Lecture 5

Concept of OOPS

OOPs (Object Oriented Programming System)

oops is a programming language organised around object

Object means a real word entity such as pen, chair, table etc. **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:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

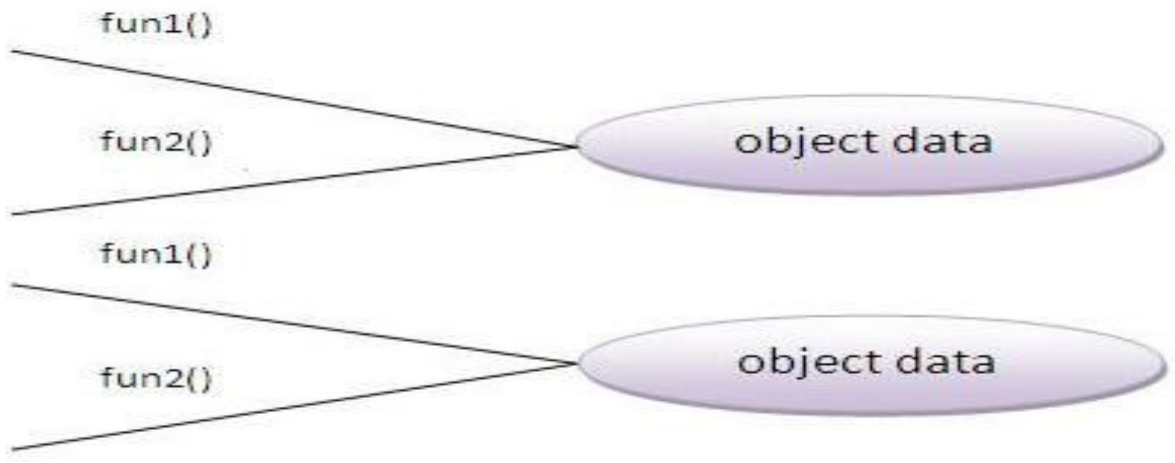
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs

1)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.

2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.

3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

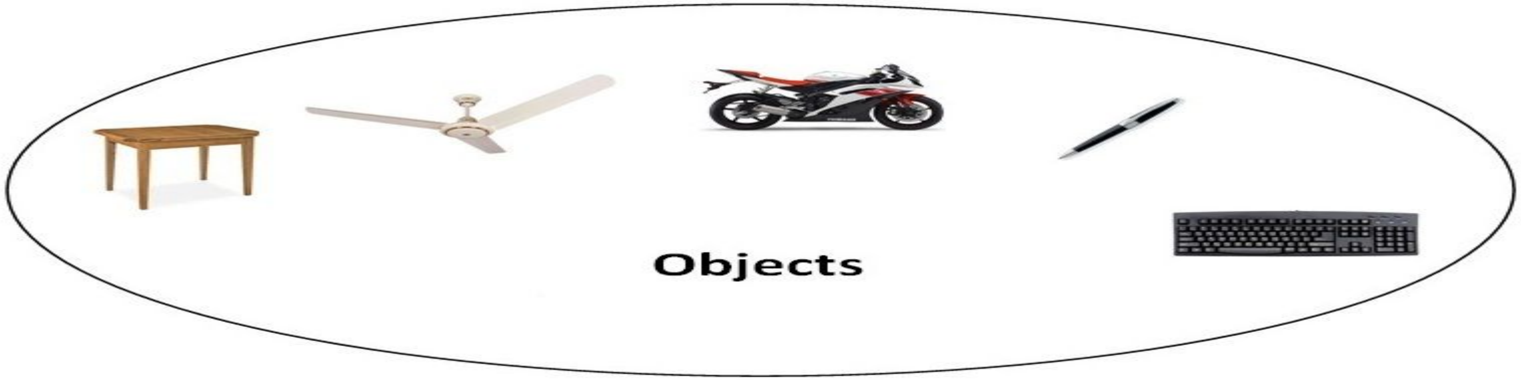


Object in Java

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.
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.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.



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.
Object Definitions:

- Object is *a real world entity*.
- Object is *a run time entity*.
- Object is *an entity which has state and behavior*.
- Object is *an instance of a class*.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
A class in Java can contain:

- **fields**
- **methods**
- **constructors**
- **blocks**
- **nested class and interface**

```
class <class_name>{

    field;

    method;

}
```

Object and Class Example:

```
class Student{
 int id;//field or data member or instance variable
 String name;

 public static void main(String args[]){
  Student s1=new Student();//creating an object of Student
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

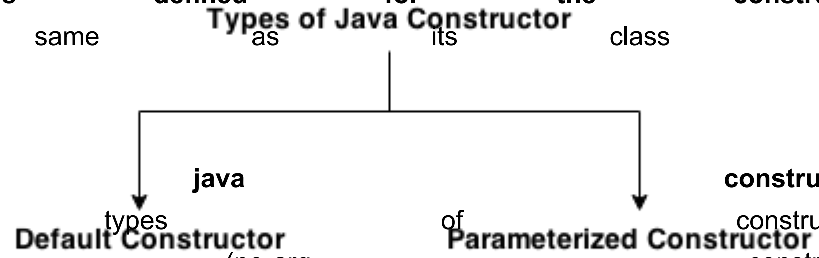
We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

```
class Student{
 int id;
 String name;
}
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Constructor in Java

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.

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



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

Java Default Constructor
A constructor that have no parameter is known as default constructor.

Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Java static variable

If you declare any variable as static, it is known static variable.

The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
The static variable gets memory only once in class area at the time of class loading.
Advantage of static variable

It makes your program memory efficient (i.e it saves memory).

Inheritance in Java

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.
Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

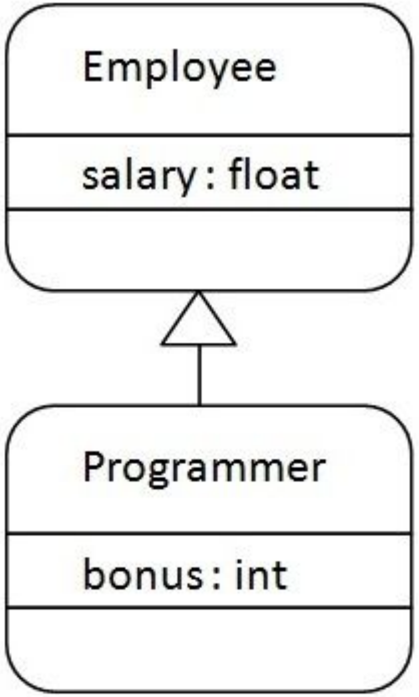
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name

{

   //methods and fields

}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

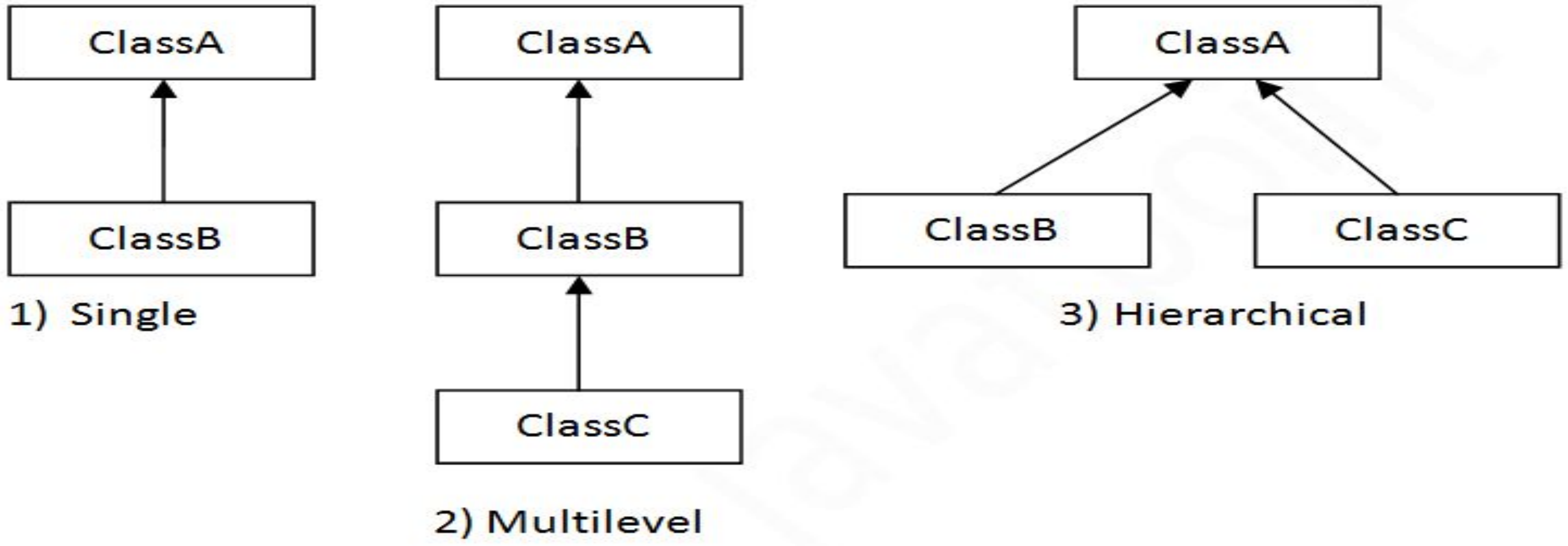
Java Inheritance Example

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```



Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



Single Inheritance Example

```
class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}
```

Multilevel Inheritance Example

```
class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class BabyDog extends Dog{

void weep(){System.out.println("weeping...");}

}

class TestInheritance2{

public static void main(String args[]){

BabyDog                              d=new                              BabyDog();

d.weep();

                                                                         d.bark();

d.eat();
```

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.
So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{

static int add(int a,int b){return a+b;}

static int add(int a,int b,int c){return a+b+c;}

}

class TestOverloading1{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}
```

Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{

static int add(int a, int b){return a+b;}

static double add(double a, double b){return a+b;}

}

class TestOverloading2{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}
```

Method Overriding in Java

1.

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.
In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

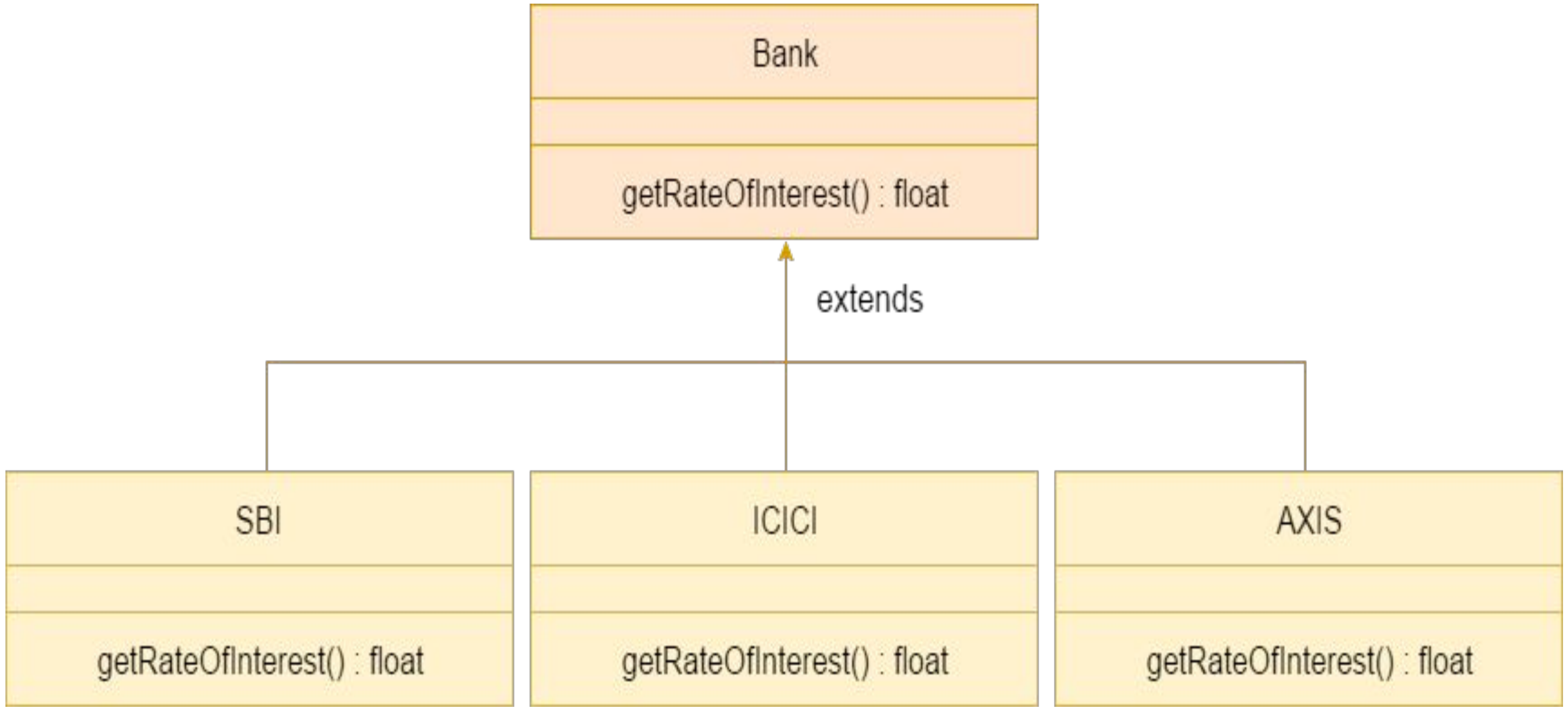
Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Example

```
class Vehicle{

void run(){System.out.println("Vehicle is running");}

}

class Bike2 extends Vehicle{

void run(){System.out.println("Bike is running safely");}


public static void main(String args[]){

Bike2 obj = new Bike2();

obj.run();

}
```

Example 2


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Final Keyword In Java

1. Final variable
2. Final method
3. Final class
4. Is final method inherited ?
5. Blank final variable
6. Static blank final variable
7. Final parameter
8. Can you declare a final constructor

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

Polymorphism in Java

Polymorphism in java is a concept by which we can perform a *single action by different ways*. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

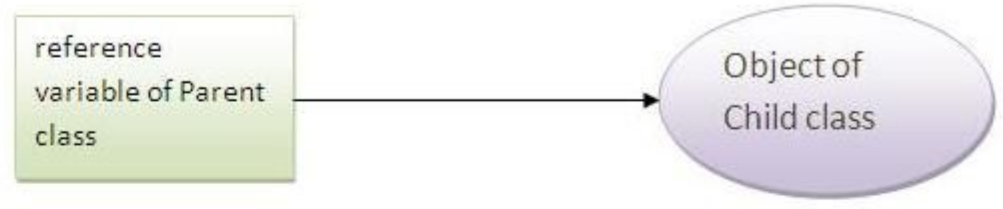
If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:



```
class A{}

class B extends A{}

A a=new B();//upcasting
```

super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Example

```
class Animal{

String color="white";  }

class Dog extends Animal{

String color="black";

void printColor(){

System.out.println(color);//prints color of Dog class

System.out.println(super.color);//prints color of Animal class  }  }

class TestSuper1{

public static void main(String args[]){

Dog d=new Dog();

d.printColor();

}}
```

Example 2 invoke parent class method

```
class Animal{

void eat(){System.out.println("eating...");}  }

class Dog extends Animal{

void eat(){System.out.println("eating bread...");}

void bark(){System.out.println("barking...");}

void work(){

super.eat();

bark();  }  }

class TestSuper2{

public static void main(String args[]){

Dog d=new Dog();

d.work();

}}
```

Example 3 invoke the parent class constructor.

```
class Animal{

Animal(){System.out.println("animal is created");}

}

class Dog extends Animal{

Dog(){

super();

System.out.println("dog is created");

}

}

class TestSuper3{

public static void main(String args[]){

Dog d=new Dog();

}}
```

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. Variable Cannot be changed
2. Method Cannot be overridden
3. Class Cannot be extended

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only.

Example of final variable

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;  /error
 }
public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}     /error
 public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Example of final class

```
final class Bike{}

class Honda1 extends Bike{  /error

  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){

  Honda1 honda= new Honda();

  honda.run();

  }

}
```

Final Method Can be Inherited but cannot be Overridden

Java instanceof

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(s instanceof Simple1);//true
 }
}
```

```
// true
```

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstaction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated.

```
abstract class A{}
```

abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

```
abstract void printStatus()
```

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Example Real time

Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.

```
abstract class Shape{

abstract void draw();  }  //In real scenario, implementation is provided by others i.e. unknown by end user

class Rectangle extends Shape{

void draw(){System.out.println("drawing rectangle");}  }

class Circle1 extends Shape{

void draw(){System.out.println("drawing circle");}  }  //In real scenario, method is called by programmer or user

class TestAbstraction1{

public static void main(String args[]){

Shape s=new Circle1();//In real scenario, object is provided through method e.g. getShape() method

s.draw();

}  }
```

Abstract class having constructor, data member, methods etc.

An abstract class can have data member, abstract method, method body, constructor and even main() method.

```
abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}   }


class Honda extends Bike{
void run(){System.out.println("running safely..");}   }
class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
obj.run();
 obj.changeGear();
}  }
```

Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**.

There can be only abstract methods in the java interface not method body.

It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

Why use Java interface

There are mainly three reasons to use interface.

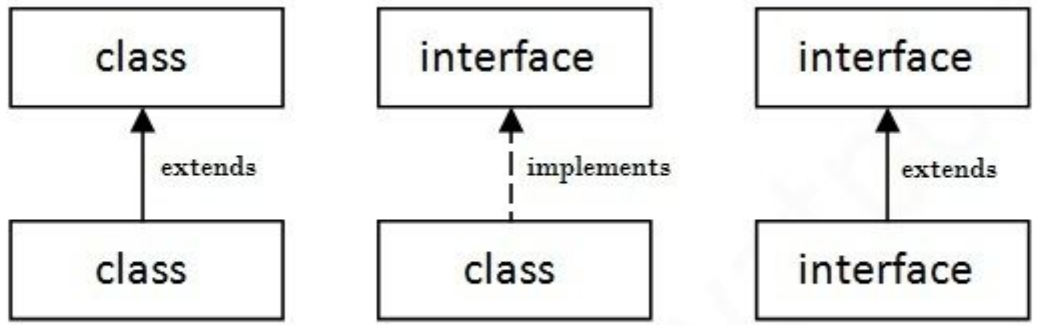
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Java 8 Interface Improvement

Since Java 8, interface can have default and static methods

Understanding relationship between classes and interface

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.

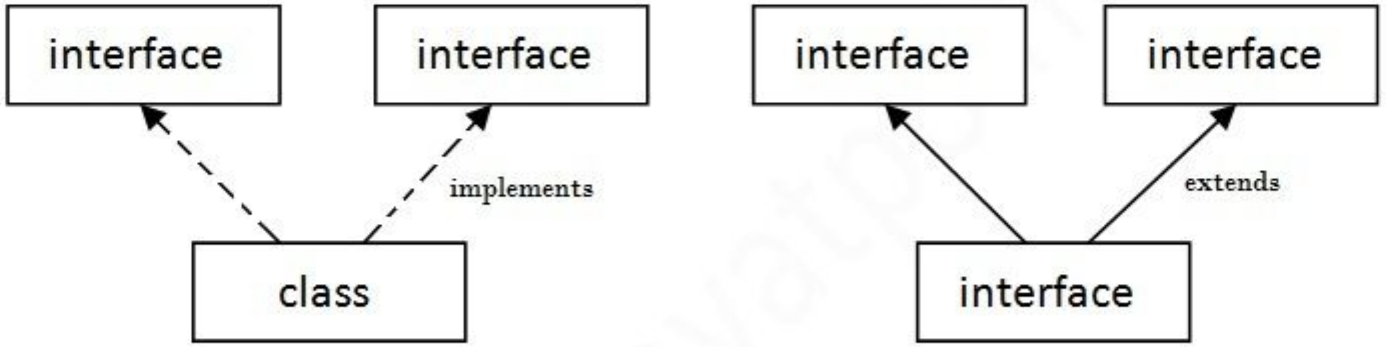


Interface Example

```
interface printable{

void print();

}

class A6 implements printable{

public void print(){System.out.println("Hello");}


public static void main(String args[]){

A6 obj = new A6();

obj.print();

 }

}
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

Multiple Inheritance

```
interface Printable{

void print();

}

interface Showable{

void show();

}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

A7 obj = new A7();

obj.print();

obj.show();   }  }
```

Multiple inheritance is not supported through class in java but it is possible by interface, why?

multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

```
interface Printable{

void print();

}

interface Showable{

void print();

}

class TestTnterface3 implements Printable, Showable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

TestTnterface1 obj = new TestTnterface1();

obj.print();   }  }
```

Java 8 Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method. Can create conflicts in multiple Inheritance.

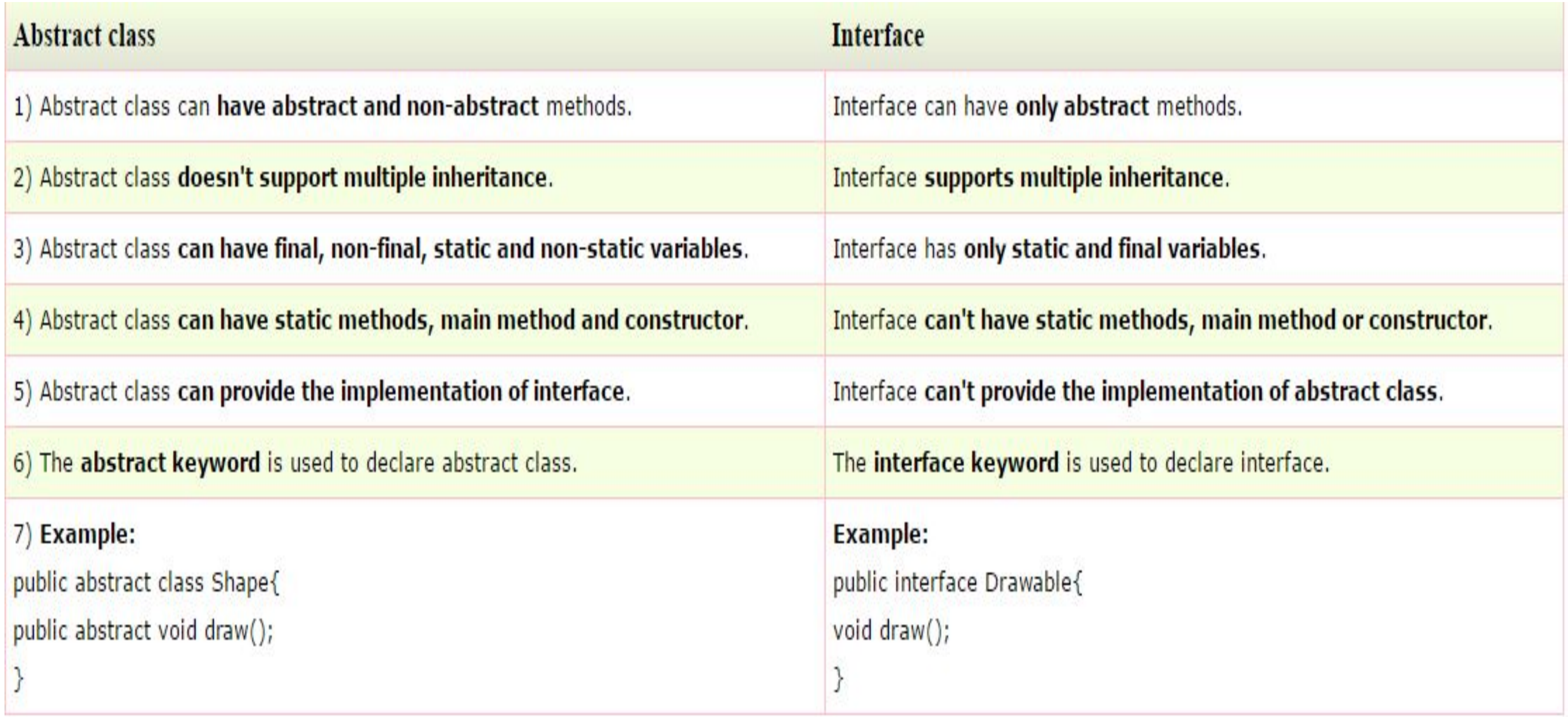
```
interface Drawable{

void draw();

default void msg(){System.out.println("default method");}

}

class Rectangle implements Drawable{

public void draw(){System.out.println("drawing rectangle");}

}

class TestInterfaceDefault{

public static void main(String args[]){

Drawable d=new Rectangle();

d.draw();

d.msg();  }}
```

Static Method in Interface

Since Java 8, we can have static method in interface.

```
interface Drawable{

void draw();

static int cube(int x){return x*x*x;}

}

class Rectangle implements Drawable{

public void draw(){System.out.println("drawing rectangle");}

}

class TestInterfaceStatic{

public static void main(String args[]){

Drawable d=new Rectangle();

d.draw();

System.out.println(Drawable.cube(3));

}}
```

Difference between abstract class and interface



Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance.**	Interface **supports multiple inheritance.**
3) Abstract class **can have final, non-final, static and non-static variables.**	Interface has **only static and final variables.**
4) Abstract class **can have static methods, main method and constructor.**	Interface **can't have static methods, main method or constructor.**
5) Abstract class **can provide the implementation of interface.**	Interface **can't provide the implementation of abstract class.**
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

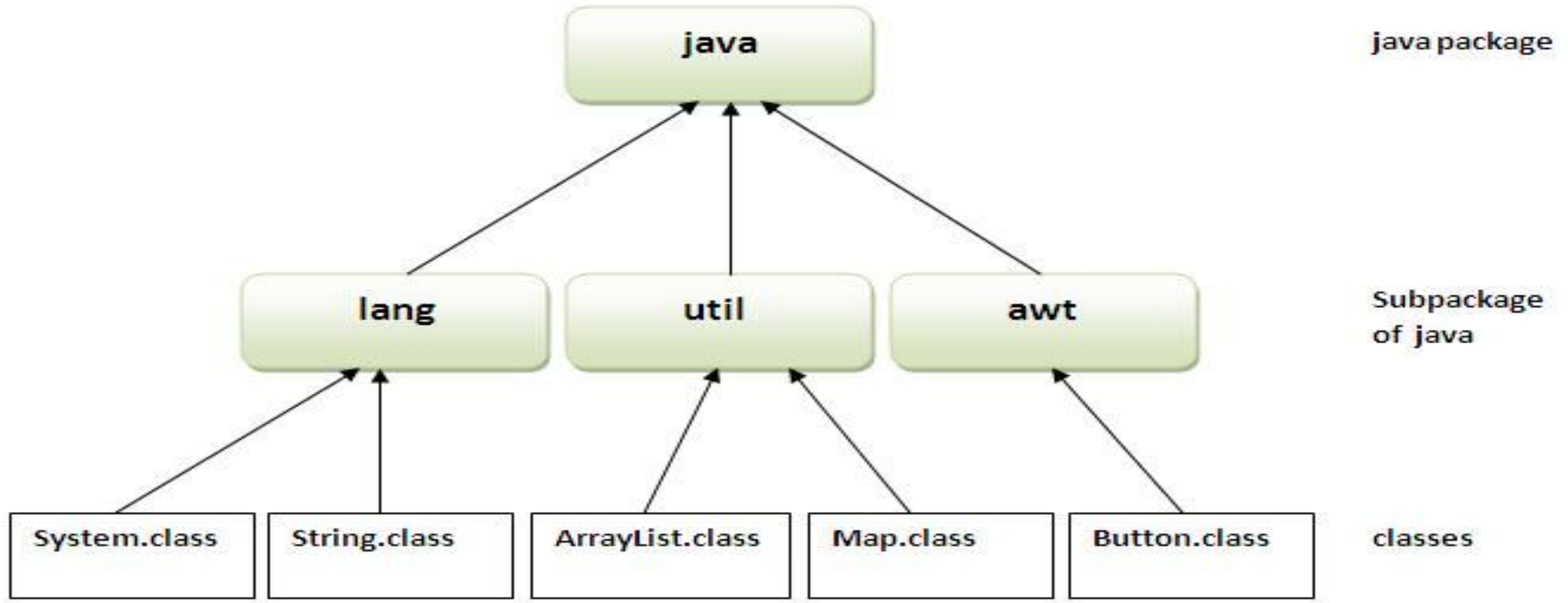
Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Map


java
java package
lang
util
awt
Subpackage of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

```
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();   }  }
```

Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.
The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

private access modifier

The private access modifier is accessible only within class.

```
class A{

private int data=40;

private void msg(){System.out.println("Hello java");}

}

public class Simple{

 public static void main(String args[]){

   A obj=new A();

  System.out.println(obj.data);//Compile Time Error

  obj.msg();//Compile Time Error

   }

}
```

Role of Private Constructor

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

default access modifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within package.

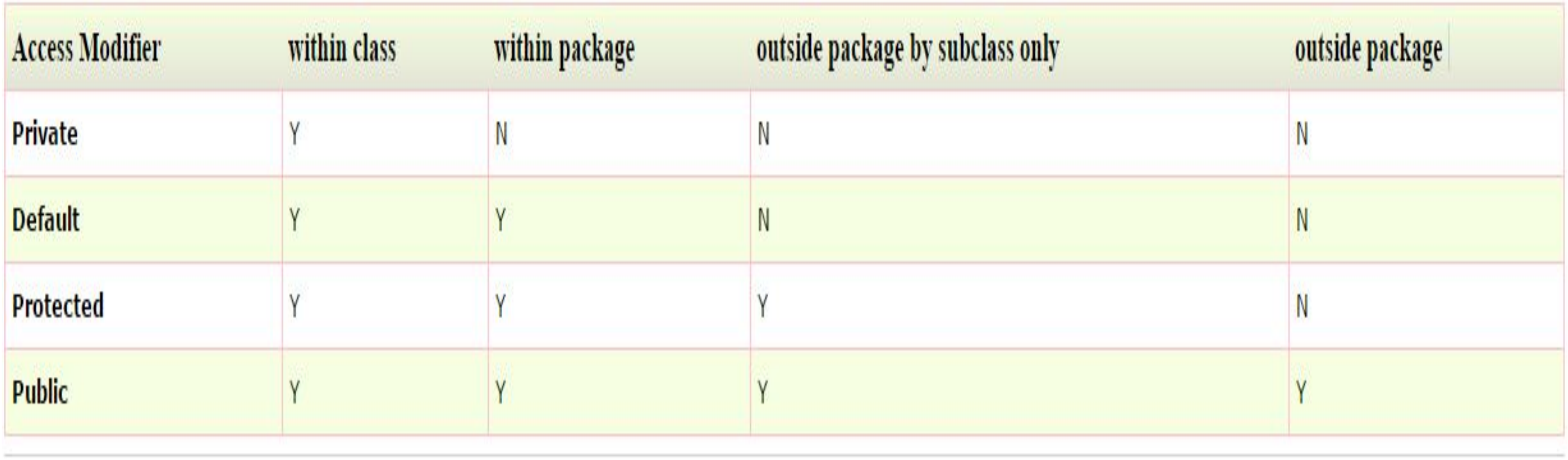
protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.
The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Understanding all java access modifiers



Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Encapsulation in Java

Encapsulation in java is a *process of wrapping code and data together into a single unit*, for example capsule i.e. mixed of several medicines.

We can create a fully encapsulated class in java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of fully encapsulated class.

Advantage of Encapsulation in java

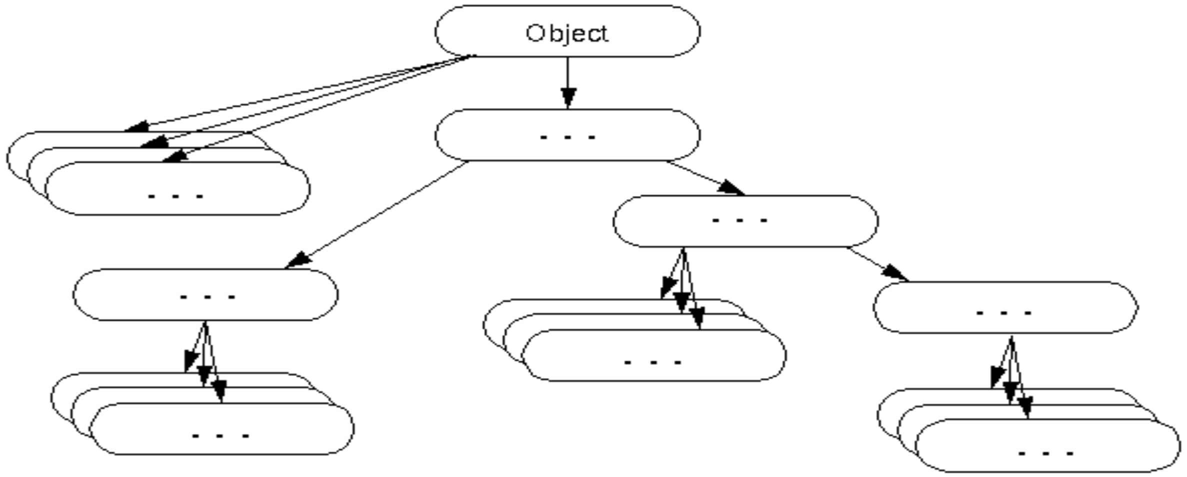
By providing only setter or getter method, you can make the class **read-only or write-only**.
It provides you the **control over the data**. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

example

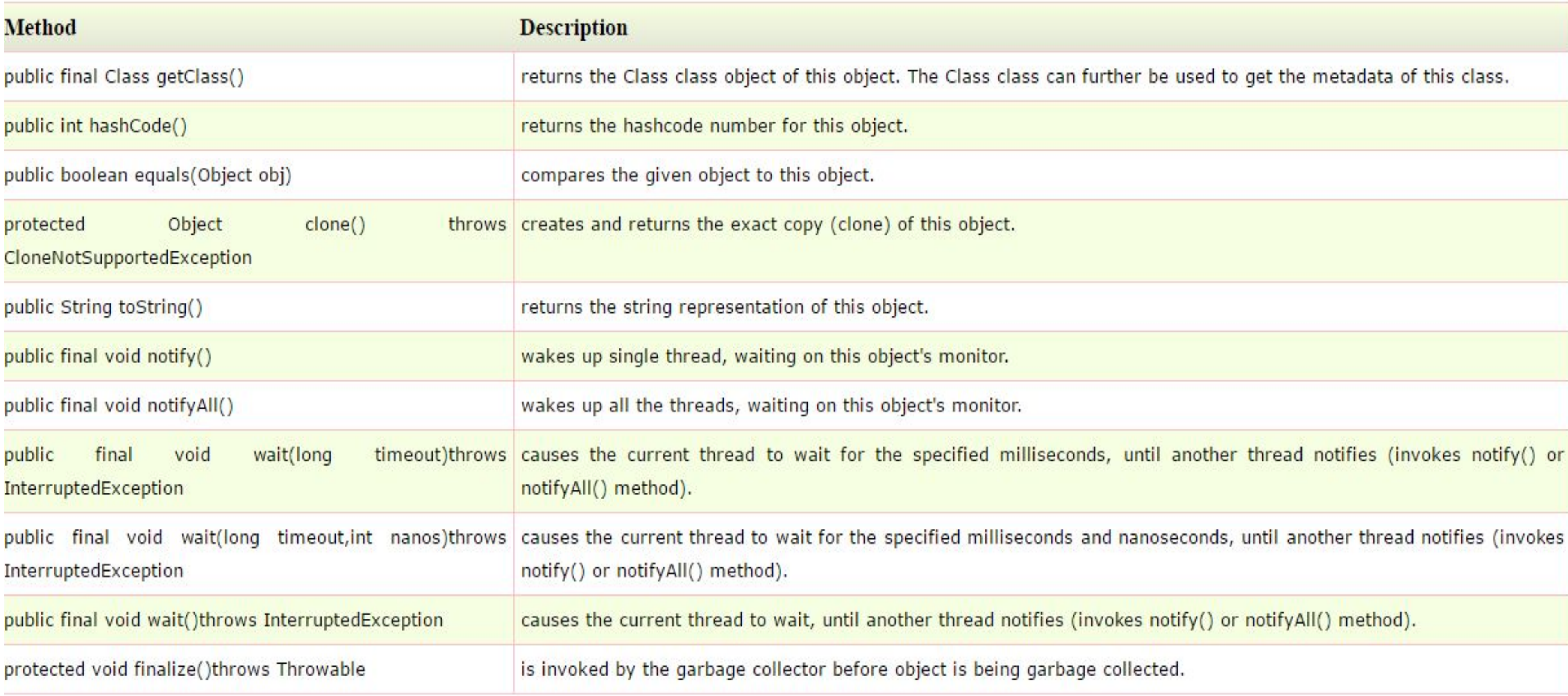
```
package com.java;
public class Student{
private String name;


public String getName(){
return name;
}
public void setName(String name){
this.name=name
}
}
```

Object class in Java

The **Object class** is the parent class of all the classes in java by default. 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. Notice that parent class reference variable can refer the child class object, know as upcasting.
The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

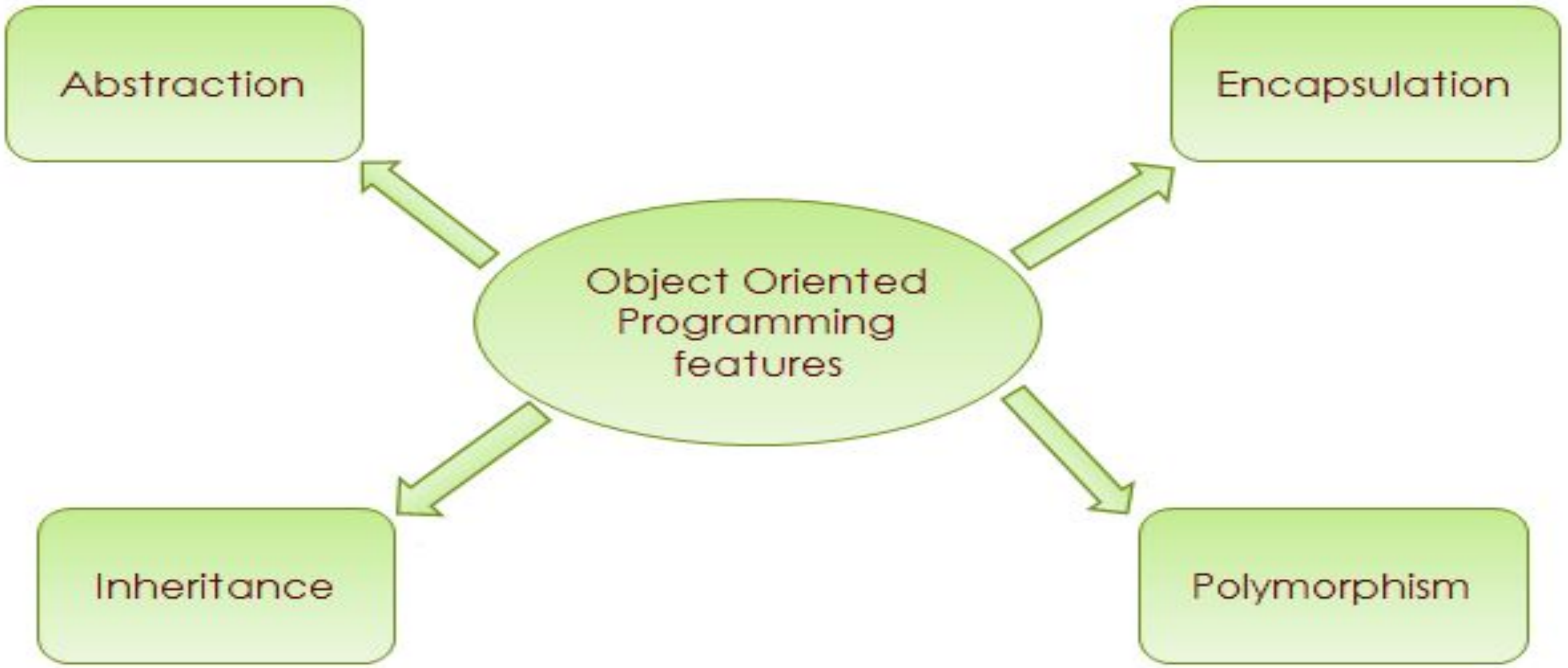


Methods of Object class



Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Review


Abstraction
Encapsulation
Object Oriented Programming features
Inheritance
Polymorphism

Home Work

1. Create an abstract class PayCalculator that has an attribute payRate given in dollars per hour. The class should also have a method computePay(hours) that returns the pay for a given amount of time.
2. Derive a class RegularPay from PayCalculator, as described in the previous exercise. It should have a constructor that has a parameter for the pay rate. It should not override any of the methods. Then derive a class HazardPay from PayCalculator that overrides the computePay method. The new method should return the amount returned by the base class method multiplied by 1.5.
3. Create an abstract class DiscountPolicy. It should have a single abstract method computeDiscount that will return the discount for the purchase of a given number of a single item. The method has two parameters, count and itemCost.
4. Derive a class BulkDiscount from DiscountPolicy, as described in the previous exercise. It should have a constructor that has two parameters, minimum and percent. It should define the method computeDiscount so that if the quantity purchased of an item is more than minimum, the discount is percent percent.

Continued

5. Create an interface MessageEncoder that has a single abstract method encode(plainText), where plainText is the message to be encoded. The method will return the encoded message.

6. Create a class SubstitutionCipher that implements the interface MessageEncoder, as described in the previous exercise. The constructor should have one parameter called shift. Define the method encode so that each letter is shifted by the value in shift. For example, if shift is 3, *a* will be replaced by *d*, *b* will be replaced by *e*, *c* will be replaced by *f*, and so on. *Hint*: You may wish to define a private method that shifts a single character.