Praktikum „SEP: Java-Programmierung“ SS 2019

Exceptions

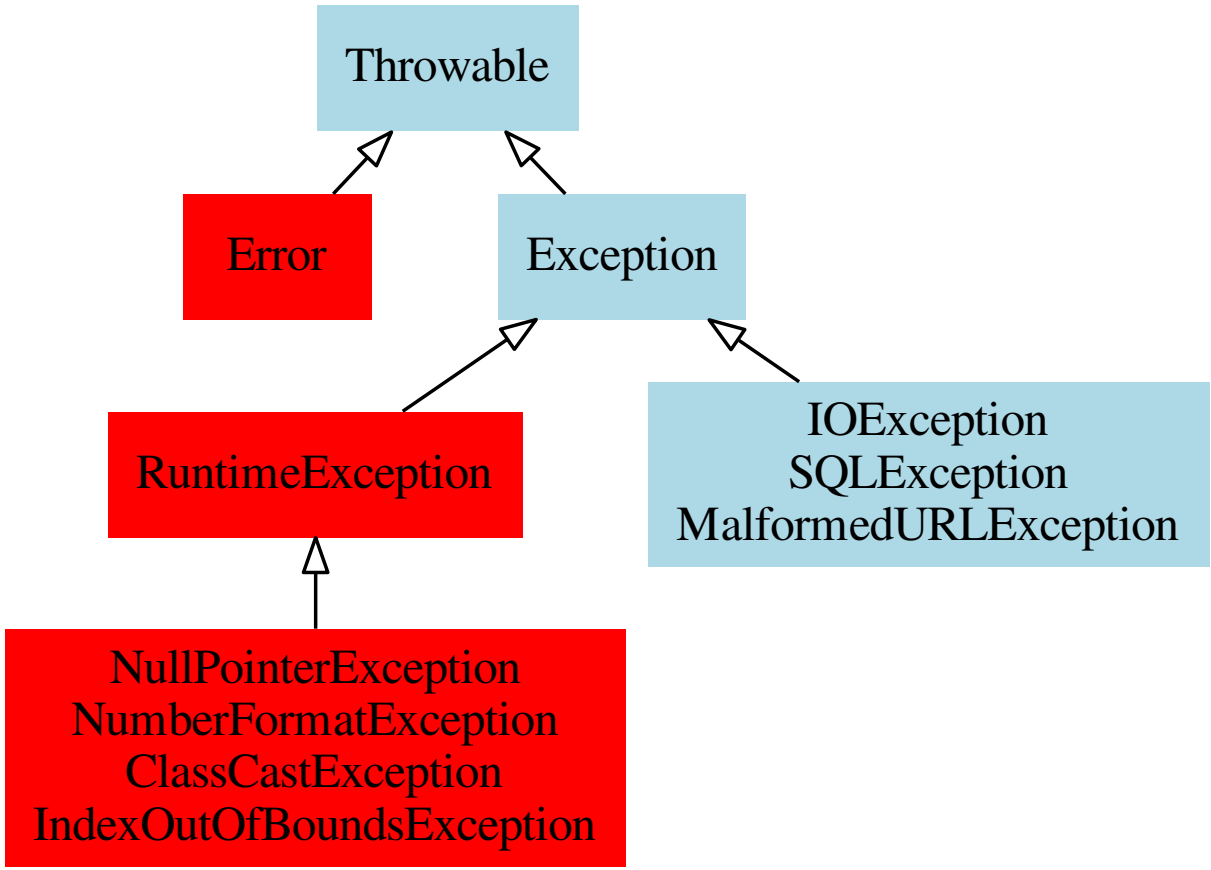
Thomas Bunk und Karlheinz Friedberger

Exception-Hierarchie in Java

Exceptions sind spezielle Java-Objekte:

- Rot: *unchecked* Exceptions
- Blau: *checked* Exceptions

Alle Exceptions beschreiben **unerwartetes** Verhalten



Exceptions: Unchecked und Checked

unchecked	**checked**
Meist keine sinnvolle Reaktion möglich	Sinnvolle Reaktion auf Ausnahme kann erwartet werden
Kann „überall“ und „jederzeit“ auftreten	Tritt unter festgelegten, wohl definierten Umständen auf
Nicht deklariert, kann immer passieren	Deklariert & erklärt
Muss nicht behandelt werden (aber darf)	Muss behandelt werden
`NullPointerException`, `NumberFormatException`, `IllegalArgumentException`, `OutOfMemoryException`,...	`IOException`, `FileNotFoundException`, `SQLException`, `TimeoutException`,...

Exceptions

Exceptions **nur** zum Signalisieren und Behandeln von Ausnahmen verwenden, und **nie** zum Lenken des normalen Kontrollfluss!

1. Bestehende Exceptions behandeln
2. Selbst Exceptions werfen/verwenden
3. Exceptions dokumentieren

Exceptions behandeln

Drei Möglichkeiten:

1. Propagieren
2. `catch` und behandeln
3. `catch` und abstraktere Exception propagieren

```java
public void handleAdd(String user, String pointsArgument) throws IOException {
  int points;
  try {
    points = Integer.parseInt(pointsArgument);
  } catch (NumberFormatException e) {
    log( m: "Invalid points argument: " + pointsArgument + " - using points = 0");
    points = 0;
  }

  addEntry(user, points); // may throw IOException (only in this example!)
}
```

Propagieren

Catch u. behandeln

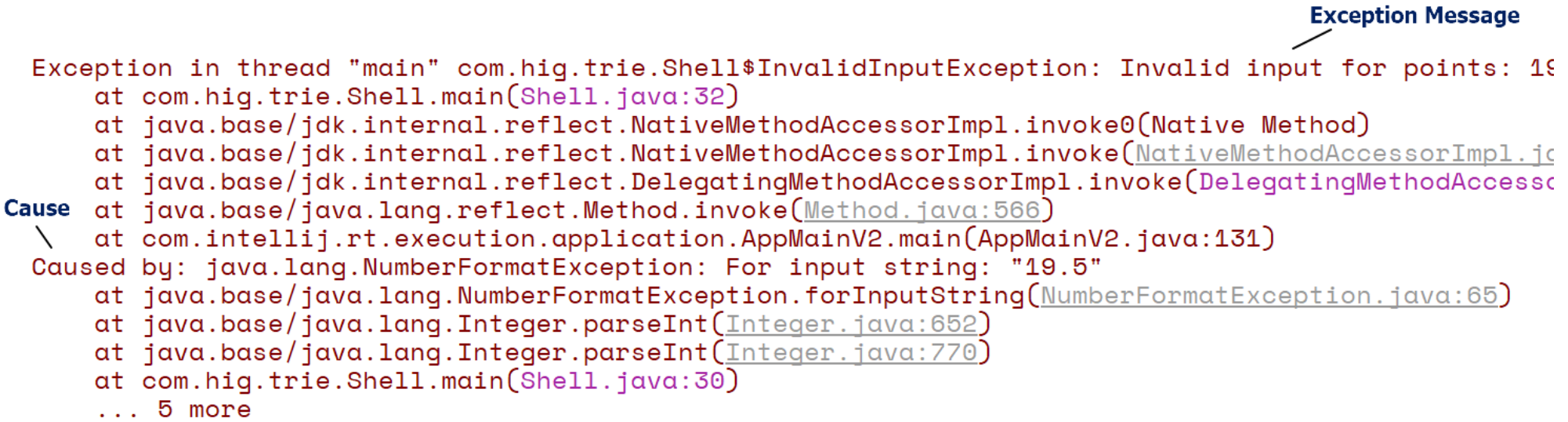
```java
  try {
    points = Integer.parseInt(pointsArgument);
  } catch (NumberFormatException e) {
    throw new InvalidInputException("Invalid input for points: " + pointsArgument, e);
  }
```

Abstraktere Exception

Exceptions propagieren

Wenn unchecked Exception aus main propagiert wird:

- Java-VM terminiert
- Stack-Trace wird auf `stderr` ausgegeben
- Exception message & causes werden gezeigt, falls existieren



Exception behandeln

Verbotene Pattern beim Behandeln von Exceptions:

```
try {
  int points = Integer.parseInt(pointsArgument);
} catch (NumberFormatException e) { // swallow
}

try {
  int points = Integer.parseInt(pointsArgument);
} catch (NumberFormatException e) { // introduce noise
  log( m: "Error! " + e.getMessage());
  throw e;
}
```

Exceptions werfen/verwenden

Erste Entscheidung: Checked oder unchecked?

- Wird erwartet, dass der Grund für die Exception sinnvoll behandelt werden kann?
  - Ungültiger User-Input, z.B. String statt Integer
    $\Rightarrow$ Fehlermeldung und weiter mit neuer Eingabe ✓
  - User-Input kann nicht gelesen werden
    $\Rightarrow$ Fehler im System oder der Programmierung X

Welche konkrete Exception werfen?

- **Niemals** nichtssagende `Exception` oder `RuntimeException`
  „Exception“ << „FileNotFoundException“
- **Niemals** einen `Error` selber werfen, außer `AssertionError`
- Existierende Exception verwenden, falls Semantik passt
- Eigene Exception-Klasse erstellen, falls nicht

Eigene Exception-Klasse

Sinnvolle Oberklasse wählen:

- checked Exception: `Exception`
- unchecked Exception: `RuntimeException`

Standard: Konstruktoren der Oberklasse implementieren

```java
public class InvalidInputException extends Exception {
  public InvalidInputException() {
  }

  public InvalidInputException(String message) {
    super(message);
  }

  public InvalidInputException(String message, Throwable cause) {
    super(message, cause);
  }

  public InvalidInputException(Throwable cause) {
    super(cause);
  }
}
```

Eigene Exception-Klasse

Erweitert: Informationen für Exception-Handling in Exception speichern

```java
public class InvalidInputException extends Exception {

  private final String failingInput;

  public InvalidInputException(String input) {
    super("Invalid user input: " + input);
    failingInput = input;
  }

  public InvalidInputException(String input, Throwable cause) {
    super("Invalid user input: " + input, cause);
    failingInput = input;
  }

  public String getFailingInput() {
    return failingInput;
  }
}
```

Häufige Exceptions

IllegalArgumentException	Gegebenes (Methoden-)Argument hat einen ungültigen Wert
IllegalStateException	Methodenaufruf im aktuellen Programmzustand ist ungültig oder unzulässig
NullPointerException	Gegebenes Object ist ungültig, weil `null`
IndexOutOfBoundsException	Ungültiger Index bei Array-/Listenzugriff
UnsupportedOperationException	Objekt unterstützt Methode nicht
AssertionError	Programminvariante wurde verletzt

Exceptions dokumentieren

Beim Werfen: Alle für Behandlung/Debugging notwendigen Informationen mitgeben

- Sinnvolle Message (gedacht für Entwickler, **nicht** User)
- Alle Variablenwerte, die Mitgrund für Ausnahme sind
  - Auf jeden Fall als Teil der Message
  - Im Idealfall als Objekt-Felder mit Getter-Methoden
  - Bsp.: Sinnvolle Infos für `IndexOutOfBoundsException`: lower bound, upper bound, used index
- Falls vorhanden: causing Exception

Exceptions dokumentieren

- In Methodensignatur: Nur checked Exceptions
- Im JavaDoc: **Alle** Exceptions, die Methode wirft
- Immer Umstände angeben, unter denen Exception geworfen wird

```java
/**
 * ...
 *
 * @throws InvalidInputException if givenPointsArgument is not a positive integer
 * @throws NullPointerException  if one of the arguments is <code>null</code>
 */
public void handleAdd(String user, String pointsArgument) throws InvalidInputException {
  if (user == null) {
    throw new NullPointerException("Given user is null");
  }
  if (pointsArgument == null) {
    throw new NullPointerException("Given points argument is null");
  }

  try {
    int points = Integer.parseInt(pointsArgument);
    addEntry(user, points);

  } catch (NumberFormatException e) {
    throw new InvalidInputException("Points param not a valid number: " + pointsArgument, e);
  }
}
```

Referenzen

- Java ist auch eine Insel: „Exceptions in try und catch“
- Java in a Nutshell: „Checked and Unchecked Exceptions“ und „Exceptions and Exception Handling“
- Effective Java, 3rd Ed.: „Exceptions“

Appendix

Exception catch

- `try-catch-finally` Blöcke

```
try {
  int points = Integer.parseInt(pointsArgument);
  doSomethingThatMayThrowIOException(user, points);

} catch (NumberFormatException e) {
  // some handling
  handle(e);

} catch (IOException e) {
  // some other handling
  handleIO(e);

} finally {
  // stuff that is always executed
  cleanUpAndCo();
}
```