Interfaces und abstrakte Klassen

Das Thema dieses Wiki-Artikels sind Interfaces und abstrakte Klassen. Das ganze basiert auf meinem Posting im Forum: implements richtig nutzen
Vorab: dieser Beitrag enthält ein bisschen Theorie aber ich gebe mir Mühe das ganze so einfach wie möglich zu schreiben ;).

== Interfaces und Objektorientierung ==
Also erstmal, wozu braucht man das Sprachmittel des Interfaces?

Damit man verstehen kann wozu Interfaces da sind, sollte man zumindest ansatzweise Objekt Orientierung verstanden haben, denn Interfaces sind die Antwort auf Probleme die man ohne Objekt Orientierung gar nicht hätte. Also nicht verzweifeln wenn ihr den Artikel nicht ganz versteht. Einfach nochmal einen Gang zurück und erstmal verstehen was Klassen und Objekte sind.
Für alle anderen geht es hier weiter.

== Beispiel ==
Ich bin der Meinung, dass man am besten an einem Beispiel lernt. Unser Beispiel wird ein kleiner Bauernhof sein, auf dem es Tiere und einen Bauern gibt.

Es gibt immer einen der eine Klasse benutzen will, in unserem Beispiel wird das Tier vom Bauern "verwendet". Alle Tier können Sachen machen. Dazu bieten sie der Außenwelt Methoden an, die jedes Tier kann. Unsere Tiere können Geräusche machen, weinen und haben ein Lieblingsessen. Aber jedes konkrete Tier (Hund, Katze oder Huhn) macht unterschiedliche Geräusche, weint anders und will etwas anderes zum essen haben. In diesem Zusammenhang spricht man auch von konkreten Implementierungen. Aber im Moment interessiert uns nur die abstrakte Definition, sprich die Sammlung der Methoden die jedes Tier kann. Dazu schreiben wir das Interface Tier:

== Interface Code ==
Quellcode

```
1. package test;
2. public interface Tier
3. {
4. public void machGeraeusch();
5. public String getLieblingsEssen();
6. public void weinen();
7. }
```

Damit haben wir jetzt definiert dass jedes Tier Geräusche macht, ein Lieblingsessen hat und weinen kann. Die Methoden enthalten keine Implementierung, sondern nur die Methodensignaturen.

== Steuerung ==
Der nächste Schritt ist, dass wir einen Bauern definieren. Der Bauer darf ruhig eine Klasse sein, denn die Methoden die wir dort definieren macht jeder Bauer gleich, egal ob es ein bayrischer oder ein niedersächsischer Bauer ist. Dieser Bauer ruft in seinen Methoden, Methoden vom Tier auf:

Quellcode

```
1. package test;
2. public class Bauer
3. {
4. public void steigAufSchwanz(Tier t)
5. {
6. t.weinen();
7. }
```

```
10. public void fuettere(Tier t)
11. {
12. System.out.println("Fülle Napf mit "+t.getLieblingsEssen();
13. }
14. public void kommeInNaehe(Tier t)
15. {
16. t.machGeraeusch();
17. }
18. }
```

Alles anzeigen

So an dieser Stelle sollte klar geworden sein, wozu man Interfaces braucht. Hätte man keine Interfaces müsste der Bauer für jedes Tier eine eigene Methode haben, also müsste es Beispielsweise eine Methode fuettere(Katze k), fuettere(Hund h), fuettere(Huhn h) geben.

== Implementierung ==
Als nächstes machen wir uns an die konkrete Implementierung der Tiere, sprich wir definieren die Methoden die das Interface Tier vorgibt für Hund und Katze.

Quellcode

```
1. package test;
2. public class Hund
3. implements Tier
4. {
5. //Methoden von Tier
6. public void machGeraeusch()
7. {
8. System.out.println("wau");
9. }
10. public String getLieblingsEssen()
11. {
12. return "Fleisch";
13. }
14. public void weinen()
15. {
16. System.out.println("*jaul*");
17. }
18. }
```

Alles anzeigen

Quellcode

```
1. package test;
2. public class Katze implements Tier
3. {
4. //Methoden von Tier
5. public void machGeraeusch()
6. {
7. System.out.println("miau");
8. }
9. public String getLieblingsEssen()
10. {
11. return "Mäuse";
```

```
14.  }
15.  public void weinen()
16.  {
17.  System.out.println("*fauch*");
18.  }
19.  }
20.
```

Alles anzeigen

Jetzt haben wir definiert, was ein Hund und eine Katze für Geräusche macht, wie sie weinen und welches Lieblingsessen sie haben.

Aber um das Thema abzurunden führe ich noch schnell die abstrakten Klassen ein. Dazu greif ich als Beispiel das Huhn auf. Ein Huhn ist ein Tier, sprich es implementiert die Methoden getLieblingsEssen und weinen. Das weibliche Huhn, die Henne, macht aber andere Geräusche wie das männliche, der Hahn.

Ein naiver Ansatz wäre nun eine Klasse Henne und eine Klasse Hahn die Tier implementieren zu schreiben. Besser ist es eine Klasse Huhn zu definieren, die die beiden Methoden weinen und getLieblingsEssen implementiert. Henne und Hahn erben dann von dieser und implementieren noch die Methode machGeraeusch. Dazu hat man in Java die Möglichkeit die Klasse Huhn als abstract zu definieren. Das heißt, dass diese Klasse noch nicht vollständig ist, und somit auch keine Objekte von der Klasse erzeugt werden dürfen.

Quellcode

```
1.  package test;
2.  public abstract class Huhn implements Tier
3.  {
4.  //Implementierte Methoden da dies bei Henne und Hahn gleich bleibt
5.  public String getLieblingsEssen()
6.  {
7.  return "Körner";
8.  }
9.  public void weinen()
10. {
11. System.out.println("*kreisch*");
12. }
13. //Offen gelassene Methode, muss nicht hingeschrieben werden
14. public abstract void geraeusch();
15. }
```

Alles anzeigen

Als letzten Schritt definieren wir noch Henne und Hahn. In denen legen wir fest, was welches Geräusch die Henne bzw. der Hahn macht:

Quellcode

```
1. package test;
2. public class Henne extends Huhn
3. {
4. public void geraeusch()
5. {
6. System.out.println("guckguck");
7. }
8. }
```

Quellcode

```
1. package test;
2. public class Hahn extends Huhn
3. {
4. public void geraeusch()
5. {
6. System.out.println("kickerikie");
7. }
8. }
```

Als letztes folgt noch der Test des ganzen:

Quellcode

```
1. package test;
2. public class Test
3. {
4. public static void main(String[] args)
5. {
6. Bauer bauer = new Bauer();
7. Tier[] tiere = new Tier[4];
8. tiere[0] = new Hund();
9. tiere[1] = new Henne();
10. tiere[2] = new Katze();
11. tiere[3] = new Hahn();
12. for (Tier tier : tiere)
13. {
14. bauer.steigAufSchwanz(tier);
15. bauer.fuetter(tier);
16. bauer.kommeInNaehe(tier);
17. }
18. }
19. }
```

Alles anzeigen

== Fazit ==
Ich hoffe euch ist einigermaßen klar geworden, wozu Interfaces und abstrakte Klassen da sind.
Zusammenfassend sind Interfaces dazu da, wenn man weiß dass mehrere Klassen die selben Methoden anbieten sollen, diese aber ein unterschiedliches Verhalten haben sollen.
Wenn Klassen teilweise das selbe Verhalten haben aber einzelne Methoden unterschiedlich sind, kann man das selbe Verhalten in einer abstrakten Klasse beschreiben.