Task: Enumerators

www.hyperiondev.com

Introduction

Collecting information

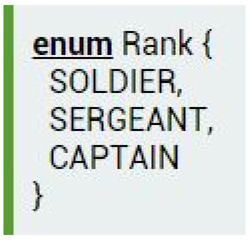
In this task you are going to learn about an enum type which is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Enumeration (Enum) was not originally available in Java though it was available in another language like C and C++, but eventually Java realized and introduced Enum on JDK 5 (Tiger) by keyword Enum. Let's get started!

-The Hyperion Team

A note from the Hyperion Team...

Defining Enumerators

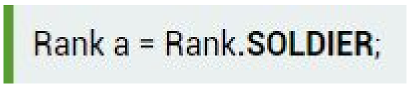
An Enum is a special type used to define collections of constants. Here is a simple Enum example:



```
enum Rank {
  SOLDIER,
  SERGEANT,
  CAPTAIN
}
```

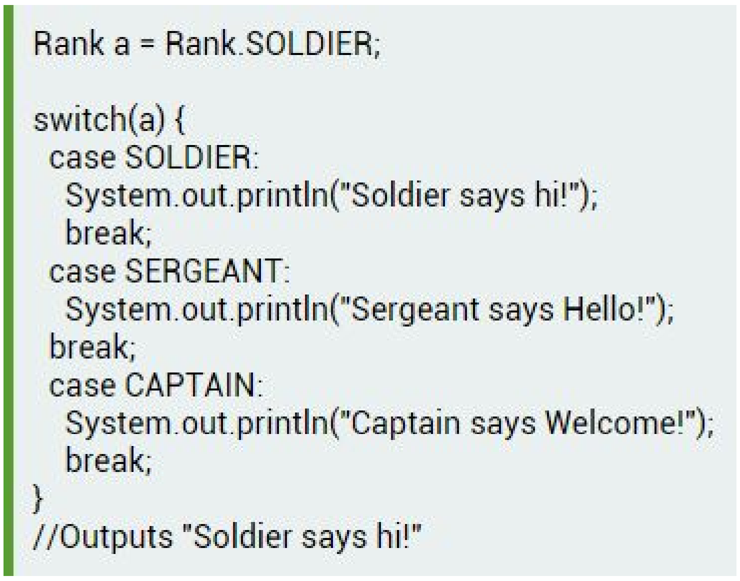
***Note**: the values are comma-separated.*

You can refer to the constants in the enum above with the dot syntax:



```
Rank a = Rank.SOLDIER;
```

Basically, Enums define variables that represent members of a fixed set. After declaring an Enum, we can check for the corresponding values with, for example, a switch statement:



```
Rank a = Rank.SOLDIER;

switch(a) {
  case SOLDIER:
    System.out.println("Soldier says hi!");
    break;
  case SERGEANT:
    System.out.println("Sergeant says Hello!");
  break;
  case CAPTAIN:
    System.out.println("Captain says Welcome!");
    break;
}
//Outputs "Soldier says hi!"
```

You should always use Enums when a variable (especially a method parameter) can only take one out of a small set of possible values. If you use Enums instead of integers (or String codes), you increase compile-time checking and avoid errors from passing in invalid constants, and you document which values are legal to use.

Note: *Some sample Enum uses include month names, days of the week, deck of cards, etc.*

The benefits of Enums

1) Enum is type-safe so you cannot assign anything else other than predefined Enum constants to an Enum variable. It is a compiler error to assign something else, unlike the public static final variables used in Enum int pattern and Enum String pattern.

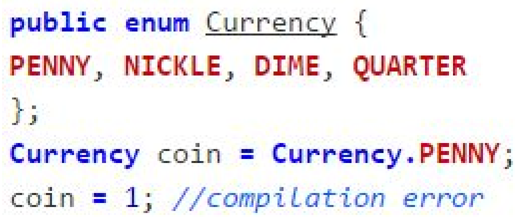
2) Enum has its own namespace.

3) The best feature of Enum is you can use Enum in Java inside Switch statement like int or char primitive data type. We will also see an example of using java enum in switch statement in this task.

4) Adding new constants on Enum in Java is easy and you can add new constants without breaking the existing code.

Enums Functionality

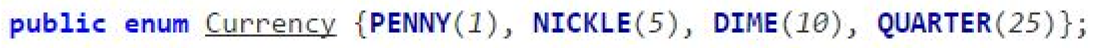
- Enums in Java are type-safe and has their own namespace. It means your enum will have a type for example "Currency" in the below example and you can not assign any value other than specified in Enum Constants.



```
public enum Currency {
PENNY, NICKLE, DIME, QUARTER
};
Currency coin = Currency.PENNY;
coin = 1; //compilation error
```

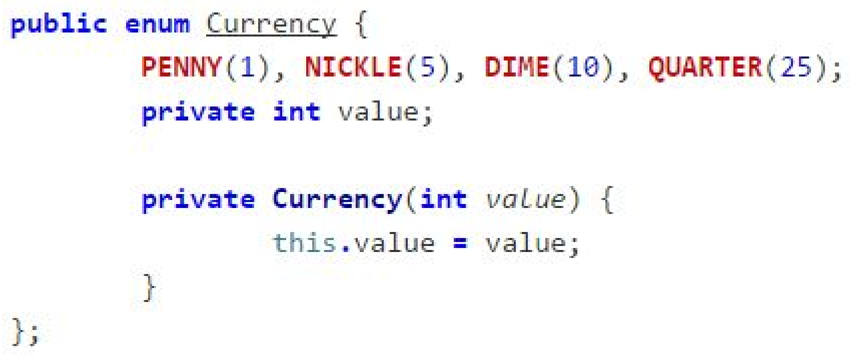
- Enums in Java are reference types like class or interface and you can define constructor, methods and variables inside java Enum which makes it more powerful than Enum in C and C++ as shown in next example of Java Enum type.

- You can specify values of enum constants at the creation time as shown in below example:



```
public enum Currency {PENNY(1), NICKLE(5), DIME(10), QUARTER(25)};
```

But for this to work you need to define a member variable and a constructor because PENNY (1) is actually calling a constructor which accepts an int value. See the example below:



```
public enum Currency {
        PENNY(1), NICKLE(5), DIME(10), QUARTER(25);
        private int value;

        private Currency(int value) {
                this.value = value;
        }
};
```

The constructor of enum in java must be private any other access modifier will result in compilation error. Now to get the value associated with each coin you can define a public getValue() method inside Java enum like any normal Java class. Also, the semicolon in the first line is optional.

- Enum constants are implicitly static and final and cannot be changed once created. For example, below code of java enum will result in compilation error:

```
Currency.PENNY = Currency.DIME;
```

  The final field EnumExamples.Currency.PENNY cannot be reassigned.

- Enum in java can be used as an argument on switch statement and with "case:" like int or char primitive type. This feature of java enum makes them very useful for switch operations. Let’s see an example of how to use java enum inside switch statement:

```
Currency usCoin = Currency.DIME;

    switch (usCoin) {
            case PENNY:
                    System.out.println("Penny coin");
                    break;
            case NICKLE:
                    System.out.println("Nickle coin");
                    break;
            case DIME:
                    System.out.println("Dime coin");
                    break;
            case QUARTER:
                    System.out.println("Quarter coin");
 }
```

- Since constants defined inside Enum in Java are final you can safely compare them using "==", the equality operator as shown in following example of Java Enum:

```
Currency usCoin = Currency.DIME;
if(usCoin == Currency.DIME){
  System.out.println("enum in java can be compared using ==");
}
```

  By the way comparing objects using == operator is not recommended, always use equals() method or compareTo() method to compare Objects.

- Java compiler automatically generates static values() method for every enum in java. Values() method returns array of Enum constants in the same order they have listed in Enum and you can use values() to iterate over values of Enum in Java as shown in below example:

```
for(Currency coin: Currency.values()){
   System.out.println("coin: " + coin);
}
```

Which produces the the output:

```
coin: PENNY
coin: NICKLE
coin: DIME
coin: QUARTER
```

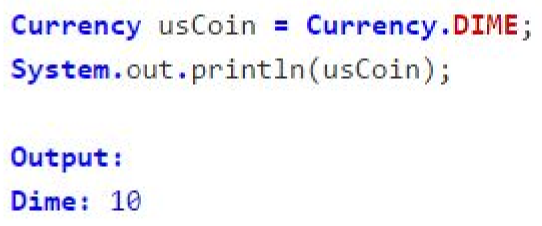
Notice the order is exactly the same as defined order in the Enum.

- In Java, Enum can also override methods. Let's see an example of overriding toString() method inside Enum in Java to provide a meaningful description for enums constants:

```
public enum Currency {
  ........

  @Override
  public String toString() {
       switch (this) {
         case PENNY:
              System.out.println("Penny: " + value);
              break;
         case NICKLE:
              System.out.println("Nickle: " + value);
              break;
         case DIME:
              System.out.println("Dime: " + value);
              break;
         case QUARTER:
              System.out.println("Quarter: " + value);
        }
  return super.toString();
 }
};
```

And here is how it looks like when displayed:



```
Currency usCoin = Currency.DIME;
System.out.println(usCoin);

Output:
Dime: 10
```

- Two new collection classes EnumMap and EnumSet are added into collection package to support Java Enum. These classes are a high-performance implementation of Map and Set interface in Java and we should use this whenever there is any opportunity.

  EnumSet doesn't have any public constructor instead it provides factory methods to create instance e.g. EnumSet.of() methods. This design allows EnumSet to internally choose between two different implementations depending upon the size of Enum constants.

  If Enum has less than 64 constants then EnumSet uses RegularEnumSet class which internally uses a long variable to store those 64 Enum constants, and if Enum has more keys than 64 then it uses JumboEnumSet. (*For interest sake*).

- You cannot create an instance of enums by using new operator in Java because the constructor of Enum in Java can only be private, and Enums constants can only be created inside Enums itself.

- An instance of Enum in Java is created when any Enum constants are first called or referenced in code.

- Enum in Java can implement the interface and override any method like normal class It’s also worth noting that Enum in java implicitly implements both Serializable and Comparable interface. Let's look at an example of how to implement interface using Java Enum:

```
public enum Currency implements Runnable{
  PENNY(1), NICKLE(5), DIME(10), QUARTER(25);
  private int value;
  ............

  @Override
  public void run() {
  System.out.println("Enum in Java implement interfaces");

   }
}
```

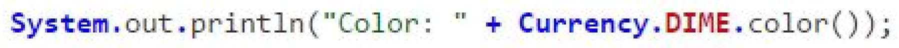
- You can define abstract methods inside Enum in Java and can also provide a different implementation for different instances of enum in java. Let's see an example of using abstract method inside enum in java:

```
public enum Currency {
       PENNY(1) {
           @Override
           public String color() {
               return "copper";
           }
       },
       NICKLE(5) {
           @Override
           public String color() {
               return "bronze";
           }
       },
       DIME(10) {
           @Override
           public String color() {
               return "silver";
           }
       },
       QUARTER(25) {
           @Override
           public String color() {
               return "silver";
           }
       };
       private int value;

       public abstract String color();

       private Currency(int value) {
           this.value = value;
       }
```

In this example since every coin will have the different color we made the color() method abstract and let each instance of Enum to define their own color. You can get color of any coin by just calling the color() method as shown in below example of Java Enum:



```java
System.out.println("Color: " + Currency.DIME.color());
```

Instructions

First read example.java, open it using jGRASP (Right click the file and select 'Open with jGRASP").

- In this folder there is a file called example.java
- Open this folder using the JGRASP program. You can either do this by right clicking on the example.java file, going to the 'Open with' menu, and selecting JGrasp.exe. Alternatively, you can run the JGRASP program on your computer, go to the top left corner of the program, select File->Open and navigate to example.java on your hard drive and double click on it to open it.
- Once example.java is open in JGRASP please read all of its content very carefully. This will help you understand the basic structure of a Java program.
- There is a compulsory task at the end of the example.java document. The instructions of what to do to complete the task can be found here. You must complete this exercise to continue onto the next task of this course.

Compulsory Task

Follow these steps:

- Create a new java file called Planet.java

- Create a public Enum called 'Planet'.

- Define the following values within the Enum:
    - 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)

- Define two private final Double variables; 'mass' and 'radius'.

- Initialise the Planet properties as such:

    -
      ```
      Planet(double mass, double radius) {
          this.mass = mass;
          this.radius = radius;
      }
      ```

- Create a public static final Double variable called 'G' and set it to 6.67300E-11.

- Define a Double method called 'surfaceGravity' which returns the result of:
    - G * mass / (radius * radius)

- Define a Double method called 'surfaceWeight' which takes a Double variable called 'otherMass', and returns:
    - otherMass * surfaceGravity()

- Create a main method which does the following:
    - Ask the user for their weight
    - Convert this String input to a Double called earthWeight
    - Create a Double variable called earthMass which is equal to:
        - earthWeight/EARTH.surfaceGravity()
    - Create a loop to iterate through each planet in the Planet Enum to display the user's weight on each planet.

- Compile, save and run your file.

Things to look out for

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **jGRASP or Java** may not be installed correctly.
2. If you are not using Windows, please ask your tutor for alternative instructions.

Still need help?

Just write your queries in your comments.txt file and your tutor will respond.

Task Statistics

Last update to task: 05/02/2016.
Authors: Jared Ping.
Main trainer: Umar Randeree.
Task Feedback link: Hyperion Development Feedback.