Objekte nutzen und testen

Andreas Zeller

Vgl. Kapitel 5-6 im BlueJ-Buch.

Dienstag, 8. Mai: keine Vorlesung!

Konzepte

(Wiederholung vom Freitag)

- Sammlungen
  - flexibler Größe: ArrayList
  - fester Größe: Array
- Iteratoren
- null
- Schleifen
  - for each
  - while
  - for
  - break
- Überlauf

Konzepte

(Wiederholung vom Freitag)

- Sammlungen
  - flexibler Größe: ArrayList
  - fester Größe: Array
- Iteratoren
- null
- Schleifen
  - for each
  - while
  - for
  - break
- Überlauf

Schleifen

for-each	`for (ElementType element: collection) {`
	`  loop body`
	`}`

while	`while (condition) {`
	`  loop body`
	`}`

for	`for (init; condition; post-body) {`
	`  loop body`
	`}`

for-each, while, for

do-while bleibt außen vor – zu fehleranfällig

Schleifen

for-each	um *alle Elemente* einer *Sammlung* zu durchlaufen
while	für eine *vorher unbekannte* Anzahl Schleifendurchläufe
for	für eine *vorher bekannte* Anzahl Schleifendurchläufe

for-each, while, for

Konzepte

(Wiederholung vom Freitag)

- Sammlungen
  - flexibler Größe: ArrayList
  - fester Größe: Array
- Iteratoren
- null
- Schleifen
  - for each
  - while
  - for
  - break
- Überlauf

Iteratoren

```
import java.util.ArrayList;

class Notes { ...
    public void listNotes() {
        for (String note : notes) {
            System.out.println(note);
        }
    }
}
```


Iteratoren

```
import java.util.ArrayList;
import java.util.Iterator;

class Notes { ...
    public void listNotes() {
        Iterator<String> it = notes.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

Kontrollstrukturen

Java
```
if (amount >= 0)
    balance = balance + amount;
```

Perl
```
if ($amount >= 0)
    $balance = $balance + $amount;
```


Kontrollstrukturen

Java
```
if (amount >= 0)
    balance = balance + amount;
```

Perl
```
unless ($amount < 0)
    $balance = $balance + $amount;
```


Kontrollstrukturen

Java
```
if (amount >= 0)
    balance = balance + amount;
```

Perl
```
do {
    $balance = $balance + $amount;
} if ($amount >= 0);
```

Kontrollstrukturen

Java

```
if (amount >= 0)
    balance = balance + amount;
```

Perl

```
do {
    $balance = $balance + $amount;
} unless ($amount < 0);
```


Kontrollstrukturen

Java

```
if (amount >= 0)
    balance = balance + amount;
```

Perl

```
$balance = $balance + $amount
    unless ($amount < 0);
```


Kontrollstrukturen

Java

```
int x[] = {1, 2, 3, 4};
while (x[i] < 4)
    i++;
```

Perl

```
@x = (1, 2, 3, 4);
while ($x[i] < 4) {
    $i++;
}
```


Wo ist der Fehler?

Kontrollstrukturen

Java

```
int x[] = {1, 2, 3, 4};
while (x[i] < 4)
    i++;
```

Perl

```
@x = (1, 2, 3, 4);
while ($x[$i] < 4) {
    $i++;
}
```


Kontrollstrukturen

Java

```
int x[] = {1, 2, 3, 4};
while (x[i] < 4)
    i++;
```

Perl

```
@x = (1, 2, 3, 4);
until ($x[$i] >= 4) {
    $i++;
}
```


Kontrollstrukturen

Java

```
int x[] = {1, 2, 3, 4};
do {
    i++;
} while (x[i] < 4)
```

Perl

```
@x = (1, 2, 3, 4);
do {
    $i++;
} while ($x[$i] < 4);
```

Kontrollstrukturen

Perl

```
BAR:
    goto (“FOO”);
FOO:
    goto (“BAR”);
```


Kontrollstrukturen

Perl

```
print “Where do you want to go today? ”;
$line = <STDIN>;
goto ($line);
```


Vgl. Kapitel 5-6 im BlueJ-Buch.

Objekte nutzen und testen

Andreas Zeller

Objekte nutzen und testen

Andreas Zeller

Vgl. Kapitel 5-6 im BlueJ-Buch.

Tech Support

Eliza

```
"D:\Documents                    \Gonzales Cenelia\Desktop\Chatterbot Eliza 2.0\Debug\Eliza.exe"
HI USER, CAN YOU PLEASE TELL ME YOUR NAME?
>my name is Gonzales.
GLAD TO MEET YOU GONZALES.
>ok Eliza!
PLEASE BE MORE SPECIFIC.
>what do you mean by that?
SORRY I CAN'T PROVIDE MORE EXPLANATION ON THIS.
>why not?
WHY ARE YOU ASKING ME THIS?
>i dont know.
HOW COME YOU DONT KNOW?
>i'm just kidding.
WHAT MAKES YOU THINK THAT YOU ARE JUST KIDDING?
>well, because it is true.
SO YOU ARE TELLING ME THE REASON.
>yes, exactly.
YOU SOUND VERY OPTIMISTIC.
>thank you for the compliment!
DONT MENTION IT.
>ok.
PLEASE BE MORE SPECIFIC.
>i will try to be more specific next time.
DO YOU LIKE TALKING ABOUT YOURSELF?
>so, you think that i was talking about myself.
YOU ARE NOT REALLY TALKING ABOUT ME, ARE YOU?
>well, not really.
```

Demo: Tech Support

Map

(Abbildung)

```
import java.util.HashMap;

HashMap<String, String> phoneBook =
    new HashMap<String, String>();

phoneBook.put(“Knut”, “+49 681 302-64011”);
phoneBook.put(“Dave”, “+1 555 425-9000”);

String number = phoneBook.get(“Knut”);
```


Map

(Abbildung)

```
import java.util.HashMap;

HashMap<String, String> phoneBook =
    new HashMap<String, String>();

phoneBook.put(“Knut”, “+49 681 302-64011”);
phoneBook.put(“Dave”, “+1 555 425-9000”);

String number = phoneBook.get(“Knut”);
```

Set

(Menge)

```
import java.util.HashSet;

HashSet<String> mySet = new HashSet<String>();

mySet.add(“one”);
mySet.add(“two”);
mySet.add(“three”);

for (String item: mySet) {
    ...
}
```


Set

(Menge)

```
import java.util.HashSet;

HashSet<String> mySet = new HashSet<String>();

mySet.add(“one”);
mySet.add(“two”);
mySet.add(“three”);

for (String item: mySet) {
    ...
}
```

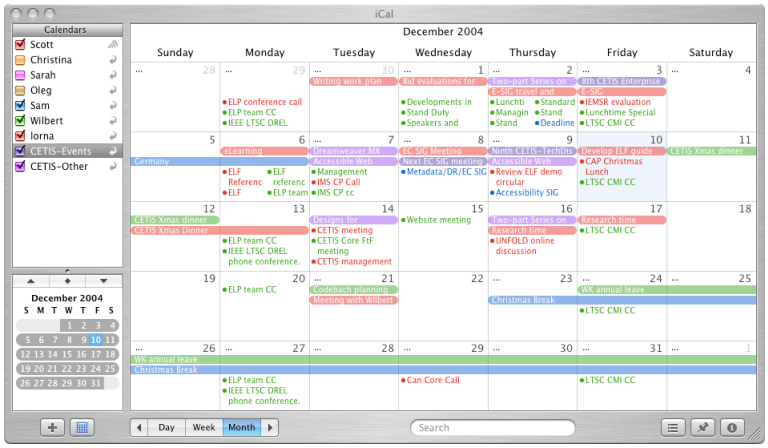

Konzepte

- Java-Bibliothek
- Schnittstelle
- Implementierung
- Unveränderlich
- Abbildung (Map)
- Menge (Set)

Objekte nutzen und testen

Andreas Zeller

Vgl. Kapitel 5-6 im BlueJ-Buch.



Demo: Kalender

Testen

- Testen ist das Ausführen eines Programms *mit dem Ziel, Fehler zu finden*
- Tests konzentrieren sich daher
  - auf erwartete *Funktionen* (positiver Test)
  - auf erwartetes *Fehlschlagen* (negativer Test)
- Tests sollten möglichst viele Funktionen und Fehlschlagen *abdecken*

JUnit

- Testwerkzeug für Java-Programme
- Test-Methoden prüfen Ergebnis mit *Zusicherungen*

```
public class HelloWorld extends TestCase
 {
   public void testMultiplication()
   {
     // Testing if 3*2=6:
     assertEquals (6, 3*2);
   }
 }
```


Zusicherungen in JUnit

- Allgemein: `assert(x)` sichert zu, dass `x` gilt
- Im Fall ¬`x` schlägt der Test fehl
  - `assertTrue(2 + 2 == 4)`
  - `assertFalse(size < 0)`
  - `assertEquals(numberOfElements(), 0)`
  - `assertNull(currentObject())`

Testklasse

- Eine Testklasse umfasst eine Menge von Test-Methoden
- Jede Test-Methode hat die Signatur

```
public void testXXX() {}
```

  und enthält wenigstens eine Zusicherung
- Mit "Test all" werden alle Test-Methoden ausgeführt

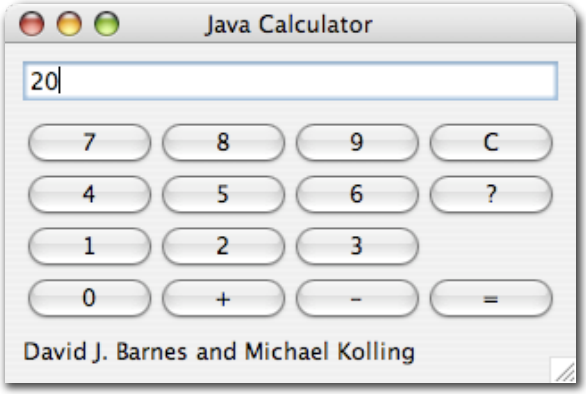
Testgerüst

(Fixture)

- Viele Tests erstellen ähnliche Objekte
- Ein *Testgerüst* stellt eine einheitliche Umgebung für alle Tests bereit
- Realisiert in der setUp()-Methode
- setUp() wird vor jedem Test aufgerufen

Dienstag, 8. Mai: keine Vorlesung!

Fehlersuche



Fehlersuche

- Manuelle Ausführung
- Zusicherungen
- Protokollierung (Logs)
- Systematische Fehlersuche

Konzepte

- Java-Bibliothek
- Schnittstelle
- Implementierung
- Unveränderliche Objekte
- Abbildung (Map)
- Menge (Set)
- Testen
- Fehlersuche
- Positives/negatives Testen
- Zusicherung
- Testgerüst (Fixture)
- Manuelle Ausführung