♥♦ 6 Java Modifiers every interviewer seems to like

Posted on August 10, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. In Java, what purpose does the key words **final**, **finally**, and **finalize** fulfill?
A1. Covered in detail at Q1-Q10: Top 50+ Core Java Interview Questions & Answers

Q2. What is the difference between '**final**' and '**const**' modifiers on a variable?
A2. This is a bit of a tricky question because the '**const**' is a reserved keyword in Java, but not used. In C++ it means a variable is a constant(i.e. its values cannot be changed).

final

Final modifier on a reference variable just means that the reference cannot be changed to reference a different object once assigned. This does not mean that the variable is a constant because the values of the object it refers to can be modified unless the values themselves are marked as final. For example, an *Employee* object marked as final may have member variables “firstName”, and “lastName”. The values of “firstName”, and “lastName” can be modified if they themselves are not marked as final.

```
1
2     final Employee emp1 = new Employee("John", "Pe
3     emp1 = new Employee(); // compile-time error.
4
5     //if 'firstName' variable in Employee class is
6     emp1.setFirstName("Simon"); //Line A - firstNa
7
8     //if 'firstName' itself marked as final then,
9
```

const

is a reserved keyword in Java to make a variable constant, so that the referenced object values also cannot be modified, but it is currently not used in Java. The compiler will complain if you use it.

Q3. What is a **volatile** key word in Java?
A3. The volatile keyword is used with object and primitive variable references to indicate that a variable’s value will be modified by different threads.

Volatile

means

- The value of this variable will never be cached locally within the thread, and all the reads and writes must go to the main memory to be **visible** to the other

threads. In other words the keyword volatile **guarantees visibility.**

- From JDK 5 onwards, writing to a volatile variable happens before reading from a volatile variable. In other words, the volatile keyword guarantees **ordering,** and prevents compiler or JVM from reordering of the code.

Q4. How does **volatile** keyword differ from the **synchronized** keyword?

A4.

1. The **volatile** keyword is applied to variables of both primitives and objects, whereas the synchronized keyword is applied to only objects.
2. The volatile keyword only guarantees visibility and ordering, but not atomicity, whereas the synchronized keyword can guarantee both visibility and atomicity if done properly. So, the volatile variable has a limited use, and cannot be used in compound operations like incrementing a variable.

Wrong use of volatile in a compound operation

```
1  volatile int counter = 0;
2
3 public void increment(){
4    counter++;
5 }
6
```

Right use of volatile. **Example1**:

```
1   volatile boolean status = false;
2
3  //...
4
5  public void process(){
6     while(!status){
7     //....
8     }
9  }
10
```

Or in lazy singleton. **Example2**: Double checked locking

```
1   public final Class MySingleton {
2
3       private static volatile MySingleton instanc
4
5       private MySingleton( ){}
6
7       public static MySingleton getInstance() {
8               if(instance == null) {
9                       synchronized (MySingleton.class
10                              if(instance == null) {
11                                      instance = new
12                              }
13                      }
14              }
15
16              return instance;
17      }
18
19  }
20
```

Important: Synchronized keyword (i.e. locking) can guarantee both visibility and atomicity, whereas volatile variables can only guarantee visibility. A synchronized block can be used in place of volatile but the inverse is not true.

So, if you are not sure where to use, then use the "synchronized" keyword, and stay clear of the volatile modifier. You can learn more in detail at "10+ Atomicity, Visibility, and Ordering interview Q&As on Java Memory Model (JMM) to understand multi-threading"

Q5. What is a "transient" modifier? Can you mark a static variable as transient?
A5. It marks a member variable not to be serialized when it is persisted to streams of bytes. It cannot be used with a static variable as a static variable belongs to a class, not to an object. You can only serialize an object.

Transient

Serialization converts an object state to serial bytes (i.e. flattening an object). Those bytes are sent over the network and the object is recreated from those bytes. Member variables marked by the java transient keyword are not transferred over the wire. A "File" object cannot be serialized.

```
1 final class SerializeExample implements Serializa
2   transient File f;
3   public Ser() throws FileNotFoundException {
4     f = new File("c:\\temp\\filename");
5   }
6 }
```

Non memory objects like sockets, file handles, etc cannot be serialized, hence mark them as “transient”.

Note: **@Transient** annotation suggests that the object should not be persisted in Hibernate.

Q6. What value will the following method return?

```
1
2 public static int getSomeNumber( ){
3    try{
4       return 2;
5    } finally {
6       return 1;
7    }
8 }
9
```

A6. 1 is returned because ‘finally’ has the right to override any exception/returned value by the try..catch block. It is a bad practice to return from a finally block as it can suppress any exceptions thrown from a try..catch block. For example, the following code will not throw an exception.

```
1
2 public static int getSomeNumber( ){
3  try{
4   throw new RuntimeException( );
5  } finally {
6   return 12;
7  }
8 }
9
```

Q7. What can prevent execution of a code in a finally block?
A7. a) An end-less loop.

```
1
2  public static void main(String[ ] args) {
3   try {
4    System.out.println("This line is printed .....
```

```
5    //endless loop
6    while(true){
7     //...
8    }
9   }
10  finally{
11   System.out.println("Finally block is reached."
12  }
13 }
14
```

b) System.exit(1) statement.

```
1
2  public class Temp {
3
4      public static void main(String[ ] args) {
5          try {
6              System.out.println("This line is pri
7              System.exit(1);
8          }
9          finally{
10             System.out.println("Finally block is
11         }
12     }
13 }
14
```

c) Thread death or turning off the power to CPU.
d) An exception arising in a finally block itself.
e) Process p = Runtime.getRuntime( ).exec(“”);

If using Java 7 or later editions, use AutoCloseable statements within the try block.

Popular Posts

♦ 11 Spring boot interview questions & answers

857 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

825 views

18 Java scenarios based interview Questions and Answers

447 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

400 views

♦ 7 Java debugging interview questions & answers

311 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

301 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

292 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold

35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ 02: ♥♦ 10 Java String class interview questions & answers

♦ 30+ FAQ Java Object Oriented Programming (i.e. OOP) interview Q&As ›

Posted in Reserved Key Words

Tags: Core Java FAQs, Java/JEE FAQs, Novice FAQs

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?

Senior Java developers must have a good handle on

open all | close all
⊞ Best Practice (6)
⊞ Coding (26)
⊞ Concurrency (6)
⊞ Design Concepts (7)
⊞ Design Patterns (11)
⊞ Exception Handling (3
⊞ Java Debugging (21)
⊞ Judging Experience I
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all
⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (2
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (2
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)
⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

Preparing for Java written & coding tests

open all | close all
- ♦ Complete the given
- Can you write code? (
- Converting from A to I
- Designing your classe
- Java Data Structures
- Passing the unit tests
- What is wrong with th
- Writing Code Home A
- Written Test Core Jav
- Written Test JEE (1)

How good are your…to go places?

open all | close all
- Career Making Know-
- Job Hunting & Resum

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

 ↑