

Practical Computer Science 2

02-Review

Sayed Ahmad Sahim

Kandahar University
Computer Science Faculty

csadjava@gmail.com

February 17, 2018

Conttents

- 1 Object Oriented Programing OOP
- 2 Java Naming conventions
- 3 Object
- 4 Class
- 5 Class Constructors
- 6 Access Modifiers
- 7 Question

Object Oriented Programing OOP

- Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.
- Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.
- In structural programing, the programming challenge was seen as how to write the logic, not how to define the data.

Object Oriented Programing OOP

- The first step in OOP is to identify all the objects the programmer wants to manipulate and how they relate to each other, an exercise often known as data modeling.
- Once an object has been identified, it is generalized as a class of objects (the idea of chairs).

Object Oriented Programing OOP

- It simplifies the software development and maintenance by providing some concepts:
 - Object
 - Class
 - Inheritance
 - Polymorphism
 - Abstraction
 - Encapsulation

Java Naming conventions

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.
- But, it is not forced to follow. So, it is known as convention not rule.
- All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

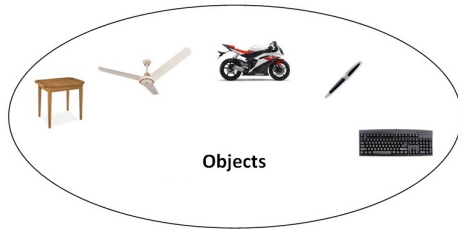
Advantage of naming conventions in java

- By using standard Java naming conventions:
 - you make your code easier to read for yourself and for other programmers.
 - Readability of Java program is very important.
 - It indicates that less time is spent to figure out what the code does.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Object

- An entity that has state and behavior is known as an object.
- e.g. chair, bike, marker, pen, table, car etc.
- It can be physical or logical (tangible and intangible).
- The example of intangible object is banking system.

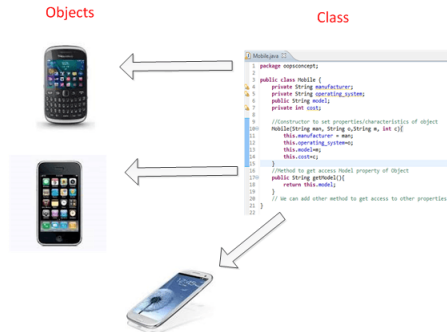


Characterstics of an object

- An object has three characteristics:
 - ❶ **state:** represents data (value) of an object.
 - ❷ **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
 - ❸ **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.
- **Object is an instance of a class:** Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

Class

- A class is a group of objects which have common properties.
- It is a template or blueprint from which objects are created.
- Class is a logical entity, It can't be physical.
- An object is an instance of a class.



Information Hiding

- In a well designed OO application, a class publicizes what it can do i.e. its method signatures
- but hides the internal details both of how it performs these services (method bodies) and
- the data (attributes) that it maintains in order to support these services

Class characteristics

- A class in Java can contain:
 - fields
 - methods
 - constructors
 - blocks
 - nested class and interface
- Syntax to declare a class:

```
class <class_name>{  
    field;  
    method;  
}
```

Class Members

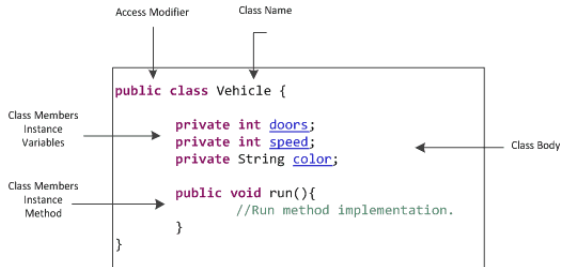
- Variables: A Java class uses variables to define data fields.
- Methods are used to define action for the object.
- Constructors are special kinds of method which initialize the data field of an object when it is created.
- Constructors are called automatically when a new object is created.

Fields

- field: A variable inside an object that is part of its state.
 - field: A variable inside an object that is part of its state.

Defining a class

- Blueprint



DEMO

Example

```
class Student{
    int id;//field or data member or instance variable
    String name;
}
class testStudent{
    public static void main(String args[]){
        Student s1=new Student();//creating an object of Student
        //The new keyword is used to allocate memory at run time.
        //All objects get memory in Heap memory area.
        System.out.println(s1.id);//accessing member through reference variable
        System.out.println(s1.name);
    }
}
```

Member Initialization

- Unitialized variables are a common source of bugs.
 - Using an uninitialized variable in method gets a compiler error.
 - Primitive data members in classes automatically get initialized to zero.
- Is the initialized value (zero) any better than a garbage value?

ways to initialize objects

- There are 3 ways to initialize object in java.
 - By reference variable
 - By method
 - By constructor

DEMO

Example

```
class Rectangle{
    int length;
    int width;
    void insert(int l, int w){
        length=l;
        width=w;
    }
    void calculateArea(){System.out.println(length*width);}
}
class TestRectangle1{
    public static void main(String args[]){
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();
        r1.insert(11,5);
        r2.insert(3,15);
        r1.calculateArea();
        r2.calculateArea();
    }
}
```

Constructor in Java

- There are basically two rules defined for the constructor.
 - ❶ Constructor name must be same as its class name
 - ❷ Constructor must have no explicit return type

Types of java constructors

- There are two types of constructors:
 - ❶ Default constructor (no-arg constructor): A constructor that have no parameter is known as default constructor.
 - ❷ Parameterized constructor: A constructor which requires parameters.

Constructor Overloading

- Constructor overloading is the same technique as we used for methods.
 - ① a class can have any number of constructors that differ in parameter lists.
 - ② .The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Rules for creating a constructor

- Constructor in java is a special type of method that is used to initialize the object.
- Java constructor is invoked at the time of object creation.
- It constructs the values i.e. provides data for the object that is why it is known as constructor.

Example

```
class Student5{
    int id;
    String name;
    int age;
    Student5(int i,String n){
        id = i;
        name = n;
    }
    Student5(int i,String n,int a){
        id = i;
        name = n;
        age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
        Student5 s1 = new Student5(111,"Karan");
        Student5 s2 = new Student5(222,"Aryan",25);
        s1.display();
        s2.display();
    }
}
```

Difference between constructor and method

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

Static keyword

- The static keyword in java is used for memory management mainly.
- We can apply java static keyword with variables, methods, blocks and nested class.
- The static keyword belongs to the class than instance of the class.

Compare the two examples

```
class Student{  
    int rollno;  
    String name;  
    String college="ITS";  
}
```

```
class Student8{  
    int rollno;  
    String name;  
    static String college ="ITS";  
  
    Student8(int r,String n){  
        rollno = r;  
        name = n;  
    }  
    void display (){System.out.println(rollno+" "+name+" "+college);}  
  
    public static void main(String args[]){  
        Student8 s1 = new Student8(111,"Karan");  
        Student8 s2 = new Student8(222,"Aryan");  
  
        s1.display();  
        s2.display();  
    }  
}
```

this keyword

- Here is given the 6 usage of java this keyword:
 - ① this can be used to refer current class instance variable.
 - ② this can be used to invoke current class method (implicitly)
 - ③ this() can be used to invoke current class constructor
 - ④ this can be passed as an argument in the method call
 - ⑤ this can be passed as argument in the constructor call.
 - ⑥ this can be used to return the current class instance from the method.

Compare the two examples

```
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee){
        rollno=rollno;
        name=name;
        fee=fee;
    }
    void display(){
        System.out.println(rollno+" "+name+" "+fee);
    }
}
class TestThis1{
    public static void main(String args[]){
        Student s1=new Student(111,"Ahmad",5000f);
        Student s2=new Student(112,"Jawid",6000f);
        s1.display();
        s2.display();
    }
}
```

```
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee){
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
    }
    void display(){
        System.out.println(rollno+" "+name+" "+fee);
    }
}
class TestThis1{
    public static void main(String args[]){
        Student s1=new Student(111,"Ahmad",5000f);
        Student s2=new Student(112,"Jawid",6000f);
        s1.display();
        s2.display();
    }
}
```

this keyword for reusing constructor

```
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee){
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
    }
    Student(int rollno,String name,String course,float fee){
        this(rollno,name,course);//reusing constructor
        this.fee=fee;
    }
    void display(){
        System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis1{
    public static void main(String args[]){
        Student s1=new Student(111,"Ahmad",5000f);
        Student s2=new Student(112,"Jawid",6000f);
        s1.display();
        s2.display();
    }
}
```


Access Modifiers

```
class Circle {  
    private double radius;  
  
    public Circle() {  
        this(1.0);  
    }  
  
    public Circle(double newRadius) {  
        radius = newRadius;  
    }  
  
    public double getArea() {  
        return radius * radius * 3.14159;  
    }  
}
```

Typically:

Attributes are
declared private

Methods are
declared public

Visibility Modifiers

- Public
 - The class, data, or method is visible to any class.
- Private
 - The data or method can only be accessed by the declaring class.

Accessing private attributes

- How can code in any other class access them?


Accessing private attributes

- How can code in any other class access them?
- Programmer can provide methods to get and set them.

Accessing private attributes

```
public class Driver {  
    public static void main(String[] args)  
    {  
        Circle s1, s2;  
        s1 = new Circle(14);  
  
        s2 = new Circle(7);  
  
        System.out.println("Radius = ", s1.radius);  
        outcome = s2.getArea(); //ok!  
    }  
}
```

Illegal: because attribute *radius* is hidden or private!




Get/Set Methods

- Get method or Accessor
 - A method that returns the value of an attribute e.g., `getUFID()` return studentUFID;
- Set method or Mutator
 - A method that changes the value of an attribute e.g., `getUFID()` return studentUFID;

Accessing private attributes

```
public class Driver {  
    public static void main(String[] args)  
    {  
        Circle s1, s2;  
        s1 = new Circle(14);  
  
        s2 = new Circle(7);  
  
        System.out.println("Radius = ", s1.radius);  
        outcome = s2.getArea(); //ok!  
    }  
}
```

Illegal: because attribute *radius* is hidden or private!



Example 1

```
public class Circle {  
    private double radius;  
  
    public double getRadius() {  
        return radius;  
    }  
  
    public void setRadius(double radius){  
        this.radius = radius;  
    }  
  
    public Circle() { this(1.0); }  
    public Circle(double r) { setRadius(r); }  
  
    public getArea() { return 3.14*radius*radius; }  
}
```


Example 2

```
public class Driver {  
  
    Circle c1 = new Circle(14);  
    Circle c2 = new Circle(7);  
  
    System.out.println(c1.getRadius()); //ok!  
    c1.setRadius(5); //ok!  
    boolean outcome = s2.getArea(); //ok!  
}
```

Example 3

```
public class Circle {  
    private double radius;  
  
    public double getRadius() {  
        return radius;  
    }  
  
    public void setRadius(double r){  
        if (r>0) radius = r;  
    }  
  
    public Circle() { this(1.0); }  
    public Circle(double r) { setRadius(r); }  
  
    public getArea() { return 3.14*radius*radius;  
}
```

Example 4

```
public class Student {  
    private String name;  
    private String ssn;  
    private float gpa;  
  
    public Student(int i, String n) {  
        setSsn(i); setName(n); setGpa(0.0f);  
    }  
  
    // other constructors and methods, not shown here  
  
    public String getName () { return name; }  
    public void setName(String newName) { name = newName;}  
  
    public String getSsn () { return ssn; }  
    public void setSsn(String s) { ssn = s;}  
  
    public float getGpa () { return gpa; }  
    public void setGpa(float newGpa) { gpa = newGpa;}  
}
```

Example 5

```
public class Student {
    private String name;
    private String ssn;
    private float gpa;
    private int numDsFs;

    public Student(int i, String n) {
        setSsn(i);    setName(n);    setGpa(0.0f);    setNumDsFs(0);
    }

    // other methods, not shown here

    public boolean isOnProbation() { if(numDsFs > 3) return true;
                                    else return false; }

    public String getName () { return name; }
    public void setName(String newName) { name = newName;}
    public String getSsn () { return ssn; }
    public float getGpa () { return gpa; }
    public void setGpa(float newGpa) { gpa = newGpa;}

    public void setNumDsFs(int n) { numDsFs = n; }
```

1

Benefits of Information Hiding

- Allows **Data Validation**
- Allows **control over the level of access** given for an attribute
- Simplifies **Code Maintenance**

Question

