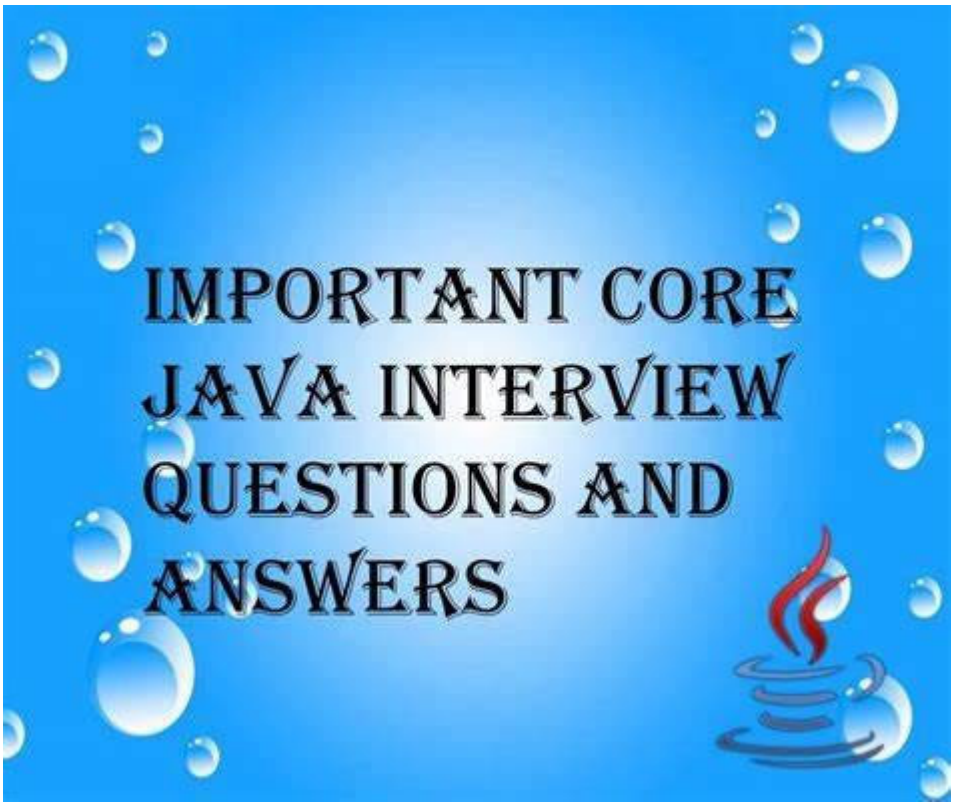
medium.com

Java tricky interview questions and answers. It covers questions with code examples | Medium

FullStackTips

5–6 minutes



Java tricky questions!

I am just adding some tricky Java questions that will not only help interview but also help you write better code in Java.

1. **Can an interface in Java have static methods?**

   Yes, starting from Java 8, an interface in Java can have static

methods.

A static method in an interface is a method that is associated with the interface itself, rather than with instances of the interface. They can be invoked without creating an instance of the interface, by using the interface name followed by the dot operator (.) and the method name.

Example:

```
interface MyInterface {
    static void staticMethod() {

    }
}
// Usage:
MyInterface.staticMethod();
```

It is worth noting that static methods in interfaces are also sometimes referred to as “static interface methods.”

A follow up question, **how many static methods can we have in the interface?**

There’s no specific limit to the number of static methods that you can have in an interface in Java. You can have as many as you need, as long as they are properly defined and their names are unique within the interface.

2. What is the difference between static method in class and static method in interface?

This can be a follow up question to the previous question. As might know a class can have a static method. A static method in a class is a method that belongs to the class itself, rather than to

instances of the class. It can be invoked without creating an instance of the class, and the method name. *A static method in a class can access and modify static variables of the class, but cannot access non-static variables or call non-static methods.*

On the other hand, a static method in an interface is a method that belongs to the interface itself, rather than to instances of the interface. It can be invoked without creating an instance of the interface, by using the interface name followed by the dot operator (.) and the method name. A static method in an interface cannot access any variables or call any methods, static or non-static.

To summarize, **a static method in a class can access class-level variables and methods, while a static method in an interface cannot**. Static methods in a class are typically used to provide utility methods or to define factory methods, while static methods in interfaces are typically used to provide common functionality that can be shared across multiple implementations of the interface.

3. What will be the output of the following program?

```
class A
{
   static int i = 1111;
    static
   {
            i = i-- - --i;
      System.out.println("In class-A-Static:"+i);
   }
    {
      i = i++ + ++i;
```

```
        System.out.println("In class-A instance:"+i);
    }
}
 class B extends A
{
    static
    {
        i = --i - i--;
        System.out.println("In class-B-static:"+i);
    }
     {
        i = ++i + i++;
        System.out.println("In class-B-init:"+i);
    }
}
 public class Main
{
    public static void main(String[] args)
    {
        B b = new B();
         System.out.println(b.i);
    }
}
```

This is primarily asked to check your understanding of class loading, static and instance initializers in Java. It also tests you analytical ability, and how the increment and decrement operators works.

The output will be.

will be…

will be… (intentionally left it, so you can think about the answer)

In class-A-Static:2
In class-B-static:0
In class-A instance:2
In class-B-init:6
6

The flow of the program execution is as follows.

1. Class `A` is loaded and its static initializer block is executed. The value of `i` is first decremented by 1 (`i--`) and then decremented by 1 again (`--i`), resulting in `i = 1111 - 1109 = 2`. If you noticed why 1109 here? because when `i` first evaluated it was 1110 so `--i` will be 1109. So `i` is 2 so far.
2. Class `B` is loaded and its static initializer block is executed. The value of `i` is first decremented by 1 (`--i`) and then assigned the value of `1- 1 . i = – i – i –` will be executed as `i = 1–1 =0`
3. Next class `A` instance initializer block is executed. So `i = i++ + ++i` will be evaluated as `i=0+2=2`
4. Next class `B` instance initializer block is executed. So `i = ++i + i++;` will be evaluated as `i=3+3=6.` because `++i` is 3 and `i++` is also 3!

   So output will be `6`

4. Can you give the output of the below program? not too tricky though

class Main

```
{
    int methodOfA()
    {
        return (true ? null : 0);
    }
    public static void main(String args[]){
        Main c = new Main();
        c.methodOfA();
    }
}
```

Because the return type of `methodOfA` is `int`, but it returns `null`, this code will result in a compile-time error: "incompatible types: <null> cannot be converted to int" or a `NullPointerException`. To fix this error, the return type of the method could be changed to `Integer` or the ternary operator could be changed to return an `int` value instead of `null`.

This blog will be updated with few more, please follow me for the updates.

If you're interested in joining Medium, use my personal referral link to become a member. You'll get unlimited access to great content and supportive community and a platform that values quality content.