Exceptions, Enums

Programmierung und Softwareentwicklung
Hörsaalübung

Schleifen – Aufgabe

Schreiben Sie eine Abfrage `double getAverageGrade()`, welche die durchschnittliche Note eines Studierenden berechnet. Gehen Sie davon aus, dass alle Noten gleich gewichtet sind. Sie können hierfür die folgenden Abfragen nutzen:

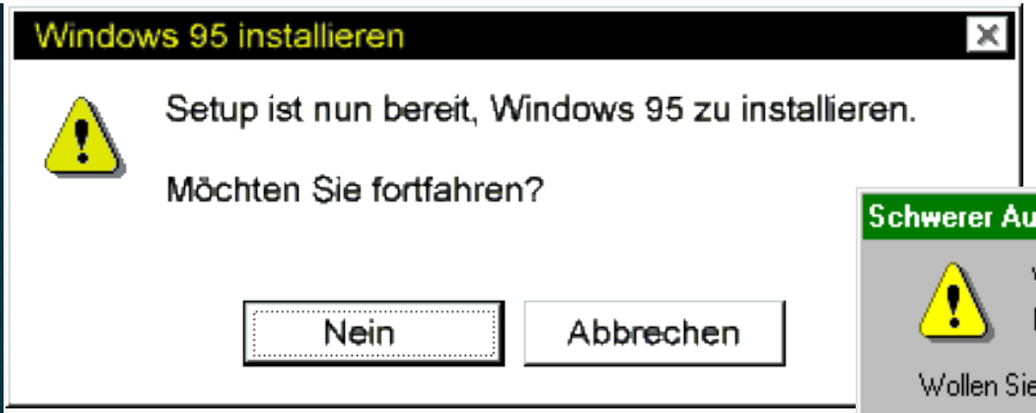
- `double getGrade(int i)` bekommt eine Zahl `i` übergeben und gibt die `i`-te Note der Studierenden zurück. Beachten Sie, dass die erste Note mit `i` = 0 zurückgegeben wird.
- `int numberOfGrades()` gibt die gesamte Anzahl an Noten übergeben.

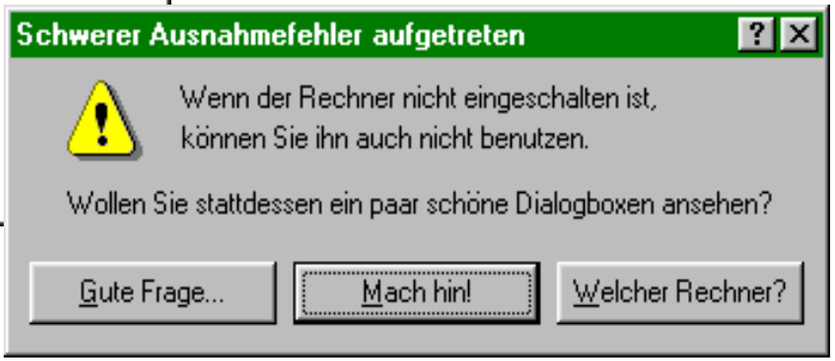
Besitzt der Studierende noch keine Noten, soll 0.0 zurückgegeben werden.

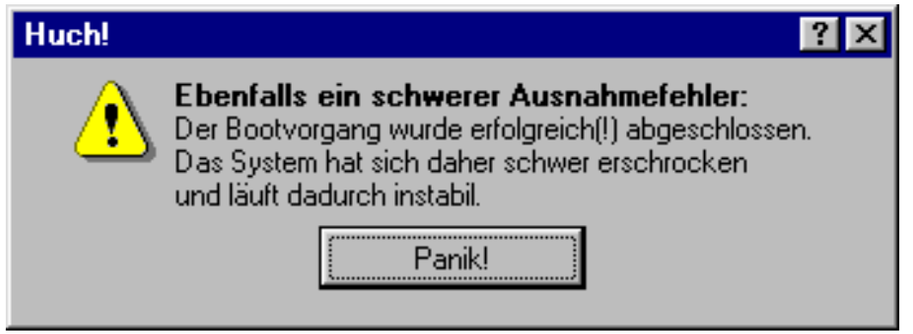
Schleifen – Aufgabe – Lösung

```
double getAverageGrade() {
    double sumOfGrades = 0.0;
    for (int i = 0; i < this.numberOfGrades(); ++i) {
        sumOfGrades += this.getGrade(i);
    }
    if (this.numberOfGrades() > 0) {
        return sumOfGrades / this.numberOfGrades();
    }
    return sumOfGrades;
}
```

Exception


Windows 95 installieren
Setup ist nun bereit, Windows 95 zu installieren.
Möchten Sie fortfahren?
Nein
Abbrechen


Schwerer Ausnahmefehler aufgetreten
Wenn der Rechner nicht eingeschalten ist,
können Sie ihn auch nicht benutzen.
Wollen Sie stattdessen ein paar schöne Dialogboxen ansehen?
Gute Frage...
Mach hin!
Welcher Rechner?


Huch!
Ebenfalls ein schwerer Ausnahmefehler:
Der Bootvorgang wurde erfolgreich(!) abgeschlossen.
Das System hat sich daher schwer erschrocken
und läuft dadurch instabil.
Panik!


Error success Info
The error was successfully performed.
Have a nice day.
OK

Problem

Was passiert, wenn man nicht genug Geld besitzt?

```
double money;
void pay(final double amount) {
    this.money -= amount;
}
```

Exceptions

Was sind Exceptions?

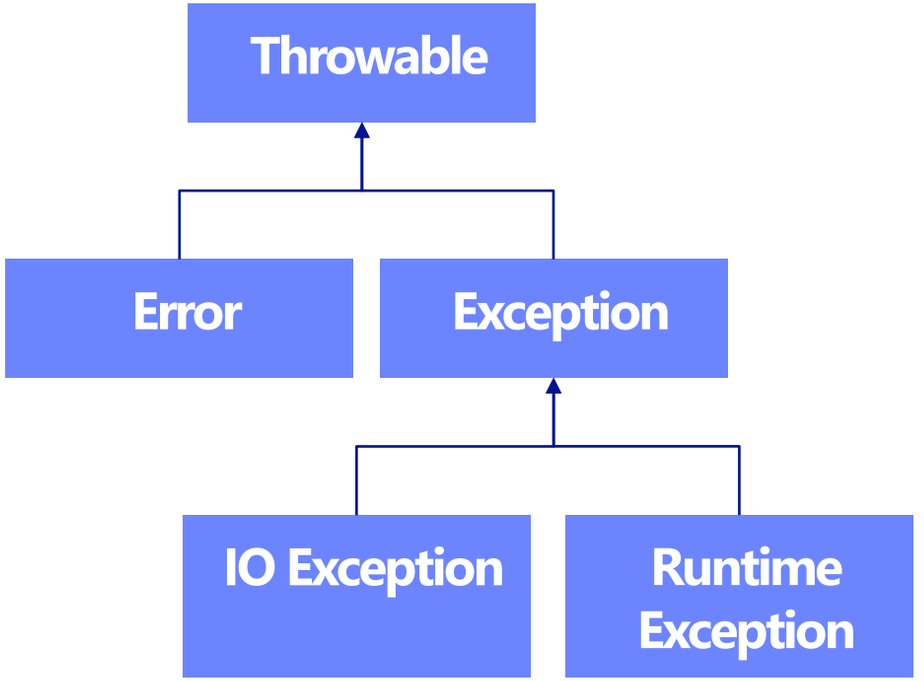
- Möglichkeit zur Fehlererkennung /-behandlung
- Machen auftretende Abstürze im Programmverlauf sicher behandelbar

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 7
        at de.unistuttgart.pse.vu.t3.Main.main(Main.java:10)
```

Fehlerklassen

Welche Fehlerklassen gibt es?

- Eigene Hierarchie für Fehlerbehandlung
- Vom Objekt `Throwable` abgeleitet
- Dutzende Ableitungen von den Hauptklassen `Error` und `Exception`
- Viele Standardoperationen zur Fehlerbehandlung



Nur ein kleiner Auszug des Hierarchiebaums!

Error & Exception

Error

- Repräsentiert schwere Fehler im System
- Programm wird vollständig beendet

Exception

- Benutzerdefinierte Ausnahmen

Exceptions werfen

```
double money;
void pay(final double amount) throws Exception {
    if (this.money < amount) {
        throw new Exception();
    }
    this.money -= amount;
}
```

Exceptions werfen

- `throw`-Befehl zum Werfen der Exception
- `new`-Befehl zum Objekt der Fehlerklasse erzeugen
- Ankündigung, dass die Operation im Laufe der Ausführung eine Exception auslösen könnte (`throws Exception`)

Exception vs RuntimeException

Exception

müssen explizit behandelt werden

- Operationsdeklaration muss `throws` beinhalten
- Oder Operation beinhaltet `try-catch` Block

```
public static void catchThat()
        throws Exception {
    throw new Exception();
}
```

RuntimeException

Programmiertechnisch vermeidbar

- NullPointerException (`obj != null`)
- ArrayIndexOutOfBoundsException (`index < size`)

```
public static void catchThat() {
    throw new RuntimeException();
}
```

Exceptions behandeln

Selber behandeln

```
void foo(Code object) {

    try {
        object.pay(5);
    } catch (Exception e) {
        System.err.println("Not enough money");
    } finally {
        System.err.println("Purchasing finished");
    }

}
```

Exceptions behandeln

Selber behandeln

`try`-Block

- überwachender Code

`catch`-Block

- Fehlermeldungen
- Kann mehrfach vorkommen, um auf alle Exceptions passend zu reagieren
- Nicht zutreffende Blöcke werden nicht ausgeführt
- Jeder Exceptiontyp darf nur einmal vorkommen

Exceptions behandeln

Selber behandeln

`finally`-Block

- optional, Bereich wird auf jeden Fall ausgeführt, egal ob Fehler in der Ausnahmenbehandlung oder nicht
- Nützlich für Ressourcenfreigabe oder beenden von I/O-Streams

```
try {
    student.work();
} catch (NoMotivationException e) {
    System.err.println("Too much to work");
} finally {
    student.drinkBeer();
}
```

Exceptions behandeln

Weiterwerfen

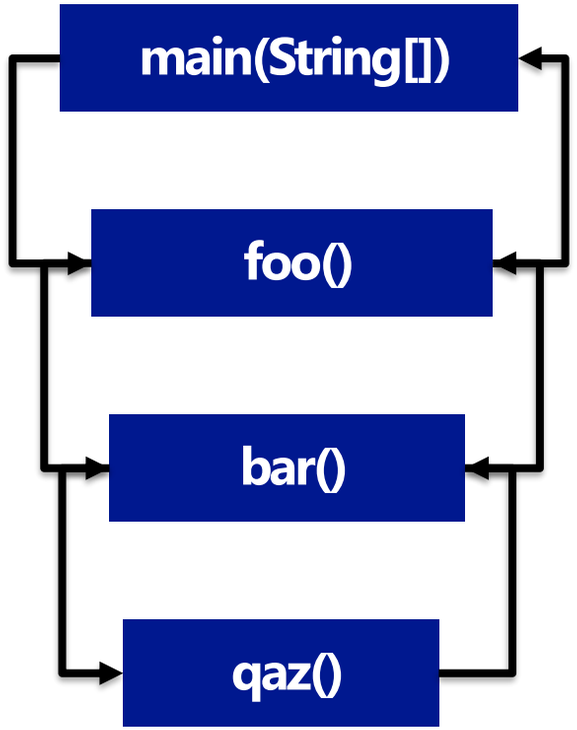
```java
public static void foo() throws Exception {
    bar();
}
```

```java
public static void bar() throws Exception {
    qaz();
}
```

```java
public static void qaz() throws Exception {
    throw new Exception();
}
```

```java
public static void main(String[] args)
                        throws Exception {
    foo();
}
```

AUFRUF



EXCEPTION

Eigene Exception

- Spezielle Fehlerbehandlungen für eigene Klassen
- Muss sich in die Exception-Hierarchie einordnen
  - Ableiten von einer Exception-Klasse der Hierarchie (am besten `Throwable`)
  - Nicht von `Error`-Klasse ableiten, da diese nur für schwere Fehler in der JVM reserviert ist!

```
class MyException extends Exception {
    public MyException(){}
    public MyException(final String s) {
        super(s);
    }
}
```

Exception - Aufgabe

Schreiben Sie eine Operation, die eine Zahl `n`, sowie ein `String` übergeben bekommt. Die Methode soll über den `String` iterieren und die ersten `n` Buchstaben einzeln auf der Konsole ausgeben.

Machen Sie sich Gedanken, ob Fehler auftreten können und behandeln Sie diese mit Exceptions.

Exception - Aufgabe - Lösung

```
public void getFirstNChars(int n, String s) throws IllegalArgumentException {
    if(n > s.length()) {
        throw new IllegalArgumentException("The number should not be greater "
            + "than string length");
    } else if (n < 0) {
        throw new IllegalArgumentException("The number should not be less than 0");
    } else if (s == null) {
        throw new IllegalArgumentException("The String must not be null");
    } else {
        for(int i = 0; i < n; ++i) {
            System.out.println(s.charAt(i));
        }
    }
}
```

Enum

Enum – Beispiel (I)

```
public enum Weekdays {
    Monday, Tuesday,
    Wednesday, Thursday,
    Friday, Saturday,
    Sunday
}
```

- Variable kann auf vordefinierte konstante Werte gesetzt werden
- Variable muss eine der Konstanten sein
- `enum` dann verwenden wenn alle Werte bereits zur Compilezeit bekannt sind
  - Bpsw: Wochentage, Planeten, Himmelsrichtungen, …
- `enum` Klassen können auch Funktionen beinhalten

Enum – Beispiel (II)

```
public enum Status {
    PASSED(1, "Passed", "The test has passed."),
    FAILED(-1, "Failed", "The test was executed but failed."),
    DID_NOT_RUN(0, "Did not run", "The test did not start.");

    private int code;
    private String label;
    private String description;

    private Status(int code, String label, String description) {
        this.code = code;
        this.label = label;
        this.description = description;
    }

    public int getCode() {
        return code;
    }
```

- `enum` kann neben Ordinalwerten auch anderer Attribute enthalten
- wird durch privaten Konstruktor automatisch gesetzt
- einzelne Attribute können durch getter-Operationen abgefragt werden

Enum – Beispiel (II) – cont'd

```
    public String getLabel() {
        return label;
    }

    public String getDescription() {
        return description;
    }

    @Override
    public String toString() {
        return this.label + " " + this.description
    }
}
```

Enum – Beispiel (III)

```java
import java.util.HashMap;
import java.util.Map;

public enum Status {
    PASSED(1, "Passed", "The test has passed."),
    FAILED(-1, "Failed", "The test was executed but failed."),
    DID_NOT_RUN(0, "Did not run", "The test did not start.");

    private int c;
    private String lbl;
    private String desc;

    private static Map<Integer, Status> cToStatusMapping;

    private Status(int c, String lbl, String desc) {
        this.c = c;
        this.lbl = lbl;
        this.desc = desc;
    }

    public static Status getStatus(int i) {
        if (cToStatusMapping == null) {
            initMapping();
        }
        return cToStatusMapping.get(i);
    }

    private static void initMapping() {
        cToStatusMapping = new HashMap<Integer, Status>();
        for (Status s : values()) {
            cToStatusMapping.put(s.c, s);
        }
    }

    public int getc() {
        return c;
    }

    public String getlbl() {
        return lbl;
    }

    public String getdesc() {
        return desc;
    }

    @Override
    public String toString() {
        return this.code + ":" + this.lbl + " - " + this.desc;
    }

    public static void main(String[] args) {
        System.out.println(Status.PASSED);
        System.out.println(Status.getStatus(-1));
    }
}
```

Enum – Aufgabe

Schreiben Sie eine Enum, welche alle 12 Monate beinhaltet

Enum – Aufgabe - Lösung

```
public enum Month {
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
}
```