upGrad

#LifeKoKaroLift

Object-Oriented Programming – Inheritance

upGrad

Course: Object-Oriented Analysis, Design & Programming

Lecture On: Object-Oriented Programming – Inheritance

Instructor: Sayan Nayak

upGrad

Topics covered in the previous class...

1. Different types of class relationships and how to represent them using code
2. Static members of a class and how to declare and access them
3. What is a static constructor (a static initializer block) and an initializer block?
4. What are enum data types and how to use them?
5. What are encapsulation and data hiding?
6. What are access modifiers or access specifiers? How to provide getters and setters?
7. What is the use of a private constructor and how to instantiate such classes?
8. Different types of nested classes and how to use them

Poll 1 (15 sec)

Which of the following is not a type of nested class?

1. Static nested class
2. Inner class
3. Singleton class
4. Local class

Poll 1 (15 sec)

Which of the following is not a type of nested class?

1. Static nested class
2. Inner class
3. **Singleton class**
4. Local class

upGrad

Homework Discussion

upGrad

Today's Agenda

- **Reduce Code Duplication Using Inheritance**
  - Recall Generalization Relationship and Inheritance
  - Types of Inheritance
  - Provide an Inheritance Relationship using the 'extends' Keyword
  - Object Class – Parent of all Classes
  - super Keyword – To Call a Parent Class Constructor
  - Abstract Classes and Abstract Methods
  - Abstract Class vs Interface

upGrad

Generalization Relationship

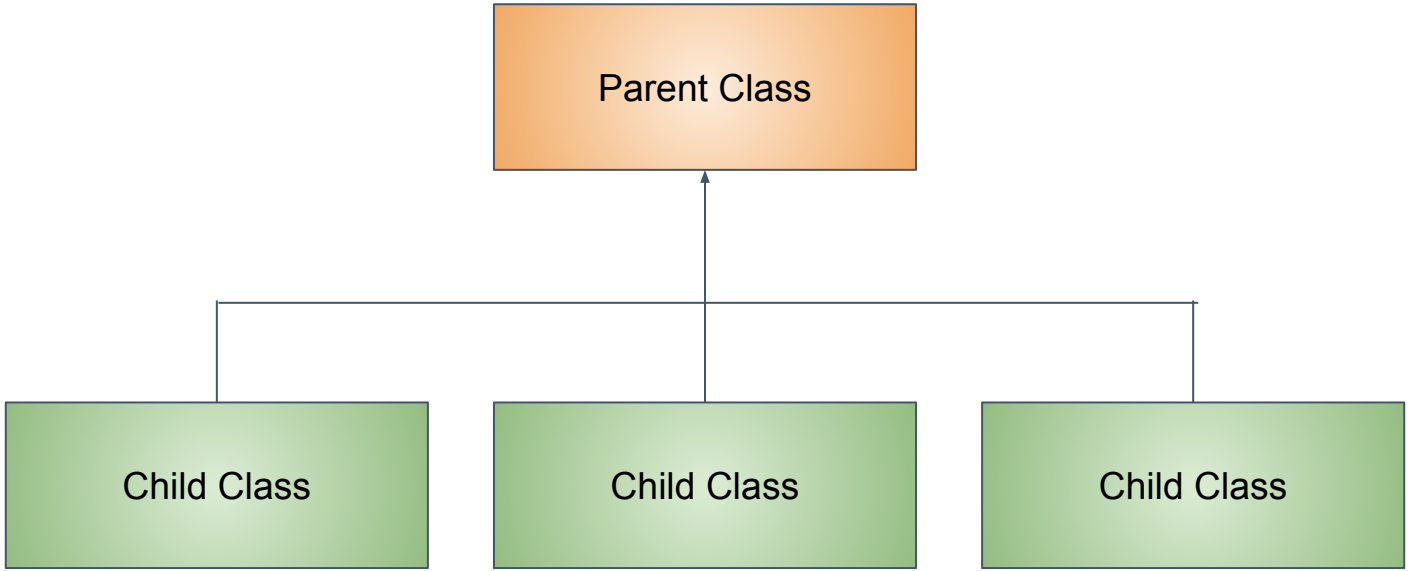
Generalization Relationship

upGrad

- While drawing the UML Class Diagram for the Inventory Management System, we realised that some of the attributes and methods were common in both the Customer and the Vendor.
- So, to reduce code duplication or to improve code reusability, we provided a separate class, BusinessPartner.
- The BusinessPartner class contains all the code (both attributes and methods) that is common to both the Vendor and the Customer.
- Therefore, BusinessPartner is a ***parent class*** and both the Vendor and the Customer are ***child classes***.

Generalization Relationship

upGrad



Inheritance

upGrad

- In Object-Oriented Programming, the generalization relationship is implemented via inheritance.
- In Java, inheritance is implemented using the ***extends*** keyword.
- So, in the Java language, instead of saying a child class inherits from the parent class, we say a child class extends the parent class.
- Let’s understand this with an example.

Inheritance

upGrad

```
class ParentClass {
    public int num = 10;
    protected String str = "XYZ";
    private boolean bool = true;
}

class ChildClass extends ParentClass {
}

public class Inheritance {
    public static void main(String[] args) {
        ChildClass childClass = new ChildClass();
        System.out.println(childClass.num);
        System.out.println(childClass.str);
    }
}
```

Output

10
XYZ

Inheritance

upGrad

- As you can see from the previous example, when there is an inheritance relationship between two classes, the child class inherits all the public and protected attributes and methods. If they are in the same package, then it inherits all the package-private attributes and methods too.
- In the previous example, we did not provide any attributes in the ChildClass, but still we were able to access the public and protected attributes from the parent class via the ChildClass object.
- The same applies to public and protected methods as well.
- When we try to access an attribute or a method, and the same is not found in the class, then JVM will search for those attributes and methods in the parent class.

Inheritance

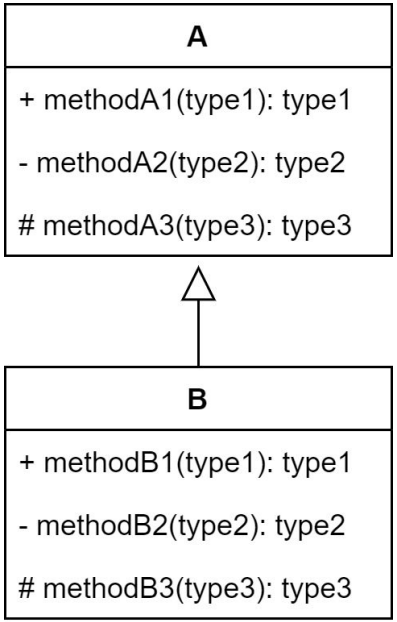
upGrad

- So, with inheritance, you can improve code reusability. You can provide common code in a class and then inherit from that class so that all the subclasses will have that piece of code.
- Inheritance can be of the following four types, of which three are supported by Java:
  - Single Inheritance
  - Multilevel Inheritance
  - Hierarchical Inheritance
  - Multiple Inheritance

Single Inheritance

upGrad

- In the following example of inheritance, the child class inherits from a single base class.

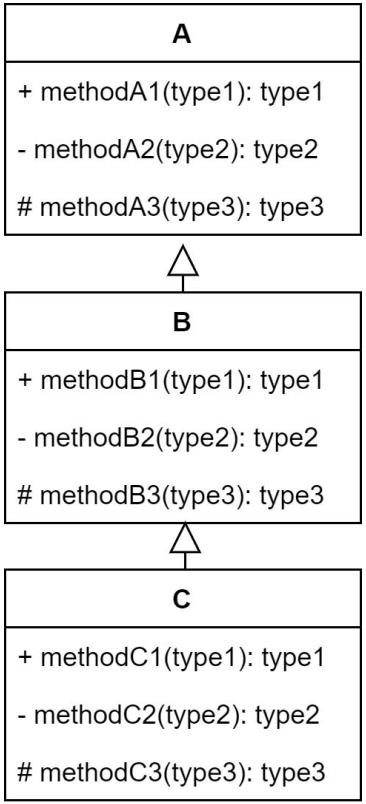


Here, B class will have the following five methods:

- methodA1(), methodA3()
- methodB1(), methodB2(), methodB3()

Multi-Level Inheritance

upGrad

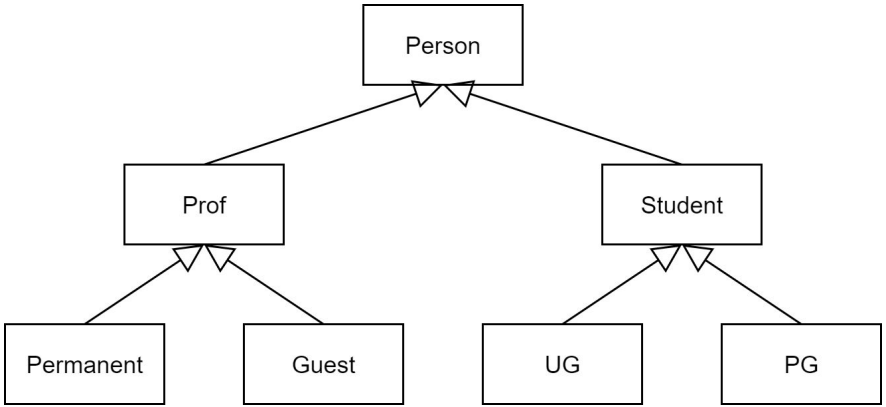


- In the following example of inheritance, the child class inherits from another child class.
- Here, B class will have the following five methods:
  - methodA1(), methodA3()
  - methodB1(), methodB2(), methodB3()
- And C class will have the following seven methods:
  - methodA1(), methodA3()
  - methodB1(), methodB3()
  - methodC1(), methodC2(), methodC3()

Hierarchical Inheritance

upGrad

- In the following example of inheritance, more than one child class will inherit from the parent class.
- The child classes will themselves be inherited by other classes.
- A class will have all the methods that it inherits from all of its parent classes, along with the ones that are declared inside that class.

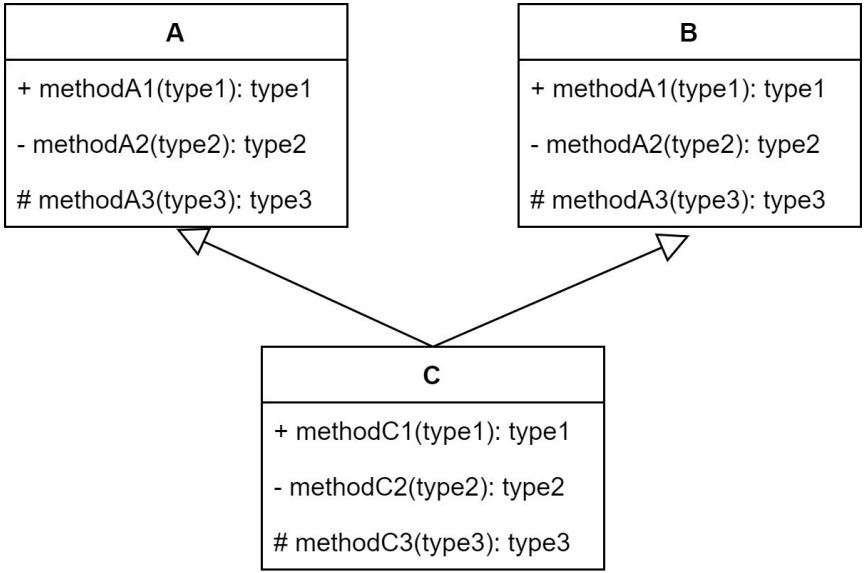


Here, the UG class will have all the public and protected methods from Person and Student, and all the methods declared inside the UG class.

Multiple Inheritance

upGrad

- In the following example of inheritance, a child class inherits from more than one parent class.
- Java does not support this relationship, because two different parent classes can provide the same method differently and this would confuse the JVM about which method to inherit.



super Keyword

upGrad

- Let’s provide the BusinessPartner class, which will contain common code for both the Customer and the Vendor, and then inherit the vendor class from the BusinessPartner class.
- Before we can do that, we have to learn about the super keyword.
- The super keyword is quite similar to the this keyword. Let’s learn about them through comparison.

super Keyword

upGrad

this	super
Used to call the constructor of the same class (should be the first statement in the constructor)	Used to call the constructor of the parent class (should be the first statement in the constructor)
Used to access the attributes and methods of the same class	Used to access the attributes and methods (only public and protected) of the parent class
Used to resolve variable name collision in methods and constructors (if the argument name is the same as the attribute name, then the attribute is accessed with the this keyword)	Used to resolve attribute and method name collision in the class (if the same attributes and methods are present in the child class and the parent class, then the parent class attributes and methods are accessed using the super keyword and the attributes and methods of the same class are accessed using the this keyword)

super Keyword

upGrad

```
class A {
    String getName() {
        return "A";
    }
}

class B extends A{
    String getName() {
        return super.getName() + "B";
    }
}

public class Inheritance {
    public static void main(String[] args) {
        B b = new B();
        System.out.println(b.getName());
    }
}
```

super Keyword

upGrad

- Now, let's provide the BusinessPartner class and make the Vendor class inherit the BusinessPartner class.
- Remember, inside the child class constructor, the first statement should be a call to the superclass constructor using the super(). This is called ***constructor chaining***.
- super() is mainly used to initialise the private attributes to the parent class. Even if we don't have any private attributes in the parent class to initialise, we still have to call the no-arg constructor.
- If we do not call any of the superclass constructors, then the JVM will call the no-arg constructor; hence, we have to make sure the no-arg constructor exists in the super class.

Code Reference

Disadvantages of Inheritance

upGrad

- You have learnt about the advantages of inheritances as it help to reduce code duplications. However, inheritance has certain disadvantages:
  - Since the child classes inherit all the public and private attributes, the JVM will allocate memory for all the attributes in heap. But you may end up not using some of the inherited attributes in a child class.
  - Inheritance makes a child class dependent on the parent class. Therefore, any change made to the base-class gets reflected in the child class.

Object Class

upGrad

- Java provides an Object class, which is a superclass of the Java classes, either direct or indirect.
- If you provide a class and it does not extend any other class, then the compiler will make it extend from the Object class.
- If you provide a class that extends a parent class, but that parent class does not extend any other class, then it will extend the Object class internally.
- The Object class provides some of the basic methods that should be present in every object, such as toString(), hashCode() and equals().
- You can verify this by providing a class without any methods and then trying to access the methods of the Object class.

Poll 2 (15 sec)

Suppose there exists a class A in a system. How to make class A the parent class of class B?

1. class B extends class A {}
2. class B extends A {}
3. class A extends class B {}
4. class A extends B {}

Poll 2 (15 sec)

Suppose there exists a class A in a system. How to make class A the parent class of class B?

1. class B extends class A {}
2. **class B extends A {}**
3. class A extends class B {}
4. class A extends B {}

Poll 3 (15 sec)

Which of the following statements regarding inheritance in Java is/are true? More than one option can be correct.

1. All classes inherit from at least one class.
2. A class inherits only attributes and methods from its direct superclass.
3. You cannot instantiate parent classes.
4. You can provide new attributes and methods in a child class.

Poll 3 (15 sec)

Which of the following statements regarding inheritance in Java is/are true? More than one option can be correct.

1. **All classes inherit from at least one class.**
2. A class inherits only attributes and methods from its direct superclass.
3. You cannot instantiate parent classes.
4. **You can provide new attributes and methods in a child class.**

Poll 4 (15 sec)

Which of the following statements regarding the super keyword in Java is/are true? More than one option can be correct.

1. super is used to call the parent class constructor.
2. With super, you can access even the private methods of the parent class.
3. super can be used to access the attributes and methods of the parent class.
4. You cannot access parent class methods with the super keyword.

Poll 4 (15 sec)

Which of the following statements regarding the super keyword in Java is/are true? More than one option can be correct.

1. **super is used to call the parent class constructor.**
2. With super, you can access even the private methods of the parent class.
3. **super can be used to access the attributes and methods of the parent class.**
4. You cannot access parent class methods with the super keyword.

Hands-On Experience

upGrad

TODO:

- Use the following command to checkout to the current state:
  git checkout 7bf3e55
- Inherit the Customer class from the BusinessPartner class.
- For now, you can remove the no-arg constructor and copy the constructor of the Customer class.
- Print the address details of the customer in the main method.

Code Reference

upGrad

Abstract Classes and Abstract Methods

Abstract Classes and Abstract Methods

upGrad

- Let’s recall Abstraction.
- While learning about abstraction, we defined it as “focusing on the essential characteristics of an object based on the context”.
- We also discussed that abstraction can be achieved in two ways:
  - **Entity Abstraction:** By providing classes for the problem domain entities
  - **Action Abstraction:** By providing the abstract methods
- While learning about encapsulation, you learnt another definition of abstraction: “ignoring unnecessary details”.

Abstract Classes and Abstract Methods

upGrad

- Let’s club the two definitions of abstraction, “focusing on essential characteristics” and “ignoring unnecessary details”, and learn how to provide action abstraction.
- Let’s take an example. Suppose we want to develop a game with two animals: dog and cat.
- In this game, the player would be a cat, which would be chased by many dogs. The cat has to survive as long as possible.
- So, the following would be the common behaviours for all the animals:
  - run()
  - jump()

Abstract Classes and Abstract Methods

upGrad

- But these common behaviours will be displayed differently; we have to play different types of animations for the cat and the dogs.
- If we have an Animal class to contain the common code, then either we do not place the run() and jump() methods in the Animal class, or if we at all place these methods in the Animal class, then we have to make them empty or provide some dummy implementation.

Abstract Classes and Abstract Methods

upGrad

- If we do not place these two methods inside the Animal class, then they we are violating the first definition, since running and jumping will be essential characteristics for all the problem domain entities.
- If we place these two methods inside the Animal class with empty body or dummy code, then we are violating the second definition, since we are focusing on unnecessary details, which is how they will be running or jumping.
- So, to apply Abstraction in this case, we have to declare these two methods inside the Animal class but implement them inside the Cat and the Dog class separately. This is called Action Abstraction and can be achieved using Abstract methods.

Abstract Classes and Abstract Methods

upGrad

- Abstract methods are those methods that are declared in a class but implemented in its subclasses.
- So, we can make the jump() and run() abstract methods in the Animal class and then implement them in the Cat and Dog classes.

Abstract Classes and Abstract Methods

upGrad

```
abstract class Animal {
    private int id;
    private String color;

    public Animal (int id, String color) {
        this.id = id;
        this.color = color;
    }

    private void detectObstacle() {
        System.out.println("Detecting Obstacles");
    }

    public abstract void run();
    public abstract void jump();
}
```

Abstract Classes and Abstract Methods

upGrad

```
class Cat extends Animal {

    public Cat(int id, String color) {
        super(id, color);
    }

    @Override
    public void run() {
        System.out.println("Cat is running.");
    }

    @Override
    public void jump() {
        System.out.println("Cat is jumping.");
    }
}
```

Abstract Classes and Abstract Methods

upGrad

- In the same way, we can also provide the implementation for the Dog class.
- While working with abstract classes and methods, you need to keep in mind the following points:
  - Abstract methods do not have any definition linked to them.

```
public abstract void abstractMethod ();
```

  - If a class contains an abstract method, then it has to be declared abstract.

```
public abstract class AbstractClass {
   public abstract void abstractMethod ();
}
```

Abstract Classes and Abstract Methods

upGrad

- You cannot instantiate abstract classes, since an abstract class contains some methods that do not have any implementation.

- An abstract class can also contain a non-abstract method.

```
public abstract class AbstractClass {
    public abstract void abstractMethod ();
    public void nonAbstractMethod () {

    }
}
```

Abstract Classes and Abstract Methods

upGrad

- An abstract class can have only non-abstract methods.

```
public abstract class AbstractClass {
    public void nonAbstractMethod () {

    }
}
```

- All the child classes have to provide implementations for all the abstract methods in the parent class.

Abstract Classes and Abstract Methods

upGrad

- If a child class does not want to provide the implementations for all of the abstract methods in the parent class, then that child class has to be declared an abstract class as well.

```
abstract class AbstractClass {
   public abstract void abstractMethod();
   public void nonAbstractMethod () {

   }
}

abstract class ChildClass extends AbstractClass {
}
```

Poll 5 (15 sec)

How are abstract methods declared?

1. `public abstract run();`
2. `public void abstract run();`
3. `public abstract void run();`
4. `public void run();`

Poll 5 (15 sec)

How are abstract methods declared?

1. `public abstract run();`
2. `public void abstract run();`
3. **`public abstract void run();`**
4. `public void run();`

Poll 6 (15 sec)

Which of the following statements regarding abstract classes and methods is true?

1. An abstract class should have at least one abstract method.
2. An abstract class cannot have non-abstract child classes.
3. An abstract class can have abstract subclasses.
4. An abstract method can exist in a non-abstract class.

Poll 6 (15 sec)

Which of the following statements regarding abstract classes and methods is true?

1. An abstract class should have at least one abstract method.
2. An abstract class cannot have non-abstract child classes.
3. **An abstract class can have abstract subclasses.**
4. An abstract method can exist in a non-abstract class.

Interface

Interface

upGrad

- In the previous section, you learnt about abstract classes.
- Abstract classes are similar to non-abstract classes (concrete classes), because:
  - Both can have attributes and methods.
  - Both can have constructors (an abstract class can have a constructor to initialise the private fields, even though abstract classes cannot be instantiated).
- Abstract classes differ from concrete classes because:
  - Abstract classes can have abstract methods (methods with body), but concrete classes cannot.
  - Concrete classes can be instantiated, whereas abstract classes cannot.

Interface

upGrad

- We can also have abstract classes that have only abstract methods. Such classes are called pure abstract classes (without attributes, constructors or non-abstract methods).

```
abstract class Animal {
    public abstract void run();
    public abstract void jump();
}
```

Interface

upGrad

- Such pure abstract classes are used to only provide the behaviours for the subclasses.
- For example, if a concrete class, Cat, extends the Animal class, then we know for sure that the Cat class has provided the implementation for the jump() and run() methods.
- We can interact with the Cat class objects by invoking the jump() and run() methods. Thus, pure abstract classes are used to provide an interface for objects or classes.
- This is why Java allows you to define such classes using an interface.

Interface

upGrad

- You can declare an interface as shown below:

```
interface Animal {
    public abstract void run();
    public abstract void jump();
}
```

- All methods inside an interface are by default public and abstract. You cannot have protected or private methods inside an interface.
- Since there is no concrete code to extend inside an interface, but you have to implement all the abstract methods of the interface, classes implement the interface, not extend them.

Interface

upGrad

```
interface Animal {
    void run();
    void jump();
}

class Cat implements Animal {
    public void run() {
        System.out.println("Cat is running");
    }

    public void jump() {
        System.out.println("Cat is jumping");
    }
}
```

Interface

upGrad

- While working with an interface, you need to keep in mind the following points:
  - Just like an abstract class, an interface cannot be instantiated.
  - All the methods of an interface are public and abstract.
  - A class implements an interface; it does not extend the interface.
  - If a class implements an interface, then it has to provide the implementation for all the methods of the interface; otherwise, that class has to be an abstract class.

```
abstract class Cat implements Animal {
}
```

Interface

upGrad

- An interface can extend another interface. In this case, the child interface will inherit methods from the parent class. If a concrete class implements the child interface, then it has to provide the implementation for the methods present in the child interface as well as the parent interface.

```
interface Interface1 {
   void method1();
}

interface Interface2 extends Interface1{
   void method2();
}
```

Interface

upGrad

- A class cannot extend more than one parent class, although it can implement more than one interface, along with extending a parent class. In this way, we can have multiple interfaces in Java, as there won't be any conflict due to different implementations for the same method, because there won't be any implementations in the interfaces.

```
class MyClass extends ParentClass implements Interface1,
Interface2, Interface3 {
    ...
}
```

- If you are extending a class as well as implementing one or more interfaces, then you first need to use the extends keyword.

Poll 7 (15 sec)

All the methods inside an interface are _____?

1. public abstract
2. private abstract
3. public non-abstract
4. private non-abstract

Poll 7 (15 sec)

All the methods inside an interface are _____?

1. **public abstract**
2. private abstract
3. public non-abstract
4. private non-abstract

Poll 8 (15 sec)

A class can _____?

1. extend just one interface.
2. extend one or more interfaces.
3. implement just one interface.
4. implement one or more interfaces.

Poll 8 (15 sec)

A class can_____?

1. extend just one interface.
2. extend one or more interfaces.
3. implement just one interface.
4. **implement one or more interfaces.**

All the codes used in today's session can be found at the link provided below:

https://github.com/ishwar-soni/fsd-ooadp-ims/tree/session5-demo

upGrad

Important Concepts and Questions

Important Questions

upGrad

1. Can an abstract class have a private constructor? How would that be useful?
2. What are the some of the benefits of having interfaces over abstract classes?
3. What are the advantages and disadvantages of inheritance in Java?
4. Why does Java not support multiple inheritance? What is a workaround for this?
5. What are the similarities and differences between an abstract class and an interface?
6. Does a class inherit the constructor of its superclass?
7. What is constructor chaining? What are the different ways of constructor chaining?

upGrad

Doubt Clearance Window

upGrad

Today, we learnt about the following:

1. Generalization Relationship and Inheritance between Classes
2. extends Keyword to Provide the Inheritance Relationship
3. Types of Inheritance – Single, Multilevel, Hierarchical and Multiple
4. super() Keyword to Call the Parent Class Constructor
5. Disadvantages of Inheritance
6. Object Class – Parent of all Java Classes
7. Abstract Class and Abstract Methods
8. Interface

TODO:

- Provide an abstract class, Box, with the following attributes:
  - ***- length: int***
  - ***- width: int***

    Here, '-' is the UML symbol for the private access modifier.
- Provide an interface, Drawable, with the following methods:
  - ***+ draw(): String***
- Provide a class, LabeledBox, which extends the Box class and implements the Drawable interface.
- This class contains one attribute:
  - ***- label: String***

TODO:

- Provide the following constructor in the LabeledBox class:
  LabeledBox(length: int, width: int, label: String)
- Implement the draw() method inside the LabeledBox class, which was inherited from the Drawable interface.
- This method should return the following output (without the double quotes):
  “Drawing Box with dimensions: (<length>, <width>) with label: <label>”
- For the output above, you have to access the length and width attributes of the Box class. You can access these two attributes using the getLength() and getWidth() methods. These two methods can be accessed directly or with the this keyword or the super keyword.

upGrad

Tasks to Complete After Today's Session

MCQs
Homework
Coding Questions
Project Checkpoint 4

upGrad

#RahoAmbitious

Thank You!