Game Development Workshop
Lab 1

Welcome to our first programming lab. In this lab we are going to learn the essentials, and basics of game development using Java. More importantly, we are going to practice programming concepts we learned in the first introductory programming course through engaging exercises. This lab assumes that you are using a machine with Java Development Kit installed and a Java Integrated Development Environment (IDE) such as Eclipse. If not, please go over the document on how to install Java.

In this lab we are going to start implementing a 2D fighting game. We are going to program some features of Mortal Kombat, a popular game developed by Midway in 1992. Throughout the labs, we will describe the game in more details and implement more features from it. For this lab we will be mainly focusing on setting up our game environment:

1. Set up the start screen with the basic graphical elements.
2. Add basic flashing animation to the start screen.
3. Add basic keyboard event handling.

In this lab we are going to use the following programming features to develop our game:

- **Variables**: to store values such as integers, Boolean, and strings.
- **Boolean logic** to allow us to represent game logic, or the ability to ask questions.
- **If statements**: to provide our code the capability to choose between one or more options.
- **I/O**: we will use the keyboard for input, and the screen for output.

INTRODUCTION TO JAVA

Let us review some of the features in Java. We will mainly concentrate on the concepts we need in the lab.

VARIABLE DECLARATION:

When you need to declare a new variable in Java you need to specify first its type, the value is optional:

```
// Java
int x;
// or
int x = 0;
```

USE OF SEMICOLON:

Most instructions in Java need to terminate using a semicolon, there are special cases where semicolon is not used:

```
// Java Assignment Statement
x = 64;
```

STRINGS:

Strings in Java can only be enclosed using double quotes, other programming languages might allow either single or double quotes:

```
// Java
String str = "This is " + "a string value";
```

GLOBAL VARIABLES:

In C++ or Python we declared and initialized our global variables directly in the global scope. In Java we declare our global variables (this is an imprecise Java terminology, we will introduce the correct one when we cover OOP) in one place and initialize their values in another place.

```
// Java
public int score;

public void init() {
     // …
     Score = 0;
}
```

Notice that we only specify the type of the variable when we declare the variable, and not when we initialize it.

IF STATEMENTS:

The if/else statement in Java is similar to other languages like C++ or Python:

```
int score1 = 29;
int score2 = 32;

if (score1 == score2)
{
     System.out.println("The two scores are equal.");
}
else
{
     System.out.println("The two scores are different.");
}
```

STYLE OF PROGRAMMING:

Java tries to enforce Object Oriented Programming (OOP). We will learn about OOP in the next lab. Until that time, you will notice that Java requires more code to accomplish similar things in C++ or Python. You only need to understand certain parts of the code, later will understand the rest of the code. In the lab, you do not need to worry about any Java code that lies below the comment:

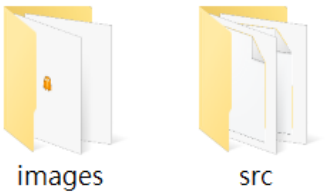
```
/*************************************************************************************
   // This part should not be modified
 * >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>*/
```

Let us next briefly discuss the features that are unique to Java Things that we will cover.

1. Opening a new project:
    a. Empty project
    b. Project with existing startup code
2. Floats vs doubles
3. Block of code
4. Local vs global variables

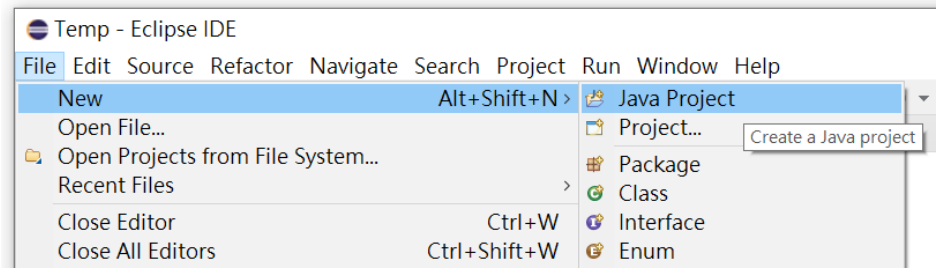
OPENING THE LAB WITH JAVA & ECLIPSE

The first step is to download the provided zip file and unzip it. In this demo, it is unzipped at "C:\Mortal Kombat\Java\Lab 1". You may unzip the provided zip file at a suitable location on your computer. Then, you will have two folders: ***images*** and ***src***.

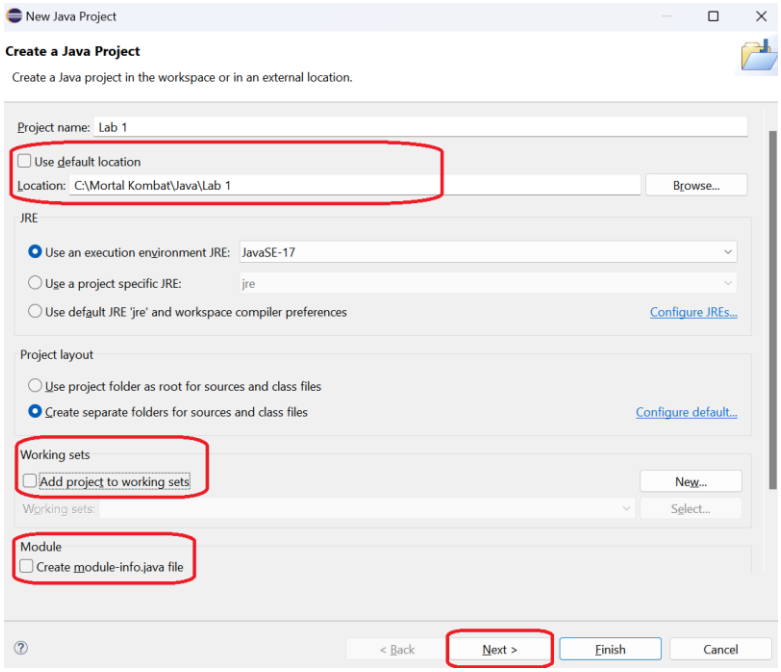


- "images" is where we store the images of the game project. You may add or modify the images in the future assignments.
- "src" is where we store the source code of the game project.

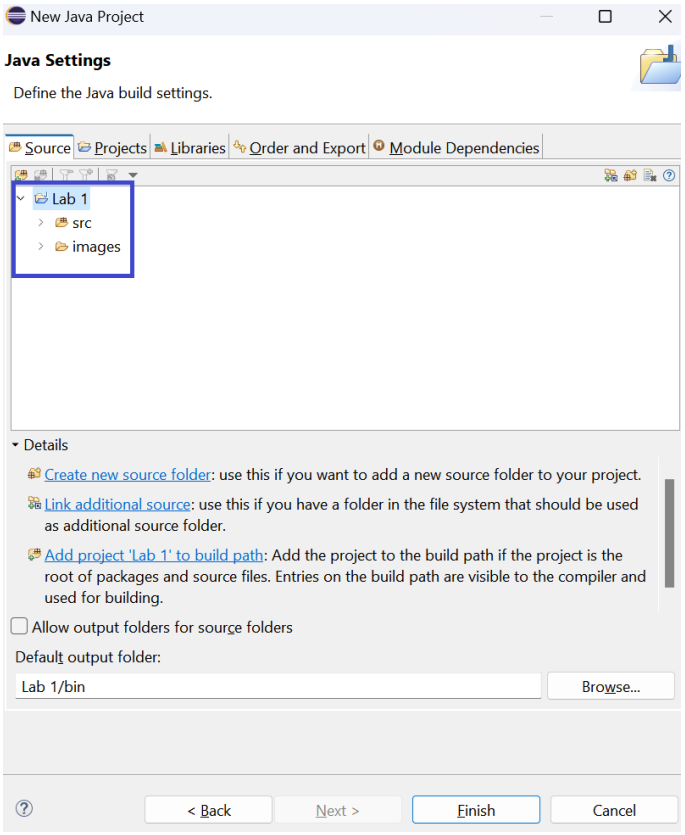
Let us run Eclipse. If you have used Eclipse before, it will automatically open the last project that you wrote. This time, let us create a new project. Choose File - New - Java Project.



In the next window, instead of using the default location, we will browse the location where Lab 1 is located. Uncheck "Use default location", click on the "browse" button, find where you unzipped Lab 1 folder is located, and choose it. Do not choose images or src. If you have done so correctly, the project name will become "Lab 1" automatically. Make sure under Module, to uncheck "Create module-info.java file".

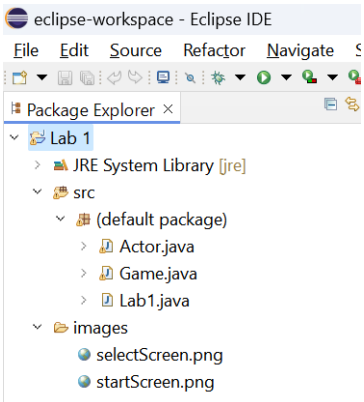


Click on "Next >". You will be directed to the next window.



Click on "Finish".

After you created your new project, make sure that you have added the Java programs, and the *images* folder so you can run the program. If you have done all the previous steps correctly, you will see that they are already there in the Package Explorer.



You can always right click inside the Package Explorer to add new files or to delete old files from your project. You can also double click on any java file to open it.

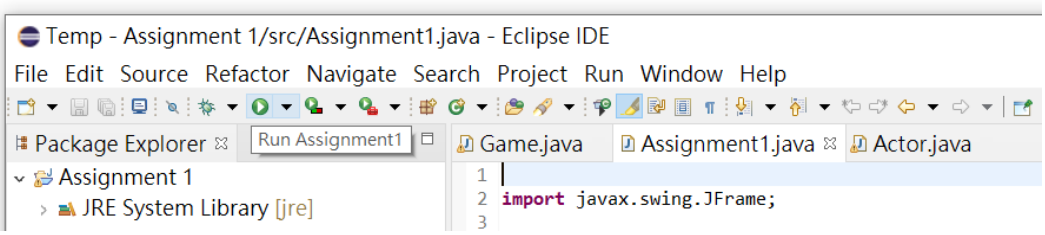
1 GAME DEVELOPMENT IN JAVA

We can easily build simple 2D games in Java with only what we learned in our first introductory programming course, which is a cool thing to do. The provided startup code will help us start with our game so let us explain it a little bit.

Let us open the provided Lab 1 project using the Eclipse IDE. Our lab should have three Java files:

- Lab1.java is where the *main* function is located. The main function is the starting point of our program, and it will call the functions from the two other java files. You **should not modify** this file.
- Actor.java is where the actor class is defined. You **should not modify** this file.
- Game.java is where we write the code of our particular video game. You are expected to add code to this file.

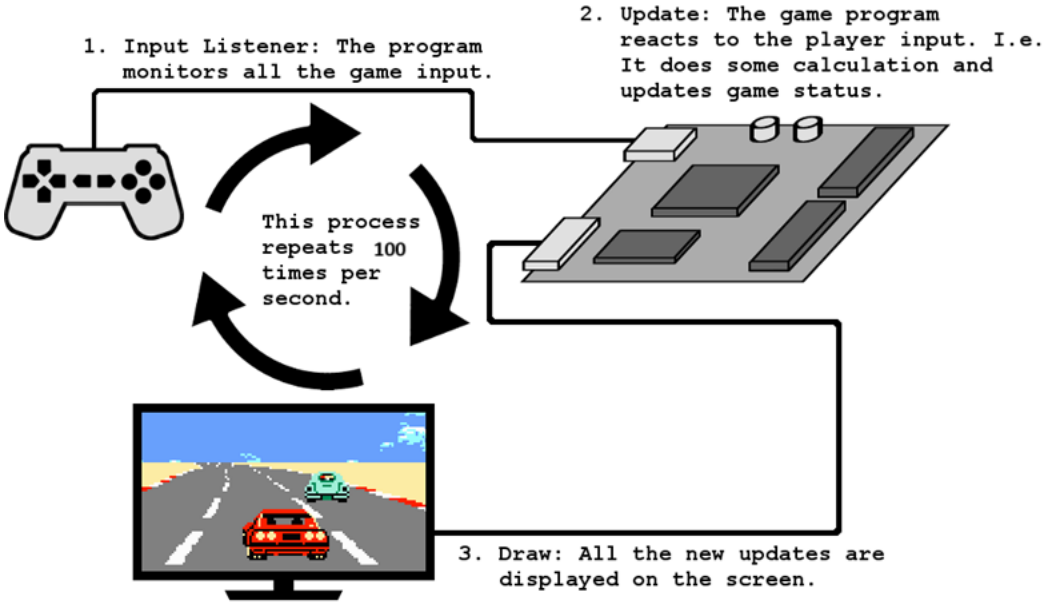
In this lab, you only need to modify **Game.java**. You should leave Actor.java and Lab1.java unmodified. Let us have a test run of the game of Lab 1. Open Lab1.java, and run it by clicking on the run button in the tool bar or the menu bar.



When we run the program, initially the screen is empty and we get the following:

After we complete the lab, the screen will look the following:

For our game to run properly, it needs a game loop:



This type of programming is called *Event-Driven Programming* because the program (our game) is constantly listening to any input (events) by the player and reacts to it, and responds as output.

Game.java has four important sections:

(A) Variable Declaration: This is where we will declare our variables that can be accessed and modified by the rest of the code. We give the variables initial value in the *init*() function.

(B) The *draw* function. This function is automatically called by the game loop, and is responsible for drawing the screen contents. Currently, the function only has two instructions that setup the font on the screen.

(C) The *keyPressed* function. This function is automatically called whenever the player presses a keyboard button.

(D) The *update* function. This function is automatically called by the game loop. This is where the game calculates and updates the game variables (such as to the score, character position and status).

In any game we develop, we need to worry about five things:

1) The screen settings that the game will be hosted on.
2) The background images, these images will be fixed on the screen.
3) The text (of type string) that will be displayed on the screen.
4) The sprites or moveable game objects.
5) Input handling such as getting the input from the keyboard.

2 THE SCREEN

We control the dimensions and main properties of the screen through three variables: TITLE, HEIGHT, and WIDTH. TITLE stores a string value that will be displayed on the Title bar. HEIGHT, and WIDTH are integer variables that represents the height and width of the screen.

EXERCISE I:

We already declared these variables in lines 13 to 15. Let us give them values:

1. Initialize the TITLE variable with a value " MK Lab I".
2. Initialize the two variables called HEIGHT, and WIDTH. The game screen should 960 x 500.

DRAWING ON THE SCREEN

The *draw* function is responsible for drawing all the objects on the screen, including text, background, and sprites. Let us first look at the instruction that draws background images:

```
blit(screen,"imageName", x,y);
```

The *blit* function requires 3 parameters: the screen, a string that represent the name of the picture located in the *images* folder, followed by the *x* and *y* positions.

We use the `drawString` function to draw text on the screen:

```
screen.drawString("text", x, y);
```

The function requires 3 arguments: the string to be printed, and the *x* and *y* position as integer.

POSITION ON THE SCREEN

The smallest unit of measurement for a graphical object is a pixel. When we want to insert information on the screen, usually we want to know the exact pixel location. We can

approximate the location of an object on the screen without knowing its exact pixel position by estimating the position with respect to the height and width of the screen. For example, if we want to have an image in the middle of the screen, then we can set its *x* value to WIDTH/2 (or 0.5 * WIDTH), and the *y* value to HEIGHT/2 (or 0.5 * HEIGHT).

EXERCISE II:

1. Modify the *draw* function so the screen now looks the following:

2. Declare a global Boolean variable called *startScreen*. Initially this variable has a true value.
3. Modify the *draw* function so the start screen is drawn only when the *startScreen* variable is set to true.

3 SPRITES (ACTORS)

One of the important elements in a video game is its characters, whether they are enemies, or the main player. What is unique about these characters is the ability to move, and do various things throughout the game. These are referred to *Sprites*, and they constitute all of the parts of a game that can change their position. In this class, we will call sprites *Actors*. In this lab we do not have actors but we will see them in the next lab.

4 GAME INPUT

We want the game to transition from the start screen to the player selection screen when the player presses the enter button.

EXERCISE III:

1. Make sure that the value of *startScreen* is initially set to true.
2. Modify and complete the *keyPressed* function so it properly handles the enter button:
```
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_ENTER)
```
3. Modify the *draw* function so the player selection screen is drawn only when the *startScreen* variable is set to false.

5 BASIC ANIMATION

Next, we want the text *PRESS ENTER* in the start screen to have a flashing appearance. This feature is usually used to catch the player's attention and to clearly pinpoint on what they have to do.

EXERCISE IV:

1. In the *draw* function, we set the font color to green. Instead of setting the color directly as a value, modify your code so that you use a global variable to store the font color. Then, set the color using that variable.
2. In the *update* function, modify the font color to black.
3. In order to implement this feature, we will take advantage of how frequent the update function is called by Java. We are going to use a counter variable to keep track of time Remember that the update function is called 100 times a second. Therefore, if we make the *update* function increment an integer variable by one, then we end up with 100 increments a second.
    a. Declare a global variable called *colorCounter*, with an initial value of 0.
    b. In the *update* function, increment the *colorCounter* by one. This should **only be done** when the game is in the start screen.
4. Modify the *update* function so that the *text* flashes four times a second.