Vererbung

Vererbung (engl. *Inheritance*) gehört zu den grundlegenden Konzepten der objektorientierten Programmierung. Dieses Konzept basiert auf Beobachteungen aus der realen Welt:

* Dinge (Objekte) kommen in verschiedenen Varianten vor, die sich hierarchisch klassifizieren lassen
* Dinge (Objekte), die hierarchisch tiefer stehen, sind speziellere Varianten der übergeordneten, generelleren Dinge
* Speziellere Dinge besitzen die Eigenschaften der generelleren Dinge plus weitere, spezifischere Eigenschaften

Häufig werden für solche Beispiele aus der Realen Welt die Klassifikationen von Tieren oder Pflanzen verwendet, z.B. Säugetiere sind eine Spezialisierung der Tiere, Katzen und Hunde wiederum sind Spezialisierungen von Säugetieren usw. Eine solche Hierarchie, wie z.B. in der folgenden Abbildung gezeigt, kommt jedenfalls in der realen Welt sehr häufig vor.

Tiere sind dabei die allgemeinste *Klasse*. Sie haben Eigenschaften, die für alle Tiere zutreffen, z.B. dass sie sich bewegen können und fortpflanzen. Wirbeltiere haben alle Eigenschaften der Tiere, aber zusätzlich noch *speziellerere* Eigenschaften, wie z.B. dass sie ein Skellett besitzen. Säugetiere besitzen alle Eigenschaften von Wirbeltieren (und also auch von Tieren) und darüber hinaus speziellere Eigenschaften, nämlich z.B. lebend gebärend usw.

Für die **Programmierung** bedeutet das, dass Klassen von anderen Klassen die Eigenschaften **erben** können, d.h.

* *Kindklassen* übernehmen (erben) Eigenschaften (Objektvaribalen und Objektmethoden) der *Elternklasse*
* *Kindklassen* können zusätzlich weitere Eigenschaften (Objektvaribalen und Objektmethoden) enthalten

Wichtig für das Verständnis der Vererbung ist, dass zwischen den Kind- und der Elternklasse eine **is-a**- (**ist-ein**-) Relation besteht. Für unser Beispiel oben bedeutet das z.B. der Hund **ist ein** Säugetier, das Säugetier **ist ein** Wirbeltier usw. Wenn wir uns also eine Vererbungshierarchie aufmalen, dann gehen die Pfleile immer von der Kind- zur Elternklasse:

Für die Vererbung gilt also:

* Die Vererbung ist eine sehr bedeutende Möglichkeit der Wiederverwendbarkeit von Klassen
* Vererbung basiert auf der Idee, dass *Elternklassen* ihren *Kindern* Variablen und Methoden vererben
* Durch Vererbung entsteht eine Klassenhierarchie:

Vererbung in Java

Angenommen, wir haben eine Klasse Elternklasse und eine Kindklasse soll von dieser Klasse erben. Wir verwenden das Schlüsselwort extends, um die Kindklasse von der Elternklasse erben zu lassen:

public class Kindklasse extends Elternklasse { }

In Java kann eine Klasse nur von **genau einer** Klasse erben. *Mehrfachvererbung* (das Erben von mehreren Klassen) ist **nicht** möglich.

Ein erstes Beispiel

Wir betrachten ein erstes Beispiel. Wir werden eine Klasse Viereck erstellen. Von dieser Klasse wird eine Klasse Rechteck **abgeleitet**, d.h. Rechteck **erbt** von Viereck. Danach erzeugen wir noch eine weitere Klasse Quadrat, die wiederum von Rechteck erbt.

Die Klasse Viereck

Als Objektvaribalen der Klasse Viereck wählen wir die vier Seitenlängen eines Vierecks a, b, c und d. Außerdem definieren wir uns noch einen Konstruktor und zwei Objektmethoden, die Methode umfang(), die den Umfang des Rechtecks zurückgibt und die Methode print(), die die vier Seitenlängen auf die Konsole ausgibt und den Umfang:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public class Viereck  {  private int a,b,c,d; // Seiten  public Viereck(int a, int b, int c, int d)  {  this.a = a;  this.b = b;  this.c = c;  this.d = d;  }  public int umfang()  {  return this.a + this.b + this.c + this.d;  }  public void print()  {  System.out.print("[ a=" + this.a + ", b=" + this.b  + ", c=" + this.c + ", d=" + this.d + " ] ");  System.out.println(" Umfang des Vierecks : " + this.umfang());  }  } |

In einer TestViereck-Klasse erzeugen wir Objekte von Viereck und testen die Methoden:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class TestViereck  {  public static void main(String[] args)  {  Viereck v1 = new Viereck(10,20,30,40);  v1.print();  Viereck v2 = new Viereck(15,20,25,20);  v2.print();  }  } |

Auf der Konsole erscheint folgende Ausgabe:

[ a=10, b=20, c=30, d=40 ] Umfang des Vierecks : 100

[ a=15, b=20, c=25, d=20 ] Umfang des Vierecks : 80

Soweit nichts Neues. Jetzt wollen wir aber von dieser Klasse erben und erzeugen eine Klasse Rechteck. Ein rechteck **ist ein** Viereck.

Die Klasse Rechteck

Die Klasse Rechteck erbt von Viereck, d.h. wir verwenden das Schlüsselwort extends. Wir erzeugen uns im gleichen package, in dem auch Viereck und TestViereck liegen, eine Klasse Rechteck:

|  |  |
| --- | --- |
| 1  2  3  4 | public class Rechteck extends Viereck  {  } |

Hinter den Klassennamen Rechteck schreiben wir nun extends und den Namen der Klasse, von der geerbt werden soll, hier Viereck.

Das ist soweit gut, jedoch wird leider ein Fehler angezeigt:

Implicit super constructor Viereck() is undefined for default constructor. Must define an explicit constructor

**Der Konstruktor von Rechteck - das Schlüsselwort super**

Wir [erinnern uns](http://freiheit.f4.htw-berlin.de/prog1/objekte/#objekte-erzeugen-der-konstruktor):

1. Wenn wir eine neue Klasse erstellen und **keinen** Konstruktor definieren, dann existiert immer ein sogenannter *impliziter* Konstruktor (oder *Standardkonstruktor*). Dieser heißt exakt wie die Klasse und ist parameterlos.
2. Wenn wir uns einen [eigenen Konstruktor definieren](http://freiheit.f4.htw-berlin.de/prog1/objekte/#ein-eigener-konstruktor), dann existiert dieser *implizite* Konstruktor (*Standardkonstruktor*) nicht mehr.

In der Klasse Viereck haben wir uns einen eigenen Konstruktor erstellt (Viereck(int a, int b, int c, int d)), d.h. es gibt keinen *impliziten* Konstruktor Vierck() (mehr). In Java gilt nun aber folgendes:

Wird in Java ein Objekt einer Kindklasse erzeugt, wird auch **immer** ein Objekt der Elternklasse erzeugt.

Wir müssen nun dafür sorgen, dass beim Erzeugen eines Objektes von Rechteck auch ein Objekt von Viereck erzeugt werden kann. Dies machen wir wie folgt:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class Rechteck extends Viereck  {  public Rechteck(int laenge, int breite)  {  super(laenge, breite, laenge, breite); // Aufruf des Konstruktors von Viereck  }  } |

Erläuterung:

* in den Zeilen 3-6 wird der Konstruktor von Rechteck definiert. bei einem Rechteck sind die gegenüberliegenden Seiten gleich lang, deshalb benötigen wir für die Seitenlängen auch nur noch zwei Werte, nämlich laenge und breite.
* in Zeile 5 wird der **Konstruktor von Viereck** aufgerufen! Das passiert in der Kindklasse nicht durch Viereck(int, int, int, int), sondern durch die Verwendung des Schlüsselwortes super. Hier wird also explizit der Konstruktor der Elternklasse aufgerufen! Wenn wir uns den Konstruktor von Viereck nochmal anschauen, dann sehen wir, dass nun der Wert von laenge der Seite a zugeordnet wird, der Wert von breite der Seite b, der Wert von laenge der Seite c und der Wert von breite der Seite d.

Wenn wir in einer *Kindklasse* einen Konstruktor definieren, sollten wir immer explizit den Konstruktor der Elternklasse aufrufen (mithilfe von super()). Dieser Aufruf muss die erste Anweisung innerhalb des Konstruktors sein!

Der Umgang mit den Konstruktoren ist schon das Komplizierteste in Bezug auf Vererbung. Wir kommen später nochmal darauf zurück.

**Rechteck hat alle Eigenschaften von Viereck geerbt**

Unsere Klasse Rechteck ist nun anwendbar. Die Klasse rechteck hat alle Eigenschaften der Klasse Viereck geerbt, d.h.

* die Objektvariablen a, b, c und d sowie
* die Objektmethoden umfang() und print().

Das testen wir gleich in der TestViereck-Klasse und erzeugen uns Objekte von Rechteck:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | public class TestViereck  {  public static void main(String[] args)  {  System.out.printf("%n%n------- Objekte von Viereck erzeugen ------------%n%n");  Viereck v1 = new Viereck(10,20,30,40);  v1.print();  Viereck v2 = new Viereck(15,20,25,20);  v2.print();  System.out.printf("%n%n------- Objekte von Rechteck erzeugen ------------%n%n");  Rechteck r1 = new Rechteck(10, 20);  r1.print();  Rechteck r2 = new Rechteck(20, 30);  r2.print();  }  } |

Auf der Konsole erscheinen folgende Ausgaben:

------- Objekte von Viereck erzeugen ------------

[ a=10, b=20, c=30, d=40 ] Umfang des Vierecks : 100

[ a=15, b=20, c=25, d=20 ] Umfang des Vierecks : 80

------- Objekte von Rechteck erzeugen ------------

[ a=10, b=20, c=10, d=20 ] Umfang des Vierecks : 60

[ a=20, b=30, c=20, d=30 ] Umfang des Vierecks : 100

Wir erzeugen uns also zwei Objekte von Rechteck und rufen für beide Objekte die Objektmethode print() auf. Diese Methode hat Rechteck von Viereck geerbt. Diese Methode gibt nun korrekt die jeweiligen Seitenlängen aus (laenge wird a und c zugewiesen und breite den Seiten b und d - siehe Aufruf des Konstruktors von Viereck im Konstruktor von Rechteck: super(laenge, breite, laenge, breite)).

In der Methode print() wird die Methode umfang() aufgerufen, die ebenfalls geerbt wurde. Wir könnten diese Methode zum Testen auch direkt in der Testklasse für die Rechteck-Objekte aufrufen, z.B.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | System.out.printf("%n%n------- Objekte von Rechteck erzeugen ------------%n%n");  Rechteck r1 = new Rechteck(10, 20);  r1.print();  System.out.println("Umfang des Rechtecks : " + r1.umfang());  Rechteck r2 = new Rechteck(20, 30);  r2.print();  System.out.println("Umfang des Rechtecks : " + r2.umfang()); |

und erhalten dann die Ausgaben:

------- Objekte von Rechteck erzeugen ------------

[ a=10, b=20, c=10, d=20 ] Umfang des Vierecks : 60

Umfang des Rechtecks : 60

[ a=20, b=30, c=20, d=30 ] Umfang des Vierecks : 100

Umfang des Rechtecks : 100

Wichtig ist, dass Rechteck alle Objektvariablen und Objektmethoden der Klasse Viereck geerbt hat. Allerdings sind die Objektvariablen a, b, c und d in Viereck als private deklariert und deshalb kann nur in Viereck auf diese Objektvariablen zugegriffen werden. Wenn wir auch in Rechteck darauf zugreifen wollen, dann müssen wir die Sichtbarkeit der Objektvariablen ändern. Das wollen wir im nächsten Schritt machen.

**Der Sichtbarkeitsmodifizierer protected**

Wir kennen bisher zwei *Sichtbarkeitsmodifizierer* (auch *Zugriffsmodifizierer*): private und public.

* auf Objektvariablen und -methoden, die als private deklariert wurden, kann nur innerhalb der Klasse zugegriffen werden,
* auf Objektvariablen und -methoden, die als public deklariert wurden, kann in allen anderen Klassen (auch Klassen aus anderen Paketen) zugegriffen werden.

Wir lernen jetzt einen weiteren Sichtbarkeitsmodifizierer kennen: protected.

Auf Objektvariablen und -methoden, die als protected deklariert wurden, kann in den Klassen einer Vererbungshierarchie zugegriffen werden.

Bis jetzt können wir in der Klasse Rechteck nicht auf die Objektvariablen a, b, c und d zugreifen, da diese in Viereck als private deklariert wurden und deshalb nur innerhalb von Viereck zugreifbar sind. Aus Gründen der [Datenkapselung](http://freiheit.f4.htw-berlin.de/prog1/objekte/#datenkapselung-information-hiding-das-schlusselwort-private) sollen diese aber auch nicht public sein. Wir deklarieren sie deshalb als protected:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public class Viereck  {  protected int a,b,c,d; // Seiten innerhalb der Vererbungshierarchie sichtbar  public Viereck(int a, int b, int c, int d)  {  this.a = a;  this.b = b;  this.c = c;  this.d = d;  }  public int umfang()  {  return this.a + this.b + this.c + this.d;  }  public void print()  {  System.out.print("[ a=" + this.a + ", b=" + this.b  + ", c=" + this.c + ", d=" + this.d + " ] ");  System.out.println(" Umfang des Vierecks : " + this.umfang());  }  } |

Nun kann auf die Objektvariablen in allen abgeleiteten Klassen (also in Rechteck) zugegriffen werden. Nicht aber in anderen Klassen. Man muss also von Viereck erben, um Zugriff auf die als protected deklarierten Variablen zu bekommen.

**Erweitern der Klasse Rechteck um eine weitere Eigenschaft**

Rechteck hat alle Eigenschaften von Viereck geerbt. Ganz am Anfang des Vererbungskapitels haben wir aber gesagt, dass speziellere Klassen auch speziellere Eigenschaften haben können. Wir wollen Rechteck nun eine weitere Eigenschaft hinzufügen: die Objektmethode flaecheninhalt():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public class Rechteck extends Viereck  {  public Rechteck(int laenge, int breite)  {  super(laenge, breite, laenge, breite); // Aufruf des Konstruktors von Viereck  }  /\*  \* neue Objektmethode  \* spezielle Eigenschaft fuer Rechteck  \* gilt nicht für Viereck  \*  \*/  public int flaecheninhalt()  {  return this.a \* this.b; // Zugriff moeglich wegen protected in Viereck  }  } |

Die Klasse Rechteck hat somit eine weitere Eigenschaft. Diese ist nicht von Viereck geerbt, sondern ist eine spezielle Eigenschaft von Rechteck. Die Klasse Viereck besitzt diese Eigenschaft nicht! Das bedeutet, dass für Objekte der Klasse Viereck existiert die Eigenschaft flaecheninhalt() nicht, für Objekte der Klasse Rechteck aber schon. In der TEstklasse können wir die neue Methode testen:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | public static void main(String[] args)  {  System.out.printf("%n%n------- Objekte von Viereck erzeugen ------------%n%n");  Viereck v1 = new Viereck(10,20,30,40);  v1.print();  Viereck v2 = new Viereck(15,20,25,20);  v2.print();  System.out.printf("%n%n------- Objekte von Rechteck erzeugen ------------%n%n");  Rechteck r1 = new Rechteck(10, 20);  r1.print();  System.out.println("Umfang des Rechtecks : " + r1.umfang());  System.out.println("Flaecheninhalt des Rechtecks " + r1.flaecheninhalt());  Rechteck r2 = new Rechteck(20, 30);  r2.print();  System.out.println("Umfang des Rechtecks : " + r2.umfang());  System.out.println("Flaecheninhalt des Rechtecks " + r2.flaecheninhalt());  } |

und erhalten dann die Ausgaben (hier nur für Rechteck gezeigt):

------- Objekte von Rechteck erzeugen ------------

[ a=10, b=20, c=10, d=20 ] Umfang des Vierecks : 60

Umfang des Rechtecks : 60

Flaecheninhalt des Rechtecks 200

[ a=20, b=30, c=20, d=30 ] Umfang des Vierecks : 100

Umfang des Rechtecks : 100

Flaecheninhalt des Rechtecks 600

Beachten Sie, es ist **nicht** möglich, die Objektmethode flaecheninhalt() für die Objekte von Viereck aufzurufen! Für diese Objekte existiert die Eigenschaft **nicht!**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public static void main(String[] args)  {  System.out.printf("%n%n------- Objekte von Viereck erzeugen ------------%n%n");  Viereck v1 = new Viereck(10,20,30,40);  v1.print();  // System.out.println("Flaecheninhalt des Vierecks " + v1.flaecheninhalt()); // existiert nicht!!!  Viereck v2 = new Viereck(15,20,25,20);  v2.print();  // System.out.println("Flaecheninhalt des Vierecks " + v1.flaecheninhalt()); // existiert nicht!!!  } |

**Überschreiben von Methoden**

Die Klasse Rechteck hat unter anderem die Objektmethode print() von der Klasse Viereck geerbt. Wir können geerbte Methoden entweder so lassen, wie wir sie geerbt haben oder wir implementieren sie neu. Das neuimplementieren von geerbten Methoden nennt sich **Überschreiben**.

Wir eine Methode von der Elternklasse geerbt, diese Methode in der Kindklasse jedoch neu implemntiert, so wird diese Methode **überschrieben**.

Die geerbte print()-Methode gefällt uns nicht wirklich gut, denn

1. erfolgt innerhalb der Methode die Ausschrift Umfang des Vierecks anstelle von Umfang des Rechtecks und
2. könnten wir die print()-Methode in Rechteck um die Ausgabe des Flächeninhaltes des Rechteckes erweitern.

Wir wollen deshalb diese Methode **überschreiben**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public class Rechteck extends Viereck  {  public Rechteck(int laenge, int breite)  {  super(laenge, breite, laenge, breite); // Aufruf des Konstruktors von Viereck  }  /\*  \* neue Objektmethode  \* spezielle Eigenschaft fuer Rechteck  \* gilt nicht für Viereck  \*  \*/  public int flaecheninhalt()  {  return this.a \* this.b; // Zugriff moeglich wegen protected in Viereck  }  @Override  public void print()  {  System.out.print("[ a=" + this.a + ", b=" + this.b  + ", c=" + this.c + ", d=" + this.d + " ] ");  System.out.print(" Umfang des Rechtecks : " + this.umfang());  System.out.println(" Flaecheninhalt des Rechtecks : " + this.flaecheninhalt());  }  } |

In den Zeilen 20-26 haben wir nun die Methode print() neu implementiert, d.h. wir haben sie überschrieben. Beachten Sie die sogenannte *Annotation* @Override, die direkt über dem Methodenkopf steht. Mit dieser *Annotation* geben wir dem Compiler an, dass die folgende Methode **überschrieben** wird. Der Compiler prüft nun, ob wir die Methode überhaupt so geerbt haben, wie wir sie überschreiben, d.h. es wird überprüft, ob

1. der Name der Methode korrekt ist (wir müssen eine Methode geerbt haben, die genau so heißt - auch hier Groß- und Kleinschreibung beachten),
2. die Anzahl und die Typreihenfolge der Parameter mit der geerbten Methode übereinstimmen (die Namen der Parameter sind egal),
3. der Rückgabetyp der Methode mit der geerbten Methode übereinstimmt (der kann innerhalb der Vererbungshierarchie geändert werden, aber das ist ein späteres Thema),
4. die Sichtbarkeit nicht eingeschränkt wird (eine Methode, die in der Elternklasse als public deklariert wurde, darf beim Überschreiben nicht protected oder private werden).

Wir müssen in der Testklasse nun gar nichts ändern, es wird für die Rechteck-Objekte die neue Implementierung der print()-Methode verwendet:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public static void main(String[] args)  {  System.out.printf("%n%n------- Objekte von Viereck erzeugen ------------%n%n");  Viereck v1 = new Viereck(10,20,30,40);  v1.print();  Viereck v2 = new Viereck(15,20,25,20);  v2.print();  System.out.printf("%n%n------- Objekte von Rechteck erzeugen ------------%n%n");  Rechteck r1 = new Rechteck(10, 20);  r1.print();  Rechteck r2 = new Rechteck(20, 30);  r2.print();  } |

erzeugt folgende Ausgaben:

------- Objekte von Viereck erzeugen ------------

[ a=10, b=20, c=30, d=40 ] Umfang des Vierecks : 100

[ a=15, b=20, c=25, d=20 ] Umfang des Vierecks : 80

------- Objekte von Rechteck erzeugen ------------

[ a=10, b=20, c=10, d=20 ] Umfang des Rechtecks : 60 Flaecheninhalt des Rechtecks : 200

[ a=20, b=30, c=20, d=30 ] Umfang des Rechtecks : 100 Flaecheninhalt des Rechtecks : 600

Wir werden auf das Überschreiben von Methoden noch weiter üben, wenn wir uns mit der Klasse Object beschäftigen. Wir merken uns zunächst, dass wir geerbte Methoden überschreiben können und dass wir die Annotation @Override verwenden sollten, wenn wir eine Methode überschreiben, um zu vermeiden, dass wir - z.B. weil wir den Namen der Methode falsch schreiben - die Methode gar nicht überschreiben, sondern eine neue Methode hinzufügen.

**Übrigens :** Wir kennen das Schlüsselwort final ja bereits von Variablen. Eine als final deklarierte Variable kann ihren einmal zugewiesenen Wert nicht mehr ändern. Wir haben mit final sogenannte *Konstanten* definiert.

Eine als final deklarierte Methode kann **nicht** überschrieben werden!

Von einer als final deklarierten Klasse kann **nicht** geerbt werden!

Die Klasse Quadrat

Das folgende Beispiel dient nur zur Wiederholung. Wir erben nun eine Klasse Quadrat von der Klasse Rechteck.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public class Quadrat extends Rechteck  {  Quadrat(int seite)  {  super(seite, seite); // Aufruf des Konstruktors von Rechteck  }  @Override  public void print()  {  System.out.print("[ a=" + this.a + ", b=" + this.b  + ", c=" + this.c + ", d=" + this.d + " ] ");  System.out.print(" Umfang des Quadrats : " + this.umfang());  System.out.println(" Flaecheninhalt des Quadrats : " + this.flaecheninhalt());  }  } |

* In Zeile 1 geben wir mithilfe von extends Rechteck an, dass die Klasse Quadrat von der Klasse Rechteck erbt.
* In den Zeilen 3-6 definieren wir den Konstruktor von Quadrat. Darin rufen wir als erstes den Konstruktor von Rechteck auf. Der Konstruktor von Quadrat erwartet nur noch eine Parameter, da im Quadrat alle Seiten gleich lang sind. Den Wert des Parameters übergeben wir dem Konstruktor von Rechteck, der zwei Parameterwerte erwartet (für laenge und breite).
* In den Zeilen 8-15 überschreiben wir wieder die print()-Methode und passen sie an das Quadrat an.

Beachten Sie, dass Quadrat alle Eigenschaften von Rechteck erbt, also

* die Objektvariablen a, b, c, d und
* die Objektmethoden print(), umfang() und flaecheninhalt().

Die Testklasse könnten wir nun wie folgt erweiteren:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | public static void main(String[] args)  {  System.out.printf("%n%n------- Objekte von Viereck erzeugen ------------%n%n");  Viereck v1 = new Viereck(10,20,30,40);  v1.print();  Viereck v2 = new Viereck(15,20,25,20);  v2.print();  System.out.printf("%n%n------- Objekte von Rechteck erzeugen ------------%n%n");  Rechteck r1 = new Rechteck(10, 20);  r1.print();  Rechteck r2 = new Rechteck(20, 30);  r2.print();  System.out.printf("%n%n------- Objekte von Quadrat erzeugen ------------%n%n");  Quadrat q1 = new Quadrat(30);  q1.print();  Quadrat q2 = new Quadrat(40);  q2.print();  } |

und bekämen folgende Ausgaben:

------- Objekte von Viereck erzeugen ------------

[ a=10, b=20, c=30, d=40 ] Umfang des Vierecks : 100

[ a=15, b=20, c=25, d=20 ] Umfang des Vierecks : 80

------- Objekte von Rechteck erzeugen ------------

[ a=10, b=20, c=10, d=20 ] Umfang des Rechtecks : 60 Flaecheninhalt des Rechtecks : 200

[ a=20, b=30, c=20, d=30 ] Umfang des Rechtecks : 100 Flaecheninhalt des Rechtecks : 600

------- Objekte von Quadrat erzeugen ------------

[ a=30, b=30, c=30, d=30 ] Umfang des Quadrats : 120 Flaecheninhalt des Quadrats : 900

[ a=40, b=40, c=40, d=40 ] Umfang des Quadrats : 160 Flaecheninhalt des Quadrats : 1600

**Success**

Mit Vererbung haben wir ein wichtiges Konzept der objektorientierten Programmierung kennengelernt. Um von einer Klasse zu erben, verwenden wir das Schlüsselwort extends. Eine Kindklasse erbt von ihrer Elternklasse alle Eigenschaften, also alle Objektvariablen und Objektmethoden. Eine geerbte Methode kann in der Kindklasse überschrieben werden. Wird eine Methode überschrieben, verwenden wir die Annotation @Override. Im Konstruktor der Kindklasse wird der Konstruktor der Elternklasse aufgerufen. Dies kann implizit erfolgen, wenn der implizite Konstruktor der Elternklasse existiert, oder es erfolgt explizit durch die Verwendung des Schlüsselwortes super. Mithilfe von Vererbung erhöhen wir die Wiederverwendbarkeit von Code und vermeiden doppelte Implementierungen. Außerdem sorgen wir für eine bessere Strukturierung des Codes.