

Einfache Datentypen vs. Klassen

Die Programmiersprache Java (die auf der Sprache C++ basiert) ist zwar objektorientiert, aber nicht vollständig. In einer **vollständig objektorientierten** Sprache (wie z.B. Smalltalk) sind alle Datentypen Klassen mit Attributen, Methoden usw.

In Java (wie im Vorgänger C++) gibt es einige einfache, häufig verwendete Datentypen wie `int`, `double` und `boolean`, die keine Klassen sind. Das heißt, wenn man eine Variable vom Typ `int` deklariert, kann man auf diese Variable keine Methoden anwenden:

```
public void beispiel()  
{  
    int a;  
    a.set(0);           // nicht möglich  
    a.increase(5);  
}
```

Man sieht den Unterschied auch an der Schreibweise: die „einfachen“ Datentypen `int`, `double` usw. beginnen mit einem Kleinbuchstaben, während alle Klassen mit Großbuchstaben beginnen.

Ein Grund, warum man in Java diesen Unterschied macht, ist, dass Klassen mit ihren Methoden zusätzlichen Zeitaufwand sowie Speicherplatz im Programmablauf verursachen. Jeder Methodenaufruf kostet beim Programmablauf einige zusätzliche Prozessorbefehle, und jedes Objekt belegt einige Bytes zusätzlich zu den eigentlichen Daten, die es speichert (den sogenannten „Overhead“). Da insbesondere Zahlen in Schleifen oft massenhaft bearbeitet werden, möchte man sich diesen Zusatzaufwand in zeit- bzw. speicherkritischen Anwendungen ersparen.

Einfache Datentypen und Datenstrukturen

Grundsätzlich ist es kein großes Problem, dass es einfache Attribute und Variablen gibt, die keine Objekte sind. In einigen Situationen ist das aber hinderlich. Möchte man zum Beispiel eine dynamische Liste von Zahlen verwenden, kann man diese nicht mit „`int`“ als Inhalts-Objekt deklarieren – denn die Klasse `List` benötigt eine Klasse als „`ContentType`“:

```
public class Zahlenliste  
{  
    private List<int> zl;    // nicht möglich!
```

Eine Lösung wäre, selbst eine Klasse zu implementieren, die ein Attribut vom Typ `int` hat, und diese dann als `ContentType` zu verwenden. Da das aber recht aufwendig ist, haben die Entwickler von Java für solche Fälle sogenannte „Wrapper-Klassen“ für die einfachen Datentypen `int`, `double` usw. entwickelt, die Programmierer in solchen Fällen verwenden können.

Wrapper-Klassen

Eine Wrapper-Klasse „verpackt“ sozusagen einen einfachen Datentyp in eine Klasse. Diese Klassen heißen ähnlich wie die einfachen Datentypen, beginnen aber mit Großbuchstaben.

Beispielhaft werden hier einige Methoden der Klassen Integer und Double vorgestellt. Die vollständige Beschreibung aller Wrapper-Klassen ist in der Java-API-Dokumentation zu finden.

Datentyp	Wrapper-Klasse
int	Integer
double	Double
boolean	Boolean
char	Character

Klasse Integer

Integer(int pValue)	Konstruktor, der den int-Wert pValue für das erzeugte Objekt übernimmt.
Integer(String pText)	Konstruktor, der die durch den Text pText dargestellte Zahl für das erzeugte Objekt übernimmt.
int intValue()	Gibt den Wert des Objekts als int zurück.
String toString()	Gibt den der Zahl entsprechenden Text zurück.

Klasse Double

Double(double pValue)	Konstruktor, der den double-Wert pValue für das erzeugte Objekt übernimmt.
Double(String pText)	Konstruktor, der die durch den Text pText dargestellte Zahl für das erzeugte Objekt übernimmt.
double doubleValue()	Gibt den Wert des Objekts als double zurück.
String toString()	Gibt den der Zahl entsprechenden Text zurück.

Automatische Umwandlung

Java wandelt automatisch zwischen einfachen Variablen bzw. Konstanten und Objekten von Wrapper-Klassen um. Man kann also Objekte von Wrapper-Klassen wie einfache Variablen in Ausdrücken, Zuweisungen und als Parameter einsetzen und Operatoren wie +, −, *, /, <, > anwenden. Die Methoden der Klassen Integer, Double usw. braucht man daher nur selten.

```
public void beispiel
{
    // Konstante 7.0 wird von double nach Double umgewandelt
    Double zahl = 7.0, ergebnis;

    // Math.sqrt erwartet double → wird von Double umgewandelt
    // Math.sqrt liefert double → wird nach Double umgewandelt
    ergebnis = Math.sqrt(zahl + 9.0);

    if (ergebnis > 0.0)
    {
        Console.println(ergebnis);
    }
}
```

Listen von Zahlen

Mit den Wrapper-Klassen kann man dynamische Datenstrukturen von Zahlen erzeugen:

```
public class ZahlenListe
{
    private List<Double> zl;
    ...
    public void beispiel()
    {
        // Objekt auf „klassische Art“ erzeugen und einfügen
        Double zahl = new Double(3.0);
        zl.append(zahl);

        // Automatische Umwandlung double → Double nutzen
        zl.append(4.0);
    }
}
```

Aufgabe

Implementiere die Klasse ZahlenListe mithilfe der ausgeteilten BlueJ-Vorlage.

Die Methode fülleFibonacci() füllt die Liste zl von Integer-Objekten mit den Werten der Fibonacci-Folge:

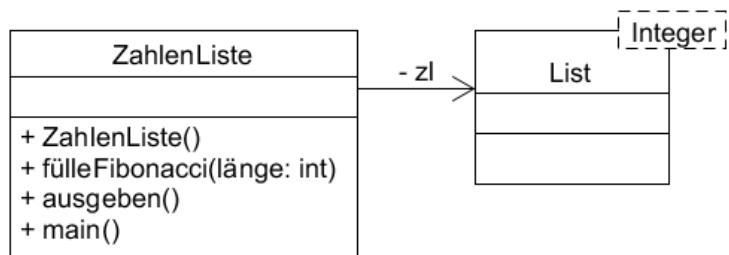
$f_1 = 1, f_2 = 1, f_3 = f_2 + f_1, f_4 = f_3 + f_2$, usw.

Allgemein: $f_n = f_{n-1} + f_{n-2}$.

Der Parameter länge gibt an, wie viele Elemente die Liste haben soll.

Die Methode ausgabe() gibt die Werte der Liste auf der Konsole aus.

Beachte, dass Integer maximal den Wert von f_{46} fassen kann. Für größere Werte müsstest du statt Integer die Klasse Long (und für Variablen den Datentyp long) verwenden. Auch hier ist bereits bei f_{92} Schluss. Für noch größere Werte gibt es die Klasse BigInteger¹, für die es dann allerdings keine automatische Umwandlung in einfache Variablen gibt.



¹ Hinweis zu BigInteger: die maximale Größe wird mit $2^{\text{Integer.MAX_VALUE}}$ angegeben. Da Integer meist als 32-Bit-Wert implementiert ist, wäre das ca. $2^{2^{\text{Mrd}}}$. Die größte mit BigInteger darstellbare Zahl würde allein also ca. 2 Mrd. Bit beanspruchen, das sind 250 Megabyte. Eine Liste von solch großen Zahlen würde schnell den gesamten Arbeitsspeicher (oder ggf. sogar die gesamte Festplatte) füllen, daher wird hier die Anzahl der Elemente der Liste durch den zur Verfügung stehenden Speicher begrenzt.