

JavaOne Hands-On Lab LAB-6400 Create Your Own Mobile Game

Copyright 2008 Sun Microsystems

LAB-6400: Create Your Own Mobile Game



LAB-6400: Create Your Own Mobile Game

Expected duration: 120 minutes

Contacts: Lukas Hasik and Fabiola Rios

Last Updated: April 17, 2008

NetBeans Mobility 6.0.1 introduced various tools for Mobile applications including new Visual Designer as well as Game Builder. Both allows you to visually develop Java ME applications and games for various MIDP-2.0-compliant mobile devices.

In this lab, participants will use the Game Builder to create a mobile game. They will create the menu for managing the application and also the game itself. NetBeans Mobile Pack helps you with connecting your game to an enterprise solution through J2ME Web Services API (JSR 172) for exchanging various data.

In this hands-on lab session, developers will gain an experience with the successful tools, technologies and the best practices to seamlessly build mobile game application using the Java ME and Java EE technologies. The result of the hands on lab will be a real world mobile game which you can take with you in your mobile phone.

Copyright

Copyright 2008 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, Solaris, Java, the Java Coffee Cup logo, JavaOne, the JavaOne logo, and all Solaris-based and Java-based marks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Prerequisites

This hands-on lab assumes you have some basic knowledge or programming experience on the following technologies.

- MIDP/CLDC development
- Mobile Media API (JSR 135)
- Web Services API (JSR 172)

System Requirements

- Supported OS: Windows Vista/ XP, Solaris 10 x86/x64,Linux,Ubuntu7.x.
- Memory requirement: 512MB minimum, 1GB recommended
- Disk space requirement: 650MB minimum, 1GB recommended
- Processor requirement: minimum 800 MHz Intel Pentium III workstation, recommended 1.4 GHz Intel Pentium III or equivalent
- JVM requirement: JDK 1.5_0 or higher

Software needed for the lab

Please download and install the following set of software. If you have any questions on installation, please feel free to send questions to the email alias mentioned below.

- Download and install NetBeans IDE 6.0.1 ALL version, it includes all the Netbeans packs and bundles application server.
 - Select *your platform and language* at the download page

Download and unzip lab file (under a directory of your choice)
 Read <lab_root>/index.html to proceed

Notations used in this documentation

- <NETBEANS_HOME>
 - It is the installation directory of NetBeans IDE with Mobility Pack
 - For Windows, the default is C:\Program Files\netbeans-6.0.1
 - For Linux, the default is ~/netbeans-6.0.1
- <APPSERVER_HOME> directory where you installed Sun Java System Application Server (or glassfish)
- <lab_root> directory into which lab zip file is unzipped
 - This document uses <lab_root> to denote the directory under which you have unzipped the lab zip file of this handson lab. The name of the lab zip file of this handson lab is 6400_mobilegame.zip.
 - Once you unzipped the lab zip file under <lab_root>, it will create a subdirectory calledmobilegame. For example, under Windows, if you have unzipped the lab zip file in the root of drive E:\, it will create E:\mobilegame. Under Linux if you have unzipped the lab zip file in the /home/sang directory, it will create /home/sang/mobilegame directory

Lab exercises

- Exercise 0: Install and configure the lab environment
- Exercise 1: Creating a MIDP application using visual tools
- Exercise 2: Create the game
- Exercise 3: Logic of the game
- Exercise 4: Make it connected

Resources

- Lab presentation
- Mobile Applications Learning Trail

Where to send questions or feedbacks on this lab and public discussion forums

- You can send technical questions via email to the authors of this Hands-on lab (and experts on the subject) or you can post the questions to the web.
- You can send your other questions to the public mailing lists and forums.
 - Mobility users mailing list archive
 - Forum: Java ME Technologies CLDC and MIDP
 - JavaOne2008 forum at hands-on-labs.dev.java.net

Exercise 0: Install and Configure Lab Environment

Please, install the full NetBeans 6.0.1 IDE. If you don't plan to generate mobile client to webservice then the NetBeans Mobility 6.0.1 distribution is enough. If you plan to connect the mobile application to server backend then you need the Mobility End to End. The easist way is to install the **All** distribution of the NetBeans 6.0.1.

• Download the NetBeans IDE

Go on to next exercise

Exercise 1: Creating a MIDP application using visual tools

Expected duration: 20 minutes

Introduction:

The goal of this exercise is to let you experience the features of NetBeans Mobility by building and running a simple mobile application and modify the flow of the game application that you will use later to build a mobile game.You will learn following things in this exercise:

- How to create a new Mobile project
- How to build and run the project
- Add Custom Component to the palette.
- How to modify the flow of the Visual game application

Background information:

NetBeans Mobility makes it easy for all programmers to start developing for the Java ME platform quickly, even if you've only had Java SE programming experience. If you are programming mobile devices for the first time, you can just think of Java ME as having a more limited version of the Java SE API, and let NetBeans Mobility Pack take care of the foreign Java ME issues. Though there are more differences than just a smaller API, it's not necessary to know all deeps before producing a working mobile application. To help you get started, here are some terms that are used throughout this document:

- Java ME Java Platform, Micro Edition is the Java platform meant to run on small devices. Currently, these devices are typically cell phones or PDAs, but Java ME is also used on other embedded systems. A configuration, profile, and optional packages are what compose a Java ME platform.
- CLDC Connected Limited Device Configuration is a Java ME configuration that is currently most often used on mobile phones. It contains a runtime environment and a core API that are appropriate for the limited processor speed and memory size of mobile devices.
- MIDP Mobile Information Device Profile is the set of APIs that provide higher level functionality required by mobile applications like displayable components and network communication.
- MIDlet A class required by all MIDP applications. It acts as the interface between the application and the device on which it is running. Similar to a main class in a Java SE project.
- Emulator a tool which can run your mobility applications. The emulator usually mimics the behavior of the real phone it is emulating.
- Platform a Java development kit used to build and run the project in the NetBeans IDE. For Mobile
 project this usually means emulator, building tools and various JARs that implement Java ME API.
- WTK Sun Java Wireless Toolkit for CLDC 2.5.2 a reference implementation of Java ME's CDLC
- configuration and MIDP profile. Bundled with NetBeans Mobility. Also includes the various building tools.
 JSR172 J2ME Web Service API. Optional package of J2ME defining API for connecting to a web service.

Steps to Follow:

- Create a Mobile Project.
- Build and run the project
- Add Custom Component to the palette.
- Define the flow of the game application

Step 1: Create a Mobile Project

The first step for any application you develop in NetBeans is to create a project. The project contains all the files that make up the mobile application.

- 1. To create a new project select **File > New Project** or press New Project toolbar icon.
- A New Project dialog box appears, asking you to choose project category and type. Select Mobile category and MIDP Application project.
- 3. Click Next and the wizard continues by asking to name the project. Name this project Hello.
- Make sure Set as Main Project and Create Hello MIDlet check boxes are marked as selected and click Next.Click Finish to create the project. Your project appears with the HelloMidlet open in the visual designer(Figure-1-1).



Figure-1-1: HelloMidlet in the visual designer

- 5. Select Alert component from the Displayables category in the Palette Window.
- 6. Drag&Drop the Alert screen from palette to the Visual Designer Flow view.
- 7. Component named **alert** appears (Figure-1-2). By this simple operation you just added a screen component to the design.



Figure-1-2: Alert component added to the flow

- 8. Now change the title of the alert, so it says "Help" to do so you can simply click on the **Title** property (in the properties window located in the right-bottom corner of the IDE) and type in the text.
- 9. The next step is to change the text the alert displays to the user e.g. "This is Help for Hello Midlet". Switch to the Screen designer (using **Screen** button).
- 10. Select the Alert component in combo box at the top of the document window. It will show the alert component in the Device Screen area of the document.
- 11. Move the mouse over the edit screen, over the < text not specified > area and a blue box should appear, informing you that the text can be edited. Clicking on the blue box. In the editor type the new text and press **Enter**.
- 12. Switch back to the flow designer and drag **Help Command** from the palette and drop it on the **form** component. **Help** message is added to **form** component.
- 13. The next task is to connect this help command with our help component (alert) you've just created. Select *helpCommand* in the *form* component and drag it to the *alert* component (you should see a line with arrow while dragging) and drop it on the **alert** component.
- 14. The components should be now connected together (Figure-1-7 Alert component is able to automatically return to the component from which it was displayed, thus two lines were actually created one from form to

alert and the second from alert back to form).



Figure-1-3: alert component is linked with Help command on helloForm.

Step 2: Build and Running the Project

- Run the application with Run > Run Main Project or simply by clicking on green arrow or hitting F6 key. This saves all files, builds the project (you should see progress of building in output window) and starts the emulator with the HelloMidlet.
- 2. Launch the application, and see the "Hello" form (Figure-1-4), press the Help button. Don't forget to close the emulator when you are done with the application (you can exit the emulator either with close button on the emulator's window or by hitting power-off button in the top-right area, just above the display).



Figure-1-4: Emulator running the HelloMidlet

If the emulator doesn't start up then it might be problem with path to the java. Please, look at the FAQ describing the problem - <u>WTK and default java</u>.

Step 3: Add Custom Component to the palette.

In this step you are going to register a canvas component to the component palette, so it can be used in the visual designer. You can find components provided by MIDP and custom components of the NetBeans Mobility in the palette. The screen components can be used to visualy design a flow of an application. Other components will be used to create user interface of particular screens. You can use your own components to design your application. A custom component has to be added to the palette before it can be used in the visual designer.

- Now you are going to use a project, which contains some functionality already. Open the project by choosing File > Open Project. A filechooser dialog appears.
 Select project MobileGameApp in <lab_root>/exercises/exercise1/ directory and ensure that Open as
- Main Project is checked.
- The project is one of the original appears in **Projects** tab in the IDE. Make sure the project is set as main (project node should be in bold. If not, right-click on the project node and choose **Set as Main Project** from pop-up menu to make it main).
 Expand the Project node >Source Packages>org.netbeans.j1.game>GameMiDlet.java and double click on
- the file. It will open the flow design of our game application.(Figure-1-5)

						-																
		Mohile	Dovi	ico		- - -			• • •		. 8	51	rgGan	neSpla	shSc	reen	. [⊒ ·				
		WODIN						:						Comr	mands	3						
	04-	ut a st	WIDIE	÷L				112			E	ISMI	SS C	OMMA	ND			1	124			
	Sta	nea					Ð.,								e							
:	Res	sumed																	:D			
		· · · · ·					· · · · ·						:						1	: . J	· · · ·	
															i 👘							
												4.			1							
													. 8		÷ _							
											60/3	svgN	ainM	enu		1						
												SVG	Menu	Eleme	ent		-				-	
											mei	nulter	n_0									
							· · · · ·			6	mei	nulter	n_1						:		· · · ·	
						•			•		mei	nulter	n_2						1			
											me	nulter	n 3									
														. 8								
			 	()								11		11 † 1					:)**		r -	
8	ve s	vgspia	snsc	reenA	noar																	
			Comr	mand	S									-							1	
L C	DISMI	ss_co)MMA	ND			5	8														
							 9					111										
		💑 S1	/gGar	neSc	ores¥	VaitSo	:reen		_	1												
				С	omma	ands									forml	JserA	ndSco	ore	_			
		SUCC	ESS	COM	MANE)				<u>.</u>						Com	mand	s				
		FAILU	RE C	- COMM	AND									okC	omma	and1				8		
	L													can	celCo	mmar	nd			8		
																1	1		1			
														:								
							· · · · ·														· · ·	
		· ·					-	-	-				-					-			-	
										•••												
							· · · ·														·	
																					-	
Figure	-1-5:	Gamel	MIDle	t in th	e flow	desid															· · · ·	

5. To add a custom component to the palette select Tools > Palette > MIDP Visual Mobile Designer. This brings up the Palette Manager (Figure-1-6).

🗊 Palette Manager	
Pale <u>t</u> te Content:	
 Palette Displayables Ommands Elements Elems Flow 	Add to Palette Move Up Move Down
 Image: Image: Im	Remove New Category
	Reset <u>P</u> alette
	⊆lose

Figure-1-6: Palette Manager

- 6. In the dialog, click on Add to Palette button. A wizard for adding new component to the palette appears.7. On the first page, select the project from which you'd like to add the component. In our case, MobileGameApp project should be selected.
- Click Next and select a class to add. By default this dialog shows only components which have sources in the select project, select the class org.netbeans.j1.game.MazeCanvas (Figure-1-7).
 In this dialog you can also specify the palette category to which the component is going to be added. For custom components, please select the Custom Components node.

Add to Palette wizard		\mathbf{X}
Steps 1. Select Project 2. Select Classes	Select Classes Found MIDP Classes - select those you want to add: org.netbeans.jl.game.MazeCanvas	
	< Back Next > Einish Cancel Help	

Figure-1-7: Registering custom component

10. Close the palette manager. MazeCanvas component is now available for the visual design (Figure-1-8).



Figure-1-8: MazeCanvas component on the Palette

Step 4: Define the flow of the game application.

In this step you are going to modify the UI of the application that will show a mobile game. When you add/fix the whole game UI and add the logic into the application, the Visual Flow Design of the application should behave in following way. When the user finish to play the game, the application will register his scores and it will require him to enter the userID. It will keep his/her scores and add it into a scores table. This table will display the top10 scores of the best players. If the user decides to improve the score he/she will have the option to update the previous score, in case that it will happened the new score will be reflected in the score table, and return to the main menu, or if he/she decides to exit the application the emulator will be close. For more details about the Visual Flow Design component (and other components) see Javadoc for NetBeans MIDP components and tutorials available on http://www.netbeans.org/kb/trails/mobility.html.

1. Go to the **Flow Designer** of the GameMIDlet. Drag&Drop the **MazeCanvas** component from palette to the Flow Designer. Component named **MazeCanvas** should appear in flow design. By this simple operation you just added a screen component to the design. Create flow of the screens in the application. In the Flow design window, click on the **menuItem_0** and drag the arrow to the **MazeCanvas** component. New transition between screens is created.(Figure 10)



Figure-1-10: Final design of the flow of the MazeCanvas application

2. Select MazeCanvas component in the Flow Design, go to properties window and click on a custom editor button of the **midlet(1.contructor-1.parameter defined** property (a small button with three dots). This opens a window with embedded editor, which lets you to write a code to be executed. In this editor, you need to change the default value <null> to **this** value (Figure-1-2)

🗊 Java Ex	pression:			×
Java Expres	sion:			
this				
	ОК	Reset to) <u>d</u> efault	Cancel

Figure-1-11: Java expression

3. The next task is to write the code, which will start the game thread. To add this functionality you need to click the Source button at top of GameMIDlet document window. The source-code editor should open Now find method svgMainMenuAction - you can either use CTRL+F or double click the method name in Navigator at left bottom corner of the IDE.

If you have problem to find the right place where to write (or paste) the code you can use **Window > Tasklist** window. Where are listed all TODOs in this hands-on lab.

4. In the method change the code in following way



Summary:

In this exercise you learned how to rapidly create mobile application from scratch using visual designer. You utilized standard MIDP components as well as additional components supplied by NetBeans Mobility 6.0.1 external project, which included a custom visual component. You learned how to add a Canvas into the custom components.

Go on to next exercise

Exercise 2: Create the game

Expected duration: 25 minutes

Introduction:

In this exercise, we'll look to the most important blocks to build up the Game UISprites, TiledLayers and Scenes. NetBeans Mobility provides set of tool for creation of this UI blocks for the game developers.

- How to create a new Visual Game Design file that encapsulates the graphics of the game
- Create new Scene
- Create new TiledLayer
- Create new Sprite

• How to design look of individual objects

Background information:

Visual editing support for **MIDP 2.0 Game API** including creation and editing of Sprites, TiledLayers and their arrangement into scenes. The main goal of the **Game Builder** module is to reduce the effort involved in creating common game components.

- **Sprite** Game Builder treats sprites as a collection of animated sequences where each sprite can be contain a number of sequences. The sprite editor allows editing and previewing of multiple animated sequences simultaneously.
- **TiledLayer** Grid of cells, each painted with one frame selected from an Image. When a new tiled layer is created it can be edited using the available visual editor. This allows the user to modify the contents of the tiled layer and receive instant visual feedback. The tiled layer editor also includes support for animated tiles.
- Scene Scenes are locations within a game environment that have distintive visual, and possible audio, characteristics that define the games look and feel. Each level in a game can have several different scenes. Conversely, a simple game may have only one scene for all of its levels.

Steps to Follow:

- Create a new Visual Game Design file
- Create new Sprites
- Create new TiledLayer
- Create new scene for our game
- Modify the background

Step 1: Create a new Visual Game Design file

In this step you will create a new Visual Game Design file, which will be developed in the Game Builder.

- 1. Right click on the package **org.netbeans.j1.game** and select **New > Visual Game Design**. A **New File** wizard for the visual game appears.
- In this wizard enter the name of the GameDesign (this name will be visible to user when launching the application) for example MazeGameDesign. Click Finish to create the visual game. When created, the file should be opened in
 Game Builder (Figure-2-1).



Figure-2-1: MazeGameDesign in the Game Builder

Step 2: Create new Sprites

Sprites are the build brick of any MIDP game. Game Builder treats sprites as a collection of animated sequences where each sprite can be contain a number of sequences.

- 1. We will create new sprite. You can create it in following ways:
 - Click Create Sprite text in the GameDesign file opened in editor. The Game Builder view has to be active.

• Click Create Sprite icon in toolbar of the GameDesign file opened in editor.

Use the /duke.png image. Adjust the **tile size to 25 pixels** with the sliders for both - width and height. Name the Sprite as **dukeS**. Click **Finish** button.

New	sprite 🗙
Sprite name: dukeS	
Select image: /bigG.png	Adjust tile size in pixels:
/duke.png /thumbsUp.png /wall.png	
Import sample images	Tile width: 25 px Tile height: 25 px
	OK Cancel <u>H</u> elp

Figure-2-2: Create new sprite

New sprite with default sequence opens in the editor.



Figure-2-3: Default duke sprite sequence

2. Change the look of the sprite sequence to look like at Figure-2-4. You can drag and drop images to desired position in the sprite sequence from the down part of the Game Builder view where you can see all the available tiles for the sprite. When you right click a frame in the sequence you can remove the frame.



Figure-2-4: Duke sprite sequence

3. We need to create one more Sprite in our game in the same way as the previous one. This time we will name the sprite **jamesS**. The image that we will use is *bigG*.

When the jamesS sprite is created it should look like on Figure-2-5.



Figure-2-5: James Gosling sprite

- 4. We need more sequences for our duke sprite. The duke can walk, he can fall and also jump. We have to create new sequences. Open the dukeS sprite in the Game Builder. Right click on any frame of the default sequence dukeSseq001.
- 5. Choose **Create sequence** from the popup menu. Name the sequence *dukeSfalling*. Create one more sequence with name *dukeSjumping*.



Figure-2-6: Sequences

Step 3: Create new TiledLayer

TiledLayer are the build brick of any MIDP game. When a new tiled layer is created it can be edited in the visual editor. This allows the user to modify the contents of the tiled layer and receive instant visual feedback. The tiled layer editor also includes support for animated tiles.

1. Create new TiledLayer. You can invoke the New TiledLayer dialog in different way. Name the TiledLayer as **maze1**. Use image wall. Adjust the tile size to the maximum heigth and width.

		New tiled layer		
TiledLayer name:	mazel			
Select image:		Adjust tile size in pixels:		<u>Z</u> or
/bigG.png /duke.png /thumbsUp.png /wall.png				
Import sample in	nages	Tile width: 63 px	Tile height: 64 px	
			ОК	Cancel <u>H</u>

Figure-2-7: New tiled layer

2. Press Finish button. Empty grid appears.



Figure-2-8: Empty tiled layer

3. You can create the tiled layer by drag and drop of images from bottom area to the grid. The IDE will construct the tiled layer table for you. Create a maze of square shape. Build up the walls of the maze. The maze should be square shape.



Figure-2-9: Example of maze walls

Step 4: Create new Scene

The scene editor is provided to organize all layers (sprites and tiled layers) together into scenes.

- 1. Create new Scene. Name the Scene as level1. Click Finish button. Empty scene grid appears.
- 2. Right click the grid. Choose Add tiled layer > maze1 from the popup menu. The maze1 appears on the grid.





3. The *maze1* tiled layer appears on grid in positions where you right clicked the mouse. We need it in 0,0 position. You can either move it the with mouse or you can set position in **Navigator** view in right bottom corner of IDE.



Figure-2-11: Setup of position

 Right click the grid again. Choose Add sprite > jamesS from the popup menu. Add dukeS sprite into the scene too. Place both the sprites to not collide with the maze walls.



Figure-2-12: Final look of the scene

When placing the sprites of duke and James into the maze remember the purpose of the game - duke is looking for James.

Summary:

In this exercise you learned how to create new sprite, tiled layer. Please, note that you can design these sprites and tiled layers in visual tool. The sprites and tiled layer are defined by arrays of positions of a tile in the png image of the sprite. NetBeans Mobility generates these boring arrays for you. If you want to see the generated source code switch to **Source** of the MazeGameDesign.

You also learned how to create sequences of sprite and how to compose the sprites and tiled layers into a scene with NetBeans Mobility.

Go on to next exercise

Exercise 3: Logic of the game

Expected duration: 20 minutes

Introduction:

The logic of our game is quite simple. There is the Duke creature walking in a maze. The Duke is looking for James Gosling who is lost somewhere in the maze. When he finds James the game ends. We created the sprites in previous exercise. We also created one level of our game.

Duke can move left and right. He can jump. When there is no floor under his feets then he starts falling. The code for movement of the duke is based on the example provided by Carol Hamer in <u>"Creating Mobile Games:</u> <u>Using Java ME Platform to Put the Fun into Your Mobile Device and Cell Phone" by Appress</u>.

Steps to follow:

- Initialize the MazeManager
- · Let the duke move

Step 1: Initialize the MazeManager

1. There are following classes created for you in the **org.netbeans.j1.game.logic** package that take care about processing the game. The GameTread.java class runs in separate thread to not block the game ui. The MazeManager.java class takes care about the interaction among the sprites and tiled layers in the game.

At first we have to initialize all the game parts. Copy and paste code in bold to init method of the MazeManager class.

```
public void init() throws Exception {
   //TODO: [Exercise 3][step 1] - initialization code. Please, don't forget to fix the missing
           mazeDesign = new MazeGameDesign(); //[Exercise3]
             jamesSprite = mazeDesign.getJamesS();//[Exercise3]
             dukeSprite = mazeDesign.getDukeS();//[Exercise3]
               dukeSprite.defineReferencePixel(dukeSprite.getWidth()
                                                                          1
                                                                              2
                                                                                  0]
              dukeSpriteAnimator = new SpriteAnimationTask(dukeSprite,
                                                                                 fal
            myWalls = mazeDesign.getMaze1();//[Exercise3]
               mazeDesign.updateLayerManagerForLevel1(this);//[Exercise3]
          timer = new Timer();//[Exercise3]
                timer.scheduleAtFixedRate(dukeSpriteAnimator,
                                                                         mazeDesigr
                                                                     0,
   // this sets the view screen so that the player is
   // in the center.
   myViewWindowX = dukeSprite.getX() - ((DISP_HEIGHT - SQUARE_WIDTH) / 2);
   myViewWindowY = dukeSprite.getY() - ((DISP HEIGHT - SQUARE WIDTH) / 2);
   System.gc();
                 }
```

2. Fix the missing import for MazeGameDesign class. You can use the hint bulb to fix the code. Just press the bulb on the glyph or ALT+ENTER on the line with errors.

Hints Figure-3-1: Hints

- 3. Create new field mazeDesign in the class. You can use the hint again.
- 4. Now the game is ready to be played. You can run the application. Right click the **MobileGameApp** project and choose **Run**.
- 5. When the emulator window appears launch the application.



Figure-3-2: Launch game

Figure-3-3: Splash screen



Figure-3-4: Main menu

Figure-3-5: Game running

Step 2: Let the duke move

We need to set up the right sequence to the duke sprite when he is walking, jumping or falling.

- It can be done in MazeManager class in method updateSprite. Just uncomment the four lines ending with //TODO: [Exercise 3][step 2]. It will add the code that changes the sequence of tiles for duke sprite. Therefore the duke will either walk or fall or jump.
- 2. The source code of the updateSprite() method will look in following way. The bold lines are the uncommented ones.

```
private void updateSprite(int hdirection, int vdirection) {
    // if the duke is moving left or right, we set
    // his image to be facing the right direction
    if (hdirection > 0) {
        dukeSprite.setTransform(Sprite.TRANS_NONE);
    } else if (hdirection < 0) {</pre>
        dukeSprite.setTransform(Sprite.TRANS_MIRROR);
    //in case that duke is moving, change the showing sprite image
    if (lastHorDirection != hdirection) {
     dukeSprite.setFrameSequence(mazeDesign.dukeSseq001); //TODO: [Exercise 3][ste]
    }
    if (lastVertDirection != vdirection) {
        if (vdirection > 0) { //he's jumping
         dukeSprite.setFrameSequence(mazeDesign.dukeSfalling);//TODO: [Exercise :
        } else if (vdirection < 0) {</pre>
          dukeSprite.setFrameSequence(mazeDesign.dukeSjumping);//TODO: [Exercise 3
        }
   }
    if ((vdirection != 0) || (hdirection != 0)) {
        dukeSpriteAnimator.forward();
    } else {
       dukeSpriteAnimator.setMoving(false);
          dukeSprite.setFrameSequence(mazeDesign.dukeSseq001); //TODO: [Exercise 3
    }
    lastHorDirection = hdirection;
    lastVertDirection = vdirection;
}
```

Summary:

You created necessary logic for the game. You can now play the game. As you can see the work with the Sprites and TiledLayers is quite simple thank to the Game API. The game logic isn't so complicated however the description of it is beyond scope of this hands-on lab class. The source code is well described with comments and javadoc therefore you can study it later.

Go on to next exercise

Go on to summary

Exercise 4: Make it connected

Expected duration: 20 minutes

Introduction:

We created the mobile game. In this exercise we will enhance it to share the high scores with other users. It common feature of games in todays mobile market. Players like to compete and it's better to compare your results with real players.

We will use Java ME Web Service API (JSR 172) to connect to the web service that collects the high scores. The same web service also provides the latest results to be displayed in the High Score table.

The NetBeans Mobility includes a wizard for generating the JSR172 stubs for the mobile client that will connect to a web service.

In this step you will not enhance the game functionality itself but you will make the application connected to a server part.

Steps to follow:

- Create new configuration
- Generate stubs

Step 1: Create new configuration

In this step you will create new configuration in your project. It will allow us to have two version of the same project. First one will be used for the online game and the second one for offline game.

- 1. Right click the MobileGameApp project. It will open the project Properties.
- In the project properties dialog, select Add Configuration... from Project configurations the combo box. It will open Add Configuration dialog.
- 3. Type in the name for new configuration OnlineGame. Finish the dialog.

Category: Category: Category: Category: Category: Category: Category: Category: Category: Category: Platform Abilities Application Desc Category: Compiling Compiling Compiling Compiling Creating JAF Creating JAF Compiling	Add Configuration Use Configuration Template: Configuration Template: Configuration Template: Configuration Templates Configuration templates Configuration templates New Configuration Name: OnlineGame	
Generating 1	OK Cancel Help	
🔂 Creating JAP 🌛 Signing 🙀 Generating J 🕨 Running	OK Cancel Help	

Figure-4-1: New configuration dialog

- 4. In project properties, select Abilities. Uncheck the Use Values from DefaultConfiguration.
- 5. Click Add button. In Add Ability dialog provide name ONLINE and value true. Click OK to finish dialog.

MobileGameApp			
Project Configuration: OnlineGame		•	Manage (
Category:			
	Use Values from	"DefaultConfiguration"	
Platform	Abilities:		
Application Descriptor	Ability	Value (optional)	
Build Add Ability	1001/11	1.0	x
Error Compiling Ability Name:	ONLINE		-
Obfuscating Value (optional): G Creating JAR	: true		
Generating Ja		OK Cancel He	elp

Figure-4-2: New ability dialog

6. Click OK to close the project properties dialog.

Step 2: Generate stubs

In this step you will generate a mobile client to the web service. We will use the wizard provided by NetBeans Mobility.

- 1. Select File > New File... from main menu. In the New File dialog select MIDP category and Java ME Web Service Client. Make sure that *MobileGameApp* project is selected in **Project** combo box. Click Next button.
- You have to specify an URL of webservice's WSDL file in this panel. The WSDL can be either a file stored locally on your hard drive or an URL of a webservice at the internet.
 - Our webservice should be running at following URL :

http://193.86.76.42:8080/DukeHighScores/HighScoresService?WSDL

If you are not able to access the web service WSDL try to check your network settings. Sometimes a firewall or antivirus shield can protect you from connecting to the web service. If the webservice URL is unreachable you can deploy the webservice to application server by yourself. The project with the webservice is at <lab root>solutions/solution4 directory. The name of the project is DukeHighScores.

- 3. Select the Running Web Service radio button.
- 4. Type the URL of the webservice to **WSDL URL** text field. Click **Retrieve WSDL** button and, if the WSDL file name automatically appears in the Local Filename field, the WSDL file is successfully downloaded and validated.

The validation of the WSDL is mandatory step. You cannot generate the Java ME Web Service Client to any webservice. There are some restrictions for the webservice defined by the JSR 172. The webservice has to be WS-I Basic Profile compliant. There is no support for asynchronous messaging. No support for SOAP messages with attachments. Support only for the literal representation of messages. No support for extensible type mapping. The wizard will report an error message when you cannot generate the J2ME webservice client for a webservice.

 The wizard will provide you a default name for the client and package. Change the Package name to org.netbeans.j1.game.client

0	New Java ME Web Service Client	thatile yo		at dealry the whiservice :	to apply -
5	teps	Java ME Web	Servi	ce Client Information	
1 2	 Choose File Type Java ME Web Service Client Information 	Select the sour Running We WSDL <u>U</u> RL: Specify the	rce of t eb <u>S</u> erv local f	the WSDL description for the web service to be adde vice :8080/DukeHighScores/HighScoresService?WSDL ilename for the retrieved WSDL.	d to this project. Retrieve <u>W</u> SI <u>P</u> roxy Setting:
		Local Filena	me: DL File a <u>m</u> e:	HighScoresService.wsdl	Browse
		<u>C</u> lient Name:	Highs	ScoresService	
		Project:	Mobil	leGameApp	
		P <u>a</u> ckage:	org.r	netbeans.j1.game.client	
		Created File:	ise3\	MobileGameApp\src/org/netbeans/j1/game/client/Hig	ghScoresService.w
		C Generate D	<u>i</u> ataBin	ding structures	
				< <u>B</u> ack Next > <u>Finish</u>	Cancel <u>F</u>

Figure-4-3: Java ME Web Service Client wizard

6. When you press Finish button the wizard will generate web service client stubs.



Figure-4-4: Generated stubs

Step 3: Use mobile client

You generated the stubs that will allow you communicate with the web service on a server. It is time to use the web service client in the game source code.

1. Add following code to the GameMIDlet.java source code. You can add it to the end of the file.

```
//TODO: [Exercise 4][step 3] add the getMobileClient() method
HighScoresService client = null;
    public HighScoresService getMobileClient() {
        if(client == null) {
            client = new HighScoresService_Stub();
        }
        return client;
}
```

This will add new method for retrieving the webservice stub.

2. Fix the imports.

3. Update the getAddHighScoreTask method to upload the scores to the server.

```
public SimpleCancellableTask getAddHighScoreTask() {
          if (addHighScoreTask == null) {
              // write pre-init user code here
               addHighScoreTask = new SimpleCancellableTask();
                 addHighScoreTask.setExecutable(new org.netbeans.microedition.util.Exec
                   public void execute() throws Exception {
//#if ONLINE == "true"
                       //TODO: [Exercise 4][step 3] set High score
                      //upload the score to WS
                               getMobileClient().setHighScore(getNameTextField().get
                        //get the high scores from WS again to synchronize with server
                             updateHighScoresTable(getMobileClient().getHighScores()
//#else
//#
                           //just add the score to the scores table
//#
                              addScoreToScoreTable(lastHighScore, getNameTextField().ge
//#endif
                       //anyway update the High scores table
                       updateSvgWithHighScores();
                 }
             });
         // write post-init user code here
        }
          return addHighScoreTask;
    }
```

4. Update the getHighScoresTask method to download the scores from the server.

```
public SimpleCancellableTask getHighScoresTask() {
         if (highScoresTask == null) {
              // write pre-init user code here
               highScoresTask = new SimpleCancellableTask();
                highScoresTask.setExecutable(new org.netbeans.microedition.util.Execut
                  public void execute() throws Exception {
                       // write task-execution user code here
//#if (ONLINE == "true")
                      //TODO: [Exercise 4][step 3] get High Scores
                       //connect to WS and get the latest scores
                              updateHighScoresTable(getMobileClient().getHighScores
                       //update the high score table
                           updateSvgWithHighScores();//update the svg image with
//#else
//#
                           //this code is used when the ONLINE ability value isn't tru
//#
                           //nothing to do. The high scores are in the highScores Value
//#
                         Thread.sleep(500);
//#endif
```

```
}
}
;
});
// write post-init user code here
}
return highScoresTask;
}
```

Step 4: Run the game

We finished the creation of the game. Even if there is just one source code we created two versions of the game. One is the OnlineGame that connects to server. The other one is the origin source code that works offline.

The online version enables you to share the high scores with other players.

1. Switch active configuration to DefaultConfiguration in configuration combobox in middle of toolbar.

Switch configuration Figure-4-5: Switch configuration

- 2. Run the project. It will run the offline version of the game.
- 3. Switch active configuration to OnlineGame in configuration combobox in middle of toolbar.

Switch configuration Figure-4-6: Switch configuration

4. Run the project. It will run the online version of the game.

Summary:

You connected the game with the server side. It is very easy to use the generated stubs. The application is now able to upload your result to server and display the list of best results on your device.

We used J2ME Webservices API to achieve this functionality but the J2ME Webservices API is not supported on lot of device in real world. It is easy to change the client calls to solution not using J2ME Webservice API. You can use the second wizard that NetBeans Mobility provides - "Mobile Client to Web Application". The difference between these two solutions is described in "LAB-6340: <Advanced Java ME Programming - Streaming Video From Server to Your Device>" Hands-on lab

Go on to summary

Congratulations! You have successfully completed the "LAB-6400: <Create Your Own Mobile Game>" Hands-on Lab.

Where to send questions or feedbacks on this lab and public discussion forums:

- You can send technical questions via email to the authors of this Hands-on lab (and experts on the subject) or you can post the questions to the web.
 - Please post questions that are relevant only to this hands-on lab. • JavaOne2008 forum at hands-on-labs.dev.java.net
- You can send your other questions to the public mailing lists and forums.

 - <u>Mobility users mailing list archive</u>
 <u>Forum: Java ME Technologies CLDC and MIDP</u>