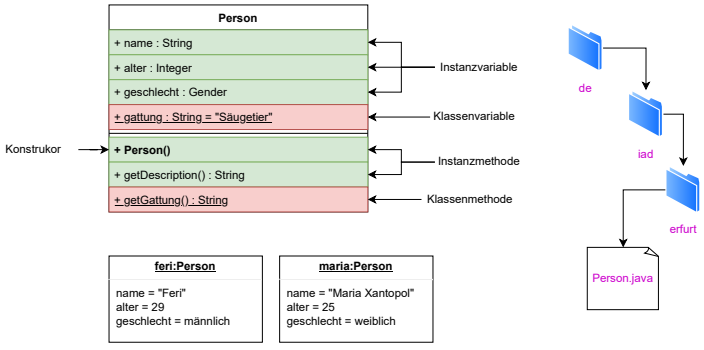
Ein Objekt ist eine Instanz einer Klasse. Erzeugt man ein neues Objekt, spricht man auch vom Instanziieren (Instanziierung).

```
// Der Operator new erzeugt ein neues Objekt einer Klasse und führt den dazugehörigen Konstruktor aus.
// Der Compiler wählt den Konstruktor anhand der angegebenen Argumente aus.
Klasse referenzVariable = new Klasse(argumente);
```

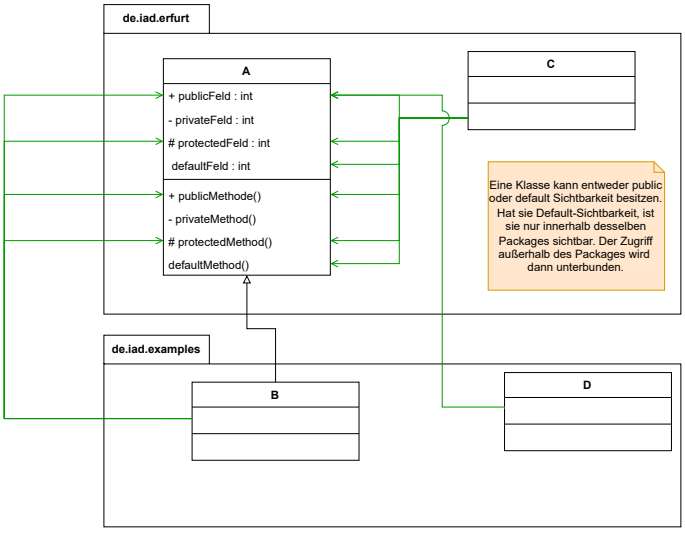
	Abrufen	Speicherort	Schlüsselwort this	Kennzeichnung in Java und C#	Zugriff auf
Klassenvariable	*Klasse*.klassenvariable	Wert wird nur einmal direkt in der Klasse gespeichert	Kann nur mit Klassenname angesprochen werden.	static	-
Instanzvariable	**objekt**.instanzvariable	Individueller Wert wird pro Objekt gespeichert	Kann mit oder ohne des Schlüsselwortes this angesprochen werden.	-	-
Klassenmethode	*Klasse*.klassenmethode()	wird in Form von Bytecode einmal zentral gespeichert	Es gibt kein this, da die Methode auf einer Klasse aufgerufen wird.	static	Klassenmethoden und Klassenfelder von *Klasse*
Instanzmethode	**objekt**.instanzmethode()	wird in Form von Bytecode einmal zentral gespeichert	this repräsentiert das **objekt** auf dem die Methode aufgerufen wurde.	-	Instanzfelder von **objekt**, Instanzmethoden von **objekt**, Klassenfelder von *Klasse*, Klassenmethoden von Klasse




```java
package de.iad.erfurt;

class Person {
 // Instanzvariablen
 public String name;
 public int alter;
 public Gender geschlecht;
 Person partner; // Zugriffsmodifizier default!
 // Klassenvariablen
 public static String gattung = "Säugetier";
 // Instanzmethoden
 public Person() {
  this.name = "Unbekannt";
  this.alter = 0;
  this.geschlecht = Gender.DIVERSE;
 }
 public String getDescription() {
  return this.name + " " + this.alter;
 }
 // Klassenmethoden
 public static String getGattung() {
  return Person.gattung;
 }
}
```

Zugriffsmodifizierer in Java (und dahinter in UML Notation)	Orte, an denen der Zugriff gestattet ist.
public (+)	überall
private (-)	nur innerhalb der Klasse, die das Feld bzw. die Methode definiert.
protected (#)	im gesamten **Package**, in dem sich die Feld-definierende Klasse befindet, sowie in allen Unterklassen bzw. Nachfahrenklassen.
default (~) (ohne Angabe eines Modifizierers gilt Default)	im gesamten **Package**, in dem sich die Feld-definierende Klasse befindet.



Man unterscheidet zwischen **konkreten** und **abstrakten** Klassen. **Eine abstrakte Klasse dient nur als Oberklasse**. Sie ist in der Regel **unvollständig** und muss von den Unterklassen ergänzt werden. Im Gegensatz zu einer konkreten Klasse, darf eine **abstrakte Klasse auch "Lücken", also nicht implementierte abstrakte Methoden besitzen**. Da abstrakte Klassen unvollständig sind, kann man **keine Objekte von ihnen erzeugen**.

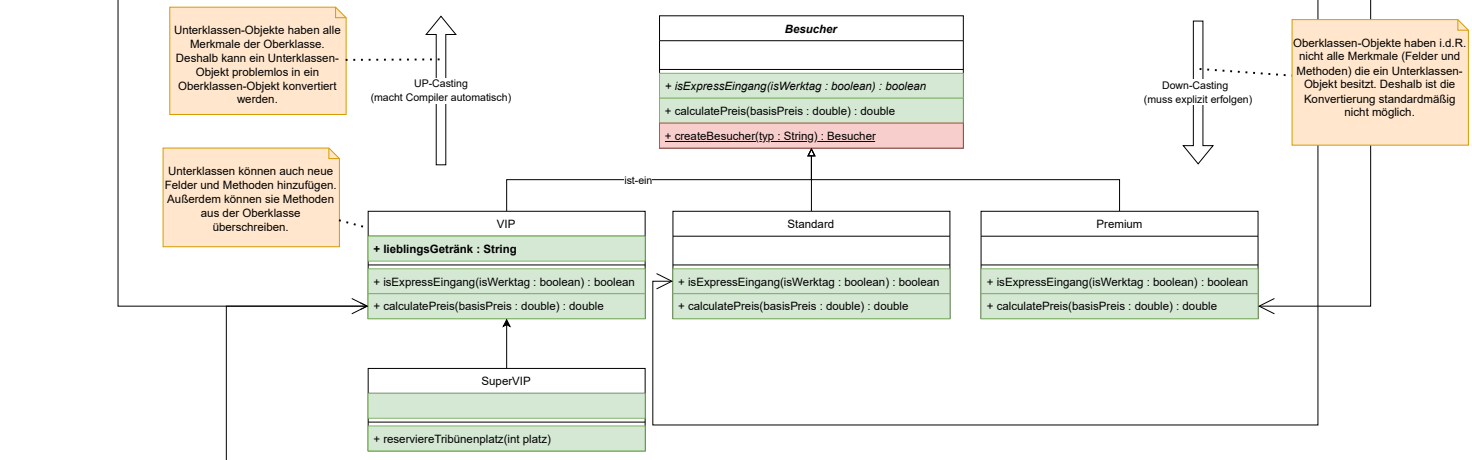
```
abstract class Besucher {
 // Konkrete Instanzmethoden (sie besizen einen Rumpf/Block, also eine Implementierung)
 public double calculatePreis(double basisPreis) {
  return basisPreis;
 }
 // Abstrakte Instanzmethoden (sie besitzen keine Implementierung und müssen von Unterklassen implementiert werden)
 public abstract boolean isExpressEingang(boolean isWerktag);
 // Klassenmethoden
 public static Besucher createBesucher(String typ) {
  return switch (typ) {
   case "VIP" -> new VIP();
   case "PREMIUM" -> new Premium();
   default -> new Standard();
  };
 }
}
```

Eine Form von **Polymorphie** ist das sogenannte **Sub-Typing**. Das bedeutet, dass einer Referenzvariablen von Klasse K ein Objekt einer beliebigen Unterklasse von K zugewiesen werden kann.
Bezogen auf das untere Beispiel heißt das: Der Variablen b vom Typ Besucher kann wahlweise ein Objekt vom Typ VIP, Standard und Premium zugewiesen werden, sowie Objekte sämtlicher Nachfahren von Besucher.

```
class Programm {
 public static void main(String[] argumente) {
  Besucher vipBesucher = Besucher.createBesucher("VIP");
  Besucher premiumBesucher = Besucher.createBesucher("PREMIUM");
  double vipPreis = vipBesucher.calculatePreis(10);
  double premiumPreis = premiumBesucher.calculatePreis(20);
  Besucher standardBesucher = new Standard(); // Automatisches Up-Casting.
  boolean isExpressEingangOffen = standardBesucher.isExpressEingang(true);
  VIP andererBesucher = new Besucher(); // Fehler: Down-Casting findet nicht automatisch statt!
  Besucher b = new VIP(); // OK: Up-Casting findet automatisch statt.
 }
}
```

DYNAMISCHES BINDEN
(zur Laufzeit wird bestimmt, zu welcher Methode tatsächlich gesprungen wird)

Die Klasse Besucher hat die Kinder VIP, Standard und Premium. VIP hat wiederum ein Kind namens SuperVIP.
VIP, Standard, Premium und SuperVIP sind **Nachfahren** von Besucher (engl. descendants).
Besucher ist ein **Vorfahre** von VIP und SuperVIP sowie Standard und Premium.




```
class Programm {
 public static void main(String[] argumente) {
  // Mit der Variable b kann nur auf die Merkmale (Felder, Methoden) der Klasse Besucher zugegriffen werden.
  // Es spielt keine Rolle, ob b dabei auf eine Unterklasse von Besucher verweist!
  Besucher b = new SuperVIP(); // OK: Up-Casting (Umwandlung von SuperVIP nach Besucher)
  b.isExpressEingang(true);
  b.calculatePreis(10);
  b.reserviereTribünenplatz(8); // Fehler: Kann nur auf Merkmale der Klasse Besucher zugreifen!
  Besucher.createBesucher("VIP");
  SuperVIP superVip = new VIP(); // Fehler: Kann ein Objekt der Oberklasse nicht einer Variablen der Unterklasse zuweisen!
  superVip.isExpressEingang(true);
  superVip.calculatePreis(10);
  superVip.reserviereTribünenplatz(8);
  superVip.lieblingsgetränk = "Pils";
}
```

Es wird immer die zuletzt überschriebene ("neuste") Variante einer Methode aufgerufen. Die Suche beginnt bei Klasse SuperVIP und geht dann klassenweise nach oben in Richtung Stammklasse.