TEKO
SCHWEIZERISCHE FACHSCHULE

Objektorientiertes Programmieren 1

Beziehungen zwischen Klassen und Objekten

Stephan Kessler, BSc in Systems Engineering

Kontakt: stephan.kessler@edu.teko.ch

Vorbereitung

Lesen Sie folgende Abschnitte im Buch ***Sprechen Sie Java?*** als Vorbereitung für dieses Thema. Beachten Sie, dass bei Angaben von Unterkapiteln nur diese auch gelesen werden müssen:

- Objektorientierung (S. 151 – 159)
  - Statische und Objektbezogene Felder und Methoden
  - Beispiel: Klasse PhoneBook
  - Beispiel: Klasse Stack
- Vererbung (S. 183-187)
  - Klassifikation

Statische und nicht statische Elemente kombiniert

```java
public class Nebenklasse
{
    public static int id_count = 0;
    public int id;

    public Nebenklasse(){
        id_count = id_count + 1;
        id = id_count;
    }

    public void printID(){
        System.out.println("Nebenklasse ID: " + id);
    }
}
```

In diesem Beispiel wurde die Klasse *Nebenklasse* so programmiert, dass sie einen statischen Integer *id_count* besitzt. Dies bedeutet, dass diese Variable Klassenbezogen ist.

Statische und nicht statische Elemente kombiniert

```
public class Nebenklasse
{
    public static int id_count = 0;
    public int id;

    public Nebenklasse(){
        id_count = id_count + 1;
        id = id_count;
    }

    public void printID(){
        System.out.println("Nebenklasse ID: " + id);
    }
}
```

Alle anderen Variablen und Methoden wurden nicht-statisch programmiert und sind somit Objektbezogen.

Statische und nicht statische Elemente kombiniert

```java
public class Nebenklasse
{
    public static int id_count = 0;
    public int id;

    public Nebenklasse(){
        id_count = id_count + 1;
        id = id_count;
    }

    public void printID(){
        System.out.println("Nebenklasse ID: " + id);
    }
}
```

Im Konstruktor werden nun beide Typen (statisch und nicht statisch) verwendet. Dabei wird beim Instanziieren die Variable *id_count* um den Wert 1 erhöht und in *id* abgespeichert.

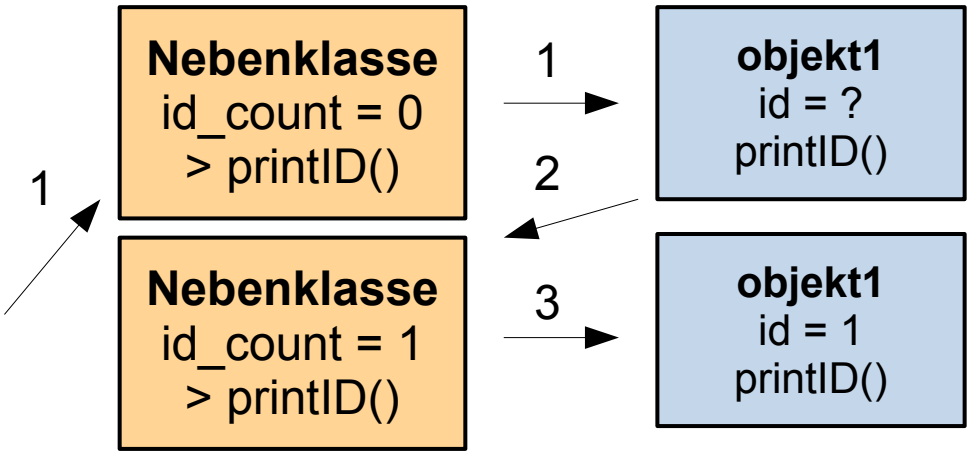
Statische und nicht statische Elemente kombiniert

```
public class Hauptklasse
{
    public static void main(String[] args){
        Nebenklasse objekt1 = new Nebenklasse();
        Nebenklasse objekt2 = new Nebenklasse();

        objekt2.printID();
        objekt1.printID();
    }
}
```

Wenn wir nun zwei Objekte des Typs *Nebenklasse* in der *Hauptklasse* instanziieren, passiert folgender Effekt.

Statische und nicht statische Elemente kombiniert

```java
public class Hauptklasse
{
	public static void main(String[] args){
		Nebenklasse objekt1 = new Nebenklasse();
		Nebenklasse objekt2 = new Nebenklasse();

		objekt2.printID();
		objekt1.printID();
	}
}
```



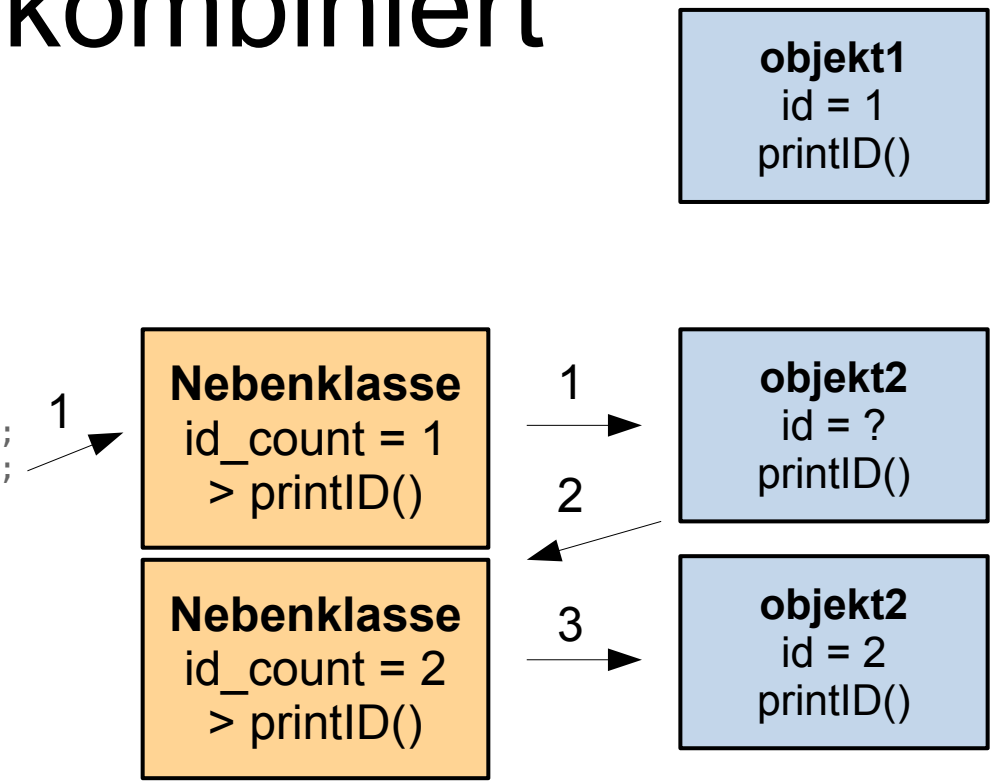
1. Aus der Klasse *Nebenklasse* wird nun ein Objekt instanziiert (*klasse1*).

2. Durch den Konstruktor wird nun die Variable *id_count* in der Klasse *Nebenklasse* um den Wert 1 erhöht.

3. Dieser Wert wird während dem Instanziieren in *id* des Objektes Abgespeichert.

Statische und nicht statische Elemente kombiniert

```
public class Hauptklasse
{
     public static void main(String[] args){
          Nebenklasse objekt1 = new Nebenklasse();
          Nebenklasse objekt2 = new Nebenklasse();

          objekt2.printID();
          objekt1.printID();
     }
}
```



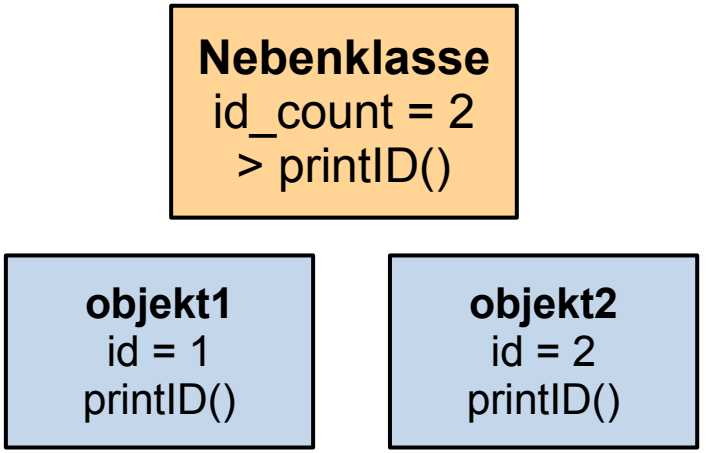
1. Nun wird das selbe mit dem Objekt *klasse2* durchgeführt. Das Objekt wird instanziiert.

2. Durch den Konstruktor wird wieder die Variable *id_count* in der Klasse *Nebenklasse* um den Wert 1 erhöht.

3. Dieser Wert wird während dem Instanziieren in *id* des Objektes Abgespeichert.

Statische und nicht statische Elemente kombiniert

```
public class Hauptklasse
{
     public static void main(String[] args){
          Nebenklasse objekt1 = new Nebenklasse();
          Nebenklasse objekt2 = new Nebenklasse();

          objekt2.printID();
          objekt1.printID();
     }
}
```

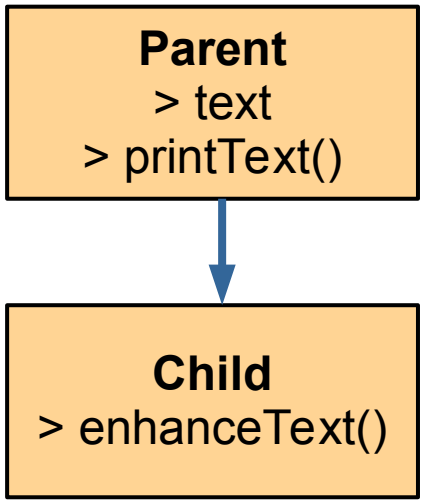


Dadurch erhält man zwei Objekte mit automatisch inkrementierter ID. Dies ist möglich, da die statische Variable **global** bestehen bleibt.

Sie können sich also merken, dass **statische Elemente** auch **globale Elemente sind**.

Die Ausgabe in diesem Beispiel ist entsprechend wie folgt (beachten Sie, dass *klasse2* zuerst ausgegeben wird):
Nebenklasse ID: 2
Nebenklasse ID: 1

Vererbung



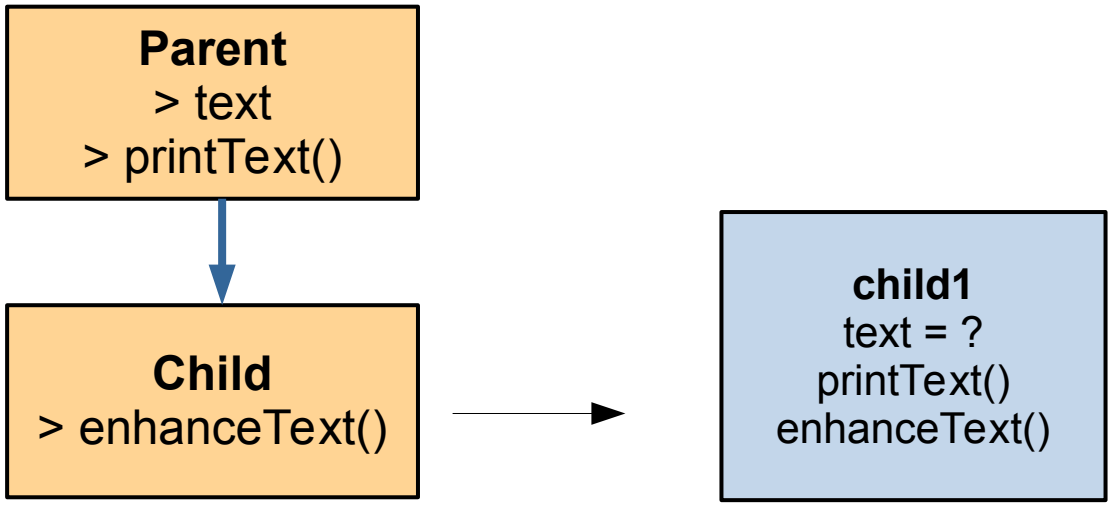
In Java haben Sie die Möglichkeit von Klassen zu erben. Das bedeutet, dass die erbenden Klassen Eigenschaften (Variablen) und Methoden der vererbten Klasse übernehmen können.

In diesem grafischen Beispiel sind zwei Klassen dargestellt:

- *Parent* mit der nichtstatischen Variable *text* und der nichtstatischen Methode print*Text*
- *Child* mit der nichtstatischen Methode enhance*Text()*

Zudem erbt *Child* von *Parent*.

Vererbung



Instanziiert man nun ein Objekt aus *Child* besitzt dieses Objekt die Eigenschaften und Methoden aus *Child* **und** *Parent*.

Vererbung

```java
public class Parent
{
    public String text;

    public void printText(){
        System.out.println(text);
    }
}
```

```java
public class Child extends Parent
{
    public void enhanceText(){
        text = text + "more";
    }
}
```

Parent und *Child* sind entsprechend nach dem grafischen Schema aufgebaut.

Vererbung

```java
public class Parent
{
    public String text;

    public void printText(){
        System.out.println(text);
    }
}
```

```java
public class Child extends Parent
{
    public void enhanceText(){
        text = text + "more";
    }
}
```

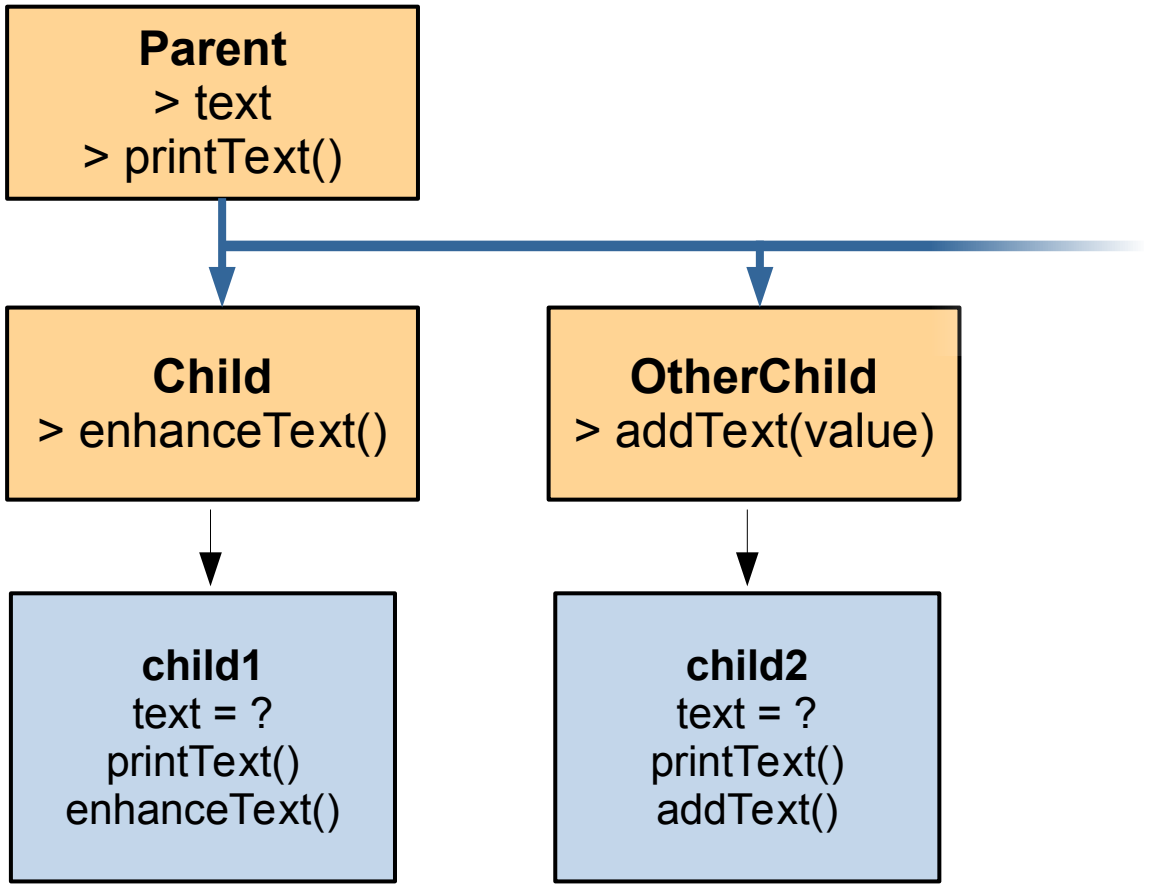
Damit nun *Child* von *Parent* erbt, wird in Java das Schlüsselwort ***extends*** verwendet.

Vererbung

```
public class Hauptklasse
{
    public static void main(String[] args){
        Child child1 = new Child();
        child1.text = "Hello";
        child1.enhanceText();
        child1.printText();
    }
}
```

In einer weiteren Klasse (im Beispiel: *Hauptklasse*) kann nun ein Objekt aus *Child* wie gewohnt instanziiert werden. Wie zu erwarten war, kann nun auch auf *text* und *enhanceText()* zugegriffen werden, obwohl diese Elemente nicht direkt in der Klasse *Child* einprogrammiert wurden. Dies ist möglich, da *Child* von *Parent* erbt und diese Elemente dort vorhanden (und erreichbar) sind.

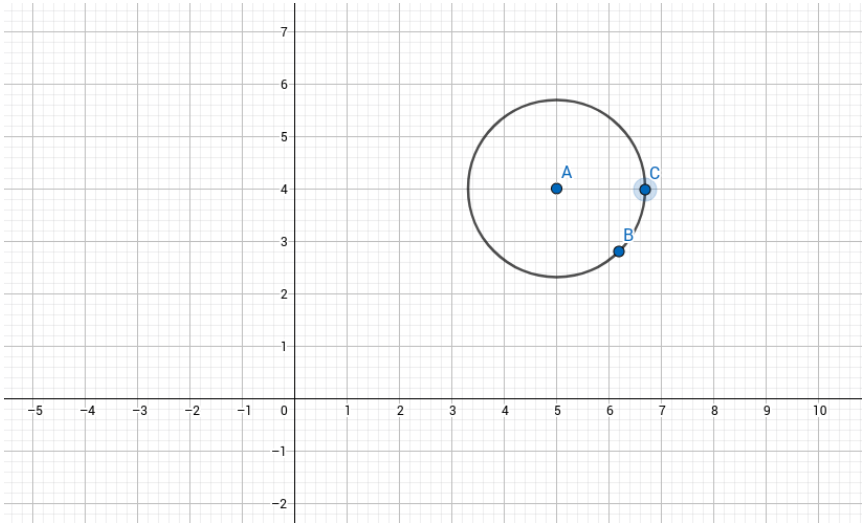
Vererbung



Natürlich können auch mehrere Klassen von der selben Klasse erben.

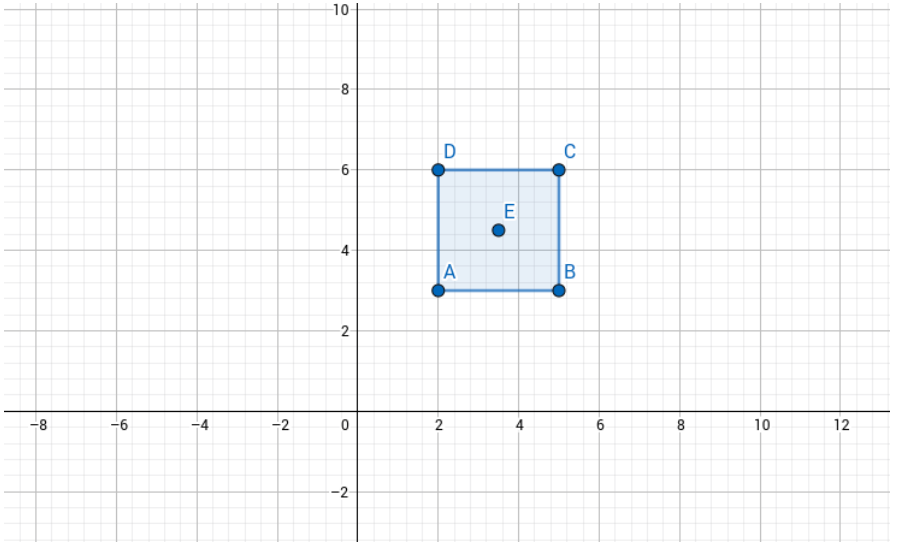
Aufgabe

- Schreiben Sie eine Klasse *KoordinatenElement* welche die folgenden nichtstatische Variablen besitzt: *x* (Float) und *y* (Float)
- Schreiben Sie nun eine Klasse *Kreis* die von *KoordinatenElement* erbt und die nichtstatische Variable *radius* besitzt.
- Schreiben Sie nun die Methode maxX*()* in der Klasse *Kreis,* die den maximal positiven X-Wert zurück gibt, welcher vom Kreis eingeschlossen ist. In der folgenden Grafik ist dies der Punkt C (6,7). Die Koordinatenwerte *x* und *y* stellen dabei die Position des Mittelpunktes des Kreises dar.



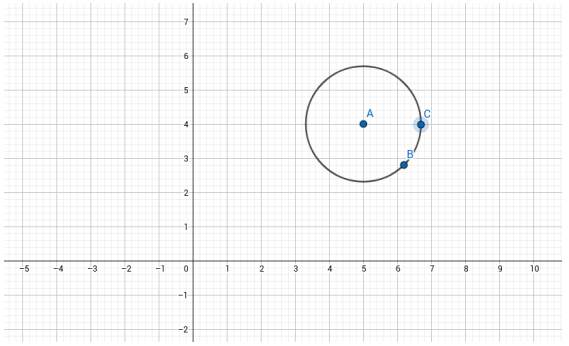
Aufgabe

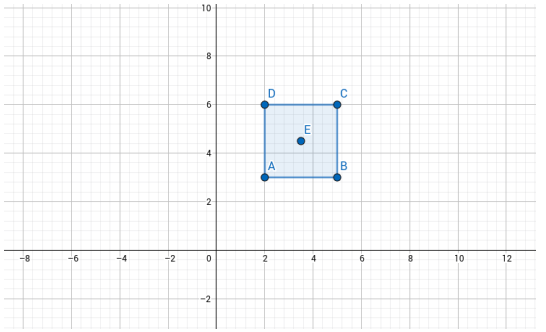
- Schreiben Sie nun eine Klasse *Viereck* welche auch von der Klasse *KoordinatenElement erbt*. Diese soll die nichtstatische Variable *seite* besitzen, welche die Seitenlänge angibt.
- Schreiben Sie eine Methode *mitteX()* in V*iereck*, welche den Mittelpunkt-X-Wert des Viereckes zurückgibt. Dies wäre der Punkt E in der Grafik. Dabei sollen die *x*- und *y*-Werte als Eckpunkt des Viereckes interpretiert werden (Punkt A in der Grafik).



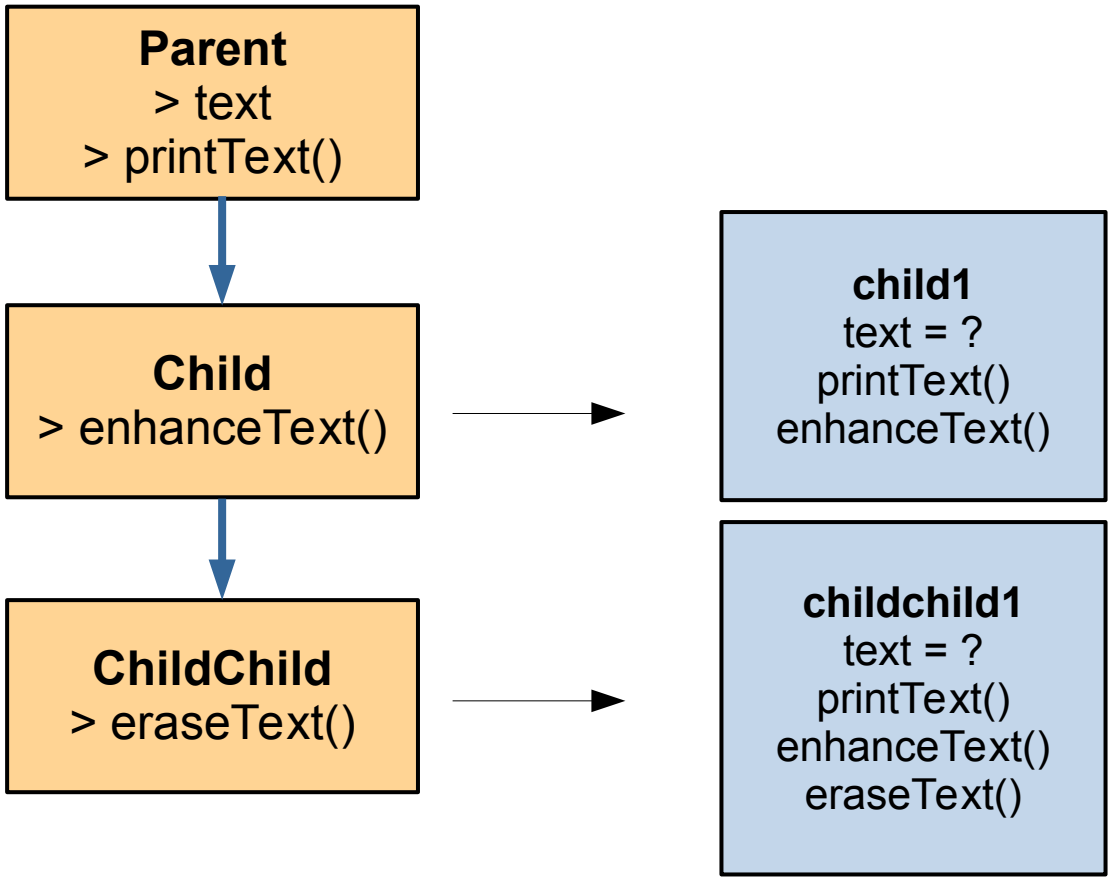
Aufgabe

- Instanziieren Sie nun jeweils ein Objekt aus *Kreis* und *Viereck*. Füllen Sie die entsprechenden Variablen und führen Sie die Methoden aus.
- Welche Vorteile ziehen Sie aus der Vererbung?



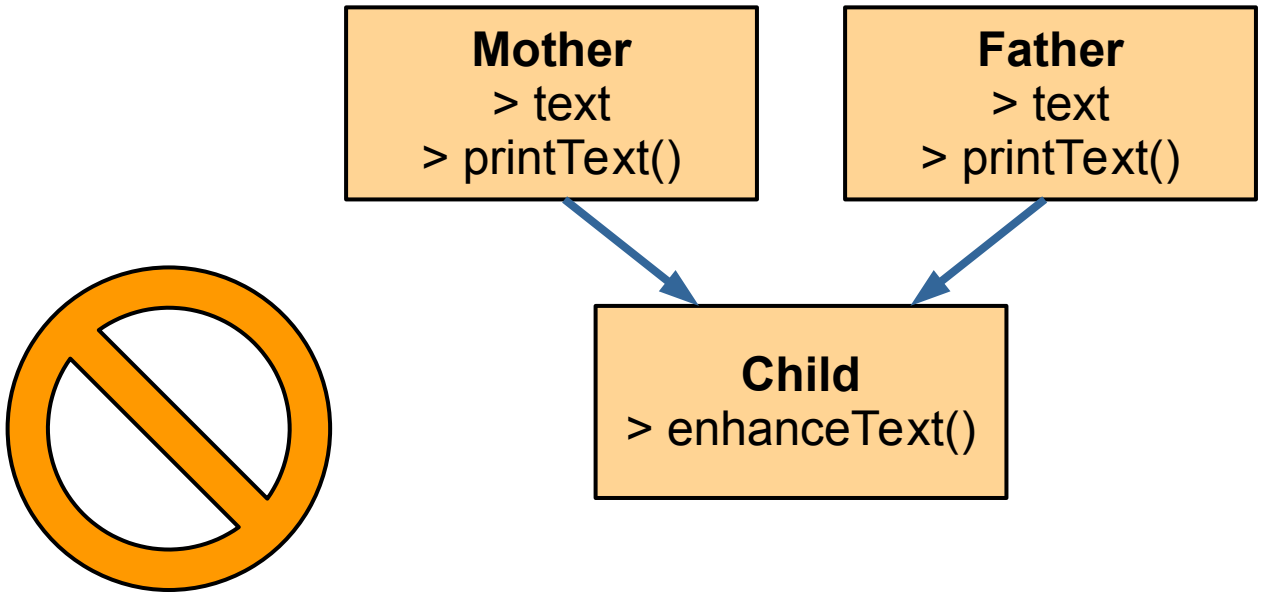


Vererbung (mehrstufiges Erben)



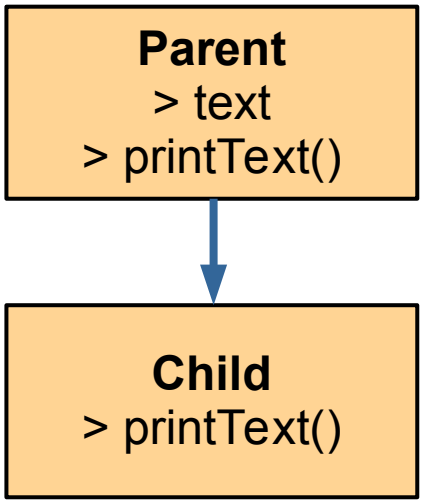
In Java kann man auch die geerbten Klassen weiter vererben.

Vererbung (Mehrfachvererbung)



In Java ist es jedoch **nicht möglich**, dass eine Klasse von mehreren Klassen erbt (Mehrfachvererbung).

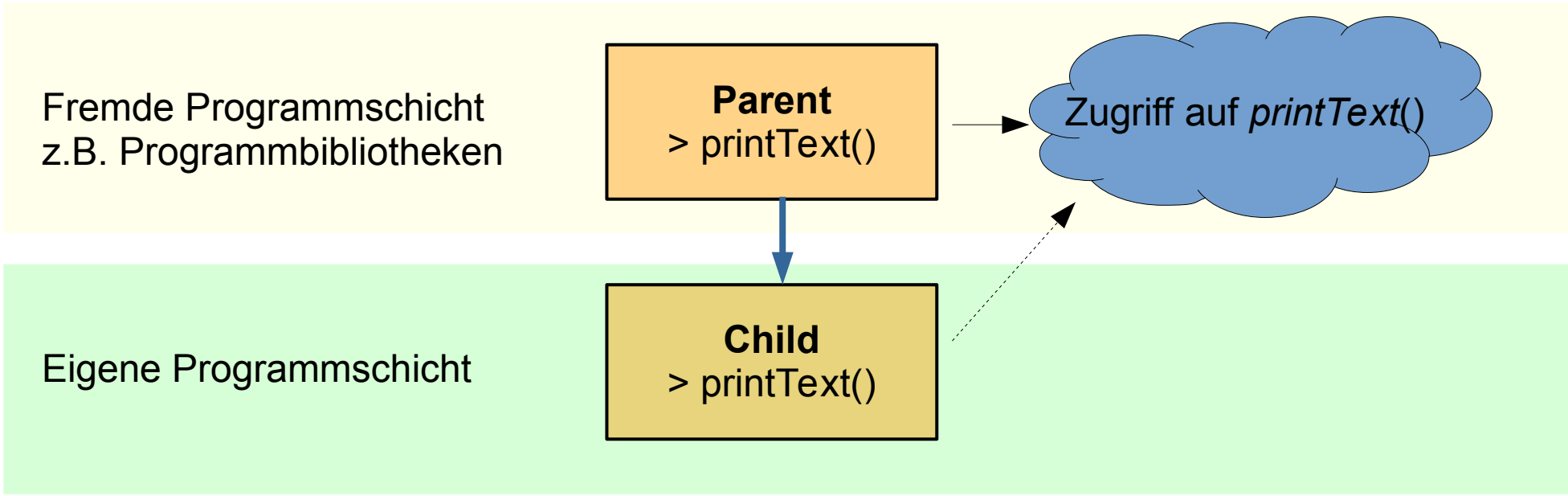
Methoden überschreiben



In Java haben Sie die Möglichkeit Methoden zu überschreiben. Dies ermöglicht die Übergabe von Algorithmen zur Laufzeit und ist z.B. für die Verwendung bei manchen fremden Programm-Bibliotheken notwendig.

(Methode gleiche Signatur, jedoch erneute Implementation → **Polymorphismus**)

Methoden überschreiben



Diese Variante macht vor allem dann Sinn, wenn im Voraus noch nicht bekannt ist, was die entsprechende Methode ausführen soll oder man bewusst die Implementierung der Methode flexibel halten möchte.

Da in diesem Beispiel *Parent* mindestens eine Methode *printText()* besitzt, kann der Zugriff auf dieser Methode bereits programmiert werden, bevor der endgültige Inhalt von *printText()* bekannt ist. Durch die Überschreibung der Methode kann der Programmierer im Nachhinein die Funktionalität der Methode bestimmen.

Methoden überschreiben

```java
public class Parent
{
    public void printText(){

    }
}
```

```java
public class Child extends Parent
{
    public void printText(){
        System.out.println("Hallo");
    }
}
```

In Java würde die Überschreibung entsprechen so aussehen. Da in der Klasse *Child* eine Methode mit exakt gleichen Namen und Parametern gewählt wurde, wird die geerbte Methode überschrieben.

Aufgabe

- Nehmen Sie die Klasse *ParentFlow* (ParentFlow.java) und *Parent* (Parent.java), die zur Verfügung gestellt wurden, in ein neues Projekt auf.

  In der Klasse *ParentFlow* können Sie eine Klasse, welche von *Parent* geerbt hat, registrieren. Dies wird über die Methode *setParent(Parent)* in der Klasse *ParentFlow* erreicht. Die Klasse *Parent* besitzt eine Methode *doIt()* welche nach der Registration in *ParentFlow* und dem Starten von *ParentFlow* alle 3 Sekunden ausgeführt wird.
- Schreiben Sie eine Klasse die von *Parent* erbt. Überschreiben Sie die Methode *doIt()* so, dass diese beim Ausführen einen Wert, welcher in Ihrer Klasse gespeichert ist, hoch zählt und den aktuellen Wert ausgibt.
- Erstellen Sie nun die Klasse *Hauptklasse*, welche eine Main-Methode enthält.
- In der Main-Methode: Instanziieren Sie nun ein Objekt aus Ihrer geerbten Klasse. Instanziieren Sie zudem ein Objekt aus der Klasse *ParentFlow*.
- In der Main-Methode: Registrieren Sie Ihr geerbtes Objekt im Objekt *ParentFlow* über dessen Methode *setParent(Parent)*. Sie können dabei, das Objekt als Parameter übergeben.
- In der Main-Methode: Starten Sie das Objekt aus *ParentFlow* mit der Methode *start()*. Wenn Sie nun das Programm ausführen, sollte der Wert des geerbten Objektes alle 3 Sekunden inkrementiert ausgegeben werden.