

Globalübung 29.11.2005

- 1 Wrapper-Klassen
- 2 Überladen von Methoden und Sichtbarkeiten
- 3 Rekursion

Überblick

- 1 Wrapper-Klassen
- 2 Überladen von Methoden und Sichtbarkeiten
- 3 Rekursion

Wrapper-Klassen (Integer, Double, Boolean, ...)

- Kapseln einen primitiven Datentyp (int, double, boolean, ...)
- Vorteile:
 - Viele nützliche Bibliotheksfunktionen (Eingabe, Ausgabe, Konvertierung, ...)
 - (Später) Containerklassen können durch Wrapper-Klassen auch primitive Datentypen aufnehmen.
- Nachteile:
 - Kein direktes Rechnen mehr möglich ($x + y$ nicht möglich)
 - Vergleiche mit "==" gefährlich, im Gegensatz zu primitiv-Datentypen.

Wrapper-Klassen (Integer, Double, Boolean, ...)

- Kapseln einen primitiven Datentyp (int, double, boolean, ...)
- Vorteile:
 - Viele nützliche Bibliotheksfunktionen (Eingabe, Ausgabe, Konvertierung, ...)
 - (Später) Containerklassen können durch Wrapper-Klassen auch primitive Datentypen aufnehmen.
- Nachteile:
 - Kein direktes Rechnen mehr möglich ($x + y$ nicht möglich)
 - Vergleiche mit "==" gefährlich, im Gegensatz zu primitiv-Datentypen.

Überblick

- 1 Wrapper-Klassen
- 2 Überladen von Methoden und Sichtbarkeiten
- 3 Rekursion

Überladen von Methoden und Sichtbarkeiten

- Überladen: Methode f (oder Konstruktor) kommt mehrfach vor

```
int f(int x) { ... }  
String f(double x) { ... }  
void f(String x) { ... }  
void f(String x, int y) { ... }
```

- (eindeutige) Auswahl anhand der *Eingabe*-Parameter

```
f(5), f(5.0), f("5"), f(null, 2)
```

- Auswahl anhand der Rückgabe *nicht* möglich

```
int f(int x) { ... }  
String f(int x) { ... }
```

```
int x = f(3); String y = f(3);
```

Überladen von Methoden und Sichtbarkeiten

- Überladen: Methode f (oder Konstruktor) kommt mehrfach vor

```
int f(int x) { ... }  
String f(double x) { ... }  
void f(String x) { ... }  
void f(String x, int y) { ... }
```

- (eindeutige) Auswahl anhand der *Eingabe*-Parameter

```
f(5), f(5.0), f("5"), f(null, 2)
```

- Auswahl anhand der Rückgabe *nicht* möglich

```
int f(int x) { ... }  
String f(int x) { ... }
```

```
int x = f(3); String y = f(3);
```

Überladen von Methoden und Sichtbarkeiten

- Überladen: Methode f (oder Konstruktor) kommt mehrfach vor

```
int f(int x) { ... }  
String f(double x) { ... }  
void f(String x) { ... }  
void f(String x, int y) { ... }
```

- (eindeutige) Auswahl anhand der *Eingabe*-Parameter

```
f(5), f(5.0), f("5"), f(null, 2)
```

- Auswahl anhand der Rückgabe *nicht* möglich

```
int f(int x) { ... }  
String f(int x) { ... }
```

```
int x = f(3); String y = f(3);
```


Überblick

- 1 Wrapper-Klassen
- 2 Überladen von Methoden und Sichtbarkeiten
- 3 Rekursion**

Rekursion

Rekursion: Abarbeitung einer Methode f führt zum erneuten Aufruf von f .

- Direkte Rekursion: Funktion ruft sich selbst auf

```
int fac(int x) { return x==0 ? 1 : x * fac(x-1); }
```

- Verschränkte Rekursion: Gegensatz zu direkter Rekursion

```
boolean even(int x) {  
    return x==0 ? true : odd(x-1);  
}  
  
boolean odd(int x) {  
    return x==0 ? false : even(x-1);  
}
```

Rekursion

- Divide & Conquer-Strategie:
Führe große Probleme auf kleinere zurück
- Erlaubt komplexere Ablaufstrukturen als Schleifen,
aber etwas langsamer als Schleifen.
- Leichte Implementierung und Verifikation *induktiver* Algorithmen

Rekursion

- **Lineare Rekursion:** Nur ein rekursiver Aufruf

```
int fac(int x) { return x==0 ? 1 : x * fac(x-1); }
```

- **Nicht-Lineare Rekursion:** Mehrere rekursive Aufrufe

```
int fib(int x) {  
    if (x <= 1) return 1;  
    return fib(x-1) + fib(x-2);  
}
```

Gefahr von exponentieller Laufzeit
(bei überlappenden Aufrufen)