Name: Ibtesam Lamine

24/05/2021 ADV JAVA AND J2EE 3GN18CS034

Q1) What is enumeration in java? With an example syntax explain enumeration

Soln:- Enumeration is a list of named constants In java, an enumeration defines a class type. In java, an enumeration can have constructors, methods and instance variables.

* An enumeration is created using the enum keyword.

```
// An enumeration of apple varieties
enum Apple {
        Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

* The identifiers Jonathan, GoldenDel, and so on, are called enumeration constants. Each is implicitly declared as a public, static final member of Apple.

* Their type is the type of enumeration in which they are declared, which is Apple in this case. These constants are called self-typed, in which "self" refers to the enclosing enumeration.

* Once you have defined an enumeration, you can create a variable of that type.

Eg:- Apple ap;

* this declares ap as a variable of enumeration type Apple. The only values that it can be assigned (or can contain) are those defined by enumeration. For eg, this assigns ap the value RedDel:

Eg:- ap = Apple.RedDel;

Q2) Implement java enumeration by using default explicit parameterized constructor and method? Explain values() and valueOf() method with suitable example

Soln:- Java enum with constructor

For example, constructor accept one string argument action

```
public enum TrafficSignal{
        RED ("wait"), GREEN ("go"), ORANGE("slow down");

        private String action;

        public String getAction(){
                return this.action;
    }

// enum constructor - cannot be public or protected
    TrafficSignal (String action){
            this.action = action;
    }
}

public class EnumConstructorExample {
            public static void main (String args[]){

                TrafficSignal[] signals = TrafficSignal.values();

                for(TrafficSignal signal : signals){

                    System.out.println("name :" +signal.name()+ "action :"
                                                    +signal.getAction());
                    }
    }
}
```

* All enumerations automatically contain two predefined methods: values() and valueOf().
* Their general forms are shown here:

```
public static enum-type[] values()
public static enum-type valueOf(String str)
```

* The values() method returns an array that contains a list of the enumeration constants.
* The valueOf() method returns the enumeration constant whose value corresponds to the string passed in str.

```
* enum Apple{
        Jonathan, GoldenDel, RedDel, WineSap, Cortland
  }
  class EnumDemo{
        public static void main(String args[])
        {
            Apple ap;
            System.out.println("Here are all Apple constants: ");
            //use values()
            Apple allapples[] = Apple.values();
            for(Apple a : allapples)
                System.out.println(a);

            System.out.println();

            //use valueOf()
            ap = Apple.valueOf("WineSap");
            System.out.println("ap contains "+ap);
        }
  }
```

The output of the program is:

Here are all Apple constants:
Jonathan
GoldenDel
RedDel
Winesap
Cortland

ap contains Winesap

Q3) Enumerations are class type. Justify with suitable example program.

Soln :- A java enumeration is a class type. Although, you don't instantiate an enum using new.
* Each enumeration constant is an object of its enumeration type.
* Constructor is called when each enumeration constant is created.

```
// Use an enum constructor, instance variable, and method
enum Apple {
    Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);

    private int price;  // price of each apple

    // Constructor
    Apple(int p) { price = p; }
    int getPrice()
    {
        return price; }
}
```

```
class EnumDemo {

    public static void main(String args[])
    {
        Apple ap;

        // Display price of Winesap.

        System.out.println("Winesap costs " + Apple.Winesap.getPrice() +
                                                "cents.\n");

        // Display all apples and prices.

        System.out.println("All apple prices:");
        for (Apple a : Apple.values())
            System.out.println(a + " costs " + a.getPrice() + " cents.");

    }
}
```

Output:

```
Winesap costs 15 cents

All apple prices:
Jonathan costs 10 cents.
GoldenDel costs 9 cents.
RedDel costs 12 cents.
Winesap costs 15 cents.
Cortland costs 8 cents.
```

Q4) With an example program describe
(a) ordinal() (b) compareTo() (c) equals() (d) Enum class

Soln: (a) ordinal :-
You can obtain a value that indicates an enumeration constant's position in the list of constants. This is called its ordinal value, and it is retrieved by calling the ordinal() method, shown below:

```
final int ordinal()
```

It returns the ordinal value of the invoking constant. Ordinal values begins at zero.

(b) CompareTo() method :- You can compare the value of two constants of the same enumeration by using the compareTo() method.

```
final int compareTo(enum-type e)
```

* Here, enum-type is the type of the enumeration and e is the constant being compared to the invoking constant.
* Remember, both the invoking constant and e must be of same enumeration.
* If the two ordinal values are same, then zero is returned.
* If the invoking constant has ordinal value less than e's, then it returns negative value. If invoking constant has ordinal value greater than e's, then a positive value is returned.

(c) equals() :- can compare an enumeration constant to any other object, those two objects will only be equal if they both refer to the same constant, within the same enumeration.
* Simply having ordinal values in common will not cause equals() to return true if the two constants are from different enumerations

Program to demonstrate ordinal(), compareTo() and equals()

```
enum Apple{
        Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
class EnumDemo{
        public static void main(String args[])
        {
            Apple ap, ap2, ap3;

            //obtain all ordinal values using ordinal()
            System.out.println("Here are all apple constants"+"and their
                                                    ordinal values: ");

            for(Apple a : Apple.values())
                    System.out.println(a+ " " +a.ordinal());

            ap = Apple.RedDel;
            ap2 = Apple.GoldenDel;
            ap3 = Apple.RedDel;
            System.out.println();

            //Demonstrate compareTo() and equals()
            if( ap.compareTo(ap2)<0)
                    System.out.println(ap+ "comes before "+ap2);

            if( ap.compareTo(ap2)>0)
                    System.out.println(ap2+ "comes before"+ap);

            if (ap.compareTo(ap3)==0)
                    System.out.println(ap+ "equals "+ap3);
            System.out.println();
```

```
if (ap.equals(ap2))
    System.out.println("Error!");

if(ap.equals(ap3))
    System.out.println(ap + "equals" + ap3);

if(ap == ap3)
    System.out.println(ap + "==" + ap3);

}

}
```

Output :-

```
Here are all apple constants and their ordinal values:
Jonathan 0
GoldenDel 1
RedDel 2
Winesap 3
Cortland 4

GoldenDel comes before RedDel
RedDel equals RedDel

RedDel equals RedDel
RedDel == RedDel
```

19
85) Describe Boxing and Unboxing in java. Write java program for auto boxing and auto unboxing.

Soln:- Boxing:- The process of encapsulating a value within an object is called boxing.

Integer iOb = new Integer(100);

It wraps the integers value 100 inside an Integer object called iOb. Then obtains this value by calling intValue() and stores the result in i.

Unboxing:- The process of extracting a value from a type wrapper is called unboxing.

eg:- int i = iOb.intValue();

This unboxes the value in iOb

Autoboxing is the process by which a primitive type is automatically encapsulated (boxed) into its equivalent type wrapper whenever an object of that type is needed. There is no need to explicitly construct an object.

Auto-unboxing is the process by which the value of a boxed object is automatically extracted (unboxed) from a type wrapper when its value is needed. There is no need to call a method such as intValue() or doubleValue

```
//Demonstrate autoboxing/unboxing
class AutoBox {
        public static void main(String args[]) {

                Integer iOb = 100;      //autobox an int
                int i = iOb;            // auto-unbox
                System.out.println(i + " " + iOb); // displays 100 100
        }
}
```

06) Write a java program for Autoboxing and Unboxing by using Boolean and character objects.

Soln :-

```
// Autoboxing/unboxing a Boolean and Character.
class AutoBox {
        public static void main(String args[]){

            // Autobox/unbox a boolean.
            Boolean b=true;

            // Below b is auto-unboxed when used in
            // a conditional expression, such as an if.
            if (b)
              System.out.println(" b is true");

            // Autobox/ unbox a char.
            Character ch='x';  // box a char
             char ch2=ch;      // unbox a char

            System.out.println("ch2 is "+ch2);

        }

}
```

Output :-

```
b is true
ch2 is x
```

7) What is Annotation? List some of the advantages in Annotations in java

Soln:- Annotation (Metadata):- a new facility added to Java that enables you to embed supplemental information into a source file. This information is called an annotation, does not change the actions of a program.

* Annotation is created through a mechanism based on the interface.

Eg:- // A simple annotation type

```
@interface MyAnno {
        String str();
        int val();
}
```

Advantages of Annotation:-

* Static type checking:- the compiler will check for you where the annotation (once defined properly) is applicable and how.
* Clean code - its much easier to see (visually) the metadata defined in annotations.
* Annotations are attached to classes and can be processed via reflection.

8) Apply the Annotation method overriding, method deprecation and warning avoidance. Describe the above with suitable program.

Soln:- Java defines many built-in annotations. There are seven general purpose. Four are imported from java.lang.annotation: @Retention, @Documented, @Target, and @Inherited.
Three — @Override, @Deprecated, and @SuppressWarnings - are included in java.lang

* @Override:- @Override is a marker annotation that can be used only on methods. A method annotated with @Override must override a method

from a superclass. If it doesn't, a compile-time error will result. It is used to ensure that a superclass method is actually overridden, and not simply overloaded.

@Deprecated :- @Deprecated is a marker annotation. It indicates that a declaration is obsolete and has been replaced by a newer form.

@SuppressWarnings :- @SuppressWarnings specifies that one or more warnings that might be issued by the compiler are to be suppressed. The warnings to suppress are ~~suppress~~ specified by name, in string form. This annotation can be applied to any type of declaration.

```
import java.util.List;

class Food{}
class Hay extends Food{}
class Animal{
        Food getPreferredFood(){
            return null;
        }
        @Deprecated
        static void deprecatedMethod(){
        }
}
class Horse extends Animal{
        Horse(){
            return;
        }
        @Override      //Compiler error if getPreferredFood
                       //overloaded, not overridden
        //Notice return type is different (covariant return type)
```

```
Hay getPreferredFood(){
        return new Hay();
    }

    @SuppressWarnings({"deprecation", "unchecked"})
    void useDeprecatedMethod(List raw){
        // deprecation warnings - suppressed
        Animal.deprecateMethod();
        // unchecked warning - suppressed
        raw.add(new Horse());
    }
}
```

Q9) What is Annotation? Explain types of Annotations.

Soln:- Java annotations are metadata (data about data) for our program source code.

Annotations can be categorized as:

(i) Predefined Annotation :-

(a) @Deprecated :- is a marker annotation that indicates the element (class, method, field, etc) is deprecated and has been replaced by a newer element.

→ Syntax :-

```
@Deprecated
accessModifier returnType deprecatedMethodName(){ . . }
```

(b) @Override :- specifies that a method of a subclass overrides the method of the superclass with the same method name, return type and parameter list.

(c) @SuppressWarnings :- annotation instructs the compiler to suppress warnings that are generated while the program executes.

* We can specify the type of warning to be suppressed. The warnings that can be suppressed are compiler-specific but there are two categories of warnings: deprecation and unchecked.

Eg:- @SuppressWarnings({"deprecated", "unchecked"})

(2) Custom Annotation :-

It is also possible to create our own custom annotations.

Syntax:-

```
[Access Specifier] @interface<AnnotationName>{
  Data Type <Method Name>() [default value];
}
```

* Annotations can be created by using @interface followed by the annotation name. The annotation can have elements that look like methods but they do not have an implementation.

* The default value is optional. The parameters can not have a null value.

(3) Meta Annotation :- are annotations that are applied to other annotations.

(a) @Retention :- specifies the level up to which the annotation will be available.

Syntax:-

```
@Retention(RetentionPolicy).
```

(b) @Documented :- is a marker interface that tells a tool than an annotation is to be documented. It is designed to be used only as an annotation to an annotation declaration.

Syntax :-

```
@Documented
public @interface MyCustomAnnotation { ... }.
```

(c) @Target :- specifies the type of declarations to which an annotation can be applied. We can restrict an annotation to be applied to a specific targets using the @Target annotation.
@Target takes one argument, which must be a constant from the ElementType enumeration.
Syntax :- @Target(ElementType)

Target Constant	Annotation Can Be Applied To.
ANNOTATION_TYPE	Another annotation
CONSTRUCTOR	Constructor
FIELD	Field
LOCAL_VARIABLE	Local variable
METHOD	Method
PACKAGE	Package
PARAMETER	Parameter
TYPE	Class, interface, or enumeration.

To specify multiple values, you must specify them within braces-delimited list.
Eg:- @Target({ElementType.FIELD, ElementType.LOCAL_VARIABLE}).

(d) @Inherited :- is a marker annotation that can be used only on another annotation declaration. It affects only annotations that will be used on class declarations. @Inherited causes the annotation for a superclass to be inherited by a subclass. Therefore, when request for a specific annotation is made to the subclass:
⇒ if that annotation is not present in subclass, then its superclass is checked.
⇒ if annotation is present in the superclass, and if it is annotated with @Inherited; then that annotation will be returned.

Eg:- @Inherited
public @interface MyCustomAnnotation{ ... }

@MyCustomAnnotation
public class ParentClass{ ... }
public class ChildClass extends ParentClass{ ... }

10) Implement a java program
(a) By applying annotation at RUNTIME
(b) By specify annotation target method
(c) Make annotation available for subclass.

Soln:- (a)
```
import java.lang.annotation.*;
import java.lang.reflect.*;

//An annotation type declaration
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno{
        String str();
        int val();
}

class Meta{
//Annotate a method.
@MyAnno(str = "Annotation Example", val = 100)
public static void myMeth(){
        Meta ob = new Meta();

        //Obtain the annotation for this method and display
        //the values of the members
        try{

        //First, get a class object that represents this class.
        Class c = ob.getClass();

        //Now, get a Method object that represents this method.
        Method m = c.getMethod("myMeth");
```

```
//Next, get annotation for this class.
MyAnno anno = m.getAnnotation(MyAnno.class);

//Finally, display the values.
System.out.println(anno.str() + " " + anno.val());
} catch (NoSuchMethodException exc){
        System.out.println("Method not found.");
}
}
public static void main(String args[]){
    myMeth();
}
}
```

Output:
Annotation Example 100

(B) By specify annotation target at method

```
@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
@interface MyAnnotation{
    int value1();
    String value2();
}
```

(c) Make annotation available for subclass

```
@Inherited
@interface ForEveryone { }  //Now it will be available to subclass also

@interface ForEveryone { }
class Superclass { }

class Subclass extends Superclass { }
```

Q11) List some of the restrictions imposed on annotations.

Soln:- There are number of restrictions that apply to annotation declaratio

⇒ First, no annotation can ~~it~~ inherit another.

⇒ Second, all methods declared by an annotation must be without paramele

⇒ Furthermore, they must return one of the following:

- A primitive type, such as int or double
- An object of type String or Class
- An enum type
- Another annotation type
- An array of one of preceding types.

⇒ Annotations can not be generic. In other words, they can not take parameters (Generics)

⇒ finally, annotation methods cannot specify a throws clause.

Q12) Write a java program for single-member annotation.

Soln:- import java.lang.annotation.*;
import java.lang.reflect.*;

14/03/24

Q1.

```
// A single-member annotation.
@Retention(RetentionPolicy.RUNTIME)
@interface MySingle {
        int value(); // this variable name must be value
}
class Single {

    // Annotate a method using a single-member annotation.
    @MySingle(100)
    public static void myMeth() {
        Single ob = new Single();
        try {
            Method m = ob.getClass().getMethod("myMeth");
            MySingle anno = m.getAnnotation(MySingle.class);
            System.out.println(anno.value()); // displays 100
        } catch (NoSuchMethodException exc) {
            System.out.println("Method not found");
        }
    }

    public static void main(String args[]) {
        myMeth();
    }
}
```

o/p:

100

Q13) Discuss Marker Annotation with a program example.

Soln:-

```
import java.lang.annotation.*;
import java.lang.reflect.*;

// A marker annotation.
@Retention(RetentionPolicy.RUNTIME)
@interface MyMarker{ }

class Marker{

    //Annotate a method using a marker
    //Notice that no() is needed.
    @MyMarker
    public static void myMeth() {

        Marker ob = new Marker();

        try{
            Method m = ob.getClass().getMethod("myMeth");

            //Determine if the annotation is present.
            if(m.isAnnotationPresent(MyMarker.class)
                System.out.println("MyMarker is present.");
        } catch(NoSuchMethodException exc) {

            catch    System.out.println("Method Not Found.");

    }

}
public static void main(String args[]){ myMeth(); } }
```

O/P:-

MyMarker is present.