Java für Fortgeschrittene

Programmierpraktikum

Melanie Dietz und Michael Petter

TU-München

Sommersemester 2006

Worum gehts hier?

Kompetenz sammeln

- Sich sicherer in Java bewegen
- Alle Sprachfeatures sicher beherrschen
- Praxiserfahrung sammeln und Routine gewinnen
- Technologien für Java kennenlernen
- Teamarbeit
- Softwareentwicklungsprozesse kennenlernen
- Größere Projekte kennenlernen
- Spass am Programmieren

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Worum gehts hier?

Themenbereiche

- Java verstehen
- Fortgeschrittene Konzepte
- Softwaretechnik
- Nebenläufigkeit
- Rechnernetze
- Softwaresicherheit
- Datenbanken

Organisation

Zeitrahmen

- Praktikumstreffen Dienstag von 14:00 bis 16:00
- Hausaufgaben im Umfang von etwa 5 Stunden
- Sprechstunde immer Donnerstags von 9:30 bis 11:30
- Abmeldung am Vortag per Email

Projektorganisation

Gruppen

- Teamarbeit zu zweit ⇒ Abstimmung
- Hausaufgabenabgabe zu zweit per Codeverwaltung ⇒ SVN
- bei Bedarf Rechnerlogin
- Korrekturen/Bewertungen im Homeverzeichnis
- Betreuung persönlich oder per Email
  `praktikumsleitung@mailseidl.informatik.tu-muenchen.de`

Hausaufgabenannahme

- Aktuelle Version richtig als Abgabe Kennzeichnen!
- Nur fehlerfrei **mit ANT** compilierende Hausaufgaben werden korrigiert
- Eine Korrektur automatisch ab Mo 12:00
- Eine Nachbesserung nach Emailankündigung möglich

Projektorganisation

Gruppen

- Teamarbeit zu zweit ⇒ Abstimmung
- Hausaufgabenabgabe zu zweit per Codeverwaltung ⇒ SVN
- bei Bedarf Rechnerlogin
- Korrekturen/Bewertungen im Homeverzeichnis
- Betreuung persönlich oder per Email
  `praktikumsleitung@mailseidl.informatik.tu-muenchen.de`

Hausaufgabenannahme

- Aktuelle Version richtig als Abgabe Kennzeichnen!
- Nur fehlerfrei **mit ANT** compilierende Hausaufgaben werden korrigiert
- Eine Korrektur automatisch ab Mo 12:00
- Eine Nachbesserung nach Emailankündigung möglich

Projektorganisation

Projektbuch

- Problemspezifikation $\Rightarrow$ Was habe ich verstanden?
- Skizzen zur Aufgabe/Lösung
- Lösungsansätze (auch verworfene)
- Tagesprotokoll $\Rightarrow$ Was habe ich erreicht?
- Links zu Resourcen/Anleitungen/Dokumentationen
- Testbuchhaltung: Sonderfälle/Tests/Ergebnissoll und -haben
- Korrekturanmerkungen und Reaktionen
- Projektabschluss: Was geht, was ist offen?

Vorraussetzungen

(Fast) Alles ist Grundwissen :)

- Prof. Seidl: *Einführung in die Informatik 1 WS0405*
- Literaturhinweise auf dem Aufgabenblatt
- Jetzt: kurze Wiederholung
  - Objektreferenzen
  - Vererbung und Polymorphismus
  - Threads
  - Generische Klassen

Referenzen in Java

Grundlagen

Erzeugen eines Datumsobjektes: `Date today = new Date();`

- Deklaration eines Objektes, d. h. Bereitstellen der Objektreferenz (`Date today`)
- Erzeugen einer Instanz ($\rightarrow$ Objektadresse) mittels `new`
- Zuweisen der zurückgegebenen Adresse an die Referenz mittels =

Übergabe von Argumenten an Methoden

- call-by-reference: Objektreferenz wird übergeben (In Java nicht möglich!)
- call-by-value: Argument wird kopiert und Argumentkopie wird übergeben.

Was passiert also beim Übergeben von Referenzen in Java?

Vererbung

Eine Unterklasse

- ist eine Spezialisierung der Oberklasse
- erbt alle Attribute und Methoden der Oberklasse
- besitzt das Schlüsselwort *extends*

Einfachvererbung bedeutet,

- dass jede Klasse maximal eine Oberklasse besitzen darf
- dass die Klassenhierarchie eine Baumstruktur ergibt
- ist – im Gegensatz zur Mehrfachvererbung – in Java erlaubt

Eine abstrakte Klasse

- kann nicht instantiiert werden
- definiert Gemeinsamkeiten einer Gruppe von Unterklassen
- besitzt das Schlüsselwort *abstract*

Polymorphismus

Überschriebene Methoden

- werden in der Unterklasse mit identischer Signatur implementiert
- sind für die Unterklasse verdeckt
- können nur durch die Notation *super* aufgerufen werden.

Identisches gilt für verdeckte Attribute.

Polymorphismus

- Methodenaufrufe können differenziert behandelt werden
- umgeht die Implementierung von Typabfragen und ermöglicht dadurch späteres Hinzufügen weiterer Klassen
- kann nur bei Vererbung eingesetzt werden

Threads

Ein Thread ist

- eine Aktivität, die zu anderen Aktivitäten nebenläufig sein kann
- jedes Programm, das von der JVM ausgeführt wird
- der Name einer Java Klasse (`java.lang.Thread`) :-)

Monitore: Das Schlüsselwort synchronized

- kann sowohl bei Methoden als auch bei Objektadressen verwendet werden
- beschränkt den Zugriff auf eine Resource: Bei einem Aufruf tritt das Objekt automatisch in den eigenen Monitor ein
- kann bei falscher Verwendung zu einer Verklemmung führen


Ich will schreiben!
Geben Sie mir Papier!
Geben Sie mir den Stift!

Generische Klassen

Allgemein

- erlauben das Abstrahieren über Datentypen - und dadurch die Trennung von Algorithmen und Daten
- erleichtern das Arbeiten mit *Containern*

Aus der Vorlesung bekannt:

- Klassen: `Class<T>`
- Methoden:
  `<T> void fromAToB(T[] a, Collection<T> b);`

Shape, Rectangle und Circle

Example

```
public abstract class Shape {
 public abstract void draw(Canvas c);
}

public class Circle extends Shape {
 private int x, y, radius;
 public void draw(Canvas c) { ... }
}

public class Rectangle extends Shape {
 private int x, y, width, height;
 public void draw(Canvas c) { ... }
}
```

Die Klasse Canvas

Example

```
public class Canvas {
 public void drawAll(List<Shape> shapes) {
  for (Shape s: shapes) { s.draw(this);
 }
}
```

Dieser Code *funktioniert nicht*, da weder `List<Rectangle>`, noch `List<Circle>` Unterklassen von `List<Shape>` sind.

Example

```
public void drawAll(List<? extends Shape> shapes)
{
 for (Shape s: shapes) { s.draw(this);
}
```