Kapitel 4.1 bis 4.3

1. zu bearbeitende Aufgaben: 4.1

   4.1:

   done

2. Was verstehen Sie unter einem "Java-Package"?

   Erweiterungen verschiedener Klassen welche in Java benutzt werden können.

3. Sie möchten in Ihrem Source-Code die Bibliotheksklasse `ArrayList` verwenden. Mit welcher Anweisung können Sie die Bibliotheksklasse `ArrayList` verfügbar machen?

   ```
   import java.util.ArrayList;
   ```

4. Machen Sie sich mit Hilfe der API Dokumentation etwas schlau über die Klasse `ArrayList`. Welche Methoden erachten Sie als besonders wichtig/interessant?

   http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html

   add(); clear(); get(); indexOf(); isEmpty(); remove(); set(); size();

Kapitel 4.4 bis 4.7

5. zu bearbeitende Aufgaben: 4.2 bis 4.9

   4.2:

   ```
   ArrayList<Book> library;
   library = new ArrayList<Book>();
   ```

   4.3:

   size gibt die Anzahl Objekte zurück: 10.

   4.4:

   ```
   items.get(4);
   ```

   4.5:

   Der Index des letzten Objektes ist 14.

   4.6:

   ```
   notes.add(meeting);
   ```

   4.7:

   ```
   dates.remove(2)
   ```

   4.8:

   Index 5

   4.9:

   ```
   /**
    * Remove a note from the notebook.
    * @param noteNumber The number of the note to be removed.
    */
   public void removeNote(int noteNumber)
   {
       if ((noteNumber >= 0) && (noteNumber < notes.size())) {
           //remove this note
           notes.remove(noteNumber);
       }
   }
   ```

6. Erläutern Sie die nachfolgende Deklaration:
`private ArrayList<Balloon> list = new ArrayList<Balloon>();`
Es ist eine private ArrayList für Objekt-Typ Ballon, der Variablenname ist list.

7. Inwiefern haben Abstraktion und `ArrayList` miteinander zu tun?
Unter Abstraktion versteht man das „verstecken, vereinfachen" von Code, resp. deren Verwendung.
Beim ArrayList wird sehr viel Code „versteckt", resp. die Verwendung der ArrayList durch die Methoden vereinfacht.

8. Inwiefern unterscheiden sich die Methoden `remove()` und `get()` von `ArrayList`?
remove() entfernt das Item aus der Liste, get() liefert den Inhalt des Items zurück.

Kapitel 4.8. und 4.9

9. zu bearbeitende Aufgaben: 4.12 bis 4.18, 4.21, 4.25

4.12:

```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    for(String note : notes) {
        System.out.println(note);
    }
}
```

4.13:

```
listNotes();
> erster Eintrag
> 222222
> bronze
```

4.14:

done

4.15:

done

4.16:

```
/**
 * Remove a note from the notebook.
 * @param noteNumber The number of the note to be removed.
 */
public void removeNote(int noteNumber)
{
    if ((noteNumber >= 0) && (noteNumber < notes.size())) {
        //remove this note
        notes.remove(noteNumber);
    }
    else {
        System.out.println("This is not a valid note number!");
    }
}
```

```
/**
 * Show a note.
 * @param noteNumber The number of the note to be shown.
 */
public void showNote(int noteNumber)
{
    if(noteNumber < 0) {
        // This is not a valid note number
        System.out.println("This is not a valid note number!");
    }
    else if(noteNumber < numberOfNotes()) {
        // This is a valid note number, so we can print it.
        System.out.println(notes.get(noteNumber));
    }
    else {
        // This is not a valid note number
        System.out.println("This is not a valid note number!");
    }
}
```

4.17:

```
/**
 * print out all multiples of 5 between 10 and 95
 */
public void multiplesOfFive()
{
    int i = 10;
    while (i <= 95) {
        System.out.println(i);
        i += 5;
    }
}
```

4.18:

```
/**
 * sum all numbers between param1 and param2
 * @param a Number1
 * @param b Number2
 * @return sum of all numbers between a and b
 */
public long sum(int a, int b)
{
    int c = 0;
    long res = 0;
    if (a > b) {
        // swap a and b
        c = b;
        b = a;
        a = c;
    }
    while (a < b) {
        res += a;
        a++;
    }
    return res;
}
```

4.21:

```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int i = 0;
    for(String note : notes) {
        System.out.println(i + ": " + note);
        i++;
    }
}
```

4.25:

```
import java.util.ArrayList;

/**
 * Store details of club memberships.
 *
 * @author  Felix Rohrer
 * @version 2011-10-22-v0
 */
public class Club
{
    // Define any necessary fields here ...
    ArrayList<Membership> members;

    /**
     * Constructor for objects of class Club
     */
    public Club()
    {
        // Initialise any fields here ...
        members = new ArrayList<Membership>();
    }

    /**
     * Add a new member to the club's list of members.
     * @param member The member object to be added.
     */
    public void join(Membership member)
    {
        members.add(member);
    }

    /**
     * @return The number of members (Membership objects) in
     *         the club.
     */
    public int numberOfMembers()
    {
        return members.size();
    }
}
```

10. Erläutern Sie möglichst ausführlich den Source-Code auf Seite 97 unten.
Die Public Methode listNotes() liefert keinen Rückgabewert. Mithilfe der foreach-Schleife wird jede note vom Array notes auf die Konsole geschrieben.

11. Kann es passieren, dass der Schleifen-Rumpf einer `while`-Schleife gar nie zur Ausführung gelangt?
Ja, z.B. while(false) { […] }

12. Geben Sie für `no++` zwei alternative Anweisungen an.
```
no = no + 1;
no += 1;
```

13. Eine `ArrayList` lässt sich mit Hilfe einer foreach-Schleife traversieren. Kennen Sie noch weitere Möglichkeiten?
Mit dem Iterator, oder mittels for / while / do-while Schleifen.

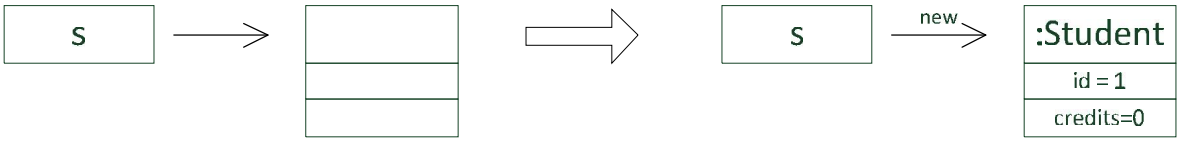
14. Ist `hasNext()` eine Methode von `ArrayList` oder `Iterator`? Wie ist der Rückgabewert von `hasNext()` zu interpretieren?
hasNext() ist eine Methode vom Iterator.
hasNext() Liefert ein Boolean zurück ob es ein weiteres Objekt in der Liste gibt welches noch nicht abgerufen wurde.

Kapitel 4.10

15. Eine Variable, deklariert für einen Klassentyp (Eine solche Variable wird auch Referenzvariable genannt.), kann den Spezialwert `null` beinhalten. Veranschaulichen diesen Fall mit einer Zeichnung. Wie sieht es aus, wenn die Variable ein Objekt speichert?

Bei der Deklaration wird erst der Speicherbereich für das Objekt reserviert. Die Referenzadresse zeigt erst auf den Speicherplatz wo später das Objekt gespeichert wird, zu diesem Zeitpunkt besitzt sie noch den Wert null. Erst wenn das Objekt instanziiert wird, wird im Speicher das Objekt erzeugt. In der Variable steht noch immer nur die Referenzadresse auf das Objekt.

```
private Student s;
s = new Student();
```



Kapitel 4.12

16. zu bearbeitende Aufgaben: 4.41 bis 4.45

4.41:

10: 227

14: 227

18: 237

4.42:

```
private Person[] people;
```

4.43:

```
private boolean[] vacant;
```

4.44:

2x hourCounts	*Variable, Array Init*
1x hourCounts.length	*Anzahl Array Elemente abfragen*
2x hourCounts[hour]	*Counter der selektierten Stunde (hour) erhöhen, resp. abfragen*

```
/**
 * Read web server data and analyse
 * hourly access patterns.
 *
 * @author David J. Barnes and Michael Kolling.
 * @version 2008.03.30
 */
public class LogAnalyzer
{
    // Where to calculate the hourly access counts.
    private int[] hourCounts;
    // Use a LogfileReader to access the data.
    private LogfileReader reader;

    /**
     * Create an object to analyze hourly web accesses.
     */
    public LogAnalyzer()
    {
        // Create the array object to hold the hourly
        // access counts.
        hourCounts = new int[24];
        // Create the reader to obtain the data.
        reader = new LogfileReader();
    }

    /**
     * Analyze the hourly access data from the log file.
     */
    public void analyzeHourlyData()
```

```
    {
        while(reader.hasMoreEntries()) {
            LogEntry entry = reader.nextEntry();
            int hour = entry.getHour();
            hourCounts[hour]++;
        }
    }

    /**
     * Print the hourly counts.
     * These should have been set with a prior
     * call to analyzeHourlyData.
     */
    public void printHourlyCounts()
    {
        System.out.println("Hr: Count");
        for(int hour = 0; hour < hourCounts.length; hour++) {
            System.out.println(hour + ": " + hourCounts[hour]);
        }
    }

    /**
     * Print the lines of data read by the LogfileReader
     */
    public void printData()
    {
        reader.printData();
    }
}
```

4.45:

Falsch:	Richtig:
[]int counts;	*int[] counts;*
boolean[5000] occupied;	*boolean[] occupied;* *occupied = new boolean[5000];* *oder:* *boolean[] occupied = new boolean[5000];*

17. Was spricht für und was spricht gegen Arrays?

Pro:

- *Kann Werte eines bestimmten elementaren Datentyps oder Objekte (Referenzen) eines bestimmten Klassentyps aufnehmen.*
- *Auf die Werte bzw. Objekte kann via Index direkt zugegriffen werden. (0 .. length-1)*
- *Ein Array selbst wird wie ein Objekt gehandhabt.*

Kontra:

- *Mit dem Erzeugen eines Arrays wird seine Länge unveränderlich festgelegt.*

18. Wie lässt sich die Länge eines Arrays ermitteln?

```
anzElemente = myArray.length;
```