Java Enum Interview Questions for Developers with Answers

Java Enum Interview Questions with Answers

Java Enum Interview Questions with Answers for ProgrammersHere is my list of questions based on different features and properties of Java Enum. You can use this list for preparing Interview or simply as FAQ of Enum. If you are new to Java than I am sure you will learn a lot about Enum and it’s useful feature.

Question 1) Can Enum implement interface in Java?

Yes, Enum can implement interface in Java. Since enum is a type, similar to class and interface, it can implement interface. This gives a lot of flexibility to use Enum as specialized implementation in some cases. See here for an example of Enum implementing an interface in Java.

Question 2) Can Enum extends class in Java?

No, Enum can not extend class in Java. Surprised, because I just said it's a type like a class or interface in Java. Well, this is why this question is a good follow-up question of previous Enum interview question. Since all Enum by default extend abstract base class java.lang.Enum, obviously they can not extend another class, because Java doesn't support multiple inheritance for classes. Because of extending java.lang.Enum class, all enum gets methods like ordinal(), values() or valueOf().

Question 3) How do you create Enum without any instance? Is it possible without compile time error?

This is one of those tricky Java question, which Interviewer love to ask. Since Enum is viewed as a collection of well defined fixed number of instances e.g. Days of Week, Month in a Year, having an Enum without any instance, may seems awkward. But yes, you can create Enum without any instance in Java, say for creating a utility class. This is another innovative way of using Enum in Java. Here is the code

Public enum MessageUtil {

; // required to avoid compiler error, also signifies no instance

public static boolean isValid() {

throw new UnsupportedOperationException("Not supported yet.");

}

}

Question 4) Can we override toString() method for Enum? What happens if we don't?

Ofcourse you can override toString in Enum, as like any other class it also extends java.lang.Object and has toString() method available, but even if you don't override, you will not going to regret much, because abstract base class of enum does that for you and return name, which is name of the enum instance itself. here is the code of toString() method from Enum class :

public String toString() {

return name;

}

name is set, when compiler emit code for creating enum in response to instance declaration in enum class itself, along with setting ordinal, as visible in this constructor of enum from java.lang.Enum class :

protected Enum(String name, int ordinal) {

this.name = name;

this.ordinal = ordinal;

}

This is the only constructor of creating enum, which is called by code, generated by compiler in response to enum type declaration in Java program.

Question 5) Can we create instance of Enum outside of Enum itself? If Not, Why?

No, you cannot create enum instances outside of Enum boundry, because Enum doesn't have any public constructor, and compiler doesn't allow you to provide any public constructor in Enum. Since compiler generates lot of code in response to enum type declaration, it doesn’t allow public constructors inside Enum, which enforces declaring enum instances inside Enum itself.

Question 6) Can we declare Constructor inside Enum in Java?

This is asked along with previous question on Java Enum. Yes, you can, but remember you can only declare either private or package-private constructor inside enum. public and protected constructors are not permitted inside enum. See here for a code example.

Question 7) What is difference in comparing Enum with == and equals() method?

private enum Shape{

RECTANGLE, SQUARE, CIRCLE, TRIANGLE;

}

private enum Status{

ON, OFF;

}

Shape unknown = null;

Shape circle = Shape.CIRCLE;

boolean result = unknown == circle; *//return false*

result = unknown.equals(circle); *//throws NullPointerException*

I agree this can be avoided by simply **comparing known to unknown** i.e. circle.equals(unknown), but this is one of the most common [coding error Java programmers make](http://javarevisited.blogspot.com/2012/02/java-mistake-1-using-float-and-double.html). By using == to compare enum, you can completely avoid it.

**2) == method provides type safety during compile time**

Another advantage of using == to compare enum is, compile time safety. Equality or == operator checks if both enum object are from same enum type or not at compile time itself, while equals() method will also return false but at runtime. Since it's always better to detect errors at compile time, == scores over equals in case of comparing enum. If you are using [Eclipse](http://javarevisited.blogspot.com/2012/10/eclipse-shortcut-to-remove-all-unused-imports-java.html) or Netbeans, you can detect these error as soon as you type. By the way Netbeans also shows warning when you call equals() method on two incomparable types, but that is completely IDE specific.

**3) == should be faster than equals method**

This is more from common sense, using operator should be a touch faster than calling method, and than using operator. Though I believe modern JIT compiler might inline equals() method, when you compare two enums in Java. Which means this would not be big difference in terms of performance.But, without any smartness from compiler or [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html), I think == should always be touch faster.

Read more: <http://javarevisited.blogspot.com/2013/04/how-to-compare-two-enum-in-java-equals.html#ixzz57noRIyj0>

Question 8) What does ordinal() method do in Enum?

Ordinal method returns the order in which Enum instance are declared inside Enum. For example in a DayOfWeek Enum, you can declare days in order they come e.g.

Public enum DayOfWeek{

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;

}

here if we call DayOfWeek.MONDAY.ordinal() it will return 0, which means it's the first instance. This ordering can be very useful to represent actual real world ordering i.e. declaring TUESDAY after MONDAY, ensures that it came after MONDAY and before WEDNESDAY. Similarly you can use enum to represent Month of year in the order they come e.g. FEBRUARY after JANUARY and before MARCH. All user defined enum inherit this method from java.lang.Enum abstract class, and it's set by compiler, when it internally call protected constructor of java.lang.Enum, which accepts name and ordinal.

Question 9) Can we use Enum with TreeSet or TreeMap in Java?

public abstract class ****Enum<E extends Enum<E>>****

extends [Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)

implements [Comparable](https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html)<E>, [Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

This is really interesting question on Java Enum, I would love to ask this to gauge knowledge of Enum. Until you know about java.lang.Enum and has looked it's code, it's more likely that you don't know that Enum implements Comparable interface, which is main requirement to be used in Sorted Collection like TreeSet and TreeMap. Since Enum by default impalement Comparable interface, they can be safely used inside TreeSet or TreeMap in Java.

Question 10) What is difference between ordinal() and compareTo() in Enum?

This is follow-up of previous question on Java Enum. Actually, compareTo() mimic ordering provided by ordinal() method, which is the natural order of Enum. In short Enum constraints are compared in the order they are declared. Also, worth remembering is that enum constants are only comparable to other enum constants of the same enum type. Comparing enum constant of one type to another type will result in compiler error.

Question 11) Can we use Enum in switch case in Java?

Yes, you can use Enum in Switch case in Java, in fact that's one of the main advantage of using Enum. Since Enum instances are compile time constant, you can safely use them inside switch and case statements. Here is an example of using our DayOfWeek enum in switch case :

public void developerState(DayOfWeek today){

switch(today){

case MONDAY:

System.out.println("Hmmmmmmmm");

break;

case TUESDAY:

System.out.println("Hmmmm");

break;

case FRIDAY :

System.out.println("Yeahhhhhh");

break;

}

}

Enum and Switch cases goes well with each other, especially if Enum has relatively small number of fixed constants e.g. 7 days in week, 12 months in a year etc, See here for another example of using switch case with Enum in Java.

Question 12) How to iterate over all instance of a Enum?

Well, if you have explored java.lang.Enum, you know that there is a values() method which returns an array of all enum constant. Since every enum type implicitly extends java.lang.Enum, they get this values() method. By using, this you can iterate over all enum constants of a certain type. See here for a Enum values Example in Java for iterating over Enum using values() and foreach loop.

Question 13) What is advantage and disadvantage of using Enum as Singleton?

Enum provides you a quick shortcut to implement Singleton design pattern, and ever since it's mentioned in Effective Java, it's been a popular choice as well. On the face, Enum Singleton looks very promising and handles lot of stuff for you e.g. controlled instance creation, Serialization safety and on top of that, it’s extremely easy to create thread-safe Singleton using Enum. You don’t need to worry about double checked locking and volatile variable anymore. See here to know about pros and cons of using Enum as Singleton in Java.

Question 14) What is advantage of using Enum over enum int pattern and enum String pattern?

If you are coding from more than 5 years, and have coded in JDK 1.3 and 1.4, you must be familiar with Enum String pattern and enum int pattern, where we used public static final constants to represent collection of well known fixed number of things e.g. DayOfWeek. There was lot of problem with that approach e.g. you don't have a dedicated enum type, Since it's String variable, which represent day of week, it can take any arbitrary value. Similarly enum int pattern can take any arbitrary value, compiler doesn't prevent those. By using Enum, you get this type-safety and compiler checking for you. There are couple of good items on this topic in Effective Java, which is once again, must read for any Java developer.

Question 15) How to convert an String to Enum in Java?

This is a day to day ask, given popularity of String and Enum in Java application development. Best way for converting Enum to String, is to declare a factory method inside Enum itself, which should take String argument and return an Enum. You can choose to ignore case as well. See here for a code example of String to Enum conversion in Java.

Ans : Enum.valueOf(String)