Kapitel 5.6

1. zu bearbeitende Aufgaben: 5.23 bis 5.30

5.23:

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

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

Eine HashMap dient zum Speichern von Key-Value (Schlüssel-Wert) Paaren. Das heisst man gibt einen Schlüssel an und kann zu diesem Schlüssel einen Wert abspeichern und zu genau diesem Schlüssel auch den Wert wieder aus der HashMap herausholen. Ein Schlüssel kann ein beliebiges Objekt sein und ist meist eine Object Id, Speicheradresse oder ein String. Wichtig ist jedoch, dass ein Schlüssel eindeutig ist, sodass niemals ein Schlüssel für zwei Werte existiert.

5.24:

Ja

Return	***Method***
Set<Map.Entry<K,V>>	*entrySet()*
V	*get(Object key)*
Set<K>	*keySet()*
V	*put(K key, V value)*
void	*putAll(Map<? extends K,? extends V> m)*
V	*remove(Object key)*
Collection<V>	*values()*

5.25:

```
import java.util.HashMap;

/**
 * HashMap Test
 *
 * @author Felix Rohrer
 * @version 1.0
 */
public class MapTester
{
    private HashMap<String, String> phoneBook;

    /**
     * Constructor for objects of class MapTester
     */
    public MapTester()
    {
        phoneBook = new HashMap<String, String>();
        phoneBook.put("Felix Rohrer (Home)",   "+41 41 123 45 67");
        phoneBook.put("Felix Rohrer (Mobile)", "+41 79 123 45 67");
    }

    /**
     * Add a new Entry to the PhoneBook
     *
     * @param   Name, Number
     */
    public void enterNumber(String name, String number)
    {
        phoneBook.put(name, number);
    }

    /**
     * Get a Number
     *
     * @param   Name
     * @return  Number
     */
    public String lookupNumber(String name)
    {
        return phoneBook.get(name);
    }
}
```

5.26:

Der alte Eintrag wird überschrieben.

5.27:

Beide Werte werden in der HashMap gespeichert.

5.28:

```
phoneBook.containsKey("Test");
phoneBook.containsKey(name);
```

5.29:

Nichts, es wird null zurückgegeben.

5.30:

```
phoneBook.size()
```

2. Für welche Art von "Lookups" sind Maps ideal?

Immer wenn Schlüssel-Wert Paare gespeichert werden müssen. Durch den Key können diese sehr schnell abgefragt werden.

Kapitel 5.7

3. zu bearbeitende Aufgabe: 5.32

5.32:

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

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

Similarities:

- *können Objekte beinhalten*
- *add() Methode*
- *remove() Methode*
- *size() Methode*
- *iterator() Methode*

Differences:

- *beim HashSet muss das Objekt eindeutig sein, in einer ArrayList kann es mehrfach vorkommen*
- *die ArrayList ist sortiert, HashSet nicht*

Kapitel 5.8

4. zu bearbeitende Aufgaben: 5.33 und 5.35

5.33:

Es kann RegEx verwendet werden.

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

```
myString.split(":");
myString.split("\t"); // tab
myString.split("\n"); // new line
```

5.35:

In einem HashSet kann ein Element nur einmal vorkommen, jedoch ist die Reihenfolge im Vergleich zu einer ArrayList nicht sichergestellt.

Kapitel 5.10

5. zu bearbeitende Aufgaben 5.43 bis 5.45.

5.43:

Done.

5.44:

@author
@version
@param
@return
Diese „key-symbols“ werden in der Dokumentation speziell dargestellt.

5.45:

http://download.oracle.com/javase/6/docs/technotes/tools/windows/javadoc.html#javadoctags

6. Kommentieren Sie eines Ihrer eigenen Projekte (z.B. Balloon von OOP2) mit Hilfe von javadoc.

Class Balloon

java.lang.Object
└ Balloon

public class Balloon
extends Object

Write a description of class Balloon here.

Version:
06.10.2011
Author:
Felix Rohrer

Constructor Summary
Balloon() Constructor 2 for objects of class Balloon
Balloon(String baloonColor) Constructor 3 for objects of class Balloon
Balloon(String baloonColor, int newNumber) Constructor for objects of class Balloon

Method Summary	
void	blowOff() Luft Ablassen
void	blowUp() Ballon aufblasen
void	explode() Ballon zerstören
String	getColor() Farbe abfragen
int	getNummer() Lizenz-Nummer abfragen
int	getPosition() Position abfragen
int	getVolume() Volumen näherungsweise ausrechnen und zurück geben
void	setAltitude()
void	setAltitude(int newAlti) Höhe setzen
void	setLocation(int pos, int alti) Position festlegen

Method Detail

blowOff

public void blowOff()

Luft Ablassen

blowUp

public void blowUp()

Ballon aufblasen

Constructor Detail

Balloon

public Balloon()

Constructor 2 for objects of class Balloon

Balloon

public Balloon(String baloonColor)

Constructor 3 for objects of class Balloon

Parameters:
baloonColor - Farbe des Balloon

Balloon

public Balloon(String baloonColor,
int newNumber)

Constructor for objects of class Balloon

Parameters:
baloonColor - Farbe des Balloon newNumber Nummer vom Balloon

Kapitel 5.13

7. zu bearbeitende Aufgabe 5.58

5.58:

```
public static final double TOLERANCE = 0.001;
private static final int PASSMARK = 40;
public static final char HELPCOMMAND = 'h';
```

Algorithmen

8. Man rufe untenstehende Methode mit einem aktuellen Parameter auf. Während der Ausführung betrage die maximale Höhe des Call-Stacks 5. Wie oft wurde demzufolge die Methode coolMethod() insgesamt aufgerufen?

```
public int coolMethod(int n)
{
  if(n >= 2) {
    return (coolMethod(n-1) + coolMethod(n-2));
  }
  else {
    return n;
  }
}
```

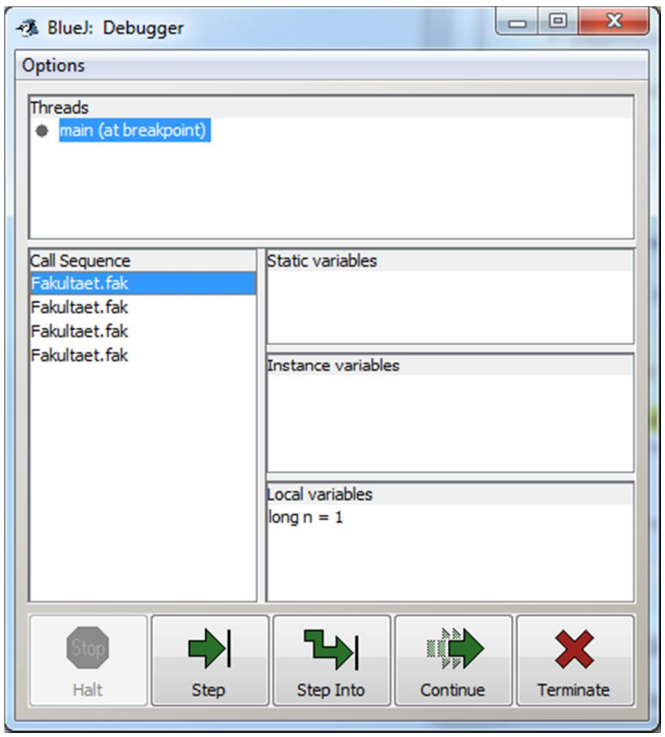
Fünfmal

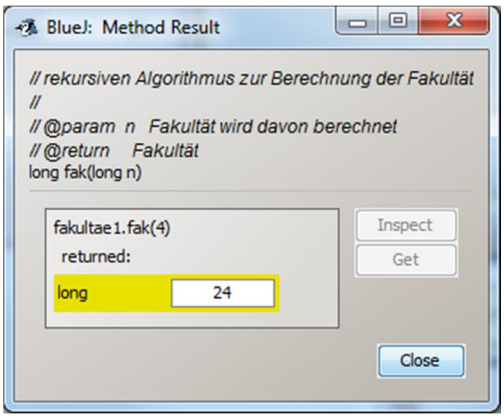
9. Implementieren Sie den rekursiven Algorithmus zur Berechnung der Fakultät , d.h. eine Methode public long fak(long n), vgl. Folie 9. Beobachten Sie die Berechnung von fak(4) mit dem Debugger.

```
/**
 * Implementieren Sie den rekursiven Algorithmus zur Berechnung der Fakultät
 *
 * @author Felix Rohrer
 * @version 1.0
 */
public class Fakultaet
{

    /**
     * Constructor for objects of class Fakultaet
     */
    public Fakultaet()
    {
        // nothing
    }

    /**
     * rekursiven Algorithmus zur Berechnung der Fakultät
     *
     * @param  n   Wert, für den die Fakultät berechnet wird (n>=0)
     * @return     n!
     */
    public long fak(long n)
    {
        if ((n == 0) || (n == 1)) {
            return 1; // Rekursionsbasis
        }
        else {
            return (n * fak(n-1)); // Rekursionsvorschrift
        }
    }
}
```





10. Implementieren Sie in Java rekursive Methoden, um ein Array vorwärts und rückwärts ausgeben zu können (Folien 30 und 31).

```
/**
 * Implementieren Sie in Java rekursive Methoden, um ein Array vorwärts und rückwärts ausgeben zu können
 * @author Felix Rohrer
 * @version 1.0
 */
public class RekArray
{
    private int[] myArray = {1,2,3,4,5,10,20,30,40,50,100};

    /**
     * Constructor for objects of class RekArray
     */
    public RekArray()
    {
        // nothing
    }

    /**
     * PrintOut Array forward
     * @param  Array  Array to print
     * @param  Index  Current Index
     */
    public void printForward(int[] a, int index)
    {
        if (index <= (a.length - 1)) {
            System.out.println("Index: " + index + ", Value: " + a[index]);
            printForward(a, index + 1);
        }
    }

    /**
     * PrintOut Array backward
     * @param  Array  Array to print
     * @param  Index  Current Index
     */
    public void printBackward(int[] a, int index)
    {
        if (index >= 0) {
            System.out.println("Index: " + index + ", Value: " + a[index]);
            printBackward(a, index - 1);
        }
    }

    /**
     * DoIt Forward
     */
    public void doItForward()
    {
        System.out.println("doItForward()...");
        printForward(myArray, 0);
        System.out.println("--------------------------------");
    }

    /**
     * DoIt Backward
     */
    public void doItBackward()
    {
        System.out.println("doItBackward()...");
        printBackward(myArray, myArray.length - 1);
        System.out.println("--------------------------------");
    }
}
```

```
doItForward()...
Index: 0, Value: 1
Index: 1, Value: 2
Index: 2, Value: 3
Index: 3, Value: 4
Index: 4, Value: 5
Index: 5, Value: 10
Index: 6, Value: 20
Index: 7, Value: 30
Index: 8, Value: 40
Index: 9, Value: 50
Index: 10, Value: 100
```

```
doItBackward()...
Index: 10, Value: 100
Index: 9, Value: 50
Index: 8, Value: 40
Index: 7, Value: 30
Index: 6, Value: 20
Index: 5, Value: 10
Index: 4, Value: 5
Index: 3, Value: 4
Index: 2, Value: 3
Index: 1, Value: 2
Index: 0, Value: 1
```

11. Implementieren Sie eine rekursive Methode, welche die Summe aller Werte eines Arrays von Integern berechnet.

```
/**
 * Calc Sum
 * @param   Array of Integer to sum up
 * @return  sum
 */
public long sum(int[] a, int index)
{
    if (index <= (a.length - 1)) {
        return a[index] + sum(a, index + 1);
    }
    else {
        return 0;
    }
}

/**
 * DoIt CalcSum
 */
public void doItCalcSum()
{
    long value;
    System.out.println("doItCalcSum()...");
    value = sum(myArray, 0);
    System.out.println("Sum:   " + value);
    System.out.println("--------------------------------");
}
```

```
doItCalcSum()...
Sum:   265
--------------------------------
```