Karlsruher Institut für Technologie

Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Lehrstuhl Prof. Walter F. Tichy

E-Mail: swt1@ipd.kit.edu

Ilias: https://ilias.studium.kit.edu/goto.php?target=crs_945596&client_id=produktiv

Web: http://ps.ipd.kit.edu

LEZ: https://lez.ipd.kit.edu

SWT1: Übungsblatt 6

Kontrollfluss-orientiertes Testen, Codeinspektion, Äquivalenzklassen und Parallelisierung

Ausgabe: Mittwoch, 3. Juli 2019
Abgabe: Mittwoch, 17. Juli 2019, 12:00 Uhr

Geben Sie Ihre Lösung mit Deckblatt (mit Name, Matrikelnummer und der Nummer Ihres Tutoriums deutlich lesbar) ab, damit Ihr Tutor Korrekturhinweise und Ihre Punkte notieren kann. Werfen Sie es in die Holzkiste vor Raum 369 im Informatik-Hauptgebäude 50.34. Verwenden Sie ausschließlich das Deckblatt zur SWT1 aus ILIAS.

Alle Theorie-Aufgaben inklusive aller Diagramme sind handschriftlich anzufertigen!
Ausgeschlossen sind auch kopierte oder ausgedruckte „handschriftliche“ Lösungen!

Aufgabe 1: Parallelisierung von HDrize (6 Punkte + 2 Bonuspunkte)

Sie haben den Fehler begangen, Ihrer Projektleiterin vom hervorragenden Bildungsangebot des KIT zu berichten. Sie hat sich umgehend für mehrere Kurse eingeschrieben und erscheint seitdem täglich mit neuen guten Ideen. *Gut* bedeutet in diesem Kontext vor allem *arbeitsaufwendig* für Sie und Ihr Team. Im letzten Kurs ging es um Parallelprogrammierung. „Das ist unsere Chance iMage mal auf Vordermann zu bringen. Die Berechnung der HDR-Bilder dauert ja ewig.“, stellt Ihre Projektleiterin fest. Zähneknirschend geben Sie Ihr recht, schließlich wissen sie selbst um das Parallelisierungspotential, das in HDrize schlummert. Ihre Projektleiterin merkt sofort, dass Sie für den Job perfekt geeignet sind und beauftragt Sie und Ihr Team mit der Umsetzung von zwei ihrer Ideen: die parallele Berechnung der einzelnen Farbkanäle für die Antwortkurve und die Verwendung des ijk-Algorithmus für die Matrix-Multiplikation.

a) Legen Sie ein neues Modul namens `iMage.HDrize.parallel` für die parallele Implementierung von HDrize an. Geben Sie `iMage.HDrize` als Abhängigkeit an. Sie können hierfür Ihre eigene Implementierung oder die bereitgestellte Musterlösung verwenden.
b) Bonusaufgabe: Beschleunigen Sie die sequenzielle Implementierung der Antwortkurvenberechnung (Klasse `CameraCurve`, Methode `calculate()`) mittels Parallelität. Legen Sie hierfür eine neue Klasse `CameraCurveParallel` an, welche die Klasse `CameraCurve` erweitert und die entsprechende Methode überschreibt. Parallelisieren Sie die Berechnung der einzelnen Farbkanäle. Verwenden Sie ein Parallelisierungskonstrukt Ihrer Wahl. Sie können (müssen aber nicht) beispielsweise Faden-Pools (engl. *Thread Pools*) – Tutorial unter https://docs.oracle.com/javase/tutorial/essential/concurrency/pools.html - verwenden. Legen Sie selbstständig eine sinnvolle Anzahl von Fäden (engl. *Threads*) fest.
c) Beschleunigen Sie die sequenzielle Implementierung der Matrix-Multiplikation (Klasse `MatrixCalculator`, Methode `multiply(Matrix a, Matrix b)`) mittels Parallelität. Legen Sie hierfür eine

neue Klasse `MatrixCalculatorParallel` an, welche die Klasse `MatrixCalculator` erweitert und die entsprechende Methode überschreibt. Parallelisieren Sie die Multiplikation der Matrizen mittels des parallelen ijk-Algorithmus. Stellen Sie einen Konstruktor bereit, mit dem man die Anzahl der zu verwendenden Fäden festlegen kann (als int-Parameter): `public MatrixCalculatorParallel(int numThreads)`. Ihre parallele Matrix-Multiplikation soll diese Anzahl an Fäden zur Berechnung verwenden. Benutzen Sie auch hier ein Parallelisierungskonstrukt Ihrer Wahl.

d) Erstellen Sie eine parallele Variante von HDrize in einer neuen Klasse `HDrizeParallel`, welche die Schnittstelle `IHDrizeParallel` (aus der aktuellsten `HDrize.base`) implementiert. Die zu implementierende Schnittstellenmethode `createRGB(InputStream[] images, int samples, double lambda)` soll dieselbe Funktionalität wie die Methode `createRGB(EnhancedImage[] enhancedImages, int samples, double lambda, IMatrixCalculator<Matrix> mtxCalc, ToneMapping mapping)` aus der Klasse `HDrize` umsetzen (konsultieren Sie ggf. die Musterlösung). Da die parallele Variante von HDrize für unterschiedliche Plattformen vertrieben wird, müssen Sie darauf achten, dass die Applikation sowohl auf langsamen Smartphone-Prozessoren mit nur 2 Kernen als auch auf Rechnern mit 8 Kernen und Hyperthreading die Parallelität möglichst optimal ausnutzt. Stellen Sie auch hier einen Konstruktor bereit, mit dem man die Anzahl der zu verwendenden Fäden festlegen kann (als int-Parameter): `public HDrizeParallel(int numThreads)`. Ein zusätzlicher Konstruktor ohne `numThreads`-Parameter soll die ideale Fadenzahl für die aktuelle Ausführungsumgebung (d.h. die verfügbare Prozessoranzahl) selbst bestimmen. Sorgen Sie dafür, dass die festgelegte (bzw. bestimmte) Fadenanzahl an die Klasse `MatrixCalculatorParallel` weitergereicht wird. Führen Sie gegebenenfalls weitere Konstruktoren und/oder Methoden in anderen Klassen ein. Wird als Anzahl der zu verwendenden Fäden 1 gewählt, soll nicht die parallele Implementierung mit einem Faden gestartet werden. Stattdessen soll die sequentielle Implementierung (siehe Übungsblatt 2) verwendet werden. Sie können hierfür Ihre eigene Implementierung oder die im ILIAS bereitgestellte Musterlösung verwenden.

e) Stellen Sie sicher, dass die parallele Version korrekte Ergebnisse erzeugt. Schreiben Sie hierfür Unit-Tests.

Beachten Sie beim Bearbeiten der Aufgabe die folgenden Hinweise:

1. Verwenden Sie nicht die GPU oder andere Spezial-Hardware, beschränken Sie sich auf die Verwendung der CPU. Verwenden Sie ausschließlich Java.
2. Beachten Sie bei der Wahl der Anzahl der Fäden für den `MatrixCalculatorParallel`, dass sofern Sie die Bonusaufgabe (A1 b)) bearbeitet haben, die `multiply`-Methode durch die parallele `calculate`-Methode aufgerufen wird.
3. Verwenden Sie ein geeignetes Testbild für Ihre Unit-Tests. Sie können – müssen aber nicht – die für Aufgabe 2 vorgesehenen Testbilder aus ILIAS verwenden (Das Ergebnisbild wurde mit folgender Konfiguration erstellt: Lambda=30, Samples=500, ToneMapping=SRGBGamma).
4. In ILIAS finden Sie eine Konfigurationsdatei (`src/assembly/src.xml`), die eingebunden werden soll.
5. Achten Sie darauf, dass Ihre Abgabedatei nicht zu groß wird (z.B. aufgrund von Testbildern): Die Abgabedatei soll nicht größer sein als 5MB.
6. Führen Sie vor der Abgabe die üblichen Plausibilitätstests durch.

Aufgabe 2: Parallelisierungswettbewerb (7 Punkte + Sonderpreis für die schnellste Implementierung)

Ihre parallele Variante von HDrize ist Ihrer Projektleiterin immer noch nicht schnell genug (auch wenn Sie bezweifeln, dass sie eine Vorstellung hat, wie schnell eine parallele Implementierung sein könnte). Da Ihre Projektleiterin ein wenig das Vertrauen in Sie zu verlieren scheint, macht Sie dem Vorstand den Vorschlag, die Aufgabe, eine schnellere Parallelisierung zu implementieren, firmenweit auszuschreiben. Für die beste Lösung wird ein noch geheimer Preis sowie der Titel „bester Parallelisierer des Monats“ ausgelobt. Sie entscheiden sich unter dem Pseudonym Ewen F. Aster am Parallelisierungswettbewerb teilzunehmen, um Ihrer Projektleiterin zu beweisen, dass Sie es doch am besten können, und beginnen sofort mit der Suche nach einer Lösung.

a) Entwickeln Sie eine möglichst schnelle parallele Variante von HDrize. Sie dürfen beliebige Programmteile parallelisieren und anderweitig optimieren. Sie können die Lösung aus Aufgabe 1 erweitern, aber auch teilweise oder völlig revidieren. Beschreiben Sie zunächst textuell grob, wie Sie bei der Parallelisierung vorgegangen sind, an welchen Stellen Sie Parallelität eingesetzt haben und warum. Wie in Aufgabe 1 gilt, dass die Applikation sowohl auf Smartphone-Prozessoren mit nur 2 Kernen als auch auf Rechnern mit 8 Kernen und Hyperthreading die Parallelität möglichst optimal ausnutzt. Erklären Sie auch, wie Sie die ideale Anzahl der Fäden ermitteln.
b) Setzen Sie nun Ihren Ansatz in Quelltext um. Erweitern Sie hierzu Ihr Modul `iMage.HDrize.parallel` aus Aufgabe 1 (oder legen Sie dieses an, falls Sie Aufgabe 1 nicht bearbeitet haben). Wie in Aufgabe 1 gilt, dass sämtliche parallelisierten Klassen (und alle ihre Unterklassen) einen Konstruktor bereitstellen müssen, mit dem man die Anzahl der zu verwendenden Fäden wählen kann (als `int`-Parameter); der Konstruktor in `HDrizeParallel` ohne `numThreads`-Parameter soll die ideale Fadenzahl für die aktuelle Ausführungsumgebung (d.h. die verfügbare Prozessoranzahl) selbst bestimmen.
c) Stellen Sie sicher, dass die parallele Version weiterhin korrekt funktioniert. Schreiben Sie hierfür Unit-Tests.

Um Ihre Lösung in einem guten Licht dastehen zu lassen, bereiten Sie für die nächste Projektsitzung einen Performanzvergleich vor, der die Vorzüge Ihrer neuen Variante dokumentiert. Sie finden in ILIAS den Testfall `IPDPerformanceTest` und darin eine Testmethode, die die Zeitmessung für `HDrize` vornimmt. Belassen Sie

diese Test-Methode unverändert, damit Ihre Projektleiterin die Ergebnisse mit denen Ihrer Kollegen vergleichbar kann. (Ändern Sie insbesondere nicht das Paket, in dem `HDrizeParallel` liegt.)

d) Erstellen Sie ein Laufzeitprofil. Gehen Sie wie folgt vor:
   I. Beschreiben Sie das System, auf dem Sie testen: Prozessortyp, Anzahl der Kerne (mit Hyperthreading?), Größe des Arbeitsspeichers und das von Ihnen verwendete Betriebssystem mit Architektur (x86/x64). Der von Ihnen verwendete Rechner sollte dabei mindestens zwei Kerne (oder einen Kern mit Hyperthreading) haben.
   II. Führen Sie pro Berechnung mindestens 5 Messungen durch und dokumentieren Sie die durchschnittliche Laufzeit für eine Berechnung. Führen Sie diesen Vorgang für Ihre ursprüngliche sequentielle Version durch sowie für die von Ihnen implementierte, jeweils mit 1, 2, 4, 8, 16, 32, 64 und 128 Ausführungsfäden. Berechnen Sie jeweils die Effizienz sowie die Beschleunigung im Vergleich zur ursprünglichen sequentiellen Methode. Verwenden Sie für die Performanzmessung die Bilder *input_1_10.jpg*, *input_1_25.jpg*, *input_1_80.jpg* aus dem ILIAS. Verwenden Sie außerdem `SRGBGamma` für das Tonemapping, 30 als Lambda-Wert sowie 500 als Anzahl der verwendeten Stichproben. (Ein Teil dieser Aufgabe ist im Testfall `IPDPerformanceTest` bereits vorbereitet.)
   III. Verwenden Sie die Tabelle aus ILIAS als Vorlage zur Dokumentation.

e) JUnit erlaubt das Setzen einer maximalen Laufzeit für einen Testfall (siehe Aufgabe 3). Schreiben Sie damit einen JUnit-Testfall, der sicherstellt, dass die parallele Implementierung mit dem Standardkonstruktor nicht länger läuft als die sequentielle.

Wählen Sie eine sinnvolle Zeitschwelle basierend auf dem Laufzeitprofil aus Aufgabenteil d); wenn Ihre parallele Implementierung langsamer ist als die sequenzielle, dann kennzeichnen Sie den Testfall vor der Abgabe mit `@Ignore` und einem entsprechenden Hinweis.

Beachten Sie beim Bearbeiten der Aufgabe neben den fünf in Aufgabe 1 gegebenen ebenfalls die folgenden Hinweise:

1. Um den Parallelisierer des Monats zu ermitteln, wird mit der sequenziellen Musterlösung verglichen.
2. Abgesehen von der Verwendung von Java und der Beschränkung auf die CPU (siehe Hinweis 1 in Aufgabe 2) sind Sie frei in der Wahl der Mittel – Beschleunigung ist das Ziel (d.h. setzen Sie auch Optimierungen ein, die im sequenziellen Fall helfen).

Geben Sie die Parallelisierungsbeschreibung und das Laufzeitprofil in Papierform ab.

Aufgabe 3: Funktionales Testen (3 Punkte)

Eine Bibliothek für Konto-Buchungen, die große Dezimalzahlen unterstützt, soll in Zukunft für die Abrechnung der Löhne und Gehälter der Pear Corp. verwendet werden und wurde zu diesem Zweck von einem Drittanbieter lizensiert. Da die Lizenz äußerst günstig erworben wurde, verfügt die Pear Corp. lediglich über die Bibliothek sowie eine Javadoc-Dokumentation – nicht aber über den Quelltext.

Beim Einsatz der Bibliothek bei der Berechnung der Vorstands-Gehälter hat sich gezeigt, dass die Bibliothek fehlerhaft ist. „So wenig kann es nicht sein, für unter 8k im Monat steh ich morgens nicht mal auf“, stellt Dr. Salzhügel fest. Eine spontane Eingabe Ihrer Projektleiterin bringt Sie auf die richtige Spur: „Die hebeAb()-Funktion hält irgendwie nicht, was die Dokumentation verspricht. Fangen Sie mal dort an zu suchen!“ Da Sie befürchten, dass die neue Bibliothek keine Relevanz mehr für Ihr Gehalt haben könnte, wenn Sie dieser „Bitte“

nicht schleunigst nachkommen, beeilen Sie sich eine Lösung zu finden. Sie beginnen damit, sich zu überlegen, wie man sinnvolle Testfälle bestimmen könnte.

```
public interface IKonto {
  private BigDecimal kontostand;
  //...

  /**
   * Hebt den spezifizierten Betrag ab, sofern der aktuelle Kontostand
   * und der angegebene Kreditrahmen dies erlauben.
   *
   * @param betrag
   *          der abzuhebende Betrag (>= 0)
   * @param kreditRahmen
   *          der Kreditrahmen des aktuellen Kontos (>= 0)
   * @return wahr falls eine Abhebung durchgeführt werden konnte, unwahr sonst
   */
  public boolean hebeAb(BigDecimal betrag, BigDecimal kreditRahmen);
  //...
}
```

a) Bestimmen Sie für die Methode `hebeAb(…)` die Äquivalenzklassen mit zugehörigem Repräsentanten. Beschreiben Sie jede der Klassen in einem Satz.

b) Bestimmen Sie zusätzlich die Grenzwerte für Ihre Äquivalenzklassen.

Aufgabe 4: Kontrollfluss-orientiertes Testen (7 Punkte + 1 Bonuspunkt)

Der Vorstand steht diesem „modernen Parallelzeugs“ immer noch kritisch gegenüber und legt fest, dass man das Ganze ausgiebig testen sollte. Zwar wurden bereits Komponentests geschrieben, Sie merken aber an, dass diese nicht unbedingt ausreichend seien, um Testvollständigkeit zu garantieren. "Da merkt man, wie viel Sie schon unter meiner Obhut gelernt haben", kommentiert Ihre Projektleiterin. Sie selbst muss für die nächsten zwei Wochen leider mit dem Vorstand auf ein technisches Kolloquium auf Teneriffa. Daher übergibt Sie die Aufgabe vertrauensvoll in die Hände „ihrer besten Arbeitskraft“. Aus Ihrer Studienzeit am KIT erinnern Sie sich an Kontrollfluss-orientiertes Testen und Testabdeckungskriterien. Als Sie die Praktikanten Knöpfle und Spätzle mit der Umsetzung beauftragen, bemerken Sie deren ahnungslose Blicke und beschließen, das Verfahren erstmal an der folgenden Java-Methode zu demonstrieren:

```
01: public boolean isIdentity(double[][] mtx) {
02:   if (mtx == null || mtx.length != mtx[0].length) {
03:     return false;
04:   }
05:   int n = mtx.length;
06:
07:   for (int r = 0; r < n; r++) {
08:     for (int c = 0; c < n; c++) {
09:       if (r == c && Math.abs(mtx[r][c] - 1) >= 1E-8) {
10:         return false;
11:       } else if (r != c & Math.abs(mtx[r][c]) >= 1E-8) {
12:         return false;
13:       }
14:     }
15:   }
16:   return true;
17: }
```

a) Erstellen Sie für die oben angegebene Methode `isIdentity(double[][] mtx)` den Kontrollflussgraphen. Schreiben Sie den Quelltext in die Kästchen, Verweise auf Zeilennummern der Methode sind nicht ausreichend.

Hinweis: Wir empfehlen die Methode zunächst in die in der Vorlesung besprochene Zwischensprache zu übersetzen.

b) Geben Sie jeweils eine minimale Testfallmenge sowie die durchlaufenen Pfade an, mit denen Anweisungsüberdeckung bzw. Zweigüberdeckung erreicht werden. Falls es nicht möglich ist das Kriterium zu erfüllen, begründen Sie, warum das Überdeckungskriterium nicht erfüllbar oder praktikabel ist.

Hinweis: „Minimal“ heißt: die Testfallmenge enthält gerade die Testeingaben, mit denen das Abdeckungskriterium erfüllt wird und keine mehr.

c) Bonusaufgabe: Begründen Sie, warum das Erreichen der Pfadüberdeckung für die angegebene Methode `isIdentity(double[][] mtx)` nicht praktikabel ist.

Aufgabe 5: Codeinspektion (4 Punkte)

Um Ihnen für die Entwicklung neuer Funktionalität für iMage mehr finanziellen Spielraum geben zu können, entschließt sich der Vorstand der Pear Corp. dafür, so wörtlich, „iMage endlich gewinnbringend zu vermarkten“. Hierzu soll ein Teil des iMage-Quelltextes als Bibliothek veröffentlicht und unter Lizenz an Mitbewerber verkauft werden. Auch wenn der Vorstand großes Vertrauen in die Fähigkeiten Ihrer Entwickler hat, soll vor der Veröffentlichung der Quelltext in seiner aktuellen Version einer weiteren manuellen Prüfung unterzogen werden. „Man will sich ja schließlich nicht vor den Mitbewerbern lächerlich machen!“, bringt es ein Vorstandsmitglied auf den Punkt.
Sie werden mit der Aufgabe betraut, den Quelltext der `Matrix`-Klasse einer Codeinspektion zu unterziehen. Ihre Projektleiterin überreicht Ihnen hierzu eine Prüfliste der einzuhaltenden Standards in der Pear Corp.

Überprüfen Sie den Quelltext der folgenden `Matrix`-Klasse auf Mängel. Ein Mangel ist dabei als eine Verletzung einer der Regeln zu verstehen, die in der Prüfliste angegeben sind. Protokollieren Sie die gefundenen Mängel in einer Tabelle folgender Form:

Regelnummer	Zeilennummer(n)	Kurzbeschreibung
⋮	⋮	⋮

Wenn Sie einen Mangel entdecken, tragen Sie ihn in die Tabelle in folgender Form ein: Geben Sie zunächst jeweils die Nummer der verletzten Regel an, anschließend die Zeilennummer(n) im Quelltext sowie eine kurze Beschreibung des Mangels.

```
01: package org.iMage.HDrize.matrix;
02:
03: import org.iMage.HDrize.base.matrix.IMatrix;
04: import org.ojalgo.matrix.store.PrimitiveDenseStore;
05:
```

```java
/**
 * A matrix.
 *
 */
public final class Matrix implements IMatrix {

  private final int rows;
  private final int cols;

  final double[][] data;

  /**
   * Create a new matrix.
   *
   * @param m the original matrix
   */
  public Matrix(IMatrix m) {
    this(m.copy());
  }

  /**
   * Create a new matrix.
   *
   */
  private Matrix(double[][] m) {
    this(m.length, m.length == 0 ? 0 : m[0].length);
    for (int r = 0; r < this.rows; r++) {
      if (m[r].length != this.cols) {
        throw new IllegalArgumentException("Rows have not equal lengths.");
      }
      for (int c = 0; c < this.cols; c++)
        this.set(r, c, m[r][c]);
    }

  }

  /**
   * Create a matrix (only zeros).
   *
   * @param rows the amount of rows
   */
  public Matrix(int rows, int cols) {
    if (rows < 1 || cols < 1) {
      throw new IllegalArgumentException("Rows and cols have to be >= 1");
    }
    this.rows = rows;
    this.cols = cols;
    this.data = new double[rows][cols];
  }

  @Override
  public double[][] copy() {
    double[][] copy = new double[this.rows][this.cols];
    for (int r = 0; r < this.rows; r++) {
      for (int c = 0; c < this.cols; c++) {
        copy[r][c] = this.data[r][c];
      }
    }
    return copy;
  }
```

```
65:
66:   @Override
67:   public int rows() {
68:       return this.rows;
69:   }
70:
71:   @Override
72:   public int cols() {
73:       return this.cols;
74:   }

75:   @Override
76:   public void set(int r, int c, double v) {
77:     this.data[r][c] = v;
78:
79:   }
80:
81:   @Override
82:   public double get(int r, int c) {
83:     return this.data[r][c];
84:   }
85:
86:   @Override
87:   public String toString() {
88:     StringBuilder builder = new StringBuilder();
89:     String klammerZu = "}";
90:     builder.append("{\n");
91:     for (int r = 0; r < this.rows(); r++) {
92:         builder.append("{");
93:         for (int c = 0; c < this.cols(); c++) {
94:           builder.append(this.get(r, c));
95:           if (c != this.cols() - 1) {
96:               builder.append(", ");
97:           }
98:         }
99:         builder.append("}\n");
100:       }
101:       builder.append("}");
102:       return builder.toString();
103:     }
104:   }
109:}
```

PRÜFLISTE FÜR JAVA-QUELLTEXT

1. Indikator: Variablen- und Konstantendeklaration

Regel 10. Sind alle Namen sprechend gewählt und in Einklang mit der Java-Namenskonvention?

Regel 11. Hat jede Variable und Konstante den richtigen Typ?

Regel 12. Ist jede Variable und Konstante richtig initialisiert?

Regel 13. Gibt es Literale, die als Konstante deklariert sein sollten?

Regel 14. Gibt es lokale Variablen oder Instanz- und Klassenvariablen, die Konstanten sein sollten?

Regel 15. Gibt es Klassen- oder Instanzvariablen, die lokale Variablen sein sollten?

Regel 16. Haben alle Klassen- und Instanzvariablen die richtigen Zugriffsmodifizierer (default, `private`, `protected`, `public`)?

Regel 17. Gibt es Instanzvariablen, die Klassenvariablen (`static`) sein sollten (und umgekehrt)?

Regel 18. Gibt es importierte Bibliotheken, die nirgends verwendet werden?

2. **Indikator: Methoden- und Konstruktorendefinition**
   - Regel 20. Sind alle Namen sprechend gewählt und in Einklang mit der Java-Namenskonvention?
   - Regel 21. Wird jeder Methodenparameter vor Gebrauch auf korrekte Eingabewerte geprüft?
   - Regel 22. Gibt jede `return`-Anweisung den korrekten Wert zurück?
   - Regel 23. Haben alle Methoden und Konstruktoren die richtigen Zugriffsmodifizierer (default, `private`, `protected`, `public`)?
   - Regel 24. Gibt es Instanzmethoden, die Klassenmethoden (`static`) sein sollten (und umgekehrt)?
3. **Indikator: Kommentare**
   - Regel 30. Haben alle Klassen sowie alle nicht-privaten Konstruktoren, Methoden, Konstanten, Klassen- und Instanzvariablen komplette JavaDoc-Kommentare?
   - Regel 31. Beschreibt der Kommentar die Funktionsweise des betreffenden Quelltextabschnitts korrekt?
4. **Indikator: Layout**
   - Regel 40. Ist die Einrückung korrekt und konsistent?
   - Regel 41. Haben alle Blöcke geschweifte Klammern?
5. **Indikator: Leistung**
   - Regel 50. Können Werte zwischengespeichert anstatt neu berechnet werden?
   - Regel 51. Wird jeder berechnete Wert auch verwendet?
   - Regel 52. Können Berechnungen aus dem Schleifenkörper genommen werden?

Hinweis zu den Werkzeugen

Git ist das in den Übungsaufgaben und in den Tutorien verwendete Werkzeug zur Versionskontrolle. Eclipse ist die in der Vorlesung und in den Tutorien verwendete Programmierumgebung. Sie können auch eine andere Programmierumgebung einsetzen, wenn Sie damit ausreichend vertraut sind und die gestellten Aufgaben genauso bearbeiten können. Es werden keine Aufgaben ausgegeben, deren Lösungen Eclipse-spezifische spezielle Fähigkeiten benötigen. Sie müssen „nur" in einer modernen Entwicklungsumgebung programmieren und Versionskontrolle einsetzen können.
Verwenden Sie bitte für alle Aufgaben Java 11.

Hinweis zu den Programmieraufgaben

Sofern nicht anders angegeben, sind die Aufgaben mit Java zu lösen. Die Einbuchungen und ihre Kommentare in Git werden bewertet (Existenz, sinnvolle Einbuchungshäufigkeit, -zeitpunkte und -dateien, sprechende Kommentare).
Konfigurieren Sie Maven mittels der Datei `pom.xml` **immer** wie folgt, um konsistente Projekte zu erhalten:

1. `ArtifactID` ist Ihre Matrikelnummer.
2. `GroupID` ist „swt1.ub<Übungsblatt-Nr.>.a<Aufgaben-Nr.>", also z. B. „swt1.ub0.a2" für Aufgabe 2 des Vorbereitungsblatts oder „swt1.ub3.a1" für Aufgabe 1 des dritten Übungsblattes.
3. Ändern Sie keine Einstellungen, die im XML-Dokument mit Kommentaren von uns versehen wurden (bspw. den Bereich DependencyManagement). Änderungen an diesen Bereichen können die automatische Analyse Ihrer Programme verhindern – in so einem Fall müssen Sie mit Punktabzug rechnen.
4. Wenn Sie Quelltexte abgeben, erzeugen Sie die Abgabedatei immer mit dem Befehl `mvn package`; laden Sie nicht die Binärfassung des Projekts hoch (*.jar), sondern die ZIP-Datei mit den Programmquellen. **Lösungen ohne Quelltext können nicht bewertet werden!**

Hinweis zur Lösungseinzugszentrale (LEZ)

Die Abgabe erfolgt per Lösungseinzugszentrale (LEZ). Sie können sich bei der LEZ mit Ihrem SCC-Benutzerkonto anmelden (https://lez.ipd.kit.edu/). Sie können pro Aufgabe eine einzelne Datei, ggf. auch ein gepacktes Archiv im ZIP-Format, abgeben. Soweit nicht anders angegeben, ist für die Programmieraufgaben hier immer `mvn package` auszuführen und die resultierende ZIP-Datei mit den Quelltextdateien Ihrer Lösung zu übertragen.
Achten Sie darauf, dass Sie eine signierte E-Mail der LEZ erhalten, in welcher eine Prüfsumme Ihrer abgegebenen Lösung enthalten ist. Diese dient bei technischen Problemen als Nachweis der Abgabe.

Hinweis zur Einzelbewertung

Die Lösungen aller Übungsblätter sind Einzelleistungen. Plagiieren von Lösungen führt zum Abzug von 25 Punkten bei allen am Plagiat Beteiligten.
Beispiel: Herr Knöpfle lässt Frau Spätzle Aufgabe 2 abschreiben, ansonsten bearbeiten sie Übungsblatt 1 aber getrennt. Beide erhalten nun 25 Minuspunkte sowie keine Punkte für die übrigen Aufgaben von Blatt 1.

Hinweis zu Aktualisierungen des Übungsblatts

Verfolgen Sie Nachrichten in ILIAS und prüfen Sie es regelmäßig auf Aktualisierungen und Korrekturen des aktuellen Übungsblatts.