What is enum and why enum?
An enum is a set of named constants those are represented as menu kind of items.
It is created by using the keyword "enum". It is internally a final class.

For example:
We should use enum in below situations
1) For creating Naresh IT courses list
2) Hotel menu items list
3) Colors list
4) Days, Months list in Calendar
5) Thread states list
6) Programming Elements List
   etc ...

How can we create enum?
For creating enum, in Java 5, a new keyword called "enum" is introduced.

```
[Accessibility modifier] enum <EnumName> {
     menu items named constants with "," seperator
                  ;
    all 10 static and non-static members which you defined in a normal class
}
```

Rule in creating named constant:
For creating named constants we must use only names, we should not use data type

Below code showing creating an enum and accessing and printing its names constants

```
//Month.java
enum Month{
        JAN, FEB, MAR
}

//Year.java
class Year{
        public static void main(String[] args){
                System.out.println( Month.JAN );
                System.out.println( Month.FEB );
                System.out.println( Month.MAR );
        }
}
```

Compilation and execution:

```
>javac Year.java
>java Year
        JAN
        FEB
        MAR
```

More examples:

1. Define an enum for storing Colors
2. Define an enum for storing Naresh ***i*** Technologies courses

```
//Color.java
enum Color {
        RED, BLUE, GREEN
}
```

```
//Course.java
enum Course{
        CRT, C, DS, ORACLE, JAVA, DOTNET, PYTHON
}
```

Exactly, what is the 'enum' and the 'named constants' inside enum?

Internally

1. the enum is final class
2. the named constant is a 'variable name' and 'value'

Compiler software converts

1. The keyword enum as 'final class' and
2. The named constants as
   public static final variables of current enum type
3. and initializes those named constant with the same named constant names

For example

In the enum Month, the named constants are converted as shown below

```
public static final Months JAN;
public static final Months FEB;
```

Compiler changed code for the enum Month in Month.class

```
final class Month extends java.lang.Enum {
   public static final Month JAN;
   public static final Month FEB;
   private static final Month[] $VALUES;

   public static Month[] values(){
      return (Month[])$VALUES.clone();
   }
   public static Month valueOf(String s){
      return (Month)Enum.valueOf(Month, s);
   }
   private Month(String name, int ordinal){
      super(name, ordinal);
   }
   static {
      JAN        = new Month("JAN",   0);
      FEB        = new Month("FEB",   1);

      $VALUES = new Month[]{ JAN, FEB };
   }
}
```

enum internals *Conclusion from above program*

1. "enum" keyword is replaced by "final & class" keywords
2. It is subclass of java.lang.Enum.
3. java.lang.Enum is an abstract class which is the default super class for every enum type classes. Also it is implementing from Comparable and Serializable interfaces
4. So every custom enum also become Comparable & Serializable type.
5. Since every enum type is Comparable, we can add its objects to collection TreeSet object, also it can be stored in file as it is Serializable type.
6. All named constants created inside enum are referenced type the current enum type. Compiler adds the missing code, and
7. These enum type variables are initialized in static block with this enum class object by using (String, int) parameter constructor
   Here
   - String parameter value is named constant name exactly as declared in its enum declaration
   - int parameter value is its position in the list. The position number starts from "ZERO"
8. In every "enum" type class compiler places private (String, int) parameter constructor with "super(s, i);" statement to call java.lang.Enum class constructor. These two values are stored in java.lang.Enum class non-static variables "name" and "ordinal" respectively, which are created in our enum class object separately for every enum variable's object.
9. Also below two additional methods are added
   1. To return all enum variable's objects
      public static <enumtype>[] values()
      For example:
      public static Months[] values()

   2. To return only the given enum object
      public static <enumtype> valueOf(String s)
      - here parameter is named constant name in String form
      For example:
      public static Months valueOf(String s)

      Months m = Months.valueOf("JAN")

10. It is also inheriting several methods from java.lang.Enum class.
    The important methods are
    1. public final String name()
       returns name of the current enum exactly as declared in its enum declaration

    2. public final int ordinal()
       returns current enum position

Q) Why compiler places no-arg constructor with no-arg super call in normal classes and parameterized constructor with (string,int) arg super class in enum?

A) For normal classes super class is java.lang.Object and it has no-arg constructor so compiler places default constructor without parameters with no-arg super call.

But for enum the super class is java.lang.Enum, it has String, int parameter constructor. So compiler places default constructor with (String, int) parameter with super call with string, int argument.

enum type class object JVM Architecture:

As we said in above points every enum type class object has minimum 2 variables

1. name *and*
2. ordinal

Every named constant variable defined inside enum type is an objects referenced variable that holds current enum class object. So, in the above Months enum class JAN, FEB are objects. Below is the Months enum class objects structure

Write a program to print name and ordinal of all enum objects of enum Months

// MonthsNameAndOrinal.java

```
class MonthsNameAndOrinal{
	public static void main(String[] args){
		Month[] months = Month.values();

		for (Month month : months){
			System.out.print( month.ordinal() + 1 +". ");
			System.out.println( month.name());
		}
	}
}
```

Q) How can we assign values to Menu items in enum? Is below syntax is valid?

```
	enum Months{
		JAN = 1, FEB = 2
	}
```

It is a wrong syntax, it leads to CE because named constants are not of type "int" they are of type "Month". So we can not assign values to named constants directly using "=" operator.

Q) Then how should we assign?

Syntax:

```
	namedconstant(value)
```

For example;

```
		JAN(1), FEB(2)
```

Rule: To assign values to names constants as shown in the above syntax, enum must have a parameterized constructor with the passed argument type. Else it leads to CE.

Q) Where these values 1, 2 are stored?
A) We must create a non-static int type variable to store these values.
So, to store these named constants values we must follow below 3 rules
In enum class

1. We must create non-static variable in enum with the passed argument type
2. Also we must define parameterized constructor with argument type
3. Named constants must end with ";" to place normal variables, methods, constructors explicitly. It acts as separator for compiler to differentiate named constants and general members.

Below code shows defining enum constants with values
//Month.java
```
enum Month{
        JAN(1), FEB(2)
                ;
        private int num;

        Month(int num){
                this.num = num;
        }
        public int getNum(){
                return num;
        }
        public void setNum(int num){
                this.num = num;
        }
}
```
Compilation:
```
>javac Month.java
        |->Month.class
```

Compiler changed code
In .class file developer given constructor code is merged with enum default constructor code that is given by compiler, as shown below

//Months.class

```
final class Months extends Enum{
   public static final Months JAN;
   public static final Months FEB;
   private int num;
   private static final Months $VALUES[];

   public static Months[] values(){
      return (Months[])$VALUES.clone();
   }
   public static Months valueOf(String s){
      return (Months)
           Enum.valueOf(Months, s);
   }

   private Months(String s, int i, int price){
      super(s, i);
      this.num = num;
   }

   public int getnum()  {
      return num;
   }

   public void setnum(int num){
      this.num= num;
   }

   static{
      JAN = new Months("JAN", 0, 1);
      FEB = new Months("FEB", 1, 2);
      $VALUES = (new Months[]{ JAN, FEB});
   }
}
```

Write a program to print above Months as how the real Months items are appeared.

Expected Output:

```
1. JAN
2. FEB
```

//Year.java

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

                Months[] menuItems = Months.values();

                for(Months menuItem : menuItems){
                   System.out.print( menuItem.getNum() + ". " );
                   System.out.println( menuItem.name() );
                }
        }
}
```

SCJP Questions (written test questions)

enum has below set of rules

Rule #1: on enum subclass

We cannot derive a subclass from enum, because it is final class

For example

```
    enum Flowers{}
    enum Rose extends Flowers{} CE: '{' expected
```

Rule #2: on enum object

We cannot instantiate enum using new keyword and constructor, it leads to CE: enum types may not be instantiated, but it is instantiated by compiler in .class file.

```
    enum Months{
        ;
        public static void main(String[] args){
            Months m1 =  new Months("JAN", 0);
        }
    }
```

Rule #3: on ";"

- Named constants must be separated with comma "," but not with semicolon ";"

  For example

  ```
  enum Color{
      RED ; BLUE ; CE: <identifier expected>
  }
  ```

  here BLUE is not considered as named constant,
  because ; represents end of all named constants.

- But we can place ; at end of all named constants, here in enum it is act as separator and also tells end of all named constants.

  ```
  enum Color{
      RED, BLUE ;
  }
  ```

- After named constants semicolon is mandatory to place normal members in enum definition. Normal members are static and non-static members

  For example

  ```
  enum Color{
      RED, BLUE

      int x = 10; ✗ CE: ',' , '}', or ';' expected
  }
  ```

  ```
  enum Color{
      RED, BLUE
      ;
      int x = 10; ✓
  }
  ```

Rule #4: on "enum members"

Inside enum we can define

1. Named constants
2. All static members- SV, SB, SM, MM
3. All Non-static members except abstract methods- NSV, NSB, NSM, Constructor
   Rule: Abstract method is not allowed because we cannot declare enum as abstract as enum object should be created to initialized named constants
4. inner class, interface, enum

For example:

```
enum Color{
        RED(15), BLUE(25), GREEN
                ;
        static int a = 10;
              int x = 20;

        static void m1(){
                System.out.println("SM");
        }
        void m2(){
                System.out.println("NSM");
        }

        static{
                System.out.println("SB");
        }
        {
                System.out.println("NSB");
        }

        Color(){
                System.out.println("no-arg constructor");
                this.x = 50;
        }
        Color(int x){
                System.out.println("int-arg constructor");
                this.x = x;
        }

        //abstract void m3(); CE:
        public static void main(String[] args){
                System.out.println("Color main");
        }
        class A{}
}// Color close
```

Mandatory

Q) How can we access enum members?

1. *named constants:* we can access named constants by using enum name (Color), because they are static
2. *static members:* we access static members by using enum name (color)
3. *non-static members*: we access non-static members by using enum objects i.e using named constants RED, BLUE

   Syntax:

   enumname.namedconstant.NSM

Below application shows accessing static and non-static members of an enum

```
class Test{
        public static void main(String[] args){
                System.out.println("Test main");

                //accessing named constants
                System.out.println(Color.RED);
                System.out.println(Color.BLUE);

                //accessing static members
                System.out.println(Color.a);
                Color.m1();

                //accessing non-static members
                System.out.println(Color.RED.x);
                System.out.println(Color.BLUE.x);
                System.out.println(Color.GREEN.x);

                Color.RED.m2();
                Color.BLUE.m2();
        }// main close
}// Test close
```

Compilation:

```
>javac Test.java
> java Color                 >java Test               RED
                             Test main                BLUE
NSB
int-arg Constructor          NSB                      10
NSB                          int-arg Constructor      SM
int-arg Constructor          NSB                      15
NSB                          int-arg Constructor      25
no-arg Constructor           NSB                      50
SB                           no-arg Constructor       NSM
Color main                   SB                       NSM
```

Rule #5: On enum members order
named constants must be first members in enum and they must be separated with *comma*

For example
Case #1: enum with Named Constant (NC), and Non-static member (NM)
```
enum Color{
        RED, BLUE
                ;
        int a = 10;
}
```

Case #2: enum with NC, NM, and NC
```
enum Color{
        RED;
        int a = 10;
        BLUE;
}
```

Case #3: enum with NM and NC
```
enum Color{
        int a = 10;
        RED,BLUE;
}
```

Case #4: empty enum
```
enum Color{
}
```

Case #5: enum with only NMs
```
enum Color{
        int a = 10;
}
```
Rule: To define enum only with NMs it must starts with ";" to tell to compiler it is not NC.
```
enum Color{
        ; int a = 10;
}
```

Rule #6: On constructor
We cannot declare explicit constructor as protected or public, because enum constructor is by default private.

Find out compile time errors
```
enum Color{
        private Color(){}
}
enum Color{
        Color(){}
}
enum Color{
        protected Color(){}
}
enum Color{
        public Color(){}
}
```

If we do not apply Accessibility Modifier to constructor compiler changes it to private.

Q) Hello where is ";" before constructor how will above enum is compiled without starting with " **;** "? **A)** ";" is optional if enum has only no-arg constructor.

Check below bits

Case #1: ";" is mandatory before constructor if we place constructor along with NCs

```
enum Months{
	JAN, FEB
	    ;
	Months(){}
}
```

Case #2: Also ";" is mandatory if we write constructor with parameters

```
enum Months{
	    ;
	Months(int a){}
}
```

Case #3: Also ";" is mandatory if we write no-arg constructor along with normal members

```
enum Months{
	    ;
	Months(){}
	int a= 10;
}
```

Case #4: Also ";" is mandatory if we write no-arg constructor as private

```
enum Months{
	    ;
	private Months(){}
}
```

Q) Can we overload constructor in enum?

We can overload constructors in enum to initialized named constant objects with different type of values.

For Example:

```
public enum Months{
	JAN(1), FEB(2L), MAR("3")
		;
	private int number;
	private Months(int num){
		this.number = num;
	}
	private Months(long num){
		this.number = (int)num;
	}
	private Months(String num){
		this.number = Integer.parseInt(num);
	}
}
```

Compiler changed code in the above program:
All three constructors are updated with name, ordinal parameters, and in body compiler places "super(s, i);" as first statement.

check below code:

```
public final class Months extends java.lang.Enum{

        public static final Months JAN;
        public static final Months FEB;
        public static final Months MAR;

        private int number;
        private static final Months $VALUES[];

        public static Months[] values(){
                return (Months[])$VALUES.clone();
        }
        public static Months valueOf(String s){
                return (Months)Enum.valueOf(Months, s);
        }

        private Months(String s, int i, int num){
                super(s, i);
                this.number = num;
        }
        private Months(String s, int i, long num){
                super(s, i);
                this.number = (int)num;
        }
        private Months(String s, int i, String num){
                super(s, i);
                this.number = Integer.parseInt(num);
        }
         static{
                JAN= new Months ("JAN", 0, 1);
                FEB = new Months ("FEB", 1, 2);
                MAR= new Months ("MAR", 1, 3);
                $VALUES = (new Months []{ JAN, FEB, MAR});
         }
}
```

Rule #7: On Named constants declaration
Named constants must be declared according to the constructors available in enum

For example:

Case #1: If we do not define constructor, it has default constructor. So, NCs must be declared without arguments

Find out compile time error

```
enum Months{
        JAN, FEB, MAR(3)
}
```

Case #2: If we define constructor with int parameter, then it does not have default constructor. So, all NCs must be declared with only int arguments

Find out compile time error

```
enum Months{
        JAN, FEB, MAR(3)
                ;
        Months(int i){}
}
```

Case #3: if we define constructors with no-arg, int and String parameters, all NCs must be declared either without argument or with int argument or with String argument

Find out compile time error

```
enum Months{
        JAN, FEB("2"), MAR(3), APR('a'), MAY(5L)
                ;
        Months(){}
        Months(int i){}
        Months(String s){}
}
```

Rule #8: allowed modifiers to enum

Only public, & strictfp modifiers are allowed for enum

Find out CEs in the below list

```
private enum Color{}
protected enum Color{}
public enum Color{}
static enum Color{}
final enum Color{}
abstract enum Color{}
strictfp enum Color{}
```

- private, protected, static are not allowed as they are only applicable to class members
- final is not allowed as it is already final
- abstract is not allowed as it should be instantiated and more over it is final

Rule #9: inner enum

We can define enum inside a class but we cannot define it inside a method

```
class A{
        enum Color{}  ✓
        void m1(){
                enum Color{} ✗
        }    CE: enum types must not be local
}
```

- private, protected, public, static, strictfp are allowed for inner enum.
- final, abstract modifiers are also not allowed for inner enum.