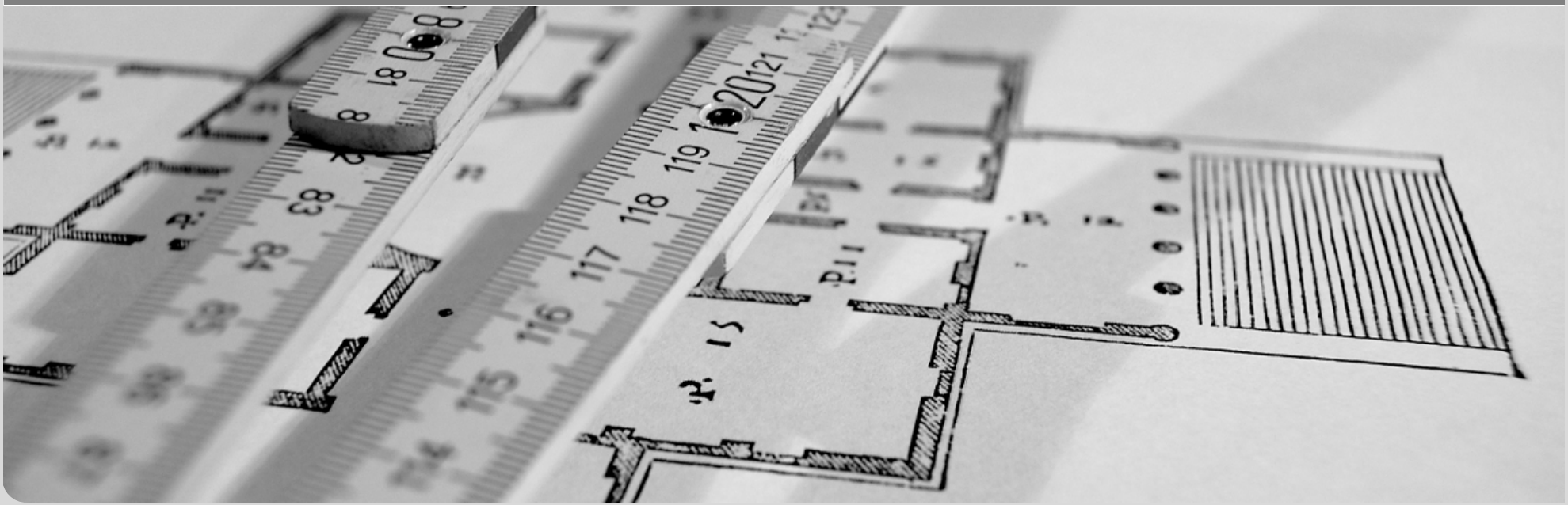
Karlsruher Institut für Technologie

Tutorium 01: Projektplanung

Softwaretechnik im SS 2011, Tut 17

Christian Jülg | 5. Mai 2011

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Was machen wir heute?

1. Organisatorisches
   - Vorstellung
   - Organisatorisches
2. Altes Übungsblatt
   - Altes Übungsblatt
   - Zum Aufwärmen ...
   - Pflichtenheft
3. Werkzeuge
   - Versionsverwaltungen
   - Checkstyle
   - Eclipse
4. UML
   - Klassendiagramm
   - Aktivitätsdiagramm
5. Ende
   - Tipps zum nächsten Übungsblatt

Wer bin ich?

- Christian Jülg
- 14-tes Semester Informatik
- swt-tutor@gmx.de
- auf `http://swt1-tut.blogspot.com/` werde ich Folien und evtl. hilfreiche Links zum Tutorium veröffentlichen
- Partnerturoren Jürgen Walter und Daniel Deckers
- ...

Wer bin ich?

- Christian Jülg
- 14-tes Semester Informatik
- swt-tutor@gmx.de
- auf `http://swt1-tut.blogspot.com/` werde ich Folien und evtl. hilfreiche Links zum Tutorium veröffentlichen
- Partnerturoren Jürgen Walter und Daniel Deckers
- ...

Wer bin ich?

- Christian Jülg
- 14-tes Semester Informatik
- swt-tutor@gmx.de
- auf `http://swt1-tut.blogspot.com/` werde ich Folien und evtl. hilfreiche Links zum Tutorium veröffentlichen
- Partnerturoren Jürgen Walter und Daniel Deckers
- ...

Wer bin ich?

- Christian Jülg
- 14-tes Semester Informatik
- swt-tutor@gmx.de
- auf `http://swt1-tut.blogspot.com/` werde ich Folien und evtl. hilfreiche Links zum Tutorium veröffentlichen
- Partnerturoren Jürgen Walter und Daniel Deckers
- ...

Wer bin ich?

- Christian Jülg
- 14-tes Semester Informatik
- swt-tutor@gmx.de
- auf `http://swt1-tut.blogspot.com/` werde ich Folien und evtl. hilfreiche Links zum Tutorium veröffentlichen
- Partnerturoren Jürgen Walter und Daniel Deckers
- ...

Organisatorisches

Übungsschein . . .

- ist keine Vorraussetzung zur Klausur, aber Vorraussetzung für Modul!
- hat 6 Übungsblätter mit insgesamt 150 Punkten
- ist mit 50 Prozent aus Übungsblättern und Programmmieraufgaben bestanden

Tutorium . . .

- findet alle 2 Wochen statt

Organisatorisches

Übungsschein . . .

- ist keine Vorraussetzung zur Klausur, aber Vorraussetzung für Modul!
- hat 6 Übungsblätter mit insgesamt 150 Punkten
- ist mit 50 Prozent aus Übungsblättern und Programmmieraufgaben bestanden

Tutorium . . .

- findet alle 2 Wochen statt

Organisatorisches

Übungsschein . . .

- ist keine Vorraussetzung zur Klausur, aber Vorraussetzung für Modul!
- hat 6 Übungsblätter mit insgesamt 150 Punkten
- ist mit 50 Prozent aus Übungsblättern und Programmmieraufgaben bestanden

Tutorium . . .

- findet alle 2 Wochen statt

Organisatorisches

Übungsschein . . .

- ist keine Vorraussetzung zur Klausur, aber Vorraussetzung für Modul!
- hat 6 Übungsblätter mit insgesamt 150 Punkten
- ist mit 50 Prozent aus Übungsblättern und Programmmieraufgaben bestanden

Tutorium . . .

- findet alle 2 Wochen statt

Organisatorisches

Übungsschein . . .

- ist keine Vorraussetzung zur Klausur, aber Vorraussetzung für Modul!
- hat 6 Übungsblätter mit insgesamt 150 Punkten
- ist mit 50 Prozent aus Übungsblättern und Programmmieraufgaben bestanden

Tutorium . . .

- findet alle 2 Wochen statt

Altes Übungsblatt

Aufgabe 1: Mailingliste

- ...

Aufgabe 2: Lastenheft

1. Zielbestimmung
2. Produkteinsatz
3. Funktionale Anforderungen
4. Produktdaten
5. Nichtfunktionale Anforderungen
6. Systemmodelle
   - Szenarien
   - Anwendungsfälle
7. Glossar (Begriffslexikon zur Beschreibung des Produktes)

Aufgabe 3 Durchführbarkeitsuntersuchung

- jeden der 6 Aspekte ansprechen
- “Probleme durch ... treten nicht auf, **da ...**” ist auch eine gute Antwort

Aufgabe 4 Hans Olo

- **Benutzt Checkstyle!**

Aufgabe 5 Vorbereitung der Programmieraufgabe

- hat jeder das Projekt runter geladen?

Wahr oder falsch?

- In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht
- Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll
- In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert
- Das Lastenheft ist eine Verfeinerung des Pflichtenheftes
- Im Pflichtenheft steht beschrieben, wie etwas zu implementieren ist. Es werden z.B. Algorithmen und Datenstrukturen beschrieben

Wahr oder falsch?

- In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht
- Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll
- In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert
- Das Lastenheft ist eine Verfeinerung des Pflichtenheftes
- Im Pflichtenheft steht beschrieben, wie etwas zu implementieren ist. Es werden z.B. Algorithmen und Datenstrukturen beschrieben

Wahr oder falsch?

- In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht
- Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll
- In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert
- Das Lastenheft ist eine Verfeinerung des Pflichtenheftes
- Im Pflichtenheft steht beschrieben, wie etwas zu implementieren ist. Es werden z.B. Algorithmen und Datenstrukturen beschrieben

Wahr oder falsch?

- In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht
- Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll
- In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert
- Das Lastenheft ist eine Verfeinerung des Pflichtenheftes
- Im Pflichtenheft steht beschrieben, wie etwas zu implementieren ist. Es werden z.B. Algorithmen und Datenstrukturen beschrieben

Wahr oder falsch?

- In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht
- Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll
- In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert
- Das Lastenheft ist eine Verfeinerung des Pflichtenheftes
- Im Pflichtenheft steht beschrieben, wie etwas zu implementieren ist. Es werden z.B. Algorithmen und Datenstrukturen beschrieben

Wahr oder falsch?

- In der Planungsphase wird die softwaretechnische Realisierbarkeit eines Produktes untersucht
- Ein Pflichtenheft beschreibt die Eigenschaften, die das Produkt aus der Sicht des Kunden erfüllen soll
- In einem UML-Anwendungsfalldiagramm werden typische Interaktionen des Benutzers mit dem System modelliert
- Das Lastenheft ist eine Verfeinerung des Pflichtenheftes
- Im Pflichtenheft steht beschrieben, wie etwas zu implementieren ist. Es werden z.B. Algorithmen und Datenstrukturen beschrieben

Pflichtenheft

- Das Pflichtenheft ist eine Verfeinerung des Lastenheftes
- Das Pflichtenheft beschreibt nicht, wie, sondern nur **was** zu implementieren ist.
  $\Rightarrow$ es werden weder Algorithmen noch Datenstrukturen festgelegt
- das Pflichtenheft definiert das Projekt so vollständig und exakt, dass Entwickler das System implementieren können, ohne nachfragen oder raten zu müssen, **was** zu implementieren ist.

Pflichtenheft

- Das Pflichtenheft ist eine Verfeinerung des Lastenheftes
- Das Pflichtenheft beschreibt nicht, wie, sondern nur **was** zu implementieren ist.

  $\Rightarrow$ es werden weder Algorithmen noch Datenstrukturen festgelegt
- das Pflichtenheft definiert das Projekt so vollständig und exakt, dass Entwickler das System implementieren können, ohne nachfragen oder raten zu müssen, **was** zu implementieren ist.

Pflichtenheft

- Das Pflichtenheft ist eine Verfeinerung des Lastenheftes
- Das Pflichtenheft beschreibt nicht, wie, sondern nur **was** zu implementieren ist.
  $\Rightarrow$ es werden weder Algorithmen noch Datenstrukturen festgelegt
- das Pflichtenheft definiert das Projekt so vollständig und exakt, dass Entwickler das System implementieren können, ohne nachfragen oder raten zu müssen, **was** zu implementieren ist.

Pflichtenheft

- Das Pflichtenheft ist eine Verfeinerung des Lastenheftes
- Das Pflichtenheft beschreibt nicht, wie, sondern nur **was** zu implementieren ist.
  $\Rightarrow$ es werden weder Algorithmen noch Datenstrukturen festgelegt
- das Pflichtenheft definiert das Projekt so vollständig und exakt, dass Entwickler das System implementieren können, ohne nachfragen oder raten zu müssen, **was** zu implementieren ist.

Versionsverwaltungen

Subversion

- von der Vorlesung unterstützte Versionsverwaltung
- Windows: Tortoise SVN
- Linux: Shell oder RabbitVCS
- Mac: Shell

Versionsverwaltungen...

Git

- Werkzeug des Tutors ;-)
- dezentrales VCS: jeder Benutzer hat lokal ein vollständiges Repository
- hat mehr Funktionen als SVN, aber auch mehr Möglichkeiten sich selbst in den Fuß zu schießen...
- Verschmelzen verschiedener Zweige und Versionen ist hier eher Normalfall als Ausnahme

weitere

- Es gibt noch viele weitere frei verfügbare dezentrale VCS
- z.B. Mercurial (Hg)

Klausur 2009

Aufgabe

Erklären Sie die beiden Begriffe „Striktes Ausbuchen“ und „Optimistisches Ausbuchen“ im Kontext einer Konfigurationsverwaltung. Nennen Sie jeweils einen Vor- sowie einen Nachteil. (4P)

Striktes Ausbuchen

Optimistisches Ausbuchen

Klausur 2009

Aufgabe

Erklären Sie die beiden Begriffe „Striktes Ausbuchen“ und „Optimistisches Ausbuchen“ im Kontext einer Konfigurationsverwaltung. Nennen Sie jeweils einen Vor- sowie einen Nachteil. (4P)

Striktes Ausbuchen

- Nur eine Ausbuchung gleichzeitig ist erlaubt
- Ausbucher hat exklusives Änderungsrecht
- Vorteil: kein Verschmelzungsaufwand beim Zurückschreiben
- Nachteil: immer nur einer kann eine Version ändern

Optimistisches Ausbuchen

Klausur 2009

Aufgabe

Erklären Sie die beiden Begriffe „Striktes Ausbuchen“ und „Optimistisches Ausbuchen“ im Kontext einer Konfigurationsverwaltung. Nennen Sie jeweils einen Vor- sowie einen Nachteil. (4P)

Striktes Ausbuchen

- Nur eine Ausbuchung gleichzeitig ist erlaubt
- Ausbucher hat exklusives Änderungsrecht
- Vorteil: kein Verschmelzungsaufwand beim Zurückschreiben
- Nachteil: immer nur einer kann eine Version ändern

Optimistisches Ausbuchen

- Mehrere Ausbuchungen gleichzeitig erlaubt
- Mehrere Entwickler Arbeiten an der gleichen Programmversion
- Vorteil: Mehrere Entwickler können eine Version ändern
- Nachteil: Aufwand beim Zusammenführen der Versionen (der Schnellere gewinnt)

Checkstyle

Das geht besser . . .

- Fast keiner von euch ist ohne Checkstyle Fehler

private Konstruktoren und utility classes

- Fehlermeldung:
  “utility classes should not have a public or default constructor”
- ist alles in einer Klasse `static`, dann sollte man diese Klasse vermutlich nicht instanziieren
- durch einen privaten Konstruktor verhindert man das versehentliche Instanziieren:

```
private MyClass() {}
```

- eine solche “utility class” sollte als `final` markiert werden:

```
public final class MyClass {
```

Checkstyle

Das geht besser . . .

- Fast keiner von euch ist ohne Checkstyle Fehler

private Konstruktoren und utility classes

- Fehlermeldung:
  “utility classes should not have a public or default constructor”
- ist alles in einer Klasse `static`, dann sollte man diese Klasse vermutlich nicht instanziieren
- durch einen privaten Konstruktor verhindert man das versehentliche Instanziieren:

```
private MyClass() {}
```

- eine solche “utility class” sollte als `final` markiert werden:

```
public final class MyClass {
```

Checkstyle

Das geht besser . . .

- Fast keiner von euch ist ohne Checkstyle Fehler

private Konstruktoren und utility classes

- Fehlermeldung:
  “utility classes should not have a public or default constructor”
- ist alles in einer Klasse `static`, dann sollte man diese Klasse vermutlich nicht instanziieren
- durch einen privaten Konstruktor verhindert man das versehentliche Instanziieren:

```
private MyClass() {}
```

- eine solche “utility class” sollte als `final` markiert werden:

```
public final class MyClass {
```

Checkstyle

Das geht besser . . .

- Fast keiner von euch ist ohne Checkstyle Fehler

private Konstruktoren und utility classes

- Fehlermeldung:
  “utility classes should not have a public or default constructor”
- ist alles in einer Klasse `static`, dann sollte man diese Klasse vermutlich nicht instanziieren
- durch einen privaten Konstruktor verhindert man das versehentliche Instanziieren:

```
private MyClass() {}
```

- eine solche “utility class” sollte als `final` markiert werden:

```
public final class MyClass {
```

Checkstyle

Das geht besser . . .

- Fast keiner von euch ist ohne Checkstyle Fehler

private Konstruktoren und utility classes

- Fehlermeldung:
  “utility classes should not have a public or default constructor”
- ist alles in einer Klasse `static`, dann sollte man diese Klasse vermutlich nicht instanziieren
- durch einen privaten Konstruktor verhindert man das versehentliche Instanziieren:

```
private MyClass() {}
```

- eine solche “utility class” sollte als `final` markiert werden:

```
public final class MyClass {
```

Checkstyle -

weitere Fehler

- Fehlermeldung:
  “X sollte auf derselben/ der nächsten Zeile stehen”
  in Eclipse mit Autoformat beheben: `Ctrl + Shift + F`
- Fehlermeldung: “Javadoc unvollständig”
  vor jeder Klasse und jeder Methode ohne Kommentar `/**` schreiben und `Enter` drücken
- Fehlermeldung: “Importe einzeln angeben”
  in Eclipse Importe vervollständigen: `Ctrl + Shift + O`

Checkstyle -

weitere Fehler

- Fehlermeldung:
  “X sollte auf derselben/ der nächsten Zeile stehen”
  in Eclipse mit Autoformat beheben: `Ctrl + Shift + F`
- Fehlermeldung: “Javadoc unvollständig”
  vor jeder Klasse und jeder Methode ohne Kommentar `/**` schreiben und `Enter` drücken
- Fehlermeldung: “Importe einzeln angeben”
  in Eclipse Importe vervollständigen: `Ctrl + Shift + O`

Checkstyle -

weitere Fehler

- Fehlermeldung:
  “X sollte auf derselben/ der nächsten Zeile stehen”
  in Eclipse mit Autoformat beheben: `Ctrl + Shift + F`
- Fehlermeldung: “Javadoc unvollständig”
  vor jeder Klasse und jeder Methode ohne Kommentar `/**` schreiben und `Enter` drücken
- Fehlermeldung: “Importe einzeln angeben”
  in Eclipse Importe vervollständigen: `Ctrl + Shift + O`

Checkstyle -

weitere Fehler

- Fehlermeldung:
  “X sollte auf derselben/ der nächsten Zeile stehen”
  in Eclipse mit Autoformat beheben: `Ctrl + Shift + F`
- Fehlermeldung: “Javadoc unvollständig”
  vor jeder Klasse und jeder Methode ohne Kommentar `/**` schreiben und `Enter` drücken
- Fehlermeldung: “Importe einzeln angeben”
  in Eclipse Importe vervollständigen: `Ctrl + Shift + O`

Checkstyle -

weitere Fehler

- Fehlermeldung:
  “X sollte auf derselben/ der nächsten Zeile stehen”
  in Eclipse mit Autoformat beheben: `Ctrl + Shift + F`
- Fehlermeldung: “Javadoc unvollständig”
  vor jeder Klasse und jeder Methode ohne Kommentar `/**` schreiben und `Enter` drücken
- Fehlermeldung: “Importe einzeln angeben”
  in Eclipse Importe vervollständigen: `Ctrl + Shift + O`

Checkstyle -

weitere Fehler

- Fehlermeldung:
  “X sollte auf derselben/ der nächsten Zeile stehen”
  in Eclipse mit Autoformat beheben: `Ctrl + Shift + F`
- Fehlermeldung: “Javadoc unvollständig”
  vor jeder Klasse und jeder Methode ohne Kommentar `/**` schreiben und `Enter` drücken
- Fehlermeldung: “Importe einzeln angeben”
  in Eclipse Importe vervollständigen: `Ctrl + Shift + O`

Eclipse Tipps

einfaches Umbenennen

- es gibt eine einfache Möglichkeit, ein beliebiges Element eines Java Programms in Eclipse umzubenennen
- dazu markiert man das gewünschte Element und drückt `Alt + Shift + R` bzw. wählt “Refactor”>”Rename”

SVN in Eclipse

- sobald in Eclipse ein SVN Plugin installiert ist, könnt ihr darauf über “Rechtsklick, Team” zugreifen
- dort findet ihr “Commit”, “Update”, “Show History” und mehr

Local History

- Eclipse merkt sich alle Änderungen an Dateien, unabhängig von SVN
- gelöschte Datien findet man z.B. über “Rechtsklick auf Projekt”, “Restore from Local History”

Eclipse Tipps

einfaches Umbenennen

- es gibt eine einfache Möglichkeit, ein beliebiges Element eines Java Programms in Eclipse umzubenennen
- dazu markiert man das gewünschte Element und drückt `Alt + Shift + R` bzw. wählt “Refactor”>”Rename”

SVN in Eclipse

- sobald in Eclipse ein SVN Plugin installiert ist, könnt ihr darauf über “Rechtsklick, Team” zugreifen
- dort findet ihr “Commit”, “Update”, “Show History” und mehr

Local History

- Eclipse merkt sich alle Änderungen an Dateien, unabhängig von SVN
- gelöschte Datien findet man z.B. über “Rechtsklick auf Projekt”, “Restore from Local History”

Eclipse Tipps

einfaches Umbenennen

- es gibt eine einfache Möglichkeit, ein beliebiges Element eines Java Programms in Eclipse umzubenennen
- dazu markiert man das gewünschte Element und drückt `Alt + Shift + R` bzw. wählt “Refactor”>”Rename”

SVN in Eclipse

- sobald in Eclipse ein SVN Plugin installiert ist, könnt ihr darauf über “Rechtsklick, Team” zugreifen
- dort findet ihr “Commit”, “Update”, “Show History” und mehr

Local History

- Eclipse merkt sich alle Änderungen an Dateien, unabhängig von SVN
- gelöschte Datien findet man z.B. über “Rechtsklick auf Projekt”, “Restore from Local History”

Eclipse Tipps

einfaches Umbenennen

- es gibt eine einfache Möglichkeit, ein beliebiges Element eines Java Programms in Eclipse umzubenennen
- dazu markiert man das gewünschte Element und drückt `Alt + Shift + R` bzw. wählt “Refactor”>”Rename”

SVN in Eclipse

- sobald in Eclipse ein SVN Plugin installiert ist, könnt ihr darauf über “Rechtsklick, Team” zugreifen
- dort findet ihr “Commit”, “Update”, “Show History” und mehr

Local History

- Eclipse merkt sich alle Änderungen an Dateien, unabhängig von SVN
- gelöschte Datien findet man z.B. über “Rechtsklick auf Projekt”, “Restore from Local History”

Eclipse Tipps

einfaches Umbenennen

- es gibt eine einfache Möglichkeit, ein beliebiges Element eines Java Programms in Eclipse umzubenennen
- dazu markiert man das gewünschte Element und drückt `Alt + Shift + R` bzw. wählt “Refactor”>”Rename”

SVN in Eclipse

- sobald in Eclipse ein SVN Plugin installiert ist, könnt ihr darauf über “Rechtsklick, Team” zugreifen
- dort findet ihr “Commit”, “Update”, “Show History” und mehr

Local History

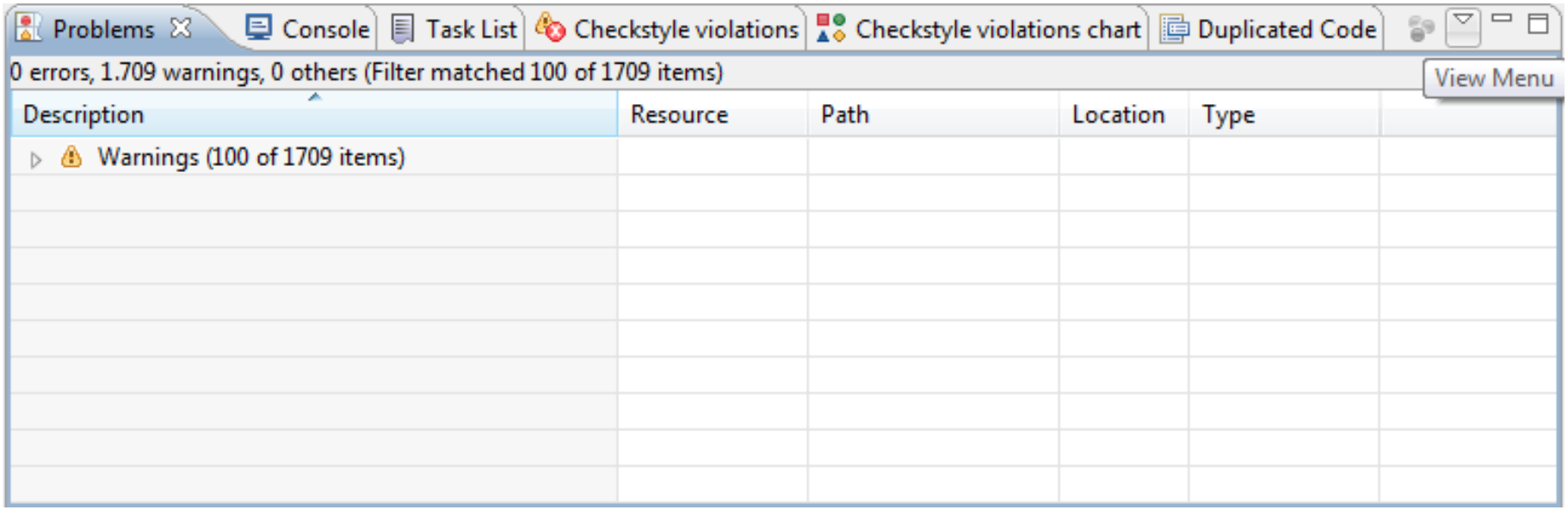
- Eclipse merkt sich alle Änderungen an Dateien, unabhängig von SVN
- gelöschte Datien findet man z.B. über “Rechtsklick auf Projekt”, “Restore from Local History”

Checkstyle in Eclipse

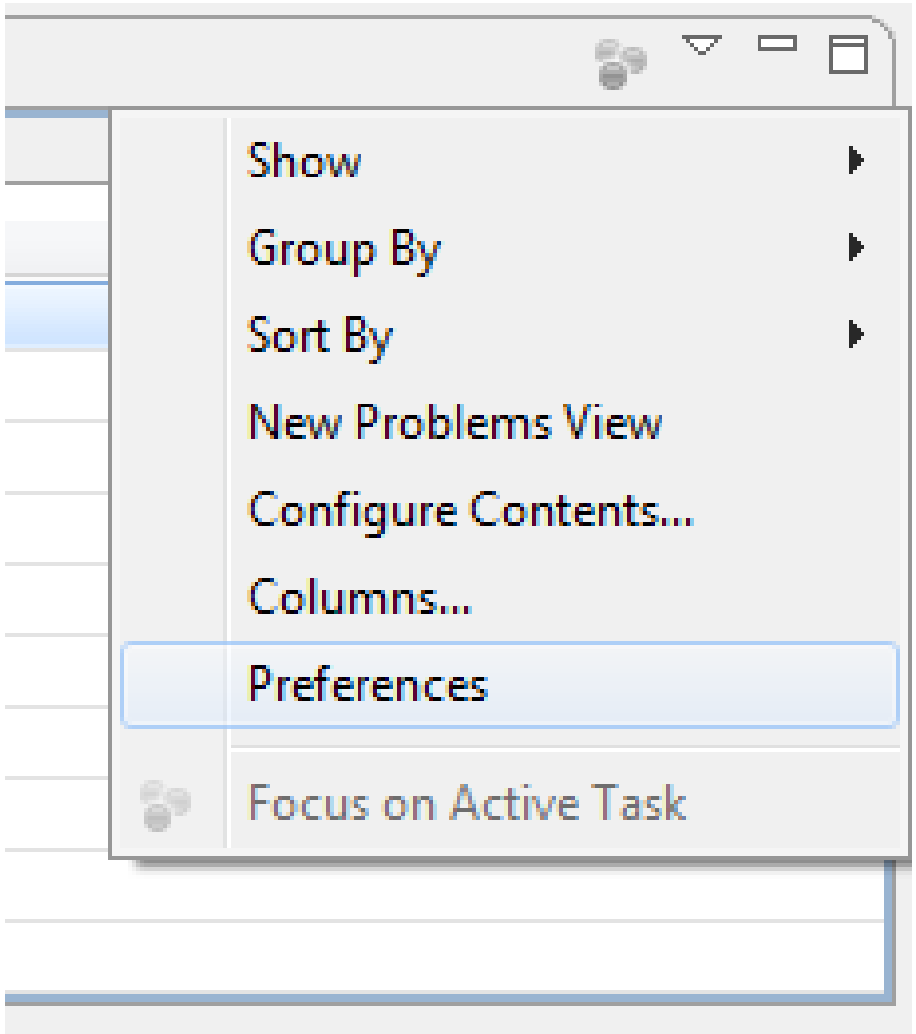
Ergebnisse anzeigen

- sobald Checkstyle richtig konfiguriert wurde, gibt es einige hilfreiche Übersichten
- `Window > Show View > Other > Checkstyle`
- hier gibt es `Checkstyle violations`, `Checkstyle violations chart` und `Duplication Code`

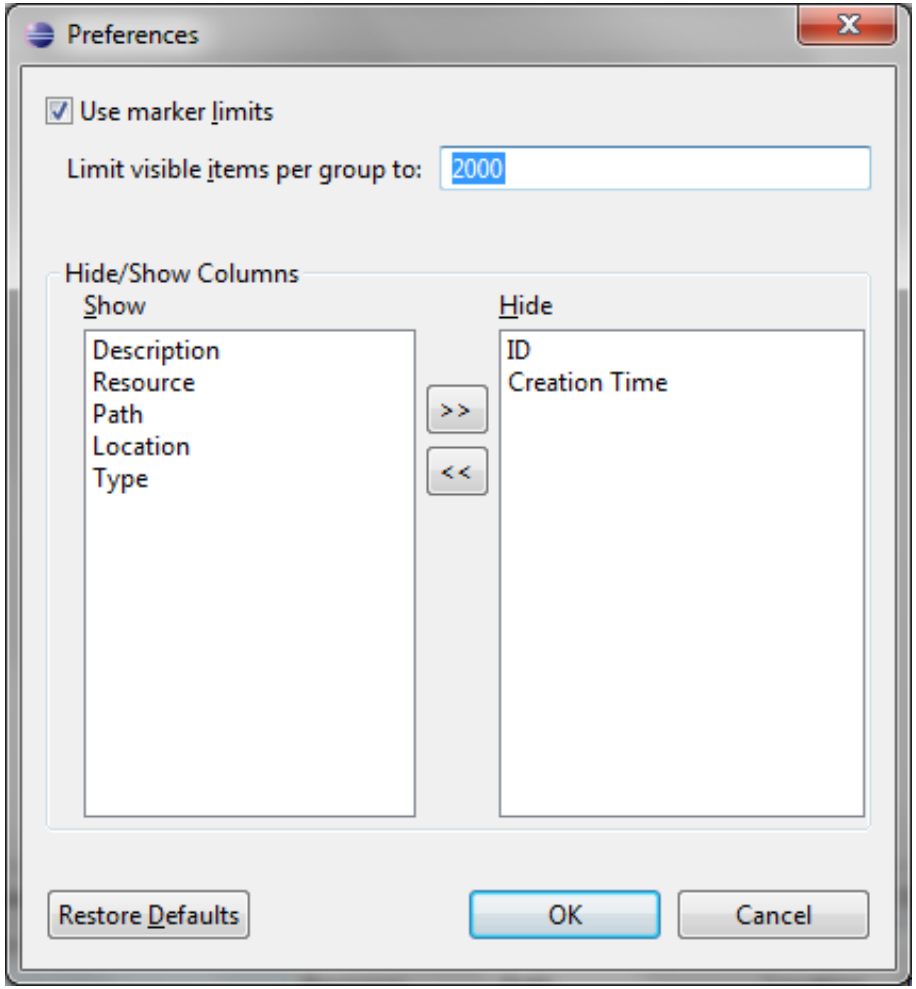
Eclipse - mehr Fehler anzeigen



Eclipse - mehr Fehler anzeigen



Eclipse - mehr Fehler anzeigen



UML

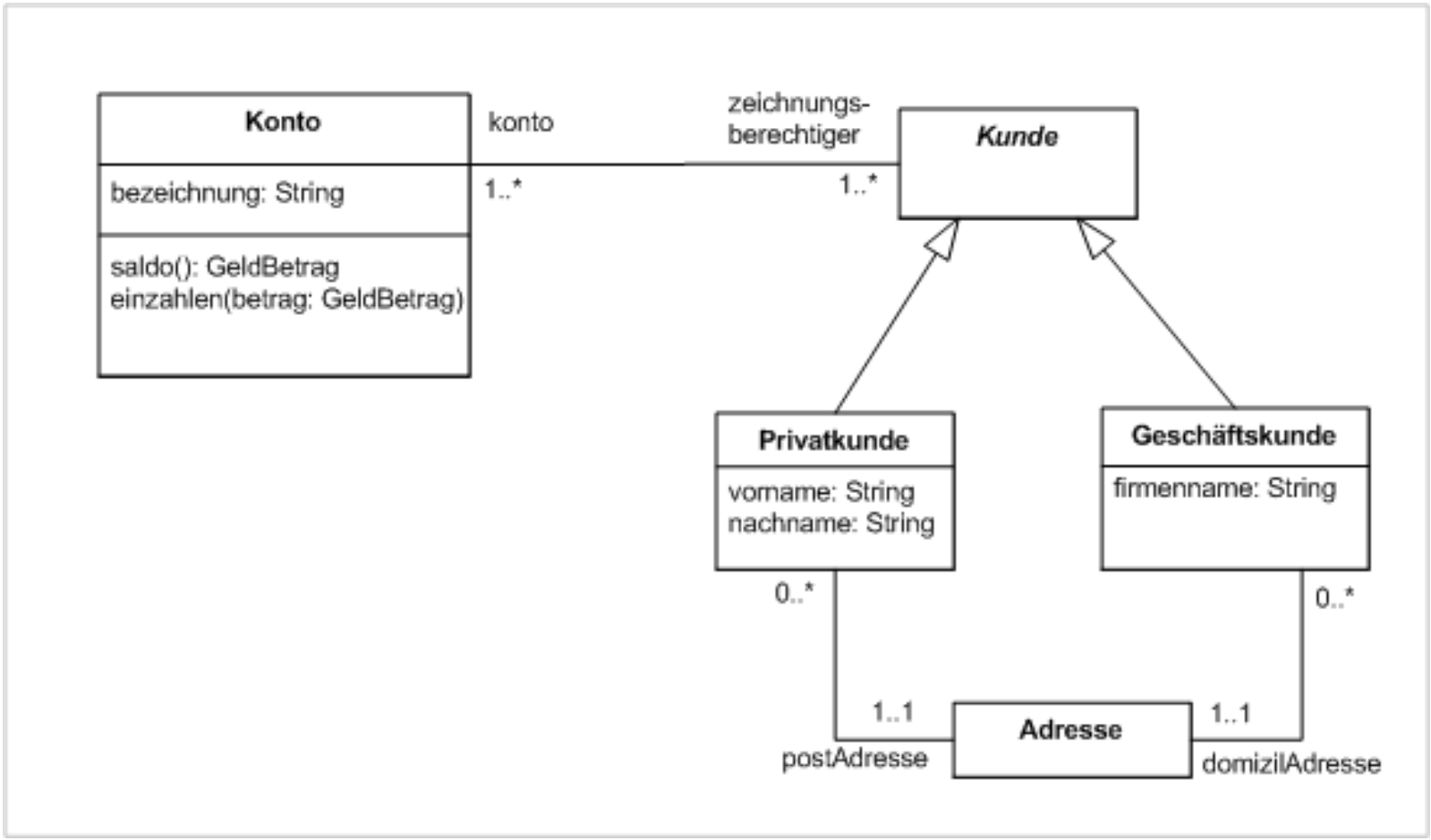
Klassendiagramm

- Was ist das?

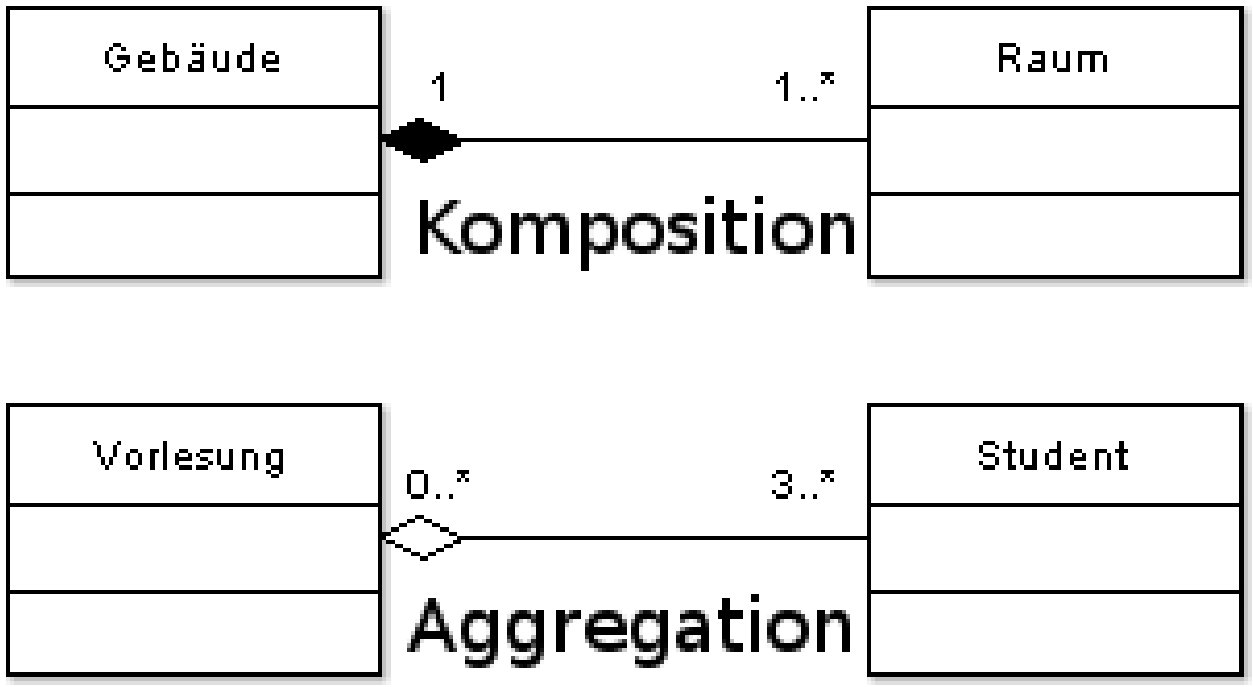
Aktivitätsdiagramm

- Was ist das?

Klassendiagramm: Erläuterung am Beispiel



Klassendiagramm: Komposition und Aggregation



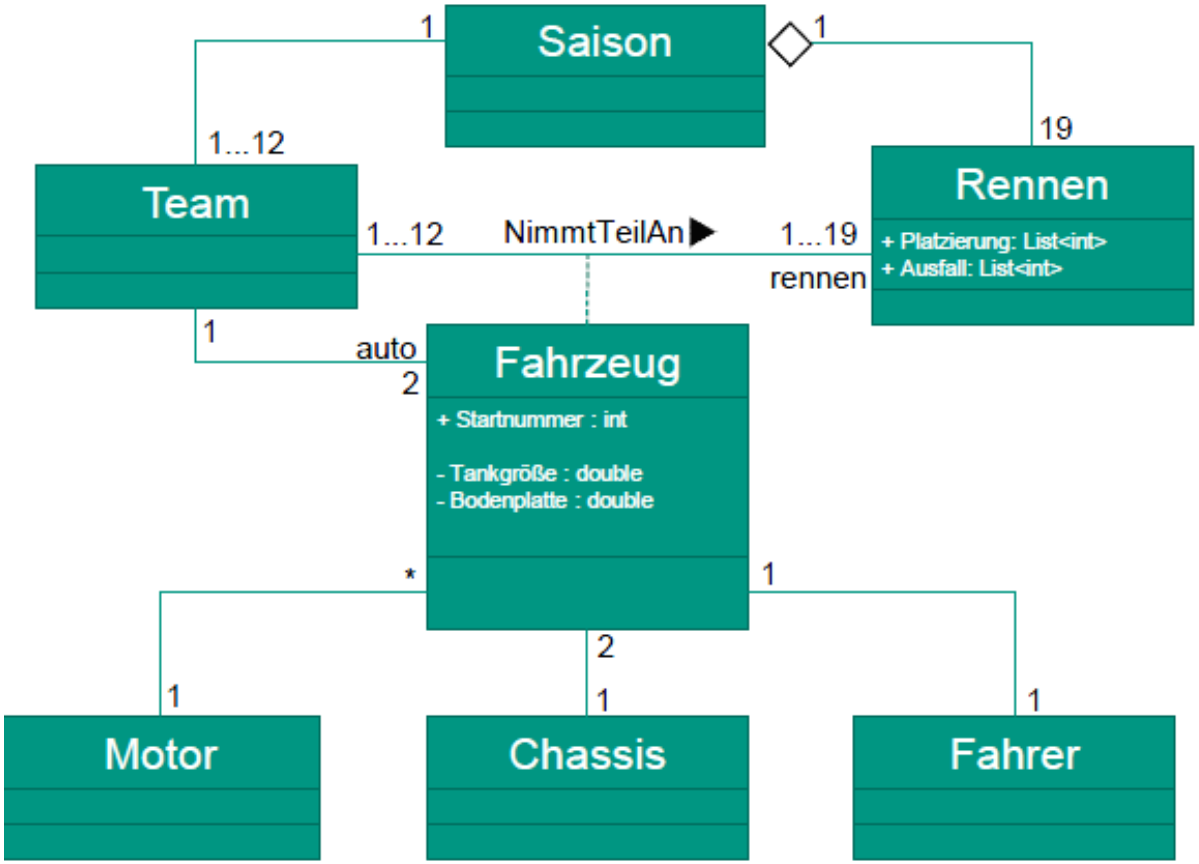
Aufgabe Klassendiagramm

Szenario

In einer Formel 1-Saison gibt es 12 Teams. Die Saison hat 19 Rennen und an jedem Rennen nehmen die Teams mit jeweils 2 Fahrzeugen teil. Während der Saison können Teams ausscheiden und von nachrückenden Teams ersetzt werden. Es kann aber niemals in dieser Saison weniger als eins oder mehr als 12 Teams geben. Zu den öffentlichen Eigenschaften eines Fahrzeugs gehört der Team- Name, der Name des Chassis, die Motorbezeichnung sowie Startnummer und der Fahrer. Die Startnummer ist eindeutig. Die geheimen Eigenschaften eines Fahrzeugs sind die Größe des Tanks und die Dicke der Bodenplatte. In dieser Saison gibt es verschiedene Motoren, die von den zwölf Teams in ihren beiden Fahrzeugen verbaut sind.

Verwenden Sie die aus der Vorlesung bekannten Vorgehensweisen zur Objektmodellierung und erstellen Sie ein Klassendiagramm. Modellieren Sie dabei Klassen, Attribute, Assoziationen und Multiplizitäten.

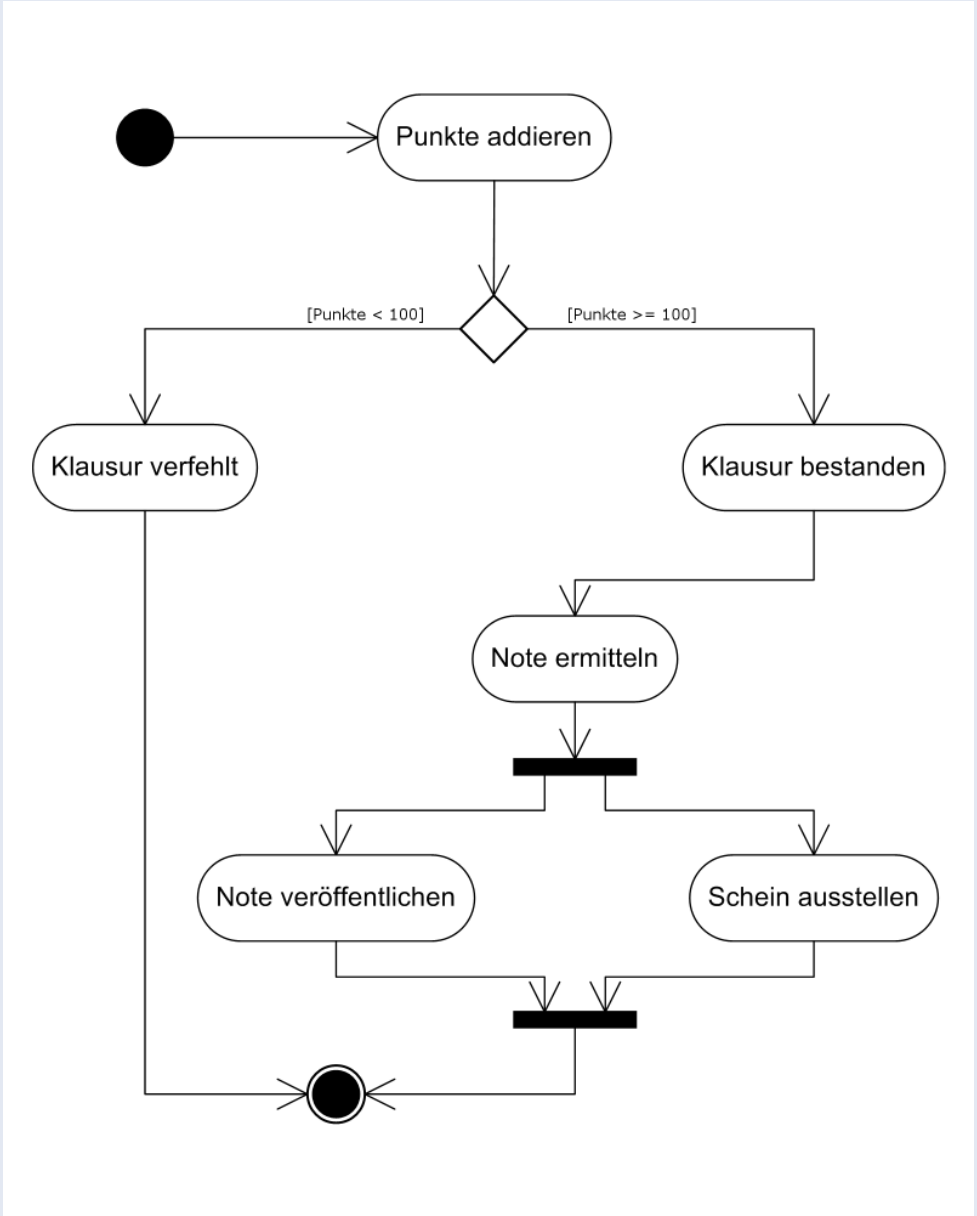
Musterlösung



Aktivitätsdiagramm

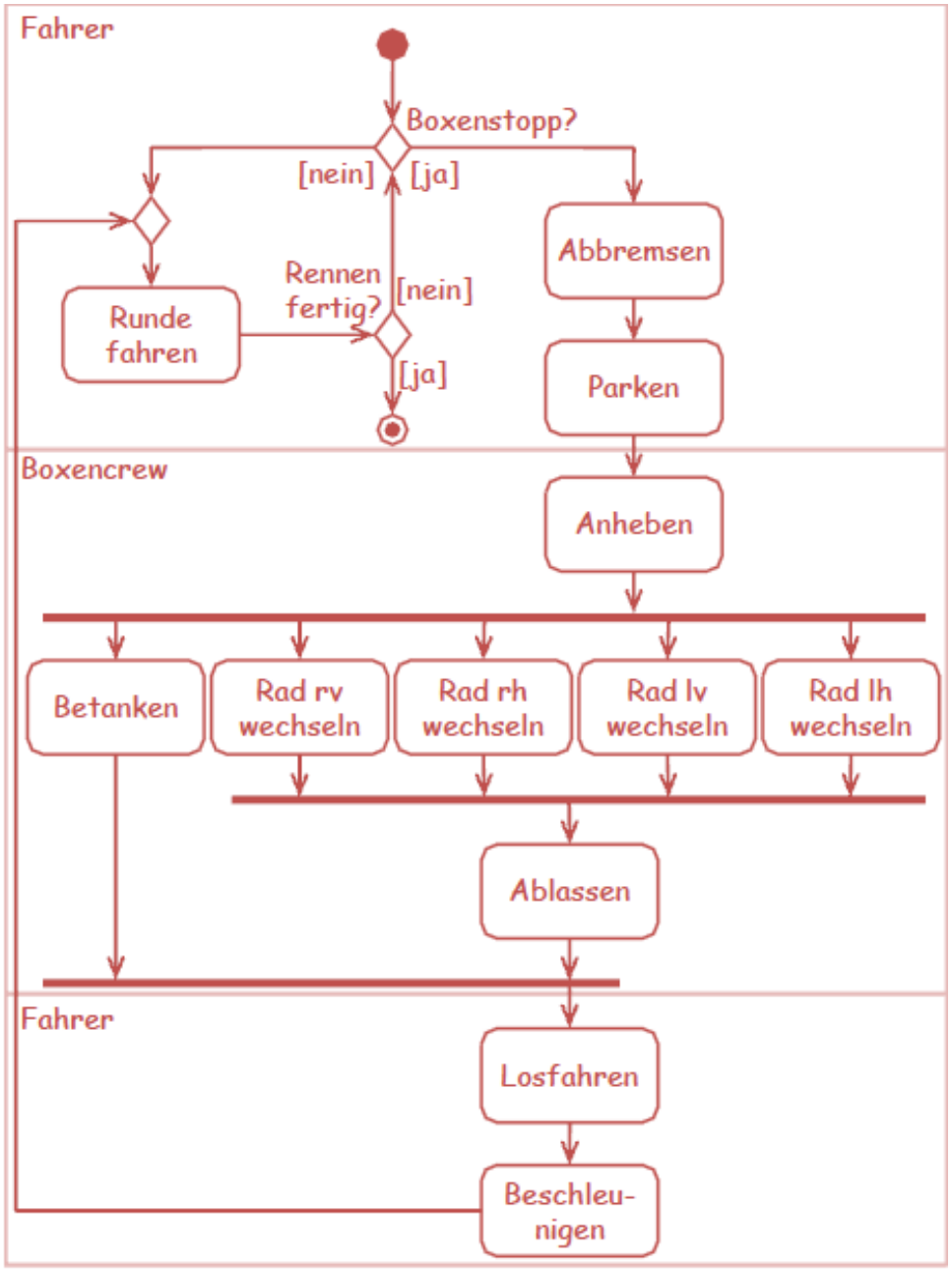
- Ein Aktivitätsdiagramm beschreibt einen Ablauf
- Besteht aus **Aktions-, Objekt-** und **Kontrollflussknoten** sowie **Objekt-** und **Kontrollflüssen**
- Elemente eines Aktivitätsdiagramms
  - Aktionen
  - Knoten (Startknoten, Endknoten, Ablaufende)
  - Entscheidungen (durch Raute dargestellt)
  - Zusammenführung
  - Aufteilung des Kontrollflusses
  - Synchronisation

Beispiel Aktivitätsdiagramm



Klausur 2010 Aktivitätsdiagramm (9P)

Nach dem Start beginnt die erste Runde. In dieser und in jeder folgenden Runde kann sich der Fahrer entscheiden, ob er an die Box kommen oder die Runde fahren möchte. Nach der letzten Runde ist das Rennen unmittelbar beendet. Der Boxenstopp beginnt damit, dass der Fahrer den Rennwagen auf die in der Boxengasse erlaubte Höchstgeschwindigkeit abbremst. Anschließend parkt er seinen Rennwagen in der Box. Sobald der Rennwagen steht, wird er von einem Mitarbeiter der Boxencrew angehoben. Danach werden von vier weiteren Mitarbeitern der Boxencrew gleichzeitig die Räder gewechselt, wobei sich jeder Mitarbeiter um jeweils ein Rad kümmert. Neben dem Radwechsel wird der Rennwagen frisch betankt, was ebenfalls ein dedizierter Mitarbeiter erledigt. Sobald alle vier Räder gewechselt sind, kann der Rennwagen abgelassen werden. Der Fahrer fährt los, sobald der Rennwagen abgelassen und der Tankvorgang beendet wurde. Sobald der Fahrer das Ende der Boxengasse erreicht hat, beschleunigt er den Rennwagen auf Renntempo und fährt die Runde zu Ende. Anschließend geht der Fahrer auf die nächste Runde oder das Rennen ist beendet.


Fahrer
Boxenstopp?
[nein]
[ja]
Abbremsen
Rennen fertig?
[nein]
Runde fahren
[ja]
Parken
Boxencrew
Anheben
Betanken
Rad rv wechseln
Rad rh wechseln
Rad lv wechseln
Rad lh wechseln
Ablassen
Fahrer
Losfahren
Beschleu-nigen

Tipps zum nächsten Übungsblatt

Aufgabe 1 - Klassendiagramm

- denkt an Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen sowie Rollen
- merkt euch den Unterschied zwischen Komposition und Aggregation!

Aufgabe 2 - Aktivitätsdiagramm

- ihr dürft die Aufgabe auf zwei Diagramme verteilen
- `http://de.wikipedia.org/wiki/Floyd-Steinberg-Algorithmus`

Tipps zum nächsten Übungsblatt

Aufgabe 1 - Klassendiagramm

- denkt an Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen sowie Rollen
- merkt euch den Unterschied zwischen Komposition und Aggregation!

Aufgabe 2 - Aktivitätsdiagramm

- ihr dürft die Aufgabe auf zwei Diagramme verteilen
- `http://de.wikipedia.org/wiki/Floyd-Steinberg-Algorithmus`

Tipps zum nächsten Übungsblatt

Aufgabe 1 - Klassendiagramm

- denkt an Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen sowie Rollen
- merkt euch den Unterschied zwischen Komposition und Aggregation!

Aufgabe 2 - Aktivitätsdiagramm

- ihr dürft die Aufgabe auf zwei Diagramme verteilen
- `http://de.wikipedia.org/wiki/Floyd-Steinberg-Algorithmus`

Tipps zum nächsten Übungsblatt

Aufgabe 1 - Klassendiagramm

- denkt an Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen sowie Rollen
- merkt euch den Unterschied zwischen Komposition und Aggregation!

Aufgabe 2 - Aktivitätsdiagramm

- ihr dürft die Aufgabe auf zwei Diagramme verteilen
- `http://de.wikipedia.org/wiki/Floyd-Steinberg-Algorithmus`

Tipps zum nächsten Übungsblatt

Aufgabe 3 - Programmieren

- args4j kann euch sehr viel Arbeit ersparen, versucht die Bedienung anhand von Configuration.java zu verstehen
- achtet darauf ob ihr Ganzzahldivision verwendet:

```
int o; ...; int p = (o + 128 ) / 256 * 255
// liefert hier 0 oder 255: wieso?
```

Tipps zum nächsten Übungsblatt

Aufgabe 3 - Programmieren

- args4j kann euch sehr viel Arbeit ersparen, versucht die Bedienung anhand von Configuration.java zu verstehen
- achtet darauf ob ihr Ganzzahldivision verwendet:

```
int o; ...; int p = (o + 128 ) / 256 * 255
// liefert hier 0 oder 255: wieso?
```

Bis zum nächsten Mal