

## Grundlagen der Informatik 4 (GIT IV) Übung

Bearbeitet von:

Name	Matrikelnummer	Nummer des Übungsblatts
Maximilian Bradstädter	1007486	7

### Aufgabe 1:

1. Die abstrakte Klasse Zahlungsart als Interface zu implementieren ist hier praktisch, da hier keine festen/nötigen Funktionsdefinitionen implementiert werden.
2. Für das Prinzip an sich ändert sich an sich nichts, da dies quasi nichts hier ändert. Jedoch muss man das „extends Zahlungsart“ zu „implements Zahlungsart“ ändern.

### Aufgabe 2:

Amphibienfahrzeug.java:

```
public class Amphibienfahrzeug implements ILandfahrzeug, IWasserfahrzeug {
```

```
    private final Landfahrzeug landModul = new Landfahrzeug();
```

```
    private final Wasserfahrzeug wasserModul = new Wasserfahrzeug();
```

```
@Override
```

```
public void stoppen() {
```

```
    System.out.println("Stoppen");
```

```
}
```

```
@Override
```

```
public void starten() {
```

```
    System.out.println("Starten");
```

```
}
```

```
@Override  
public void fahren() {  
    landModul.fahren();  
}  
  
{
```

```
@Override  
public void schwimmen() {  
    wasserModul.schwimmen();  
}  
}
```

IFahrzeug.java:

```
public interface IFahrzeug {  
    void starten();  
  
    void stoppen();  
}
```

ILandfahrzeug.java:

```
public interface ILandfahrzeug extends IFahrzeug {  
    void fahren();  
}
```

IWasserfahrzeug.java:

```
public interface IWasserfahrzeug extends IFahrzeug{  
    void schwimmen();  
}
```

Landfahrzeug.java:

```
public class Landfahrzeug implements ILandfahrzeug{  
    @Override  
    public void stoppen() {  
        System.out.println("Stoppen");  
    }  
  
    @Override  
    public void starten() {  
        System.out.println("Starten");  
    }  
  
    @Override  
    public void fahren() {  
        System.out.println("fahren");  
    }  
}
```

Wasserfahrzeug.java:

```
public class Wasserfahrzeug implements IWasserfahrzeug {  
    @Override  
    public void stoppen() {  
        System.out.println("Stoppen");  
    }  
  
    @Override  
    public void starten() {  
        System.out.println("Starten");  
    }  
  
    @Override  
    public void schwimmen() {  
        System.out.println("schwimmen");  
    }  
}
```

Aufgabe 3:

```
import java.util.Stack;  
  
public class StapelVerteiler {  
    public void verteileZahlen(  
        Stack<Integer> stapel1,
```

```
Stack<Integer> stapel2,
Stack<Integer> stapel3 {
    if (stapel1.isEmpty()) {
        return;
    }

    int tmp = stapel1.pop();

    if (tmp % 2 == 0) {
        stapel2.push(tmp);
    } else {
        stapel3.push(tmp);
    }
}

verteileZahlen(stapel1, stapel3, stapel3);
}

public static void main(String[] args) {
    Stack<Integer> stapel1 = new Stack<>();
    Stack<Integer> stapel2 = new Stack<>();
    Stack<Integer> stapel3 = new Stack<>();
    // Hier Beispielwerte in Stapel 1 hinzufügen
    // ...
    for (int i = 1; i <= 8; i++) {
        stapel1.push(i);
    }
}

StapelVerteiler verteiler = new StapelVerteiler();
```

```
verteiler.verteileZahlen(stapel1, stapel2, stapel3);  
System.out.println("Gerade Zahlen: " + stapel2);  
System.out.println("Ungerade Zahlen: " + stapel3);  
}  
  
}
```