Object Oriented Concepts and Constructors

Reference: Liang, Y., 2015. Intro To Java Programming. 10th ed. Pearson Education Limited.

Unit Topics

1. Simple Program to Add Two Numbers
2. Objects
3. Input Value from Student
4. Relation Between Class and Object
5. UML for Circle
6. Constructor

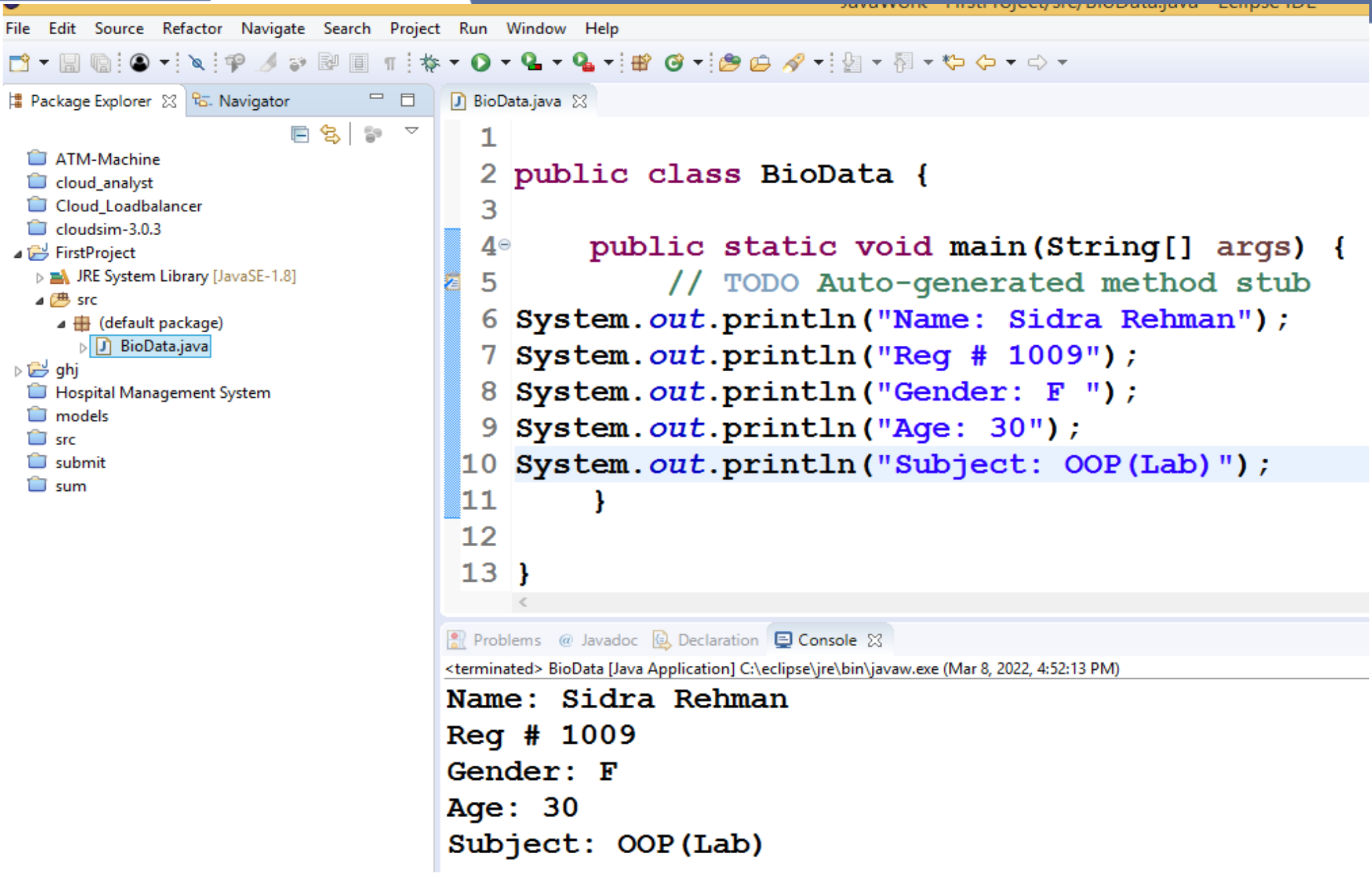
Creating UMLs

Online Tool:

https://online.visual-paradigm.com/diagrams/features/erd-tool/

Offline Softwares:

- Adobe Spark.
- Edraw Max.
- Moqups.
- Microsoft Visio.
- Lucidchart.
- ConceptDraw.
- StarUML.
- Umbrello.


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Navigator
ATM-Machine
cloud_analyst
Cloud_Loadbalancer
cloudsim-3.0.3
FirstProject
JRE System Library [JavaSE-1.8]
src
(default package)
BioData.java
ghj
Hospital Management System
models
src
submit
sum
BioData.java
1
2 public class BioData {
3
4 public static void main(String[] args) {
5 // TODO Auto-generated method stub
6 System.out.println("Name: Sidra Rehman");
7 System.out.println("Reg # 1009");
8 System.out.println("Gender: F ");
9 System.out.println("Age: 30");
10 System.out.println("Subject: OOP(Lab)");
11 }
12
13 }
Problems Javadoc Declaration Console
<terminated> BioData [Java Application] C:\eclipse\jre\bin\javaw.exe (Mar 8, 2022, 4:52:13 PM)
Name: Sidra Rehman
Reg # 1009
Gender: F
Age: 30
Subject: OOP(Lab)

Simple Program to Add two numbers:

```
public class AddTwoNumbers
{
public static void main(String[] args) {
int num1 = 5, num2 = 15, sum;
sum = num1 + num2;
System.out.println("Sum of these numbers: "+ sum);
}
 }
```

Output:

Sum of these numbers: 20

Input number from User

```
import java.util.Scanner;
public class AddTwoNumbers2 {
public static void main(String[] args) {
int num1, num2, sum;
Scanner sc = new Scanner(System.in);
System.out.println("Enter First Number: ");
num1 = sc.nextInt();
System.out.println("Enter Second Number: ");
num2 = sc.nextInt();
sc.close();
sum = num1 + num2;
System.out.println("Sum of these numbers: "+sum);
 }
}
Output:
Enter First Number: 121 Enter Second Number: 19
Sum of these numbers: 140
```

Print the average of three numbers entered by user by creating a class named avg to calculate and print the average.

```
import java.util.Scanner;
public class avg {
  public static void main(String[] args)
    {
double num1,num2,num3, avg;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the first number: ");
        num1 = sc.nextDouble();
        System.out.print("Enter the second number: ");
        num2 = sc.nextDouble();
        System.out.print("Enter the third number: ");
        num3 = sc.nextDouble();
        sc.close();
        avg =(num1+num2+num3)/3;
        System.out.print("The average of entered numbers is:" +avg );
    }
}
```

Objects

- An *object* represents an entity in the real world that can be distinctly identified. An object has a unique identity, state, and behavior.
- Examples:
  a student, a desk, a circle, a button, and a loan.
- The **state** of an object (also known as its properties or attributes) is represented by data fields with their current values.
- The **behavior** of an object (also known as its actions) is defined by methods.
- An object is an **instance** of a class. Many instances can be created from a single class.
- The terms *object and instance are often interchangeable.*

object Syntax

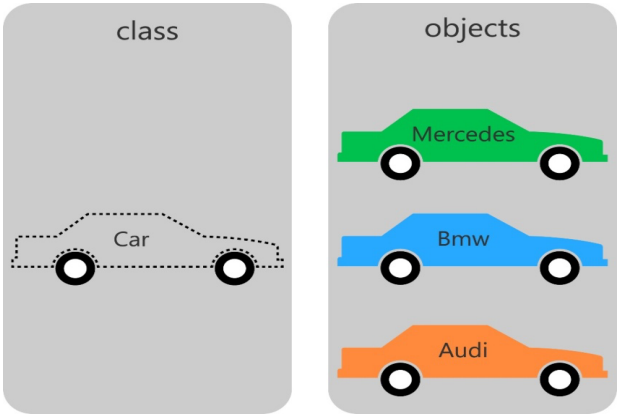

```
ClassName object_name;
object_name = new ClassName();

or simply
ClassName object_name = new ClassName();

or with parameters in constructor
ClassName object_name = new ClassName(type parameter);
```

Relation between Class & Object

- *Classes are definitions for objects and objects are created from classes.*
- A class creates a new *data type* that can be used to create objects.
- The relationship between classes and objects is analogous to that between a recipe and dish prepared by it.
- Many objects can be created from the same class.



new operator

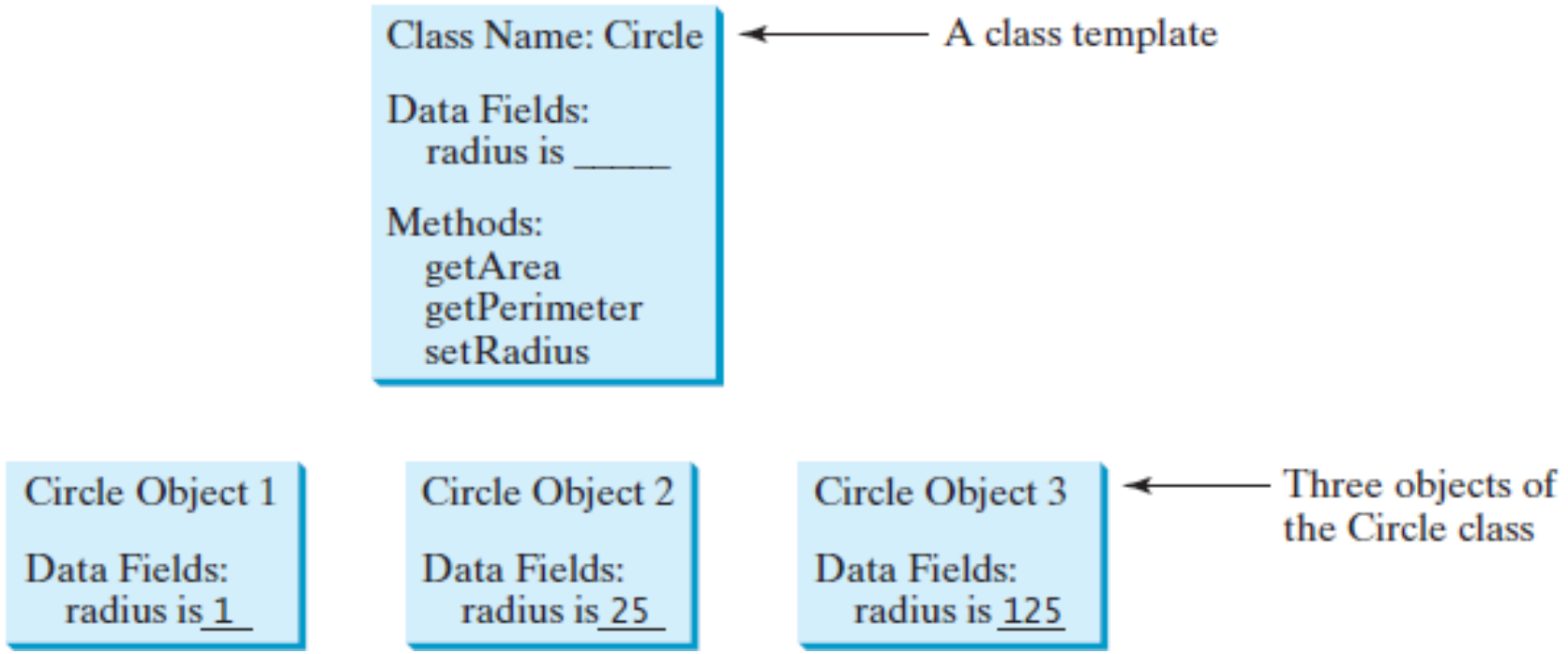
- The *new* operator dynamically allocates memory for an object.
- It has this general form:

  class-var= new classname();

- *new* allocates memory for an object during run time.
- The advantage of this approach is that a program can create as many or as few objects as it needs during the execution.

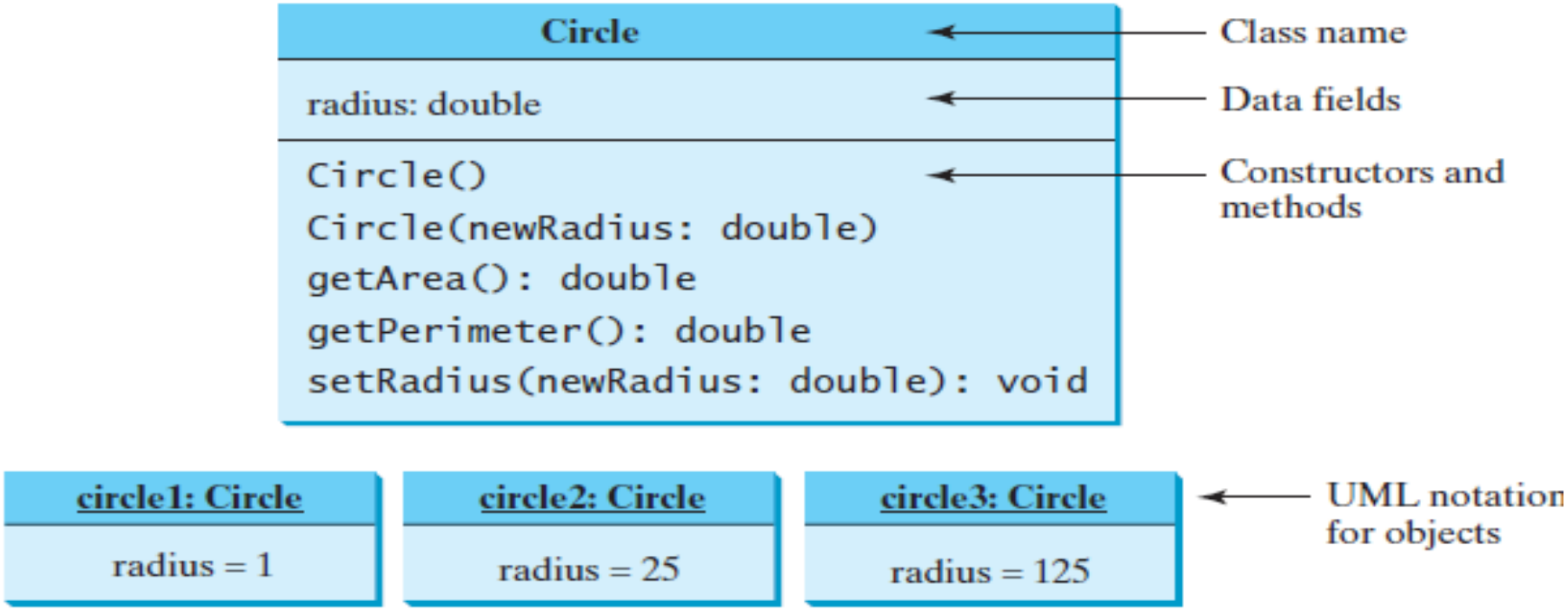
circle – an example

- A **circle** object has a data field **radius**, which is the property that characterizes a circle. Similarly methods can be defined like **getArea()** and **getPerimeter()** for circle objects.



UML Class Diagram

. Classes and objects can be represented using UML notation.



Constructor

- A *constructor* is invoked to create an object using the *new* operator.
- Constructors are a special kind of method.
- They have three **properties**:
  1. A constructor must have the same name as the class itself.
  2. Constructors do not have a return type—not even void.
  3. Constructors are invoked when an object is created. Constructors play the role of initializing objects.
- A constructor initializes an object immediately upon creation.
- The implicit return type of a constructor is the class type itself.
- To construct an object from a class, invoke a constructor of the class using the **new** operator, as follows:

new ClassName(arguments);

```
class ClassName {
    ClassName() {
    }
}
```

Constructor

Is it a valid constructor for class **Circle**?

```
void Circle() {
}
```

- It is a method, not a constructor.
- Constructor does not have a return type.
- It is a common mistake to put the **void** keyword in front of a constructor.

```
void Circle() {        ⟶        Circle() {
}                                }
```

Java allows two types of constructors namely :

- No argument Constructors
- Parameterized Constructors

No argument Constructors

- As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.
- Example

```
Public class MyClass {
    Int num;
    MyClass() {
        num = 100;
    }
}
```

You would call constructor to initialize objects as follows

```
public class ConsDemo {
   public static void main(String args[]) {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      System.out.println(t1.num + " " + t2.num);
   }
}
```

This would produce the following result

```
100 100
```

Non-Parameterized Constructor

Defining a Non-Parameterized Constructor

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

  // This is the constructor for Box.
  Box() {
    System.out.println("Constructing Box");
    width = 10;
    height = 10;
    depth = 10;
  }
}
```

Calling a Non-Parameterized Constructor

```
class BoxDemo {
  public static void main(String args[]) {
    // declare, and initialize Box objects
    Box mybox1 = new Box();
    }
  }
```

Parameterized Constructors

- Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.
- Example

Here is a simple example that uses a constructor –

```
// A simple constructor.
class MyClass {
   int x;

   // Following is the constructor
   MyClass(int i ) {
      x = i;
   }
}
```

You would call constructor to initialize objects as follows –

```
public class ConsDemo {
   public static void main(String args[]) {
      MyClass t1 = new MyClass( 10 );
      MyClass t2 = new MyClass( 20 );
      System.out.println(t1.x + " " + t2.x);
   }
}
```

This would produce the following result –

```
10 20
```

Parameterized Constructor

Defining a Parameterized Constructor

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

  // This is the constructor for Box.
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
}
```

Calling a Parameterized Constructor

```
class BoxDemo  {
  public static void main(String args[]) {
    // declare, initialize Box objects
    Box mybox1 = new Box(10, 20, 15);
    }
  }
```


THANK YOU