Kapitel 2.6

1. Was ist ein Header? Was ist ein Body?

   Header:

   ```
   public Account(String newOwner, int newPin)
   ```

   Body:

   Alles im Block darunter:

   ```
   {
      [...]
   }
   ```

2. Geben Sie die Methodensignaturen der TicketMachine (Code 2.1) (gemäss der Definition in SW 3) an.

   ```
   TicketMachine(int ticketCost)
   getPrice()
   getBalance()
   insertMoney(int amount)
   printTicket()
   ```

3. Wo können Anweisungen und Deklaration stehen?

   In der Klasse und der einzelnen Methoden darin.

4. Was ist ein Block?

   ```
   { […] }
   ```

5. Wie viele return Anweisungen finden Sie in Code 2.1?

   Zwei (bei getPrice() und getBalance())

6. zu bearbeitende Aufgaben: 2.21 bis 2.27

   2.21:

   getPrice() liefert den Wert von price zurück.

   getBalance() liefert den Wert von balance zurück.

   2.22:

   Wie viel wurde bereits eingeworfen?

   2.23:

   Nein, bei „return“ wird der Rückgabewert zurück gegeben, egal ob der Methodenname geändert wurde.

   2.24:

   ```
   public int getTotal()
   {
      return total;
   }
   ```

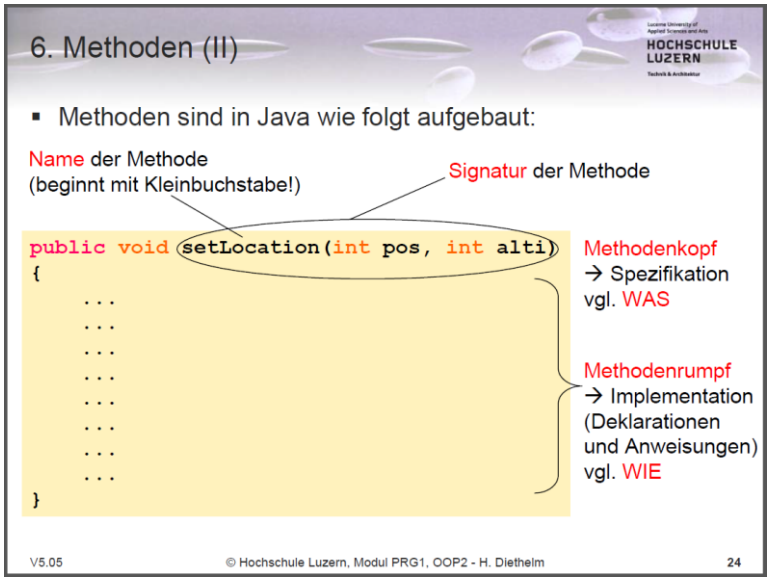
   2.25:

   missing return statement

2.26:

Abgesehen vom Namen der Methode sind beide Signaturen gleich.

getBalance() hätte jedoch einen Rückgabewert (int), printTicket() besitzt keinen Rückgabewert (void).



2.27:

Nein, sie haben keinen Rückgabewert.

Gemäss Beschreibung liefern sie ebenfalls keinen Rückgabewert und ändern nur die Instanz-Variablen.

Kapitel 2.7

7. Was bedeutet der return-Typ void?
   Kein Rückgabewert

8. Auf S. 33 ist in der Fussnote der compound assignment operator beschrieben.
   Es gilt: a += b entspricht a = a+b.
   Füllen Sie analog dazu folgende Tabelle aus:

compound assignment	**assignment**
a += b	*a = a + b*
a -= b	*a = a - b*
*a *= b*	*a = a * b*
a /= b	*a = a / b*

9. Im Code der TicketMachine (Code 2.1) gibt es 2 Stellen, an denen Sie den compound assignment operator verwenden können. Finden Sie diese beiden Stellen.

```
insertMoney(int amount)
{
   balance += amount;
}


public void printTicket()
{
    // Update the total collected with the balance.
    total += balance;
}
```

10. zu bearbeitende Aufgaben: 2.29 bis 2.32
    2.29:

    Anhand des Rückgabewertes (void). Ein Konstruktor hat keinen Rückgabewert!

2.30:

```
/**
 * Set the price of a ticket.
 * @param ticketCost
 */
public void setPrice(int ticketCost)
{
    price = ticketCost;
}
```

2.31:

```
/**
 * Increase score by the given number of points.
 * @param points
 */
public void increase(int points)
{
    score += points;
}
```

2.32:

```
/**
 * Reduce price by the given amount.
 * @param amount
 */
public void discount(int amount)
{
    price -= amount;
}
```

Kapitel 2.8

11. zu bearbeitende Aufgaben: 2.34 bis 2.38

2.34:

```
/**
 * Print ticket price
 */
public void showPrice()
{
    System.out.println("The price of a ticket is " + price + " cents.");
}
```

2.35:

```
The price of a ticket is 5 cents.
```

The price of a ticket is 15 cents.

Der Preis ist in einer Instanzvariable gespeichert, die können je nach Instanz unterschiedlich sein.

2.36:

Es wird der String „price“ ausgegeben und nicht der Wert welcher in der Variable „price“ gespeichert ist.

```
##################
# The BlueJ Line
# Ticket
# price cents.
##################
```

2.37:

Gleich wie bei 2.36

```
##################
# The BlueJ Line
# Ticket
# price cents.
##################
```

2.38:

Nein, beides mal wird es als String interpretiert und nicht als eine Variable.

Kapitel 2.11

12. Beschreiben Sie das conditional statement in pseudo-code auf S. 40 unten in Deutsch. Übersetzen Sie hierzu alle Beschreibungen ausser den 2 Worten „if“ und „else“ in Deutsch.

if (balance grösser gleich als price) {
// In der Variable total den Wert Total + preis speichern
total = total + preis
// In der Variable balance den Wert balance – price speichern
balance = balance – price;
}
else {
// Nachricht Ausgeben
System.out.println(„Bitte mindestens „ + (price – balance) + „ zusätzliche Rappen einwerfen“);
}

13. zu bearbeitende Aufgaben: 2.43 und 2.44

2.43:

```
Use a positive amount: -5
Use a positive amount: 0
```

Die Balance wird nur geändert wenn ein Wert > 0 eingegeben wird.

2.44:

Bei 0 wird nun keinen Error mehr ausgegeben. Die Balance ändert sich natürlich dadurch nicht. (x+ 0 = x)

Kapitel 2.12

14. zu bearbeitende Aufgaben: 2.46.

2.46:

```
    public void printTicket()
    {
        if(balance >= price) {
            // Simulate the printing of a ticket.
            System.out.println("##################");
            System.out.println("# The BlueJ Line");
            System.out.println("# Ticket");
            System.out.println("# " + price + " cents.");
            System.out.println("##################");
            System.out.println();

            // Update the total collected with the price.
            total = total + price;
            // Reduce the balance by the prince.
            balance = balance - price;
        }
        else {
            System.out.println("You must insert at least: " +
                               (price - balance) + " more cents.");

        }
    }
```

Es wird nur ein Ticket gedruckt wenn genügend Geld eingeworfen wurde. Andernfalls wird eine Meldung ausgegeben wie viel noch fehlt.

Der total sowie balance Wert wird nun richtig gerechnet und nicht einfach auf 0 gesetzt.

Kapitel 2.13

15. zu bearbeitende Aufgaben: 2.53 und 2.54

2.53:

Die Variable balance wird auf 0 gesetzt und danach als Returnwert verwendet, dieser ist somit immer 0.

2.54:

```
Compiler Error: unreachable statement
```

Bei einem Return wird die Methode sofort beendet, allfälig folgende Befehle würde nie ausgeführt.

16. Unter Pitfall auf S. 43 unten steht eine sehr wichtige Information. Übersetzen Sie den ersten Satz als Merksatz ins Deutsche.
Wenn eine lokale Varibael den gleichen Namen hat, wie eine Instanzvariable, dann kann nicht auf die lokale zugegriffen werden.

Kapitel 2.14

17. Füllen Sie die Tabelle als Zusammenfassung aus.

	Field / Attribut	**formaler Parameter**	**lokale Variable**
kann Werte speichern? (Ja/Nein)	Ja	Nein	Ja
Wo wird er/sie/es definiert?	Class	Signatur	Methode
Wie lange existiert er/sie/es? Wie ist die Lebensdauer?	Solange es die Klasse gibt.	Während des Aufrufes der Methode	Während des Aufrufes der Methode
Von wo kann auf ihn/sie/es zugegriffen werden? Wie ist die Sichtbarkeit?	Je nach AccessModifier, überall	Innerhalb der Methode	Innerhalb der Methode