Handout: Ausnahmebehandlung in Java

Einführung

Ausnahmebehandlung (Exception Handling) in Java ist ein Mechanismus, um auf Fehler und unerwartete Zustände während der Programmausführung zu reagieren. Java bietet eine klare Struktur, um Laufzeitfehler abzufangen, zu behandeln und ggf. weiterzugeben.

Fehlertypen in Java

Fehlertyp	Beschreibung	Beispiele	Empfohlene Handlung
Checked Exceptions	Fehler, die vom Compiler überprüft werden. Müssen im Code behandelt oder deklariert werden.	`IOException`, `SQLException`	Verwende `try-catch` oder `throws`, um den Fehler abzufangen.
Unchecked Exceptions	Fehler, die zur Laufzeit auftreten. Werden nicht vom Compiler überprüft.	`NullPointerException`, `ArrayIndexOutOfBoundsException`	Vermeide fehlerhafte Logik oder behandle sie optional mit `try-catch`.
Errors	Schwere Fehler, die auf Systemebene auftreten. Können meist nicht durch den Code behoben werden.	`OutOfMemoryError`, `StackOverflowError`	Fehlerprotokollierung und Neustart des Systems.

Grundkonzepte der Ausnahmebehandlung

1. **Ausnahmen abfangen und behandeln**: Mit `try-catch` Blöcken können Fehler kontrolliert abgefangen und verarbeitet werden.
2. **Ausnahmen weitergeben**: Mit `throws` können Methoden deklarieren, dass sie bestimmte Fehler nicht selbst behandeln.
3. **Ausnahmen auslösen**: Mit `throw` kann eine Ausnahme manuell erzeugt werden.

Ausnahmebehandlung in Java: Beispiele

1. Exceptions abfangen und behandeln

Der `try-catch`-Block ermöglicht es, bestimmte Fehler zu erkennen und darauf zu reagieren.

```
```

```
public class AusnahmeBeispiel {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // Division durch null
        } catch (ArithmeticException e) {
            System.out.println("Fehler: Division durch null ist nicht erlaubt!");
        }
    }
}
```

Erklärung:

- Die Division durch null löst eine `ArithmeticException` aus.
- Der `catch`-Block fängt diese Ausnahme ab und verhindert einen Absturz des Programms.

---

2. Ausnahmen weitergeben

Eine Methode kann deklarieren, dass sie bestimmte Fehler nicht selbst behandelt, sondern an den Aufrufer weitergibt.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class AusnahmeWeitergeben {
    public static void main(String[] args) throws FileNotFoundException {
        leseDatei("nicht_existierende_datei.txt");
    }

    public static void leseDatei(String dateiname) throws FileNotFoundException {
        Scanner scanner = new Scanner(new File(dateiname));
        System.out.println(scanner.nextLine());
    }
}
```

Erklärung:

- Die Methode `leseDatei` verwendet `throws`, um eine `FileNotFoundException` an den Aufrufer weiterzugeben.
- Der Aufrufer (`main`) muss diese Ausnahme entweder abfangen oder selbst deklarieren.

---

3. Ausnahmen auslösen

Mit `throw` kann eine Ausnahme manuell erzeugt werden.

```
```

```
public class AusnahmeAuslösen {
    public static void main(String[] args) {
        try {
            überprüfeAlter(-5); // Ungültiges Alter
        } catch (IllegalArgumentException e) {
            System.out.println("Fehler: " + e.getMessage());
        }
    }

    public static void überprüfeAlter(int alter) {
        if (alter < 0) {
            throw new IllegalArgumentException("Alter darf nicht negativ sein.");
        }
        System.out.println("Alter ist gültig: " + alter);
    }
}
```

Erklärung:

- Wenn das Alter negativ ist, wird eine `IllegalArgumentException` ausgelöst.
- Der Fehler wird im `catch`-Block behandelt.

---

4. Eigene Exceptions erstellen

Benutzerdefinierte Exceptions können durch die Erweiterung der Klasse `Exception` erstellt werden.

```
public class EigeneException {
    public static void main(String[] args) {
        try {
            prüfePasswort("123"); // Unsicheres Passwort
        } catch (UngültigesPasswortException e) {
            System.out.println("Fehler: " + e.getMessage());
        }
    }

    public static void prüfePasswort(String passwort) throws
UngültigesPasswortException {
        if (passwort.length() < 6) {
            throw new UngültigesPasswortException("Das Passwort ist zu kurz.");
        }
        System.out.println("Passwort ist sicher.");
    }
}

// Benutzerdefinierte Exception
class UngültigesPasswortException extends Exception {
    public UngültigesPasswortException(String nachricht) {
        super(nachricht);
    }
}
```

Erklärung:

- Die Klasse `UngültigesPasswortException` erweitert `Exception` und definiert eine benutzerdefinierte Fehlermeldung.
- Die Methode `prüfePasswort` wirft diese Ausnahme, wenn das Passwort unsicher ist.

Laufzeitfehler (Unchecked Exceptions)

Fehlertyp	Beschreibung
NullPointerException	Aufruf einer Methode oder eines Attributs auf einem `null`-Objekt.
ArrayIndexOutOfBoundsException	Zugriff auf einen Index außerhalb der Array-Grenzen.
ArithmeticException	Fehler bei arithmetischen Operationen (z. B. Division durch null).

Beispiel: NullPointerException abfangen

```
public class NullPointerBeispiel {
    public static void main(String[] args) {
        try {
            String text = null;
            System.out.println(text.length()); // Führt zu NullPointerException
        } catch (NullPointerException e) {
            System.out.println("Fehler: Ein null-Objekt hat keine
Eigenschaften!");
        }
    }
}
```

Zusammenfassende Tabelle

Konstrukt	Verwendung
`try-catch`	Fängt Exceptions ab und verarbeitet sie.
`finally`	Führt einen Block immer aus, unabhängig davon, ob eine Ausnahme auftritt oder nicht.
`throws`	Deklariert, dass eine Methode eine Ausnahme an den Aufrufer weitergeben kann.
`throw`	Löst eine Ausnahme manuell aus.
Checked Exceptions	Müssen behandelt oder deklariert werden.
Unchecked Exceptions	Können optional behandelt werden. Treten typischerweise aufgrund von Programmierfehlern auf.

Konstrukt	Verwendung
Errors	Systemfehler, die nicht durch den Code behandelt werden können.