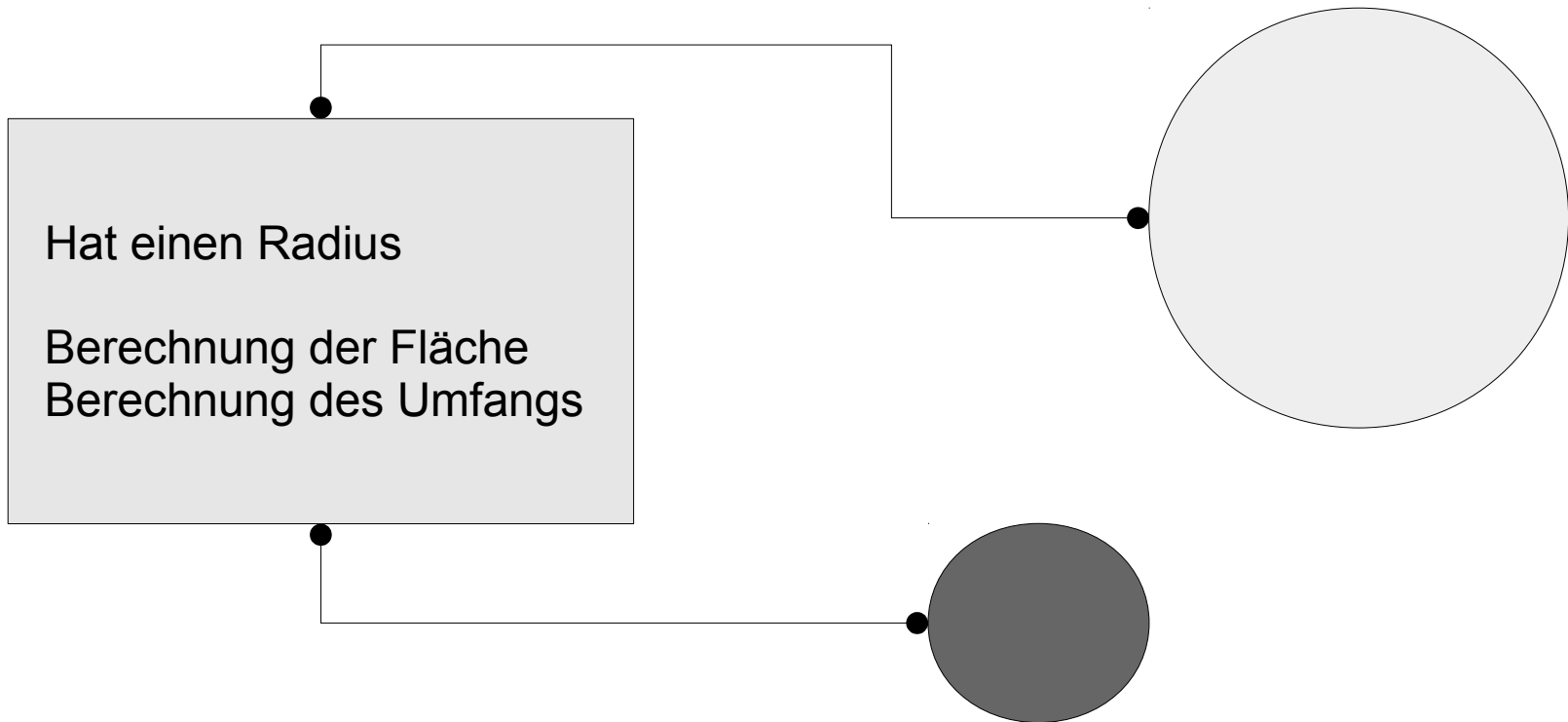


Java - Klassen und Objekte



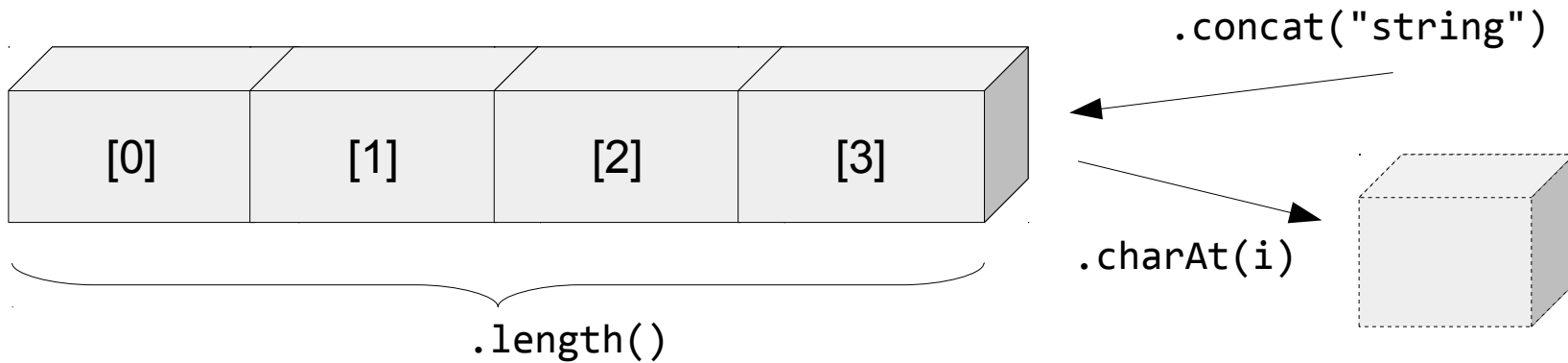
Objektorientierte Programmierung

- Abstraktion von Gegenständen der realen Welt mit Hilfe von Klassen.
- Daten und die dazugehörigen Methoden werden in Klassen zusammengefasst.
- Daten können nicht direkt von außen verändert werden.
- Klassen können Daten und Methoden vererben.

Klasse

- Abstraktion von Dingen aus der realen Welt.
- Definition eines konkreten Objekts. Welche Daten werden zur Beschreibung des Objekts benötigt? Wie kann das Verhalten des Objekts allgemeingültig beschrieben werden?
- Formale Beschreibung einer bestimmten Objektart.
- Vorlage für die Erzeugung eines Objektes.

Klasse „String“

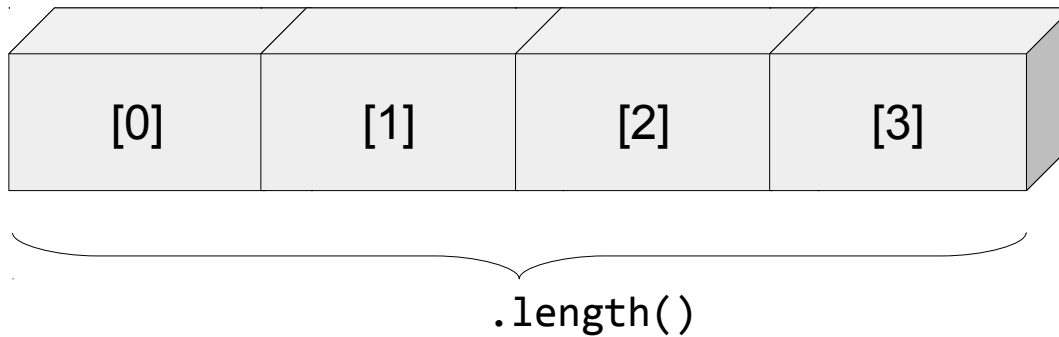


- Die Klasse `String` beschreibt allgemein die Funktionalität einer Zeichenkette.
- Eine Zeichenkette besteht aus beliebig vielen Elemente in einer bestimmten Zeichenkodierung.

Attribute (Member, Instanzvariablen)

- Beschreibung eines Gegenstandes, einer Person, etc.
- Allgemeingültige Beschreibung für einen bestimmten Objekttyp.
- Jedes Objekt einer Klasse hat die gleichen Attribute.
- Jedes Objekt einer Klasse unterscheidet sich aber in mindestens einem Attribut-Wert von allen anderen Objekten.
- Die Attribute werden für jedes Ding einmal im Speicher abgelegt.

Klasse „String“

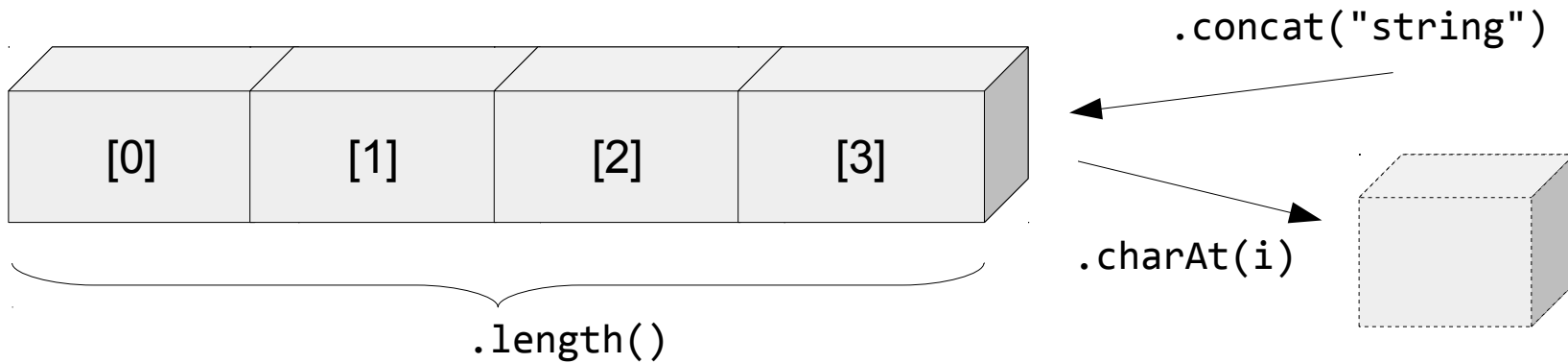


- Von außen kann man auf Attribute einer Klasse nur mit Hilfe einer Methode zugreifen.
- In diesem Beispiel wird auf das Attribut „Länge der Zeichenkette“ mit Hilfe einer Methode zugegriffen.

Methoden

- Alle Methoden gemeinsam beschreiben das Verhalten eines Objekts.
- Der Bauplan wird mit den Funktionen des zu bauenden Elements ausgeliefert.
- Lesen und modifizieren eines konkreten Dings in Abhängigkeit von definierten Regeln.
- Ein Zugriff von außen auf die Attribute eines Objekts erfolgt nur mit Hilfe von Methoden.
- Methoden werden für alle Objekte einer Klasse gemeinsam im Speicher abgelegt.

Klasse „String“

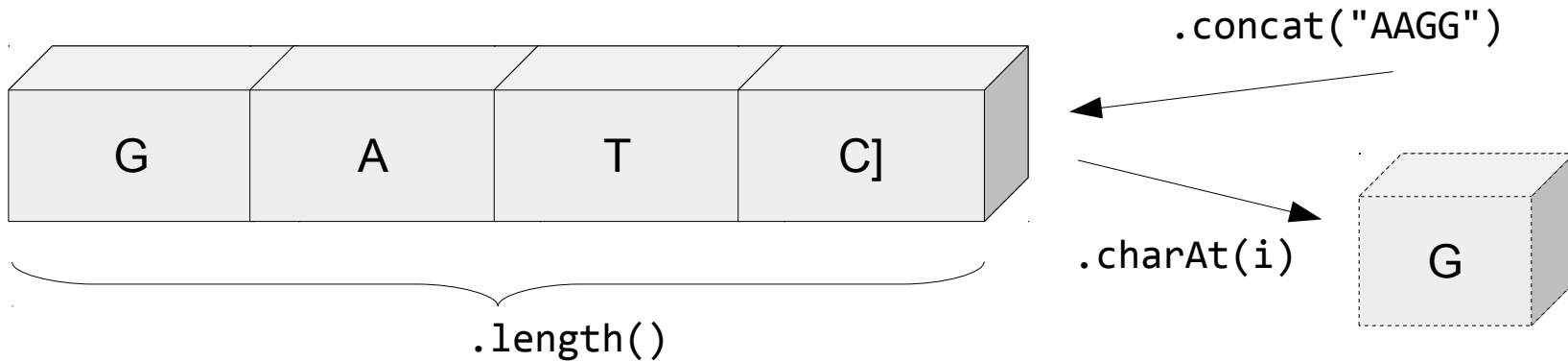


- Mit Hilfe der Methode `.concat()` wird ein String mit einem anderen String verknüpft.
- Die Methode `.charAt()` gibt ein Zeichen aus dem String zurück, verändert aber den String nicht.

Objekte

- Ein Ding (Exemplar, Instanz) aus der realen Welt.
- Substantive in einer Textbeschreibung.
- Erzeugung zur Laufzeit eines Programms über Klassen.
- Instanz einer Klasse.
- Alle Elemente einer Kategorie von Dingen haben die gleichen Eigenschaften, aber in unterschiedlichen Ausprägungen. Sie nutzen die gleichen Methoden zum Ändern ihrer Ausprägungen.

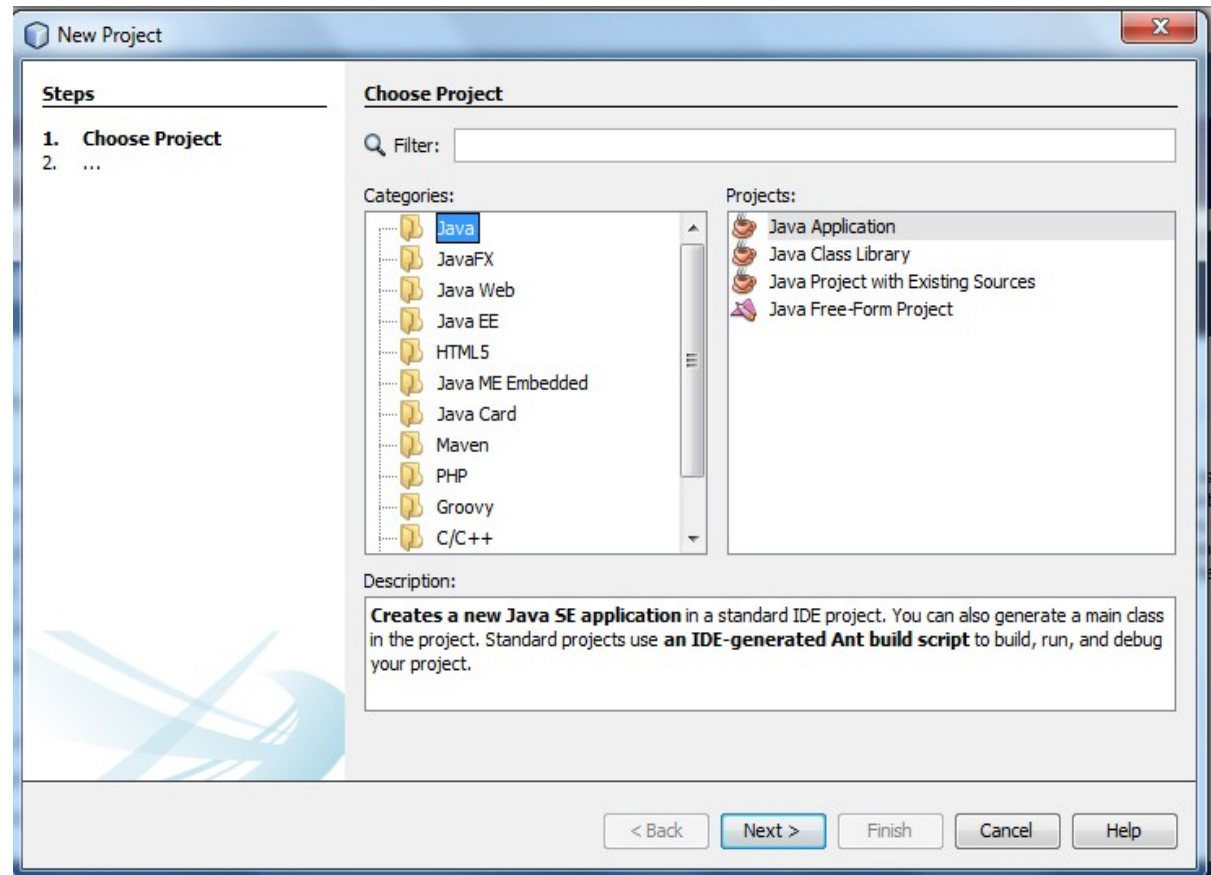
Objekt „zitterroche“



- Das Objekt „zitterroche“ speichert die ersten vier Basen.
- Der Instanz wird mit der Methode `.concat()` weitere vier Basen hinzugefügt.
- Mit Hilfe von `.charAt()` wird die erste Base ausgegeben.

Java-Projekt anlegen

- *File – New Project.*



1. Schritt: Auswahl einer Projektkategorie

- *Categories* „Java“. *Projects* „Java Application“.
- Das Grundgerüst eines Projekts wird entsprechend der ausgewählten Kategorie erzeugt.
- In diesem Beispiel werden die Ordner und Dateien für eine Konsolen-Anwendung automatisiert erstellt.

2. Schritt: Name und Speicherort des Projekts

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

< Back Next > Finish Cancel Help

Erläuterung

- *Project Name* legt den Namen des Ordners fest. Der Paketname des Java-Projekts wird festgelegt.
- *Project Location* und *Project Folder* legen den Speicherort des Java-Projekts fest.
- *Create Main Class* erzeugt die Klasse, die die Start-Methode enthält. Der Klassennamen beginnt mit einem Großbuchstaben und entspricht standardmäßig dem Paketnamen.

Deklaration von Objekt-Variablen

String	strWort	;
Kreis	kreis01	;
Klasse	variablenname	;

- Statt eines Datentyps wird ein Klassenname als Typ angegeben. Der Klassenname muss exakt wie im Kopf (`public class klasseName`) der Klasse beschrieben, geschrieben werden.
- Der Variablenname ist frei wählbar.

Initialisierung von Objekt-Variablen

kreis01	=	new	Kreis	()	;
variablenname	=	new	Klasse	()	;

- Das Schlüsselwort `new` erzeugt mit Hilfe eines Konstruktors eine Instanz von einer Klasse.
- Der Objekt-Variablen wird ein Verweis auf das neu erstellte Objekt übergeben.

Konstruktoren

- Spezielle Methode zur Erzeugung eines Objekts.
- Konstruktor und Klasse haben den gleichen Namen.
- Konstruktoren kann eine Parameterliste übergeben werden. Die Liste folgt direkt im Anschluss des Klassennamens. Die Parameterliste wird durch die runden Klammern begrenzt.
- Einem Konstruktor können Parameter übergeben werden. Mit Hilfe der Parameter werden Attribute des Objekts initialisiert. Wenn die Liste leer ist, werden die Attribute des Objekts nicht mit Werten von außen initialisiert.

Hinzufügung einer Klasse

- Rechter Mausklick auf den Projektnamen im Projekt-Explorer.
- *New – Java Class*.
- Eingabe eines Klassennamens in das Textfeld *Class Name*. Klassennamen beginnen immer mit einem Großbuchstaben.
- Klick auf die Schaltfläche *Finish*. Das Gerüst einer Klasse wird automatisiert erstellt.

Klassen

```
public class Kreis {  
  
}
```

- Jedes Java-Programm enthält mindestens eine Klasse. Diese Klasse enthält die Methode `main()` zum Starten des Projekts. In der Methode `main()` können wiederum Objekt-Variablen von anderen Klassen deklariert werden.
- Ein Projekt kann mehrere Klassen enthalten. Jede Datei mit der Endung „.java“ enthält eine Klasse.

Aufbau einer Klasse

```
public class Kreis {  
    double radius;  
    String farbe;  
  
    public Kreis(){  
        radius = 0;  
        linienfarbe = "schwarz";  
    }  
  
    public double getFlaeche(double radius){  
        double flaeche;  
  
        flaeche = pi * (this.radius *  
                        this.radius);  
        return flaeche;  
    }  
}
```

} Attribute / Eigenschaften

} Konstruktor

} Methoden

Kopf einer Klassen

```
public class Kreis {
```

- Jede Klasse wird durch das Schlüsselwort `class` gekennzeichnet.
- Jede Klasse hat einen eindeutigen Namen. In diesem Beispiel wird die Klasse `Kreis` implementiert.
- Das Schlüsselwort vor dem Namen regelt den Zugriff auf die Klasse von außen her.

Klassenname

- Beginn mit einem Großbuchstaben. Zum Beispiel „Kreis“.
- Nutzung der Kamel-Notation. Zum Beispiel „GeometrieKreis“. Jedes Wort beginnt mit einem Großbuchstaben.
- Der Name spiegelt die Bezeichnung eines Dinges in der realen Welt wieder.
- Der Name ist innerhalb eines Paketes eindeutig.

Öffentliche Klassen

```
public class Kreis {
```

- Das Schlüsselwort `public` kennzeichnet öffentliche Klassen.
- Die Klasse kann von jedem Paket genutzt werden.
- Die Klasse kann von allen anderen Klassen verwendet werden.

Private Klassen

```
private class Kreis {
```

- Das Schlüsselwort `private` kennzeichnet eine private Klasse.
- Wenn kein Zugriffsmodifikator angegeben wird, ist die Klasse nicht öffentlich.
- Die Klasse kann nur in dem Paket genutzt werden, in dem sie definiert ist.

Attribute (Instanzvariablen)

```
public class Kreis {  
    private final double PI = 3.14159;  
    private final String LINIENFARBE;  
  
    private double radius;  
    private String fuellfarbe;
```

- Daten in einer Klasse.
- Beschreibung eines Objekts.
- Variable oder konstante Attribute
- Attribute sind immer privat.

Deklaration von Variablen

<code>double</code>	<code>radius</code>	<code>;</code>
<code>typ</code>	<code>variablenname</code>	<code>;</code>

- Der Variablenname ist frei wählbar.
- Entsprechend des angegebenen Datentyps wird Speicher bereitgestellt.
- Es können primitive Datentypen oder Klassen genutzt werden.
- Die Variablen können im Konstruktor initialisiert werden. Variablen werden in Methoden verändert.

Deklaration von Konstanten

<code>final</code>	<code>double</code>	<code>PI</code>	<code>=</code>	<code>3.14159</code>	<code>;</code>
<code>final</code>	Datentyp	Konstanten-Name	<code>=</code>	wert	<code>;</code>

- Deklarationen von Konstanten beginnen mit dem Schlüsselwort `final`.
- Konstanten können von Methoden nicht verändert werden.
- Konstanten können gleichzeitig deklariert und initialisiert werden. Konstanten können aber auch in einem Konstruktor initialisiert werden.

Zugriffsmodifikatoren für Instanzvariablen

- Wenn kein Zugriffsmodifikator angegeben ist, kann das Attribut von allen Objekten aller Klassen im selben Paket genutzt werden.
- Private Attribute (`private`) können von keinen Objekten verwendet werden. Sie sind in der Klasse gekapselt. Die Attribute können nur mit Hilfe von öffentlichen Methoden verändert werden.
- Öffentliche Attribute (`public`) können von allen anderen Objekten verwendet werden. Attribute in einer Klasse sollten nie öffentlich sein.

Methoden in Java

```
public String getFuellfarbe()  
{  
    return this.fuellfarbe;  
}  
  
public void setFuellfarbe(String farbe)  
{  
    this.fuellfarbe = farbe;  
}
```

Methoden

- Abfolge von Anweisungen, die der Computer versteht.
- Zusammenfassung von Aktionen.
- Schnittstellen zum Benutzer.
- Lesen und Schreiben von Werten der gekapselten Instanzvariablen.

Hinweise

- Die Methoden sollten intuitiv zu bedienen sein .
- Die Methoden beschreiben das Verhalten eines Objekts so wie in der realen Welt. Die Addition von Werten subtrahiert in einer eigenen Klasse keine Werte.
- In der Beschreibung „Ein Kreis von einem Radius von 5 cm wurde mit schwarzer Farbe gezeichnet und mit einer roten Farbe ausgefüllt.“ deuten Verben auf bestimmte Methoden hin. Das Verb „zeichnen“ und „füllen“ haben Auswirkungen auf das Objekt „Kreis“. Der Radius kann verändert werden.
- Es werden nur die Methoden implementiert, die für die gestellte Aufgabe benötigt werden.

Aufbau von Methoden

```
public double getFlaeche(double radius)
```

```
{  
    double flaeche;  
  
    if (radius != this.radius)  
    {  
        this.radius = radius;  
    }  
  
    flaeche = PI * (this.radius * this.radius);  
    return flaeche;  
}
```

Methoden-
kopf

Methoden-
rumpf

Methodenrumpf

- Beginn und Ende mit den geschweiften Klammern.
- Zusammenfassung von Anweisungen für eine bestimmte Aktion.
- Die Aktion wird in dem Methodennamen beschrieben.

Variablen im Methodenrumpf

```
public double getFlaeche(double radius)
{
    double flaeche;
```

- Die, in einem Methodenrumpf deklarierten Variablen sind Platzhalter für lokale Werte.
- Die Variablen sind nur in der Methode gültig und sichtbar, in der sie deklariert sind.

Instantvariablen im Methodenrumpf

```
public double getFlaeche(double radius)
{
    double flaeche;
    flaeche = PI * (this.radius * this.radius);
}
```

- Instantvariablen beschreiben Attribute einer Klasse.
- Innerhalb der Klasse können Instantvariablen genauso wie lokale Variablen in einem Methodenrumpf genutzt werden.

Zuordnung zu einem Objekt

```
public double getFlaeche(double radius)
{
    double flaeche;
    flaeche = PI * (this.radius * this.radius);
}
```

- Instanzvariablen werden mit Hilfe des Punktoperators einem Objekt zugeordnet.
- Das Schlüsselwort `this` ist ein Platzhalter für das Objekt, welches die Methode aufgerufen hat. Zum Beispiel ruft das Objekt `kreisBlau` diese Methode auf. Das Attribut `radius` des Objekts `kreisBlau` wird zur Berechnung der Fläche genutzt.

Existenz von Variablen in Methoden

```
public double getFlaeche(double radius)
{
    double flaeche;
    flaeche = PI * (this.radius * this.radius);
}
```

- Eine Instanzvariable ist in der Klasse existent, in der sie deklariert ist.
- Eine lokale Variable ist nur in dem Block existent, in dem sie definiert wurde.

Sichtbarkeit von Variablen in Methoden

```
public double getFlaeche(double radius)
{
    double flaeche;
    flaeche = PI * (this.radius * this.radius);
}
```

- Eine lokale Variable ist nur in dem Block sichtbar, in dem sie definiert wurde.
- Eine Instanzvariable kann von einer lokalen Variablen gleichen Namens überdeckt werden. NetBeans zeigt eine Warnung an.
- Falls eine lokale Variable und eine Instanzvariable den gleichen Namen haben, kann die Instanzvariable nur in Verbindung mit `this` verwendet werden.

Methodenkopf

<code>public</code>	<code>void</code>	<code>setFuellfarbe</code>	<code>(</code>		<code>)</code>
zugriff	datentyp	methodenname	<code>(</code>		<code>)</code>

- Jede Methode beginnt mit dem Zugriffsmodifikator. Methoden sind häufig öffentlich (`public`).
- Dem Zugriffsmodifikator folgt der Datentyp der Methode.
- Dem Datentyp folgt der Methodenname. Der Methodenname ist eindeutig in einer Klasse.
- Dem Methodennamen folgt in runden Klammern die Parameterliste.

Methodennamen

- Beginn mit einem Kleinbuchstaben.
- Zusammengesetzte Namen: Das erste Wort beginnt mit einem Kleinbuchstaben. Alle anderen Wörter beginnen mit einem Großbuchstaben.
- Der Name spiegelt die Aktion wieder. Häufig wird ein Verb für einen Methodennamen genutzt.
- Der Name ist eindeutig in der Klasse.

Weitere Hinweise

- Methoden, die Werte für Instanzvariablen setzen, beginnen häufig mit „setze“ oder „set“.
- Methoden, die Werte für Instanzvariablen lesen, beginnen häufig mit „liefere“ oder „get“..
- Methoden, die zwei Werte vergleichen beginnen häufig mit „is“.

Zugriffsmodifikatoren

<code>public</code>	<code>void</code>	<code>setFuellfarbe</code>	<code>(</code>		<code>)</code>
<code>zugriff</code>	<code>datentyp</code>	<code>methodenname</code>	<code>(</code>		<code>)</code>

- Methoden sollten öffentlich (`public`) sein. Die Methode kann aus allen anderen Klassen aufgerufen werden.
- Private (`private`) Methoden werden für interne Berechnungen genutzt. Diese Methoden können nur von der Klasse aufgerufen werden, in der sie definiert sind.
- Wenn kein Zugriffsmodifikator angegeben ist, kann die Methode aus allen anderen Klassen im selben Paket aufgerufen werden.

Parameterliste

void	setFuellfarbe	(String fuellfarbe)
typ	methodenname	(typ para01, typ para02)

- Die Parameterliste beginnt und endet mit den runden Klammern.
- Die Parameterliste folgt direkt dem Namen der Methoden.
- Die Parameterliste kann beliebig viele Parameter von beliebigen Typ enthalten.

Parameter in der Parameterliste

<code>void</code>	<code>setFuellfarbe</code>	<code>(</code>	<code>String fuellfarbe</code>	<code>)</code>
<code>typ</code>	<code>methodenname</code>	<code>(</code>	<code>typ para01, typ para02</code>	<code>)</code>

- Jeder Parameter ist von einem bestimmten Typ. Dies kann ein primitiver Datentyp oder eine Klassenname sein.
- Die Parameter werden durch ein Komma getrennt.
- Parameter können nur in dem Methodenrumpf verwendet werden, zu dem der Methodenkopf gehört.
- Parameter sind lokale Variablen einer Methode.

Prozeduren

public	void	setFuellfarbe	()
zugriff	void	methodenname	()

- Jede Prozedur hat den Datentyp void.
- Prozeduren geben kein Wert an den Aufrufer zurück.
- Häufig setzen Prozeduren Instanzvariablen.

Aufruf von Prozeduren

```
Kreis kreisBlau;  
kreisBlau = new Kreis();  
  
kreisBlau.setFuellfarbe("blau");
```

- Objekt und Methode werden mit dem Punktoperator verbunden.
- Die Methode bezieht sich auf das Objekt, welches links vom Punkt steht.
- Entsprechend des Methodenkopfs werden der Methode Anfangswerte in der Parameterliste übergeben. Wenn der Methode keine Parameter übergeben werden, sind die Klammern leer.

Zuordnung der Parameter

- Die Parameter im Aufruf werden den Parameter im Methodenkopf von links nach rechts zugeordnet.
- Der erste Parameter beim Aufruf wird dem ersten Parameter im Methodenkopf zugeordnet und so weiter.
- Die Parameter beim Aufruf sollten entsprechend der Parameter im Methodenkopf interpretierbar sein.

Funktionen

public	double	getFlaeche	()
public	datentyp	methodenname	()

- Funktionen sind von einem primitiven Datentyp oder einer Klasse.
- Funktionen geben einen Wert an den Aufrufer zurück.
- Funktionen, deren Name mit „is“ beginnen, geben häufig einen boolschen Wert zurückgeben

Aufruf von Funktionen

```
Kreis kreisBlau;  
double flaeche;  
kreisBlau = new Kreis();  
  
flaeche = kreisBlau.getFlaeche(2);
```

- Objekt und Methode werden mit dem Punktoperator verbunden.
- Die Methode bezieht sich auf das Objekt, welches links vom Punkt steht.
- Entsprechend des Methodenkopfs werden der Methode Anfangswerte in der Parameterliste übergeben.
- Der Rückgabewert kann in einer Variablen gespeichert werden.

Rückgabewert in der Methode

```
public String getFuellfarbe()  
{  
    return this.fuellfarbe;  
}
```

- Mit Hilfe des Schlüsselwortes `return` wird ein Wert zurückgegeben.
- Das Schlüsselwort beendet die Methode.
- Der Rückgabewert kann entsprechend des Datentyps der Methode interpretiert werden.

Nutzung in switch-Anweisungen

```
public String getFuellfarbeToRGB()  
{  
    switch(this.fuellfarbe)  
    {  
        case "blau":  
            return "0,0,255";  
        case "rot":  
            return "255,0,0";  
        case "grün":  
            return "0,255,0";  
        default:  
            return "unbekannte Farbe";  
    }  
}
```

... beim Aufruf der Funktion

```
Kreis kreisBlau;  
double flaeche;  
kreisBlau = new Kreis();  
  
flaeche = kreisBlau.getFlaeche(2);
```

- Die Funktion kann überall dort verwendet werden, wo eine Variable verwendet werden kann.
- Falls der Rückgabewert in einer Variablen gespeichert wird, sollte der Datentyp der Variablen und der Funktion gleich sein.

Instanzmethode

```
public String getInfo()  
{  
    String ausgabe;  
    ausgabe = "\nFläche: " + this.getFlaeche(radius);  
}
```

- Innerhalb der Methode können weitere Methoden aufgerufen werden.
- Methoden, die in der gleichen Klasse definiert sind, werden als Instanzmethoden bezeichnet. In diesem Beispiel wird die Methode `getFlaeche()` aufgerufen.

... aufrufen

```
public String getInfo()  
{  
    String ausgabe;  
    ausgabe = "\nFläche: " + this.getFlaeche(radius);  
}
```

- Die Methode `getInfo()` wird durch ein Objekt aufgerufen. Zum Beispiel wird die Methode durch `blauKreis` aufgerufen. Das Schlüsselwort `this` ist ein Platzhalter für das Objekt `blauKreis`.
- Die Instanzmethode `getFlaeche()` soll auf die gleichen Attribute zugreifen. Der Punktoperator verbindet die Methode mit dem Objekt, welches die Methode `getInfo()` aufgerufen hat.

Konstrukturen

- Methoden zur Erzeugung von Instanzen.
- Zum Bau eines Objekts werden Anfangswerte übergeben.
- Der Name eines Konstruktoren ist immer gleich der Klasse, von der ein Objekt konstruiert werden soll.
- Automatischer Aufruf durch das Schlüsselwort `new()`.

Beispiel

```
public Kreis()  
{  
    radius = 0;  
    LINIENFARBE = "schwarz";  
}
```

- Konstruktoren sind immer öffentlich. Der Kopf eines Konstruktors beginnt immer mit dem Zugriffsmodifikator `public`.
- Dem Zugriffsmodifikator folgt der Name der Klasse, in dem der Konstruktor definiert wird.
- Dem Namen folgt die Parameterliste.

Parameterloser Konstruktor

```
public Kreis()  
{  
    radius = 0;  
    LINIENFARBE = "schwarz";  
}
```

- Die Parameterliste ist leer.
- In dem Rumpf des Konstruktors werden die Variablen und Konstanten auf allgemein gültige Werte gesetzt.
- Falls für eine Klasse kein Konstruktor vorhanden ist, wird automatisiert ein parameterloser Konstruktor angelegt.

Initialisierung von Instanzvariablen

```
public class Kreis {  
    private double radius;  
    private String fuellfarbe;  
  
    public Kreis(){  
        radius = 0;  
        LINIENFARBE = "schwarz";  
    }  
}
```

- Instanzvariablen können, müssen aber nicht in Konstruktoren initialisiert werden.

Initialisierung von Instanzkonstanten

```
public class Kreis {  
    private final double PI = 3.14159;  
    private final String LINIENFARBE;  
  
    public Kreis(){  
        radius = 0;  
        LINIENFARBE = "schwarz";  
    }  
}
```

- Konstanten, die nicht gleichzeitig deklariert und initialisiert wurden, müssen in einem Konstruktor initialisiert werden.
- Mit Hilfe des Zuweisungsoperators wird der Konstanten ein Wert zugewiesen.

Aufruf des parameterlosen Konstruktor

```
Kreis kreisBlau;  
kreisBlau = new Kreis();
```

- Mit Hilfe des Schlüsselwortes `new` wird ein Objekt von einer Klasse erzeugt.
- Dem Schlüsselwort folgt der Name der Klasse, die als Basis für die Instanz dient.
- Dem Namen folgen die leeren, runden Klammern. Die Parameterliste ist leer. Der parameterlose Konstruktor wird aufgerufen.

Konstruktor mit Parametern

```
public Kreis(double dblRadius, String linie)
{
    radius = dblRadius;
    LINIENFARBE = linie;
}
```

- Die Parameterliste folgt dem Namen des Konstruktors.
- In den runden Klammern werden die gewünschten Parameter, getrennt durch Kommata aufgelistet.
- Jeder Parameter hat einen eindeutigen Namen und Datentyp.
- Die Werte der Parameter werden Instanzvariablen und Konstanten zugewiesen.

Aufruf des Konstruktors


```
Kreis kreisBlau;  
kreisBlau = new Kreis(5, fuellfarbe);
```

- Mit Hilfe des Schlüsselwortes `new` wird ein Objekt von einer Klasse erzeugt.
- Dem Schlüsselwort folgt der Name der Klasse, die als Basis für die Instanz dient.
- Dem Namen folgen die leeren, runden Klammern. Die Parameterliste enthält die Parameter. Die Parameter werden durch ein Komma getrennt.

Zuordnung der Parameter

```
Kreis kreisBlau;  
kreisBlau = new Kreis(5, fuellfarbe);
```

```
public Kreis(double dblRadius, String linie)  
{
```



- Der erste Parameter im Aufruf wird dem ersten Parameter im Kopf des Konstruktors zugeordnet und so weiter.
- In Abhängigkeit der Anzahl der Parameter und deren Datentyp wird automatisiert der passende Konstruktor aufgerufen.

Klassenvariablen

```
static int anzahlKreise;
```

- Klassenvariablen sind Attribute, die alle Instanzen einer Klasse gemeinsam nutzen.
- Klassenvariablen haben für alle Instanzen der Klasse den gleichen Wert.
- Klassenvariablen behalten ihre Gültigkeit, solange wie ein Objekt von der Klasse existiert.

... deklarieren

```
static int anzahlKreise;
```

- Die Deklaration beginnt mit dem Schlüsselwort `static`.
- Dem Schlüsselwort folgt der Datentyp der Klassenvariablen.
- Der Name der Variablen muss eindeutig in der Klasse sein.
- Klassenvariablen müssen nicht initialisiert werden.

... setzen

```
public Kreis()  
{  
    Kreis.anzahlKreise++;  
}
```

- Klassenvariablen werden häufig in Konstruktoren gesetzt.
- Mit Hilfe des Punktoperators wird die Variable und die dazugehörige Klasse verbunden.

Klassenmethoden

- Statische Methoden können ausgeführt werden, bevor ein Objekt erzeugt wurde.
- Methoden, die unabhängig von den Instanzen einer Klasse sind.
- Klassenmethoden sind für die Klasse selbst und deren Instanzen gültig.

Beispiel: Start-Prozedur „main“

```
public static void main(String[] args) {  
    Kreis kreisBlau;  
    Kreis kreisRot;  
    Kreis kreisGruen;  
  
    kreisBlau = new Kreis();  
    kreisRot = new Kreis(5, "Rot");  
    kreisGruen = Kreis.copyKreis(kreisRot);  
}
```

Beispiel: „Kopieren von Objekten“

```
static Kreis copyKreis(Kreis quelle)
{
    if (quelle == null) {
        return null;
    }
    else
    {
        return new Kreis(quelle.radius,
            quelle.LINIENFARBE, quelle.fuellfarbe);
    }
}
```

Beispiel: Lesen von statischen Attributen

```
static int getAnzahl()  
{  
    return Kreis.anzahlKreise;  
}
```

- Hinweis: Statische Attribute können nur in Konstruktoren oder statischen Methoden gelesen oder gesetzt werden.

Methodenkopf

<code>static</code>	<code>Kreis</code>	<code>copyKreis</code>	<code>(</code>		<code>)</code>
<code>static</code>	<code>datentyp</code>	<code>methodenname</code>	<code>(</code>		<code>)</code>

- Klassenmethoden beginnen immer mit dem Schlüsselwort `static`.
- Dem Schlüsselwort folgt der Datentyp der Klassenmethode.
- Dem Datentyp folgt der Name der Methode.
- Dem Namen folgt die Parameterliste.

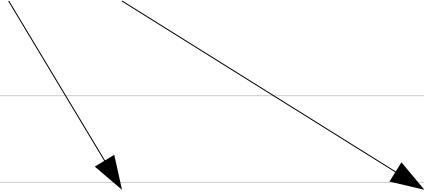
... aufrufen

```
Kreis kreisGruen;  
kreisGruen = Kreis.copyKreis(kreisRot);
```

- Die statische Methode wird über den Namen der Klasse aufgerufen.
- Der Punktoperator verbindet die Methode und die Klasse, in der die Methode definiert ist.

Primitive Datentypen

```
public static void main(String[] args) {  
    clsPunkt punktA = new clsPunkt();  
    double x = 2;  
    double y = 3;  
    punktA.setzePunkt(x, y);  
}
```



```
public void setzePunkt(double x, double y){  
    this.xPunkt = x;  
    this.yPunkt = y;  
}
```


Erläuterung

- Primitive Datentypen speichern Werte.
- Wenn primitive Datentypen an Methoden übergeben werden, wird in dem Parameter im Methodenkopf eine Kopie des Wertes abgelegt.
- Änderungen an dem Parameter (`x` in der Methode `.setzePunkt`) führt zu keinen Änderungen an dem Original (`x` in der Methode `.main`).

Referenzen

```
public static void main(String[] args) {  
    clsPunkt punktA = new clsPunkt();  
    clsPunkt punktB = new clsPunkt();  
  
    punktA.tauschePunkt(punktB);  
}
```

```
public void tauschePunkt(clsPunkt punkt){  
    clsPunkt tmpPunkt = this;  
  
    this.xPunkt = punkt.xPunkt;  
    this.yPunkt = punkt.yPunkt;  
    punkt.xPunkt = tmpPunkt.xPunkt;  
    punkt.yPunkt = tmpPunkt.yPunkt;  
}
```

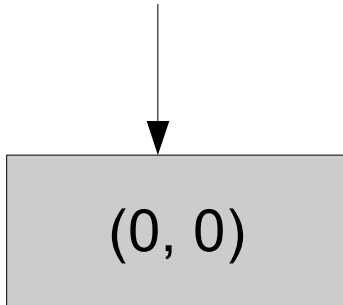


Was sind Referenzen?

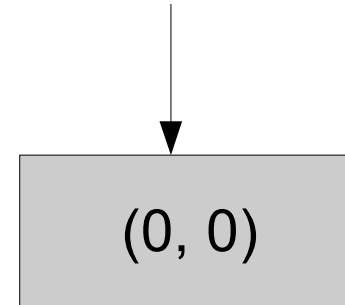
- Objekt-Variablen speichern Referenzen auf Objekte.
- Eine Referenz verweist auf die Adresse, an der das Objekt im Speicher abgelegt wurde.

Erzeugung und Setzen des Punkts

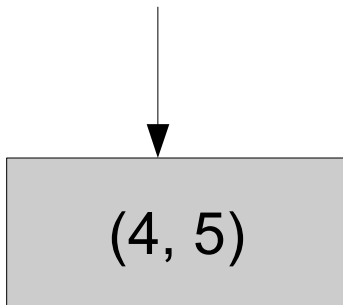
```
punktA = new clsPunkt()
```



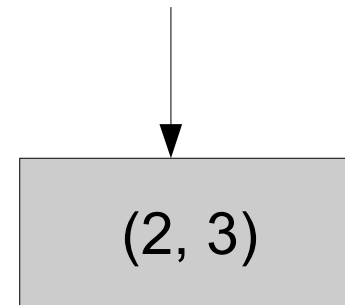
```
punktB = new clsPunkt()
```



```
punktA.setzePunkt()
```

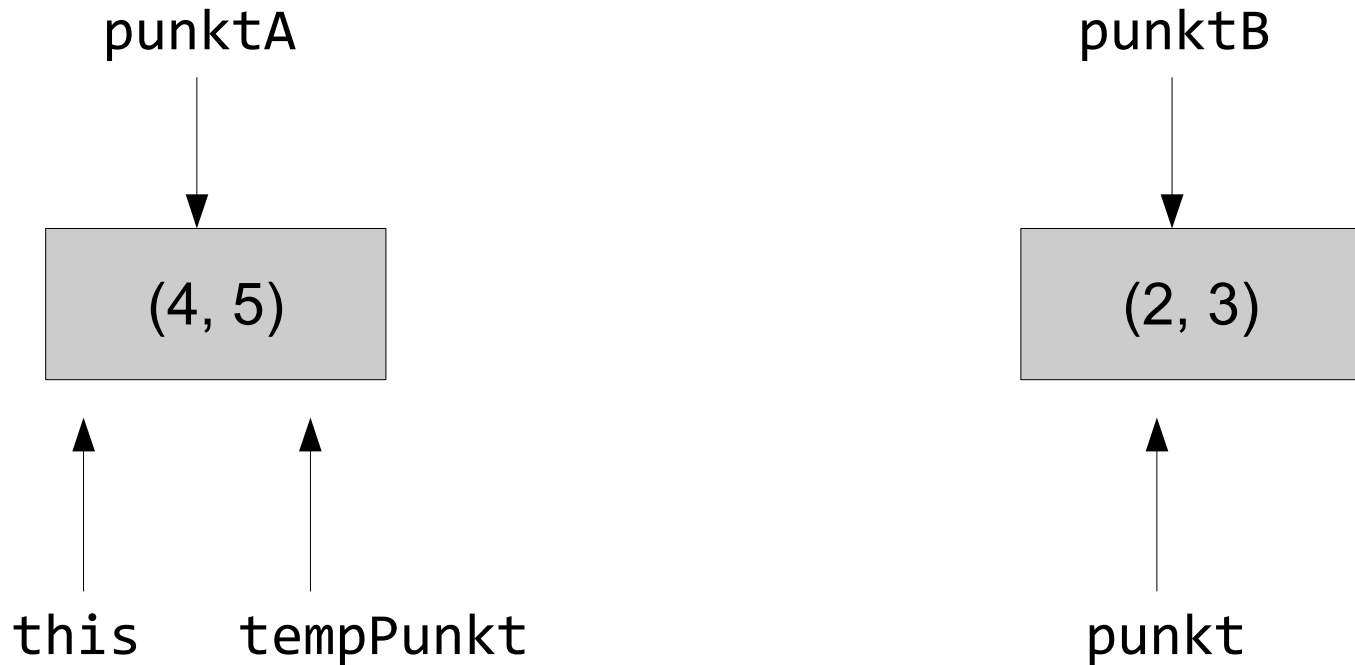


```
punktB.setzePunkt()
```



Aufruf der Methode

```
punktA.tauschePunkt(punktB);
```



Erste Zuweisung

```
this.xPunkt = punkt.xPunkt;
```

