CARL VON OSSIETZKY universität OLDENBURG

Sommersemester 2020

Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

CARL VON OSSIETZKY universität OLDENBURG

Exceptions und Fehlerbehandlung

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Fehler- und Ausnahmebehandlung
- Exception, try, catch, finally
- Java Exceptions

CARL VON OSSIETZKY universität OLDENBURG

Motivation

```
public class Domino {
    public void setDots(int dots) {
        this.dots = dots;
    }
}
```

```
Domino piece = new Domino();
piece.setDots(-17);
```

```
public class Domino {
    public void setDots(int dots) {
        if (dots <= 0) {
            System.out.println("Invalid number of dots! Must be > 0.");
        }
        this.dots = dots;
    }
}
```

Was passiert bei der Verwendung von grafischen Oberflächen?
Wie kann man programmatisch auf solche Fälle reagieren?

CARL VON OSSIETZKY universität OLDENBURG

Verwendung von Bibliotheken

```
class MathLib {

    public static int factorial(int n) {
        if (n < 0) {
            System.err.println("Invalid parameter!");
            return -1;
        } else if (n == 0) {
            return 1;
        } else {
            return n * MathLib.factorial(n - 1);
        }
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Verwendung von Bibliotheken

```
public class MathApp {

    public static void main(String[] args) {
        int number = 30 - 20 * 5;
        // gemeint war: (30 - 20) * 5 = 50
        // heraus kommt aber: 30 - (20 * 5) = -70

        number = MathLib.factorial(number);
        // keine Rueckmeldung von der Bibliothek
        // MathApp merkt nichts und rechnet einfach weiter

        number = number * 3;
        // ...
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Exceptions

CARL VON OSSIETZKY universität OLDENBURG

Fehlerbehandlung

- Syntaxfehler, Typfehler
  - fehlendes Semikolon, Zugriff auf nicht deklarierte Variable
  - inkompatible Typen
  - ➢Compiler
- Logische Fehler
  - ungewollte oder falsche Ergebnisse bzw. Verhalten
  - ➢Testen, Verifikation
- Laufzeitfehler
  - Division durch 0, Zugriff auf nicht-existierendes Array-Element
  - Lesen aus nicht-existierende Datei
  - ➢Exceptions

CARL VON OSSIETZKY universität OLDENBURG

Exceptions

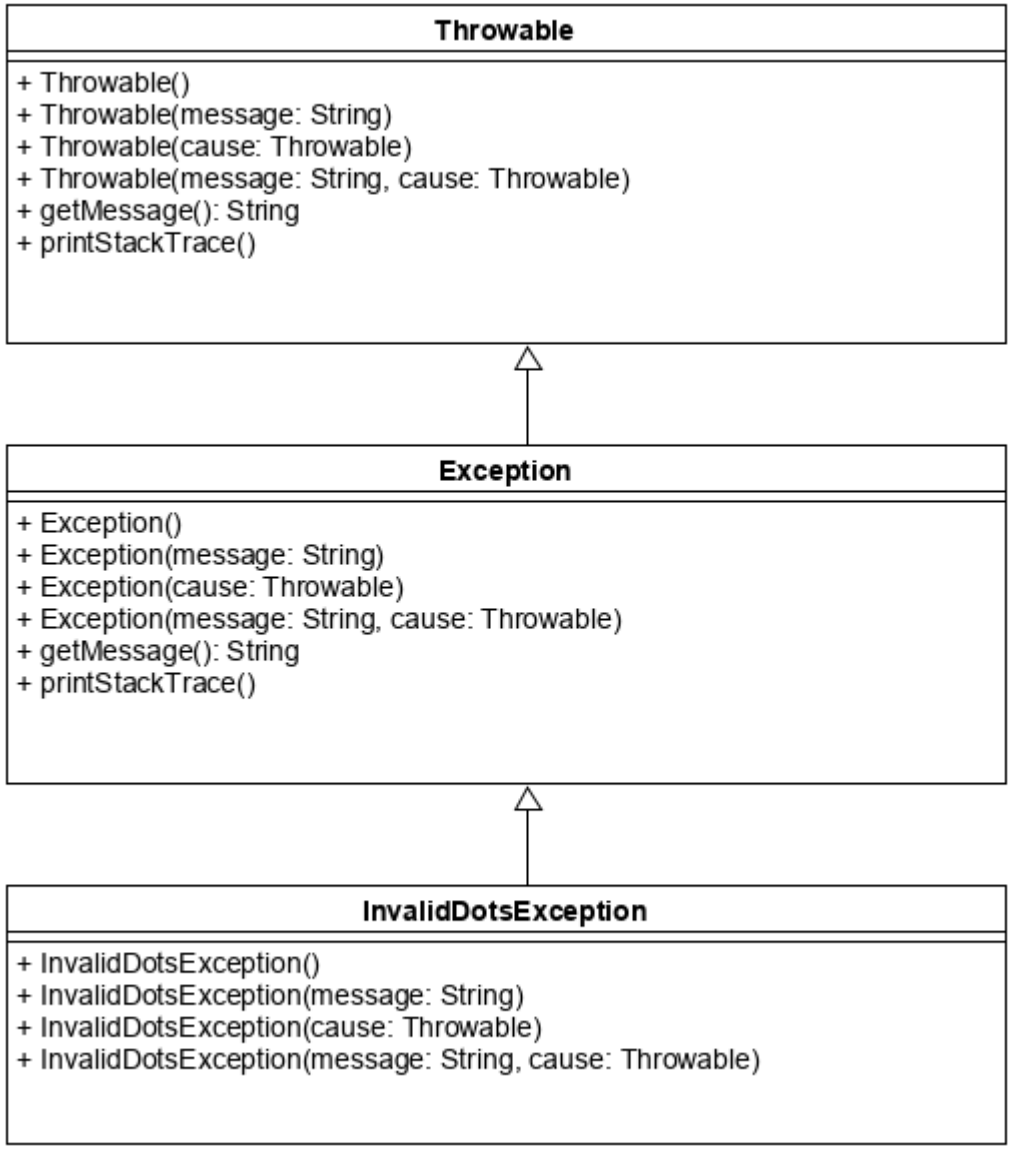
- Exceptions (Ausnahmen) steuern das Verhalten von Programmen in Ausnahmesituationen
  - unerwartete Werte von Parametern
  - unerwartetes Verhalten von Hardware
  - sonstige Situationen, die für den „normalen“ Programmablauf nicht vorgesehen sind
- Exceptions brechen die Ausführung der aktuellen Methode ab, sobald sie geworfen (ausgelöst) werden
  - dies geschieht schrittweise für die aufrufenden Methoden
    - bis die main-Methode abgebrochen wird, oder
    - bis die Exception gefangen (und behandelt) wird
- Eine Methode muss alle Exceptions deklarieren, die sie werfen kann (Ausnahme: Runtime Exceptions)

CARL VON OSSIETZKY universität OLDENBURG

Exceptions (2)

- Exceptions sind ein Mittel zur Fehlerbehandlung
  - sie werden eingesetzt, um mit Ausnahmesituationen umzugehen
  - ➢Situationen, die nicht dem „normalen“ Programmablauf entsprechen
- Dazu greifen Exceptions in den Kontrollfluss einer Anwendung ein
  - die reguläre Ausführung wird bis zur Behandlung der Exception abgebrochen
- Exceptions dürfen abgesehen von der Fehlerbehandlung aber **nicht** zur Kontrollflusssteuerung verwendet werden!
  - z.B. gezielter Abbruch einer Schleife oder einer Methode

Exceptions: Beispiel



CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Beispiel (Definition)

```
public class InvalidDotsException extends Exception {

    public InvalidDotsException() {
        super();
    }

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

    public InvalidDotsException(Throwable cause) {
        super(cause);
    }

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

}
```

Exceptions in Java sind Klassen.

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Beispiel (Werfen)

```
public class Domino {

    public void setDots(int dots) throws InvalidDotsException {
        if (dots <= 0) {
            throw new InvalidDotsException("Dots must be > 0.");
        }
        this.dots = dots;
    }

}
```

Exceptions in Java müssen instanziiert werden, bevor sie geworfen werden.
Nach dem Werfen der Exception wird die Methode sofort abgebrochen.
Es können nur Exceptions geworfen werden (**`throw`**), die auch deklariert wurden (**`throws`**).

Exceptions: Beispiel (Fangen)

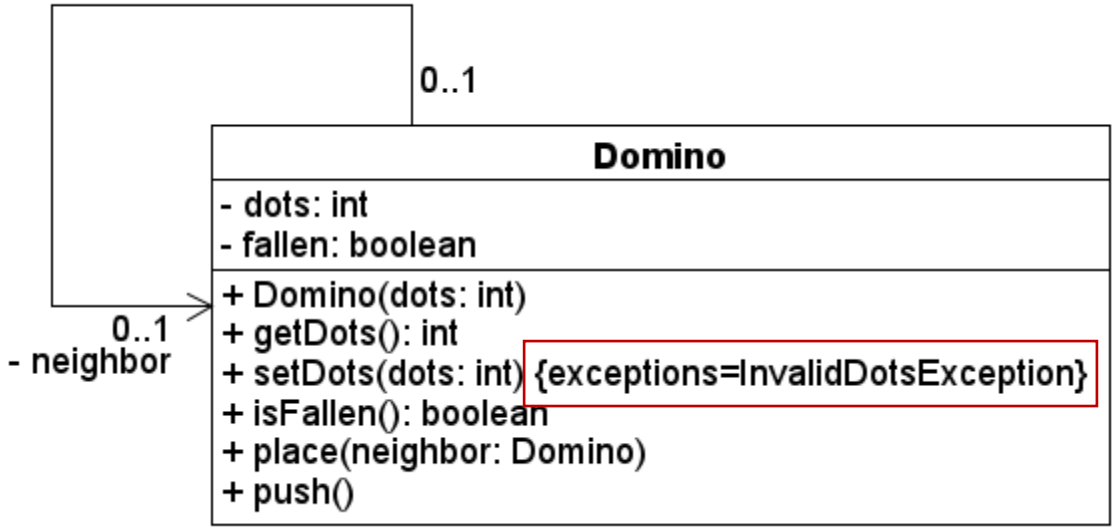
```
Domino piece = new Domino();
try {
    piece.setDots(-17);
} catch (InvalidDotsException e) {
    e.printStackTrace();
}
```

Fehlerbehandlung: Block, der die passende Exception fängt.

```
InvalidDotsException: Dots must be > 0.
at Domino.setDots(Domino.java:32)
at Domino.main(Domino.java:87)
```

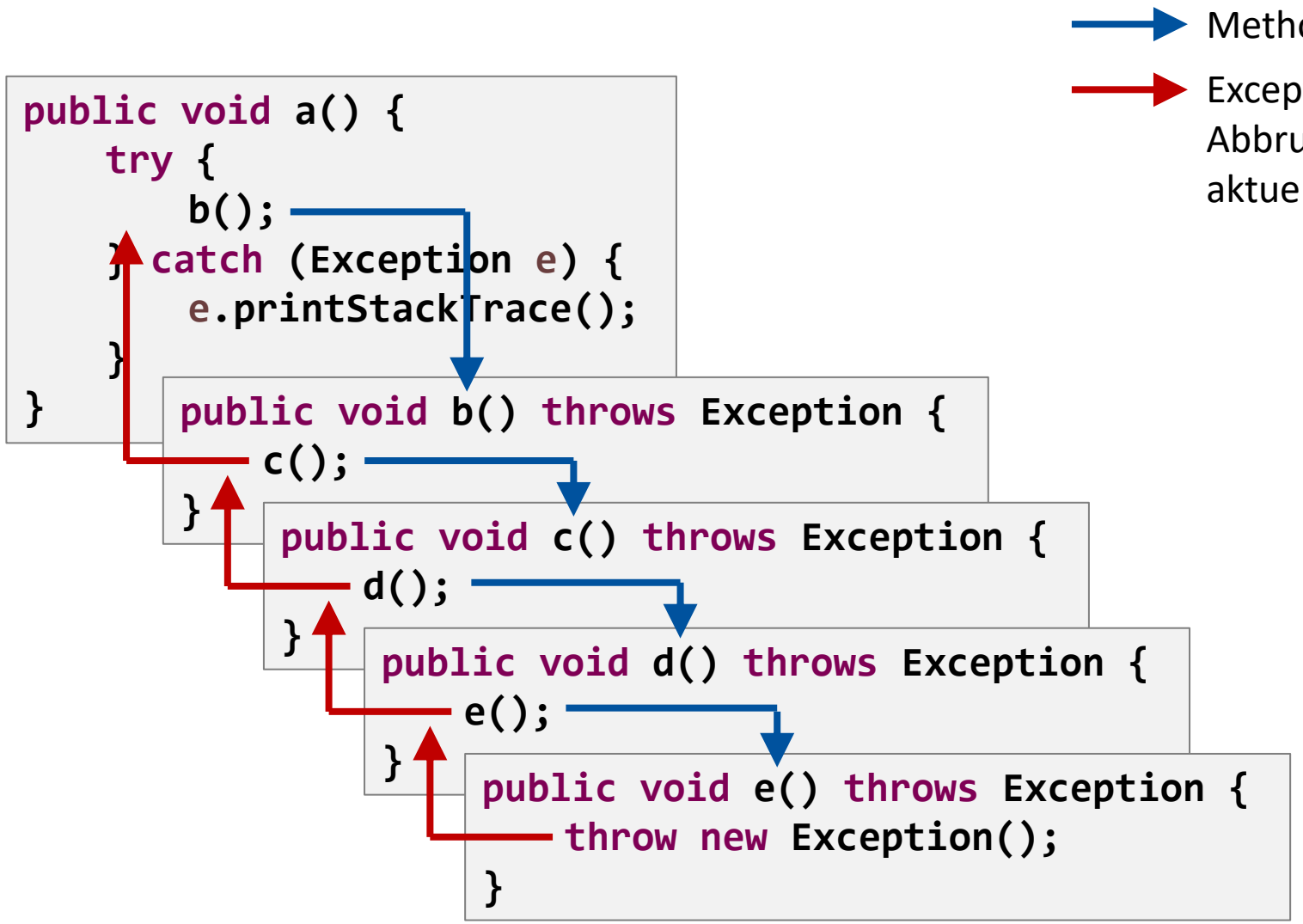
Bei fertigen Anwendungen: **`e.getMessage()`** ausgeben.
Bei grafischen Anwendungen: Fehlerdialog anzeigen!

Exceptions in der UML



Mehrere Exceptions werden durch Kommata getrennt.
Die Exception selbst wird als Klasse dargestellt.

Werfen/Fangen von Exceptions: Kontrollfluss



CARL VON OSSIETZKY universität OLDENBURG

Verwendung von Exceptions

- Bei der Klassen-/Methodendefinition
  - überlegen, was für Fehler auftreten können
  - Definition und Implementierung geeigneter Exceptions
  - Erweiterung der Methodensignatur um Angabe möglicher Exceptions
  - erzeugen und werfen von Exceptions im Fehlerfall
- Bei der Klassen-/Methodennutzung
  - beim Aufruf einer Methode ermitteln, welche Exceptions prinzipiell auftreten können
  - zunächst versuchen (**`try`**), die Methode auszuführen
  - falls keine Exception auftritt, normal weitermachen
  - sonst die Exception fangen und geeignete Fehlerbehandlung einleiten

CARL VON OSSIETZKY universität OLDENBURG

Welche Fehler können auftreten?

```
class MathLib {

    public static int factorial(int n) {
        if (n < 0) {
            ???
        } else if (n == 0) {
            return 1;
        } else {
            return n * MathLib.factorial(n - 1);
        }
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Definition geeigneter Exceptions

```
public class InvalidParameterException extends Exception {

    private int value;

    public InvalidParameterException(int value) {
        super("Invalid parameter");
        this.value = value;
    }

    public int getValue() {
        return value;
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Erweiterung der Methodensignatur

```
class MathLib {

    public static int factorial(int n)
                throws InvalidParameterException {
        if (n < 0) {
            ???
        } else if (n == 0) {
            return 1;
        } else {
            return n * MathLib.factorial(n - 1);
        }
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Werfen der Exception im Fehlerfall

```
class MathLib {

    public static int factorial(int n)
                throws InvalidParameterException {
        if (n < 0) {
            throw new InvalidParameterException(n);
        } else if (n == 0) {
            return 1;
        } else {
            return n * MathLib.factorial(n - 1);
        }
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Aufruf: Welche Exceptions möglich?

```
public class MathApp {

    public static void main(String[] args) {
        int number = 30 - 20 * 5;
        // gemeint war: (30 - 20) * 5 = 50
        // heraus kommt aber: 30 - (20 * 5) = -70

        number = MathLib.factorial(number);
```

Unhandled Exception type **InvalidParameterException**

```
        // MathApp merkt nichts und rechnet einfach weiter

        number = number * 3;
        // ...
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Versuchen, Methode auszuführen…

```
public class MathApp {

    public static void main(String[] args) {
        int number = 30 - 20 * 5;
        // gemeint war: (30 - 20) * 5 = 50
        // heraus kommt aber: 30 - (20 * 5) = -70

        try {
            number = MathLib.factorial(number);
        }

        number = number * 3;
        // ...
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

… und Exception fangen und bearbeiten

```
public class MathApp {

    public static void main(String[] args) {
        int number = 30 - 20 * 5;
        // gemeint war: (30 - 20) * 5 = 50
        // heraus kommt aber: 30 - (20 * 5) = -70

        try {
            number = MathLib.factorial(number);
        } catch (InvalidParameterException e) {
            System.err.println(e.getMessage() + ": " + e.getValue());
            number = 0; // definierter Folgezustand
        }

        number = number * 3;
        // ...
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Beispiel

```
Domino piece = new Domino();
Scanner scanner = new Scanner(System.in);
boolean init = true;
while (init) {
    System.out.print("How many dots? ");
    int dots = scanner.nextInt();
    try {
        piece.setDots(dots);
        init = false;
    } catch (InvalidDotsException e) {
        System.out.println("The number of dots must be > 0.");
    }
}
scanner.close();
```

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Verschachteln

```
Domino piece = new Domino();
try {
    piece.setDots(-17);
} catch (InvalidDotsException e1) {
    Random rand = new Random();
    try {
        piece.setDots(rand.nextInt(12) + 1);
    } catch (InvalidDotsException e2) {
        e2.printStackTrace();
    }
}
```

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Fehlerunabhängige Ausführung

```
Domino piece = new Domino();
try {
    piece.setDots(-17);
} catch (InvalidDotsException e1) {
    …
} finally {
    // wird immer ausgefuehrt
    piece.place();
}
```

Ein **`finally`**-Block wird immer ausgeführt:

- wenn im **`try`**-Block keine Exception geworfen wird
- wenn im **`try`**-Block eine Exception geworfen und im **`catch`**-Block behandelt wird
- wenn im **`try`**-Block eine Exception geworfen und im **`catch`**-Block nicht behandelt wird
- wenn im **`catch`**-Block eine neue Exception geworfen wird

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Mehrere mögliche Fehler

```
public void setDots(int dots)
            throws NegativeDotsException, TooManyDotsException {
    // ...
}
```

```
Domino piece = new Domino();
try {
    piece.setDots(-17);
} catch (NegativeDotsException e) {
    …
} catch (TooManyDotsException e) {
    …
}
```

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Weiterwerfen

```
public static void main(String[] args) throws InvalidDotsException {
    Domino piece = new Domino();
    piece.setDots(-17);
}
```

CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Weiterwerfen (2)

```
class DominoException extends Exception {
    public DominoException(Exception cause) {
        super(cause);
    }
}
```

```
public static void main(String[] args) throws DominoException {
    Domino piece = new Domino();
    try {
        piece.setDots(-17);
    } catch (InvalidDotsException e) {
        throw new DominoException(e);
    }
}
```

Auch hier wird vor dem Werfen der **`DominoException`**
ein ggf. existierender **`finally`**-Block ausgeführt.

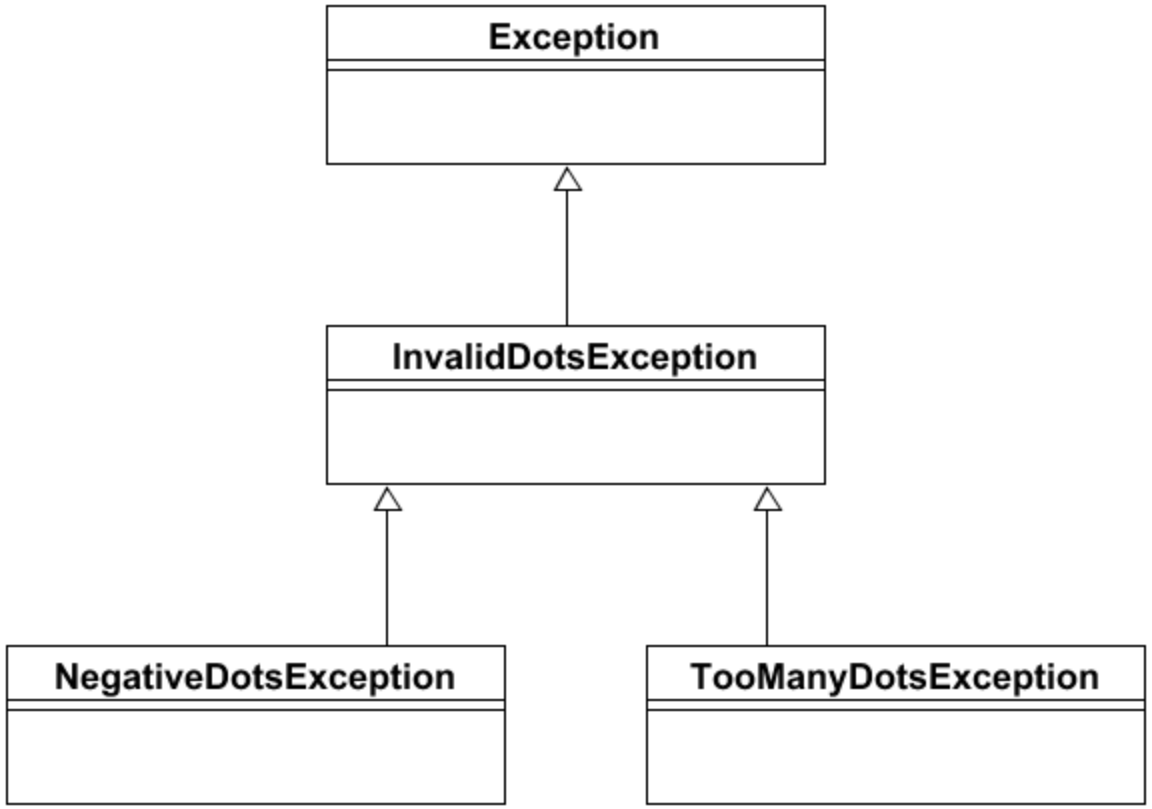
CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Weiterwerfen (3)

```
public void setDots(int dots)
            throws NegativeDotsException, TooManyDotsException {
    // ...
}
```

```
public class Domino {
    public static void main(String[] args)
                throws TooManyDotsException {
        Domino piece = new Domino();
        try {
            piece.setDots(-17);
        } catch (NegativeDotsException e) {
            …
        }
    }
}
```

Exceptions: Abgeleitete Exceptions



CARL VON OSSIETZKY universität OLDENBURG

Exceptions: Abgeleitete Exceptions

```
public void setDots(int dots) throws InvalidDotsException { … }
```

```
Domino piece = new Domino();
try {
    piece.setDots(-17);
} catch (NegativeDotsException e) {
    …
} catch (TooManyDotsException e) {
    …
} catch (InvalidDotsException e) {
    …
}
```

Schrittweise Überprüfung, ob ein Fehlerbehandlungsblock zu der Exception passt.
Es wird höchstens ein Fehlerbehandlungsblock ausgeführt.

CARL VON OSSIETZKY universität OLDENBURG

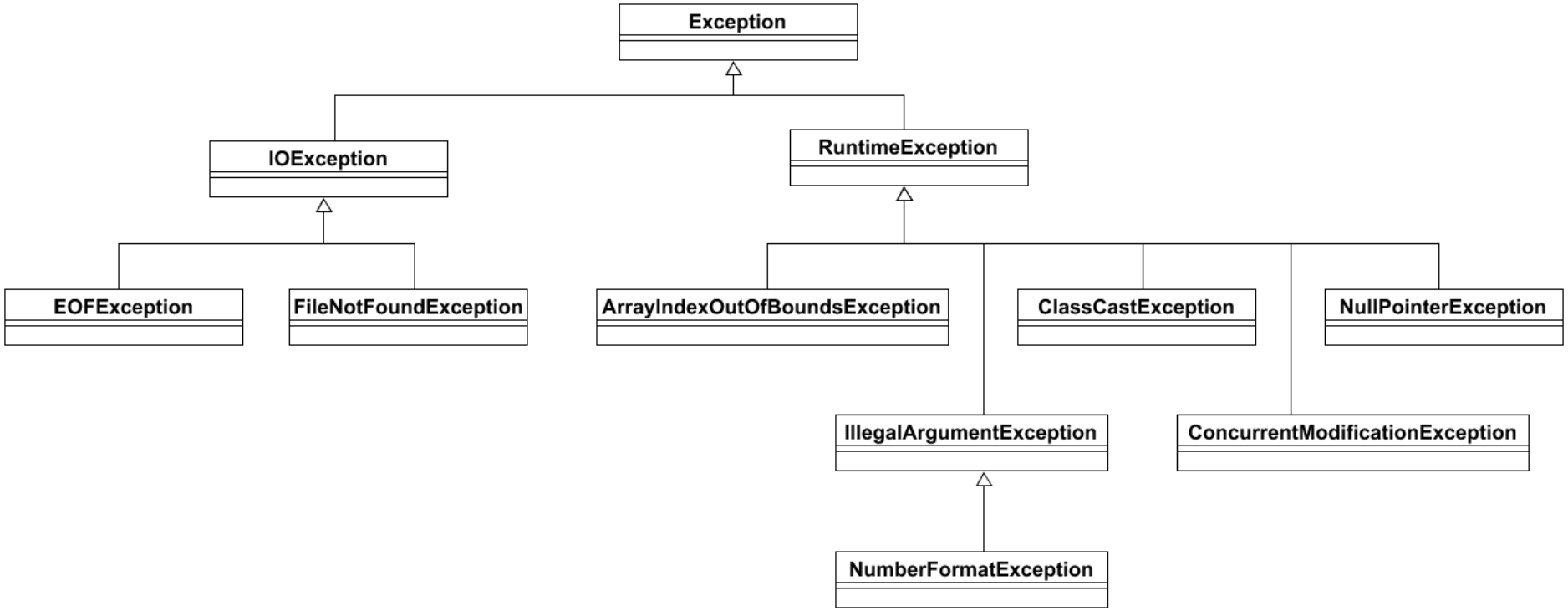
Exceptions: Überschriebene Methoden

```
class Domino {
    public void setDots(int dots) throws InvalidDotsException { … }
}
```

```
class ExtendedDomino extends Domino {
    public void setDots(int dots) throws NegativeDotsException { … }
}
```

Werden Methoden, die Exceptions werfen, in abgeleiteten Klassen überschrieben,
dann müssen die geworfenen Exceptions übereinstimmen, oder
die geworfenen Exceptions müssen von den Exceptions der
überschriebenen Methode abgeleitet sein.

Exceptions: Auswahl



RuntimeExceptions (und ihre Spezialisierungen) können überall geworfen werden,
ohne dass die Methode das deklarieren muss.
Eigene Exceptions sollten i.A. **nicht** von RuntimeException abgeleitet werden.

CARL VON OSSIETZKY universität OLDENBURG

Fallstricke: Serialisierung

```
class InvalidDotsException extends Exception {
```

The serializable class **`InvalidDotsException`** does not declare a static final **`serialVersionUID`** field of type **`long`**

```
}
```

CARL VON OSSIETZKY universität OLDENBURG

Fallstricke: Serialisierung

```
class InvalidDotsException extends Exception {

    private static final long serialVersionUID = 1L;

}
```

Exceptions sind serialisierbar (**`implements Serializable`**) → später
Serialisierbare Klassen müssen eine Serialisierungs-Versionsnummer angeben.

CARL VON OSSIETZKY universität OLDENBURG

Fallstricke (2): Gültigkeitsbereich

```
public class Domino {

    public static void main(String[] args) {
        Domino piece = new Domino();
        // ...

        try {
            Domino neighbour = piece.getNeighbour();
        } catch (InvalidNeighbourException e) {
            // ...
        }
        if (neighbour != null) {
```

neighbour cannot be resolved to a variable

```
        }
    }

    public Domino getNeighbour() throws InvalidNeighbourException {
        // ...
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Fallstricke (2): Gültigkeitsbereich

```
public class Domino {

    public static void main(String[] args) {
        Domino piece = new Domino();
        // ...

        Domino neighbour = null;
        try {
            neighbour = piece.getNeighbour();
        } catch (InvalidNeighbourException e) {
            // ...
        }
        if (neighbour != null) {
            // ...
        }
    }

    public Domino getNeighbour() throws InvalidNeighbourException {
        // ...
    }

}
```

Fallstricke (3): Zu viel Beifang

```
public class Domino {

    public static void main(String[] args) {
        Domino piece = new Domino();
        try {
            piece.setDots(5);
        } catch (Exception e) {
            System.out.println("Invalid number of dots: " + e.getMessage());
        }
    }

    public void setDots(int dots) throws InvalidDotsException {
        if (dots <= 0) {
            throw new InvalidDotsException("Dots must be > 0.");
        }
        this.dots = dots;
        if (neighbour.dots == dots) {
            System.out.println("Same dots as the neighbour!");
        }
    }

}
```

```
Invalid number of dots: null
```

Fallstricke (3a): Zu viel Beifang

- Das Fangen von **`Exception`** (oder **`Throwable`**) ist generell nicht sinnvoll, weil es Programmierfehler verdeckt und zu ungewolltem Verhalten führt.
  Außerdem kann so keine Fehlerabhängige Fehlerbehandlung durchgeführt werden.
- Ausnahme: Im Hauptprogramm, um einen unkontrollierten Programmabsturz zu vermeiden.
- Aus dem gleichen Grund dürfen auch keine Instanzen von **`Exception`** (oder **`Throwable`**) geworfen werden!
- Stattdessen: Eigene **`Exception`** mit sprechendem Namen definieren.

Fallstricke (4): Stacktrace im Production Code

- **`printStackTrace()`** ist nützlich während der Entwicklung, sollte aber vor der Auslieferung entfernt werden.
- Stattdessen:
  - sinnvolle Fehlermeldungen ausgeben
  - Debug-Informationen in einer Log-Datei speichern

CARL VON OSSIETZKY universität OLDENBURG

Fallstricke (5): Exceptions Ignorieren

```
public class MathApp {

    public static void main(String[] args) {
        int number = 30 - 20 * 5;

        try {
            number = MathLib.factorial(number);
        } catch (InvalidParameterException e) {
            // who cares?
        }

        number = number * 3;
        // ...
    }

}
```

Fallstricke (6): Exceptions Verlieren

```
Domino piece = new Domino();
try {
    piece.setDots(-5);
} finally {
    piece.setDots(-6);
}
```

```
Exception in thread "main": Dots must be > 0, was -6.
```

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Fehler- und Ausnahmebehandlung
- Exception, try, catch, finally
- Java Exceptions