Tutorial 2(Module-02)

SMITA SANKHE

Assistant Professor

Department of Computer Engineering

Contents

- Classes
- Objects
- Methods
- Static and Non-Static methods
- Constructors
- Constructor overloading
- Constructor Vs Method
- Types of Constructors
- This keyword
- Destructor
- Access Specifiers in java

Objects

- **Object -** Objects have states and behaviors. Object is the **physical** as well as **logical** entity. Objects are created using following keywords.

  By new keyword- New keyword is used to allocate memory at runtime.

  By newInstance() method

  By clone() method

  By factory method etc.

- An object that have no reference is known as annonymous object and it can be used when object is used only once.

Eg:- new Classname().

- **Object is an instance of a class.** An object has three characteristics:
  - **state:** represents data (value) of an object.
  - **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
  - **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.

Class , Object, Method, Constructors

SMITA SANKHE

Assistant Professor

Department of Computer Engineering

Example

```
class Student1{
 int id;//data member (also instance variable)
 String name;//data member(also instance variable)
 public static void main(String args[]){
Student1 s1=new Student1();//creating an object of Stude
nt
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Ctd..

```
class Rectangle{
 int length;
 int width;

 void insert(int l,int w){
  length=l;
  width=w;
 }

 void calculateArea(){System.out.println(length*width);}

 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects

  r1.insert(11,5);
  r2.insert(3,15);

  r1.calculateArea();
  r2.calculateArea();
}
}
```

Constructors

- A constructor **initializes an object** when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.
- We use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

Types of Constructors

- There are two types of constructors:
  - **Default constructor** (no-arg constructor): Default constructor provides the default values to the object like 0, null etc. It will be invoked at the time of object creation.depending on the type. <class_name>(){}
  - **Parameterized constructor**: A constructor that have parameters. Parameterized constructor is used to provide different values to the distinct objects.
- There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

Constructor overloading

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Method

- A Java method is a collection of statements that are grouped together to perform an operation. Syntax is

modifier returnType nameOfMethod (Parameter List) { // method body }

- Modifier is optional. There are two ways in which a method is called i.e. method returns a value or returning nothing

Si no	Constructor	Method
1	Constructor is used to initialize the **state** of an object.	Method is used to expose **behavior** of an object.
2	Constructor must not have return type	Method must have return type.
3	Constructor is invoked implicitly.	Method is invoked explicitly.
4	The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
5	Constructor name must be same as the class name	Method name may or may not be same as class name.

Parametarized constructor

•A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation

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

- **this** is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor. Using *this* you can refer the members of a class such as constructors, variables and methods.

- usage of java this keyword**.**
    1. this keyword can be used to refer current class instance variable.
    2. this() can be used to invoke current class constructor.
    3. this keyword can be used to invoke current class method (implicitly)
    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 keyword can also be used to return the current class instance.

Destructor

- A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Finalize()

- The **finalize()** method is equivalent to a **destructor** of C++. When the job of an object is over, or to say, the object is no more used in the program, the object is known as **garbage**. The process of removing the object from a running program is known as **garbage collection**. finalize() method can be best utilized by the programmer to close the I/O streams, JDBC connections or socket handles etc.

Access Specifiers in Java

- **Public:** A **class, method, constructor, interface** etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. However if the public class we are trying to access is in a different package, then the public class still need to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
- **Protected: Variables, methods and constructors** which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier **cannot be applied to class and interfaces**. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.
- **Private: Methods, Variables and Constructors** that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. **Class and interfaces cannot be private**. Variables that are declared private can be accessed outside the class if public getter methods are present in the class. Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.
- **Default:** Default access modifier means we do not explicitly declare an access modifier for a **class, field, method,** etc. A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Scope of access specifiers

Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes

List of Java Object class methods.

- The **Object class** is the parent class of all the classes in java bydefault. In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know.

clone() - Creates and returns a copy of this object.

equals() - Indicates whether some other object is "equal to" this one.

finalize() - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

getClass() - Returns the runtime class of an object.

hashCode() - Returns a hash code value for the object.

notify() - Wakes up a single thread that is waiting on this object's monitor.

notifyAll() - Wakes up all threads that are waiting on this object's monitor.

toString() - Returns a string representation of the object. wait() - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

Dot operator in java

- It enables you to access instance variables of any objects within a class. It is used to call object methods. "*the* ***dot***" connects classes and objects to members.
- when you are connecting a **class** name to one of its **static** fields. An example of this is the dot between "System" and "out" in the statements we use to print stuff to the console window. System is the name of a class included in every Java implementation. It has an object reference variable that points to a *PrintStream* object for the console. So, "System.out.println("text") invokes the println() method of the System.out object.

Example

```
class Exp07
{
public static void main(String args[])
{
 Scanner src=new Scanner(System.in);
 System.out.println("enter distance1 in feet and inches");
 double m=src.nextDouble();
 double n=src.nextDouble();
 Distance d1=new Distance(m,n);
 System.out.println("enter distance2 in feet and inches");
 m=src.nextDouble();
 n=src.nextDouble();
 Distance d2=new Distance(m,n);
 Distance d3=new Distance();
 d3.add(d1,d2);
 d3.display();
 if(d1.compare(d2))
 System.out.println("distances are equal");
 else
 System.out.println("distances are not equal"); }}
```