Übungen zur Vorlesung “Grundlagen der Informatik” (GDI) von Manuel Gedack (Tutorium WS 2011/2012)

Aufgabe 1 (2 Punkte)
Thema: Schleifen
Schwierigkeitsstufe: Sehr leicht

Sie arbeiten in einem Call-Center eines Auktionshauses und nehmen telefonisch Gebote an. Da die Auktionen nicht live sind, ist das aktuelle Höchstgebot unbekannt. Da Sie sich aber merken müssen, was das höchste Gebot Ihrer Leitung war schreiben Sie sich ein Java Programm mit einer Klasse ***SucheMaximum***, die die Methode ***public static void verarbeiteEingabe(int eingabe)*** enthält. Diese Methode soll die Benutzereingabe überprüfen und sich ggf. das Maximum merken.
Die Methode ***public static int holeMaximum()*** soll Ihnen am Ende das eingegebene Maximum liefern.
Schreiben Sie außerdem eine passende ***main*** Methode zum Ausführen, die es Ihnen erlaubt ganze Zahlen einzugeben, die Eingabe mit einer negativen Zahl zu beenden und dann das Maximum der eingegebenen Zahlen mit Hilfe der zu entwickelnden Methoden ausgibt.

Beispiel Ausgabe auf der Konsole:

```
Bitte geben Sie die nächste Zahl ein (oder eine negative Zahl zum Beenden): 20
Bitte geben Sie die nächste Zahl ein (oder eine negative Zahl zum Beenden): 10000
Bitte geben Sie die nächste Zahl ein (oder eine negative Zahl zum Beenden): 3000
Bitte geben Sie die nächste Zahl ein (oder eine negative Zahl zum Beenden): -8
Das Maximum der eingegebenen Zahlen war: 10000
```

Aufgabe 2 (4 Punkte)
Thema: Schleifen, Modulo
Schwierigkeitsstufe: Leicht

Der Programmierer Cody Coder muss für ein Projekt mit der NASA den Umfang von Schwarzen Löchern im All exakt berechnen. Dafür benötigt er die Kreiszahl Pi. Aus der Schulzeit ist ihm bekannt, dass man mit Hilfe der folgenden Formel die Kreiszahl Pi näherungsweise berechnen kann:

Pi / 2 = (2/1) * (2/3) * (4/3) * (4/5) * (6/5) * (6/7) * (8/7) * (8/9) * (10/9) * (10/11) * (12/11) ...

Da Cody sich sehr unsicher ist, ob das so korrekt ist, werden Sie gebeten ihm zu helfen. Dazu schreiben Sie eine Java Klasse ***PiBerechnung***, die eine Methode ***public static double berechnePi (int n)*** enthält. Mit Hilfe der oben genannten Formel sollen Sie Pi bis zum n-ten Faktor berechnen und zurückzugeben.
Anschließend schreiben Sie eine Methode ***public static void zeigePi(int n)***, die Pi mithilfe der ***berechnePi*** Methode berechnet und jeweils das Ergebnis, mit der Java ***Math.PI*** Konstante verglichen, ausgibt. Pi soll dabei bis zum n-ten Faktor berechnet werden. Der Faktor bis zu dem Pi zu berechnen ist, ist gleich der Anzahl an Durchläufen, welche vom Benutzer eingegeben werden können.

Beispiel Ausgabe auf der Konsole:

```
Anzahl Durchläufe für Pi Bestimmung:
```

100
Pi berechnet: 2.0
Math.PI: 3.141592653589793
Pi berechnet: 4.0
Math.PI: 3.141592653589793
...
...
...
Pi berechnet: 3.125766292325387
Math.PI: 3.141592653589793
Pi berechnet: 3.157339689217563
Math.PI: 3.141592653589793

Aufgabe 3 (6 Punkte)

Thema: Files, Schleifen, boolean
Schwierigkeitsstufe: Mittel

Der Informatik Student Harry Hacker muss bis zum nächsten GDI Testat die Anzahl der *Lines of Code* seiner programmierten Java Klassen vorlegen. Zu den *Lines of Code* zählen in diesem Fall alle nicht leeren Zeilen der Java Quellcode Datei, die keine Kommentare enthalten. Man kann davon ausgehen, dass in einer Zeile **entweder** Java Code **oder** Kommentar enthalten ist.
Da Harry Hacker schlecht im Kopfrechnen ist, sollen Sie ihm helfen, indem Sie eine Java Klasse mit dem Namen ***LinesOfCode*** schreiben, welche eine Methode ***public static int countLines(String pfadZurDatei)*** enthält, die eine Java Quellcode Datei einließt, die *Lines of Code* zählt und anschließend ausgibt. Der Name der Datei soll vom Benutzer nach Start des Programms erfragt werden.

Beispiel Ausgabe auf der Konsole:

Bitte geben Sie den Pfad zur Java Quellcode Datei an:
C:\JavaFiles\Beispiel.java
Anzahl Lines of Code: 27

Aufgabe 4 (18 Punkte)

Thema: Schleifen, boolean, Arrays
Schwierigkeitsstufe: Schwer

Gegeben Sei folgendes einfache Labyrinth aus ASCII Raute („#“) Zeichnen:

```
##########
#        #
# ###### #
#     ## #
##### ## A
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Der Eingang ist durch ein ***E***, der Ausgang durch ein ***A*** und die Wand durch eine # gekennzeichnet. Der freie Platz im inneren des Labyrinths ist somit der begehbare Weg.

a)

Schreiben Sie eine Klasse ***EinfachesLabyrinth***, die eine Methode ***public static void zeichneLabyrinth(String[][] labyrinth)*** enthält, welche das Labyrinth auf die Konsole ausgibt.

b)

Erweitern Sie die Klasse ***EinfachesLabyrinth*** um eine Methode ***public static int[] setzeSpielfigurZufaellig(String[][] labyrinth)***, die eine durch ein ***S*** dargestellte Spielfigur beim Aufrufen zufällig auf dem Weg des Labyrinths positioniert.

c)

Erweitern Sie die Klasse ***EinfachesLabyrinth***, um folgende Methoden, welche es der Spielfigur ***S*** ermöglichen sich in alle Richtungen (Aufwärts, Abwärts, Links, Rechts) zu bewegen: ***public static boolean laufeAufwaerts(int[] position, String[][] labyrinth)***, ***public static boolean laufeAbwaerts(int[] position, String[][] labyrinth)***, ***public static boolean laufeLlinks(int[] position, String[][] labyrinth)***, ***public static boolean laufeRechts(int[] position, String[][] labyrinth)***.

d)

Erweitern Sie die Klasse ***EinfachesLabyrinth*** um eine Methode ***public static void findeAusgang()***, die mit Hilfe der Methoden aus ***a)*** – ***c)*** die zufällig positionierte Spielfigur ***S*** zum Ausgang ***A*** führt. Der bereits gelaufene Weg soll mit einem ***O*** gekennzeichnet werden. Nach jedem Schritt soll das Labyrinth ausgegeben werden.

Beispiel Ausgabe des Labyrinths auf der Konsole:

1.

```
##########
#        #
# ###### #
#     ## #
##### ## A
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Leeres Labyrinth.

2.

```
##########
#      S #
# ###### #
#     ## #
##### ## A
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Zufällig gesetzte Spielfigur.

3.

```
##########
#      OS#
# ###### #
#     ## #
##### ## A
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Ein Schritt nach rechts.

4.

```
##########
#      OO#
# ######S#
#     ## #
##### ## A
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Ein Schritt nach unten.

5.

```
##########
#      OO#
# ######O#
#     ##S#
##### ## A
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Ein weiterer Schritt nach unten.

6.

```
##########
#      OO#
# ######O#
#     ##O#
##### ##SA
#   # ####
# # #    #
# # #### #
# #      #
#E########
```

Mit diesem Schritt nach unten ist der Ausgang erreicht.