Object Oriented Programming
OOP 4 : Objects and Classes

Hirra Anwar

Contents by Dr. Abdul Ghafoor

National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science Islamabad

Agenda

- Objects
- Classes
- Method Overloading
- Constructor
- this()

Agenda

- Objects
- Classes
- Method Overloading
- Constructor
- this()

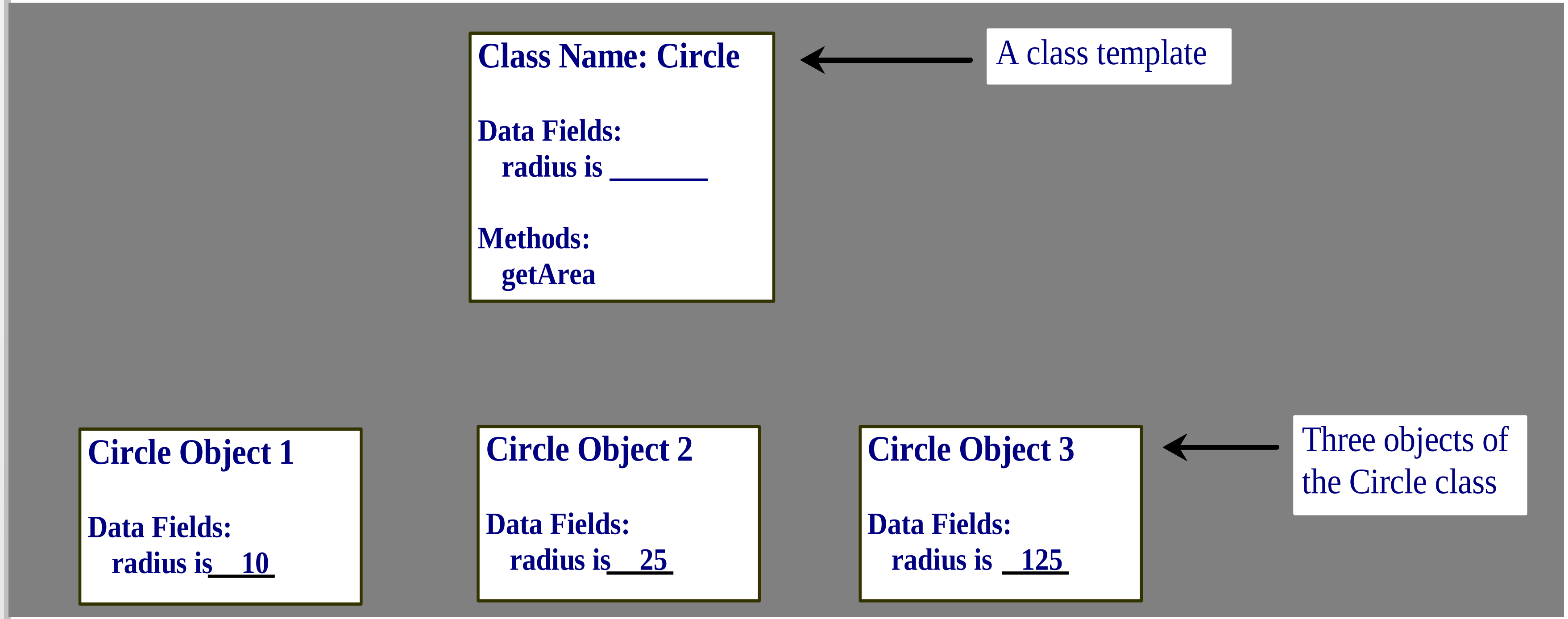
Objects

- An object has a unique identity, state, and behaviors.
- The state of an object consists of a set of data fields (also known as properties) with their current values.
- The behavior of an object is defined by a set of methods.

Classes

- Classes are constructs that define objects of the same type.
- A Java class uses variables to define data fields and methods to define behaviors.
- Additionally, a class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.

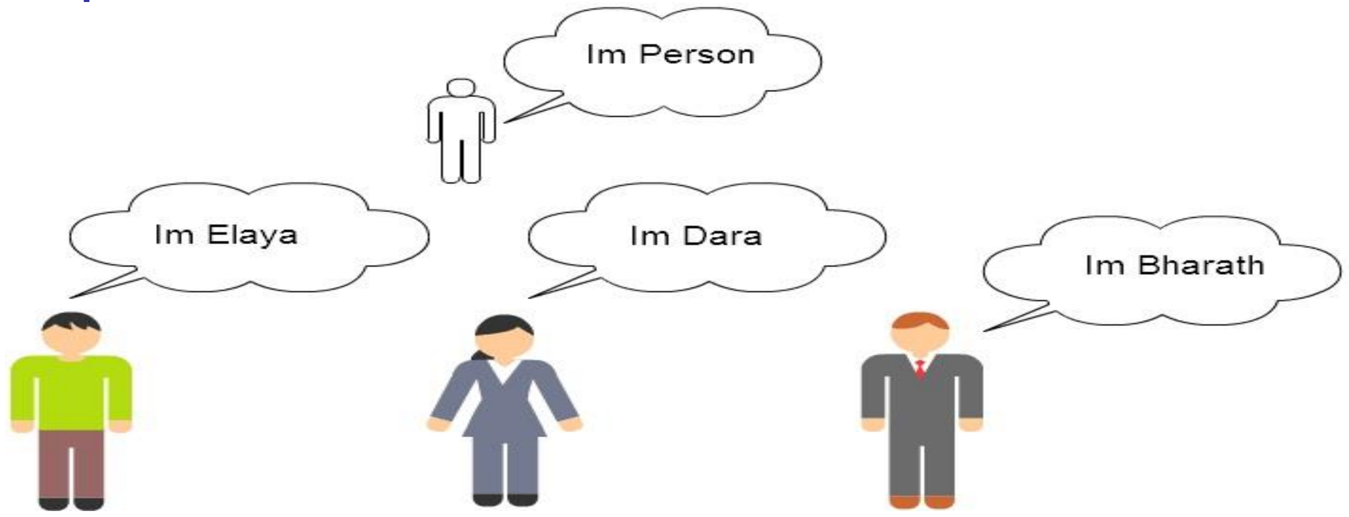
Objects



- An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

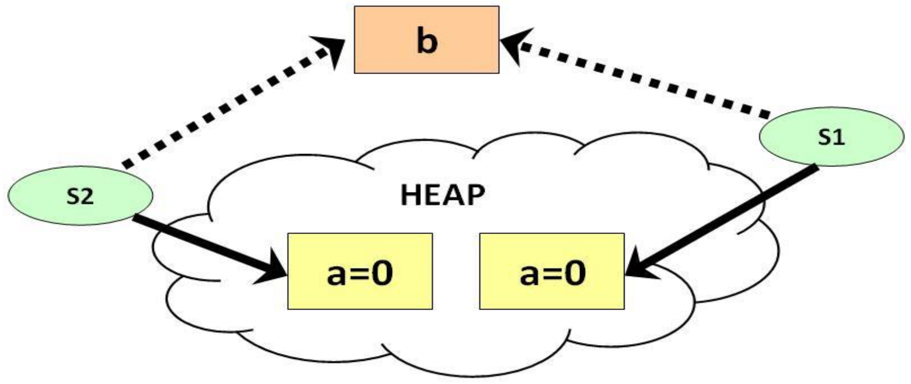
Instance variables

- Instance variable are those variables that are associated with each object uniquely.
  - Each instance of the class will have its own copy of each of these variables.
  - Each object will have its own values for each instance variables that differentiate one object from the other of the same class type.
  - Declared in the usual way and can have an initial value specified.



Class variables

- Class variables are associated with the class and is shared by all objects of the class.
  - There is only one copy of each of these variables no matter how many class objects are created.
  - They exist even if no objects of class have been created.
  - These variables are also called static fields because we use the keyword static when we declare them.



Instance and Class Methods

- **Instance Methods**
  - These methods can only be executed in relation to a particular object
  - If no object exists, no instance method can be executed
  - Note: Although instance methods are specific to objects of a class, there is only one copy of an instance method in memory that is shared by all the objects of the class.
- **Class Methods**
  - You can execute class methods even when no objects of a class exist.
  - Like class variables, these are declared using the keyword static, so also called static methods
  - Static methods cannot refer to instance variables or call instance methods. Why?
  - Why main is always declared static?

Instance Variables

```
public class StudentRecord
{
    // Instance variables
    private String name;
    private int age;
    private double mathGrade;
    private double englishGrade;
    private double average;
    //we'll add more code here later
}
```

- Declare instance variables as private so that only class methods can access them directly.

Class (static) variables

```
public class StudentRecord
{
    //static variables
    private static int studentCount;
    //we'll add more code here later
}
```

– we use the keyword static to indicate that a variable is a static variable.

Accessor (Getter) Method

```
public class StudentRecord
{
    private String name;
     // some code
     // An example in which the business logic is
     // used to return a value on an accessor method
    public double getAverage()
    {
            double result = 0;
            result=(mathGrade+englishGrade+scienceGrade)/3;
            return result;
    }
}
```

Mutator (Setter) Method

```
public class StudentRecord
{
   private String name;
   public void setName( String temp )
   {
         name = temp;
   }
}
```

Static Methods

```
public class StudentRecord
{
    private static int studentCount;
    public static int getStudentCount()
    {
            return studentCount;
    }
}
```

- static-means that the method is static and should be called by typing,*[ClassName].[methodName]*.

- For example, in this case, we call the method StudentRecord.getStudentCount()

When to Define Static Method?

- When the logic and state does not involve specific object instance
  - Computation method
  - add(int x, int y) method
- When the logic is a convenience without creating an object instance
  - Integer.parseInt(…);

Example

```
public class StudentRecord
 {
    // Instance variables
    private String name;
    private String address;
    private int age;
    private double mathGrade;
    private double englishGrade;
    private double scienceGrade;
    private double average;
    private static int studentCount;
    StudentRecord()
    {
              studentCount++;
    }
   public String getName()
   {
        return name;
   }
```

```
    public void setName( String temp )
    {
         name = temp;
    }

    public double getAverage()
    {
       double result = 0;
       result =(mathGrade+englishGrade+scienceGrade )/3;
       return result;
    }

    public static int getStudentCount()
    {
        return studentCount;
    }
    // write main mehod here
}
```

Sample Source Code that uses StudentRecord Class

```
public static void main( String[] args )
{
   //create three objects for Student record
   StudentRecord aRecord = new StudentRecord();
   StudentRecord bRecord = new StudentRecord();
   StudentRecord cRecord = new StudentRecord();
   //set the name of the students
   aRecord.setName(“Junaid");
   bRecord.setName(“Imran");
   cRecord.setName(“Waseen Akram");
   //print name
   System.out.println( aRecord.getName() );
   //print number of students
   System.out.println("Count="+StudentRecord.getStudentCount());
}
```

Output

- Junaid
- Count=3

An example class-2

```
public class BankAccount
 {
   //--instance variables
  private String ID;
  private double balance;
   // Constructor to initialize the state
   public BankAccount(String initID, double initBalance)
   {
       ID = initID;
       balance = initBalance;
   }
   // Credit this account by depositAmount
    public void deposit(double depositAmount)
   {
       balance = balance + depositAmount;
    }
     // Debit this account by withdrawalAmount
     public void withdraw(double withdrawalAmount)
    {
       balance = balance - withdrawalAmount;
    }
```

class BankAccount

```
  public String getID()
  {
    return ID;
  }

  public double getBalance()
  {
    return balance;
  }
  public String toString( )
  {
    return ID + " $" + balance;
  }
} // End class BankAccount
```

Sample Messages

```
BankAccount anAcct = new BankAccount("Moss", 500.00);

anAcct.withdraw(60);
anAcct.deposit(147.35);
```

Agenda

- Objects
- Classes
- Method Overloading
- Constructor
- this()

Method Overloading

- Method overloading
  - allows a method with the same name but different parameters, to have different implementations and return values of different types
  - can be used when the same operation has different implementations.
- Always remember that overloaded methods have the following properties:
  - the same method name
  - different parameters or different number of parameters
  - return types can be different or the same

Example

```
public void print( String temp )
{
    System.out.println("Name:" + name);
    System.out.println("Address:" + address);
    System.out.println("Age:" + age);
}
public void print(double eGrade, double mGrade, double sGrade)
{
    System.out.println("Name:" + name);
    System.out.println("Math Grade:" + mGrade);
    System.out.println("English Grade:" + eGrade);
    System.out.println("Science Grade:" + sGrade);
}
```

Example

```
public static void main( String[] args )
{
    StudentRecord aRecord = new StudentRecord();
    aRecord.setName(“Ahmed");
    aRecord.setAddress(“Pakistan");
    aRecord.setAge(15);
    aRecord.setMathGrade(80);
    aRecord.setEnglishGrade(95.5);
    aRecord.setScienceGrade(100);

    //overloaded methods
    aRecord.print( aRecord.getName() );
    aRecord.print( aRecord.getEnglishGrade(),aRecord.getMathGrade(), aRecord.getScienceGrade());
}
```

Output

we will have the output for the first call to print,

```
Name:Ahmed
Address:Pakistan
Age:15
```

- we will have the output for the second call to print,

```
Name:Ahmed
Math Grade:80.0
English Grade:95.5
Science Grade:100.0
```

Agenda

- Objects
- Classes
- Method Overloading
- Constructor
- this()

Default constructor (no-arg constructor)

- is the constructor without any parameters.
- If the class does not specify any constructors, then an implicit default constructor is created.

Constructor Overloading

- Classes can have more than one constructor
- All constructors have the same name (the class name)
- Each constructor differs from the others in either the number or types of its arguments
- `new` is used when using a constructor to create a new object

Constructor Overloading

```
public StudentRecord()
{
    //some initialization code here
}

public StudentRecord(String temp)
{
    name = temp;
}
public StudentRecord(String name, String address_)
{
    name = name;
    address = address_;
}
public StudentRecord(double mGrade, double eGrade,double sGrade)
{
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
}
```

Using Constructors

To use these constructors, we have the following code,

```
public static void main( String[] args )
{
    //create three objects for Student record
    StudentRecord aRecord=new StudentRecord("Ahmed");
    StudentRecord bRecord=new StudentRecord("BIT", "Pakistan");
    StudentRecord cRecord=new StudentRecord(80,90,100);
    //some code here
}
```

Example-3

```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Data field

Constructors

Method

Agenda

- Objects
- Classes
- Method Overloading
- Constructor
- this()

“this()” constructor call

Constructor calls can be chained, meaning, you can call another constructor from inside another constructor.

- We use the this() call for this
- There are a few things to remember when using the this() constructor call:

  – When using the this constructor call, *IT MUST OCCUR AS THE FIRST STATEMENT* in a constructor

  – It can ONLY BE USED IN A CONSTRUCTOR DEFINITION. The this call can then be followed by any other relevant statements

Example

```
public StudentRecord()
{
    this("some string");
}

 public StudentRecord(String temp)
{
    this.name = temp;
}

public static void main( String[] args )
{
     StudentRecord aRecord = new StudentRecord();
}
```

“this” reference

The *this* reference

– refers to current object instance itself

– used to access the instance variables shadowed by the parameters.

- To use the this reference, we type, **this.<nameOfTheInstanceVariable>**
- You can only use the this reference for instance variables and NOT static or class variables

"this" reference

The this reference is assumed when you call a method from the same object

```
public class MyClass
{
    void aMethod()
    {
            // same thing as this.anotherMethod()
            anotherMethod();
    }
    void anotherMethod()
    {
            // method definition here...
    }
}
```

Example

```
public void setAge( int age )
{
    this.age = age;
}
```

Reading Material

- Reading Material:
  - Chapter 3 : 72-97