

Programmieren I

Klassen und Objekte



Heusch 12-13
Ratz 7-9

Institut für Automation und angewandte Informatik

```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

Klassen, Objekte und Instanzen (Wiederholung)

- **Klasse**: Sammlung aller Eigenschaften und Methoden der Objekte. Konstruktoren legen deren Erzeugung fest.
- „**Instanz** einer Klasse“: tatsächliches Objekt.
(Klasse = abstrakte, Instanz = konkrete Darstellung)
- Jede Klasse besteht im allgemeinen aus zwei Komponenten: Attribute und Methoden.
- Die **Daten** (Eigenschaften) werden durch Variablen (Attribute) definiert (Instanzvariablen und Klassenvariablen).
- Das **Verhalten** wird durch Methoden bestimmt. Methoden regeln, was Objekte alles machen können und was man mit Objekten machen kann. Methoden sind Funktionen, die innerhalb von Klassen definiert (Instanzmethoden und Klassenmethoden) sind.




Klassen – selbstdefinierte Datentypen erstellen

- Einfache Datentypen können nur einen Wert speichern.
- Eine Klasse beschreibt einen "neuen Datentyp", der sich durch verschiedene Attribute und eine spezielle Funktionalität auszeichnet.
- Java baut als **objektorientierte Programmiersprache** auf diesem Konzept auf. Sämtlicher Programmcode befindet sich innerhalb von Klassenbeschreibungen. Attribute und Methoden sind immer innerhalb einer Klasse beschrieben.
- Syntax einer Klassendefinition in Java:

```
class class_name {  
    { attribute }  
    { constructor }  
    { method }  
}
```

Attribute versus lokale Variablen

- Lokale Variablen werden innerhalb einer Methode in einem Anweisungsblock definiert und sind auch nur in diesem Bereich gültig.
- Neben lokalen Variablen (innerhalb von Methoden) kennt Java so genannte **Attribute**. Attribute werden außerhalb einer Methode in der Klassenbeschreibung definiert.

```
class Rectangle {  
    int width;  Attribute  
    int height;  Attribute  
  
    // ...  
    public static void main(String args[]) {  
        int i = 1;  Lokale Variable  
        // ...  
    }  
}
```

Instanzvariablen

- **Attribute** gehören standardmäßig (keine Verwendung des Modifikators `static`) zu den Objekten der Klasse. Sie werden deshalb auch **Instanzvariablen** genannt.
- Instanzvariablen stellen die Eigenschaften des Objektes dar.
- Sie existieren erst wenn ein Objekt existiert.
- Jedes Objekt einer Klasse hat die gleichen Instanzvariablen, besitzt aber eine eigene Identität.

Konstruktoren

- Konstruktoren werden nur bei der Instanziierung (Erzeugung) eines Objektes der Klasse aufgerufen. Sie tragen den Namen der Klasse.

```
class Rectangle {  
  
    int width;    // Attribut für die Breite  
    int height;   // Attribut für die Höhe  
  
    public Rectangle() { // Konstruktor ohne Parameter  
        width = 0;  
        height = 0;  
    }  
  
    public Rectangle(int w, int h) { // Konstruktor mit Parametern  
        width = w;  
        height = h;  
    }  
  
    // ...  
}
```



Heusch 12.4
Ratz 8.4

Erzeugung von Objekten, Referenzvariablen (1)

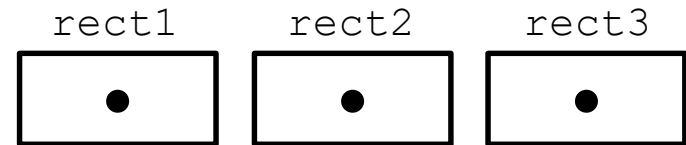
- Klassen werden zur *Übersetzungszeit* angelegt (im erzeugten Programmcode).
- Objekte werden hingegen erst zur *Laufzeit* erzeugt → Mechanismus zur Instanziierung notwendig.
- In Java gibt es keine Variablen, deren Wert ein Objekt ist (wie bei primitiven Datentypen).
- Es gibt nur Variablen, in denen eine *Referenz* - ein Verweis - auf ein Objekt abgelegt werden kann (Referenzvariablen).
- Die Objektreferenz wird beim Erzeugen des Objekts geliefert.

Erzeugung von Objekten, Referenzvariablen (2)

- *Deklaration* von Referenzvariablen. Syntax und Beispiel:

```
class_name variable_name {, variable_name } ;
```

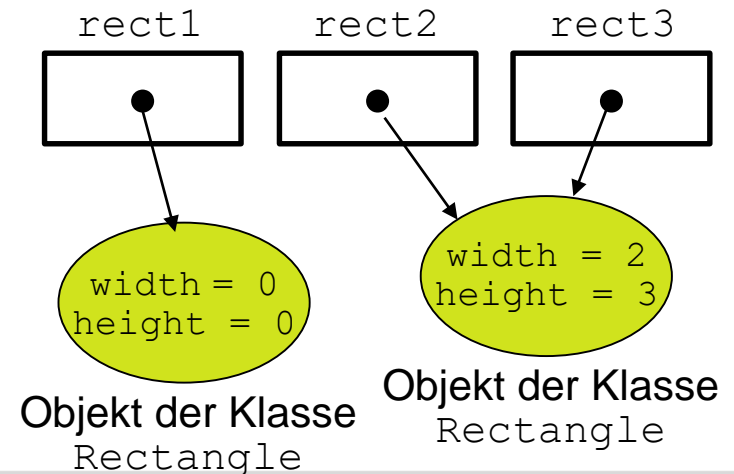
```
Rectangle rect1;
Rectangle rect2, rect3;
```



- *Erzeugung* von Objekten mit Hilfe von **new** und Zuweisung der Objekt-Referenzen an Referenzvariablen:

```
variable_name = new class_name ( [ parameter_list ] ) ;
```

```
rect1 = new Rectangle();
rect2 = new Rectangle(2, 3);
rect3 = rect2;
```



Attribute von Objekten: Zugriff

- Der Zugriff auf Attribute (und auch Methoden) erfolgt mit Hilfe des **Dereferenzierungsoperators** "." (Punkt).

- Syntax:

```
variable_name . attribut_name
```

variable_name: Name der Variablen, in der die Referenz auf das Objekt gespeichert ist.

- Beispiel:

```
class Geometry {  
    public static void main(String[] args) {  
        Rectangle a = new Rectangle(2, 3);  
        Rectangle b = new Rectangle(4, 5);  
  
        a.height = b.height;  
        b.height = 10;  
    }  
}
```

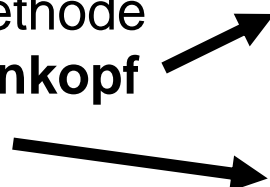
Methoden – die Funktionalität der Klassen

- Neben den Attributen können auch Methoden zu einer Klasse gehören.
- Mit Hilfe von Methoden wird das **Verhalten** (**Funktionalität**) der Objekte einer Klasse ausgedrückt.
 - Methoden werden wie die Attribute innerhalb einer Klasse beschrieben.
 - Methoden gehören immer zu der Klasse.
 - Eine Klasse kann mehrere Methoden haben, wobei die Methoden in beliebiger Reihenfolge beschrieben werden können.
 - Methoden können nicht geschachtelt werden.



Heusch 9.2
Ratz 6.1


Methoden deklarieren

- Beschreibung einer Methode erfolgt durch **Methodenkopf** und **Methodenrumpf**.

- Methodenkopf besteht aus dem Namen, einer Parameterliste, dem Typ des Rückgabewertes und einem Sichtbarkeitsmodifikator, z.B. „public“ oder „private“.
- Der **Methodenrumpf** besteht aus einem Block mit Anweisungen, der in geschweifte Klammern { } gesetzt wird.
- Mit der Anweisung **return** *Ausdruck*; wird ein **Rückgabewert** festgelegt, dessen Typ dem deklarierten Rückgabe-Typ entsprechen muss.
- Wird anstelle eines Rückgabe-Typs das Schlüsselwort **void** verwendet, hat die Methode keinen Rückgabewert.

```
int getArea()  
{  
    int result = height * width;  
    return result;  
}
```

Beispiel: Klasse Rectangle

```
public class Rectangle {  
    int width;  
    int height;  
  
    public Rectangle(int w, int h) {  
        width = w;  
        height = h;  
    }  
  
    public int getArea(){  
        return height * width;  
    }  
  
    public void scale(int factor){  
        width = width * factor;  
        height *= factor;  
    }  
  
    public static void main(String[] args) {  
        Rectangle a = new Rectangle(3, 4);  
        a.scale(3);  
        System.out.println(a.getArea());  
    }  
}
```


> java Rectangle
108

Weitere Methoden der Klasse Rectangle

```
public class Rectangle {  
    int width;  
    int height;  
    // ...  
  
    public void setWidth(int width) {  
        this.width = width;  
    }  
    public void setHeight(int h) {  
        height = h;  
    }  
  
    @Override  
    public String toString() {  
        String res;  
        res = "Rectangle Width: " + width + "\n";  
        res += "           Height: " + height + "\n";  
        return res;  
    }  
    //...  
}
```

Instanz- bzw. Objektmethoden (1)

- Die bisher behandelten Methoden müssen von einem Objekt aus aufgerufen werden.
Man nennt sie deshalb *Objekt- bzw. Instanzmethoden*.
Der Aufruf erfolgt in der Form

```
object_name . method_name ( [ parameter_list ] )
```

```
Rectangle a = new Rectangle(3, 4);  
a.scale(3);
```

- Methodennamen beginnen üblicherweise mit einem Kleinbuchstaben (Konvention, nicht Pflicht).
- Wird ein Attribut oder eine Methode aus einer Methode derselben Klasse aufgerufen, braucht keine Objektreferenz angegeben werden.

Instanz- bzw. Objektmethoden (2)

- Im Gegensatz zum Zugriff auf Attribute wird der Zugriff auf Methoden durch folgende runde Klammern () hinter dem Namen gekennzeichnet.
- Objekte müssen nach ihrer Benutzung nicht explizit gelöscht werden, dies übernimmt der Garbage Collector.


Beispiel: Rectangle-Objekte verwenden

```
public class Geometry {  
  
    public static void main(String[] args) {  
        Rectangle a = new Rectangle(3, 4);  
        Rectangle b = new Rectangle(1, 2);  
  
        System.out.println(a.toString());  
        System.out.println(b);  
        a.scale(3);  
        b.setWidth(2);  
        System.out.println(a.getArea());  
        int area = b.getArea();  
        System.out.println(area);  
  
        System.out.println(a);  
        System.out.println(b.toString());  
    }  
}
```

? Ausgabe?

Verwendung von `this` bei Kapselung von Attributen

- Von der Entwicklungsumgebung erzeugte getter- und setter-Methoden.

 **Eclipse:** Source (Shift+Alt+S) > Generate Getters and Setters

 **NetBeans:** Refactor > Encapsulate Fields (Ctrl+Alt+Shift+E)

```
public class Fraction {  
  
    private int numerator = 0;  
    private int denominator = 1;  
  
    public void setDenominator(int d) {  
        if (d != 0)  
            denominator = d;  
    }  
  
    public int getDenominator() {  
        return denominator;  
    }  
  
    public int getNumerator() {  
        return numerator;  
    }  
}
```

```
    public void setNumerator(int numerator) {  
        this.numerator = numerator;  
    }  
  
    public double getValue(){  
        return getNumerator()  
            / (double)getDenominator();  
    }  
  
    public static void main  
        (String[] args) {  
        Fraction f = new Fraction();  
        f.setNumerator(1);  
        f.setDenominator(2);  
        System.out.println  
            ("Value: " + f.getValue());  
    }  
}
```

Überladen

- Innerhalb einer Klasse können mehrere Methoden (und auch Konstruktoren) mit demselben Namen existieren.
- Sie müssen sich jedoch hinsichtlich der Anzahl und/oder des Datentyps der Parameter unterscheiden.
- Die Programmierung mehrerer Methoden innerhalb einer Klasse mit gleichem Namen und unterschiedlicher Anzahl von Parametern wird oft auch **Überladen (Overloading)** genannt. Die Auswahl der geeigneten Methode geschieht zur Compilezeit.
- Beispiel für das Überladen von Methoden:

```
public void resize(int factor) {  
    height *= factor;  
    width *= factor;  
}  
  
public void resize(int newWidth, int newHeight) {  
    width = newWidth;  
    height = newHeight;  
}
```



Heusch 12.3.4
Ratz 6.1.5

Mehrere Konstruktoren und Zuweisung von Objektreferenzen

```
public class Circle {
    // Instanzvariablen
    public double x, y, r;

    // Konstruktoren
    public Circle(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    public Circle(double r) {
        this(0, 0, r);
    }

    public Circle(Circle c) {
        this(c.x, c.y, c.r);
    }

    public Circle() {
        this(0, 0, 1);
    }

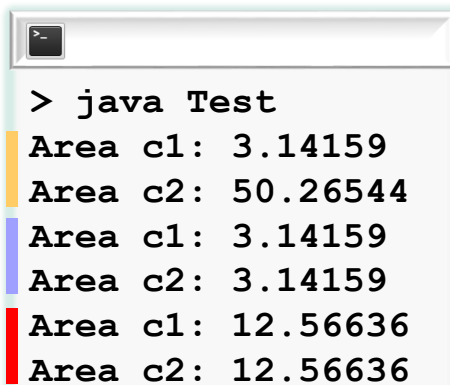
    // Methoden
    public double getArea() {
        return 3.14159 * r * r;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        Circle c2 = new Circle(2,2,4);

        System.out.println("Area c1: "+c1.getArea());
        System.out.println("Area c2: "+c2.getArea());

        c2 = c1;
        System.out.println("Area c1: "+c1.getArea());
        System.out.println("Area c2: "+c2.getArea());

        c1.r = 2;
        System.out.println("Area c1: "+c1.getArea());
        System.out.println("Area c2: "+c2.getArea());
    }
}
```



```
> java Test
Area c1: 3.14159
Area c2: 50.26544
Area c1: 3.14159
Area c2: 3.14159
Area c1: 12.56636
Area c2: 12.56636
```

Constructor Chaining

- verschiedene Konstruktoren können von anderen Konstruktoren aus aufgerufen werden
- Verwendung des Schlüsselworts `this`
- Parameterliste des jeweiligen Konstruktors wird anschließend übergeben
- Beispiele siehe eingekreiste Aufrufe in vorheriger Folie

ÜBUNG

Klassenvariablen und -methoden

- Werden definiert durch Verwendung des Schlüsselworts `static`.
- Sind nicht an die Existenz eines Objektes einer Klasse gebunden.
- Im Gegensatz zu Objektvariablen gehören statische Variablen zu der Klasse selbst. Sie werden daher auch **Klassenvariablen** genannt.
- Klassenvariablen existieren genau einmal, wobei es nicht erforderlich ist, dass Objekte (Instanzen) von der Klasse existieren
- Ebenso ist die Festlegung von statischen Methoden (Klassenmethoden) möglich.
- Statische Methoden arbeiten unabhängig von den Attributen der Klasse.
- Der Zugriff auf statische Variablen und Methoden erfolgt über den Klassennamen.



Heusch 12.8
Ratz 8.3

Syntax der Deklaration von Klassenvariablen und -methoden

■ Deklaration von Klassenvariablen und Beispiel:

```
{ modifier } static type variable_name [= value]  
    {, variable_name [= value]} ;
```

```
public static int numCircles = 0;
```

■ Deklaration von Klassenmethoden und Beispiel:

```
{ modifier } static type method_name ( [type parameter_name  
    {, type parameter_name}] ) {  
    method_body  
}
```

```
public static double abs(double x) {  
    if (x < 0) {  
        return -x;  
    } else {  
        return x;  
    }  
}
```

Beispiel: Klassenvariablen

```
public class Circle {  
    // Klassenvariable: erzeugte Kreise  
    public static int numCircles = 0;  
  
    // Konstante  
    public static final double PI = 3.14159265;  
  
    // Instanzvariablen  
    public double x, y, r;  
  
    // Konstruktor  
    public Circle(double x, double y, double r) {  
        this.x = x;  
        this.y = y;  
        this.r = r;  
        numCircles++;  
    }  
  
    // Methoden  
    public double getCircumference() {  
        return 2 * PI * r;  
    }  
    public double getArea() {  
        return PI * r * r;  
    }  
}
```

```
// Zugriff auf Klassenvariablen  
System.out.println(  
    "Circles: " + Circle.numCircles  
);  
  
System.out.println("Pi: "+Circle.PI);
```


Beispiel: Klassenmethoden (1)

```
public class Circle {  
  
    // Klassenvariable: erzeugte Kreise  
    public static int numCircles = 0;  
    // Konstante  
    public static final double PI = 3.14159265;  
    // Instanzvariablen  
    public double x, y, r;  
  
    // Konstruktor  
    // ...  
  
    // Liegt Punkt (a,b) innerh. des Kreises?  
    public boolean isInside(double a, double b) {  
        double dx = a - x;  
        double dy = b - y;  
        double distance = Math.sqrt(dx*dx+dy*dy);  
        return distance <= r;  
    }  
    // ...  
}
```

```
// Aufruf der Instanzmethode  
Circle a = new Circle(1,2,3);  
System.out.println(  
    "inside: " + a.isInside(0,0)  
);
```

\swarrow
`Math.sqrt()` ist ein Aufruf einer Klassenmethode (statische Methode), die in der Klasse `Math` definiert ist.

Beispiel: Klassenmethoden (2)

```
public class Circle {  
    // ...  
    // Instanzvariablen  
    public double x, y, r;  
    // ...  
  
    // Instanzmethode,  
    // liefert den größeren Kreis  
    public Circle bigger(Circle c) {  
        return c.r > r ? c : this;  
    }  
  
    // Klassenmethode, liefert den größeren Kreis  
    public static Circle bigger(Circle a, Circle b) {  
        if ( a.r > b.r )  
            return a;  
        else  
            return b;  
    }  
  
    // ...  
}
```

```
// Aufruf der Instanzmethode  
Circle a = new Circle(2);  
Circle b = new Circle(3);  
Circle c = a.bigger(b);  
// oder: b.bigger(a);
```

```
// Aufruf der Klassenmethode  
Circle a = new Circle(2);  
Circle b = new Circle(3);  
Circle c = Circle.bigger(a,b);
```

Statischer Initialisierer (static-Block)

- Ein static-Block ist eine Art Konstruktor für das Klassenobjekt selbst (und nicht für eine Instanz der Klasse).
- In ihm können statische Komponenten einer Klasse, z.B. Klassenvariablen, initialisiert werden.
- Syntax:

```
static {  
    { statement }  
}
```

- Statische Initialisierer haben nur Zugriff auf statische Komponenten einer Klasse, nicht auf Instanzkomponenten.
- Ein static-Block wird beim Start ausgeführt (wenn die Klasse geladen wird, zur Übersetzungszeit).

Beispiel: Initialisierung von Klassenvariablen

```
public class RandomNumbers {  
  
    private static final int MAX = 20;  
    private static int[] randomNumbers;  
  
    static {  
        RandomNumbers.randomNumbers = new int[RandomNumbers.MAX];  
        // Lokale Variable rand ist nicht static!  
        Random rand = new Random();  
        for (int i = 0; i < RandomNumbers.MAX; i++) {  
            RandomNumbers.randomNumbers[i] = rand.nextInt(50);  
        }  
    }  
  
    // ...  
  
}
```


Klasse mit gemischten Attributen

```
public class Mix {  
    protected int a;  
    protected static int b;  
  
    public Mix(int val) {  
        a = val;  
        b = 17;  
    }  
  
    public void print() {  
        System.out.println(a + " " + b);  
    }  
  
    public void setAundB(int val1, int val2) {  
        a = val1;  
        b = val2;  
    }  
  
    public static void setAundBStatic(int val1, int val2) {  
        a = val1; // Compiler-Fehler!  
        b = val2;  
    }  
}
```

Zugriff auf ein nicht-statisches Attribut **a** (Instanzvariable) aus einer statischen Methode heraus.

Methode toString()

```
public class Circle {  
  
    public double x, y, r; // Instanzvariablen  
  
    // Konstruktor  
    public Circle(double x, double y, double r) {  
        this.x = x;  
        this.y = y;  
        this.r = r;  
    }  
  
    @Override  
    public String toString() {  
        return "Circle with center at: (" + x + "," + y + ") and radius: " + r;  
    }  
  
    public static void main(String args[]) {  
        Circle kreis = new Circle(3.0, 3.0, 2.0);  
        System.out.println(kreis.toString());  
        System.out.println(kreis);  
    }  
}
```

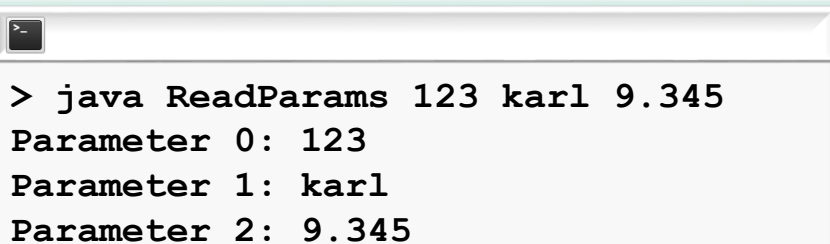

> java Circle
Circle with center at: (3.0,3.0) and radius: 2.0
Circle with center at: (3.0,3.0) and radius: 2.0

Die main-Methode

- Eine Java-Applikation benötigt eine main-Methode (auch Hauptprogramm genannt).

```
public class ReadParams {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            System.out.println("Parameter " + i + ": " + args[i]);  
        }  
    }  
}
```

- Die Methode `main(String args[])` verwendet Argumente. Auf diese Weise lassen sich beim Programmstart Parameter übergeben.
- Beispiel für Aufruf der Anwendung mit Übergabe der Parameter:



```
> java ReadParams 123 karl 9.345  
Parameter 0: 123  
Parameter 1: karl  
Parameter 2: 9.345
```



Heusch 4.1.12
Ratz 6.3

Parameterübergabe per IDE - Eclipse

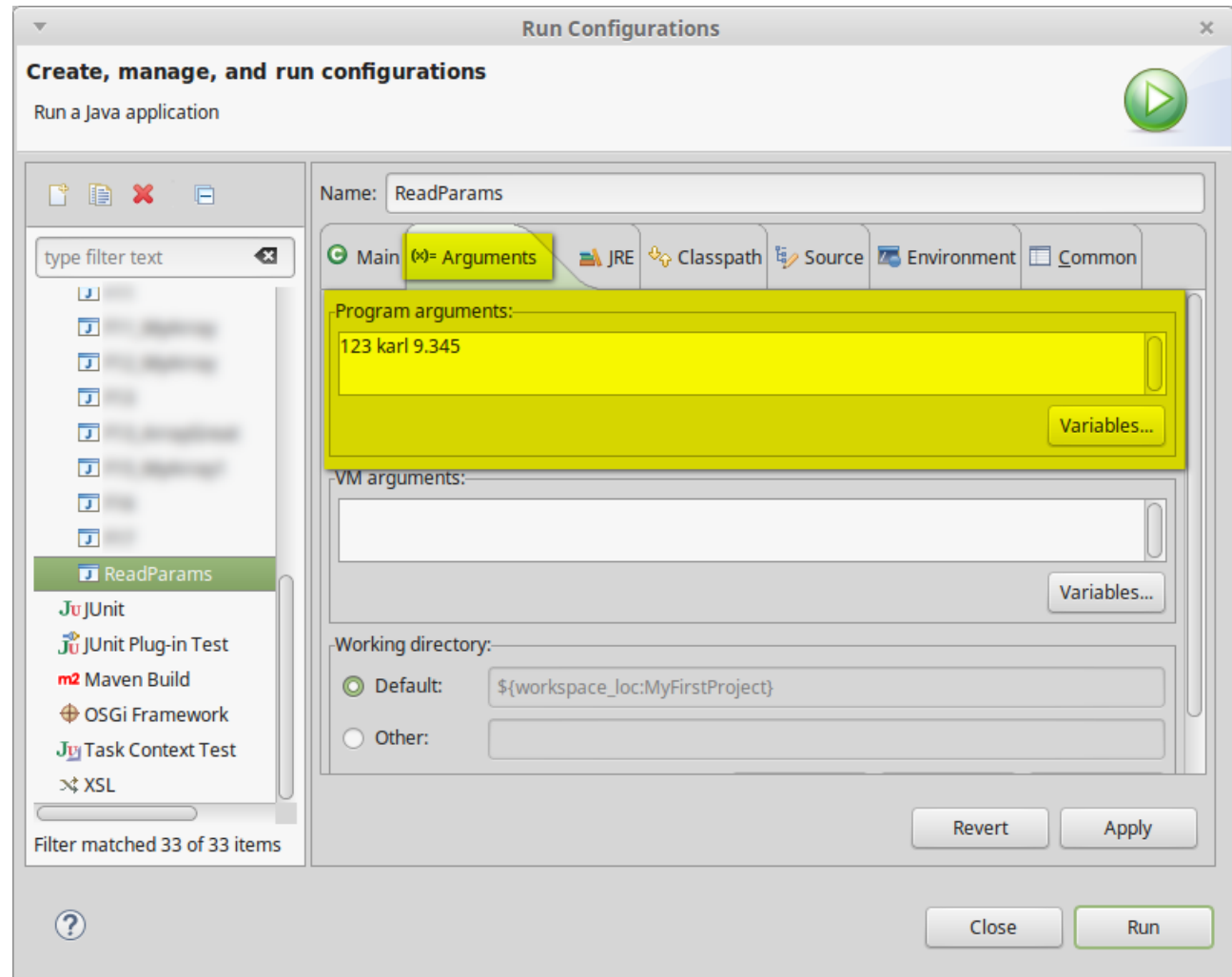
- Rechtsklick auf die Klasse im Package Explorer



Run as..



Run Configurations
(Reiter „Arguments“)

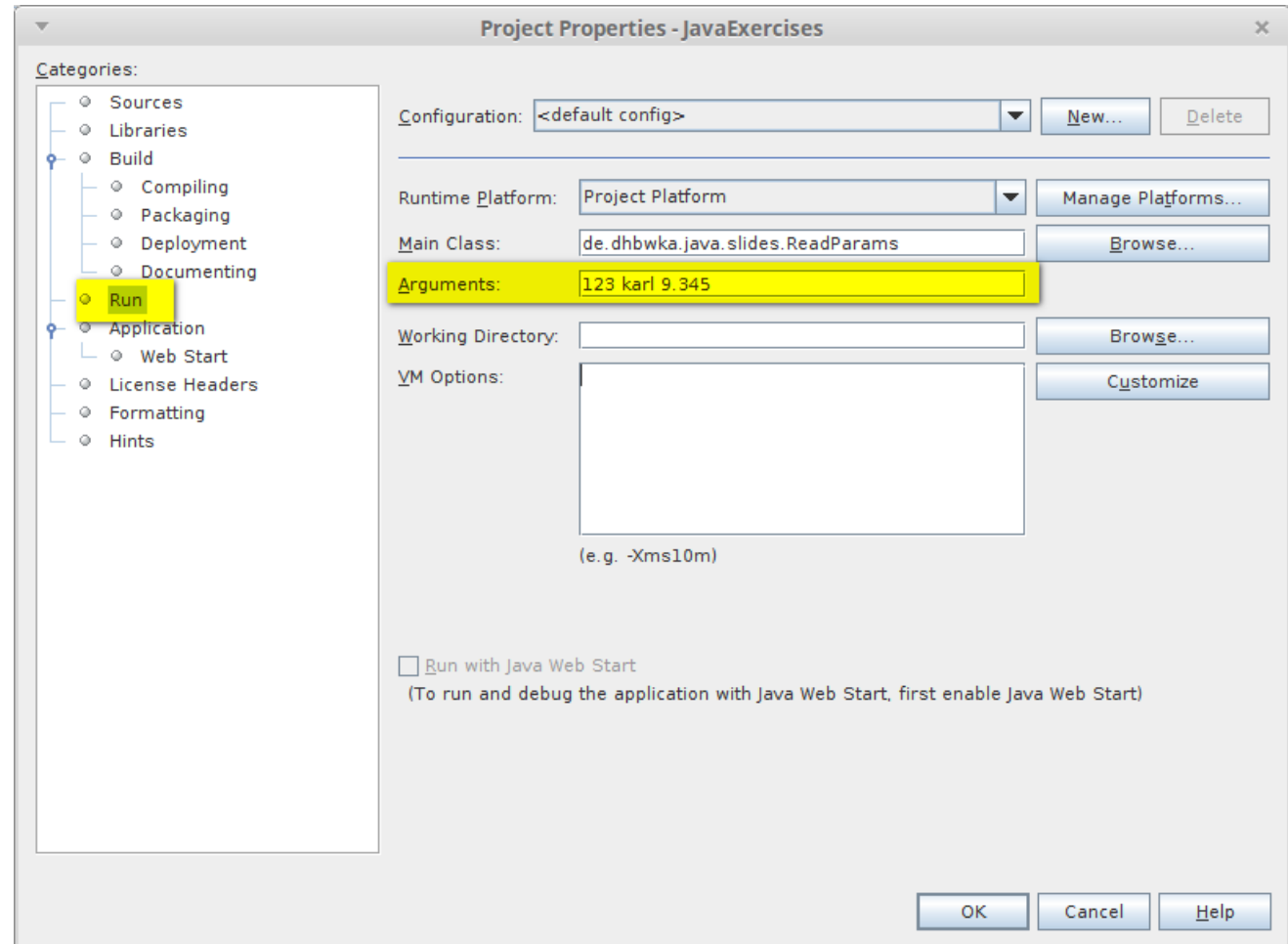


Parameterübergabe per IDE - NetBeans

- Rechtsklick
aufs Projekt

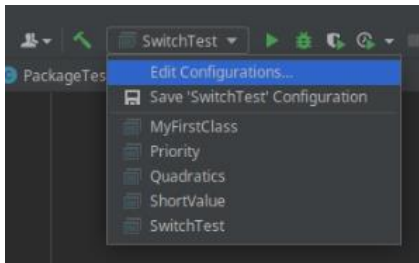


Properties
(Unterpunkt „Run“)

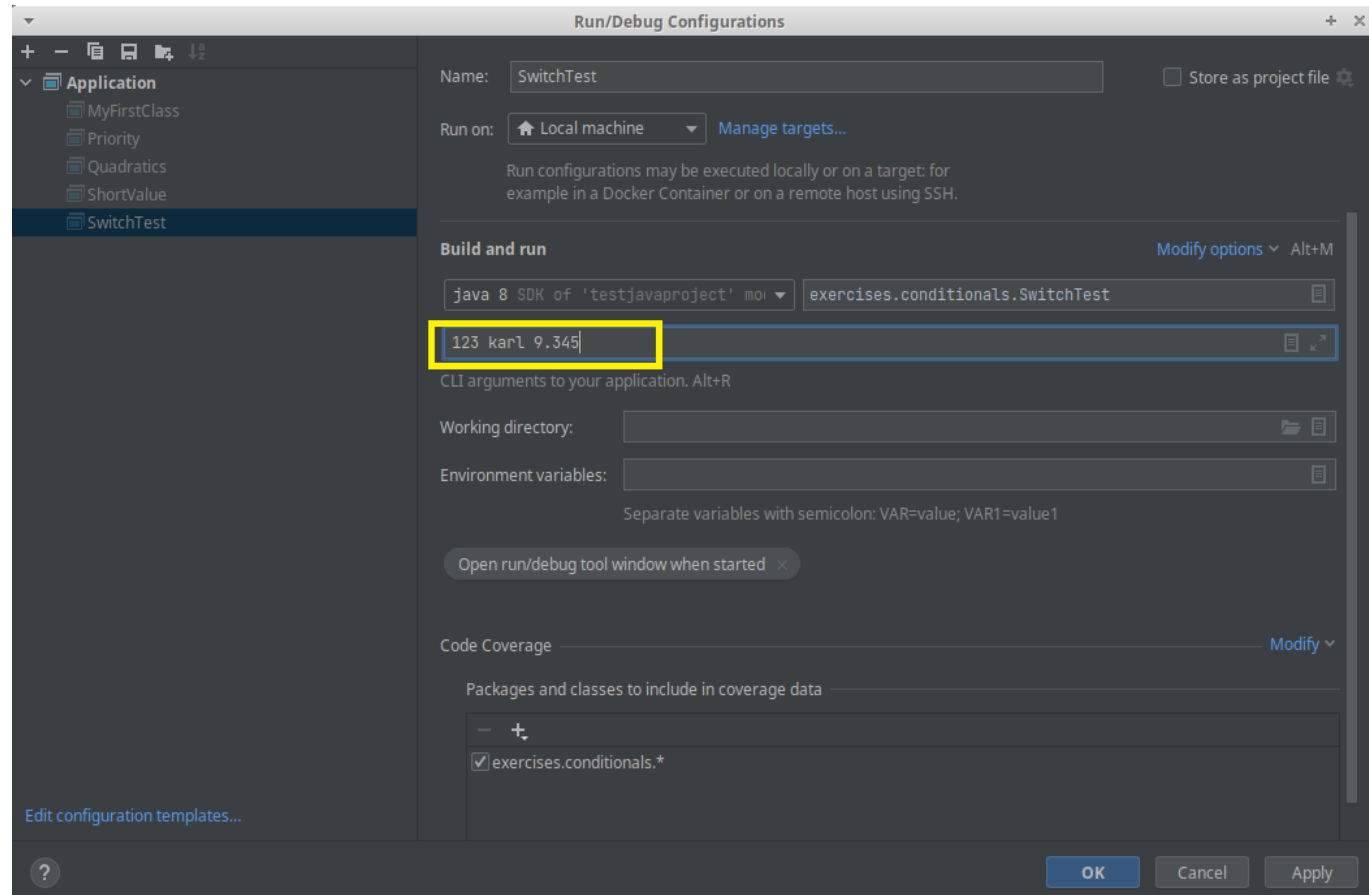


Parameterübergabe per IDE - IntelliJ

- Gewünschte config auswählen



Edit Configurations



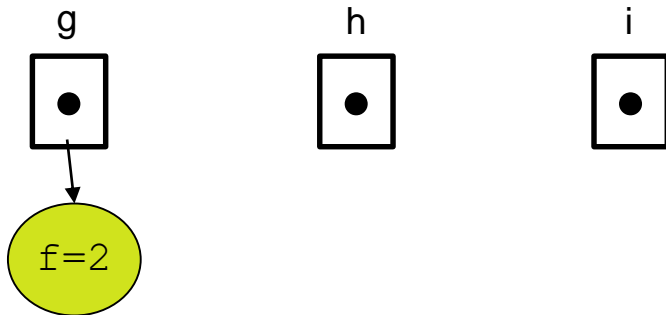
Was gibt dieses Programm aus?

```
public class Exam {  
  
    float f;  
  
    Exam(float z) {  
        f = z;  
    }  
  
    public Exam method1(float i) {  
        return new Exam(f * i);  
    }  
  
    public Exam method2(float i) {  
        f += i;  
        return this;  
    }  
  
    public Exam method3(Exam h) {  
        f = h.getValue() + this.getValue();  
        h.f = h.getValue() + getValue();  
        return this;  
    }  
  
    public float getValue() {  
        return f;  
    }  
}
```

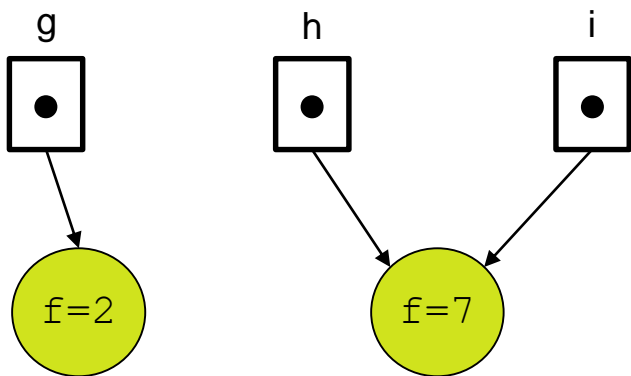
```
    public String toString() {  
        return "Exam: " + f;  
    }  
  
    public static void main(String[] args) {  
        Exam g = new Exam(2);  
        Exam h;  
        Exam i;  
        h = g.method1(3);  
        System.out.println(g);  
        System.out.println(h);  
        i = h.method2(1);  
        System.out.println(g);  
        System.out.println(h);  
        System.out.println(i);  
        i = g.method3(h);  
        System.out.println(g);  
        System.out.println(h);  
        System.out.println(i);  
    }  
}
```

Ergebnis

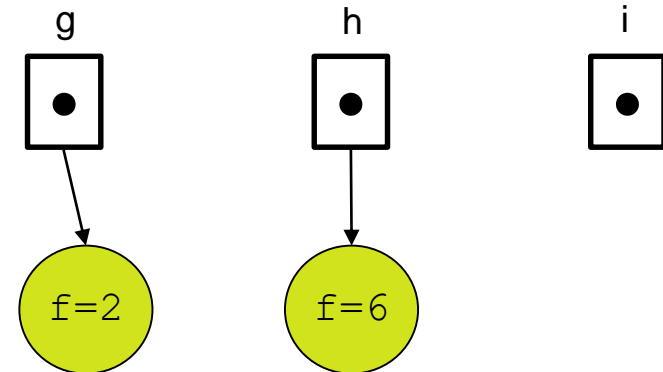
```
Exam g = new Exam(2);
Exam h;
Exam i;
```



```
i = h.method2(1);
```



```
h = g.method1(3);
```



```
i = g.method3(h);
```

