Lecture-2

Defining Classes and Objects

Introduction

- Classes are the most important language feature that make *object-oriented programming* (*OOP*) possible
- Programming in Java consists of defining a number of classes
  - Every program is a class
  - All helping software consists of classes
  - All programmer-defined types are classes
- Classes are central to Java

Class Definitions

- You already know how to use classes and the objects created from them, and how to invoke their methods
  - For example, you have already been using the predefined `String` and `Scanner` classes
- Now you will learn how to define your own classes and their methods, and how to create your own objects from them

A Class Is a Type

- A class is a special kind of programmer-defined type, and variables can be declared of a class type
- A value of a class type is called an object or *an instance of the class*
  - If A is a class, then the phrases "bla is of type A," "bla is an object of the class A," and "bla is an instance of the class A" mean the same thing
- A class determines the types of data that an object can contain, as well as the actions it can perform

Primitive Type Values vs. Class Type Values

- A primitive type value is a single piece of data
- A class type value or object can have multiple pieces of data, as well as actions called *methods*
  - All objects of a class have the same methods
  - All objects of a class have the same pieces of data (i.e., name, type, and number)
  - For a given object, each piece of data can hold a different value

The Contents of a Class Definition

- A class definition specifies the data items and methods that all of its objects will have
- These data items and methods are sometimes called *members* of the object
- Data items are called *fields* or *instance variables*
- Instance variable declarations and method definitions can be placed in any order within the class definition

Class Definition

- Syntax

```
public class Class_Name {
        Instance_Variable_Declaration_1
        Instance_Variable_Declaration_2
                …
        Instance_Variable_Declaration_Last

        Method_Definition_1 Method_Definition_2
                ...
        Method_Definition_Last
}
```

The **`new`** Operator

- An object of a class is named or declared by a variable of the class type:

```
ClassName  classVar;
```

- The **`new`** operator must then be used to create the object and associate it with its variable name:

```
classVar = new ClassName();
```

- These can be combined as follows:

```
ClassName classVar = new ClassName();
```

Instance Variables and Methods

An object's data and methods can be invoked using dot operator(.) known as ***object member access operator***

- Instance variables can be defined as in the following two examples
  - Note the `public` modifier (for now):

```
public String  instanceVar1;
public int  instanceVar2;
```

- In order to refer to a particular instance variable, preface it with its object name as follows:

```
objectName.instanceVar1
objectName.instanceVar2
```

Instance Variables and Methods

- An invocation of a method that returns a value can be used as an expression anyplace that a value of the **`typeReturned`** can be used:

```
typeReturned tRVariable;
tRVariable = objectName.methodName();
```

- An invocation of a **`void`** method is simply a statement:

```
objectName.methodName();
```

State of an Object

- The state of an object (also known as its properties or attributes) is represented by data fields with their current values.
- Default state of object after its creation is default values in its attributes

```
class Student {
         String name;
         int age;
         boolean
         char gender;
         double marks;
}
```

```
public class Main{
         public static void main(String args[]){

                  Student std1 = new Student();
                  std1.name = "Riz";
                  System.out.println(std1.age);
                  Student std2 = new Student();
                  std2.name = "Ali";
                  System.out.println(std1.name);
                  System.out.println(std2.name);
         }
}
```

Reference Data Fields and the null Value

- Default state of an object when it is created means, default values in data fields.
  - **Reference Type** has default value ***null*** ; when reference type variable is not referencing any object
  - **Numeric Type** has default value 0
  - **Boolean Type** has default value true
  - **Char Type** has \u0000
- Java assigns **no default** value to a local variable inside a method

Example

```
public class Main{
       public static void main(String args[]){

              Student std1 = new Student();
              System.out.println(std1.name);
              System.out.println(std1.age);
              System.out.println(std1.isScienceMajor);
              System.out.println(std1.gender);
              System.out.println(std1.marks);

       }
}
class Student {
       String name; // name has the default value null
       int age; // age has the default value 0
       boolean isScienceMajor; //isScienceMajor default value false
       char gender; // gender has default value '\u0000'
       double marks; // marks has default value 0.0
}
```

File Names and Locations

- Reminder: a Java file must be given the same name as the class it contains with an added **`.java`** at the end
  - For example, a class named **`MyClass`** must be in a file named **`MyClass.java`**
- For now, your program and all the classes it uses should be in the same directory or folder