perplexity

1. Interview-Style Opening

"This is a classic Java interview question that tests the understanding of **Polymorphism** versus **Static Binding**.

The short answer is **No**, you cannot override static methods in Java.
However, you can declare a static method with the exact same signature in the subclass, but that is called **Method Hiding**, not Overriding. This distinction is crucial because 'Overriding' implies runtime polymorphism (dynamic dispatch), whereas 'Hiding' relies on compile-time types."

2. Problem Understanding and Clarification

We need to distinguish between two concepts that look similar in code but behave differently in the JVM:

1. **Overriding (Instance Methods):** The method to call is decided at *Runtime* based on the actual object (e.g., `new Child()`).
2. **Hiding (Static Methods):** The method to call is decided at *Compile Time* based on the reference type (e.g., `Parent p`).

3. High-Level Approach (The "Why")

Why does Java work this way?

- **Static methods belong to the Class**, not the Object instance.
- Overriding is about objects responding differently to the same message. Since static methods don't use the `this` pointer or instance state, the JVM doesn't look at the object at runtime to decide which one to run. It simply looks at the class name on the left side of the dot.

4. Visual Explanation (Overriding vs. Hiding)

Diagram: Runtime vs. Compile Time

```
Parent p = new Child();
p.doWork();

   |
   v
```

```
  [ Is doWork() Static? ]
       |
       +------ YES (Static) ----->  Look at Reference Type (Parent)
       |                            Result: Calls Parent.doWork()
       |                            (This is METHOD HIDING)
       |
       +------ NO (Instance) ---->  Look at Actual Object (Child)
                                    Result: Calls Child.doWork()
                                    (This is METHOD OVERRIDING)
```

5. Java Code (Interview-Ready Example)

Here is the definitive code example I use to prove this behavior.

```
class Parent {
    // 1. Static Method
    public static void staticMethod() {
        System.out.println("Parent Static");
    }

    // 2. Instance Method
    public void instanceMethod() {
        System.out.println("Parent Instance");
    }
}

class Child extends Parent {
    // This is METHOD HIDING, not overriding.
    // If you add @Override here, the compiler will throw an error.
    public static void staticMethod() {
        System.out.println("Child Static");
    }

    // This is METHOD OVERRIDING.
    @Override
    public void instanceMethod() {
        System.out.println("Child Instance");
    }
}

public class Test {
    public static void main(String[] args) {
        // The Tricky Part: Reference is Parent, Object is Child
        Parent p = new Child();

        // Q1: Which static method is called?
        // Answer: Parent's. Because 'p' is of type Parent.
        // The "new Child()" part is ignored for static calls.
        p.staticMethod();

        // Q2: Which instance method is called?
        // Answer: Child's. Because 'p' points to a Child object.
        // This is Runtime Polymorphism.
        p.instanceMethod();
```

```
    }
}
```

Output:

```
Parent Static
Child Instance
```

6. Code Walkthrough

- `p.staticMethod()`: The compiler sees `p` is of type `Parent`. Since `staticMethod` is static, it binds the call to `Parent.staticMethod()` right there at compile time. It doesn't care that the object is actually a `Child` at runtime.
- `@Override` **on Static**: If you uncomment `@Override` above `Child.staticMethod`, the compiler will fail with "Method does not override method from its superclass," proving the point.

7. How to Explain This to the Interviewer

"To explain this, I like to use the concept of **Binding**.

Instance methods use **Dynamic Binding**. When I say `p.instanceMethod()`, the JVM waits until the code runs to check what `p` actually points to.

Static methods use **Static Binding**. When I say `p.staticMethod()`, the compiler decides *immediately* during compilation. It sees `p` is a `Parent` reference, so it hardcodes a call to `Parent.staticMethod`.

So, if I define a static method in the Child class with the same name, I am simply 'hiding' the Parent's version from anyone who accesses it via the Child class directly. But I haven't replaced it in the polymorphic sense."

8. Edge Cases and Follow-Up Questions

Q: Can you override a private method?
A: No. Private methods are not visible to subclasses, so you can't override or hide them. If you define a method with the same name in the Child, it's just a completely new, unrelated method.

Q: Can you overload a static method?
A: Yes! You can have multiple static methods with the same name but different parameters. This is standard overloading.

Q: What happens if I change the Child method to non-static?
A: Compilation Error. You cannot "override" a static method with an instance method, nor can you hide an instance method with a static one. The 'static-ness' must match.

⁂

1. https://www.geeksforgeeks.org/java/can-we-overload-or-override-static-methods-in-java/
2. https://discuss.codecademy.com/t/why-doesnt-java-allow-overriding-of-static-methods/808567
3. https://docs.oracle.com/javase/tutorial/java/landl/override.html
4. https://www.geeksforgeeks.org/java/overriding-in-java/
5. https://www.scaler.com/topics/can-we-override-static-method-in-java/
6. https://crunchify.com/java-method-hiding-and-overriding-override-static-method-in-java/
7. https://hyperskill.org/university/java/java-method-overriding
8. https://prepbytes.com/blog/can-we-override-static-method-in-java/
9. https://www.youtube.com/watch?v=WoAsR_8Rosw
10. https://www.reddit.com/r/java/comments/3v399x/12_rules_of_overriding_in_java_you_should_know/