Klassendefinitionen

Grundbegriffe:

Felder

Konstruktoren

Methoden

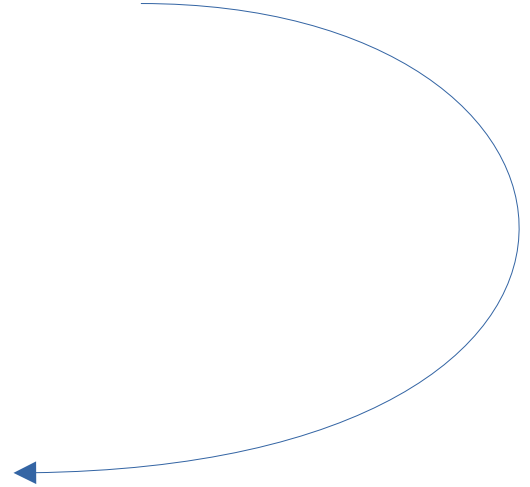
Parameter

Zuweisungsoperationen

Struktur einer Klasse

```
public class TicketMachine
{
    [...]
}
```

```
public class ClassName
{
    Felder
    Konstruktor
    Methoden
}
```



Schlüsselworte (Reservierte Worte)

Keywords oder reservierte Worte „gehören dem Compiler“…

Zum Beispiel:

```
public
class
private
int
```

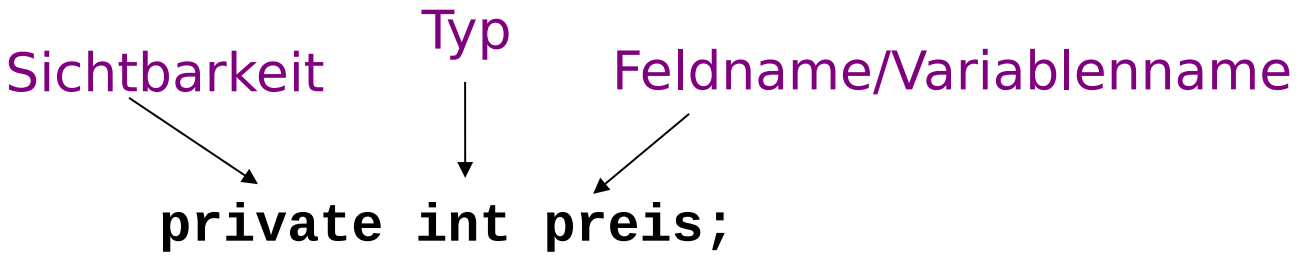
Felder (Instanzvariablen)

„Felder“, „Feldvariablen“ oder **„Instanzvariablen“** speichern Wert für ein Objekt, also eine Instanz einer Klasse.

Felder legen den **Zustand** eines Objekts fest

Mit **Introspektion** kann man den Zustand eines Objekts betrachten, in BlueJ dient dazu die Funktion **Inspect** im Kontextmenü

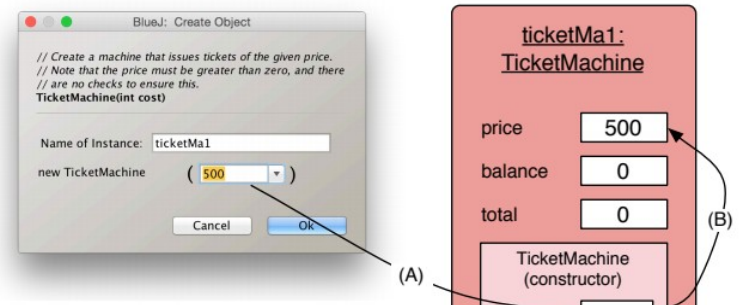
```
public class TicketAutomat
{
    private int preis;
    private int kontostand;
    private int gesamt;

    [...]
}
```



Konstruktor

```
public TicketAutomat(int ticketPreis)
{
    price = ticketPreis;
    balance = 0;
    total = 0;
}
```

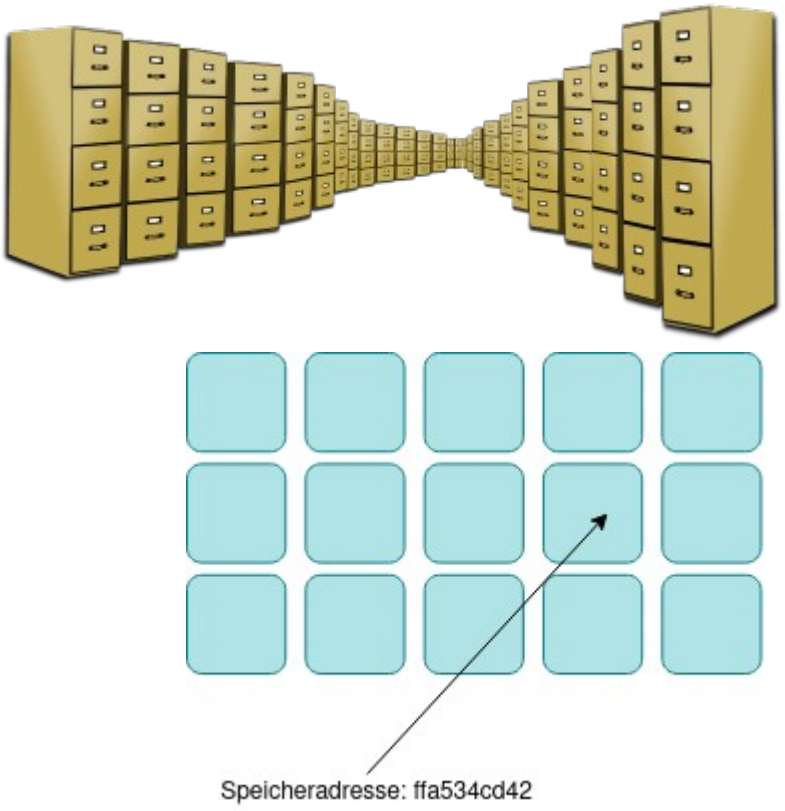
- Name der **Klasse** und des **Konstruktors** sind gleich
- Werden beim Instanziieren/Erzugen eines neuen Objekts ausgeführt, **initialisieren** das neue Objekt
- Haben eine enge **Verbindung zu den Instanzvariablen**: Meist werden diese im Konstruktor sinnvoll vorbelegt.
- Aus diesem Grund haben Konstruktoren oft **Parameter**, so dass man beim Erzeugen von Objekten entsprechende Objekteigenschaften beeinflussen kann.



Wertzuweisungen und Variablen

Um Werte in Feldern und anderen Variablen zu speichern, verwendet man **Wertzuweisungen**

```
variablenname = wert;

kosten = 7;
summe = 4 + 7;
summe = 6-3;
```



Sinnvolle Namensgebung!

preis, anzahl, name, alter …

a1, t, k, yxz u.ä.

Methoden

- **Methoden** legen das Verhalten von Objekten fest
- Methoden bestehen aus einem Methodenkopf und dem Methodenkörper
- Zwei große Typen werden unterschieden:
  - **„Getter“**: Ermitteln Informatioen zum Zustand des Objekts und geben diese an den Aufrufenden zurück
  - **„Setter“**: Verändern den Zustand eines Objekts, in dem Sie z.B. den Wert eines Feldes verändern.
- Weitere Methoden können verschiedenste Aufgaben innerhalb einer Klasse übernehmen

„Kapselung“

```
So:

private feld1          + Getter & Setter

Nicht so:

public  feld2          + automat3.feld2 = 5 (Niemals!)
```

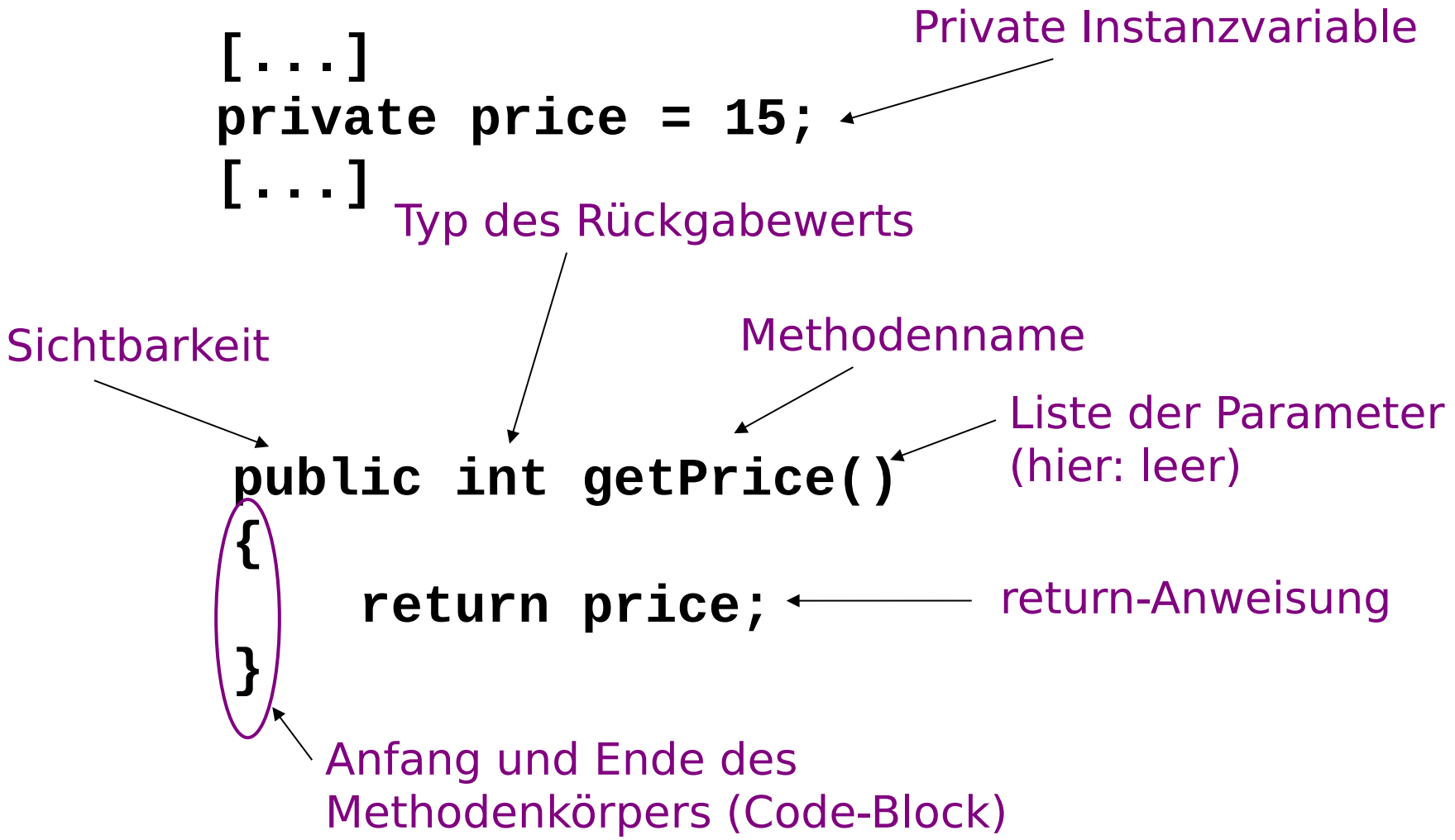
Methodenkopf

```
public int getPrice()

public void setName(String name)

private void calculateSum(int u, int u, int w)
```

Am Kopf kann man folgende Infos ablesen:

- Sichtbarkeit der Methode
- Gibt die Methode ein Ergebnis zurück?
- Den Methodennamen
- Ob und welche Parameter die Methode entgegennimmt

Ein einfacher „Getter“



- Ein „Getter“ hat stets einen nicht-leeren Rückgabetypen (nicht void)
- Ein Getter gibt einen Wert zurück
- Die Methode enthält stets ein return-Statement

Quick-Quiz

5 Fehler – welche?

```
public class CokeMachine
{
private price;

public CokeMachine()
{
   price = 300
}

public int getPrice
{
   return Price;
}
```

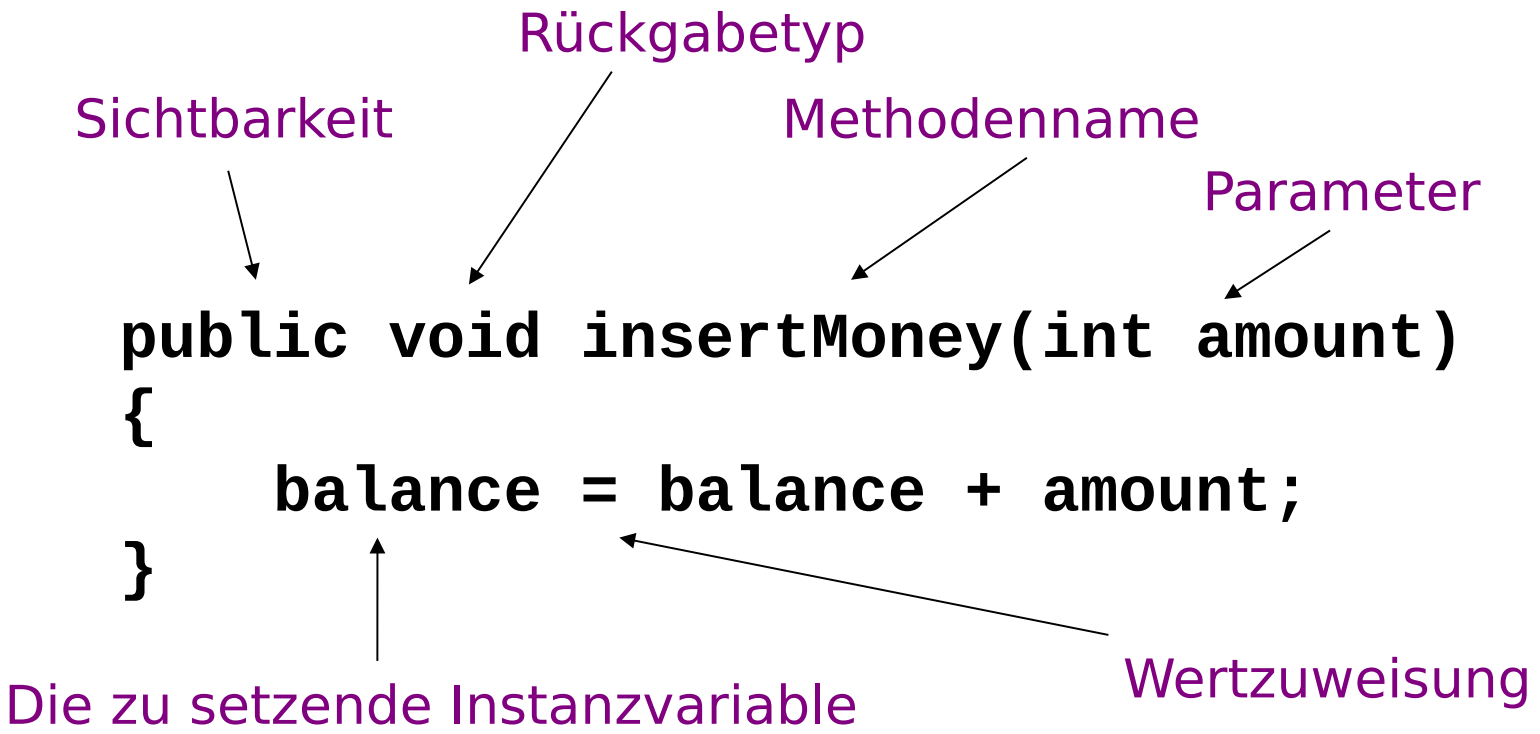
Quick-Quiz

```
public class CokeMachine
{
private price;

public CokeMachine()
{
   price = 300
}

public int getPrice
{
   return Price;
}
```

5 Fehler:

```
public class CokeMachine
{
private int price;

public CokeMachine()
{
   price = 300;
}

public int getPrice()
{
   return price;
}
}
```

„Setter“ Methoden



Ein einfacher „Setter“ hat folgende Eigenschaften

- Rückgabetyp **void**
- Methodenname stellt eine Beziehung zu der zu verändernden Instanzvariable her.
- Ein einzelner Parameter vom selben Typ wie die Instanzvariable
- Eine Wertzuweisung

„Setter“ Methoden

```
public void setDiscount(int amount)
{
    discount = amount;
}
```

Deklariere **`discount`**.

„Setter“ Methoden

```
public void setDiscount(int amount)
{
    discount = amount;
}
```

Deklariere **`discount`**.

```
private int discount;
```

Protektive „Setter“ Methoden

- Häufig werden in einer Setter-Methode vor der Wertzuweisung weitere Prüfungen vorgenommen, um das Attribut vor falschen/unsinnigen Werten zu „schützen“.
- Ein solcher Setter „schützt“ gewissermaßen die Instanzvariablen.
- Auch das gehört zum Prinzip der Kapselung

```
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println(
            "Use a positive amount: " +
            amount);
    }
}
```

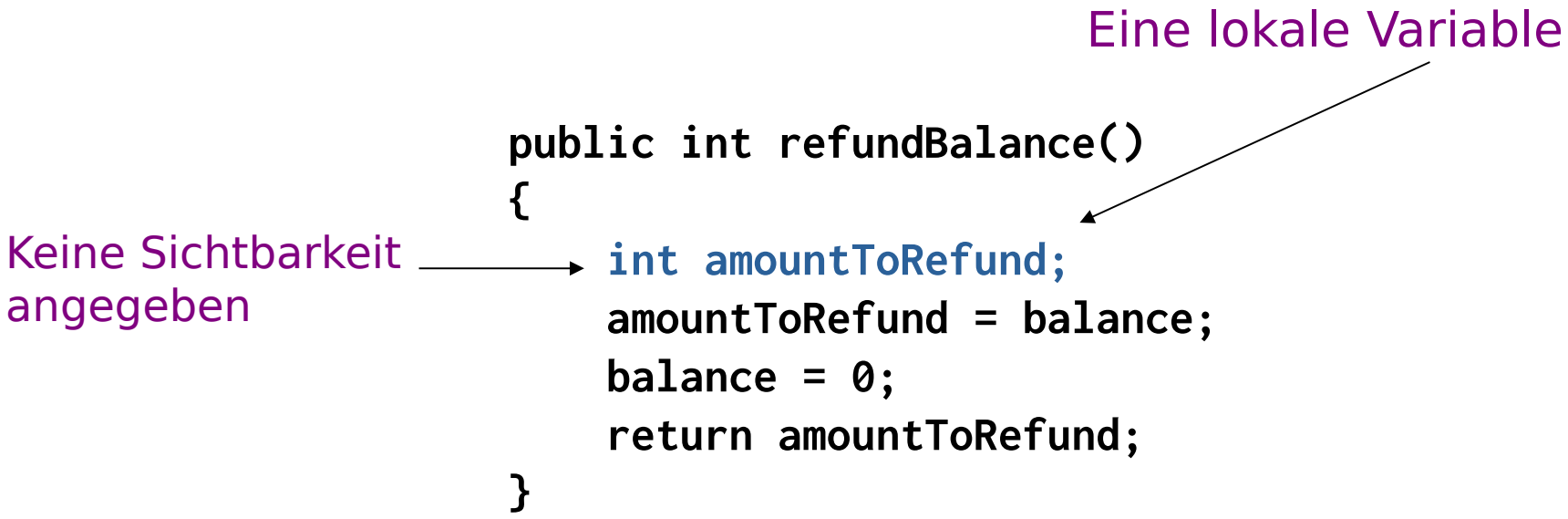
Scope & Lifetime (Sichtbarkeit und Lebenszeit)

- Jeder Codeblock (**{ }**)definiert einen neuen „Scope“ („Sichtbarkeitsbereich“)
- „Scopes“ können verschachtelt sein:

```
public void methode() {
        if (a>5) {
                for (int i; i<=6;i++) {

                }
        } else {

        }

}
```

- „Scope“ ist statisch (bei der Eingabe des Quelltexts)
- „Lebenszeit“ ist dynamisch (Zur Laufzeit des Programms)

Lokale Variablen



Scope und Lebenszeit der lokalen Variablen?

Scope & Lifetime II

Die Sichtbarkeit einer Instanzvariablen (Feld, Attribut) ist stets die gesamte Klasse.

Die Lebenszeit einer Instanzvariablen entspricht der Lebenszeit des Objekts, in welcher sie enthalten ist

Die Sichtbarkeit einer lokalen Variablen ist der Codeblock, in dem sie definiert wurde.

Die Lebenszeit einer lokalen Variablen entspricht der Ausführungszeit des Codeblocks, in welcher sie enthalten ist