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 3

Plug-Ins, Aktivitäts-, Sequenz-, Zustandsdiagramm und Substitutionsprinzip

Ausgabe: Mittwoch, 22. Mai 2019
Abgabe: Mittwoch, 5. Juni 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: Einschub-Architektur (4 Punkte)

Seit der letzten Projekt-Sitzung mit dem Software-Architekten ist Ihre Projektleiterin der Meinung, dass neue Funktionalitäten in iMage (siehe z.B. Blatt 2) unbedingt als Einschub (Plug-Ins) entwickelt werden sollten. Sie wissen, dass eine Anwendung zunächst vorbereitet werden muss, bevor Einschübe integriert werden können. Sie machen sich sofort ans Werk und gehen dabei wie folgt vor:

a) Informieren Sie sich über die ServiceLoader-API von Java: https://docs.oracle.com/javase/tutorial/ext/basics/spi.html
b) Holen Sie sich die Vorlage aus ILIAS und fügen Sie sie in Ihr JMJRST-Modul ein.
c) Implementieren Sie in der Klasse `PluginForJMJRST` die nötigen Methoden, um Einschübe nach der Länge ihres Klassennamens vergleichbar zu machen.
d) Implementieren Sie die Methode des `PluginManagements`, welche die verfügbaren Einschübe ermittelt und gemäß der Reihenfolge der Klassennamen, beginnend mit dem kürzesten, zurückgibt.
e) Fügen Sie JMJRST zwei neue Menüs hinzu, in denen später die Einschübe gestartet bzw. konfiguriert werden können. Die Menüs sollen als neue Untermenüs des „Options“-Menüs über dem Trennstrich (und unter dem Menüpunkt „Option“) eingefügt werden und die Namen „Start Plug-In“ und „Configure Plug-In“ tragen. Trennen Sie die Menüs mit einem weiteren Trennstrich vom Menüpunkt „Option“ ab.
f) Fügen Sie die verfügbaren Einschübe gemäß der in Aufgabenteil d) definierten Ordnung in das „Start Plug-In“- und das „Configure Plug-In“-Untermenü ein (beginnend oben mit dem Einschub, mit dem kürzesten Namen). Einschübe, die nicht konfiguriert werden können, sollen nicht im „Configure Plug-In“-Untermenü aufgeführt werden. Durch Klicken auf die entsprechenden Einträge soll die jeweilige Funktion des Einschubs aufgerufen werden.

Führen Sie die üblichen Plausibilitätstests vor der Abgabe durch.

Aufgabe 2: Instagrim-Einschub für iMage (4 Punkte)

Der Vorstand sieht das Projekt iMage als vollen Erfolg an (und rechnet diesen natürlich ausschließlich Ihrer Projektleiterin zu). Nun gilt es, die Verbreitung für die Zwecke der Pear Corp. zu nutzen. Ein hochrangiges Vorstandsmitglied hat auf einer Cocktail-Party gehört, dass man die Meinungsbilder heutzutage sehr einfach mit sogenannten „Bots“ steuern kann. Nun schwebt ihm vor, von Bots automatisch positive Kommentare generieren zu lassen und von iMage generierte Bilder automatisch damit zu versehen, um die Außenwahrnehmung von iMage noch positiver zu gestalten. Sie bekommen die Aufgabe, einen Prototypen zu erstellen, der automatisch Freitext-Kommentare erzeugt, welche iMage (subtil) loben. Das Vorstandsmitglied gibt Ihnen als Beispiele für solche Kommentare noch folgende mit auf den Weg: „Wow, scharfes Pic. Mit iMage erstellt?“ oder „Wusste gar nicht, dass sowas mit iMage möglich ist. Voll krasse Farben.“ Sie implementieren den Prototyp als Einschub und testen somit gleichzeitig die in Aufgabe 1 erstellte Einschub-Infrastruktur:

a) Erstellen Sie ein neues Modul namens `iMage.Instagrim-plugin` für einen Demonstrations-Prototypen eines Einschubs, welcher die grundlegenden meinungsbildenden Möglichkeiten von iMage zeigt. Der Einschub soll die abstrakte Klasse `iMage.PluginForJmjrst` beerben und die nötigen Methoden implementieren.
b) Fügen Sie der `pom.xml` eine Abhängigkeit zu Ihrem JMJRST-Modul hinzu (dann können Sie auf die benötigten Schnittstellen zugreifen) und denken Sie an die üblichen benötigten Konfigurationsdateien.
   Damit Maven die Abhängigkeit zu Ihrem Projekt erfassen kann, müssen Sie die Lösung von Aufgabe 1 in Ihr lokales Repository installieren (mit `mvn install`). Falls dies hartnäckig scheitert, können Sie stattdessen eine Abhängigkeit zum Projekt mit der ArtifactID `jmjrst-plugin-helper`, der GroupID `edu.kit.ipd.swt1` und der Version `0.0.2-SNAPSHOT` in Ihre `pom.xml` hinzufügen.
c) Das Plug-In soll beim Initialisieren auf der Standard-Fehlerausgabe folgendes ausgeben: „iMage: Der Bildverschönerer, dem Influencer vertrauen! Jetzt bist auch Du Teil unseres Teams, “. An diesen Text soll noch der Benutzername des aktuell angemeldeten Benutzers angehängt werden.
d) Der Einschub soll beim Konfigurieren ein Fenster öffnen und darin alle vorhandenen Freitext-Kommentare auflisten (siehe Aufgabenteil e)). Sorgen Sie dafür, dass die Kommentare zeilenweise angezeigt werden.
e) Der Einschub soll beim Ausführen auf der Standardausgabe einen Freitext-Kommentar ausgeben. Dieser soll zufällig aus einer Menge von mindestens sechs Kommentaren gezogen werden.
f) Verpacken Sie Ihr Plug-In mit Maven entsprechend der ServiceLoader-API in ein JAR. Erzeugen Sie die benötigte Service-Datei, indem Sie in die `pom.xml` das folgende Projekt eintragen und entsprechend der Dokumentation verwenden: http://metainf-services.kohsuke.org/

Führen Sie die üblichen Plausibilitätstests vor der Abgabe durch.

Aufgabe 3: iMage-Bundle (2 Punkte)

Ihre Projektleiterin ist begeistert von der Aufteilung des iMage-Projektes in einzelne Teilmodule. „Teile und herrsche, war schon immer mein Credo!“, stellt sie fest. „Man muss aber stets darauf bedacht sein, die delegierten Teile auch zusammenzufügen!“, belehrt sie Sie. Sie erahnen, wohin dieser Monolog führt, und beginnen noch während sie redet, proaktiv damit, ein neues Modul anzulegen, das die Module zusammenfasst, die ausgeliefert werden können.

a) Legen Sie ein neues Maven-Modul namens `iMage.bundle` mit den üblichen Einstellungen und Dateien für ein Java-Projekt an.
b) Tragen Sie als Abhängigkeit die JMJRST-Version aus Aufgabe 1 dieses Blatts ein sowie den Instagrim-Einschub aus Aufgabe 2.
c) Implementieren Sie nun die `main()`-Methode der `App`-Klasse so, dass JMJRST gestartet wird.
   Tragen Sie die `App`-Klasse als Hauptklasse im Manifest ein und lassen Sie Maven die Einstellungen zum Klassenpfad ins Manifest schreiben (siehe Blatt 2). Damit es aufgrund der von uns vorgegebenen ArtifactID nicht zu Konflikten kommt (denn diese ist ja für alle Module gleich), setzen Sie zudem die Option `classpathLayoutType` auf `repository`.
   Konfigurieren Sie das copy-dependencies-Plugin so, dass die Abhängigkeiten passend in das Verzeichnis `target` kopiert werden, wenn Sie `mvn package` ausführen. Anschließend sollten Sie die JAR-Datei ohne weitere Angaben mit `java -jar 1234567.jar` ausführen können (natürlich mit Ihrer Matrikelnummer und ohne die Bibliotheken aufzuführen) und in der GBO von JMJRST den Instagrim-Einschub benutzen können.

Führen Sie die üblichen Plausibilitätstests vor der Abgabe durch.

Aufgabe 4: UML-Aktivitätsdiagramm (6 Punkte)

Die Geschäftsführung ist äußerst zufrieden mit Ihrer Umsetzung von HDrize. Als Sie in diesem Zuge nach einer Gehaltserhöhung fragen, antwortet Ihnen der stellvertretende Geschäftsführer, er finde zwar die berechneten Bilder sehr hübsch, habe aber noch nicht verstanden, wie ein Nutzer HDrize jetzt benutzt und vor allem wie die Pear Corp. hiermit Geld verdiene. Sie erinnern sich an das von Ihnen ausgearbeitet Lastenheft, verschweigen aber im Sinne Ihrer Altervorsorge, dass Sie diese Information eigentlich bereits dort dargelegt haben. Um den Ablauf diesmal nachhaltig zu verdeutlichen, entscheiden Sie sich für eine verständlichere Darstellung und passen das Szenario dem momentanen Projektstand an:

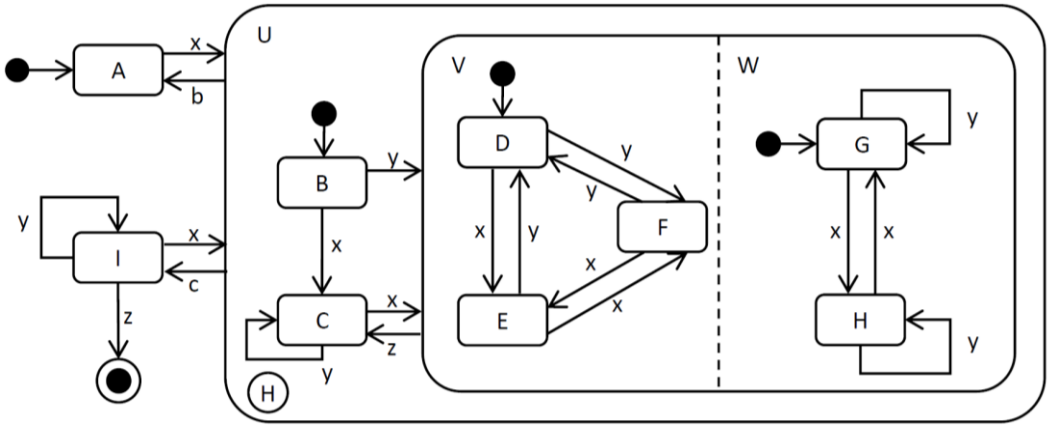
> *Zu Beginn zeigt die Applikation die Startseite an. Der Nutzer kann sich per Klick auf ein HDR-Bild von der Applikation eine Vorschau erzeugen und anzeigen lassen. Daraufhin betätigt der Nutzer die „Teilen“-Schaltfläche, um das Bild auf Facezine hochzuladen. Dies sorgt dafür, dass das entsprechende Bild auf dem Pear-Corp-Zentralserver gesucht und von dort an Facezine gesendet wird. Gleichzeitig mit dem Senden kehrt die Applikation auf die Startseite zurück. Über das Klicken der Schaltfläche „Neues HDR-Bild“ gelangt der Nutzer zu einer Übersicht der auf dem Mobiltelefon gespeicherten Bilder. Der Nutzer wählt drei Bilder einer Bildreihe aus und bestätigt seine Auswahl mit einem Klick auf „HDR-Bild erstellen“.*
> *Die Eingabebilder werden anschließend auf den Pear-Corp-Zentralserver hochgeladen und das HDR-Bild erzeugt. Zur Erzeugung des HDR-Bildes rekonstruiert der Server zunächst die Antwortkurve der Kamera der Eingabebilder. Daraufhin kombiniert er die Eingabebilder zu einem HDR-Bild. Hierzu iteriert der Server über die Pixel der Bilder und bestimmt für jeden Pixel den neuen Farbwert. Dazu kombiniert er zunächst die gewichteten Einzelfarbwerte mit der Antwortkurve. Anschließend wird über die Summe der Gewichte normiert. Wurde über alle Pixel iteriert, bildet der Server das Bild in den RGB-Farbraum ab und speichert sowohl das HDR- als auch das RGB-Bild auf dem Server. Die Applikation wartet, solange die Erzeugung nicht beendet ist. Per Klick auf die Schaltfläche „HDR-Bild abholen“ wird dem Nutzer das Vorschaubild und Kaufoptionen angezeigt. Hat der Nutzer eine Kaufoption gewählt, speichert die Applikation das Bild im Bilderordner des Mobiltelefons. Besitzt der Nutzer bereits ein Monatsabonnement, sollen keine Kaufoptionen, sondern direkt eine Schaltfläche „HDR-Bild jetzt speichern“ angezeigt werden.*

Zur Darstellung des Ablaufs kommen Ihnen UML-Aktivitätsdiagramme in den Sinn. Setzen Sie die obenstehende Szenario-Beschreibung als Aktivitätsdiagramm um.

Hinweis: Verwenden Sie bei Ihrer Modellierung korrekte UML-Notation. Modellieren Sie mindestens das, was in der Beschreibung dargestellt wird. Verwenden Sie – falls nötig – sinnvolle Partitionen. Sie können den Ablauf der Kombination der Eingabebilder in ein separates Diagramm auslagern. Sie dürfen das Diagramm gerne sinnvoll erweitern.

Aufgabe 5: UML-Zustandsdiagramm (5 Punkte)

Gegeben sei folgendes UML-Zustandsdiagramm:



Geben Sie an in welchem Zustand sich der Automat nach der folgenden Eingabe befindet. Geben Sie außerdem die durchlaufenen Zustände an:

a) x, y, x, y, y, z, b, x, y, c, y, x
b) x, x, x, x, x, z, x, y, x, x
c) Wandeln Sie den Automaten in einen äquivalenten Automaten ohne Historie um.

Aufgabe 6: UML-Sequenzdiagramm (6 Punkte)

Der große Erfolg von HDrize führt dazu, dass die Geschäftsführung der Pear Corp. beschließt dem Projekt weitere Mitarbeiter zuzuteilen. Als Teamleiter ist es Ihre Aufgabe, neue Mitarbeiter mit der Implementierung von HDrize bekannt zu machen. Sie entschließen sich ein UML-Sequenzdiagramm zu erstellen, um neuen Mitarbeiter leichter mit den Aufrufreihenfolgen der bestehenden Implementierung vertraut zu machen. Da Sie selbst nur (andere) einzelne Bestandteile implementiert haben (siehe Übungsblatt 2, Aufgabe 4), lassen Sie sich von Ihren Team-Mitgliedern eine Beschreibung geben:

> *Die Kommandozeilen-Applikation (Klasse* `App`*) ruft in der Klasse* `HDrize` *die* `createRGB`*-Methode (*`createRGB(EnhancedImage[] enhancedImages, int samples, double lambda, IMatrixCalculator<Matrix> mtxCalc, ToneMapping mapping)`*) auf. Diese Methode ruft wiederum die* `createHDR(…)`*-Methode (ebenfalls in* `HDrize`*) mit den entsprechenden Parametern auf. In* `createHDR(…)` *wird zunächst die Methode* `createCurve(…)` *(auch in* `HDrize`*) aufgerufen. Diese erzeugt eine neue Instanz der Klasse* `CameraCurve`*. Anschließend wird auf dieser Instanz die Methode* `calculate(…)` *mit den entsprechenden Parametern aufgerufen. Dann wird auf der gleichen Instanz* `isCalculated()` *aufgerufen, um zu prüfen, ob die Kurve berechnet wurde. Wurde die Kurve berechnet, wird die Instanz an* `createHDR(…)` *zurückgeliefert, andernfalls wird* `null` *zurückgeliefert.* `createHDR(…)` *ruft anschließend die Methode* `createHDR(…)` *in der Klasse* `HDRCombine` *auf. Diese liefert ein* `HDRImage` *zurück. Das erhaltene HDR-Bild wird anschließend an die* `createRGB(…)`*-Methode zurückgeliefert. Dort wird die statische Methode* `createRGB(…)` *in der Klasse* `HDRImageIO` *aufgerufen, die ein* `BufferedImage`

zurückliefert. Dieses wird von der Methode `createRGB(…)` *in* `HDrize` *wiederum an die Kommandozeile zurückgeliefert.*

Erstellen Sie anhand der obigen Beschreibung ein UML-Sequenzdiagramm.

Hinweis: Sie können (oder sollten) zur Bearbeitung der Aufgabe den Quelltext des Moduls `image.HDrize` zurate ziehen. Modellieren Sie aber nur die in der Beschreibung gegebenen Aufrufe (d.h. ignorieren Sie Überprüfungen auf null und ähnliches). Außerdem können Sie davon ausgehen, dass bereits eine Instanz der Klasse `HDrize` (und damit auch der Klasse `HDRCombine`) erzeugt wurde.

Aufgabe 7: Augen auf bei der Bibliothekswahl (3 Bonuspunkte)

Eine Entwicklerin Ihres Teams hat vorgeschlagen zur einfacheren Fehlerbehebung und besserer Nachverfolgbarkeit des Programmablaufs einen Logger einzusetzen und hat bereits eine passende Logging-Bibliothek gefunden. Allerdings ist sie sich nicht ganz sicher, „[...] ob die nicht irgendwie kaputt ist“ und fragt Sie um Rat. Sie hat bereits die Stellen, die möglicherweise nicht in Ordnung sind, zusammenkopiert:

Klasse `Logger`:

```
01: public class Logger {
02:     Layout layout;
03:
04:     public void setLayout(Layout layout) { this.layout = layout; }
05:
06:     public Layout getLayout() { return layout; }
07: }
```

Klasse `AdvancedLogger`:

```
08: public class AdvancedLogger extends Logger {
09:   AdvancedLayout layout = new AdvancedLayout();
10:
11:   @Override public AdvancedLayout getLayout() {
12:     // we are sure, that we've got an AdvancedLayout at hand!
13:     return layout;
14:   }
15:
16:   @Override public void setLayout(Layout layout) {
17:     // ignore the setter, we use AdvancedLayouts always!
18:   }
19: }
```

Klasse `Layout`:

```
20: public class Layout {
21:   // creates the layout of the logger output
22: }
```

Klasse `AdvancedLayout`:

```
23: public class AdvancedLayout extends Layout {
24:   // adds some "sugar" to the logger output
25: }
```

Bonusaufgabe: Verletzt die obenstehende Implementierung einer Logging-Bibliothek das Liskov’sche Substitutionsprinzip? Begründen Sie Ihre Antwort!

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: Piggeldy lässt Frederick 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.