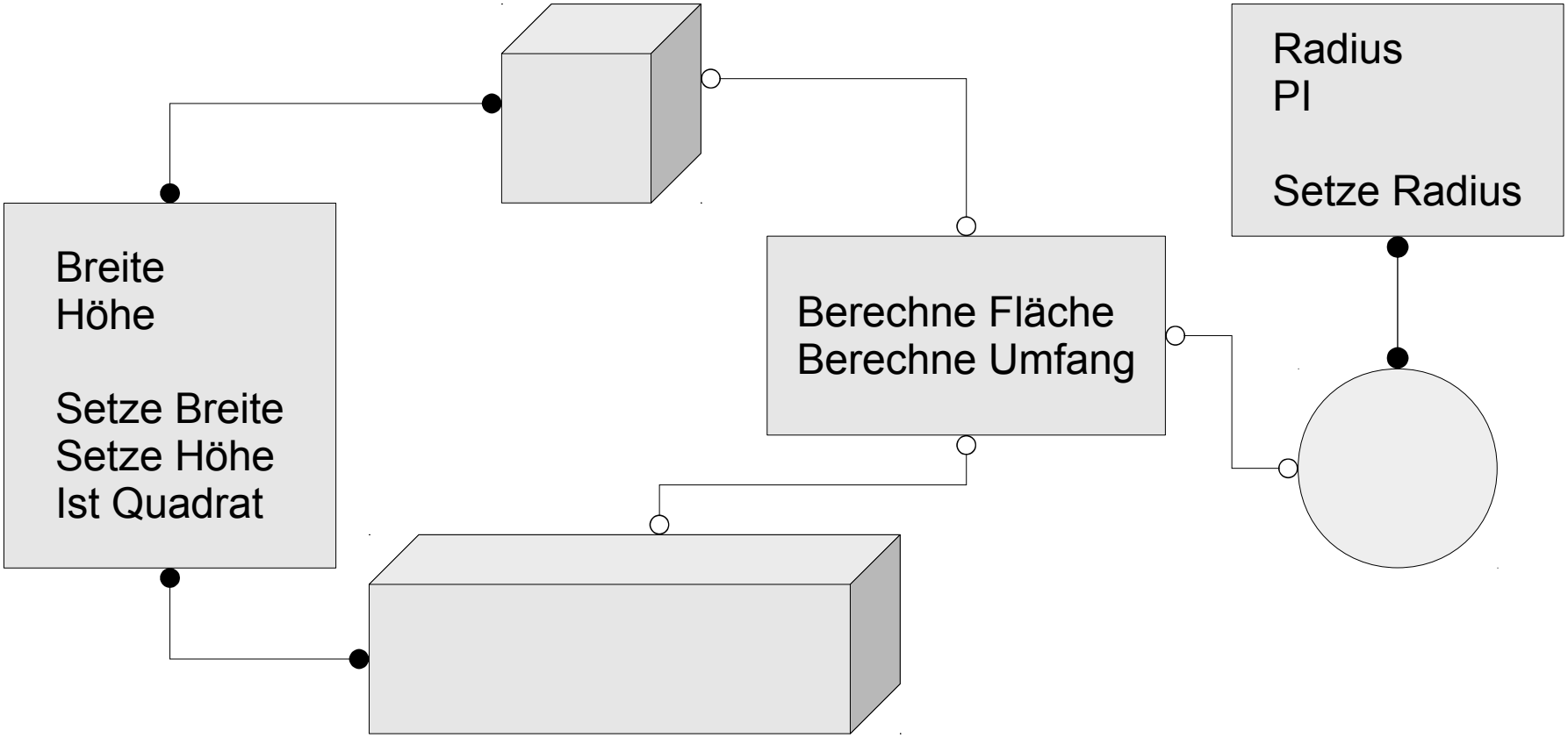
Java - Abstrakte Klassen und Interfaces



Abstrakte Klasse

- Vorlage für eine bestimmte Sammlung von Klassen.
- Basisklasse, von der keine Instanz mit Hilfe von `new()` erzeugt werden kann.
- Bereitstellung von Funktionalitäten, die jede Subklasse implementieren muss.
- Kategorisierung von Objekten. Rechtecke, Kreise etc. sind geometrische Formen.
- Generalisierung von Klassen.

… in Java

```
abstract class basisForm {
    public abstract double umfang();
    public abstract double flaeche();

    public void drucke()
    {
      System.out.printf("Fläche: %.2f", this.flaeche());
      System.out.printf("\nUmfang: %.2f", this.umfang());
      System.out.print("\n");
    }
}
```

Hinweise

- Abstrakte Klassen enthalten Attribute, Methoden und abstrakte Methoden.
- Instanzen können nicht von einer abstrakten Klasse erzeugt werden.
- Abstrakte Klassen können nur von einer abstrakten Klasse abgeleitet werden.
- Eine abstrakte Klasse kann nicht konstant sein.

Kopf einer abstrakten Klasse

`abstract`	`class`	`basisForm`
`abstract`	`class`	`Name`

- Jede abstrakte Klasse wird mit dem Schlüsselwort `abstract` gekennzeichnet.
- Dem Schlüsselwort `abstract` folgt das Schlüsselwort `class`. Eine abstrakte Klasse wird erstellt.
- Der Name der Klasse ist frei wählbar.

Abstrakte Methoden

`public`	`abstract`	`double`	`umfang`	`(`		`)`	`;`
`zugriff`	`abstract`	`datentyp`	`methodenname`	`(`		`)`	`;`

- Abstrakte Methoden werden immer mit dem Schlüsselwort `abstract` gekennzeichnet sein.
- Abstrakte Methoden werden immer mit dem Semikolon abgeschlossen.
- Abstrakte Methoden besitzen keinen Methodenrumpf. Sie werden in einer abstrakten Klasse deklariert.

Abgeleitete Klasse

```
public class clsKreis extends basisForm {
    private double radius;

    public clsKreis()
    {
        this.radius = 0;
    }

    public double umfang()
    {
        return(2 * Math.PI * this.radius);
    }

}
```

Kopf einer Subklasse

```
public class clsKreis extends basisForm {
```

- Jede Klasse wird durch das Schlüsselwort `class` gekennzeichnet.
- Jede Klasse hat einen eindeutigen Namen. In diesem Beispiel wird die Klasse `clsKreis` implementiert.
- Vor dem Schlüsselwort `class` wird ein Zugriffsmodifikator angegeben.

Angabe der Basisklasse

```
public class clsKreis extends basisForm {

```

- Die Subklasse basiert immer auf einer Basisklasse.
- Dem Schlüsselwort `extends` folgt der Name der Basisklasse.
- Die Basisklasse kann abstrakt sein.

Überschreiben der abstrakten Methode

```
public double umfang()
{
    return(2 * Math.PI * this.radius);
}
```

- Jede abstrakte Methode der Basisklasse muss überschrieben werden.
- Abstrakte Methoden müssen in der Subklasse definiert werden.

Interfaces (Schnittstellen)

- Schnittstellen sind immer öffentlich.
- Schnittstellen können von anderen Schnittstellen abgeleitet werden.
- Variablen können vom Typ einer Schnittstelle sein.
- Klassen können mehrere Schnittstellen implementieren.

… in Java

```
public interface IForm {
    public double umfang();
    public double flaeche();
    public void drucke();

}
```

Kopf einer Schnittstelle

`public`	`interface`	`IForm`
`zugriff`	`interface`	`Name`

- Jede Schnittstelle wird mit dem Schlüsselwort `interface` gekennzeichnet.
- Eine Schnittstelle muss immer öffentlich (`public`) sein.
- Der Name der Schnittstelle ist frei wählbar.

Konstruktoren in Schnittstellen

- Schnittstellen enthalten keine Konstruktoren.
- Variablen vom Typ einer Schnittstelle können deklariert werden.
- Aber Instanzen werden nicht von einer Schnittstelle erzeugt.

Methoden in Schnittstellen

- Schnittstellen enthalten nur abstrakte oder öffentliche Methoden.
- Der Name der Methode, die zu übergebenen Parameter und der Typ des Rückgabewertes wird festgelegt.
- Der Rumpf der Methode wird in der Klasse definiert, in der die Schnittstelle implementiert ist.

Attribute in Schnittstellen

- Schnittstellen habe keine Attribute.
- Alle Variable einer Schnittstelle müssen öffentlich und / oder konstant sein.
- Die Zugriffsrechte `private` oder `protected` für Variablen sind in einer Schnittstelle nicht erlaubt.

Implementierung von Schnittstellen

```
public class clsKreis implements IForm {
    private double radius;


    public void drucke()
    {
         System.out.printf("Radius %.2f", this.radius);
         System.out.printf("\nFläche: %.2f",
                           this.flaeche());
         System.out.printf("\nUmfang: %.2f",
                            this.umfang());
         System.out.print("\n");
    }
}
```

Kopf einer Klasse

```
public class klasse
{

```

- Jede Klasse wird durch das Schlüsselwort `class` gekennzeichnet.
- Jede Klasse hat einen eindeutigen Namen.
- Vor dem Schlüsselwort `class` wird ein Zugriffsmodifikator angegeben.

Angabe der Basisklasse

```
public class klasse extends basisklasse  {

}
```

- Die Subklasse basiert immer auf einer Basisklasse.
- Dem Schlüsselwort `extends` folgt der Name der Basisklasse.
- Die Basisklasse kann abstrakt sein.

Angabe der Schnittstelle

```
public class klasse extends basisklasse
                    implements interface
{

```

- Dem Schlüsselwort `implements` folgt eine Auflistung von Schnittstellen, die die Subklasse implementiert.
- Zuerst wird die Subklasse um die Basisklasse erweitert. Dann wird die Subklasse um die, zu implementierende Schnittstellen ergänzt.

Überschreiben der Schnittstellen-Methoden

```
public void drucke()
 {
      System.out.printf("Radius %.2f", this.radius);
      System.out.printf("\nFläche: %.2f",
                        this.flaeche());
      System.out.printf("\nUmfang: %.2f",
                        this.umfang());
      System.out.print("\n");
 }
```

Erläuterung

- Jede Methode, die in der zu implementieren Schnittstelle deklariert ist, muss definiert werden.

Nutzung von Annotationen

```
@Override
public void drucke()
{
      System.out.printf("Radius %.2f", this.radius);
      System.out.printf("\nUmfang: %.2f",
                        this.umfang());
      System.out.print("\n");
}
```

- Die Annotation `@Override` gibt einen Fehler aus, wenn die zu überschreibende Methode nicht in der Basisklasse vorhanden ist.

Annotationen

- Einführung mit Java 5.0.
- Beginn mit dem Add-Zeichen.
- Annotationen werden beim Kompilieren ausgewertet.
- Tutorial:
  https://docs.oracle.com/javase/tutorial/java/annotations/

Lambda-Ausdruck

- Einführung mit Java 8.
- Methoden ohne Namen und definierte Klasse.
- Transformierung von Methoden.

Funktionale Schnittstelle

```
@FunctionalInterface
public interface IPunkt {

    public abstract double verschiebePunkt
                (double punkt, double verschiebung);
}
```

- Funktionale Schnittstellen enthalten exakt eine abstrakte Methode.
- Die Schnittstellen werden mit der Annotation `@FunctionalInterface` gekennzeichnet.

Angabe der funktionalen Schnittstelle

```
public class clsKreis implements IForm, IPunkt { }
public class clsRechteck  implements Iform{ }
```

- Dem Schlüsselwort `implements` folgt eine Auflistung von Schnittstellen, die die Subklasse implementiert.
- Die Schnittstellen werden durch Kommata getrennt.
- Wenn Lambda-Ausdrücke genutzt werden, müssen funktionale Schnittstellen nicht in einer Klasse implementiert werden.

Überschreiben der abstrakten Methode

```
public class clsKreis  implements IForm, IPunkt {

  @Override
  public double verschiebePunkt(double punkt,
                                double verschiebung)
  {
       return (punkt + verschiebung);
  }
```

Erläuterung

- Im Klassenkopf ist die Schnittstelle `IPunkt` implementiert.
- Die, in der Schnittstelle definierte abstrakte Methode muss überschrieben werden.
- Die überschriebene Methode wird mit der Annotation `@Override` gekennzeichnet.

Nutzung eines Lambda-Ausdrucks

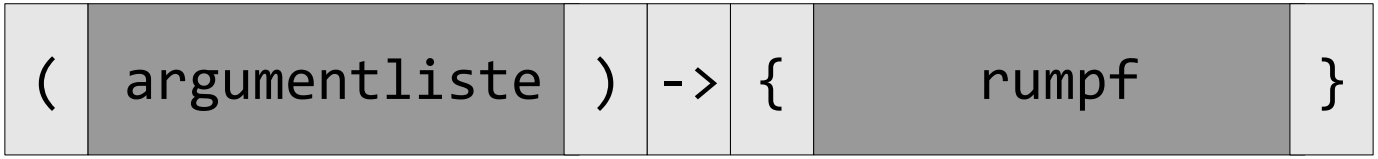
```
public class clsRechteck  implements Iform{

   IPunkt verschieben =
        (double koordinate, double wert)→
        {
                double neuKoordinate = koordinate + wert;
                return neuKoordinate;
        };
```

Erläuterung

- Falls ein Lambda-Ausdruck genutzt wird, muss Schnittstelle `IPunkt` im Klassenkopf nicht implementiert werden.
- Die, in der Schnittstelle definierte abstrakte Methode wird mit Hilfe des Lambda-Ausdrucks definiert.
- Die Referenz auf den Lambda-Ausdruck wird in einer Variablen gespeichert, die vom Typ der Schnittstelle `Ipunkt` ist.

Aufbau eines Lambda-Ausdrucks



- Der Pfeil-Operator verbindet die Argumentliste mit dem Rumpf der Funktion.
- Links vom Pfeil-Operator werden die Argumente des Ausdrucks mit Hilfe der runden Klammern zusammengefasst.
- Rechts von der Pfeilliste wird in den geschweiften Klammern die, in der Schnittstelle definierte abstrakte Methode überschrieben.

Argumentliste des Lambda-Ausdrucks

```
IPunkt verschieben =
        (double koordinate, double wert)→
        {
        };
```

- Die runden Klammern fassen Argumente eines Lambda-Ausdrucks zusammen.
- In den runden Klammern werden die Argumente durch ein Komma getrennt.
- Die Argumente werden in der Form `datentyp name` definiert.

Rumpf des Lambda-Ausdruckes

```
IPunkt verschieben =
      (double koordinate, double wert)→
      {
        double neuKoordinate = koordinate + wert;
        return neuKoordinate;
       };
```

- Der Rumpf, rechts vom Pfeil-Operator, beschreibt die gewünschte Funktionalität
- Der Rumpf enthält Anweisungen, die mit einem Semikolon beendet werden.
- Der Code-Block wird durch die geschweiften Klammern zusammengefasst.

Variable von der Schnittstelle ...

```
IPunkt verschieben =
      (double koordinate, double wert)→
      {
            double neuKoordinate = koordinate + wert;
            return neuKoordinate;
      };
```

- Die Variable `verschieben` ist vom Typ `IPunkt`.
- In dieser Schnittstelle ist die zu implementierende abstrakte Methode deklariert.
- Die Variable verweist auf einen Lambda-Ausdruck, der die Methode definiert.

… in der Klasse nutzen

```
public void verschiebeKoordinateX(double abstand)
{
   this.xKoordinate =
      verschieben.verschiebePunkt(this.xKoordinate,
                                  abstand);
}
```

Erläuterung

- Die abstrakte Methode `verschiebePunkt` in der Schnittstelle `IPunkt` wird mit Hilfe einer Variablen `verschiebe` aufgerufen.
- Die Variable `verschiebe` verweist auf einen Lambda-Ausdruck.
- Dieser Lambda-Ausdruck wird als Implementierung der abstrakten Methode `verschiebePunkt` genutzt.