

**Teil**

**Objektorientierte Programmierung**

**Unterrichtseinheit 25**

**Klassen und Abstrakte Datentypen**

**Dr. Dietrich Boles**

- Abstrakter Datentyp
- Klassen als Abstrakte Datentypen
  - ADT-Klasse Bruch
  - Definitionen
  - ADT-Klasse Int
  - ADT-Klasse Stack
  - ADT-Klasse ArrayList
- Zusammenfassung

# „Abstrakter Datentyp“ (1)

```
class Bruch {  
    int zaehler = 1;  
    int nenner = 1;  
}
```

**ADT =**  
**(gekapselte) Datenstruktur**  
**+**  
**Funktionen auf der Datenstruktur**

```
class BruchRechnung {  
  
    static void init(Bruch bruch, int z, int n) {  
        bruch.zaehler = z;  
        bruch.nenner = n;  
        kuerzen(bruch);  
    }  
}
```

```
static void multTo(Bruch b1, Bruch b2) {  
    b1.zaehler *= b2.zaehler;  
    b1.nenner *= b2.nenner;  
    kuerzen(b1);  
}
```

```
static void addTo(Bruch b1, Bruch b2) {  
    b1.zaehler =  
        b2.nenner * b1.zaehler +  
        b1.nenner * b2.zaehler;  
    b1.nenner = b1.nenner * b2.nenner;  
    kuerzen(b1);  
}
```

```
static String getString(Bruch b) {  
    return b.zaehler + "/" + b.nenner;  
}
```

```
static void kuerzen(Bruch b) {  
    int ggt = ggT(b.zaehler, b.nenner);  
    b.zaehler /= ggt;  
    b.nenner /= ggt;  
}
```

```
static int ggT(int z1, int z2) {  
    if (z2 == 0) {  
        return z1;  
    } else {  
        return ggT(z2, z1 % z2);  
    }  
}
```

```
public static void main(String[] args) {
    int zaehler = IO.readInt("Bruch 1 (Zaehler): ");
    int nenner = IO.readInt("Bruch 1 (Nenner): ");
    Bruch b1 = new Bruch();
    init(b1, zaehler, nenner);
    IO.println("Bruch 1: " + getString(b1));

    zaehler = IO.readInt("Bruch 2 (Zaehler): ");
    nenner = IO.readInt("Bruch 2 (Nenner): ");
    Bruch b2 = new Bruch();
    init(b2, zaehler, nenner);
    IO.println("Bruch 2: " + getString(b2));

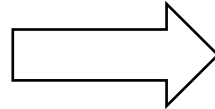
    multTo(b1, b2); // b1 = b1 * b2;
    IO.println("Bruch 1: " + getString(b1));
    IO.println("Bruch 2: " + getString(b2));

    addTo(b1, b2); // b1 = b1 + b2;
    IO.println("Bruch 1: " + getString(b1));
    IO.println("Bruch 2: " + getString(b2));
} }
```

- Abstrakter Datentyp = Datenstruktur + Funktionen (+ mehr)
- Klasse =  
(syntaktisches) Konzept, Datenstruktur und Funktionen zu einer Einheit zusammenzufassen

Bisher:

```
class X {  
    „Datenstruktur“  
}  
  
class XFunktionen {  
    „Funktionen für X“  
}
```



Nun:

```
class X {  
    „Datenstruktur“  
    +  
    „Funktionen“  
}
```

# Klassen als ADT / Motivationsbeispiel (1)

```
public static void main(String[] args) {  
    int zaehler = IO.readInt("Bruch 1 (Zaehler): ");  
    int nenner = IO.readInt("Bruch 1 (Nenner): ");  
    Bruch b1 = new Bruch();  
    init(b1, zaehler, nenner);                b1.init(zaehler, nenner);  
    IO.println("Bruch 1: " + getString(b1));    b1.getString()  
  
    zaehler = IO.readInt("Bruch 2 (Zaehler): ");  
    nenner = IO.readInt("Bruch 2 (Nenner): ");  
    Bruch b2 = new Bruch();  
    init(b2, zaehler, nenner);                b2.init(zaehler, nenner);  
    IO.println("Bruch 2: " + getString(b2));    b2.getString()  
  
    multTo(b1, b2);                            b1.multTo(b2);  
    IO.println("Bruch 1: " + getString(b1));    b1.getString()  
    IO.println("Bruch 2: " + getString(b2));    b2.getString()  
  
    addTo(b1, b2);                            b1.addTo(b2);  
    IO.println("Bruch 1: " + getString(b1));    b1.getString()  
    IO.println("Bruch 2: " + getString(b2));    b2.getString()  
} }
```



# Klassen als ADT / Motivationsbeispiel (2)

```
class Bruch {  
    // Datenstruktur  
    int zaehler;  
    int nenner;
```

- **this** ist das Objekt, für das die Methode aufgerufen wird.
- lediglich andere syntaktische Schreibweise

```
    // Funktionen auf der Datenstruktur = „Methoden“  
    // void init(int z, int n) {  
    Bruch(int z, int n) { // Konstruktor statt init  
        this.zaehler = z;  
        this.nenner = n;  
        this.kuerzen();  
    }
```

bisher:

```
static void init(Bruch this,  
                 int z, int n) {  
    this.zaehler = z;  
    this.nenner = n;  
    kuerzen(this);  
}
```

```
Bruch bruch1 = new Bruch();  
init(bruch, 4, 7); // bisher
```

```
Bruch bruch1 = new Bruch(4,7); // nun
```

# Klassen als ADT / Motivationsbeispiel (3)

```
void multTo(Bruch b2) {
    this.zaehler *= b2.zaehler;
    this.nenner *= b2.nenner;
    this.kuerzen();
}
static void multTo(Bruch b1, Bruch b2) {
    b1.zaehler *= b2.zaehler;
    b1.nenner *= b2.nenner;
    kuerzen(b1);
}
```

```
void addTo(Bruch b2) {
    this.zaehler = b2.nenner * this.zaehler +
                  this.nenner * b2.zaehler;
    this.nenner = this.nenner * b2.nenner;
    this.kuerzen();
}
static void addTo(Bruch b1, Bruch b2) {
    b1.zaehler = b2.nenner * b1.zaehler +
                  b1.nenner * b2.zaehler;
    b1.nenner = b1.nenner * b2.nenner;
    kuerzen(b1);
}
```

```
String getString() {
    return this.zaehler + "/" + this.nenner;
}

static String getString(Bruch b) {
    return b.zaehler + "/" + b.nenner;
}

void kuerzen() {
    int ggt = Bruch.ggT(this.zaehler, this.nenner);
    this.zaehler /= ggt;
    this.nenner /= ggt;
}

static void kuerzen(Bruch b) {
    int ggt = ggT(b.zaehler, b.nenner);
    b.zaehler /= ggt;
    b.nenner /= ggt;
}

static int ggT(int z1, int z2) {
    return (z2 == 0) ? z1 : Bruch.ggT(z2, z1 % z2);
}
```

# Klassen als ADT / Motivationsbeispiel (5)

```
// Testprogramm
public static void main(String[] args) {
    int zaehler = IO.readInt("Bruch 1 (Zaehler): ");
    int nenner = IO.readInt("Bruch 1 (Nenner): ");
    Bruch b1 = new Bruch(zaehler, nenner);
    IO.println("Bruch 1: " + b1.getString());

    zaehler = IO.readInt("Bruch 2 (Zaehler): ");
    nenner = IO.readInt("Bruch 2 (Nenner): ");
    Bruch b2 = new Bruch(zaehler, nenner);
    IO.println("Bruch 2: " + b2.getString());

    b1.multTo(b2);
    IO.println("Bruch 1: " + b1.getString());
    IO.println("Bruch 2: " + b2.getString());

    b1.addTo(b2);
    IO.println("Bruch 1: " + b1.getString());
    IO.println("Bruch 2: " + b2.getString());
} }
```

- Klasse = Zusammenfassung von Daten und darauf arbeitenden Funktionen zu einer (syntaktischen) Einheit
- Objekt = Instanz einer Klasse  
(= konkreter Verbund + Funktionen)

```
class Int { // Realisierung von int-Werten als Objekte
```

```
    // Datenstruktur  
    int wert;
```

```
    // Konstruktoren  
    Int() { this.wert = 0; }
```

```
    Int(int w) { this.wert = w; }
```

```
    // Methoden
```

```
    void add(Int obj2) {  
        this.wert += obj2.wert;  
    }
```

```
    void add(int wert) {  
        this.wert += wert;  
    }
```

```
String getString() {  
    return "" + this.wert;  
}
```

```
int intWert() {  
    return this.wert;  
}
```

```
// Testprogramm
```

```
public static void main(String[] args) {  
    int w1 = IO.readInt("Zahl: ");  
    Int i1 = new Int(w1);  
    Int i2 = new Int();  
    i1.add(i2);  
    i1.add(w1);  
    i1.add(i2.intWert());  
    IO.println("Ergebnis: " + i1.getString());  
} }
```

# ADT-Klasse Stack (1)

```
class Stack { // Protokoll
    Stack(int size) // Konstruktor
    void push(int value)
    int pop()
    boolean isFull()
    boolean isEmpty()
}
```

47
-843
56

```
// Hauptprogramm
public static void main(String[] args) {
    Stack stack = new Stack(IO.readInt("Stackgroesse: "));
    // Stack fuellen
    while (!stack.isFull())
        stack.push(IO.readInt("Zahl: "));

    // Stack leeren
    while (!stack.isEmpty())
        IO.println(stack.pop());
}
```



# ADT-Klasse Stack (2)

```
class Stack { // Implementierung
    // Datenstruktur
    int[] store; // zum Speichern von Daten
    int current; // aktueller Index
```

```
// Methoden
```

```
Stack(int size) { // Konstruktor
```

```
    this.store = new int[size];
```

```
    this.current = -1;
```

```
}
```

```
boolean isFull() {
```

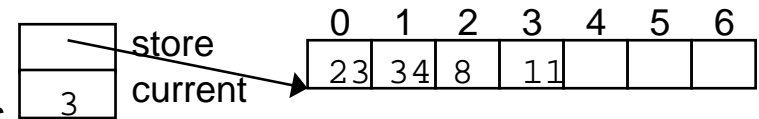
```
    return this.current == (this.store.length-1);
```

```
}
```

```
boolean isEmpty() {
```

```
    return this.current == -1;
```

```
}
```



## ADT-Klasse Stack (3)

```
void push(int value) {  
    this.store[++this.current] = value;  
}
```

```
int pop() {  
    return this.store[this.current--];  
}
```

```
// Testprogramm
```

```
public static void main(String[] args) {  
    Stack stack = new Stack(IO.readInt("Stackgroesse: "));  
    // Stack fuellen  
    while (!stack.isFull()) {  
        stack.push(IO.readInt("Zahl: "));  
    }  
    // Stack leeren  
    while (!stack.isEmpty()) {  
        IO.println(stack.pop());  
    } } }
```

- gesucht: Klasse ArrayList, die ein Array beliebiger Größe simuliert
- wichtig ist das Protokoll, nicht die interne Datenstruktur

```
class ArrayList {  
    void add(int value)  
    boolean isElem(int value)  
}  
  
public static void main(String[] args) { // Test  
    ArrayList l = new ArrayList();  
    int input = IO.readInt("Number: "); // Fuellen  
    while (input > 0) {  
        l.add(input);  
        input = IO.readInt("Number: ");  
    }  
    while (true) { // auf Liste arbeiten  
        int check = IO.readInt("Check: ");  
        if (l.isElem(check)) IO.println("Is Element");  
        else IO.println("Is NOT Element");  
    }  
}
```

# ADT-Klasse ArrayList / Lösung 1 (1)

```
class ListElem { // Hilfsklasse
    int value;    // Speicher fuer den Wert
    ListElem next; // Verweis auf naechstes Element

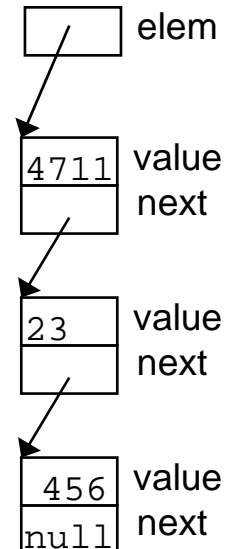
    ListElem(int v) { value = v; next = null; }
}
```

```
class ArrayList {

    // Datenstruktur
    ListElem elem; // verkettete Liste

    // Konstruktoren

    ArrayList() { // Default-Konstruktor
        this.elem = null;
    }
}
```



```
// Methoden
void add(int v) {
    if (this.elem == null) {
        this.elem = new ListElem(v);
    } else {
        ListElem help = this.elem;
        while (help.next != null)
            help = help.next;
        help.next = new ListElem(v);
    }
}

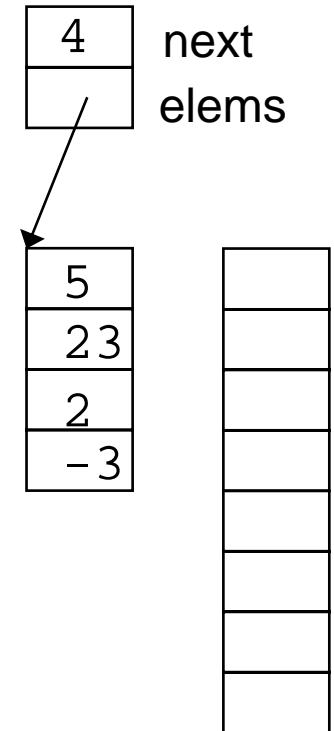
boolean isElem(int v) {
    ListElem help = this.elem;
    while (help != null) {
        if (help.value == v) return true;
        help = help.next;
    }
    return false;
}}
```

```
class ArrayList {
    int next;
    int[] elems; // "austauschbares" Array

    ArrayList() {
        this.elems = new int[4]; this.next = 0;
    }

    void add(int v) {
        if (this.next == this.elems.length) {
            int[] copy = new int[this.elems.length * 2];
            for (int i=0; i<this.elems.length; i++)
                copy[i] = this.elems[i];
            this.elems = copy; // Austausch des Arrays
        }
        this.elems[this.next++] = v;
    }

    boolean isElem(int v) {
        for (int i=0; i<this.next; i++)
            if (this.elems[i] == v) return true;
        return false;
    }
}
```



- ADT: (gekapselte) Datenstruktur + Funktionen auf der Datenstruktur
- Klasse: (syntaktisches) Konzept, Datenstruktur und Funktionen zu einer Einheit zusammenzufassen
- Objekt: Instanz einer Klasse (= konkreter Verbund + Funktionen)