Lecture 6: Class vs Object, Constructor , Getter and Setter

Object Oriented Concepts and Programming (CSC244)

By

Dr. Tabbasum Naz

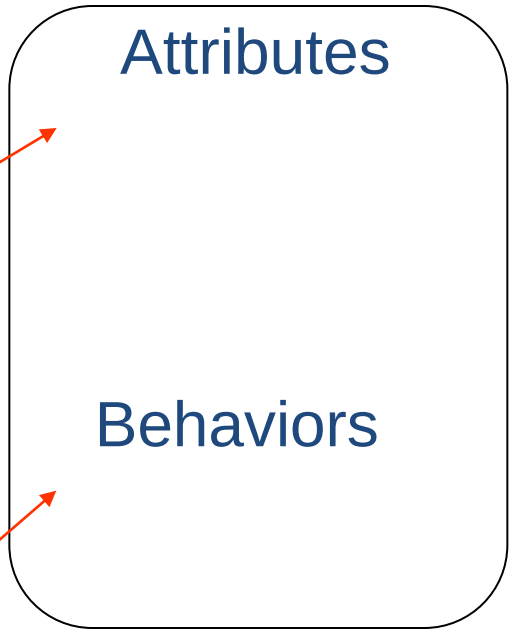
tabbasum.naz@ciitlahore.edu.pk

Class vs Object

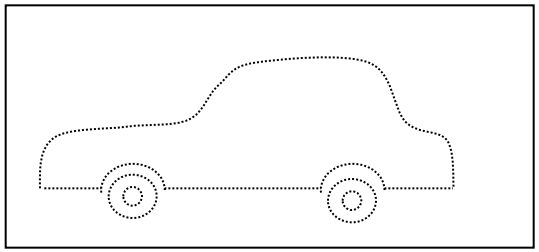
Class

- A blueprint for objects of a particular type
- Defines the structure (number, types) of the attributes
- Defines available behaviors of its objects

Object



Class: Car



Attributes:

- String model
- Color color
- int numPassengers
- double amountOfGas

Behaviors:

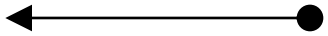
- Add/remove a passenger
- Get the tank filled
- Report when out of gas

Object: a car

Attributes:

- model = "Mustang"
- color = Color.YELLOW
- numPassengers = 0
- amountOfGas = 16.5

Behaviors:



Class vs. Object

Class	Object
• A piece of the program's source code	• An entity in a running program
• Written by a programmer	• Created when the program is running (by the main method or a constructor or another method)

Class vs. Object

- Specifies the structure (the number and types) of its objects' attributes — the same for all of its objects
- Specifies the possible behaviors of its objects

- Holds specific values of attributes; these values can change while the program is running
- Behaves appropriately when called upon

What is a Constructor?

- Constructor is a special method that gets invoked "automatically" at the time of object creation.
- Constructor is normally used for initializing objects with default values unless different values are supplied.
- Constructor has the same name as the class name.
- Constructors do not have a return type (not even void) and they do not return a value.
- A class can have more than one constructor as long as they have different signature (i.e., different input arguments syntax).
- Constructors have special syntax:
  - must always have the same name as their class
  - by convention, always start with capital letter (unlike other methods)

Default Constructor

- There is always at least one constructor in every class
- If you do not define any constructors for a class, Java writes one for you
  - called *default* constructor
  - default constructor will initialize each instance variable to its default value
- A constructor is called whenever an object is created.
- If the programmer does not supply any constructors, the default constructor will be present automatically
  - The default constructor takes no arguments
  - The default constructor takes no body
- Enable you to create object instances with new XX()
- Notes: If you add a constructor declaration with arguments to a class that previously had no explicit constructors, you lose the default constructor.

Defining a Constructor

- Like any other method

```
public class ClassName {

    // Data Fields…

    // Constructor
    public ClassName()
    {
        // Method Body Statements initialising Data Fields
    }

    //Methods to manipulate data fields
}
```

- Invoking:
  - There is NO explicit invocation statement needed: When the object creation statement is executed, the constructor method will be executed automatically.

Adding a Multiple-Parameters Constructor to our Circle Class

```
public class Circle {
      public double x,y,r;
    // Constructor
    public Circle(double centreX, double centreY, double radius)
    {
            x = centreX;
            y = centreY;
            r = radius;
      }
    //Methods to return circumference and area
      public double circumference()
{
                  return 2*3.14*r;
}
      public double area()
{
return 3.14 * r * r; }
}
}
```

Defining a Constructor: Example

```
public class Counter {
    int CounterIndex;

    // Constructor
    public Counter()
    {
        CounterIndex = 0;
    }
    //Methods to update or access counter
    public void increase()
    {
        CounterIndex = CounterIndex + 1;
    }
    public void decrease()
    {
        CounterIndex = CounterIndex - 1;
    }
    int getCounterIndex()
    {
      return CounterIndex;
    }
}
```

Trace counter value at each statement and What is the output ?

```
public class Counter {
    int CounterIndex;

    // Constructor
    public Counter()
    {
        CounterIndex = 0;
    }
    //Methods
    public void increase()
    {
        CounterIndex = CounterIndex + 1;
    }
    public void decrease()
    {
        CounterIndex = CounterIndex - 1;
    }
    int getCounterIndex()
    {
      return CounterIndex;
    }
}
```

```
class MyClass {
  public static void main(String args[])
  {
Counter counter1 = new Counter();
counter1.increase();
int a = counter1.getCounterIndex();
counter1.increase();
int b = counter1.getCounterIndex();
if ( a > b )
      counter1.increase();
else
      counter1.decrease();

System.out.println(counter1.getCounterI
  }
}
```

Multiple Constructors

- Sometimes want to initialize in a number of different ways, depending on circumstance.

- This can be supported by having multiple constructors having different input arguments. This is called **constructor overloading.**

More on Constructor

- Constructors are used to assign initial values to instance variables of the class. Constructors have the following characteristics:
- They are called only once when the class is being instantiated.
- They must have the same name as the class itself.
- They do not return a value and you do not have to specify the keyword void.
- Constructor are declared just like as we declare methods, except that the constructor don't have any return type. Constructor can be overloaded provided they should have different arguments because JVM(Java Virtual Machine) differentiates constructors on the basis of arguments passed in the constructor.
- Whenever we assign the name of the method same as class name. Remember this method should not have any return type.
- In the example below we have made three overloaded constructors each having different arguments types so that the JVM can differentiates between the various constructors.

Constructor Overloading Example

```
class Rectangle{
  int l, b;
  float p, q;
  public Rectangle(int x, int y)//First constructor
  {
  l = x;
  b = y;
  }
  public int first()//Method 1
  {
  return(l * b);
  }
  public Rectangle(int x)//Constructor 2
  {
  l = x;
  b = x;
  }
  public int second() //Method 2
  {
  return(l * b);
  }
  public Rectangle(float x)//Constructor 3
  {
  p = x;
  q = x;
  }
  public float third()//Method 3
 {
  return(p * q);
  }
  public Rectangle(float x, float y)//Constructor 4
 {
  p = x;
  q = y;
  }
  public float fourth()//Method 4
 {
  return(p * q);
  }
}
```

Constructor Overloading Example

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

  Rectangle rectangle1=new Rectangle(2,4);
  int areaInFirstConstructor=rectangle1.first();
  System.out.println(" The area of a rectangle in
  first constructor is :  " + areaInFirstConstructor);

  Rectangle rectangle2=new Rectangle(5);
  int areaInSecondConstructor=rectangle2.second();
  System.out.println(" The area of a rectangle in
   first constructor is :  " + areaInSecondConstructor);

  Rectangle rectangle3=new Rectangle(2.0f);
  float areaInThirdConstructor=rectangle3.third();
  System.out.println(" The area of a rectangle in first
  constructor is :  " + areaInThirdConstructor);

  Rectangle rectangle4=new Rectangle(3.0f,2.0f);
  float areaInFourthConstructor=rectangle4.fourth();
  System.out.println(" The area of a rectangle in first
  constructor is :  " + areaInFourthConstructor);
  }
}
```

```
C:\java>java
ConstructorOverloading
The area of a rectangle in first
constructor is : 8
The area of a rectangle in first
constructor is : 25
The area of a rectangle in first
constructor is : 4.0
The area of a rectangle in first
constructor is : 6.0
```

Overloading Constructors

```
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }

  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

```
class OverloadCons {
  public static void main(String args[]) {
    // create boxes using the various constructors
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box();
    Box mycube = new Box(7);

    double vol;

    // get volume of first box
    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);

    // get volume of second box
    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);

    // get volume of cube
    vol = mycube.volume();
    System.out.println("Volume of mycube is " + vol);
  }
}
```

```
Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0
Volume of mycube is 343.0
```

Public, private, protected

- We use these keywords to specify access levels for member variables, or for member functions (methods).
- **Public** variables, are variables that are visible to all classes.
- **Private** variables, are variables that are visible only to the class to which they belong.
- **Protected** variables, are variables that are visible only to the class to which they belong, and any subclasses.
- Deciding when to use private, protected, or public variables is sometimes tricky. You need to think whether or not an external object (or program), actually needs direct access to the information. If you do want other objects to access internal data, but wish to control it, you would make it either private or protected, but provide functions which can manipulate the data in a controlled way.

```
public class bank_balance {
public String owner;
public int balance;
 public bank_balance( String name, int dollars )
 {
owner = name;
if (dollars >= 0)
balance = dollars;
else
dollars =0;
 }
}
```

- Any object in the system can change the balance (setting it to zero, or even giving us a negative balance). This could cause the program to fall over, even though we wrote code in our constructor to prevent negative balances.

```
public class bank_balance {
public String owner;
private int balance;
public bank_balance( String name, int dollars )
 {
   owner = name;
   if (dollars >= 0)
   balance = dollars;
   else
   dollars =0;
}
 public int getBalance()
{
 return balance;
}
 public void setBalance(int dollars)
 {
 if (dollars >= 0)
balance = dollars;
else dollars = 0;
}
 }
```

Accessors (getter) and Mutators (setter)

```
1  // Fig. 3.10: GradeBook.java
2  // GradeBook class with a constructor to initialize the course name.
3
4  public class GradeBook
5  {
6     private String courseName; // course name for this GradeBook
7
8     // constructor initializes courseName with String supplied as argument
9     public GradeBook( String name )
10    {
11       courseName = name; // initializes courseName
12    } // end constructor
13
14    // method to set the course name
15    public void setCourseName( String name )
16    {
17       courseName = name; // store the course name
18    } // end method setCourseName
19
20    // method to retrieve the course name
21    public String getCourseName()
22    {
23       return courseName;
24    } // end method getCourseName
25
26    // display a welcome message to the GradeBook user
27    public void displayMessage()
28    {
29       // this statement calls getCourseName to get the
30       // name of the course this GradeBook represents
31       System.out.printf( "Welcome to the grade book for\n%s!\n",
32          getCourseName() );
33    } // end method displayMessage
34
35 } // end class GradeBook
```

Fig. 3.10 | GradeBook class with a constructor to initialize the course name.

Accessors (getter) and Mutators (setter)

```
1  // Fig. 3.8: GradeBookTest.java
2  // Create and manipulate a GradeBook object.
3  import java.util.Scanner; // program uses Scanner
4
5  public class GradeBookTest
6  {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       // create a GradeBook object and assign it to myGradeBook
14       GradeBook myGradeBook = new GradeBook();
15
16       // display initial value of courseName
17       System.out.printf( "Initial course name is: %s\n\n",
18          myGradeBook.getCourseName() );
19
20       // prompt for and read course name
21       System.out.println( "Please enter the course name:" );
22       String theName = input.nextLine(); // read a line of text
23       myGradeBook.setCourseName( theName ); // set the course name
24       System.out.println(); // outputs a blank line
25
26       // display welcome message after specifying course name
27       myGradeBook.displayMessage();
28    } // end main
29
30 } // end class GradeBookTest
```

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

Fig. 3.8 | Creating and manipulating a GradeBook object.