CSE-2005 (Object Oriented Programming Systems)

Module 1:Object-Oriented Programming – Fundamentals

Dr. Arundhati Das

Different Programming Paradigms

Functional/procedural programming:

- program is a list of instructions to the computer : e.g. C, Cobol, Fortran,Pascal

Object-oriented programming

- program is composed of a collection objects that communicate with each other: e.g. Java, Python, C++

Concepts of OOPS

1. Object
2. Class
3. Constructor
4. Polymorphism
5. Inheritance
6. Abstraction
7. Encapsulation

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc.

A dog is an object because it has states like color, name as well as behaviors like barking, eating.

Class

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

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Student (Class) {

Defined by an instance (Object)

Name, Roll number, Branch (Properties of the object)

Read(), Write() and Play() (Tasks Performed)

}

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance.

It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism and Abstraction

Polymorphism:

If one task is performed in different ways, it is known as polymorphism.

To draw something, for example, shape, triangle, rectangle, etc.

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

Abstraction:

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

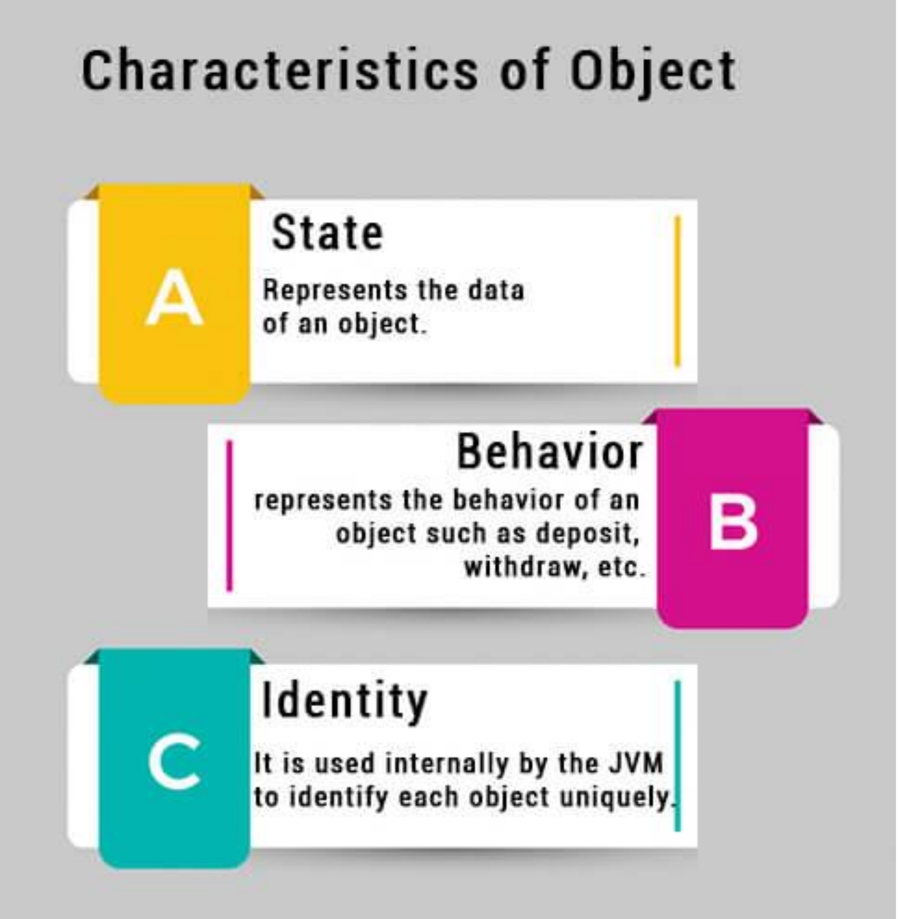
In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

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

A java class is the example of encapsulation.

Object



Characteristics of Object

A

State

Represents the data of an object.

B

Behavior

represents the behavior of an object such as deposit, withdraw, etc.

C

Identity

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, known as its state.

It is used to write, so writing is its behavior.

P can be identifier

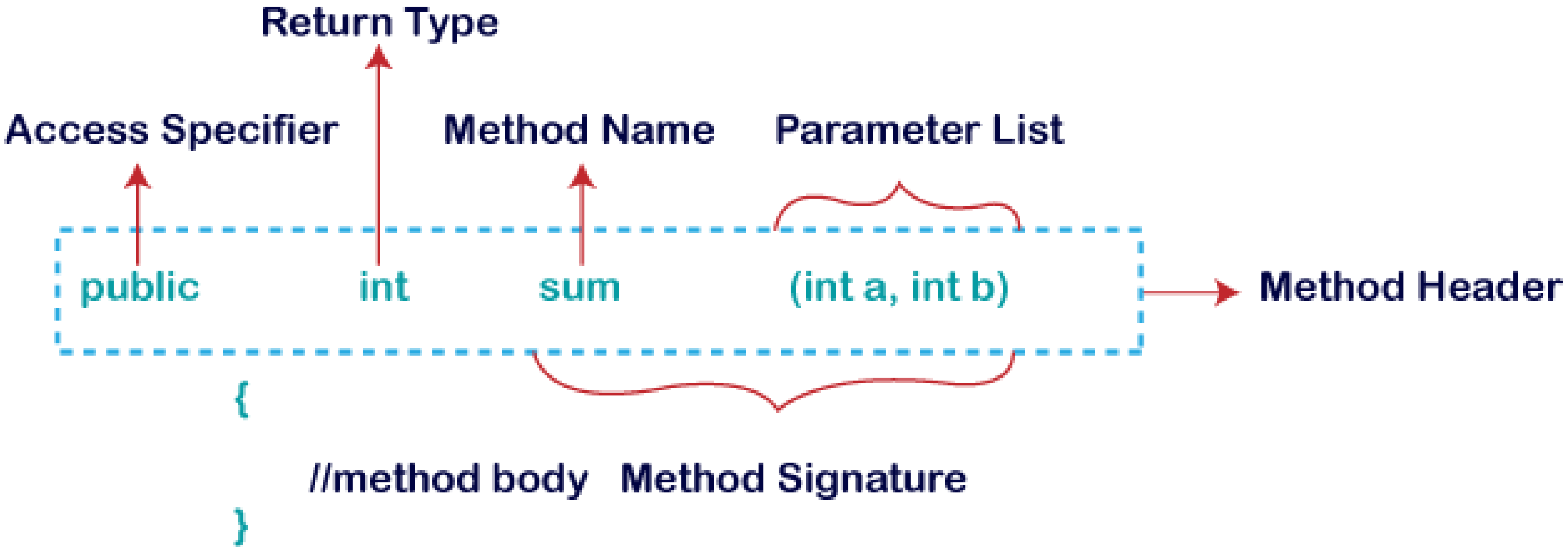
Class

- 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, not physical.

Instance variable, methods, new keyword

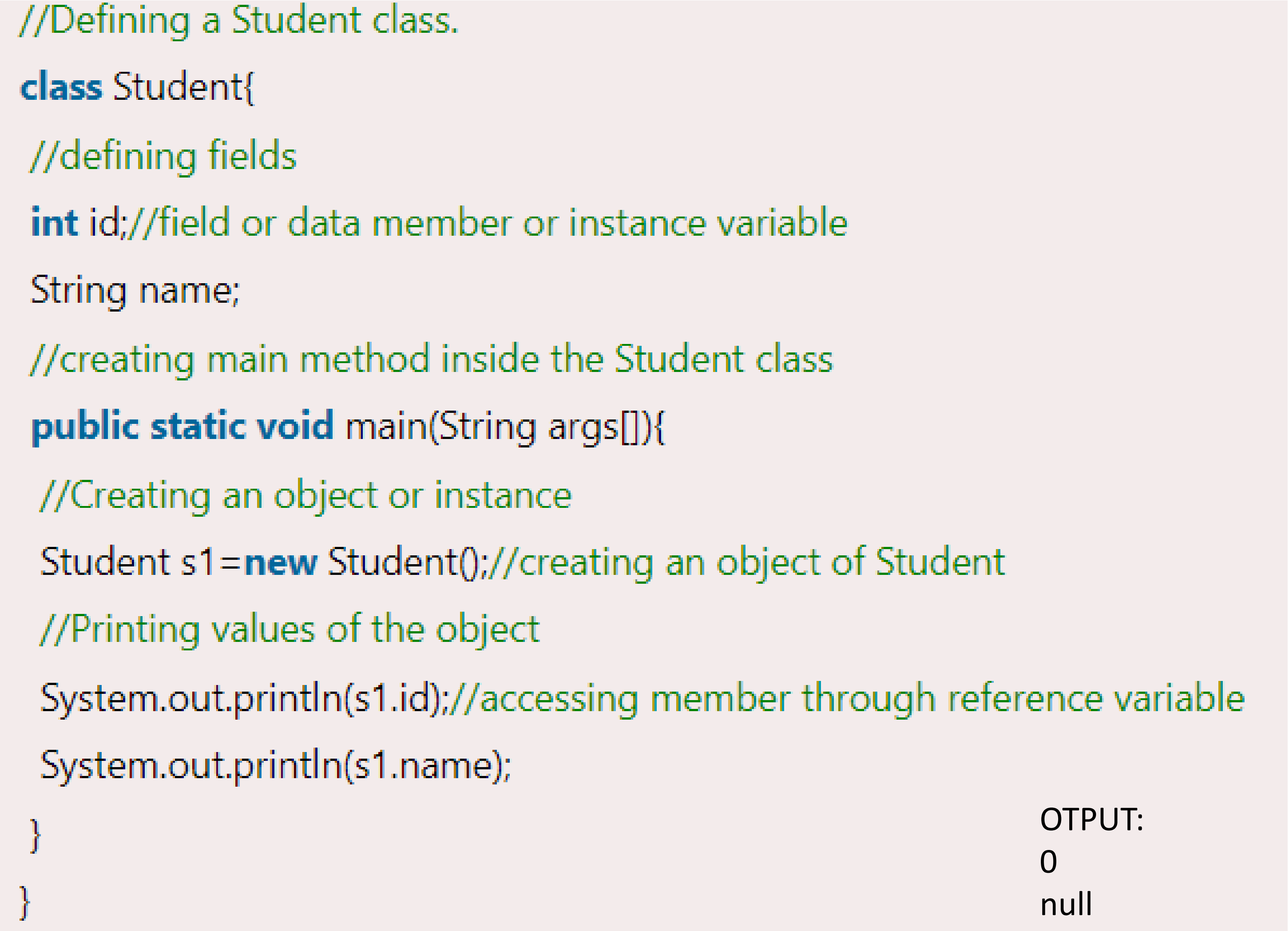
Method:

A method is like a function which is used to expose the behavior of an object.



New:

The new keyword is used to allocate memory at runtime.

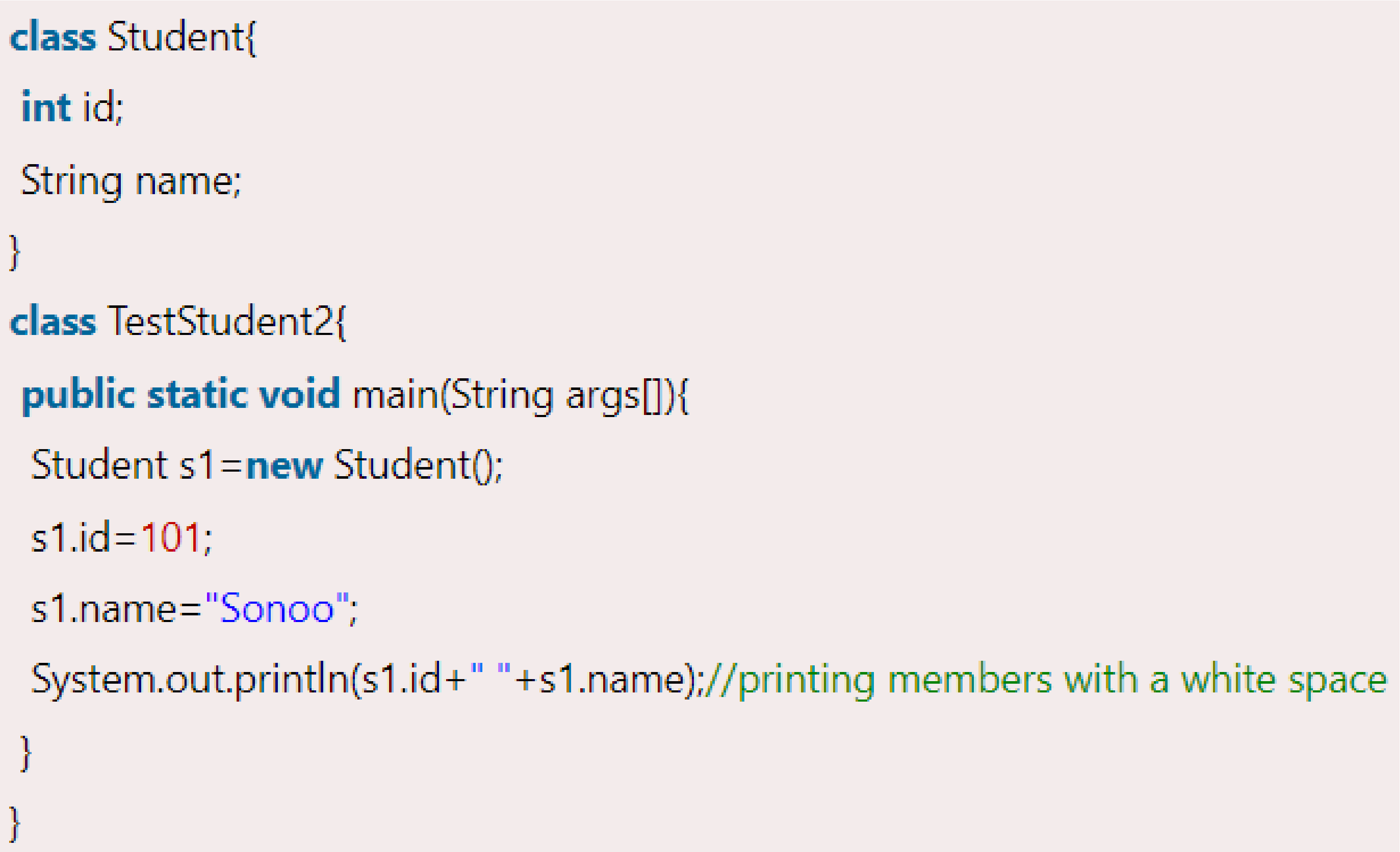


```
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

OTPUT:

```
0
null
```

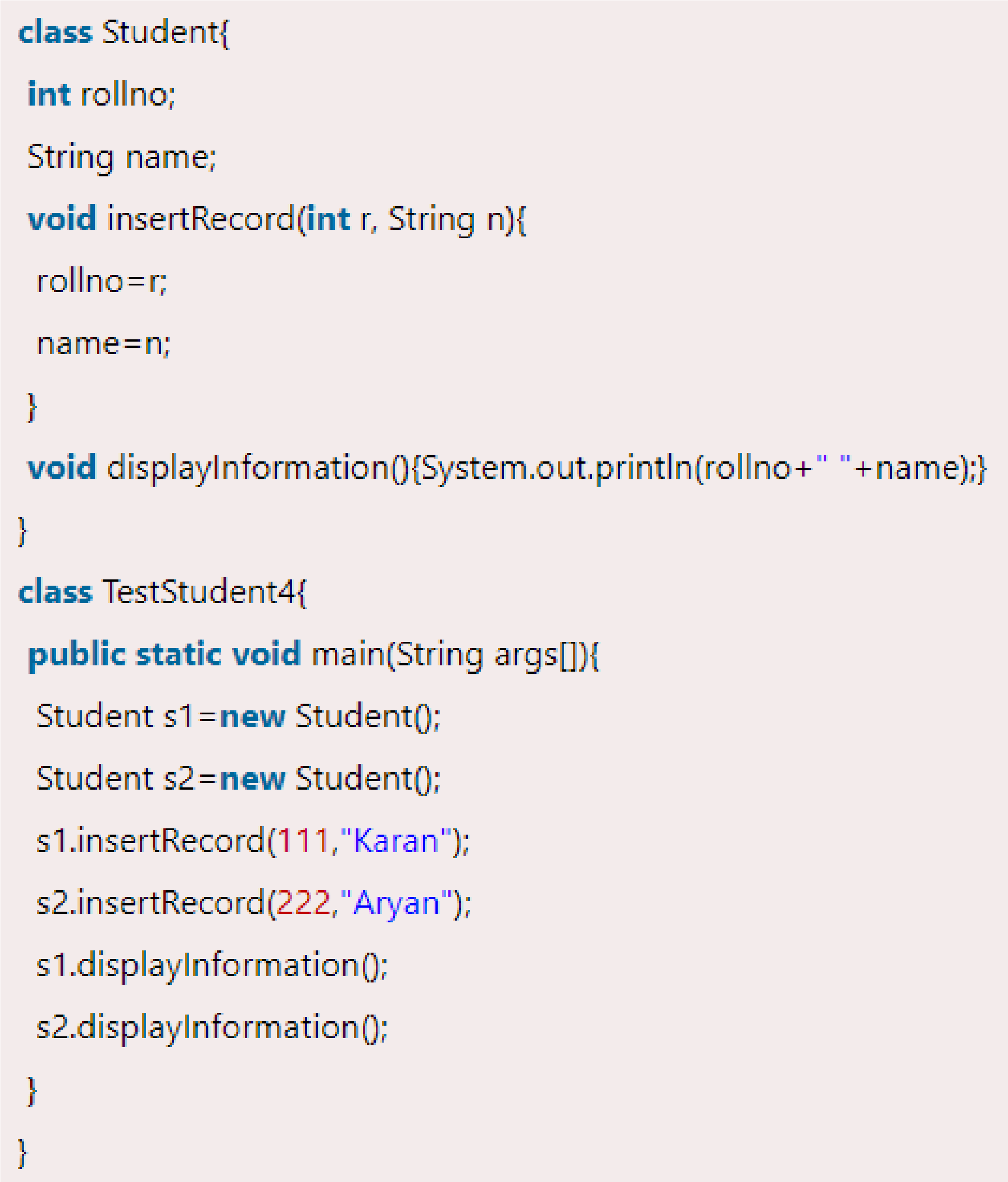
Object and Class Example: Initialization through reference



```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

```
OTPUT:
101 Sonoo
```

Initialization through method

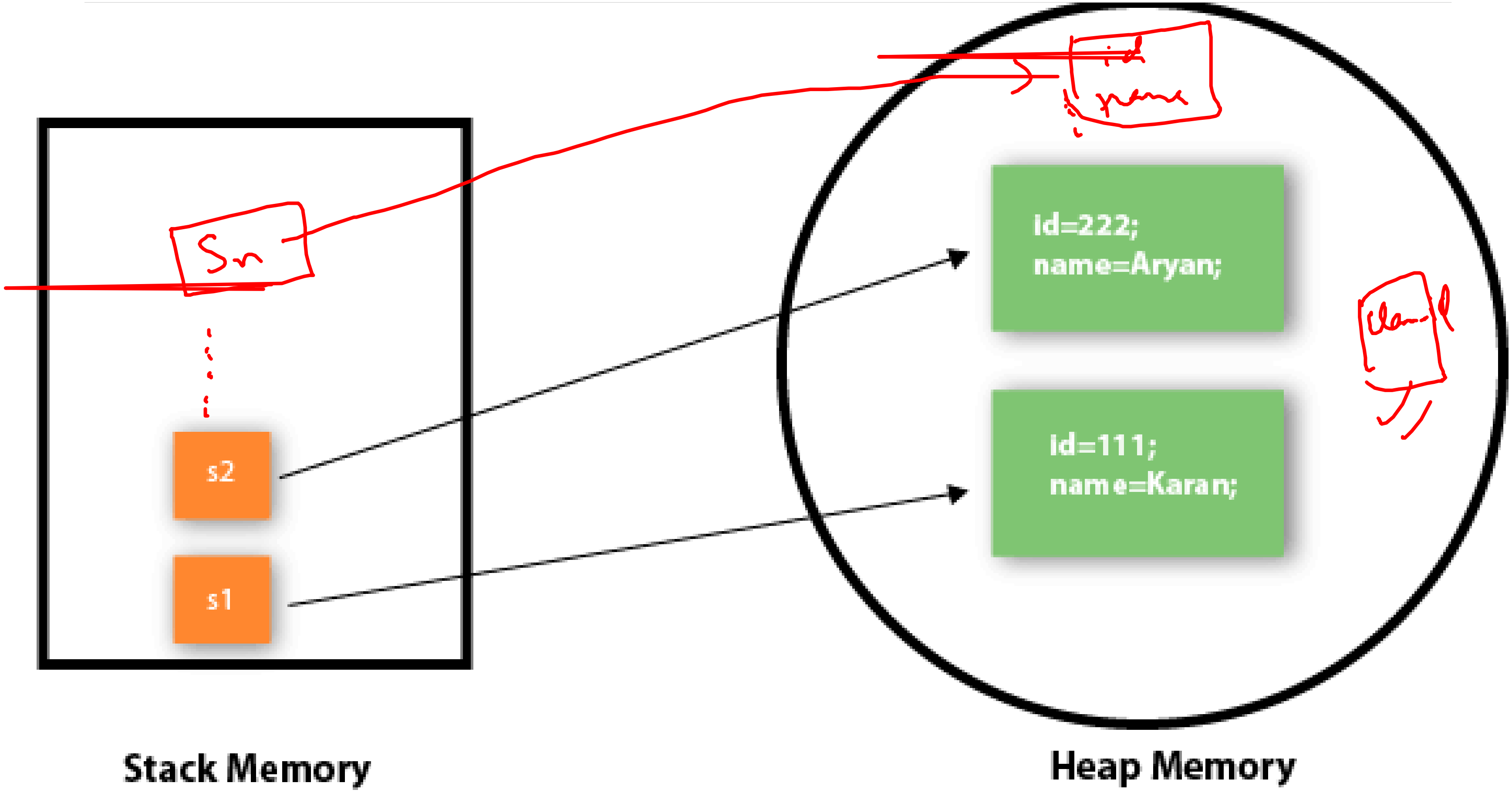


```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

OTPUT:

```
111 Karan
222 Aryan
```

Memory Allocation



Types of Variables

There are three types of variables in java:

- **local variable:**

A variable declared inside the body of the method is called local variable.

- **Instance variable:**

A variable declared inside the class but outside the body of the method, is called instance variable.

- **Static variable/Class Variable**

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Types of Variables

```
class A
{
  int data=50; //instance variable
  static int m=100; //static variable
         void method()
         {
         int n=90;// local variable
         }
}
```

Types of Variables

```
class Simple
{
        public static void main(String[] args)
        {
        int a=10;
        int b=10;
        int c=a+b;
        System.out.println(c);
        }

 }
```

Types of Variables

3.14

Math.PI

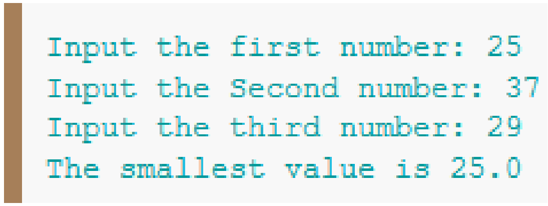
Write a Java method to find the smallest number among three numbers.

Input the first number: 25

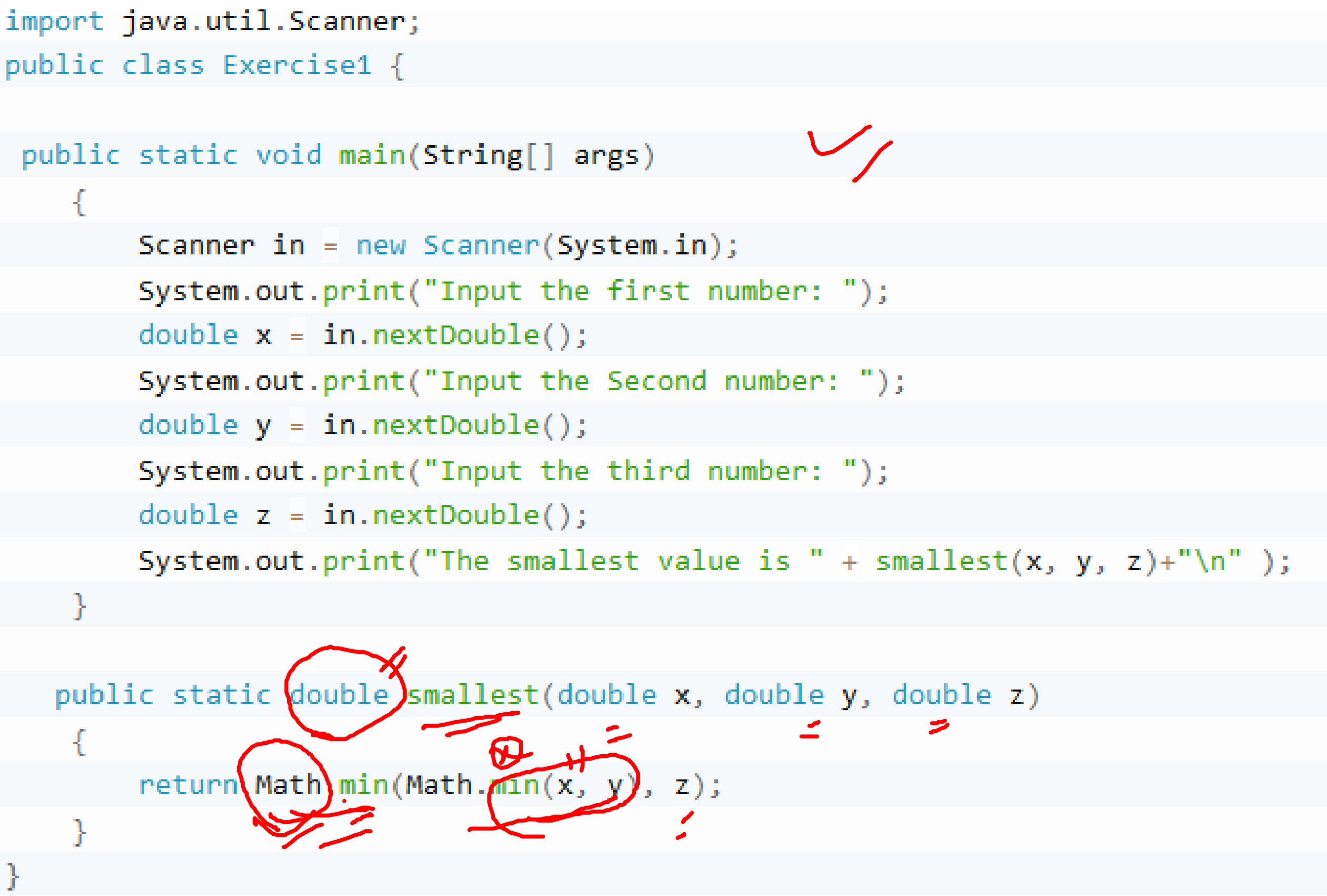
Input the Second number: 37

Input the third number: 29

Expected Output: 25.0 ✓

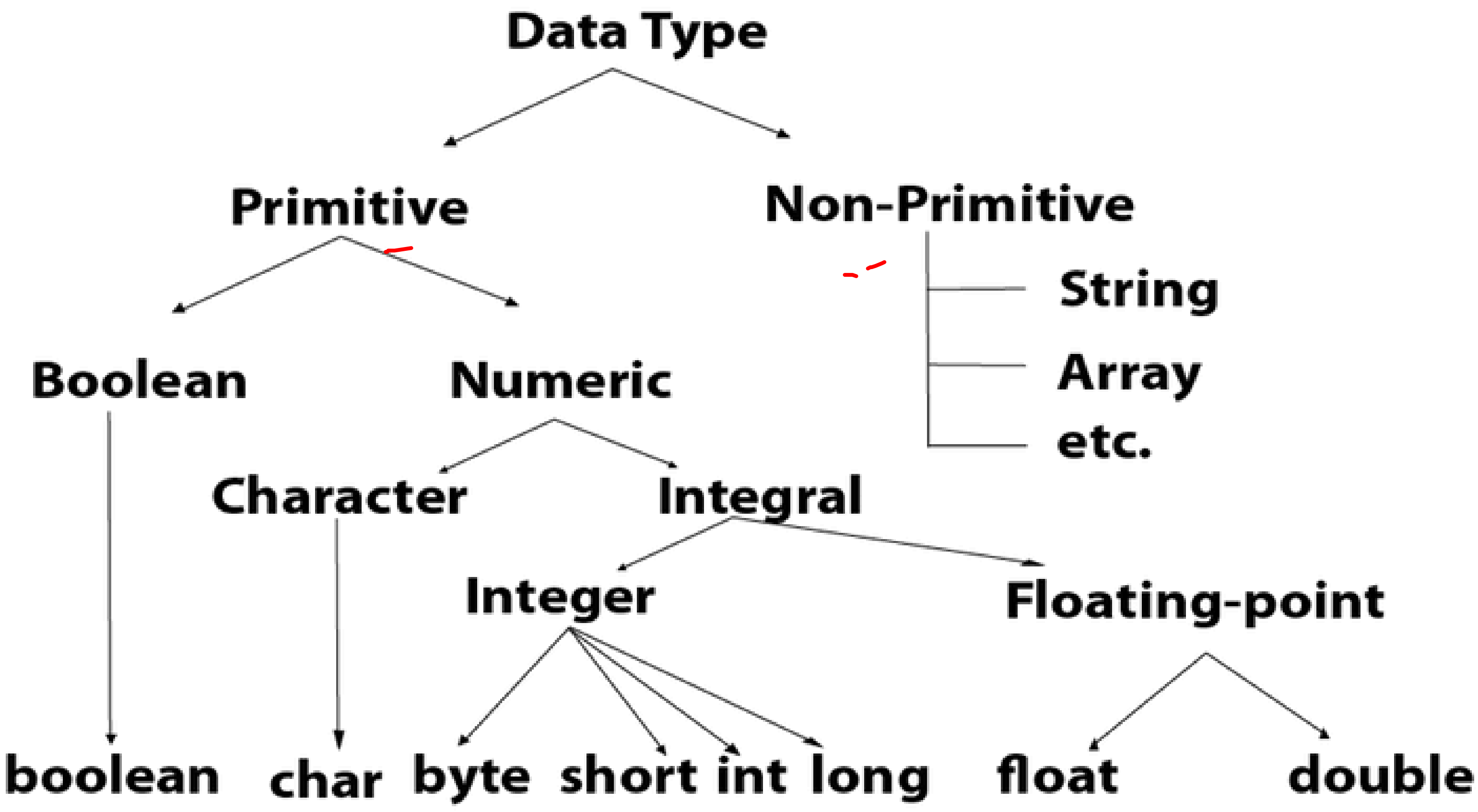


```
Input the first number: 25
Input the Second number: 37
Input the third number: 29
The smallest value is 25.0
```



```
import java.util.Scanner;
public class Exercise1 {

 public static void main(String[] args)
   {
       Scanner in = new Scanner(System.in);
       System.out.print("Input the first number: ");
       double x = in.nextDouble();
       System.out.print("Input the Second number: ");
       double y = in.nextDouble();
       System.out.print("Input the third number: ");
       double z = in.nextDouble();
       System.out.print("The smallest value is " + smallest(x, y, z)+"\n" );
   }

  public static double smallest(double x, double y, double z)
   {
       return Math.min(Math.min(x, y), z);
   }
}
```

Data Types



Data Type
Primitive
Non-Primitive
Boolean
Numeric
String
Array
etc.
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Primitive types

1.	int	4 bytes
2.	short	2 bytes
3.	long	8 bytes
4.	byte	1 byte
5.	float	4 bytes
6.	double	8 bytes
7.	char	Unicode encoding (2 bytes)
8.	boolean	{true, false}

Click to add text

Behaviors is exactly as in C++

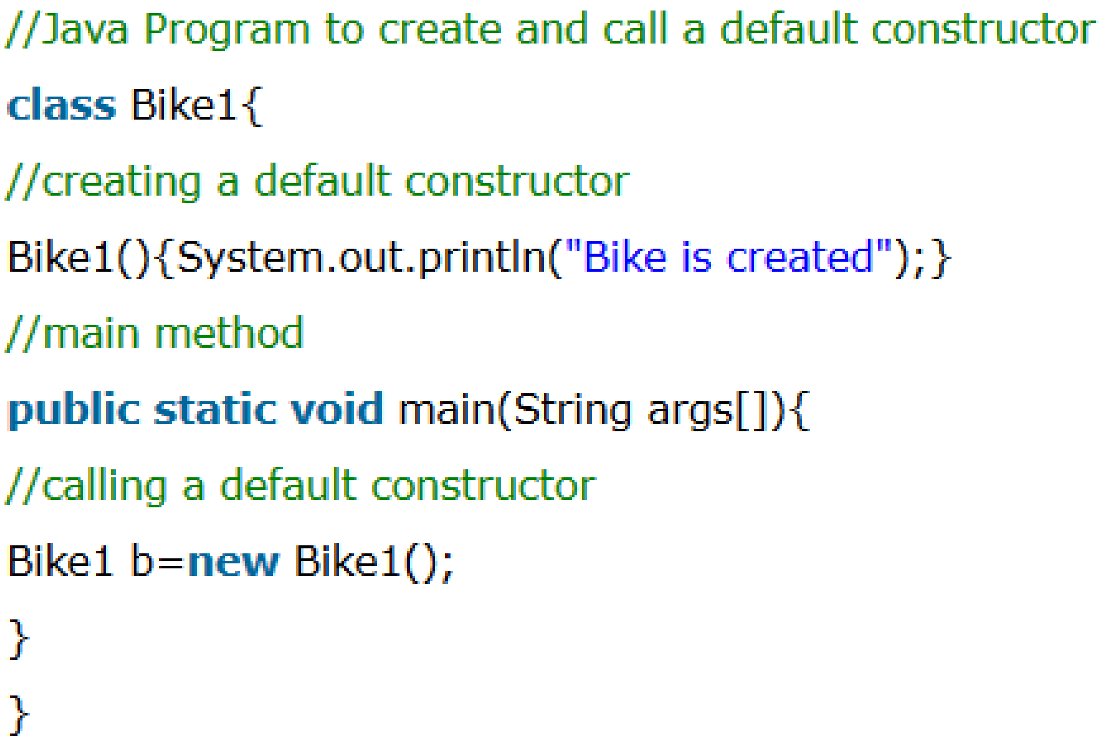
Note:
Primitive type always begin with lower-case

Constructors in Java

1. A constructor is a block of codes similar to the method. It is called when an instance of the class is created.
2. It is a special type of method which is used to initialize the object.
3. Every time an object is created using **the new()** keyword, at least one constructor is called.
4. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
5. There are two types of constructors in Java: no-arg (non-parameterized) constructor, and parameterized constructor.

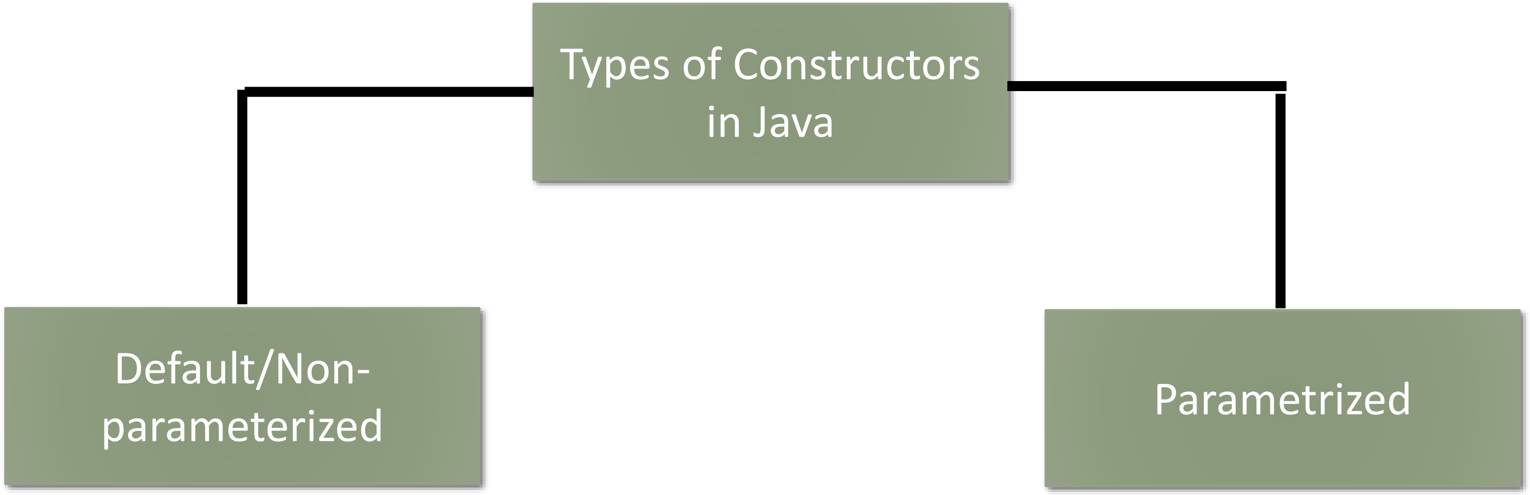
Constructors in Java

Default Constructor:



```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Constructors in Java



Types of Constructors in Java

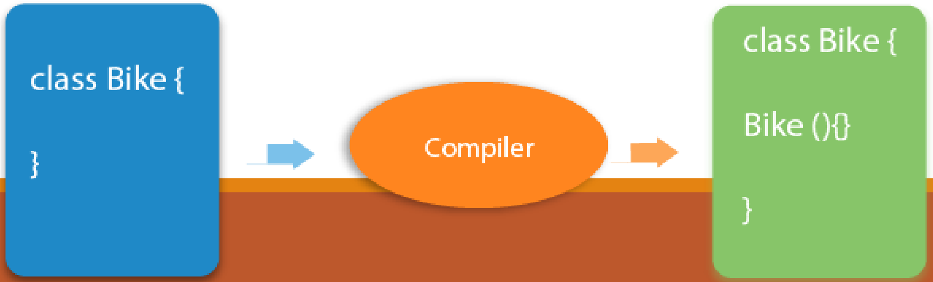
Default/Non-parameterized

Parametrized

Constructors in Java

There are some rules defined for the constructor.

1. Constructor name must be the same as its class name.
2. A Constructor must have no explicit return type.
3. Constructors can be called only once (when the object is created) unlike other method or functions which we can call repeatedly
4. A Java constructor cannot be **abstract**, **static**, and **final**.
5. If there is no constructor in a class, compiler automatically creates a default constructor.
6. Unlike C++, we don't need to write destructor in Java. Garbage collector automatically deletes objects, variables etc, so no need to write destructor in class.



```
class Bike {

}
```

Compiler

```
class Bike {

Bike (){}

}
```

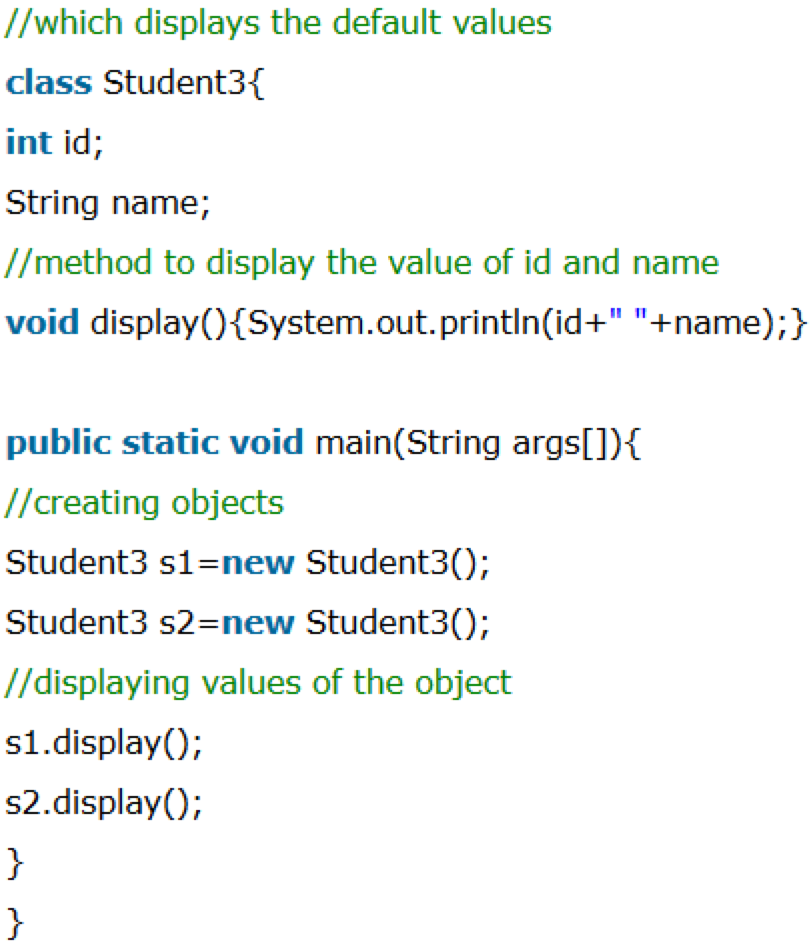
Default Constructor:

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

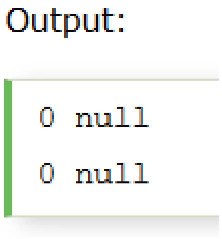
Output: Bike is created

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

Default Constructor:



```
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

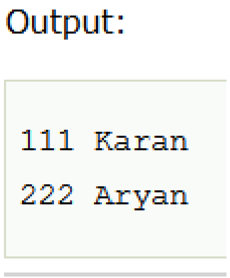


Output:

```
0 null
0 null
```

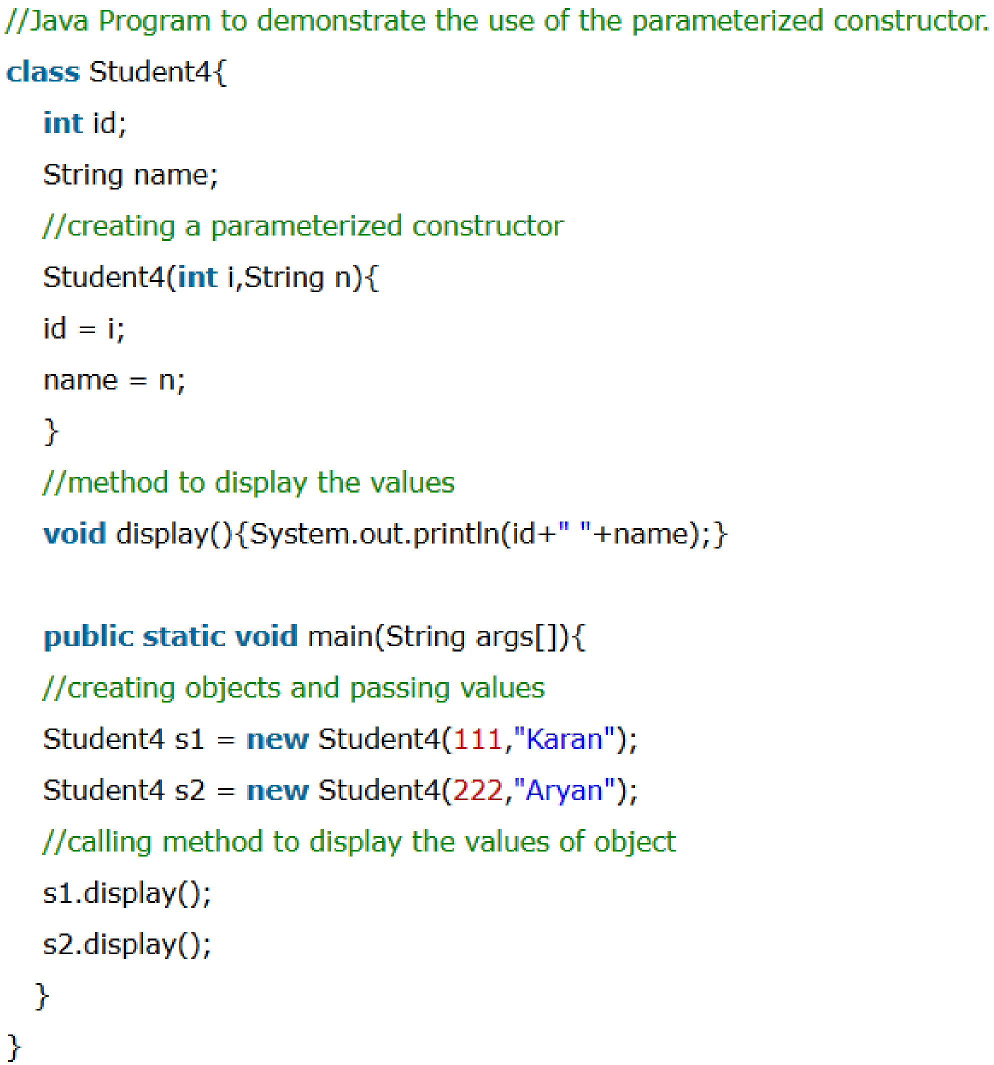
Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Parameterized Constructor:



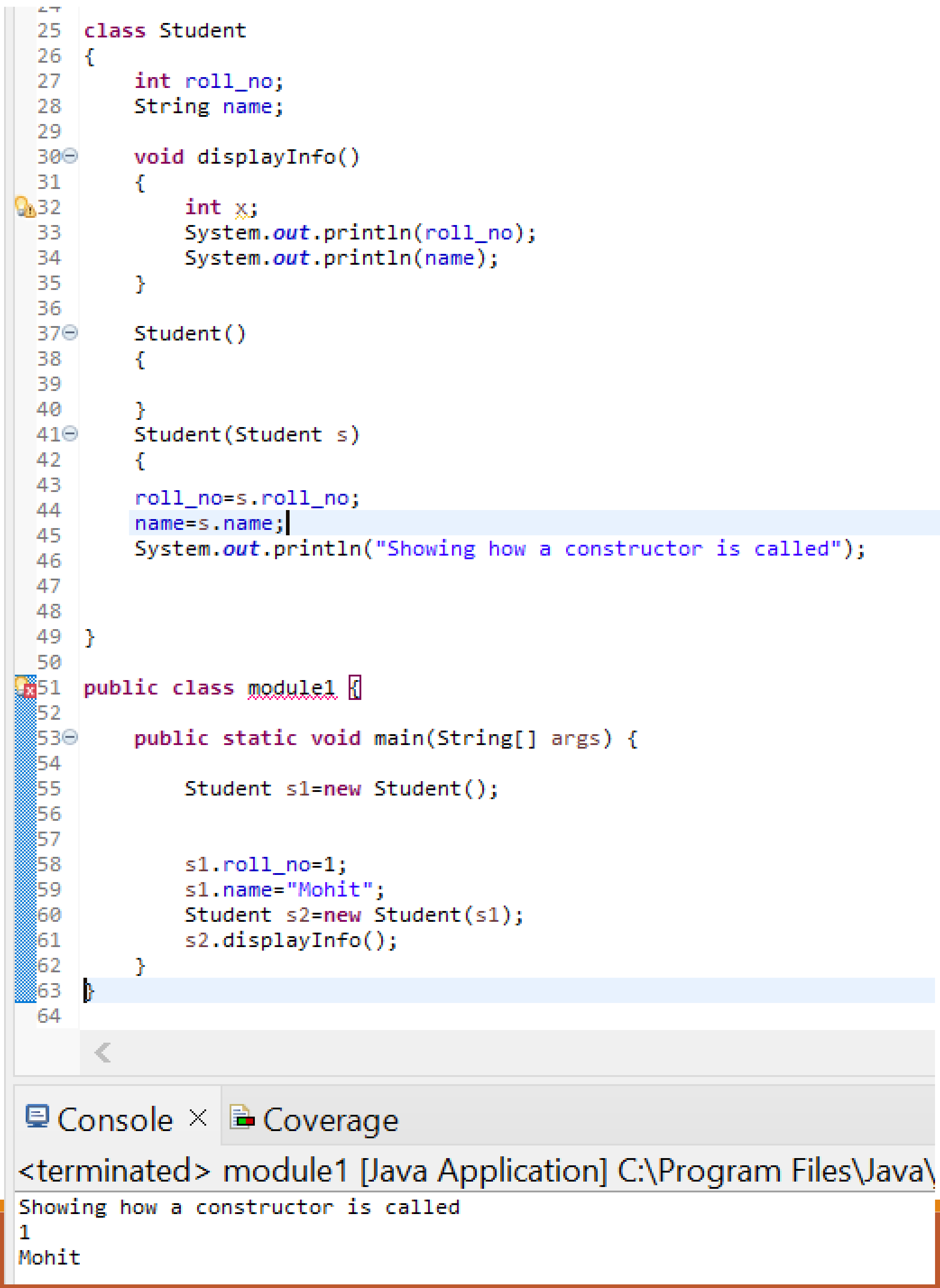
Output:

```
111 Karan
222 Aryan
```



```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
  int id;
  String name;
  //creating a parameterized constructor
  Student4(int i,String n){
  id = i;
  name = n;
  }
  //method to display the values
  void display(){System.out.println(id+" "+name);}

  public static void main(String args[]){
  //creating objects and passing values
  Student4 s1 = new Student4(111,"Karan");
  Student4 s2 = new Student4(222,"Aryan");
  //calling method to display the values of object
  s1.display();
  s2.display();
 }
}
```

Copy constructor:



```java
class Student
{
    int roll_no;
    String name;

    void displayInfo()
    {
        int x;
        System.out.println(roll_no);
        System.out.println(name);
    }

    Student()
    {

    }
    Student(Student s)
    {
    roll_no=s.roll_no;
    name=s.name;
    System.out.println("Showing how a constructor is called");



}

public class module1 {

    public static void main(String[] args) {

        Student s1=new Student();


        s1.roll_no=1;
        s1.name="Mohit";
        Student s2=new Student(s1);
        s2.displayInfo();
    }
}

```

```
<terminated> module1 [Java Application] C:\Program Files\Java\
Showing how a constructor is called
1
Mohit
```

Polymorphism: Method overloading

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.

So, we perform method overloading to figure out the program quickly.

Method overloading

Advantage of method overloading

Method overloading increases the readability of the program.

There are three ways to overload the method in OOPs

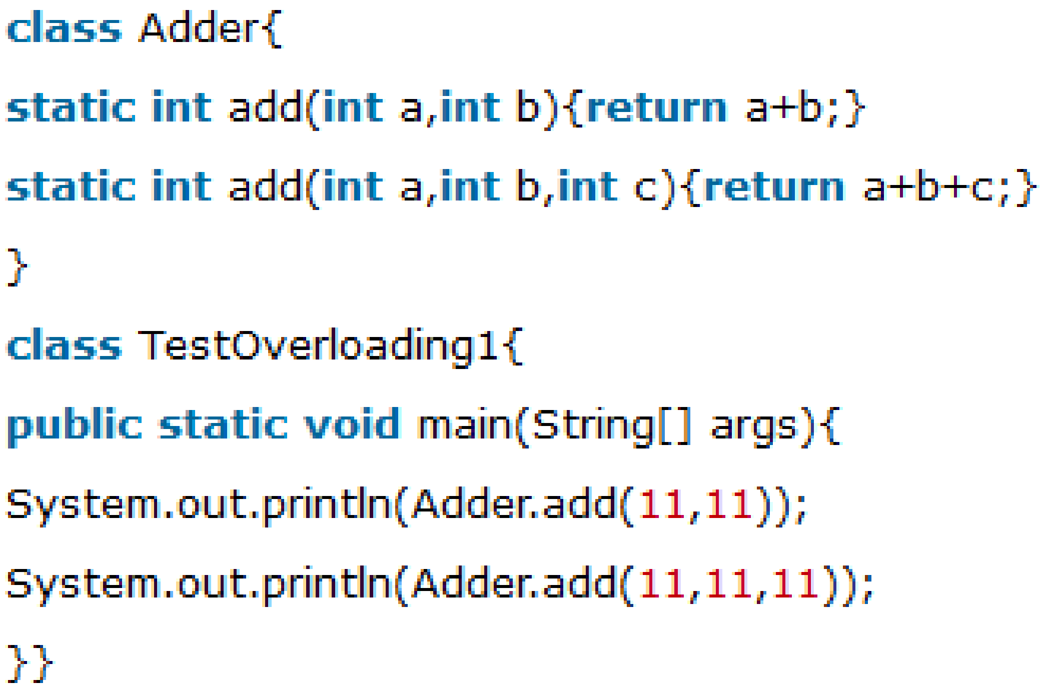
1. By changing number of arguments
2. By changing the data type of arguments
3. By changing the return type

In Java, Method Overloading is not possible by changing the return type of the method only.

Method Overloading

There are two ways to overload the method in java

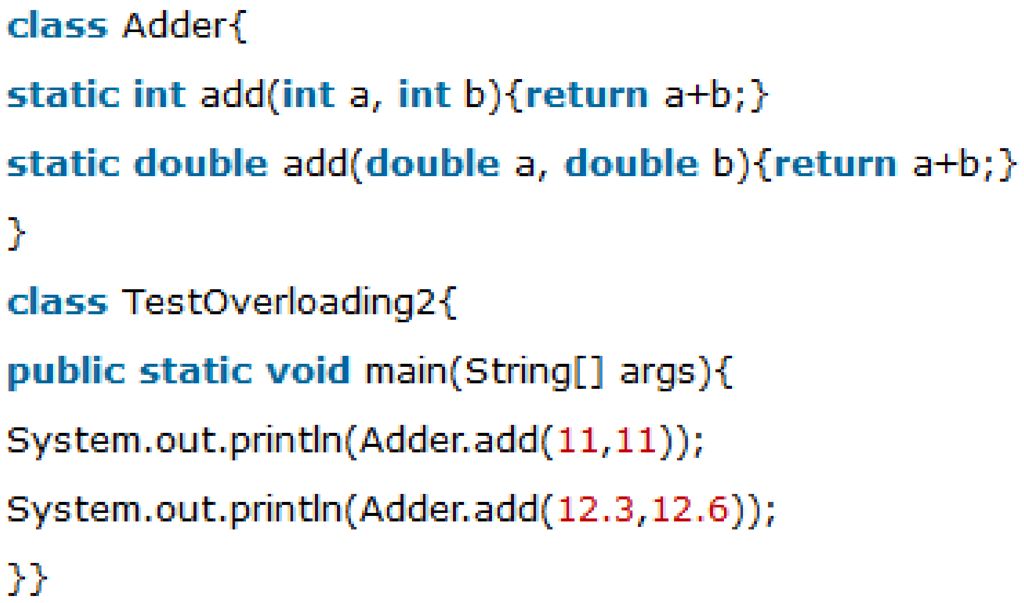
1. By changing number of arguments



```
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

2. By changing the data type



```
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));
}}
```

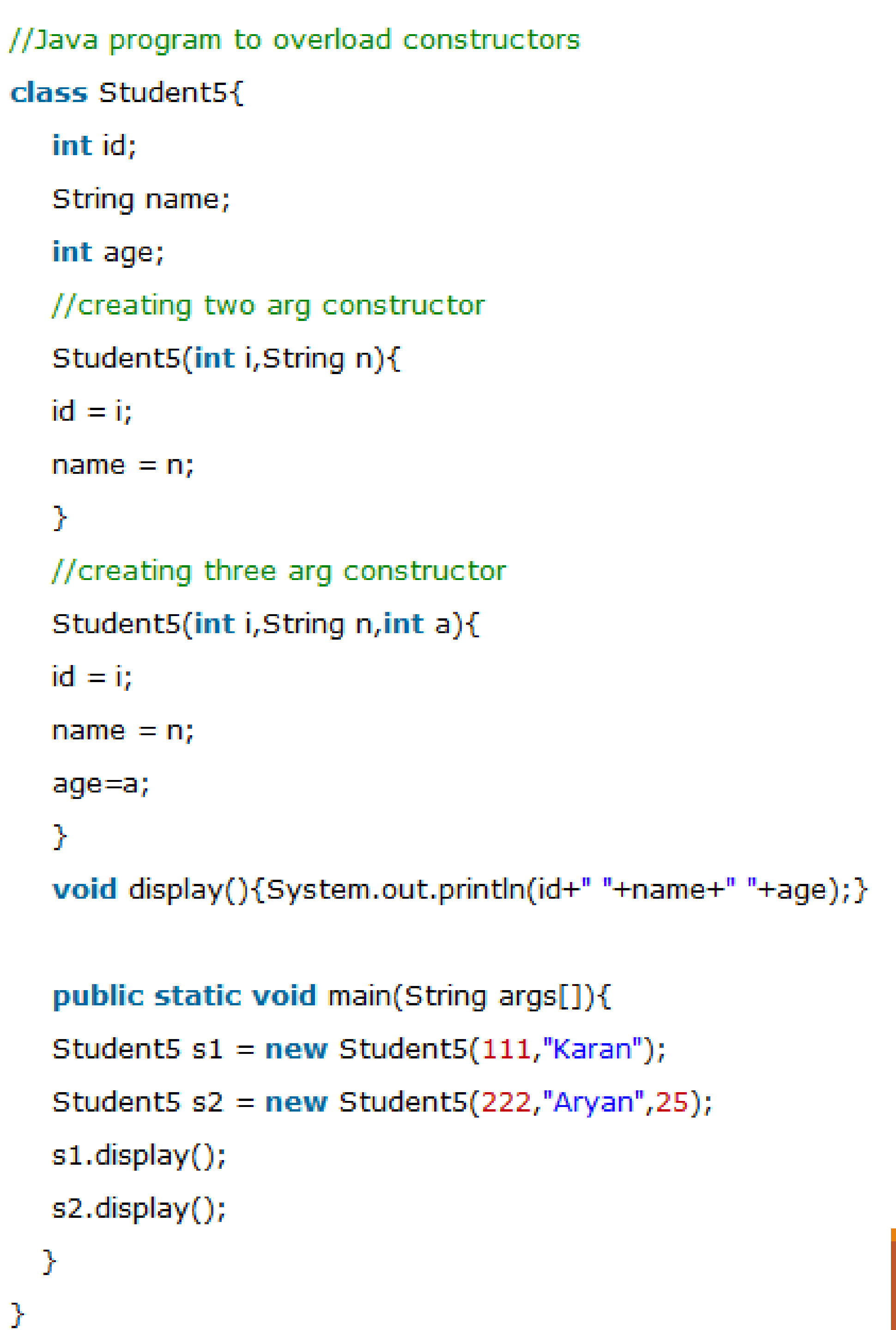
Output:

```
22
24.9
```

Constructor Overloading

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

They are differentiated by the compiler by the number of parameters in the list and their types.



```
//Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n){
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Difference between constructor and method

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Static Keyword

The static keyword in Java is used for memory management mainly.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Static Block

1) Java static variable

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

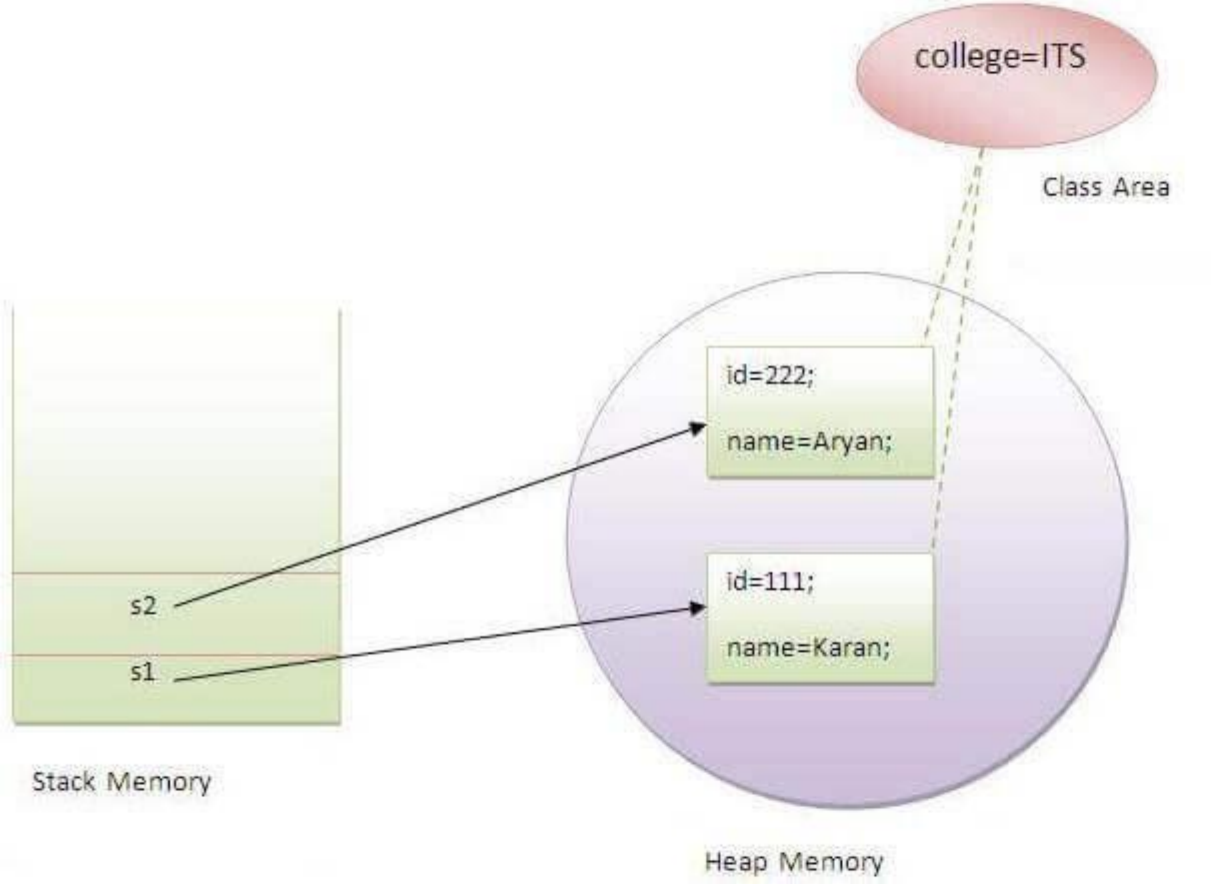
Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created.

All students have its **unique rollno and name**, so **instance data member is good** in such case.

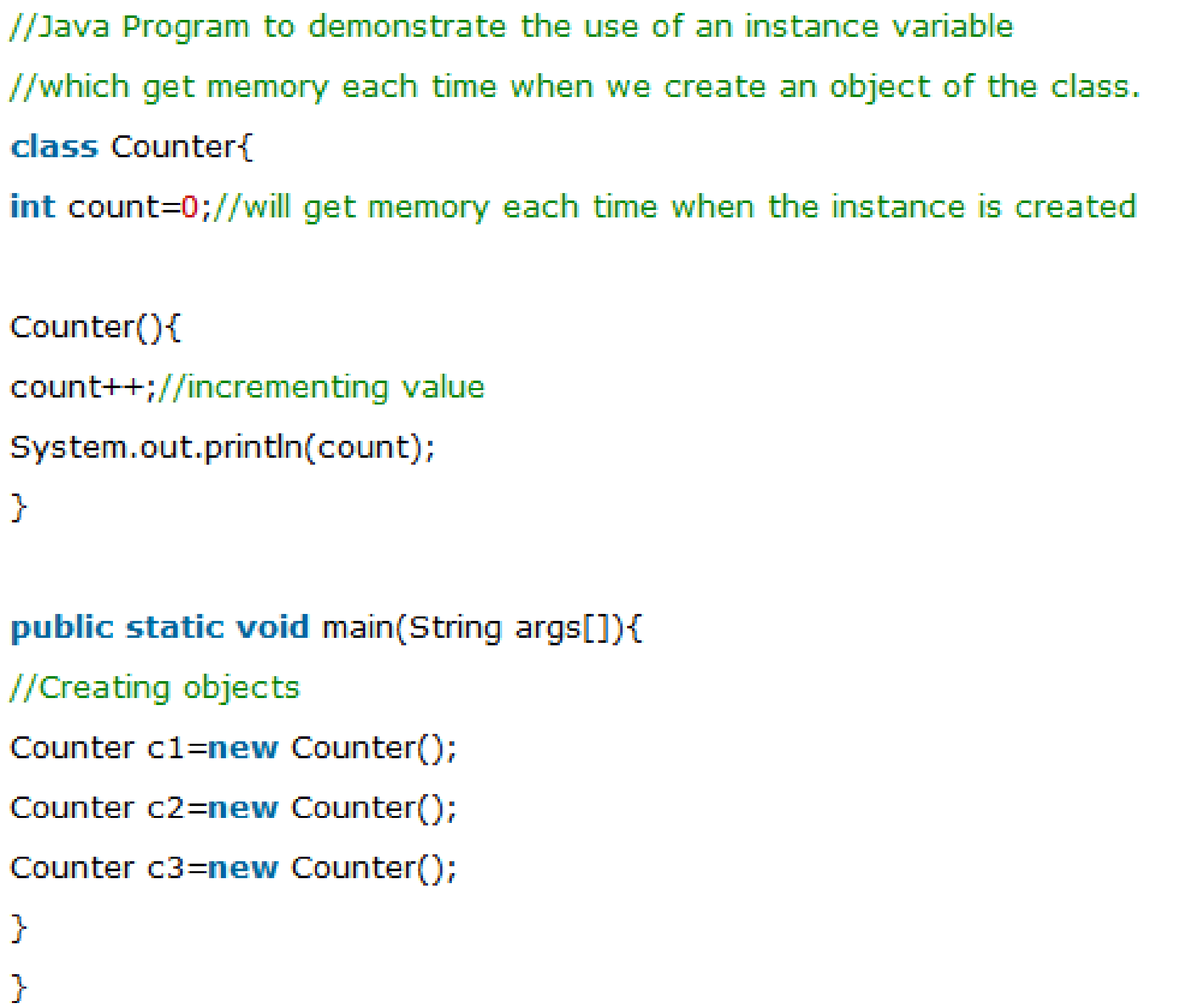
Here, "college" refers to the **common property of all objects**. If we make it static, this field will get the memory only once.

Static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

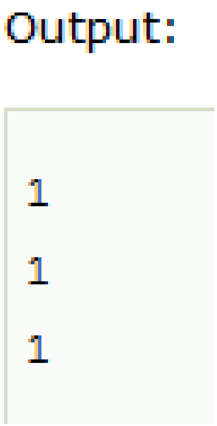
1) Java static variable



1) Java static variable



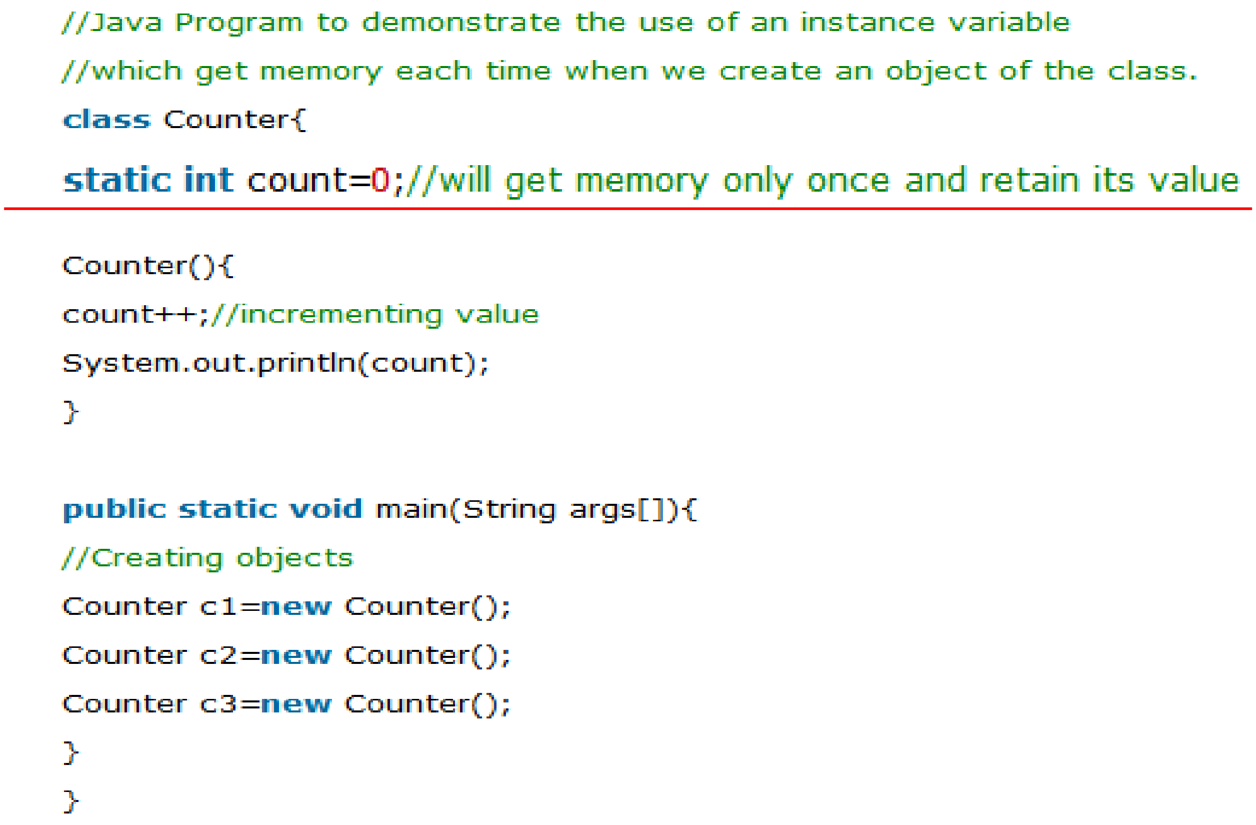
```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter{
int count=0;//will get memory each time when the instance is created

Counter(){
count++;//incrementing value
System.out.println(count);
}

public static void main(String args[]){
//Creating objects
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```



Output:

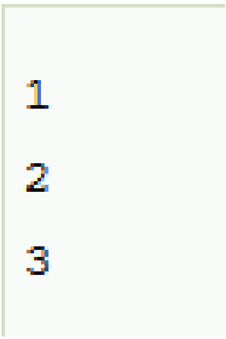
```
1
1
1
```

1) Java static variable



```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter{
static int count=0;//will get memory only once and retain its value

Counter(){
count++;//incrementing value
System.out.println(count);
}

public static void main(String args[]){
//Creating objects
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Output:

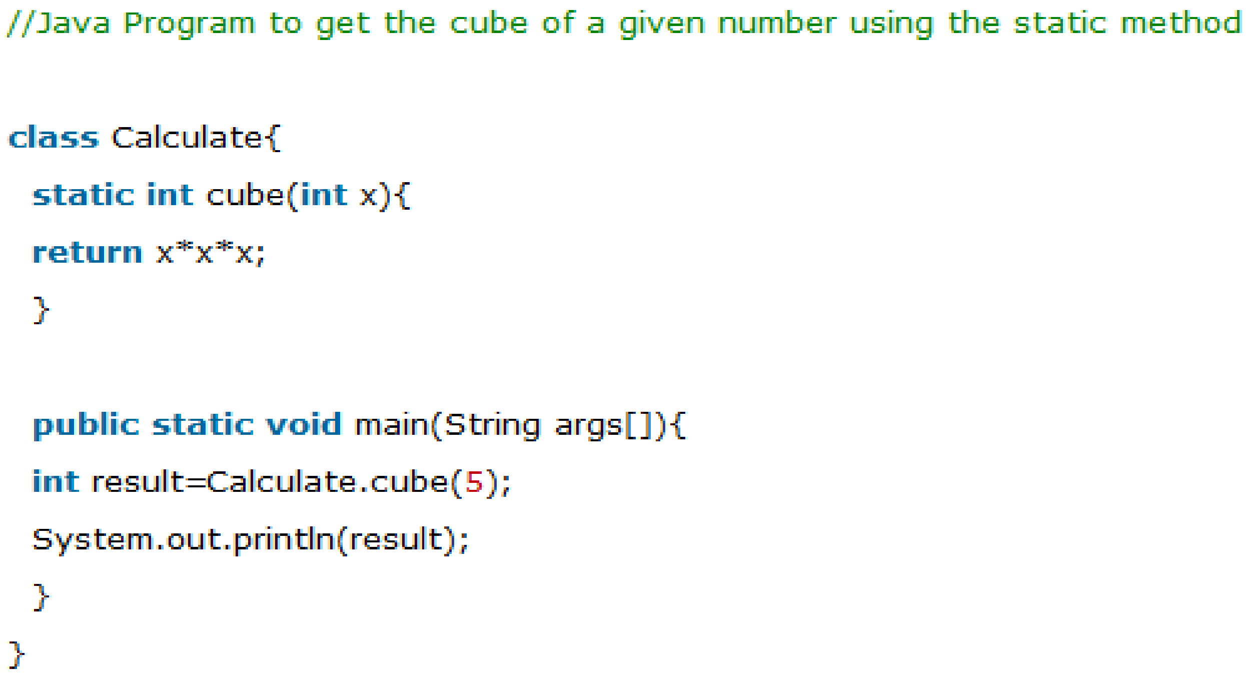


```
1
2
3
```

2) Java static method

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

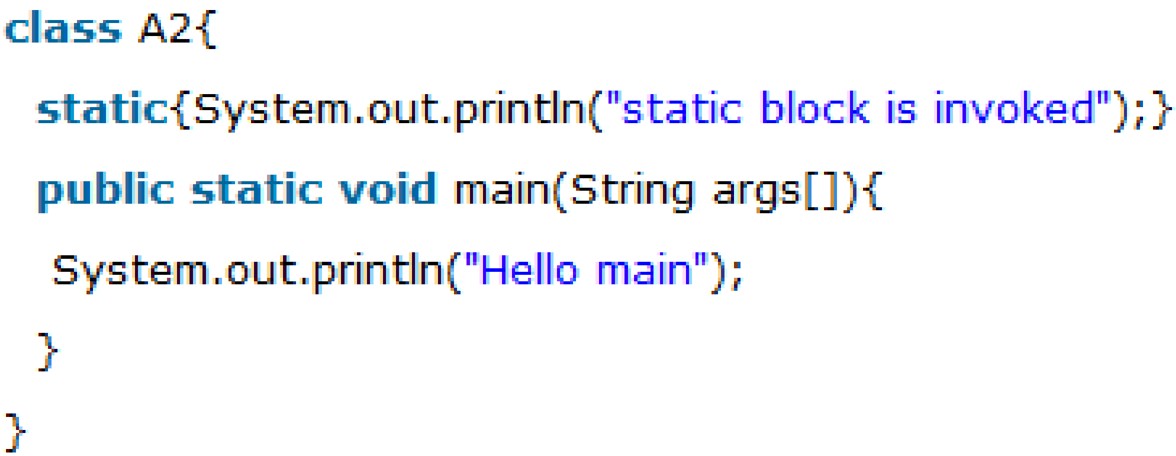
1. A static method belongs to the class rather than the object of a class.
2. A static method can be invoked without the need for creating an instance of a class.
3. A static method can access static data member and can change the value of it.



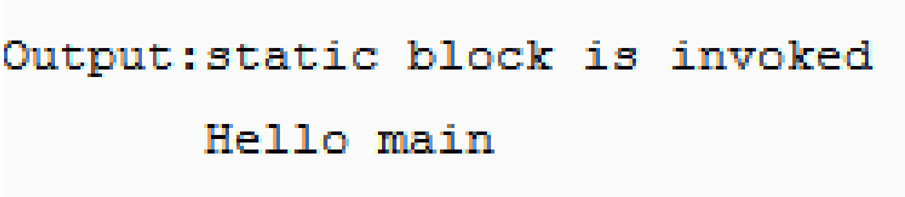
```
//Java Program to get the cube of a given number using the static method

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

  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
```

3) Java static block

1. Is used to initialize the static data member.
2. It is executed before the main method at the time of class loading.



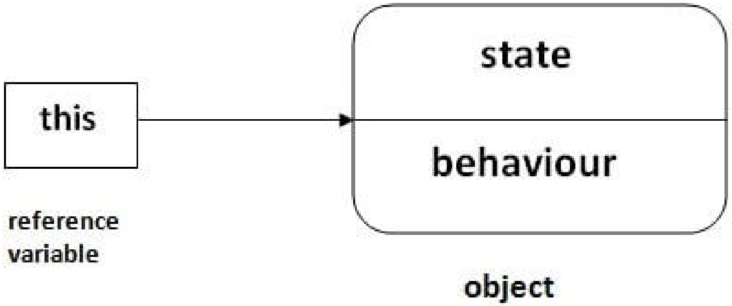
```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```



```
Output:static block is invoked
       Hello main
```

this keyword in java

1. **'this'** is a **reference variable** that refers to the current object



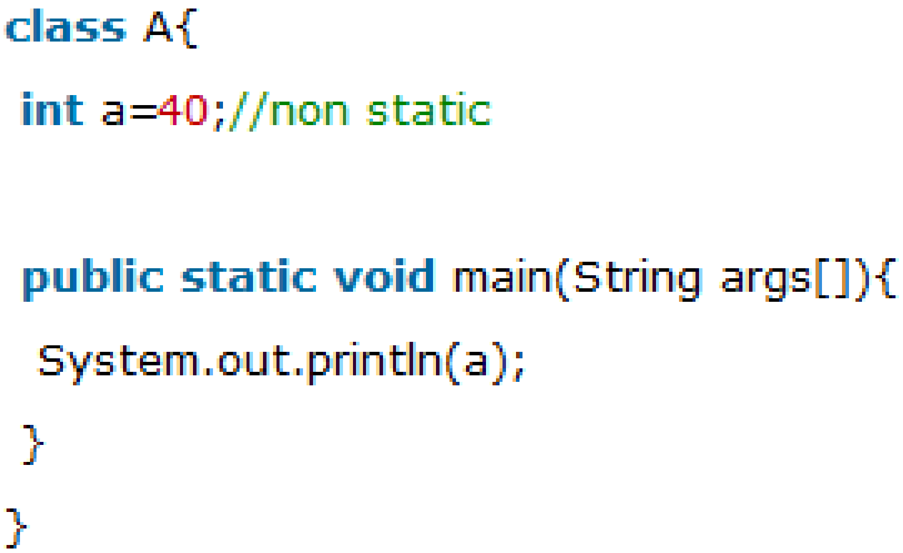
2. this can be used to refer current **class instance variable**.
3. this can be used to invoke **current class method** (implicitly)
4. this() can be used to invoke **current class constructor**.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

2) Java static method

There are two main restrictions for the static method. They are:

The static method can not use non static data member or call non-static method directly.

this and super cannot be used in static context.

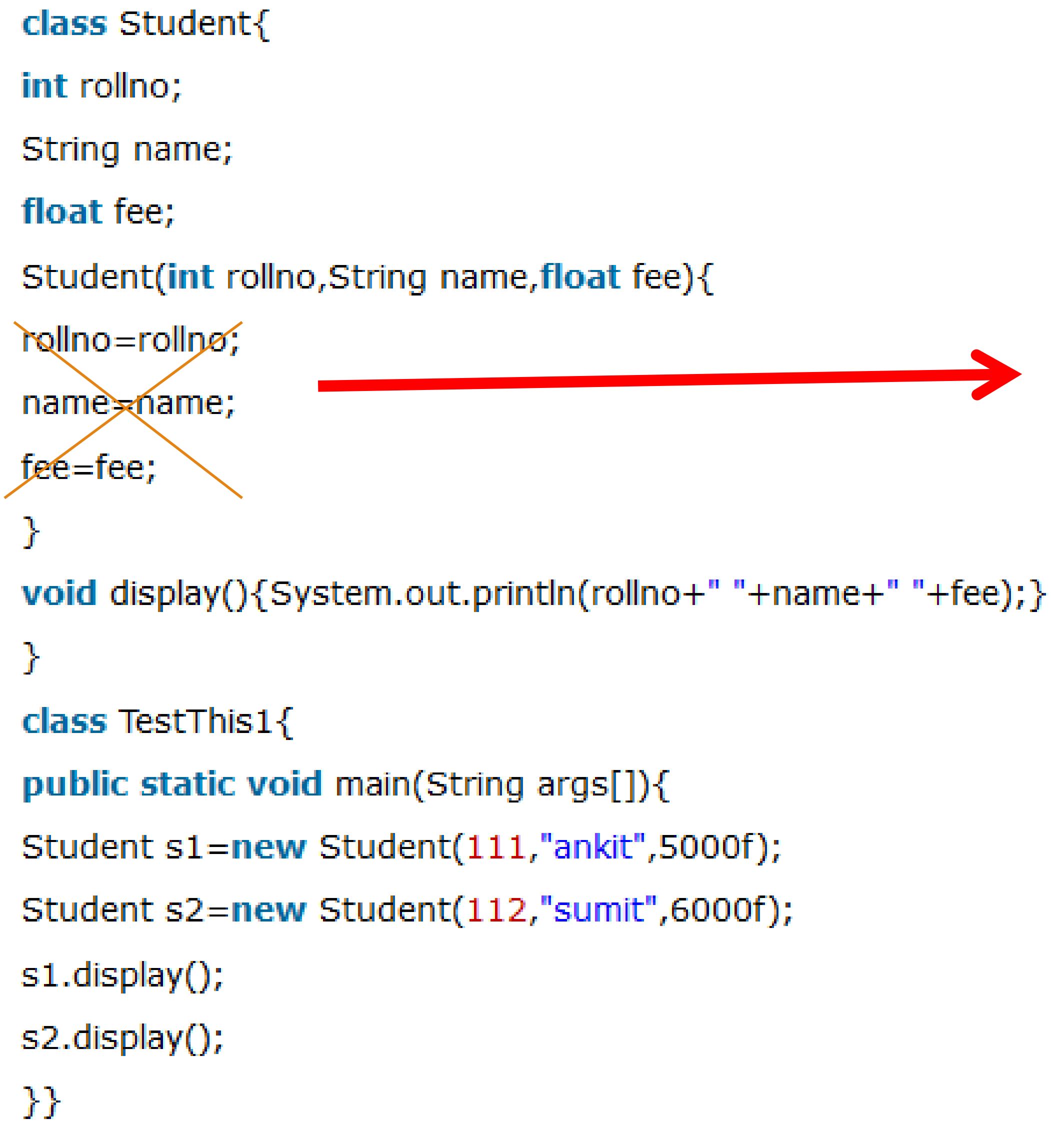


```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
 }
}
```

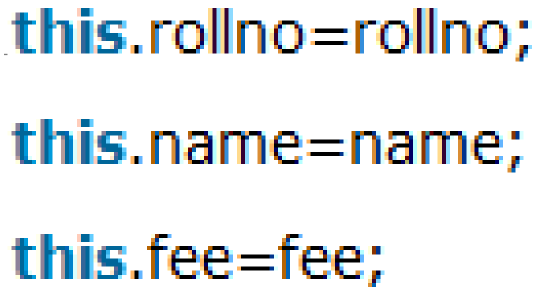
```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
0 null 0.0
0 null 0.0
```

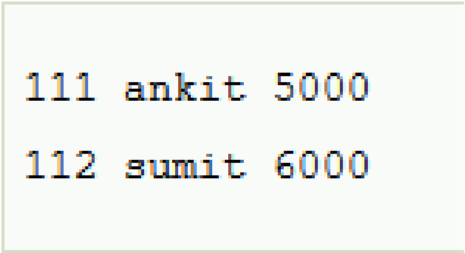


```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
~~rollno=rollno;~~
~~name=name;~~
~~fee=fee;~~
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```



```
this.rollno=rollno;
this.name=name;
this.fee=fee;
```

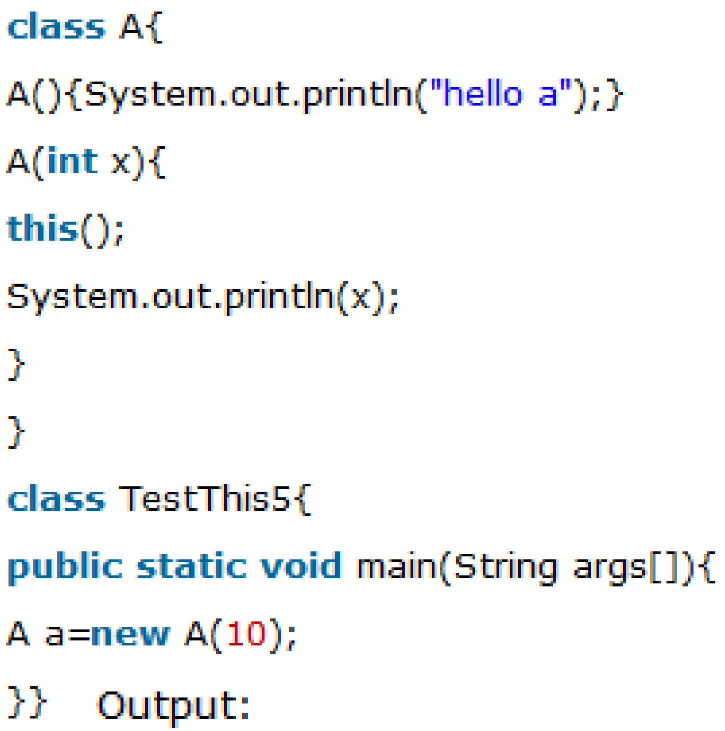
Output:



```
111 ankit 5000
112 sumit 6000
```

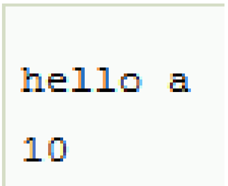
this keyword in java

Calling Default constructor



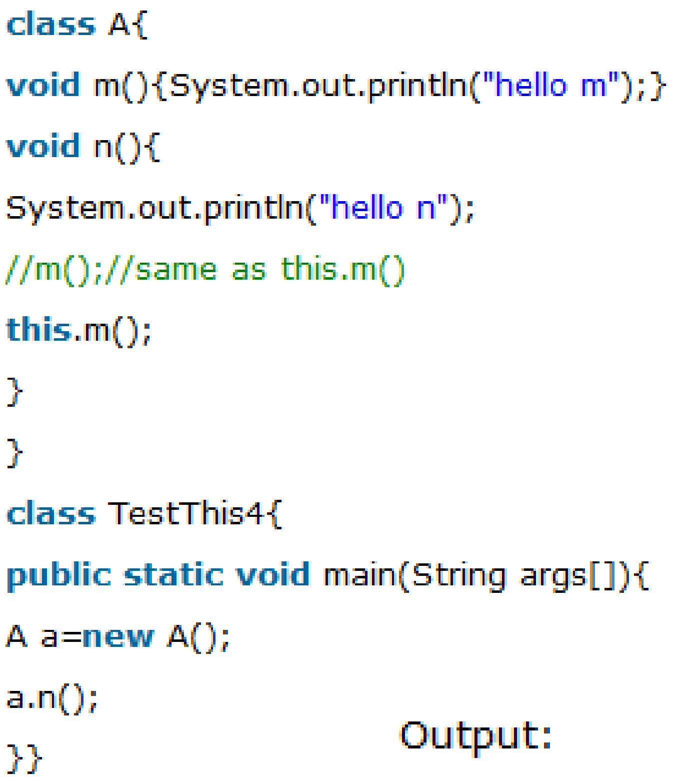
```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```

Output:



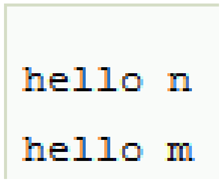
```
hello a
10
```

Calling current class method

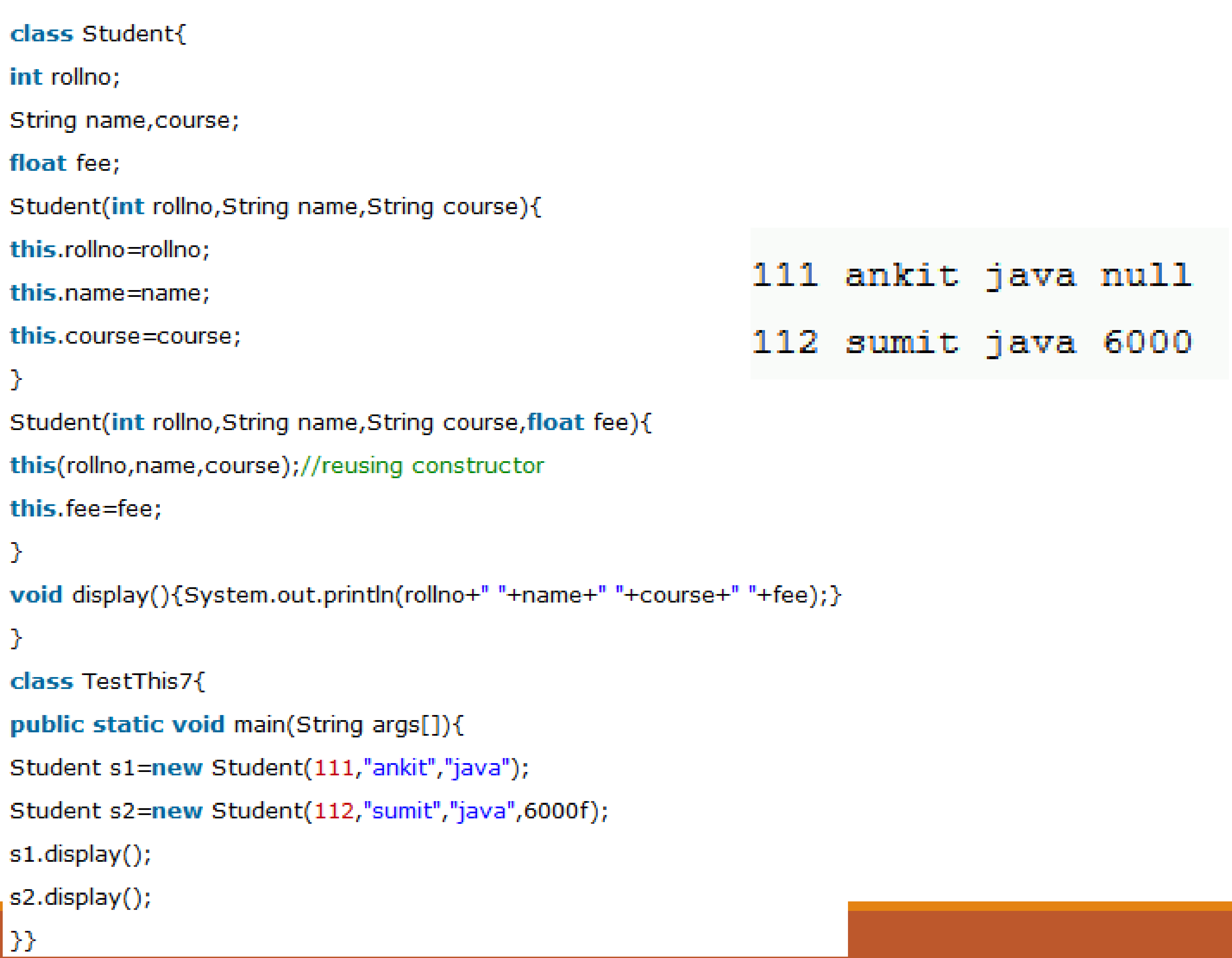


```
class A{
void m(){System.out.println("hello m");}
void n(){
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();
}}
```

Output:



```
hello n
hello m
```



```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this(rollno,name,course);//reusing constructor
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis7{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

```
111 ankit java null
112 sumit java 6000
```

Inheritance in Java

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

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

Inheritance in Java

- **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:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

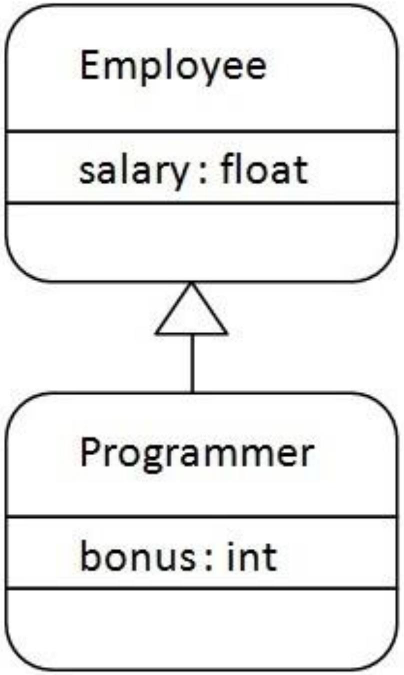
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Inheritance in Java

Programmer is the **subclass** and Employee is the superclass.

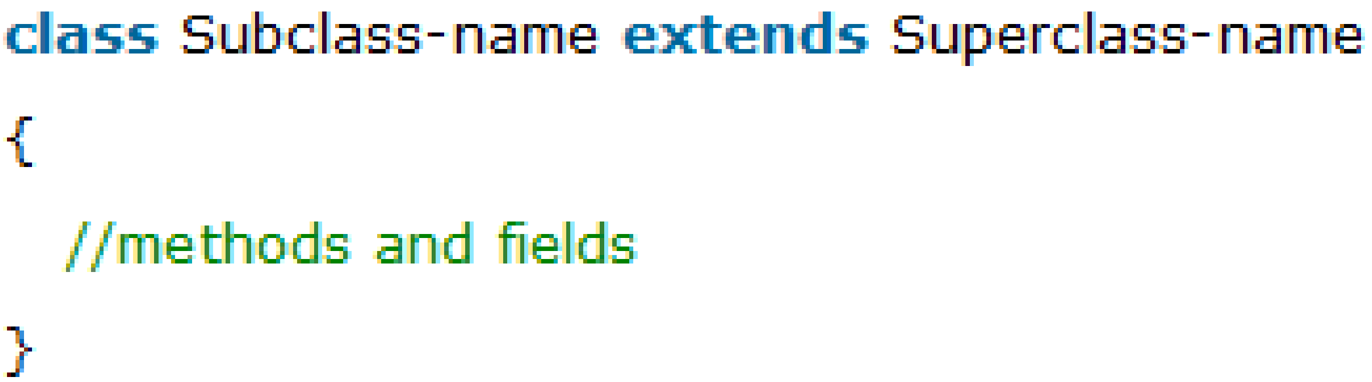
The relationship between the two classes is Programmer IS-A Employee.

It means that Programmer is a type of Employee.



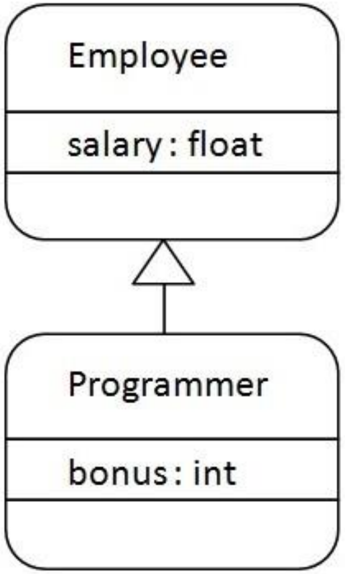
Inheritance in Java

The syntax of Java Inheritance

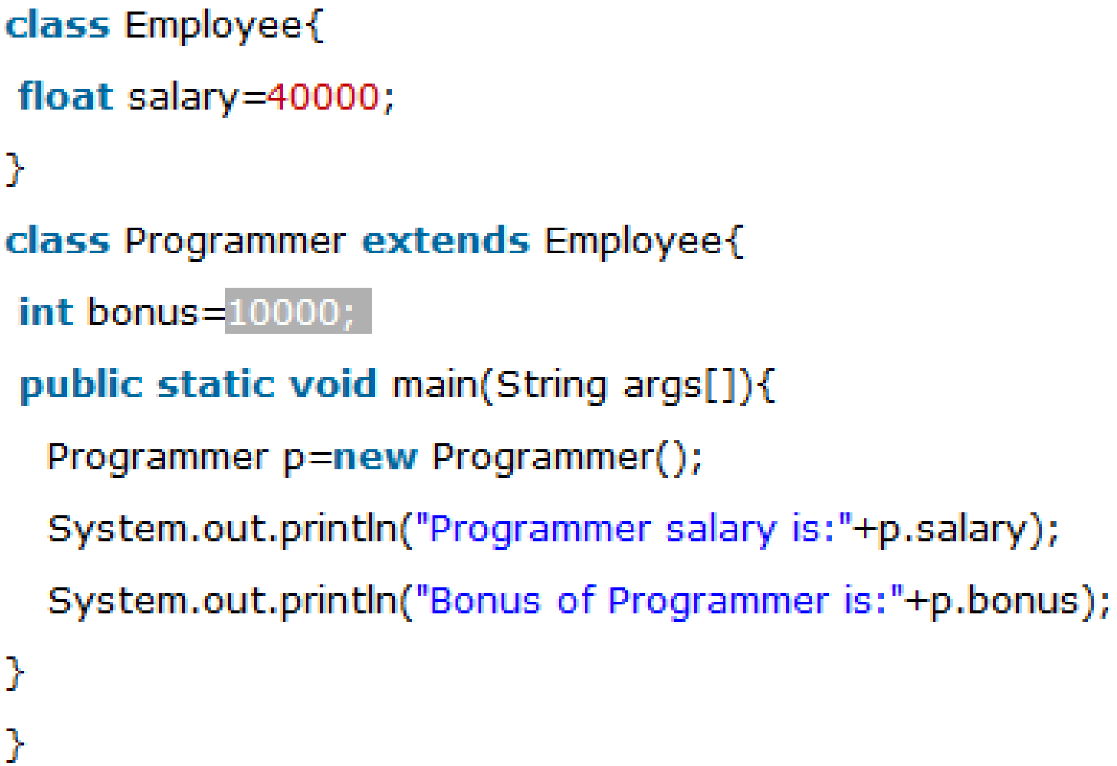


```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

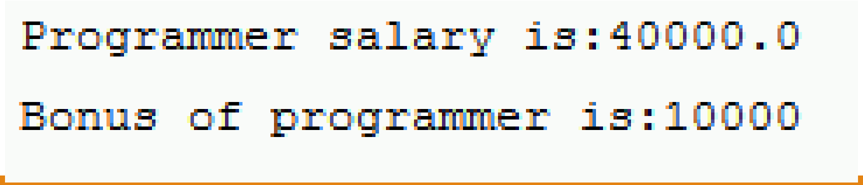
Inheritance in Java



Programmer object can access the field of **own** class as well as of **Employee class** i.e. code reusability.

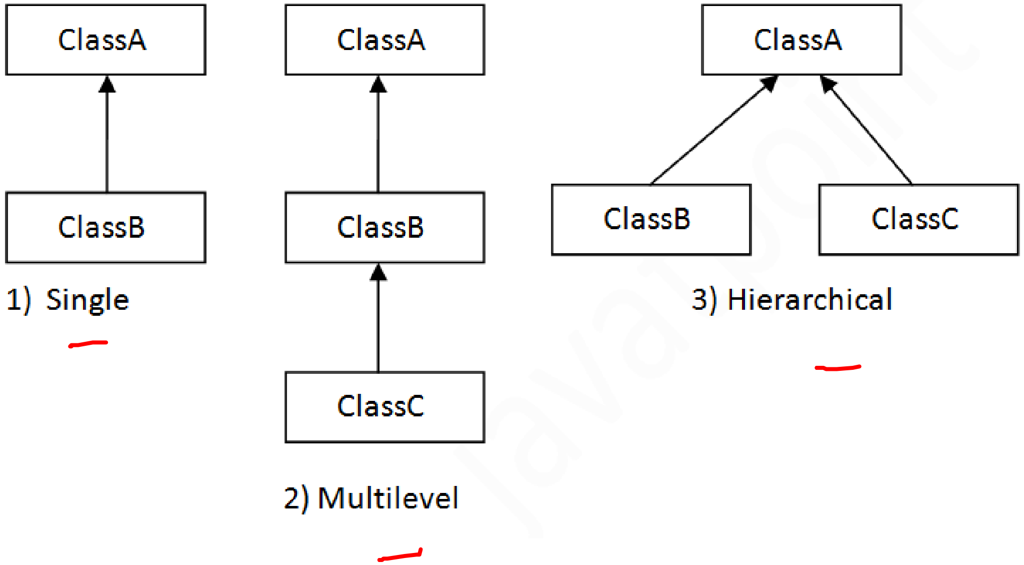


```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```



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

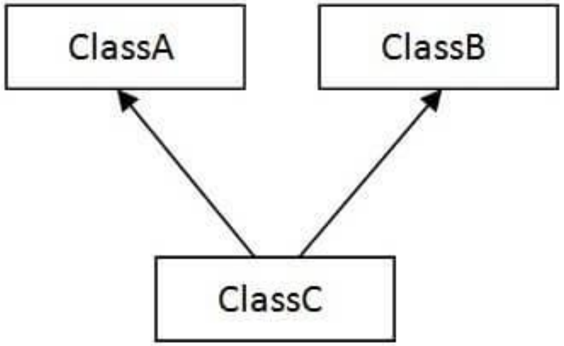
Inheritance in Java



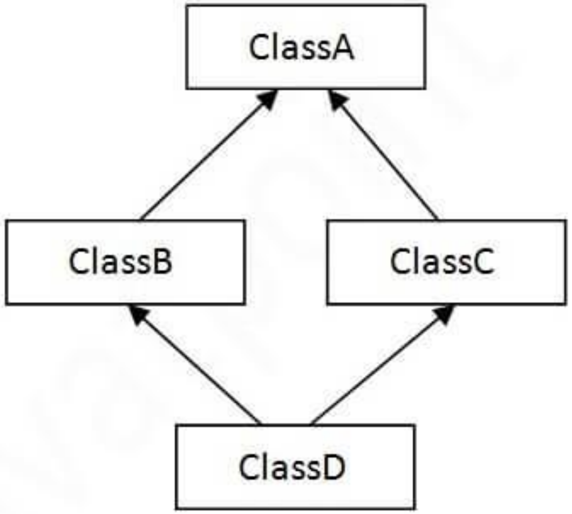
Supported in JAVA

Inheritance in Java

Not Supported in JAVA

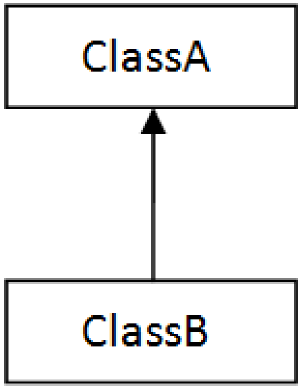


4) Multiple

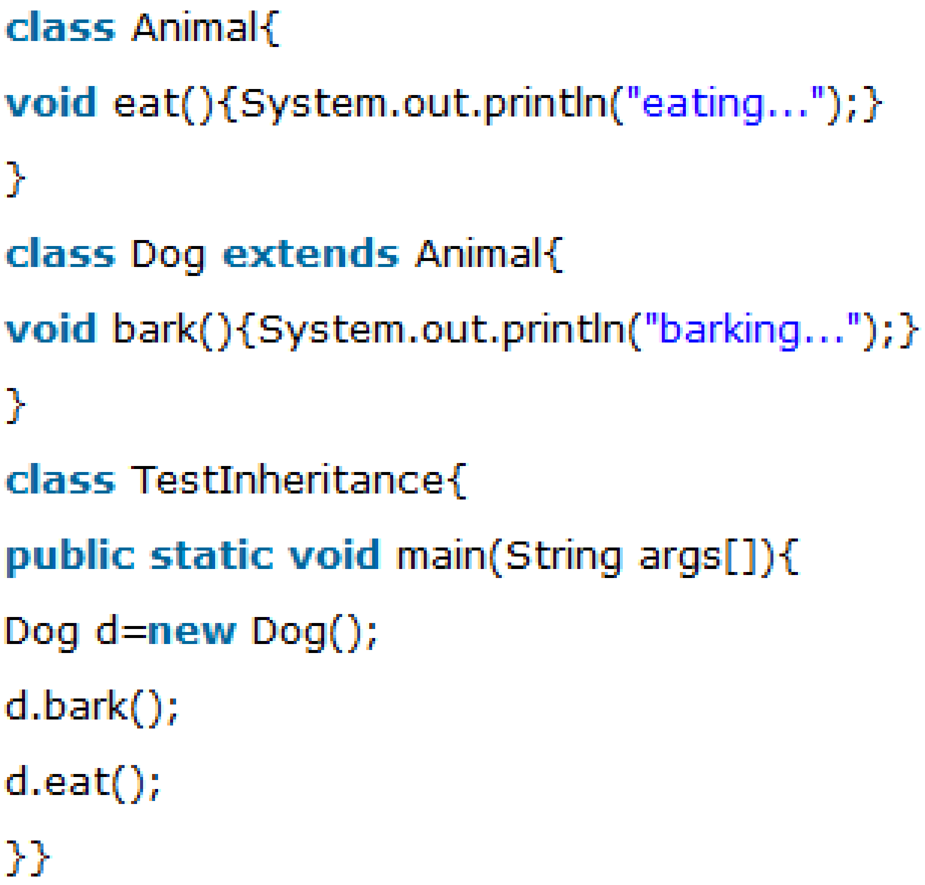


5) Hybrid

Inheritance in Java Single Level



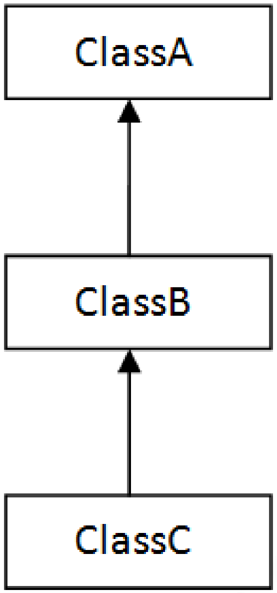
1) Single



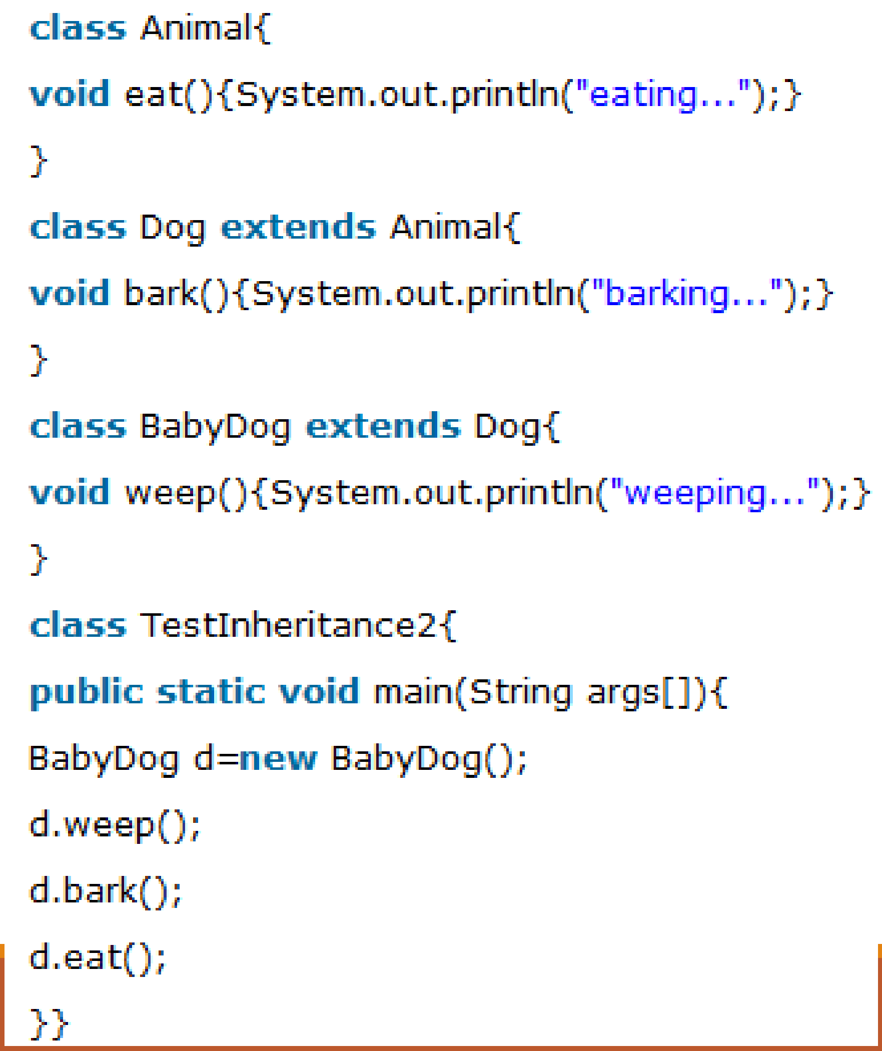
```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

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

Inheritance in Java: Multilevel Level



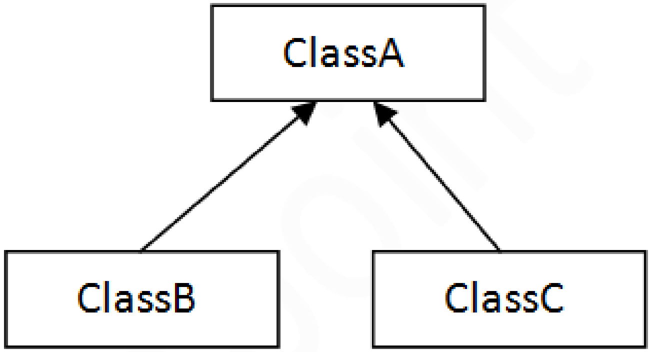
2) Multilevel



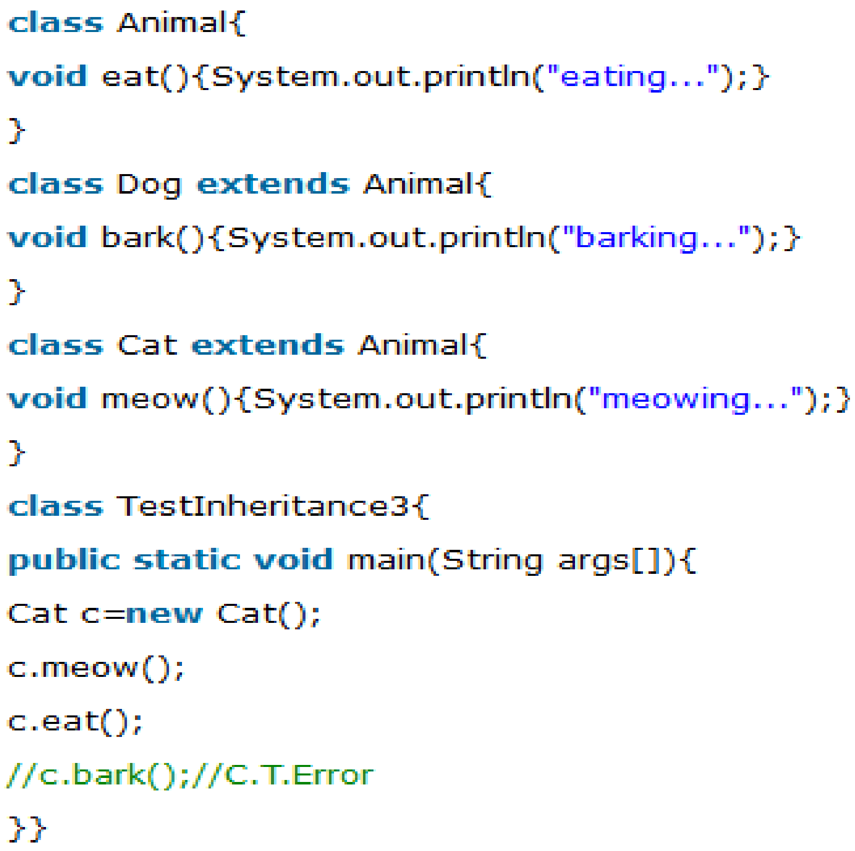
```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

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

Inheritance in Java Hierarchical



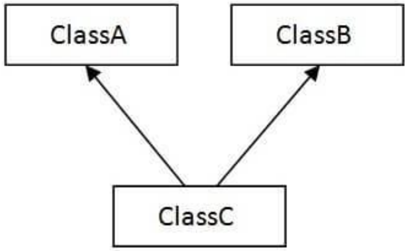
3) Hierarchical



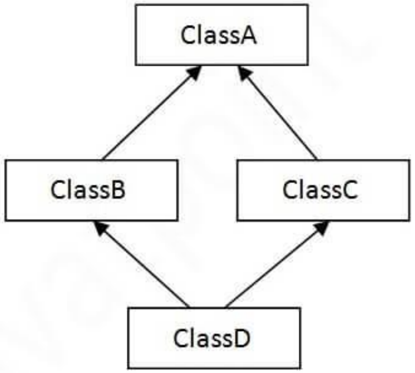
```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();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

```
meowing...
eating...
```

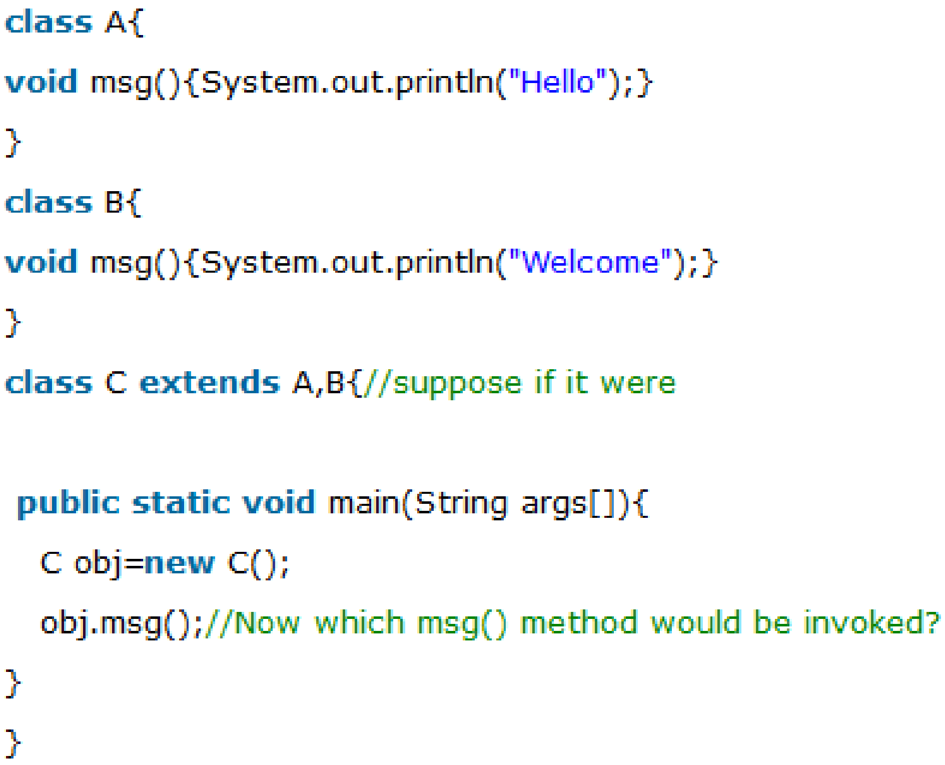
Why multiple inheritance is not supported in java?



4) Multiple



5) Hybrid



```
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?
}
}
```

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**.

Super Keyword in Java

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

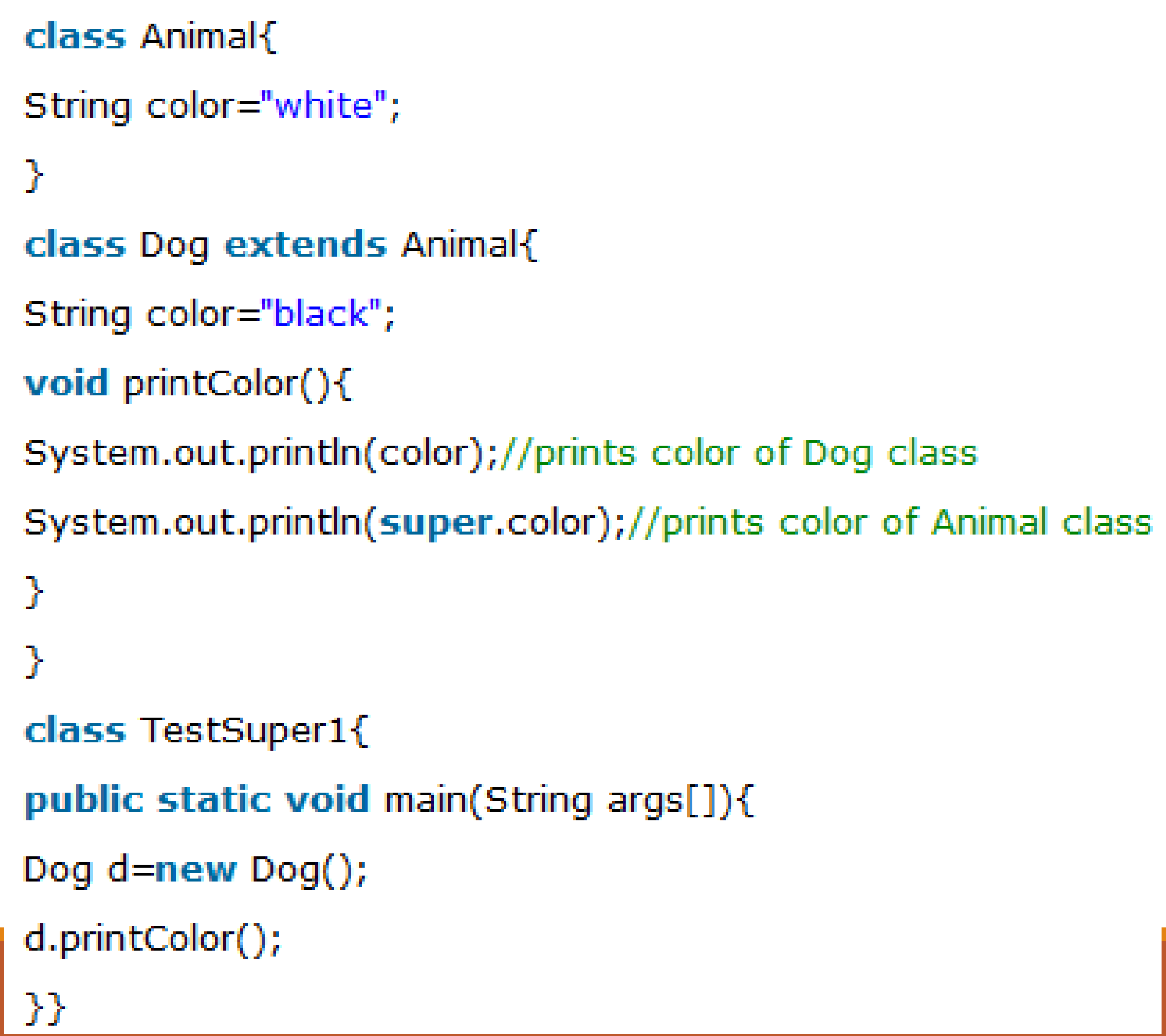
Usage of Java super Keyword

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.

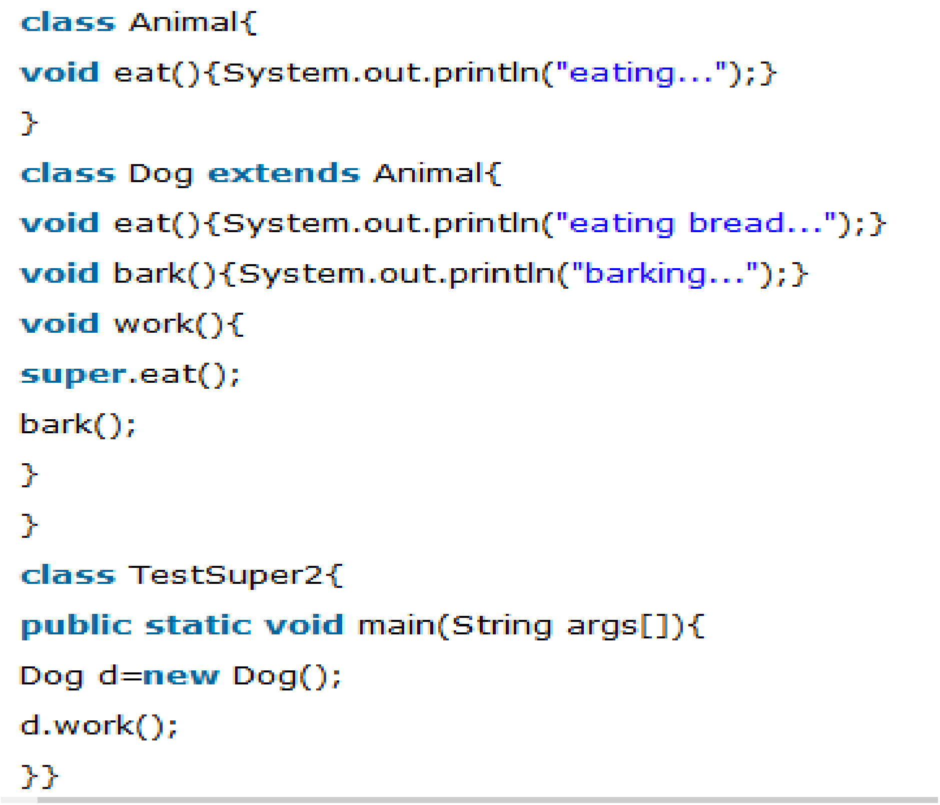
Super Keyword : Parent class instance variable



```
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();
}}
```

black
white

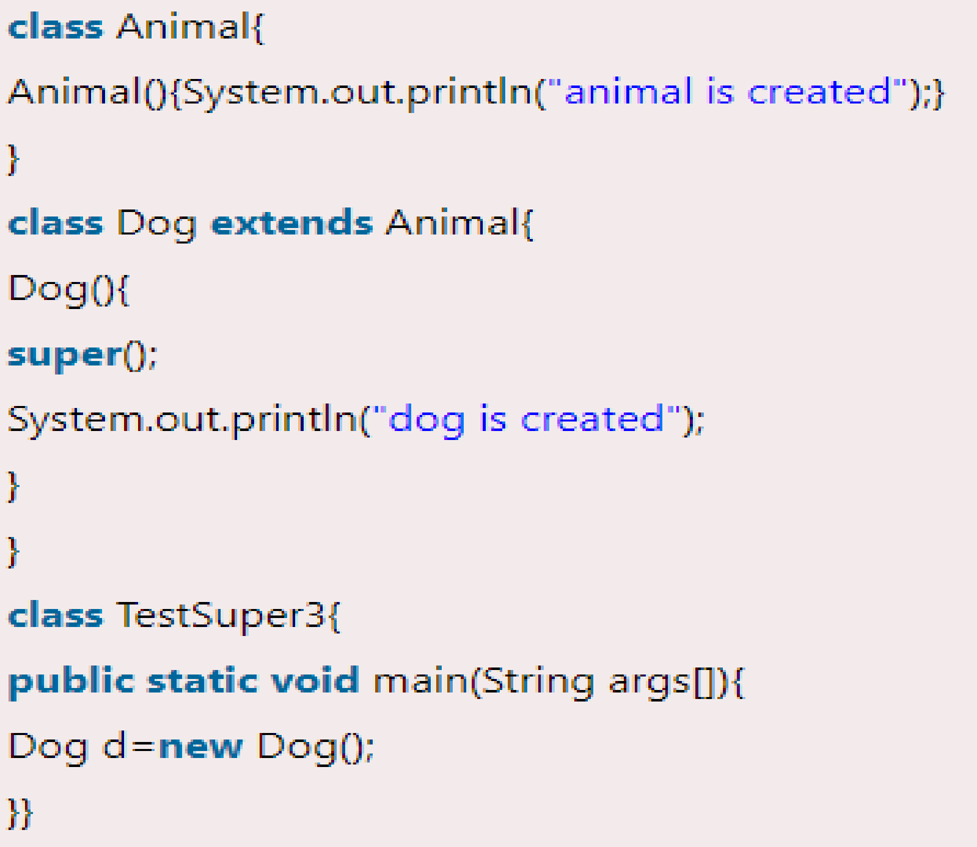
Super Keyword in Java



```
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();
}}
```

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

Super Keyword in Java

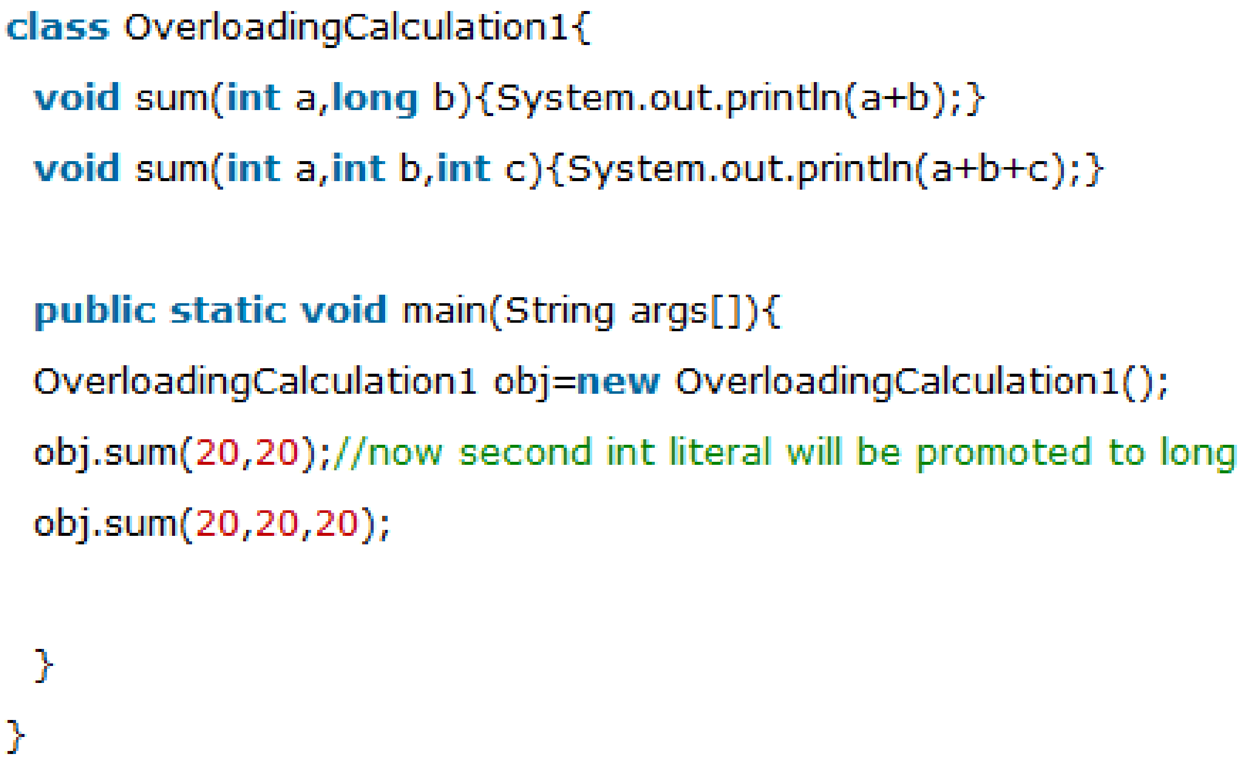


```java
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

```
animal is created
dog is created
```

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found.

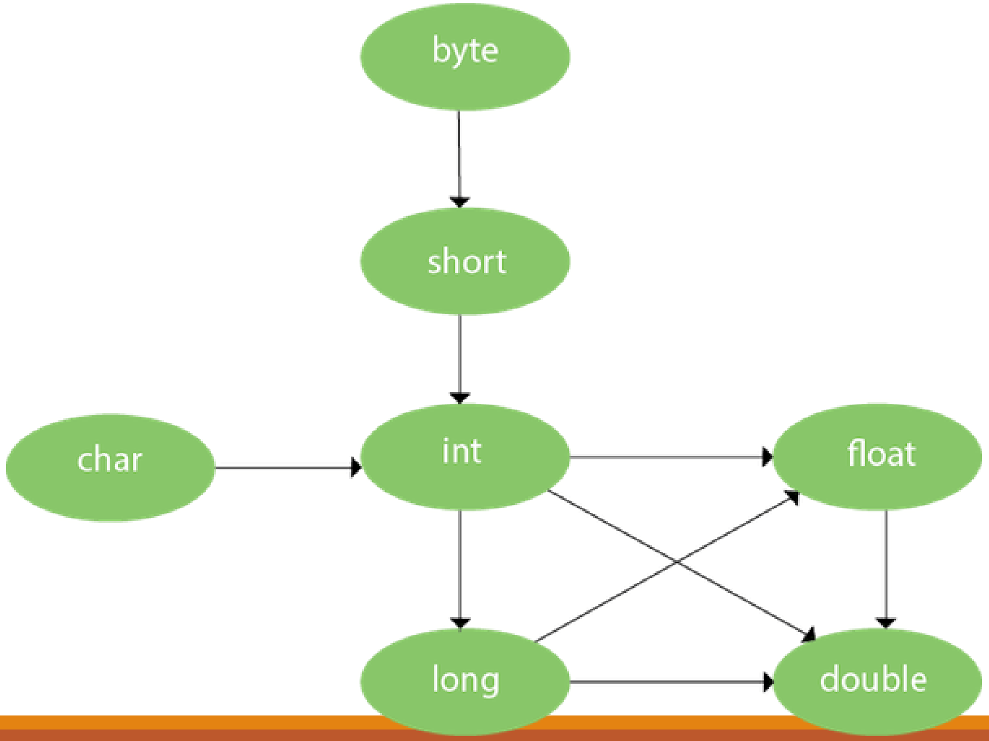


```
class OverloadingCalculation1{
  void sum(int a,long b){System.out.println(a+b);}
  void sum(int a,int b,int c){System.out.println(a+b+c);}

  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);


  }
}
```

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found.

byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double.



Final Keyword In Java

The **final keyword** in java is used to restrict the user.

Final can be:

1. Variable:

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

Click to add text

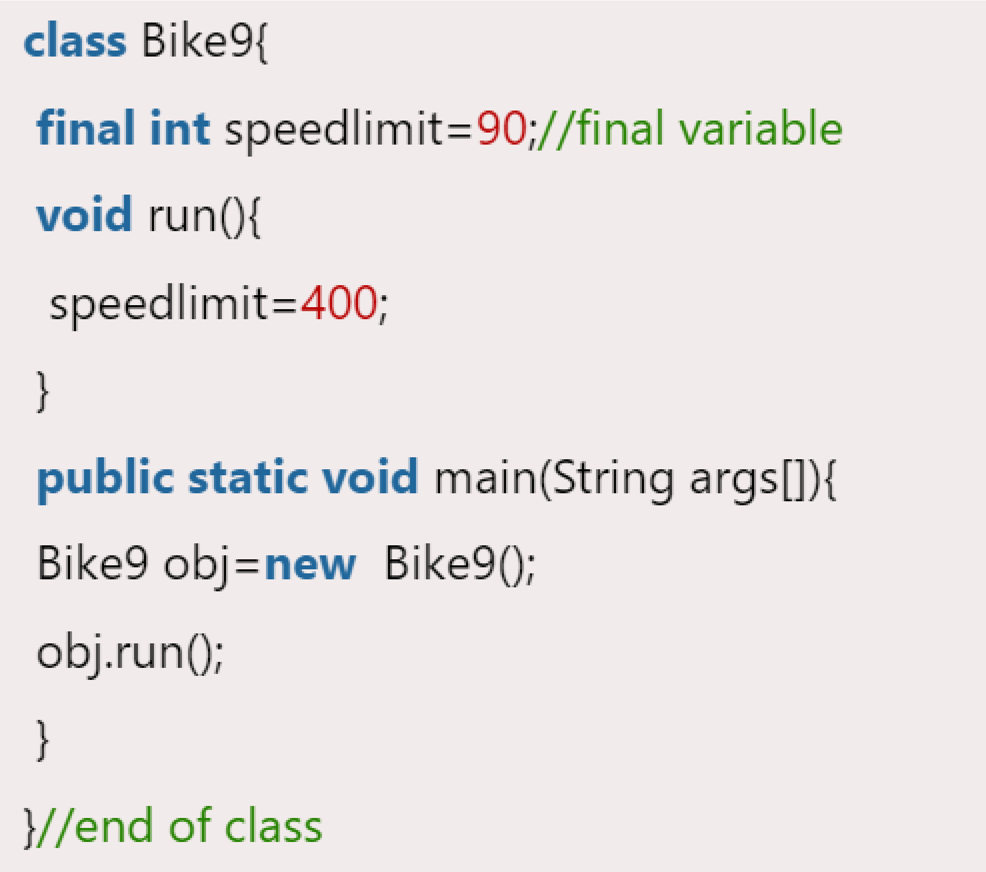
2. Method:

    If you make any method as final, you cannot override it. A final method can be inherited but cannot override.

3. Class:

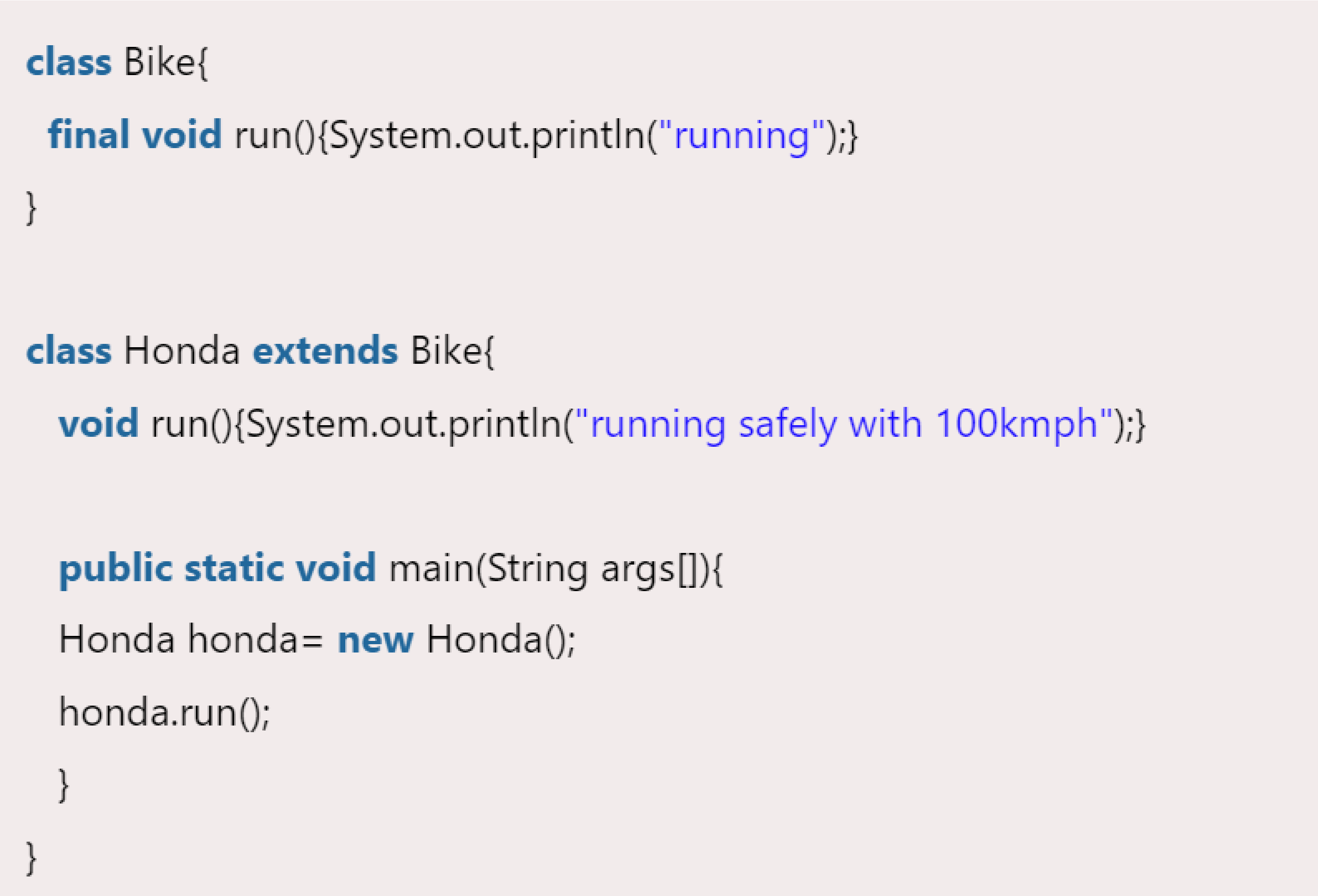
    If you make any class as final, you cannot extend it.

Final Variable: Output will be compile time error



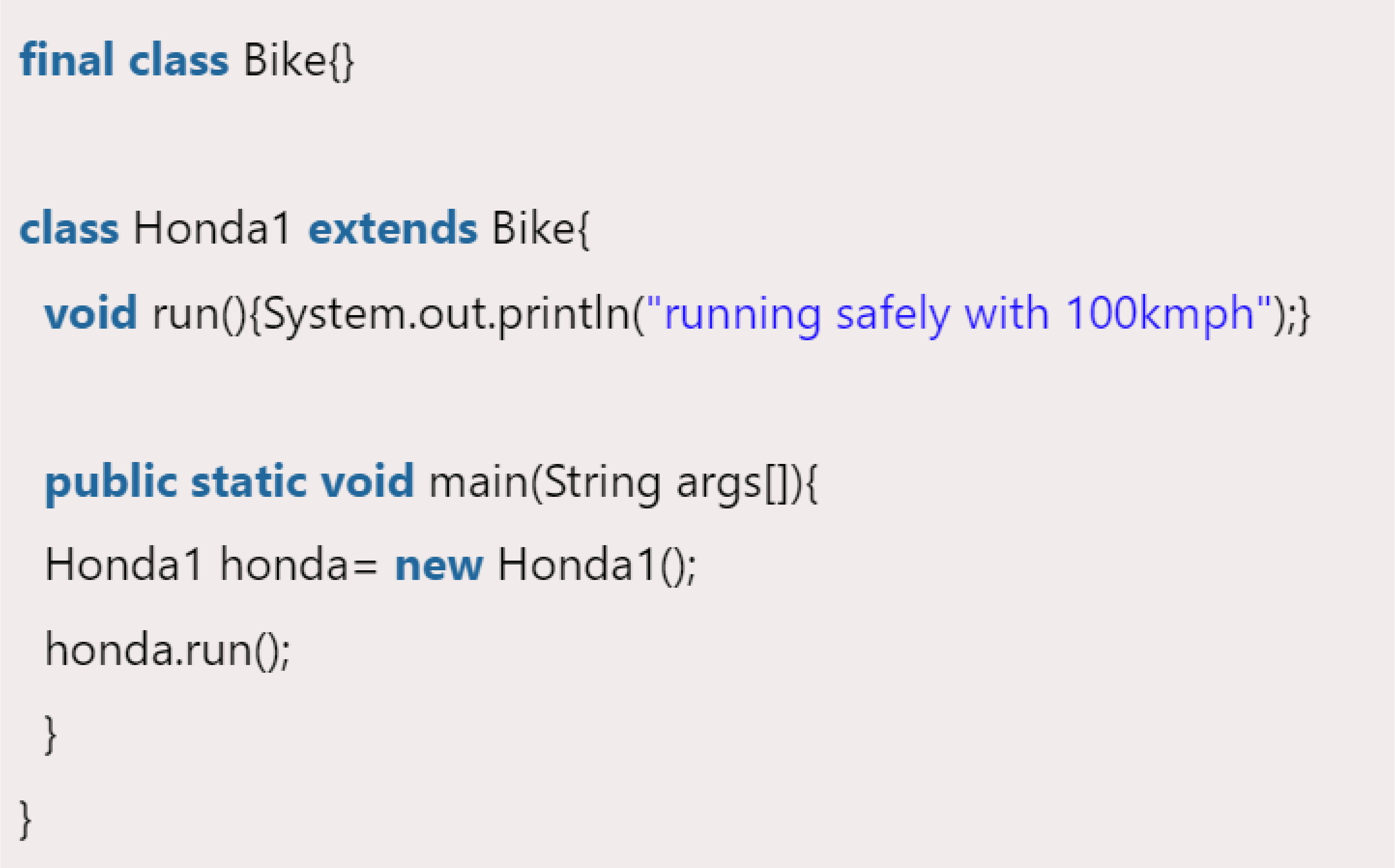
```java
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Final Method : Output will be compile time error



```
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();
  }
}
```

Final Class : Output will be compile time error



```
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();
 }
}
```