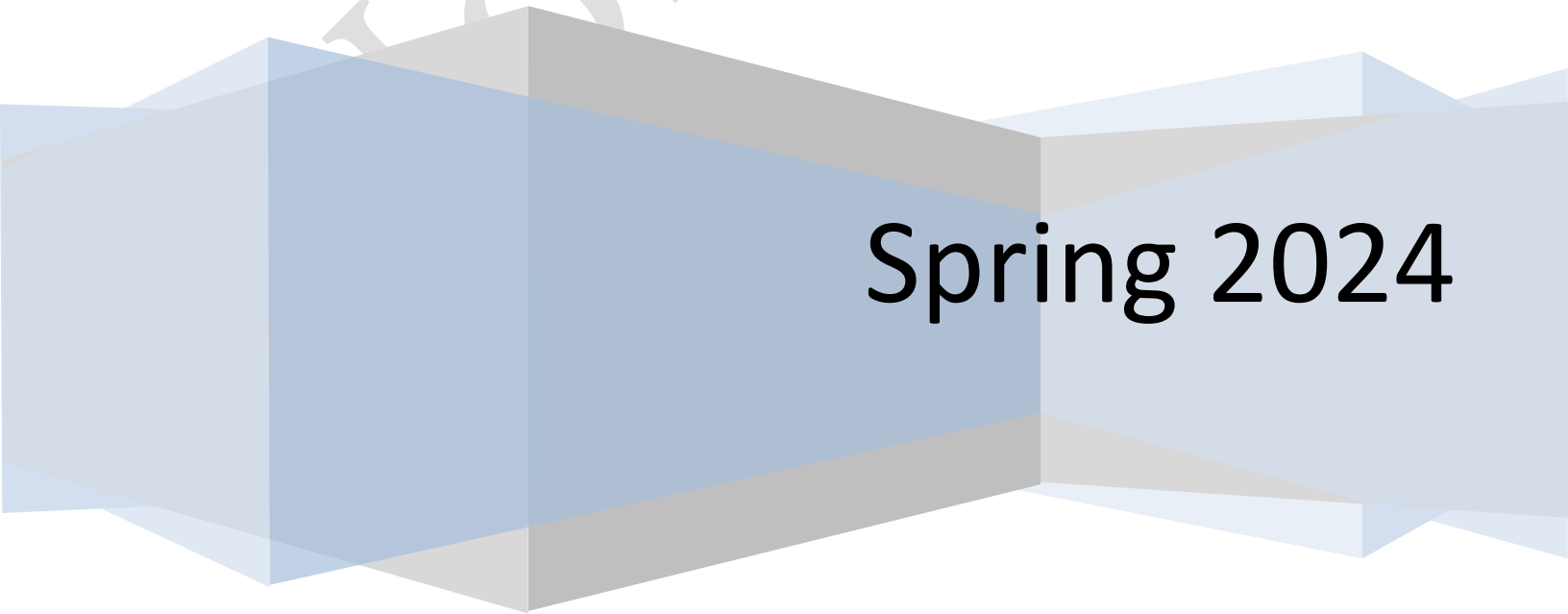
Northern University, Nowshera

Inheritance

Week # 11 - Lecture 21- 22



Spring 2024

Learning Objectives:

1. **Assignment Solution**
2. **Inheritance (recap)**
3. **is-a relationship**
4. **protected keyword**
5. **method overriding**
6. **types of inheritance**
    a. **single inheritance**
    b. **multi-level inheritance**
    c. **Hierarchical inheritance**
7. **Rules of inheritance**

1. Assignment Solution

Q#1: Write code for the following classes.

Animal Class: Animal class has attributes: String eats and int no of legs. Write constructors to set attributes eat and no of legs. Write a function getNoOfLegs() and getEats() to return required attributes.

Cat Class: Write another class cat having attribute color of type string that inherits animal class. Write appropriate constructors and a function for getColor() to return color property of cat. Use the concept of calling constructors of parent class in child class.

Example main () is given below. Complete the code by considering the given main().

```
public class AnimalInheritanceTest {
```

```
	public static void main(String[] args) {
		Cat cat = new Cat("milk", 4, "black");
		System.out.println("Cat eats " + cat.getEats());
		System.out.println("Cat has " + cat.getNoOfLegs() + " legs.");
		System.out.println("Cat color is " + cat.getColor());
	}

}
```

Assignment Solution:

```
class Animal
{

    private String eats;
    private int noOfLegs;
    public Animal()
    {

    }
    public Animal(String e, int no)
    {
        this.eats=e;
        this.noOfLegs=no;
    }
    public int getNoOfLegs()
    {
        return noOfLegs;
    }
    public String getEats()
    {
        return eats;
    }
}
class Cat extends Animal
{
    private String color;
    public Cat()
    {
        super();
        color=null;
    }
    public Cat(String e, int no, String col)
    {
        super(e,no);
        this.color=col;
    }
    public String getColor()
    {
        return color;
    }
```

```
}
class AnimalInheritanceTest {
    public static void main(String[] args) {
        Cat cat = new Cat("milk", 4, "black");
        System.out.println("Cat eats " + cat.getEats());
        System.out.println("Cat has " + cat.getNoOfLegs() + " legs.");
        System.out.println("Cat color is " + cat.getColor());
    }
}
```

2. Inheritance Revision

In below example of inheritance, class Bicycle is a base class, class MountainBike is a derived class which extends Bicycle class and class Test is a driver class to run program.

Example 1: A Bicycle example

```
//Java program to illustrate the
// concept of inheritance
// base class
class Bicycle {
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed) {
        this.gear = gear;
        this.speed = speed;
    }



    // the Bicycle class has three methods
    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString() {
        return ("No of gears are " + gear
                + "\n"
```

```
                + "speed of bicycle is " + speed);
    }
}

// derived class
class MountainBike extends Bicycle {

    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear, int speed,
                        int startHeight) {
        // invoking base-class(Bicycle) constructor


        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }

    // overriding toString() method
    // of Bicycle to print more info
    @Override
    public String toString() {
        return (super.toString() +
                "\nseat height is " + seatHeight);
    }

}
// driver class
public class Test {
    public static void main(String args[]) {

        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());

    }
}
```

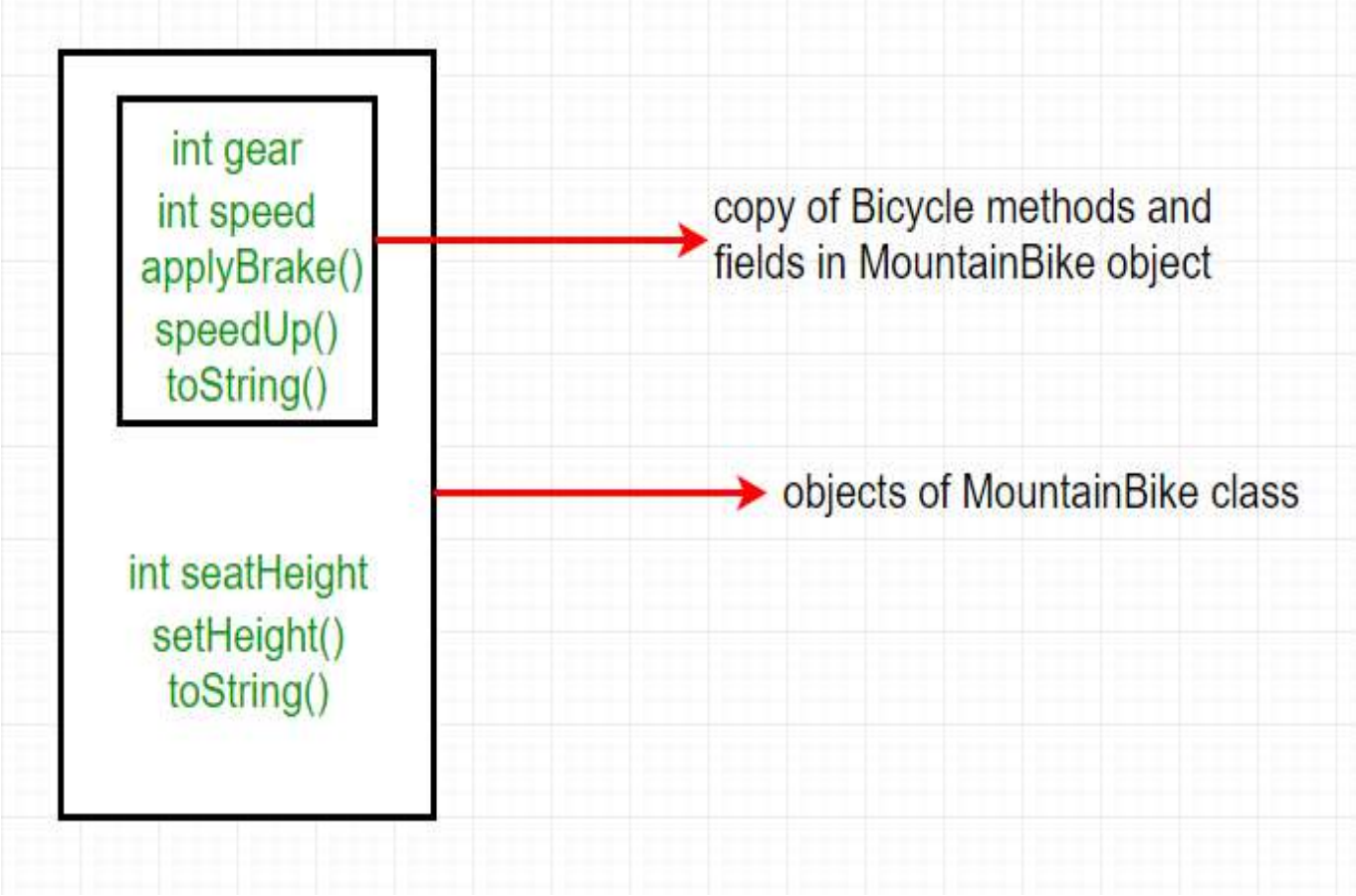
OUTPUT

```
No of gears are 3
speed of bicycle is 100
seat height is 25
Brake applied
```

```
No of gears are 3
speed of bicycle is 80
seat height is 25
```

In above program, when an object of MountainBike class is created, a copy of the all methods and fields of the superclass acquire memory in this object. That is why, by using the object of the subclass we can also access the members of a superclass.
Please note that during inheritance only object of subclass is created, not the superclass. For more, refer Java Object Creation of Inherited Class.

Illustrative image of the program:



3. is-a relationship

Inheritance is an **is-a** relationship. We use inheritance only if an **is-a** relationship is present between the two classes. Here are some examples:

- A car is a vehicle.
- Orange is a fruit.
- A surgeon is a doctor.
- A dog is an animal.

Example 2: is- a relationship

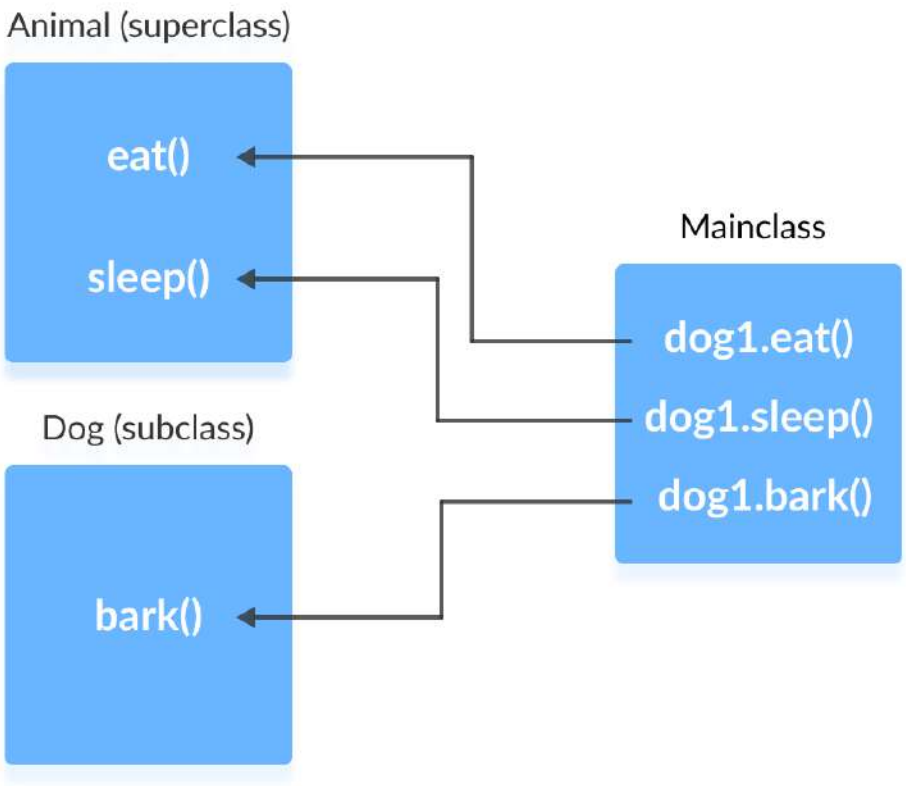
```
class Animal {

   public void eat() {
      System.out.println("I can eat");
   }

   public void sleep() {
      System.out.println("I can sleep");
   }
}

class Dog extends Animal {
   public void bark() {
      System.out.println("I can bark");
   }
}

class Main {
   public static void main(String[] args) {

      Dog dog1 = new Dog();

      dog1.eat();
      dog1.sleep();

      dog1.bark();
   }
}
```

OUTPUT

```
I can eat
I can sleep
I can bark
```

Here, we have inherited a subclass Dog from superclass Animal. The Dog class inherits the methods eat() and sleep() from the Animal class.

Hence, objects of the Dog class can access the members of both the Dog class and the Animal class.



4. protected Keyword

We learned about `private` and `public` access modifiers in previous lectures.

- `private` members can be accessed only within the class
- `public` members can be accessed from anywhere

You can also assign methods and fields `protected`. Protected members are accessible

- from within the class
- within its subclasses

within the same package

Here's a summary from where access modifiers can be accessed.

	Class	Package	subclass	World
public	Yes	Yes	Yes	Yes
private	Yes	No	No	No
protected	Yes	Yes	Yes	No

Example 3: protected Keyword

```
class Animal {
   protected String type;
   private String color;

   public void eat() {
      System.out.println("I can eat");
   }

   public void sleep() {
      System.out.println("I can sleep");
   }

   public String getColor(){
      return color;
```

```
    }

    public void setColor(String col){
        color = col;
    }
}

class Dog extends Animal {
    public void displayInfo(String c){
        System.out.println("I am a " + type);
        System.out.println("My color is " + c);
    }
    public void bark() {
        System.out.println("I can bark");
    }
}

class Main {
    public static void main(String[] args) {

        Dog dog1 = new Dog();
        dog1.eat();
        dog1.sleep();
        dog1.bark();

        dog1.type = "mammal";
        dog1.setColor("black");
        dog1.displayInfo(dog1.getColor());
    }
}
```

OUTPUT

```
I can eat
I can sleep
I can bark
I am a mammal
My color is black
```

Here, the `type` field inside the `Animal` class is protected. We have accessed this field from the `Main` class using

```
dog1.type = "mammal";
```

It is possible because both the `Animal` and `Main` classes are in the same package (same file).

5. Java Method overriding

From the above examples, we know that objects of a subclass can also access methods of its super class. **What happens if the same method is defined in both the superclass and subclass?**

Well, in that case, the method in the subclass overrides the method in the superclass. For example:

Example 4: Method overriding Example

```
class Animal {
   protected String type = "animal";

   public void eat() {
      System.out.println("I can eat");
   }

   public void sleep() {
      System.out.println("I can sleep");
   }
}

class Dog extends Animal {

      public void eat() {
      System.out.println("I eat dog food");
   }

   public void bark() {
```

```
        System.out.println("I can bark");
    }
}

class Main {
    public static void main(String[] args) {

        Dog dog1 = new Dog();
        dog1.eat();
        dog1.sleep();
        dog1.bark();
    }
}
```

OUTPUT

```
I eat dog food
I can sleep
I can bark
```

- Here, eat() is present in both the superclass Animal and subclass Dog. We created an object dog1 of the subclass Dog.
- When we call eat() using the dog1 object, the method inside the Dog is called, and the same method of the superclass is not called. This is called method overriding.

If we need to call the eat() method of Animal from its subclasses, we use the super keyword.

Example 5: use of Super Keyword

```
class Dog extends Animal {
    public Dog(){
        super();
        System.out.println("I am a dog");
    }

    public void eat() {
      super.eat();
      System.out.println("I eat dog food");
   }
```

```
   public void bark() {
      System.out.println("I can bark");
   }
}


class Main {
   public static void main(String[] args) {
      Dog dog1 = new Dog();

      dog1.eat();
      dog1.bark();
   }
}
```

OUTPUT

```
I am an Animal
I am a dog
I can eat
I eat dog food
I can bark
```

Here, we have used the super keyword to call the constructor using super(). Also, we have called the eat() method of Animal superclass using super.eat().

Note the difference in the use of super while calling constructor and method.

6. Types of inheritance in java

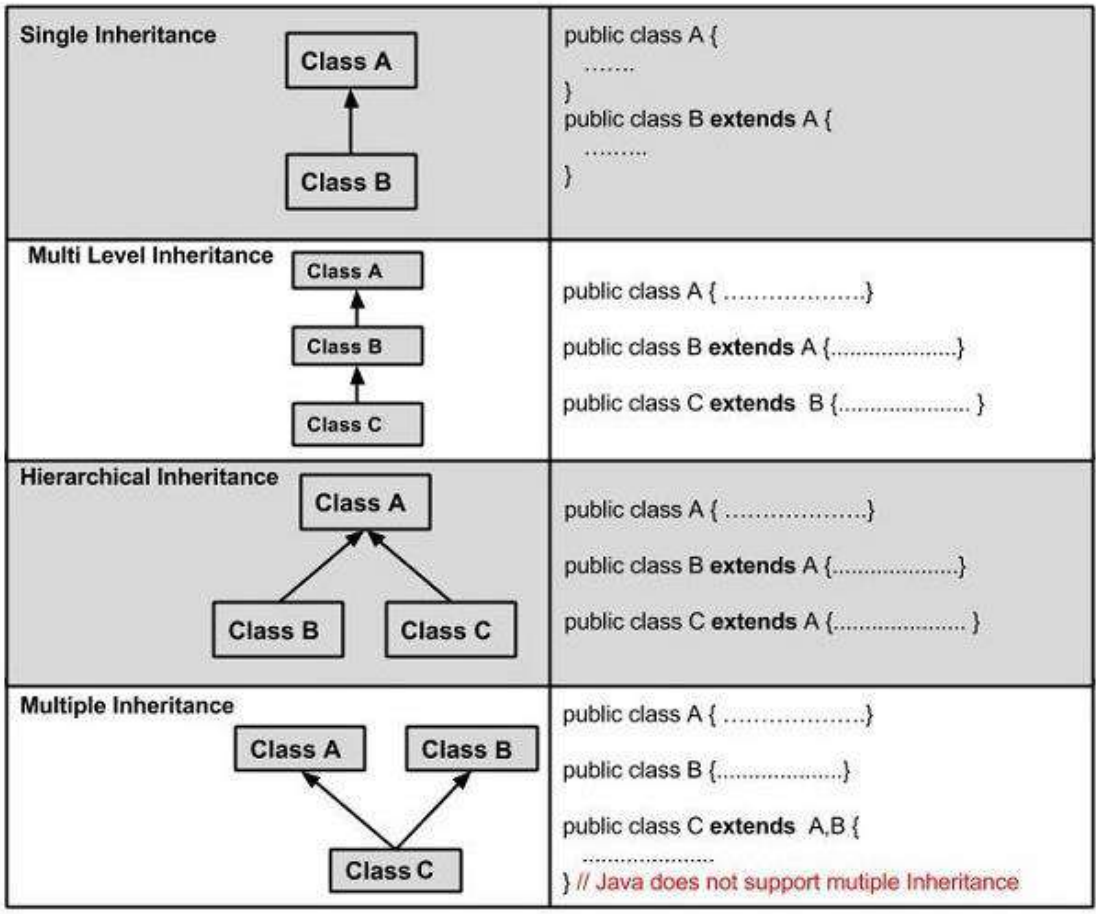
On the basis of class, there can be three types of inheritance in java:

- single,
- multilevel and
- hierarchical.

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

Types of Inheritance

There are various types of inheritance as demonstrated below.



Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

Example 6: 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...
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

Example 7: Multilevel inheritance Example

```
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...
```

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Example 8: 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...
```

Q) 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.

Example 9: multiple inheritance (not supported)

```
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?
    }
}
```

OUTPUT

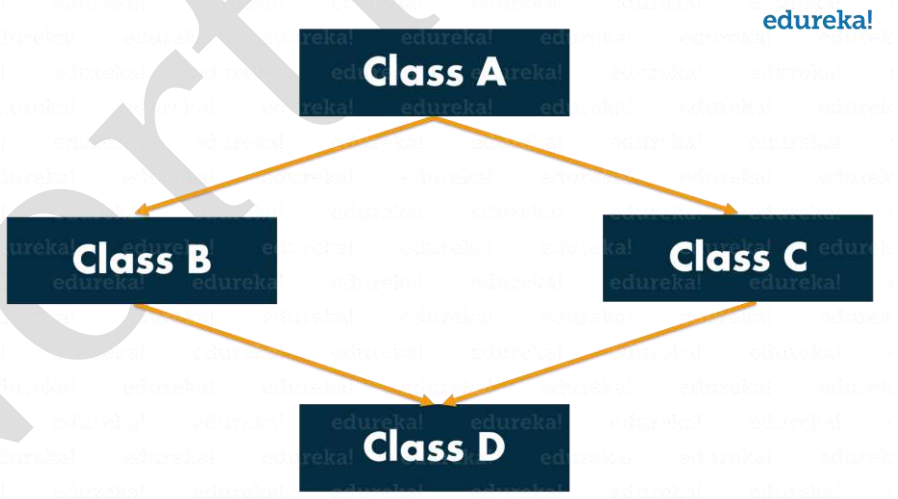
```
Compile Time Error
```

7. Rules of Inheritance in Java

Consider the following rules while inheriting classes.

RULE 1: Multiple Inheritance is NOT permitted in Java

.



Multiple inheritance refers to the process where one child class tries to extend more than one parent class. In the above illustration, Class A is a parent class for Class B and C, which are further extended by class D. This is results in Diamond Problem. Why?

Say you have a method show() in both the classes B and C, but with different functionalities. When Class D extends class B and C, it automatically inherits the characteristics of B and C

including the method show(). Now, when you try to invoke show() of class B, the compiler will get confused as to which show() to be invoked (either from class B or class C). Hence it leads to ambiguity.

For Example:

```
class Demo1 {
//code here
}

class Demo2 {
//code here
}

class Demo3 extends Demo1, Demo2 {
//code here
}

class Launch {
    public static void main(String args[]) {
//code here
    }
}
```

In the above code, Demo3 is a child class which is trying to inherit two parent classes Demo1 and Demo2. This is not permitted as it results in a diamond problem and leads to ambiguity.

RULE 2: Cyclic Inheritance is NOT permitted in Java

It is a type of inheritance in which a class extends itself and form a loop itself. Now think if a class extends itself or in any way, if it forms cycle within the user-defined classes, then is there any chance of extending the Object class. That is the reason it is not permitted in Java.

For Example:

```
class Demo1 extends Demo2 {
//code here
}
class Demo2 extends Demo1 {
```

```
//code here
}
```

In the above code, both the classes are trying to inherit each other's characters which is not permitted as it leads to ambiguity

RULE 3: <u>Constructors cannot be inherited in Java</u>

A constructor cannot be inherited, as the subclasses always have a different name.

```
class A {
  A();
}
class B extends A{
  B();
}
```

You can do only:

```
B b = new B();  // and not new A()
```

Methods, instead, are inherited with "the same name" and can be used. You can still use constructors from A inside B's implementation though:

```
class B extends A{
    B()
    {
     super();
    }
 }
```

Assignment #10

READ THE FOLLOWING INSTRUCTIONS CAREFULLY:

- Attempt the Assignment after reading the lecture notes, watching the videos lectures and doing self-learning of the topic from web.
- Submit your assignment via email /Google class room to respective teacher by the end of this week, dated 11th May 2020 before: 11:59 PM
- Your Assignment should be a single file either pdf or MS Word (handwritten scanned document) and must follow the naming format, your Name, Reg#, Section, subject and assignment number, i.e. AliAhmed_2018-Arid-0001-CS2A_OOP_Asgn4

Q#1: Write code for the following classes.

Person Class: Animal class has attributes: String name, address and int age. Write setperson() function to set values and getPerson() to Print attributes. Also write appropriate constructors.

Employee Class: Write another class Employee having attributes department and salary of type string and double. Write methods setEmployee(), getEmployee() and appropriate constructors for Employee class.

Student Class: Write a class Student having attributes registration number and GPA of type string and float. Also write setStudent(), getStudent() methods and required constructors.

Use the concept of inheritance to achieve the above functionality. Write a main() function to display the information of employee and student.

Note: Call the constructors/methods of parent class in child class where required.