Java (Core Java)

Inheritance

Today's Topics

- OOPS concepts
- Inheritance and Aggregation
  - What is Aggregation (HAS-A)
  - What is inheritance (IS-A)
  - Types of inheritance
  - Demonstrate how methods from superclass are available
- Overriding in inheritance

OOPS - Concepts

- OOPS – Object Oriented Programming Systems
- Various OOP concepts
  - Inheritance and Aggregation
  - Encapsulation
  - Polymorphism
  - Abstraction

Inheritance and Aggregation

```java
public class Dog extends Animal{
    String breedOfDog;

    // Constructor
    public Dog() {
        System.out.println("A dog is created");
    }

    public void bark() {
        System.out.println("Barking");
    }

    // Overloading
    public void bark(String pitch) {
        System.out.println("Barking " + pitch);
    }
}
```

- INHERITANCE (IS-A)
- Dog **IS-A** Animal
- Dog extends Animal

```java
public class ClassRoom {
    static int noOfAC = 1;
    static int noOfProjectors = 1;
    static int noOfWhiteboards = 1;
    static String brandOfProjector = "Canon";
    int capacity = 0;
    int noOfFans = 0;

    Teacher teacher;
    Student [] students;
}
```

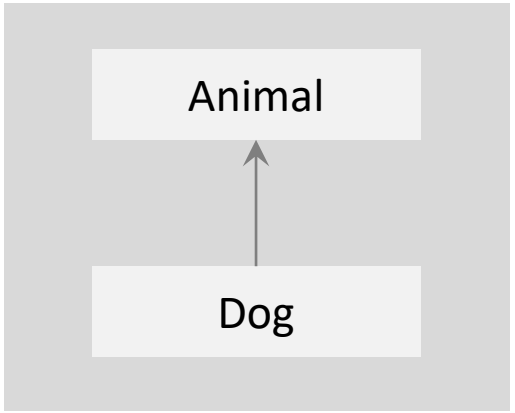
- AGGREGATION (HAS-A)
- ClassRoom **HAS-A** Teacher
- ClassRoom **HAS-A** number of Students

Inheritance - Example

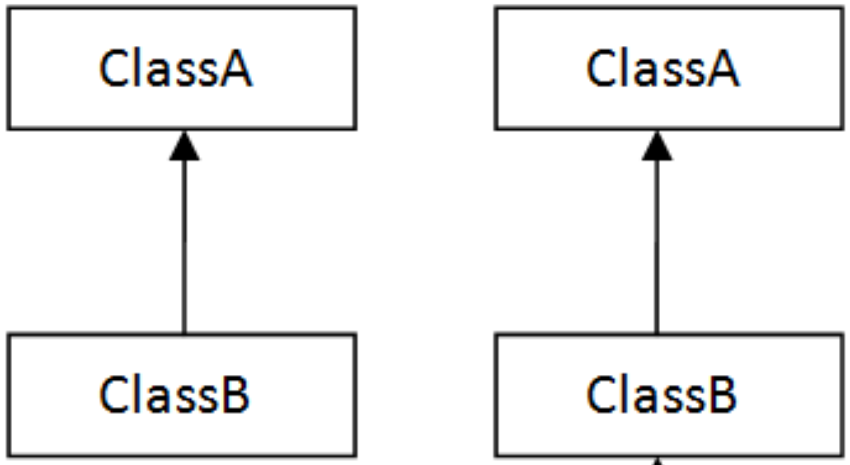
```
public class Dog extends Animal{
    String breedOfDog;

    // Constructor
    public Dog() {
        System.out.println("A dog is created");
    }

    public void bark() {
        System.out.println("Barking");
    }

    // Overloading
    public void bark(String pitch) {
        System.out.println("Barking " + pitch);
    }
}
```

This is how you incorporate inheritance

- Animal is the super class, or the base class
- Dog is the sub class or the inherited class

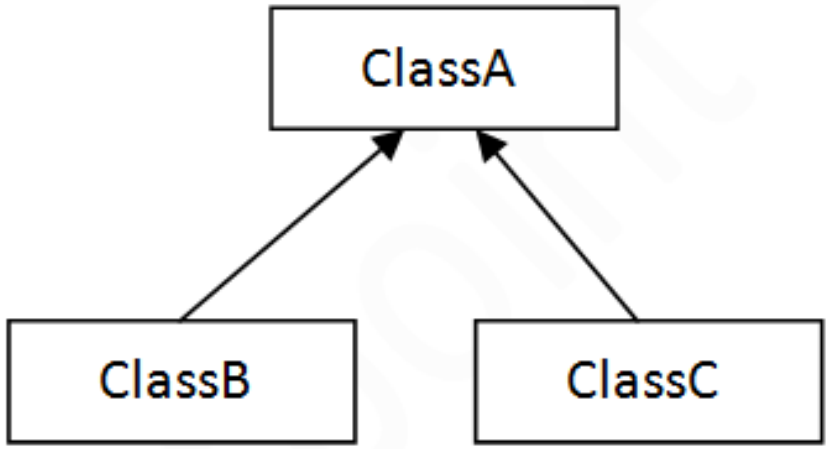


Types of Inheritance



1) Single

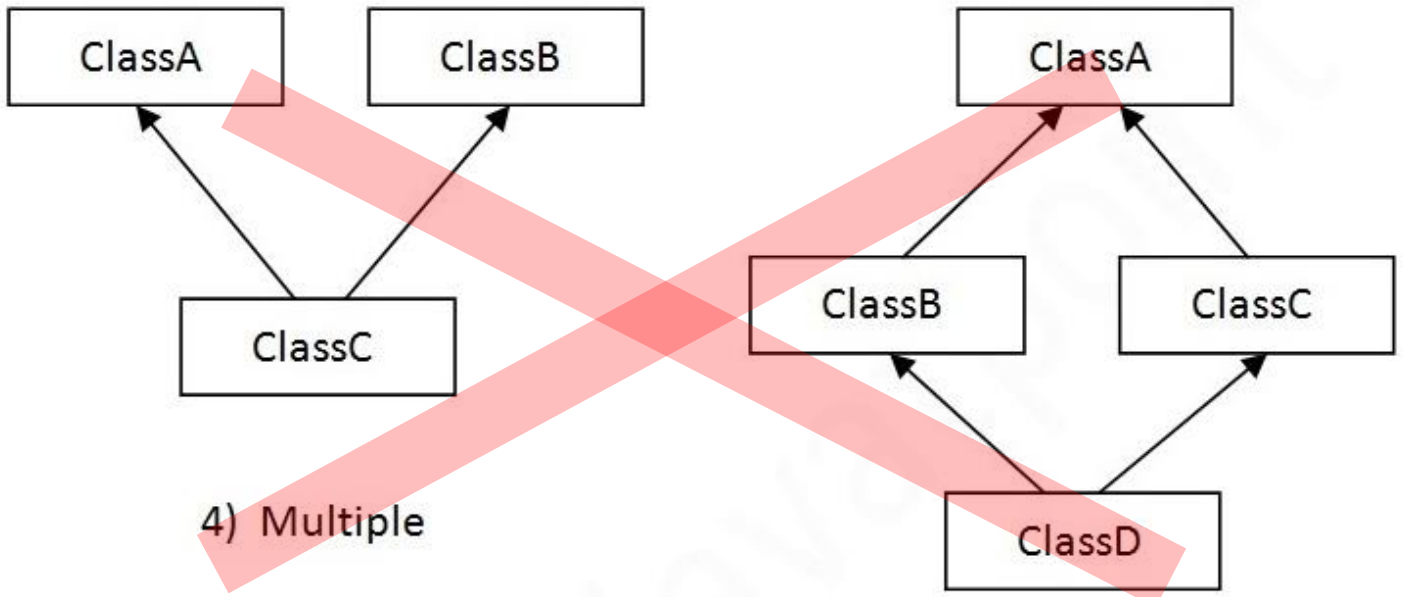
2) Multilevel



3) Hierarchical

Let us discuss some examples…

Types of Inheritance – Not Allowed



Why are these not allowed?

What happens during inheritance?

- Non private attributes from super class are available to sub class
- Non private methods from super class are available to sub class

Let us see some examples

Overriding methods

- Sometimes behavior belongs to both super class as well as sub class but implementation is different.

```
public class Animal {
    String color;

    public Animal() {
        System.out.println("\n\nAn animal is created");
    }

    public void eat() {
        System.out.println("Eating");
    }
}
```

Every Animal eats, and so does cat. But seems like cat has a special way of eating, and hence it overrides the eat method of Animal

```
public class Cat extends Animal{
    // Overriding eat()
    public void eat() {
        super.eat(); // What is this?
        System.out.println("Actually slurping milk");
    }

    public void mew() {
        System.out.println("Mew");
    }

    // Constructor
    public Cat() {
        System.out.println("A cat is created");
    }
}
```

Can you think of some other examples?

Overriding methods - continued

If you have instance of the sub class, then it will invoke the overloaded method. Once method is overridden in sub class, there is no way to invoke the method of parent class through object of sub class

To be able to override methods, following needs to be same for both super as well as sub class

- Method name
- Return type
- Arguments

```
public class Animal {
    String color;

    public Animal() {
        System.out.println("\n\nAn animal is created");
    }

    public void eat() {
        System.out.println("Eating");
    }
}
```

```
public class Cat extends Animal{
    // Overriding eat()
    public void eat() {
        super.eat(); // What is this?
        System.out.println("Actually slurping milk");
    }

    public void mew() {
        System.out.println("Mew");
    }

    // Constructor
    public Cat() {
        System.out.println("A cat is created");
    }
}
```

Let’s relate what we have learnt so far

```
public class Shape {
    int noOfSides;
    int lengthOfOneSide;

    public void perimeter() {
        System.out.println("Perimeter is : " + this.noOfSides * this.lengthOfOneSide);
    }

    public void area() {
        System.out.println("Not enough info..");
    }

    // Parameterized Constructor
    Shape (int noOfSides, int lengthOfOneSide){
        this.noOfSides = noOfSides;
        this.lengthOfOneSide = lengthOfOneSide;
    }

    // Default constructor
    Shape(){
    }
}
```

Questions:

- I need to create objects called Square, and Rectangle.
- What could be the inheritance hierarchy?
- What type of inheritance it is?
- Which method needs to be overridden where?
- I want to print the generic shape name like quadrilateral, triangle etc.. Every time a shape with a size is created. How do I accomplish this?

Recap of this and super

- `super` points to the superclass of the current instance
- `super` keyword can only be used in subclasses
- `this` keyword points to the current instance of the class

Basically, `subclass.super` **is same as** `superclass.this`

```
// Copy Constructor
public Student(Student refStudent) {
    this.rollNo = refStudent.rollNo;
    this.nameOfStudent = refStudent.nameOfStudent;
    this.yearOfBirth = refStudent.yearOfBirth;
    this.age = refStudent.age;
    this.gender = refStudent.gender;
}
```

Usage of `this` for copy constructor