MOD -02 -> OOPs

LUMINAR
TECHNOLAB

Class, Object & Constructor

LUMINAR
TECHNOLAB

OOPs (Object Oriented Programming System)

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

It simplifies the software development and maintenance by providing some concepts:

1. Object
2. Class
3. Data Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism
7. Dynamic Binding

Class

1. User defined datatype
2. Once a class has been defined, we can create any number of objects belonging to that class.
3. A class is thus a collection of objects of the same type.

Object

1. Objects are variables of the type class.
2. Objects are the basic run time entities in an object oriented system.
3. An instance of class.

Data Abstraction (reduce the complexity)

1. The process of revealing only the essential things to the user without showing the internal implementation details.
2. Implemented using the abstract class and interfaces.

Encapsulation (data security)

1. The process of binding data and thereby restricting access to some of its components.
2. Implemented using access modifiers. Eg. Java Bean class.

Data hiding

1. The process of hiding data of one entity from another entity.
2. The purpose of data hiding is to prevent correction of data by other entities.
3. Made possible by encapsulation.

Inheritance

1. The process by which one class acquire the properties of another class.

Polymorphism

1. An operation may exhibits different behaviors in different instances.

Dynamic Binding/Dynamic Polymorphism/ Dynamic method dispatch/ Runtime Polymorphism

1. The code associated with a given procedure call is not known until the time of call at runtime.

Advantages

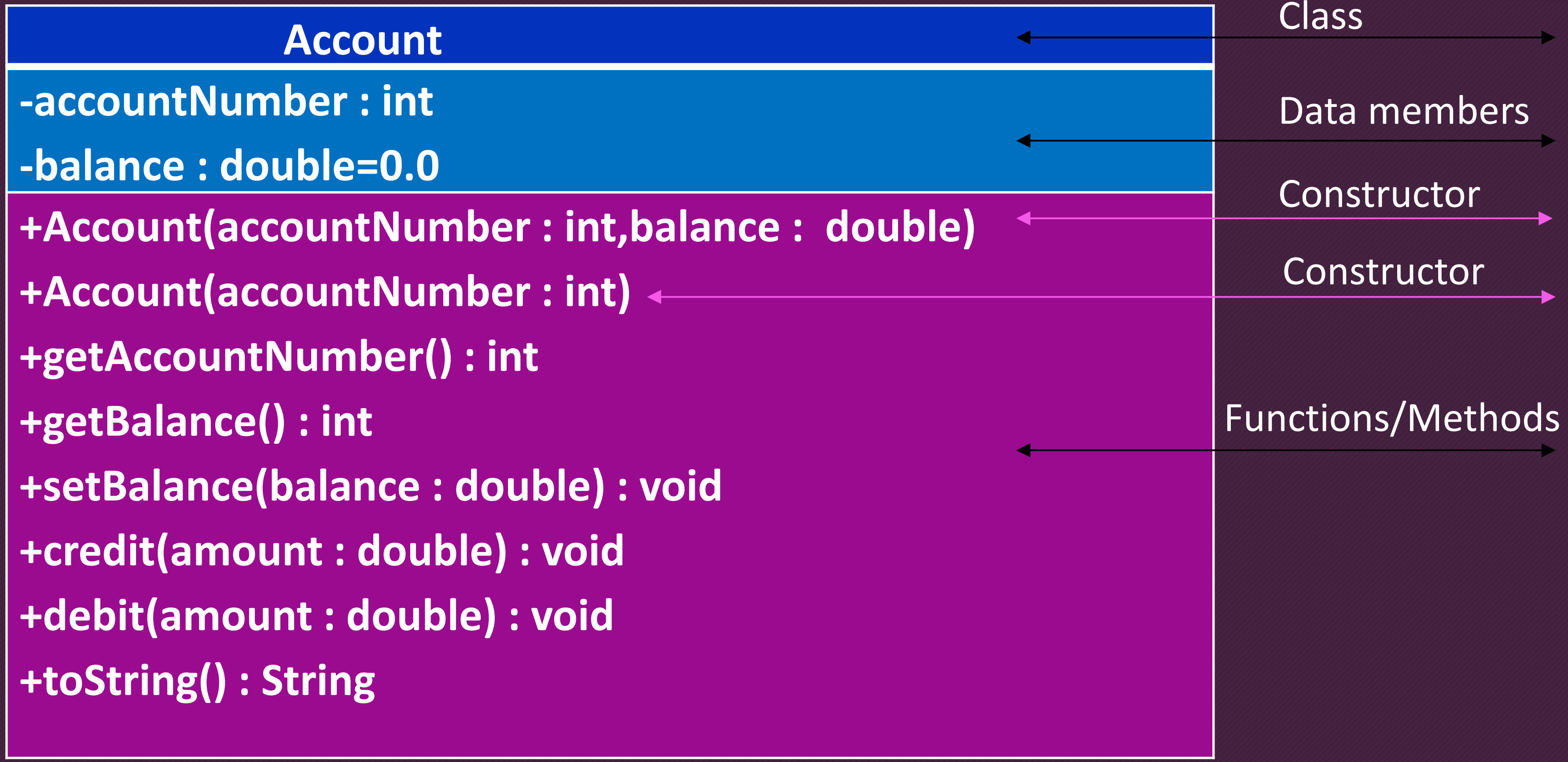
- OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
- OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- OOPs provide ability to simulate real-world event much more effectively.

The Java class has mainly two type of access level

`Default`: class objects are accessible only inside the package.
`Public`: class objects are accessible in code in any package.

Note: If no access modifier used it is taken as default.

Class & its members



```
Account
-accountNumber : int
-balance : double=0.0
+Account(accountNumber : int,balance :  double)
+Account(accountNumber : int)
+getAccountNumber() : int
+getBalance() : int
+setBalance(balance : double) : void
+credit(amount : double) : void
+debit(amount : double) : void
+toString() : String
```

Objects

Object Creation

```
<class name> <reference variable name>=new <constructor>;
```

Declaring Objects

```
classname classVar = new classname( );
```

In some cases we can declare reference and later allocate object.

- `classname classVar` //declare reference to object
- `classVar = new classname( );` // allocate an object

Constructor

Constructor is a special member of a class which is used to initialize the state of an object. It provides the values to the data members at the time of object creation that is why it is known as constructor.

- A constructor initializes an object immediately upon creation.
- Same name as the class
- They have no return type, not even void
- When you do not explicitly define a constructor for a class, then Java creates a default constructor for the class.
- Once you define your own constructor, the default constructor is no longer used.

Characteristics of constructor

1. A constructor must have the same name as of its class.
2. It is invoked at the time of object creation and used to initialize the state of an object.
3. It does not have an explicit return type.

Types of constructor

1. Default or no-argument constructor.
2. Parameterized constructor.

Default or no-argument constructor

A constructor with no parameter is known as default or no-argument constructor. If no constructor is defined in the class then compiler automatically creates a default constructor at the time of compilation.

Why default constructor is used?

Default constructor is used to provide default values to the object properties i.e. to provide default state of an object.

NOTE

If no constructor is defined in the class then compiler automatically creates a default constructor at the time of compilation.

Does a constructor return any value?

Yes, a constructor implicitly returns the instance of the current class.

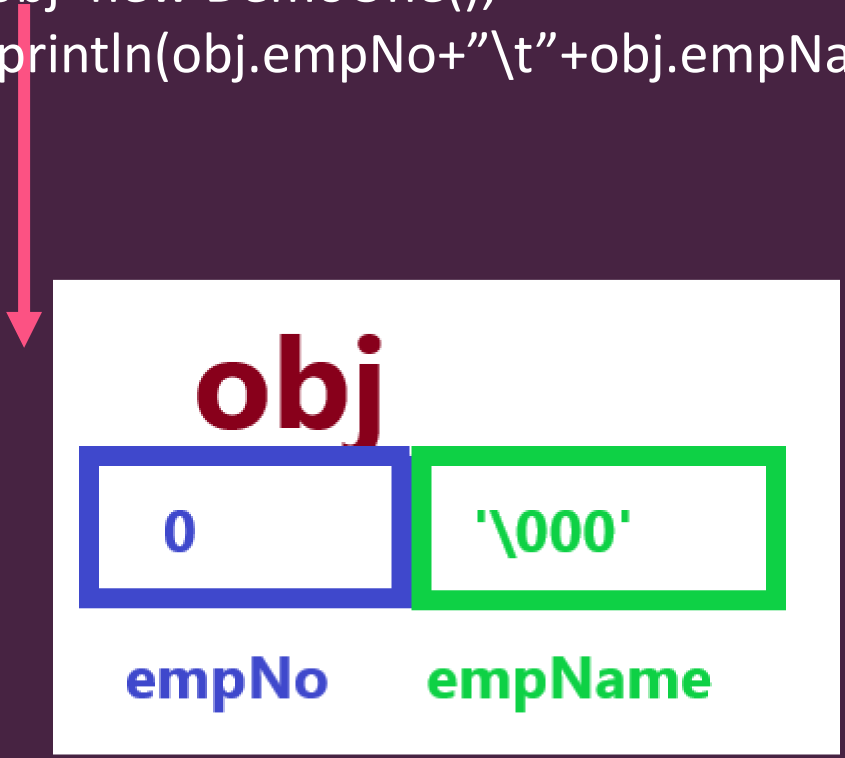
Default constructor in Java

Java (compiler)automatically creates default constructor if there is no default or parameterized constructor written by user.

1. main() in same class

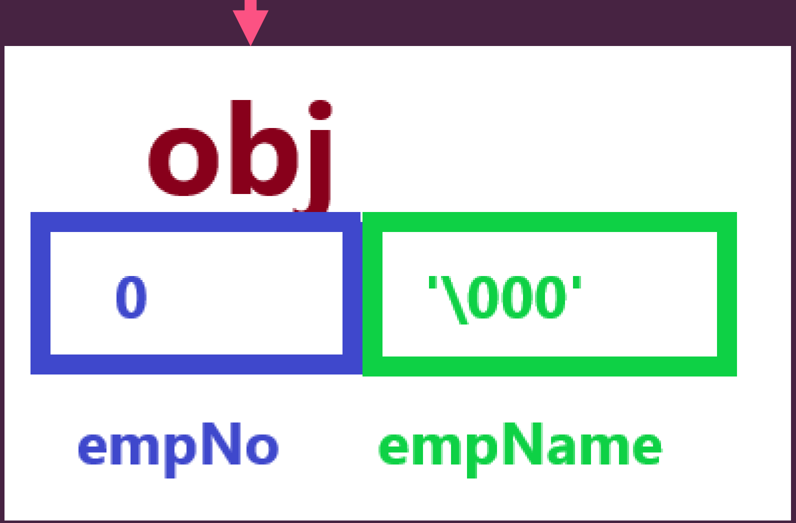
```
class DemoOne{
      int empNo;
      String empName;

      public static void main(String[] arg){
              DemoOne obj=new DemoOne();
             System.out.println(obj.empNo+"\t"+obj.empName);
      }
}
```



2. main() in separate class

```
class Employee{
        int empNo;
        String empName;
}
public class DemoTwo{
        public static void main(String[] arg){
                Employee obj=new Employee();
                System.out.println(obj.empNo+”\t”+obj.empName);
        }
}
```



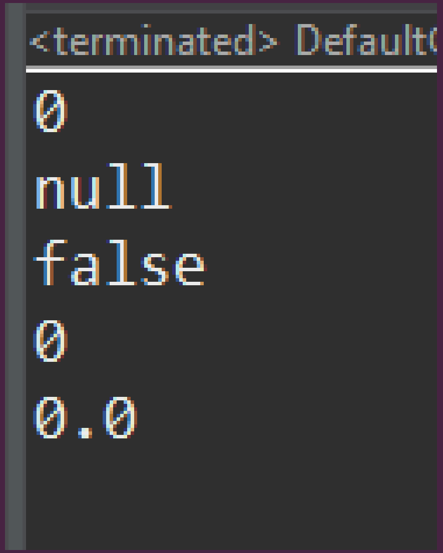
```java
package oopspack;


class Test{
    int no;
    Test test;
    boolean flag;
    byte byteValue;
    float amount;
}

public class DefaultConstructorTest {
    public static void main(String[] args) {
    Test obj=new Test();
    System.out.println(obj.no);
    System.out.println(obj.test);
    System.out.println(obj.flag);
    System.out.println(obj.byteValue);
    System.out.println(obj.amount);

}


}
```



```
<terminated> Default
0
null
false
0
0.0
```

3. Default constructor written by user

```
class Employee{
	int empNo;
	String empName;
	Employee(){}

}
public class DemoThree{
	public static void main(String[] arg){
		Employee obj=new Employee();
		System.out.println(obj.empNo+”\t”+obj.empName);
	}
}
```

4. Default constructor written by user

```
class Employee{
    int empNo;
    String empName;
    Employee(){
        empNo=0;
        empName=null;
    }

}
public class DemoFour{
    public static void main(String[] arg){
        Employee obj1=new Employee();
        System.out.println(obj1.empNo+”\t”+obj1.empName);
        Employee obj2=new Employee();
        System.out.println(obj2.empNo+”\t”+obj2.empName);

    }
}
```

5. Default constructor written by user

```
class Employee{
        int empNo;
        String empName;
        Employee(){
                empNo=23;
                empName="bini";
        }

}
public class DemoFive{
        public static void main(String[] arg){
                Employee obj1=new Employee();
                System.out.println(obj1.empNo+"\t"+obj1.empName);
                Employee obj2=new Employee();
                System.out.println(obj2.empNo+"\t"+obj2.empName);
        }
}
```

6. Default constructor written by user

```
class Employee{
        int empNo;
        String empName;
        Employee(){
                empNo=23;
        }

}
public class DemoSix{
        public static void main(String[] arg){
                Employee obj1=new Employee();
                System.out.println(obj1.empNo+”\t”+obj1.empName);
                Employee obj2=new Employee();
                System.out.println(obj2.empNo+”\t”+obj2.empName);
        }
}
```

Parameterized Constructors

- A constructor with one or more arguments is known as parameterized constructor.

Why parameterized constructor is used?

Parameterized constructor is used to provide values to the object properties. By use of parameterized constructor different objects can be initialize with different states.

- While a `constructor` in a class does initialize an `object`, it is not very useful all `objects` have the same dimensions.
- What is needed is a way to construct `objects` of various dimensions. The easy solution is to add parameters to the constructor.

NOTE

If a class contains parameterized constructor and default constructor is needed than default constructor has to be defined explicitly. In this case compiler will not provide default constructor.

7. Parameterized constructor with direct value

```java
package oopspack;


class ParameterTest{
    int no;
    ParameterTest(int num){
        no=num;
    }
}


public class ParameterizedConstructorTest {


    public static void main(String[] args) {
        ParameterTest obj=new ParameterTest(34);
        System.out.println(obj.no);

    }

}
```

8. Default & Parameterized constructor

```
package oopspack;
class Trial{
        int no;
        Trial(int num){
            num=no;
        }
        //Trial(){}
}
public class DefaultConstructorExplicit {


    public static void main(String[] args) {
        Trial trail=new Trial(10);
        System.out.println(trail.no);
        Trial trail=new Trial(); //The constructor Trial() is undefined
    }


}
```

9. Parameterized constructor with dynamic value

```
Import java.util.Scanner;
class Employee{
      int empNo;
      String empName;
      Employee(int empId){
             empNo= empId;

      }

}
public class DemoSeven{
      public static void main(String[] arg){
            int id;
            Scanner sc=new Scanner(System.in);
            System.out.println(“Enter the id”);
            id=sc.nextInt();
            Employee obj1=new Employee(id);
            System.out.println(obj1.empNo+”\t”+obj1.empName);
        }
}
```

Copy constructor in Java

```java
package oopspack;
class Admin{
    int empRoll;
    String empName;
    public Admin(int roll,String name) {
        empRoll=roll;
        empName=name;
    }
    public Admin(Admin admin) {
        empRoll=admin.empRoll;
        empName=admin.empName;
    }
    public void printDetails() {
        System.out.println(empRoll+" "+empName);
    }
}
```

```java
public class CopyConstructor {
    public static void main(String[] args) {
        Admin objOne=new Admin(23,"Arun");
        objOne.printDetails();
        Admin objTwo=new Admin(objOne);
        objTwo.printDetails();
    }
}
```

Difference between constructor and method

Constructor	Method
1. It has same name as of class. 2. Invoked implicitly. 3. Must not have any explicit return type. 4. It is used to initialize the state of an object.	1. It may or may not have same name as of class. 2. Invoked explicitly. 3. Must have a return type. 4. It is used to show behavior of an object.

Thank you ☺ Happy coding ☺