ENUMERATION

OUTLINE

- Enums
- Working with Enums
- Conclusion

ENUMS ?

- Java 5 introduced the data type *enum*.
- Enums are type-safe constructs, meaning that represents a **FIXED** list of values and can't be reassigned.

KEY POINT

- Enums are singletons.
  - You cannot create an instance of an enum by using the new operator.
    - A constructor in an *enum* class can only be specified as private.
      Enums are not allowed to have a public constructor, or the compiler will complain with following message: "Illegal modifier for the enum constructor; only private is permitted".
- An instance of an enum is created when the enum is first referenced.
- Enums are thread-safe by default (meaning that you don't need to do double checks when creating them).

RULE

- Enums are implicitly declared public, static, and final, which means you cannot extend because all enums extend from java.lang.Enum

```
// A enum can't extend a class
public enum Volume extends AClass { ... }
```

- **CAN** do is implement interfaces:

```
public enum Volume implements AnInterface { ... }
```

RULE

- When working with enums is overriding methods and implementing abstract methods.

```
public enum Volume {
    HIGH(100) {
       public void printValue() {
           System.out.println("** Highest value**");
       }
       public void printDescription() {
           System.out.println("High Volume");
       }
    }, MEDIUM(50) {
        public void printDescription() {
           System.out.println("Medium Volume");
        }
    }, LOW(20) {
        public void printDescription() {
            System.out.println("Low Volume");
        }
    };
    private int value;

    private Volume(int value) {
        this.value = value;
    }

    public void printValue() {
        System.out.println(value);
    }
    public abstract void printDescription();
}
```

RULE

- If you declare an enum within a class, then it is by default static.
- You can compare two enumerations for equality using == operator.
- If enumeration constants are from two different enumerations, the equals() method does not return true.
- Enumeration constants cannot be cloned. An attempt to do so will result in a CloneNotSupportedException.

RULE

- Enums are required to have a semicolon after the list of value.

```
4    public enum AnimalClasses {
5        MAMMAL(true), FISH(Boolean.FALSE), BIRD(false),
6        REPTILE(false), AMPHIBIAN(false), INVERTEBRATE(false)
7
8        boolean hasHair;
9
10       public AnimalClasses(boolean hasHair) {
11           this.hasHair = hasHair;
12       }
13
14       public boolean hasHair() {
15           return hasHair;
16       }
17
18       public void giveWig() {
19           hasHair = true;
20       }
21   }
```

```
4    public enum AnimalClasses {
5        MAMMAL(true), FISH(Boolean.FALSE), BIRD(false),
6        REPTILE(false), AMPHIBIAN(false), INVERTEBRATE(false)
7
8        boolean hasHair;
9
10       public AnimalClasses(boolean hasHair) {
11           this.hasHair = hasHair;
12       }
13
14       public boolean hasHair() {
15           return hasHair;
16       }
17
18       public void giveWig() {
19           hasHair = true;
20       }
21   }
```

HOW TO GET A REFERENCE OF AN ENUM.

- Reference directly

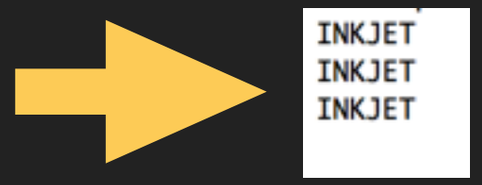
```
enum PrinterType {
    DOTMATRIX, INKJET, LASER
}

PrinterType type = PrinterType.DOTMATRIX;
PrinterType type2 = PrinterType.valueOf("DOTMATRIX");
PrinterType type3 = PrinterType.valueOf("DotmatriX"); // Run-time exception
```

- You can apply the valueOf() and name() methods to the enum element to return the name of the enum element.
- Get an enum from a string (be careful* case sensitive)

WORKING WITH ENUMS

- name() method its equivalent to invoke the `name()` method that all enums have

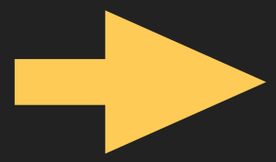
```
System.out.println(PrinterType.INKJET);
// equivalent to invoke the name()
System.out.println(PrinterType.INKJET.name());
// in other words
System.out.println(PrinterType.INKJET.toString());
```



```
INKJET
INKJET
INKJET
```

- ordinal() method of an enum returns its corresponding int value. Like arrays, enums are zero based. Remember that the index of an enum may change when you recompile the code and should not be used for comparison.

```
public enum Volume {
    HIGH, MEDIUM, LOW
}
```



```
System.out.println(Volume.HIGH.ordinal());
System.out.println(Volume.LOW.ordinal());
```

The output will be:

```
0
2
```

- The static values() method in the Enum class returns an array of the enumeration constants when called on an enumeration type.

```
for(Volume v: Volume.values()) {
   System.out.print(v.name() + ", ");
}
```

The output:

```
HIGH, MEDIUM, LOW,
```

SWITCH-CASE

- A case statement on an enum data type must be the unqualified name of an enumeration constant. For example, case VANILLA would be valid. You cannot use the ordinal equivalents. Therefore, the code does not compile.

```
Volume level = Volume.HIGH;
...
// Or Volume.HIGH.equals(level)
if(Volume.HIGH == level) {
    ...
}
switch(level) {
    // Notice that the only the name of the enum is used,
    // in fact, Volume.HIGH for example, won't compile
    case HIGH: ...
    case MEDIUM: ...
    case LOW: ...
}
```

CONCLUSION

- E*num* is a type that represents a **FIXED** list of values, providing type-safe.
- Enums can define constructors, but they must be private.
  - You cannot create an instance of an enum by using the new operator.
- Enums are implicitly `public`, `static` and `final`.
- Enums can be compared against other enums using the == operator and the `equals()` method.
- Enums can be used in `switch` statements.
- Enums can implement interfaces, but they cannot extend from a class since they implicitly extend from `java.lang.Enum`.
- Enums are the easiest way to implement singletons.