# Musterlösung Übung 6

## Abstrakte Klassen und Interfaces in Java, Entwurfsmuster „Strategie“

Anhand der Figuren in einem Schachspiel soll das Entwurfsmuster „Strategie“ nachvollzogen werden.

Als Basisklasse soll die folgende verwendet werden:

```
public abstract class Figur {
 ZugMoeglichkeit zugMoeglichkeit;
 public void zieheFigur() { zugMoeglichkeit.ziehe(); }
}
```

1. Tippen Sie die Basisklasse `Figur.java` ab und schreiben Sie das Interface `ZugMoeglichkeit`, das die virtuelle Methode `ziehe()` bereitstellt. *Siehe **`Figur.java`** und **`ZugMoeglichkeit.java`**:*

   ```
   public abstract class Figur {
     ZugMoeglichkeit zugMoeglichkeit;
     public void zieheFigur() {
      // Kleine Erweiterung: Figur ausgeben
      System.out.print(this.toString() + ": ");
      zugMoeglichkeit.ziehe();
     }
   }

   public interface ZugMoeglichkeit {
    void ziehe();
   }
   ```

2. In einem Schachspiel gibt es die Figuren `Bauer`, `Springer`, `Laeufer`, `Turm`, `Dame` und `Koenig`, die als Subklasse von `Figur` implementiert werden sollen. Schreiben Sie jede dieser von `Figur` abgeleiteten Klasse in eine entsprechende JAVA-Datei. *Siehe **`Bauer.java`**, **`Springer.java`**,*

   ***`Laeufer.java`**,*

   ***`Turm.java`***

   *und **`Koenig.java`**.*

   ```
   public class Bauer extends Figur {
    public Bauer() {
     zugMoeglichkeit = new EinsVorZug();
    }
   }
   ```

```
public class Springer extends Figur {
 public Springer() {
  zugMoeglichkeit = new SpringerZug();
 }
}

public class Laeufer extends Figur {
 public Laeufer() {
  zugMoeglichkeit = new SchraegZug();
 }
}

public class Turm extends Figur {
 public Turm() {
  zugMoeglichkeit = new GeradeZug();
 }
}

public class Dame extends Figur {
 public Dame() {
  zugMoeglichkeit = new GeradeOderSchraegZug();
 }
}

public class Koenig extends Figur {
 public Koenig() {
  zugMoeglichkeit = new EinsGeradeOderSchraegZug();
 }
}
```

3. Als Zugmöglichkeit verwenden wir vorerst einfach eine Bildschirmausgabe (wir wollen es nicht übertreiben), also z.B. ein `System.out.println("zwei-vor-eins-zur-seite");` für die Figur `Springer`. Wenn Sie die Züge der Schachfiguren nicht kennen, nehmen Sie als Text einfach `"Springerzug"` oder ähnliches. Die zu einer Figur gehörenden Zugmöglichkeiten sollen in der Methode `ziehe()` berücksichtigt werden (ähnlich, wie wir es in der Vorlesung am Beispiel von Bewegungsmöglichkeiten der Tierarten durchgespielt hatten).

***EinsGeradeOderSchraegZug.java***

```
public class EinsGeradeOderSchraegZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("eins-vor-oder-eins-diagonal");
 }
}
```

***EinsGeradeZug.java***

```
public class EinsGeradeZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("eins-geradeaus");
 }
}
```

***EinsSchraegZug.java***

```
public class EinsSchraegZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("eins-diagonal");
 }
}
```

***EinsVorZug.java***

```
public class EinsVorZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("eins-vorwärts");
 }
}
```

***GeradeOderSchraegZug.java***

```
public class GeradeOderSchraegZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("geradeaus-oder-diagonal");
 }
}
```

***GeradeZug.java***

```
public class GeradeZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("geradeaus");
 }
}
```

***SchraegZug.java***

```
public class SchraegZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("diagonal");
 }
}
```

***SpringerZug.java***

```
public class SpringerZug implements ZugMoeglichkeit {
 public void ziehe() {
  System.out.println("zwei-geradeaus-eins-zur-seite");
 }
}
```

4. Welche Vor- und Nachteile hat dieses Entwurfsmuster bezüglich des Problems „Beschreibung von Schachfiguren“? (Überlegen Sie, was getan werden müsste, wenn eine neue Figur mit anderen Zugmöglichkeiten hinzukommt).

   **Vorteile:**

   - *Das Einfügen einer neuen Figur mit anderer Zugmöglichkeit ist sehr einfach, da bestehende Klassen unverändert bleiben können, und lediglich eine Klasse für die Zugart (die das Interface* ***`ZugMoeglichkeit`*** *benutzt) und eine Klasse für die neue Figur selbst (als* ***`extends Figur`****) hinzukommt.*

- *Die Implementierung eines tatsächlichen Zuges kann mit Interface-Methoden geschehen, die separat vom Rest des Programmes in den* ***ZugMoeglichkeit****en eingebaut werden.*
- *Durch die Trennung von Spielfigur und Zugregeln (die durchaus auch kombiniert und auf mehrere Figuren angewandt werden könnten) ist der Aufbau des Programmes sehr übersichtlich.*
- *Besonders praktisch: Wenn ein Bauer die Grundline auf der Gegnerseite erreicht, darf er sich in eine andere Figut „verwandeln", dies kann durch Neuzuweisung der* ***zugMoeglichkeit*** *elegant gelöst werden.*

**Nachteile:**

- *Abhängigkeiten zwischen den Figuren und Zugmöglichkeiten sind alleine mit dem Strategie-Entwurfsmuster schwer zu realisieren, z.B. die Tatsache, dass eine Figur u.U. nicht ziehen darf, wenn der König im Schach steht.*
- *Die Positionen der Figuren auf dem Brett müssten noch in Form von Klassenattributen vermerkt werden.*
- *Einschränkungen der Zugmöglichkeiten (z.B. Doppelzug des Bauern nur in der Anfangsposition möglich, Rochade nur wenn König und Turm noch nicht gezohen haben und König nicht über Schach zieht) sind aufgrund fehlender Abhängigkeiten schwer zu realisieren. Hier müssten weitere Interfaces definiert, oder ein anderes Muster gewählt werden.*
- *Event-Steuerung, also z.B. Feststellen, dass das Spiel zu Ende ist, oder dass bestimmte Sonderregeln angewendet werden müssten, ist in diesem Entwurfsmuster nicht vorgesehen. Das Spiel läuft quasi endlos, und es werden keine Regeln außer den unveränderlichen Zugregeln angewandt.*

5. Wenn Sie möchten, implementieren Sie ein Testprogramm, das ein paar Figuren ziehen lässt. In einem Schachspiel gibt es übrigens von einigen Figuren mehrere, so dass man vom code reuse z.B. beim `weißerLaeufer` und `schwarzerLaeufer` profitieren könnte.

   ***Main.java*** *soll ein kurzes Spiel mit zwei Spielern simulieren. Tatsächliche Züge mit Anfangs- und Endposition sind in dem einfachen Modell zwar nicht möglich, aber die Zugreihenfolge ist angegeben, und alle am Spiel beteiligten Figuren-Objekte werden zu Beginn erzeugt.*

```
public class Main {

 public static void weiss() { // Weiß ist am Zug
  System.out.print("Weiß: ");
 }

 public static void schwarz() { // Schwarz ist am Zug
  System.out.print("Schwarz: ");
 }

 public static void main(String[] args) {
  // Eine Figur pro Spieler, einige davon gibt's mehrfach.
  Figur koenig[] = new Koenig[2], dame[] = new Dame[2], turm[][] = new Turm[2][2],
laeufer[][] = new Laeufer[2][2], springer[][] = new Springer[2][2], bauer[][] = new Bauer[2][8];
```

```
  // Das Schachbrett aufbauen
  for(int spieler=0; spieler<2; spieler++) koenig[spieler] = new Koenig();
  for(int spieler=0; spieler<2; spieler++) dame[spieler]   = new Dame();
  for(int spieler=0; spieler<2; spieler++) for(int i=0; i<2; i++) turm[spieler][i] = new Turm();
  for(int spieler=0; spieler<2; spieler++) for(int i=0; i<2; i++) laeufer[spieler][i] = new Laeufer();
  for(int spieler=0; spieler<2; spieler++) for(int i=0; i<2; i++) springer[spieler][i] = new Springer();
  for(int spieler=0; spieler<2; spieler++) for(int i=0; i<8; i++) bauer[spieler][i] = new Bauer();

  // Ein Spiel

  System.out.println("Spielbeginn.");

  weiss();   bauer[0][4].zieheFigur(); bauer[0][4].zieheFigur(); // Weiß zieht den 5. Bauer auf e4
  schwarz(); bauer[1][4].zieheFigur(); bauer[1][4].zieheFigur(); // Schwarz zieht den 5. Bauer auf e7

  weiss();   laeufer[0][1].zieheFigur(); // Weiß zieht den 2. Läufer auf c4
  schwarz(); bauer[1][3].zieheFigur();   // Schwarz zieht den 4. Bauer auf d6

  weiss();   dame[0].zieheFigur();        // Weiß zieht die Dame auf f3
  schwarz(); springer[1][0].zieheFigur(); // Schwarz zieht den 1. Springer auf c6

  weiss();   dame[0].zieheFigur(); // Weiß zieht die Dame auf f7 und gibt Schach

  System.out.println("Schwarz ist schachmatt!");
 }
}
```

6. Stellen Sie die programmierten Klassen und Interfaces als Klassendiagramm dar.

   *Nach Importieren und etwas Umsortieren im Darstellungsfenster von* ***argouml*** *ergibt sich folgendes Bild:*