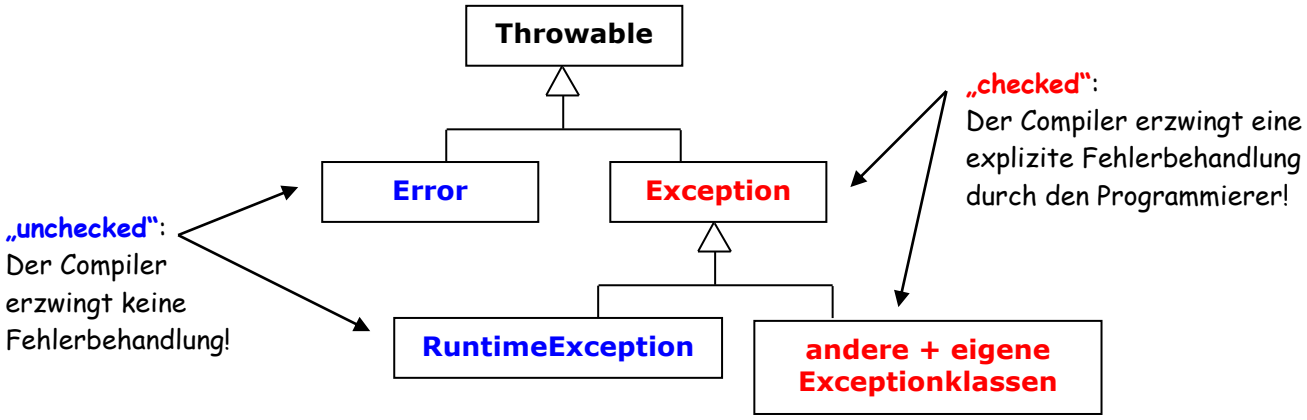
Ausnahmebehandlung mit Exceptions

Während der Ausführung eines Java-Programms können Fehler auftreten, die zum Übersetzungszeitpunkt des Java-Quelltextes nicht absehbar waren. Solche Fehler werden „Laufzeitfehler" genannt.
Tritt ein Laufzeitfehler auf, liegt eine Ausnahmesituation vor, es wird daher eine Ausnahme (Exception) ausgelöst (entweder durch die Java-Laufzeitumgebung oder durch eine entsprechende Codierung des Programmieres). Exceptions sind durch eine Klassenhierarchie in Java abgebildet:



Klasse	Bedeutung/Hinweise
`Error`	Bei schweren Systemfehlern werden von der JVM Error-Objekte ausgelöst (z.B. OutOfMemoryError, StackOverflowError,…). Hier besteht praktisch für den Programmierer keine Möglichkeit mehr den Abbruch des Programms zu verhindern.
`RuntimeException`	Logische Programmierfehler (z.B. ArithmeticException bei Division durch Null, IndexOutOfBoundsException beim Zugriff auf Arrayinhalte mit falschem Index, …) sollten vom Programmierer durch entsprechende vorsichtige Programmierung bzw. Überprüfungen vermieden werden.
`Exception`	Basisklasse für alle Fehler die zur Laufzeit auftreten können und auf die der Programmierer keinen Einfluss hat. Für diese Exceptions erzwingt der Compiler eine entsprechende Fehlerbehandlung. (siehe Folgeseite)

Tritt eine entsprechende Ausnahmesituation auf, so wird ein passendes **Exception-Objekt** erzeugt, das **Informationen über den Fehler beinhaltet**. Diese Informationen können bei der Behandlung von dem übergebenen Exception-Objekt über die folgenden Methoden abgefragt und ausgewertet werden:

Was ist passiert?	
`String getMessage();`	Abfrage der internen Fehlermeldung einer Exception.
Wie ist es passiert?	
`void printStackTrace();`	Ausgabe der Aufrufreihenfolge aller Methoden bis zu der Methode, in der die Exception ausgelöst wurde.

"checked" Exceptions

- Was muss ich machen, damit der Compiler nicht meckert?
- Was geworfen werden kann muss gefangen und behandelt werden!

```
try
{
  ...
  riskanteOperation();
  ...
}
catch(Exception e)        ← Notruf fangen
{
  ...                     ← Fehlerbehandlung
}
finally
{
  ...                     ← Aufräumarbeiten
}
```

Exceptions try-Varianten

- **Multicatch**: Exceptions unterschiedlicher Klassen behandeln:

```
try
{
  ...
}
catch(ExceptClass1 | ExceptClass2 e)
{  ...  }
```

- **Try with Resource**: Von Ressourcen, die vor dem try-Block erzeugt werden, wird automatisch am Ende des Blocks die **close()**-Methode aufgerufen:

```
try( FileReader r = new FileReader(…) )
{
  int c = r.read(); …
} // r.close() wird automatisch aufgerufen
catch(…){…}
```

Die Alternative für "verantwortungslose Feiglinge"

- Statt eine Exception zu Fangen und zu Behandeln kann sie durch Deklaration an die aufrufende Operation weitergereicht werden.

```
public void meineMethode() throws XYZException {
      ...
   RiskanterCode
      ...
}
```

- Die aufrufende Operation muss dann eine Fehlerbehandlung durchführen oder die Exception selbst weiterreichen.
- Vorsicht: wird die Exception bis zur main-Methode durchgereicht, wird die Exception nie behandelt → Absturz

Deklarieren und Behandeln von Exceptions

Beispiel für die **Definition** einer **eigenen Exception-Klasse.**
(Naturlich sollte der Name nicht „MyException", lauten, sondern im Namen gleich Aufschluss über die Art des Fehlers geben.)

```
public class MyException extends Exception {
  // ein Konstruktor genügt
  public MyException() {
    super("Fehlermeldung was passiert ist…");
  }
}
```

Hier wird ein Objekt der eigenen Exception-Klasse erzeugt und ausgelöst („**geworfen**").

```
public class MyApplication {
  …
  public void myMethod() throws MyException {
    if (…) {
      // alles OK
    } else {
      // Fehlerzustand
      throw new MyException();
    }
  }
  …
}
```

Deklaration, dass die Methode **myMethod** eine Exception vom Typ **MyException** werfen kann (nicht muss). Anhand dieser Information prüft der Compiler, ob der Anwender dieser Methode eine entsprechende Fehlerbehandlung durchführt.

Da **myMethod** mit einer möglichen **MyException** deklariert wurde, erzwingt der Java-Compiler eine Fehlerbehandlung im aufrufenden Programm. D.h. Aufrufe von Methoden, die Excepetions auslösen können, müssen in einen **try**-Block eingebettet werden.
Nach dem Ende des try-Blocks schließt sich direkt mindestens ein **catch**-Block an, in dem die Exception ausgewertet und wenn möglich sinnvoll behandelt wird.

```
public class Main{
 public static void main(String[] args){
      MyApplication app;
      app = new MyApplication();
      try{
        …
        app.myMethod();
        …
      }
      catch( MyException e ){
        Systen.out.println(e.getMessage());
        e.printStackTrace();
      }
      catch ( Exception e ) {
        …
      }
      finally {
        …
      }
  }
}
```

Es können auch **mehrere** Ausnahmebehandlungen für unterschiedliche Exceptions angefügt werden.
Aber Reihenfolge beachten: Zuerst die speziellen Exceptions, dann die Allgemeineren (der Exception-Basisklassen).

In einem (optionalen) **finally**-Block kann Quelltext angefügt werden, der in jedem Fall vor Verlassen der Methode ausgeführt wird, unabhängig, ob eine Exception ausgelöst wurde. (**Ausnahme**: Ein vorausgehendes `System.exit(0)` beendet die Anwendung sofort, ohne die Ausführung der finally-Anweisungen)

Einführungsbeispiel für Erstellung und Verwendung einer eigenen Exceptionklasse:

```
public class BierFass {

    double vorratInLiter;

    public BierFass(){
        vorratInLiter = 5;
    }

    public void zapfen(double liter) throws BierWarnung{
         if (vorratInLiter >= liter){
              // Alles Prima, Bier wird getrunken und vom Vorrat abgezogen
              vorratInLiter = vorratInLiter - liter;
              System.out.println("Es wurden " + liter + " getrunken.");
              System.out.println("Neuer Vorrat: " + vorratInLiter);
         }else{
              // Biervorrat reicht nicht mehr aus
              throw new BierWarnung();
         }
    }

    public static void main(String[] args) {

        BierFass bier = new BierFass();

        try {
            // Fussballmannschaft betritt die Kneipe und trinkt bier
            bier.zapfen(0.33);
            bier.zapfen(1);
            bier.zapfen(0.5);
            bier.zapfen(0.5);
            bier.zapfen(0.33);
            bier.zapfen(0.2);
            bier.zapfen(1);
            bier.zapfen(0.5);
            bier.zapfen(0.5);
            bier.zapfen(0.33);
            bier.zapfen(0.33);
            bier.zapfen(0.2);

        }
        // Fehler wird abgefangen
        catch (BierWarnung e) {
            e.printStackTrace();
            // Es wird neues Bier nachgekauft! (10 Liter)
            bier.vorratInLiter = bier.vorratInLiter + 10;
            System.out.println();
            System.out.println("Es wurde Bier eingekauft! Neuer Vorrat: " +
 bier.vorratInLiter + " Liter.");
        }

    }

}

public class BierWarnung extends Exception{
    public BierWarnung() {
        super("Bier ist leer! Bitte neues Kaufen!!! Rothaus schmeckt am
Besten!");
   }

}
```

In Anlehnung an: http://blog.mynotiz.de/programmieren/java-exceptions-beispiel-tutorial-278/