

## Lecture 2 - Java Inheritance

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java  
Course Page: [\[Click\]](#)

# Introduction to Inheritance

## High-Level Ideas:

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

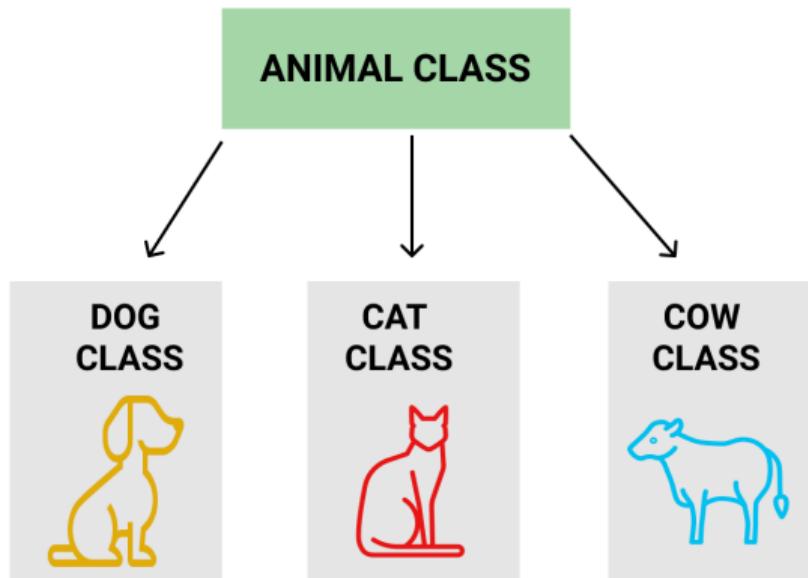


香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Introduction to Inheritance

An example of inheritance:



Dog, cat and cow are animals. Animals must eat, sleep, and breed babies.



# Introduction to Inheritance

Let's imagine that you are building a rocket for SpaceX.

Class RocketFalcon1 ()

public Void Engine ()

public Void Appearance ()

public Void ControlAlgorithm ()

.....



Class RocketFalcon9 ()

public Void Appearance ()

public Void Carry ()

.....



inherit



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Introduction to Inheritance

## Advantages:

- **Reusability.** When you inherit from an existing class, you can reuse methods and fields of the parent class.
- **Addition.** You can add new methods and fields in your current class also.



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Terminologies in Inheritance

## 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 that 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.



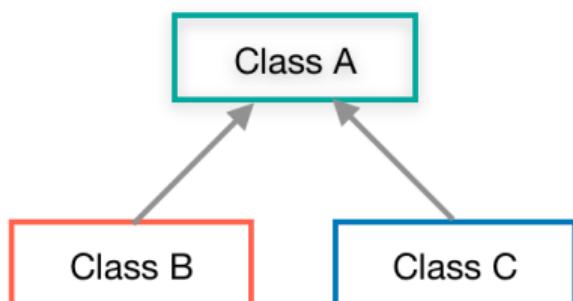
香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Java Inheritance

**Relationship.** Inheritance defines the **IS-A** relationship, e.g., parent-child relationship.

- *The syntax of Java Inheritance.* The `extends` keyword indicates that you are making a new class that derives from an existing class.

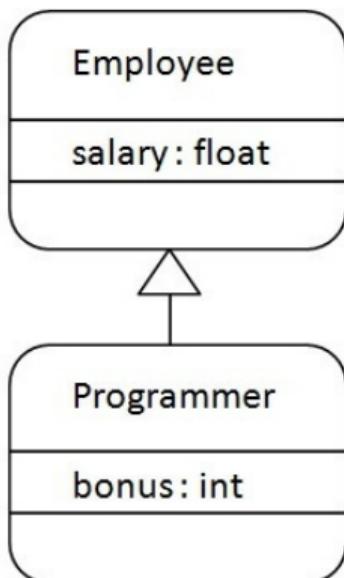


Hierarchical  
Inheritance

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

# Java Inheritance Example

The programmer is the **subclass** and the employee is the **superclass**. The relationship between the two classes is **Programmer IS-A Employee**.

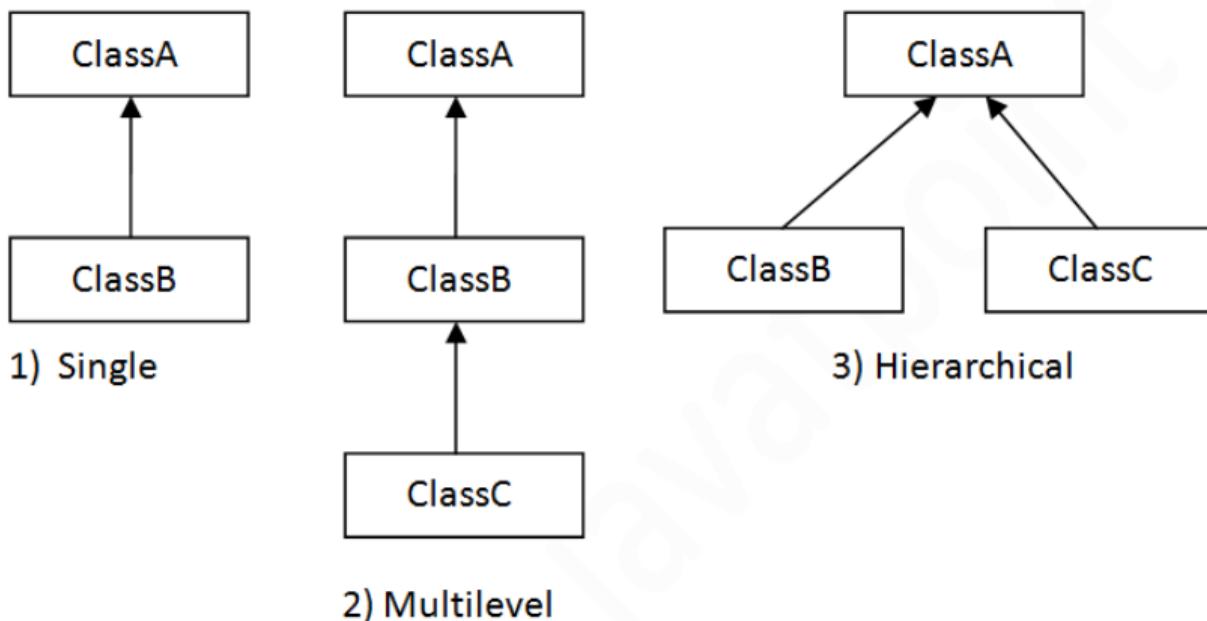


```
class Employee{  
    float salary=40000;  
}  
class Programmer extends Employee{  
    int bonus=10000;  
    public static void main(String args[]){  
        Programmer p=new Programmer();  
        System.out.println("Programmer salary is:"+p.salary);  
        System.out.println("Bonus of Programmer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

Shenzhen

# Types of inheritance in java

There can be three types of inheritance in Java: **single**, **multilevel**, and **hierarchical**.



# Single Inheritance Example

**Single Inheritance** happens when a class inherits another class.

In this example, the Dog class inherits the Animal class, so there is a 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();
    }
}
```



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Single Inheritance Example

**Single Inheritance** happens when a class inherits another class.

In this example, the Dog class inherits the Animal class, so there is a 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();
    }
}
```

## Output:

barking...

eating...



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Multilevel Inheritance Example

**Multilevel Inheritance** happens when there is a chain of inheritance.

e.g., the BabyDog class inherits the Dog class which again inherits the Animal class.

```
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();
    }
}
```



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Multilevel Inheritance Example

**Multilevel Inheritance** happens when there is a chain of inheritance.

e.g., the BabyDog class inherits the Dog class which again inherits the Animal class.

```
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();
    }
}
```

## Output:

weeping...

barking...

eating...



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Hierarchical Inheritance Example

**Hierarchical Inheritance** happens when two or more classes inherit a single class.

E.g., Dog and Cat classes inherit 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(); //C.T.Error
    }
}
```



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Hierarchical Inheritance Example

**Hierarchical Inheritance** happens when two or more classes inherit a single class.

E.g., Dog and Cat classes inherit 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(); //C.T.Error
    }
}
```

## Output:

meowing...

eating...



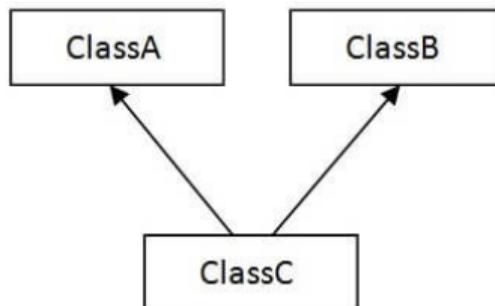
香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

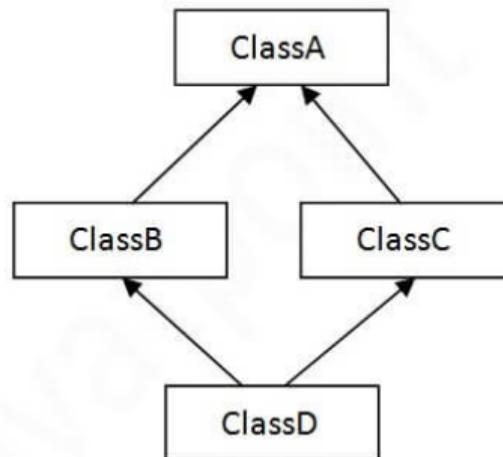
# Types of inheritance not supported in Java

**Multiple Inheritances** happens when one class inherits multiple classes.

**Hybrid Inheritances** happens when one merges multiple inheritances with other structures. Multiple or Hybrid inheritances are **not supported** in Java.



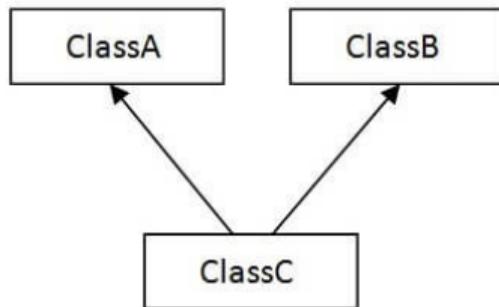
4) Multiple



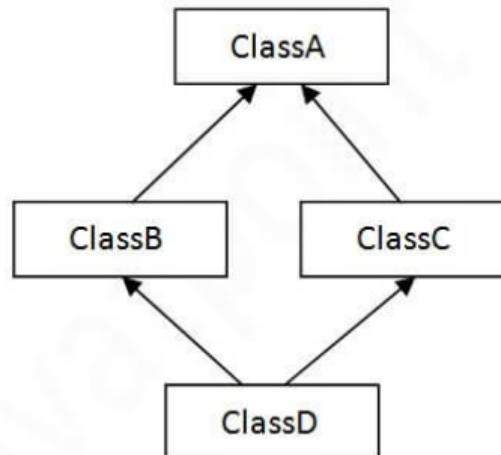
5) Hybrid

## Types of inheritance not supported in java

Consider a scenario (in 4) above where A, B, and C are three classes. **The C class inherits the A and B classes.** If A and B classes have the **same method** and you call it from the C class object, there will be **ambiguity** in calling the method of A or B class.



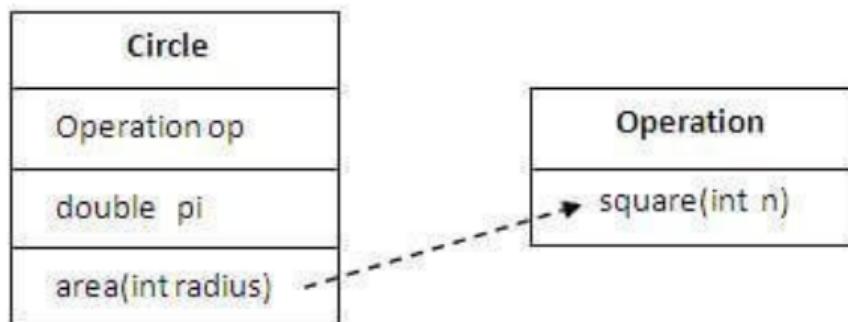
4) Multiple



5) Hybrid

# Java Aggregation

If a class has an **entity reference**, it is known as **Aggregation**. Aggregation represents **HAS-A** relationship.



Why use Aggregation? For Code Reusability.



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Java Aggregation

```
class Operation{  
    int square(int n){  
        return n*n;  
    }  
}  
  
class Circle{  
    Operation op;//aggregation  
    double pi=3.14;  
  
    double area(int radius){  
        op=new Operation();  
        int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).  
        return pi*rsquare;  
    }  
}
```

```
public static void main(String args[]){  
    Circle c=new Circle();  
    double result=c.area(5);  
    System.out.println(result);  
}  
}
```



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Java Aggregation

```
class Operation{  
    int square(int n){  
        return n*n;  
    }  
}  
  
class Circle{  
    Operation op;//aggregation  
    double pi=3.14;  
  
    double area(int radius){  
        op=new Operation();  
        int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).  
        return pi*rsquare;  
    }  
}
```

```
public static void main(String args[]){  
    Circle c=new Circle();  
    double result=c.area(5);  
    System.out.println(result);  
}  
}
```

**Output:** 78.5



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Java Interface

The **interface** in Java is a mechanism to **achieve abstraction**. There can be only abstract methods in the Java interface, not the method body.

```
interface printable{
    void print();
}

class A6 implements printable{
    public void print(){System.out.println("Hello");}
}

public static void main(String args[]){
    A6 obj = new A6();
    obj.print();
}
```

**Output:** Hello

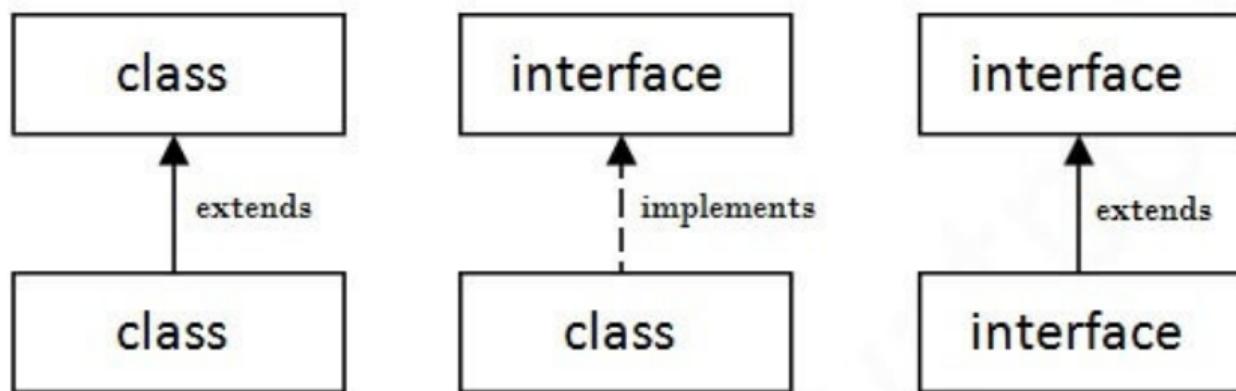


香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Java Interface

The **interface** in Java is a mechanism to **achieve abstraction**. There can be only abstract methods in the Java interface, not the method body.



# Question and Answering (Q&A)



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen