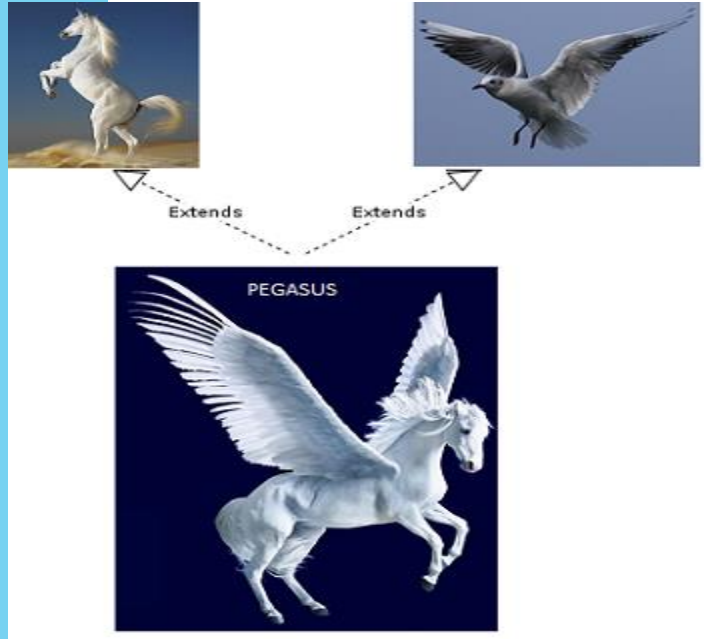
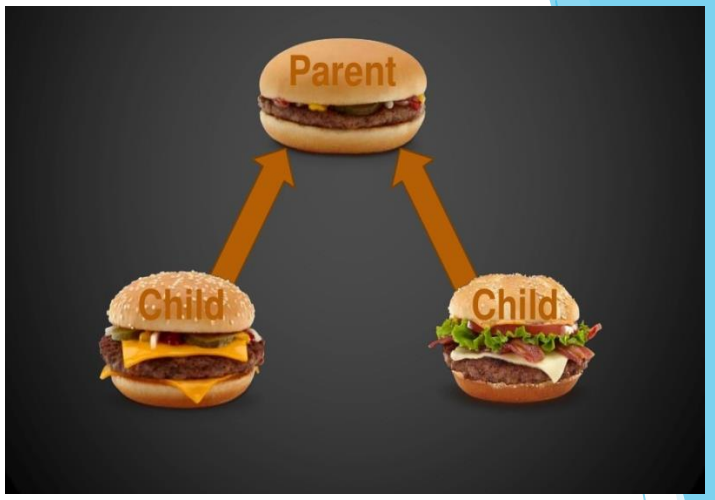


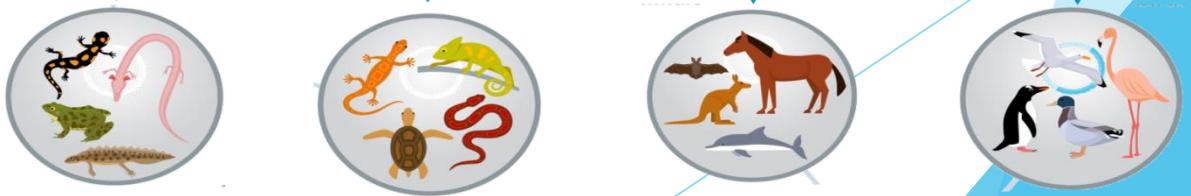


Inheritance

Super Class


Animals

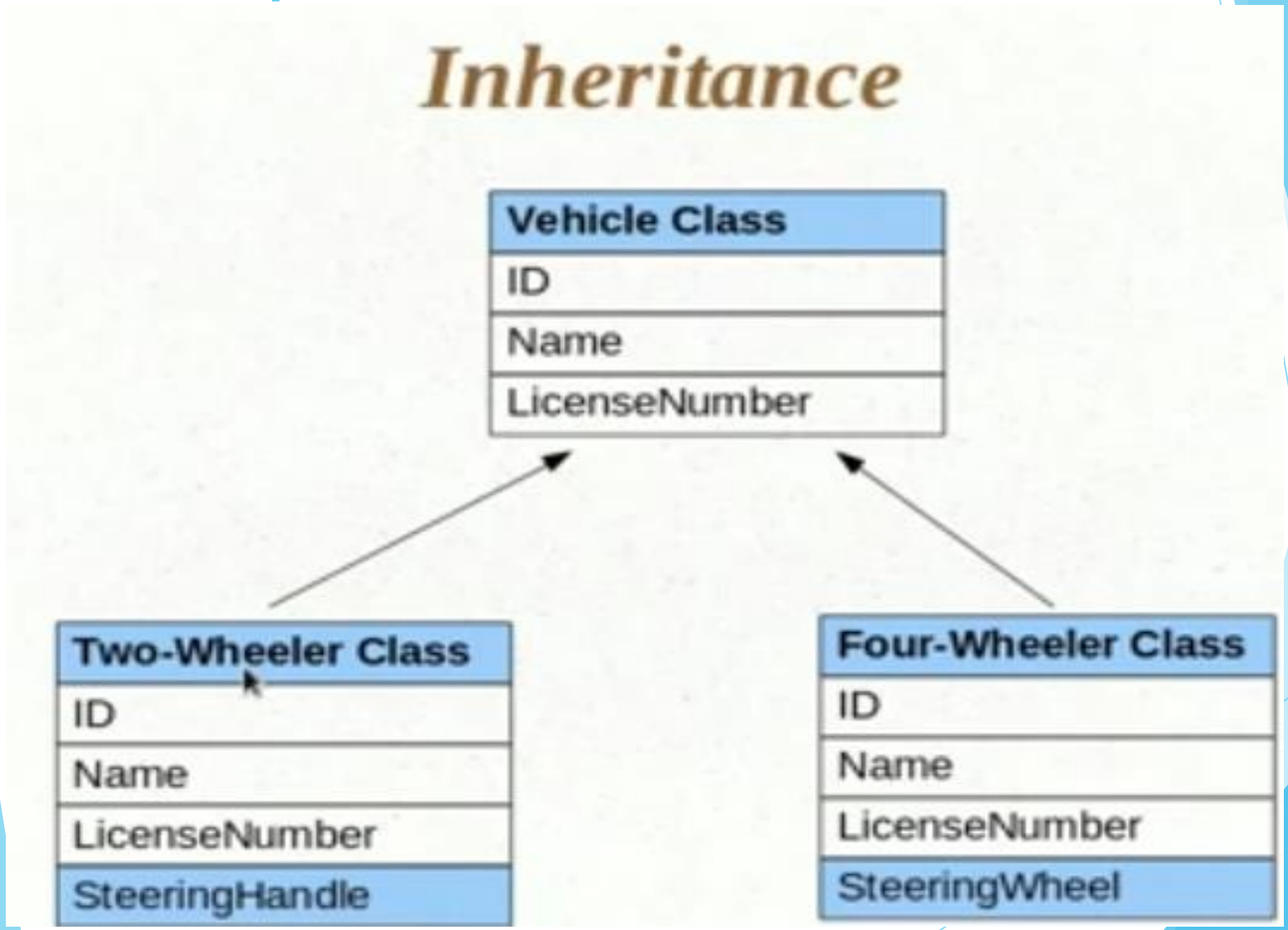
Child Class



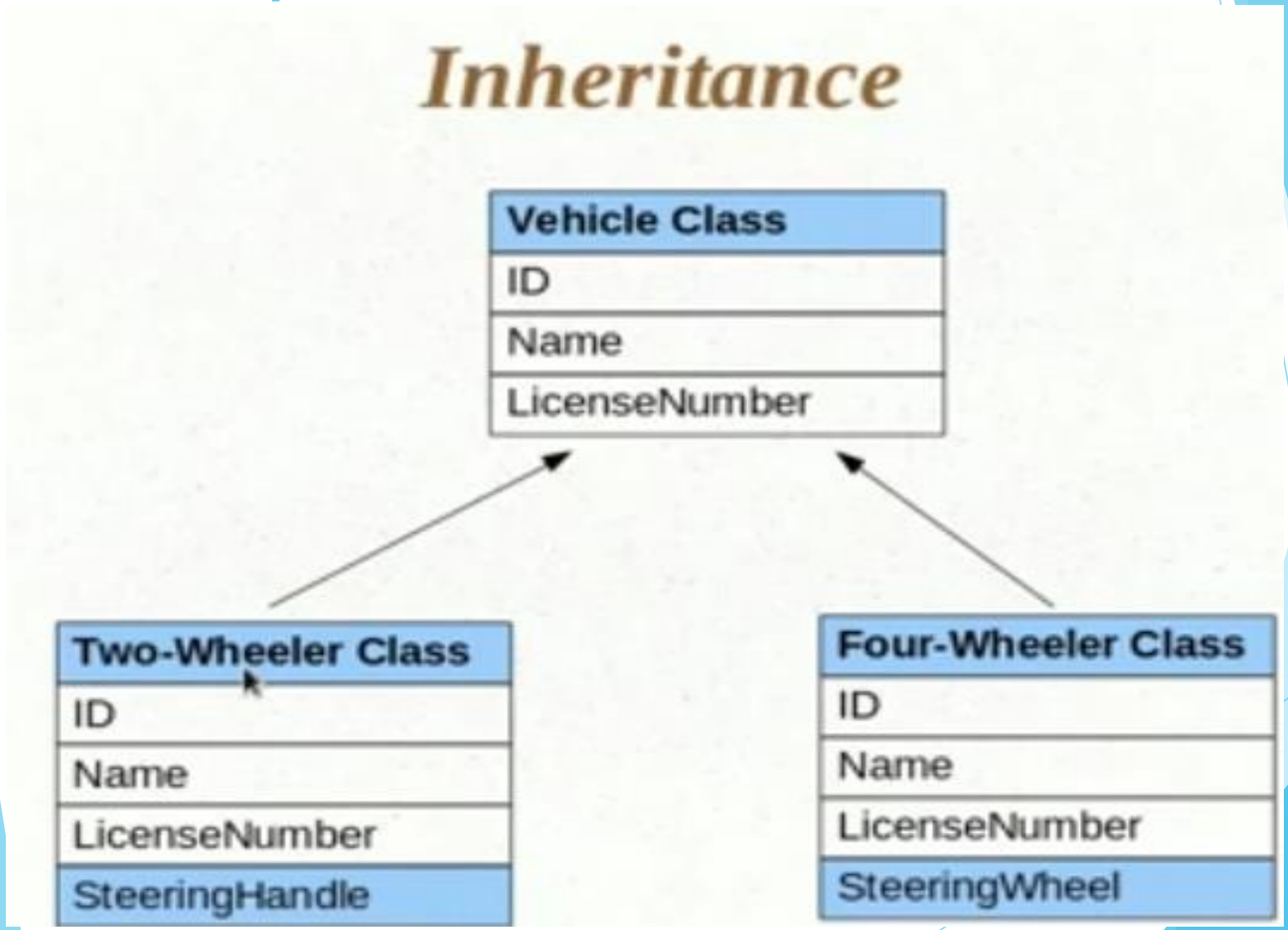
Introduction

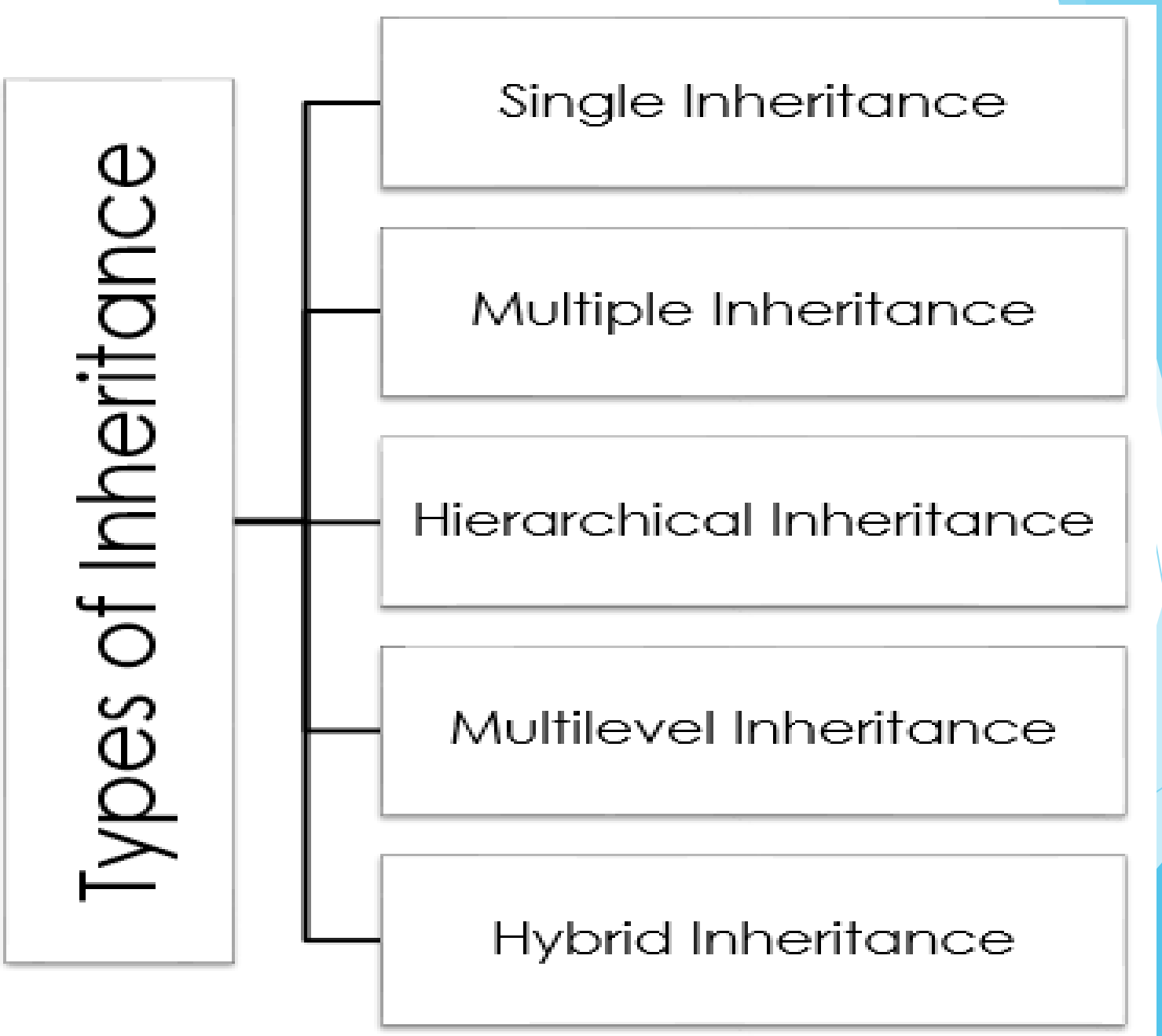
- The mechanism of deriving a new class from an old one is called **Inheritance**.
- In the inheritance the class(old class) which is give data members and methods is known as base or super or parent class.
- The class(new class) which is taking the data members and methods is known as sub or derived or child class.
- The concept of inheritance is also known as re-usability or extendable classes or sub classing or derivation.

Example


Inheritance
Vehicle Class
ID
Name
LicenseNumber
Two-Wheeler Class
ID
Name
LicenseNumber
SteeringHandle
Four-Wheeler Class
ID
Name
LicenseNumber
SteeringWheel

Example


Inheritance
Vehicle Class
ID
Name
LicenseNumber
Two-Wheeler Class
ID
Name
LicenseNumber
SteeringHandle
Four-Wheeler Class
ID
Name
LicenseNumber
SteeringWheel


Types of Inheritance
Single Inheritance
Multiple Inheritance
Hierarchical Inheritance
Multilevel Inheritance
Hybrid Inheritance

Single inheritance

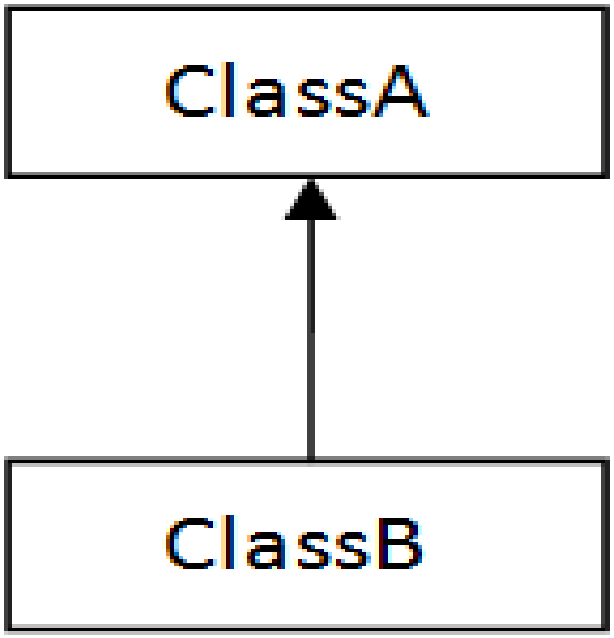
- In single inheritance there exists single base class and single derived class.
- Syntax:

class *subclassname* extends *superclassname*

{

classmembers;

}



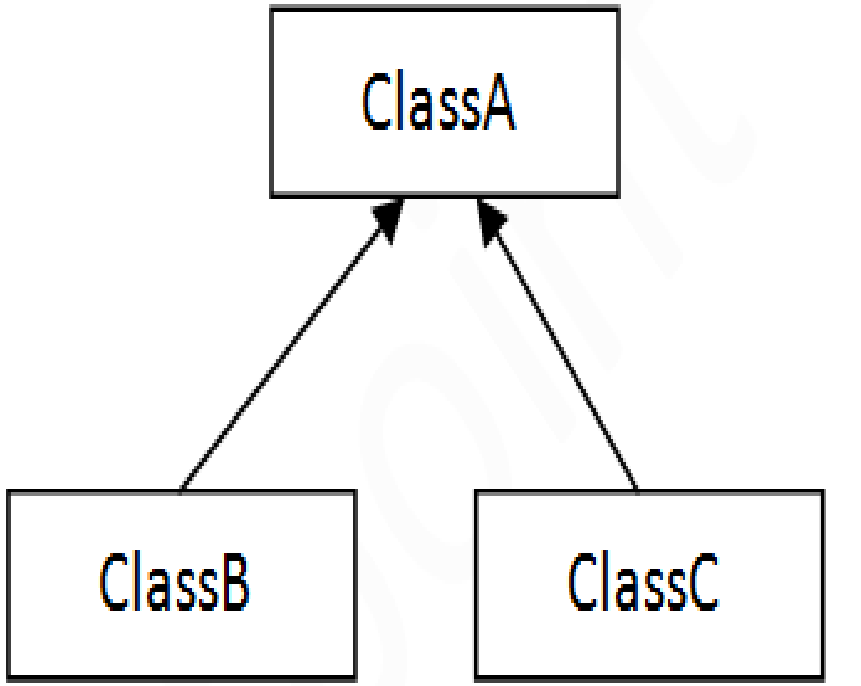
Example

```
class Employee
{
 float salary=40000;
}
class Programmer extends Employee
{
 int bonus=10000;
}
class EmployeeProgrammer
{
 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);
} }
```

Hierarchical inheritance

- When there is only one base class and more than one derived classes then it is called Hierarchical inheritance.
- Syntax :

```
class A
{… }
class B extends A
{ ….}
class C extends A
{…}
```



Example

```
class A
{
  public void methodA()
  {
     System.out.println("method of Class A");
  }
}
class B extends A
{
  public void methodB()
  {
     System.out.println("method of Class B");
  }
}
```

Contd…

Contd…

```
class C extends A
{
 public void methodC()
 {
     System.out.println("method of Class C");
 }
}
class D extends A
{
  public void methodD()
  {
      System.out.println("method of Class D");
  }
}
```

Contd…

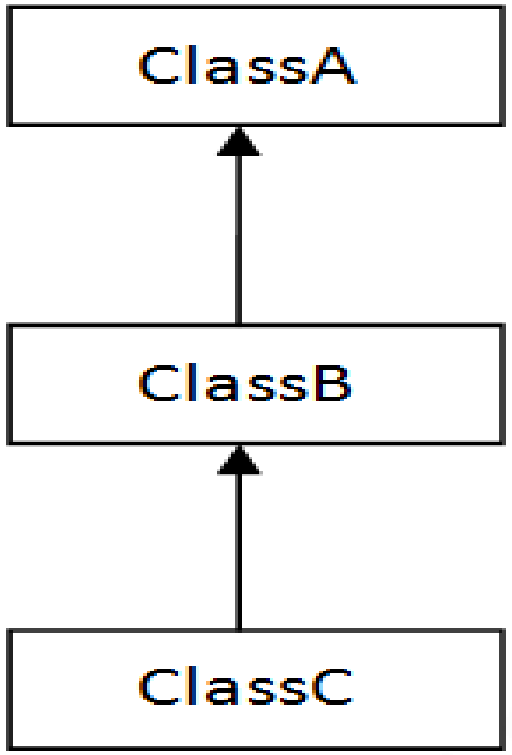
Contd…

```
class MyClass
{
public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     obj1.methodA();
     obj2.methodA();
     obj3.methodA();
  }
}
```

Multilevel inheritance

- Deriving a new class from a derived class is called multilevel inheritance.
- In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.
- Syntax :

```
class A
{… }
class B extends A
{ ….}
class C extends B
{…}
```



Example

```
class Car
{
    void vehicleType()
    {
        System.out.println("Vehicle Type: Car");

    }
}
 class Maruti extends Car
{
    void brand()
    {
        System.out.println("Brand: Maruti");
    }
}
```

Contd…

Contd...

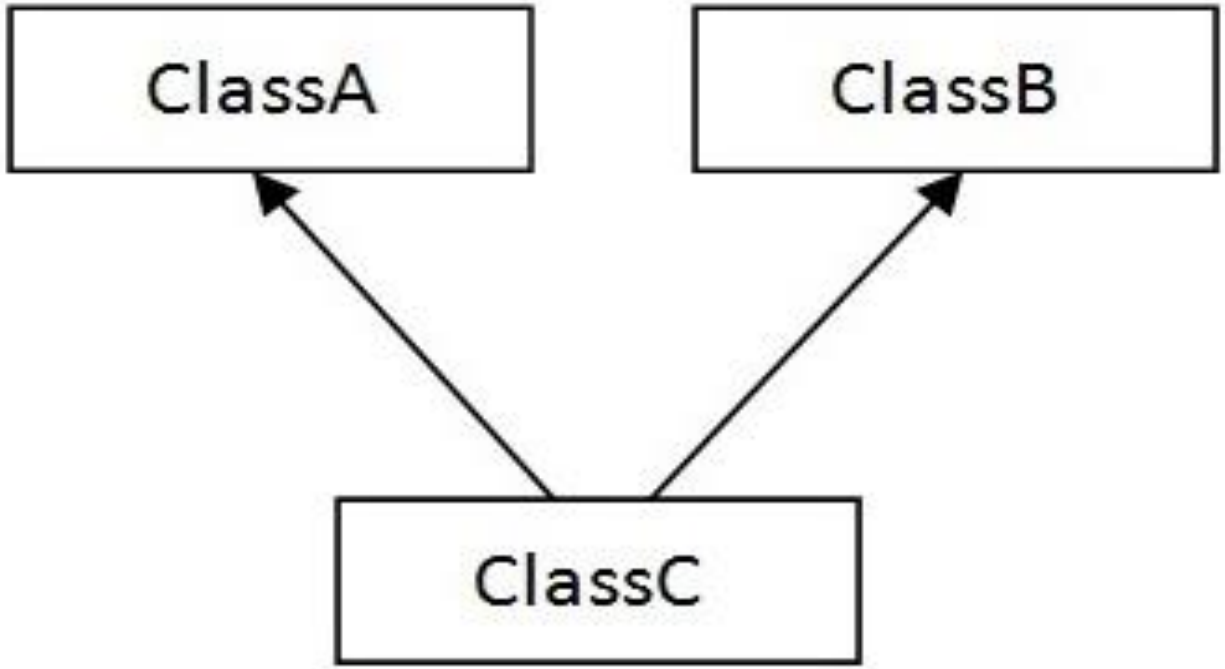
```
class Maruti800 extends Maruti
{
    void model()
    {
        System.out.println("Maruti Model: 800");

    }
    void speed()
    {
        System.out.println("Max: 80Kmph");
    }
```

```
public static void main(String args[])
    {
        Maruti800 obj=new Maruti800();
        obj.vehicleType();
        obj.brand();
       obj.model();
        obj.speed();
    }
}
```

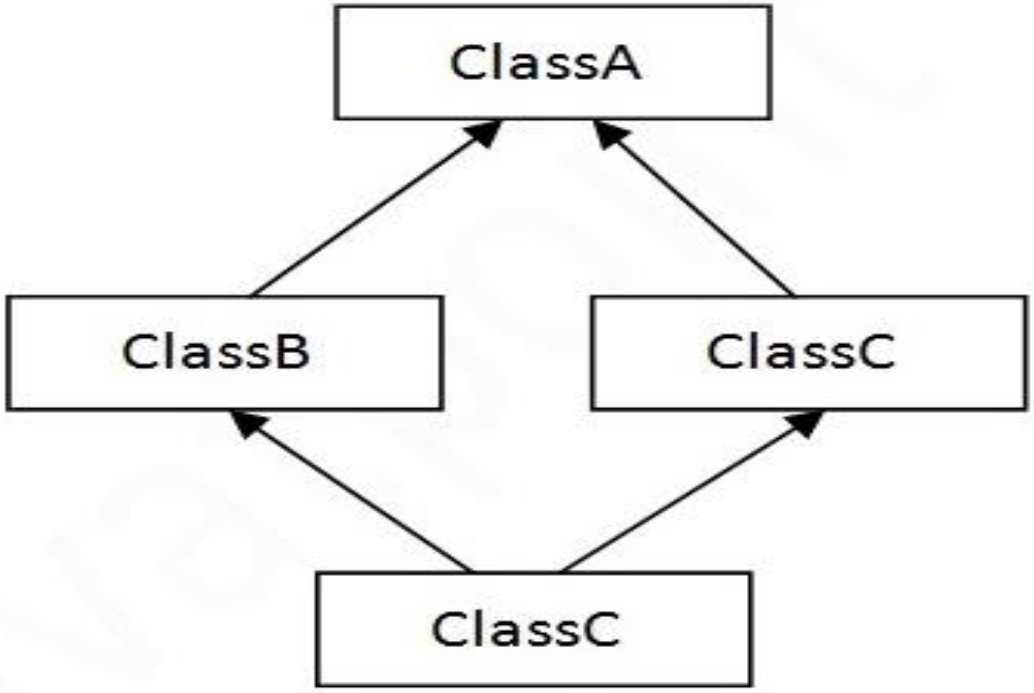
Multiple inheritance

- In multiple inheritance there exist multiple base classes and single derived class.
- The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.



Hybrid inheritance

- If there is combination of any two type of inheritance then it is called as Hybrid Inheritance.
- In the combination, if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.



Super keyword

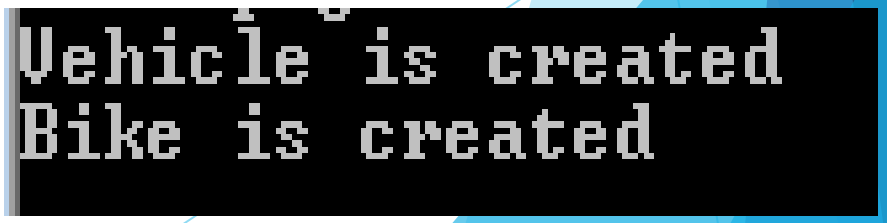
- The **super** keyword in java is a reference variable that is used to refer immediate parent class object.
- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.

Example of super that is used to refer immediate parent class instance variable.

```
class Vehicle
{
  int speed=50;
}
class Bike4 extends Vehicle
{
  int speed=100;
  void display()
    {     System.out.println(super.speed);
     } //will print speed of Vehicle now
  public static void main(String args[])
{
   Bike4 b=new Bike4();
   b.display();
}
}
```

Example where super is used to invoke parent class constructor.

```
class Vehicle
{
  Vehicle()
  {    System.out.println("Vehicle is created");    }
  }
class Bike extends Vehicle
{
    Bike()
    {
        super();          //will invoke parent class constructor
        System.out.println("Bike is created");
     }
     public static void main(String args[])
    {
          Bike b=new Bike();
     }
}
```



Example where super can be used to invoke parent class method

```
class Person
{
    void message()
   {       System.out.println("welcome");   }
   }
class Student extends Person
{
    void message()
    {    System.out.println("welcome to java");
    }
    void display()
    {         message();        //will invoke current class message() method
              super.message();    //will invoke parent class message() method
    }
    public static void main(String args[])
    {
        Student s=new Student();
        s.display();
     }
}
```

```
welcome to java
welcome
```

Example

- In a garden, trees are maintained. A tree has following set of attributes:-Tree code, heights, base and amount spent on the tree so far.
- Define Tree class, its constructor, update() that updates tree information and display()
- Define derive class Mango tree that has additional yield attribute. Define Garden class and display information of a tree and a Mango Tree.(use super keyword)

solution

```
import java.util.Scanner;
class Tree
{
  Scanner sc=new Scanner(System.in);
  int Tree_code,height,base;
  float amount_spent;
  Tree(int tc,int h,int b,float am)
  {
     Tree_code=tc;
      height=h;
      base=b;
      amount_spent=am;
   }
```

contd...

Contd…

```
  void update()
  {
    System.out.println("update tree code, height, base and amount spent:");
    Tree_code=sc.nextInt();
    height=sc.nextInt();
     base=sc.nextInt();
     amount_spent=sc.nextFloat();
  }
 void display()
  {
   System.out.println("Tree code :"+Tree_code);
   System.out.println("Tree height :"+height);
   System.out.println("Tree base :"+base);
   System.out.println("Amount spent on tree :"+amount_spent);
   }
}
```

contd…

Contd…

```
class Mango extends Tree
{
 int yield;
 Mango(int tc,int h,int b,float am,int y)
 {
  super(tc,h,b,am);
  yield=y;
 }
 void update()
  {
   super.update();
   System.out.println("update yeild:");
   yield=sc.nextInt();
  }
```

contd…

contd...

```
    void display()
    {
     System.out.println("Tree code :"+Tree_code);
     System.out.println("Tree height :"+height);
     System.out.println("Tree base :"+base);
     System.out.println("Amount spent on tree :"+amount_spent);
     System.out.println("Yield :"+yield);
     }
 }
class Garden
{
  public static void main(String args[])
  {
    Mango m=new Mango(11,200,50,4000,50000);
    m.display();
    m.update();
    m.display();
   }
}
```

Question

Write the code to implement the concept of inheritance for Vehicles.

- You are required to implement inheritance between classes.
- You have to write four classes in java i.e. one superclass, two sub classes and one class.
- Vehicle is the super class whereas Bus and Truck are sub classes of Vehicle class.
- Transport is a driver class which contains main method.

Solution

```
class Vehical
{
        void type()
       {     System.out.println("Vehicle Type: Car");       }
        void speed()
       {     System.out.println("Max: 80Kmph");    }
}
class Bus extends Vehical
{
       void type()
       {      System.out.println("Vehicle Type: Bus");    }
        void speed()
       {     System.out.println("Max: 80Kmph");          }
}
```

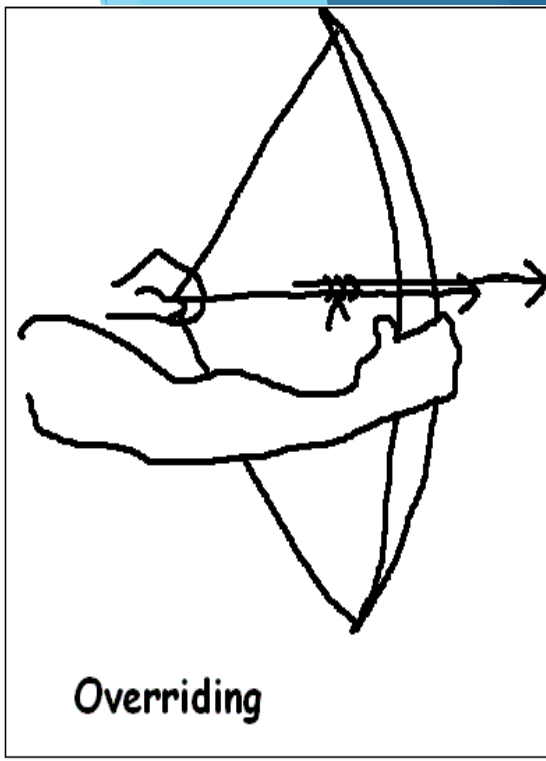
contd...

Contd…

```
class Truck extends Vehical
{
        void type()
        {     System.out.println("Vehicle Type: Truck");     }
        void speed()
       {      System.out.println("Max: 90Kmph");       }
}
public class Transport
{
   public static void main(String args[])
        {
              Bus b=new Bus();
                b.type();
                b.speed();
              Truck t=new Truck();
                t.type();
                t.speed();
        }
}
```

Method Overriding

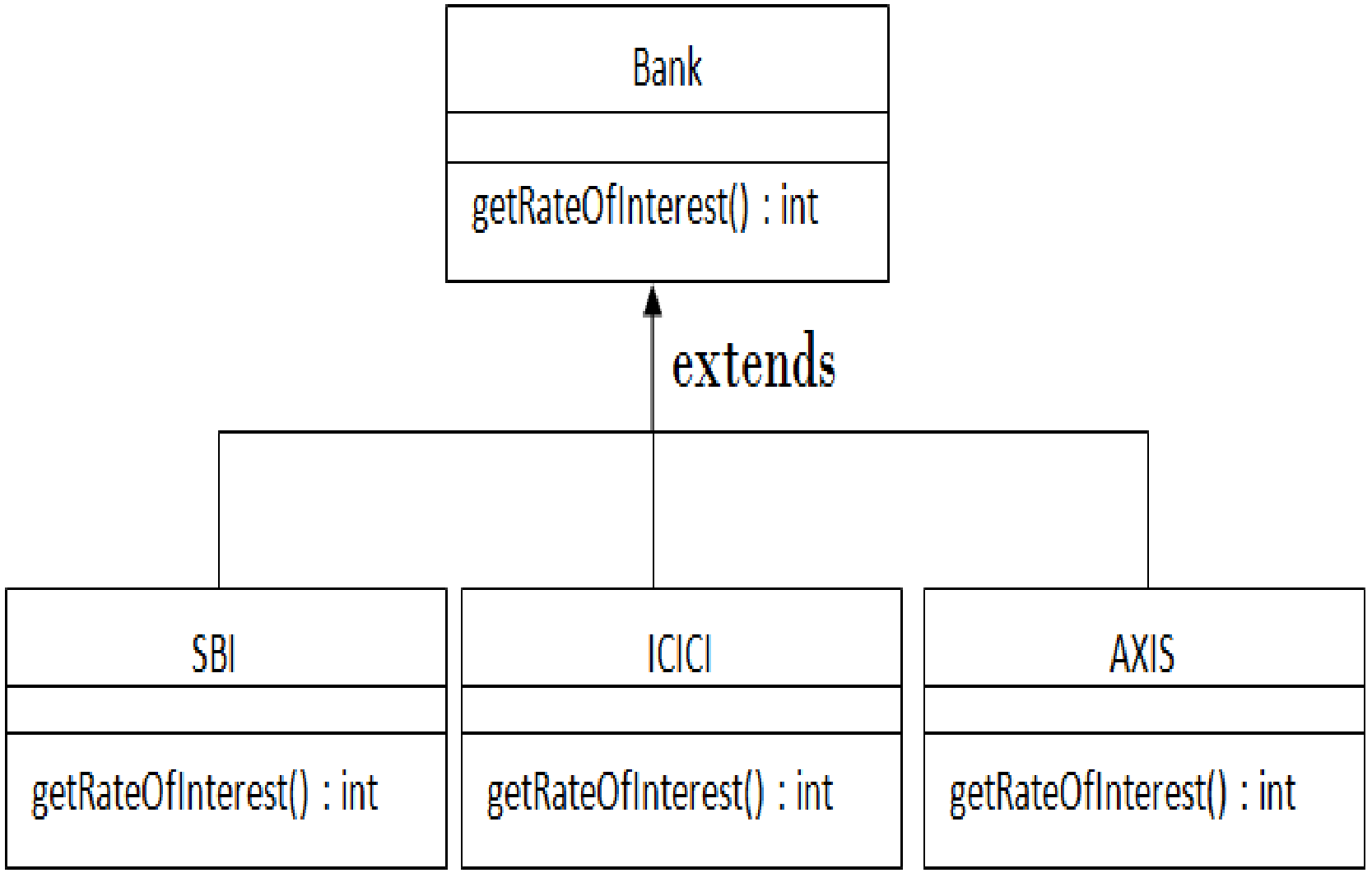
- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.
- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism.



Rules for Java Method Overriding

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

Example


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

example

```
class Bank
{
    int getRateOfInterest()
   {  return 0;  }
}
class SBI extends Bank
{
     int getRateOfInterest()
     {    return 8;  }
}
class ICICI extends Bank
{
     int getRateOfInterest()
    {    return 7;  }
}
```

Contd...

Contd…

```
class AXIS extends Bank
{
   int getRateOfInterest()
   {   return 9;  }
}
class TestOverride
{
     public static void main(String args[])
    {
       SBI s=new SBI();
       ICICI i=new ICICI();
       AXIS a=new AXIS();
      System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
      System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
      System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
   }
}
```

Method Overriding in dynamic method dispatch

- Dynamic method dispatch is a technique which enables us to assign the base class reference to a child class object.

Question

WAP to create a parent class Person and child classes teacher and Students. Write a method work() in parent class and override it in child classes. Call the methods in driver class using dynamic method dispatch.

Solution

```
class Person
{
  void work()
  {    System.out.println("working");  }
}
class Teacher extends Person
{
  void work()
  {    System.out.println("Teacher teaches students");  }
}
class Student extends Person
{
  void work()
  {    System.out.println("Student studies");    }
}
```

contd...

Contd…

```
class DynMethodDispatch
{
  public static void main(String args[])
  {
    Person t=new Teacher();
    t.work();
    Person s=new Student();
    s.work();
   }
}
```

this Keyword

- In java, ***this*** is a reference variable that refers to the current object.
- ***this*** keyword can be used to refer current class instance variable.
- ***this()*** can be used to invoke current class constructor.
- ***this*** keyword can be used to invoke current class method (implicitly)
- ***this*** can be passed as an argument in the method call.
- ***this*** can be passed as argument in the constructor call.
- ***this*** keyword can also be used to return the current class instance.

Example

```
class Student
{
    int id;
    String name;
    Student(int id, String name)
    {
    this.id = id;
    this.name = name;
    }
    void display()
    {   System.out.println(id+""+name);    }
    public static void main(String args[])
   {
      Student s1 = new Student(111,"Karan");
      Student s2 = new Student(222,"Aryan");
      s1.display();
      s2.display();
    }
}
```

Final variable

- final variables are nothing but constants.
- We cannot change the value of a final variable once it is initialized.
- Final variables behave like class variables and they do not take any space on individual objects of the class.
- E.g.

```
final int SIZE=100;
```

Final Method

- A final method cannot be overridden.
- Which means even though a sub class can call the final method of parent class without any issues but it cannot override it.
- E.g.

```
final void showstatus()
{……}
```

Example

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

Output: Compile Time Error

Final class

- If you make any class as final, you cannot extend it.
- Declaring a class final prevents any unwanted extension to the class.
- E.g.

```
final class A {……}

final class B extends D {…..}
```

Example

```
final class Bike
{    }
class Honda1 extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}
public static void main(String args[])
{
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error

Abstract Methods

- A method that is declared as abstract and does not have implementation is known as abstract method.
- E.g. abstract void printStatus(); //no body and abstract

Abstract Classes

- A class that is declared as abstract is known as abstract class.
- It needs to be extended and its method implemented.
- It cannot be instantiated.(i.e. Object of abstract class can not be created)
- When a class contains one or more abstract methods, it should also declared abstract.
- e.g. abstract class shape

  {…..}

Example

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

Visibility Control

- It may be necessary in some situations to restrict the access to certain variables , methods and classes from outside the class.
- This is achieved in java by applying visibility modifiers to the instance variables and methods.
- There are three visibility modifier in java:
  - Public
  - Private
  - Protected

Levels of protection provided by visibility modifiers

- Public Access
- Friendly Access
- Protected Access
- Private Access
- Private Protected Access

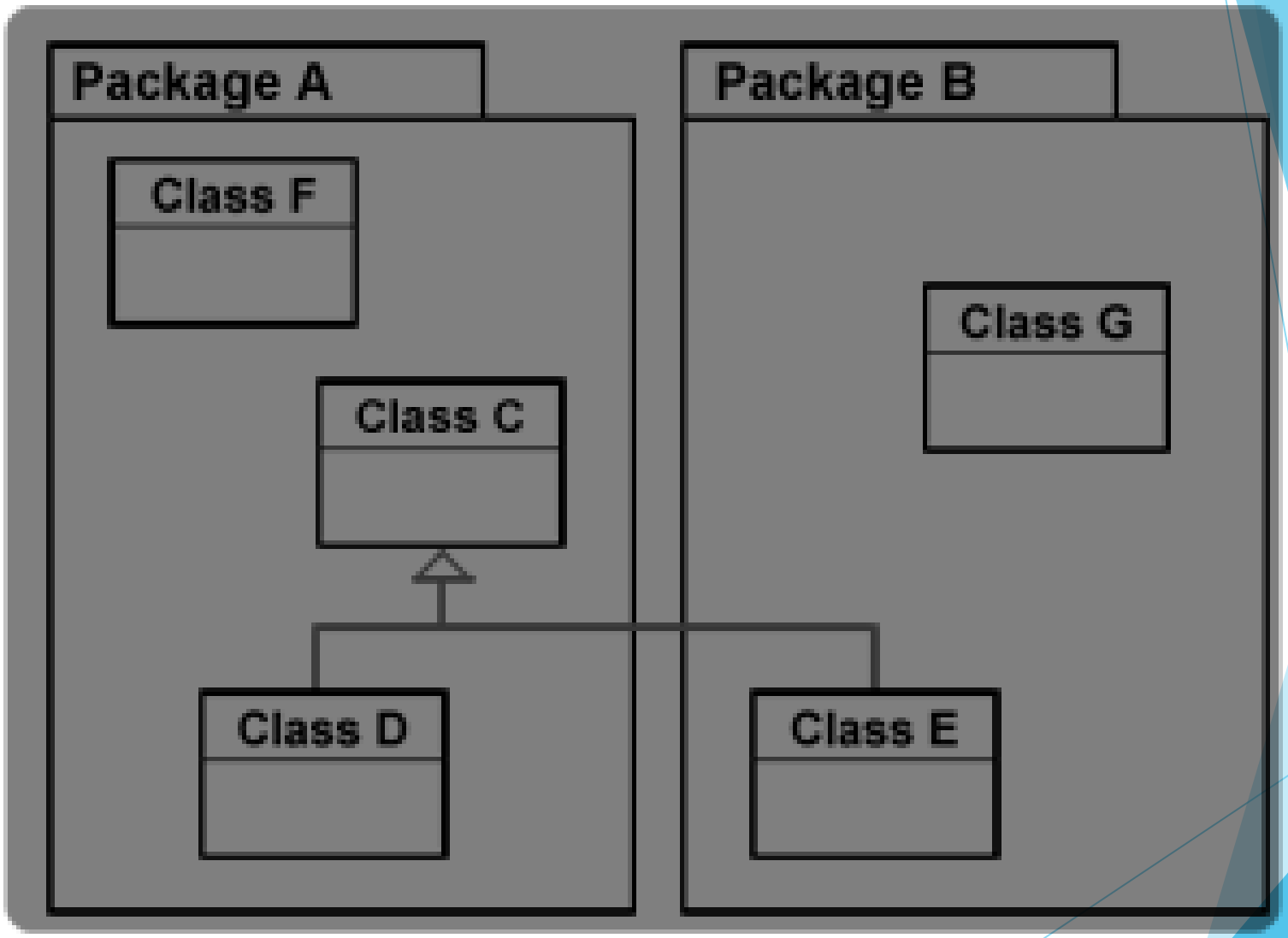
Public Access

- 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.

Public



Note: Darken area is visibility.

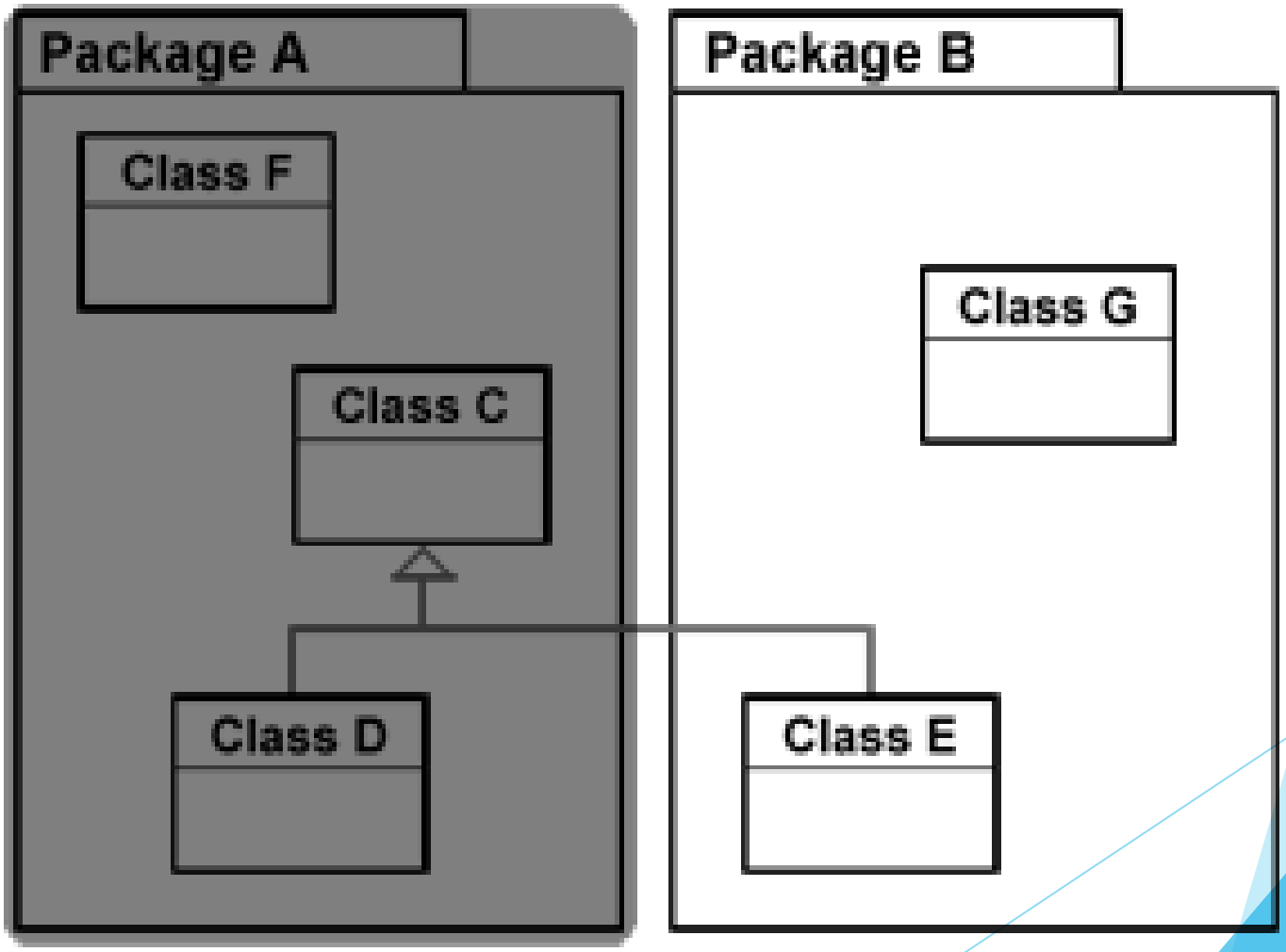
Friendly Access

- When no access modifier is specified, the member defaults to a limited version of public accessibility known as friendly level of access.

- Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

- The difference between the public access and the friendly access is that the public modifier makes fields visible in all classes, regardless of their packages while the **friendly access makes fields visible only in the same package**.

Friendly / Default



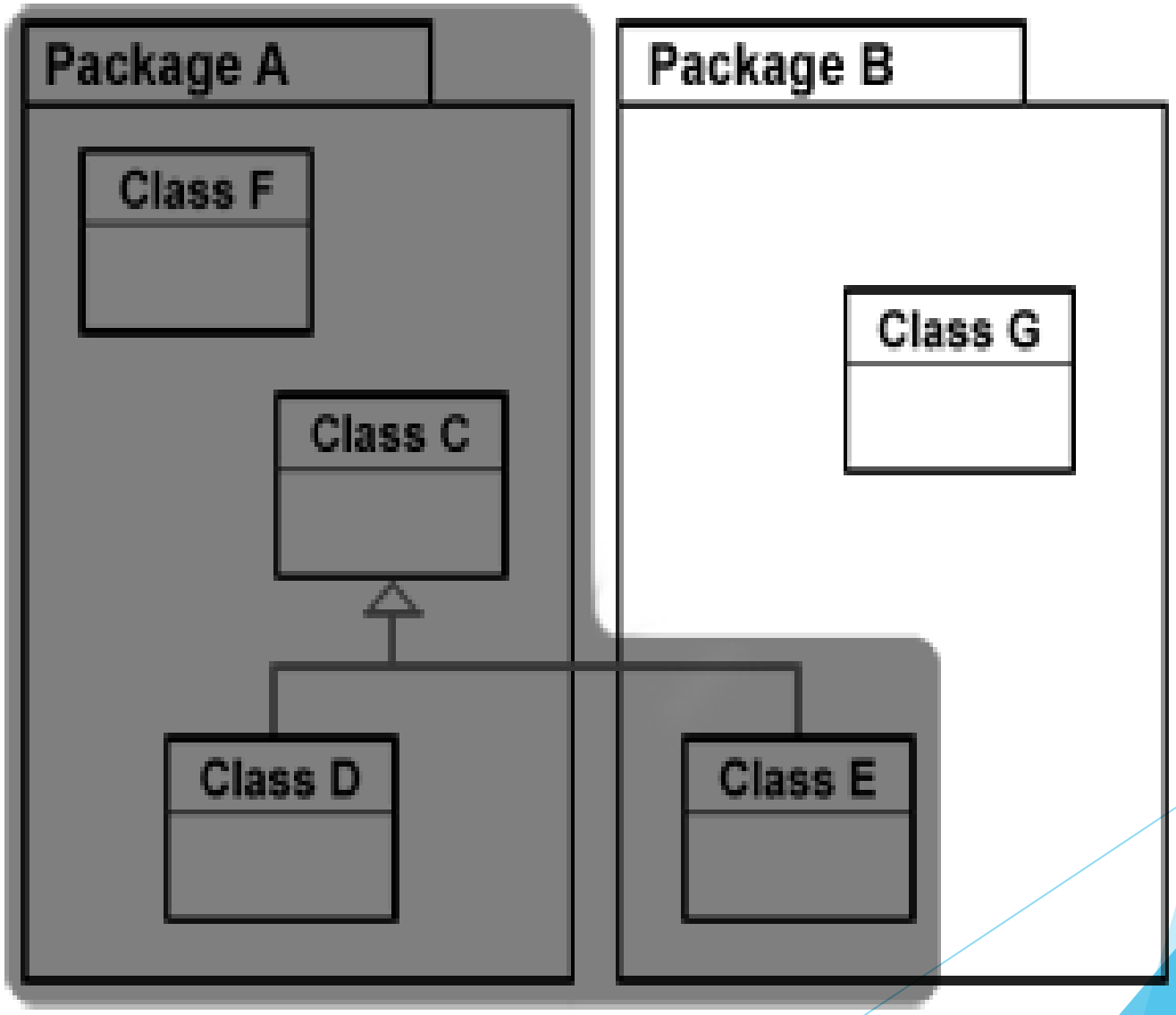
Protected Access

- 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.

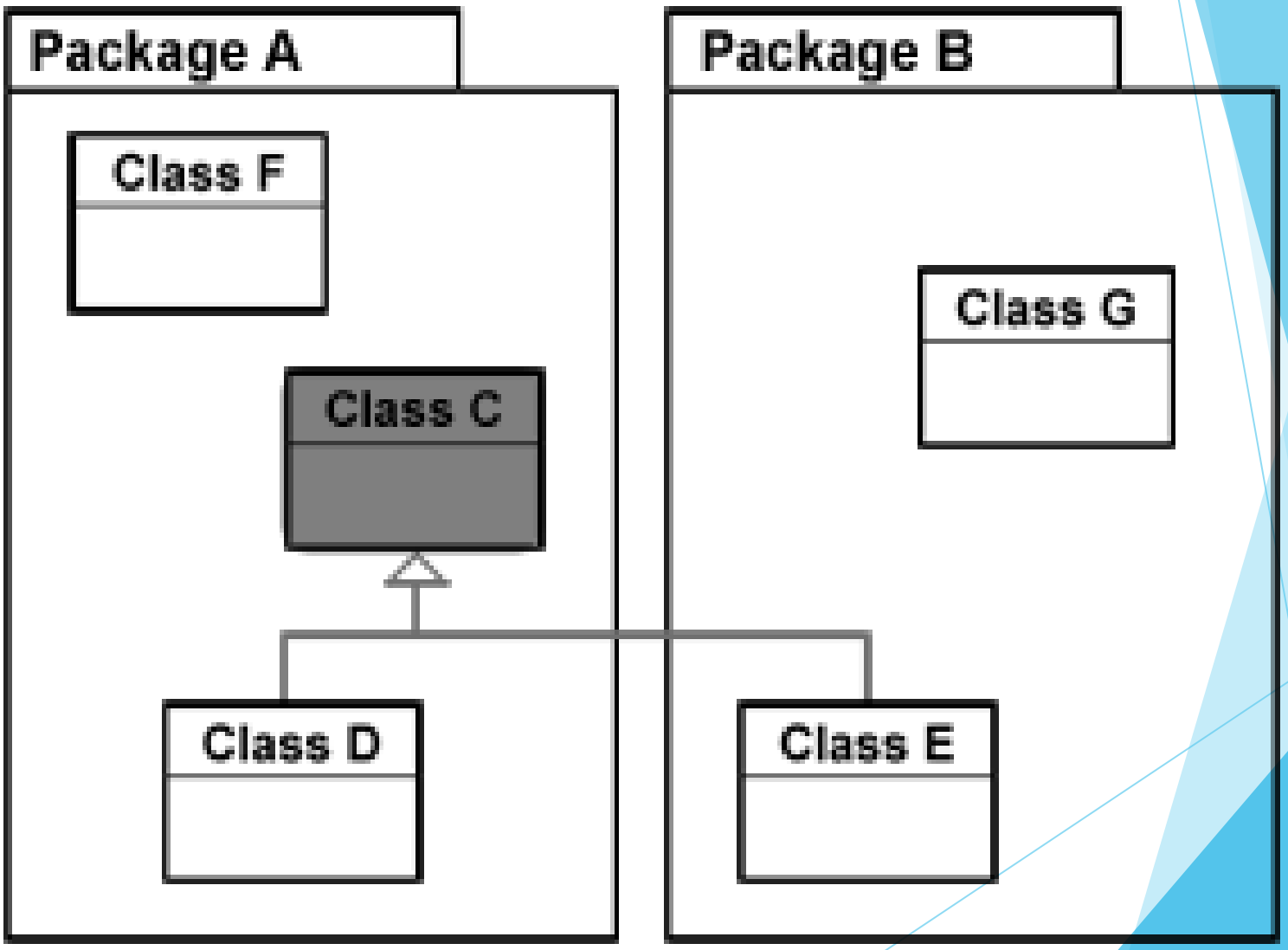
Protected



Private Access

- Members 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.
- Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.

Private

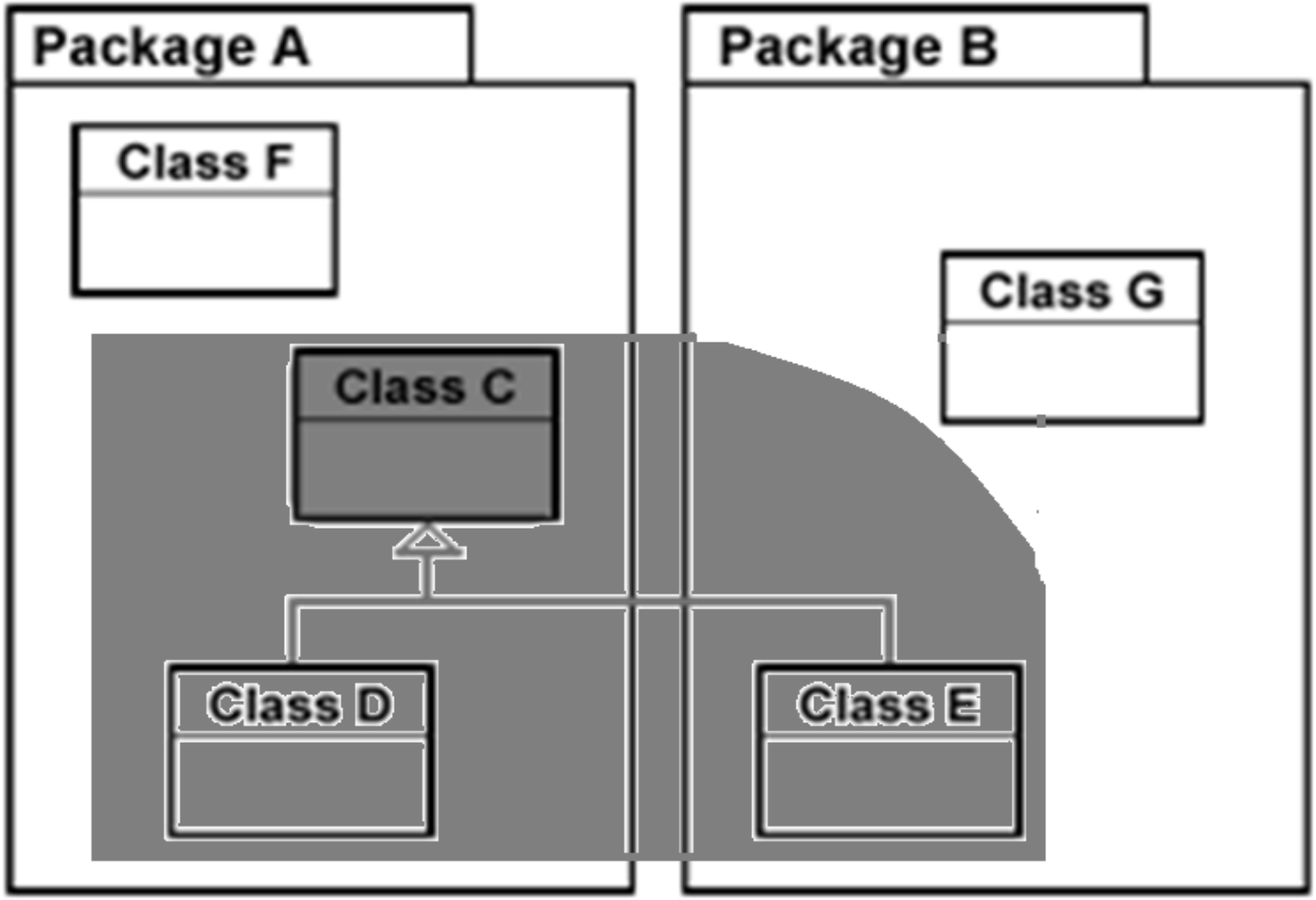


Private Protected Access

- A field can be declared with two keyword private and protected together.
- This gives visibility level in between the protected access and private access.
- This modifier makes the fields visible in all subclasses regardless of what package they are in.
- These fields are not accessible by other classes in the package.
- E.g.

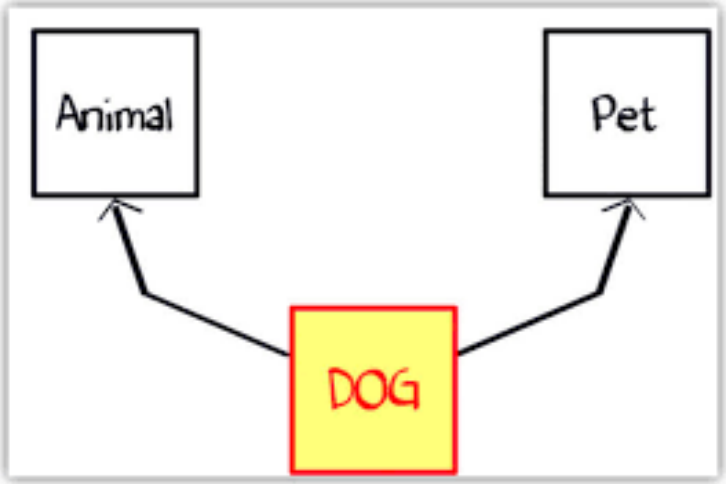
```
private protected int codeNumber;
```

Private protected

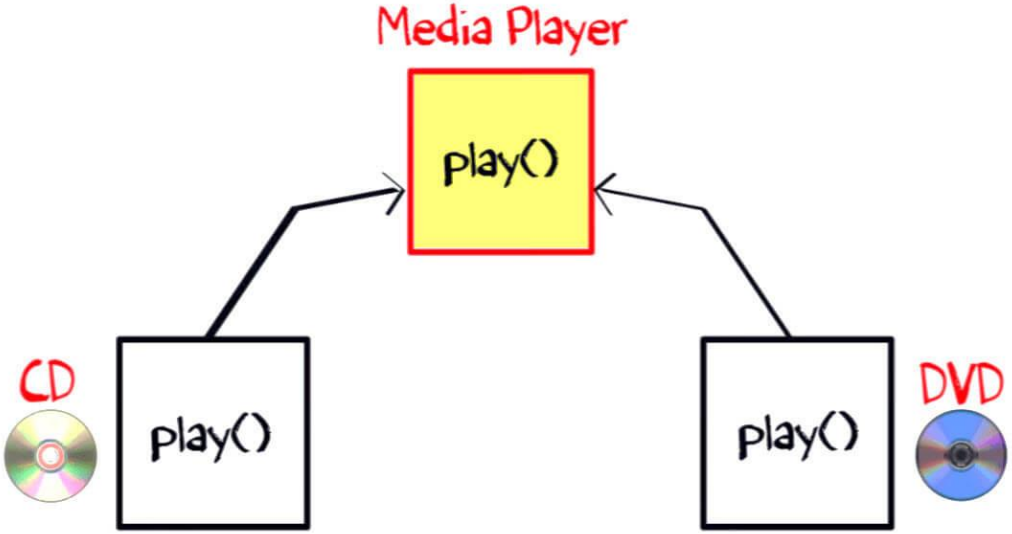


Visibility of field in a class

	public	protected	default	private protected	private
Same class	Yes	Yes	Yes	Yes	Yes
Same package sub class	Yes	Yes	Yes	Yes	No
Same package Non-sub class	Yes	Yes	Yes	No	No
Different package sub class	Yes	Yes	No	Yes	No
Different package Non-sub class	Yes	No	No	No	No



Interface



Introduction

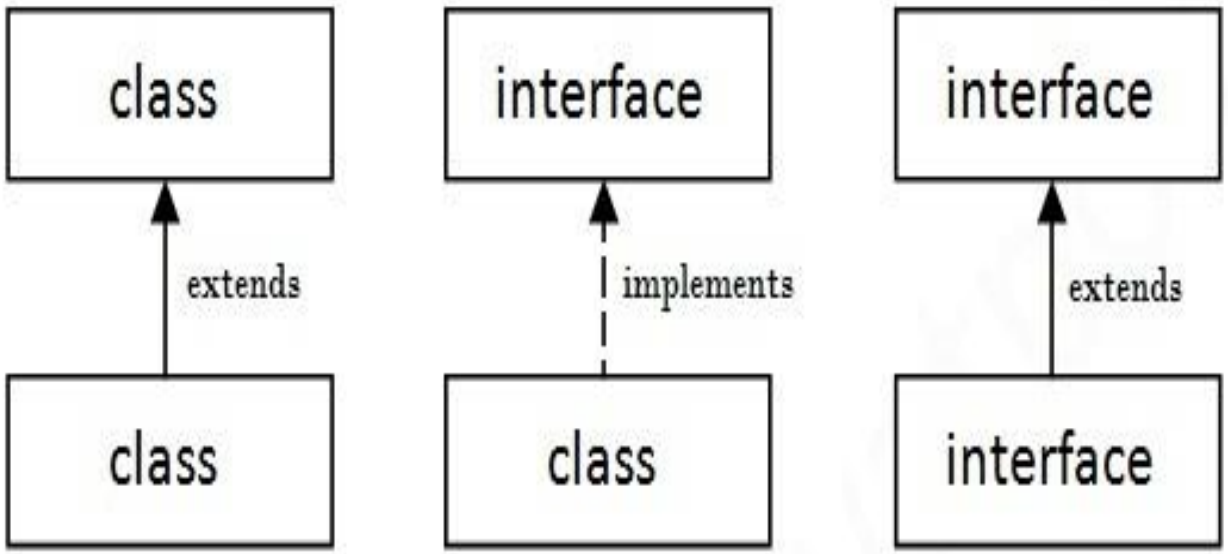
- The interface in java is **a mechanism to achieve fully abstraction**.
- There can be only abstract methods in the java interface not method body.
- It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.
- Interface fields are public, static and final by default, and methods are public and abstract.

Difference between abstract class

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();
	}

Relationship between classes and interfaces



Defining Interfaces

Syntax :

```
interface NameOfInterface
{
    //Any number of final, static fields
    //Any number of abstract method declarations
}
```

e.g.

```
interface Item
{
    static final int code=1001;
    void display();
}
```

Extending Interfaces

Syntax :

interface *interfacename2* extends *interfacename1*

{ body of interfacename2 }

Example :

```
interface ItemConstants
{
    int code=1001;
}
interface Item extends ItemConstants
{
    void display();
}
```

Implementing Interfaces

Syntax :

class *classname* implements *interfacename*

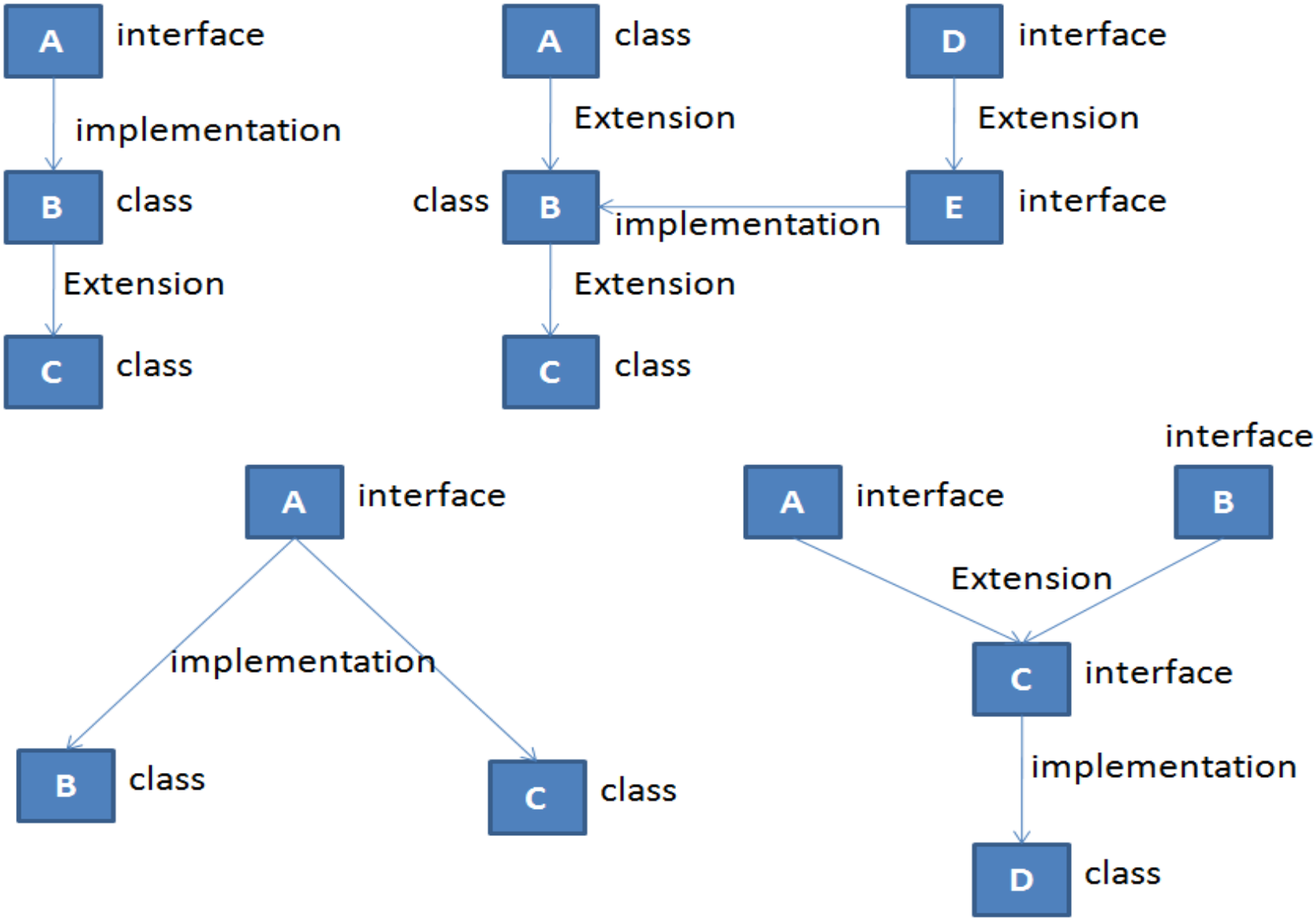
{ body of classname }

A class can extend another class while implementing interfaces.

class *classname* extends superclass implements *interface1,interface2….*

{ body of classname }

various forms of interface implementation



Example 1

```
interface MyInterface
{
   public void method1();
}
class XYZ implements MyInterface
{
      public void method1()
     {
           System.out.println("implementation of method1");
      }
     public static void main(String arg[])
     {
           MyInterface obj = new XYZ();
            obj. method1();
      }
}
```

Question

- Write a program to implement Animal using class Mammal.
- class Mammal implements eat and travel methods of interface Animal.

Solution

```
interface Animal
{
   public void eat();
   public void travel();
}
class Mammal implements Animal
{
     public void eat()
    {              System.out.println("Mammal eats");          }
    public void travel()
    {              System.out.println("Mammal travels");          }
      public static void main(String args[])
      {
            Mammal m = new Mammal();
            m.eat();
            m.travel();
        }
}
```

Multiple inheritance with interfaces

```
interface Exam
{     void percent_cal();        }
class Student
{
      String name="Ramesh";
      int roll_no=01;
       int mark1=93;
       int mark2=84;
}
class Result extends Student implements Exam
{
      public void percent_cal()
      {
            int total=(mark1+mark2);
            float percent=total*100/200;
            System.out.println ("Percentage: "+percent+"%");
      }
```

contd...

Contd…

```
void display()
    {
        System.out.println ("Name of Student: "+name);
        System.out.println ("Roll No. of Student: "+roll_no);
        System.out.println ("Marks of Subject 1: "+mark1);
        System.out.println ("Marks of Subject 2: "+mark2);
    }
}
class MultipleInheritance
{
    public static void main(String args[])
    {
        Result R=new Result();
        R.display();
        R.percent_cal();
    }
}
```

Question

- Animal implements Movable and Crawlable.
- Movable has method moveFast and Crawlable has Crawl.
- write a java program to implement the interface.

Solution

```
interface Moveable
{     public void moveFast();      }
interface Crawlable
{     public void crawl();     }
class Animal implements Moveable, Crawlable
{
    public void moveFast()
   {    System.out.println("I am moving fast, buddy !!");    }
    public void crawl()
    {       System.out.println("I am crawling !!");    }
    public static void main(String[] args)
    {
        Animal self = new Animal();
       self.moveFast();
       self.crawl();
    }
}
```

Hybrid inheritance using interface

```
interface A
{
    public void methodA();
}
interface B extends A
{
    public void methodB();
}
interface C extends A
{
    public void methodC();
}
```

contd…

Contd...

```
class D implements B, C
{
    public void methodA()
    {       System.out.println("MethodA");    }
    public void methodB()
    {       System.out.println("MethodB");    }
    public void methodC()
    {       System.out.println("MethodC");    }
    public static void main(String args[])
    {
        D obj1= new D();
        obj1.methodA();
        obj1.methodB();
        obj1.methodC();
     }
}
```

Hybrid inheritance using interface and classes

```
interface A
{
    public void methodA();
}
class B implements A
{
    public void methodB()
    {
        System.out.println("MethodB");
    }
    public void methodA()
    {
        System.out.println("MethodA");
    }
}
```

contd...

Contd…

```
interface C extends A
{
    public void methodC();
}
class D extends B implements C
{
   public void methodC()
   {       System.out.println("MethodC");   }
   public static void main(String args[])
   {
       D obj1= new D();
       obj1.methodA();
       obj1.methodB();
       obj1.methodC();
    }
}
```

Question

Write a java program to find area of square, rectangle and circle using interface

Solution

```
interface  Area
{
double pi=3.14;
double compute(double x,double y);
}
class Rect implements Area
{
        public double compute(double x,double y)
      {          return x*y;              }
}
class Tri implements Area
{
        public double compute(double x,double y)
     {          return x*y/2;          }
}
```

Contd…

Contd…

```
class Circle implements Area
{
        public double compute(double x,double y)
       {        return pi*x*x;        }
}
class Area_Demo
{
        public static void main(String arg[])
       {
            Rect r=new Rect();
            Tri t=new Tri();
            Circle c=new Circle();
           System.out.println("The Area of rectangle is:"+r.compute(10,20));
           System.out.println("The Area of Triangle  is :"+t.compute(10,12));
           System.out.println("The Area of Circle  is :"+c.compute(10,12));
}
}
```

Question

- WAP to implement an interface Bank. SBI and PNB are child classes of Bank.
- Implement a method rateofinterest of Bank into the child classes.

Solution

```
interface Bank
{
float rateOfInterest();
}
class SBI implements Bank  {
public float rateOfInterest()
    {      return 9.15f;       }
}
class PNB implements Bank  {
public float rateOfInterest()
 {       return 9.7f;       }
}
class TestInterface2
{
public static void main(String[] args)
{
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```