L37

Thursday, February 3, 2022 11:23 PM

Abstract Class and usages of abstract keyword, Enum Types

Abstract Class and usages of abstract keyword

Q1. In Java, a class when declared with abstract keyword becomes an abstract class. For example:
public abstract class A {
}
As per the dictionary, **abstract** means **not concrete**. In object oriented languages also, when a class is marked with abstract keyword, it indicates that the class is not concrete. **It cannot be instantiated**. Meaning, the Java compiler will given an error, saying abstract class A cannot be instantiated, if we write a statement like A a = new A();.

Abstract classes like normal (concrete) classes can have fields, constructors and methods.

Abstract classes can also have one or more abstract methods. An abstract method should have the keyword abstract in its signature and it should not have a method body. For example:
public abstract int sum(int num1, num2);
A concrete class is not allowed to have a method declared as abstract. Java compiler will throw an compilation error if an attempt is made to declare an abstract method in a concrete class.

In interfaces, the method declarations with signatures and without method bodies are by default marked abstract. However, in a abstract class we have to mark a method explicitly with abstract keyword.

Abstract classes are present so only to be inherited by other classes. Abstract classes usually contain common code that can be shared by the subclasses.

See and retype the below code to understand how to write and use abstract classes.

After executing, from the output you will realize that the classes EnglishGreeting and SpanishGreeting have inherited the implementation of getStandardMessage method from the abstract class AbstractGreeting.

If the classes EnglishGreeting and SpanishGreeting were written such that they did not extend from the abstract class and have instead implemented the Greeting interface directly, they would have been forced to individually provide the implementation for getStandardMessage method, resulting in duplication of code.

```java
package q11285;
public class AbstractDemo {
    public static void main(String[] args) {
        Greeting english = new EnglishGreeting();
        Greeting spanish = new SpanishGreeting();
        System.out.println(english.getStandardMessage("Winston"));
        System.out.println(english.getCustomMessage("Winston"));
        System.out.println(spanish.getStandardMessage("Martin"));
        System.out.println(spanish.getCustomMessage("Martin"));
    }
}
interface Greeting {
    public String getStandardMessage(String name);
    public String getCustomMessage(String name);
}
abstract class AbstractGreeting implements Greeting {
    public String getStandardMessage(String name) {
        return "Hi " + name;
    }
    public abstract String getCustomMessage(String name);
}
class EnglishGreeting extends AbstractGreeting {
    public String getCustomMessage(String name) {
        return "Hello " + name;
    }
}
class SpanishGreeting extends AbstractGreeting {
    public String getCustomMessage(String name) {
        return "Holla " + name;
    }
}
```

Q2. Write a Java program to illustrate the **abstract class** concept.

Create an abstract class CalcArea and declare the methods **triangleArea(double b, double h)**, **rectangleArea(double l, double b)**, **squareArea(double s)**, **circleArea(double r)**.

Create a class FindArea which extends the abstract class CalcArea used to find areas of triangle, rectangle, square, circle.

Write a class Area with the **main()** method which will receive **two** arguments and convert them to **double** type.

If the input is given as command line arguments to the **main()** as **"1.2","2.7"** then the program should print the output as:
Area of triangle : 1.62
Area of rectangle : 3.24
Area of square : 1.44
Area of circle : 22.890600000000006

```java
package q11286;
public class Area {
    public static void main(String args[]) {
        FindArea area = new FindArea();
        area.triangleArea(Double.parseDouble(args[0]), Double.parseDouble(args[1]));
        area.rectangleArea(Double.parseDouble(args[0]), Double.parseDouble(args[1]));
        area.squareArea(Double.parseDouble(args[0]));
        area.circleArea(Double.parseDouble(args[1]));
    }
}

// Write all the classes with definitions
class CalcArea {
    public void triangleArea(double b, double h) {
        System.out.println("Area of triangle : " + (b * h) / 2);
    }
    public void rectangleArea(double l, double b) {
        System.out.println("Area of rectangle : " + l * b);
    }
    public void squareArea(double s) {
        System.out.println("Area of square : " + s * s);
    }
    public void circleArea(double r) {
        System.out.println("Area of circle : " + 3.14 * r * r);
    }
}
class FindArea extends CalcArea {
    public void triangleArea(double b, double h) {
        super.triangleArea(b, h);
    }
    public void rectangleArea(double l, double b) {
        super.rectangleArea(l, b);
    }
    public void squareArea(double s) {
        super.squareArea(s);
    }
    public void circleArea(double r) {
        super.circleArea(r);
    }
}


```

Q3. Write a Java program to illustrate the **abstract class** concept.

Create an abstract class Shape, which contains an empty method **numberOfSides()**.

Define three classes named Trapezoid, Triangle and Hexagon extends the class Shape, such that each one of the classes contains only the method **numberOfSides()**, that contains the number of sides in the given geometrical figure.

Write a class AbstractExample with the **main()** method, declare an object to the class Shape, create instances of each class and call numberOfSides() methods of each class.

Sample Input and Output:
Number of sides in a trapezoid are 4
Number of sides in a triangle are 3
Number of sides in a hexagon are 6

```java
package q11287;
class Shape {
    public void numberOfSides() {
    }
}
class Trapezoid extends Shape {
    public void numberOfSides() {
        System.out.println("Number of sides in a trapezoid are 4");
    }
}
class Triangle extends Shape {
    public void numberOfSides() {

        System.out.println("Number of sides in a triangle are 3");
    }
}
class Hexagon extends Shape {
    public void numberOfSides() {
        System.out.println("Number of sides in a hexagon are 6");
    }
}
public class AbstractExample {
    public static void main(String[] args) {
        Shape s;
        s = new Trapezoid();
        // Call the method
        s.numberOfSides();
        s = new Triangle();
        // Call the method
        s.numberOfSides();
        s = new Hexagon();
        // Call the method
        s.numberOfSides();
    }
}
```

Enum Types

Q1. Enumeration means a list of named constant. In Java, enumeration defines a class type.

An enum is defined similarly to how a class would be defined. It uses the enum instead of the class keyword. The constant values are listed in the enum's body. It is common to write the constants in all uppercase letters to make it easier to recognize them.

This is an example of how to define an enum.

```
enum Subject {
        JAVA, CPP, C, DBMS
}
```

1. The elements inside the **Subject** enum are called **enumeration constants**
2. By default each enumeration constant is public, static and final.
3. Variables of Enumeration can be defined directly without using new keyword. Ex Subject = sub1;
4. Variables of Enumeration type can have only enumeration constants as value. We define a variable in enum as **enum_variable = enum_type.enum_constant;**. This can be done as follows. From the above example we can define variable as
   sub1 = Subject.JAVA;
5. Enum declaration can be done either outside a Class or inside a Class. But, we cannot declare Enum inside the method
6. Enum can have fields, constructors and methods
7. Enum may implement many interfaces but cannot extend any class because it internally extends java.lang.Enum, so it can't extend other classes

In the next lesson we will learn how to Implement enums with switch case.

- [x] Enumeration means list of named constant
- [ ] An enum can be defined using class keyword
- [ ] Enumeration constants are private by default.
- [ ] Enum can extend class
- [x] Enum internally extends a class called java.lang.Enum

Q2. In **Java**, along with **primitives**, **classes**, **interfaces** and **abstract classes**, there is yet another type called enum.

An **enum** is essentially a set of named values called as members or elements. Like a class, an **enum** can also have fields and methods.

For example, let us consider the four directions East, West, North and South. We can represent them as primitive int constants or as String literals. Below example uses int constants:

```
final int EAST = 1;
final int WEST = 2;
final int NORTH = 3;
final int SOUTH = 4;
int direction = SOUTH;
switch (direction) {
        case EAST:
                System.out.println("East stands for Light!");
                break;
        case WEST:
                System.out.println("West stands for Strength!");
                break;
        case NORTH:
                System.out.println("North stands for Silence!");
                break;
        case SOUTH:
                System.out.println("South stands for Love!");
                break;
        default:
                System.out.println("Unknown direction value : " + direction);
}
```

In the above example there is no restriction on a value like **5** or **6** being assigned to variable direction. There is no way for the compiler to syntactically enforce the bounds, saying we can assign only values between 1 and 4.

In such situations a more appropriate way to represent directions would be using enums. Enums are used when we are dealing with a known finite set of elements.

See and retype the below code to understand the usage of enum. Enums facilitate type checking during compile type, where the compiler will flag an error if one of the enum elements are not used where that enum type is specified.

```java
package q11288;
public class EnumDemo {
    public static void main(String[] args) {
        System.out.println(printDirectionMessage(Direction.EAST));
        System.out.println(printDirectionMessage(Direction.WEST));
        System.out.println(printDirectionMessage(Direction.NORTH));
        System.out.println(printDirectionMessage(Direction.SOUTH));
    }
    public static String printDirectionMessage(Direction direction) {
        switch (direction) {
            case EAST:
                return "East stands for Light!";
            case WEST:
                return "West stands for Strength!";
            case NORTH:
                return "North stands for Silence!";
            case SOUTH:
                return "South stands for Love!";
        }
        return "Unknown direction. This case will never occur when we use enums.";
    }
}
enum Direction {
    EAST, WEST, NORTH, SOUTH;
}

```

Q3. **values()**:

We can obtain an array of all the possible values of a Java enum type by calling its static method **values()**. All enum types get a static values() method automatically by the Java compiler. Here is an example of iterating all values of an enum:

```
enum Subject{
JAVA, C, PYTHON
}
```

Iteration can be done for the above enum class using the method value() can be done by

```
for(Subject sub : Subject.values()) {
```

System.out.println(sub);
}

```java
package q24199;
public class Test {
    public static void main(String args[]) {

        // Iterate over the elements in enum and print them
        for(Languages sub : Languages.values()){
            System.out.println(sub);
        }


    }
}
enum Languages {
    JAVA,
    PYTHON,
    C,
    CPP,
    DBMS
}
```

Q4. valueOf():

The valueOf() method can be used to obtain an instance of the enum class for a given String value. Here is an example.

enum Subject{
JAVA, C, PYTHON
}

Iteration can be done for the above enum class using the method value() can be done by

Subject sub = Subject.valueOf("JAVA");
}

```java
package q24200;

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

        // print the instances of all enumeration constants in enum class Languages

        Languages sub = Languages.valueOf("JAVA");
        System.out.println(sub);

        Languages sub1 = Languages.valueOf("PYTHON");
        System.out.println(sub1);

        Languages sub2 = Languages.valueOf("C");
        System.out.println(sub2);

        Languages sub3 = Languages.valueOf("CPP");
        System.out.println(sub3);

        Languages sub4 = Languages.valueOf("DBMS");
        System.out.println(sub4);
    }
}
enum Languages {
    JAVA,
    PYTHON,
    C,
    CPP,
    DBMS
}
```

Q5. A Java enum type can have a private constructor that can be used to initialize instance fields. Which means Java Enum can have a constructor to pass data while creating Enum constants. This feature allows you to associate related data together. Let us consider this with example.

```
public enum TrafficSignal{

   RED("wait"), GREEN("go"), ORANGE("slow down"); //this will call enum constructor with one String argumentprivate String action;

   public String getAction(){ // enum constructor - can not be public or protected
return this.action;
   }


   TrafficSignal(String action){
      this.action = action;
   }
}
```

In the above example, Enum, where we pass the action to each Enum instance e.g. GREEN, is associated with go, RED is associated with stop and ORANGE is associated with the slowdown.. We can also provide one or more constructor to Enum as it also supports constructor overloading like normal Java classes.

Just remember that constructor in enums an only be either private or package level it can't be public or protected hence access modifier public and protected are not allowed to Enum constructor, it will result in compile-time error

- [ ] Enum constructor can be public
- [x] In enum the constructor is only private
- [x] Constructor overloading can be applied in enum

Q6. The below program has an enum class with Student which has marks of different students. Define a constructor and get the marks of each student in the following way.

Total Marks:

Radha got 90 marks.
Seetha got 62 marks.
Ram got 50 marks.

```java
package q24202;

enum Student {
    Radha(90), Seetha(62), Ram(50);

    private int total;

    // write your code here
    Student(int total) {
        this.total = total;
    }
    // define a constructor Student and get total marks
    int display() {
        return total;
    }
}

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

        System.out.println("Total Marks: ");
        // write your code here

        Student s = Student.Radha;
        System.out.println("Radha got " + s.display() + " marks.");

        Student s1 = Student.Seetha;
        System.out.println("Seetha got " + s1.display() + " marks.");

        Student s2 = Student.Ram;
        System.out.println("Ram got " + s2.display() + " marks.");
    }
}

```