Fehlerbehandlung in Java I — Exceptions

Laufzeitfehler, z.B. durch fehlerhafte Benutzereingaben, sind kaum zu vermeiden und schwer vorhersehbar. Mit dem Konzept der Exceptions (Ausnahmen) ermöglicht Java dem Programmierer eine gezielte Fehlerbehandlung.

Fehlerquellen im Projekt `DoME`

Aufgabe 1: Erzeugen Sie ein `Datenbank`–Objekt. Erfassen Sie ein Video und eine CD. Versuchen Sie das Medium mit dem Index `4` zu entfernen. Notieren und interpretieren Sie die Fehlermeldung des Sytems.

Aufgabe 2: Rufen Sie die Methode `erfasseMedium` auf und übergeben Sie anstatt eines Mediums einen Nullzeiger (`null`). Rufen Sie danach die Methode `auflisten` auf. Notieren und interpretieren Sie die Fehlermeldung.

Exceptions — programmtechnisch gesehen

Alle Exceptions sind Unterklassen der Klasse `Exception`. Bei einem Laufzeitfehler wird zunächst ein Objekt der entsprechenden Exceptionklasse erzeugt. Anschließend wird die Exception ausgelöst. Die laufende Methode bricht sofort ab und kehrt zur aufrufenden Methode zurück.

Aufgabe 3: Sehen Sie sich in der Java–Klassendokumentation die Klasse `Exception` und einige ihrer Unterklassen an.

Aufgabe 4: Erkunden Sie in der Klassendokumentation von `ArrayList`, welche Exceptions die Methode `remove(int index)` auslöst (Abschnitt *Method Detail*).

Aufgabe 5: Erklären Sie anhand des Fehlergeschehens aus Aufgabe 1 das Auslösen einer Exception.

Fehlerbehandlung in Java II — Exceptions abfangen

Nicht jeder Laufzeitfehler soll zum Programmabbruch führen. Gerade bei Benutzereingaben wäre das ausgesprochen lästig. Deshalb kann man Exceptions gezielt abfangen und behandeln. Die fehlerträchtigen Anweisungen werden in einen `try`–Block eingeschlossen, in einem oder mehreren `catch`–Blöcken werden Anweisungen für die Fehlerfälle festgelegt, z.B.:

```
ArrayList <Freund> freunde = new ArrayList();
Freund besterFreund;
...
try {
  besterFreund = freunde.get(5);
  besterFreund.anzeigen();
catch(IndexOutOfBoundException e) {       // Kein Element mit Index 5 vorhanden
  System.out.println("Freund nicht vorhanden");
}
catch(NullPointerException e) {           // Element mit Index 5 war null
  besterFreund = new Freund("Name unbekannt",0);
}
...
```

Ein `catch`–Block fängt jedoch nur Exceptions der angegebenen Klassen und ihrer Unterklassen ab. So führt eine `IllegalArgumentException` im obigen Beispiel zu einem Programmabbruch. Anweisungen nach der `try`–`catch`–Klausel werden dann nicht mehr ausgeführt.

Anweisungen, die im Fehlerfalle unbedingt noch ausgeführt werden sollen, z.B. das Sichern von Daten, schließt man deshalb in einen optionalen abschließenden `finally`–Block ein, z.B.:

```
finally {
  freunde.speichern();
}
```

Aufgabe 6: Fangen Sie in den Methoden `auflisten` und `entferneMedium` mögliche Exceptions ab und schreiben Sie geeignete Fehlerbehandlungen.

Aufgabe 7: Sichern Sie weitere gefährdete Abschnitte im Programmtext ab.

Fehlerbehandlung in Java III — Exceptions selbst auslösen

Anweisungen wie:

```
besterFreund = new Freund(null,-23);
```

sind programmtechnisch korrekt, inhaltlich jedoch unsinnig: Der beste Freund ist quasi noch nicht geboren, sein Name mit `null` anstatt mit einem „echten“ String belegt. Durch das Auslösen geeigneter Exceptions lässt sich die Erzeugung eines solchen `Freund`–Objektes unterbinden:

```
public Freund(String name,byte alter) {
  if(alter < 0) {
    throw new IllegalArgumentException("Negatives Alter");
  }
  else {
    this.alter = alter;
  }
  if(name == null) {
    throw new NullPointerException("Ungültiger Name");
  }
  else {
    this.name = name;
  }
}
```

Im Fehlerfall wird zuerst mit `new` ein Exceptionobjekt erzeugt und dann mit `throw` die Exception ausgelöst.

Aufgabe 8: Erkunden Sie, wo im Programmtext der Klasse `Medium` und ihrer Subklassen die Eingabe unsinniger Attributwerte möglich ist.

Aufgabe 9: Lösen Sie in diesen Abschnitten geeignete Exceptions aus.

Fehlerbehandlung in Java IV — Exceptions definieren

Selbst definierte Exceptionklassen lassen eine verfeinerte Fehlerbehandlung zu. So sagt z.B. eine `IllegalArgumentException` wenig darüber aus, inwiefern ein Wert ungültig ist. Aussagekräftiger sind eine `NegativesAlterException`, eine `LeererNameException` usw.

Eigene Exceptions sind direkte oder indirekte Subklassen von `Exception`, z.B.:

```
public class NegativesAlterException extends IllegalArgumentException
{
  public NegativesAlterException() {
    super();
  }
  public NegativesAlterException(String fehlermeldung) {
    super(fehlermeldung);
  }
}
```

Im Konstruktor der Klasse `Freund` wird jetzt die speziellere Exception ausgelöst:

```
throw new NegativesAlterException("Korrigieren Sie das Alter!");
```

Aufgabe 10: Erkunden Sie die verschiedenen Typen von Fehleingaben in der Klasse `Medium` und ihren Unterklassen.

Aufgabe 11: Definieren Sie geeignete Exceptionklassen. Nutzen Sie diese in den Medienklassen.