Hochschule Ingolstadt WS2010/2011

Studiengänge: Elektro- und Informationstechnik, Mechatronik,
Informatik, Flug- und Fahrzeuginformatik, Wirtschaftsinformatik

Prüfung
Grundlagen der Programmierung 2
Objektorientierte Programmierung
Softwareentwicklung 2

Prüfer:	Prof. Glavina, Prof. Hahndel, Prof. Schmidt, Prof. Windisch
Zeit:	90 Minuten
Hilfsmittel:	keine

Studiengang	Dozent	Matrikel-nummer	Semester	Raum	Platz

(Bitte sorgfältig schreiben!)

Aufgabe	1	2	3	4	5	$\sum$	Note
Punkte							

Schreiben Sie Ihre Antworten direkt in die dafür vorgesehenen Stellen der Seiten dieser Prüfungsangabe. Die Angabe besteht (inkl. Deckblatt) aus 14 Seiten. (**Wichtig:** Bitte geben Sie wieder alle Blätter ab, auch wenn einzelne Seiten nicht beschrieben sein sollten. Tragen Sie bitte SOFORT Ihre persönlichen Angaben sowie den Dozenten oben vor diesem Absatz ein.

Viel Erfolg!

Aufgabe 1: (Allgemeine Fragen, ca. 20%)

a) Warum muss die Methode main() eine Klassenmethode sein und keine Instanzmethode?

b) Was versteht man unter dem Begriff Bytecode und welchen Zweck hat dieser?

c) Mit welchem Operator wird der Speicherplatz für ein neues Objekt bereitgestellt ?

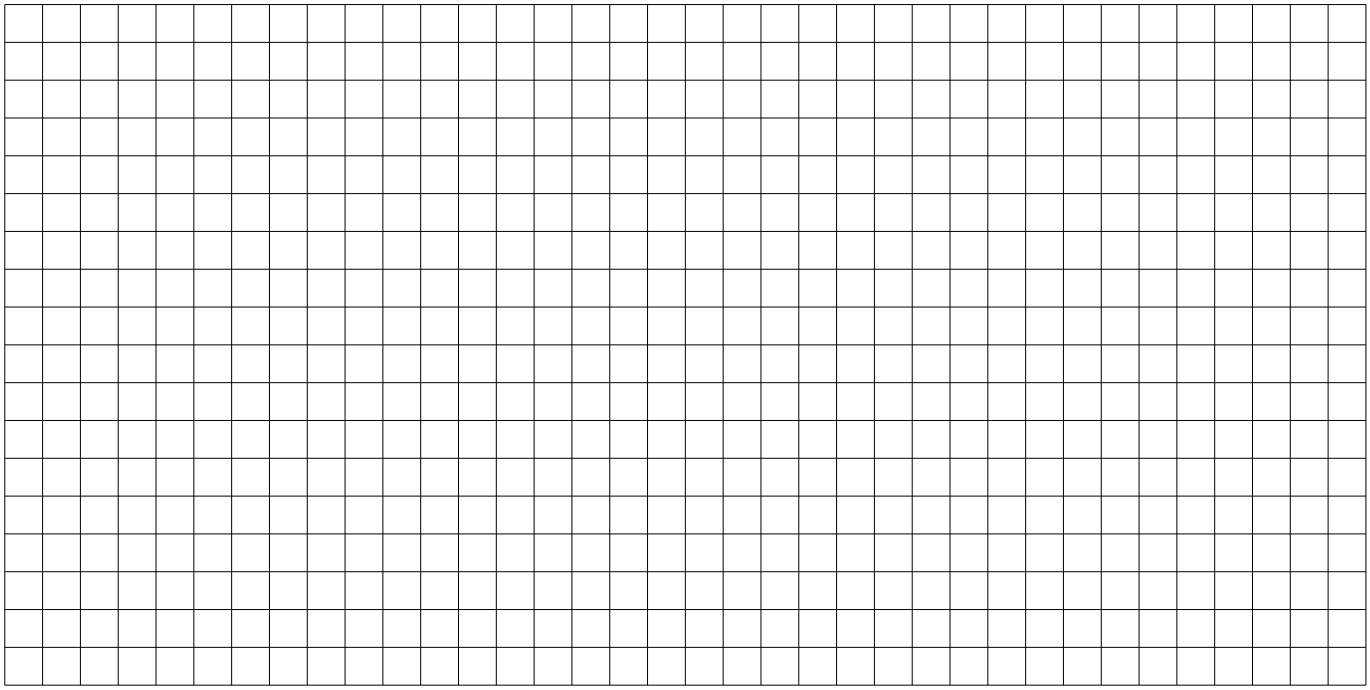
d) Nennen Sie zwei Unterschiede zwischen einer abstrakten Klasse und einem Interface.

e) Der folgende Java-Code enthält mehr als 10 Fehler, von denen Sie mindestens 7 verschiedene Fehler finden, erklären und berichtigen sollen. Hier ein Beispiel:
Fehler in Zeile 7: Ein Konstruktor muss genauso heißen, wie die Klasse. Groß-/Kleinschreibung wird unterschieden! berichtigte Zeile 7: `public Complex(float r, float i) {`

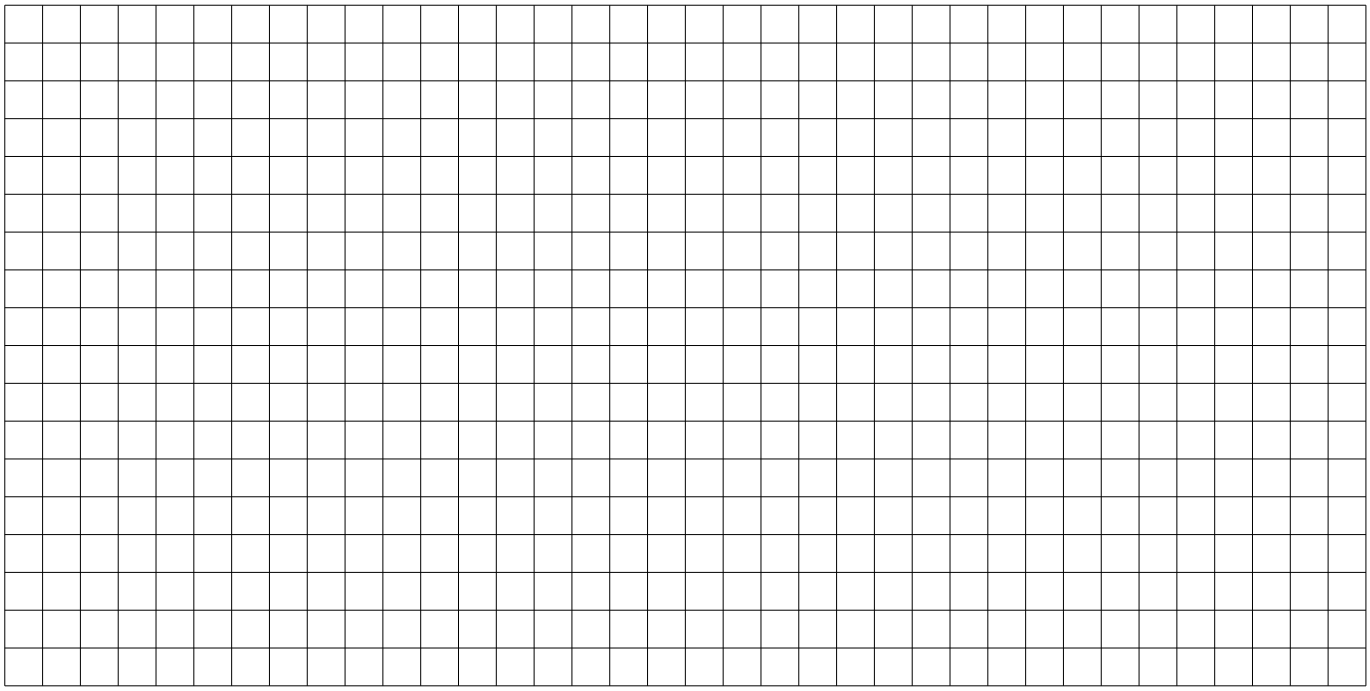
```
1  // Klasse Complex fuer komplexe Zahlen
2  class Complex {
3          private float x;
4          private float y;
5
6          //Konstruktor fuer die Klasse Complex
7          public complex(float r, float i) {
8                  float x = r;
9                  float y = i;
10         }
11         // Getter:
12         public int Re() { return x;}
13         public int Im() { return y;}
14 }
15
16         // Unterklasse von Complex
17 class BComplex implements Complex {
18         public void setRe(float r) {
19                 x = r;
20                 return null;
21        }
22       public void setIm(float i) {
23                 y = i;
24                 return;
25     }
26 }
27
28  //Client class: Java program zum testen der beiden obigen Klassen
29  class TestComplex {
30      public void main(String arg) {
31         Complex c = new BComplex(1.0f,2.0f);
32         c.setRe(3.0);
33         System.out.println(c.x);
34         System.out.println(c instanceof BComplex);
35      }
36  }
```

Aufgabe 2: (Interfaces und Generics, ca. 20%)

a) Schreiben Sie ein generisches Interface `Function` mit zwei Typparametern `X` und `Y`. Dieses Interface spezifiziert eine Methode `apply` mit einem Parameter vom Typ `X` und einem Rückgabewert vom Typ `Y`



b) Implementieren Sie das Interface `Function` in einer Klasse `StringFunction<String,Integer>`, so daß die Methode `apply` die Länge a eines Stringparameters zurückgibt.



c) Ergänzen Sie die unten gegebene Klasse `MapApp` um eine statische Methode `map` mit zwei Parametern:

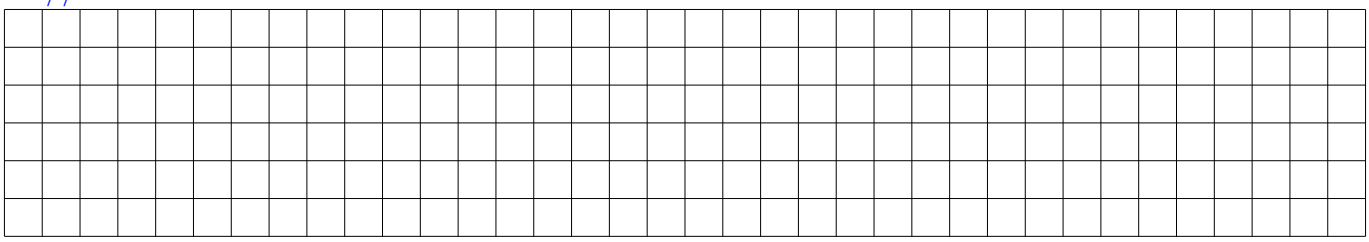
- einem Objekt der Klasse `StringFunction`
- einem Objekt welches das Standardinterface `java.util.List` implementiert.
- Das Ergebnis von `map` soll eine neue Liste sein, die ensteht, wenn man die Methode `apply` des ersten Parameters auf jedes Element des zweiten Parameters anwendet.

```
import java.util.*;

public class MapApp {
    public static void main(String[] args) {
        StringFunction f = new StringFunction();
        List<String> l = new ArrayList<String>();
        List<Integer> intListe = map(f,l);
        // ...
    }
}
```

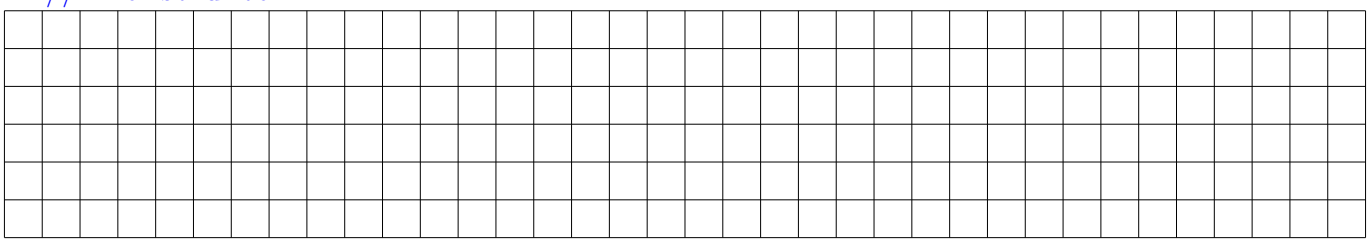
Aufgabe 3: (Abstrakte Klassen, Polymorphismus, ca. 23%)

Gegeben seien die folgenden Klassendefinitionen für ein Brettspiel in Java:

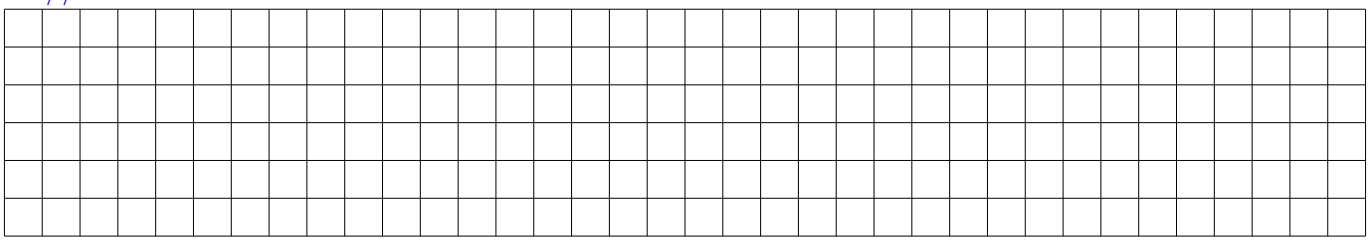
```
enum Color { rot, gruen, blau, gelb }

abstract class Spielfigur {
   protected int wert;
   protected Color farbe;

   // Konstruktor und Getter
   Spielfigur(Color farbe){ this.farbe = farbe; }
   int getWert(){ return wert; }
   Color getFarbe() { return farbe; }
}

class Maus extends Spielfigur {
   // Konstruktor:
```



```
   // ... weitere Methoden nicht aufgabenrelevant
}
class Katze extends Spielfigur {
   // Konstruktor:
```



```
   // ... weitere Methoden nicht aufgabenrelevant
}
class Hund extends Spielfigur {
   // Konstruktor:
```



```
   // ... weitere Methoden nicht aufgabenrelevant
}
```

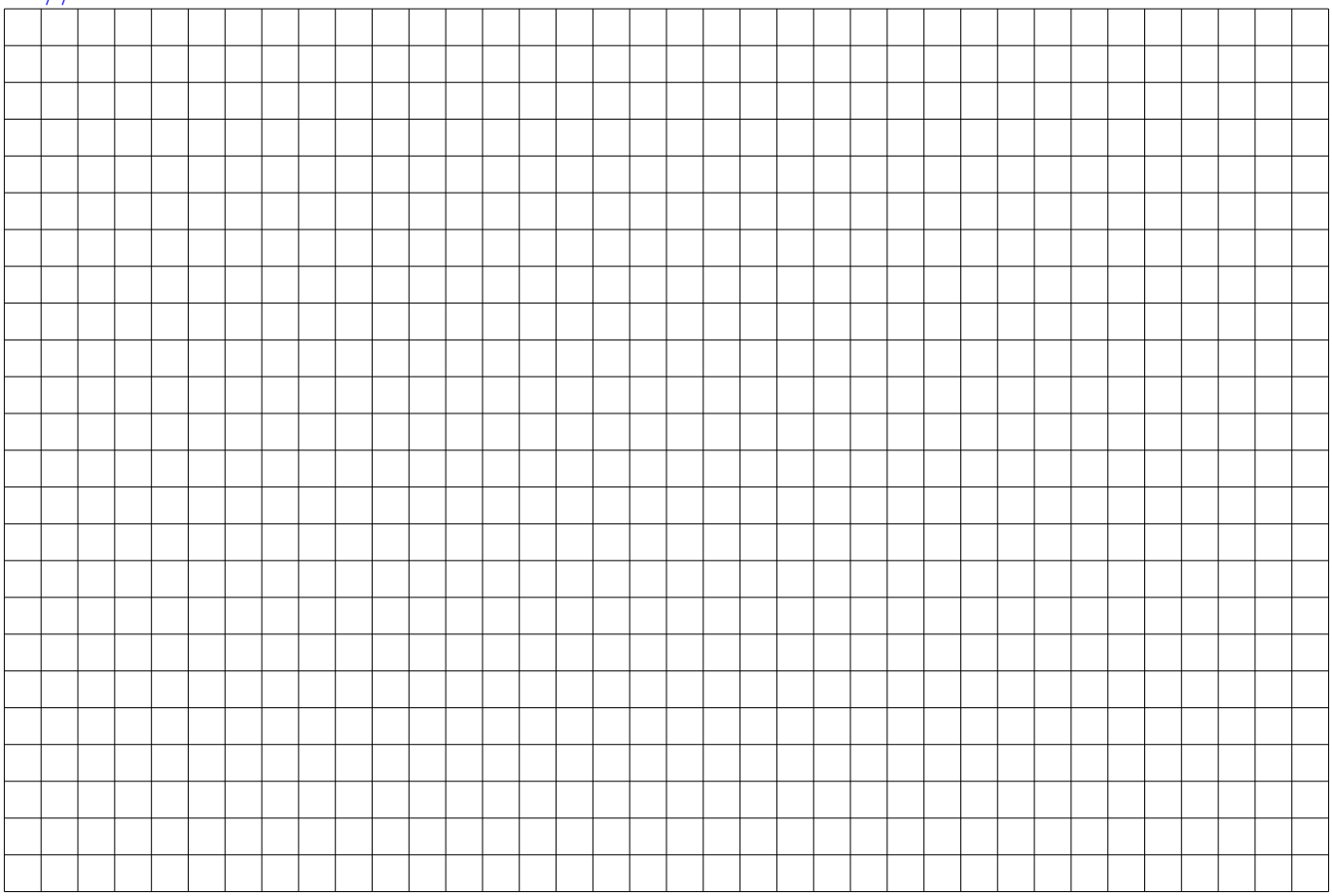
a) Ergänzen Sie die Klassen `Maus`, `Hund` und `Katze` um einparametrige Konstruktoren, die Farbe und Wert für die Spielfigur setzen. Der Parameter ist vom Typ `Color`. Die Spielfigur Maus hat den Wert 2, Katze den Wert 4 und Hund den Wert 6. Verwenden Sie dazu eine geeignete Konstruktorverkettung.

b) Ergänzen Sie das unten gezeigte Klassenfragment `Spielbrett` um eine `main`-Methode, die ein zweidimensionales (10x10)-Array vom Typ `Spielfigur` erzeugt. Dieses soll wie folgt vorbelegt werden:

- Eine `Maus` mit Farbe `Color.blau` auf Position (0,1)
- Eine `Maus` mit Farbe `Color.rot` auf Position (0,2)
- Eine `Hund` mit Farbe `Color.gelb` auf Position (0,3)
- Eine `Katze` mit Farbe `Color.rot` auf Position (0,4)

Anschließend wird mit dem Array ein Objekt vom Typ `Spielbrett` erzeugt und der Integerwert ausgegeben, den folgender Methodenaufruf zurück gibt:
`spielbrett.gesamtwert(Color.rot)`

```
class Spielbrett {
   private Spielfigur [][] spielbrett;

   Spielbrett(Spielfigur[][] stellung) {
        spielbrett = stellung;
   }

   // Schreiben Sie hier Ihre main-Methode:
```



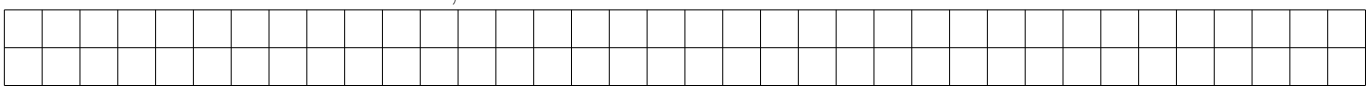
```
   // Methode gesamtwert wird in der naechsten Teilaufgabe erstellt
   // ... restliche Methoden ohne Belang
}
```

c) Schreiben Sie für die Klasse `Spielbrett` eine Methode `gesamtwert`, die ein Argument vom Typ `Color` bekommt und die Summe der Einzelwerte aller Spielfiguren zurückgibt, die dieser Farbe entsprechen.

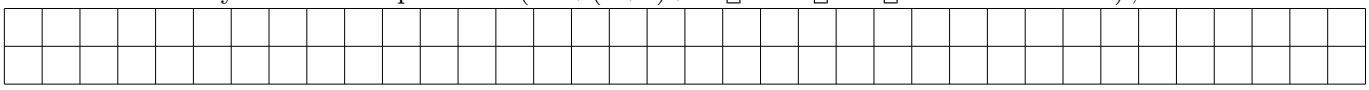
Aufgabe 4: (Exceptions , ca. 15%)

Gegeben sei folgendes Programm, die wobei die Klasse `IPAdresse` eine IP-Adresse beschreibt:

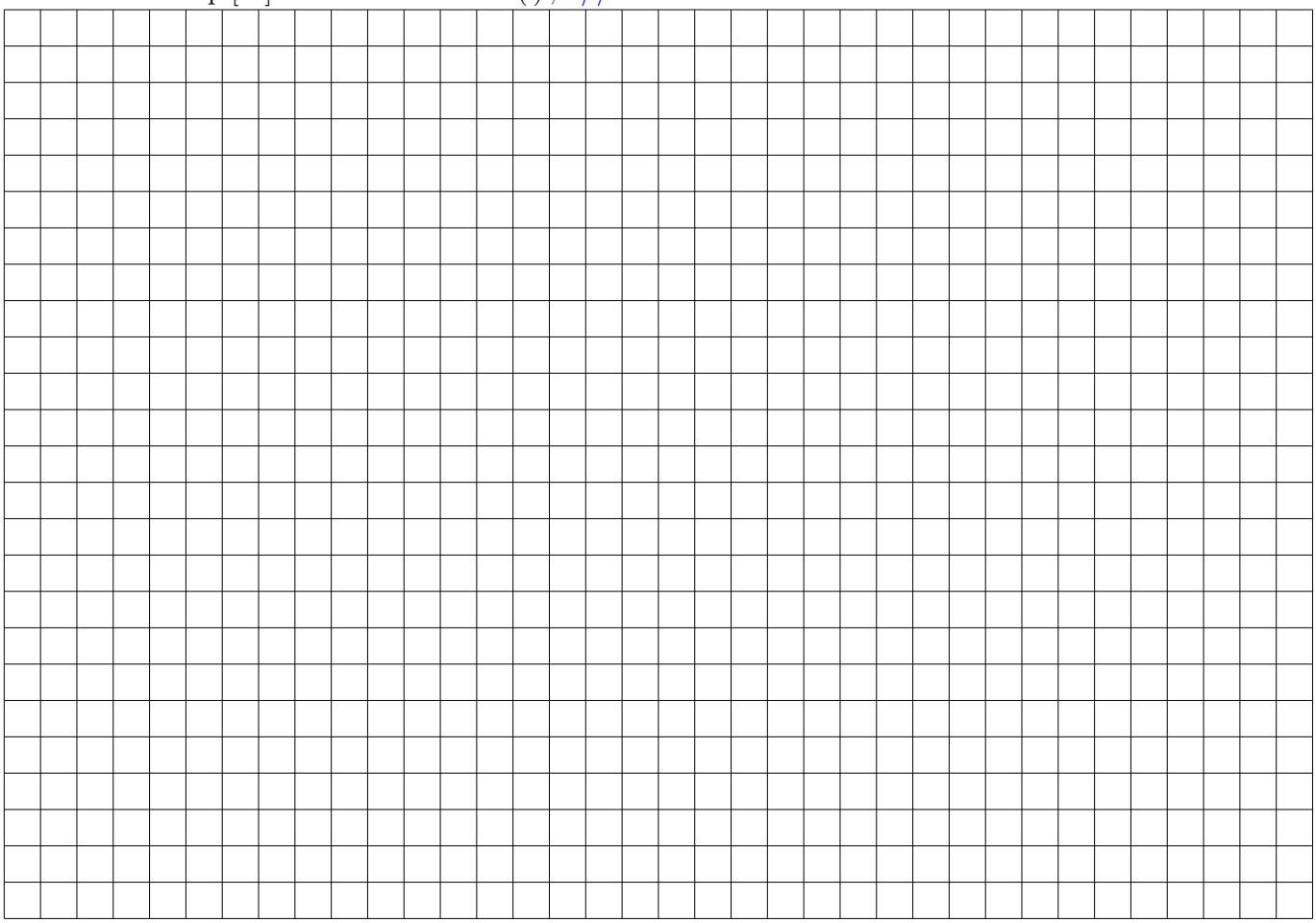
```
import java.io.*;

class ExceptionApp {
     public static void main(String[] args) {
        // Werte von der Tastatur einlesen:
        Scanner din = new Scanner(System.in);

        int [] ip = new int[4];  // Array fuer Adresskomponenten
        IPAdresse ipAdresse = null;
        boolean nochmal = true;

        for (int i = 0; i<4; i++){
            nochmal = true;
```



```
            System.out.println(""+(i+1)+". Teil der IP-Adresse:");
```



```
            ip[i] = din.nextInt(); // ***
```

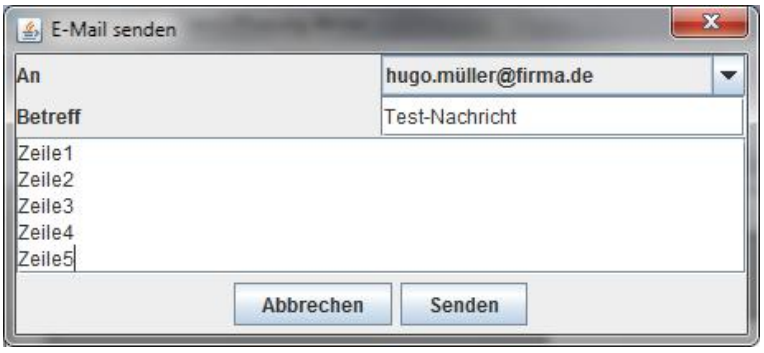


```
        }// end for
            System.out.println("Neue IP-Adresse-Objekt erzeugen:");
            ipAdress = new IPAdresse(ip); // IP-Adresse erzeugen
            // Weitere Bearbeitung ..
     }// end main
}// end class definition
```

a) Bei `***` kann eine `NumberFormatException` oder eine `IOException` auftreten, Außerdem darf der Benutzer nur eine Zahl zwischen 0 und 255 eingeben. Ergänzen Sie das obige Programmfragment um eine geeignete Exceptionbehandlung die die beiden vorgenannten Exceptions geeignet abfängt und eine erneute Eingabe der Zahl verlangt. Falls der Nutzer eine Zahl außerhalb des erlaubten Bereichs eingibt soll eine `NoIPPartException` erzeugt und derart abgefangen werden, daß die entsprechende Zahl ebenfalls neu eingelesen wird. Nutzen Sie dazu unter Verwendung der vorgegebenen Variablen `nochmal` ein geeignetes Schleifenkonstrukt.

b) Geben Sie nachfolgend eine passende Definition für `NoIPPartException` an.

Aufgabe 5: (GUI-Programmierung, ca. 23%)



Der in der obigen Abbildung dargestellte Dialog zum Erstellen und Senden von E-Mail-Nachrichten soll mit Java-Swing realisiert werden. Folgende Hinweise sind dabei zu beachten: Der Nachrichtenempfänger (s. An-Feld) wird aus einer Liste ausgewählt. Die Liste wird mit Hilfe der statischen öffentlichen Methode `String[] gibAdressen()` der Klasse `MailadressenDatenbank` initialisiert. Die Nachricht selbst wird in ein mehrzeiliges Textfeld eingetragen. Dieses bietet Platz für 5 Zeilen mit je 40 Spalten. Das Klicken des Abbrechen-Buttons führt zum Beenden des Programms, bei Klicken auf den Senden-Button werden die Eingaben einfach auf die Konsole ausgegeben. Die zur obigen Abbildung dazugehörige Ausgabe sieht wie folgt aus:

```
An: hugo.müller@firma.de
Betreff: Test-Nachricht
Nachricht: Zeile1
Zeile2
Zeile3
Zeile4
Zeile5
```

Geben Sie eine Java-Klasse an, die den oben beschriebenen E-Mail-Dialog realisiert. Der Dialog soll mit dem Konstruktor-Aufruf angezeigt werden, wobei der Aufruf selbst nicht zu programmieren ist. Import-Anweisungen sind nicht anzugeben.

Hinweis: Die Listenauswahl kann mit der GUI-Klasse `JComboBox` realisiert werden die über einen einparametrigen Konstruktor verfügt, mit dem die Liste vorinitialisiert werden kann, sowie über eine Methode `Object getSelectedItem()`, welches die aktuelle Auswahl zurückliefert.