

Softwaretechnik 1 - 6. Tutorium

Tutorium 17 Felix Bachmann | 23.07.2019

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)

- 1 Orga
 - Feedback
- Werbung
- 3 Testen
 - Definitionen
- 4 Wiederholung und Klausuraufgaben
 - Planung & Definition
 - Entwurf
 - Implementierung
 - Testen
 - Abnahme, Einsatz & Wartung
 - Rest
- 6 Ende
 - Feierabend

Termine



Klausur, Übungsschein

- Hauptklausur am 01.08.19, 11:00
- Nachklausur wahrscheinlich am 07.10.19
- Anmeldung sollte nun für alle möglich sein

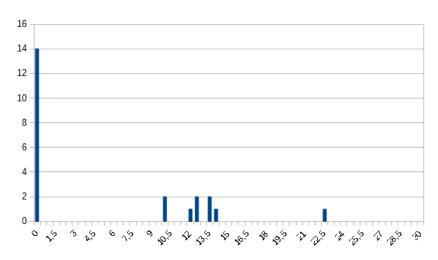
Nicht abgeholte Übungsblätter



- bringe ich zu den Übungsleitern (R346, in der Nähe vom Holzkasten)
 - siehe https://ps.ipd.kit.edu/3273.php
- falls ihr die noch haben wollt, holt sie euch dort ab
- Übungsleiter beißen (meistens) nicht

6. Übungsblatt Statistik







Aufgabe 1+2: Parallelisierung

Ende

6/35



Aufgabe 1+2: Parallelisierung

■ 4+1 Abgabe...

Ende



Aufgabe 1+2: Parallelisierung

- 4+1 Abgabe...
- Anzahl Prozessoren (inkl. HT) berechnen
 - int nproc =
 Runtime.getRuntime().availableProcessors();



Aufgabe 1+2: Parallelisierung

- 4+1 Abgabe...
- Anzahl Prozessoren (inkl. HT) berechnen
 - int nproc =
 Runtime.getRuntime().availableProcessors();

Aufgabe 3: Äquivalenzklassen-Tests

- Grenzwerte "möglichst nah" an Grenze
- z.B. ungültiger Betrag: Grenzwert sowas wie -1 oder -0.01, nicht -20



Aufgabe 4: Kontrollfluss-orientiertes Testen

Ende



Aufgabe 4: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist
 - insb. auch return x; (nicht zusammenfassen!)



Aufgabe 4: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist
 - insb. auch return x; (nicht zusammenfassen!)
 - ansonsten bei Überdeckung gemogelt
 - wo hört man beim Zusammenfassen auf? sind auch alle c++ ein Block?
 - ein return-Block betreten reicht dann schon für Überdeckung aller returns
 - dann können wir es gleich lassen :D



Aufgabe 4: Kontrollfluss-orientiertes Testen

- alles außer Kontrollfluss-Zeug so lassen wie es ist
 - insb. auch return x; (nicht zusammenfassen!)
 - ansonsten bei Überdeckung gemogelt
 - wo hört man beim Zusammenfassen auf? sind auch alle c++ ein Block?
 - ein return-Block betreten reicht dann schon für Überdeckung aller returns
 - dann können wir es gleich lassen :D
 - Code 1:1 übernehmen, ; nicht vergessen



Aufgabe 5: Codeinspektion

Felix Bachmann - SWT1



Aufgabe 5: Codeinspektion

- JavaDoc bei @Override nicht nötig
 - solange Unterklasse nicht etwas nennenswert anders macht
 - sollte sie aber eh nicht (Substitutionsprinzip undso)



Aufgabe 5: Codeinspektion

- JavaDoc bei @Override nicht nötig
 - solange Unterklasse nicht etwas nennenswert anders macht
 - sollte sie aber eh nicht (Substitutionsprinzip undso)
- throws in Signatur und JavaDoc
 - nur bei CheckedExceptions nötig
 - bei UncheckedExceptions = Unterklassen von RuntimeException nicht
 - z.B. NullPointerException, IllegalArgumentException, ArrayIndexOutOfBoundsException

23.07.2019



Aufgabe 5: Codeinspektion

- JavaDoc bei @Override nicht nötig
 - solange Unterklasse nicht etwas nennenswert anders macht
 - sollte sie aber eh nicht (Substitutionsprinzip undso)
- throws in Signatur und JavaDoc
 - nur bei CheckedExceptions nötig
 - bei UncheckedExceptions = Unterklassen von RuntimeException nicht
 - z.B. NullPointerException, IllegalArgumentException, ArrayIndexOutOfBoundsException
- data = new double[rows][cols]; füllt das Array implizit mit Nullen
 - also keine Verletzung des Verhaltens wie in JavaDoc beschrieben



Aufgabe 5: Codeinspektion

- JavaDoc bei @Override nicht nötig
 - solange Unterklasse nicht etwas nennenswert anders macht
 - sollte sie aber eh nicht (Substitutionsprinzip undso)
- throws in Signatur und JavaDoc
 - nur bei CheckedExceptions nötig
 - bei UncheckedExceptions = Unterklassen von RuntimeException nicht
 - z.B. NullPointerException, IllegalArgumentException, ArrayIndexOutOfBoundsException
- data = new double[rows][cols]; füllt das Array implizit mit Nullen
 - also keine Verletzung des Verhaltens wie in JavaDoc beschrieben
- final-Variablen können erst deklariert werden und im Konstruktor initialisiert



Was?

- Übergang von SWT1 zu PSE
- in kleinen Gruppen ein 2D-Jump'n'Run erweitern



Was?

- Übergang von SWT1 zu PSE
- in kleinen Gruppen ein 2D-Jump'n'Run erweitern
- <zeige werbungs-slides>



Was?

- Übergang von SWT1 zu PSE
- in kleinen Gruppen ein 2D-Jump'n'Run erweitern
- <zeige werbungs-slides>

Wann?

- steht noch nicht 100%ig fest
- vermutlich 1./2. VL-Woche vom n\u00e4chsten Semester
- bevor das PSE losgeht



Was?

- Übergang von SWT1 zu PSE
- in kleinen Gruppen ein 2D-Jump'n'Run erweitern
- <zeige werbungs-slides>

Wann?

- steht noch nicht 100%ig fest
- vermutlich 1./2. VL-Woche vom nächsten Semester
- bevor das PSE losgeht

Warum?

- SWT ist dann schon über 2 Monate her ;)
- und außerdem: warum nicht?

Arten von Fehlern



- Was verursacht was?
- Defekt, Irrtum, Versagen

Arten von Fehlern



- Was verursacht was?
- Defekt, Irrtum, Versagen
- Irrtum (Aktion) → Defekt (Bug) → Ausfall (failure)
- dumme Eselsbrücke: IDA

Arten von Fehlern



- Was verursacht was?
- Defekt, Irrtum, Versagen
- Irrtum (Aktion) → Defekt (Bug) → Ausfall (failure)
- dumme Eselsbrücke: IDA
- "Testing shows the presence of bugs, not their absence." (Edsger W. Dijkstra)



Dynamische Verfahren

■ Testfälle schreiben und ausführen (z.B. mit JUnit)



- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing



- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing



- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests



- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests



Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Statische Verfahren

Inspektion ("scharfes Hinsehen", Code Review, etc.)



Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Statische Verfahren

- Inspektion ("scharfes Hinsehen", Code Review, etc.)
- statische Analyse mit Tools (formale Verifikation, SpotBugs, etc.)



Dynamische Verfahren

- Testfälle schreiben und ausführen (z.B. mit JUnit)
- white box testing
 - kontrollflussorientiert
 - datenflussorientiert
- black box testing
 - funktionale Tests
 - Leistungstests

Statische Verfahren

- Inspektion ("scharfes Hinsehen", Code Review, etc.)
- statische Analyse mit Tools (formale Verifikation, SpotBugs, etc.)
- Programm wird nicht ausgeführt!

Disclaimer



Ich kenne die Klausur auch nicht!

Ende

Disclaimer



- Ich kenne die Klausur auch nicht!
 - ⇒ alles, was ich zum Inhalt der Klausur sage ist Spekulation
 - basierend auf Altklausuren

Disclaimer



- Ich kenne die Klausur auch nicht!
 - ⇒ alles, was ich zum Inhalt der Klausur sage ist Spekulation
 - basierend auf Altklausuren
- kein Anspruch auf Vollständigkeit der Wiederholung



- 4 Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen



- 4 Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen
- 2 meistens Aufgaben zu:
 - UML-Diagrammen

23.07.2019



- 4 Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen
- meistens Aufgaben zu:
 - UML-Diagrammen
 - Entwurfsmustern



- 4 Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen
- meistens Aufgaben zu:
 - UML-Diagrammen
 - Entwurfsmustern
 - Parallelität



- Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen
- meistens Aufgaben zu:
 - **UML-Diagrammen**
 - Entwurfsmustern
 - Parallelität
 - Testen bzw. Qualitätssicherung (KFG!)

23.07.2019



- Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen
- meistens Aufgaben zu:
 - UML-Diagrammen
 - Entwurfsmustern
 - Parallelität
 - Testen bzw. Qualitätssicherung (KFG!)
 - Rest (z.B. Objektorientierung, Git/Versionskontrolle, Prozessmodelle...)



- Aufgabe 1: Aufwärmen
 - Wahr-/Falsch-Fragen
 - ein paar gesammelt auf https://github.com/malluce/ swt1-tut/blob/master/multiple_choice.txt
 - Wissensfragen
- meistens Aufgaben zu:
 - UML-Diagrammen
 - Entwurfsmustern
 - Parallelität
 - Testen bzw. Qualitätssicherung (KFG!)
 - Rest (z.B. Objektorientierung, Git/Versionskontrolle, Prozessmodelle...)
- bisher: $\frac{1}{3} \pm \epsilon$ der Punkte reichen zum Bestehen
 - kann sich eventuell ändern, wegen längerer Bearbeitungszeit

Aufwärmaufgabe



Hauptklausur SS2011 A1

Ende



■ Lastenheft, Pflichtenheft

Felix Bachmann - SWT1

Wiederholung und Klausuraufgaben



- Lastenheft, Pflichtenheft
 - Phasen zuordnen

23.07.2019



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen

23.07.2019



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm
 - Syntax kennen!



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm
 - Syntax kennen!
 - gegebenen Text/Code in Diagramm umwandeln

15/35



- Lastenheft, Pflichtenheft
 - Phasen zuordnen
 - Gliederung kennen
 - Beispiele geben
- UML-Diagramme
 - Klassendiagramm
 - Aktivitäts-, Sequenz-, Zustandsdiagramm
 - Anwendungsfalldiagramm
 - Syntax kennen!
 - gegebenen Text/Code in Diagramm umwandeln
 - bei Zustandsdiagramm
 - Umwandeln hierarchisch ⇔ nicht-hierarchisch
 - Umwandeln parallel ⇔ nicht-parallel

15/35

Zustandsdiagramm-Aufgabe



Hauptklausur SS2012 A2

Testen

Ende



Architekturstile



- Architekturstile
- Entwurfsmuster



- Architekturstile
- Entwurfsmuster
 - möglichst viele, bestenfalls alle kennen und verstehen



- Architekturstile
- Entwurfsmuster
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen



- Architekturstile
- Entwurfsmuster
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)



- Architekturstile
- Entwurfsmuster
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)
 - aus Klassendiagrammen/Code Entwurfsmuster erkennen



- Architekturstile
- Entwurfsmuster
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)
 - aus Klassendiagrammen/Code Entwurfsmuster erkennen
 - Code für Muster schreiben



- Architekturstile
- **Entwurfsmuster**
 - möglichst viele, bestenfalls alle kennen und verstehen
 - Kategorien kennen
 - Klassendiagramm hinzeichnen (auch auf Szenario angepasst)
 - aus Klassendiagrammen/Code Entwurfsmuster erkennen
 - Code für Muster schreiben
 - Code-Schnipsel auf mögliche Verbesserung durch EM untersuchen

Entwurfsmuster-Aufgabe



Hauptklausur SS2010 A3



UML-Abbildung

Ende



- UML-Abbildung
- Parallelität

Felix Bachmann - SWT1



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore

23.07.2019



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (Tafel))



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (Tafel))
- gegebenen Code thread-safe machen



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (Tafel))
- gegebenen Code thread-safe machen
- Lösungsvorschläge zur Synchronisation bewerten



- UML-Abbildung
- Parallelität
 - grundlegendes Prinzip
 - in Java
 - critical sections/ race conditions
 - deadlock
 - Monitore, wait & notify
 - Semaphore
- Rechnungen können (Speedup, Amdahls Law (Tafel))
- gegebenen Code thread-safe machen
- Lösungsvorschläge zur Synchronisation bewerten
- eigenen Code schreiben

Parallelität-Aufgabe



Hauptklausur SS2011 A3

Ende

20/35



- thread-safe Datenstrukturen ("Concurrent Collections")
 - kann man in parallelen Programmen verwenden, falls mehrere Threads drauf zugreifen
 - "normale" Collections sind nicht thread-safe
 - was soll passieren, wenn ein Thread über Daten iteriert, ein anderer sie gleichzeit modifiziert?
 - sind "fail-fast": werfen ConcurrentModificationException



- thread-safe Datenstrukturen ("Concurrent Collections")
 - kann man in parallelen Programmen verwenden, falls mehrere Threads drauf zugreifen
 - "normale" Collections sind nicht thread-safe
 - was soll passieren, wenn ein Thread über Daten iteriert, ein anderer sie gleichzeit modifiziert?
 - sind "fail-fast": werfen ConcurrentModificationException
 - statt HashMap, verwende ConcurrentHashMap
 - statt ArrayList, verwende CopyOnWriteArrayList
 - Achtung: kopiert bei jeder Veränderung das gesamte Array
 - daher nur benutzen, wenn öfter gelesen/iteriert wird als geschrieben
 - usw.



- thread-safe Datenstrukturen ("Concurrent Collections")
 - kann man in parallelen Programmen verwenden, falls mehrere Threads drauf zugreifen
 - "normale" Collections sind nicht thread-safe
 - was soll passieren, wenn ein Thread über Daten iteriert, ein anderer sie gleichzeit modifiziert?
 - sind "fail-fast": werfen ConcurrentModificationException
 - statt HashMap, verwende ConcurrentHashMap
 - statt ArrayList, verwende CopyOnWriteArrayList
 - Achtung: kopiert bei jeder Veränderung das gesamte Array
 - daher nur benutzen, wenn öfter gelesen/iteriert wird als geschrieben
 - usw.
 - Achtung: Concurrent != Synchronized
 - Datenstrukturen lassen ggf. mehrere Threads zu (JavaDoc lesen)



- thread-safe Datenstrukturen ("Concurrent Collections")
 - kann man in parallelen Programmen verwenden, falls mehrere Threads drauf zugreifen
 - "normale" Collections sind nicht thread-safe
 - was soll passieren, wenn ein Thread über Daten iteriert, ein anderer sie gleichzeit modifiziert?
 - sind "fail-fast": werfen ConcurrentModificationException
 - statt HashMap, verwende ConcurrentHashMap
 - statt ArrayList, verwende CopyOnWriteArrayList
 - Achtung: kopiert bei jeder Veränderung das gesamte Array
 - daher nur benutzen, wenn öfter gelesen/iteriert wird als geschrieben
 - usw.
 - Achtung: Concurrent != Synchronized
 - Datenstrukturen lassen ggf. mehrere Threads zu (JavaDoc lesen)
 - Code-Beispiel



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute(Runnable runnable)



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute (Runnable runnable)
 - ExecutorService extends Executor bietet weitere Methoden

23.07.2019



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute(Runnable runnable)
 - ExecutorService extends Executor bietet weitere Methoden
 - Executors bietet Methoden zum Erstellen von ExecutorService-Impl.
 - newFixedThreadPool(int nThreads)
 - newCachedThreadPool()



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute(Runnable runnable)
 - ExecutorService extends Executor bietet weitere Methoden
 - Executors bietet Methoden zum Erstellen von ExecutorService-Impl.
 - newFixedThreadPool(int nThreads)
 - newCachedThreadPool()
- Callable<T>, Future<T>
 - Problem an Runnable: keine Rückgabe möglich (void)

22/35



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute(Runnable runnable)
 - ExecutorService extends Executor bietet weitere Methoden
 - Executors bietet Methoden zum Erstellen von ExecutorService-Impl.
 - newFixedThreadPool(int nThreads)
 - newCachedThreadPool()
- Callable<T>, Future<T>
 - Problem an Runnable: keine Rückgabe möglich (void)
 - Callable<T>#call() gibt T zurück



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute (Runnable runnable)
 - ExecutorService extends Executor bietet weitere Methoden
 - Executors bietet Methoden zum Erstellen von ExecutorService-Impl.
 - newFixedThreadPool(int nThreads)
 - newCachedThreadPool()
- Callable<T>, Future<T>
 - Problem an Runnable: keine Rückgabe möglich (void)
 - Callable<T>#call() gibt T zurück
 - Problem: "man weiß nicht wann call() fertig"
 - daher: Future<T> ist Abstraktion der Rückgabe
 - mittel Future<T>#get() kann man auf Ergebnis warten



- Executor, ExecutorService, Executors
 - abstrahiert von Thread-Management
 - Executor bietet Methode void execute (Runnable runnable)
 - ExecutorService extends Executor bietet weitere Methoden
 - Executors bietet Methoden zum Erstellen von ExecutorService-Impl.
 - newFixedThreadPool(int nThreads)
 - newCachedThreadPool()
- Callable<T>, Future<T>
 - Problem an Runnable: keine Rückgabe möglich (void)
 - Callable<T>#call() gibt T zurück
 - Problem: "man weiß nicht wann call() fertig"
 - daher: Future<T> ist Abstraktion der Rückgabe
 - mittel Future<T>#get() kann man auf Ergebnis warten
 - z.B.: ExecutorService#submit(Callable<T> task) gibt Future<T> zurück

23.07.2019

java.util.concurrent.atomic



- bietet atomare Datentypen
- "lock-free thread-safe programming on single variables"
- erlaubt z.B. atomares i++;
 - normalerweise: tmp = i; tmp = tmp + 1; i = tmp;
 - problematisch bei mehreren Threads (s. Bsp. letztes Tut)
- Bsp: AtomicInteger#incrementAndGet()

java.util.concurrent.atomic



- bietet atomare Datentypen
- "lock-free thread-safe programming on single variables"
- erlaubt z.B. atomares i++;
 - normalerweise: tmp = i; tmp = tmp + 1; i = tmp;
 - problematisch bei mehreren Threads (s. Bsp. letztes Tut)
- Bsp: AtomicInteger#incrementAndGet()
- Code-Beispiel

Semaphoren



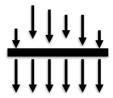
- Semaphore(int n)
 - kritische Abschnitte mit Kapazität n statt 1
 - ein kritischer Abschnitt kann von n Threads betreten werden
 - nützlich, wenn z.B. n Resourcen verarbeitet werden sollen
 - aquire(): versuchen reinzukommen, blockieren falls nicht möglich
 - tryAquire() wie aquire(), aber nicht blockierend
 - release() gibt einen "Platz" wieder frei

24/35

CyclicBarrier



- CyclicBarrier(int n)
 - await() blockiert aufrufenden Thread
 - falls await() n-mal aufgerufen, dürfen alle Threads weitermachen
 - Cyclic = Barriere kann danach wiederbenutzt werden

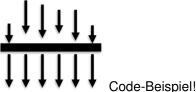


23.07.2019

CyclicBarrier



- CyclicBarrier(int n)
 - await() blockiert aufrufenden Thread
 - falls await() n-mal aufgerufen, dürfen alle Threads weitermachen
 - Cyclic = Barriere kann danach wiederbenutzt werden





- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest

23.07.2019



- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - Kontrollflussgraph



- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - Kontrollflussgraph !!!
 - absolute Standardaufgabe



- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - Kontrollflussgraph !!!
 - absolute Standardaufgabe
 - in 33 von 36 der letzten Altklausuren
 - in allen Klausuren seit SS14
 - im Schlaf können ("Schema F", lässt sich sehr gut üben...)



- Testphasen
 - Komponententest, Integrationstest, Systemtest, Abnahmetest
- Testverfahren
 - Kontrollflussgraph !!!
 - absolute Standardaufgabe
 - in 33 von 36 der letzten Altklausuren
 - in allen Klausuren seit SS14
 - im Schlaf können ("Schema F", lässt sich sehr gut üben...)
- Testhelfer
- Definitionen kennen (Fehlerarten...)

KFG: Klausuraufgabe



Hauptklausur SS2016 A6

Ende



Aufgaben der verschiedenen "Subphasen" kennen



- Aufgaben der verschiedenen "Subphasen" kennen
- viel Text zum Lernen, aber nicht schwierig...



- Aufgaben der verschiedenen "Subphasen" kennen
- viel Text zum Lernen, aber nicht schwierig...
- Wartung vs. Pflege

23.07.2019



- Aufgaben der verschiedenen "Subphasen" kennen
- viel Text zum Lernen, aber nicht schwierig...
- Wartung vs. Pflege
- selten eigene Aufgabe dazu, eher Ankreuzaufgaben in A1



Schätzmethoden



- Schätzmethoden
- Prozessmodelle

Ende



- Schätzmethoden
- Prozessmodelle
- Agile Prozesse

Werbung



- Schätzmethoden
- Prozessmodelle
- Agile Prozesse
- verschiedene Modelle kennen (XP, Scrum,...)

Felix Bachmann - SWT1



- Schätzmethoden
- Prozessmodelle
- Agile Prozesse
- verschiedene Modelle kennen (XP, Scrum,...)
- auch eher Ankreuzaufgaben, Wissensfragen

Git: Klausuraufgabe



Hauptklausur SS18 A4

Ende

Git: Klausuraufgabe Local Remote working remote staging local repo directory area repo git add git commit git push git fetch git checkout git merge

Backup: Klassendiagramm-Aufgabe



Nachklausur SS2011 A5

Felix Bachmann - SWT1

23.07.2019

Ende

Lernen



- Klausuren rechnen && Folien anschauen
 - Übung nötig, für sehr gute Note aber auch viel Wissen

Lernen



- Klausuren rechnen && Folien anschauen
 - Übung nötig, für sehr gute Note aber auch viel Wissen
- diesmal habt ihr mehr Zeit für den gleichen Umfang an Aufgaben wie in früheren Klausuren
 - vermutlich wird Bestehensgrenze hochgesetzt

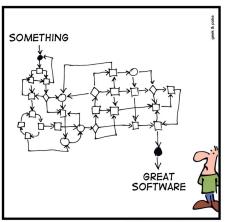
33/35

Das war's dann wohl...



- danke für Eure Anwesenheit und Mitarbeit!
- Viel Erfolg bei der Klausur und im weiteren Studium! :)

SIMPLY EXPLAINED



DEVELOPMENT PROCESS