Introduction to Programming I

Lab 9

Alexey Shikulin, Munir Makhmutov, Sami Sellami and Furqan Haider

Agenda

- Enums, Assertions, Annotations
- Warm-up exercises
- Solving Problems
- Q&A

Learning outcome:

- Enhance your knowledge on:
    - Enums, Assertions, Annotations

Enums in C++

SPONGEBOB DAILY
Check it out.

Enums in Java

How to hardcode data

Enums in Java

An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).

```
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;
    }
  }
}
```

Enums in Java. Fields, Methods

```java
public enum Level {
   HIGH  (3),  //calls constructor with value 3
   MEDIUM(2),  //calls constructor with value 2
   LOW   (1)   //calls constructor with value 1
   ; // semicolon needed when fields / methods follow

   private final int levelCode;

   Level(int levelCode) {
       this.levelCode = levelCode;
   }

   public int getLevelCode() {
       return this.levelCode;
   }
}
```

You can add fields to a Java enum. Thus, each constant enum value gets these fields.

The field values must be supplied to the constructor of the enum when defining the constants.

You can add methods to a Java enum too

Enums in Java. Fields, Methods

It is possible for a Java enum class to have abstract methods too. If an enum class has an abstract method, then each instance of the enum class must implement it.

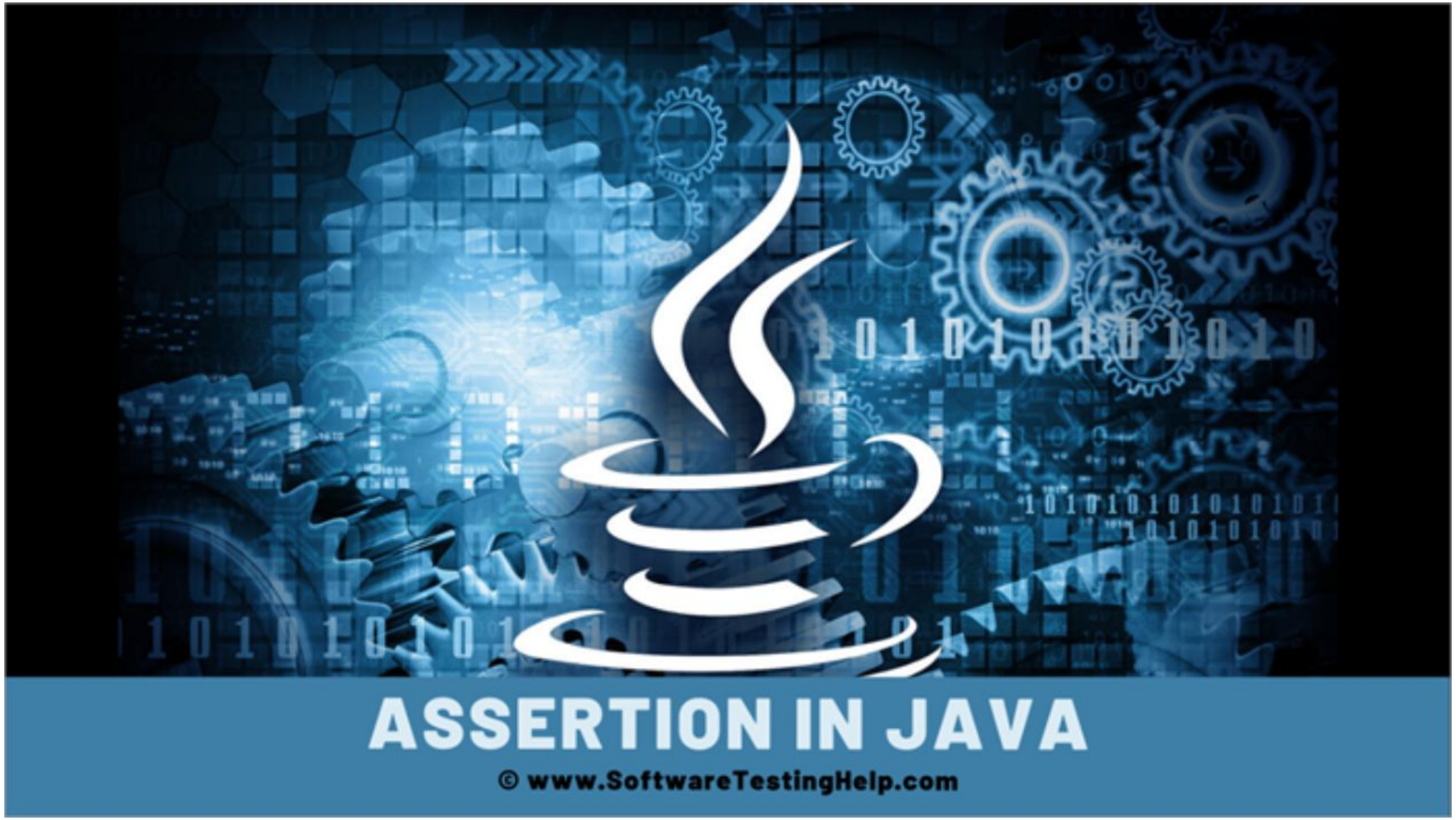
Notice the abstract method declaration at the bottom of the enum class.

Notice also how each enum instance (each constant) defines its own implementation of this abstract method.

Using an abstract method is useful when you need a different implementation of a method for each instance of a Java enum.

```java
public enum Level {
   HIGH {
       @Override
       public String asLowerCase() {
           return HIGH.toString().toLowerCase();
       }
   },
   MEDIUM {
       @Override
       public String asLowerCase() {
           return MEDIUM.toString().toLowerCase();
       }
   },
   LOW {
       @Override
       public String asLowerCase() {
           return LOW.toString().toLowerCase();
       }
   };

   public abstract String asLowerCase();
}
```


ASSERTION IN JAVA
© www.SoftwareTestingHelp.com

Assertions

The Java **assert** keyword allows developers to quickly verify certain assumptions or state of a program.

```
class Test {
   public static void main( String args[] ){
       int value = 15;
       assert value >= 20 : " Underweight";
       System.out.println("value is "+value);
   }
}
```

Assertions. Enabling / Disabling

By default, assertions are disabled. We need to run the code as given. The syntax **for enabling** assertion statement in Java source code is:

```
java –ea Test //shorter way          or
java –enableassertions Test //default way
```

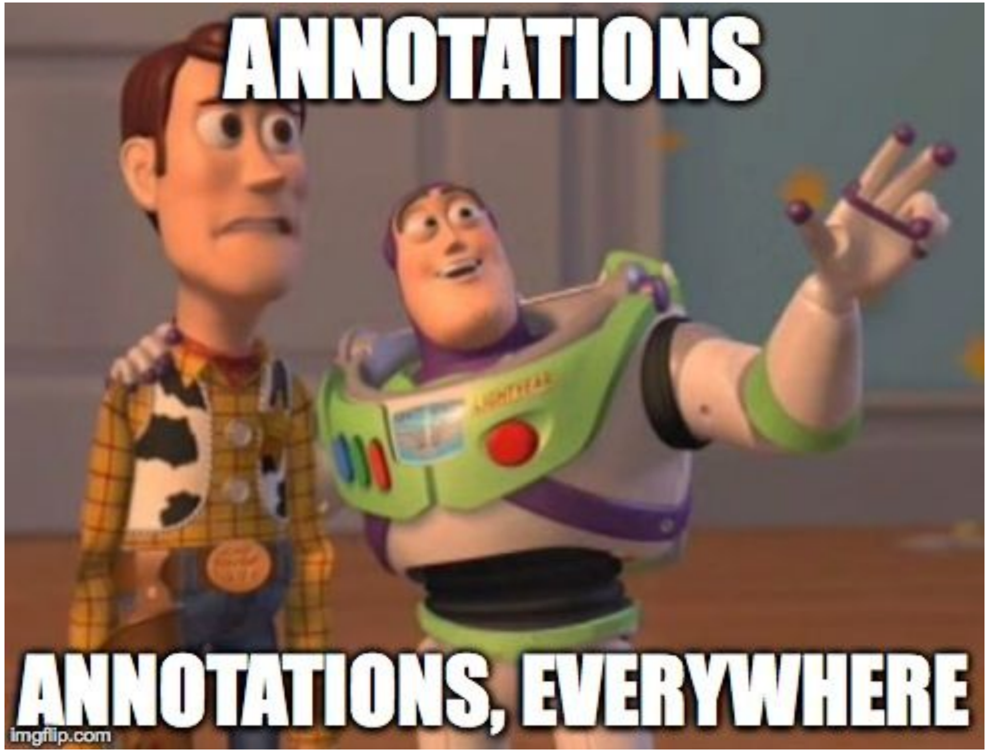
The syntax for **disabling** assertions in java are:

```
java –da Test //shorter way          or
java -disableassertions Test //default way
```

Enabling assertions in IntelliJ :

`https://se-education.org/guides/tutorials/intellijUsefulSettings.html`

When you Work in a Large Java Project…



Annotations

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

- **Information for the compiler** — Annotations can be used by the compiler to detect errors or suppress warnings.
- **Compile-time and deployment-time processing** — Software tools can process annotation information to generate code, XML files, and so forth.
- **Runtime processing** — Some annotations are available to be examined at runtime.

```
            @Entity
```

The @ character signals to the compiler that this is an annotation. The name following the @ character is the name of the annotation. In the example above the annotation name is Entity.

Annotations

Built-in annotations:

- @Override: Is a Java annotation used above methods that override methods in a superclass. If the method does not match a method in the superclass, the compiler will give you an error. This annotation is not necessary in order to override a method in a superclass. It is a good idea to use it still.
- @Deprecated: It is used to mark a class, method or field as deprecated, meaning it should no longer be used. If your code uses deprecated classes, methods or fields, the compiler will give you a warning.
- @SuppressWarnings: This annotation makes the compiler suppress warnings for a given method. For instance, if a method calls a deprecated method, or makes an insecure type cast, the compiler may generate a warning. You can suppress these warnings by annotating the method containing the code with the @SuppressWarnings annotation.

Annotations. Custom Annotations

To add metadata with an annotation, you must first define the *annotation type*.

Annotation types are a form of *interface*,

The body of the annotation definition contains *annotation type element* declarations, which look a lot like methods.

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
   // Note use of array
   String[] reviewers();
}
```

```
@ClassPreamble (
        author = "John Doe",
        date = "3/17/2002",
        currentRevision = 6,
        lastModified = "4/12/2004",
        lastModifiedBy = "Jane Doe",
        // Note array notation
        reviewers = {"Alice", "Bob", "Cindy"}
)
public class Generation3List {

// class code goes here

}
```

After the annotation type is defined, you can use annotations of that type, with the values filled in.

Warm Up Exercises

Warm Up Exercises

1. Difference between Enums and Classes?
2. Why And When To Use Enums?
3. An Enum type can be a subclass of java.lang.String. Is this statement true?

Warm Up Exercises

1. Difference between Enums and Classes?

Answer: 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).
An enum cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

2. Why And When To Use Enums?

Answer: Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

3. An Enum type can be a subclass of java.lang.String. Is this statement true?

Answer: False. 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.

Exercise 1: Enums

The gravity you would experience on each of the planets in the solar system if you were standing on the surface or, in the case of the ice giants, floating in the atmosphere, is different. Because of that, your weight is different in each planet.

- Write a simple program that takes your weight on earth (in Kg) and calculates and prints your weight on all of the planets.

  A planet is defined by mass and radius

  Universal Gravitational Constant G = 6.67408 × 10^{-11} m^3 kg^{-1} s^{-2}

  Surface Gravity = G * mass / radius²

  Person's mass = Weight on Earth / Earth's Surface Gravity

  Surface Weight = Person's mass * Surface Gravity

Planet	**Mercury**	**Venus**	**Earth**	**Mars**	**Jupiter**	**Saturn**	**Uranus**	**Neptune**
Mass (Kg)	$3.303*10^{23}$	$4.869*10^{24}$	$5.976*10^{24}$	$6.421*10^{23}$	$1.9*10^{27}$	$5.688*10^{26}$	$8.686*10^{25}$	$1.024*10^{26}$
Radius (Km)	$2.4397*10^{6}$	$6.0518*10^{6}$	$6.37814*10^{6}$	$3.3972*10^{6}$	$7.1492*10^{7}$	$6.0268*10^{7}$	$2.5559*10^{7}$	$2.4746*10^{7}$

Exercise 1: Solution

```java
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) {
       if (args.length != 1) {
           System.err.println("Usage: java Planet <earth_weight>");
           System.exit(-1);
       }
       double earthWeight = Double.parseDouble(args[0]);
       double mass = earthWeight/EARTH.surfaceGravity();
       for (Planet p : Planet.values())
         System.out.printf("Your weight on %s is %f%n", p, p.surfaceWeight(mass));
   }
}
```

Exercise 2: Enums

- Write a simple *Vending Machine* program, which allows money insertion and buying a single drink and returning the money (unlimited in machine). Before money insertion, the *Vending Machine* should show the menu with prices
- Create enum *Drinks* with beverage drinks (Coke Cola, Sprite, Fanta) with parameters *name* and *price*. Create enum *Money* with applicable banknotes with parameter *denomination*. Assume that coins cannot be used
- Handle exceptions if needed (not enough money, negative values, etc.), if familiar with exceptions. Otherwise, use error messages
- Assume that if the *Vending Machine* cannot return the money, because of missing such a banknote in *Money* enum, it will return banknote with the closest lesser nomination, *e.g.* instead of 6$ the customer will be returned 5$. Provide adequate interaction with the customer

Exercise 3: Enums

- Extend your “Equipment rental” system (Lab 9) adding allowed sizes of the equipment and make sure the user can retrieve information about sizes available.
- Add enum about status of each equipment:
    - reserved / available / damaged / taken

Exercise 4: Annotations

- Try to add @Override and @Deprecated methods to exercises previously implemented during the course

Exercise 5: Assertion

- Copy the code given as an example for assertions, enable assertions and run the code. Try to answer what could be used instead of assertions in this case

References

- https://www.w3schools.com/java/java_enums.asp
- https://www.baeldung.com/java-assert
- https://www.geeksforgeeks.org/assertions-in-java/