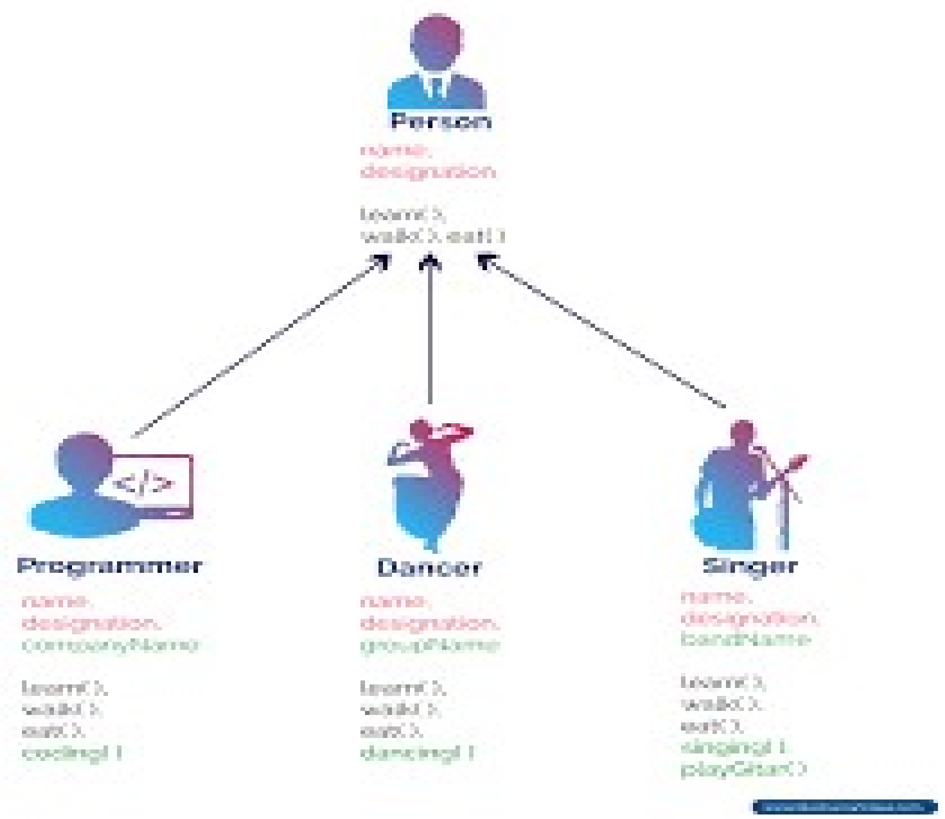
Inheritance

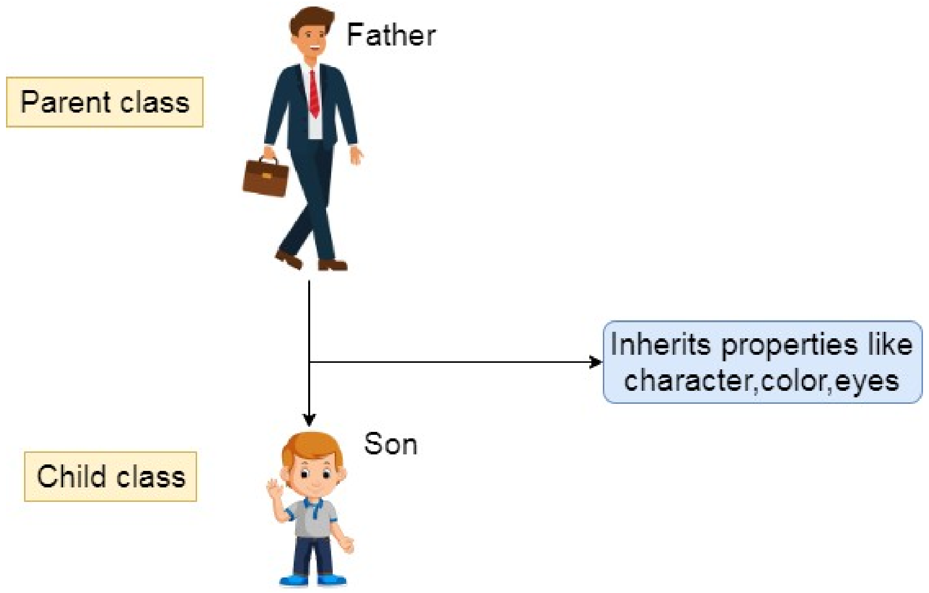
Introduction

- 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).
- 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 the parent class. Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship.



Introduction

- n Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:
  - **subclass** (child) - the class that inherits from another class
  - **superclass** (parent) - the class being inherited from
- To inherit from a class, use the ***extends*** keyword.
- In the example aside, the Son class (subclass) inherits the attributes and methods from the Father class (superclass):



Introduction

- **The syntax of Java Inheritance**

  class Subclass-name extends Superclass-name {

  //methods and fields

  }

```
class Vehicle {
  protected String brand = "Ford";
  public void horn() {
    System.out.println("Tuut, tuut!");
  }
}
class Car extends Vehicle {
  private String modelName = "Mustang";
  public static void main(String[] args) {
    Car obj1 = new Car();
    obj1.horn();
    System.out.println(obj1.brand + " " +
obj1.modelName);
  }
}
```

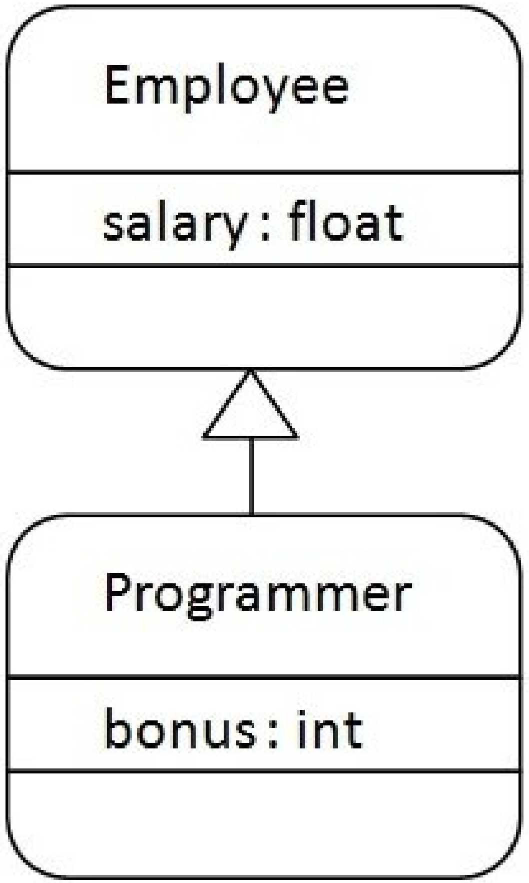
Introduction

- **Why And When To Use "Inheritance"?**
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Introduction

- **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: 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.

Introduction



- *As displayed in the above figure, 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.*

Introduction

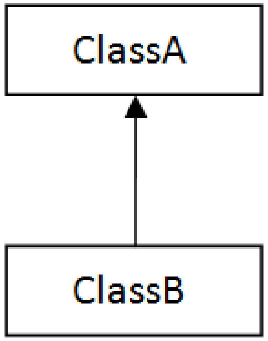
- In the example aside,
  - **Programmer** object can access the field of own class
  - as well as of **Employee** class i.e. code reusability.

```
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);
}
}
```

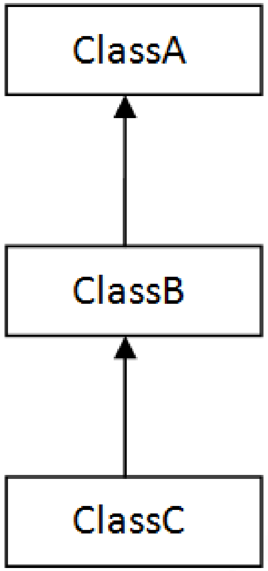
Types of Inheritance

- 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.
  - We will learn about interfaces later.

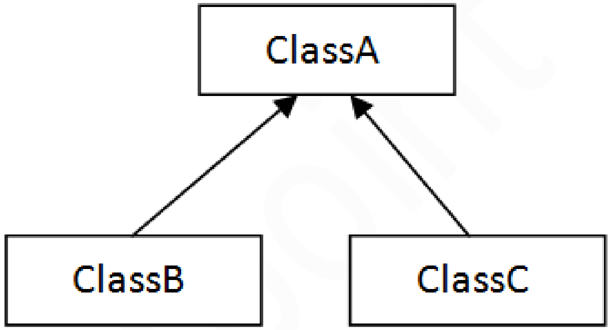
Types of Inheritance



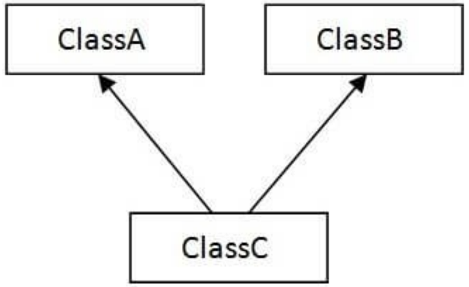
1) Single



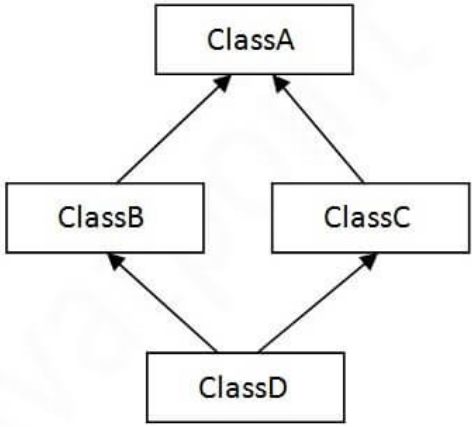
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Single Inheritance

- When a class inherits another class, it is known as a single inheritance.
- In the example given aside, 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();
    }
}
```

Multilevel Inheritance

- When there is a chain of inheritance, it is known as multilevel inheritance.
- As you can see in the example given aside,
  - 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();
    }
}
```

Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as hierarchical inheritance.
- In the example given aside, 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(); // This line will result in a compile-time error
   }
}
```

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 the same method and you call it from child class object, there will be ambiguity to call the 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.

```
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?
}
}
```

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 a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

- *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*

Method Overriding

- 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

- Let's understand the problem that we may face in the program if we don't use method overriding.
- Problem is that we have to provide a specific implementation of **run()** method in subclass that is why we use method overriding.

```
// Creating a parent class
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 the child class
        Bike obj = new Bike();
        // calling the method with the child class instance
        obj.run(); // This calls the overridden run() method
in the Bike class
    }
}
```

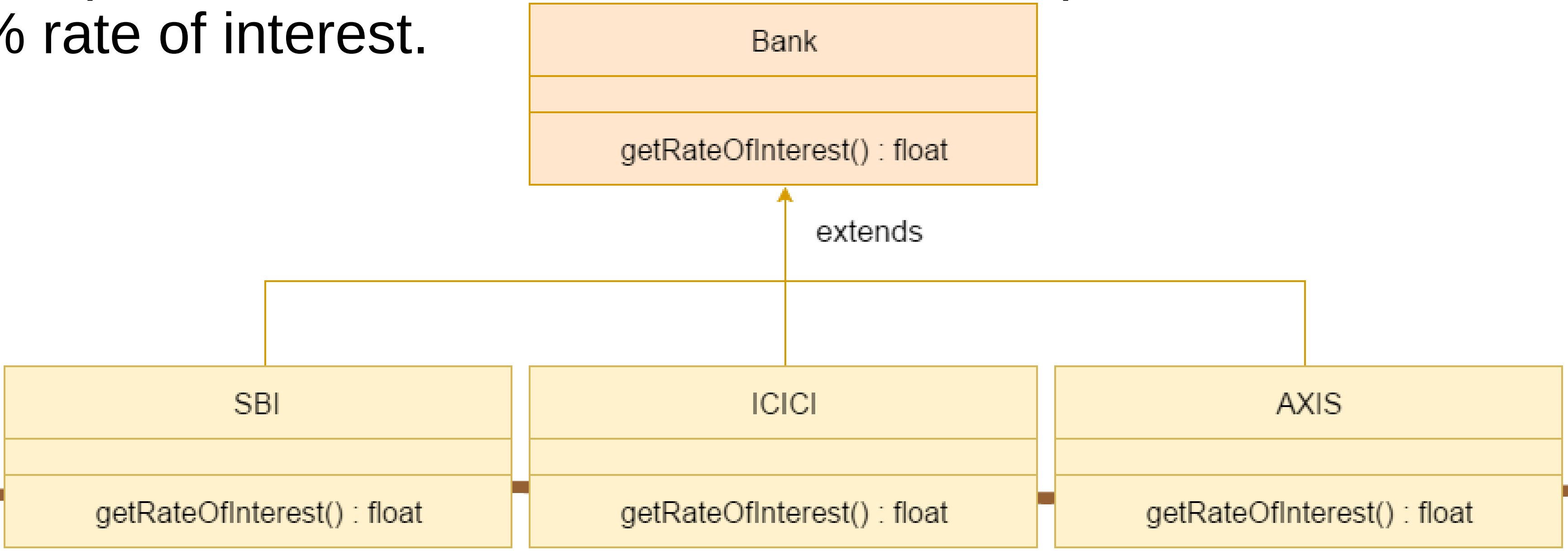
Method Overriding

- In this example, we have defined the ***run()*** method in the subclass as defined in the parent class but it has some specific implementation.
- The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
// Creating a parent class.
class Vehicle {
    // defining a method with a single parameter
    void run(String message) {
        System.out.println("Vehicle is " + message);
    }
}
// Creating a child class
class Bike2 extends Vehicle {
    // overriding the method with a single parameter
    void run(String message) {
        System.out.println("Bike is " + message + " safely\n");
    }
    public static void main(String args[]) {
        Bike2 obj = new Bike2(); // creating object
        obj.run("running"); // calling method with a single parameter
        Vehicle o = new Vehicle();
        o.run("not running");
    }
}
```

A real example of Java Method Overriding

- 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.


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

Why can we not override static method?

- It is because the static method is bound with class whereas instance method is bound with an object.
- Static belongs to the class area, and an instance belongs to the heap area.

- **Homework**:
  - Write the differences between method Overloading and Method Overriding in java

Using Super keyword

- In Java, the super keyword is used to refer to the immediate parent class object.
- It is used to call the methods of the parent class,
  - access the fields of the parent class, and
  - invoke the parent class constructor.
- Here are a few use cases of the super keyword in inheritance:

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}
class Dog extends Animal {
    void eat() {
        System.out.println("Dog is eating");
        super.eat(); // Invoking the eat method of the parent class using super
    }
}
public class TestSuperKeywords {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();
    }
}
```

Using Super keyword

- In Java, the super keyword is used to refer to the immediate parent class object.
- It is used to
  - call the methods of the parent class,
  - access the fields of the parent class, and
  - invoke the parent class constructor.
- Here are a few use cases of the super keyword in inheritance:

```
class Animal {
    String color = "White";
}
class Dog extends Animal {
    String color = "Black";
    void displayColors() {
        System.out.println("Dog color: " + color);      // Accessing the color field of the Dog class
        System.out.println("Animal color: " + super.color); // Accessing the color field of the Animal class using super
    }
}
public class TestSuperKeyword {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.displayColors();
    }
}
```

Using Super keyword

- In Java, the super keyword is used to refer to the immediate parent class object.
- It is used to
  - call the methods of the parent class,
  - access the fields of the parent class, and
  - invoke the parent class constructor.
- Here are a few use cases of the super keyword in inheritance:

```
class Animal {
    Animal() {
        System.out.println("Animal constructor");
    }
}
class Dog extends Animal {
    Dog() {
        super(); // Invoking the constructor of the parent class using super
        System.out.println("Dog constructor");
    }
}
public class TestSuper {
    public static void main(String[] args) {
        Dog myDog = new Dog();
    }
}
```

Execution of Constructors in Multilevel Inheritance

- In multilevel inheritance, where one class extends another, and then another class extends the second class, the constructors are executed in a specific order.
- The order is from the topmost (parent) class to the bottommost (child) class.
- Let's take an example to illustrate the execution of constructors in multilevel inheritance:

```
class Animal {
    Animal() {
        System.out.println("Constructor of Animal class");
    }
}
class Mammal extends Animal {
    Mammal() {
        System.out.println("Constructor of Mammal class");
    }
}
class Dog extends Mammal {
    Dog() {
        System.out.println("Constructor of Dog class");
    }
}
public class TestMulti {
    public static void main(String[] args) {
        Dog myDog = new Dog();
    }
}
```

Abstract classes and abstract methods

- In Java, abstract classes and abstract methods are used to achieve abstraction. Abstraction is the process of hiding the implementation details and showing only the functionality.
- Abstract classes and methods allow you to define a common interface for a group of related classes while leaving the specific implementation details to the individual subclasses.

Abstract classes

- Abstract Class Definition:
  - An abstract class is declared using the abstract keyword.
  - An abstract class can have abstract methods as well as concrete methods.
  - Abstract classes cannot be instantiated on their own; they are meant to be subclassed.

- Abstract Method Definition:
  - An abstract method is a method without a body (no implementation).
  - Abstract methods are declared with the abstract keyword.
  - Subclasses must provide an implementation for all abstract methods of the superclass.

Abstract class and abstract methods

```
abstract class Shape {
    // Abstract method - to be implemented by subclasses
    abstract double calculateArea();
    // Concrete method
    void display() {
        System.out.println("This is a shape.");
    }
}
class Circle extends Shape {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    // Implementation of the abstract method
    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

```
class Square extends Shape {
    double side;
    Square(double side) {
        this.side = side;
    }
    // Implementation of the abstract method
    @Override
    double calculateArea() {
        return side * side;
    }
}
public class AbstractClass {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        Square square = new Square(4);
        circle.display();
        System.out.println("Area of Circle: " + circle.calculateArea());
        square.display();
        System.out.println("Area of Square: " + square.calculateArea());
    }
}
```