Grundlagen der Informatik 1
WS 2008/09

Prof. Mühlhäuser, Dr. Rößling, Melanie Hartmann, Daniel Schreiber
http://proffs.tk.informatik.tu-darmstadt.de/gdi1

Übung 12 - Lösungsvorschlag Version: 1.1 26.01.2009

---

1 Mini Quiz

☐ Die Reihenfolge der catch-Blöcke ist nicht von Bedeutung.

☐ Ein Compiler kann alle Arten von Fehlern (bis auf Laufzeitfehler) entdecken und signalisieren.

☒ Die beim Ausführen einer Methode auftretenden Ausnahmen können im catch Block behandelt werden.

☐ Fehler, die während der Laufzeit des Programms auftreten müssen behandelt werden.

☐ Ausnahmen müssen unmittelbar in der Methode behandelt werden, in der sie auftreten.

☐ Man kann leicht nachweisen, dass ein Programm fehlerfrei ist.

2 Verständnis

1. Welche Gemeinsamkeiten besitzen die Ausnahmeklassen Exception und Error? Was unterscheidet sie? Welche von beiden sollte man abfangen?

   Lösungsvorschlag:
   Beide Klassen sind direkte Erben der Klasse Throwable, außerdem treten beide zur Laufzeit des Programmes auf.

   - *Errors sind "schwerwiegende" Fehler, diese führen oft direkt zum Programmabsturz, so dass eine Behebung und planmäßige Programmfortsetzung* **nicht mehr möglich** *ist. Hierzu zählt z.B. eine abgerissene (aber erforderliche) Netzwerkverbindung.*
   - *Exceptions stehen hingegen für "leichtgewichtige" Fehler. Ausnahmen der Klasse Exception können innerhalb des Programms abgefangen und behoben werden.* **Eine Behandlung und planmäßige Programmfortsetzung ist somit möglich**.

2. Erklären Sie, was Syntax-, Laufzeit-, Intentions- und lexikalische Fehler sind. Welche dieser Fehler werden vom Compiler erkannt?

   Lösungsvorschlag:

   - **Syntaxfehler** *entstehen durch eine falsche Anordnung von Worten.*
   - **Laufzeitfehler** *treten, wie der Name schon sagt, zur Laufzeit des Programms auf und stören den Ablauf, etwa wenn kein freier Arbeitsspeicher mehr verfügbar ist oder auf nicht mit einem Objekt belegten Referenzen (null) eine Aktion ausgeführt werden soll.*

- **Intentionsfehler:** *Das Programm läuft, jedoch ist das Ergebnis falsch. Der Grund dafür ist in der Regel eine syntaktisch zwar fehlerfreie, aber nicht den Erwartungen entsprechende Implementierung (einfacher gesagt: Denkfehler in der Implementierung). Daher führt der Computer zwar fehlerfrei eine Berechnung aus, aber eben nicht mit der gewünschten Funktionalität.*
- **Lexikalische Fehler** *sind in der Regel falsche oder unbekannte Worte.*

Nur die Fehlerarten syntaktisch / lexikalisch können bereits vom Compiler erkannt werden.

3. Wie hängen die Begriffe Throwable, throws und throw zusammen? Wo sollte man was verwenden?

Lösungsvorschlag:

- **Throwable***: Ist die oberste Klasse in der Fehlerhierarchie, sowohl Error als auch Exception sind Subklassen von Throwable.*
- **throws***: Ist Bestandteil der Methodensignatur und gibt an, welche Ausnahmen (Exceptions) die Methode zum Aufrufer weitergibt.*
- **throw***: Ist der Befehl, um eine Ausnahme (Exception) auszulösen.*

Folgende Beispiele verdeutlichen die Verwendung, wo was genau verwendet wird.

- **Throwable** *wird normalerweise nicht direkt genutzt. Jede selbstdefinierte Exception muss von der Klasse java.lang.***Throwable** *oder einer ihrer Subklassen abgeleitet sein. In den meisten Fällen wird man als Superklasse entweder* `java.lang.Exception` *oder* `java.lang.RuntimeException` *verwenden.*

```
1 class MyException extends Exception{
2   MyException(Parameterliste){
3     //...
4   }
5 }
```

- **throws** *gehört in die Methodensignatur:*

```
1 String readFirstLineOfFile(String filename) throws
    FileNotFoundException, IOException {
2   RandomAccessFile f = new RandomAccessFile(filename, "r");
3   return f.readLine();
4 }
```

- **throw** *gehört in die Methode:*

```
1 int faculty(int arg) {
2   if (arg < 0) throw new IllegalArgumentException("Illegal␣
      argument␣<0");
3 }
```

4. Ist es sinnvoll, eigene Ausnahmeklassen zu definieren? Welche Vorteile ergeben sich hieraus?

Lösungsvorschlag:
Es ist sogar sehr sinnvoll, eigene Ausnahmeklassen zu definieren. Dadurch lassen sich Fehler leichter spezifizieren (verschiedene Fehlerkontexte) und, da sie nicht alle unter dem gleichen Namen laufen, auch besser behandeln.

3 Fehlersuche

1. Betrachten Sie folgenden Java-Code. Beheben Sie alle syntaktischen Fehler.

```
1  public static int[] reverseArray (int[] source) throw Exception {
2    int length = source.length();
3    int[] inverted;
4    int i=0;
5
6    try {
7      inverted = new int[length] ;
8
9      while (i < length){
10       inverted[i] = source[i];
11       i++;
12     }
13   }
14   catch (Exception) {
15     System.out.println("Caught Exception...");
16   }
17   catch (IndexOutOfBoundsException e) {
18     System.out.println("Caught Exception: " + e);
19   }
20   final {
21     inverted = new int[length()];
22     inverted = source ;
23   }
24   return inverted;
25 }
```

Lösungsvorschlag:

- *Zeile 1: throws anstatt throw.*
- *Zeile 2: length ist keine Methode sondern ein Attribut. Die Klammern sind daher falsch.*
- *Zeile 16: Fehlender Name für das Exception Objekt.*
- *Zeile 23: finally anstatt final.*
- *Zeile 24: length anstatt length()*

2. Was passiert generell beim Aufruf der Funktion reverseArray?

   Die Funktion wird von Java nicht kompiliert - auch **ohne** vorhandene syntaktische Fehler, weswegen? Wie können Sie die Funktion dennoch kompilieren, was müsste man dazu beheben? **Hinweis**: Betrachten Sie bitte nochmal die Mini Quiz Fragen...

   Lösungsvorschlag:
   Es wird eine möglicherweise auftretende Exception abgefangen, inverted mit source überschrieben und danach zurückgegeben.

   Der Java-Compiler wird wegen nicht erreichbarem Programmcodes einen Fehler melden. Das Abfangen einer IndexOutOfBoundsException im zweiten Catch-Block ist nicht möglich, da alle Ausnahmen bereits durch den allgemeineren Typ Exception im ersten Catch-Block abgefangen werden. Das Problem kann durch die Änderung der Anordnung der Exception gelöst werden.

3. Suchen und beheben Sie die vorhandenen Intentionsfehler (Fehlerquelle 'Mensch').

   Lösungsvorschlag:

   - *Zeile 10: Austauschen von source[i] gegen source [length-i-1] damit auch wirklich invertiert wird.*
   - *Initialisierung des Arrays zur Deklaration nehmen - damit können wir den finally-Block aufgeben und können trotzdem immer eine initialisierte Variable zurückgeben.*

4 Funktionen höherer Ordnung in Java

Erinnern Sie sich noch an die Funktion fold aus dem Scheme-Teil der Vorlesung? Zur Erinnerung, der Vertrag lautet:

```
;; fold : (X Y -> Y) Y -> ((list of X) -> Y)
```

Machen Sie sich nochmal kurz klar, welche Aufgabe die folgende Variante von fold erfüllt.

```
(define (fold f n)
 (lambda (x)
   (if (empty? x)
     n
     (f (first x) ((fold f n) (rest x ))))))
```

Wir wollen nun eine ähnliche Funktion in Java implementieren. Zunächst stellen wir fest, dass eine direkte Übersetzung nach Java nicht möglich ist, da Java keine Funktionen höherer Ordnung unterstützt. Mit Hilfe eines Interfaces kann jedoch ein solches Verhalten simuliert werden.

1. Deklarieren Sie ein Interface namens BinaryOp, das eine binäre Operation `apply` über ganzen Zahlen deklariert. Sowohl Argumente als auch der Rückgabewert sollen vom Typ int sein.

   Lösungsvorschlag:

   ```
   public interface BinaryOp {
     int apply(int a, int b);
   }
   ```

2. Schreiben Sie jetzt die Klassen Adder und Multiplier, die das Interface BinaryOp implementieren und in der Methode `apply` die Addition bzw. Multiplikation zur Verfügung stellen.

   Lösungsvorschlag:

   ```
   class Adder implements BinaryOp {
     public int apply(int a, int b){
       return a + b;
     }
   }

   class Multiplier implements BinaryOp {
     public int apply(int a, int b) {
       return a * b ;
     }
   }
   ```

3. Implementieren Sie nun eine Methode mit folgender Signatur:

```
int jfold(int[] a, BinaryOp op, int s)
```

Diese Methode soll analog zu fold die ihr übergebene Operation *op* schrittweise auf alle Elemente des Arrays *a* anwenden. Dabei entspricht *s* dem Argument *n* in fold.

Lösungsvorschlag:

```
int jfold(int[] a, BinaryOp op, int s) {
  int result = s;

  for (int i = 0; i < a.length; i++)
    result = op.apply(result, a[i]);

  return result;
}
```

4. Erklären Sie nun, worin sich jfold und fold unterscheiden.

Lösungsvorschlag:
Ein Unterschied zwischen jfold und fold besteht darin, dass BinaryOp und damit auch jfold nur binäre Operationen auf ganzen Zahlen unterstützt. fold hingegen akzeptiert binäre Operationen, die auf **beliebigen** *Datentypen arbeiten.*

Ein weiterer Unterschied besteht darin, dass fold eine Funktion zurückliefert, während jfold eine Zahl liefert. An dieser Stelle sei nochmal erwähnt, dass Java es nicht ohne Weiteres ermöglicht, Funktionen zurückzugeben!

5 Polymorphie

Betrachten Sie folgenden Java Code:

```
interface MyInterface {
  public int g();
}

class Base implements MyInterface {
  public int i = 1;
  public int g() { return f(); }
  protected int f() { return i; }
}

class Derived extends Base {
  public int i = 2;
  protected int f() { return -i; }
  protected int h() { return i + i; }
  protected int p() { return i * i; }
}

public class Client {
  public static void main (String args[]){
    Base base = new Base() ;
```

```
21      Derived derived = new Derived();
22      MyInterface restricted = derived;
23
24      int temp = base.i;      // a)
25      temp = base.g();        // b)
26      base = derived;
27      temp = base.i;          // c)
28      temp = base.g() ;       // d)
29      temp = restricted.g(); // e)
30      restricted = base;
31      temp = restricted.g(); // f)
32      System.out.println(temp);
33    }
34  }
```

1. Welchen statischen und dynamischen Typ hat die Variable base zum Zeitpunkt a) bzw. c)?

   Lösungsvorschlag:

   - *Zeitpunkt a: Der statische und dynamische Typ ist Base.*
   - *Zeitpunkt c: Statischer Typ ist Base, dynamischer Typ ist Derived.*

2. Welchen Wert hat die Variable **temp** zu den Zeitpunkten a), b), c), d) e) und f)?

   Lösungsvorschlag:

   a) 1

   b) 1, Ergebnis von Base.g()=base.f()=base.i)

   c) 1 , die Attribute werden statisch gebunden

   d) -2, da Aufruf von base.g() auf f() in Derived abbildet

   e) -2, siehe d)

   f) -2, siehe d)

3. Betrachten Sie nun die folgenden Aufrufe:

```
1  Base base = new Base();
2  Derived derived = new Derived();
3  int temp = base.g();
4  derived = base;
5  base = new Derived();
6  MyInterface t = new Derived();
7  t = base;
8  temp = t.f();
9  derived = t;
10 base = t;
11 t = derived;
12 temp = t.p();
13 temp = t.g();
14 temp = derived.p();
```

Welche Aufrufe sind erlaubt, welche werfen einen Fehler? Treten die Fehler zur Laufzeit oder bereits beim Kompilieren auf? Es gibt keine Folgefehler - kommentieren Sie verbotene Aufrufe aus und ignorieren Sie diese bei der weiteren Fehlersuche.

Lösungsvorschlag:

- *Zeile 4: derived kann nur Referenzen von Objekten der Klasse Derived oder einer ihrer Subklassen aufnehmen.*
- *Zeile 8: das Interface MyInterface kennt keine Funktion f.*
- *Zeile 9: gleiches Problem wie in Zeile 4.*
- *Zeile 10: der statische Typ von t ist MyInterface, der dynamische Typ ist Derived. Hier wäre eine explizite Cast-Anweisung nötig: base = (Base) t;*
- *Zeile 12: das Interface MyInterface kennt keine Methode p.*

Es handelt sich um Syntaxfehler, die bereits vom Java-Compiler erkannt werden.

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren.

Abgabe: Bis spätestens Fr, 06.02.09, 16:00 Uhr.

6 Geografische Welten (10 Punkte)

Viele geografische Anwendungen bedienen sich heutzutage verschiedener Tricks aus der Informatik, woraus sich auch ein eigener Zweig entwickelt hat, die Geoinformatik. Wir schauen uns in dieser Hausübung eine stark vereinfachte Praxisanwendung aus diesem Gebiet an, in der es darum geht, die Küsten der Niederlande näher unter die Lupe zu nehmen.
In den Niederlanden gibt es immer wieder Sturmfluten, die das Land wegen seiner geringen Höhe überschwemmen. So hat die letzte große Sturmflut von 1953 fast ganz Zeeland (im Süden des Landes) verwüstet. Um sich gegen diese Naturgewalt zu schützen, behelfen sich die Küstenbewohner mithilfe von Deichen entlang der Nordseeküste. In dieser Aufgabe wollen wir eine solche Sturmflut auf einem gegebenen Gebiet simulieren.

6.1 Grundstruktur (0.5P)

Erstellen Sie zunächst die Klasse `GeoSimulation`. Diese soll als interne Datenstruktur area ein `char[][]` verwenden; diese Datenstruktur repräsentiert eine Karte des Gebiets. Eine korrekte Karte ist ein rechteckiges Gebiet, wobei jeder Teil durch eines von drei Symbolen repräsentiert wird: '#' für ein Stück Deich, '0' für ein Stück Land und '~' für ein Wassergebiet. Implementieren Sie die folgenden Methoden der Klasse `GeoSimulation`:

- Konstruktor GeoSimulation(**char** [][] area)

- Getter und Setter Methoden für die interne area Datenstruktur
- Eine Methode toString(), die eine Stringrepräsentation der Area ausgibt. Eine solche Repräsentation kann z.B. folgendermaßen aussehen:

```
1 ~~~~~#00000
2 ~~~~#000000
3 ~~~#0000000
4 ~~~~#000000
```

6.2 Gültige Karten (2P)

Damit eine Karte gültig ist, müssen mindestens ein Deich, ein Stück Land und Wasser vorhanden sein. Schreiben Sie dazu die Methoden **boolean** hasLeveeElement() (*levee* ist die amerikanische Bezeichnung für einen Deich), **boolean** hasGroundElement() und **boolean** hasWaterElement(). Sobald eine neue Area angegeben wird (d.h. im Konstruktor oder durch setArea()) soll geprüft werden, ob die Karte gültig ist und ansonsten eine entsprechende Exception geworfen werden (NoLeveeElementException, NoGroundElementException bzw. NoWaterElementException). Diese Exceptions sollen von der Exception InvalidAreaException erben.

6.3 Deichbruch (3P)

Implementieren Sie die Methode **void** floodArea(**int** x, **int** y), die die Koordinaten eines möglichen Deichbruchs entgegennimmt und berechnet, welche Teile des Gebiets danach unter Wasser stehen. Dazu sollen alle Landelemente ('0') der aktuellen Karte, die von der Überschwemmung betroffen sind, durch Wasserelemente ('~') ersetzt werden. Dabei erfolgt die Überschwemmung nur horizontal und vertikal, ausschließlich diagonal angrenzende Elemente sind nicht betroffen. Die Koordinaten entsprechen dabei den Zählern des Arrays, d.h. mit den Koordinaten (x,y) ist das Element area[x][y] gemeint.
Hinweis: Sie können sich beim Lösen am Floodfill-Algorithmus (siehe `http://de.wikipedia.org/wiki/Floodfill`) orientieren.
Beispiel: floodArea(4,2)

```
1 0#000#              0#~~~#
2 0#00#~              0#~~#~
3 #000#~      —>      #~~~~~
4 000#~~              ~~~#~~
5 000#~~              ~~~#~~
```

floodArea soll zwei verschiedene Exceptions auslösen können:

- InvalidCoordinatesException , wenn die angegebenen Koordinaten nicht gültig sind, d.h. außerhalb der Karte liegen.
- NoLeveeException, wenn die angegebenen Koordinaten nicht auf ein Deichstück zeigen.

6.4 Gültige Karten II (2.5P)

Ein weitere Bedingung für die Gültigkeit der Karte ist, dass darauf ein dichter Deich dargestellt ist. Schreiben Sie dazu die Methode **boolean** hasLeakProofLevee(), die genau dies bietet. Sie dürfen dabei davon ausgehen, dass alle Wasserelemente zusammenhängen, es also nicht zwei getrennte

Seen oder Meere gibt. Ergänzen Sie den Konstruktor und setArea um diese Überprüfung, und lösen Sie gegebenenfalls eine LeakyLeveeException aus, die auch von InvalidAreaException erbt.
Hinweis: Sie können zur Lösung dieser Aufgabe einen ähnlichen Ansatz wie in der vorigen Aufgabe verwenden...

6.5 Benutzerinteraktion (2P)

Schreiben Sie eine Klasse GeoDialog (das Gerüst wird im Portal bereitgestellt), die den Benutzer mit dem GeoSimulator interagieren lässt. Die Klasse soll die Area mit vorgegebenen Werten initialisieren und den Benutzer über eine InvalidAreaException mit einer passenden Fehlermeldung informieren, außer es handelt sich um eine LeakyLeveeException. Danach soll die Klasse das aktuelle Gebiet anszeigen, vom Benutzer die x- und y-Koordinaten eines möglichen Deichbruchs erfragen, diesen simulieren und das resultierende Gebiet anzeigen. Dabei auftretende Fehlermeldungen sollen dem Benutzer auch kurz mitgeteilt werden.

Lösungsvorschlag:

```
1  /**
2   * Simulation of an area (consisting of water ~, ground 0 and levees #
3   * It can simulate a crevasse if its coordinates are given
4   *
5   * @author melanie
6   *
7   */
8  public class GeoSimulation {
9     private char area[][];
10
11    /**
12     * Initializes the world with the given array
13     * @param arr Chararray that represents the world
14     * @throws LeakyLeveeException
15     * @throws NoLeveeElementException
16     * @throws NoGroundElementException
17     * @throws NoWaterElementException
18     */
19    public GeoSimulation(char arr[][]) throws InvalidAreaException  {
20       setArea(arr);
21    }
22
23    /**
24     * Sets the world to the given array
25     * @param arr Chararray that represents the world
26     * @throws NoWaterElementException
27     * @throws NoGroundElementException
28     * @throws NoLeveeElementException
29     * @throws LeakyLeveeException
30     */
31    public void setArea(char arr[][]) throws InvalidAreaException  {
32       this.area = arr;
33       if (!hasWaterElement()) throw new NoWaterElementException();
34       if (!hasGroundElement()) throw new NoGroundElementException();
35       if (!hasLeveeElement()) throw new NoLeveeElementException();
36       if (!hasLeakProofLevee()) throw new LeakyLeveeException();
37
38    }
```

```
39
40      /**
41       * Gets the current area
42       * @return the area
43       */
44      public char[][] getArea() {
45          return area;
46      }
47
48
49      /**
50       * Takes coordinates of a crevasse and simulates it.
51       * @param x coordinate of the crevasse
52       * @param y coordinate of the crevasse
53       * @throws NoLeveeException
54       * @throws InvalidCoordinatesException
55       */
56      public void floodArea(int x, int y) throws NoLeveeException,
          InvalidCoordinatesException {
57          int height = this.area.length,
58          width = this.area[0].length;
59          if (area[x][y]!='#')
60              throw new NoLeveeException("There was no levee to break");
61          if ((x < 0 || x >= height) && (y < 0 || y >= width))
62              throw new InvalidCoordinatesException("Invalid Coordinates");
63          //Destroy the levee
64          area[x][y] = '0';
65
66          floodAreaRec(x,y);
67      }
68
69
70      private void floodAreaRec(int x, int y){
71          if (x<0 || x>=area.length || y<0 || y>=area[0].length) return;
72          if (area[x][y]=='#' || area[x][y]=='~') return;
73
74          area[x][y] = '~'; // set flood value first, before we start the
75          // recursion..
76
77          floodAreaRec(x + 1, y);
78          floodAreaRec(x - 1, y);
79          floodAreaRec(x, y + 1);
80          floodAreaRec(x, y - 1);
81      }
82
83
84
85      private boolean hasElement(char element){
86          for (int x=0; x<area.length; x++){
87              for (int y=0; y<area[0].length; y++){
88                  if (area[x][y]==element) {
89                      return true;
90                  }
91              }
92          }
93          return false;
94      }
95
```

```
96      /**
97       * Returns whether the current area has a ground element
98       * @return
99       */
100     private boolean hasGroundElement(){
101        return hasElement('0');
102     }
103
104     /**
105      * Returns whether the current area has a water element
106      * @return
107      */
108     private boolean hasWaterElement(){
109        return hasElement('~');
110     }
111
112     /**
113      * Returns whether the current area has a levee element
114      * @return
115      */
116     private boolean hasLeveeElement(){
117        return hasElement('0');
118     }
119
120     /**
121      * Returns a point representing the first water element
122      * @return point that represents the first water element
123      */
124     private Point getWaterElement(){
125        for (int x=0; x<area.length; x++){
126           for (int y=0; y<area[0].length; y++){
127              if (area[x][y]=='~') {
128                 return new Point(x,y);
129              }
130           }
131        }
132        return null;
133     }
134
135     /**
136      * Returns whether the border of the levee is leak-proof
137      * @return true if the border of the levee is leak-proof
138      */
139     public boolean hasLeakProofLevee(){
140
141        //get an arbitrary water element ...
142        Point point = getWaterElement();
143        //... and start flooding from there
144        boolean result = hasLeakProofLeveeRec(point.getX(), point.getY());
145
146        //Unmark all visited water elements v -> ~
147        for (int x=0; x<area.length; x++){
148           for (int y=0; y<area[0].length; y++){
149              if (area[x][y]=='v') area[x][y]='~';
150           }
151        }
152
153        return result;
```

```
154     }
155
156
157     private boolean hasLeakProofLeveeRec(int x, int y){
158         if (x<0 || x>=area.length || y<0 || y>=area[0].length) return true;
159         if (area[x][y]=='#') return true;
160         if (area[x][y]=='0') return false;
161         if (area[x][y]=='v') return true;
162         //mark all visited water elements '~' -> 'v' to avoid that they are
                visited again
163         area[x][y]='v';
164
165         return (hasLeakProofLeveeRec(x + 1, y) &&
166                 hasLeakProofLeveeRec(x - 1, y) &&
167                 hasLeakProofLeveeRec(x, y + 1) &&
168                 hasLeakProofLeveeRec(x, y - 1));
169     }
170
171     /**
172      * Returns a string representation of the curren map
173      */
174     public String toString() {
175         StringBuffer result = new StringBuffer();
176
177         for (int y = 0; y < this.area[0].length; y++) {
178             for (int x = 0; x < this.area.length; x++) {
179                 result.append(this.area[x][y]);
180             }
181             result.append("\n");
182         }
183         return result.toString();
184     }
185
186
187
188
189 }
```

```
1
2
3  /**
4   *  Represents a point consisting of x and y value
5   * @author melanie
6   *
7   */
8  public class Point {
9
10     private int x, y;
11
12     public Point(int x, int y) {
13         this.x = x;
14         this.y = y;
15     }
16
17     public int getX() {
18         return x;
19     }
20
```

```
21      public void setX(int x) {
22          this.x = x;
23      }
24
25      public int getY() {
26          return y;
27      }
28
29      public void setY(int y) {
30          this.y = y;
31      }
32
33
34
35  }
```

```
1
2
3   import acm.program.DialogProgram;
4
5   public class GeoDialog extends DialogProgram{
6
7       /**
8        * Loads a map and interacts with the user
9        */
10      public void run()  {
11          //Attention: This is columnwise
12          char arr[][] = { { '#', '~', '#' }, { '0', '#', '0' }, { '0', '0',
              '0' },
13                  { '0', '#', '0' }, { '#', '0', '#' } };
14
15          GeoSimulation obj=null;
16          try {
17              obj = new GeoSimulation(arr);
18          } catch(LeakyLeveeException e){
19
20          }catch (InvalidAreaException e) {
21              println("Your map is not valid");
22          }
23
24          int x = 0;
25          int y = 0;
26          while(true){
27              println(obj);
28              println("X: ");
29              x = readInt();
30              if (x==-1) System.exit(0);
31              println("Y: ");
32              if (y==-1) System.exit(0);
33              y = readInt();
34              try {
35                  obj.floodArea(x, y);
36              } catch (NoLeveeException e) {
37                  println(e.getMessage());
38              } catch (InvalidCoordinatesException e) {
39                  println(e.getMessage());
40              }
41          }
```

```
42
43
44      }
45
46       /**
47         * Main method. This is automatically called by the environment when
              your
48         * program starts.
49         *
50         * @param args
51         */
52        public static void main(String[] args) {
53          new GeoDialog().start();
54        }
55
56
57  }
```

```
1   /**
2    * An InvalidAreaSyntaxException represents an invalid
3    * area definition, i.e. it does not contain at least
4    * one ~, # and 0 element.
5    *
6    * @author Melanie Hartmann for GdI / ICS 1
7    * @version 1.0
8    */
9
10  public class InvalidAreaException extends Exception {
11
12  }
```

```
1
2
3   /**
4    * NoWaterElementException ist thrown if an area does not
5    * contain a water element (~)
6    *
7    * @author melanie
8    *
9    */
10  public class NoWaterElementException extends InvalidAreaException {
11
12  }
```

```
1
2   /**
3    * NoLeveeElementException ist thrown if an area does not
4    * contain a levee element (#)
5    *
6    * @author melanie
7    *
8    */
9   public class NoLeveeElementException extends InvalidAreaException {
10
11  }
```

```
1   /**
```

```
 * NoGroundElementException ist thrown if an area does not
 * contain a levee element (0)
 *
 * @author melanie
 *
 */
public class NoGroundElementException extends InvalidAreaException {

}
```

```


/**
 * A LeakyBorderException is thrown in the given area contains
 * a leaky levee
 *
 * @author melanie
 *
 */
public class LeakyLeveeException extends InvalidAreaException {

}
```

```


/**
 * InvalidCoordinatesException is thrown if invalid coordinates were
 * entered by the user
 * @author melanie
 *
 */
public class InvalidCoordinatesException extends Exception {

    /**
     * Constructor that takes an errormessage and stores it
     * @param string
     */
    public InvalidCoordinatesException(String string) {
        super(string);
    }


}
```

```


/**
 * NoLeveeException is thrown if the user did not enter
 * the coordinates of a levee element (#)
 * @author melanie
 *
 */
public class NoLeveeException extends Exception{

    /**
     * Constructor that takes an errormessage and stores it
     * @param string
```

```
14      */
15     public NoLeveeException(String string) {
16         super(string);
17     }
18
19
20 }
```