LAB 7

This lab continues the case study of Simplified Blackjack with amendments of design. The student has to demonstrate their skills and understanding with the implementation of interface class, static methods and attributes and updating UML design.

*By: Emma McKay-Fikri*

Lab Guide and Worksheet

BITP 3113 Object Oriented Programming

Table of Contents

[Learning Outcomes 2](#_Toc401667574)

[Java Packages and API 2](#_Toc401667575)

[import Statement 5](#_Toc401667576)

[package Statement 6](#_Toc401667577)

[Organizing Java Project Using Eclipse 7](#_Toc401667578)

[Creating Java Project 7](#_Toc401667579)

[Creating Package 8](#_Toc401667580)

[Creating Java Class 9](#_Toc401667581)

[Coding Java in Eclipse 10](#_Toc401667582)

[Executing Java Program 12](#_Toc401667583)

[static 12](#_Toc401667584)

[final 13](#_Toc401667585)

[Case Study 14](#_Toc401667586)

[Exercise 1 14](#_Toc401667587)

[Exercise 2 14](#_Toc401667588)

[Exercise 3 15](#_Toc401667589)

[Exercise 4 16](#_Toc401667590)

[Exercise 5 17](#_Toc401667591)

[Exercise 6 17](#_Toc401667592)

[Exercise 7 17](#_Toc401667593)

[Exercise 8 18](#_Toc401667594)

[Exercise 9 19](#_Toc401667595)

[Exercise 10 20](#_Toc401667596)

[How are you doing? 21](#_Toc401667597)

# 

# 7Advanced Class Features

# Learning Outcomes

At the end of this lab, the student is able to:-

* Use Java API documentation
* Organize Java project using Eclipse
* Use Java package, static and final properties

# Java Packages and API

Java provides a set of reusable classes and interfaces for application development which is known as Application Programming Interface (API). API is a collection of classes and interfaces that are group into packages according to its categories, functionalities or reusability. A package is similar to a folder. Figure 1 shows core java packages for Java API. The packages provides classes for various types of application for example network programming (net), database programming (sql), graphical user interface (awt) and distributed application (rmi).

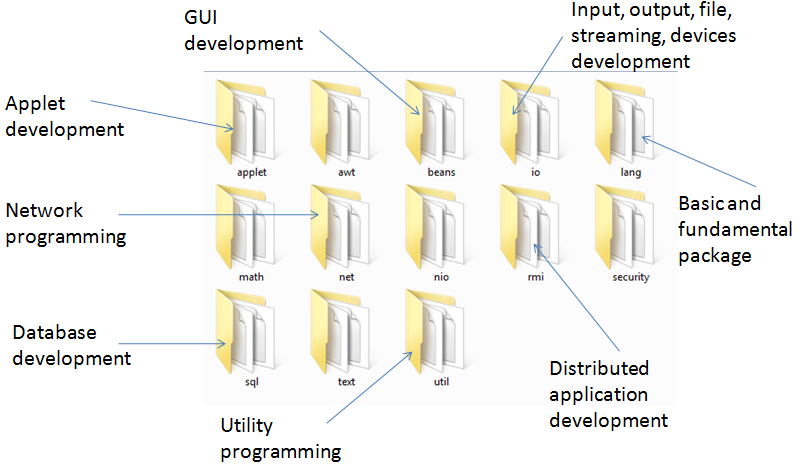


Figure : Core packages in Java

The packages consist of subpackages and compile units of classes, interfaces and abstract classes. These description of the packages and its content is describe in the Java API documentation which can be found online at <http://download.oracle.com/javase/7/docs/api/> and downloaded from <http://www.oracle.com/technetwork/java/javase/downloads/index.html#docs>. Figure 2 shows a screenshot of Java API documentations.

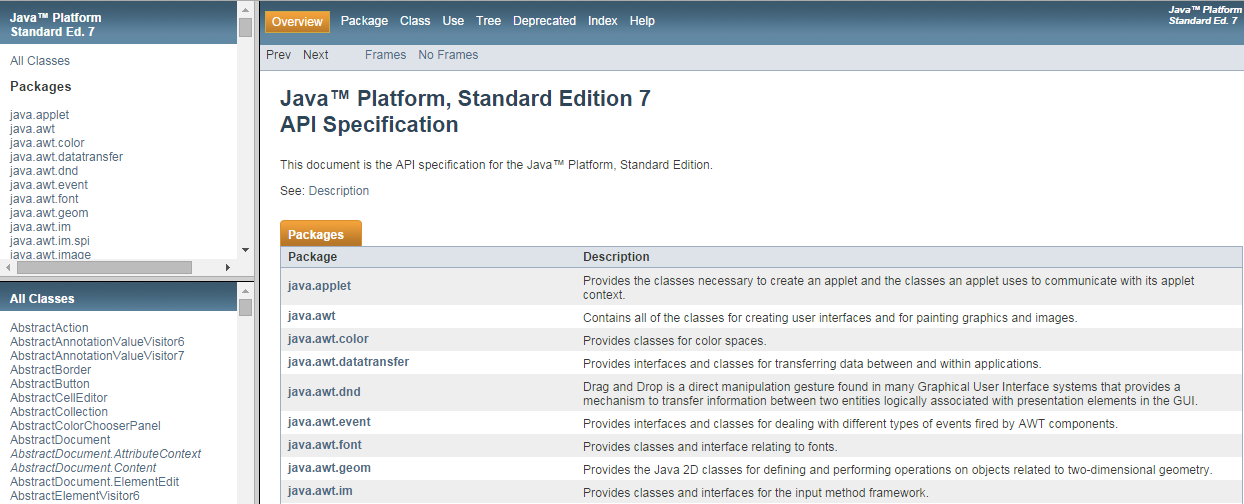


Figure : Java API SE 7 Specifications

All the classes are provided with hyperlinks to allow the user to view the specification of a class. Figure 3 shows description for Class String. It describes the purpose of the class and examples of object manipulation from the class.

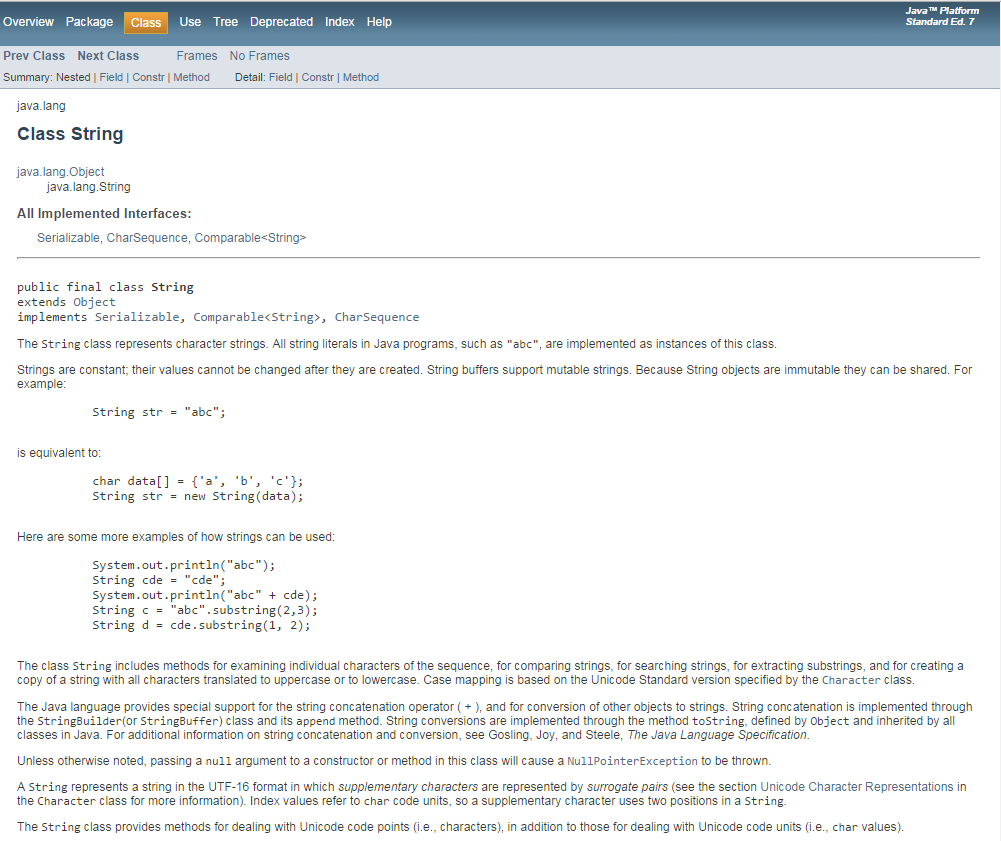


Figure : Class String description

The specification also consist detail description properties, constructors and methods. Figure 4 and Figure 5 shows list of constructors and methods to create and manipulate a String object.

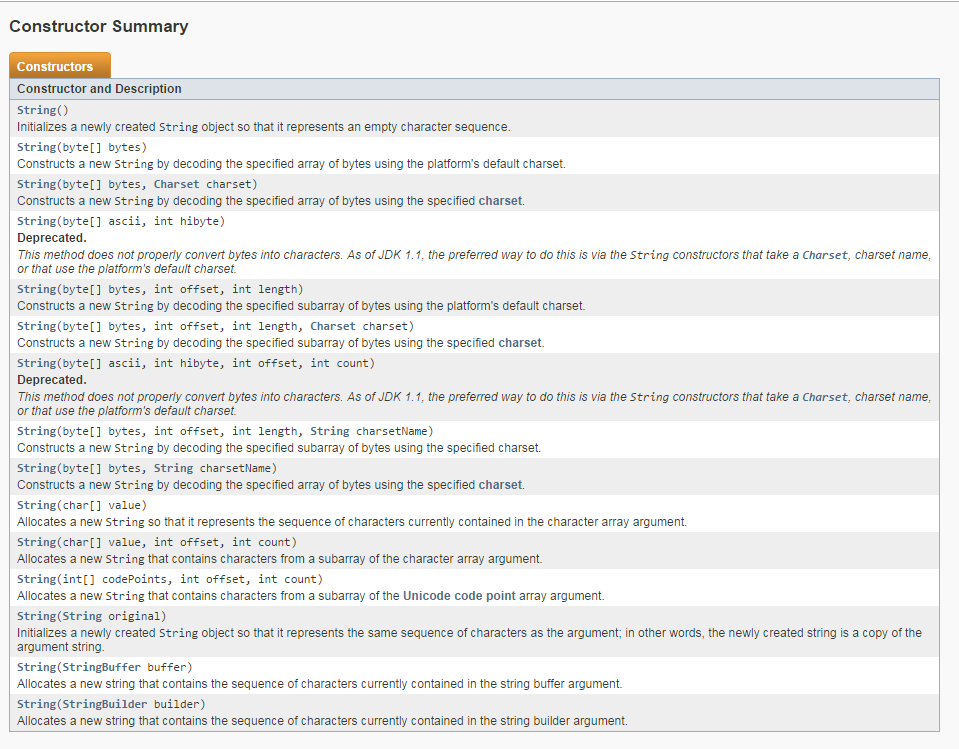


Figure : Constructors of Class String

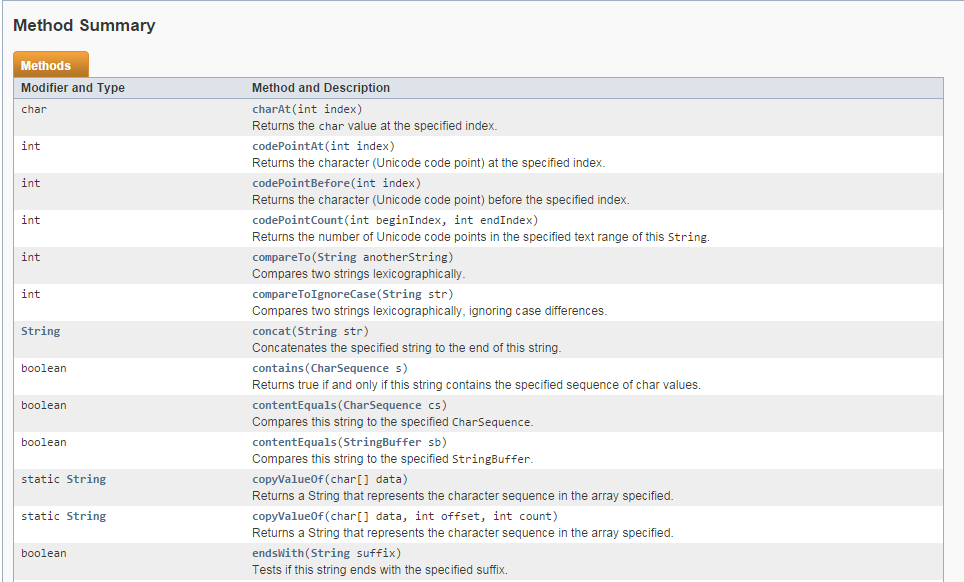


Figure : Methods of Class String

# import Statement

An import statement is used when the class to be used in the current program is located in other package. Figure 6 shows the syntax to use an import statement. import statements must be written before class declaration.

|  |
| --- |
| **import *packageName.className*;** |

Figure : Syntax of import statement

Figure 7 writes Hello World into a text file named output.txt. The program uses File class and PrintWriter class – these classes are provided by Java API. File and PrintWriter located in io package. Therefore import statement is used in this program.

|  |
| --- |
| import java.io.File;  import java.io.PrintWriter;  public class DemoFile {  public static void main(String[] args) throws Exception {    PrintWriter pw = new PrintWriter(new File("output.txt"));  pw.write("Hello World");  pw.flush();  pw.close();  }  } |

Figure : DemoFile imports class File and class PrintWriter

lang package is automatically included in the program. Therefore any of the classes in lang package without additional statement. String is class that belongs to lang package.

# package Statement

Java classes are organized into packages (whether it is for API or not) for ease of maintenance. A programmer can define his own package and group all the classes into one. A package statement tells the JVM where the class should reside. The statement should be written before any import statement. Figure 8 shows syntax of package statement.

|  |
| --- |
| **package *packageName*;** |

Figure : Syntax of package statement

Figure 10 shows an example of the package statement usage. HelloWorld.java must be saved in a folder named world as declared by the package statement.

|  |
| --- |
| package world;  public class HelloWorld {  public static void main (String args[]) {  System.out.println("Hello World");  }  } |

Figure : HelloWorld.java in package world

# Organizing Java Project Using Eclipse

Eclipse is an Integrated Development Environment (IDE) used to manage various kind of software development project for various platform. Eclipse is a free IDE which you can download from [www.eclipse.org](http://www.eclipse.org). Managing Java project using Eclipse is easier than using Notepad. Compilation and execution of Java program can be done within Eclipse IDE.

## Creating Java Project

|  |  |  |
| --- | --- | --- |
| 1. Launch Eclipse IDE. 2. Select File > New > Java Project. A new window as shown Figure 10 in will be displayed on the screen. 3. Enter Project Name. For example MyProject. 4. Click Project from existing workspace. Workspace is the place where all the .java and .class files will be stored. Specify the location using Browse button. 5. Click Finish. 6. MyProject will appear in Package Explorer. Note that if you have created other project previously, it will appear in the Package Explorer as shown in Figure 11. |  | Figure 10: New Java Project |

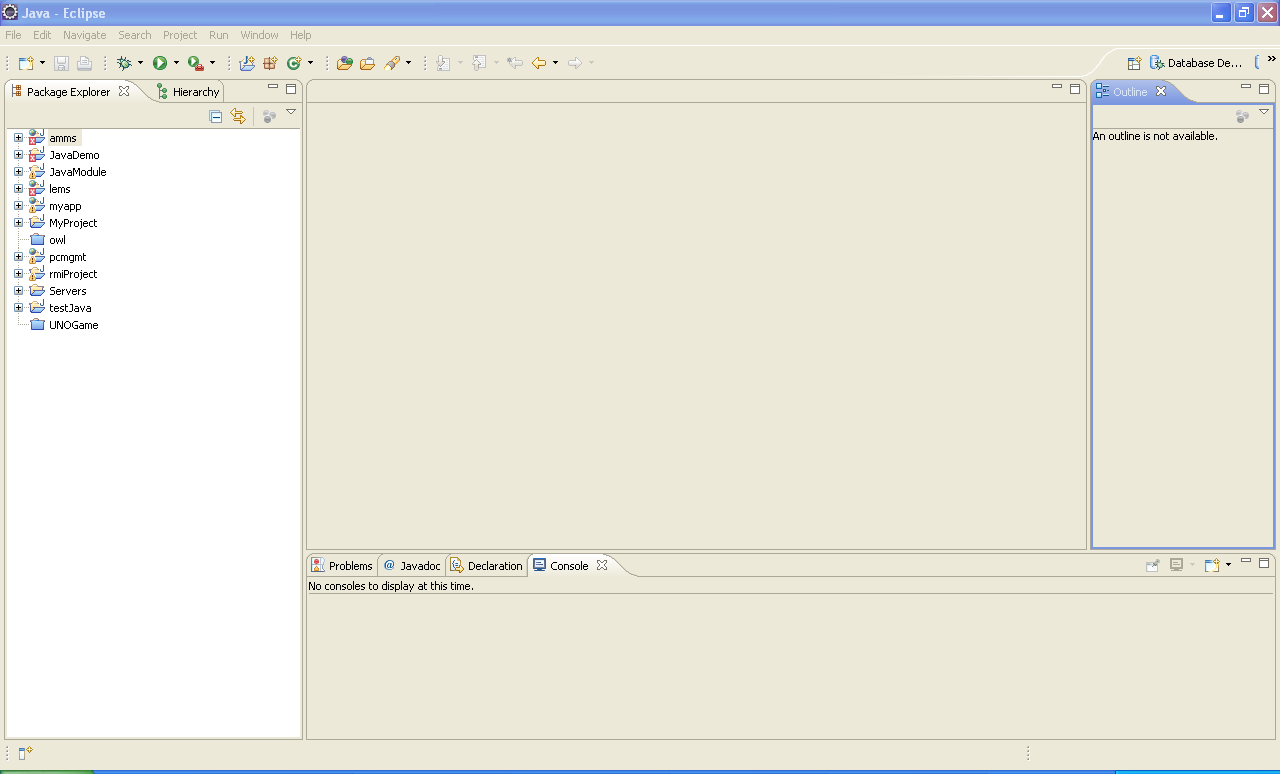


Figure : Eclipse’s Workspace

## Creating Package

1. Right click on MyProject in Package Explorer.
2. Select New > Package as shown in Figure 12. A new window named New Java Package as shown in will be displayed on the screen as shown in Figure 14.

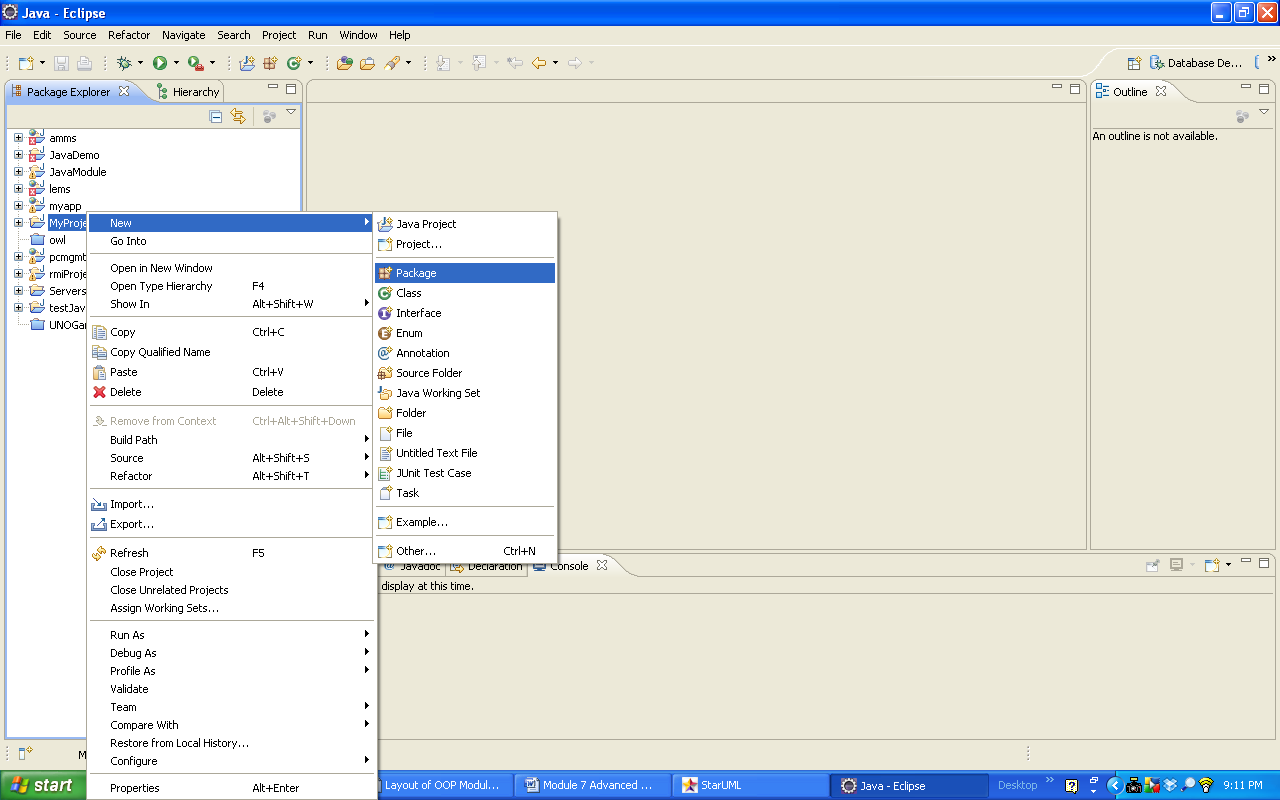


Figure : Menu to create new package

|  |  |  |
| --- | --- | --- |
| 1. Enter Package Name. For example world. 2. Click Finish. A new package named world will be created in MyProject as shown in Figure 13.     Figure 13: New world package |  | Figure 14: New Java Package window |

## Creating Java Class

|  |  |  |
| --- | --- | --- |
| 1. Right click on world packege in Package Explore. 2. Select New > Package as shown in Figure 15. A new window named New Java Class as shown in Figure 16 will be displayed on the screen.     Figure 15: Menu to create new class   1. Enter Class Name. For example HelloWorld. 2. Check on public static void main(String[ ] args) as shown in Figure 16. 3. Click Finish. A new class named HelloWorld will be created in world package and the new class will be displayed in the workspace as shown in Figure 17. |  | Figure 16: Window to create new Java class |

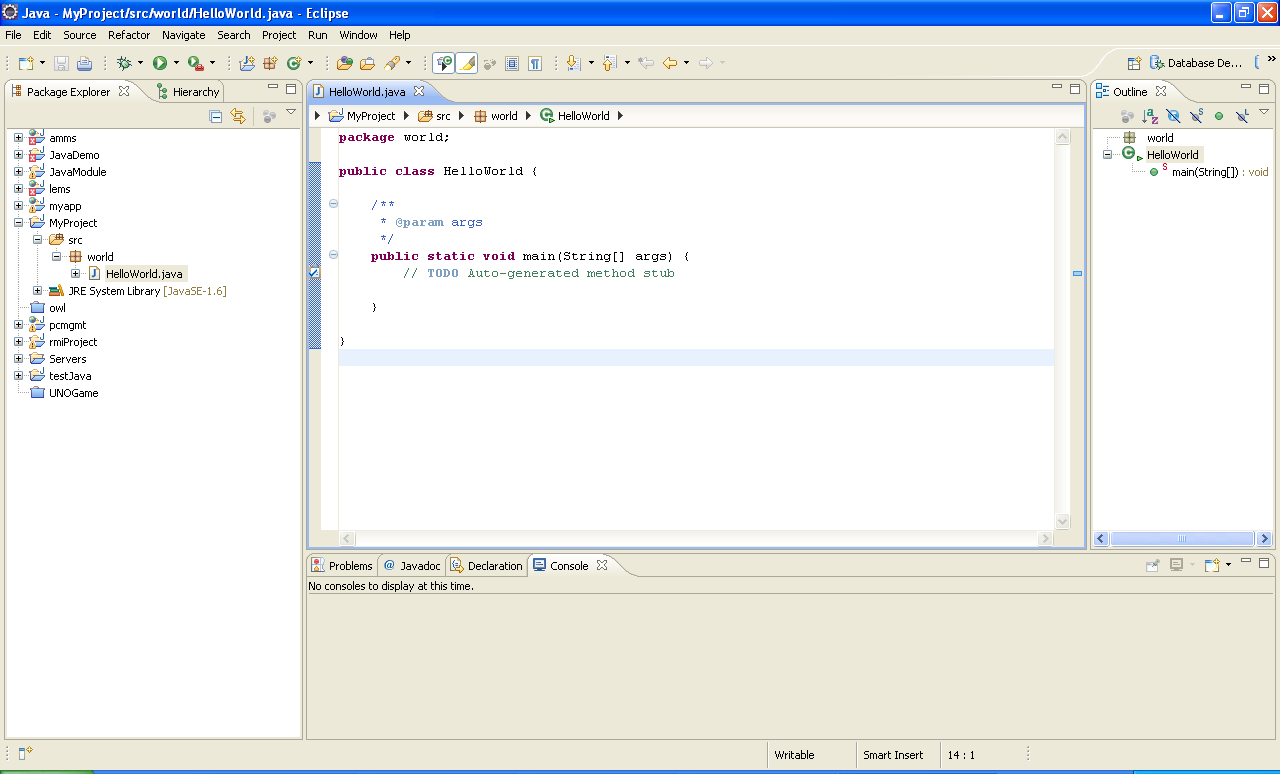


Figure : Class HelloWorld with main() method is created

## Coding Java in Eclipse

1. Write System.out.println(Hello World) in main() method body.
2. Click  (Save button) on the toolbar to save the class.
3. Notice that there is a curly line at the end of the statement as shown in Figure 18. This indicates an error and need to be fixed.
4. You can view the details of the error in Problem tab at the bottom of the Eclipse workspace as shown in Figure 18.
5. Put a semicolon to fix the error and click Save. Notice that the red curly line is missing and the error message in the Problem tab has been clear.

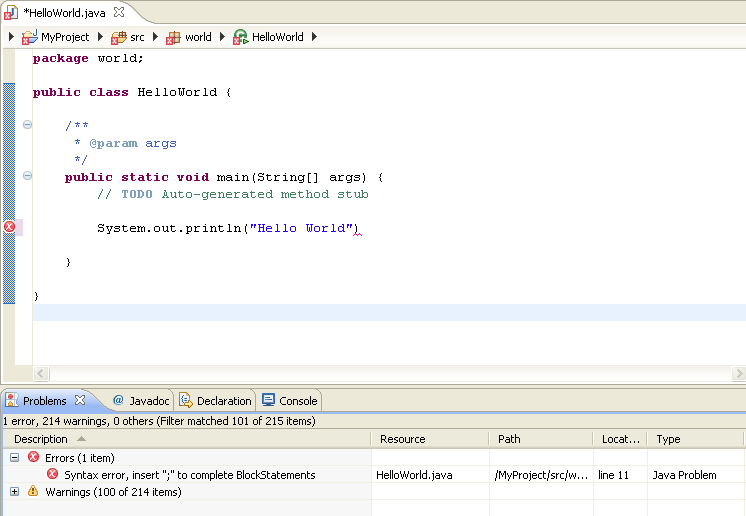
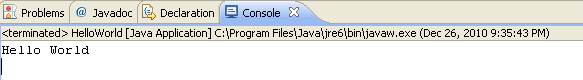


Figure : Writing Java codes for class HelloWorld

## Executing Java Program

1. Click  to execute HelloWorld.
2. The output will be displayed in Console tab as shown in



# static

static is a keyword applied at attribute or method level. A static attribute is known as class attribute. The syntax to declare a static attribute as shown in Figure 19.

|  |
| --- |
| **[accessSpecifier] [static] type attribute;** |

Figure : Syntax to declare static attribute

A static method is a class method. The syntax to declare a static method as shown in Figure 20.

|  |
| --- |
| [accessSpecifier] [static] returnType methodName ([list of parameters]) {  } |

Figure : Syntax to declare static method

A static attribute can be access using class name. The purpose is to give uniform value to all copies of object. For example, class Color in java.awt. package has a number of static property. Figure 21 shown the correct way to access static property in Color.

|  |
| --- |
| color = Color.White; |

Figure : The correct way to use static variable

The same way is applied to access static method. For example, method getColor() in class Color is a public static method. Use class name to call the method as shown in Figure 22.

|  |
| --- |
| Color currentColor = Color.getColor(“White); |

Figure : The correct way to use static method

# final

final is a keyword used at class, attribute and method level. The syntax of final is shown in Figure 23.

|  |
| --- |
| [accessSpecifier] [final] className {  [accessSpecifier] [final] type attribute;  [accessSpecifier] [final] returnType methodName ([list of parameters]) {  }  } |

Figure : Syntaxes used to declare final class, attributes and methods.

There are several implications using final to the class, attribute and method.

1. A final class cannot be extended from a subclass.
2. A final attribute is similar to a constant variable which the value cannot be change.
3. A final method cannot be overridden.

# Case Study

The Simplified Black Jack application has become bigger. The System Analyst has organized all classes into several packages. The new design as in Figure 24.

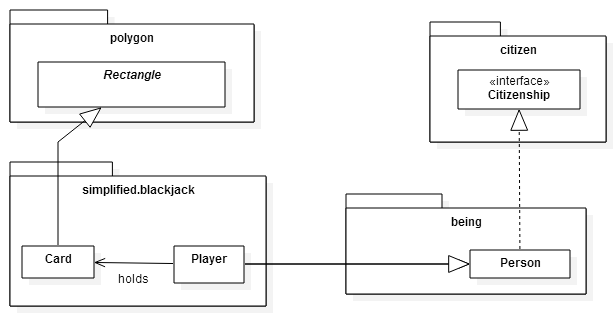


Figure : Packages for Simplified Black Jack application

## Exercise 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Implement the new design in Figure 24 using Eclipse. 2. You might need to add more codes to the exiting classes and fix all errors. 3. Save the classes and record the result in Table 1. | Table 1 : Compilation result for all classes and packages   |  |  | | --- | --- | | Expected Result | Result | | All classes are successfully compiled. | Pass / Fail | |

## Exercise 2

1. Put DemoBlackJackImplements.java that you have created in Exercise 6 of Lab 6 in a package named apps.blackjack.
2. Amend the program. Add a line to display “This program is developed in Eclipse”.
3. Fix all errors and saved the class.
4. Run DemoBlackJackImplements.java.
5. Record the result in Table 2.

Table : Output from DemoBlackJackImplements.java

|  |  |
| --- | --- |
| Expected Result | Result |
| This is to demo Black Jack game that uses Interface and Abstract class  ----------------------------------------------------------------------  Player 1 : Emma McKay @ Melaka (Total point : 14)  Player 2 : Ahmad Ismail @ Denmark (Total point : 18)  Player 3 : Mr Zidane @ Algeria (Total point : 22)  Player 4 : Madhu Taneja @ Mumbai, India (Total point : 21)  -----> Player 1 Emma McKay is not eligible.  \*\*\* Winner is Player 2 : Ahmad Ismail  +----------+  |D |  | |  | |  | |  | 2|  +----------+  +----------+  |S |  | |  | |  | |  | K|  +----------+  +----------+  |S |  | |  | |  | |  | 6|  +----------+  -----> Player 4 Madhu Taneja is not eligible.  This program is developed in Eclipse | Pass / Fail |

## Exercise 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Open Player.java. 2. Add a static private attribute to keep track the number of players created. 3. Add codes to capture the number of players created in all constructors. 4. Add a getter method to retrieve the static attribute created in (2). 5. Add appropriate comments to the method. 6. Fix all errors and save Player.java 7. Record the result in Table 3. | Table 3 : Player.java compilation result   |  |  | | --- | --- | | Expected Result | Result | | Player.java compiles successfully | Pass / Fail | |

## Exercise 4

1. Open DemoBlackJackImplements.java.
2. Add additional codes to produce output as shown in Table 4.
3. Add appropriate comments to the method.
4. Fix all errors and save DemoBlackJackImplements.java
5. Record the result in Table 4.

Table : Output from DemoBlackJackImplements.java manipulating Player static variable and method

|  |  |
| --- | --- |
| Expected Result | Result |
| This is to demo Black Jack game that uses Interface and Abstract class  ----------------------------------------------------------------------  Player 1 : Emma McKay @ Melaka (Total point : 14)  Player 2 : Ahmad Ismail @ Denmark (Total point : 18)  Player 3 : Mr Zidane @ Algeria (Total point : 22)  Player 4 : Madhu Taneja @ Mumbai, India (Total point : 21)  -----> Player 1 Emma McKay is not eligible.  \*\*\* Winner is Player 2 : Ahmad Ismail  +----------+  |D |  | |  | |  | |  | 2|  +----------+  +----------+  |S |  | |  | |  | |  | K|  +----------+  +----------+  |S |  | |  | |  | |  | 6|  +----------+  -----> Player 4 Madhu Taneja is not eligible.  Number of Players : 4  This program is developed in Eclipse | Pass / Fail |

## Exercise 5

1. Open Card.java.
2. Add a static private attribute to keep track the cards of cards created.
3. Add codes to capture the number of cards created in all constructors.
4. Add a getter method to retrieve the static attribute created in (2).
5. Add appropriate comments to the method.
6. Fix all errors and save Card.java
7. Record the result in Table 5.

Table : Card.java compilation result

|  |  |
| --- | --- |
| Expected Result | Result |
| Card.java compiles successfully | Pass / Fail |

## Exercise 6

1. Open DemoBlackJackImplements.java.
2. Add additional codes to produce output as shown in Table 7.
3. Add appropriate comments to the method.
4. Fix all errors and save DemoBlackJackImplements.java
5. Record the result in Table 7.

## Exercise 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Create a new Java interface as shown in Figure 25. 2. Add appropriate comments to the interface and methods. 3. Fix all errors and save AlphabetCalculator.java. 4. Record the result in Table 6.   Table 6 : AlphabetCalculator.java compilation result   |  |  | | --- | --- | | Expected Result | Result | | AlphabetCalculator.java compiles successfully | Pass / Fail | | Figure 25: Interface AlphabetCalculator |

Table : Output from DemoBlackJackImplements.java manipulating Player and Card static variable and method

|  |  |
| --- | --- |
| Expected Result | Result |
| This is to demo Black Jack game that uses Interface and Abstract class  ----------------------------------------------------------------------  Player 1 : Emma McKay @ Melaka (Total point : 14)  Player 2 : Ahmad Ismail @ Denmark (Total point : 18)  Player 3 : Mr Zidane @ Algeria (Total point : 22)  Player 4 : Madhu Taneja @ Mumbai, India (Total point : 21)  -----> Player 1 Emma McKay is not eligible.  \*\*\* Winner is Player 2 : Ahmad Ismail  +----------+  |D |  | |  | |  | |  | 2|  +----------+  +----------+  |S |  | |  | |  | |  | K|  +----------+  +----------+  |S |  | |  | |  | |  | 6|  +----------+  -----> Player 4 Madhu Taneja is not eligible.  Number of Players : 4  Number of cards used in this game : 12  This program is developed in Eclipse | Pass / Fail |

## Exercise 8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Open Player.java. 2. Class Player implements interface AlphabetCalculator. 3. All methods in AlphabetCalculator need to be overridden. 4. Override method calculateAlphabet( ) to calculate the number of vowels, non-vowels and spaces in a Player’s name. You need to use some of the methods in class String. Refer to Java API to find the appropriate method. 5. Additional attributes are required for requirement in (4). 6. Add new getters that return all the attributes in (5). 7. Add appropriate comments to the method. 8. Fix all errors and save Player.java. 9. Record the result in Table 8. | Table 8 : Player.java compilation result   |  |  | | --- | --- | | Expected Result | Result | | Player.java compiles successfully after implementing interface AlphabetCalculator.java | Pass / Fail | |

## Exercise 9

1. Open DemoBlackJackImplements.java.
2. Add additional codes to produce output as shown in Table 9.
3. Add appropriate comments to the method.
4. Fix all errors and save DemoBlackJackImplements.java
5. Record the result in Table 9.

Table : Output from DemoBlackJackImplements.java

|  |  |
| --- | --- |
| Expected Result | Result |
| This is to demo Black Jack game that uses Interface and Abstract class  ----------------------------------------------------------------------  Player 1 : Emma McKay @ Melaka (Total point : 14)  Alphabets Count : # Vowels - 3, # Non-Vowels - 6, # Spaces - 1 and Length : 10  Player 2 : Ahmad Ismail @ Denmark (Total point : 18)  Alphabets Count : # Vowels - 5, # Non-Vowels - 6, # Spaces - 1 and Length : 12  Player 3 : Mr Zidane @ Algeria (Total point : 22)  Alphabets Count : # Vowels - 3, # Non-Vowels - 5, # Spaces - 1 and Length : 9  Player 4 : Madhu Taneja @ Mumbai, India (Total point : 21)  Alphabets Count : # Vowels - 5, # Non-Vowels - 6, # Spaces - 1 and Length : 12  -----> Player 1 Emma McKay is not eligible.  \*\*\* Winner is Player 2 : Ahmad Ismail  +----------+  |D |  | |  | |  | |  | 2|  +----------+  +----------+  |S |  | |  | |  | |  | K|  +----------+  +----------+  |S |  | |  | |  | |  | 6|  +----------+  -----> Player 4 Madhu Taneja is not eligible.  Number of Players : 4  Number of cards used in this game : 12  This program is developed in Eclipse | Pass / Fail |

## Exercise 10

1. Download blackjacksimplified.umlj from ulearn.
2. Update the UML diagrams to fulfill the requirement in Exercise 3, 5, 7 and 8.
3. Save the file.
4. Submit the new design to ulearn.

# How are you doing?

This section is to self- evaluate your skills to implements interface, organize project in Eclipse and update UML diagram.

|  |  |  |
| --- | --- | --- |
| Item | Number of PASS result | Number of FAIL result |
| Calculate the number of PASS and FAIL (including incomplete) from Exercise 1 until 9. |  |  |
| Are you able to update the class diagram according to the requirements? | □Yes | □No |
| Total |  |  |

**Score Guide**

If (Total Score for 3rd column == 0){

“You’re damn good!”

} Else If (Total Score for 2nd column > Total Score for 3rd column) {

“You’re good. You can do better. Finish all your exercises.”

} Else {

“Work harder, dude! You should finish all the exercises.”

}

Complete the table below and email it to your lab coordinator.

|  |  |  |
| --- | --- | --- |
| NAME | MATRIC NO | SECTION |
|  |  |  |