Topic: Enumerations.
OOP concepts involved: Enumerations, Class Instantiation and Declaration, Class Members, Polymorphism, Class Modifiers.
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:

```java
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

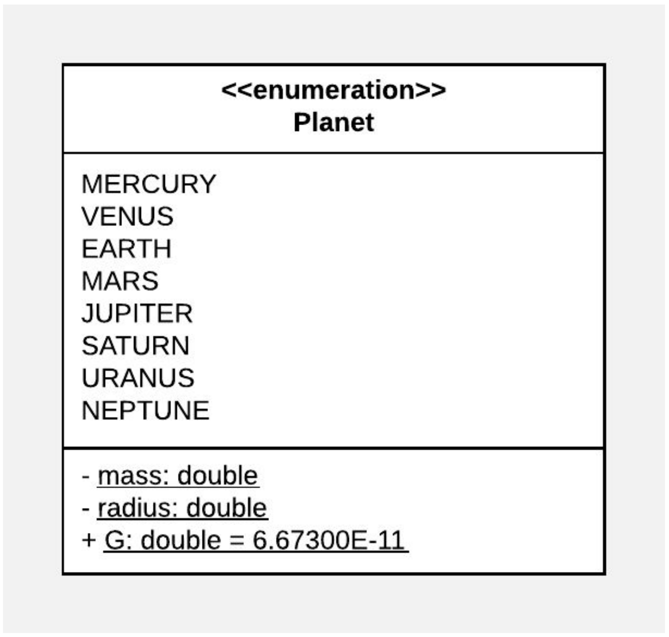
```
enum Color
{
    RED, GREEN, BLUE;
}

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

Make an implementation of an enumeration that represents the Planets of the Solar System, naming each planet as a part of the enum literals (constants). This enum needs to have two private final attributes (mass and radius) related to the planet itself, also it needs to have a universal gravitational constant. In addition of having attributes and a list of enum literals, implement a method for calculating the surfaceGravity() and another for calculating surfaceWeight() of the planet constant that has been choosed.

➢ Class design (UML)



➢ Program Code

Planet.java

```java
public enum Planet {
        // Declaring enum literals (constants) with its predefined values
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

      // private final attributes for keeping constant the selected mass
      // and radius attributes of the enum (the planet)
    private final double mass;   // in kilograms
    private final double radius; // in meters
    public static final double G = 6.67300E-11; // universal gravitational constant
(m3 kg-1 s-2)

    // Private constructor method for assigning the predetermined
    // constant values to the mass and radius attributes
    private Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }

    // Definition of Methods
    double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Main.java

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
      // Declaration of variables to save the data to be captured
      double earthWeight;
      double mass;

      // Declaring and instantiating a planet class Enum
      // Assigning constant value of Planet.Earth
      Planet earth = Planet.EARTH;

      // Instantiating a scanner object for data reading
      Scanner scanner = new Scanner(System.in);
```

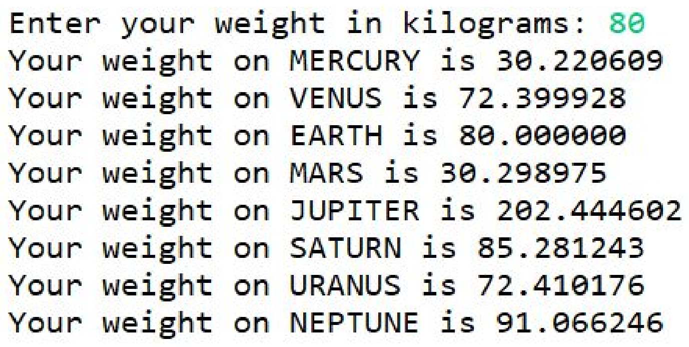
```
       System.out.print("Enter your weight in kilograms: ");
       earthWeight = scanner.nextDouble();

        mass = earthWeight/earth.surfaceGravity();

        // Using For-Each and values() static method together
        // for printing the values of the Planet Enumeration constants
        for (Planet p : Planet.values())
           System.out.printf("Your weight on %s is %f%n",
                             p, p.surfaceWeight(mass));
        // Closing scanner flow
        scanner.close();
    }
}
```

➢ **Program execution**

In the above Program Code, each enum literal (constant) has its own predefined values, corresponding to the mass (1rst value) and radius (2nd value) of the respective Planet of the listing.

Those values are assigned to the mass and radius private final attributes at the time the constructor is called implicitly (when the enum is instantiated by the creation of an Planet Object). This means that when we declare a variable of Planet type and assign the value of the enum constant (Ej. Planet.VENUS), we are calling the constructor, wich receives the 2 values associated with the enum constant literal.

In the Main method the value of earthWeight is captured and used for the calculation of the mass (after being divided by the earth.surfaceGravity() result). After, a For-Each is implemented for printing the name of the constant literal with its respective surfaceWeight() method return value depending on the value that surfaceGravity() method returns.



```
Enter your weight in kilograms: 80
Your weight on MERCURY is 30.220609
Your weight on VENUS is 72.399928
Your weight on EARTH is 80.000000
Your weight on MARS is 30.298975
Your weight on JUPITER is 202.444602
Your weight on SATURN is 85.281243
Your weight on URANUS is 72.410176
Your weight on NEPTUNE is 91.066246
```

➢ **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.