COM2104: Advanced Programming

LECTURE 12: ENUM

Objective

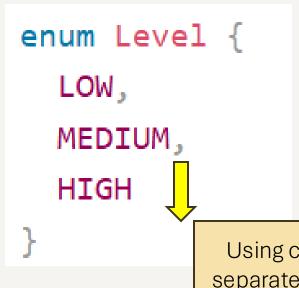
- Know what is enum
- Know how to use enum
- Know how to create fields, constructor and methods in enum types.

What is enum?

- An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).
- To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma.
- Enum is short for "enumerations", which means "specifically listed".



All constants in the enum should be capitalized



- Using enum keyword.
- Level is the name of the enum.
- Level has three constants: 1) LOW, 2) MEDIUM

AND 3) HIGH.

Using comma to separate constants

Access enum constants with the dot syntax

Level is the name of enum

Level myVar = Level.MEDIUM;





MyVar has the value of MEDIUM.

Enum inside a Class

```
public class Main {
  enum Level {
    LOW,
                                                  We put one enum
   MEDIUM,
                                                  inside one class.
    HIGH
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
                                                 The output will be:
    System.out.println(myVar);
                                                   MEDIUM
```



6

Enum in a Switch Statement

Here, enum is created outside of the class Main.

Using switch-case structure to judge the value of myVar.

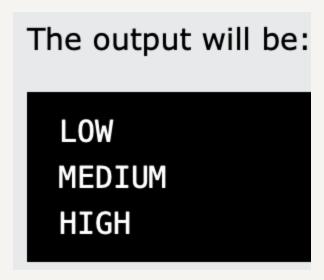


```
enum Level {
  LOW,
 MEDIUM,
  HIGH
public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
```

Loop Through an Enum

The enum type has a values() method, which returns an array of all enum constants. This method is useful when you want to loop through the constants of an enum:

```
for (Level myVar : Level values()) {
   System.out.println(myVar);
}
```



8

One extended example: Creating one enum outside of one class

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
```

We could use **public** keyword for our enum Day



```
public class EnumTest {
                                             Day can be
   Day day;
                                             treated with
    public EnumTest(Day day) {
                                              one class.
        this.day = day;
    public void tellItLikeItIs() {
        switch (day) {
            case MONDAY:
                System.out.println("Mondays are bad.");
                break:
            case FRIDAY:
                System.out.println("Fridays are better.");
                break;
            case SATURDAY: case SUNDAY:
                System.out.println("Weekends are best.");
                break;
            default:
                System.out.println("Midweek days are so-so.");
                break;
```

Continue

```
public static void main(String[] args) {
    EnumTest firstDay = new EnumTest(Day.MONDAY);
    firstDay.tellItLikeItIs();
    EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);
    thirdDay.tellItLikeItIs();
    EnumTest fifthDay = new EnumTest(Day.FRIDAY);
    fifthDay.tellItLikeItIs();
    EnumTest sixthDay = new EnumTest(Day.SATURDAY);
    sixthDay.tellItLikeItIs();
    EnumTest seventhDay = new EnumTest(Day.SUNDAY);
    seventhDay.tellItLikeItIs();
```

The constants of Day can be used for creating objects of class EnumTest



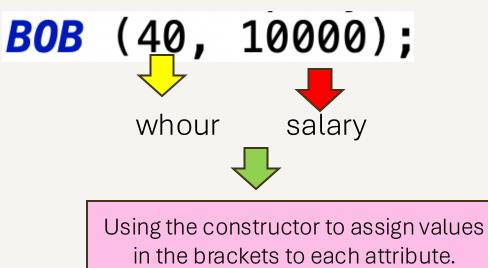
The output is:

Mondays are bad.
Midweek days are so-so.
Fridays are better.
Weekends are best.
Weekends are best.

Fields and Constructors in Enums

- When creating constants of enums, we could use () to include values of attributes.
- For example, you want to create one enum name Employee which contains one person named BOB. If you want to add weekly working hours and monthly salary to BOB, you can do it as follows:

```
public enum Employee {
    BOB (40, 10000);
    final private double whour;
    final private int salary;
    Employee(double whour,int salary){
        this.whour = whour;
        this.salary = salary;
    }
}
```



Methods in Enum

 Methods in enum are similar to those in a class. You could add any methods needed to one enum, including main method. Such as:

```
public enum Employee {
   BOB (40, 10000);
    final private double whour;
    final private int salary;
    Employee(double whour,int salary){
        this.whour = whour;
        this.salary = salary;
    private double getWhour() {
        return whour;
    private int getSalary() {
        return salary;
```

One more example about powerful Enums

We can define constructors, methods, and fields inside enum types, which makes them very powerful.

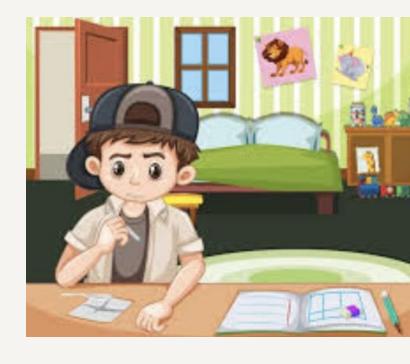
- You could treat
 Enumexample as one class. It used
 enum keyword.
- Two final attributes: day and wh
- A Constructor of Enumexample
- Two private methods could return value of each attribute, respectively.
- main method could allow us the do sth based on the constants.

```
public enum Enumexample {
   MONDAY (1, 8),
    TUESDAY(2, 6),
                                   The final keyword makes the
   WESDESDAY(3, 6),
    THURSDAY(4, 6);
                                 attribute immutable after it has
    final private int day;
                                     been assigned a value.
    final private int wh;
    Enumexample(int day, int wh) {
        this.day = day;
        this.wh = wh;
    private int getDay() {return day;}
    private int getWH() {return wh;}
    public static void main(String[] args) {
        for(Enumexample p: Enumexample.values()) {
            System.out.printf("The working hours for day of %s is %d\n'
                    p, p.getWH());
```

13

One more example about powerful Enums

```
public enum Planet {
   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 double mass; // in kilograms
   private final double radius; // in meters
   Planet(double mass, double radius) {
       this.mass = mass;
       this.radius = radius:
   private double mass() { return mass; }
   private double radius() { return radius; }
   // universal gravitational constant (m3 kg-1 s-2)
   public static final double G = 6.67300E-11;
   double surfaceGravity() {
       return G * mass / (radius * radius);
   double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
   public static void main(String[] args) {
       for(Planet p: Planet.values()) {
            System.out.printf("Gravity on %s is %.3f%n",
                   p, p.surfaceGravity());
```



Output:

Gravity on MERCURY is 3.703
Gravity on VENUS is 8.871
Gravity on EARTH is 9.803
Gravity on MARS is 3.713
Gravity on JUPITER is 24.806
Gravity on SATURN is 10.450
Gravity on URANUS is 8.873
Gravity on NEPTUNE is 11.159

Some notes

Difference between Enums and Classes

- o An enum can, just like a class, have attributes and methods. The only difference is that enum constants are public, static and final (unchangeable cannot be overridden).
- o An enum cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).
- Why And When To Use Enums?
 - o Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

