Candidate: Craig Wilkins

Interviewed by: Mark Knowlton at Thu, Sep 28 2023 02:37PM EDT

TechScreen™ AI Verify Score: 94/100

Play the interview recording

AI Verify Summary: The interview revolved around various aspects of Java programming, specifically 'final', 'finally', and 'finalize' keywords, garbage collection, virtual methods, and thread safety in Singleton. The interviewee demonstrated a good understanding of the topics, delivering comprehensive and accurate explanations. In discussing 'final', 'finally', and 'finalize', the interviewee accurately captured their concepts and behaviors, providing an in-depth understanding of their applications. However, the interviewee omitted the detail that 'finalize' is not guaranteed to run, a crucial point in the context. Despite this, the content and delivery were rated high due to the detailed explanations and understanding demonstrated. The explanation on garbage collection was both correct and detailed, earning high scores in content and delivery. The interviewee outlined the automated memory management process, as well as a two-phase approach to garbage collection. When discussing virtual methods, the interviewee correctly explained that all functions in Java are virtual by default and illustrated how dynamic method lookup works. Despite the thorough explanation, the delivery could have been more concise and contrasting viewpoints or evidence of subject mastery were lacking. Lastly, in answering how to make a Singleton thread safe, the interviewee correctly identified two methods but the explanation lacked depth and detail. Despite this, both the content and delivery scores were high as the answer was correct and matched the context. Overall, the interviewee demonstrated a good understanding of Java, providing accurate and detailed explanations. However, there is room for improvement in terms of incorporating contrasting viewpoints, examples, and demonstrating a deeper understanding of the topics.

[Advanced / Java]: Identify at least 4 benefits of using Interfaces and Abstract classes

- [x] An Interface-defined type can be implemented by any class in a class hierarchy and can be extended by another Interface
- [x] An Abstract-class-defined type can be implemented only by classes that subclass the Abstract class
- [x] An Interface-defined type can be used well in Polymorphism
- [x] Abstract classes evolve more easily than Interfaces
- [ ] If you add a new concrete method to an Abstract class, the hierarchy system is still working
- [ ] If you add a method to an Interface, the classes that rely on the Interface will break when recompiled
- [ ] Use Interfaces for flexibility
- [ ] Use Abstract classes for ease of evolution (like expanding class functionality)

Delivery:

⭐⭐⭐

Candidate confidently gave parts of answers, but didn't present them in an organized narrative.

AI Verify Explanation: The interviewee's response correctly identified several key benefits of using Interface-defined types and Abstract-class-defined types, aligning with the correct answers provided in the documents. These include the ability of any class in the hierarchy to implement Interface-defined types, their use in Polymorphism, and the fact that only classes that subclass the Abstract class can implement Abstract-class-defined types. Additionally, the interviewee correctly noted that Abstract classes evolve more easily than Interfaces. However, the delivery of the answer could have been improved. Although the interviewee provided a fundamental explanation, the response lacked contrasting viewpoints, examples, or evidence of subject mastery, which could have enriched the discussion and demonstrated a deeper understanding of the topic.

[Advanced / Java]: Explain the purpose of Final, Finally and Finalize?

☑ Finalize descends from the Java.Lang class and every object has one
☑ Finalize is called when an object is no longer being used (i.e., when there are no further references to it)
☑ Finalize performs cleanup of non-Java resources and frees system resources not handled by Java's garbage-collection system, prior to the JVM running GC
☑ Finalize is not guaranteed to run because an eligible object may not have a non-Java resource associated with it
☑ The Final keyword is a modifier that may be applied to classes, methods, and variables
☑ Final Class: It can't be sub-classed
☑ Final Method: Can't be overridden
☑ Final Variable: Can't be changed. One word of caution - making methods Final restrict later design decisions
☑ The Finally block is optional, and appears after the try block, and after zero or more catch blocks
☑ The code in a Finally block is executed once, regardless of how the code in the try block executes
☑ In normal execution, control reaches the end of the try block and proceeds to the Finally block, which generally performs any necessary cleanup
☑ Finally provides exception handling and routine cleanup for code in a try block
Delivery:
⭐⭐⭐⭐
Candidate presented an organized explanation but only gave a high-level description without details.

AI Verify Explanation: The interviewee's response is comprehensive and accurately captures the concepts of 'final', 'finally', and 'finalize' in Java. The explanation about 'final' being a keyword that can be applied to classes, methods, and variables aligns with the context information provided. The interviewee precisely describes the behavior of 'final' when applied to different elements and gives an in-depth understanding of how 'finally' works in a try-catch block. The discussion about 'finalize' and its role in the garbage collection process is detailed and demonstrates a deep understanding of the topic. However, the response lacks the information that 'finalize' is not guaranteed to run, which is an important aspect mentioned in the context information. Overall, the response is well-delivered with a clear and detailed explanation of the concepts, but it could be enhanced with the inclusion of the missing detail about 'finalize'.

[Advanced / Java]: Explain how Garbage Collection works and can you reference at least one GC algorithm?

☑ It is a way to automate memory management
☑ It is a low-priority daemon thread that runs in the background
☑ A developer creates objects that reside in the 'New' space of the heap, which stores objects in memory
☑ Objects in memory are tracked by the Java Virtual Machine, which keeps track of live references to obects
☑ The developer CAN'T force Garbage Collection, but they can make system calls 'system.gc' or 'runtime.gc' when they want to suggest that the JVM run the process
☑ The other way a developer can help influence is by setting references to an object as Null, so it will be marked as eligible as soon as it loses any active reference
☑ The JVM will look at objects that no longer have a reference and using the Mark and Sweep approach, will mark an object as 'eligible' for collection
☐ Objects that maintain a weak reference, like through a HashMap, can still be collected like any unreferenced object
☑ The JVM will make a second pass and 'sweep' or collect unreferenced objects to reclaim the memory that had been allocated so it can be re-allocated
☑ Before the Garbage Collector runs, the objects Finalize() method is called to clean up any non-Java resources (a database connection or connection to a file system, etc.) associated with the eligible object
☑ While objects remain in memory the JVM will separate them into separate regions of the Heap, referred to as Generations
☑ There are three separate sections: The New Generation, which is made up of the Eden Space and two different Survivor spaces, the Tenured Space and the Aging space, often referred to as the Perm Gen
☑ The Generational approach helps minimize the performance hit of reclaiming memory by introducing the concepts of Minor and Major GC. Minor GC just cleans out the Young Generation, where most objects are created and get marked as 'eligible'
Delivery:
⭐⭐⭐⭐
Candidate presented an organized explanation but only gave a high-level description without details.

AI Verify Explanation: The response given provides a correct and detailed explanation of garbage collection and how it functions. It refers to the 'Mark and Sweep' approach, which is a garbage collection algorithm, fulfilling the question's requirements. The explanation covers key points like automated memory management, the role of the Java Virtual Machine (JVM), the notion of object references, and the two-step process of garbage collection. The delivery of the answer is clear and well-structured, detailing the process of automated memory management, the role of the JVM, the creation of objects in the 'new' heap space, and the elimination

of completed tasks. It also outlines the two-step approach to garbage collection, including the 'mark' phase and the 'sweep' phase. However, the answer could have been improved with the inclusion of contrasting viewpoints or examples and a more in-depth understanding of the subject matter.

[Advanced / Java]: Can you have a Virtual Function or Method in Java?

☑ All functions (or methods) in Java are Virtual by default
☑ A Virtual function or method has behavior that can be overridden. An inheriting class that has a function with the same signature (method name) will override that method to provide the polymorphic behavior
Delivery:
⭐⭐⭐⭐
Candidate presented an organized explanation but only gave a high-level description without details.

AI Verify Explanation: The interviewee correctly explained that all functions in Java are virtual by default, aligning with the correct answers provided in the context. They gave a comprehensive explanation of how dynamic method lookup or dynamic binding works in Java, allowing a method in a lower class to override a method in its parent class. They even used an example of an abstract class and a drive class to illustrate this. However, while the response was accurate, it could have been more concise and straight to the point. The interviewee didn't provide contrasting viewpoints or display deep subject mastery in their response.

[Advanced / Java]: What are some ways you can make a Singleton thread safe?

☑ There are two ways to make the Singleton thread safe
☑ The first simply synchronizes the getInstance() method: public class Singleton {private static Singleton instance
☑ The second way is to declare a constant Singleton attribute on the Singleton class itself:
Delivery:
⭐⭐⭐
Candidate confidently gave parts of answers, but didn't present them in an organized narrative.

AI Verify Explanation: The response matches the context information well in terms of content, as it accurately answers the given question about making a Singleton thread safe. It correctly identifies and explains the two methods mentioned in the original document, although the question was slightly misphrased. However, the delivery of the answer could be improved. While it provides a basic explanation, it lacks depth and detail. There are no contrasting viewpoints or examples provided to illustrate the methods, and it does not demonstrate a deep understanding or mastery of the topic.