Karlsruher Institut für Technologie

Vorlesung Softwaretechnik I
Übung 1

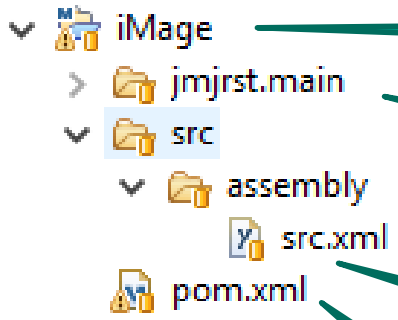
SWT I – Sommersemester 2019
Walter F. Tichy, Sebastian Weigelt, Tobias Hey

IPD Tichy, Fakultät für Informatik

Altsoftware vorbereiten

AUFGABE 1

Aufgabe 1
Das iMage-Projekt



Stammverzeichnis des Projekts iMage

Projektverzeichnis für das Modul jmjrst.main → Details auf späterer Folie

Konfigurationen aus lez-test-dummy (Steuert die Erzeugung der ZIP-Datei)

pom.xml von iMage
- Packaging: pom
- Parent-pom.xml: SWT1-Übungsparent (GroupID: edu.kit.ipd.swt1.ss2019, ArtifactID: uebungsparent, Version: 0.0.1-SNAPSHOT)
- Eintrag für das SWT1-Maven-Depot

Aufgabe 1
JMJRST für Maven vorbereiten

- Maven-Verzeichnisstruktur anlegen
  - `src/main/java`
  - `src/main/resources`

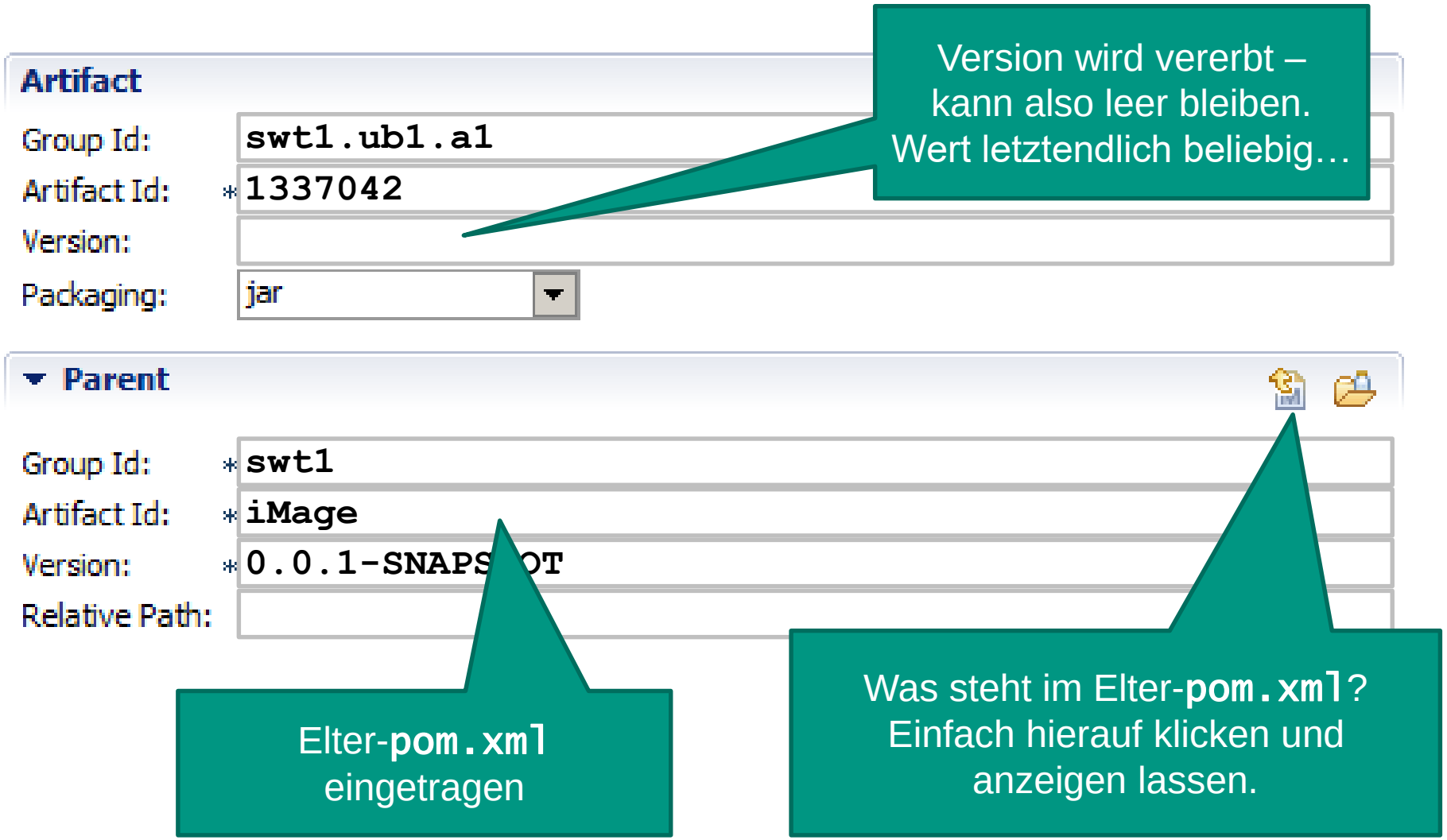
Dazwischen – in sinnvollen Abständen (bspw. Teilaufgaben) – ins Git einbuchen!

- JMJRST-Dateien entsprechend einsortieren
  - Quelltexte aus `src/` nach `src/main/java`
  - Ressourcendateien aus `src/` nach `src/main/resources`
    - `src/icons`
      → `src/main/resources/icons`
    - `src/org/jis/*.properties`
      → `src/main/resources/org/jis/*.properties`
  - `templates`
    → bleibt dort, wo es ist; siehe `GallerieDialog`, Zeilen 379 ff.
- Patch einspielen, um `ImageFormatException` zu entfernen
- Geforderte Dateien aus `lez-test-dummy` kopieren
- pom.xml anpassen

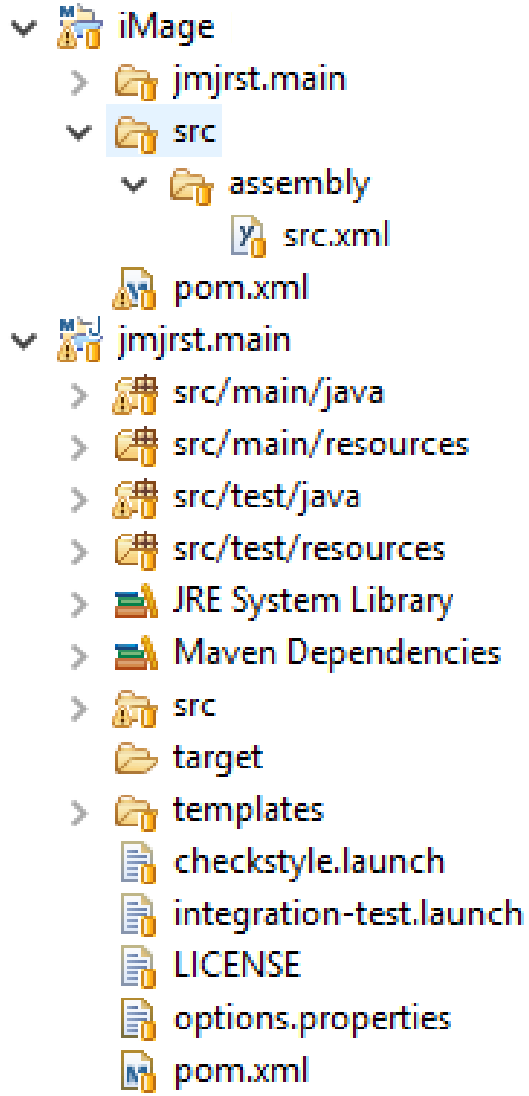
Aufgabe 1
Ein Auszug aus dem Patch

```
---
a/iMage/jmjrst.main/src/main/java/org/jis/generator/Consumer.java
+++
b/iMage/jmjrst.main/src/main/java/org/jis/generator/Consumer.java
@@ -59,10 +57,6 @@
          {
            process(obj);
          }
-         catch (ImageFormatException e)
-         {
-           e.printStackTrace();
-         }
          catch (IOException e)
          {
            e.printStackTrace();
```

Aufgabe 1
Die pom.xml von iMage als Elter-`pom.xml`



Aufgabe 1
Das iMage-Projekt mit jmjrst.main-Modul



Aufgabe 1
JMJRST – maven-jar-plugin

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-jar-plugin</artifactId>
 <configuration>
  <archive>
   <manifest>
    <mainClass>org.jis.Main</mainClass>
    <addClasspath>true</addClasspath>
    <addDefaultImplementationEntries>
     true
    </addDefaultImplementationEntries>
    <addDefaultSpecificationEntries>
     true
    </addDefaultSpecificationEntries>
   </manifest>
  </archive>
 </configuration>
</plugin>
```

Damit man das JAR einfach starten kann.

Fügt Versionsinformationen zum JAR-Manifest hinzu

Aufgabe 1
JMJRST – maven-jar-plugin

Nachschauen: Das JAR ist ein ZIP!
Manifest-Datei liegt unter
`JAR/META-INF/MANIFEST.MF`

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven
Built-By: weigelt
Build-Jdk: 11.0.1
Main-Class: org.jis.Main
Specification-Title: 1337042
Specification-Version: 0.0.1-SNAPSHOT
Specification-Vendor: KIT - IPD Tichy
Implementation-Title: 1337042
Implementation-Version: 0.0.1-SNAPSHOT
Implementation-Vendor-Id: swt1.ub1.a1
Implementation-Vendor: KIT - IPD Tichy
```

Damit man das JAR einfach starten kann.

Versionsinformationen ☺

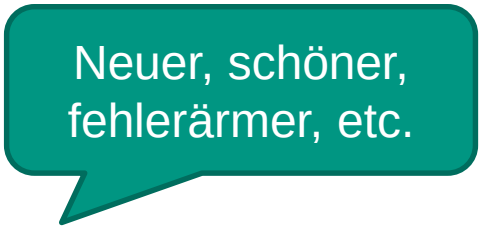
Aufgabe 1
JMJRST – Checkstyle

- Von CheckStyle gemeldete Fehler (TOP-10):

Checkstyle violation type	Marker count
'X' should be on the previous line.	467
Missing a Javadoc comment.	195
Name 'X' must match pattern 'X'.	152
'X' construct must use '{}'s.	151
‘X’ should be on the same line.	124
Line is longer than X characters.	86
Catching 'X' is not allowed.	57
Variable 'X' must be private and have accessor methods.	37
'X' is not followed by whitespace.	17
Executable statement count is X (max allowed is X).	12

Randnotiz zu JMJRST

- Voraussetzung für das SWT1-Projekt
  - Echtes Programm, kein synthetisches Lehrbuch-Beispiel
  - Quelloffen
  - Einigermaßen sinnvoll
  - Lauffähig
  - Von Umfang und Komplexität geeignet
  - Java
  - Keine Bibliothek, sondern eine Anwendung mit grafischer Oberfläche
  - Sinnvoll erweiterbar
  - Parallelisierbare Algorithmen



- Falls Sie mit diesen Anforderungen ein (besseres!) Projekt finden
  - Sagen Sie Bescheid
  - Sichern sie sich den Dank der folgenden SWT1-Generationen
  - Werden Sie Teil des SWT1-Teams ☺

Modultests

AUFGABE 2

Aufgabe 2
JUnit – Allgemeines

- Abhängigkeit in pom.xml eintragen
- Verzeichnisse anlegen
  - `src/test/java`
  - `src/test/resources`

GroupID: `junit`
ArtifactID: `junit`
Version: leer (Elter-`pom.xml`)
Scope: `test`

- `beforeClass()`-Methode
  - `@BeforeClass`-Annotation, damit Verzeichnis für Testbilder angelegt wird
- `setUp()`-Methode
  - `@Before`-Annotation, damit der `Generator` immer frisch erzeugt wird
  - Klassenvariable für den `Generator` und das `Testbild`, damit alle Tests darauf zugreifen können
- `tearDown()`-Methode
  - `@After`-Annotation zum Speichern der durch die Testmethoden erstellten Bilder

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2c)

```
private Generator generator;
private BufferedImage testImage;
private String imageName;
private static final String IMAGE_FILE = "/image.jpg";

@Before
public void setUp() {
  generator = new Generator(null, 0);
  final URL imageResource =
    this.getClass().getResource(IMAGE_FILE);
  imageName = extractFileNameWithoutExtension(
    new File(imageResource.getFile()));
  testImage = ImageIO.read(imageResource);   // oder Code
                                             // aus rotate()
}
```

Generator und Image werden als Klassen-variablen angelegt

Für Aufgabe 2 h)

Aufgabe 2
JUnit – Bildname (Aufgabe 2h)

```
private String extractFileNameWithoutExtension(
  File file) {
  String fileName = file.getName();
  if (fileName.indexOf(".") > 0) {
    return fileName.substring(0,
      fileName.lastIndexOf("."));
  } else {
    return fileName;
  }
}
```

Aufgabe 2
JUnit – Testverzeichnis (Aufgabe 2h)

```
@BeforeClass
public static void beforeClass() {
  if (TEST_DIR.exists()) {
    for (File f : TEST_DIR.listFiles()) {
      f.delete();
    }
  } else {
    TEST_DIR.mkdirs();
  }
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2d)

```
@Test
public void testRotateImage0 () {
  rotatedImage = generator.rotateImage(testImage, 0);
  assertTrue(imageEquals(testImage, rotatedImage));
}

@Test
public void testRotateImageNull () {
  rotatedImage = generator.rotateImage(null, 0);
  assertNull(rotatedImage);
}
```

Aufgabe 2
JUnit – Testfallbeispiel (`imageEquals`)

```
public static boolean imageEquals(
      BufferedImage expected, BufferedImage actual) {
  if (expected == null || actual == null) {
    return false;
  }

  if (expected.getHeight() != actual.getHeight()) {
    return false;
  }

  if (expected.getWidth() != actual.getWidth()) {
    return false;
  }
  …
```

Aufgabe 2
JUnit – Testfallbeispiel (`imageEquals`)

```
public static boolean imageEquals(
      BufferedImage expected, BufferedImage actual) {
  …
  for (int i = 0; i < expected.getHeight(); i++) {
    for (int j = 0; j < expected.getWidth(); j++) {
      if (expected.getRGB(j, i) !=
        actual.getRGB(j, i)) {
          return false;
      }
    }
  }
  return true;
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2e)

```
@Test(expected = IllegalArgumentException.class)
public void testRotateImageIAE() {
      generator.rotateImage(testImage, 0.42);
}
```

Erwartet eine `IllegalArgumentException`

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2f)

```
@Test
public void testRotateImage90degrees() {
  rotatedImage = generator.rotateImage(
      testImage, Math.toRadians(90));

  assertEquals(testImage.getHeight(),
      rotatedImage.getWidth());

  assertEquals(testImage.getWidth(),
      rotatedImage.getHeight());

  …
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2f)

- Wie stellt man sicher, dass es sich um dasselbe (gedrehte) Bild handelt?
- Antwort: pixelweises Vergleichen

```
@Test
public void testRotateImage90degrees() {
  …
  for (int i = 0; i < IMAGE_HEIGHT; i++) {
    for (int j = 0; j < IMAGE_WIDTH; j++) {
      assertEquals(testImage.getRGB(j, i),
        rotatedImage.getRGB(IMAGE_HEIGHT - 1 - i, j));
    }
  }
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2f)

- Wie stellt man sicher, dass es sich um dasselbe (gedrehte) Bild handelt?
- Antwort: pixelweises Vergleichen

```
@Test
public void testRotateImage270degrees() {
  …
  for (int i = 0; i < IMAGE_HEIGHT; i++) {
    for (int j = 0; j < IMAGE_WIDTH; j++) {
      assertEquals(testImage.getRGB(j, i),
        rotatedImage.getRGB(i, IMAGE_WIDTH - 1 - j));
    }
  }
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2g)

```
@Test
public void testRotateImageM90degrees() {
  rotatedImage = generator.rotateImage(
      testImage, Math.toRadians(-90));

  assertEquals(testImage.getHeight(),
      rotatedImage.getHeight());

  assertEquals(testImage.getWidth(),
      rotatedImage.getWidth());

  …
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2g)

- Wie funktioniert der pixelweise Vergleich in diesem Fall?

```
@Test
public void testRotateImageM90degrees() {
  …
  for (int i = 0; i < IMAGE_HEIGHT; i++) {
    for (int j = 0; j < IMAGE_WIDTH; j++) {
      assertEquals(testImage.getRGB(j, i),
        rotatedImage.getRGB(i, IMAGE_WIDTH - 1 - j));
    }
  }
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2g)

- Wie funktioniert der pixelweise Vergleich in diesem Fall?

```
@Test
public void testRotateImageM270degrees() {
  …
  for (int i = 0; i < IMAGE_HEIGHT; i++) {
    for (int j = 0; j < IMAGE_WIDTH; j++) {
      assertEquals(testImage.getRGB(j, i),
        rotatedImage.getRGB(IMAGE_HEIGHT – 1 – i, j));
    }
  }
}
```

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2g)

- **`IllegalArgumentException:`** *„degree must be a mutiple of 90°!"*
- Wenig sinnvoll → Erweitern der Methode `rotateImage(…)`

```
// Generator.java
public BufferedImage rotateImage(BufferedImage image,
      double rotate) {
  …
  if (rotate < 0) {
      rotate += Math.toRadians(360);
    }
  if (rotate == Generator.ROTATE_90) {
    …
  } else if (rotate == Generator.ROTATE_270) {
    …
```

Verschiebt negative Gradzahlen auf entsprechende positive

Aufgabe 2
JUnit – Testfallbeispiel (Aufgabe 2h)

```
@After
public void tearDown() {
  SimpleDateFormat sdf =
    new SimpleDateFormat("MM-dd_HH.mm.ss.SSS");
  String time = sdf.format(new Date());

  File outputFile = new File(MessageFormat.format(
    "{0}/{1}_rotated_{2}.jpg", TEST_DIR, imageName,
    time));

  if (rotatedImage != null) {
    // speichern siehe rotate()…
  }
}
```

Aufgabe 2
JUnit – Patch erzeugen: Befehl und Meta-Infos

```
C:\swt1-ss19>git format-patch -1 --stdout

From 31a13bae25f3e8ccbaf51c2964c4c738a0a   c4f Mon Sep
17 00:00:00 2001
From: Tobias Hey
<hey@kit.edu>
Date: Mon, 22 April 2019 19:01:44 +0200
Subject: [PATCH] test

---
…
```

Aufruf funktioniert hier, weil die Änderung im vorangegangenen Commit war, der Patch also die Änderung zwischen „HEAD“ und Vorgänger (–1) darstellt.
(Aufruf abweichend – aber möglich – für Patches zwischen zwei Commits.)

Aufgabe 2
JUnit – Patch erzeugen (2): Wo wurde geändert

```
src/main/java/org/jis/generator/Generator.java |    2
++
 1 files changed, 3 insertions(+), 0 deletions(-)

diff --git
a/src/main/java/org/jis/generator/Generator.java
b/src/main/java/org/jis/generator/Generator.java
index 8fbe910..f90a390 100644
--- a/src/main/java/org/jis/generator/Generator.java
+++ b/src/main/java/org/jis/generator/Generator.java
```

Aufgabe 2
JUnit – Patch erzeugen (3): Was wurde geändert

```
   int width = image.getWidth(null);
   int height = image.getHeight(null);

+  if (rotate < 0) {
+    rotate += Math.toRadians(360);
+  }
   if (rotate == Generator.ROTATE_90) {
     transform.translate(height, 0);
     transform.rotate(Generator.ROTATE_90);
```

Testüberdeckung

AUFGABE 3

Aufgabe 3
Testüberdeckung – generateImage


```
@Test
public void testGenerateImage_ModusDefault() {
  Options.getInstance().setModus(Options.MODUS_DEFAULT);
  Options.getInstance().setAntialiasing(false);
  Options.getInstance().setCopyMetadata(false);
  Options.getInstance().setCopyright(false);
  File imageFileLandscape =
    new File(TEST_DIR, "landscape.png");
  try {
    ImageIO.write(testImage, "png", imageFileLandscape);
    generator.generateImage(imageFileLandscape,
      TEST_DIR, false, IMAGE_WIDTH / 2,
      IMAGE_HEIGHT / 2, "scaled-");
    …
```

Aufgabe 3
Testüberdeckung – generateImage

```
    File targetImage = new File(TEST_DIR,
      "scaled-landscape.png");
    BufferedImage loaded = ImageIO.read(targetImage);

    assertEquals(testImage.getHeight() / 2, loaded.getHeight());
    assertEquals(testImage.getWidth() / 2, loaded.getWidth());

    imageFileLandscape.delete();
    targetImage.delete();
  } catch (IOException e) {
    fail(e.getMessage());
  }
}
```

Aufgabe 3
Testüberdeckung – rotateFile

@Ignore um Testfall zu ignorieren

```
// Note: ken here. Therefore we skip the test to
// avoid blems.
@Ignore @Test
public void testRotateFileInt() {
  File testFile = new File(TEST_DIR, "test_rotateFileInt.jpg");
  try {
    // we need to copy the image because the original
    // is being transformed...
    Files.copy(IMAGE_FILE.toPath(), testFile.toPath(),
      StandardCopyOption.REPLACE_EXISTING);

    // method under test
    generator.rotate(testFile, 90);

    BufferedImage actual = ImageIO.read(testFile);
```

Aufgabe 3
Testüberdeckung – rotateFile

```
    // The rotate function using the int parameter is
    // broken as well, so we will get these results:
    // assertEquals(256, actual.getWidth());
    // assertEquals(128, actual.getHeight());

    // These are the correct results
    assertEquals(128, actual.getWidth());
    assertEquals(256, actual.getHeight());

    testFile.delete();
  } catch (IOException e) {
    fail(e.getMessage());
  }
}
```

Aufgabe 3: Bonusaufgabe
Mockito – pom.xml

- Hinzufügen der Abhängigkeit zur pom.xml-Datei

```
<dependencies>
        …
        <dependency>
                <groupId>org.mockito</groupId>
                <artifactId>mockito-all</artifactId>
        </dependency>
</dependencies>
```

Aufgabe 3: Bonusaufgabe
Mockito – Einbinden in Testklasse

```
@RunWith(MockitoJUnitRunner.class)
public class GeneratorTest {
  @Mock
  private static Main mockMain;

  @BeforeClass
  public static void beforeClass() {
    mockMain = Mockito.mock(Main.class);
  }

  @Before
  public void setUp() throws Exception {
    generator = new Generator(mockMain, 100);
  }
```

Aufgabe 3: Bonusaufgabe
Mockito – Anwendung

```
@Test
public void testGenerateText() {
  mockMain.mes = new Messages(
    Options.getInstance().getLocal());
    …

  generator.generateText(targetFolder, targetFolder,
      IMAGE_WIDTH / 2, IMAGE_HEIGHT / 2);
    …
}
```

Nachahmen der `Messages`-Instanzvariable