Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf

yousuf@juniv.edu

Constructor in Java

- C**onstructor 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.

Constructor in Java

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

Constructor in Java

- Types of java constructors
  - There are two types of constructors:
    - Default constructor (no-arg constructor)
    - Parameterized constructor

Java Default Constructor

- A constructor that have no parameter is known as default constructor.

```
class Bike1{
  Bike1()
   {
       System.out.println("Bike is created");
   }
  public static void main(String args[]){
  Bike1 b=new Bike1();
  }
}
Output:
Bike is created
```

- **Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**

Java parameterized constructor

- A constructor that have parameters is known as parameterized constructor.

- Why use parameterized constructor?
  - Parameterized constructor is used to provide different values to the distinct objects

Example of parameterized constructor

```
class Student4{
    int id;
    String name;

    Student4(int i, String n){
    id = i;
    name = n;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Constructor Overloading in Java

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.

- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
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();
   }  }
```

Output:
111 Karan 0
222 Aryan 25

Difference between constructor and method in java

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.

Java Copy Constructor

- There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.
- There are many ways to copy the values of one object into another in java. They are:
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

```
class Student6{
   int id;
   String name;

   Student6(int i,String n){
   id = i;
   name = n;
   }
   Student6(Student6 s){
   id = s.id;
   name =s.name;
   }
   void display(){System.out.println(id+" "+name);}

   public static void main(String args[]){
   Student6 s1 = new Student6(111,"Karan");
   Student6 s2 = new Student6(s1);
   s1.display();
   s2.display();
  }
}
```

In this example, we are going to **copy the values of one object into another using java constructor.**

Output:

111 Karan
111 Karan

Copying values without constructor

```
class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;
    }
    Student7(){}
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student7 s1 = new Student7(111,"Karan");
    Student7 s2 = new Student7();
    s2.id=s1.id;
    s2.name=s1.name;
    s1.display();
    s2.display();
   }
}
```

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

Output:

111 Karan
111 Karan

Java Object as parameter

```
// Objects may be passed to methods.
class Test {
  int a, b;

  Test(int i, int j) {
    a = i;
    b = j;
  }

  // return true if o is equal to the invoking object
  boolean equals(Test o) {
    if(o.a == a && o.b == b) return true;
    else return false;
  }
}

class PassOb {
  public static void main(String args[]) {
    Test ob1 = new Test(100, 22);
    Test ob2 = new Test(100, 22);
    Test ob3 = new Test(-1, -1);

    System.out.println("ob1 == ob2: " + ob1.equals(ob2));
    System.out.println("ob1 == ob3: " + ob1.equals(ob3));
  }
}
```

This program generates the following output:
ob1 == ob2: true
ob1 == ob3: false

Java Object as parameter

- When you pass an object to a method, the situation changes dramatically, **because objects are passed by what is effectively call-by-reference**.
- Keep in mind that when you create a variable of a class type, you are only creating a reference to an object. Thus, when you pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
- This effectively means that objects act as if they are passed to methods by use of call-by-reference. Changes to the object inside the method do affect the object used as an argument. For example, consider the following program:

```
// Objects are passed through their references.

class Test {
  int a, b;

  Test(int i, int j) {
    a = i;
    b = j;
  }

  // pass an object
  void meth(Test o) {
    o.a *= 2;
    o.b /= 2;
  }
}

class PassObjRef {
  public static void main(String args[]) {
    Test ob = new Test(15, 20);
    System.out.println("ob.a and ob.b before call: " +
                       ob.a + " " + ob.b);

    ob.meth(ob);

    System.out.println("ob.a and ob.b after call: " +
                       ob.a + " " + ob.b);
  }
}
```

This program generates the following output:

ob.a and ob.b before call: 15 20
ob.a and ob.b after call: 30 10

In this case, the actions inside meth( ) have affected the object used as an argument.

Returning Objects

- A method can return any type of data, including class types that you create. For example, in the following program, the **incrByTen( )** method returns an object in which the value of **a** is ten greater than it is in the invoking object.

```
// Returning an object.
class Test {
  int a;

  Test(int i) {
    a = i;
  }

  Test incrByTen() {
    Test temp = new Test(a+10);
    return temp;
  }
}

class RetOb {
  public static void main(String args[]) {
    Test ob1 = new Test(2);
    Test ob2;

    ob2 = ob1.incrByTen();
    System.out.println("ob1.a: " + ob1.a);
    System.out.println("ob2.a: " + ob2.a);
    ob2 = ob2.incrByTen();
    System.out.println("ob2.a after second increase: "
                       + ob2.a);
  }
}
```

The output generated by this program is shown here:
ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22

each time incrByTen( ) is invoked, a new object is created, and a reference to it is returned to the calling routine.

Thank you