Übungen zu Kapitel 16

16.1 Abfangen von Ausnahmen: Im folgenden Programm werden - getriggert durch einen Zufallsgenerator - verschiedene Ausnahmen ausgelöst. Erweitern Sie das Coding, so dass diese Ausnahmen einzeln abgefangen und die ***exception message*** am Bildschirm ausgegeben wird.

```
public class ExceptionTest {
   public static void main( String[] args ) {
      Object obj = null;
      int rd = (int)(Math.random() * 3);
      if (rd == 0)
         rd = 7/0;
      if (rd == 1)
         new String( "Hallo" ).charAt(13);
      if (rd == 2)
         obj.equals( obj );
   }
}
```

16.2 Ausnahme und Endlos-Schleife: Die uns aus früheren Kapiteln bekannte Klasse `IO` stellt die Methode

```
public static int promptAndReadInt( String s )
```

zur Verfügung, die den Benutzer mit dem `String s` zur Eingabe auffordert und die Benutzereingabe als `int`-Wert zurück gibt. Kann die Eingabe des Benutzers nicht in korrekt konvertiert werden, wird eine `NumberFormatException` ausgelöst.

Vervollständigen Sie im folgenden Programm die Methode `promptAndReadInt()`, so dass durch inkorrekte Benutzereingaben verursachte Ausnahmen abgefangen werden und der Benutzer nochmals zur Eingabe aufgefordert wird. Nach dem ersten Fehlversuch soll anstelle von `String s1` der `String s2` geprompted werden. Selbstverständlich kann die Klasse `IO` verwendet werden.

Tipp: Fangen Sie die Ausnahme innerhalb einer Endlos-Schleife ab, die nur im Erfolgsfall verlassen wird.

```
public class IOTest {
   public static void main( String[] args ) {
      int y;
      y = promptAndReadInt( "Bitte geben Sie Ihr Alter ein: ",
         "Bitte geben Sie Ihr Alter als ganze Zahl ein: " );
      System.out.println( "Sie sind " + y + " Jahre alt!" );
   }
   public static int promptAndReadInt( String s1, String s2 ) {
   // ...
   }
}
```

16.3 Ausnahmen und Benutzereingabe: Vervollständigen Sie in der folgenden Klassen die Methode `prompt4Bruch()`, die den Benutzer auffordert einen Bruch als Zähler und Nenner einzugeben. Ausnahmen des Typs `NumberFormatException` oder `IllegalArgumentException` sollen so lange abgefangen werden, bis der Benutzer erlaubte Werte eingibt. Die Klassen `IO` und `Bruch` können verwendet werden. Beachten Sie auch die Übung 16.2.

```
public class BruchTest {
   public static void main( String[] args ) {
      Bruch b1 = prompt4Bruch();
      Bruch b2 = prompt4Bruch();
      System.out.println( "Die Summe von " + b1 + " und " b2 +
                          " ist " + b1.add( b2) );
   }
   static Bruch prompt4Bruch() {
      int z, n;
      // Zähler und Nenner beim Benutzer erfragen (Ausnahme abfangen)
      // ...
      // Bruch erzeugen und zurückgeben (Ausnahme abfangen)
      // ...
   }
}
```

16.4 Erneutes Auslösen: Im folgenden Programm wird die Methode `throwExc()` aufgerufen, die zufällig und mit gleicher Wahrscheinlichkeit eine dieser Anweisungen ausführt:

- Auslösen einer `RuntimeException`
- Auslösen einer `Exception`
- kein Auslösen einer Ausnahme, einfaches `return`

Erweitern Sie die Methode `main()`, so dass nur kontrollierte Ausnahmen abgefangen werden. Wird also eine nicht-kontrollierte Ausnahme ausgelöst, soll das Programm mit der automatischen Ausgabe des ***exception stack*** terminieren. Zudem soll das Programm Informationen ausgeben, wenn eine kontrollierte Ausnahme abgefangen wurde oder wenn überhaupt keine Ausnahme aufgetreten war. Fügen Sie schließlich einen `finally`-Block hinzu, der Informationen über das Programmende ausgibt.

Tipp: Formulieren Sie `catch`-Blöcke für die beiden Ausnahmeklassen und lösen Sie abgefangene Ausnahmen vom Typ `RuntimeException` mittels `throw` erneut aus.

```
public class ThrowAgain {
   public static void main( String[] args ) {
   }
   static void throwExc() throws Exception {
      int rd = (int)(Math.random() * 3);
      if( rd == 0 )
         throw new RuntimeException();
      if( rd == 1 )
         throw new Exception();
      return;
   }
}
```

16.5 Finally und return: Ein `finally`-Block sollte nicht mit `return` abgeschlossen werden, da dieser implizit alle ausgelösten Ausnahmen abfängt. Verifizieren Sie diesen Sachverhalt, indem Sie im vorigen Programm den `finally`-Block entsprechend ergänzen.

16.6 Kontrollierte Ausnahmen: Erstellen Sie die Klassen `AException` und `BException`, die beide direkt von `Exception` erben, sowie ein Testprogramm mit den folgenden Methoden:

```
public static void throwAE() {
   throw new AException();
}
public static void throwBE() {
   throw new BException();
}
public static void callAB() {
   throwAE();
   throwBE();
}
```

Das Programm lässt sich in der vorliegenden Form nicht übersetzen, da in den Methoden kontrollierte Ausnahmen ausgelöst werden und die entsprechende `throws`-Deklaration im Methodenkopf fehlt. Ergänzen Sie in den Methoden die Zusätze

```
... throws AException ...
... throws BException ...
... throws AException, BException ...
```

und übersetzen Sie das Programm. Was passiert, wenn man bei der ersten Methode die Deklaration `throws AException` durch `throws Exception` ersetzt?

Wie verhält sich der Compiler, wenn man in `throwBE()` keine Ausnahme auslöst, den Zusatz `throws BException` im Methodenkopf aber beibehält?

Erklären Sie die Logik, mit der der Compiler solche kontrollierte Ausnahmen hinsichtlich `throws`–Deklaration überprüft.