JAVA PROGRAMMING

Chapter 2

Classes and Objects

Chaskar R. R.

OBJECT

- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.
- An object has three characteristics:
- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object.
- **Identity:** An object identity is typically implemented via a unique ID.

CLASS

- A class is a collecton of objects which have common properties.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

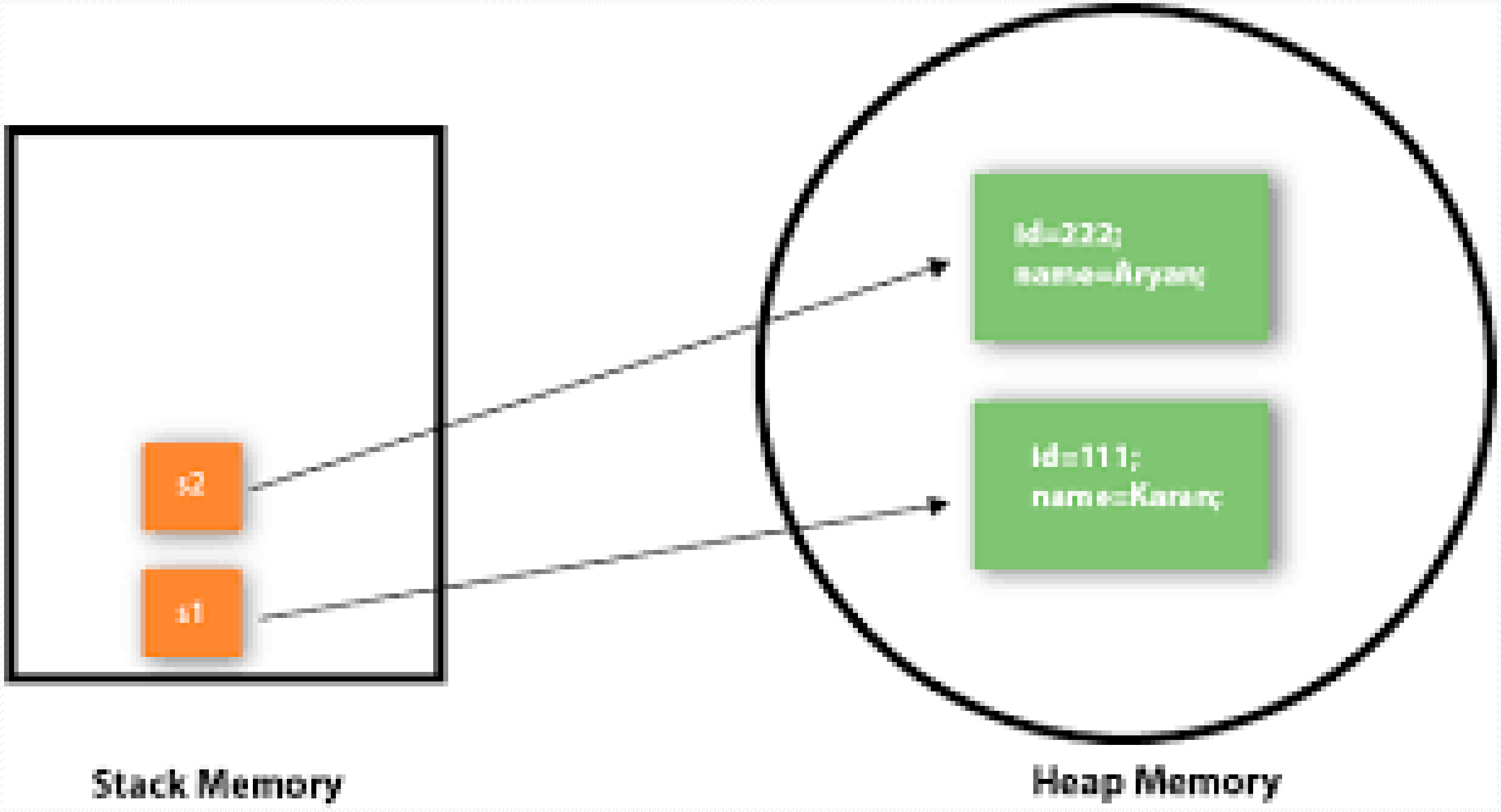
Syntax

```
class <class_name>
{
    field;
    method;
}
```

- Eg

```
class Student
{
 int id;
 String name;
}
class TestStudent2
{
 public static void main(String args[])
{
  Student s1=new Student();
  s1.id=101;
  s1.name=“ABC";
  System.out.println(s1.id+" "+s1.name);
 }
}
```

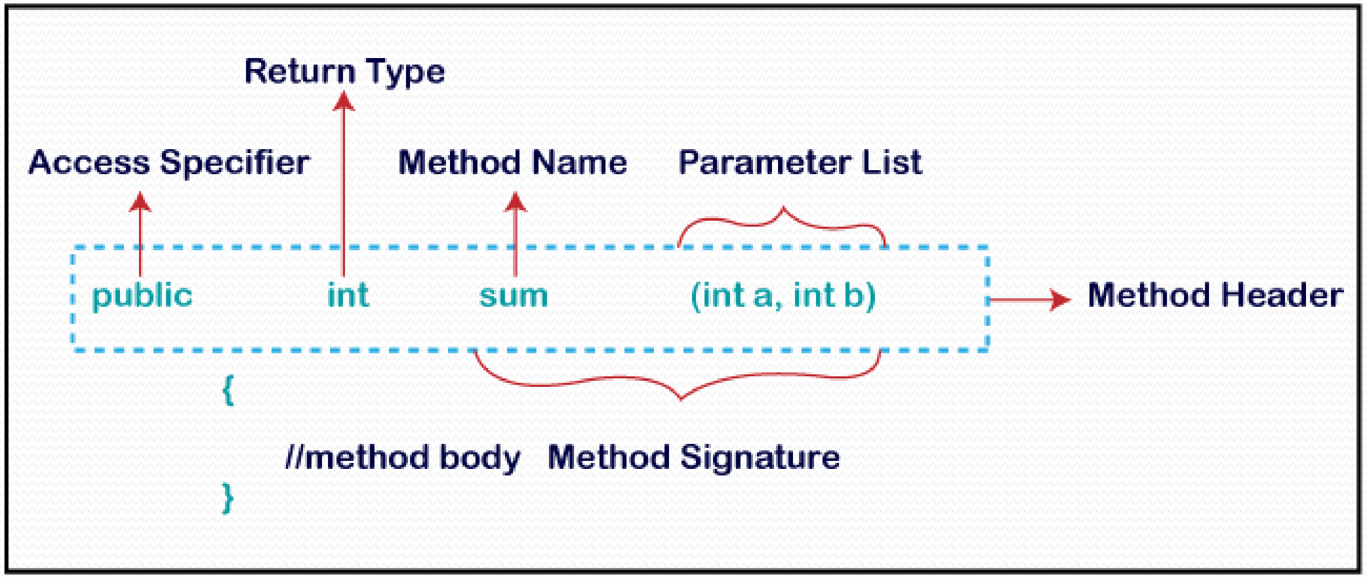
MEMORY ALLOCATION FOR OBJECTS



Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.

Method Declaration



Access Specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Types of Method

There are two types of methods in Java:

- Predefined Method-

length(), equals(), compareTo(), sqrt()

- User-defined Method

create(), display(), show()

CONSTRUCTOR

- In Java, a constructor is a block of codes similar to the method.
- ***Types of constructors***

1. Default constructor :-

A constructor is called "Default Constructor" when it doesn't have any parameter.

Example:-

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

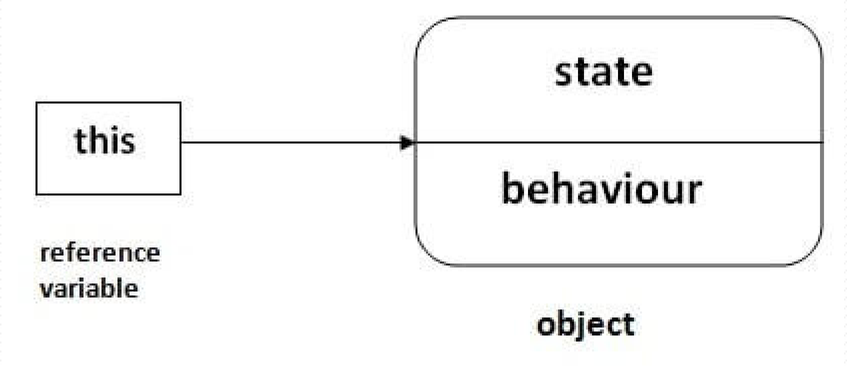
Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

```
class Student
{
    int id;
    String name;        //creating a parameterized constructor
    Student4(int i,String n)
{
    id = i;
    name = n;
 }
    void display()
{
System.out.println(id+" "+name);
}
    public static void main(String args[])
{
    Student s1 = new Student(111,"Karan");   //creating objects and passing values
    Student s2 = new Student(222,"Aryan");
        s1.display();
        s2.display();
}  }
```

this keyword

In java, this is a **reference variable** that refers to the current object.



Usage of java this keyword

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.

- If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Program

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

Inheritance

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

Terms used in Inheritance

Class: A class is a group of objects which have common properties.

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.

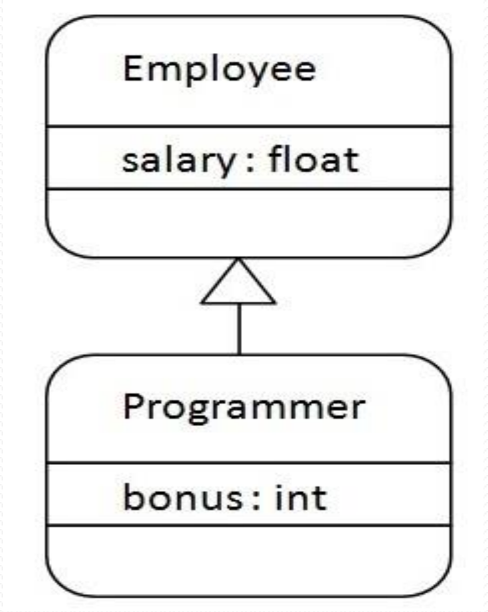
Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

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.

Syntax

```
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.

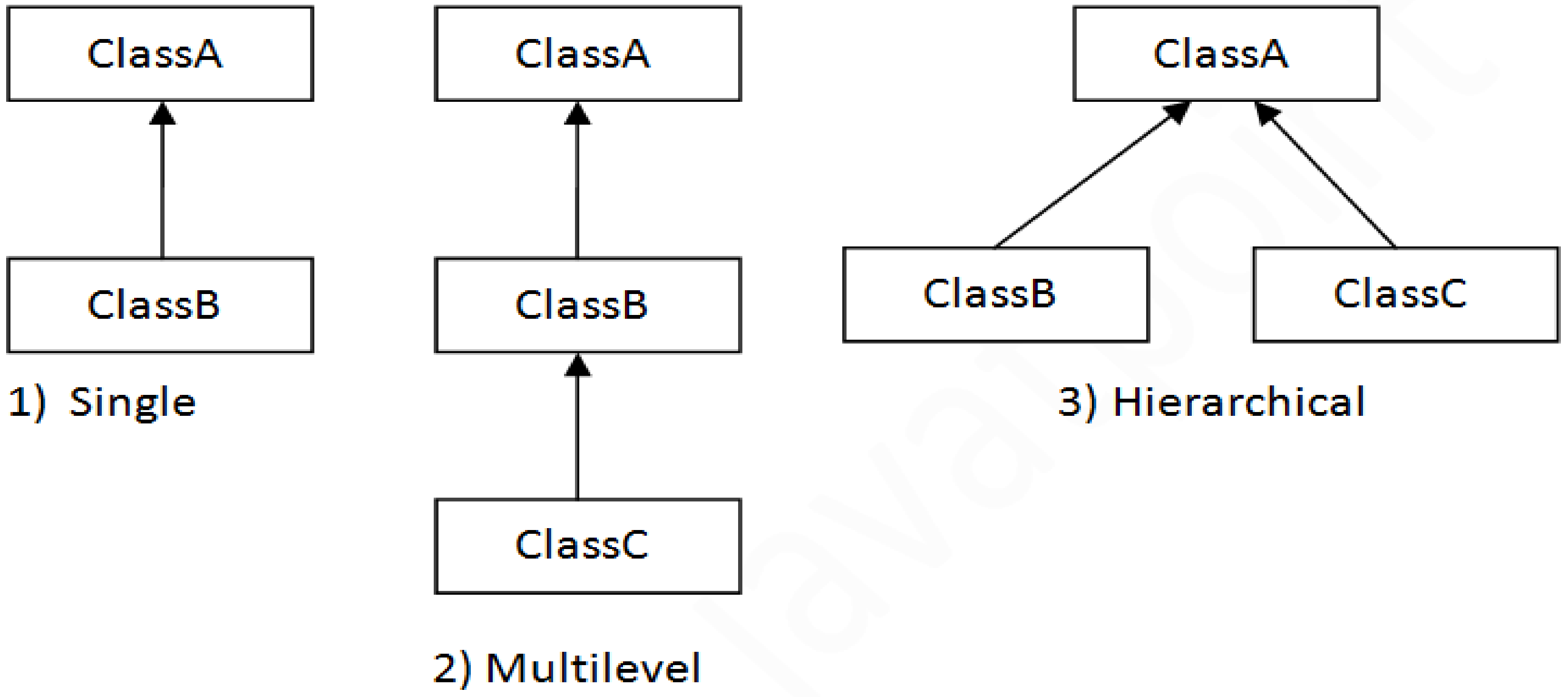


Program

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

Types of inheritance

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



Single Inheritance

When a class inherits another class, it is known as a *single inheritance*.

Program

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

Multilevel Inheritance

When there is a chain of inheritance, it is known as *multilevel inheritance*.

Program

```
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();
}}
```

Hierarchical Inheritance

When two or more classes inherits a single class, it is known as *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
}}
```

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.

Abstract class

- A class which is declared with the abstract keyword is known as an abstract class in Java.
- A class which is declared with the abstract keyword is known as an abstract class in Java.

RULES

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

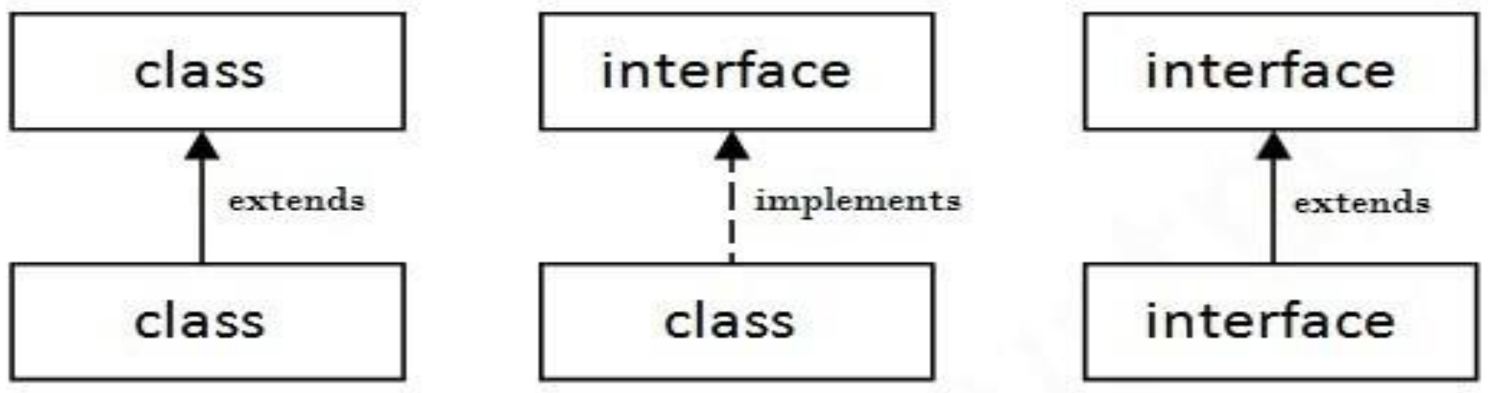
PROGRAM

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

Interface in Java

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body.
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

The relationship between classes and interfaces



```
interface Drawable
{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable
{
public void draw()
{
System.out.println("drawing rectangle");
} }
class Circle implements Drawable
{
public void draw()
{
System.out.println("drawing circle");
} }
//Using interface: by third user
class TestInterface1
{
public static void main(String args[])
{
Drawable d=new Circle();
//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

Polymorphism

- polymorphism means many forms.
- here are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

Runtime Polymorphism

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

PROGRAM

```
class Bike
{
  void run()
{
System.out.println("running");
}  }
class Splendor extends Bike
{
  void run()
{
System.out.println("running safely with 60km");
}
  public static void main(String args[])
{
    Bike b = new Splendor();//upcasting
    b.run();
  } }
```