Topic: Enumerations.
OOP concepts involved: Enumerations, Class Instantiation and Declaration, Class Members, Polymorphism, Class Modifiers, Abstract Methods.
Programming generic concepts involved: Access Modifiers, Functions, Primitive Variables, Variable Declaration, Function Parameters.

➢ Theoric introduction

WHAT IS A JAVA ENUM?

A Java Enum (short for Enumeration) is a special Java type used to define collections of constants. More precisely, a Java enum type is a special kind of Java class. An enum can contain constants, methods, etc. *Java enums were added in Java 5*.

Because they are constants, the names of an enum type's fields are in uppercase letters.
In the Java programming language, you define an enum type by using the **enum** keyword. For example, you would specify a days-of-the-week enum type as:

```
public enum Day {
// Java Enum of the days of the week
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

Notice the **enum** keyword which is used in place of class or interface. The Java enum keyword signals to the Java compiler that this type definition is an enum.

Note: If there is a need for an enum to have a constructor, the constructor for an enum type must be package-private or private access. It automatically creates the constants that are defined at the beginning of the enum body. You cannot invoke an enum constructor yourself.

WHY ARE ENUMS USEFUL?

Enums are used when we know all possible values at **compile time**, such as choices on a menu, rounding modes, command line flags, etc. It is not necessary that the set of constants in an enum type stay fixed for all time.

Java programming language enum types are much more powerful than their counterparts in other languages. The enum declaration defines a class (called an enum type). The enum class body can include methods and other fields. The compiler automatically adds some special methods when it creates an enum. For example, they have a static **values method** that returns an array containing all of the values of the enum in the order they are declared. This method is commonly used in combination with the for-each construct to iterate over the values of an enum type. **For example, this code from the Day enum example below iterates over all the days of the week.**

```
System.out.println("These are the days of the week");
for (Day d: Day.values()) {
      System.out.println(d);
}
```

Note: All enums implicitly extend java.lang.Enum. Because a class can only extend one parent the Java language does not support multiple inheritance of state, and therefore an enum cannot extend anything else.

DECLARATION OF ENUM IN JAVA

You can define an enum type either independently outside of any Class or as part of a Class, but not inside a method.

ENUM DECLARATION INSIDE A CLASS

```
public class Test
{
    enum Color
    {
        RED, GREEN, BLUE;
    }

    // Driver method
    public static void main(String[] args)
    {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

ENUM DECLARATION OUTSIDE A CLASS

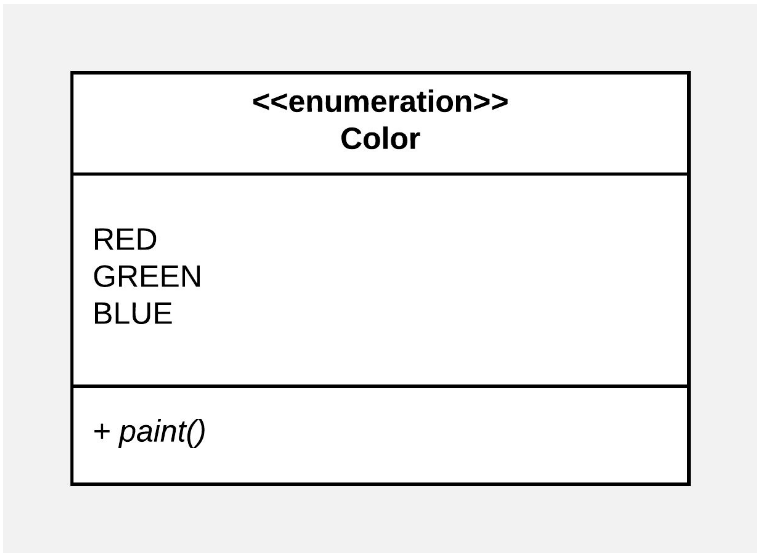
```
enum Color
{
    RED, GREEN, BLUE;
}

public class Test
{
    // Driver method
    public static void main(String[] args)
    {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

Note: Java requires that the constants be defined first, prior to any fields or methods. Also, when there are fields and methods, the list of enum constants must end with a semicolon.

➢ Statement

Based on the above Color enum example, modify it and add to each Color constant an implementation of an abstract method called **paint**, which needs to be within the same Color Enum.

➢ Class design (UML)



➢ Program Code

Color.java

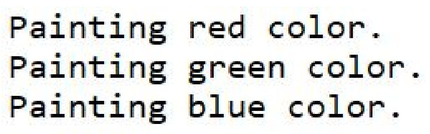
```java
public enum Color {
        // Defining constants with its own abstract method paint() implementation
        RED {
                @Override
                public void paint() {
                        System.out.println("Painting red color.");
                }
        }, GREEN {
                @Override
                public void paint() {
                        System.out.println("Painting green color.");

                }
        }, BLUE {
                @Override
                public void paint() {
                        System.out.println("Painting blue color.");
                }
        };

        // Abstract method
        public abstract void paint();
}
```

Main.java

```java
public class Main {

    public static void main(String[] args)
    {
      // Creating 3 different Color Enum instances to test each paint()
      // abstract method implementation
        Color red = Color.RED;
        Color green = Color.GREEN;
        Color blue = Color.BLUE;

        // Using each color constant method of the enum
        red.paint();
        green.paint();
        blue.paint();
    }

}
```

- ➢ **Program execution**

In the above Program Code, each enum instance (each constant) defines its own implementation of the abstract method paint declaration at the bottom of the enum class. Using an abstract method is useful when you need a different implementation of a method for each instance of a Java enum.



```
Painting red color.
Painting green color.
Painting blue color.
```

➢ Conclusions

Enumerations are very useful when we need to have a finite collection of predetermined constant literals (values) that we know we are going to use at the compile time.

Java enums can contain constants, attributes, methods, etc. We can add values to the enum constants, having the ability to predefine the value before the constructor is called. Methods and Attributes can be added to the enum class, so that we can perform a series of instructions related to its own enum literals.

And as in the previous example we can use abstract methods within the enum class so that each enum constant can define its own implementation.