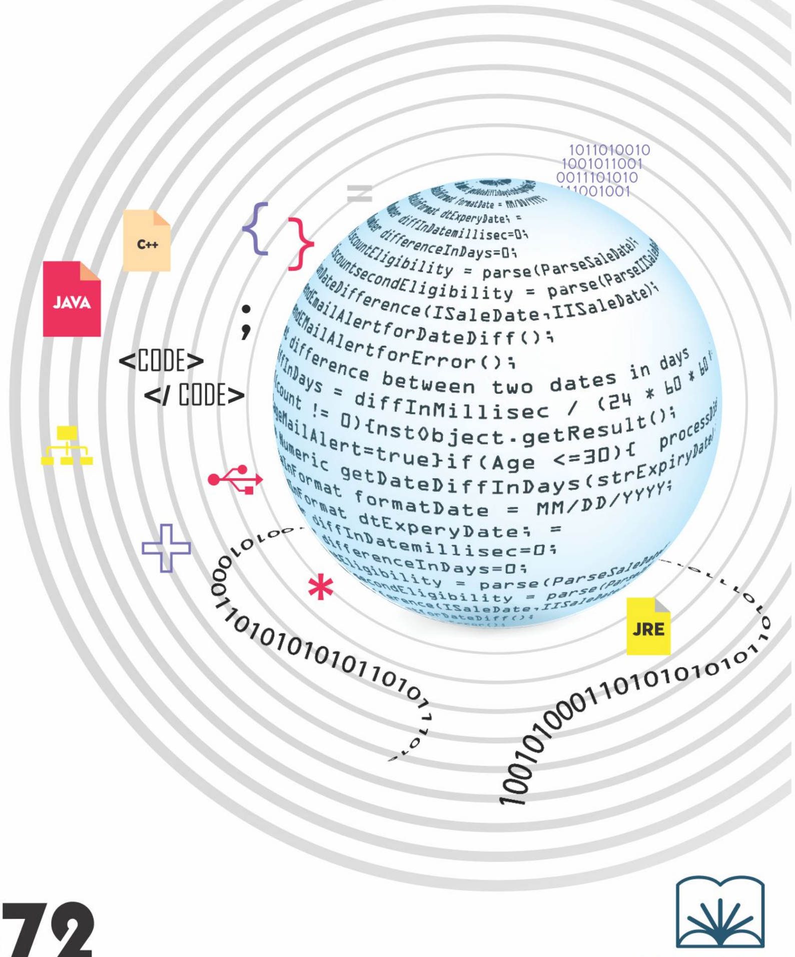
Job Interview Questions Series

ADVANCED JAVA

INTERVIEW QUESTIONS
YOU'LL MOST LIKELY BE ASKED



372
Interview Questions

VIBRANT
PUBLISHERS

Job Interview Questions Series

2e

ADVANCED JAVA

INTERVIEW QUESTIONS
YOU'LL MOST LIKELY BE ASKED

372
Interview Questions

VIBRANT
PUBLISHERS

Advanced JAVA

Interview Questions
You'll Most Likely Be Asked

ISBN-10: 1-946383-22-8
ISBN-13: 978-1-946383-22-8

Library of Congress Control Number: 2011907325

Table of Contents

Advanced Java Interview Questions

Review these typical interview questions and think about how you would answer them. Read the answers listed; you will find best possible answers along with strategies and suggestions.

This page is intentionally left blank.

Chapter 1

Object Serialization

1: What is Object Serialization? How is it done?

Answer:

Object Serialization is writing the object's properties and behaviour into a byte stream or a file. All the serializable objects referred inside it are also written into the file. This makes the object constant or invariable. This is usually used when an object has to be sent over a network. An object is Serializable when the class implements the Externalizable or Serializable interface. The object can then be passed on to the ObjectOutputStream which in turn passes it to the file output stream. When your object implements the Serializable interface which is a marker interface, you don't have to implement its methods whereas when the Externalizable interface is implemented you have to implement the readExternal() and writeExternal() methods.

2: Differentiate between Externalizable and Serializable interfaces?

Answer:

Serializable interface is a marker interface so you don't have to override the methods in it. But you have to override the readExternal() and writeExternal() methods of the Externalizable interface.

When you implement the Serializable interface, the JVM takes care of file streaming which can be inefficient at times. But since you are overriding the streaming methods in Externalizable interface, it is a more efficient way to file streaming.

If you are not sure of how to perform the IO streaming efficiently for your application it is better to implement the Serializable interface. If you can efficiently perform IO streaming specific to your application, Externalizable interface implementation is the best way.

When all or most of the attributes of an object has to be serialized, implementing the Serializable interface with use of transient variable as required will be more efficient. But when you have to serialize some dynamic attributes of large Java objects with too many attributes, implementing the Externalizable interface is a better way as you can specify what all have to be serialised efficiently in the overridden methods.

3: What is a serialVersionUID? How is it used?

Answer:

A Serial Version UID is a unique identifier for a serializable class to make sure that the serialised and the deserialised object refers to the same version. The Java compiler creates a unique serialVersionUID if it is not defined in the program. It is best to define a serialVersionUID for every serializable class as otherwise the JVM will not able to identify the class when its version changes. Everytime you change an attribute of the serializable class, the JVM creates a new serialVersionUID if it is not user-defined. So the best practice is to define a serialVersionUID for all serializable classes in Java so that the multiple versions will refer to the same object.

4: What is the next best option is we do not go for Object Serialization?

Answer:

Java uses Object Serialization to store the data permanently in the system's storage. The same can be done using other methods such as database, XML and JSON which is a comparatively recent method that uses Javascript. Using a database to store objects is a very common method. You can use the ORM or Object Relational Mapping to store and retrieve objects from and to the database. XML based data storage and transfer is now being commonly used by many web services. This is probably the most popular way to transfer data over the internet. The JSON data transfer is a relatively new format in use. Implementation of JSON is quite simple and it is integrated to most of the web browsers as it is based on Javascript.

5: How can the Java objects exist beyond the lifetime of the virtual machine?

Answer:

Object serialization allows Java objects to live beyond the lifetime of JVM.

6: I wish to serialize a collection. Is it a must to have all its members Serializable?

Answer:

Yes. All members of a collection or an array must be Serializable in order to Serialize it.

7: How to exclude certain variables from object's serialized state?

Answer:

To exclude variables from the process of serialization, they should be marked as transient.

Example: Transient private String variable;

8: Is it true to say that during object serialization class and instance variables that are not marked as transient are also serialized?

Answer:

No. Static variables belonging to the class are not saved as the part of serialized object. Also the transient class variables will not be part of serialized object.

9: What will happen if I try to serialize an object of a class that implements Serializable interface, but also includes a non-Serializable object?

Answer:

java.io.NotSerializableException will be thrown at runtime.

10: Once de-serialization is done, what values will transient variables get?

Answer:

Transient variables get default values after de-serialization.

11: You are making a class Serializable by implementing Serializable interface. Which methods are to be implemented?

Answer:

None.

12: Consider the following scenario:
public class ClassA extends ClassB implements Serializable{
ClassA extends from ClassB.
ClassA implements Serializable.
ClassA instance is serialized.
ClassA instance is de-serialized.
What values will the variables of ClassB get?

Answer:

ClassB's constructor will run and the variables will get the values assigned during the construction of the object.

13: What is the purpose of using serialVersionUID?

Answer:

serialVersionUID provides versioning system for every Serializable class. During de-serialization process, serialVersionUID provides means of checking whether data read from the input stream is compatible with the current class definition.

14: serialVersionUID is a static field declared in a Serializable class. Is it also serialized?

Answer:

serialVersionUID is a static field that is also serialized along with the other data. During de-serialization, the de-serialized serialVersionUID has to match to the serialVersionUID declared in the class definition.

15: What happens if serialVersionUID of de-serialized object does not match to the one declared in the class definition?

Answer:

java.io.InvalidClassException is thrown.

16: Consider that you have a Serializable class without serialVersionUID. How compiler will handle this?

Answer:

Java compiler adds serialVersionUID automatically and its value is based on the fields declared in the class.

17: What methods are there in Serializable interface?

Answer:

Serializable interface is a marker interface and has no methods. It simply tells the object serialization tools that the class is Serializable.

18: In which scenarios serialization should be used?

Answer:

Objects are serialized when sending over the network.

19: What can be the cause of NotSerializableException during object serialization?

Answer:

If a class is to be serialized, it has to implement Serializable interface. It is also important that all the objects included in that class are also Serializable. NotSerializableException is thrown if any of the included objects is not Serializable.

20: Which methods can be overridden to change default serialization behavior in order to control complex object serialization process?

Answer:

writeObject() and readObject() can be implemented to control complex object serialization process. By doing so, we can provide additional information to serialize and de-serialize objects.

21: Which interface is to be implemented if we want to have complete control over your class's serialization process?

Answer:

java.io.Externalizable interface is to be implemented.

22: Name the methods that are to be overridden when Externalizable is implemented.

Answer:

readExternal and writeExternal are to be overridden when Externalizable is implemented.

23: Consider the following scenario:
public class Parent implements Serializable
..
public class Child extends Parent
..
Is Child class Serializable?

Answer:

Child class inherits serialization from its object hierarchy and it is Serializable.

24: Is it correct to say that when an object is de-serialized, its constructor is called?

Answer:

De-serialization means restoring the serialized object and not re-constructing it. The constructor is not called in the de-serialization process.

25: I have a class called Student that is Serializable. It has an instance variable of type String called 'name'. Since 'name' is not of primitive type, should we expect serialization failure?
..
public class Student implements Serializable{
private String name;
..

Answer:

java.lang.String itself is Serializable, therefore Student class can be serialized.

26: Are all primitive wrapper classes Serializable?
Answer:
Yes. All primitive wrapper classes implement Serializable interface.

27: Assume you have a Serializable class where its super class does not implement Serializable interface. The super class defines a no- argument constructor and a String argument constructor. Which constructor of the super class will get called during de-serialization?
Answer:
No-argument constructor will get called. It should be accessible to the Serializable subclass class.

28: Consider the following scenario:
You have a Serializable class without serialVersionUID defined. You serialize it. You add a new instance variable in your Serializable class and de-serialize its already serialized instance. What will happen?
Answer:
Since serialVersionUID was not defined in the Serializable class, JVM will generate that. Once you add or remove instance variables in the Serializable class, the value of serialVersionUID will change. In the given scenario, java.io.InvalidClassException will be thrown.

This page is intentionally left blank.

Chapter 2

Generics

29: Explain Java Generics.

Answer:

Java Generics allows creating collections that accept only specific type of data and thus you can eliminate type-casting. Another advantage of Generics is that you can create generic algorithms or methods that works for different data types without having to program separately for each data type.

Example:

```
List<String> listGen = new ArrayList<String>();  // We are declaring the ArrayList to store string objects

listGen.add("hello");  // adding a string

String strGen = listGen.get(0);  // Since the ArrayList was declared for string objects, no type-casting is required.
```

In the above code, if you add the following line, it will throw a compile time error:

```
listGen.add(21); // compile time error as 21 is not a string
```

30: Write a program to explain Generic Methods.

Answer:

```
public class exampleForGenericMethod{
    public static < E > void printArrayElements(E[] elementsVal) {
    for ( E elementVal : elementsVal){
        System.out.println(elementVal);
    }
    System.out.println();
        }
public static void main( String abcs[] ) {
    Integer[] myIntArray = { 15, 25, 35, 45, 55 };
    Character[] myCharArray = { 'H', 'E', 'L', 'L', 'O', 'W', 'O', 'R', 'L', 'D'
    };
    System.out.println( "The Integer Array is:" );
    printArrayElements (myIntArray  );
    System.out.println( "The Character Array is: " );
    printArrayElements ( myCharArray );
        }
        }
```

This program will print:

```
The Integer Array is:
15
25
35
45
55
The Character Array is:
```

```
H
E
L
L
O
W
O
R
L
D
```

We have used the same method and avoided method overloading also and instead used Java Generics.

31: Differentiate between List<? extends T> and List <? super T>

Answer:

The symbol ? represents a wildcard meaning any type. So *? extends T* means the list will accept any object which extends T or which is a sub-class of T. *? super T* means List will accept any object which is a super class of T. So List <? extends Number> will accept Integer type and Float types meaning List<Integer> and List<Float> will be fine. List <?> will mean that a list of any type can be created. List<?> means List<String> and List<Integer> are fine.

32: Explain the following code:

Case 1:

List listORTExample = new ArrayList(); // Line 1

```
listORTExample.add("abc");  // Line 2
listORTExample.add(123);  // Line 3
String strItem = (String) listORTExample.get(0); // Line 4
strItem = (String) listORTExample.get(1); // Line 5
```

Case 2:

```
List<String> listOSExample = new ArrayList(); // Line 6
listOSExample.add("abcd");  // Line 7
listOSExample.add(1234); // Line 8
strItem = listOSExample.get(0); // Line 9
```

Answer:

Line 3 – Though the compiler will allow this a runtime exception will be thrown

Line 4 - Explicit cast is required as the List was not declared as String Type

Line 5 – Will throw ClassCastException because Integer cannot be cast in String

Line 8 – This line will throw a compiler error which is better than runtime Exception as against Line 3

Line 9 – Thanks to Line 6 where the Arraylist is declared for String Objects, no explicit casting is required

This is a classic example of how Java Generics improves the program.

33: Explain the Type Parameters in Generics.

Answer:

A Type Parameter is a place holder for argument types. Java has 5 type parameters

T denotes a Type

E denotes an Element

K denotes a Key

N denotes a Number

V denotes a Value

A Type parameter can be used for an argument and return type of methods, a target type in type-cast and an open type of argument for parameterized methods. A Type parameter cannot be used to create an object, array or in exception handling. It cannot be static and cannot be used with instanceOf operator. It cannot be used as a supertype or a class literal.

34: How to avoid casting when retrieving elements from a collection?

Answer:

Casting can be avoided using Generic collections.

Example:

```
List<String> gList = new ArrayList<String>();
```

35: Review the following code. Will it compile? Will it run?

```
public static void main(String[] args) {
List<String> typeSafeList = new ArrayList<String>();
typeSafeList.add(new String("string1"));
addElementsToList(typeSafeList);
String a = typeSafeList.get(1);
}
private static void addElementsToList(List list){
list.add(new Integer(10));
```

}

Answer:

The code will compile fine but will fail with the following exception:

java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String

36: Can you use more than one parameterized type in a declaration? Give an example.

Answer:

It is possible.

Example:

public class UseTwo<T, X> { }

37: How can you declare a generic method using a type say: T, which is not defined in the class?

Answer:

public <T> void makeList(T t) { }

The syntax is a bit tricky. This method has void as return type and declares the type before the return type.

38: Can you instantiate an array of a generic type?

Answer:

Generic arrays can be declared but cannot be instantiated just like a normal array. It gives a compile-time error since the type of the array is not known during compilation. Instead, it has to be instantiated using the new object() and type cast to an array. Another option is to create a parameterised array without mentioning the type. For example, the following code works:

```
arrType<String> [] abc = new arrType[50];
```

But this will not work:

```
arrType2<xyz> [] abc = new arrType<xyz>[50];
```

This page is intentionally left blank.

Job Interview Questions Series

ADVANCED JAVA

INTERVIEW QUESTIONS
YOU'LL MOST LIKELY BE ASKED

New Questions Added

Advanced JAVA Interview Questions You'll Most Likely Be Asked is a perfect companion to stand ahead above the rest in today's competitive job market. Rather than going through comprehensive, textbook-sized reference guides, this book includes only the information required immediately for job search to build an IT career. This book puts the interviewee in the driver's seat and helps them steer their way to impress the interviewer.

The following is included in this book:

- **297 Advanced JAVA** Interview Questions, Answers and proven strategies for getting hired as an IT professional
- **Dozens** of examples to respond to interview questions
- **75 HR** Questions with Answers and proven strategies to give specific, impressive, answers that help nail the interviews
- **2 Aptitude Tests** download available on www.vibrantpublishers.com

VIBRANT
PUBLISHERS

www.vibrantpublishers.com

ISBN 978-1-946383-22-8
90000>
9 781946 383228