Handout: Einführung in OOP mit Java, Interfaces und Adapterklassen

Einführung in die Objektorientierte Programmierung (OOP)

Objektorientierte Programmierung (OOP) ist ein Programmierparadigma, das auf der Struktur von Klassen und Objekten basiert. Es umfasst vier Hauptprinzipien:

1. **Abstraktion**: Vereinfachung komplexer Systeme durch die Reduzierung auf wesentliche Merkmale.
2. **Kapselung**: Verbergen der Implementierungsdetails eines Objekts.
3. **Vererbung**: Weitergabe von Eigenschaften und Methoden an Unterklassen.
4. **Polymorphismus**: Nutzung der gleichen Schnittstelle für unterschiedliche Datentypen.

Klassenprinzip und Vererbung in Java

In Java ist eine Klasse eine Blaupause für Objekte. Vererbung ermöglicht es, eine Klasse von einer anderen abzuleiten.

Beispiel:

```
// Basisklasse
class Fahrzeug {
    String typ;

    public Fahrzeug(String typ) {
        this.typ = typ;
    }

    public void fahren() {
        System.out.println(typ + " fährt.");
    }
}

// Unterklasse
class Auto extends Fahrzeug {
    public Auto() {
        super("Auto");
    }
}
```

Nachteile und Probleme der Vererbung

1. **Eingeschränkte Mehrfachvererbung**: Java erlaubt keine Mehrfachvererbung von Klassen.
2. **Enger Zusammenhang**: Änderungen in der Oberklasse können Unterklassen beeinträchtigen.
3. **Wiederverwendbarkeit**: Vererbung kann zu starren Abhängigkeiten führen.

Beispiel: Warum Mehrfachvererbung problematisch ist

```
// In Java nicht möglich:
class Elektroauto extends Auto, Elektrisch {
    // Compilerfehler: Mehrfachvererbung ist nicht erlaubt
}
```

Lösung: Interfaces in Java

Ein Interface ist ein Vertrag, der festlegt, welche Methoden eine Klasse implementieren muss. Es ist eine Alternative zur Mehrfachvererbung.

Vorteile von Interfaces

1. Ermöglicht Mehrfachimplementierung.
2. Entkoppelt die Implementierung vom Verhalten.
3. Fördert Flexibilität und Wiederverwendbarkeit.

Definition eines Interfaces:

```
interface Fahrbar {
    void starten();
    void stoppen();
}
```

Implementierung eines Interfaces:

```
class Benzinauto implements Fahrbar {
    @Override
    public void starten() {
        System.out.println("Benzinauto startet.");
    }

    @Override
    public void stoppen() {
        System.out.println("Benzinauto stoppt.");
    }
}

class Elektroauto implements Fahrbar {
    @Override
    public void starten() {
        System.out.println("Elektroauto startet.");
    }

    @Override
```

```
    public void stoppen() {
        System.out.println("Elektroauto stoppt.");
    }
}
```

Direkte Methodenimplementierung in Interfaces

Seit Java 8 können Methoden in Interfaces mit `default` implementiert werden.

```
interface Wartung {
    default void status() {
        System.out.println("Wartungsstatus: Alles in Ordnung.");
    }
}

class Sportwagen implements Wartung {
    // Kann die default-Methode optional überschreiben
}
```

Anwendung von Interfaces: Beispiel Auto und Elektroauto

Szenario:

Ein Auto kann verschiedene Arten von Fahrzeugen sein, wie z. B. Benzin- oder Elektroauto. Beide haben jedoch ein gemeinsames Verhalten: sie können fahren.

```
interface Fahrzeug {
    void fahren();
}

class Benzinauto implements Fahrzeug {
    @Override
    public void fahren() {
        System.out.println("Das Benzinauto fährt.");
    }
}

class Elektroauto implements Fahrzeug {
    @Override
    public void fahren() {
        System.out.println("Das Elektroauto fährt.");
    }
}

public class Main {
    public static void main(String[] args) {
        Fahrzeug auto1 = new Benzinauto();
```

```
        Fahrzeug auto2 = new Elektroauto();

        auto1.fahren();
        auto2.fahren();
    }
}
```

Zusammenfassung: Unterschiede zwischen Vererbung und Interfaces

Aspekt	**Vererbung**	**Interfaces**
Anzahl der Elternklassen	Eine	Mehrere
Modifikatoren	public, private, protected	Nur public
Verwendung	Implementierung gemeinsamer Logik	Definition eines gemeinsamen Verhaltens
Flexibilität	Gering (starke Kopplung)	Hoch (lose Kopplung)

Quizfragen zur Selbstkontrolle

1. **Was sind die Hauptprobleme der Vererbung in Java?**
2. **Warum bietet ein Interface eine Lösung für Mehrfachvererbungsprobleme?**
3. **Was ist der Unterschied zwischen einer abstrakten Klasse und einem Interface?**
4. **Können Interfaces private Methoden haben?**

Lösungen zum Quiz

1. Probleme der Vererbung:
    - Keine Mehrfachvererbung.
    - Änderungen in der Oberklasse können Unterklassen beeinflussen.
    - Kann zu starren Abhängigkeiten führen.
2. Interfaces lösen das Problem, da sie Mehrfachimplementierungen erlauben und nur das Verhalten spezifizieren, nicht die Implementierung.
3. Eine abstrakte Klasse kann sowohl abstrakte als auch konkrete Methoden enthalten, während ein Interface nur abstrakte Methoden (bis Java 7) oder default-Methoden (seit Java 8) erlaubt.
4. Seit Java 9 können Interfaces private Methoden enthalten.