

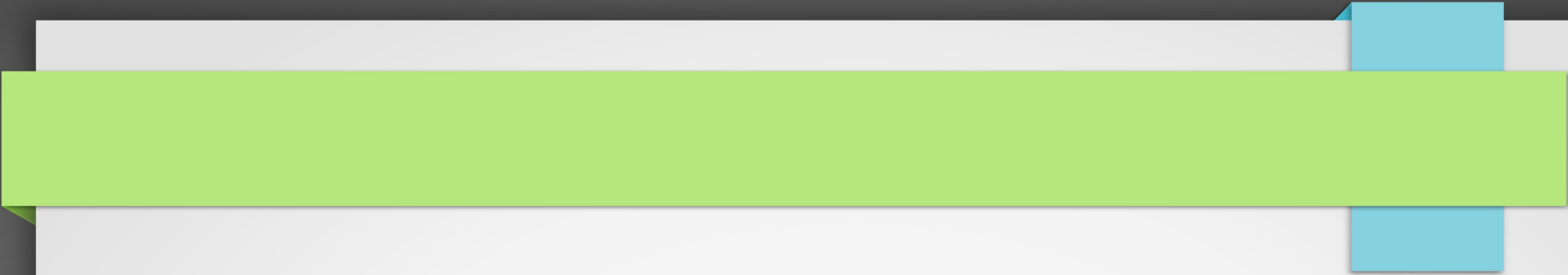
# **INTRODUCTION**

**210301301**

**CORE JAVA**

**210301305**

**PRACTICAL ON CORE JAVA**



# **Unit – III**

## **Inheritance, Interface, Package**

# Introduction

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

# Introduction

- Why use inheritance in java
  - For Method Overriding
  - For Code Reusability.

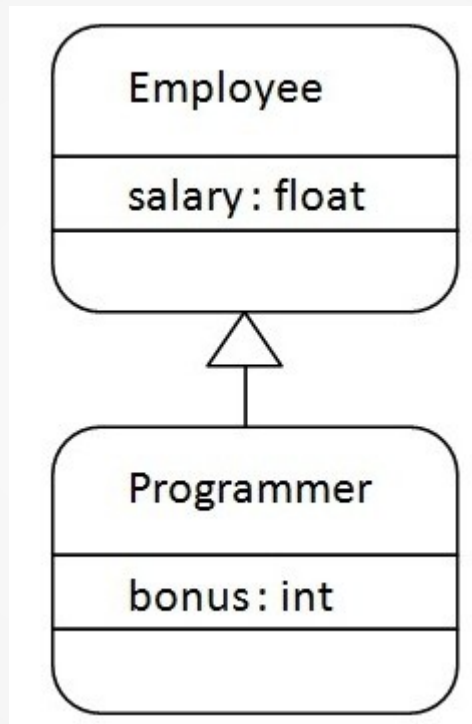
## Syntax of Java Inheritance

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

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

In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

# Introduction

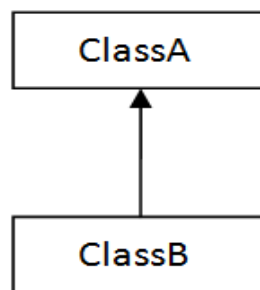


As displayed in the above figure, Programmer is the subclass and Employee is the superclass. **Relationship between two classes is Programmer IS-A Employee.**

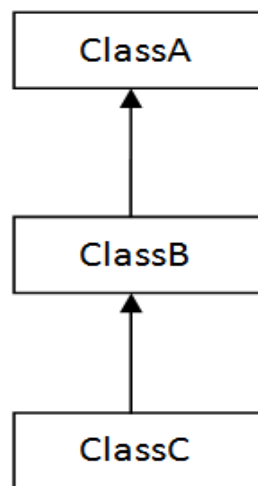
It means that Programmer is a type of Employee.

# Types of Inheritance

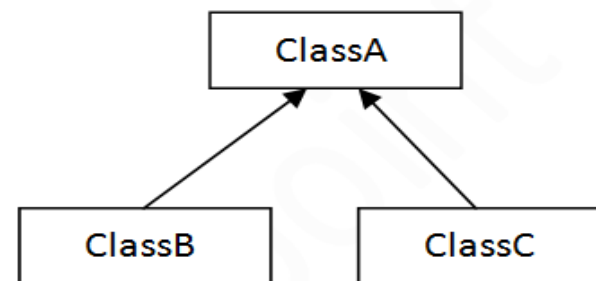
- On the basis of class, there can be **three types of inheritance in java**: single, multilevel and hierarchical.
- In java programming, multiple and hybrid inheritance is supported through interface only.
- **Multiple inheritance is not supported in java through class.**



1) Single

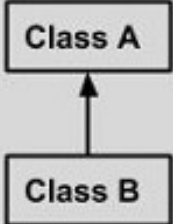
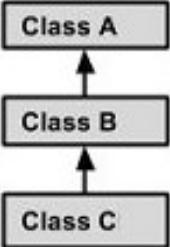
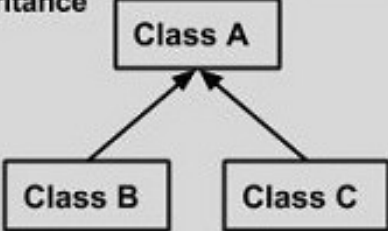
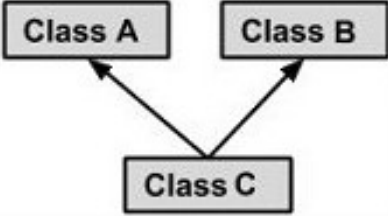


2) Multilevel



3) Hierarchical

# Types of Inheritance

|  |   |
|--|---|
| <b>Single Inheritance</b><br> <pre>graph BT; B[Class B] --&gt; A[Class A]</pre>                             | <pre>public class A {<br/>    .....<br/>}<br/>public class B extends A {<br/>    .....<br/>}</pre>  |
| <b>Multi Level Inheritance</b><br> <pre>graph BT; C[Class C] --&gt; B[Class B]; B --&gt; A[Class A]</pre>   | <pre>public class A { .....}<br/>public class B extends A {.....}<br/>public class C extends B {..... }</pre>   |
| <b>Hierarchical Inheritance</b><br> <pre>graph BT; B[Class B] --&gt; A[Class A]; C[Class C] --&gt; A</pre> | <pre>public class A { .....}<br/>public class B extends A {.....}<br/>public class C extends A {..... }</pre>   |
| <b>Multiple Inheritance</b><br> <pre>graph BT; C[Class C] --&gt; A[Class A]; C --&gt; B[Class B]</pre>    | <pre>public class A { .....}<br/>public class B {.....}<br/>public class C extends A,B {<br/>    .....<br/>} // Java does not support mutiple Inheritance</pre> |

# Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, **multiple inheritance is not supported in java.**
- 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 same method and you call it from child class object, there will be ambiguity to call method of A or B class.
- Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes.
- So whether you have same method or different, there will be compile time error now.

Inheritance5.java



# Method Overriding

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

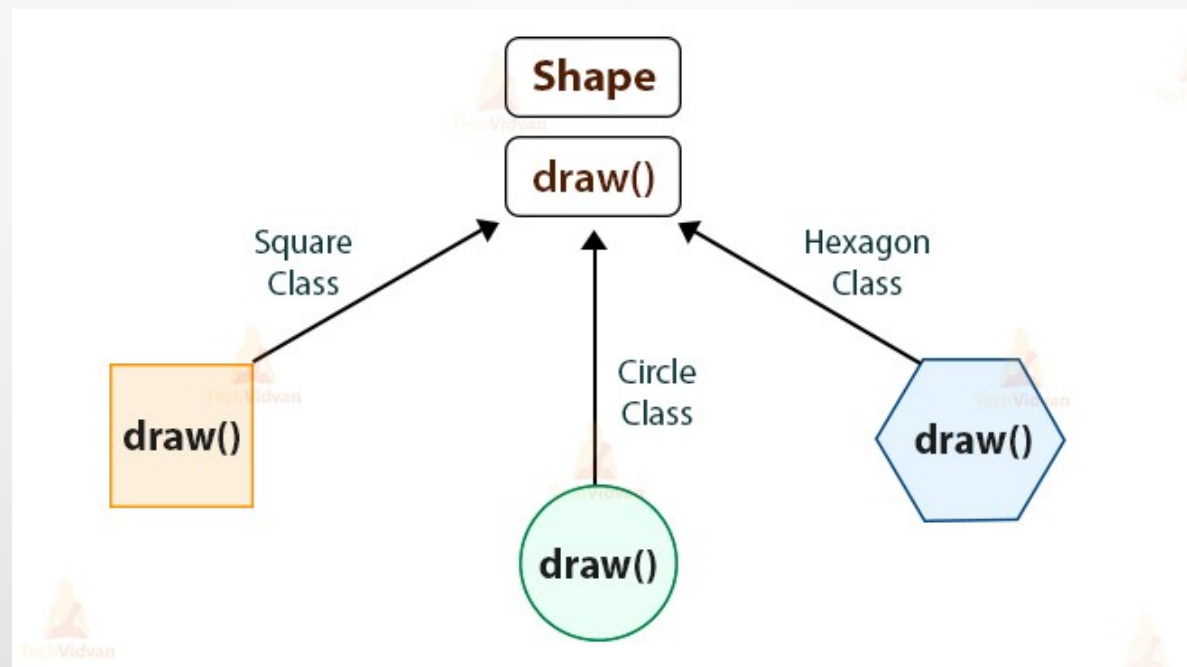
## Usage of Java Method Overriding

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

# Example of Method Overriding

## 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).



# Super keyword in java

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

## Usage of java super Keyword :

- 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() is added in each class constructor automatically by compiler if there is no super() or this().

# Final keyword in java

- The final keyword in java is **used to restrict the user.**
- The java final keyword can be used in many context.
- **Final can be:**
  - **Variable**
  - **Method**
  - **Class**
- The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable.
- **It can be initialized in the constructor only.**
- The blank final variable can be static also which will be initialized in the static block only.

# Final keyword

- Final Variable
  - If you make any variable as final, you cannot change the value of final variable(It will be constant).
- Final method
  - If you make any method as final, you cannot override it.
- Final class
  - If you make any class as final, you cannot extend it.

# Final Keyword

## Java Final Keyword

- ⇒ Stop Value Change
- ⇒ Stop Method Overriding
- ⇒ Stop Inheritance

[javatpoint.com](http://javatpoint.com)

Inheritance14\_1.java

**Final Variable** → To create constant variables

**Final Methods** → Prevent Method Overriding

**Final Classes** → Prevent Inheritance

# Abstract keyword

- A class that is declared with abstract keyword, is known as abstract class in java.
- It can have abstract and non-abstract methods (method with body).
- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- There are two ways to achieve abstraction in java
  - Abstract class
  - Interface

# Abstract keyword

## Abstract class

- 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.

- **Example**

```
abstract class A{
```



# Some rules of Abstract Class

- 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.

# Some rules of Abstract Method

- If there is any abstract method in a class, that class must be abstract.
- If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

# Abstract Method

## Abstract method

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

`abstract void printStatus();//no body and abstract`

# Why Interfaces?

```
public class extends Animal, Mammal{}
```

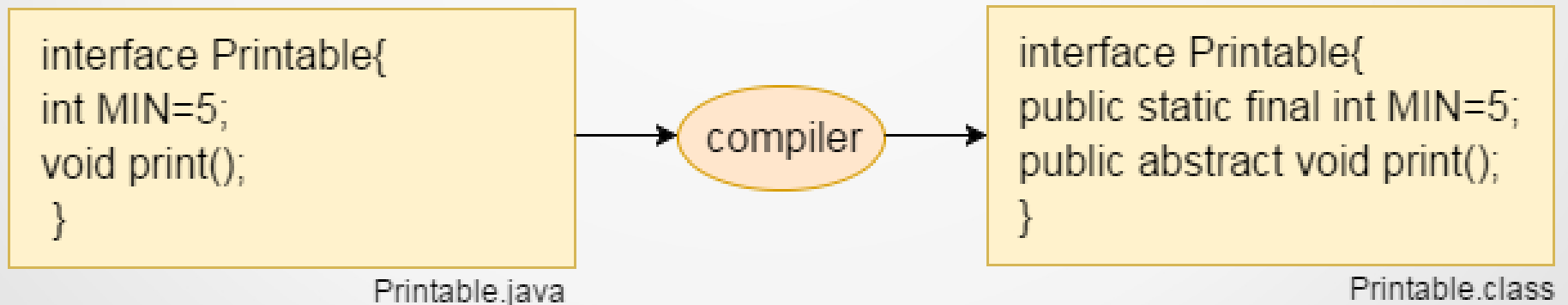
- However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritances.
- **The reason behind this is to prevent ambiguity.**
- Consider a case where class B extends class A and Class C and both class A and C have the same method display().
- Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java.

# Interfaces in Java

- An interface in java is a blueprint of a class.
- It has static constants and abstract methods.
- The interface in java is a mechanism to achieve abstraction.
- There can be only abstract methods in the java interface not method body.
- It is used to achieve abstraction and multiple inheritance in Java.
- Java Interface also represents IS-A relationship.
- It cannot be instantiated just like abstract class.
- By interface, we can support the functionality of multiple inheritance.

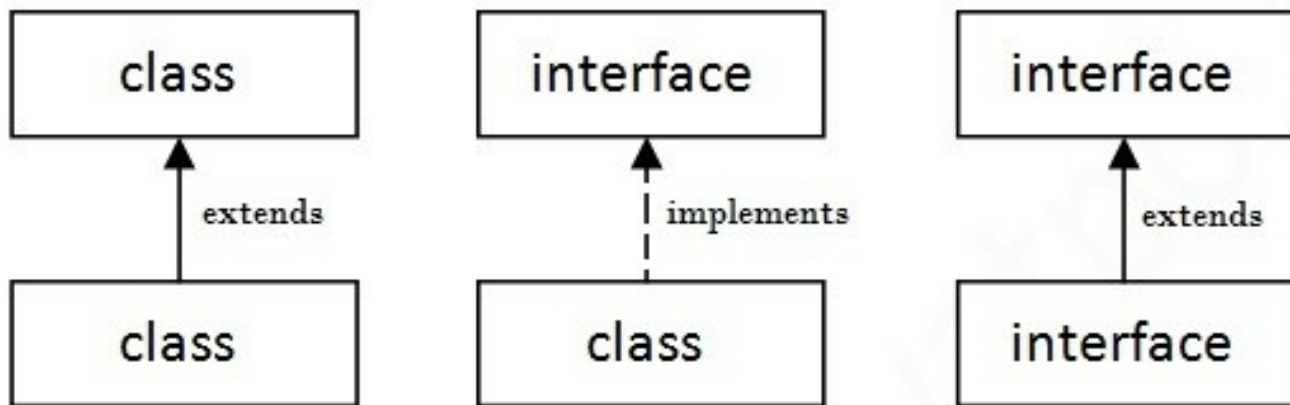
# Interfaces in Java

- The java compiler adds public and abstract keywords before the interface method. More, it adds public, static and final keywords before data members.
- In other words, Interface fields are public, static and final by default, and methods are public and abstract.



# Understanding relationship between classes and interfaces

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

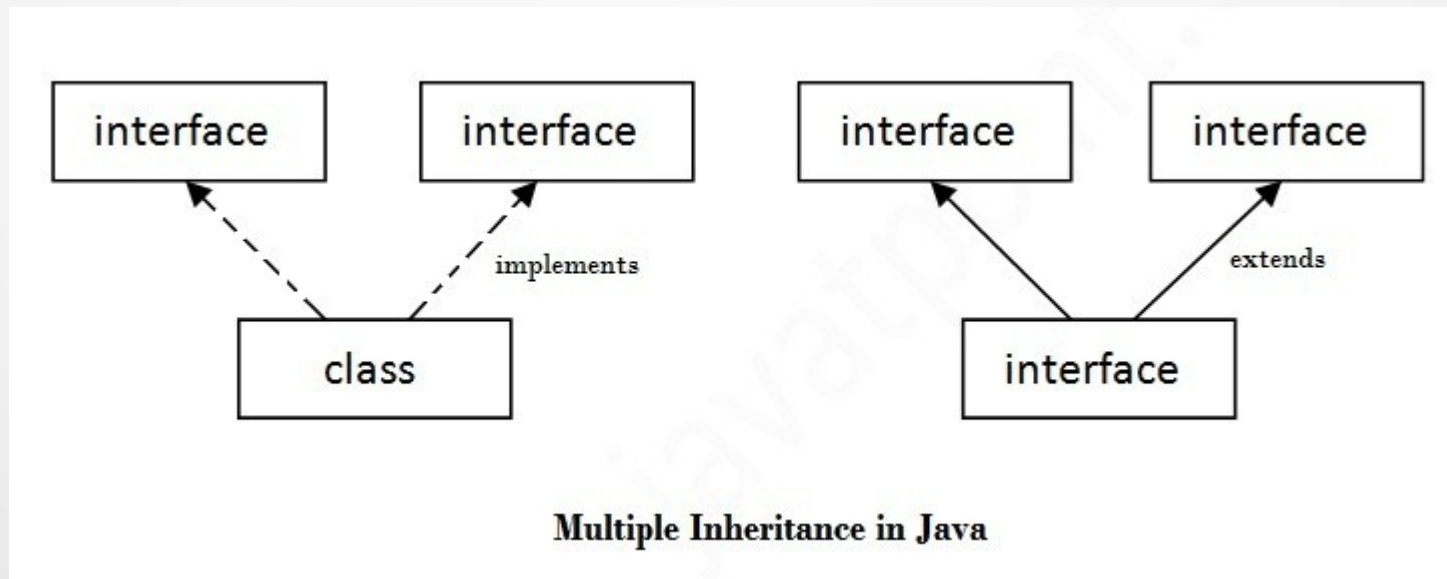


Inheritance21.java

Inheritance22.java

# Multiple Inheritance in Java

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



Inheritance23.java

Inheritance24.java

Inheritance25.java



# Interfaces in Java

- A class implements interface but one interface extends another interface .

# Interface v/s Abstract class

| Interface  | Abstract Class  |
|--|---|
| Declared using the keyword interface.  | Declared using the keyword abstract.  |
| Multiple Inheritance is possible.  | Multiple Inheritance is not possible.                                       |
| <b>Implements</b> keyword is used to inherit an inheritance  | <b>Extends</b> keyword is used to inherit a class.                          |
| By default, all methods in an interface are <b>public and abstract</b> ; need to tag it as public and abstract | Methods have to be tagged as <b>public or abstract</b> or both if required. |
| Interfaces have no implementation at all.  | Abstract classes can have partial implementation.                           |
| All methods of an interface need to be overridden.   | Only abstract methods need to be overridden.                                |
| All variables declared in an interface are by default public, static or final.                                 | Variables have to be declared as public, static or final.                   |
| Interfaces do not have any constructors.   | Abstract classes can have constructors.                                     |
| Methods in an interface cannot be static.  | Non-abstract methods can be static.   |
| Inheritance26.java   | Inheritance20.java  |

## Abstract class

## Interface

1) Abstract class can **have abstract and non-abstract** methods.

Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.

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 provide the implementation of interface**.

Interface **can't provide the implementation of abstract class**.

5) The **abstract keyword** is used to declare abstract class.

The **interface keyword** is used to declare interface.

6) An **abstract class** can extend another Java class and implement multiple Java interfaces.

An **interface** can extend another Java interface only.

7) An **abstract class** can be extended using keyword "extends".

An **interface** can be implemented using keyword "implements".

8) A Java **abstract class** can have class members like private, protected, etc.

Members of a Java interface are public by default.

9) **Example:**

```
public abstract class Shape{  
    public abstract void draw();  
}
```

**Example:**

```
public interface Drawable{  
    void draw();  
}
```