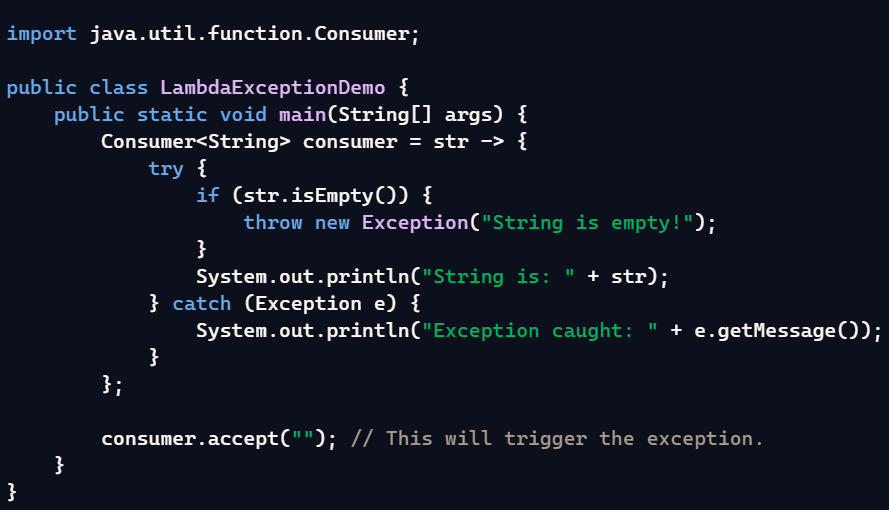
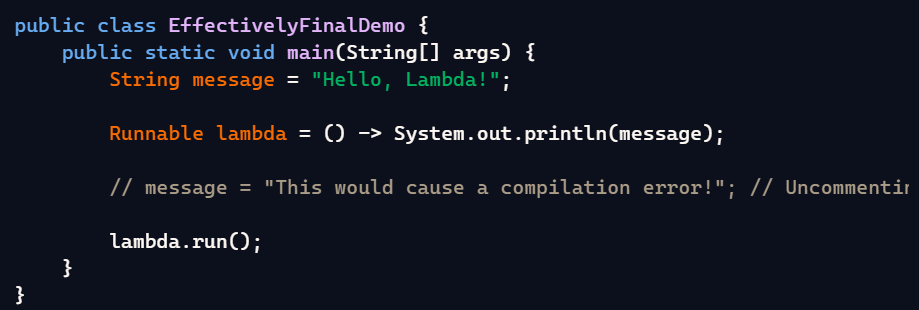
Yes, you can throw an exception in a lambda expression, but there are some nuances to keep in mind.

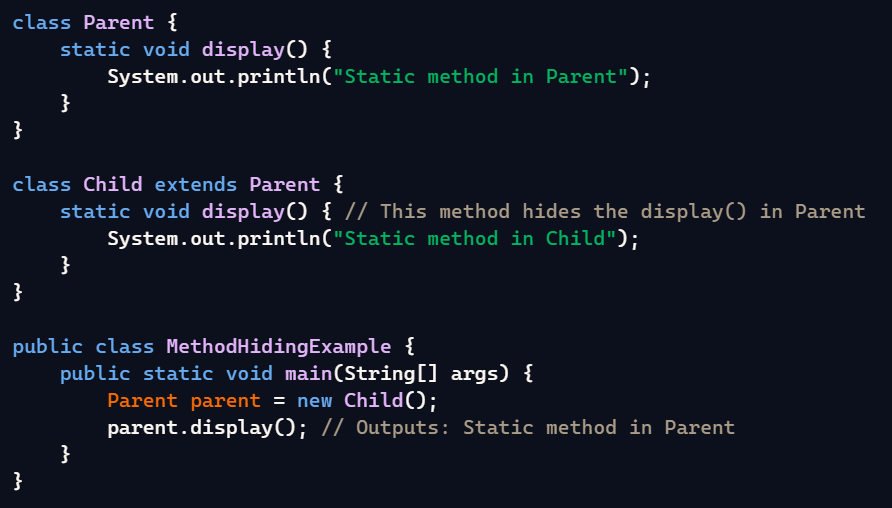
In Java, lambda expressions can throw exceptions, but the exception must align with the method's functional interface. If the interface method does not declare the exception in its throws clause, you will need to handle the exception within the lambda expression itself.



In Java, the term **"effectively final"** refers to a variable that is not modified after it is initialized, even though it is not explicitly declared as final. This concept is crucial when working with lambda expressions because lambda expressions can only use variables that are either explicitly final or effectively final.



Method hiding occurs when a **static method** in a subclass has the same name and parameter list as a static method in its superclass. In this case, the method in the subclass hides the method in the superclass. Unlike method overriding (which applies to instance methods), method hiding is determined at **compile-time**, and the method that's called is based on the type of the reference, not the object.



Variable hiding occurs when a **variable** in a subclass has the same name as a variable in its superclass. When this happens, the subclass variable hides the superclass variable. However, variable hiding is resolved based on the **reference type** (not the actual object) at **compile-time**.

A computer screen shot of text

AI-generated content may be incorrect.

Static Block: When static and non-static methods are in the main method, which method will compile first?

**Execution Order:**

1. **Static block**: Static blocks execute first, as they are initialized when the class is loaded into memory by the JVM. These are executed only once, before any methods (including main) or objects are created.
2. **Main method**: After the static blocks, the main method is executed, which serves as the entry point for the program.
3. **Instance (non-static) methods and blocks**: These are executed later if they are invoked by the main method or other parts of the program.

Covariant Return Type.

In Java, **covariant return types** refer to the ability of an overriding method in a subclass to return a type that is a subclass of the return type declared in the method of the superclass. This feature makes code more flexible and expressive while maintaining type safety.

**Key Points of Covariant Return Types:**

1. **Introduced in Java 5:** This feature allows the return type of the overriding method to differ from the superclass method, as long as it's compatible (i.e., it is a subclass of the original return type).
2. **Enhanced Flexibility:** It enables you to return more specific types in a subclass without forcing a cast on the caller.

In Java, it is necessary to maintain the same return type when overriding a method, but with an important exception related to covariant return types.

1. Exact Return Type in Method Overriding:

When you override a method, the return type must be exactly the same or a subtype (in case of covariant return types) of the original method's return type. In other words, the return type should either be the same or a more specific type (a subclass of the original return type).

2. Covariant Return Types:

Java allows covariant return types in method overriding. This means that you can override a method and change the return type to a subtype of the original method's return type, but not to a completely different type.

Default methods in Interface.

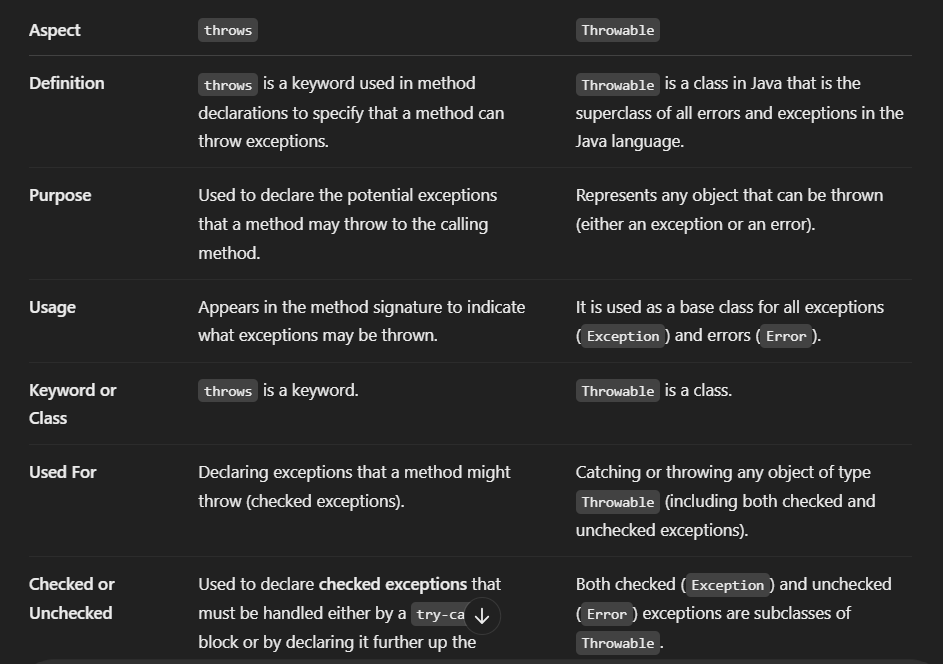
Ans.

In Interfaces A default method has a body, so you can provide a default implementation directly in the interface. This means that classes implementing the interface do not have to implement the method if they are okay with the default behaviour.

Variable types in Interface?

All variables in an interface are implicitly public, static, and final, regardless of whether you specify these modifiers explicitly or not.

Throw vs Throwable



A screenshot of a black screen

AI-generated content may be incorrect.

