Java Kurs 5/13

Florian Pix 2018

Inhalt

- Interfaces
- Exceptions

Interfaces

Interfaces

In einem Interface kann man Variablen und Methoden definieren, die andere Klassen implementieren müssen.

Man kann auf Objekte über ihre Interfaces zugreifen.

Interfaces - Beispiel

Die Post möchte Briefe, Postkarten und Pakete verschicken.

Mit dem Interface “Trackable” möchten wir diese Positionen vereinigen.

Uns ist egal wie ein einzelner Brief seine Position bestimmt.
Es ist viel wichtiger, dass es diese kommunizieren kann durch die Methoden des Interface.

Interfaces - Beispiel

- Interfaces beinhalten sog. Methodensignaturen, d.h. die Definition ohne Implementierung

```
public interface Trackable{
    public int getStatus (int identifier);
    public Position getPosition (int identifier);
}
public interface Buyable {
    public float tax = 1.19f;
    public float getPrice () ;
}
```

Interfaces - Beispiel

- Eine Klasse kann mehrerer Interfaces implementieren

```
public class Postcard implements Buyable , Trackable {
	public Position position ;
	private int identifier ;
	private float priceWithoutVAT ;

	public float getPrice () {
		return priceWithoutVAT * tax ;
	}
	public int getStatus ( int identifier ) {
		return this . identifier ;
	}
	public Position getPosition (int identifier ) {
		return this . position ;
	}
}
```

Interfaces - Beispiel

```
public class Main {
     public static void main ( String [] args ) {
          Postcard postcard_P = new Postcard () ;
          Trackable postcard_T = new Postcard () ;
          Buyable postcard_B = new Postcard () ;

          //postcard T can access Trackable.
          postcard_T . getStatus (1234) ;
          //postcard B can access Buyable.
          postcard_B . getPrice () ;
          //postcard P can access both interfaces.
          postcard_P . getStatus (1234) ;
          postcard_P . getPrice () ;
     }
}
```

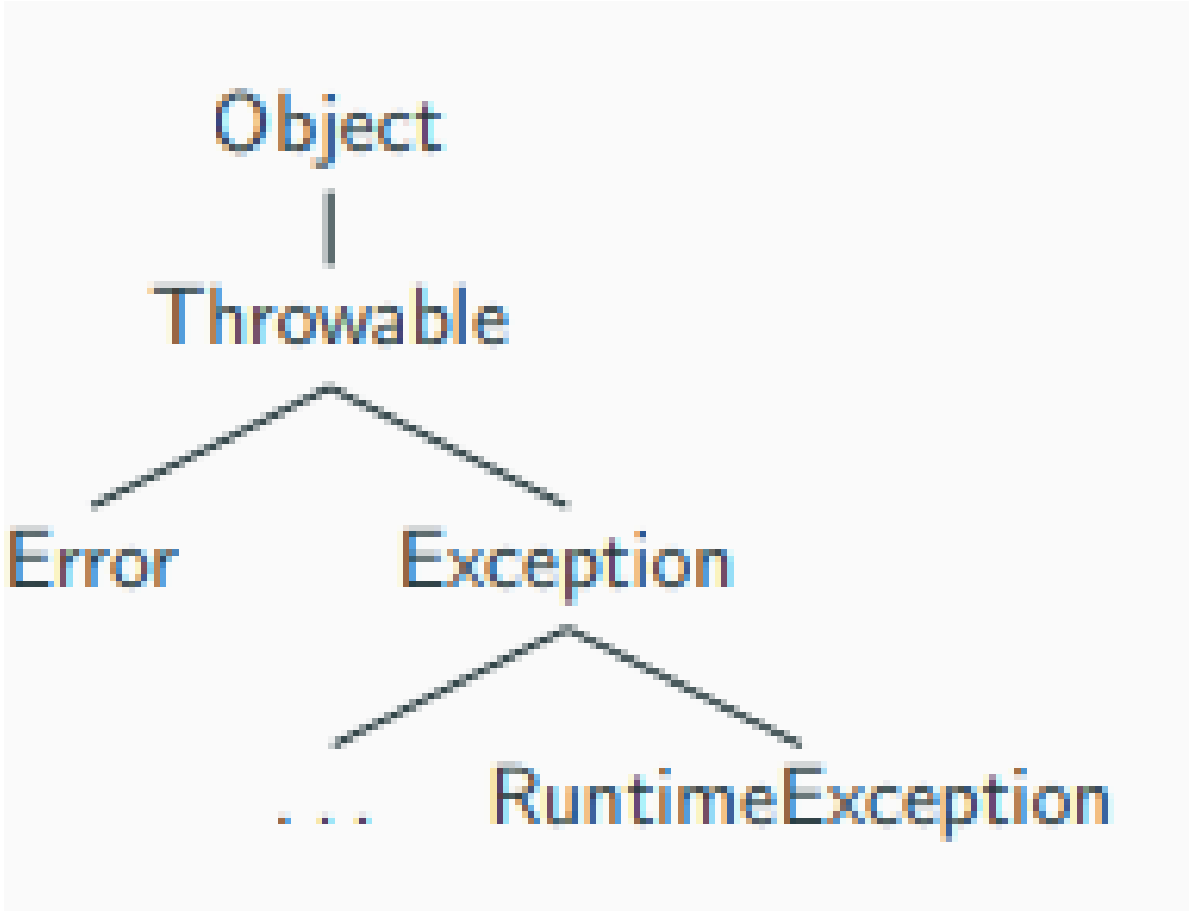
Exceptions

Exceptions

- viele Dinge können schiefgehen
- wir müssen irgendwie mit den Fehlern umgehen
- Java bietet von Haus aus “exception handling” an

Exception Hierarchie

- jede Exception ist Subklasse von Throwable
- Errors werden als ernste Fehler behandelt (z.B. : Virtual Machine Error)



Checked Exceptions

- Jede Exception außer RuntimeException und deren Subklassen sind sog. checked exceptions.
- Das bedeuted das sie gehandelt oder denotiert werden müssen.
- Die Ursachen für diese Exceptions liegen meist außerhalb eures Programms.

Unchecked Exceptions

- Unchecked Exceptions müssen nicht gehandelt oder denotiert werden, aber können es.
- Errors sind ebenfalls unchecked

Beispiel

```
1: public class Calc {
2:     public static void main ( String [] args ) {
3:         int a = 7 / 0; // will cause an ArithmeticException
4:         System . out . println (a) ;
5:     }
6: }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Calc.main(Calc.java:3)
```

Try und Catch

```
public class Calc {
    public static void main ( String [] args ) {
        try {
            int a = 7 / 0;
        } catch ( ArithmeticException e) {
            System . out . println (" Division by zero .");
        }
    }
}
```

Der catch-block, auch exception handler genannt, wird aufgrufen wenn die spezifizierte Exception im try-Block auftaucht. Man kann auch mehrere Exceptions handeln.

Stack Trace

```
public class Calc {
    public static void main ( String [] args ) {
        try {
            int a = 7 / 0;
        } catch ( ArithmeticException e) {
            System . out . println (" Division by zero .");
            e.printStackTrace();
        }
    }
}
```

Der stack trace zeigt euch die Reihenfolge der Methoden an, die zu der Exception führten.

Exceptions weitergeben

```
public class Throw {
    public static int divide ( int divident , int divisor ) throws ArithmeticException {
        return divident / divisor ;
    }
}
```

Die Methode divide(...) gibt die exception weiter an die Methode die sie aufrief.

Demo Exceptions weitergeben

```
Stacktrace:
Division by zero .
java.lang.ArithmeticException: / by zero
at PropagateExceptions1.divide(PropagateExceptions1.java:3)
at PropagateExceptions1.main(PropagateExceptions1.java:9)
```

Eigene Exception

```
public class DivisionByZeroException extends Exception {
}
```

https://stackify.com/java-custom-exceptions/