Arbeitsblatt 06 Wichtige Klassen

1 Einleitung

Java ist eine Programmiersprache die seit 1995 eingesetzt und weiterentwickelt wird. Es gibt die Standard-Edition (SE) und die Enterprise-Edition (EE). Nur wenige Programmiersprachen haben sich so lange am Markt behaupten können, es erstaunt deshalb nicht, dass sich in Java eine riesige Bibliothek von Klassen über die Jahrzehnte angesammelt hat. Waren es in der Java Version 1.3.1 noch 1840 Klassen sind es heute bei Java Version 18 schon mehr als 5000 Klassen.

Einige dieser Klassen werden Sie immer wieder über den Weg laufen, in diesem Arbeitsblatt werden Sie folgende vier der wichtigsten Klassen von Java etwas besser kennenlernen:

1. Die String Klasse
2. Die Date & Time Klassen
3. Die System Klasse
4. Exceptions

2 Die String Klasse

Die String Klasse gehört zum automatisch importierten java.lang Paket.

Das String-Objekt geniesst eine besondere Stellung unter den Objekten in Java. Es wird nicht wie andere Objekte mit dem *new* Schlüsselwort instanziiert, sondern mit dem Zuweisungsoperator =. Da es sich beim String-Typ um ein Objekt handelt, ist sein Wert bei der Deklarierung nicht etwa eine leere Zeichenkette «» sondern null (was die Absenz eines Wertes darstellt). Mehrere Zeichenketten können mit dem Operator + verbunden werden. Java führt dabei für viele Datentypen eine implizite Typkonvertierung durch, z.B. werden Zahlen und Character zu Zeichenketten umgeformt.

```
char c = 'a';
int n = 5;
String s = «ABC» + n + c;
```

Output:
ABC5a

Eine weitere besondere Eigenschaft von Zeichenkettenobjekten in Java ist, dass diese unveränderlich (immutable) sind. Das heisst alle Operationen, die Sie an einem String in Java vornehmen, generieren immer ein neues String-Objekt. Wird ein String-Objekt nicht mehr referenziert, wird dieses durch die Funktion des Garbage-Collectors automatisch aus dem Memory entfernt.

`s = s + «tu»` → erzeugt ein neues Zeichenkettenobjekt

Da bereits die kleinste Änderung an einem Zeichenkettenobjekt ein neues Zeichenkettenobjekt zur Folge hat, gibt es für die Durchführung vieler kleiner Mutationen an Zeichenketten die sog. ***StringBuilder***-Klasse.

Aufgabe 2.1) Stringzähler

Schreiben Sie ein Programm welches alle R's innerhalb folgender Zeichenkette zählt, unabhängig davon, ob es grosse R's oder kleine r's sind:
«RhabarberbaRbarabaRbarbaRenbartbaRbierbierbaRbärbel»

Aufgabe 2.2) Lord Von Tyrel

Unser Lord «John Tyrion Jamie Aria Schnee Lannister Stark Von Tyrell» möchte ein Programm, das ihm hilft, seine Initialen auszugeben, weil er diese immer wieder vergisst oder in falscher Reihenfolge hinschreibt.

Schreiben Sie ein Java-Programm welches Lord Von Tyrel erlaubt seinen gesamten Namen in Form von Initialen auszugeben. Schauen Sie jedoch, dass dabei nur ein String-Objekt verwendet wird. Nutzen Sie dafür ein StringBuilder-Objekt.

3 Die Date & Time Klassen

Die Java Date und die Java Time Klassen gehören zum java.time Paket.

Mit Java 8 wurden die neuen APIs für Datum und Zeit eingeführt, welche die Unzulänglichkeiten der alten java.util.Date und java.util.Calender Klassen ersetzen sollten. Die neuen APIs beinhalten unter anderen die Klassen wie LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Period, Duration. Die beiden alten Klassen *Date* und *Calendar* waren im Fehleranfällig bei Multi-Threaded-Application, dies wurde gelöst, indem die neuen Klassen gleich wie die Stringobjekte immutable also unveränderbar gemacht wurden. Ausserdem richtete man sich bei dem neuen Date und Time Klassen aus Java 8 an die internationalen ISO-Standards für die Arbeit mit Zeit und Datum. In den alten Klassen mussten eigener Code geschrieben werden, um Komplikationen wie Zeitzonen gerecht werden zu können, die neuen handeln diese Herausforderung mit den Klassen Local und ZonedDate/Time APIs.

Hier finden Sie die offizielle Dokumentation zu den Date & Time Klassen aus Java 8.

Aufgabe 3.1) Heute in zwei Wochen

Schreiben Sie ein Java-Programm, welches Ihnen den aktuellen Zeitstempel (TimeStamp) und das Datum von heute in zwei Wochen auf der Konsole ausgibt.

Aufgabe 3.2) Sie ist rund wie die Uhr

Schreiben Sie ein Java-Programm, welches Ihnen die aktuelle Uhrzeit in allen verfügbaren Zeitzonen ausgibt.

Aufgabe 3.3) Datum als String

Schreiben Sie eine Java-Methode, welche einen String Parameter date entgegennimmt und diesen anschliessend in einen Date-Type umwandelt und zurückgibt.

4 Die System Klasse

Die Systemklasse gehört zum automatisch importierten java.lang Paket.

Die Systemklasse gibt es bereits seit der JDK Version 1.0 und ist eine der ältesten Klassen in Java. Die Systemklasse ist vollkommen statisch, kann also nicht instanziiert werden. Wichtige Funktionalitäten, welche die Systemklasse anbietet, sind unter anderen Standard Input und Output sowie Zugriff auf extern definierte Eigenschaften wie bspw. Umgebungsvariablen.

Verschaffen Sie sich einen Überblick über die System-Klasse und lösen Sie die nachfolgenden Aufgaben.

Aufgabe 4.1) Error

Lassen Sie die Fehlermeldung: «Dies ist eine Fehlermeldung!» auf der Konsole über den Standard Error Output Stream ausgeben.

Aufgabe 4.2 Systeminformationen

Schreiben Sie ein Java-Programm, welches Ihnen angibt, welches Betriebssystem verwendet wird und unter welchem Verzeichnis sich die aktive Java-Version befindet (JAVA_HOME).

```
Das verwendete OS ist: Mac OS X
Java befindet sich im Verzeichnis: /Library/Java/JavaVirtualMachines/jdk-18.jdk/Contents/Home
```

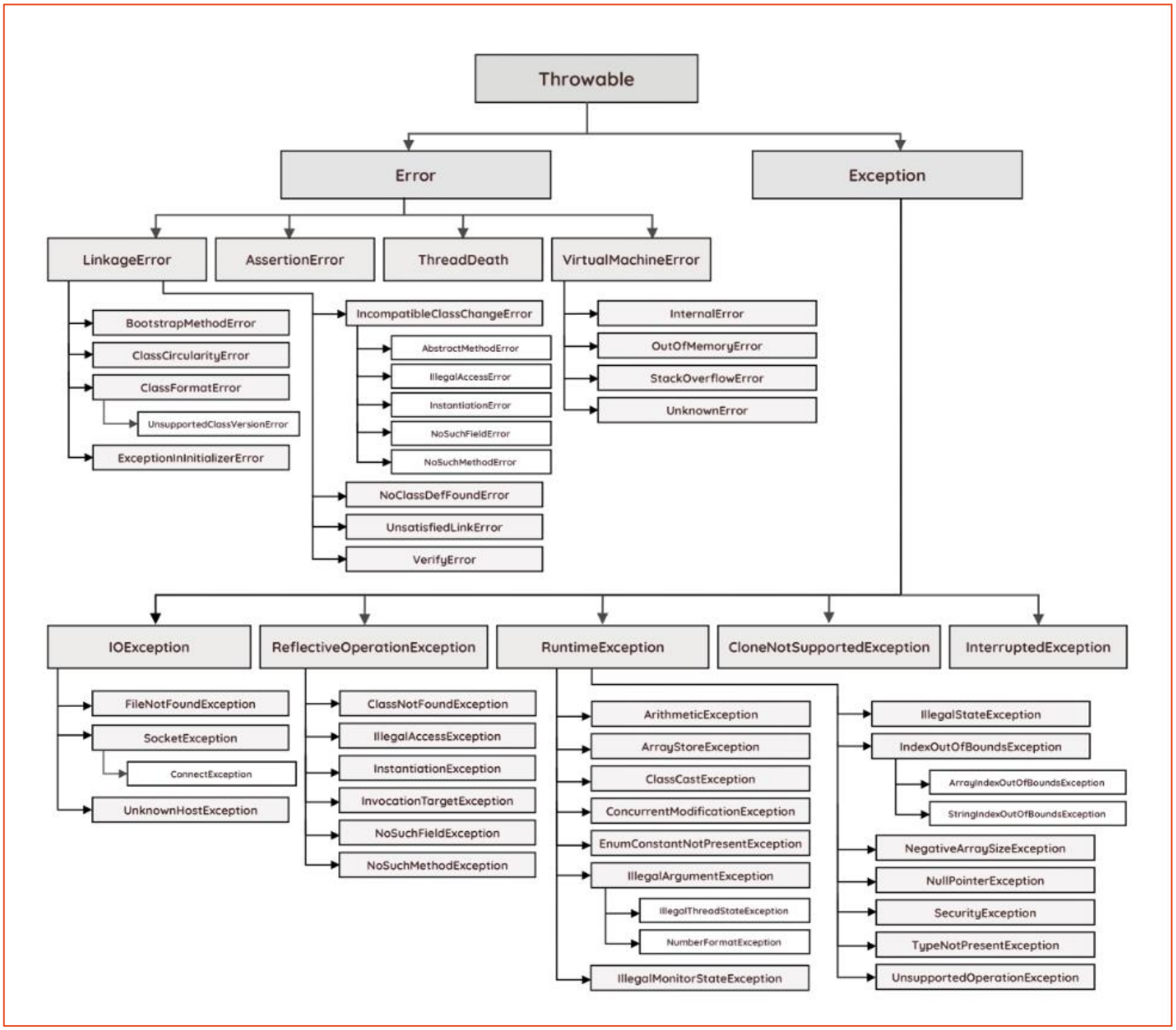
5 Exceptions

Die Throwable Klasse, von der die Error- und Exceptions-Klasse abstammen, gehört zum automatisch importierten java.lang Paket.

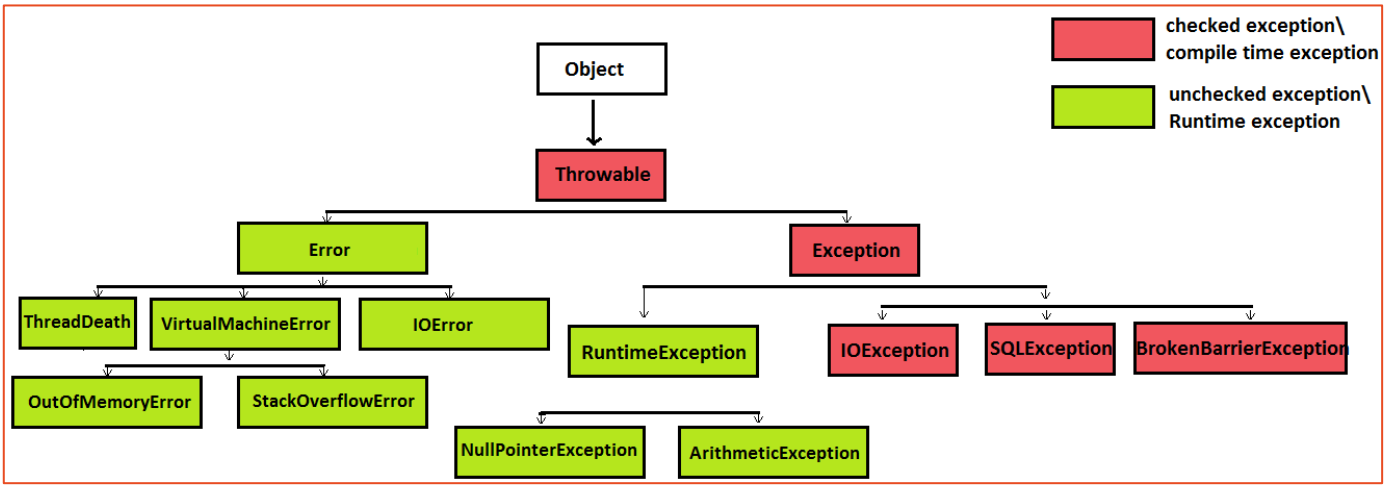
Die Exceptions-Klassen sind verantwortlich für die Fehlermeldungen in Java. Immer wenn eine Exception, also ein Fehler, auftritt wird im Hintergrund ein Exception-Objekt erzeugt und Java wechselt in einen speziellen Ausnahmezustand, in eine sogenannte Exception. Für Sie als Programmierer sind solche Exceptions eine feine Sache, sie geben Ihnen die Möglichkeit, auf ungeplante Fehler zu reagieren und das Programm in vielen Fällen sogar fortzusetzen. Sie haben verschiedene Möglichkeiten auf auftretende Exceptions zu reagieren, Sie können diese entweder selber verarbeiten z.B. mit einem ***try {}*** und ***catch {} Codeblock*** oder Sie können die Fehlerverarbeitung an eine andere Stelle, oder Methode, im Code weitergeben.

Je länger Sie mit Java arbeiten werden Sie merken, dass Sie der Compiler bei Methoden, die fehleranfälligen Code beinhalten dazu gezwungen werden, diesen Code abzusichern. Auch haben Sie selbst die Möglichkeit Methoden zu schreiben, die eine Fehlerabsicherung oder ein Exception-Handling verlangen.

In der Java Standardbibliothek sind Dutzende verschiedene Error- und Exception-Klassen definiert.



Java unterscheidet zwischen Fehlern, die im Code abgesichert werden müssen (rot), und solchen, bei denen die Absicherung optional ist (grün).



Ein Error oder eine Exception erbt seine Funktionalität und Eigenschaften von der Throwable-Klasse, die wiederum von der Java-Ursprungsklasse Object erbt. Die Throwable-Klasse implementiert wichtige Methoden, die beim Auslesen der Fehlerdaten helfen:

Methode	Bedeutung
getCause()	Gibt die Ursache des Fehlers an
getLocalizedMessage()	Liefert eine übersetzte Fehlermeldung
getMessage()	Liefert die englische Fehlermeldung
getStackTrace()	Enthält die zuvor aufgerufenen Methoden
printStackTrace()	Gibt die Stack-Trace-Daten in der Konsole aus
toString()	Liefert eine kurze Fehlerbeschreibung

Die Error-Klassen:
Die von der Error-Klasse abgeleiteten Klassen beschreiben schwerwiegende Probleme, z.B. dass der Java Interpreter nicht genug Speicherplatz zur Ausführung des Programms findet. Solche Fehler abzusichern ist meistens nicht einfach und mithilfe von Codemöglichkeiten auch gar nicht möglich, deshalb müssen diese Fehler im Code nicht abgesichert werden.

Die RuntimeException-Klassen:
Eine RuntimeException deutet in der Regel auf Programmierfehler (Flüchtigkeitsfehler) hin. Es wird davon ausgegangen, dass solche während dem Testen und dem Entstehungsprozess der Softwareentwicklung behoben werden, deshalb müssen auch diese nicht abgesichert werden.

Gewöhnliche Exception / Common Exceptions:
Alle Exceptions, die nicht von Error oder RuntimeException abgeleitet sind, gelten als gewöhnliche Exceptions. Methoden, die solche Exceptions auslösen können, müssen mit ***throws*** entsprechend markiert werden. Solche Methoden müssen mit ***try {}*** und ***catch {}*** abgesichert werden oder die übergeordnete Methode muss ebenfalls mit ***throws XxxException*** markiert werden.

try {}, catch {} und finally {}
Mit ***try*** leiten Sie einen Block ein, der Java-Anweisungen enthält, die eventuell einen Fehler auslösen können. Eine Reihe von ***catch***-Blöcken enthält Code, in dem Sie auf Fehler reagieren. Wenn es mehrere ***catch***-Blöcke gibt, zwingt der Java-Compiler Sie dazu, zuerst die speziellen und erst danach allgemeine Exception-Klassen anzugeben.

Der ***finally***-Block enthält schliesslich Code, der immer ausgeführt wird, unabhängig davon, ob vorher ein Fehler aufgetreten ist.

Schauen Sie sich folgenden Code an, der durch einen ***try*** und ***catch*** Block abgesichert wurde:

```
public class Main {

    public static void main(String[] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        } finally {
            System.out.println("The 'try catch' is finished.");
        }
    }
}
```

Überlegen Sie sich was für ein Output auf der Konsole ausgegeben wird.

Die korrekte Ausgabe wäre:

Something went wrong.
The 'try catch' is finished.

Die Weitergabe von Exceptions:
Sie müssen gewöhnliche Exceptions entweder durch ***try-catch*** absichern oder die übergeordnete Methode mit ***throws XxxException*** kennzeichnen.

Im folgenden Beispiel ruft ***main()*** die Methode ***m1()*** auf, und ***m1()*** ruft dann ***m2()*** auf. Dort tritt wegen einer schlampig programmierten Schleife eine ***ArrayIndexOutOfBoundsException*** auf.

```java
package ch.csbe.modul404;

public class Main {

    public static void main(String[] args) {
        try {
            m1();
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    public static void m1() {
        m2();
    }

    public static void m2() {
        int[] ar = {1, 2, 3};
        int sum = 0;
        for (int i = 1; i<= 3; i++) {
            sum += ar[i];
        }
    }
}
```

Selbst Exceptions werfen

Vielleicht erinnern Sie sich an die WageCalculator-Klasse die wir zu Beginn des Moduls gemeinsam programmiert haben. Hier werfen wir bei der Eingabe eines unzulässigen Wertes die RuntimeException ***IllegalArgumentException*** und geben eine eigens definierte Fehlermeldung mit.

Mit dem Schlüsselwort ***throw*** können Sie selbst eine Exception auslösen (werfen):

```java
package ch.csbe.modul404;

public class WageCalculator {
    private int baseSalary;
    private int extraHours;
    private int balanceRate = 30;

    public void setBaseSalary(int baseSalary) {
        if (baseSalary <= 0)
            throw new IllegalArgumentException("BaseSalary cannot be 0 or
less.");
        this.baseSalary = baseSalary;
    }

    public void setExtraHours(int extraHours) {
        if (extraHours < 0)
            throw new IllegalArgumentException("ExtraHours cannot be less
than 0.");
        this.extraHours = extraHours;
    }

    public int getBaseSalary() {
        return baseSalary;
    }

    public int getExtraHours() {
        return extraHours;
    }

    public int calculateWage() {
        return baseSalary + (extraHours * balanceRate);
    }
}
```

Bei nicht abgesichertem Code, wie oben dargestellt, wird damit die aktuelle Methode verlassen und die Exception an die übergeordnete Methode weitergeleitet. Wenn Sie ***throw*** innerhalb einer ***try-catch***-Konstruktion auslösen, wird der Code im ersten passenden ***catch***-Block fortgesetzt.

Nach Möglichkeit sollten Sie versuchen, eine der vielen vordefinierten Exceptions aus der Java-Standardbibliothek zu verwenden. Bei Bedarf können Sie für Ihr Programm aber auch unkompliziert eine eigene Exception-Klasse definieren. Hierfür müssen Sie jedoch das dritte wichtige Grundkonzept der objektorientierten Programmierung kennen, nämlich die **Vererbung**.

Das Thema Vererbung wird im Modul 404 leider nicht in die Tiefe angeschaut, das Schlüsselwort ist jedoch ***extends*** und bedeutet so viel wie erweitert

Aufgabe 5.1) Werfen Sie eine Exception

Schreiben Sie ein Java-Programm, welches die Eingabe eines Vornamens und eines Nachnamens erwartet. Wenn jedoch nur ein Vorname oder ein Nachname, also nur ein Wort angegeben wird, soll eine IllegalArgumentException geworfen werden.

Aufgabe 5.2) Korrekt dividieren

Schreiben Sie ein Java-Programm, welches zwei Zahlen dividiert. Die erste Zahl soll durch die zweite Zahl dividiert werden. Wird bei der zweiten Zahl der Wert 0 mitgegeben, soll die ArithmeticException gehandelt werden und der Benutzer die Chance auf eine valide Eingabe erhalten.

Aufgabe 5.3) Einlesen und Ausgeben

In dieser Aufgabe lesen Sie den Text aus dem Gefunden.txt File unter «Teams → Modul 404 → Kursmaterialien → 08 Java → Aufgaben → Aufgabe 5.3 → Gefunden.txt» In ihr Java-Programm ein und geben den Text anschliessend auf der Konsole aus.

Verwenden Sie zur Lösung dieser Aufgabe die **FileReader-Klasse** und sichern Sie Ihren Code mithilfe der try, catch Anweisung ab. Schreiben Sie zwei Methoden, eine welche die Inhalte des Files einliest und eine welche die gelesenen Inhalte auf der Konsole ausgibt. Kennzeichnen Sie die Methode, welche die Inhalte einliest, dass hierbei eine **IOException** auftreten könnte.