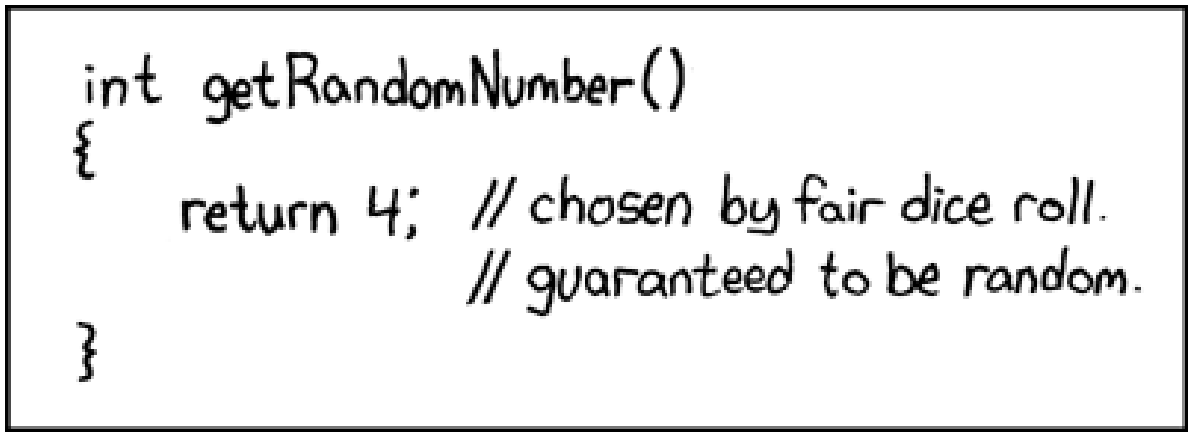
Universität Stuttgart
MINT-Kolleg Baden-Württemberg

Methoden

Vorkurs Informatik

Universität Stuttgart
Karlsruher Institut für Technologie
F. Schweiner
MINT-KOLLEG BADEN-WÜRTTEMBERG

Getting started ...



©Randall Munroe, http://xkcd.com/221/

Outline

(nach Folien von L. Vettin, V. Weidler, W. Kessler)

Outline

Rückblick

- Was haben wir bisher gelernt?
  - Grundlegender Aufbau eines Programmes
  - Ausgabe auf der Konsole mit `System.out.println(...)`
  - Variablen und Datentypen
    (`int`, `double`, `boolean`, `char`, `String`, `...`)
  - Fallunterscheidungen `if`, `if-else`
  - Schleifen `for`, `while`, `...`

Ein einfaches Java-Programm

- Unser Javaprogramm hatte bisher diese Struktur:

```
public class Test {

   public static void main (String[] args) {

      // hier Anweisungen einfügen

   }
}
```

- Ein Programm besteht aus Anweisungen (Befehlen).
- Die Anweisungen stehen immer zwischen den geschweiften Klammern hinter `main`.
- Hinter jeder Anweisung steht ein Semikolon.
- Die Auswertung eines Programms geschieht Zeile für Zeile.

Problemstellung

- Wir möchten $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$ für verschiedene Zahlen ausrechnen.

```
public class Test {

   public static void main (String[] args) {

      int result=1;
      for( int i=1; i<=5; i++){ result = result*i; }
      System.out.println("5! is: " + result);

      result=1;
      for( int i=1; i<=8; i++){ result = result*i; }
      System.out.println("8! is: " + result);

      ... // usw
   }
}
```

Funktionen/Methoden

- Bei sehr vielen verschiedenen Zahlen wird der Code sehr groß.
- Das muss doch einfacher gehen!

Funktionen/Methoden

- Bei sehr vielen verschiedenen Zahlen wird der Code sehr groß.
- Das muss doch einfacher gehen!
- Müssen dieselben Anweisungen immer wieder (nur mit unterschiedlichen Werten) durchgeführt werden, kann man den Programmteil auch (in eine eigene Klasse) auslagern.
- Für das Problem mit $n!$ sieht das wie folgt aus ...

Funktionen/Methoden

```java
public class Calculator {

   public int factorial (int n) {
      int result=1;
      for( int i=1; i<=n; i++){ result = result*i; }
      return result;
   }

}
```

```java
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();

      System.out.println("5! is: " + calc.factorial(5));
      System.out.println("8! is: " + calc.factorial(8));
      ... // usw
   }
}
```

Funktionen/Methoden

- Bei dem ausgelagerten Programmteil spricht man von einer Funktion oder Methode

Funktionen/Methoden

- Bei dem ausgelagerten Programmteil spricht man von einer Funktion oder Methode
- Funktionen stehen in einer eigenen Klasse!

Funktionen/Methoden

- Bei dem ausgelagerten Programmteil spricht man von einer Funktion oder Methode
- Funktionen stehen in einer eigenen Klasse!
- Funktionen werden ganz allgemein geschrieben (hier für beliebiges n) ...

Funktionen/Methoden

- Bei dem ausgelagerten Programmteil spricht man von einer Funktion oder Methode
- Funktionen stehen in einer eigenen Klasse!
- Funktionen werden ganz allgemein geschrieben (hier für beliebiges n) ...

```
public class Calculator {

   public int factorial (int n) {
      int result=1;
      for( int i=1; i<=n; i++){ result = result*i; }
      return result;
   }

}
```

Funktionen/Methoden

- Der Funktionsaufruf geschieht innerhalb der `main`-Funktion.
- Erst beim Funktionsaufruf werden konkrete Zahlenwerte eingesetzt!
- Funktionen können Zahlen, Text, . . . zurückliefern.).

```
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();

      System.out.println("5! ist: " + calc.factorial(5));
      System.out.println("8! ist: " + calc.factorial(8));
      ... // usw
   }
}
```

- Um die Funktion aufrufen zu können, benötigen wir ein Objekt (hier: `calc`) der Klasse, in der die Funktion steht (hier: `Calculator`)

Funktionen/Methoden

Wir benötigen in Zukunft also zwei Klassen!

```
public class Calculator {

   public int factorial (int n) {
      int result=1;
      for( int i=1; i<=n; i++){ result = result*i; }
      return result;
   }

}
```

```
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();

      System.out.println("5! is: " + calc.factorial(5));
      System.out.println("8! is: " + calc.factorial(8));
      ... // usw
   }
}
```

Rückgabetypen

Rückgabetyp

Der Rückgabetyp einer Methode gibt an, was diese Methode beim Aufruf zurückliefert.

Rückgabetypen

Rückgabetyp

Der Rückgabetyp einer Methode gibt an, was diese Methode beim Aufruf zurückliefert.

- Der Rückgabetyp steht vor dem Methodennamen, z. B. `int factorial`.
- Vor dem Rückgabetyp sollte noch `public` stehen. Wir werden später klären, was das bedeutet.
- Jede Methode muss einen Rückgabetyp haben.
- Als Rückgabetypen werden Datentypen angegeben.
- Der spezielle Rückgabetyp `void` wird verwendet, wenn eine Methode nichts zurückgibt, sondern nur eine Aktion ausführt.

Rückgabetyp vs. Rückgabewert

- Der Rückgabe**typ** einer Methode gibt an, von welchem Datentyp das zu erwartende result des Methodenaufrufs ist.
- Wenn eine Methode aufgerufen wird, wird als Ergebnis dieses Aufrufs ein spezifischer Rückgabe**wert** geliefert, der den angegebenen Datentyp hat.
- Beispiel:
  - `int factorial()` hat als Rückgabetyp `int`.
  - Bei einem Aufruf bekommt man als Rückgabewert eine ganze Zahl, z. B. 1, 24, 120.
- Welcher Wert zurückgegeben werden soll, steht hinter dem Befehl `return`.
- Ist der Rückgabetyp `void`, darf es kein `return <Wert>;` geben.

Rückgabewerte

```java
public class Calculator {

   public int factorial (int n) { // factorial gibt einen int zurueck
      int result=1;
      for( int i=1; i<=n; i++){ result = result*i; }
      return result; // Rueckgabewert
   }

}
```

```java
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();

      // factorial gibt einen int zurueck
      int x = calc.factorial(5);
      System.out.println("5! is: " + x);
   }
}
```

Rückgabewerte

- Alles nach `return <Wert>;` wird ignoriert!

```
public class Calculator {

   int add(int x, int y) {
      return x + y;
      System.out.println("Hello");
   }

}
```

```
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();
      System.out.println(calc.add(1,2));
   }
}
```

- Ausgabe:

```
3
```

Rückgabewerte

```
public class Calculator {

   int add(int x, int y) {
      System.out.println("Hello");
      return x + y;
   }

}
```

```
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();
      System.out.println(calc.add(1,2));
   }
}
```

- Ausgabe:

```
Hello
3
```

Rückgabewert

- Man beachte einen wichtigen Unterschied:
  - Eine Funktion gibt mit `return` einen Wert zurück
  - Dieser Wert kann dann in der `main`-Methode weiterverwendet werden.
  - Der Befehl `System.out.println(...)` gibt etwas auf der Konsole aus
- Folgender Code macht also etwas komplett anderes:

```
public class Calculator {

   public void factorial (int n) {
      int result=1;
      for( int i=1; i<=n; i++){ result = result*i; }
      System.out.println(result); // keine Rueckgabe!
   }

}
```

Anderes Beispiel

- Hier wird mit dem result der Methode noch weitergerechnet:

```
public class Calculator {

   public double min(double a, double b) {
      if(a<b){ return a;}
      else   { return b;}
   }

}
```

```
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();

      double minimum = calc.min(5,8);
      minimum = minimum / 2;
      System.out.println(minimum);
   }
}
```

Parameter

Parameter

Ein Parameter ist ein Mechanismus, um einer Methode zusätzliche Daten zu übergeben. Parameter haben einen Typ und einen Namen.

Parameter

Parameter

Ein Parameter ist ein Mechanismus, um einer Methode zusätzliche Daten zu übergeben. Parameter haben einen Typ und einen Namen.

- Die Parameterliste steht in den Klammern nach dem Methodennamen, z. B. `int factorial(int n)`.
- Die Parameterliste kann leer sein, d. h. eine Methode muss keine Parameter haben.
- Beim Aufruf der Methode muss für jeden Parameter ein passender Wert angegeben werden (Argument), z. B. die Zahl 3 für `n` beim Aufruf von `factorial`.

Parameter

- Innerhalb einer Methode kann auf die Parameter zugegriffen werden wie auf normale Variablen. Die Werte der Parameter sind die Argumente, die beim Methodenaufruf übergeben werden.

```
public class Calculator {

   public void sayHelloTo(String name, int times) {
      for (int i=0; i<times; i++) {
         System.out.println("Hello " + name);
      }
   }
}
```

```
public class Test {

   public static void main (String[] args) {

      Calculator calc = new Calculator();
      calc.sayHelloTo("Anna",2);
   }
}
```

Parameter vs. Argument

- Die **Parameter** einer Methode geben an, wie viele Werte von welchem Datentyp und in welcher Reihenfolge man beim Methodenaufruf an die Methode übergeben muss.
- Die **Argumente** einer Methode sind die beim Methodenaufruf konkret verwendeten Werte. Anzahl und Typ der Argumente muss mit den als Parameter erwarteten Werten übereinstimmen.
- Beispiel:
  - `void sayHelloTo(String name, int times)` hat zwei Parameter:
    - `name` vom Typ `String`
    - `times` vom Typ `int`
  - Beim Aufruf `calc.sayHelloTo("Anna",2)` werden zwei Argumente übergeben:
    - `"Anna"` an den Parameter `name`
    - `2` an den Parameter `times`

Methodendefinition vs. Methodenaufruf

- Definition der Methoden (mit Rückgabetyp, Parametern, Code):

```
public class Calculator {

    public void sayHelloTo(String name, int times) {
       for (int i=0; i<times; i++) {
          System.out.println("Hello " + name);
       }
    }

    public int add(int x, int y) {
       return x+y;
    }
}
```

- Aufrufe der Methoden (mit Argumenten, Rückgabewert):

```
calc.sayHelloTo("Anna",2);
int sum = calc.add(3,8);
```

Methoden

- In einer Methode kann beliebiger Code stehen.
- Methoden können sich gegenseitig aufrufen.

```
public void sayGoodbye() {
   System.out.println("Goodbye!");
}

public sayHelloTo(String name, int times) {
   for (int i=0; i<times; i++) {
      System.out.println("Hello " + name);
   }
   this.sayGoodbye(); // Aufruf der anderen Methode
}
```

- Ruft man eine Methode aus derselben Klasse auf, schreibt man `this` davor.

Methodensignatur

Methodensignatur

Die Spezifikation einer Methode, die Auskunft über ihren Namen, ihren Rückgabetyp und ihre Parameter gibt, heißt Signatur.

Methodensignatur

Methodensignatur

Die Spezifikation einer Methode, die Auskunft über ihren Namen, ihren Rückgabetyp und ihre Parameter gibt, heißt Signatur.

Beispiele:

- `int factorial(int n)`

Methodensignatur

Methodensignatur

Die Spezifikation einer Methode, die Auskunft über ihren Namen, ihren Rückgabetyp und ihre Parameter gibt, heißt Signatur.

Beispiele:

- `int factorial(int n)`
  - *Name*: `factorial`
  - *Parameter*: `int n`
  - *Rückgabetyp*: `int`

Methodensignatur

Methodensignatur

Die Spezifikation einer Methode, die Auskunft über ihren Namen, ihren Rückgabetyp und ihre Parameter gibt, heißt Signatur.

Beispiele:

- `int factorial(int n)`
  - *Name*: `factorial`
  - *Parameter*: `int n`
  - *Rückgabetyp*: `int`
- `double min(double a, double b)`

Methodensignatur

Methodensignatur

Die Spezifikation einer Methode, die Auskunft über ihren Namen, ihren Rückgabetyp und ihre Parameter gibt, heißt Signatur.

Beispiele:

- `int factorial(int n)`
  - *Name*: `factorial`
  - *Parameter*: `int n`
  - *Rückgabetyp*: `int`
- `double min(double a, double b)`
  - *Name*: `min`
  - *Parameter*: `double a, double b`
  - *Rückgabetyp*: `double`

Quiz: Methoden

Welche Aussagen sind korrekt?

- Es gibt Methoden ohne Rückgabetyp.
- Es gibt Methoden ohne Rückgabewert.
- Bei Parametern spielt die Reihenfolge keine Rolle.
- Jede Methode hat Parameter, aber sie müssen nicht immer angegeben werden.
- Parameter einer Methode haben einen festen Datentyp.
- Beim Aufruf einer Methode, die Parameter hat, müssen Argumente angegeben werden, die konkrete Werte für diese Parameter liefern.

Quiz: Methoden

Welche Aussagen sind korrekt?

- Es gibt Methoden ohne Rückgabetyp. (falsch, 'void' ist ein spezieller Typ)
- Es gibt Methoden ohne Rückgabewert. (korrekt, jene mit 'void')
- Bei Parametern spielt die Reihenfolge keine Rolle. (falsch)
- Jede Methode hat Parameter, aber sie müssen nicht immer angegeben werden. (falsch)
- Parameter einer Methode haben einen festen Datentyp. (korrekt)
- Beim Aufruf einer Methode, die Parameter hat, müssen Argumente angegeben werden, die konkrete Werte für diese Parameter liefern. (korrekt)

Quiz: Eigene Methoden

```
public int getFactorial(String name, int n, int f) {
   int fak = 1;
   for (int i=1; i<n; i++) {
      fak = fak * i; }
   return fak;
}
```

Was passiert bei Aufruf `f = calc.getFactorial("Test", 4, 3);`?

- Dem Parameter `name` wird der Wert `"Test"` zugewiesen.
- Dem Parameter `f` wird der Wert `4` zugewiesen.
- Die Schleife in Zeile 3 läuft von 1 bis 3.
- Die Variable `f` hat nach dem Aufruf den Wert 24.
- Die Variable `f` muss vom Typ `int` sein.
- `System.out.println(fak);` gibt danach das result aus.

Quiz: Eigene Methoden

```
public int getFactorial(String name, int n, int f) {
   int fak = 1;
   for (int i=1; i<n; i++) {
      fak = fak * i; }
   return fak;
}
```

Was passiert bei Aufruf `f = calc.getFactorial("Test", 4, 3);`?

- Dem Parameter `name` wird der Wert `"Test"` zugewiesen. (ja)
- Dem Parameter `f` wird der Wert `4` zugewiesen. (nein)
- Die Schleife in Zeile 3 läuft von 1 bis 3. (ja)
- Die Variable `f` hat nach dem Aufruf den Wert 24. (nein)
- Die Variable `f` muss vom Typ `int` sein. (ja)
- `System.out.println(fak);` gibt danach das result aus. (nein)

Outline

JavaDoc

- Dokumentation ist extrem wichtig, um Wiederverwendbarkeit von Code sicherzustellen.
- Es geht meist schneller, die Erklärung zu lesen was der Code macht, als den Code komplett nachzuvollziehen.
- Oft gibt die Dokumentation wichtige Hinweise auf erlaubte Argumente oder zu erwartende Rückgabewerte.

JavaDoc

- Ein Dokumentationskommentar ist ein spezieller Kommentar.
- Er steht über der Methodensignatur und enthält alle wichtigen Informationen über die Methode: Was sie macht, welche Parameter sie hat und welche Rückgabe sie liefert.
- Beispiel:

```
/**
 * Calculates the factorial of the given number n
 *
 * @param n positive integer number
 * @return the factorial of n
 */
public int factorial(int n){...}
```

Outline

Gültigkeitsbereiche

Gültigkeitsbereich, Geltungsbereich, Sichtbarkeitsbereich

Der Gültigkeitsbereich einer Variablen ist der Codeabschnitt, der diese Variable “sehen” (verwenden, darauf zugreifen) kann. Der Gültigkeitsbereich ist immer der durch {} begrenzte Block, in dem die Variable deklariert wurde.

Beispiele für Gültigkeitsbereiche

```
for (int i=0; i<10; i++) { // i ist sichtbar
   for (int j=0; j<i; j++) { // i und j sind sichtbar
   }  // j existiert nicht mehr
} // i existiert nicht mehr
```

Beispiele für Gültigkeitsbereiche

```
for (int i=0; i<10; i++) { // i ist sichtbar
   for (int j=0; j<i; j++) { // i und j sind sichtbar
   }  // j existiert nicht mehr
} // i existiert nicht mehr
```

```
// number existiert nicht
public void test (int number) {
   System.out.println(number); // number ist sichtbar
}
// number existiert nicht
```

Lokale und globale Variablen

- Variablen, die nur in einem kleinen Block sichtbar sind, nennt man auch *lokale* Variablen - im Gegensatz zu *globalen* Variablen (Attributen), die überall sichtbar sind.
- Eine lokale Variable ist dazu da, einen temporären Wert für eine kurze Zeit zu halten. Globale Variablen halten permanente Werte zum Zustand eines Objekts.
- Es ist guter Stil, den Gültigkeitsbereich einer Variable immer so klein wie möglich zu wählen, um versehentliche Benutzung zu vermeiden.

Outline

Zusammenfassung: Methoden, Gültigkeitsbereiche

- Mit **Methoden** können Programmteile ausgelagert werden, wenn diese immer wieder ausgeführt werden sollen.
- Methoden stehen außerhalb der `main`-Funktion und werden ganz allgemein geschrieben (ohne konkrete Zahlenwerte anzugeben)
- Methoden können mit `return` Werte zurückgeben. Der **Rückgabetyp** steht vor dem Methodennamen
- Hinter dem Methodennamen steht die **Parameterliste**.
- Beim Aufruf der Methode muss für jeden Parameter ein passender Wert angegeben werden (**Argument**).

Universität Stuttgart
MINT-Kolleg Baden-Württemberg

Vielen Dank!

Frank Schweiner

E-Mail frank.schweiner@mint-kolleg.de
Telefon +49 (0) 711 685-84326
Fax —

Universität Stuttgart
MINT-Kolleg Baden-Württemberg

Azenbergstr. 12
70174 Stuttgart