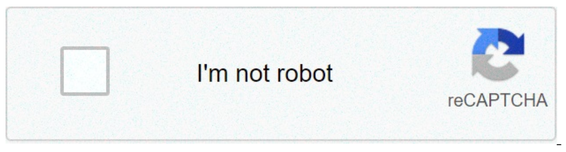

Continue

C and java interview questions and answers pdf

Content Writer at Truemark Technology. Company Website Link - and beginners often get stuck in their career as a Java developer and every work need a certain effort and practice to master. Developers often ask questions to their team or the public for solutions. As a Java developer, there are thousands of problems that they face every day. This problem could be critical or minor.Java is a general-purpose object-oriented programming language. For a Java Developer or any developer, the errors keep on occurring. Most commonly programmers tend to face such errors while practicing or experimenting.So, we have created a list of most asked questions about Java to help you guys. Here is a list of questions about Java.10 Most Asked Questions About JavaIf you are also a Java developer and if you have also faced errors while coding and if you have some questions about Java, then please first have a look at the solutions below. The below questions about Java are described with the best solutions as possible.1. Is Java “pass-by-reference” or “pass-by-value”?Answer:Java is always pass-by-value. Unfortunately, when we pass the value of an object, we are passing the reference to it. This is confusing to beginners as these questions about java are frequently asked by beginners and it goes like this:public static void main(String[] args) { Dog aDog = new Dog("Max"); Dog oldDog = aDog; // we pass the object to foo foo(aDog); // aDog variable is still pointing to the "Max" dog when foo(...) returns aDog.getName().equals("Max"); // true aDog.getName().equals("Fifi"); // false aDog == oldDog; // true } public static void foo(Dog d) { d.getName().equals("Max"); // true // change d inside of foo() to point to a new Dog instance "Fifi" d = new Dog("Fifi"); d.getName().equals("Fifi"); // true }In the example above aDog.getName() will still return "Max". The value aDog within main is not changed in the function foo with the Dog "Fifi" as the object reference is passed by value. If it were passed by reference, then the aDog.getName() in main would return "Fifi" after the call to foo.Likewise:public static void main(String[] args) { Dog aDog = new Dog("Max"); Dog oldDog = aDog; foo(aDog); // when foo(...) returns, the name of the dog has been changed to "Fifi" aDog.getName().equals("Fifi"); // true // but it is still the same dog: aDog == oldDog; // true } public static void foo(Dog d) { d.getName().equals("Max"); // true // this changes the name of d to be "Fifi" d.setName("Fifi"); }In the above example, Fifi is the dog’s name after a call to foo(aDog) because the object’s name was set inside of foo(...). Any operations that foo performs on d are such that, for all practical purposes, they are performed on aDog, but it is not possible to change the value of the variable aDog itself.Additional Answer:The Java Spec says that everything in Java is pass-by-value. There is no such thing as “pass-by-reference” in Java.The key to understanding this is that something likeis not a Dog; it’s actually a pointer to a Dog.What that means, is when you haveDog myDog = new Dog("Rover"); foo(myDog);you’re essentially passing the address of the created Dog object to the foo method.Suppose the Dog object resides at memory address 42. This means we pass 42 to the method.if the Method were defined aspublic void foo(Dog someDog) { someDog.setName("Max"); // AAA someDog = new Dog("Fifi"); // BBB someDog.setName("Rowlf"); // CCC }let’s look at what’s happening.the parameter someDog is set to the value 42at line “AAA” someDog is followed to the Dog it points to (the Dog object at address 42) that Dog (the one at address 42) is asked to change his name to Maxat line “BBB” a new Dog is created. Let’s say he’s at address 74 we assign the parameter someDog to 74at line “CCC”someDog is followed to the Dog it points to (the Dog object at address 74) that Dog (the one at address 74) is asked to change his name to Rowlfthen, we returnNow let’s think about what happens outside the method:Did myDog change?There’s the key.Keeping in mind that myDog is a pointer, and not an actual Dog, the answer is NO. myDog still has the value 42; it’s still pointing to the original Dog (but note that because of the line “AAA”, its name is now “Max” – still the same Dog; myDog’s value has not changed.)It’s perfectly valid to follow an address and change what’s at the end of it; that does not change the variable, however.Java works exactly like C. You can assign a pointer, pass the pointer to a method, follow the pointer in the method, and change the data that was pointed to. However, you cannot change where that pointer points.In C++, Ada, Pascal,m, and other languages that support pass-by-reference, you can actually change the variable that was passed.If Java had pass-by-reference semantics, the foo method we defined above would have changed where myDog was pointing when it assigned someDog on line BBB.Think of reference parameters as being aliases for the variable passed in. When that alias is assigned, so is the variable that was passed in.2. How to read/convert an InputStream into a String in Java?Answer:A nice way to do this is using Apache commons IOUtils to copy the InputStream into a StringWriter , something likeStringWriter writer = new StringWriter(); IOUtils.copy(inputStream, writer, encoding); String theString = writer.toString();or even// NB: does not close inputStream, you'll have to use try-with-resources for that String theString = IOUtils.toString(inputStream, encoding); Alternatively, you could use ByteArrayOutputStream if you don’t want to mix your Streams and WritersAdditional Answer:Also, there are various ways to convert InputStream into a string in JavaUsing IOUtils.toString (Apache Utils)String result = IOUtils.toString(inputStream, StandardCharsets.UTF_8);Using CharStreams (Guava)String result = CharStreams.toString(new InputStreamReader(inputStream, Charsets.UTF_8));Scanner s = new Scanner(inputStream).useDelimiter("\\A"); String result = s.hasNext() ? s.next() : "";Using Stream API (Java 8). Warning: This solution converts different line breaks (like \r) to .String result = new BufferedReader(new InputStreamReader(inputStream)) .lines().collect(Collectors.joining(""));Using parallel Stream API (Java 8). Warning: This solution converts different line breaks (like \r) to .String result = new BufferedReader(new InputStreamReader(inputStream)).lines() .parallel().collect(Collectors.joining(""));Using InputStreamReader and StringBuilder (JDK)final int bufferSize = 1024; final char[] buffer = new char[bufferSize]; final StringBuilder out = new StringBuilder(); Reader in = new InputStreamReader(stream, StandardCharsets.UTF_8); int charsRead; while((charsRead = in.read(buffer, 0, buffer.length)) > 0) { out.append(buffer, 0, charsRead); } return out.toString();Using StringWriter and IOUtils.copy (Apache Commons)StringWriter writer = new StringWriter(); IOUtils.copy(inputStream, writer, "UTF-8"); return writer.toString();Using ByteArrayOutputStream and inputStream.read (JDK)ByteArrayOutputStream result = new ByteArrayOutputStream(); byte[] buffer = new byte[1024]; int length; while ((length = inputStream.read(buffer)) != -1) { result.write(buffer, 0, length); } // StandardCharsets.UTF_8.name() > JDK 7 return result.toString("UTF-8");Using BufferedReader (JDK). Warning: This solution converts different line breaks (like \r) to line.separator system property (for example, in Windows to “\r”).String newLine = System.getProperty("line.separator"); BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream)); StringBuilder result = new StringBuilder(); boolean flag = false; for (String line; (line = reader.readLine()) != null;) { result.append(flag? newLine: "").append(line); flag = true; } return result.toString();Using BufferedInputStream and ByteArrayOutputStream (JDK)BufferedInputStream bis = new BufferedInputStream(inputStream); ByteArrayOutputStream buf = new ByteArrayOutputStream(); int result = bis.read(); while(result != -1) { buf.write((byte) result); result = bis.read(); } // StandardCharsets.UTF_8.name() > JDK 7 return buf.toString("UTF-8");Using inputStream.read() and StringBuilder (JDK). Warning: This solution has problems with Unicode, for example with Russian text (works correctly only with non-Unicode text)int ch; StringBuilder sb = new StringBuilder(); while((ch = inputStream.read()) != -1) sb.append((char)ch); reset(); return sb.toString();Warning:Solutions 4, 5, and 9 convert different line break to one.Solution 11 can’t work correctly with Unicode textPerformance testsPerformance tests for small String (length = 175), (mode = Average Time, system = Linux, score 1,343 is the best): Benchmark Mode Cnt Score Error Units 8. ByteArrayOutputStream and read (JDK) avgt 10 1,343 ± 0,028 us/op 6. InputStreamReader and StringBuilder (JDK) avgt 10 6,980 ± 0,404 us/op 10. BufferedInputStream, ByteArrayOutputStream avgt 10 7,437 ± 0,735 us/op 11. InputStream.read() and StringBuilder (JDK) avgt 10 8,977 ± 0,328 us/op 7. StringWriter and IOUtils.copy (Apache) avgt 10 10,613 ± 0,599 us/op 1. IOUtils.toString (Apache Utils) avgt 10 10,605 ± 0,527 us/op 3. Scanner (JDK) avgt 10 12,083 ± 0,293 us/op 2. CharStreams (guava) avgt 10 12,999 ± 0,514 us/op 4. Stream Api (Java 8) avgt 10 15,811 ± 0,605 us/op 9. BufferedReader (JDK) avgt 10 16,038 ± 0,711 us/op 5. parallel Stream Api (Java 8) avgt 10 21,544 ± 0,583 us/opPerformance tests for big String (length = 50100) (mode = Average Time, system = Linux, score 200,715 is the best): Benchmark Mode Cnt Score Error Units 8. ByteArrayOutputStream and read (JDK) avgt 10 200,715 ± 18,103 us/op 1. IOUtils.toString (Apache Utils) avgt 10 300,019 ± 8,751 us/op 6. InputStreamReader and StringBuilder (JDK) avgt 10 347,616 ± 130,348 us/op 7. StringWriter and IOUtils.copy (Apache) avgt 10 352,791 ± 105,337 us/op 2. CharStreams (guava) avgt 10 420,137 ± 59,877 us/op 9. BufferedReader (JDK) avgt 10 632,028 ± 17,002 us/op 5. parallel Stream Api (Java 8) avgt 10 662,999 ± 46,199 us/op 4. Stream Api (Java 8) avgt 10 701,269 ± 82,296 us/op 10. BufferedInputStream, ByteArrayOutputStream avgt 10 740,837 ± 5,613 us/op 3. Scanner (JDK) avgt 10 751,417 ± 62,026 us/op 11. InputStream.read() and StringBuilder (JDK) avgt 10 2919,350 ± 1101,942 us/opGraphs (performance tests depending on Input Stream length in Windows 7 system)Performance test (Average Time) depending on Input Stream length in Windows 7 system: length 182 546 1092 3276 9828 29484 58968 test8 0.38 0.938 1.868 4.448 13.412 36.459 72.708 test4 2.362 3.609 5.573 12.769 40.74 81.415 159.864 test5 3.881 5.075 6.904 14.123 50.258 129.937 166.162 test9 2.237 3.493 5.422 11.977 45.98 89.336 177.39 test6 1.261 2.12 4.38 10.698 31.821 86.106 186.636 test7 1.601 2.391 3.646 8.367 38.196 110.221 211.016 test1 1.529 2.381 3.527 8.411 40.551 105.16 212.573 test3 3.035 3.934 8.606 20.858 61.571 118.744 235.428 test2 3.136 6.238 10.508 33.48 43.532 118.044 239.481 test10 1.593 4.736 7.527 20.557 59.856 162.907 323.147 test11 3.913 11.506 23.26 68.644 207.591 600.444 1211.5453. How to Avoid != null statements?Answer:There are two instances where null checking comes up:Where null is a valid response in terms of the contract; andWhere it isn’t a valid response.(2) is easy. Either use assert statements (assertions) or allow failure Assertions are a highly-underused Java feature that was added in 1.4. The syntax is:orassert : where is a boolean expression and is an object whose toString() method’s output will be included in the error.An assert statement throws an Error (AssertionError) if the condition is not true. By default, Java ignores assertions. You can enable assertions by passing the option -ea to the JVM. You can enable and disable assertions for individual classes and packages. This means that you can validate code with the assertions while developing and testing, and disable them in a production environment, although my testing has shown next to no performance impact from assertions.Not using assertions, in this case, is OK because the code will just fail, which is what will happen if you use assertions. The only difference is that with assertions it might happen sooner, in a more meaningful way and possibly with extra information, which may help you to figure out why it happened if you weren’t expecting it.(1) Is a little harder. If you have no control over the code you’re calling then you’re stuck. If null is a valid response, you have to check for it.If it’s code that you do control, however (and this is often the case), then it’s a different story. Avoid using nulls as a response. With methods that return collections, it’s easy: return empty collections (or arrays) instead of nulls pretty much all the time.With non-collections, it might be harder. Consider this as an example: if you have these interfaces:public interface Action { void doSomething(); } public interface Parser { Action findAction(String userInput); }Where Parser takes raw user input and finds something to do, perhaps if you’re implementing a command-line interface for something. Now you might make the contract that it returns null if there’s no appropriate action. That leads the null checking you’re talking about.An alternative solution is to never return null and instead use the Null Object pattern:public class MyParser implements Parser { private static Action DO_NOTHING = new Action() { public void doSomething() { /* do nothing */ } }; public Action findAction(String userInput) { // ... if (/* we can't find any actions */) { return DO_NOTHING; } } }Compare:Parser parser = ParserFactory.getParser(); if (parser == null) { // now what? // this would be an example of where null isn't (or shouldn't be) a valid response } Action action = parser.findAction(someInput); if (action == null) { // do nothing } else { action.doSomething(); }toParserFactory.getParser().findAction(someInput).doSomething();which is a much better design because it leads to more concise code.That said, perhaps it is entirely appropriate for the findAction() method to throw an Exception with a meaningful error message — especially in this case where you are relying on user input. It would be much better for the findAction method to throw an Exception than for the calling method to blow up with a simple NullPointerException with no explanation.try { ParserFactory.getParser().findAction(someInput).doSomething(); } catch(ActionNotFoundException anfe) { userConsole.err(anfe.getMessage()); }Or if you think the try/catch mechanism is too ugly, rather than Do Nothing your default action should provide feedback to the user.public Action findAction(final String userInput) { /* Code to return requested Action if found */ return new Action() { public void doSomething() { userConsole.err("Action not found: " + userInput); } } }4.Differences between HashMap and Hashtable?Answer:In Java, there are several differences between HashMap and Hashtable :Hashtable is synchronized, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.One of HashMap’s subclasses is LinkedHashMap, so in the event that you’d want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn’t be as easy if you were using Hashtable.Since synchronization is not an issue for you, then HashMap is better. If synchronization becomes an issue, you may also look at oncurrentHashMap.Additional Answer:A very common idiom is to “check then put” — i.e. look for an entry in the Map, and add it if it does not already exist. This is not in any way an atomic operation whether you use Hashtable or HashMap.An equivalently synchronized HashMap can be obtained by:Collections.synchronizedMap(myMap);But to correctly implement this logic you need additional synchronization of the form:synchronized(myMap) { if (!myMap.containsKey("tomato")) myMap.put("tomato", "red"); }Even iterating over a Hashtable’s entries (or a HashMap obtained by Collections.synchronizedMap) is not thread-safe unless you also guard the Map from being modified through additional synchronization.Implementations of the ConcurrentMap interface (for example ConcurrentHashMap) solve some of this by including thread-safe check-then-act semantics such as:ConcurrentMap.putIfAbsent(key, value);5. How and when should UserManager.isUserAGoat() be used?Answer:From the source, the method used to return false until it was changed in API 21./** * Used to determine whether the user making this call is subject to * teleportations. * @return whether the user making this call is a goat */ public boolean isUserAGoat() { return false; }It looks like the method has no real use for us as developers.In API 21 the implementation was changed to check if there is an installed app with the package com.coffeestainstudios.goatsimulator/** * Used to determine whether the user making this call is subject to * teleportations. * * As of {@link android.os.Build.VERSION_CODES#LOLLIPOP}, this method can * now automatically identify goats using advanced goat recognition technology. * * @return Returns true if the user making this call is a goat. */ public boolean isUserAGoat() { return mContext.getPackageManager() .isPackageAvailable("com.coffeestainstudios.goatsimulator"); }6. Why don’t Java’s +=, -=, *=, /= compound assignment operators require casting?Answer:In this case §15.26.2 Compound Assignment Operators. An extract:A compound assignment expression of the form E1 op= E2 is equivalent to E1 = (T) ((E1) op (E2)), where T is the type of E1, except that E1 is evaluated only once. Errors in casting can lead to critical failure.An example cited from §15.26.2and results in x having the value 7 because it is equivalent to:short x = 3; x = (short)(x + 4.6);In other words, i += j; is shortcut for i =(type of i) (i + j).7. How to create ArrayList from array?Answer:In Java, array list can be created from an array using the code which something looks like thisnew ArrayList(Arrays.asList(array)); Also the Simplest way can beList list = Arrays.asList(array);Alternative Answer:Given:Element[] array = new Element[] { new Element(1), new Element(2), new Element(3) };The simplest answer is to do:List list = Arrays.asList(array);This will work fine. But some caveats:The list returned from asList has fixed size. So, if you want to be able to add or remove elements from the returned list in your code, you’ll need to wrap it in a new ArrayList. Otherwise, you’ll get an UnsupportedOperationException.The list returned from asList() is backed by the original array. If you modify the original array, the list will be modified as well. This may be surprising.8. How to generate random integers within a specific range in Java?Answer:In Java 1.7 or later, the standard way to do this is as follows:import java.util.concurrent.ThreadLocalRandom; // nextInt is normally exclusive of the top value, // so add 1 to make it inclusive int randomNum = ThreadLocalRandom.current().nextInt(min, max + 1);See the relevanpt JavaDoc. This approach has the advantage of not needing to explicitly initialize a java.util.Random instance, which can be a source of confusion and error if used inappropriately.However, conversely, there is no way to explicitly set the seed so it can be difficult to reproduce results in situations where that is useful such as testing or saving game states or similar. In those situations, the pre-Java 1.7 technique shown below can be used.Before Java 1.7, the standard way to do this is as follows:import java.util.Random; /** * Returns a pseudo-random number between min and max, inclusive. * The difference between min and max can be at most * Integer.MAX_VALUE - 1. * * @param min Minimum value * @param max Maximum value. Must be greater than min. * @return Integer between min and max, inclusive. * @see java.util.Random#nextInt(int) */ public static int randInt(int min, int max) { // NOTE: This will (intentionally) not run as written so that folks // copy-pasting have to think about how to initialize their // Random instance. Initialization of the Random instance is outside // the main scope of the question, but some decent options are to have // a field that is initialized once and then re-used as needed or to // use ThreadLocalRandom (if using at least Java 1.7). // // In particular, do NOT do 'Random rand = new Random()' here or you // will get not very good / not very random results. Random rand; // nextInt is normally exclusive of the top value, // so add 1 to make it inclusive int randomNum = rand.nextInt((max - min) + 1) + min; return randomNum; }See the relevant JavaDoc. In practice, the java.util.Random class is often preferable to java.lang.Math.random().In particular, there is no need to reinvent the random integer generation wheel when there is a straightforward API within the standard library to accomplish the task.9. Why is char[] preferred over String for passwords?Answer:Strings are immutable. That means once you’ve created the String, if another process can dump memory, there’s no way (aside from reflection) you can get rid of the data before garbage collection kicks in.With an array, you can explicitly wipe the data after you’re done with it. You can overwrite the array with anything you like, and the password won’t be present anywhere in the system, even before garbage collection.So yes, this is a security concern – but even using char[] only reduces the window of opportunity for an attacker, and it’s only for this specific type of attack.It’s also possible that arrays being moved by the garbage collector will leave stray copies of the data in memory. The garbage collector may clear all memory as it goes, to avoid this sort of thing. Even if it does, there’s still the time during which the char[] contains the actual characters as an attack window.10. How to efficiently iterate over each entry in a Java Map?Answer:To efficiently iterate over each entry in a Java, use the following code: Map map = ... for (Map.Entry entry : map.entrySet()) { System.out.println(entry.getKey() + "/" + entry.getValue()); } Additional Answer:Also, this description with example may be useful for youFor example, if we want to find the sum of all of the keys and values of a map, we can write:Using iterator and Map.Entrylong i = 0; Iterator it = map.entrySet().iterator(); while (it.hasNext()) { Map.Entry pair = it.next(); i += pair.getKey() + pair.getValue(); }Using foreach and Map.Entrylong i = 0; for (Map.Entry pair : map.entrySet()) { i += pair.getKey() + pair.getValue(); }Using forEach from Java 8final long[] i = {0}; map.forEach((k, v) -> i[0] += k + v);long i = 0; for (Integer key : map.keySet()) { i += key + map.get(key); }Using keySet and iteratorlong i = 0; Iterator itr2 = map.keySet().iterator(); while (itr2.hasNext()) { Integer key = itr2.next(); i += key + map.get(key); }long i = 0; for (Iterator entries = map.entrySet().iterator(); entries.hasNext();) { Map.Entry entry = entries.next(); i += entry.getKey() + entry.getValue(); }Using the Java 8 Stream APIfinal long[] i = {0}; map.entrySet().stream().forEach(e -> i[0] += e.getKey() + e.getValue());Using the Java 8 Stream API parallelfinal long[] i = {0}; map.entrySet().stream().parallel().forEach(e -> i[0] += e.getKey() + e.getValue());Using IterableMap of Apache Collectionslong i = 0; MapIterator it = iterableMap.mapIterator(); while (it.hasNext()) { i += it.next() + it.getValue(); }Using MutableMap of Eclipse (CS) collectionsfinal long[] i = {0}; mutableMap.forEachKeyValue((key, value) -> { i[0] += key + value; });In ConclusionThis is the list of 10 most asked questions about Java. Hope this article helped you. Also, don’t forget to share this post since others like you also may have similar problems related to Java.Also, please feel free to ask any questions related to Java, we will be glad to help you.This post was originally posted on DevPost.Join Hacker Noon Create your free account to unlock your custom reading experience.