Objektorientierte Programmierung

Objektvariablen

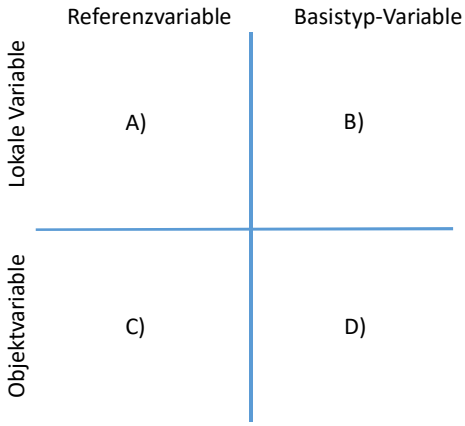
Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

Klassifikation von Variablen

- Unterscheidung nach Typ der Werte, die zugeordnet werden können:
  - **primitive Variablen** (haben Basistypen (= primitive Datentypen))
  - **Referenzvariablen** (haben Referenztypen)
- Unterscheidung nach Kontext, in dem sie eingesetzt werden:
  - **Objektvariablen**: Definition: Klasse, Verwendung: Objekt
  - **lokale Variablen**: Definition und Verwendung: Methode. (Auch Parameter sind lokale Variablen!)

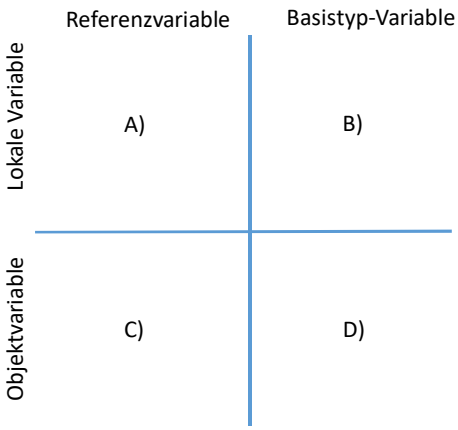
Übung – Einordnung der Variablen

```java
class Account {

    // Kontonummer
    private int accountNumber;

    // aktueller Kontostand
    private double balance = 0;

    // Einzahlungen in das Konto
    void deposit(double amount) {
        balance = balance + amount;
    }

    // Abfrage des aktuellen Kontostands
    // (Getter)
    double getBalance() {
        return balance;
    }

}
```



Übung – Einordnung der Variablen

```java
public class Main {

    public static void main(String[] args) {
        // Initialisierung eines Account-Objekts
        Account accountMax = new Account();

        // Aufruf der Methode getBalance
        // 1. Ergebnis wird an Variable übergeben
        double balanceMax = accountMax.getBalance();

        // oder alternativ Ergebnis wird direkt
        // verwendet.
        System.out.println(accountMax.getBalance());

        // Aufruf der Methode deposit.
        // Die Methode liefert kein Ergebnis zurück.
        accountMax.deposit(200);

    }
}
```



Die null-Referenz

- null steht für "kein Objekt".
- Ist ein Wert, der explizit jeder Referenzvariable zugewiesen werden:

```
Account account = null;
```

- null ist ein wohldefinierter Wert, er kann beispielsweise verglichen werden.

```
if(account == null) {
        System.out.println("kein Account vorhanden");
}
```

- Neu definierte Objektvariablen mit Referenztyp sind mit null vorinitialisiert.
- Neu definierte lokale Variablen sind nicht initialisiert, unabhängig vom Typ.

Gültigkeit von Variablen (Wichtig zur Compile-Zeit)

- Gültigkeit: Wo kann eine Variable verwendet werden?
- Lokale Variablen:
  - Gültig ab Definition bis Ende des umgebenden Blocks.
  - Auch gültig in untergeordneten Blöcken.
- Objektvariablen:
  - Gültig innerhalb der gesamten Klasse.
  - Insbesondere in allen Methoden der Klasse.

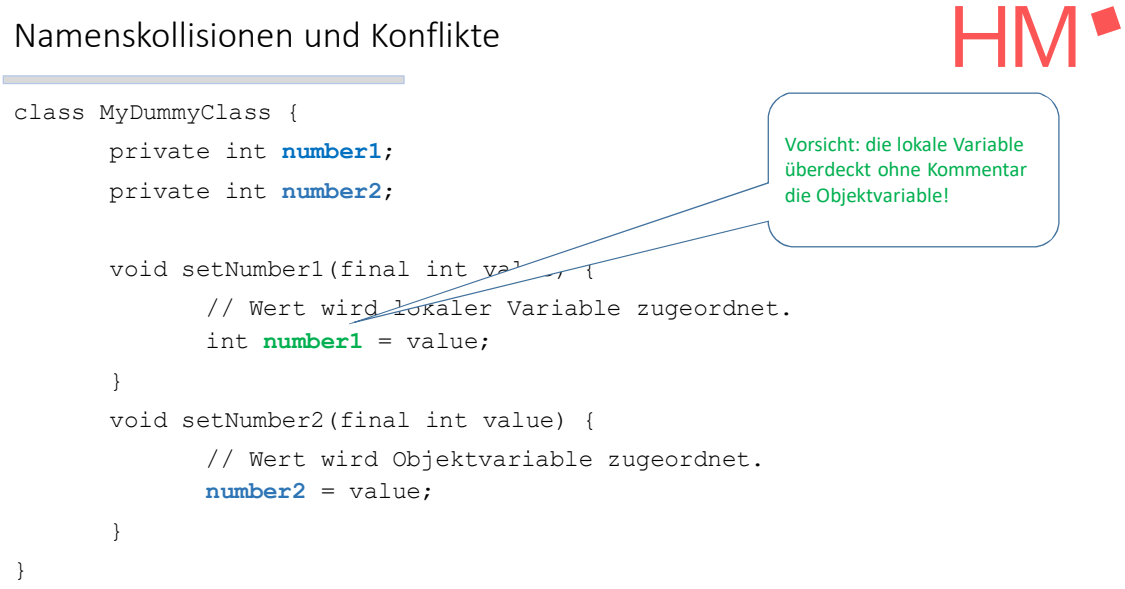
Lebensdauer von Variablen (Wichtig zur Laufzeit)

- Lebensdauer: Wo und wie lange „lebt“ eine Variable im Programm?
- Lokale Variable:
  - Ausschließlich in der Methode, in der sie definiert wird.
  - wird ‚erschaffen‘, wenn das Programm die Definition erreicht und zerstört mit Erreichen des Block-Endes.
- Objektvariable:
  - im Objekt, das mit **new** erzeugt wird.
  - wird ‚erschaffen‘ bei der Objektinitialisierung und zerstört bei der Zerstörung des umgebenden Objekts.

Namenskollisionen und Konflikte

- Zwischen lokalen Variablen:
  - Neudefinition einer Variable im gleichen Block oder in einem untergeordneten Block mit gleichem Namen führt zu einem Fehler. Das gilt auch für Parameter!
  - Neudefinition einer Variable in einem unabhängigen Block ist ohne Probleme möglich, sollte jedoch aus Gründen der Übersichtlichkeit eher vermieden werden.
- Zwischen Objektvariable und lokaler Variable
  - Neudefinition einer lokalen Variable mit dem Namen einer Objektvariable ergibt keinen Fehler!
  - **Aber** ....

Namenskollisionen und Konflikte

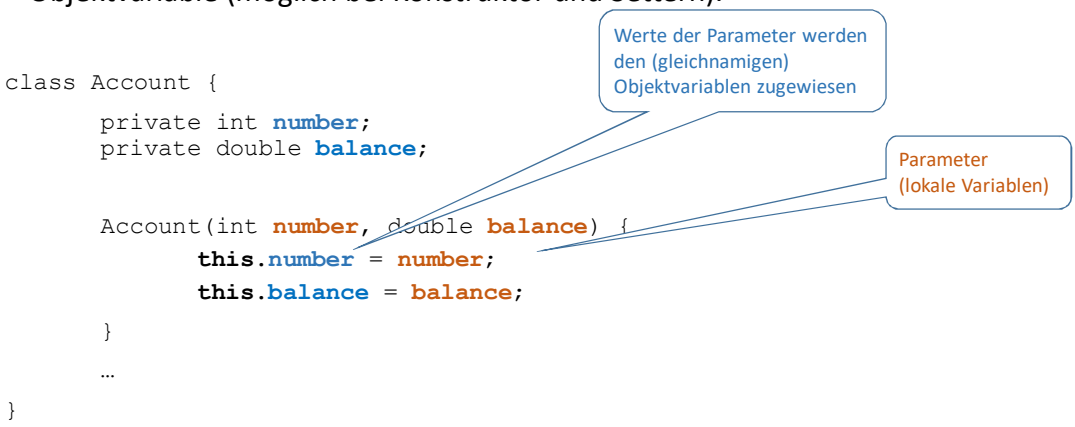


Selbstreferenz mit this

- Die Variable **this** enthält eine Referenz auf das aktuelle Objekt.
- **this** ist automatisch definiert und kann unmittelbar verwendet werden.
- Verwendung:
  - Auflösen von Namenskonflikten zwischen Objektvariablen und lokalen Variablen (typischerweise bei Parametern).
  - (Weitere Verwendung im Rahmen der Vererbung).

Selbstreferenz mit this

- Typisches Beispiel: Parametername wird gleich benannt wie zugehörige Objektvariable (möglich bei Konstruktor und Settern).




```
class Account {
        private int number;
        private double balance;

        Account(int number, double balance) {
                this.number = number;
                this.balance = balance;
        }
        …
}
```


Umgang mit Referenzvariablen

- Referenzvariablen sind Zeiger auf Objekte im Speicher. Jedes Objekt hat seinen unabhängigen Speicherbereich.
- Änderungen an einzelnen Objekten haben keinen Einfluss auf andere Objekte gleichen Typs.
- Beispiel:

```
Account account1 = new Account(1234, 200);
Account account2 = new Account(5678, 20);
account1.deposit(50);
account2.deposit(400);
account1.getBalance();      ->      250
account2.getBalance();      ->      420
```

Vergleich von primitiven Variablen mit ==

- Bisher kennen wir einen Vergleichsoperator ==
- Operanden sind numerische Werte. Der Operator prüft die numerischen Werte auf Gleichheit.
- Verwendung zur Formulierung von Bedingungen, z.B.

```
int i = Integer.parseInt(args[0]);
if (i == 0) {
    // tu etwas
}
```

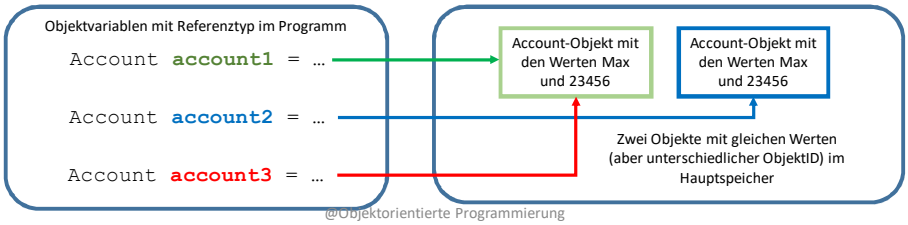
- Der Operator == kann auch zum Vergleich von Referenzvariablen verwendet werden. Aber .....

Vergleich von Referenzvariablen mit ==

- Referenzvariablen repräsentieren Objekte.
- Bei Objekten muss unterschieden werden zwischen
  - **identischen** Objekten und
  - **verschiedenen** Objekten mit **gleichen Werten**.
- Der Operator == vergleicht Objekte auf Identität, nicht (nur) auf Gleichheit der Werte.
- Der Vergleich von verschiedenen Objekten auf gleiche Werte muss explizit über den Vergleich der einzelnen Objektvariablen erfolgen (wir lernen hierzu noch die Methode equals() kennen).

Beispiel – Gleichheit versus Identität von Objekten

```
Account account1 = new Account("Max", 23456);
Account account2 = new Account("Max", 23456);
Account account3 = account1;

if(account1 == account2) {
        // Wird nie erreicht, da account1 und account2 zwar
        // gleiche Werte haben, aber nicht identisch sind.
} else if (account1 == account3) {
        // Wird erreicht, da die Variablen account1 und
        // account3 auf das gleiche Objekt im Speicher zeigen.
}
```



final bei lokalen und Objektvariablen

- final bei Objektvariablen:
  - Der Wert darf nur einmal zu Beginn gesetzt werden (Initialisierung mit Wert oder im Konstruktor). Danach kann der Wert nicht mehr geändert werden.
- final bei lokalen Variablen:
  - Der Wert darf nur einmal gesetzt und danach nicht mehr verändert werden.
- final bei Parametern:
  - Der Wert darf nur einmal gesetzt werden. Dies geschieht hier immer bei Aufruf der Methode. Innerhalb der Methode darf nur noch lesend auf den Parameter zugegriffen werden.

Zusammenfassung

HM

- Die null-Referenz ist ein gültiger Wert und kann jeder Referenzvariable zugewiesen werden. Es wird KEIN Objekt im Speicher angelegt.
- Objektvariablen sind innerhalb der Klasse gültig. Alle Methoden können auf die Objektvariablen der Klasse zugreifen und deren Werte lesen bzw. schreiben.
- Zwischen lokalen Variablen und Objektvariablen treten keine Namenskollisionen auf. Stattdessen können lokale Variablen Objektvariablen gleichen Namens überdecken.
- Die Selbstreferenz `this` ist ein Hilfsmittel um innerhalb von Methoden auf ‚dieses' Objekt zuzugreifen. Sie ist beispielsweise hilfreich zur Vermeidung von Namensüberdeckungen durch lokale Variablen.
- Der Vergleich von Referenzvariablen mit dem Vergleichsoperator == ist möglich. Es wird allerdings die Identität geprüft. Zum Vergleich auf Wertegleichheit sind andere Mechanismen erforderlich.