Aufzählungstypen
enum

Beispiel

```
public class EnumKlasse {
enum Richtungen {
  links, rechts, vor, zurueck
};
}
public class Fahrzeug {
      private EnumKlasse.Richtungen richtung;
      …
public static void main(String[] args) {
            EnumKlasse.Richtungen richtung = null;
            Fahrzeug auto = new Fahrzeug();
            auto.setRichtung(richtung.links);
            System.out.println(auto);
}
```

Zwischenaufgabe

Ändern Sie den Status des Smartphones in ein enum mit den drei Zuständen „ein“, „aus“, „standby“.

Klassenattribute, -methoden

Unabhängig von Instanzen, Schlüsselwort **`static`**

Beispiel: Math.*PI*

Eigenes Beispiel: Instanzenzähler

```
public class Smartphone {
     …
     private static int anzahl; // Klassenattribut
     public static int getAnzahl() { // Klassenmethode
        return anzahl;
     }
     …
     public Smartphone(int lautstaerke, String telefonnummer) {
          …
          anzahl++;
     }
     public Smartphone() {
          …
          anzahl++;
     }
}
```

Aufruf

```
// Array von Smartphones
Smartphone [] smartphones = new Smartphone [10];
for (int i = 0; i < smartphones.length; i++) {
   smartphones[i] = new Smartphone(); //
   Instanziierung
}

// Aufruf der Klassenmethode direkt auf Klassenname
System.out.println("Es wurden " +
   Smartphone.getAnzahl() + " Smartphones
   instanziiert");
```

Zwischenaufgabe

1. Erstellen Sie in einer Klasse StaticTest zwei Zähler, einen als Instanzenattribut, den anderen als Klassenattribut, sowie für beide den zugehörigen Getter. Beide Attribute sollen im Konstruktor um eins erhöht werden.
2. Testen Sie in einer Klasse mit main-Methode die Erzeugung mehrerer Instanzen der Klasse StaticTest und geben die Werte der Zählerattribute aus.

Vererbung

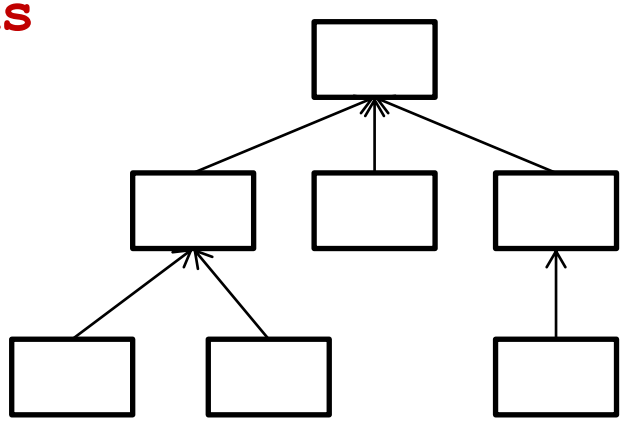
Zweck:

- Zusammenfassen von Gemeinsamkeiten mehrerer Klassen in einer neuen Oberklasse (**Generalisieren**)
- Ableiten von einer existierenden Oberklasse um deren Eigenschaften und/oder Methoden nutzen zu können (**Spezialisieren**): `extends`

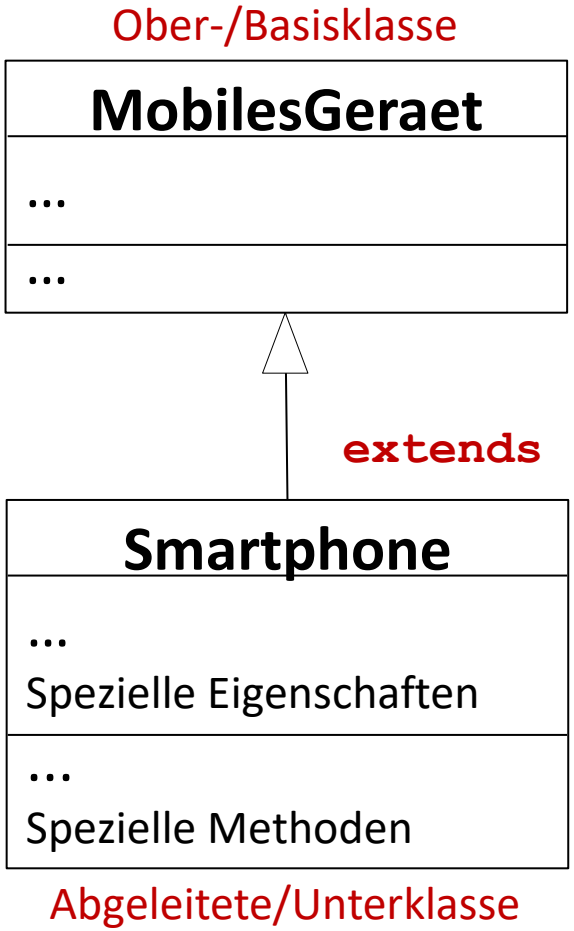
Fortgesetzte Vererbung
→ **Vererbungshierarchie**

KEINE Mehrfachvererbung

Vererbung verbieten: `final`

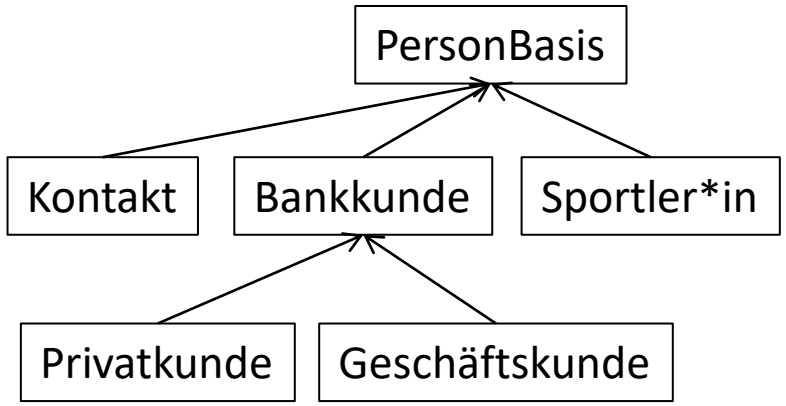


Beispiel 1


Ober-/Basisklasse
MobilesGeraet
...
...
extends
Smartphone
...
Spezielle Eigenschaften
...
Spezielle Methoden
Abgeleitete/Unterklasse

Beispiel 2: Oberklasse

```
public class PersonBasis {
	// Instanzenvariable
	private String vorname;
	private String nachname;
	// Konstruktoren
	public PersonBasis() {
		this.vorname = "";
		this.nachname = "";
	}
	public PersonBasis(String vorname, String nachname)
	{
		this.vorname = vorname;
		this.nachname = nachname;
	}
	// Setter und Getter, toString
	...
}
```



Beispiel 2: Unterklasse

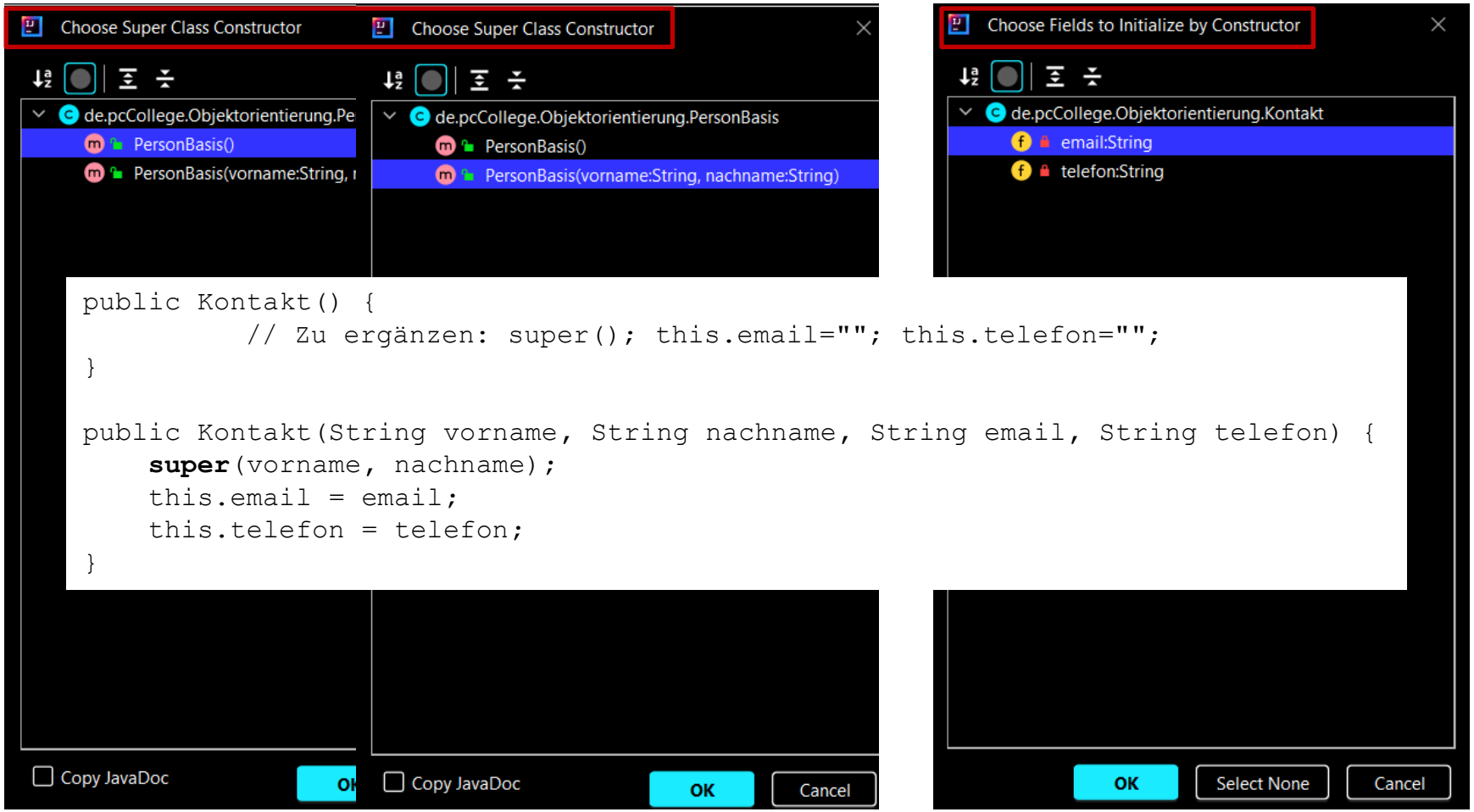
```
public class Kontakt extends PersonBasis {
    // Instanzenvariable
    private String email;
    private String telefon;
    // Konstruktoren
        ...
    // Setter und Getter
       public String getEmail() {
           return email;
       }

       public void setEmail(String email) {
           this.email = email;
       }

       public String getTelefon() {
           return telefon;
       }

       public void setTelefon(String telefon) {
           this.telefon = telefon;
       }
}
```

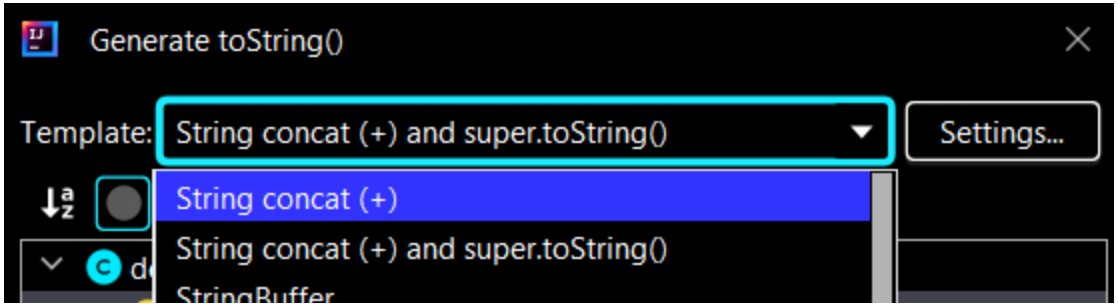
Beispiel 2: Konstruktoren




```
public Kontakt() {
          // Zu ergänzen: super(); this.email=""; this.telefon="";
}

public Kontakt(String vorname, String nachname, String email, String telefon) {
    super(vorname, nachname);
    this.email = email;
    this.telefon = telefon;
}
```

Beispiel 2: `toString`




```
@Override
public String toString() {
        return "Kontakt{" +
                "email='" + email + '\'' +
                ", telefon='" + telefon + '\'' +
                "} " + super.toString();
}
```

Bedeutung von `@Override`:

Gleichlautende Methoden der Unterklasse können anders implementiert sein

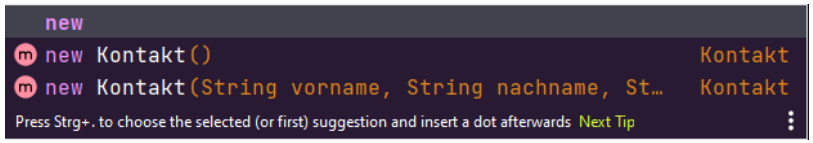
Methoden der Oberklasse(n)

- Eltern: `super`
- Großeltern: `super.super`

Vgl. www.geeksforgeeks.org/accessing-grandparents-member-in-java-using-super

Beispiel 2: Instanziierung

```
public class Adressbuch {
	public static void main(String[] args) {
		Kontakt [] k = new Kontakt[2];
		k[0] = new Kontakt();
		k[0].setVorname("Hans");
		k[0].setEmail("h@web.de");
		k[1] = new Kontakt("Susi", "Maier",
   "s@web.de", "12345");
		for (int i = 0; i < k.length; i++) {
			System.out.println(k[i]);
		}
	}
   }
```

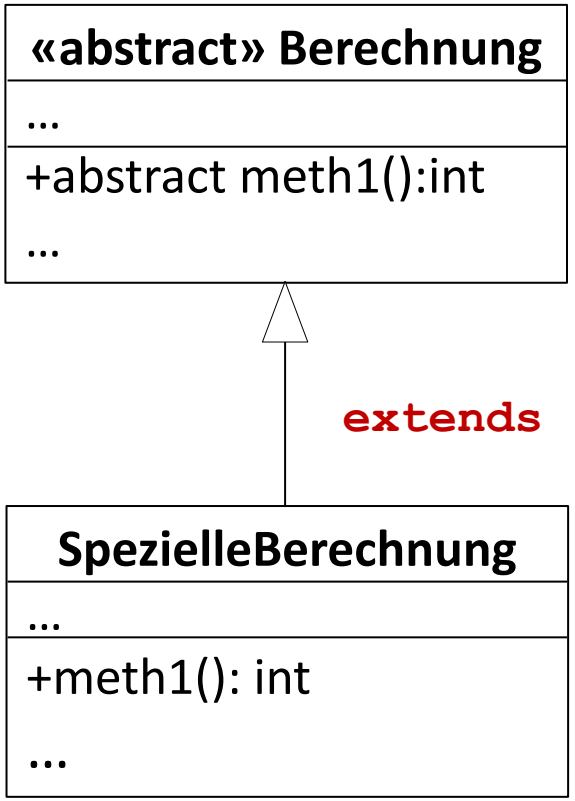


Aufgabenblatt 5

Siehe pdf-Datei

Abstrakte Klasse

- Nicht instanziierbar
- Enthält abstrakte Methoden
  (nur Methodenkopf)
- Verpflichtet nicht abstrakte Unterklassen
  zur Implementierung
  der abstrakten Methoden
- Kann auch Attribute, Konstruktoren
  und Methoden mit Implementierung
  enthalten
- Unterklassen können nur
  von einer (abstrakten) Klasse erben



Beispiel

```
public abstract class Ticket {
      private String eventOrt;
      private String eventName;
      protected int basisPreis;
      protected int ticketPreis;
      public Ticket (String ort, String name,
           int preis) {
          eventOrt = ort;
          eventName = name;
          basisPreis = preis;
      }
      // Abstrakte Methode
      public abstract int
  berechneTicketpreis();
}
```

Implementierungen

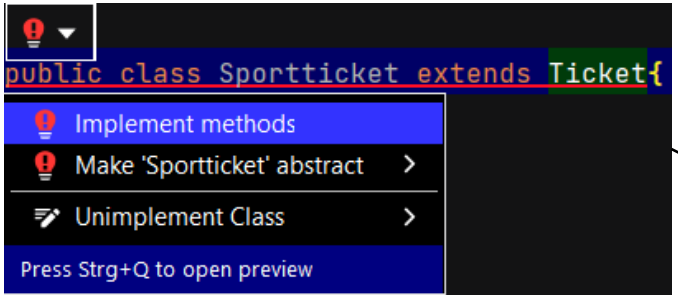
Klasse	Instanzenattribut, Konstruktor	Methode berechneTicketpreis

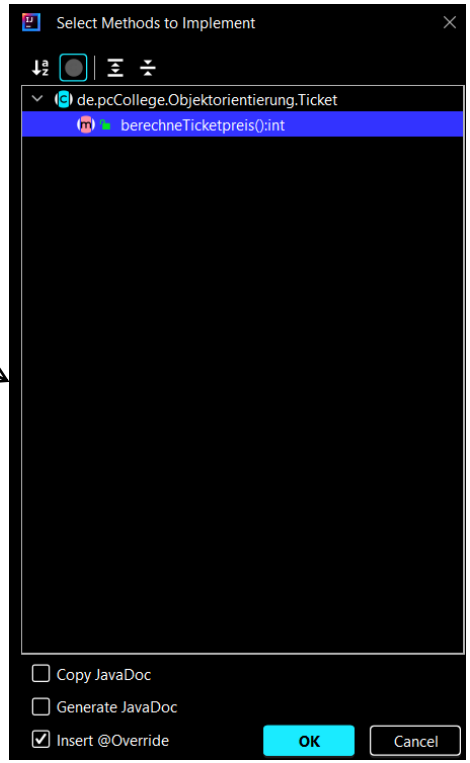
SportTicket	`private int pokalStufe;`
	`public SportTicket(String ort, String name, int preis, int stufe) {`
	`  super(ort, name, preis);`
	`  pokalStufe = stufe;`
	`}`
		`ticketPreis = basisPreis + (10*pokalStufe);`
		`return ticketPreis;`

KonzertTicket	**`private int sitzReihe;`**
	`public KonzertTicket(String ort, String name, int preis, int reihe) {`
	`  super(ort, name, preis);`
	`  sitzReihe = reihe;`
	`}`
		`ticketPreis = basisPreis*(1 + 1/sitzReihe);`
		`return ticketPreis;`

KinoTicket	**`private int filmDauer;`**
	`public KinoTicket(String ort, String name, int preis, int dauer) {`
	`  super(ort, name, preis);`
	`  filmDauer = dauer;`
	`}`
		`ticketPreis = basisPreis;`
		`if (filmDauer > 150) {`
		`  ticketPreis += 3;`
		`}`
		`return ticketPreis;`

Beispiel: Sportticket





Ergebnis:

```
@Override
public int berechneTicketpreis()
{
        return 0;
}
```

Ist durch korrekte Implementierung zu ersetzen

Aufrufende Klasse TicketVerkauf

```
public static void main(String[] args) {
   Ticket [] vTickets = new Ticket[8];
   vTickets[0] = new SportTicket("LONDON", "CHE-MAN", 50, 4);
   vTickets[1] = new KonzertTicket("BERLIN", "BACH", 74, 37);
   vTickets[2] = new KinoTicket("DRESDEN", "HONEY", 9, 165);
   vTickets[3] = new SportTicket("ERFURT", "ERF-DYN", 12, 0);
   vTickets[4] = new KonzertTicket("STUTTGART", "TARZAN", 99, 8);
   vTickets[5] = new SportTicket("BARCELONA", "BAR-MAD", 125, 7);
   vTickets[6] = new KonzertTicket("PORTO", "MARIA PIRES", 79, 1);
   vTickets[7] = new KinoTicket("PARIS", "PANEM", 10, 142);
   double umsatz = 0;
   for (int i = 0; i < vTickets.length; i++) {
        umsatz += vTickets[i].berechneTicketpreis();
   }
   System.out.println("Umsatz " + umsatz);
}
```

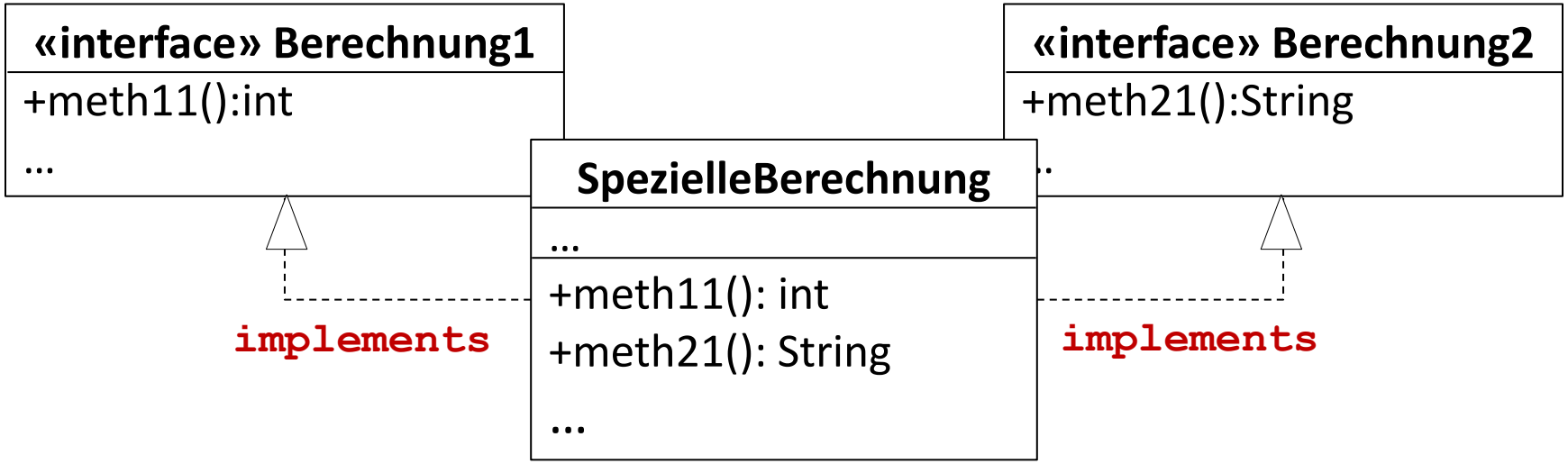
Aufgabenblatt 6

Siehe pdf-Datei

Interface

Charakterisierung

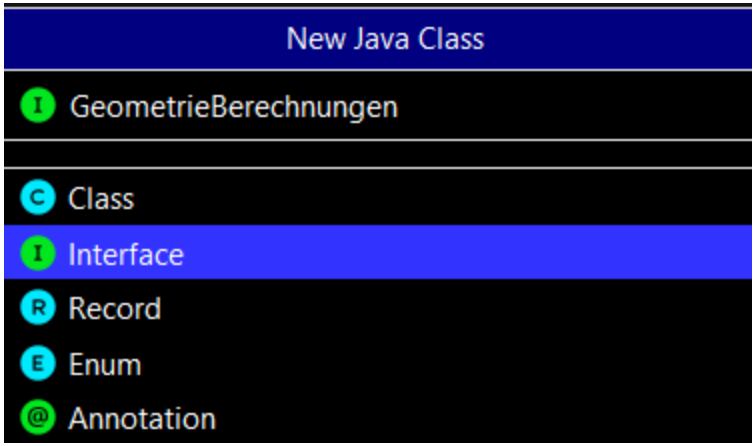
- Nicht instanziierbar
- Enthält **nur** Methodenköpfe:
  `Zugriffsmodifizierer Datentyp/void Methodenname(Aufrufparameter);`
- Verpflichtet Unterklassen zur Implementierung aller Methoden
- Klassen können **mehrere** Interfaces implementieren



Verfügbare Interfaces

- Selbst programmierte
- Java API, z. B.
  - `Comparable` } Package java.lang
  - `Runnable` } Package java.lang
  - `Serializable` Package java.io
  - GUI-Listener Package java.awt.event

Beispiel:
Erstellung eines eigenen Interfaces

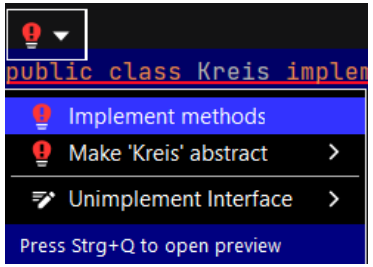


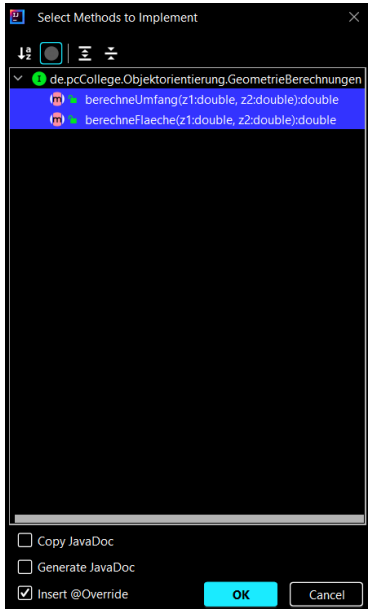

```
public interface GeometrieBerechnungen
{
   // Methodenköpfe
   public double berechneUmfang(double z1, double z2);
   public double berechneFlaeche(double z1, double z2);
}
```

Beispiel: Einbindung eines Interfaces

```
public class Kreis implements GeometrieBerechnungen{
   @Override
   public double berechneUmfang(double z1,
                                double z2)
   {
       return 0;
       // Ersetzen durch return 2*Math.Pi*z1;
   }

   @Override
   public double berechneFlaeche(double z1,
                                 double z2)
  {
       return 0;
       // Ersetzen durch return Math.Pi*z1*z1;
   }
}
```





Aufgabenblatt 7

Siehe pdf-Datei