

Sun Certified Mobile Application Developer (311-110)

Sun Microsystems, Inc.

Exam Notes

Sathya Srinivasan

02/20/2004

Table of Contents

Portability Requirements.....	3
Limitations.....	3
Performance Issues.....	3
High-Level API.....	4
Concurrency.....	4
Portability.....	4
Structure.....	4
Interplay with Application Manager.....	4
Low-Level API.....	5
Font Support.....	5
Repainting.....	5
Coordinate System.....	5
High-Level Event Handling.....	6
Low-Level Event Handling.....	6
Repainting Issues.....	6
Threading Issues.....	6
Classes in javax.microedition.lcdui package.....	8
Displayable.....	8
Display.....	8
Command.....	8
Canvas.....	8
Choice.....	8
Alert and AlertType.....	9
ChoiceGroup.....	9
DateField.....	9
Form.....	9
Gauge.....	10
ImageItem.....	10
List.....	10
Spacer.....	10
TextField.....	10
Difference between high-level and low-level APIs.....	11
Class hierarchy and relationship in javax.microedition.lcdui package.....	12

Chapter 8 MIDP UI API

Given a scenario, develop MIDP 2.0-compliant user interfaces, recognizing portability requirements and limitations (e.g. double-buffering not guaranteed), and performance issues (e.g. using inner classes, freeing memory buffers, etc.).

Portability Requirements

- The display of the application should be device-independent. Since devices have varying screen sizes, care has to be taken so that absolute coordinates are not used, and even if used, they are adjusted taking the screen size of the device into consideration.
- Availability of input devices (for example, pointing devices) should not be assumed as they might not be available in some devices (like cell phones).

Limitations

- Only one `Displayable` may be visible at a given time.

Performance Issues

- `Image` buffers must be used sparingly.
- `Graphics` object must be created once before displaying and should be reused thereafter.

Discuss the MIDP user interface high-level API including concurrency, portability, structure of the API, and interplay with the application manager.

High-Level API

High-level APIs are used to provide access to UI components in a device-independent manner. These APIs are typically used by business-type applications that need forms, etc. for input.

Concurrency

- UI API is thread-safe.
- Implementation generally does not hold locks to visible objects and hence any object can be used as a lock for synchronization

Portability

- High-level APIs are portable across devices since the implementation takes care of rendering.

Structure

- Used by business applications.
- Drawing of components in device display is performed by the implementation.
- Applications cannot define visual appearance of components.
- Primitive interactions like navigation and scrolling are handled by the implementation (like scrolling of a really long `List`).
- Applications cannot access input devices, like keys.
- All `Screen` sub-classes fall under this category.

Interplay with Application Manager

- The following guarantees are given by the Application Management Software (AMS).
 - The `Display.getDisplay()` method can be called from the start of the MIDlet till it is destroyed (from constructor to `destroyApp()` call of the MIDlet).
 - The `Display` object is the same throughout the lifecycle of a MIDlet run.
 - The `Displayable` set using `Display.setCurrent()` will not be changed by the AMS.
- The following behaviors are assumed by the AMS with respect to the application.
 - The application may set the first screen in the `startApp()` method. Since this method can be called multiple times (application may be paused and resumed), initialization should not be performed in this method (done in constructor instead).
 - The application should release as many threads as possible in `pauseApp()` method. If starting with another screen, it is set here.
 - The application should delete created objects using `destroyApp()` method.

Explain the MIDP user interface low-level API including font support, repainting, and coordinate system.

Low-Level API

Low-Level APIs provide access to the more basic features of the device, like drawing on the screen and the screen co-ordinates. These APIs are not necessarily portable for this reason (each device might have different screen size) and hence must be used with caution. Typically, game applications use these APIs.

- Used by game applications.
- Full control of device display is available to the application.
- Access provided to input devices (like key presses and releases).
- `Canvas` (and `GameCanvas`) and `Graphics` fall under this category.
- There are some parts in low-level APIs which are device-independent abstractions and hence portable (like `Key` definitions). It is preferable to use these as much as possible.

Font Support

- Implementation and support for Fonts is device-specific. If the requested type of Font is not available, the nearest Font is returned.
- Fonts can be requested in the following manner.
 - By Size: `SMALL`, `MEDIUM`, or `LARGE`
 - By Face: `PROPORTIONAL`, `MONOSPACE`, or `SYSTEM`
 - By Style: `PLAIN`, `BOLD`, `ITALIC`, or `UNDERLINED`.

Repainting

- The `paint(Graphics g)` method has to be implemented by the `Canvas` sub-class.
- The `repaint()` method queues up a request for repaint. `serviceRepaints()` method schedules the repainting thread.

Coordinate System

- While using coordinates, use the `Canvas.getWidth()` and `Canvas.getHeight()` to find out the screen area of the device and adjust accordingly to provide maximum portability.
- The origin (0,0) is defined in the upper-left corner of the display.
- The width and height of the display can be obtained using `Canvas.getWidth()` and `Canvas.getHeight()` methods. These are **static** methods.
- Pixel ratio is 1:1. That is, gaps between two horizontal and two vertical pixels are the same.

Given a set of requirement, develop interactive MIDP 2.0 user interface code with proper event-handling (including both the high-level and low-level APIs, repainting and threading issues).

High-Level Event Handling

- High-level APIs use `Command` objects and `CommandListener` to listen to abstract actions.
 - `Command` is an abstract action that can be displayed in a device-specific manner by the implementation.
 - `CommandListener` can be used to find out if the user invoked a command, and if so, which command was invoked and act appropriately (like `Cancel Command` being pressed).
 - `ItemCommandListener` can be used to listen to commands invoked on `Item` objects.
 - `ItemStateListener` can be used to listen to state changes in `Item` objects.
- All UI callbacks are serialized.
- `Timer` events are not considered as UI events.
- There can be only one type of listener associated with a 'listenable' object. For example, there cannot be two `CommandListeners` for the same `Displayable` object.
- **CommandListener Interface**
 - `public void commandAction(Command c, Displayable d);`
- **ItemCommandListener Interface**
 - `public void commandAction(Command c, Item i);`
- **ItemStateListener Interface**
 - `public void itemStateChanged(Item item);`

Low-Level Event Handling

- Low-level APIs include events for key presses and key releases.
- Events for pointer actions are available if such an input device is present in the device.
- Key events in `Canvas`
 - `public void keyPressed(int keyCode);`
 - `public void keyReleased(int keyCode);`
 - `public void keyRepeated(int keyCode);`
 - May not be available for all devices.
 - Can be checked using `hasRepeatEvents()` method.
- Pointer events in `Canvas`. This may not be available in all devices.
 - `public void pointerPressed(int x, int y);`
 - `public void pointerReleased(int x, int y);`
 - `public void pointerDragged(int x, int y);`
 - Availability of pointer events can be checked using `hasPointerEvents()` and `hasPointerMotionEvents()` methods.

Repainting Issues

- Repainting is done automatically for `Screen` objects (high-level APIs) but not for `Canvas` objects (low-level APIs).
- In `Canvas`, repainting is done asynchronously.
- Multiple `repaint()` requests can be clubbed together during the actual repaint by the device.
- Repaint is actually triggered only by the `Canvas.serviceRepaints()` call. `repaint()` call is only a request.

Threading Issues

- Caller of the `Canvas.serviceRepaints()` method must not hold any locks needed by the `paint()` method.

- Since `paint()` called by the `serviceRepaints()` method may work on a thread other than this thread, it might want to obtain a lock. If the caller of the `serviceRepaints()` method has the same lock, there will be a deadlock.
- The `Display.callSerially(Runnable r)` method can be called to ensure that the `Runnable` implementor's `run()` method is called in serial order along with other events.
- Typically used by animations to ensure that repaints are properly done.

Identify correct and incorrect statements or examples about the classes (including the class hierarchy) within the `javax.microedition.lcdui` package.

Classes in `javax.microedition.lcdui` package

Displayable

- A `Displayable` can have a title, a `Ticker`, and a set of `Command` objects associated with it.
- When a `Displayable` is made visible using the `Display.setCurrent()` method, the previous one is replaced.
- Title may contain line breaks.

Display

- There is only one `Display` object for a `MIDlet`.
- `Display.setCurrentItem`

Command

- The type of `Command` determines where the `Command` will be placed in the `Screen`.

Command Type	Intent
BACK	Command to return to logically previous screen.
CANCEL	Negative reply pertaining to the current action or screen. For example, if this is a form where user enters data to a data store, data should be discarded.
EXIT	Command to exit the application.
HELP	Command to provide help for a given screen or action.
ITEM	Command pertaining to the currently focussed item.
OK	Positive reply pertaining to the current action or screen.
SCREEN	Command pertaining to the currently displayed screen.
STOP	Command to stop a currently running process, like stopping a network access.

- The priority of `Command` determines the ordering of multiple commands in a `Screen`.
 - The lower the priority value of a `Command`, the higher its importance.

Canvas

- Key constants are defined for the ITU-T keypad (0-9, #, *).
- Key constants are defined for game keys (UP, DOWN, LEFT, RIGHT, FIRE, `GAME_{A/B/C/D}`).
- `Canvas` can have a full screen mode where title and `Ticker` are not displayed.

Choice

- If an element in a `Choice` implementation is too long, it may be truncated.
- A `Choice` can be `IMPLICIT`, `EXPLICIT`, `MULTIPLE`, or `POPUP` (valid only for `ChoiceGroup` and not for `List`).
- The state of a selected `Item` in a `Choice` remains with it even if elements are added or removed in the `Choice`.
- Since `Choice` interface may change in future, this should not be realized directly by an application class.

- The `CommandListener` method should return immediately since commands may be queued and multiple threads might not be used by the implementation, which will result in the method becoming a bottleneck.

Alert and AlertType

- If an `Alert` is set to be a timed alert and too much data is given such that it cannot be displayed fully in a screen (scrolling will be enabled automatically), then the `Alert` becomes a modal alert.
- An `Alert` has an implicit `Command` called `DISMISS_COMMAND`.
 - If more than one `Command` is added to the `Alert`, the `Alert` automatically becomes modal.
 - If all user `Commands` are removed from the `Alert`, the `DISMISS_COMMAND` is automatically added.
 - If an explicit user `Command` is added, then the implicit `DISMISS_COMMAND` is removed.
- A `Gauge` can be added to an `Alert` using `setIndicator()` method to show progress.
 - The `Gauge` added must be non-interactive.
 - The `Gauge` must not be owned by another container.
 - It must not have any `Commands`.
 - It must not have an `ItemCommandListener`.
 - Its label must be `null`.
 - Its preferred width and height must be unlocked.
 - Its layout must be `LAYOUT_DEFAULT`.
- If there is a `Ticker` in an `Alert`, it may not be displayed on all devices.
- If an explicit `CommandListener` is added, then the implicit listener is disabled.
- Timeout in an `Alert` is specified in milliseconds or can be `Alert.FOREVER`.
- `AlertTypes` are `INFO`, `WARNING`, `ERROR`, `CONFIRMATION`, and `ALARM`.

ChoiceGroup

- `ChoiceGroup` is displayed typically as a radio-button (Cell Phone) or drop-down menu (PDA) for `EXPLICIT` type and check boxes for `MULTIPLE` type.
- Elements in a `ChoiceGroup` must be non-null.
- `IMPLICIT` type is not allowed.

DateField

- `DateField` can be in `DATE`, `TIME`, or `DATE_TIME` mode.

Form

- An `Item` can be contained in only one `Form`. Otherwise, `IllegalStateException` will be thrown.
- `Form` layout is similar to AWT `FlowLayout` algorithm. The `Form` grows vertically when new `Items` are added. Typically, more than one element is present in a single row only when there is enough space.

Gauge

- A `Gauge` can be interactive or non-interactive.
- A non-interactive gauge can have a definite or indefinite range. An interactive gauge always has a definite range.
- Gauge update types have special meaning for non-interactive gauges with indefinite range.

Update Type	Example
CONTINUOUS_IDLE	Waiting for network transfer
CONTINUOUS_RUNNING	Waiting for network transfer
INCREMENTAL_IDLE	
INCREMENTAL_UPDATING	Downloading a file of unknown size
INDEFINITE	Waiting for network transfer

ImageItem

- Used to add a mutable or immutable `Image` to a `Form`.
- If an `Image` is mutable, a snapshot of the `Image` at the time of calling `ImageItem.setImage()` is used. This snapshot is not changed even if the underlying mutable `Image` is modified unless an explicit `ImageItem.setImage()` call is made.
- The `LAYOUT_*` constants in an `ImageItem` has been moved to `Item` and is present here only for compatibility.

List

- An item selection in an `IMPLICIT List` can be found by listening to the `List.SELECT_COMMAND` command.
- A `List` can be of type `EXPLICIT`, `IMPLICIT`, or `MULTIPLE`. It cannot be `POPUP`.

Spacer

- `Spacers` are used to space out other `Items` in a `Form`.
- `Commands` cannot be added to `Spacers`. Hence the methods `addCommand()` and `setDefaultCommand()` have been overridden to throw `IllegalStateException`.

TextField

- Display of the contents of a `TextField` may be different from the data. For example, a phone number may be displayed as (123) 456 7890, whereas the data might be 1234567890.
- The text constraints `DECIMAL` is added in MIDP 2.0.
- The `ANDable` text constraints `UNEDITABLE`, `SENSITIVE`, `NON_PREDICTIVE`, `INITIAL_CAPS_WORD`, and `INITIAL_CAPS_SENTENCE` have been added in MIDP 2.0.

Compare and contrast high-level and low-level APIs, including layout techniques.

Difference between high-level and low-level APIs

Feature	High-Level API	Low-Level API
Thread Safety	Safe.	Safe, except a specific case of <code>serviceRepaints()</code> call, which may cause deadlock.
Display	User has no control.	User has full control.
Repainting	Automatic.	Must be implemented by user.
Event Handling	Through Listeners.	Through event callbacks.
Input Device Access	No.	Yes.
Device Independence	Yes.	To an extent.

Explain requirements, issues, class hierarchy, and relationships between items and screens.

Class hierarchy and relationship in javax.microedition.lcdui package

