GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT302

OBJECT ORIENTED PROGRAMMING

UNIT 6

INHERITANCE AND POLYMORPHISM

Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

- Inheritance
- Creating Subclasses
- Method Overriding
- Dynamic Method Dispatch
- super Keyword
- final keyword
- Final Variables
- Static Class ,Methods, Block and Variables

Inheritance

- Inheritance is the mechanism in java by which one class is allow to inherit the features (fields and methods) of another class.
- It is process of deriving a new class from an existing class.
- A class that is inherited is called a superclass and the class that does the inheriting is called a subclass.
- Inheritance represents the IS-A relationship, also known as parent-child relationship.
- The keyword used for inheritance is extends.

is-a relationship

- Inheritance is an is-a relationship. We use inheritance only if an is-a relationship is present between the two classes.

Here are some examples:

- A car is a vehicle.
- Orange is a fruit.
- A surgeon is a doctor.
- A dog is an animal.

Inheritance

Terms used in Inheritance

- **Class:**

  A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

- **Sub Class/Child Class:**

  Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

- **Super Class/Parent Class:**

  Super class is the class from where a subclass inherits the features. It is also called a base class or a parent class.

- **Reusability:**

  As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Inheritance

- **The syntax of Java Inheritance**

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

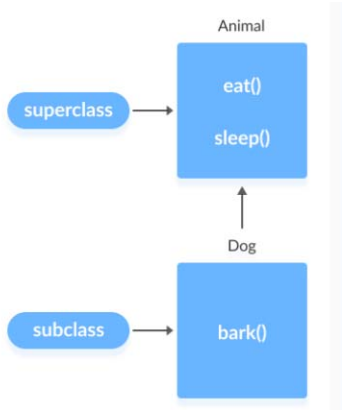
The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

- A class which is inherited is called a parent or superclass, and the new class is called child or subclass.

For example,

```
class Animal
{
// eat() method
// sleep() method
}
class Dog extends Animal
{
// bark() method
}
```

- In Java, we use the extends keyword to inherit from a class. Here, we have inherited the Dog class from he Animal class



Example

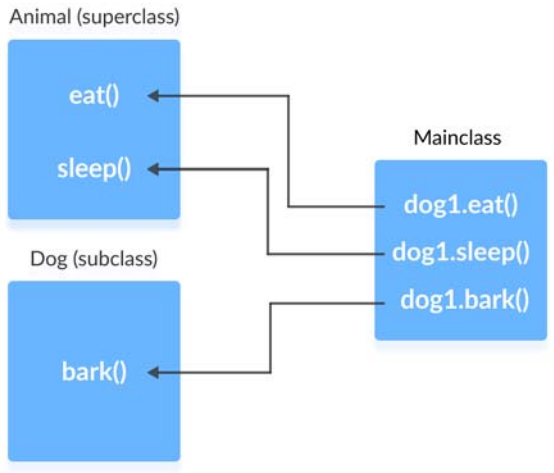
```
class Animal
{
        public void eat()
        {
        System.out.println ("I can eat");
        }
        public void sleep()
        {
        System.out.println ("I can sleep");
        }
}
class Dog extends Animal
 {
        public void bark()
        {
        System.out.println ("I can bark");
        }
 }
```

```
class Main
 {
 public static void main(String[] args)
{
Dog dog1 = new Dog();
dog1.eat();
dog1.sleep();
dog1.bark();
}
}
```

Output

I can eat
I can sleep
I can bark

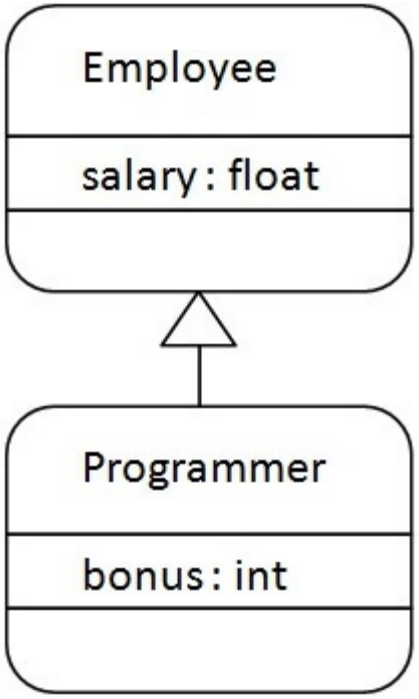
Here, we have inherited a subclass Dog from superclass Animal. The Dog class inherits the methods eat() and sleep() from the Animal class. Hence, objects of the Dog class can access the members of both the Dog class and the Animal class.



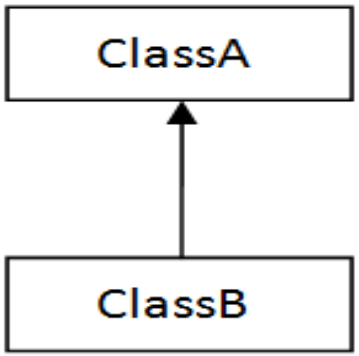
Example **Programmer IS-A Employee**

```
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);

}
}
```

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

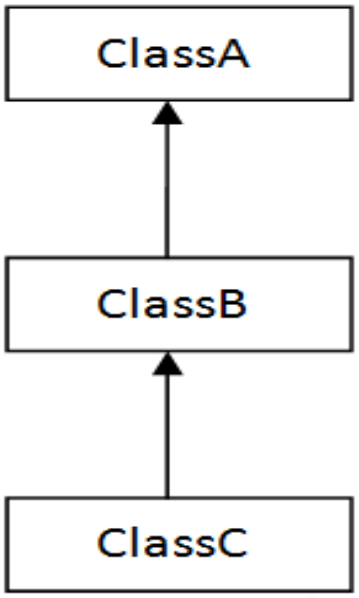


Types of inheritance in java

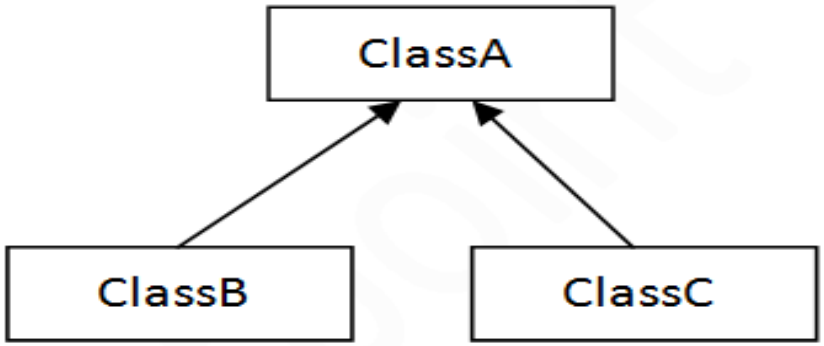


1) Single

Single inheritance - Class B extends from class A only.



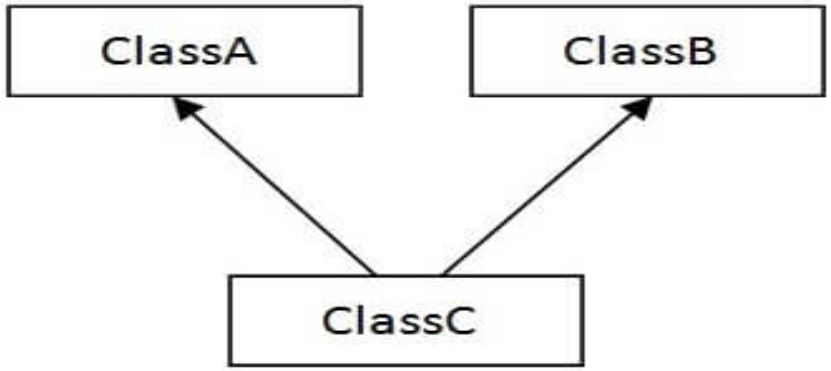
2) Multilevel



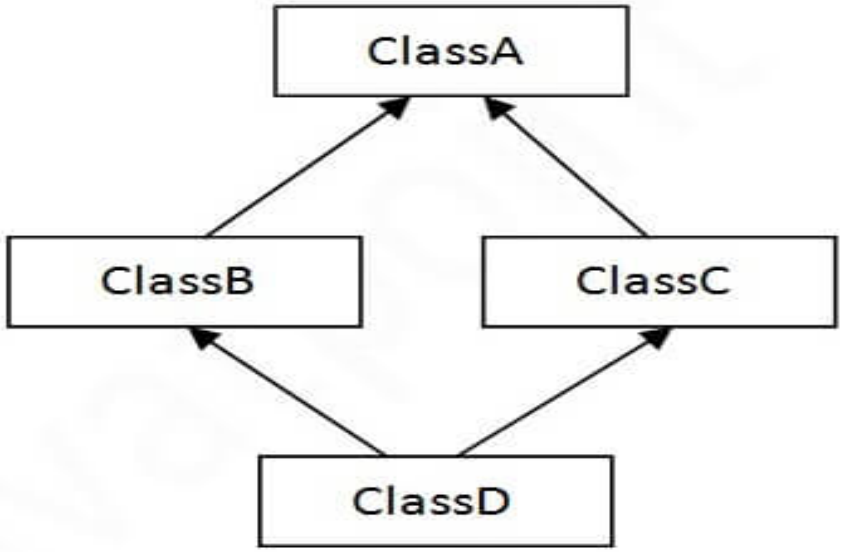
3) Hierarchical

Multilevel inheritance - Class B extends from class A; then class C extends from class B.
Hierarchical inheritance - Class A acts as the super class for classes B, C.

Types of inheritance in java



4) Multiple



5) Hybrid

Multiple inheritance - Class C extends from interfaces A and B.

Hybrid inheritance - Mix of two or more types of inheritance.

Single Inheritance

- When a class inherits another class, it is known as a *single inheritance*.
- In this example, Dog class inherits the Animal class, so there is the single inheritance.

```
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();
}}
```

Output:
barking...
eating...

Multilevel Inheritance

- When there is a chain of inheritance, it is known as *multilevel inheritance*.
- As you can see in this example, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```
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();
}}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.
- In this example, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:
meowing...
eating...

Multiple inheritance

- Multiple inheritance is not supported in java.....why ??
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

```
class A
{
void msg(){System.out.println("Hello");}
}
class B
{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were


 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Compile Time Error

Polymorphism

- **Polymorphism in Java** is a concept by which we can perform a ***single action in 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 (Method overloading)
  - runtime polymorphism (method overriding)

Method Overloading in Java

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- 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.

Method Overloading

Different ways to overload the method

There are two ways to overload the method in java

- By changing number of arguments
- 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));
}
}
```

Output:
22
33

Method Overloading

- **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));
}}
```

Output:
22
24.9

Method Overloading

- **Why Method Overloading is not possible by changing the return type of method only?**
- In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b)
{
return a+b;
}
static double add(int a,int b)
{
return a+b;
}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Output:
Compile Time Error: method add(int,int) is already defined in class Adder

System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

Method overriding

- Declaring a method in **sub class** which is already present in **parent class** is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.
- In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

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

Method overriding

Understanding the problem without method overriding

```
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

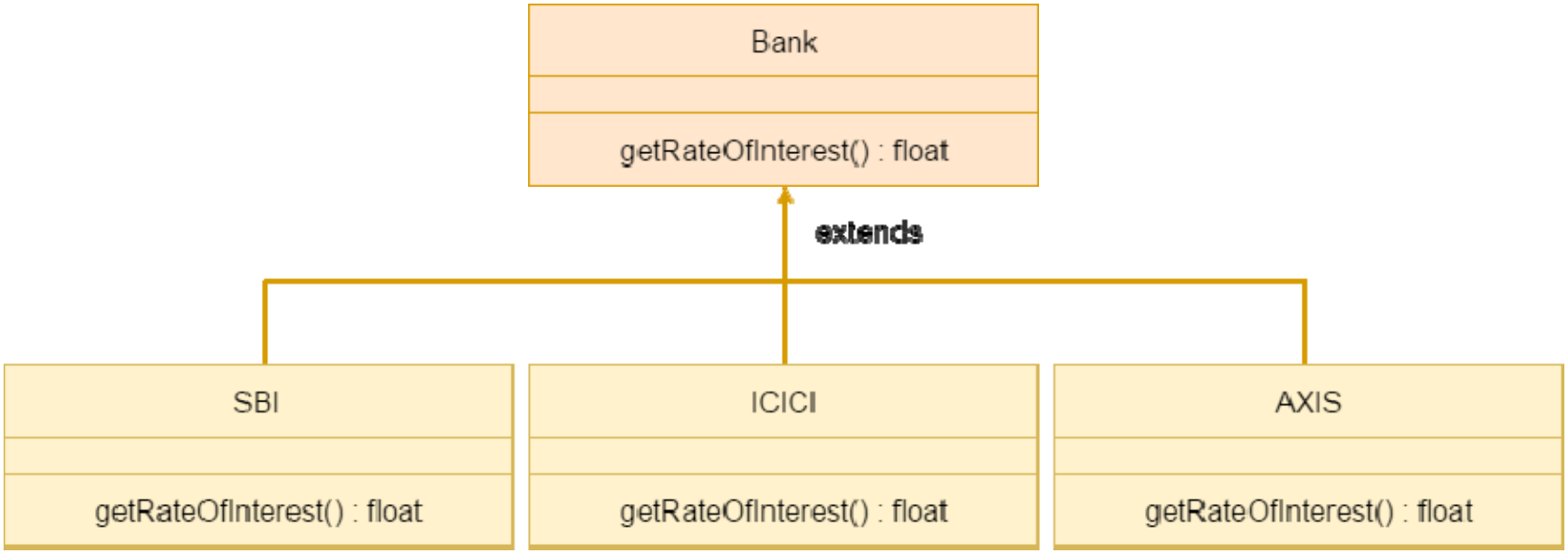
Output:
Vehicle is running

Example of method overriding

```
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output:
Bike is running safely

Example of Java Method Overriding



Example

- Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.

```
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Test2{
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();

}
}
```

Output:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Example: Shape

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
```

```
class Polymorphism{
public static void main(String args[])
{
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Method overriding

Advantage of method overriding

- The main advantage of method overriding is that the class can give its own specific implementation to a inherited method **without even modifying the parent class code.**
- This is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code.

Difference between Overloading and Overriding

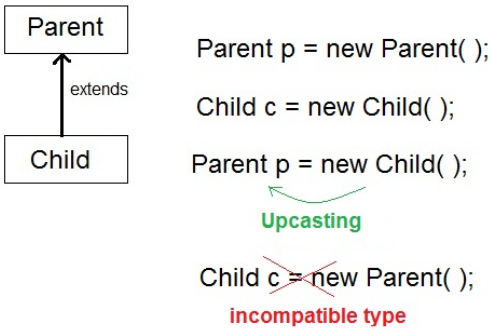
Method Overloading	Method Overriding
Parameter must be different and name must be same.	Both name and parameter must be same.
Compile time polymorphism.	Runtime polymorphism.
Increase readability of code.	Increase reusability of code.
It is performed between two classes using inheritance relation.	It requires always inheritance.
It should have methods with the same name but a different signature.	It should have methods with same name and signature.
It uses static binding	It uses the dynamic binding.
It is code refinement technique.	It is a code replacement technique.

Dynamic Method Dispatch

- Method overriding is one of the ways in which Java supports Runtime Polymorphism.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- A superclass reference variable can refer to a subclass object, This is also known as **upcasting**.

Upcasting in Java

- When **Parent** class reference variable refers to **Child** class object, it is known as **Upcasting.** In Java this can be done and is helpful in scenarios where multiple child classes extends one parent class. In those cases we can create a parent class reference and assign child class objects to it.



Example

```
class Animal
{
    public void move()
    {
        System.out.println("Animals can move");
    }
}
class Dog extends Animal
{
    public void move()
    {
        System.out.println("Dogs can walk and run");
    }
}
```

```
public class TestDog
{
    public static void main(String args[])
    {
    Animal a = new Animal(); // Animal reference and object
    Animal b = new Dog(); // Animal reference but Dog object
    a.move(); // runs the method in Animal class
    b.move(); // runs the method in Dog class
    }
}
```

output :

Animals can move

Dogs can walk and run

Dynamic Method Dispatch (Conclusion)

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs.
- Thus, this determination is made at run time.

Super Keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Usage of Java super Keyword

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

Super Keyword

- **super is used to refer immediate parent class instance variable.**

  We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
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();
        }
}
```

Output:
black
white

Super Keyword

- **super can be used to invoke parent class method**

  The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class.

```
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();
    }
}
```

Output:

eating...

barking...

Super Keyword

- **super is used to invoke parent class constructor.**

  The super keyword can also be used to 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();
}

}
```

Output:
animal is created
dog is created

Final keyword

- 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

     If you make any variable as final, you **cannot change the value** of final variable(It will be constant).
  2. Method

     If you make any method as final, you **cannot override** it.
  3. Class

     If you make any class as final, you **cannot extend** it.

Java final variable

```
class Bike9{
 final int speedlimit=90;//final variable

 void run(){
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
```

Output: Compile Time Error

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

Final keyword

Java final method

```
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

If you make any method as final, you cannot override it.

Final keyword

Java final class

```
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 Honda1();
  honda.run();
  }
}
```

Output: Compile Time Error

If you make any class as final, you cannot extend it.

Static Keyword

- The **static keyword** in Java is used for memory management mainly.
- We can apply static keyword with variables, methods, blocks and nested classes.
- The static keyword belongs to the class than an instance of the class.

The static can be:

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class

Java static variable

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

problem without static variable

```
class Student{
    int rollno;
    String name;
    String college="UVPCE";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Static Keyword

- **Example of static variable**

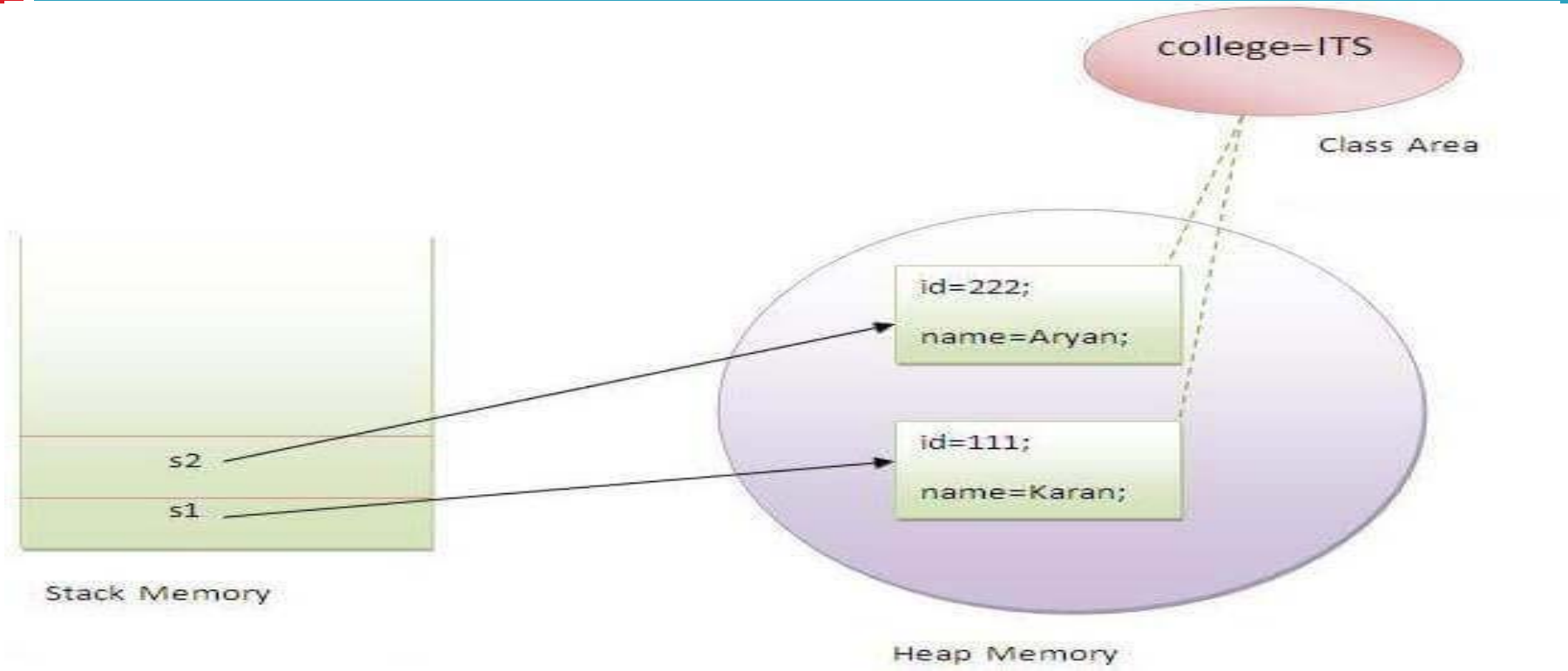
```
class Student{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   //constructor
   Student(int r, String n){
   rollno = r;
   name = n;
   }
   //method to display the values
   void display (){System.out.println(rollno+" "+n
      ame+" "+college);}
}
```

```
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");

 Student s2 = new Student(222,"Aryan");


 s1.display();
 s2.display();
 }
}
```

Output:

```
111 Karan ITS
222 Aryan ITS
```

Example


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

Static Keyword

Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Example

```
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = “UVPCE";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
   s2.display();
   s3.display();
   }
}
```

Output:
111 Karan UVPCE
222 Aryan UVPCE
333 Sonoo UVPCE

Static Keyword

Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

```
class A2
{
  static
{System.out.println("static block is invoked");}
  public static void main(String args[])
{
   System.out.println("Hello main");
  }
}
```

Output:
static block is invoked
Hello main


THANK YOU!
ANY QUESTIONS?