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 int verarbeiteEingabe(int eingabe, int maxZahl)*** enthält.
Diese Methode soll die Benutzereingabe überprüfen und sich ggf. das Maximum merken. 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 Methode 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:

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

Hinweis:
Ist n ungerade gilt = ((n+1)/n)), Beispiel: n=1 => ((1+1)/1) = (2/1)
Ist n gerade gilt = ((n)/(n+1)), Beispiel: n=2 => ((2)(2+1)) = (2/3)

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. Diese Methode soll mit Hilfe der oben genannten Formel Pi berechnen und zurückgeben. Pi soll dabei bis zum n-ten Faktor berechnet werden (Siehe Hinweis), welcher vom Benutzer eingegeben werden kann.
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 (Siehe Beispiel Ausgabe.).

Beispiel Ausgabe auf der Konsole:

Bis zu welchem Faktor (n) soll Pi bestimmt werden?:
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 ***LinesOfCodeCounter*** schreiben, welche eine Methode ***public int countLines(String pfadZurDatei)*** enthält, die eine Java Quellcode Datei einließt, die *Lines of Code* zählt und anschließend zurück gibt. Schreiben Sie außerdem eine Klasse ***CodeCounterHauptprogramm*** mit einer geeigneten ***Main*** Methode, welche den Namen der Datei vom Benutzer nach Start des Programms erfragt und am Ende die *Lines of Code* ausgibt.

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 das Feld ***private String[][] labyrinth*** enthält, welche das Labyrinth beinhaltet.
Außerdem soll es eine Methode ***public void zeichneLabyrinth()*** geben, welche das Labyrinth auf die Konsole ausgibt.
Die Methode ***public String[][] getLabyrinth()*** soll das Labyrinth zurückliefern können und die Methode ***public void setLabyrinth(String[][] labyrinth)*** soll es ermöglichen, Änderungen am Labyrinth vorzunehmen.

b)

Schreiben Sie eine Klasse ***Position***, welche die Felder **private int xPosition** und ***private int yPosition*** enthält. Diese werden später benötigt, um der Spielfigur eine Position innerhalb des Labyrinths zu geben. Die Klasse soll außerdem die Methoden ***public int getXPosition()*** und ***public int getYPosition()*** die die horizontale (X) Position bzw. die vertikale (Y) Position auf dem Spielfeld zurückgeben. Um die horizontale (X) Position bzw. die vertikale (Y) Position zu setzen muss die Klasse auch die Methoden ***public void setXPosition(int xPosition)*** und ***public void setYPosition(int yPosition)*** enthalten.

c)

Schreiben Sie eine Klasse ***Spielfigur***, die eine Methode ***public void setzeSpielfigurZufaellig()***, die eine durch ein ***S*** dargestellte Spielfigur beim Aufrufen zufällig auf dem Weg des Labyrinths positioniert. Für die Positionsverwaltung der Spielfigur soll die Klasse ***Position*** aus ***b)*** verwendet werden.

d)

Erweitern Sie die Klasse ***Spielfigur***, um folgende Methoden, welche es der Spielfigur ***S*** ermöglichen sich in alle Richtungen (sofern möglich) innerhalb des Labyrinths (Aufwärts, Abwärts, Links, Rechts) zu bewegen: ***public boolean laufeLinks()***,***public boolean laufeRechts()***,***public boolean laufeAufwaerts()***,***public boolean laufeAbwaerts()***.

e)

Erweitern Sie die Klasse ***Spielfigur***, um eine Methode ***private boolean checkObAusgangErreicht()***, welche überprüft ob der Ausgang erreicht wurde.

f)

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

g)

Entwerfen Sie sich außerdem eine geeignete Hauptklasse, welche die Klassen bzw. Methoden aus ***a)*** – ***f)*** nutzt um zu simulieren, wie ein zufällig gesetzter Spieler auf dem Labyrinth den Ausgang finden. Dabei soll jeder Schritt des Spielers nachvollziehbar sein (Siehe Beispiel). Bei Programmstart soll ein Labyrinth erzeugt werden. Bei der Erzeugung des Spielers soll es möglich sein, dem Spieler das zuvor erzeugte Labyrinth zu übergeben und anschließend den Spieler den Ausgang finden zu lassen. Erweitern sie ggf. die Klasse Spielfigur um einen geeigneten Konstruktor, der dafür sorgt, dass die Spielfigur eine Position hat, ein Labyrinth auf dem Sie laufen kann und dort zufällig gesetzt wird.

Hinweis:
Bei diesem einfachen Labyrinth ist es ausreichend die Umgebung der aktuellen Position zu überprüfen und zu testen in welche Richtung gelaufen werden kann und wo man bereits war. Ist man am Eingang angekommen darf zum ursprünglichen Startpunkt gesprungen werden und von dort aus in die andere Richtung weitergelaufen werden, bis der Ausgang erreicht ist.

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.