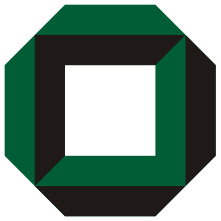
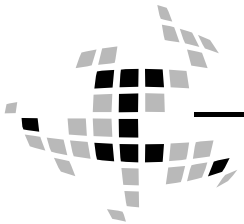


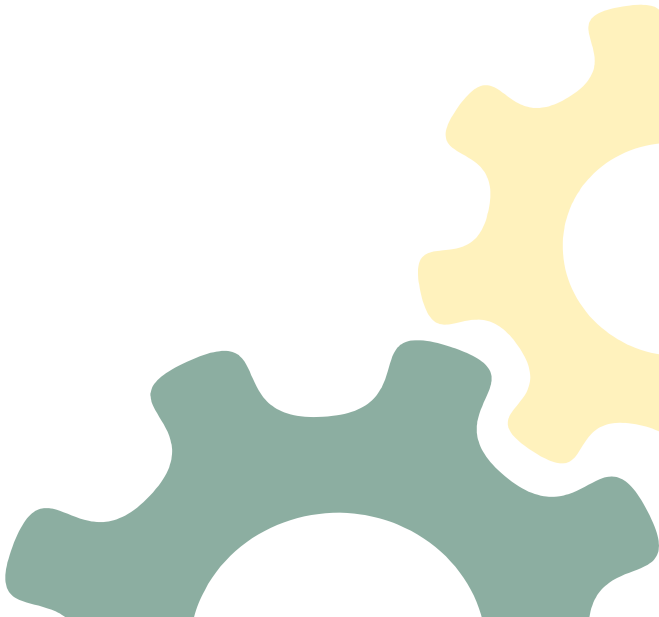
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

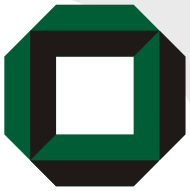
# Softwaretechnik 1
# Übung „Code-Inspektion“

06.07.2009



Fakultät für **Informatik**
Lehrstuhl für Programmiersysteme



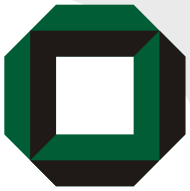


# Ablauf der Übung

1. Vorbereitung
2. Individuelle Fehlersuche
3. Gruppensitzung
4. Nachbereitung
5. Prozessverbesserung

# 1. Vorbereitung

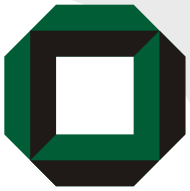

- Teilnehmer und ihre Rollen festlegen.
  - Alle anwesenden Studenten und Studentinnen
- Dokumente und Formulare vorbereiten
  - Liste für gefundene Defekte (engl. defect log, issue log), die Prüfliste sowie der Quelltext werden gleich ausgeteilt
- Lesetechniken festlegen
  - Prüfliste (s.o.)
- Zeitlichen Ablauf planen



# 2. Individuelle Fehlersuche

- Jeder Inspektor prüft für sich das Dokument
  - In dieser Übung: 4 Seiten pro Inspektor
- Benutzt vereinbarte Lesetechnik
- Trägt alle Problempunkte und die genaue Stelle im Dokument in das ausgeteilte Formular ein (→ defect log)
- Problempunkte: (mögliche) Defekte, Verbesserungsvorschläge, Fragen

# … Und los gehts



# 3. Gruppensitzung

- Individuell gefundene Problempunkte sammeln
- Jeden einzelnen Problempunkt besprechen
- Fragen zum Dokument klären
- Weitere Problempunkte sammeln, die während der Diskussion gefunden werden
- Verbesserungsvorschläge zur Durchführung von Inspektionen sammeln

# 3. Gruppensitzung – Problempunkte

```
package mandelbrot;

/**
 * Sequential implementation of the Mandelbrot set computation.
 * Multiple OOMandelbrot instances with different dimensions may exist.
 *
 * @author K. Putt
 */
public class OOMandelbrot {
  private static final double MAGNITUDE = 4.0;
  private static final int MAX_ITER = 1500;
  private static final double TOP_LEFT_REAL = -2;
  private static final double TOP_LEFT_IMAG = 1.5;
  private static final double BOTTOM_RIGHT_REAL = 1;
  private static final double BOTTOM_RIGHT_IMAG = -1.5;

  protected static int width;
  protected static int height;
```

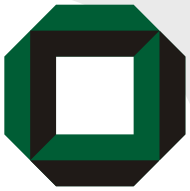
Dürfen nicht **`static`** sein!
Typ: 1

# 3. Gruppensitzung – Problempunkte

```
public OOMandelbrot(int width, int height) { ... }

/**
 * Triggers the computation of the Mandelbrot set. It also prints the
 * computation's duration in seconds to the console.
 *
 * @return the Mandelbrot
 */
public final int[][] computeMandelbrot() {
  long start = System.nanoTime();
  int[][] temp = compute();
  System.out.print((System.nanoTime() - start) / 1000000);
  return temp;
}
```

Hier stehen Sekunden…

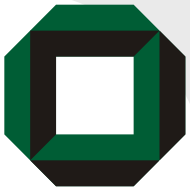
berechnet werden aber Millisekunden. Typ: 10



# 3. Gruppensitzung – Problempunkte

```
/**
 * Computes the Mandelbrot set. This method has to be implemented by all
 * sub-classes.
 *
 * @return the Mandelbrot
 */
protected final int[][] compute() {
  int[][] result = new int[width][height];
  Complex c;
  for (int x = 0; x <= width; x++) {
    for (int y = 0; y <= height; y++) {
      c = translate(new Complex(x, y));
      result[x][y] = computeMandelbrotPoint(c);
    }
  }
  return result;
}
```

Abbruchbedingung der Schleifen ist nicht korrekt: <= statt <
Typ: 4



# 3. Gruppensitzung – Problempunkte

```
/**
 * Translates the complex number c.
 *
 * @return the translated complex number
 */
protected final Complex translate(Complex c) {
    double real = TOP_LEFT_REAL + (double) c.real() / height
        * (BOTTOM_RIGHT_REAL - TOP_LEFT_REAL);
    double imag = TOP_LEFT_IMAG + (double) c.imag() / width
        * (BOTTOM_RIGHT_IMAG - TOP_LEFT_IMAG);

    return new Complex(real, imag);
}
```

Kommentar für Parameter **c** fehlt.
Typ: 10

Einrückung zu tief.
Typ: 11

# 3. Gruppensitzung – Problempunkte

```
/**
 * Computes one point of the Mandelbrot set.
 *
 * @param c
 *      the point to compute
 * @return the number of iterations after which the computation stopped. If
 *     this number exceeds the maximal number of iterations 0 is
 *     returned.
 */
protected final int computeMandelbrotPoint(Complex c) {
  Complex z = new Complex(0, 0);
  int i = 0;
  do {
    i++;
    c = transform(z, c);
  } while (((z.modsq()) <= MAGNITUDE) & (i <= MAX_ITER));
  return i % MAX_ITER;
}
```

Ergebnis von `transform(z, c)` muss `z` zugewiesen werden.
Typ: 5

Hier sollte besser „&&“ stehen
Typ: 6 (+ 14)

# 3. Gruppensitzung – Problempunkte

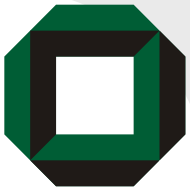
```
  /**
   * Computes one iteration of the Mandelbrot polynomial z(n+1) <- z(n)^2 + c
   * @param z   z
   * @param c   c
   * @return z(n+1)'s
   */
  protected final Complex transform(Complex z, Complex c) {
    Complex C = (z.square()).add(c);
    return c;
  }

  /**
   * @return the height of the Mandelbrot set
   */
  public final int getWidth() {
    return width;
  }

  /**
   * @return the height of the Mandelbrot set
   */
  public int getHeight() {
    return height;
  }
}
```

Die Benennung ist ungünstig.
Typ: 1

Hier sollte `C` statt `c` zurückgegeben werden.
Typ: 1

Hier sollte **`width`** stehen.
Typ: 10

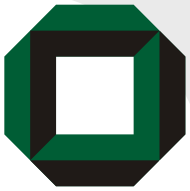
Die Methode sollte **`final`** sein.
Typ: 2



# 3. Gruppensitzung – Problempunkte

```
package mandelbrot;

/**
 * Bibliotheksklasse zum Rechnen mit komplexen Zahlen. Alle Berechnungen werden
 * mit einem Fehler von h&ouml;chstens 10<sup>-5</sup> durchgef&uuml;hrt.
 *
 * @author D. Fäckt
 *
 */
public final class Complex {
  private double real;
  private double imag;

  public Complex(Complex c) { ... }
  public Complex(double real, double imag) { ... }
  public double real() { ... }
  public double imag() { ... }

  public Complex add(Complex c) { ... }
```

Die Kommentare dieser Klasse sind Deutsch statt Englisch.
Typ: 10 (nicht in der Liste)



# 3. Gruppensitzung – Problempunkte

```
/**
 * Subtrahiert die komplexe Zahl <code>c</code> von der komplexen Zahl
 * <code>this</code>. Achtung! Die komplexe Zahl <code>this</code> wird bei
 * dieser Operation ver&auml;ndert und als Ergebnis zur&uuml;ckgeliefert.
 * Benutzen Sie den Kopierkonstruktor, um die komplexe Zahl vor dem
 * Ausf&uuml;hren der Operation zu sichern.
 *
 * @param c
 *      der Subtrahend
 * @return das Ergebnis der Subtraktion
 */
public Complex sub(Complex c) {
  double r = real + c.real;
  double i = imag + c.imag;
  real = r;
  imag = i;
  return this;
}
```

Hier sollte „-“ statt „+“ stehen.
Typ: 5

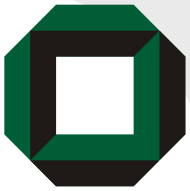
# 3. Gruppensitzung – Problempunkte

```
/**
 * Multipliziert die komplexe Zahl <code>this</code> mit der komplexen Zahl
 * <code>c</code>. Achtung! Die komplexe Zahl <code>this</code> wird bei
 * dieser Operation ver&auml;ndert und als Ergebnis zur&uuml;ckgeliefert.
 * Benutzen Sie den Kopierkonstruktor, um die komplexe Zahl vor dem
 * Ausf&uuml;hren der Operation zu sichern.
 *
 * @param c
 *      der Multiplikator
 * @return das Ergebnis der Multiplikation
 */
public Complex mul(Complex c) {
  real = real * c.real - imag * c.imag;
  imag = imag * c.real + real * c.imag;
  return this;
}
```

Alten Wert für **real** nicht Zwischengespeichert!
Typ: 5

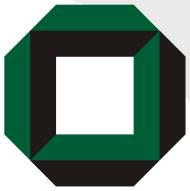
# 3. Gruppensitzung – Problempunkte

```
  /**
   * Quadriert die komplexe Zahl <code>this</code>. Achtung! Die komplexe Zahl
   * <code>this</code> wird bei dieser Operation ver&auml;ndert und als
   * Ergebnis zur&uuml;ckgeliefert. Benutzen Sie den Kopierkonstruktor, um die
   * komplexe Zahl vor dem Ausf&uuml;hren der Operation zu sichern.
   *
   * @return das Quadrat
   */
  public Complex square() {
    double r = real * real - imag * imag;
    double i = 2 * real * imag;
    real = r;
    imag = i;
    return this;
  }
}
```



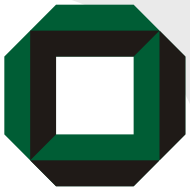
# 3. Gruppensitzung – Problempunkte

```
  /**
   * Berechnet das Quadrat des Betrags der komplexen Zahl <code>this</code>.
   *
   * @return das Quadrat des Betrags
   */
  public double modsq() {
    return real * real + imag * imag;
  }

  /**
   * Erstellet eine textuelle Repr&auml;sentation der komplexen Zahl
   * <code>this</code> in der Form (&lt;Realteil&gt; +
   * &lt;Imagin&auml;rteil&gt;i).
   *
   * @return die textuelle Repr&auml;sentation
   */
  @Override
  public String toString() {
    return real + " + " + imag + "i";
  }
}
```



# 4. Nachbereitung

- Liste mit allen Problempunkten wird an den Editor des Dokuments weitergeleitet
- Editor identifiziert tatsächliche Defekte und klassifiziert sie
- Editor leitet Änderung des Dokuments ein
- Alle Problempunkte werden bearbeitet
- Deren Bearbeitung wird überprüft
- Schätze Restdefekte
  - # nicht entdeckte Defekte ≈ # entdeckte Defekte
  - 1/6 der Korrekturen ist fehlerhaft/verursacht neuen Defekt
- Dokument wird freigeben wenn #geschätzte Def. < N



# 5 Prozessverbesserung

- Prüflisten und Szenarien anpassen
- Standards für Dokumente erarbeiten
- Defekt-Klassifikationsschema anpassen
- Formulare verbessern
- Planung und Durchführung verbessern