Einführung in die Programmierung

by André Karge

Übung - Operatoren, Methoden, Primitives

Notizen

- Standard Linux Befehle: http://images.linoxide.com/linux-cheat-sheet.pdf
- Java ist auch eine Insel: http://openbook.rheinwerk-verlag.de/javainsel

letzte Woche

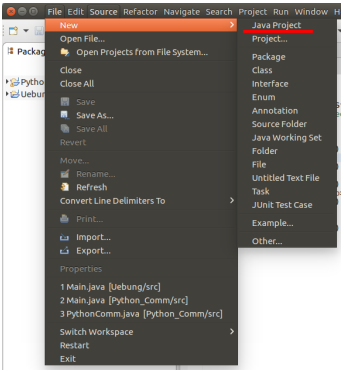
- Java JDK installiert
- Hello World Programm in einem Editor geschrieben
- Hello World Programm per Hand kompiliert
- Eclipse IDE installiert
- Hello World Programm in Eclipse geschrieben und kompiliert

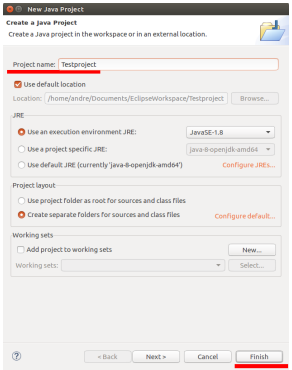
Heute

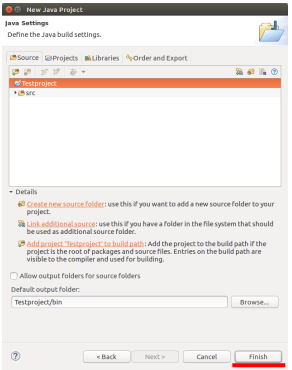
- Eclipse Infos
- Primitives
- Java Command Line I/O
- Operatoren
- Type Casting
- Methoden
- Einer-, Zweierkomplement

Eclipse

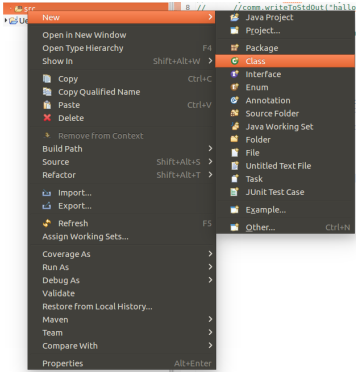
Eclipse - Projekt erstellen - Insel 1.7.2



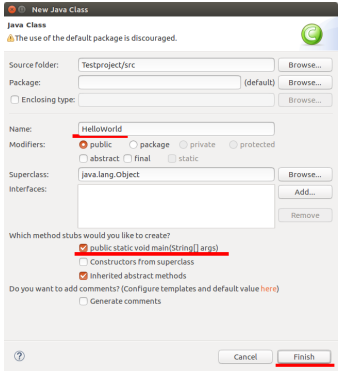




Eclipse - Klasse erstellen - Insel 1.7.4



→



Primitives

Primitives - Insel 2.3.1

Typ	**Beschreibung**
byte	8bit Ganzzahl
short	16bit Ganzzahl
int	32bit Ganzzahl
long	64bit Ganzzahl
float	32bit Fließkommazahl
double	64bit Fließkommazahl
char	16bit Unicode-Zeichen
boolean	1bit Wahrheitswert

Primitives - Insel 2.3.1

Beispiele

```
byte myByte = 10;
byte anderesByte = -10;

short myShort = 10;
short anderesShort = -10

int myInt = 42;
int anderesInt = 21;

long myLong = 56;

float myFloat = 0.0;
float anderesFloat = -16.315;

double myDouble = 3.1416
double anderesDouble = -42.4242;

char character = 'a'
boolean istWahr = true;
```

String

- ein komplexer Datentyp zur Speicherung von Zeichenketten
- wird sehr häufig verwendet
- Zusammengesetzt aus einer Liste von *char* Zeichen

String

- ein komplexer Datentyp zur Speicherung von Zeichenketten
- wird sehr häufig verwendet
- Zusammengesetzt aus einer Liste von *char* Zeichen

```
char a = 'a'; //Characters werden mit single quotes (') gekennzeichnet
String meinText = "Hallo Welt"; //Zeichenketten werden mit double quotes (") gekennzeichnet
```

Java Command Line I/O

Java I/O - Insel 2.3.3

Output

```
//gebe eine Zeichenkette auf der Kommandozeile aus
System.out.println("Hello World!");
String test = "Dies ist ein test";
System.out.println(test);
```

Java I/O - Insel 2.3.3

Output

```
//gebe eine Zeichenkette auf der Kommandozeile aus
System.out.println("Hello World!");
String test = "Dies ist ein test";
System.out.println(test);
```

Input

```
//lese eine Zeichenkette von der Kommandozeile
String text = new java.util.Scanner(System.in).nextLine();
//lese eine Ganzzahl von der Kommandozeile
int zahl = new java.util.Scanner(System.in).nextInt();
//lese eine Fließkommazahl von der Kommandozeile
double kommazahl = new java.util.Scanner(System.in).nextDouble();
```

Java I/O - Insel 2.3.3

Output

```
//gebe eine Zeichenkette auf der Kommandozeile aus
System.out.println("Hello World!");
String test = "Dies ist ein test";
System.out.println(test);
```

Input

```
//lese eine Zeichenkette von der Kommandozeile
String text = new java.util.Scanner(System.in).nextLine();
//lese eine Ganzzahl von der Kommandozeile
int zahl = new java.util.Scanner(System.in).nextInt();
//lese eine Fließkommazahl von der Kommandozeile
double kommazahl = new java.util.Scanner(System.in).nextDouble();
```

- Wichtig: beim Einlesen kommt es darauf an, die Eingabe dem korrekten Typen zuzuweisen

Operatoren

Operatoren - Insel 2.4

Zuweisungsoperator =

```
int i = 1; // speichert den Wert 1 in eine Variable vom Typ Integer mit dem Bezeichner i
String text = "Text"; // speichert die Zeichenkette in eine Variable vom Typ String mit Bezeichner text
int i = j = k = 5; // weist mehreren Variablen den Wert 5 zu
```

Arithmetische Operationen

- Addition +
- Subtraktion -
- Multiplikation *
- Division /
- Restwert %

Arithmetische Operatoren - Insel 2.4.2

```
System.out.println(3 + 4); // = 7
System.out.println(3 - 4); // = -1
System.out.println(3 * 4); // = 12
System.out.println(3 / 4); // = 0
```

Arithmetische Operatoren - Insel 2.4.2

```
System.out.println(3 + 4); // = 7
System.out.println(3 - 4); // = -1
System.out.println(3 * 4); // = 12
System.out.println(3 / 4); // = 0
```

```
// division mit Ganzzahlen wird zur nächsten Ganzzahl abgerundet
System.out.println(3.0 / 4); // = 0.75
System.out.println(3 / 4.0); // = 0.75
System.out.println(3.0 / 4.0); // = 0.75

System.out.println(9 % 2); // = 1
System.out.println(9.0 % 3); // = 0.0
```

Arithmetische Operatoren

```
// unärer minus operator -
int i = 1;
i = - i;

// verbundsoperatoren +=, -=, *=, /=
a = 1;
a += 2; // ist das selbe wie a = a + 2;
a -= 2; // a = a - 2;
a *= 2; // a = a * 2;
a /= 2; // a = a / 2;

// Postfix Increment/Decrement
i = 1;
i++; // ist das selbe wie i = i + 1;
i--; // ist das selbe wie i = i - 1;
```

Relationale Operatoren - Insel 2.4.6

- Größer-Als $>$
- Kleiner-Als $<$
- Größer-Gleich $>=$
- Kleiner-Gleich $>=$
- Gleich $==$
- Ungleich $!=$

Relationale Operatoren - Insel 2.4.6

- Größer-Als $>$
- Kleiner-Als $<$
- Größer-Gleich $>=$
- Kleiner-Gleich $>=$
- Gleich $==$
- Ungleich $!=$

Relationale Operatoren werden für Aussagenlogik gebraucht um 2 Werte zu vergleichen:

```
System.out.println(2 > 3); // = false
System.out.println(2 < 3); // = true

System.out.println(2 < 2); // = false
System.out.println(2 <= 2); // = true

System.out.println(2 == 3); // = false
System.out.println(2 != 3); // = true
```

Logische Operatoren - Insel 2.4.7

- logisches Und **&**
- logisches Oder |
- logische Negation **!**
- logisches XOR ^

```
System.out.println(true & true); // = true
System.out.println(true & false); // = false
System.out.println(true | false); // = true
System.out.println(!true); // = false
System.out.println(true ^ false); // = true
```

Type Casting - Insel 2.4.10

Type Casting = Umwandlung von Datentypen

Implizites Type Casting

```
int a = 100;
long b = a; // impliziter cast von int zu long
float f = a // impliziter cast von int zu float
```

Type Casting - Insel 2.4.10

Type Casting = Umwandlung von Datentypen

Implizites Type Casting

```
int a = 100;
long b = a; // impliziter cast von int zu long
float f = a // impliziter cast von int zu float
```

Achtung: geht nur, wenn der Ursprungstyp einen niedrigeren Wertebereich als der Zieltyp hat!
byte < short < int < long

Explizites Type Casting

```
float a = 1000.04f;
int b = (int)a; // expliziter cast von float zu int - Informationsverlust
```

Type Casting - Insel 2.4.10

Type Casting = Umwandlung von Datentypen

Implizites Type Casting

```
int a = 100;
long b = a; // impliziter cast von int zu long
float f = a // impliziter cast von int zu float
```

Achtung: geht nur, wenn der Ursprungstyp einen niedrigeren Wertebereich als der Zieltyp hat!
byte < short < int < long

Explizites Type Casting

```
float a = 1000.04f;
int b = (int)a; // expliziter cast von float zu int - Informationsverlust
```

Wir schreiben den Zieltypen explizit aus

Methoden

Methoden - Insel 2.7

- Methode = Beschreibung einer Prozedur von Anweisungen
- Wiederkehrende Programmteile sollten nicht immer wieder Programmiert werden
- Zerlegung komplexer Prozeduren in kleinere und simplere Prozeduren
- Definieren das *Verhalten* einer Klasse

Methoden - Bestandteile

Methodenbeispiel

```
public static void main(String[] args) {
  System.out.println("Hallo Welt");
}
```

Methodenkopf

```
public static void main(String[] args)
```

Methodenrumpf

```
{
  System.out.println("Hallo Welt");
}
```

Methoden - Methodenkopf

```
public static void main(String[] args)
```

- Bezeichner: *main*

Methoden - Methodenkopf

```
public static void main(String[] args)
```

- Bezeichner: *main*
- Rückgabetyp: *void*
  - Rückgabetyp *void* gibt an, dass die Methode keine Rückgabe hat
  - kann jeder beliebige Typ sein (Primitive und Komplexe Typen) Bsp.: *public static* ***int*** *main*

Methoden - Methodenkopf

```
public static void main(String[] args)
```

- Bezeichner: *main*
- Rückgabetyp: *void*
  - Rückgabetyp *void* gibt an, dass die Methode keine Rückgabe hat
  - kann jeder beliebige Typ sein (Primitive und Komplexe Typen) Bsp.: *public static* ***int*** *main*
- Methoden-Modifizierer: *public static*
  - public: gibt an, dass die Methode von anderen Klassen aus sichtbar ist (OOP)
  - static: gibt an, dass die Methode statisch ist(OOP)

Methoden - Methodenkopf

```
public static void main(String[] args)
```

- Bezeichner: *main*
- Rückgabetyp: *void*
  - Rückgabetyp *void* gibt an, dass die Methode keine Rückgabe hat
  - kann jeder beliebige Typ sein (Primitive und Komplexe Typen) Bsp.: *public static* ***int*** *main*
- Methoden-Modifizierer: *public static*
  - public: gibt an, dass die Methode von anderen Klassen aus sichtbar ist (OOP)
  - static: gibt an, dass die Methode statisch ist(OOP)
- Übergabeparameter *(String[] args)*
  - eine Liste aus Variablen (mit Komma getrennt), die in die Methode gegeben werden (brauchen einen Typen und einen Bezeichner)

Methoden - Methodenrumpf

```
{
  System.out.println("Hallo Welt");
}
```

- besteht aus allen Anweisungen, die die Prozedur ausmachen
- wenn der Rückgabetyp im Methodenkopf *void* ist, brauchen wir keine Rückgabe
- ansonsten muss ein Wert am Ende der Prozedur zurückgegben werden:

```
public static int getTheAnswer() {
  return 42;
}
```

Methoden - Aufruf von Methoden

```
public void writeHello() {
  System.out.println("Hello");
}
public void writeText(String text) {
  System.out.println(text);
}
public int add(int a, int b) { // int a, int b = Parameterliste
  return a + b;
}

public static void main(String[] args) {
  writeHello(); // call der ersten Methode
  writeText("etwas anderes") // call der zweiten Methode mit Argument
  System.out.println(add(21, 21)); // call der dritten Methode als Argument für println
}
```

- Variablendeklarationen im Methodenkopf = **Parameter**
- Übergabe von Werten beim Aufruf einer Methode = **Argumente**

Methoden - Signaturen

- Methodenname und Parameter bestimmen **Signatur** einer Methode in Java
- Bezeichner können deshalb mehrfach vergeben werden, solang die Parameterliste unterschiedlich ist
- wird auch *Überladen* von Methoden genannt

```
public void writeText() {
  System.out.println("Hello");
}
public void writeText(String text) {
  System.out.println(text);
}
public void writeText(String text, int zahl) {
  System.out.println(text + zahl);
}
```

Methoden - Gültigkeitsbereiche

```
public int methode1() {
  int a = 2;
  int b = 3;
  return a + b;
}
public int methode2() {
  int c = a; // nicht möglich, da a in einer anderen Methode deklariert wurde
  int d = 15;
  return c + d;
}
public int methode3() {
  int a = 2;
  int c = a; // möglich, da a in dieser Methode deklariert wurde
  int d = 15;
  return c + d;
}
```

Dieser Gültigkeitsbereich wird **Scope** genannt

Fragen?