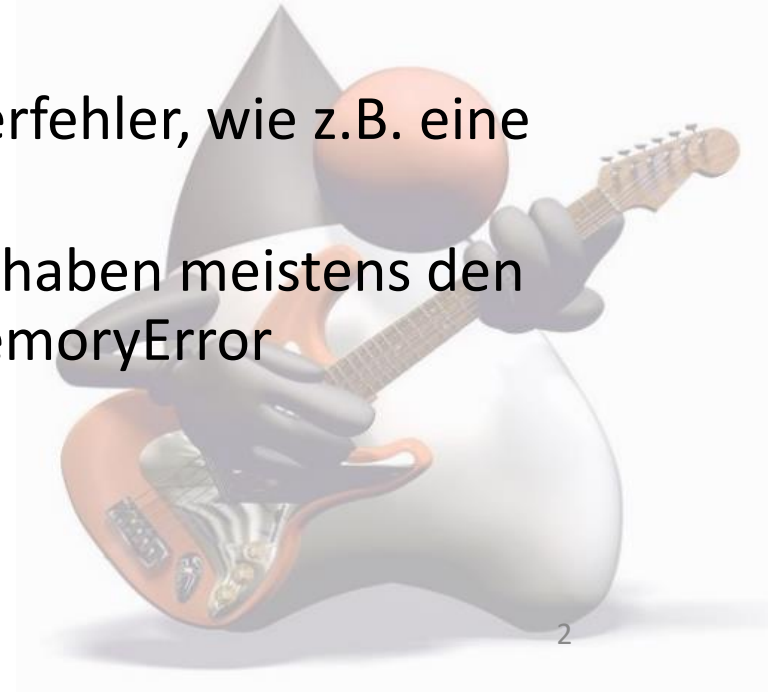
Java Fundamentals

Exception Handling

Exception Handling

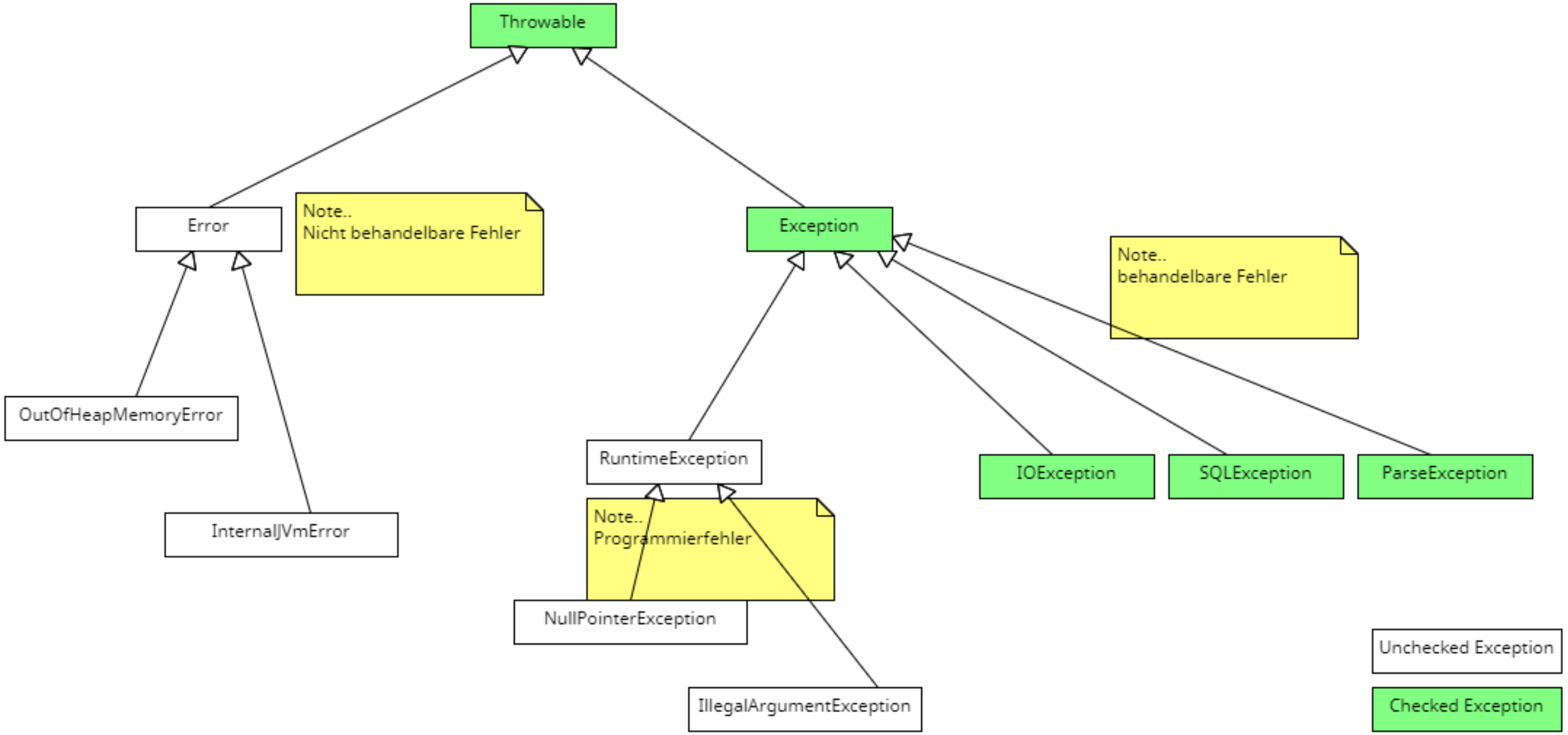
- Eine Exception tritt bei einem Fehler in einer laufenden Applikation auf.
- Es gibt 3 verschiedene Arten von Exceptions
  - Checked Exception: Müssen vom Programmierer behandelt werden. IO Exceptions
  - Unchecked Exceptions: Sind normalerweise Programmierfehler, wie z.B. eine NullPointerException oder IllegalArgumentException
  - Errors: Sind Exceptions die von der JVM kommen. Diese haben meistens den negativen Effekt das die JVM abstürzt, wie z.B. OutOfMemoryError



Unchecked Exception / Checked Exception

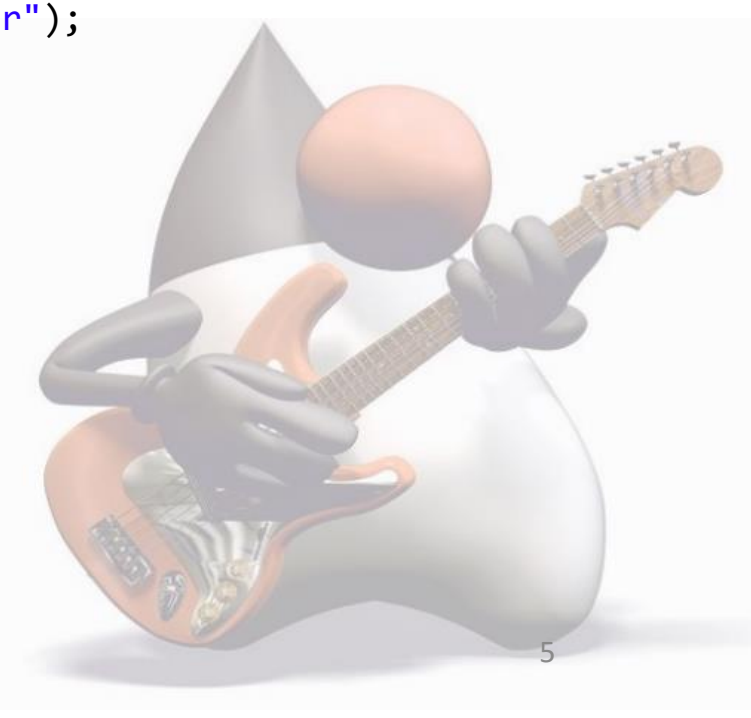
- Unchecked Exceptions: Subklassen von Runtime Exception
- Checked Exceptions: Subklassen von Exception

Exceptions Vererbungshierarchie



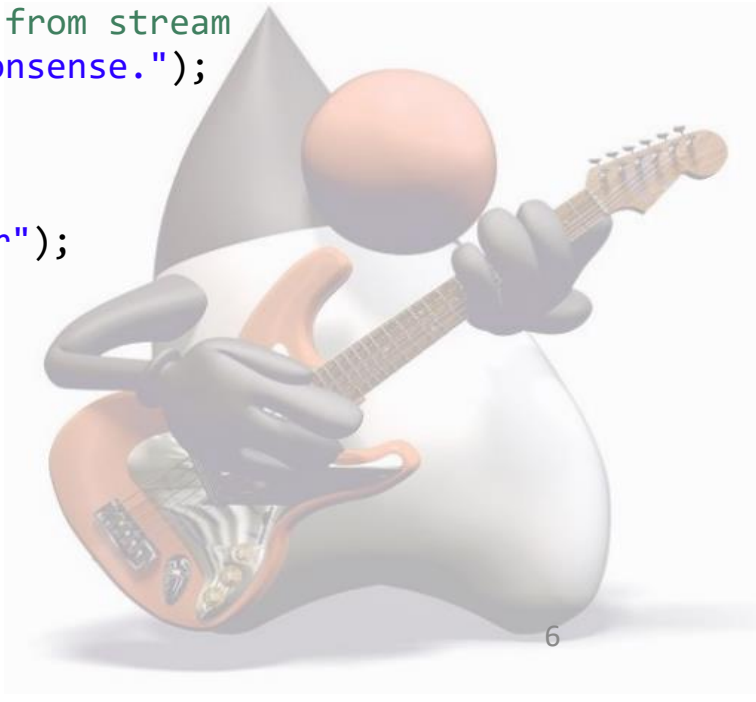
```java
public class GuessNumber {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		int guessedNumber = 0;
		int randomNumber = (int)((Math.random() * 100) + 1);
		System.out.println("Guess a number between 1 and 100");
		while (guessedNumber!=randomNumber){
			System.out.println("Your Guess:");
			guessedNumber = scanner.nextInt();
			if (guessedNumber>randomNumber){
				System.out.println("This number is too large.");
			} else if (guessedNumber<randomNumber){
				System.out.println("This number is too small.");
			}
		}
		System.out.println("Congratulations, that is the right number");
	}
}
```

```
Guess a number between 1 and 100
Your Guess:
50
This number is too large.
Your Guess:
12a
Exception in thread "main" java.util.InputMismatchException
	at java.util.Scanner.throwFor(Scanner.java:909)
	at java.util.Scanner.next(Scanner.java:1530)
	at java.util.Scanner.nextInt(Scanner.java:2160)
	at java.util.Scanner.nextInt(Scanner.java:2119)
	at at.java.GuessNumber.main(GuessNumber.java:13)
```



```java
public class GuessNumber {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		int guessedNumber = 0;
		int randomNumber = (int) ((Math.random() * 100) + 1);
		System.out.println("Guess a number between 1 and 100");
		while (guessedNumber != randomNumber) {
			System.out.println("Your Guess:");
			try {
				guessedNumber = scanner.nextInt();
				if (guessedNumber > randomNumber) {
					System.out.println("This number is too large.");
				} else if (guessedNumber < randomNumber) {
					System.out.println("This number is too small.");
				}
			} catch (InputMismatchException ex) {
				scanner.next(); //remove the invalid data from stream
				System.out.println("Please stop talking nonsense.");
			}

		}
		System.out.println("Congratulations, that is the right number");
	}
}
```

```
Guess a number between 1 and 100
Your Guess:
blablaIHateJava
Please stop talking nonsense.
Your Guess:
```



Unterschiedliche Exceptions

- Es können mehrere Exceptions von verschiedenen Typen in einem Try-Catch Block gefangen werden

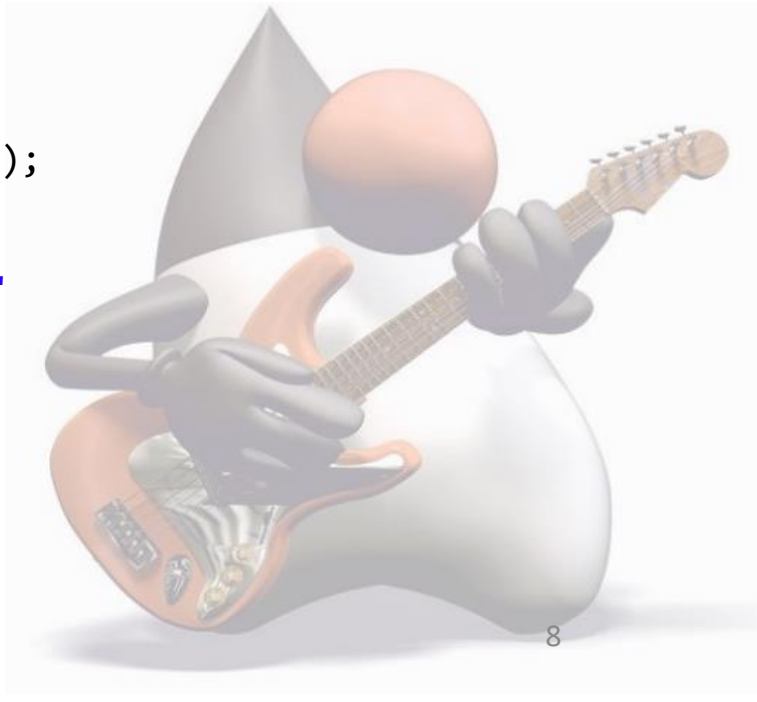
```
public static void main(String[] args) {
        try {
                // do code in which Exceptions can occur
        } catch (IOException ex) {
                // do error handling for IOExceptions
        } catch (IllegalArgumentException ex){
                // do error handling for IllegalArgumentExceptions
        }
}
```

Checked Exceptions

- Darum ***MUSS*** sich der Programmierer kümmern, sonst lässt sich das Programm nicht kompilieren

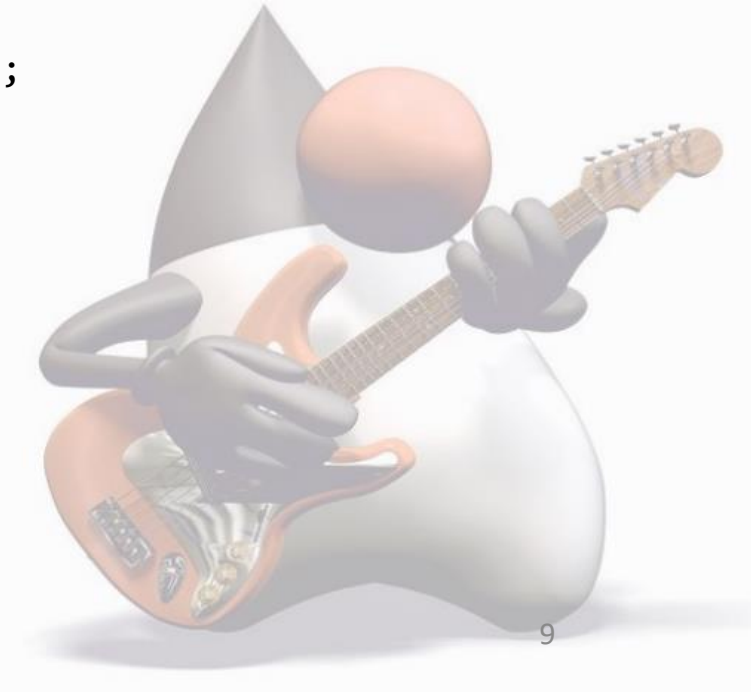
```
public class JustAClass {

        public static void main(String[] args) {
                JustAClass jac = new JustAClass();
                jac.parseDate("01.01.2011");
        }

        public Date parseDate(String dateString) {
                SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
                Date result = null;
                        result = format.parse(dateString);
                        System.out.println("The date was not in format + "
                                                + format.toPattern());
                return result;
        }
}
```

```
Unhandled exception type ParseException
```



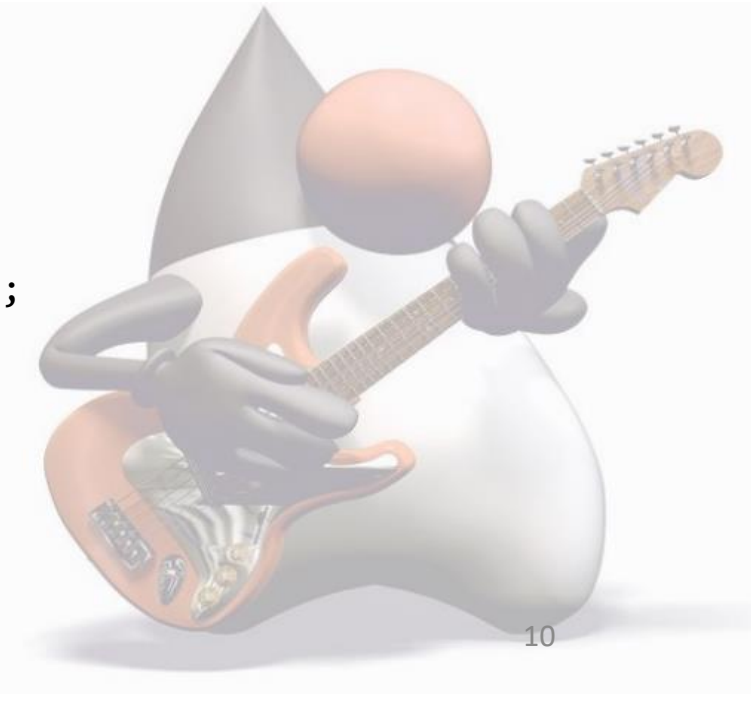
Exception Handling

```
public class JustAClass {

        public static void main(String[] args) {
                JustAClass jac = new JustAClass();
                jac.parseDate("01.01.2011");

        }

        public Date parseDate(String dateString) {
                SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
                Date result = null;
                try {
                        result = format.parse(dateString);
                } catch (ParseException ex) {
                        System.out.println("The date was not in format + "
                                        + format.toPattern());
                }

                return result;
        }
}
```



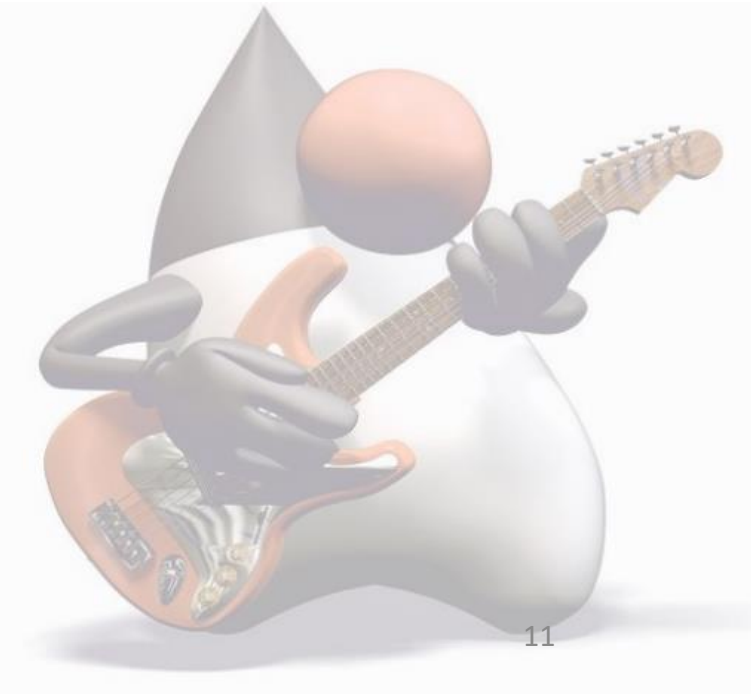
Wir könnten die Methode auch in der aufrufenden Methode fangen. Dazu muss nur deklarieren dass parseDate() eine ParseException wirft.

```
public class JustAClass {

        public static void main(String[] args) {
                JustAClass jac = new JustAClass();
                try {
                        jac.parseDate("01.01.2011");
                } catch (ParseException ex){
                        System.out
                        .println("The date was not in the right format");
                }

        }

        public Date parseDate(String dateString) throws ParseException {
                SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
                Date result = null;
                result = format.parse(dateString);
                return result;
        }
}
```



Handle-Or-Declare Rule

- Wenn eine Methode aufgerufen wird die eine Exception wirft muss man eines von zwei Dingen tun.
  - mittels Try-Catch Block die Exception fangen und behandeln
  - throws in Methoden Signatur, um zu deklarieren, dass die Methode diese Exception wirft



Woher weiß man welche Exceptions bei einer Methode geworfen werden?

- Checked Exceptions: Schreibt der Compiler vor
- Unchecked Exceptions: In der API Dokumentation sind die möglichen Exceptions, die auftreten können, gelistet

parse

```
public Date parse(String source)
           throws ParseException
```

Parses text from the beginning of the given string to produce a date. The method may not use the entire text of the given string.

See the `parse(String, ParsePosition)` method for more information on date parsing.

Parameters:

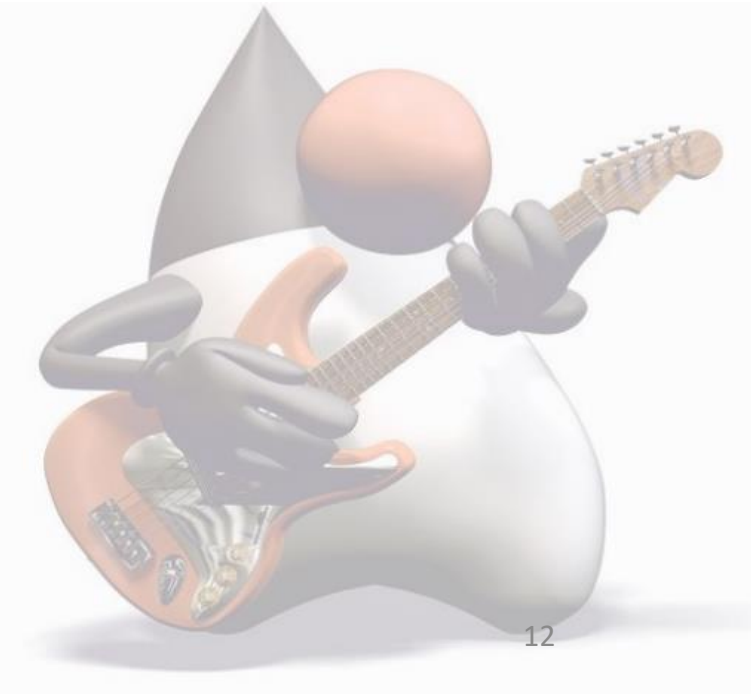
`source` - A `String` whose beginning should be parsed.

Returns:

A `Date` parsed from the string.

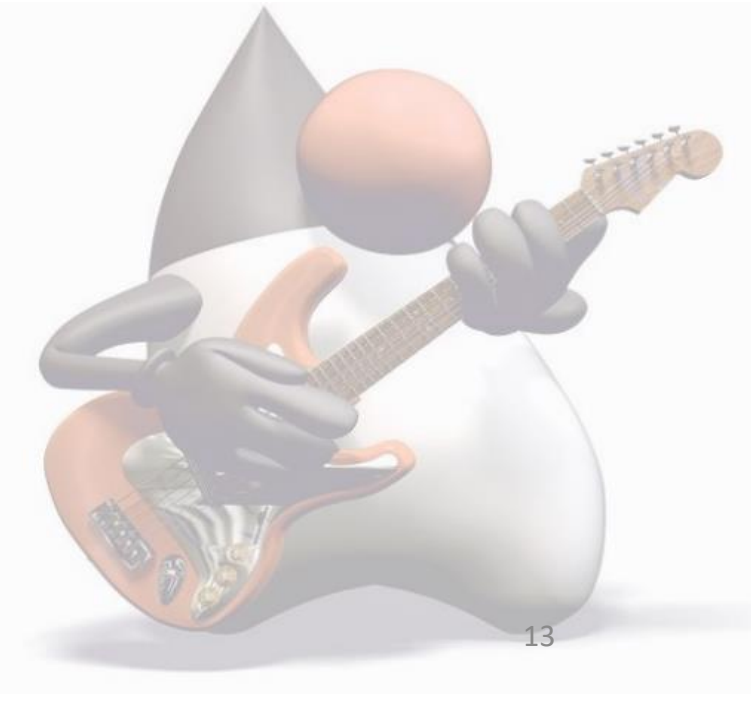
Throws:

`ParseException` - if the beginning of the specified string cannot be parsed.



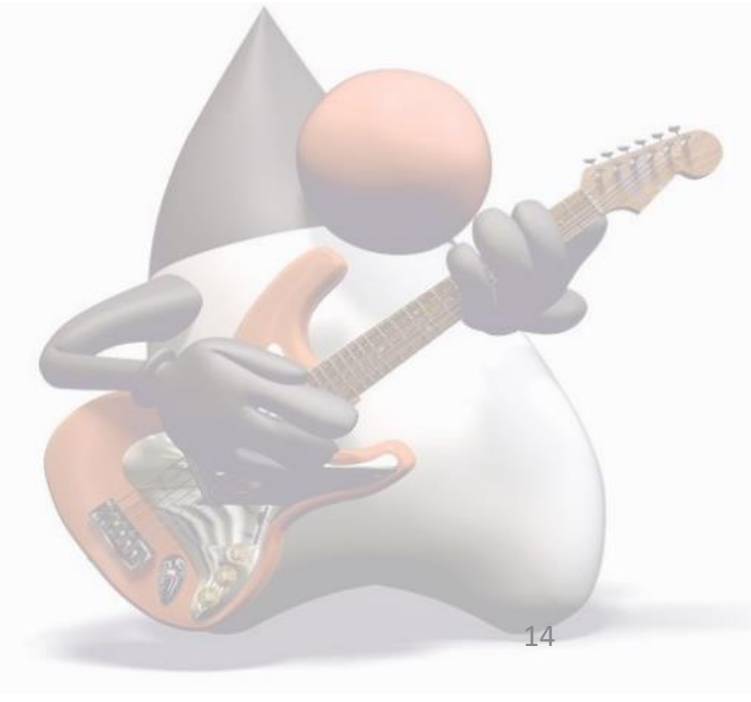
Finally Block wird immer ausgeführt

```
public class JustAClass {
        public static void main(String[] args)  {
                BufferedReader reader = null;
                try {
                        reader = new BufferedReader(new FileReader("C:\\Windows\\win.ini"));
                        String line = "";
                        while (null != (line = reader.readLine())) {
                                System.out.println(line);
                        }
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        if (reader!=null){
                                try {
                                        reader.close();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }
}
```



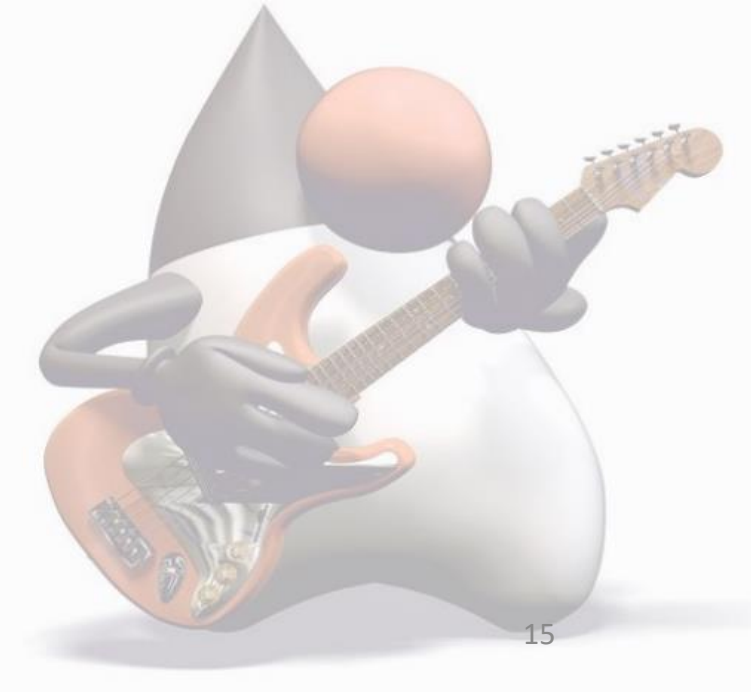
Eigene Exception Klassen erstellen

```
public class OverdraftException extends Exception{
        BigDecimal amount;

        public OverdraftException(String message, BigDecimal amount){
                super(message);
                this.amount = amount;
        }

        public BigDecimal getAmount() {
                return amount;
        }
}
```



Eigene Exception werfen

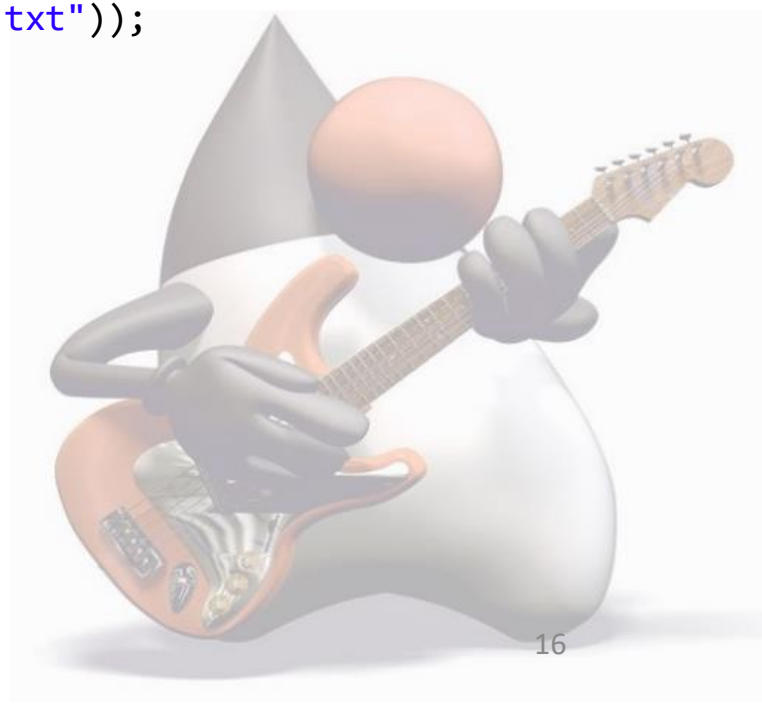
```
public boolean withdraw(double amount) throws OverdraftException {
        if (balance - amount < overdraftLimit) {
                throw new OverdraftException("Overdraft limit reached.",
                                        -overdraftLimit + balance - amount);
        }
        balance -= amount;
        return true;
}
```



Autoclosables – try with resources

- Dateien, Netzwerkverbindungen, Datenbankverbindungen müssen nach dem Zugriff mittels *close()* geschlossen werden. Das führt oft zu schlecht wartbarem Code

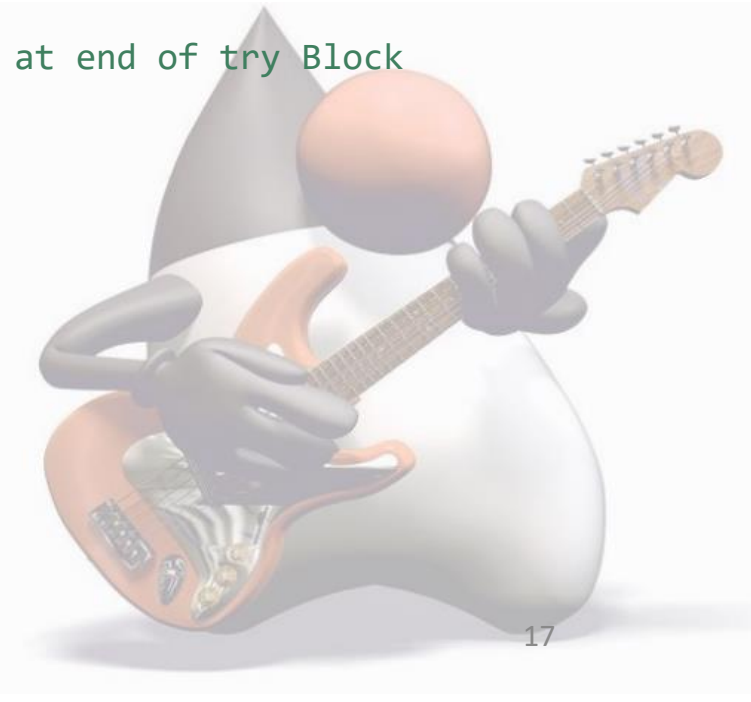
```
public class FileReader {
        public static void main(String[] args) {
                BufferedReader br = null;
                try {
                        br = Files.newBufferedReader(Paths.get("myFile.txt"));
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                br.close();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }
}
```



Autoclosables – try with resources

- Seit Java 7 steht das Try-With-Resources-Statement zur Verfügung, bei dem offene Ressourcen automatisch geschlossen werden

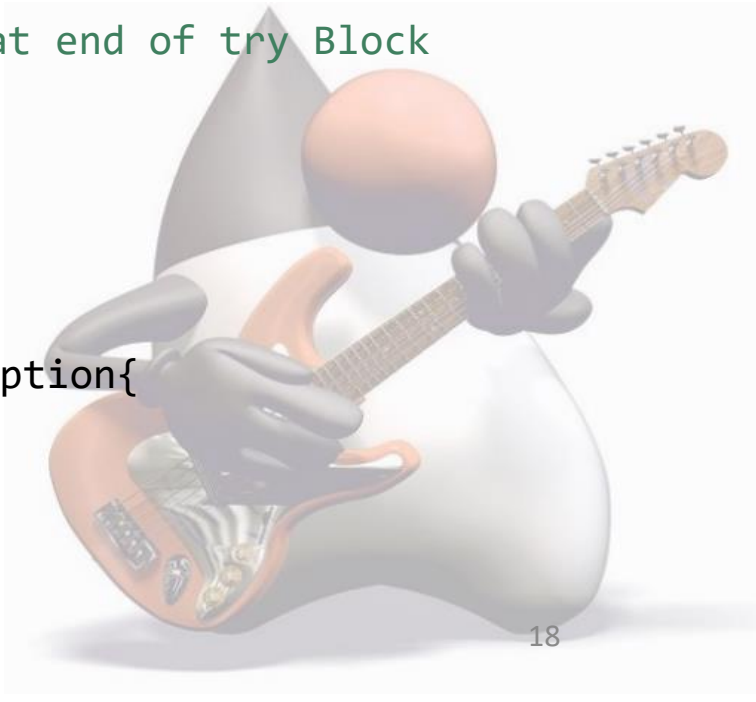
```
public class FileReader {
        public static void main(String[] args) {
                try (BufferedReader br = Files.newBufferedReader(Paths.get("myFile.txt"))){

                        //...
                        // Buffered Reader will be closed automatically at end of try Block
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```



Multicatch

- Seit Java 7 steht auch das Multi-Cach-Statement zur Verfügung, bei dem mehrere, nicht in einer Vererbungslinie stehende Exceptions behandelt werden können

```
public class FileReader {
        public static void main(String[] args) {
                try (BufferedReader br = Files.newBufferedReader(Paths.get("myFile.txt"))){
                        doDatabaseStuff();
                        // Buffered Reader will be closed automatically at end of try Block
                } catch (IOException | SQLException e) {
                        e.printStackTrace();
                }
        }

        private static void doDatabaseStuff() throws SQLException, IOException{
                // Access the database
        }
}
```



Übungen

- Baue ein sinnvolles Exception Handling in Übung 6 ein.
- Implementiere eine OverdarftException als Subclass von Exception
- Wirf die OverdraftException wenn die Methode Account.withdraw nicht erfolgreich abläuft.