στην οθόνη της συσκευής όλες τις διαθέσιμες ιδιότητες του γεγονότος. Για το σκοπό αυτό, πρέπει πρώτα να υλοποιήσετε μια πρόσθετη μέθοδο write () που βοηθά την μέθοδο paint () να προσδιορίσει την τρέχουσα y θέση στην οθόνη. Αυτό είναι απαραίτητο επειδή η DrawText () δεν προχωράει σε μια νέα γραμμή αυτόματα. Η write () μέθοδος γράφει μια συμβολοσειρά στη δεδομένη y θέση και επιστρέφει τη y θέση μαζί με το ύψος της γραμμής της τρέχουσας γραμματοσειράς, ώστε η paint () να γνωρίζει πού να τοποθετήσει την επόμενη γραμμή:

```
public int write (Graphics g, int y, String s) {
 g.drawString (s, 0, y, Graphics.LEFT|Graphics.TOP);
 return y + g.getFont ().getHeight ();
}
```

Η paint () μέθοδος αναλύει το keyCode και εμφανίζει το αποτέλεσμα καλώντας την write () μέθοδο που ορίστηκε προηγουμένως, όπως φαίνεται στο παρακάτω σχήμα.

```
public void paint (Graphics g) {
    g.setGrayScale (255);
    g.fillRect (0, 0, getWidth (), getHeight ());
```

```
        g.setGrayScale (0);

        int y = 0;
        y = write (g, y, "Key "+ eventType);
        if (keyCode == 0) return;

        y = write (g, y, "Char/Code: "+ ((keyCode < 0) ? "N/A" : ""
          +(char) keyCode) + "/" + keyCode);
        y = write (g, y, "Name: "+getKeyName (keyCode));
        String gameAction;
        switch (getGameAction (keyCode)) {
        case LEFT: gameAction = "LEFT"; break;
        case RIGHT: gameAction = "RIGHT"; break;
        case UP: gameAction = "UP"; break;
        case DOWN: gameAction = "DOWN"; break;
        case FIRE: gameAction = "FIRE"; break;
        case GAME_A: gameAction = "GAME_A"; break;
        case GAME_B: gameAction = "GAME_B"; break;
        case GAME_C: gameAction = "GAME_C"; break;
        case GAME_D: gameAction = "GAME_D"; break;
        default: gameAction = "N/A";
        }
        write (g, y, "Action: "+gameAction);
       }
    }
```



Εικόνα 34: Έξοδος του KeyDemo παραδείγματος όταν το πλήκτρο "Fire" απελευθερώθηκε.

4.5.2 Γεγονότα Pointer

Μέθοδοι pointerPressed (), pointerDragged () και pointerReleased ()

Για συσκευές που υποστηρίζουν μια συσκευή δείκτη (pointer), όπως μια γραφίδα, ή οθόνη αφής, η κλάση Canvas παρέχει τρεις μεθόδους ειδοποίησης: pointerPressed (), pointerDragged () και pointerReleased (). Αυτές οι μέθοδοι δουλεύουν όπως και οι μέθοδοι γεγονότων πληκτρολογίου, εκτός από το ότι παρέχουν δύο ακέραιες παραμέτρους, που δηλώνουν τη x και y θέση του δείκτη, όταν προκαλείται το αντίστοιχο γεγονός. (Σημειώστε ότι η υποστήριξη δείκτη είναι προαιρετική στο MIDP, έτσι η εφαρμογή δεν πρέπει να βασίζεται στην ύπαρξη ενός δείκτη. Τέτοιες συσκευές (δείκτες) είναι ασυνήθιστες για συσκευές όπως τα κινητά τηλέφωνα.) Το ακόλουθο παράδειγμα δείχνει τη χρήση των τριών μεθόδων:

```
import javax.microedition.lcdui.*;

class PointerDemoCanvas extends Canvas {

 String eventType = "Press Pointer!";
```

```
        int x;
        int y;

        public void pointerPressed (int x, int y) {
         eventType = "Pointer Pressed";
         this.x = x;
         this.y = y;
         repaint ();
        }

        public void pointerReleased (int x, int y) {
         eventType = "Pointer Released";
         this.x = x;
         this.y = y;
         repaint ();
        }

        public void pointerDragged (int x, int y) {
         eventType = "Pointer Repeated";
         this.x = x;
         this.y = y;
         repaint ();
        }

        public void paint (Graphics g) {
         g.setGrayScale (255);
         g.fillRect (0, 0, getWidth (), getHeight ());

         g.setGrayScale (0);

    g.drawString (eventType + " " +x +"/"+y, 0, 0,Graphics.TOP|Graphics.LEFT);
         g.drawLine (x-4, y, x+4, y);
         g.drawLine (x, y-4, x, y+4);
        }
       }
```

Foreground and Background Ειδοποιήσεις

Για διάφορους λόγους, η Canvas μπορεί να μετακινηθεί στο background, όπως για παράδειγμα αν η απεικόνιση έχει ρυθμιστεί σε άλλο displayable αντικείμενο ή αν η συσκευή εμφανίζει ένα παράθυρο διαλόγου του συστήματος. Σε αυτές τις περιπτώσεις, η Canvas ειδοποιείται από την μέθοδο hideNotify (). Όταν η Canvas γίνεται ορατή (και πάλι), η αντίστοιχη μέθοδος showNotify (), καλείται.

4.6 Javagochi Παράδειγμα

Τώρα που είστε εξοικειωμένοι με το Canvas αντικείμενο και τις βασικές μεθόδους σχεδίασης της κλάσης Graphics, είστε έτοιμοι να αναπτύξετε μια μικρή διαδραστική εφαρμογή, την Javagochi.

Το Javagochi ουσιαστικά είναι ένα παιχνίδι στο οποίο εμφανίζονται τυχαία γράμματα του πληκτρολογίου κάτω από μια φάτσα (το Javagochi), η οποία ανάλογα με το βάρος της, σε σχέση με ένα ιδανικό βάρος που ορίζουμε αρχικά, αλλάζει έκφραση, χρώμα και μορφή(λεπταίνει ή παχαίνει). Θα τρέφεται με βιταμίνες, οι οποίες θα αποτελούν τα γράμματα του πληκτρολογίου(π.χ. a). Όσο το ταΐζουμε με τις βιταμίνες που χρειάζεται, μέσα σε ένα καθορισμένο χρονικό διάστημα, το Javagochi παχαίνει. Αντίστιχα, θα αδυνατίζει όσο δεν τρέφεται με τις βιταμίνες αυτές ή αν του παρέχουμε την λάθος βιταμίνη(πληκτρολογούμε άλλο γράμμα από αυτό που ζητείται).

Το ζητούμενο του παιχνιδιού είναι να παραμένει το Javagochi χαρούμενο διατηρώντας το στο ιδανικό βάρος. Όσο του παρέχουμε τις σωστές βιταμίνες το score αυξάνεται, ενώ το Javagochi μπορεί να πεθάνει (οπότε και να τερματιστεί το παιχνίδι) αν παχύνει ή αδυνατίσει πάρα πολύ.

Όπως μπορείτε να δείτε στον παρακάτω κώδικα, η MIDlet υλοποίηση του Javagochi έχει ήδη ολοκληρωθεί, αλλά η κλάση Face λείπει:

```
public class Javagochi extends MIDlet {

 static final int IDEAL_WEIGHT = 100;

 Display display;
 Face face = new Face (this);
 int weight = IDEAL_WEIGHT;
 Timer consumption;
 int score;
```

Πριν ξεκινήσουμε την ανάπτυξη, ας αρχίσουμε να αναλύουμε πρώτα τον τρόπο λειτουργίας του Javagochi. Θα έχει ένα βάρος που έχει αρχικοποιηθεί με την IDEAL_WEIGHT τιμή του. Διαθέτει επίσης στιγμιότυπα των κλάσεων Display, Face, και Timer, τα οποία θα εξηγήσουμε αργότερα.

Η χαρά του Javagochi καθορίζεται από την απόκλιση του τρέχοντος βάρους του από το ιδανικό βάρος, η οποία διακυμαίνεται από 0 έως 10:

```
public int getHappiness () {
    int happiness = 20 - (weight > IDEAL_WEIGHT
        ? 10 * weight / IDEAL_WEIGHT
        : 10 * IDEAL_WEIGHT / weight);
    if (happiness < 0) happiness = 0;
    else if (happiness > 10) happiness = 10;
    return happiness;
   }
```

Το Javagochi πεθαίνει μόνο όταν το επίπεδο χαράς του φτάσει στο μηδέν:

```
 public boolean isDead () {
```

```
        return getHappiness () <= 0;
       }
```

Δεδομένου ότι μια αλλαγή βάρους μπορεί να αλλάξει την εμφάνιση του Javagochi, ένα repaint ζητείται στην μέθοδο transform ():

```
      public void transform (int amount) {
       if (!isDead ()) {
        weight += amount;
        face.repaint ();   } }
```

Όταν το Javagochi MIDlet ξεκινήσει, εμφανίζει τον εαυτό του και ξεκινά ένας Timer, που παρακολουθεί τον κύκλο ζωής του Javagochi, παρακολουθεί δηλαδή τα χρονικά διαστήματα μεταξύ των πληκτρολογήσεων:

```
     public void startApp () {
             display = Display.getDisplay (this);
      display.setCurrent (face);
      consumption = new Consumption (this);
      consumption.start();
     }
```

Όταν το MIDlet τίθεται σε κατάσταση παύσης, το Javagochi τίθεται σε κατάσταση ύπνου, κι έτσι ο Timer πρέπει να τερματιστεί:

```
     public void pauseApp () {
      consumption.leave = true;
      }

      public void destroyApp (boolean forced) {
      }
```

Η κλάση Consumption είναι μια ξεχωριστή κλάση που παρακολουθεί τις ανάγκες του Javagochi για να ζήσει. Στην μέθοδο run (), το score ενημερώνεται κάθε 0.5 δευτερόλεπτα, ανάλογα με την χαρά του Javagochi και μαζί και η μορφή του:

```
class Consumption extends Thread {

Javagochi javagochi;
public boolean leave = false;

public Consumption (Javagochi javagochi) {
 this.javagochi = javagochi;
}

public void run () {
 while (!leave) {
  try {
   sleep (500);
  }
  catch (InterruptedException e) {break;}
  javagochi.score += 10 - javagochi.weight / javagochi.IDEAL_WEIGHT;
  javagochi.transform (-5);
 }
}
}
```

Τώρα που ξέρετε πώς ένα Javagochi λειτουργεί, πρέπει να δώσετε την κατάλληλη εμφάνιση στο Javagochi με την υλοποίηση της κλάσης Face.

4.6.1 Scaling and Fitting

Σε πολλές περιπτώσεις, είναι καλή ιδέα η κλιμάκωση του μεγέθους των απεικονιζόμενων γραφικών, ανάλογα με το πραγματικό μέγεθος της οθόνης. Σε αντίθετη περίπτωση, η απεικόνιση θα φαίνεται ωραία σε ένα συγκεκριμένο τύπο συσκευής, αλλά δεν θα χωράει στην οθόνη συσκευών με χαμηλότερη ανάλυση οθόνης ή θα γίνεται χωρίς λόγο μικρή σε συσκευές με υψηλότερη ανάλυση οθόνης.

Τώρα θα δείξουμε πώς λειτουργεί η κλιμάκωση για το Javagochi παράδειγμα. Μια εικόνα ενός Javagochi φαίνεται στο παρακάτω σχήμα. Θα ξεκινήσετε από την σχεδίαση του σχήματος του προσώπου, μια απλή έλλειψη. Σε αυτή την περίπτωση, η έλλειψη θα αντανακλά το βάρος του Javagochi. Εάν το Javagochi είναι στο ιδανικό βάρος του, η έλλειψη γίνεται κύκλος.

Εικόνα 35: Ένα χαρούμενο javagochi στο ιδανικό του βάρος [13]

Για να αφήσετε λίγο χώρο για το Javagochi να αναπτυχθεί, η διάμετρος του ιδανικού κύκλου είναι το μισό από το ελάχιστο πλάτος και ύψος της οθόνης. Έτσι, το ύψος του Javagochi υπολογίζεται χρησιμοποιώντας την ακόλουθη γραμμή κώδικα:

```
int height = Math.min (getHeight (), getWidth ()) / 2;
```

Με βάση το τρέχον βάρος, το ιδανικό βάρος και το υπολογισμένο ύψος, το οποίο είναι επίσης η διάμετρος του "ιδανικού" Javagochi, μπορείτε τώρα να υπολογίσετε το πλάτος του Javagochi:

```
int width = height * javagochi.weight / javagochi.IDEAL_WEIGHT;
```

Άλλες εφαρμογές μπορεί φυσικά να εξαρτώνται και από άλλες παραμέτρους πέρα από το πραγματικό μέγεθος της οθόνης, αλλά αυτό το παράδειγμα θα πρέπει να αρκεστεί στο να δείξει την γενική ιδέα.

Το χρώμα του Javagochi εξαρτάται από την χαρά του. Εάν το Javagochi είναι χαρούμενο, το χρώμά του είναι έντονο κίτρινο. Με την μείωση της χαράς του, το Javagochi γίνεται χλωμό. Αυτό αντικατοπτρίζεται από την ακόλουθη setColor () εντολή:

```
setColor (255, 255, 255-javagochi.happiness *25);
```

Χρησιμοποιώντας το δοσμένο πλάτος και ύψος, μπορείτε τώρα να υλοποιήσετε την πρώτη έκδοση της κλάσης Face του Javagochi:

```
import javax.microedition.lcdui.*;
  class Face extends Canvas {
   Javagochi1 javagochi;

   Face (Javagochi1 javagochi) {
    this.javagochi = javagochi;
   }

   public void paint (Graphics g) {
    g.setColor (255, 255, 255);
    g.fillRect (0, 0, getWidth (), getHeight ());

    int height = Math.min (getHeight (), getWidth ()) / 2;
    int width = height * javagochi.weight / javagochi.IDEAL_WEIGHT;

    g.translate (getWidth () / 2, getHeight () / 2);

    g.setColor (255, 255, 255 - javagochi.getHappiness () * 25);
    g.fillArc (- width / 2, - height / 2, width, height, 0, 360);

    g.setColor (0, 0, 0);
```

```
        g.drawArc (- width / 2, - height / 2, width, height, 0, 360);


        g.drawString ("Score: "+javagochi.score, 0, -getHeight ()/2,
    Graphics.TOP|Graphics.HCENTER);
      }
    }
```

Για να απλουστευθεί η κεντραρισμένη απεικόνιση του Javagochi, ορίζετε την αρχή του συστήματος συντεταγμένων στο κέντρο της οθόνης χρησιμοποιώντας τη μέθοδο translate (). Το περίγραμμα του προσώπου του Javagochi σχεδιάζεται στη συνέχεια χρησιμοποιώντας την drawArc () μέθοδο.

Για την σχεδίαση των ματιών δημιουργείτε μια ξεχωριστή μέθοδο. Η drawEye() μέθοδος λαμβάνει το αντικείμενο Graphics, τις συντεταγμένες του ματιού και μια παράμετρο μεγέθους:

```
    void drawEye (Graphics g, int x, int y, int size) {
    if (javagochi.isDead ()) {
     g.drawLine (x - size/2, y, x + size/2, y);
     g.drawLine (x, y - size/2, x, y + size/2);
    }
    else
     g.drawArc (x-size/2, y-size/2, size, size, 0, 360);
   }
```

Τώρα μπορείτε να εισάγετε το υπόλοιπο του κώδικα σχεδίασης στην μέθοδο paint (), αμέσως μετά την μέθοδο drawArc (). Θα ξεκινήσετε με τα μάτια, καλώντας την drawEye () μέθοδο που ορίστηκε προηγουμένως. Χρησιμοποιώντας κλάσματα του τρέχοντος πλάτους και ύψους του Javagochi, τοποθετούνται τα μάτια και μετριέται το μέγεθος τους σωστά:

```
drawEye (g, - width / 6, - height / 5, height / 15 + 1);
drawEye (g, width / 6, - height / 5, height / 15 + 1);
```

Τώρα σχεδιάζετε το στόμα, ανάλογα με την τρέχουσα χαρά του Javagochi. Και πάλι, χρησιμοποιείτε κλάσματα του μεγέθους του Javagochi για την τοποθέτηση και την μέτρηση του μεγέθους του στόματος:

```
  switch (javagochi.getHappiness () / 3) {
  case 0:
  case 1: g.drawArc (-width/6, height/7, width/3, height/6, 0, 180);
  break;
  case 2: g.drawLine (-width/6, height/7, width/6, height/7);
  break;
default: g.drawArc (-width/6, height/7, width/3, height/6, 0, -180);
  }
```

4.6.2 Απλή Αλληλεπίδραση

Προφανώς, χρειάζεστε έναν τρόπο ώστε όταν το Javagochi χρειάζεται επειγόντως τροφή, να είστε σε θέση να αντιδράσετε γρήγορα. Έτσι, απλά χρησιμοποιείτε το γεγονός πλήκτρου που αντιστοιχεί στην ενέργεια του παιχνιδιού FIRE για να ταΐσετε το Javagochi:

```
   public void keyPressed (int keyCode) {
   if (getGameAction (keyCode) == FIRE)
    javagochi.transform (10);
}
```

Τώρα μπορείτε να σώσετε το Javagochi από την πείνα χρησιμοποιώντας το πλήκτρο παιχνιδιού FIRE.

Canvas και Text Input

Ας υποθέσουμε ότι το απλό τάισμα δεν είναι αρκετό για το Javagochi σας. Ανάλογα με την τρέχουσα κατάσταση του, χρειάζεται ειδικές βιταμίνες, που συμβολίζονται με γράμματα από το a ως το z. Για τηλέφωνα που παρέχουν τα πλήκτρα 0 έως 9 μόνο, αυτό είναι ένα πρόβλημα. Η μόνη λύση είναι η προσομοίωση του μηχανισμού εισαγωγής από πληκτρολόγιο στο λογισμικό.

Στα κινητά τηλέφωνα, συνήθως υπάρχουν τρία με τέσσερα γράμματα τυπωμένα στα αριθμητικά πλήκτρα. Στην κατάσταση εισαγωγής κειμένου, το πάτημα ενός αριθμού κάνει το πρώτο γράμμα να εμφανιστεί. Αν ο ίδιος αριθμός πατηθεί ξανά σε ένα περιορισμένο χρονικό διάστημα, το δεύτερο γράμμα θα εμφανιστεί αντί για το πρώτο. Με αυτό τον τρόπο μπορείτε κυκλικά να χρησιμοποιήσετε όλα τα γράμματα ενός αριθμητικού πλήκτρου. Όταν το πλήκτρο δεν ξαναπατηθεί για περίπου τρία τέταρτα του δευτερολέπτου, ή ένα άλλο πλήκτρο πατηθεί, το τρέχον γράμμα που απεικονίζεται, επιβεβαιώνεται ως εισαγωγή πληκτρολογίου.

Για την προσομοίωση του μηχανισμού αυτού, ορίζετε τα γράμματα των πλήκτρων 2 έως 9 σε έναν πίνακα String μέσα στην κλάση Face:

```
public static final String[] keys = {"abc", "def", "ghi", "jkl",
        "mno", "pqrs", "tuv", "wxyz"};
```

Χρειάζεστε επίσης έναν μετρητή για να μετράται ο χρόνος μέχρι την επιβεβαίωση του τρέχοντος πλήκτρου. Ο μετρητής είναι αποθηκευμένος ως keyTimer. Οι μεταβλητές keyMajor και keyMinor περιέχουν τον δείκτη στον πίνακα πλήκτρων και τον δείκτη στο εσωτερικό του αντίστοιχου string, αντίστοιχα. Η μεταβλητή needed αποθηκεύει την βιταμίνη που χρειάζεται το Javagochi:

```
Timer keyTimer;
int keyMajor = -1;
int keyMinor;
char needed = 'a';
```

Τι κάνετε όταν ένα αριθμητικό πλήκτρο πατηθεί; Αν έχετε ήδη έναν μετρητή που εκτελείται, τον ακυρώνετε από την στιγμή που ένα πλήκτρο πατηθεί. Στη συνέχεια, αφαιρείτε τον κωδικό του πλήκτρου 2 από τον τρέχον κωδικό πλήκτρου για τον υπολογισμό του δείκτη στον πίνακα πλήκτρων. Αν το συγκεκριμένο γεγονός δεν αντιπροσωπεύει ένα αριθμητικό πλήκτρο μεταξύ 2 και 9, θέτετε το keyMajor στην ειδική τιμή -1, δηλώνοντας ότι ένας μη έγκυρος χαρακτήρας έχει εισαχθεί. Διαφορετικά, ελέγχετε αν το πλήκτρο είναι ίδιο με το τελευταίο πλήκτρο. Αν ναι, ο keyMinor αυξάνεται, ώστε να διαπεράσει όλα τα γράμματα που έχουν

ανατεθεί σε ένα και μόνο αριθμητικό πλήκτρο. Αν ένα άλλο πλήκτρο πατηθεί, ο keyMajor αλλάζει αναλόγως και ο keyMinor τίθεται ξανά σε 0. Ένας νέος μετρητής προγραμματίζεται για μισό δευτερόλεπτο αργότερα:

```
public synchronized void keyPressed (int keyCode) {

 if (keyTimer != null) keyTimer.cancel ();

 int index = keyCode - KEY_NUM2;

 if (index < 0 || index > keys.length)
  keyMajor = -1;
 else {
  if (index != keyMajor) {
   keyMinor = 0;
   keyMajor = index;
  }
  else {
   keyMinor++;
   if (keyMinor >= keys [keyMajor].length ())
    keyMinor = 0;
  }

  keyTimer = new Timer ();
  keyTimer.schedule (new KeyConfirmer (this), 500);
 }
 repaint ();
}
```

Τώρα πρέπει να υλοποιήσετε έναν TimerTask (μετρητή) που επιβεβαιώνει το γράμμα εάν δεν πατηθεί άλλο πλήκτρο για μισό δευτερόλεπτο. Στην περίπτωση αυτή, η κλάση KeyConfirmer απλά καλεί την keyConfirmed () στην αρχική κλάση Face:

```
import java.util.*;
```

```
class KeyConfirmer extends TimerTask {

 Face face;

 public KeyConfirmer (Face face) {
  this.face = face;
 }

 public void run () {
  face.keyConfirmed ();
 }
}
```

Πίσω στην κλάση Face, μπορείτε τώρα να υλοποιήσετε τη λειτουργικότητα που εκτελείται όταν το γράμμα τελικά επιβεβαιώνεται. Απλά συγκρίνετε το γράμμα με την βιταμίνη που χρειάζεται το Javagochi. Αν δοθεί η σωστή βιταμίνη, το βάρος του Javagochi αυξάνεται κατά 10 μονάδες καλώντας την transform ():

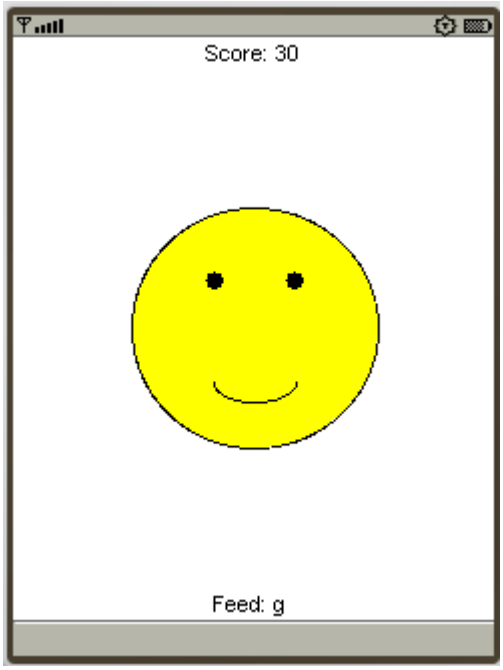
```
synchronized void keyConfirmed () {
  if (keyMajor != -1) {

   if (keys [keyMajor].charAt (keyMinor) == needed) {
    javagochi.score += javagochi.getHappiness ();

    if (!javagochi.isDead ())
     needed = (char) ('a'
      + ((System.currentTimeMillis () / 10) % 26));

    javagochi.transform (10);
   }

   keyMajor = -1;
   repaint ();
  }
```

```
    }
}
```

Τέλος, προσθέτετε κάποιες πληροφορίες κατάστασης σχετικά με την τρέχουσα βαθμολογία και το επιλεγμένο πλήκτρο στην paint () μέθοδο της κλάσης Face. Απλά εισάγετε τον ακόλουθο κώδικα στο τέλος της προηγούμενης υλοποίησης της paint ():

```
String keySelect = "";
if (keyMajor != -1) {
 String all = keys [keyMajor];
 keySelect = all.substring (0, keyMinor) + "[" + all.charAt (keyMinor)
      + "]" + all.substring (keyMinor+1);
}

g.drawString ("Feed: " + needed + " " + keySelect, 0,
   getHeight ()/2, Graphics.BOTTOM|Graphics.HCENTER);
g.drawString ("Score: "+javagochi.score, 0,
   -getHeight ()/2, Graphics.TOP|Graphics.HCENTER);
```

Το σχήμα που ακολουθεί παρουσιάζει το Javagochi χαρούμενο. Ο πλήρης πηγαίος κώδικας ακολουθεί:



Εικόνα 36: Έξοδος του javagochi παραδείγματος

Javagochi.java

Ο Ολοκληρωμένος Πηγαίος Κώδικας του Javagochi παραδείγματος

```
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

class Consumption extends TimerTask {

 Javagochi javagochi;

 public Consumption (Javagochi javagochi) {
  this.javagochi = javagochi;
 }


 public void run () {
  javagochi.transform (-1 - javagochi.score/100 );
 }
}
```

```
class KeyConfirmer extends TimerTask {

 Face face;

 public KeyConfirmer (Face face) {
  this.face = face;
 }

 public void run () {
  face.keyConfirmed ();
 }
}

class Face extends Canvas {

 public static final String[] keys = {"abc", "def", "ghi", "jkl",
          "mno", "pqrs", "tuv", "wxyz"};

 Javagochi javagochi;
 Timer keyTimer;

 int keyMajor = -1;
 int keyMinor;
 char needed = 'a';

 Face (Javagochi javagochi) {
  this.javagochi = javagochi;
 }

 public void paint (Graphics g) {
  g.setColor (255, 255, 255);
  g.fillRect (0, 0, getWidth (), getHeight ());

  int height = Math.min (getHeight (), getWidth ()) / 2;
```

```
        int width = height * javagochi.weight  / javagochi.IDEAL_WEIGHT;

        g.translate (getWidth () / 2, getHeight () / 2);

        g.setColor (255, 255, 255 - javagochi.getHappiness () * 25);
        g.fillArc (- width / 2, - height / 2, width, height, 0, 360);

        g.setColor (0, 0, 0);
        g.drawArc (- width / 2, - height / 2, width, height, 0, 360);

        String keySelect = "";
        if (keyMajor != -1) {
         String all = keys [keyMajor];
         keySelect = all.substring (0, keyMinor) + "[" + all.charAt (keyMinor) + "]" +
    all.substring (keyMinor+1);
        }

        g.drawString ("Feed: " + needed + " " + keySelect,
            0, getHeight ()/2, Graphics.BOTTOM|Graphics.HCENTER);

        g.drawString ("Score: "+javagochi.score, 0, -getHeight ()/2,
    Graphics.TOP|Graphics.HCENTER);

        drawEye (g, - width / 6, - height / 5, height / 15 + 1);
        drawEye (g, width / 6, - height / 5, height / 15 + 1);

        switch (javagochi.getHappiness () / 3) {
        case 0:
        case 1:
         g.drawArc (-width/6, height/7, width/3, height/6, 0, 180);
         break;
        case 2:
         g.drawLine (-width/6, height/7, width/6, height/7);
         break;
        default:
```

```
    g.drawArc (-width/6, height/7, width/3, height/6, 0, -180);
   }
  }

  void drawEye (Graphics graphics, int x0, int y0, int w) {
   if (javagochi.isDead ()) {
    graphics.drawLine (x0 - w/2, y0, x0 + w/2, y0);
    graphics.drawLine (x0, y0 - w/2, x0, y0 + w/2);
   }
   else
    graphics.fillArc (x0-w/2, y0-w/2, w, w, 0, 360);
  }

  public synchronized void keyPressed (int keyCode) {

   int index = keyCode - KEY_NUM2;

   if (keyTimer != null) keyTimer.cancel ();

   if (index < 0 || index > keys.length)
    keyMajor = -1;
   else {
    if (index != keyMajor) {
     keyMinor = 0;
     keyMajor = index;
    }
    else {
     keyMinor++;
     if (keyMinor >= keys [keyMajor].length ())
      keyMinor = 0;
    }

    keyTimer = new Timer ();
    keyTimer.schedule (new KeyConfirmer (this), 500);
   }
```

```
        repaint ();
       }

       synchronized void keyConfirmed () {
        if (keyMajor != -1) {

         if (keys [keyMajor].charAt (keyMinor) == needed) {
          javagochi.score += javagochi.getHappiness ();

          if (!javagochi.isDead ())
           needed = (char) ('a'  + ((System.currentTimeMillis () / 10) % 26));

          javagochi.transform (10);
         }

         keyMajor = -1;
         repaint ();
        }
       }
      }

      public class Javagochi extends MIDlet {

       static final int IDEAL_WEIGHT = 100;

       Display display;
       Face face = new Face (this);
       int weight = IDEAL_WEIGHT;
       Timer consumption;
       int score;

       public int getHappiness () {
        int happiness = 20 - (weight > IDEAL_WEIGHT
            ? 10 * weight / IDEAL_WEIGHT
            : 10 * IDEAL_WEIGHT / weight);
```

```
 if (happiness < 0) happiness = 0;
 else if (happiness > 10) happiness = 10;
 return happiness;
}

public boolean isDead () {
 return getHappiness () == 0;
}

public void transform (int amount) {
 if (!isDead ()) {
  weight += amount;
  face.repaint ();
 }
}

public void startApp () {
 display = Display.getDisplay (this);
 display.setCurrent (face);
 consumption = new Timer ();
 consumption.scheduleAtFixedRate (new Consumption (this), 500, 500);
}

public void pauseApp () {
 consumption.cancel ();
}

public void destroyApp (boolean forced) {
} }
```

Σύνοψη Κεφαλαίου

Σ’ αυτό το κεφάλαιο αναπτύχθηκε το χαμηλού επιπέδου API, το οποίο επιτρέπει τον πλήρη έλεγχο της απεικόνισης του MID σε επίπεδο pixel. Είδατε τρόπους σχεδίασης, σύνταξης κειμένου και αλληλεπίδρασης με τον χρήστη.

Ερωτήσεις

1.Ποιά μέθοδος καλείται κάθε φορά που ο διαχειριστής εφαρμογής κρίνει απαραίτητο τον σχεδιασμό της οθόνης και σε ποιά κλάση ανήκει αυτή η μέθοδος;

2.Ποιές μέθοδοι χρησιμοποιούνται για α)Σχεδίαση γραμμών, β)Σχεδίαση γεμμάτης ορθογώνιας περιοχής και γ)Σχεδίαση συμβολοσειρών. Ποιές παραμέτρους λαμβάνει αυτή η μέθοδος; Σε ποιά κλάση ανήκουν οι παραπάνω μέθοδοι;

3.Ποιές είναι οι μέθοδοι με τις οποίες μπορεί να επηρεαστεί το χρώμα της σχεδίασης; Τι παραμέτρους μπορούν να λάβουν αυτές οι μέθοδοι;

4.Με ποιά μέθοδο μπορεί να επηρεαστεί το στύλ σχεδίασης; Τι παραμέτρους μπορεί να λάβει;

5.Με ποιά μέθοδο μπορεί να αλλάξει η αρχή του συστήματος συντεταγμένων;

6.Με ποιά μέθοδο μπορεί να περιοριστεί η περιοχή όπου η σχεδίαση μπορεί να απεικονιστεί και πως ονομάζεται η περιοχή αυτή; Με ποιά μέθοδο μπορεί να μεγεθυνθεί η περιοχή αυτή; Αναφέρετε τις μεθόδους με τις οποίες μπορείτε να μάθετε τις διαστάσεις τις περιοχής αυτής.

7.Με ποιά μέθοδο επιτρέπεται η σχεδίαση ενός Image; Ποιές οι παράμετροι της; Ποιά μορφοποίηση αρχείου εικόνας είναι υποχρεωτική για το MIDP;

8.Ποιές μέθοδοι επανάκλησης παρέχονται από την Canvas για εισαγωγή απ’ το πληκτρολόγιο και πότε καλείται η κάθε μία;

9. Ποιές μέθοδοι ειδοποίησης παρέχονται από την Canvas για συσκευές που υποστηρίζουνε συσκευή δείκτη (pointer) και πότε καλείται η κάθε μια;

10.Με ποιά μέθοδο μπορεί να προσδιοριστεί η σημασία του πατήματος ενός πλήκτρου για ένα παιχνίδι;

11.Με ποιά μέθοδο μπορεί να ζητηθεί το “name” που ανατέθηκε σε ένα πλήκτρο;

ΚΕΦΑΛΑΙΟ 5

5. Animation

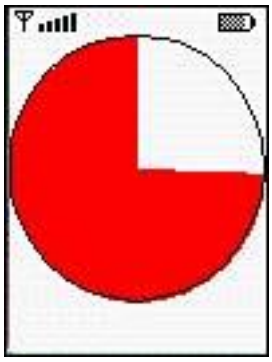
Με τα animation, υπάρχουν συνήθως δύο βασικά προβλήματα: το flickering, δηλαδή οι παράλληλες διάφανες γραμμές σαν σπασίματα στην εικόνα, και ο συγχρονισμός της απεικόνισης με τον υπολογισμό των νέων πλαισίων. Πρώτα θα εξετάσουμε πώς να θέσουμε την πραγματική απεικόνιση και την λογική της εφαρμογής σε συγχρονισμό, και στη συνέχεια πως να επιλύσουμε το πιθανό flickering.

5.1 Συγχρονισμός του Υπολογισμού Πλαισίων και της Σχεδίασης

Όταν εκτελείτε animations, μπορείτε αρχικά να υπολογίσετε το περιεχόμενο απεικόνισης και στη συνέχεια να καλέσετε την repaint (), προκειμένου να σχεδιάσετε το νέο πλαίσιο. Αλλά πώς ξέρετε ότι η κλήση της paint() έχει ολοκληρωθεί; Μία δυνατότητα θα ήταν να καλέσετε την serviceRepaints (), η οποία μπλοκάρει μέχρι να τελειώσουν όλες οι εκκρεμείς ενημερώσεις απεικόνισης. Το πρόβλημα με την serviceRepaints () είναι ότι η paint () μπορεί να κληθεί και από ένα άλλο νήμα. Αν το νήμα, που καλεί την serviceRepaints () κατέχει οποιοδήποτε στοιχείο το οποίο απαιτείται στην paint(), μπορεί να δημιουργηθεί αδιέξοδο.

Επίσης, η κλήση της serviceRepaints () έχει νόημα όταν γίνεται από ένα μόνο νήμα, εκτός από την περίπτωση χειρισμού γεγονότος νήματος. Σε αντίθετη περίπτωση, τα γεγονότα πληκτρολογίου μπορεί να μπλοκαριστούν μέχρι να τελειώσει το animation. Μια εναλλακτική της serviceRepaints () είναι η κλήση της callSerially () στο τέλος της paint () μεθόδου. Η μέθοδος callSerially () σας επιτρέπει να τοποθετήσετε Runnable αντικείμενα στην ουρά των γεγονότων. Έτσι η μέθοδος run () ενός Runnable αντικειμένου θα εκτελεστεί σειριακά όπως και κάθε άλλη μέθοδος χειρισμού γεγονότος. Στην run () μέθοδο, μπορεί να δημιουργηθεί το επόμενο πλαίσιο και να ζητηθεί εκεί ένα νέο repaint.

Προκειμένου να δείξουμε την εκτέλεση αυτού του μοντέλου, θα κατασκευάσουμε ένα απλό χρονόμετρο που μετρά αντίστροφα ένα συγκεκριμένο αριθμό δευτερολέπτων και απεικονίζει το αντίστοιχο κομμάτι πίτας χρησιμοποιώντας την fillArc () μέθοδο, όπως φαίνεται στο παρακάτω σχήμα.



Εικόνα 37: Ένα πολύ απλό χρονόμετρο [13]

Η υλοποίηση της Canvas αποθηκεύει το τρέχον κομμάτι σε μοίρες, τον χρόνο εκκίνησης, το συνολικό ποσό των δευτερολέπτων και την απεικόνιση του MIDlet σε τοπικές μεταβλητές. Για να χρησιμοποιήσει την callSerially (), η Canvas υλοποιεί την Runnable διεπαφή:

```
class StopWatchCanvas extends Canvas implements Runnable {
 int degree = 360;
 long startTime;
 int seconds;
 Display display;
```

Όταν η StopWatchCanvas δημιουργηθεί, αποθηκεύετε την δεδομένη απεικόνιση και τα δευτερόλεπτα. Στη συνέχεια, προσδιορίζεται η τρέχουσα ώρα και αποθηκεύεται, επίσης:

```
StopWatchCanvas (Display display, int seconds) {
 this.display = display;
 this.seconds = seconds;
 startTime = System.currentTimeMillis ();    }
```

Στην paint () μέθοδο, καθαρίζετε την οθόνη. Εάν πρέπει να σχεδιάσετε περισσότερες από 0 μοίρες, μπορείτε να γεμίσετε το αντίστοιχο τόξο με κόκκινο χρώμα και να ζητήσετε νέο υπολογισμό του κομματιού της πίτας χρησιμοποιώντας την μέθοδο callSerially(). Τέλος, σχεδιάζετε το περίγραμμα του χρονομέτρου με τον καθορισμό του χρώματος σε μαύρο και καλώντας την drawArc ():

```
public void paint (Graphics g) {
 g.setGrayScale (255);
 g.fillRect (0, 0, getWidth (), getHeight ());

 if (degree > 0) {
  g.setColor (255, 0, 0);
  g.fillArc (0,0, getWidth (), getHeight (), 90, degree);
  display.callSerially (this);
 }
 g.setGrayScale (0);
 g.drawArc (0, 0, getWidth ()-1, getHeight ()-1, 0, 360);
}
```

Η μέθοδος αυτή καλείται από το νήμα χειρισμού γεγονότος ως αποτέλεσμα της προηγούμενης display.callSerially (this) δήλωσης. Σε αυτή την περίπτωση, απλά υπολογίζει ένα νέο κομμάτι πίτας και ζητά ένα repaint ():

```
 public void run () {
  int permille = (int) ((System.currentTimeMillis ()
       - startTime) / seconds);
  degree = 360 - (permille * 360) / 1000;
  repaint ();
 }
}
```

Όπως πάντα, χρειάζεστε ένα MIDlet για την πραγματική απεικόνιση της StopWatchCanvas υλοποίησης. Ο ακόλουθος κώδικας δημιουργεί ένα χρονόμετρο που ορίζεται στα 10 δευτερολέπτα όταν ξεκινά η εφαρμογή:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class StopWatch extends MIDlet {

 public void startApp () {
  Display display = Display.getDisplay (this);
  display.setCurrent (new StopWatchCanvas (display, 10));
 }

 public void pauseApp () {
 }

 public void destroyApp (boolean forced) {
 }
}
```

5.2 Αποφυγή του Flickering

Σε ορισμένες συσκευές, η εφαρμογή του χρονομέτρου θα τρεμοπαίζει. Αυτό οφείλεται στο γεγονός ότι η οθόνη καθαρίζεται εντελώς πριν ένα νέο χρονόμετρο σχεδιαστεί. Ωστόσο, σε μερικές άλλες συσκευές, το χρονόμετρο δεν θα τρεμοπαίζει επειδή οι εν λόγω συσκευές παρέχουν αυτοματοποιημένο double buffering. Πριν η απεικόνιση ενημερωθεί, όλες οι μέθοδοι σχεδίασης εκτελούνται σε μια κρυφή περιοχή του buffer (της προσωρινής δηλαδή μνήμης). Στη συνέχεια, όταν η paint () μέθοδος τελειώσει, η απεικόνιση ενημερώνεται αμέσως από το offscreen buffer(δηλαδή την κρυφή περιοχή του buffer που λειτουργεί παρασκηνιακά). Η μέθοδος isDoubleBuffered () της κλάσης Canvas είναι σε θέση να καθορίσει αν η οθόνη της συσκευής είναι double buffered.

Προκειμένου να αποφευχθεί το flickering των animation σας σε όλες τις περιπτώσεις, μπορείτε να προσθέσετε τη δική σας offscreen εικόνα, η οποία εντοπίζεται μόνο αν το σύστημα δεν παρέχει double buffering:

```
Image offscreen = isDoubleBuffered () ? null :
Image.createImage (getWidth (), getHeight ());
```

Στην paint () μέθοδο, απλά ελέγχετε αν η offscreen εικόνα δεν είναι null, και εάν είναι, αναθέτετε όλη την σχεδίαση στον offscreen buffer σας. Ο offscreen buffer σχεδιάζεται αμέσως, στο τέλος της paint () μεθόδου, χωρίς πρώτα να καθαριστεί η οθόνη. Ο καθαρισμός της οθόνης δεν είναι αναγκαίος στην περίπτωση αυτή, δεδομένου ότι ο offscreen buffer καθαρίστηκε πριν από την σχεδίαση και γεμίζει κάθε pixel της οθόνης:

```
public void paint (Graphics g) {
 Graphics g2 = offscreen == null ? g : offscreen.getGraphics ();

 g2.setGrayScale (255);
 g2.fillRect (0, 0, getWidth (), getHeight ());

 if (degree > 0) {
  g2.setColor (255, 0, 0);
  g2.fillArc (0,0, getWidth (), getHeight (), 90, degree);

  display.callSerially (this);
 }

 g2.setGrayScale (0);
 g2.drawArc (0, 0, getWidth ()-1, getHeight ()-1, 0, 360);

 if (offscreen != null)
  g.drawImage (offscreen, 0, 0, Graphics.TOP | Graphics.RIGHT);
}
```

BufferedStopWatch.java

Ακολουθεί ο πλήρης πηγαίος κώδικας για το buffered χρονόμετρο (που λειτουργεί με double buffering) [13].

```
import javax.microedition.midlet.*;
```

```
import javax.microedition.lcdui.*;

class BufferedStopWatchCanvas extends Canvas implements Runnable {
 int degree = 360;
 long startTime;
 int seconds;
 Display display;
 Image offscreen;

 BufferedStopWatchCanvas (Display display, int seconds) {
  this.display = display;
  this.seconds = seconds;

  if (!isDoubleBuffered () && false)
   offscreen = Image.createImage (getWidth (), getHeight ());

  startTime = System.currentTimeMillis ();
 }

 public void paint (Graphics g) {

  Graphics g2 = offscreen == null
   ? g
   : offscreen.getGraphics ();

  g2.setGrayScale (255);
  g2.fillRect (0, 0, getWidth (), getHeight ());

  if (degree > 0) {
   g2.setColor (255, 0, 0);
   g2.fillArc (0,0, getWidth (), getHeight (), 90, degree);

   display.callSerially (this);
  }
```

```
 g2.setGrayScale (0);
 g2.drawArc (0, 0, getWidth ()-1, getHeight ()-1, 0, 360);

 if (offscreen != null)
  g.drawImage (offscreen, 0, 0, Graphics.TOP | Graphics.RIGHT);
}

public void run () {
 int permille = (int) ((System.currentTimeMillis ()
      - startTime) / seconds);
 degree = 360 - (permille * 360) / 1000;
 repaint ();
}
}


public class BufferedStopWatch extends MIDlet {

public void startApp () {
 Display display = Display.getDisplay (this);
 display.setCurrent (new BufferedStopWatchCanvas (display, 10));
}

public void pauseApp () {
}

public void destroyApp (boolean forced) {
}          }
```

Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό μάθατε τα δύο βασικά προβλήματα των animations καθώς και τρόπους αντιμετώπισης ή αποφυγής τους.

Ερωτήσεις

1.Ποιά είναι συνήθως τα προβλήματα με τα animation;

2.Ποιοί οι τρόποι αντιμετώπισης ή αποφυγής των παραπάνω προβλημάτων;

Επίλογος

Προκειμένου να επιτύχουμε τον τελικό μας στόχο, δηλαδή να δημιουργήσουμε ένα βοήθημα για ανάπτυξη εφαρμογών σε φορητές συσκευές που υποστηρίζουν java, χρησιμοποιώντας την J2ME γλώσσα προγραμματισμού και να μπορέσει αυτό να αποτελέσει χρήσιμο εργαλείο για διδάσκοντες, διδασκόμενους ή απλά ενδιαφερόμενους στην ανάπτυξη τέτοιου είδους εφαρμογών, έχουμε εργαστεί πάνω σε ένα θεωρητικό και ένα πρακτικό κομμάτι για την ολοκλήρωση αυτής της εργασίας.

Στο θεωρητικό κομμάτι, γίνεται εισαγωγή σε έννοιες που αφορούν την J2ME και χρήση αυτής της γλώσσας προγραμματισμού για ανάπτυξη εφαρμογών. Αναλυτικότερα, έχετε δεί τον γενικό κύκλο ζωής των MIDP εφαρμογών. Γνωρίζετε πώς να κατασκευάσετε μια διεπαφή χρήστη χρησιμοποιώντας τα υψηλού επιπέδου lcdui widgets, και πώς να αλληλεπιδράσετε χρησιμοποιώντας τον μηχανισμό ακροατή. Έχετε μάθει να δημιουργείτε γραφικά χρησιμοποιώντας το χαμηλού επιπέδου API, συμπεριλαμβανομένων των ανεξαρτήτως συσκευής flicker-free animation και τον συντονισμό του υπολογισμού γραφικών και σχεδίασης.

Στο πρακτικό κομμάτι, έχουμε κατεβάσει και εγκαταστήσει το απαραίτητο λογισμικό, που περιλαμβάνει το Wireless Toolkit, το JCreator καθώς και κάποιες επιπλέον χρήσιμες βιβλιοθήκες. Χρησιμοποιώντας τα παραπάνω, εκτελέσαμε και δείξαμε τα αποτελέσματα όλων των παραδειγμάτων που ήταν απαραίτητα για την κατανόηση όλου του θεωρητικού τμήματος της εργασίας. Τέλος, εγκαταστήσαμε και χρησιμοποιήσαμε την πλατφόρμα Moodle προκειμένου να μπορέσουν να χρησιμοποιηθούν τα παραπάνω υπό την μορφή μαθήματος για eLearning.

ΠΗΓΕΣ

[1] Topley Kim, March 2002, *J2me in a nutshell*, O'Reilly Media, ISBN: 0-596-00253-X

[2] Michael Kenteris - Damianos Gavalas - Daphne Economou, 25 September 2007, *An innovative mobile electronic tourist guide application,* Springer London, Greece

[3] Rifat Shahriyar-Enamul Hoque-Iftekhair Naim-S M Sohan-Mostofa Akbar, February 12-15, 2008, Controlling Remote System using Mobile Telephony, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) ICST, Brussels, Belgium, Belgium Innsbruck, Austria

[4] PAUL COULTON, OMER RASHID, REUBEN EDWARDS, AND ROBERT THOMPSON, Year of Publication: 2005, Creating Entertainment Applications for Cellular Phones,Year of Publication: 2005, Lancaster, U.K.

[5] James White, Year of Publication: 2001, An Introduction to Java 2 Micro Edition (J2ME); Java in Small Things, IEEE Computer Society Washington, DC, USA

[6] Dreamtech Software Team, January 2002, Wireless programming with j2me (Cracking the code), John Wiley & Sons, ISBN: 0-7645-4885-9

[7] 18/08/2009, J2ME: Step by step, ibm.com/developerWorks

[8] http://java.sun.com/

[9] 24/08/2009, Java Essentials: What Is Wireless Java? By Steve Anglin, http://onjava.com/pub/a/onjava/2002/11/06/wireless.html

[10] 17/09/2009, J2ME, Your first cell phone application, http://aiti.mit.edu/courses/labs/B09-J2ME.pdf

[11]11/08/2009, Simple J2ME MIDlet Example,

http://sprintdevelopers.com/node/9

[12] http://www.javabeat.net/

[13] 11/09/2009, Java 2 Micro Edition (J2ME) Application Development : MIDP Programming, http://www.stardeveloper.com/