Leibniz Universität IT Services

Leibniz Universität Hannover

Java - Generische Programmierung

Generische Programmierung

- Entwicklung von wiederverwendbaren Bibliotheken.
- Entwurf von allgemeinen Funktionen unabhängig vom Datentyp.
- Schreiben von Algorithmen, die unabhängig von einer bestimmten Datenstruktur sind.
- Nutzung in Sortieralgorithmen, Bäumen, Listen etc..

Klasse ohne generische Programmierung

```
public class clsPunkt {
    private int x;
    private int y;

    public clsPunkt(){}
    public clsPunkt(int xKoord, int yKoord){}

    public String toString(){}
    public void setPunkt(int xKoord, int yKoord){}
}
```

Nachteile

- Es können nur (Integer)-Koordinaten gespeichert werden.
- Für jede andere Datentyp-Kombination muss eine neue Klasse erzeugt werden.

Generische Klassen

```
public class clsPunkt<T> {
    private T x;
    private T y;

    public clsPunkt(){}
    public clsPunkt(T xKoord, T yKoord){}

    public String toString(){}
    public void setPunkt(T xKoord, T yKoord){}
}
```

Kopf einer Klassen

```
public class clsPunkt<T> {

```

- Jede Klasse wird durch das Schlüsselwort `class` gekennzeichnet.
- Jede Klasse hat einen eindeutigen Namen. In diesem Beispiel wird die Klasse `clsPunkt` implementiert.
- Das Schlüsselwort vor dem Namen regelt den Zugriff auf die Klasse von außen her.
- In eckigen Klammern wird eine Typ-Variable für die Klasse definiert.

Klassenname

- Beginn mit einem Großbuchstaben. Zum Beispiel „Kreis“.
- Nutzung der Kamel-Notation. Zum Beispiel „GeometrieKreis“. Jedes Wort beginnt mit einem Großbuchstaben.
- Der Name spiegelt die Bezeichnung eines Dinges in der realen Welt wieder.
- Der Name ist innerhalb eines Paketes eindeutig.

Öffentliche Klassen

```
public class clsPunkt<T> {

```

- Das Schlüsselwort `public` kennzeichnet öffentliche Klassen.
- Die Klasse kann von jedem Paket genutzt werden.
- Die Klasse kann von allen anderen Klassen verwendet werden.

Typ-Variable

```
public class clsPunkt<T> {
```

- Die Angabe der Typ-Variablen erfolgt nur einmal im Kopf der Klasse hinter dem Klassennamen.
- Die Typ-Variable wird in spitze Klammern gesetzt.

Formale Typ-Parameter

- Einzelne Großbuchstaben wie T.
- Häufig genutzte Parameter: **T**yp, **E**lement, **K**ey (Schlüssel) , **V**alue (Wert).

Attribute in der generischen Klasse

```
private T x;
private T y;
```

- Attribute werden wie Variablen in einer Klasse deklariert.
- Instanzvariablen werden am Anfang einer Klasse im Rumpf deklariert.

Deklaration von Variablen

T	x	;
typ	variablenname	;

- Der Variablenname ist frei wählbar.
- Die, im Klassenkopf angegebene Typ-Variable, primitive Datentypen oder definierte Klassen werden als Datentyp genutzt.
- Die Variablen können im Konstruktor initialisiert werden. Variablen werden in Methoden verändert.

Standard-Konstruktor

```
public clsPunkt()
{
    x = null;
    y = null;
}
```

- Konstruktoren werden bei der Erzeugung einer Instanz automatisiert aufgerufen.
- Konstruktoren und die Klasse haben den gleichen Namen.
- Jede Klasse hat einen geerbten oder definierten parameterlosen Konstruktor.
- Attribute vom Datentyp der Typ-Variablen der Klasse werden auf null gesetzt.

Weitere Konstruktoren

```
public clsPunkt(T xKoord, T yKoord)
{
      this.x = xKoord;
      this.y = yKoord;
}
```

- Konstruktoren können überladen werden. Sie werden mit Hilfe der Argumentliste unterschieden.
- In diesem Beispiel werden Argumente übergeben, die als Datentyp die Typ-Variable der Klasse nutzen.
- Der Parameter wird dem passenden Attribut mit Hilfe des Gleichheitszeichen zugewiesen. Attribut und Argument sollten vom gleichen Datentyp sein.

Erzeugung der Objekte

```
clsPunkt<Integer> punkt01 = new clsPunkt(1, 3);
clsPunkt<Double> punkt02 = new clsPunkt();
```

- Objekt-Variablen sind von einem bestimmten Typ „Klasse“.
- Objekt-Variablen können wie in diesem Beispiel gleichzeitig deklariert und initialisiert werden.

Zuweisung an Attribute

```
public clsPunkt(T xKoord, T yKoord)
{
      this.x = xKoord;
      this.y = yKoord;
 }
```

- Attribute werden mit Hilfe des Punktoperators einem Objekt zugeordnet.
- Das Schlüsselwort `this` ist ein Platzhalter für das momentan zu konstruierende Objekt. In Methoden beschreibt es die Instanz, die die Methode aufgerufen hat.

Deklaration von Objekt-Variablen

`clsPunkt`	`<`	`Integer`	`>`	`punkt01`	`;`
`Template`	`<`	`Klasse`	`>`	`variablenname`	`;`

- Statt eines Datentyps wird ein Klassenname als Typ angegeben. Der Klassenname muss exakt wie im Kopf (`public class clsPunkt`) der Klasse beschrieben, geschrieben werden.
- Der Variablenname ist frei wählbar.
- Zwischen Datentyp und Klassenname werden die Argumente für die Typ-Variable in spitzen Klammern angegeben. Primitive Datentypen können nicht genutzt werden.

Initialisierung von Objekt-Variablen

`punkt01`	`=`	`new`	`clsPunkt`	`(`		`)`	`;`
`variablenname`	`=`	`new`	`Template`	`(`		`)`	`;`

- Das Schlüsselwort `new` erzeugt mit Hilfe eines Konstruktors eine Instanz von einer Klasse.
- Der Objekt-Variablen wird ein Verweis auf das neu erstellte Objekt übergeben.
- In den runden Klammern wird entsprechend des gewählten Konstruktors eine Argumentliste angegeben.

Methoden in der Klasse

```
public void setPunkt(T xKoord, T yKoord)
{
    this.x = xKoord;
    this.y = yKoord;
}
```

Aufbau von Methoden

```
public void setPunkt(T xKoord, T yKoord)
```

Methodenkopf

```
{
  this.x = xKoord;
        this.y = yKoord;
}
```

Methodenrumpf

Methodenrumpf

- Beginn und Ende mit den geschweiften Klammern.
- Zusammenfassung von Anweisungen für eine bestimmte Aktion.
- Die Aktion wird in dem Methodennamen beschrieben.

Methodenkopf

`public`	`void`	`setPunkt`	`(`		`)`
`zugriff`	`datentyp`	`methodenname`	`(`		`)`

- Jede Methode beginnt mit dem Zugriffsmodifikator. Methoden sind häufig öffentlich (`public`).
- Dem Zugriffsmodifikator folgt der Datentyp der Methode.
- Dem Datentyp folgt der Methodenname. Der Methodenname ist eindeutig in einer Klasse.
- Dem Methodennamen folgt in runden Klammern die Parameterliste.

Parameterliste

`void`	`setPunkt`	`(`	`T xKoord, T yKoord`	`)`
`typ`	`methodenname`	`(`	`typ para01, typ para02`	`)`

- Die Parameterliste beginnt und endet mit den runden Klammern.
- Die Parameterliste folgt direkt dem Namen der Methoden.
- Die Parameterliste kann beliebig viele Parameter von beliebigen Typ enthalten.

Parameter in der Parameterliste

`void`	`setPunkt`	`(`	`T xKoord, T yKoord`	`)`
`typ`	`methodenname`	`(`	`typ para01, typ para02`	`)`

- Jeder Parameter ist von einem bestimmten Typ. Dies kann eine Typ-Variable, die im Klassenkopf definiert ist, sein. Es können primitive Datentypen oder eine definierte Klasse als Typ genutzt werden.
- Die Parameter werden durch ein Komma getrennt.
- Parameter können nur in dem Methodenrumpf verwendet werden, zu dem der Methodenkopf gehört.
- Parameter sind lokale Variablen einer Methode.

Rückgabewerte von Methoden

```
public void setPunkt(T xKoord, T yKoord){
    this.x = xKoord;
    this.y = yKoord;
}
```

```
public T getPunktY(){
    return this.y;
}
```

```
public String toString(){
    return this.x + ", " + this.y;
}
```

Erläuterung

- Eine Methode vom Datentyp `void` gibt kein Wert an den Aufrufer zurück.
- Methoden können aber mit Hilfe des Schlüsselwortes `return` einen Wert an den Aufrufer zurückgeben. Dieser Wert kann vom Datentyp einer Typ-Variable, die im Klassenkopf definiert ist, sein. Es können primitive Datentypen oder eine definierte Klasse als Rückgabewert genutzt werden.

Konvertierung von Typ-Parametern

```
Integer zahl;
zahl  = (Integer)this.wert;
```

- (Wrapper-Klasse)[Typ-Parameter].
- Typ-Parameter können in jedes Objekt von einem bestimmten Typ implizit konvertiert werden.
- Eine Konvertierung in primitive Datentypen ist nicht möglich.

Konvertierung von Typ-Variablen

```
public Integer getPunktY()
{
    return (Integer)this.y;
}
```

- Typ-Variablen können nicht in primitive Datentypen umgewandelt werden.
- In diesem Beispiel wird die Instanzvariable y in den Datentyp `Integer` konvertiert.
- Die Klasse, in die die Typ-Variable umgewandelt wird, wird in runden Klammern vor dem umzuwandelnden Wert angegeben.

Aufruf von Methoden

```
String = punkt01.toString();
punkt02.setPunkt(2.4, 1.03);
```

- Objekt-Variable und Methode werden mit einem Punkt verbunden.
- Die Objekt-Variable ist von dem Typ „Klasse / Template“, in dem die Methode definiert ist.
- In den runden Klammern wird der Methode eine Argumentliste übergeben.
- Die Methode kann einen Wert zurückgeben, muss aber nicht. Der Rückgabewert kann in einer Variablen gespeichert werden.

Typ-Variablen als Rückgabewerte

```
if (punkt01.getPunktX() instanceof Integer){
    Integer x = punkt01.getPunktX();
    System.out.println("X-Koordinate: " + x);
}
```

- Mit Hilfe des Schlüsselwortes `instanceof` wird überprüft, ob der Rückgabewert von der Klasse ist. Die Bedingung liefert einen boolschen Wert zurück.
- In diesem Beispiel wird überprüft, ob der Rückgabewert von der Wrapper-Klasse Integer ist. Falls die Bedingung `True` liefert, wird der Wert in einer Integer-Variablen gespeichert.

Nutzung von x Typ-Variablen

```
public class clsDictonary<K, V> {
    private K schluessel;
    private V wert;

    public clsDictonary(){}
    public clsDictonary(K key, V value){}

    public String toString() {}
    public K getSchluessel() {}
    public V getWert() {}
    public void setElement(K key, V value){}
}
```

Kopf einer Klassen

```
public class clsDictonary<K, V> {
```

- In eckigen Klammern werden x Typ-Variablen angegeben.
- Die Typ-Variablen werden durch ein Komma getrennt.
- Die Typ-Variablen stehen jeweils für einen bestimmten Datentyp.

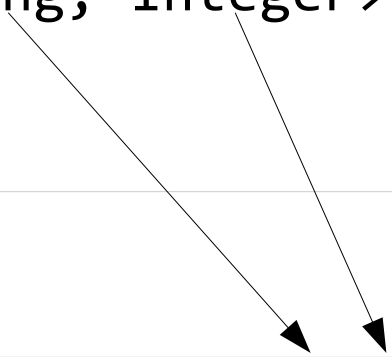
Deklaration der Objekt-Variablen

```
clsDictonary<String, Integer> schmelzpunkt
clsDictonary<Integer, Integer> siedepunkt
```

- Der Name der Objekt-Variablen ist frei wählbar. In diesem Beispiel werden die Variablen `schmelzpunkt` und `siedepunkt` erzeugt.
- Die Objekt-Variablen sind von einem bestimmten Datentyp. In diesem Beispiel nutzen die Variablen die Klasse `clsDictonary`.
- Zwischen den Datentyp und dem Variablenname werden die genutzten Datentypen in spitzen Klammern angegeben.

Zuordnung

```
clsDictonary<String, Integer> schmelzpunkt
```



```
public class clsDictonary<K, V> {
```

- In Abhängigkeit der Position werden die Datentypen den Typ-Variablen zugeordnet.