Java Interfaces

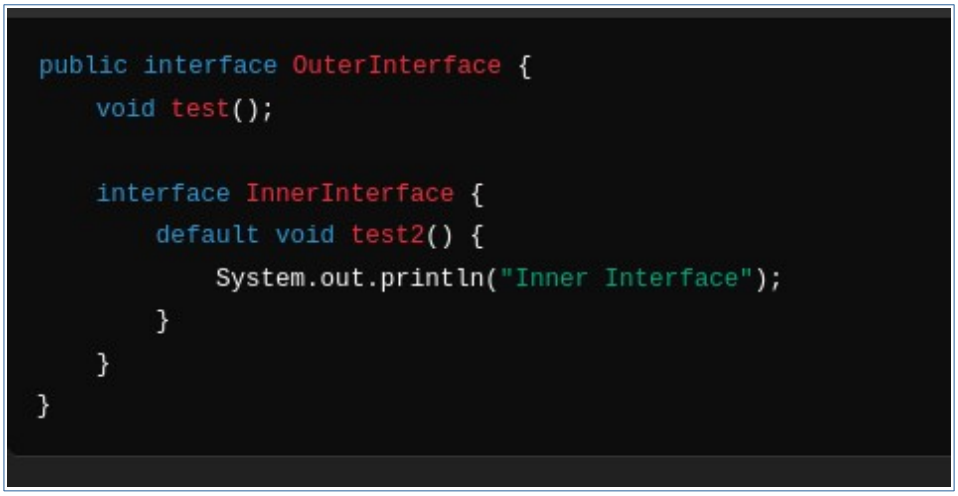
Ein Interface in Java ist eine besondere Form von Klassen, die hauptsächlich zur Deklaration von abstrakten Methoden dient. Es ähnelt abstrakten Klassen, hat aber einige wesentliche Unterschiede:

1. **Mehrfachvererbung mit Interfaces:**
    - Java erlaubt Mehrfachvererbung nur durch die Implementierung mehrerer Interfaces.
2. **Attribute in Interfaces:**
    - Alle Attribute in einem Interface sind **automatisch `public`, `static` und `final`**. Sie müssen also immer Konstanten sein.
3. **Methoden in Interfaces:**
    - Fast alle Methoden in einem Interface sind **automatisch `public` und `abstract`**.
    - Eine Ausnahme bilden **`default`-Methoden**, die in Unterklassen mit dem Schlüsselwort `public` implementiert werden können **(Müssen aber nicht).** Diese Methoden können überschrieben werden.
    - Interfaces können auch **statische Methoden** haben, die eine Implementierung enthalten. Diese Methoden sind ebenfalls `public`. aber sie **können nicht überschrieben** werden, weil sie nicht an Instanzen von Klassen gebunden sind, sondern zur **Schnittstelle selbst** gehören.
    - **Finale Methoden** sind in Interfaces nicht erlaubt, da Methoden in einem Interface abstrakt sind und überschrieben werden sollen.
    - **Private Methoden** sind erlaubt, aber nur innerhalb des Interfaces selbst zugänglich. Sie dienen dazu, Code zu strukturieren, der von mehreren `default`- oder `static`-Methoden wiederverwendet werden kann.
4. **Konstruktoren:**
    - Ein Interface kann **keinen Konstruktor** haben.
5. **Implementierung in der Unterklasse:**
    - Eine Klasse, die ein Interface implementiert, muss alle Methoden des Interfaces implementieren oder selbst als `abstract` deklariert werden.

Spezielle Interface-Typen

1. Nested Interface (Verschachtelte Interfaces):

- Ein Interface kann innerhalb eines anderen Interfaces deklariert werden.
- Beispiel:



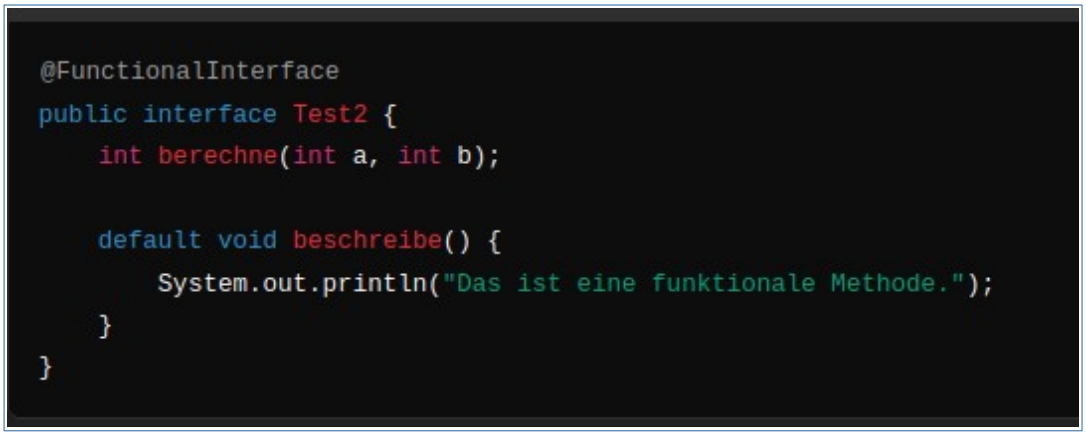
```java
public interface OuterInterface {
    void test();

    interface InnerInterface {
        default void test2() {
            System.out.println("Inner Interface");
        }
    }
}
```

- Um ein verschachteltes Interface zu implementieren, würde man das äußere Interface gefolgt vom inneren Interface angeben, z.B. `OuterInterface.InnerInterface`.

2. Functional Interface:

- Ein **funktionales Interface** ist ein Interface mit genau einer abstrakten Methode. Es wird mit der Annotation `@FunctionalInterface` markiert.
- Beispiel:



```java
@FunctionalInterface
public interface Test2 {
    int berechne(int a, int b);

    default void beschreibe() {
        System.out.println("Das ist eine funktionale Methode.");
    }
}
```

- Funktionale Interfaces können beliebig viele `default`, `static` und `private` Methoden haben, solange es genau **eine abstrakte Methode** gibt.

Marker oder Tagging Interface:

- Ein Marker-Interface (auch Tagging-Interface genannt) ist ein Interface, das **keine Methoden oder Konstanten** deklariert.
- Es dient dazu, eine Klasse "zu markieren", sodass sie bestimmte Fähigkeiten oder Eigenschaften hat (z.B. `Serializable`, `Cloneable`).

Wichtige Regeln:

- Eine Unterklasse **`extends`** eine Oberklasse (normal oder abstrakt).
- Eine normale oder finale Klasse kann ein Interface mit **`implements`** implementieren.
- Ein Interface kann ein anderes Interface mit **`extends`** erweitern.

Welche Java-Typen können ein Interface implementieren?

1. Klassen (normale oder abstrakte Klassen)
2. Abstrakte Klassen
3. Verschachtelte Klassen (Nested Classes)
4. Enums
5. Dynamische Proxys