Inheritance in Java

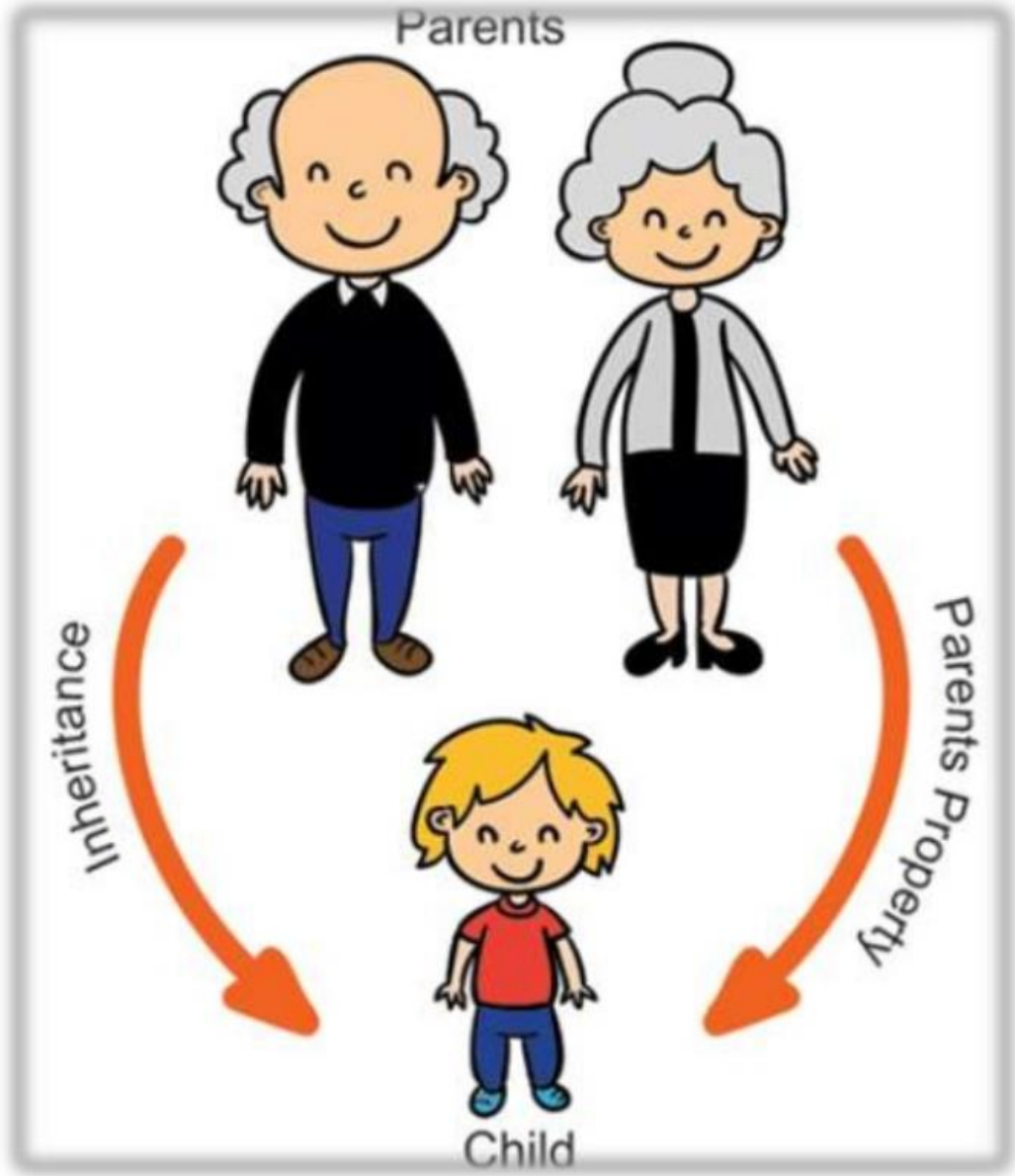
Inheritance in Java

- **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.
- When a Class extends another class it inherits all non-private members including fields and methods. Inheritance in Java can be best understood in terms of Parent and Child relationship, also known as Super class(Parent) and Sub class(child) in Java language.
- Inheritance defines is-a relationship between a Super class and its Sub class. extends and implements keywords are used to describe inheritance in Java.

Inheritance in Java

- Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.
- **Why use Inheritance ?**
- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usability

Inheritance in Java



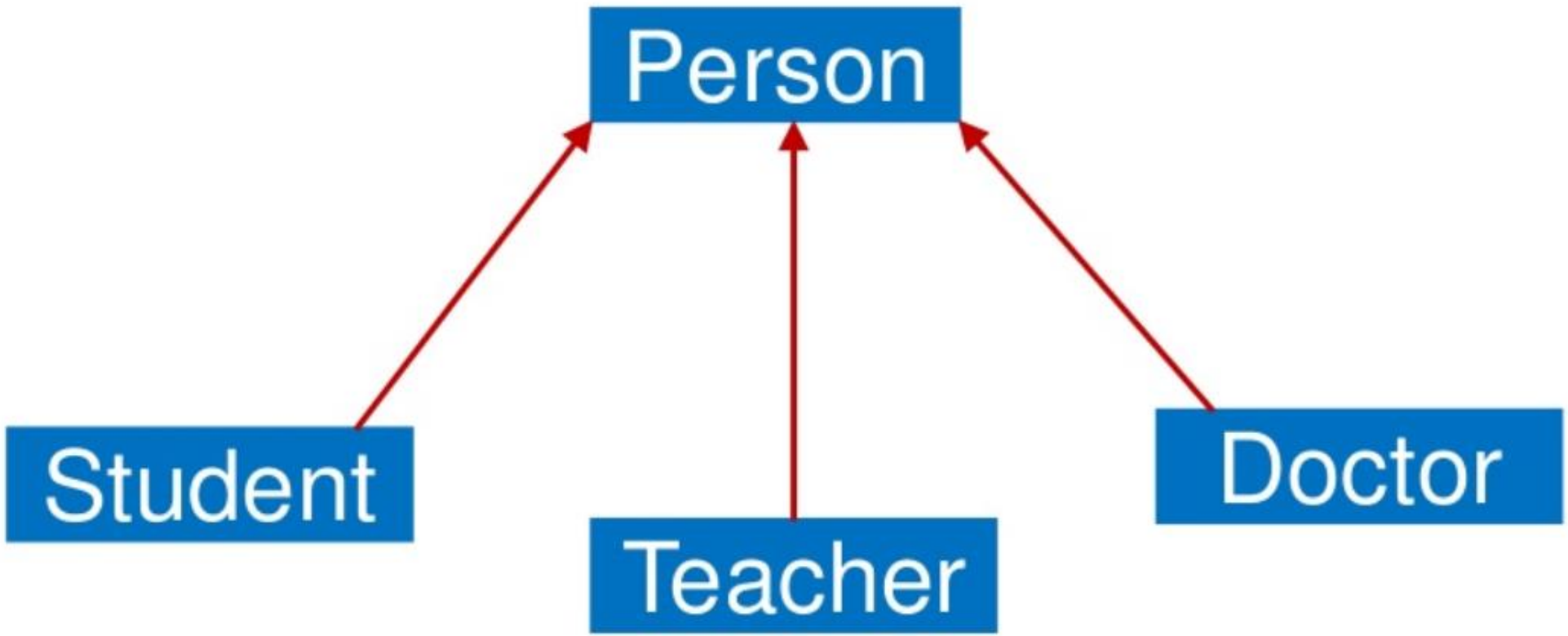
Inheritance in Java

- **Important points**
- In the inheritance the class which is give data members and methods is known as base or super or parent class.
- The class which is taking the data members and methods is known as sub or derived or child class.
- The data members and methods of a class are known as features.
- The concept of inheritance is also known as re-usability or extendable classes or sub classing or derivation.

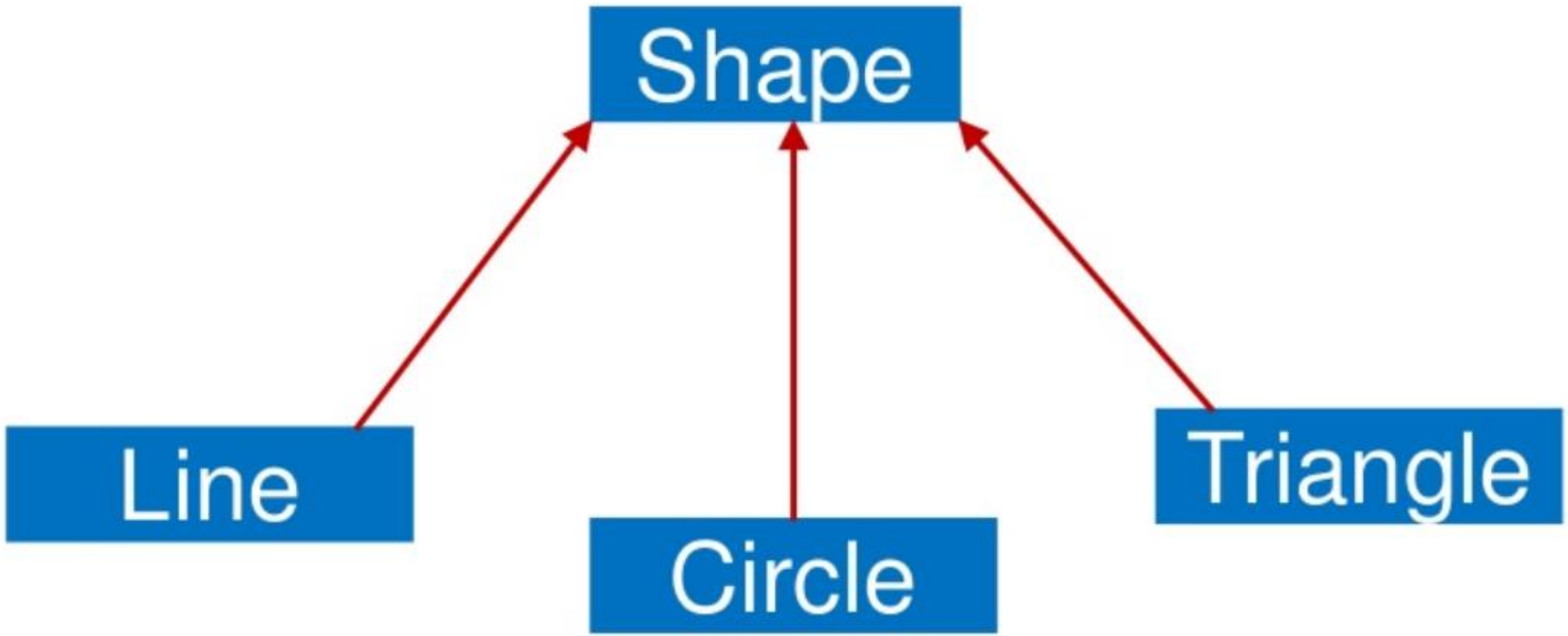
Inheritance in Java

No	Term	Definition
1	Inheritance	**Inheritance** is a process where one object acquires the properties of another object
2	Subclass	Class which inherits the properties of another object is called as **subclass**
3	Superclass	Class whose properties are inherited by subclass is called as **superclass**
4	Keywords Used	extends and implements

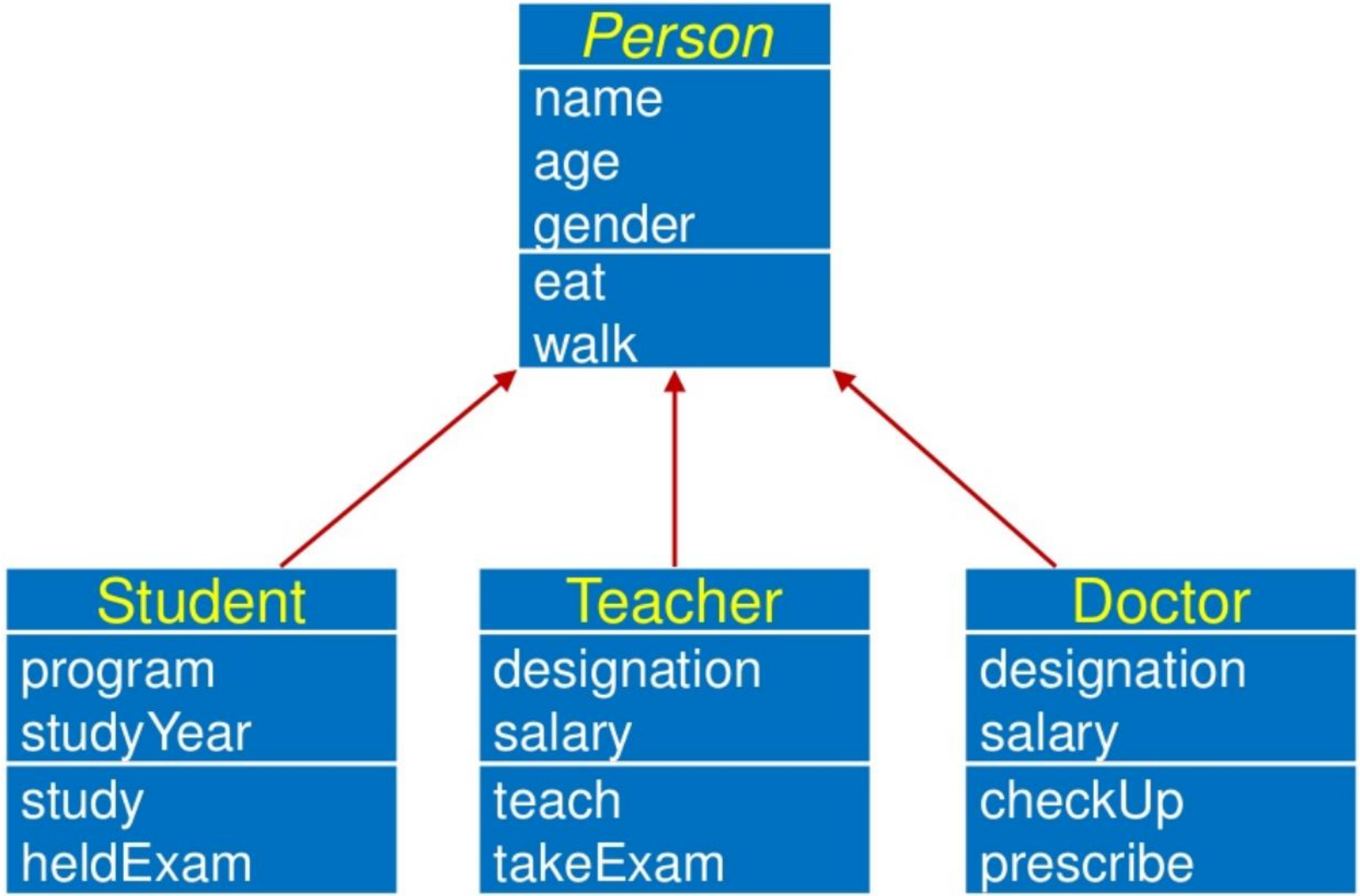
Example – Inheritance



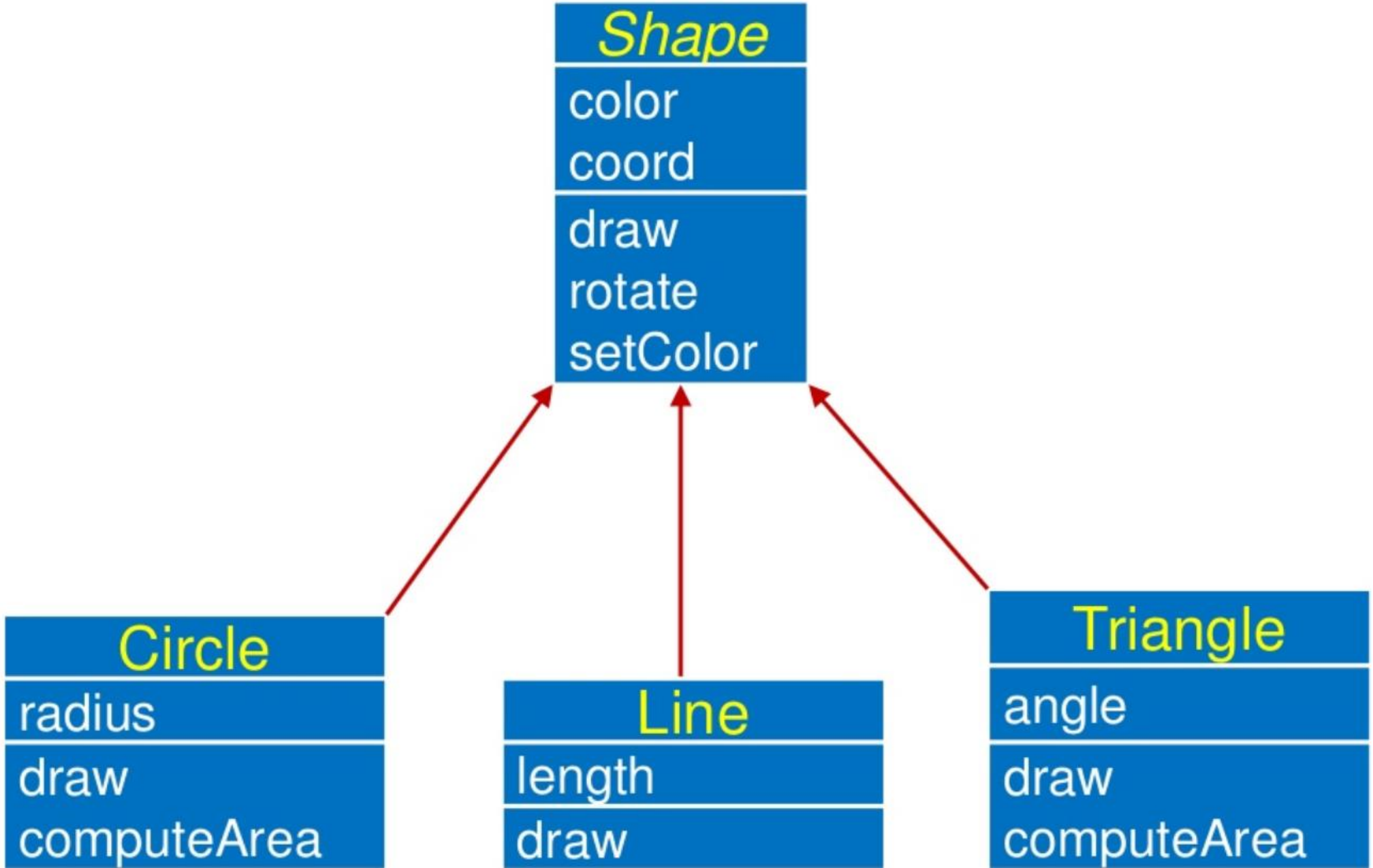
Example – Inheritance



Example – "IS A" Relationship



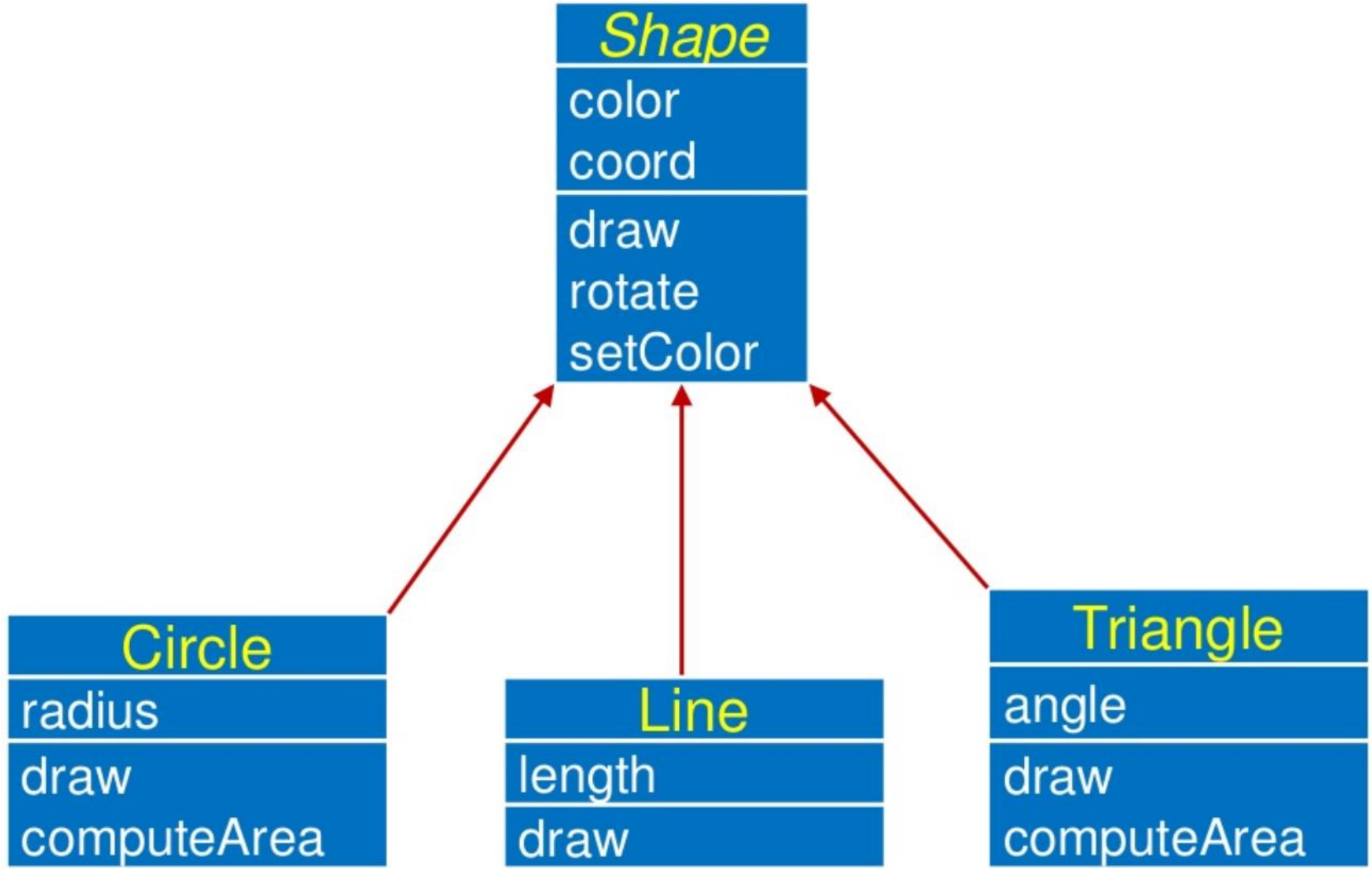
Example – "IS A" Relationship



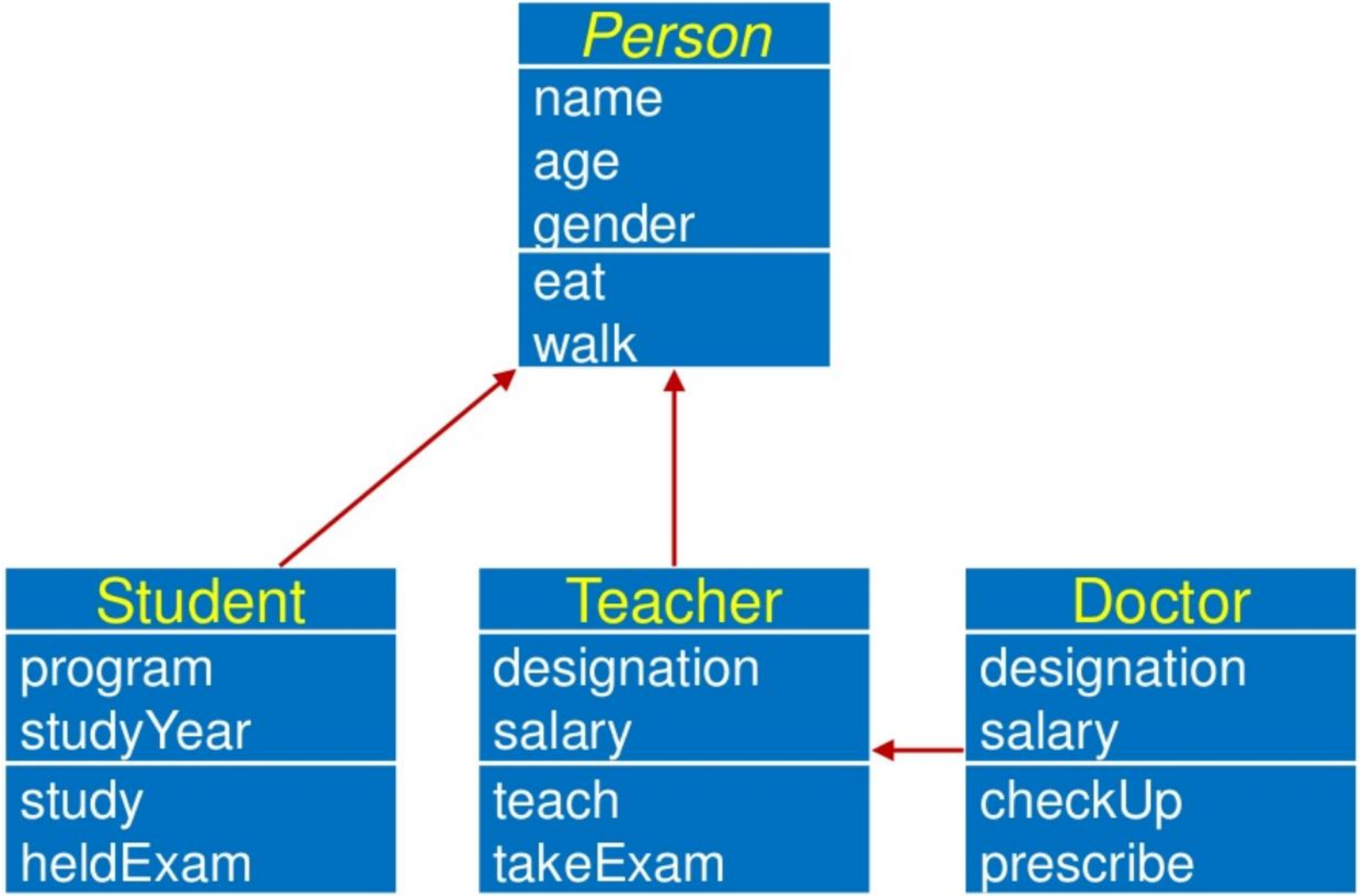
Reuse with Inheritance

- Main purpose of inheritance is reuse
- We can easily add new classes by inheriting from existing classes
  - Select an existing class closer to the desired functionality
  - Create a new class and inherit it from the selected class
  - Add to and/or modify the inherited functionality

Example Reuse



Example Reuse-1



Concepts Related with Inheritance

- Generalization
- Subtyping (extension)
- Specialization (restriction)

Concepts Related with Inheritance

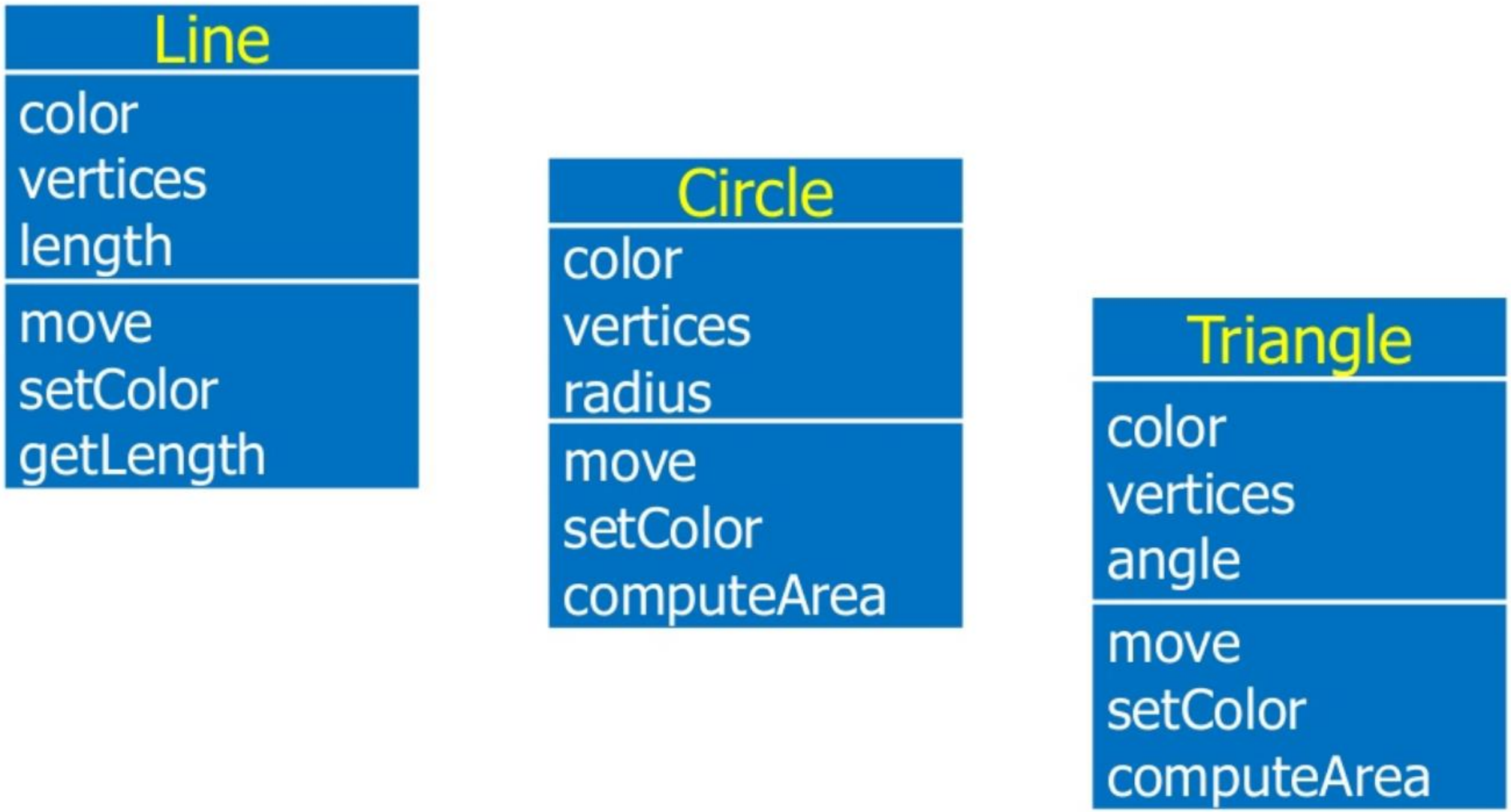
- **Generalization**

- In OO models, some classes may have common characteristics

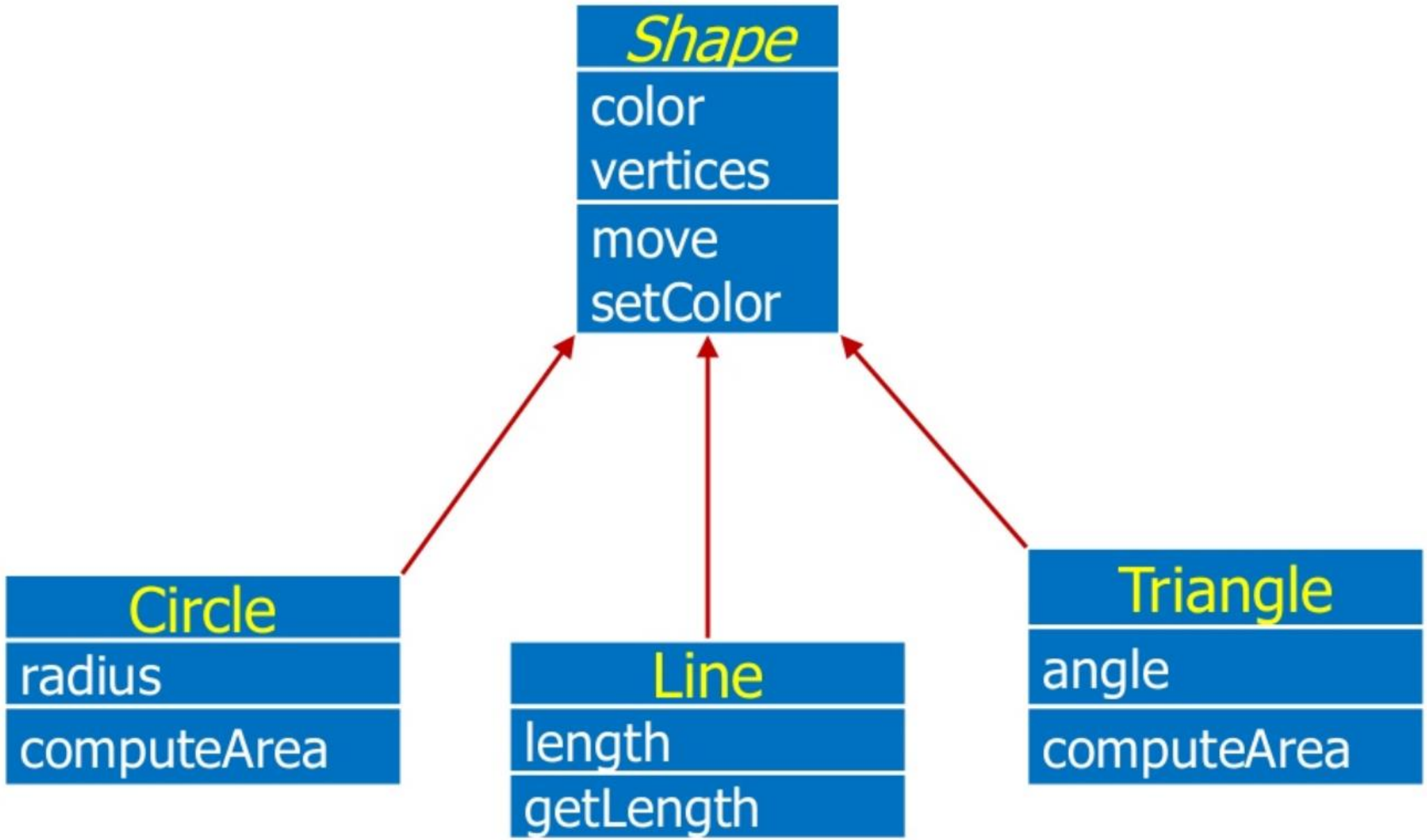
- We extract these features into a new class and inherit original classes from this new class

- This concept is known as Generalization

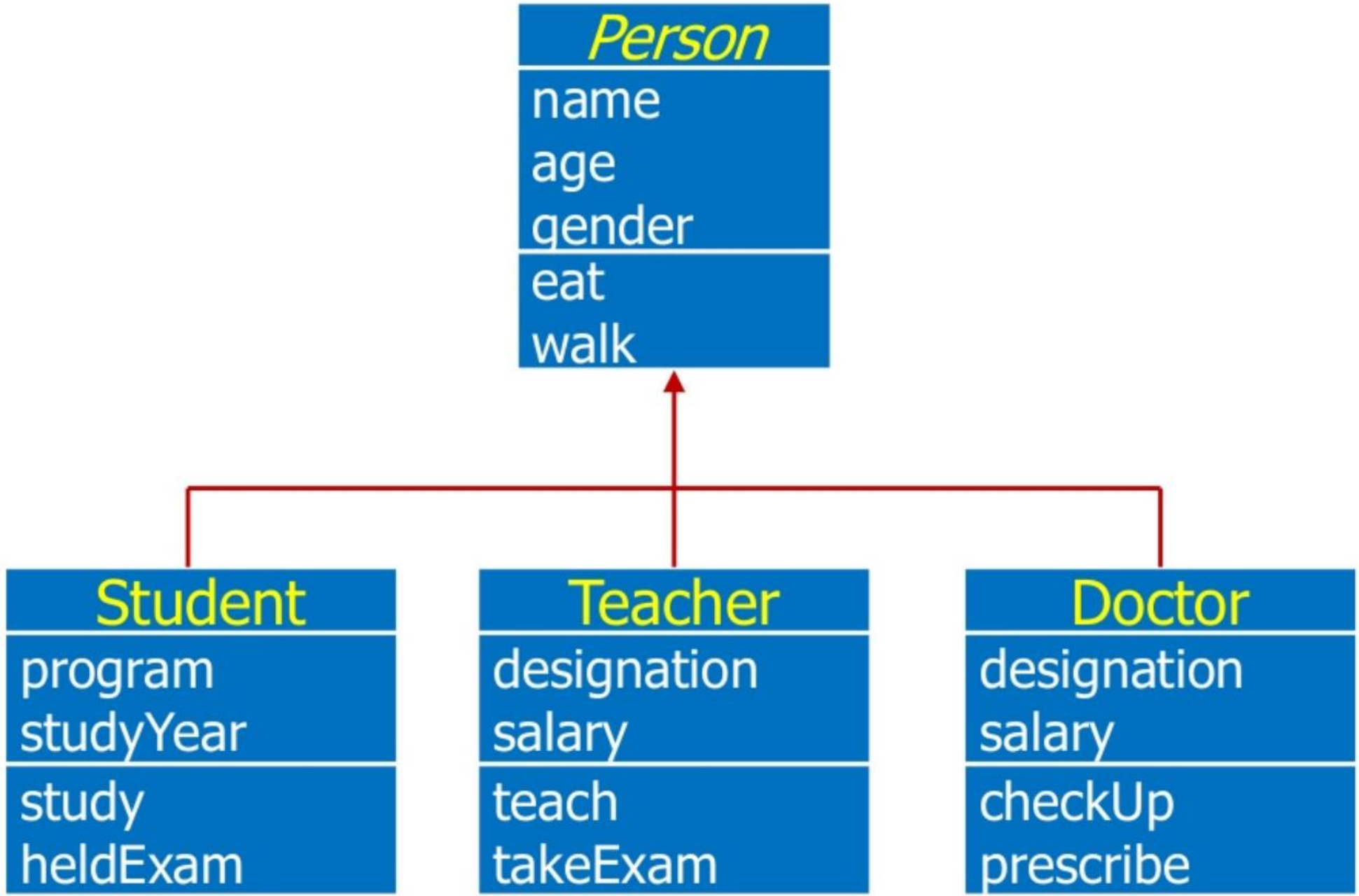
Example – Generalization



Example – Generalization



Example – Generalization



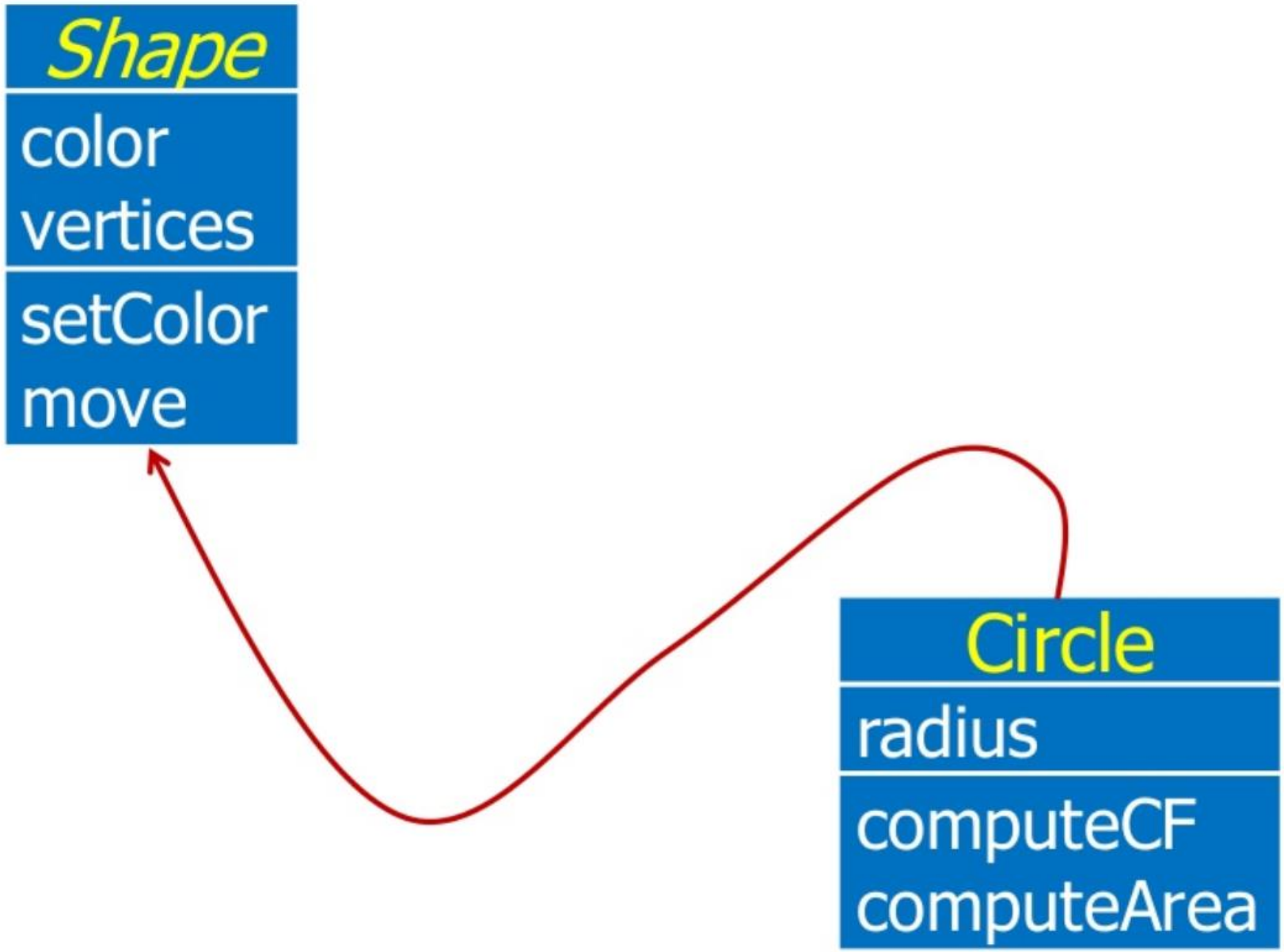
Sub-typing & Specialization

- We want to add a new class to an existing model
- Find an existing class that already implements some of the desired state and behaviour
- Inherit the new class from this class and add unique behaviour to the new class

Sub-typing (Extension)

- Sub-typing means that derived class is behaviourally compatible with the base class
- Behaviourally compatible means that base class can be replaced by the derived class

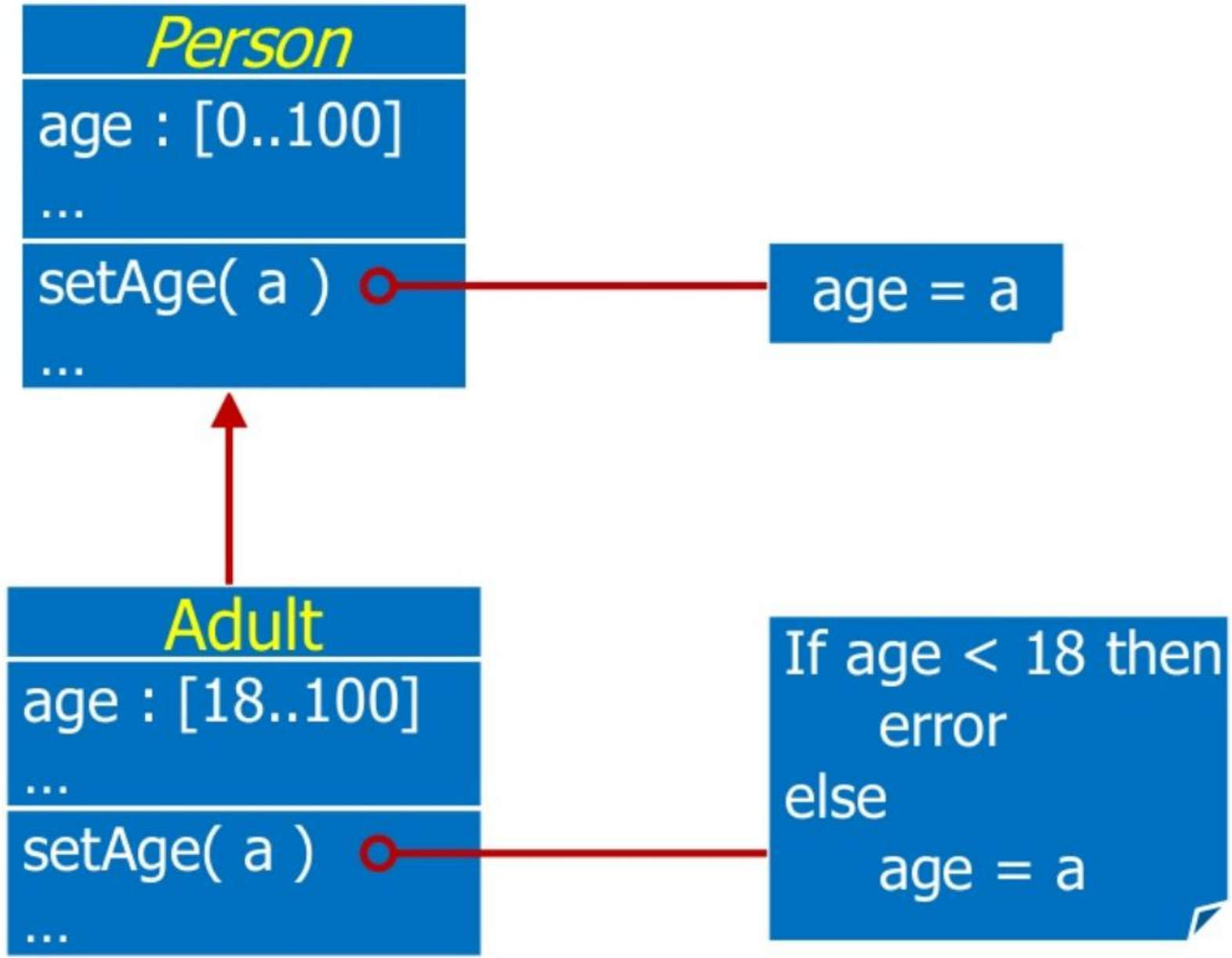
Example –Sub-typing (Extension)



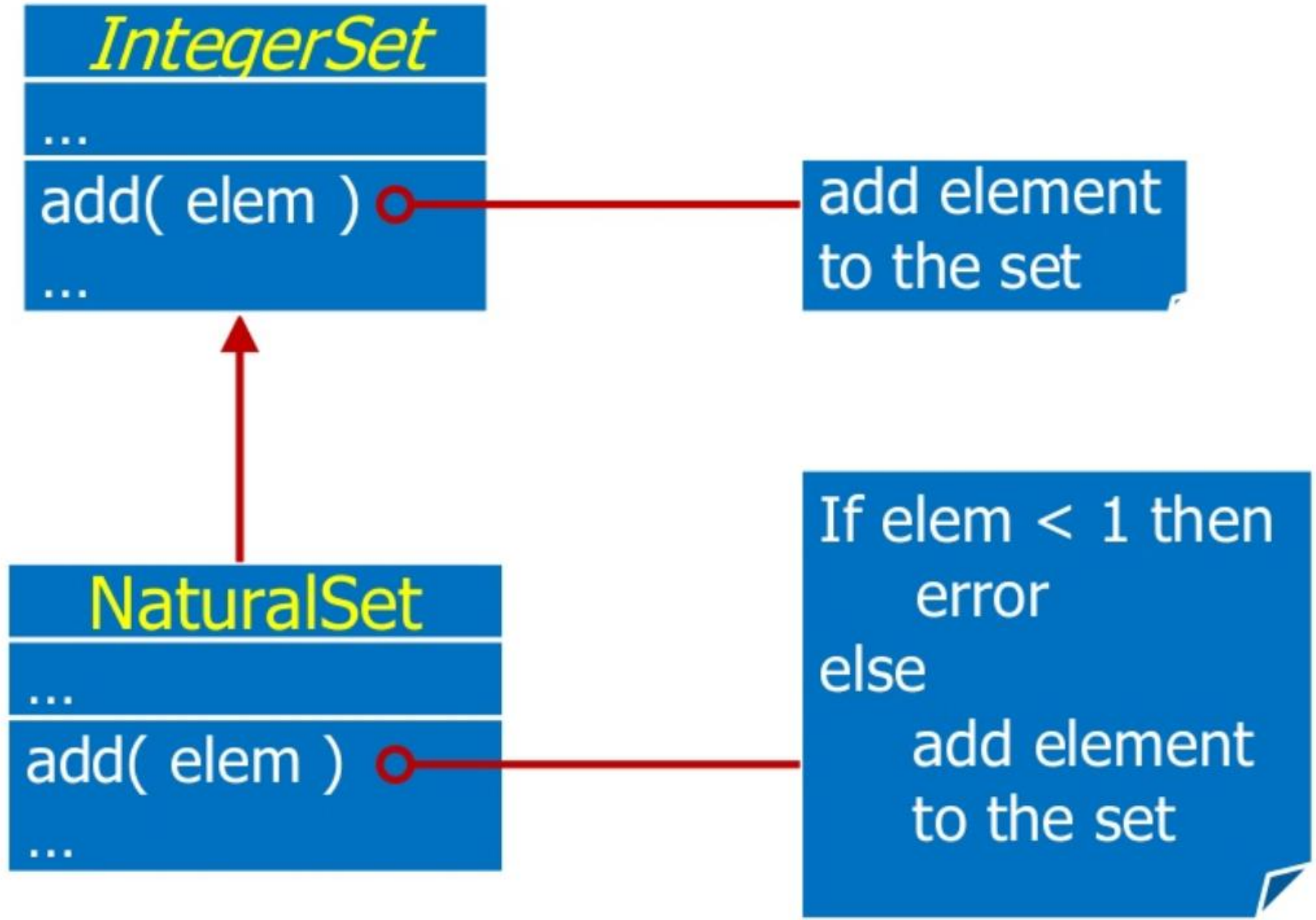
Specialization (Restriction)

- Specialization means that derived class is behaviourally incompatible with the base class
- Behaviourally incompatible means that base class can’t always be replaced by the derived class

Example – Specialization (Restriction)



Example – Specialization (Restriction)



Inheritance in Java

- **Inheritance in Java**
- Inheritance in Java is done using
  - **extends** – In case of Java class and abstract class
  - **implements** – In case of Java interface.
- **What is inherited**
  - In Java when a class is extended, sub-class inherits all the **public**, **protected** and **default (Only if the sub-class is located in the same package as the super class)** methods and fields of the super class.
- **What is not inherited**
  - **Private** fields and methods of the super class are not inherited by the sub-class and can't be accessed directly by the subclass.
  - Constructors of the super-class are not inherited. There is a concept of constructor chaining in Java which determines in what order constructors are called in case of inheritance.

Inheritance in Java

- **Syntax of 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.

Inheritance in Java

- **extends Keyword**
- **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

```
class Super {
   .....
   .....
}
class Sub extends Super {
   .....
   .....
}
```

Inheritance in Java

- **IS-A Relationship with Example**
- IS-A is a way of saying : This object is a type of that object.

```
public class Vehicle{
}

public class FourWheeler extends Vehicle{
}

public class TwoWheeler extends Vehicle{
}

public class Car extends FourWheeler{
}
```

Inheritance in Java

```
public class Vehicle{
}

public class FourWheeler extends Vehicle{
}

public class TwoWheeler extends Vehicle{
}

public class Car extends FourWheeler{
}
```

- **Conclusions from above Example :**
- Vehicle is the **superclass** of TwoWheeler class.
- Vehicle is the **superclass** of FourWheeler class.
- TwoWheeler and FourWheeler are **subclasses** of Vehicle class.
- Car is the **subclass** of both FourWheeler and Vehicle classes.

Inheritance in Java

```
public class Vehicle{
}

public class FourWheeler extends Vehicle{
}

public class TwoWheeler extends Vehicle{
}

public class Car extends FourWheeler{
}
```

IS-A relationship of above example is:

TwoWheeler IS-A Vehicle

FourWheeler IS-A Vehicle

Car IS-A FourWheeler

Hence

Car IS-A Vehicle

- **Conclusions from above Example :**
- Vehicle is the **superclass** of TwoWheeler class.
- Vehicle is the **superclass** of FourWheeler class.
- TwoWheeler and FourWheeler are **subclasses** of Vehicle class.
- Car is the **subclass** of both FourWheeler and Vehicle classes.

Inheritance in Java

- **IS-A Relationship**

```
public class Animal {
}
public class Mammal extends Animal {
}
public class Reptile extends Animal {
}
public class Dog extends Mammal {
}
```

- Based on the above example, in Object-Oriented terms which is base Class and which is Derived Class…???

Inheritance in Java

- Based on the above example, in Object-Oriented terms, the following are:
  - Animal is the superclass of Mammal class.
  - Animal is the superclass of Reptile class.
  - Mammal and Reptile are subclasses of Animal class.
  - Dog is the subclass of both Mammal and Animal classes.
- Now, if we consider the IS-A relationship, we can say:
  - Mammal IS-A Animal
  - Reptile IS-A Animal
  - Dog IS-A Mammal
  - Hence: Dog IS-A Animal as well
- With the use of the extends keyword, the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass.

Simple Example of Inheritance

```
class Parent
{
   public void p1() {
      System.out.println("Parent method");
   }
}
public class Child extends Parent {
   public void c1() {
      System.out.println("Child method");
   }
   public static void main(String[] args) {
      Child cobj = new Child();
      cobj.c1();  //Calling method of Child class
      cobj.p1();  //Calling method of Parent class
   }
}
```

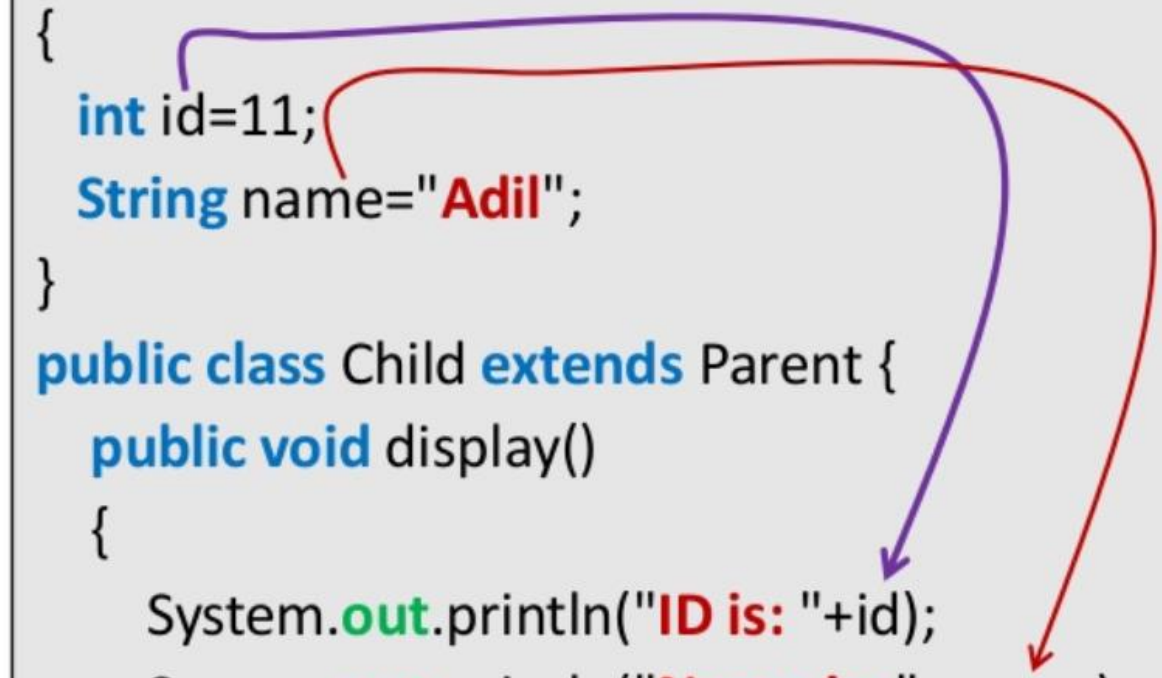
Explanation of Previous Program

- When child class inherit the propertied of parent class the child class look like this (just for understanding)

```
public class Child extends Parent {
   public void c1() {
      System.out.println("Child method"); }
    public void p1() {
      System.out.println("Parent method"); }
   public static void main(String[] args) {
      Child cobj = new Child();
      cobj.c1();  //Calling method of Child class
      cobj.p1();  //Calling method of Parent class
   }
}
```

Another example of Inheritance

```
class Parent
{
   int id=11;
   String name="Adil";
}
public class Child extends Parent {
    public void display()
    {
        System.out.println("ID is: "+id);
        System.out.println("Name is: "+name);
    }
    public static void main(String[] args)  {
        Child obj = new Child();
        obj.display();
    }
}
```



Here in Child Class we Can Inherit Or Access the Properties of Parent Class

Access Control and Inheritance

- A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the members of derived classes should be declared private in the base class.

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Access Control and Inheritance

- The following rules for inherited methods are enforced:

1. Methods declared public in a superclass also must be public in all subclasses.

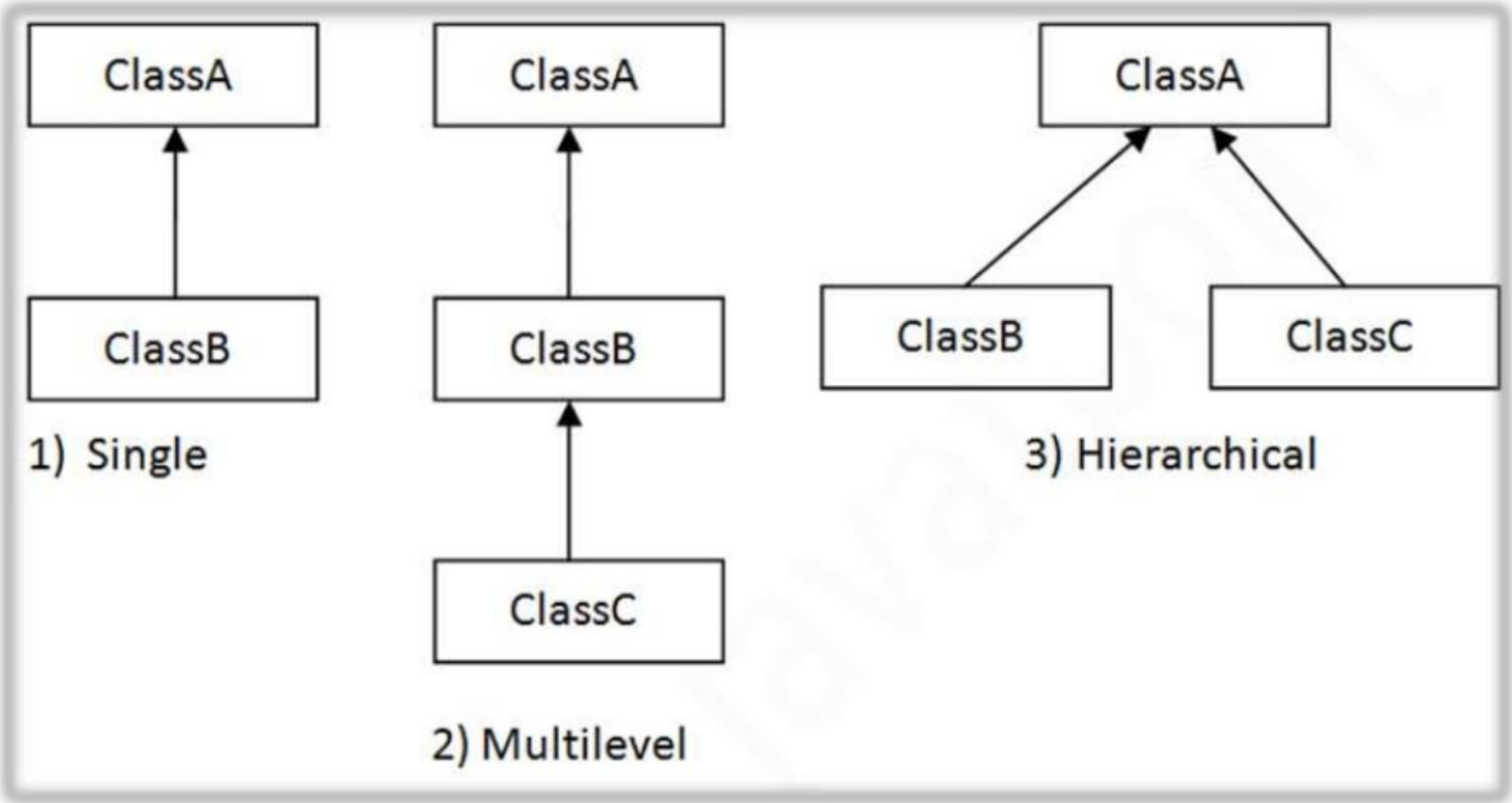
2. Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.

3. Methods declared private are not inherited at all, so there is no rule for them.

Inheritance in Java

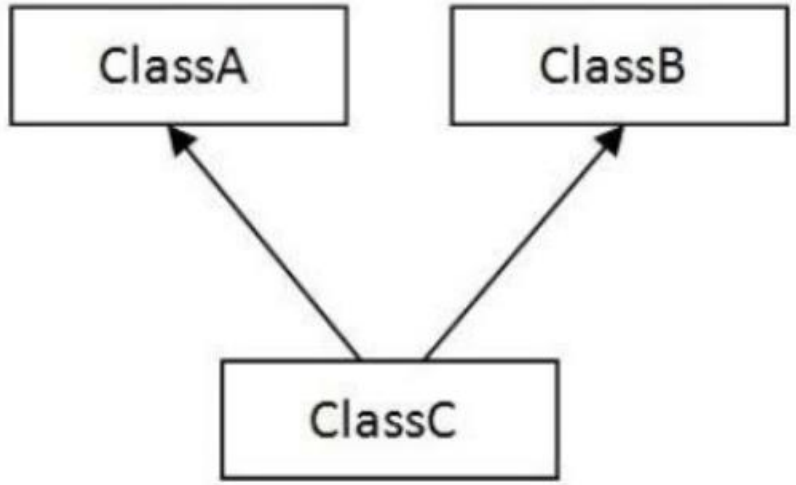
- **Types of Inheritance**
- Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance; they are:
- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

Types of Inheritance

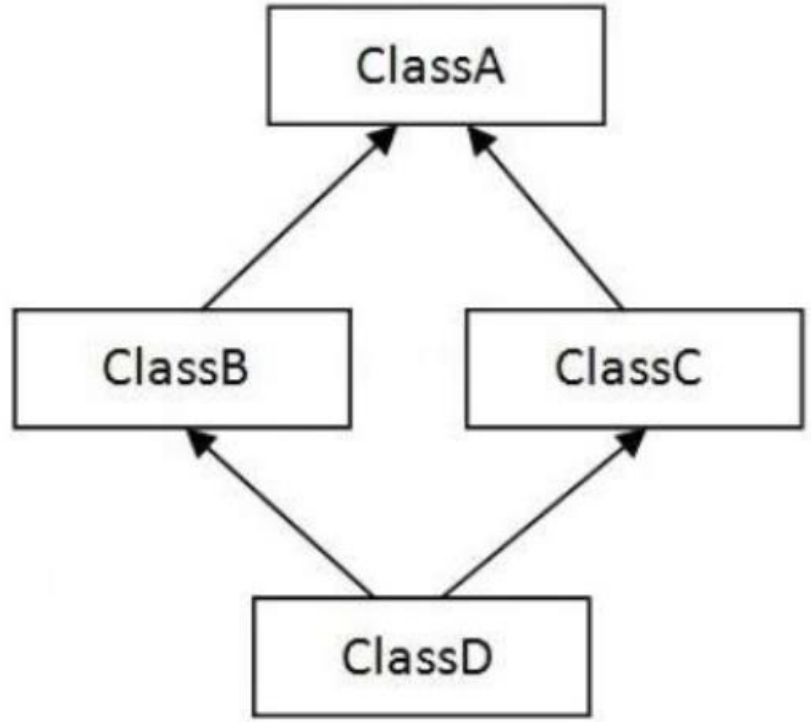


Types of Inheritance

- In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



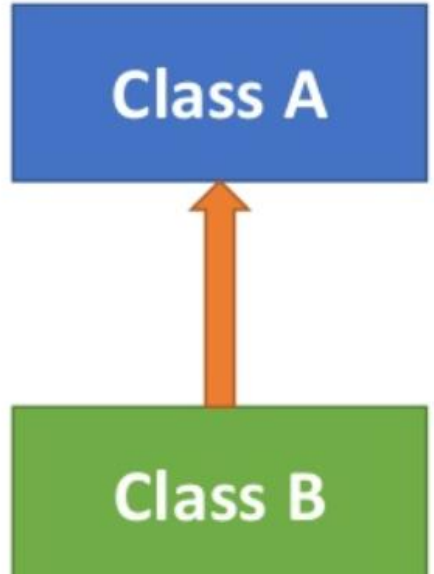
4) Multiple



5) Hybrid

Types of Inheritance

- **1) Single Inheritance**
- **Single inheritance** is easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



Single Inheritance Example

```
class A
{
.....
}

Class B extends A
{
.....
}
```

Single Inheritance Example

```
class A {
  int data=10;
}
class B extends A {
   public void display()
   {
     System.out.println("Data is:"+data);
   }
   public static void main(String args[])
   {
     B obj = new B();
     obj.display();
  }
}
```

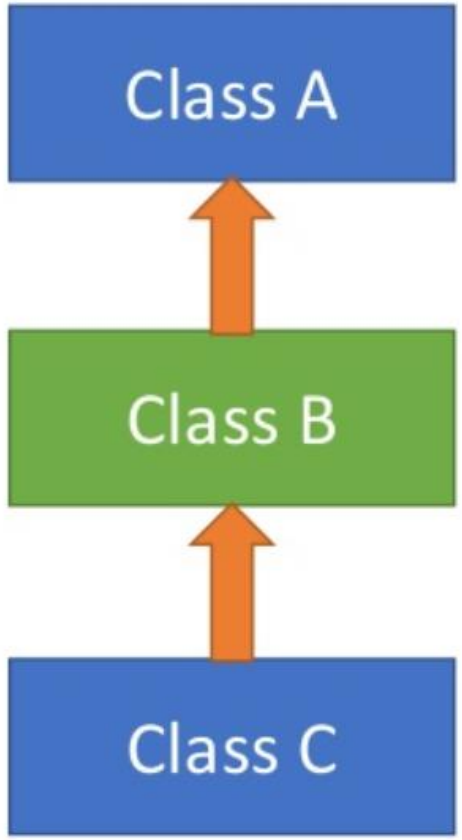
Another Single Inheritance Example

```
class Faculty {
   float salary=30000;
}
class Science extends Faculty {
   float bonus=2000;
public static void main(String args[]) {
  Science obj=new Science();
  System.out.println("Salary is:"+obj.salary);
  System.out.println("Bonus is:"+obj.bonus);
 }
}
```

Types of Inheritance

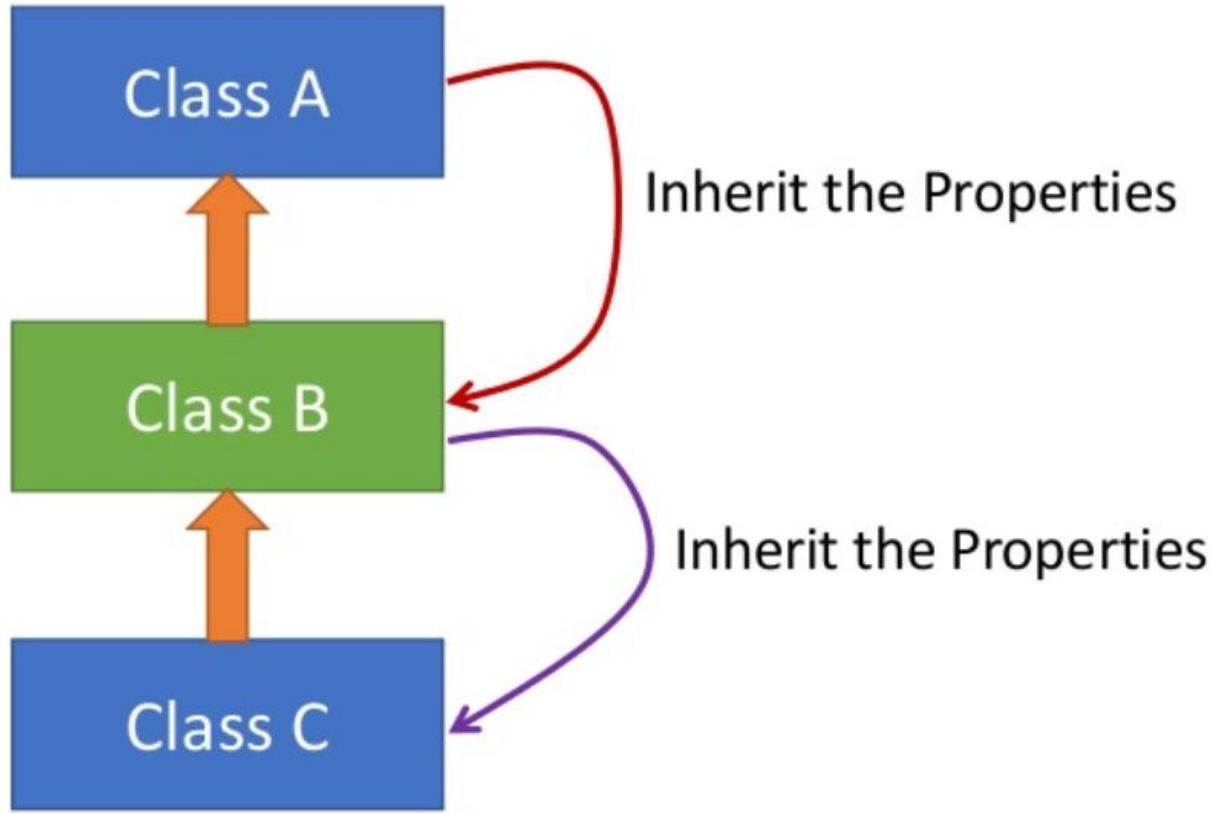
- **2) Multilevel Inheritance**
- In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.
- **Single base class + single derived class + multiple intermediate base classes.**
- **Intermediate base classes**
- An intermediate base class is one in one context with access derived class and in another context same class access base class.

Multilevel Inheritance



- Here class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and method of class A along with B that's what is called multilevel inheritance.

Multilevel Inheritance



- Here class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and method of class A along with B that's what is called multilevel inheritance.

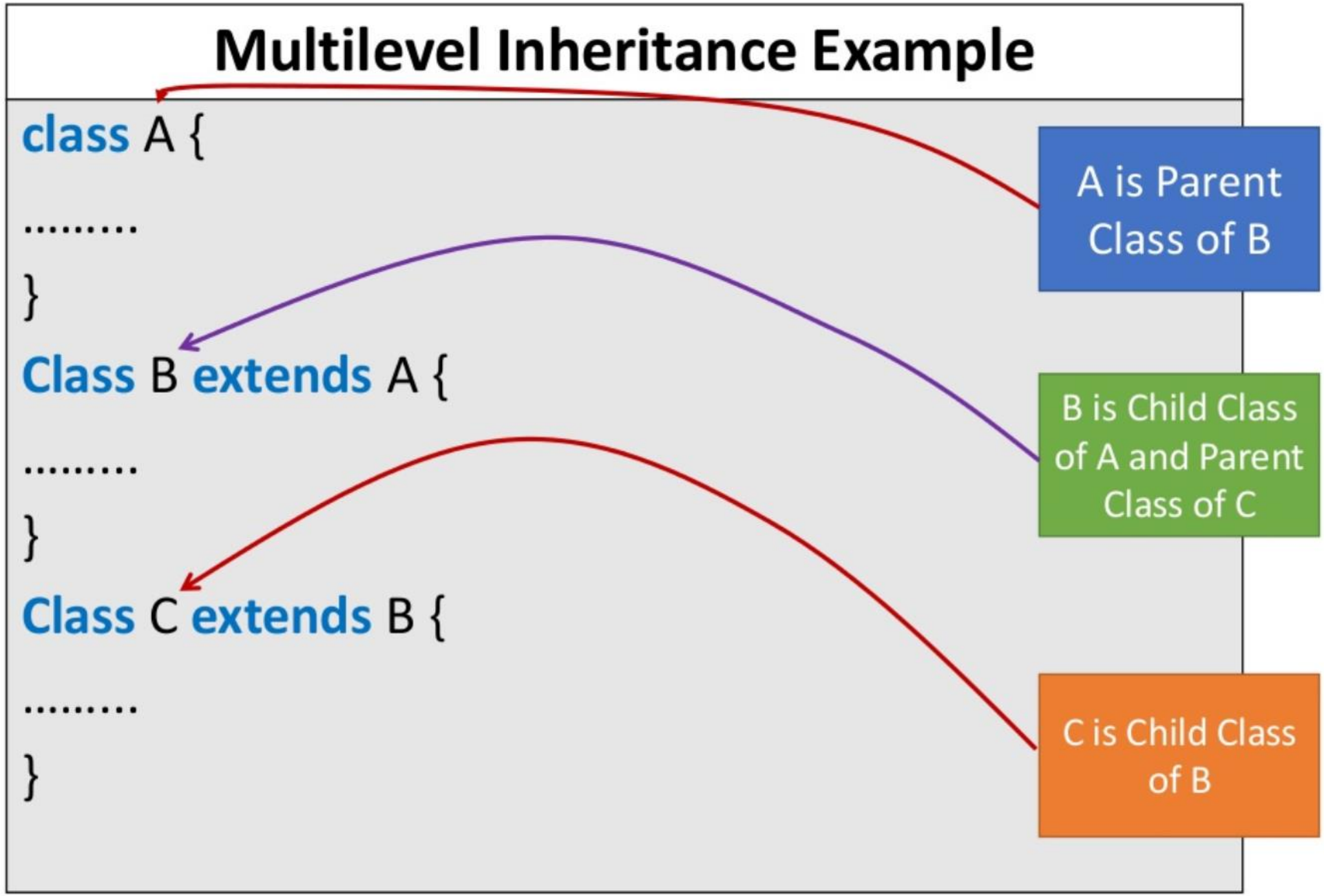
Multilevel Inheritance

- Class C can inherit the Members of both Class A and B Show below :

C Contains B Which Contains A

Multilevel Inheritance Example

```
class A {
.....
}
Class B extends A {
.....
}
Class C extends B {
.....
}
```



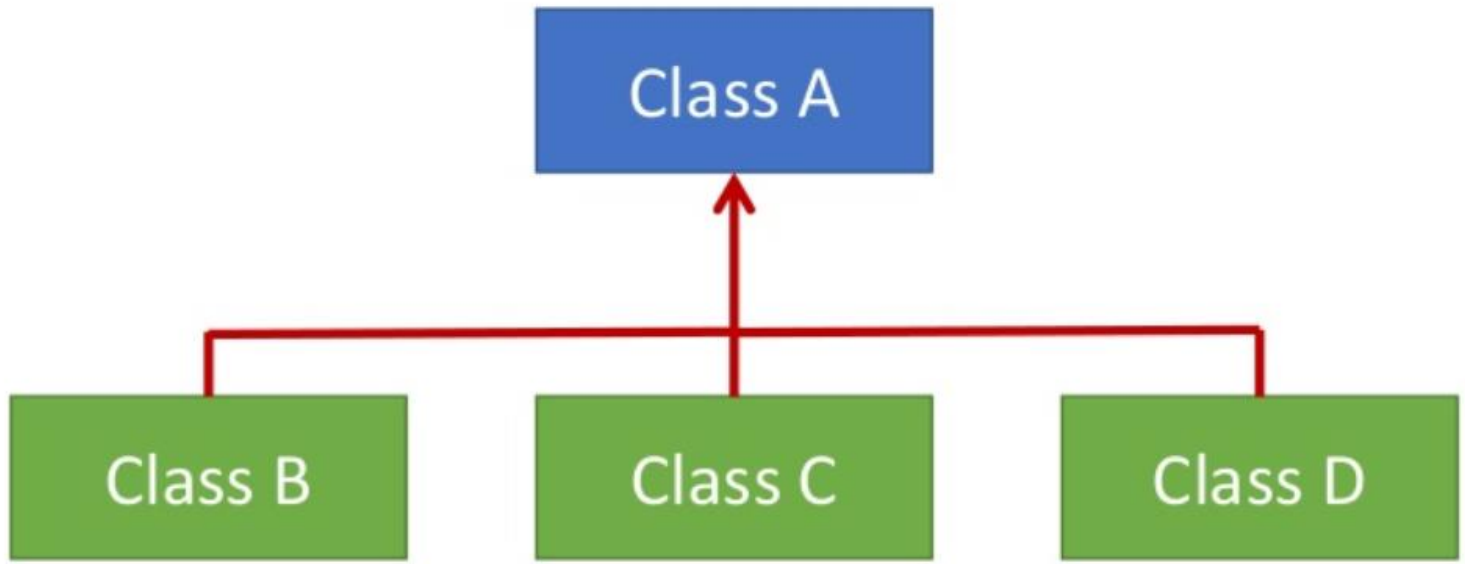
Another Multilevel Inheritance Example

```
class Faculty {
  float total_sal=0, salary=1000;
}
class HRA extends Faculty {
  float hra=2000;
}
class DA extends HRA {
  float da=3000;
}
class Science extends DA {
  float bonus=4000;
public static void main(String args[]) {
  Science obj=new Science();
  obj.total_sal=obj.salary+obj.hra+obj.da+obj.bonus;
  System.out.println("Total Salary is:"+obj.total_sal);
 }
```

Here we can create an Object of Class “Science”

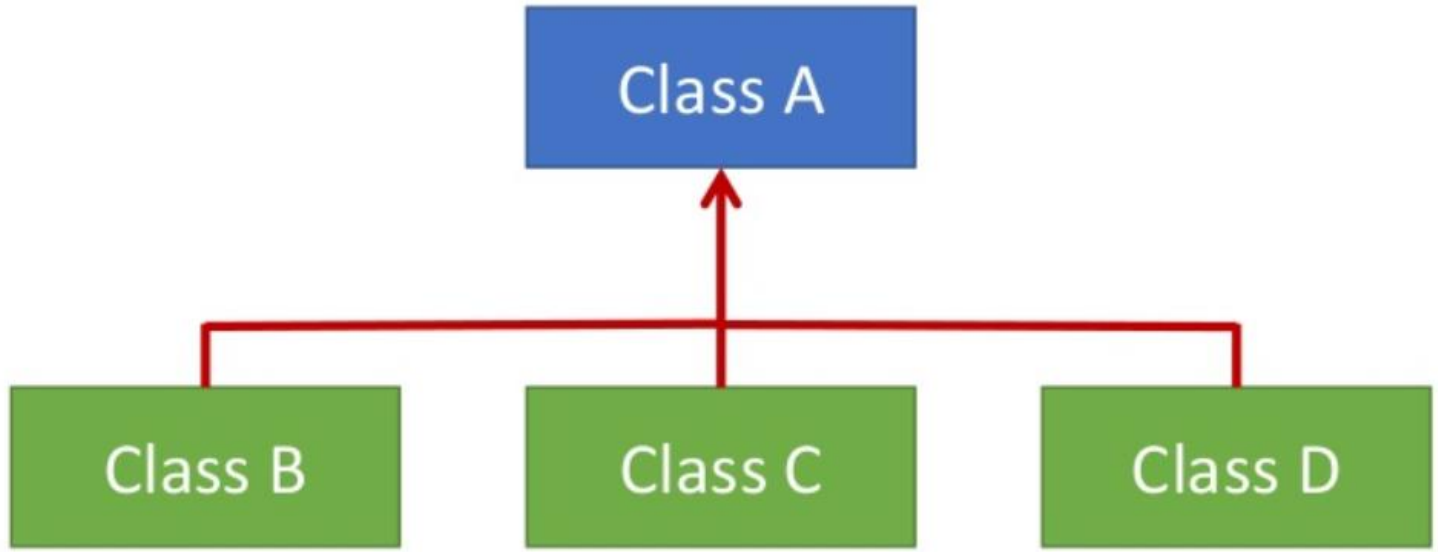
Types of Inheritance

- **3) Hierarchical Inheritance**
- In this **inheritance** multiple classes inherits from a **single** class i.e there is one super class and **multiple** sub classes. As we can see from the below diagram when a same class is having more than one sub class (or) more than one sub class has the same parent is called as **Hierarchical Inheritance**.



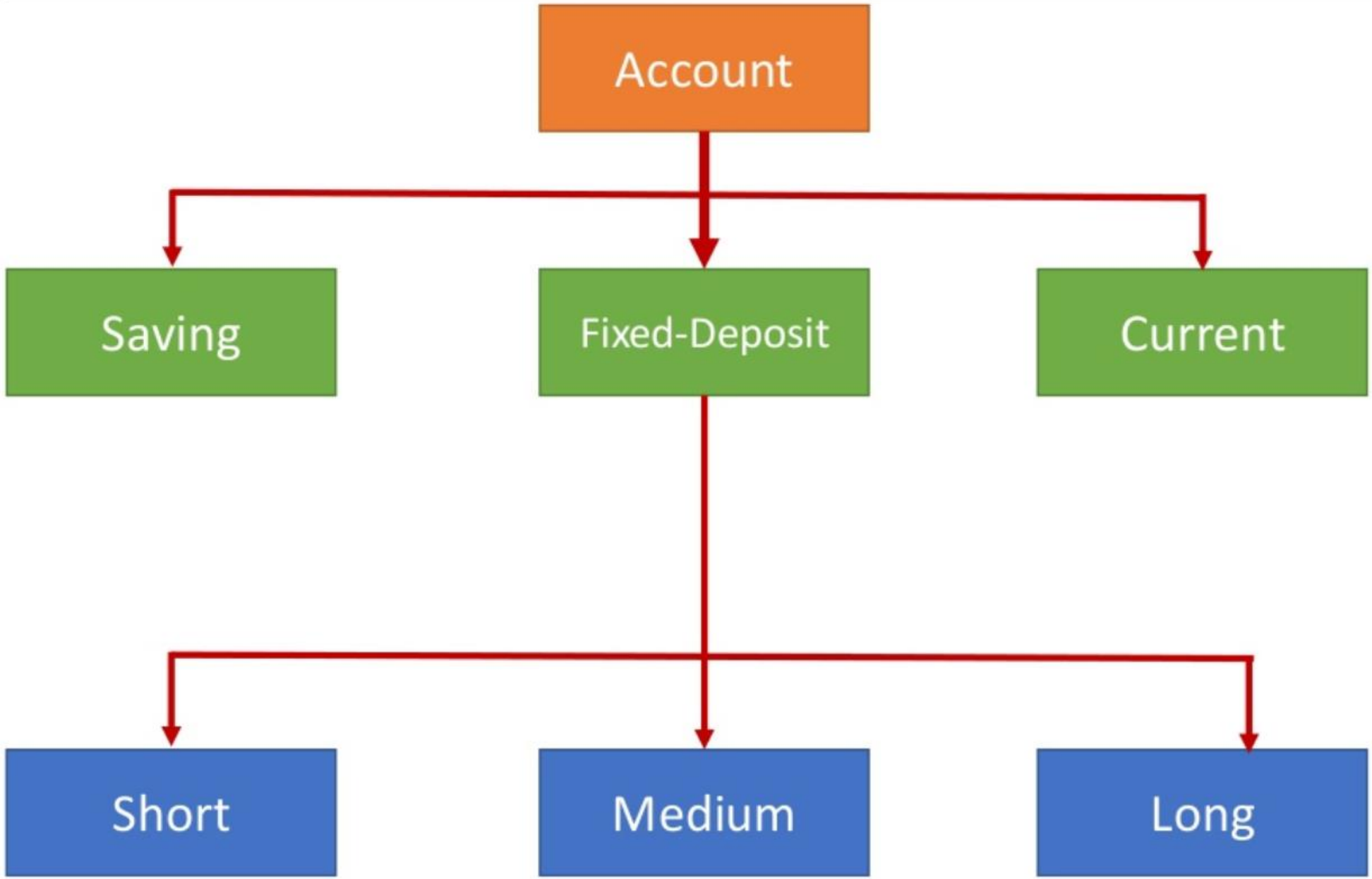
Types of Inheritance

- **3) Hierarchical Inheritance**



Here **Class A** acts as the **parent** for sub classes **Class B, Class C** and **Class D**

Hierarchical Inheritance(Real time Example)



Hierarchical Inheritance Example

```
class A {
  int data=10;
}
class B extends A{
}
class C extends A  {
}
class D extends A  {
 public static void main(String args[]) {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
   System.out.println("Data in Class B is: "+obj1.data);
   System.out.println("Data in Class C is: "+obj2.data);
   System.out.println("Data in Class D is: "+obj3.data);
  }
}
```

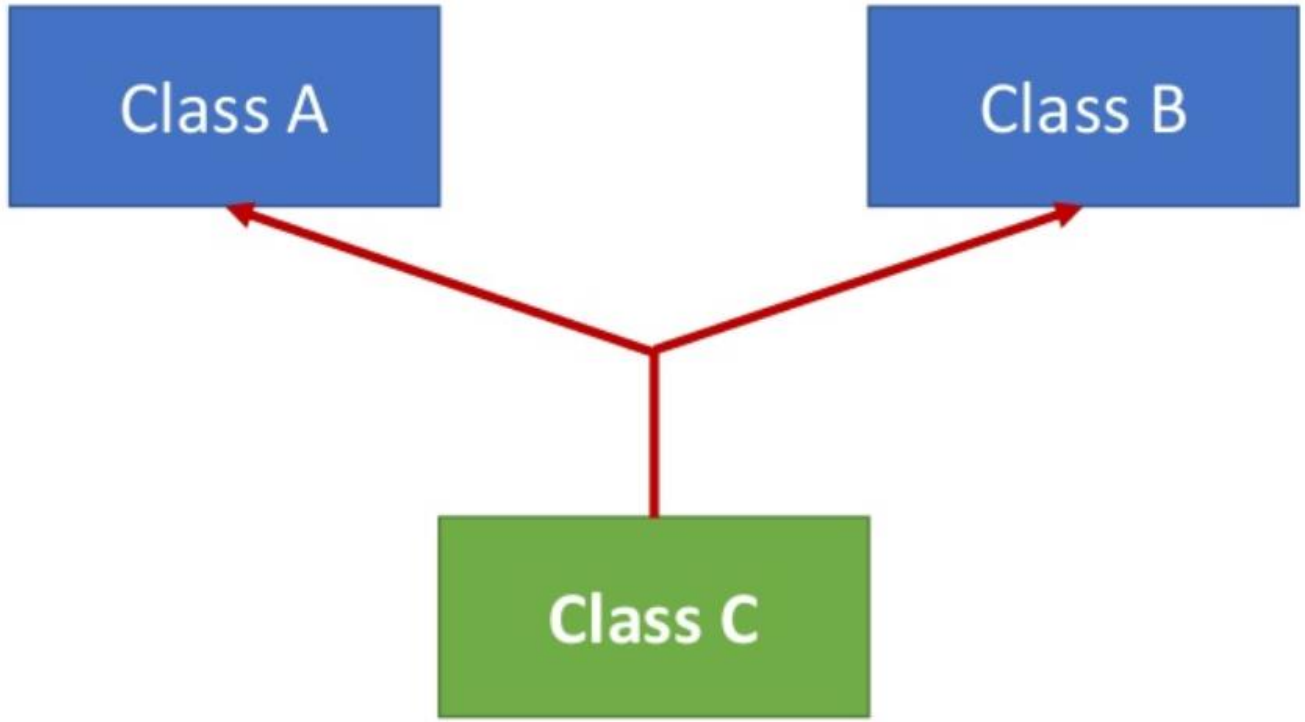
Another Hierarchical Inheritance Example

```
class A {
  void methodA() {
   System.out.println("Method of Class A");   }
}
class B extends A{ }
class C extends A{ }
class D extends A  {
 public static void main(String args[]) {
    B obj1 = new B();
    C obj2 = new C();
    D obj3 = new D();
    obj1.methodA();
    obj2.methodA();
    obj3.methodA();
  }
}
```

Types of Inheritance

- **4) Multiple Inheritance**
- In java programming, multiple and hybrid inheritance is not supported through classes but supported through interface only. We will learn about interfaces later.
- **Q) Why multiple inheritance is not supported in java?**
- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Multiple Inheritance



Here Class A, B and C are three Classes. The C Class inherits A and B classes.in other words Class C inherit the properties of both Class A and Class B.

Multiple Inheritance

- **Why 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.

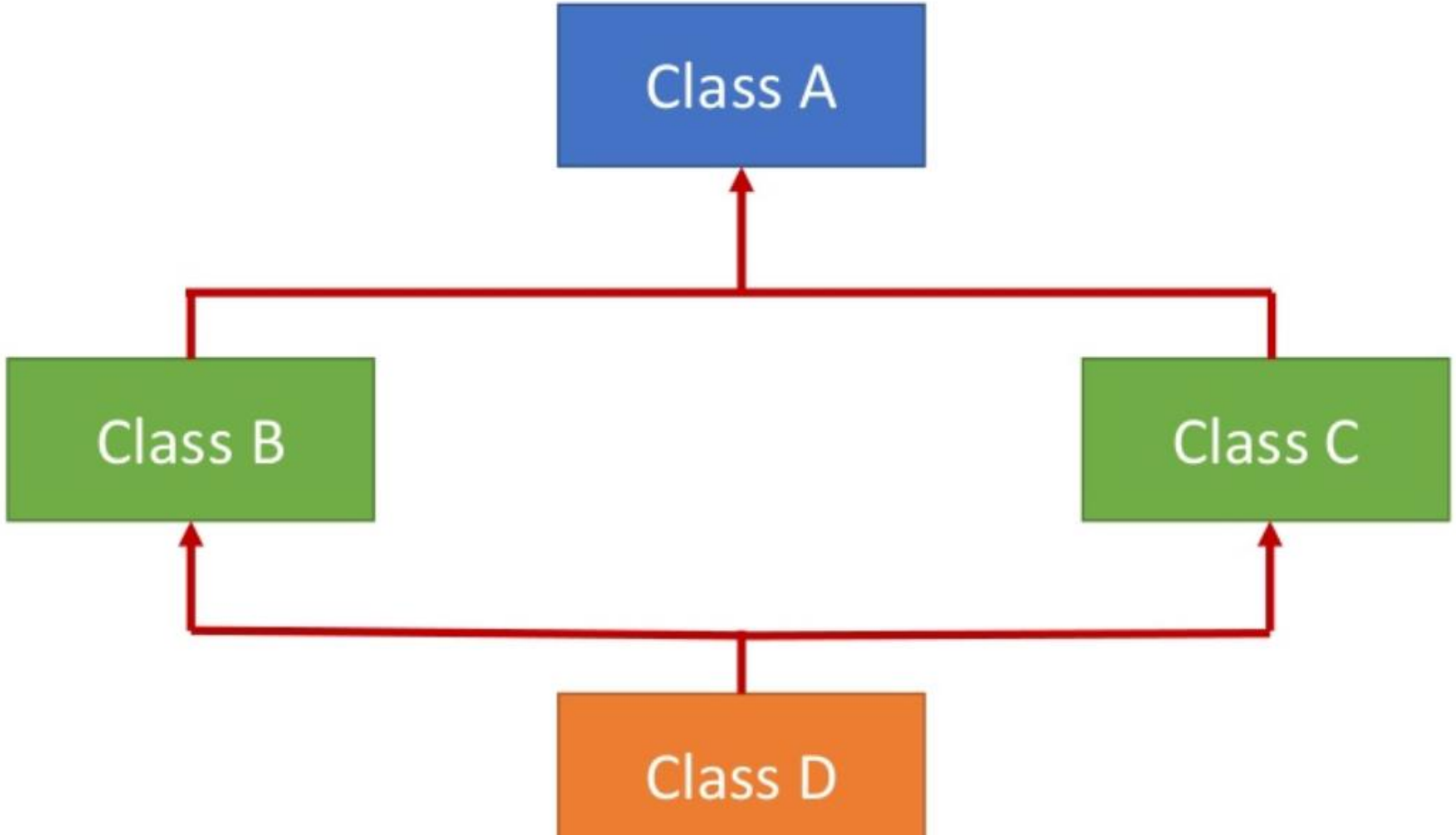
Multiple Inheritance Example

```
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?
 }
}
```

Types of Inheritance

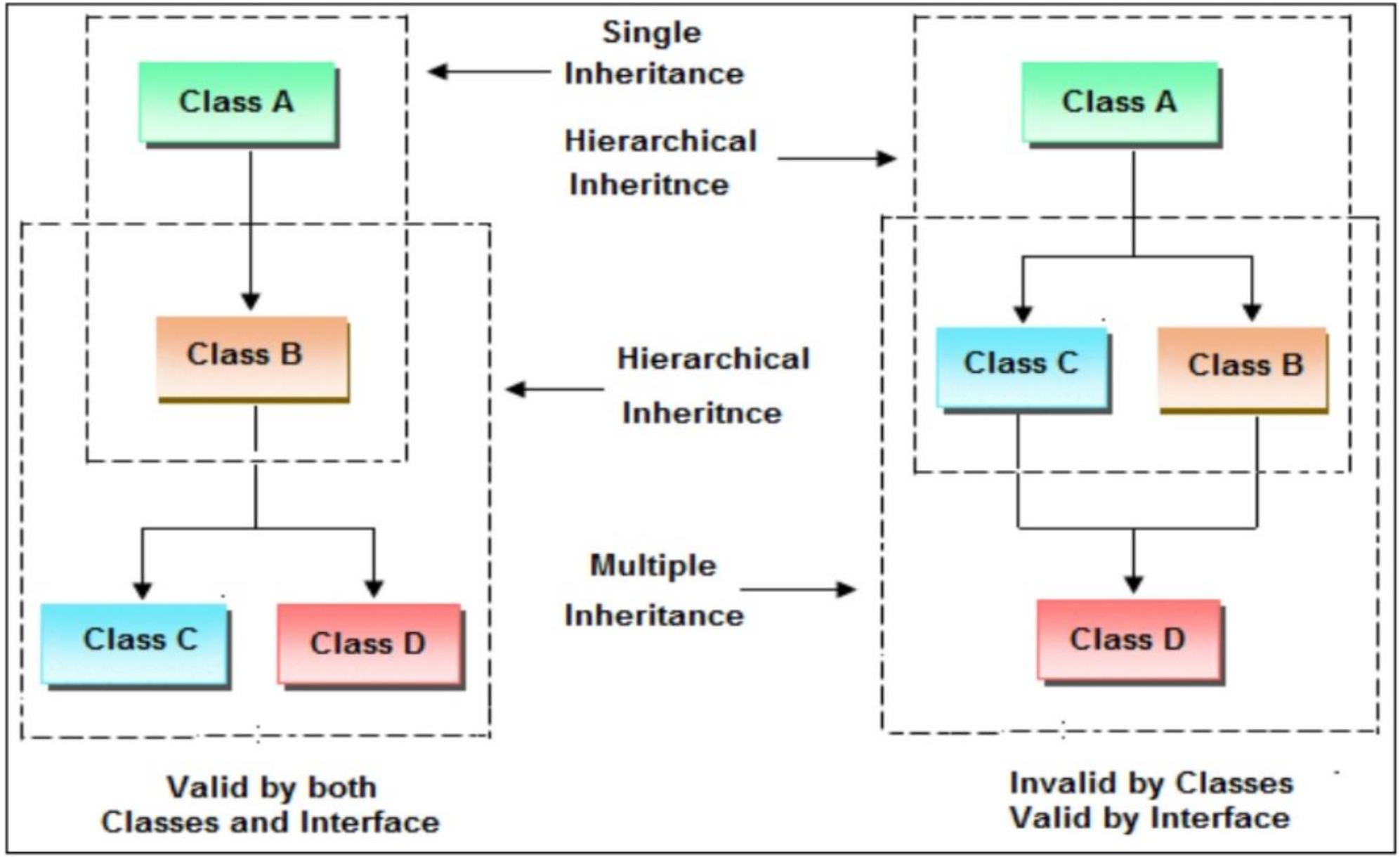
- **5) Hybrid inheritance**
- Any combination of previous three inheritance (single, hierarchical and multi level) is called as hybrid inheritance.



Hybrid Inheritance

- In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple** inheritance
- A typical flow diagram would look like in the previous slide.
- A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right.
- By using **interfaces** we can have multiple as well as **hybrid inheritance** in Java.

Hybrid inheritance



Inheritance in Java

- **Advantage of inheritance**
- If we develop any application using concept of Inheritance than that application have following advantages,
- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

Inheritance in Java

- **Disadvantages of Inheritance**
- Inheritance base class and child classes are tightly coupled. Hence If you change the code of parent class, it will get affects to the all the child classes.
- In class hierarchy many data members remain unused and the memory allocated to them is not utilized. Hence affect performance of your program if you have not implemented inheritance correctly.

Method Overriding in Java

- Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.

OR

- In other words If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java.**

OR

- 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.

Method Overriding in Java

- Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.

OR

- In other words If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java.**

OR

- 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.

Without Inheritance Method Overriding is not Possible

Method Overriding in Java

- **Advantage 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

- **Rules for 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).

Problem Without Method Overriding

```
class Vehicle{
  void run() {
   System.out.println("Vehicle is running");
   }
}
class Bike extends Vehicle{

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

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

Example of Method Overriding

```
class Vehicle{
  void run(){
  System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
  void run(){
  System.out.println("Bike is running safely");}

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

Example of Method Overriding

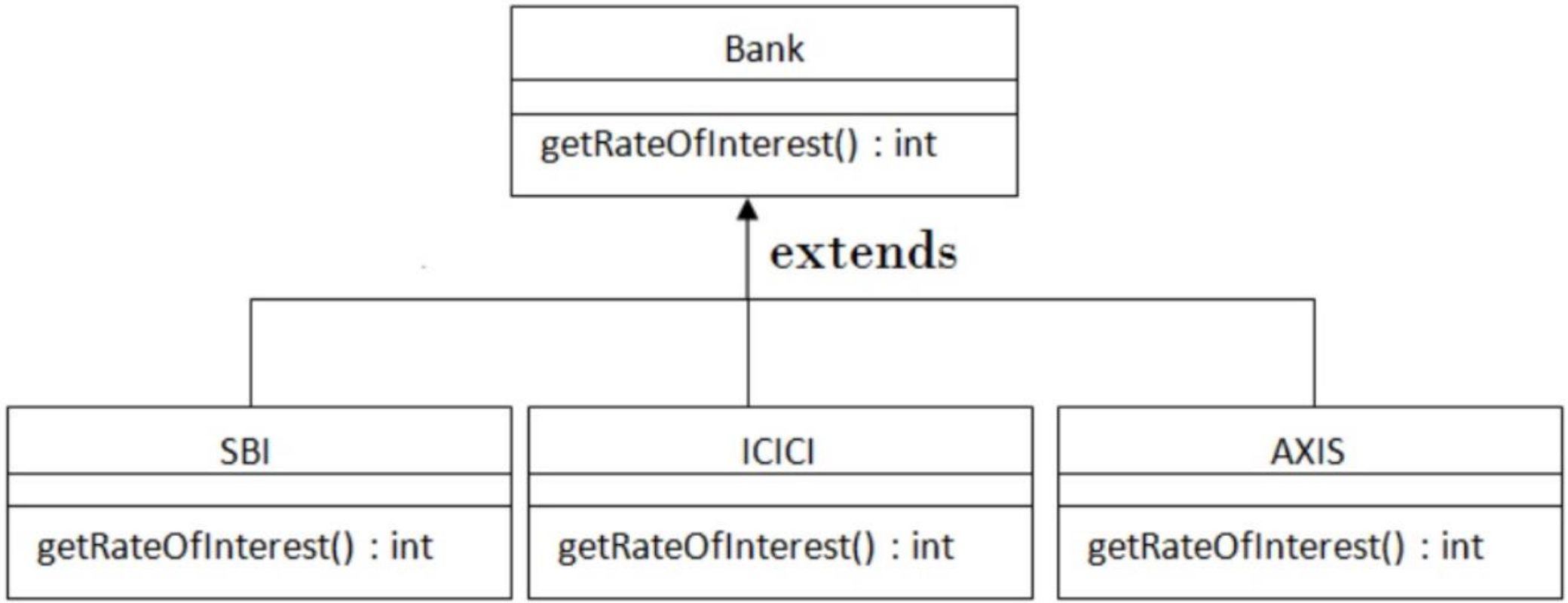
```
class Vehicle{
  void run(){
  System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
  void run(){
  System.out.println("Bike is running safely");}

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

Method name are same

Real Example of Java Method Overriding

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



Real Example of Java Method Overriding-1

```
class Bank{
  int getInterest(){return 0;}
}
class SBI extends Bank{
  int getInterest(){return 8;}
}

class ICICI extends Bank{
  int getInterest(){return 7;}
}
class AXIS extends Bank{
  int getInterest(){return 9;}
}
```

Real Example of Java Method Overriding-2

```
class Test{
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.getInterest());
  System.out.println("ICICI Rate of Interest: "+i.getInterest());
  System.out.println("AXIS Rate of Interest: "+a.getInterest());
  }
}
```

Output is:

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

Method Overriding (Same Argument List)

```
public class Base {
    public int calculate(int num1,int num2) {
      return num1+num2;
    }
}
class Derived extends Base {
    public int calculate(int num1,int num2) {
      return num1*num2;
    }
    public static void main(String[] args) {
      Derived b1 = new Derived();
      int result = b1.calculate(10, 10);
      System.out.println("Result : " + result);
    }
}
```

Method Overriding (Same Argument List)

```
public class Base {
    public int calculate(int num1,int num2) {
        return num1+num2;
    }
}
class Derived extends Base {
    public int calculate(int num1,int num2) {
        return num1*num2;
    }
    public static void main(String[] args) {
        Derived b1 = new Derived();
        int result = b1.calculate(10, 10);
        System.out.println("Result : " + result);
    }
}
```

Both Methods Having same number of Parameters List

Argument List is also same as Parameter List

Output is:

Result : 100