

Softwarepraktikum

1. Frontalveranstaltung

20.10.2017



TECHNISCHE
UNIVERSITÄT
DARMSTADT



ES Real-Time Systems Lab

Prof. Dr. rer. nat. Andy Schürr

Dept. of Electrical Engineering and Information Technology

Dept. of Computer Science (adjunct Professor)

Dr. Malte Lochau

Malte.Lochau@es.tu-darmstadt.de

www.es.tu-darmstadt.de

- Einordnung des Praktikums
- Vorstellung des Frameworks EVS
- Zeitplan, Organisatorisches
- Erste Schritte
- Aufgabenblock 1

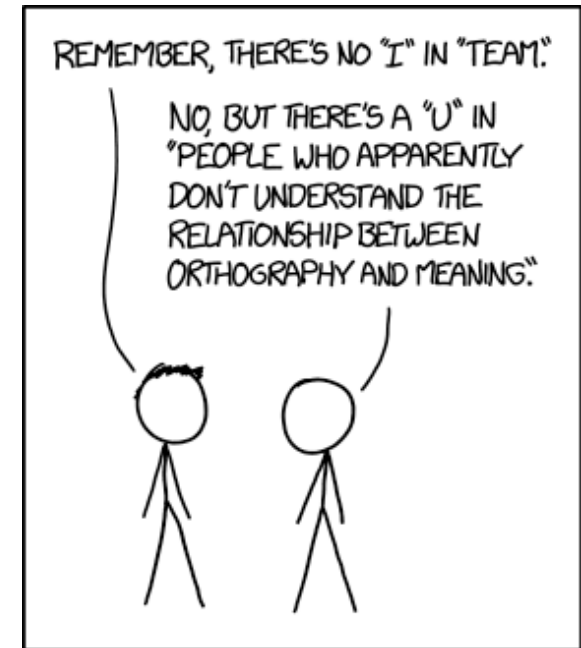
Ziele des Praktikums

■ Teamarbeit

- Kommunikation
- Persönlichkeitsentwicklung
- Konfliktlösungskompetenz
- Selbstorganisation

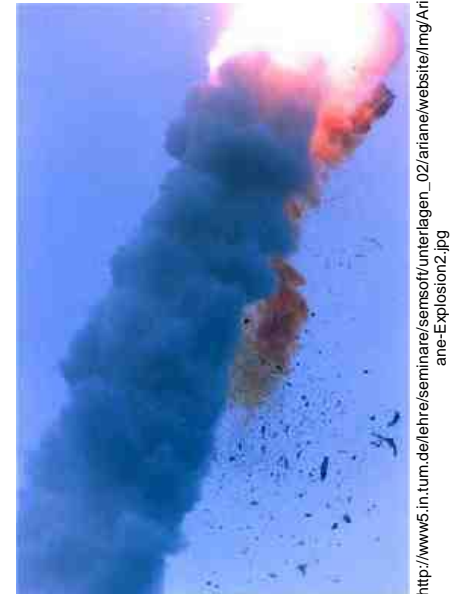
■ Software-Entwicklung

- *Voraussetzung:* Grundlegende Java-Kenntnisse
- Ausbau und Vertiefung Ihrer Kenntnisse
- Umgang mit Eclipse
- Entwickeln in einem bereitgestellten Framework
- Aber: Software-Entwicklung ist mehr als „nur“ programmieren



Explosion der Ariane-5-Rakete

```
...
declare
  vertical_veloc_sensor: float;
  horizontal_veloc_sensor: float;
  vertical_veloc_bias: integer;
  horizontal_veloc_bias: integer;
  ...
begin
  declare
    pragma suppress(numeric_error, horizontal_veloc_bias);
  begin
    sensor_get(vertical_veloc_sensor);
    sensor_get(horizontal_veloc_sensor);
    vertical_veloc_bias := integer(vertical_veloc_sensor);
    horizontal_veloc_bias := integer(horizontal_veloc_sensor);
    ...
  exception
    when numeric_error => calculate_vertical_veloc();
    when others => use_irs1();
  end;
end irs2;
```



**Overflow Error →
500 Mio. \$ Schaden**

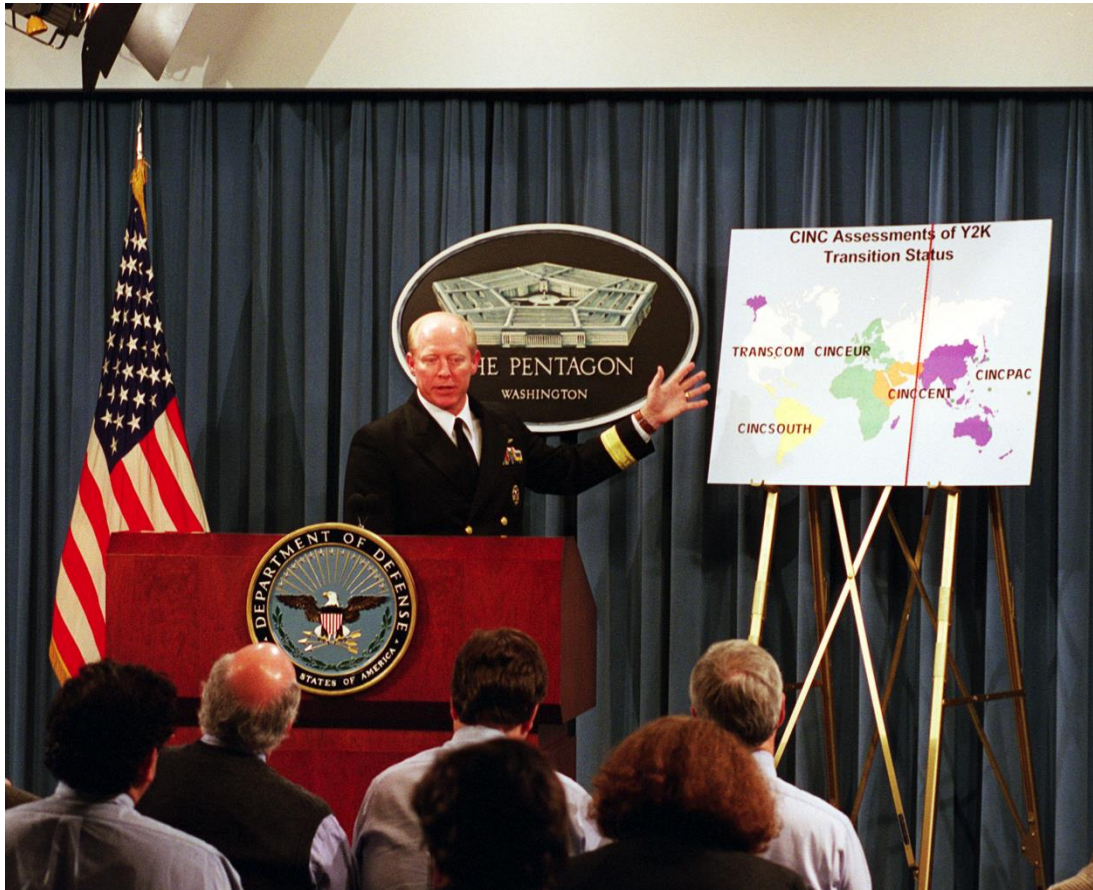
<https://web-docs.gsi.de/~giese/swr/ariane5.html>



Jahr 2000 / Y2K - Bug

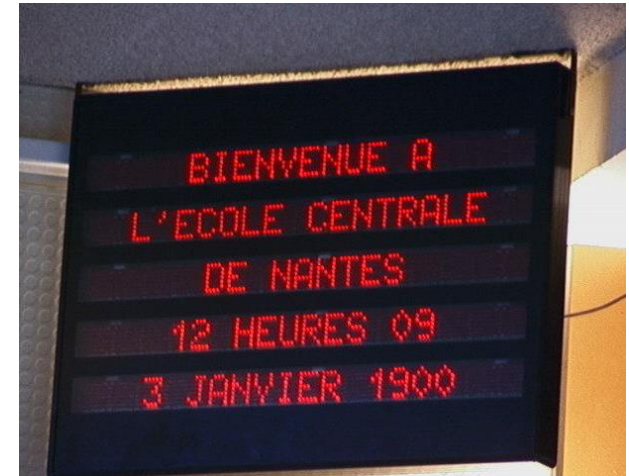


TECHNISCHE
UNIVERSITÄT
DARMSTADT

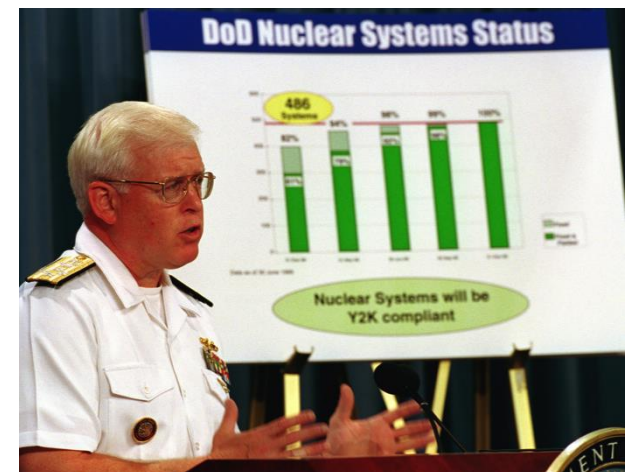


R. D. Ward Wikimedia Commons

Pressekonferenz der Y2K Task Force, US Defense Department



Bug de l'an 2000 Wikimedia Commons



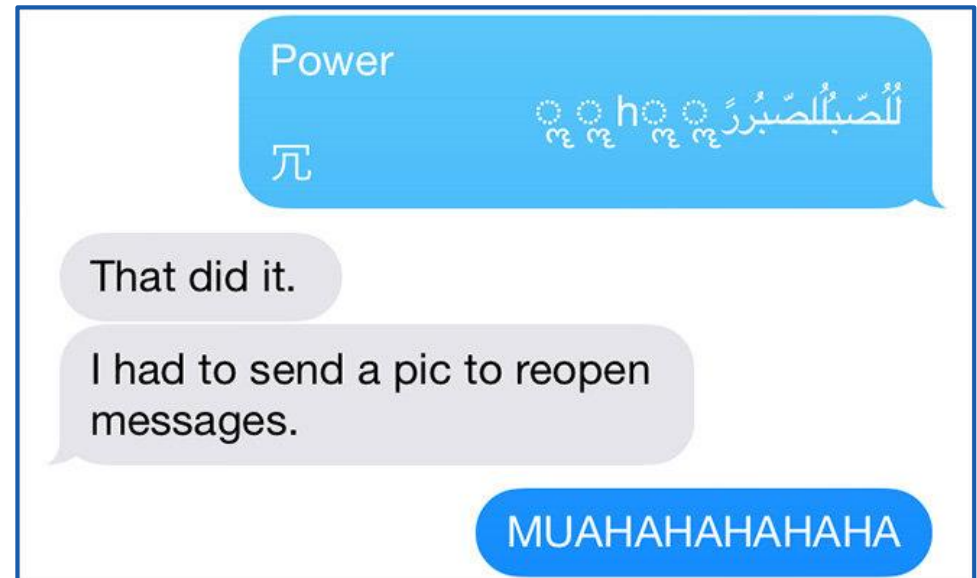
DoD Wikimedia Commons



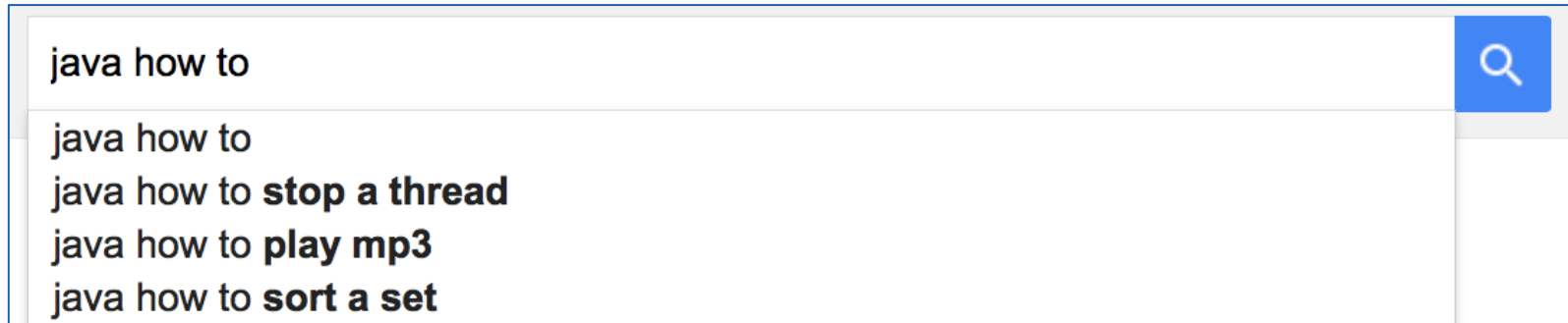
Software-Fehler in mobilen Applikationen

iPhone Unicode Bug im Mai 2015,
betroffen: >> Millionen Nutzer

- „death-string“ als SMS
→ iPhone Crash und Reboot
- Ursache: Unicode Zeichenkette
zu lang, nicht darstellbar in
Nachrichtenvorschau



Selbständiges Arbeiten



Wer/was kann weiterhelfen?

- Tutoren → auch freiwillige Termine aktiv nutzen!
- zusätzliche Frontalveranstaltungen (Repetitorien)
- Kommunikation innerhalb der Gruppe
- Material auf Moodle
- Internet-Tutorials (Vorschläge auf Moodle)
- Java-Dokumentation



Plagiatsprüfung!



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Verschärfte Ahndung von Plagiaten!

Neufassung der Allgemeinen Prüfungsbestimmungen der TU Darmstadt

§ 38 Täuschung und Ordnungswidrigkeiten (Abs. 3)

Prüflinge, die die Anweisungen über die Arbeits- und Hilfsmittel nicht befolgen oder sich auf andere Weise einer

Täuschungshandlung schuldig machen, sind durch Beschluss der Prüferinnen und Prüfer von der weiteren Teilnahme auszuschließen.

Ist dies nicht unmittelbar möglich, muss die Aufsicht das bisherige Arbeitsergebnis sichern und den Abschluss der schriftlichen Prüfung unter Vorbehalt ermöglichen.

Plagiate führen zu ...

- **Ausschluss** des gesamten Teams aus dem Praktikum
- Meldung des **Täuschungsversuchs**
- Wiederholte Verstöße → **Exmatrikulation** aus dem Studiengang

- Alle Lösungen müssen von Ihrem Team im aktuellen Semester komplett selbst erstellt werden.
- Jedes Team-Mitglied muss alle gemeinsamen Lösungen dem Tutor im Detail erklären können.

- Einordnung des Praktikums
- Vorstellung des Frameworks EVS
- Zeitplan, Organisatorisches
- Erste Schritte
- Aufgabenblock 1

Vorstellung des Frameworks

Energie-Versorgungs-Szenario (EVS)

- Ausbau und Regelung einer zuverlässigen Stromversorgung planen und ausführen
- Planung von Budget, Kraftwerksbau, Stromleitungen
- Bausteine der Gesamt-Software werden schrittweise von AB1 bis AB6 von Ihnen entwickelt
- Ausführliche Spielanleitung in → moodle



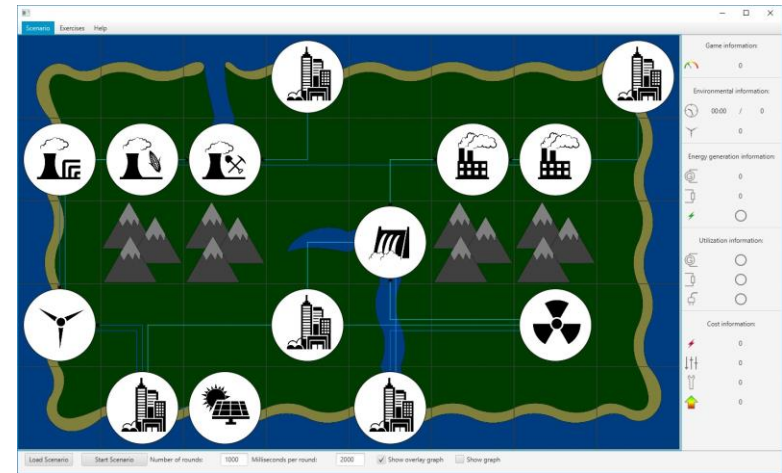
Creative Commons Stefan Kühn



Creative Commons Philipp Hertzog

Aufbau des Spiels

- Ausbau eines autarken Energieversorgungs-Netzes auf einer von der Außenwelt abgeschlossenen Insel
- Energieproduzenten (**Producer**) und Energiekonsumenten (**Consumer**) sind über Stromleitungen (**PowerLine**) miteinander verbunden
- Ausbau von unbebauten Umspannwerken (**TransformerStation**) zu Kraftwerken (**Producer**) zur Erhöhung der Energieproduktion
- Ausbau bestehender Stromleitungen (**PowerLine**) zur Erhöhung der Leitungskapazität



Ziel des Spiels

Alle Konsumenten sollen jederzeit bestmöglich mit der **angeforderten Leistung** versorgt werden, wobei

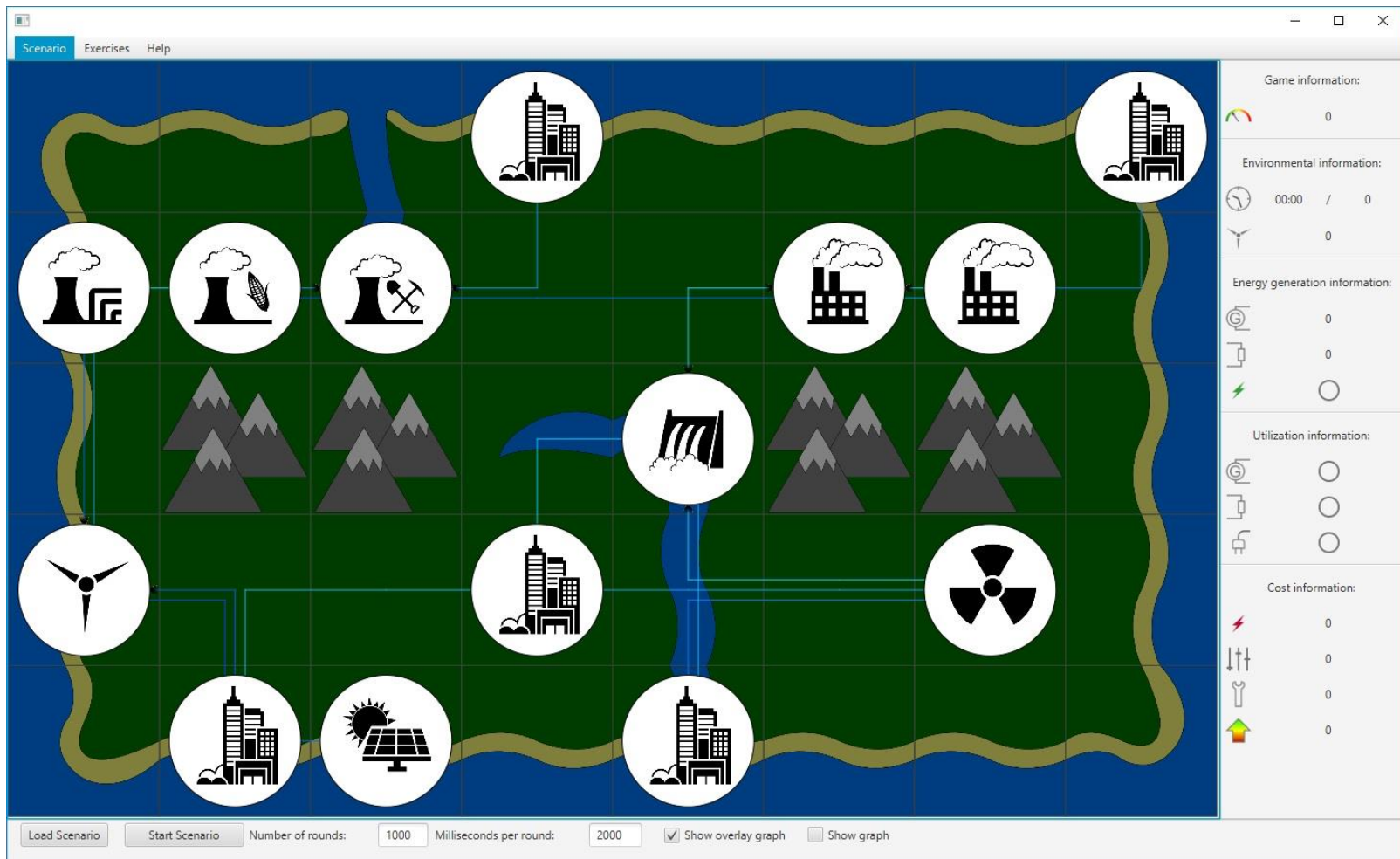
1. **nachhaltige Energieformen** bevorzugt werden sollen und
2. **wenig Kosten** verursacht werden sollen

Mögliche (potentiell gegenläufige) Strategien:

- Kostengünstiger Bau von Kraftwerken und Ausbau von Leitungen an strategisch wichtigen Stellen im Energie-Netz
- Geschickte Regelung von Produzenten und Konsumenten, um Überproduktion zu vermeiden. Dabei möglichst vorausschauend regeln, da Regelungsaktionen ebenfalls Kosten verursachen
- Möglichst hohen Anteil erneuerbarer Energieformen erzielen, da jede produzierte Energie-Einheit entsprechend gewichtet wird



Spielfeld – Übersicht



Spielfeld – Kachelarten

Es gibt fünf Kachelarten (**PlayfieldElement**):

- Wiese (Grassland)



- Fluss (River)



- Berg (Mountain)



- Strand (Beach)



- Meer (Sea)



Spielfeld – Produzenten und Konsumenten

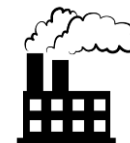
- Eine Kachel kann zusätzlich zum Kacheltyp mit einem Konsumenten, Produzenten oder Umspannwerk versehen sein
- Der jeweilige Kacheltyp hat Einfluss auf die Baukosten und Produktionskapazitäten von Kraftwerken auf dieser Kachel



Spielfeld – Konsumenten

Es gibt drei Konsumententypen:

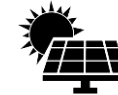
- Städte (City)
- Gewerbepark (CommercialPark)
- Industrieanlage (IndustrialPark)



Spielfeld – Produzenten

Es gibt sieben Produzententypen:

- Solarkraftwerk (SolarPowerPlant)
- Windkraftwerk (WindPowerPlant)
- Biogaskraftwerk (BioGasFiredPowerPlant)
- Gaskraftwerk (GasFiredPowerPlant)
- Wasserkraftwerk (HydroPowerPlant)
- Kohlekraftwerk (CoalFiredPowerPlant)
- Atomkraftwerk (NuclearPowerPlant)





Bei den Energieformen der verschiedenen Produzententypen wird unterschieden zwischen:

- regenerativ – nicht-regenerativ
- regelbar – nicht-regelbar
- konstante Energieproduktion – schwankende Energieproduktion
- Beispiele:
 - WindPowerPlant ist regenerativ, nicht-regelbar und schwankend
 - NuclearPowerPlant ist nicht-regenerativ, regelbar und konstant

Umspannwerke dienen als Verbindungsknoten im Leitungsnetz und können zu Kraftwerken ausgebaut werden:

- Umspannwerk (TransformerStation)



Konsumenten, Produzenten und Umspannwerke sind im Energienetz über Leitungen miteinander verbunden.

Es gibt drei mögliche Leitungskapazitäten:

- Hochspannung (**HighVoltage**)
- Mittelspannung (**MediumVoltage**)
- Niederspannung (**LowVoltage**)



- Für die Leistung eines Produzenten gilt:
 - $\text{abgegebene} \leq \text{bereitgestellte} \leq \text{maximal produzierbare Leistung}$
 - $\text{distributedPower} \leq \text{providedPower} \leq \text{maxPower}$
- Für die Leistung eines Konsumenten gilt:
 - $\text{erhaltene} \leq \text{angeforderte} \leq \text{maximal nutzbare Leistung}$
 - $\text{receivedPower} \leq \text{requiredPower} \leq \text{maxPower}$



Leistungsbedarfe von Konsumenten

Die erhaltene Leistung eines Konsumenten wird durch folgende Faktoren beeinflusst:

1. Die Kapazität des Leitungsnetzes zwischen Produzenten und Konsumenten,
2. die durch Produzenten bereitgestellte Leistung und
3. die angeforderte Leistung, die nicht überschritten werden kann und je nach Konsumententyp rundenabhängig variieren kann



Angeforderte Leistung von Konsumenten

- Die angeforderte Leistung für den Konsumententyp **IndustrialPark** ist konstant und entspricht immer dem durch Regelung festgelegten Wert
- Die angeforderte Leistung für die Konsumententypen **City** und **CommercialPark** variiert je nach Tageszeit, überschreitet aber nie die maximal nutzbare Leistung (→ *Spielanleitung: Tab. 2*)
- Eine Runde entspricht jeweils einer Stunde. Ein Szenario startet stets bei 0 Uhr. Die Tageszeit ergibt sich somit aus

$$\text{Tageszeit} = \text{Rundenzahl} \bmod 24$$





Bereitgestellte Leistung von Produzenten

- Die bereitgestellte Leistung von Produzenten der Typen **NuclearPowerPlant**, **BioGasFiredPowerPlant**, **GasFiredPowerPlant**, **CoalFiredPowerPlant** und **HydroPowerPlant** ist konstant und entspricht immer dem durch Regelung festgelegten Wert
- Die bereitgestellte Leistung des nicht regelbaren Produzententyps **SolarPowerPlant** variiert je nach Tageszeit
(→ *Spielanleitung: Tab. 3*)
- Die bereitgestellte Leistung des nicht regelbaren Produzententyps **WindPowerPlant** variiert zufällig



Der Ablauf eines Spiels besteht aus zwei aufeinanderfolgenden Phasen

1. Bauphase (Build Phase)

- Bau von Kraftwerken 
- Ausbau von Leitungen 

2. Ausführungsphase (Execution Phase)



Simulation
Tageszyklus



Regelung



Leistung
Produzenten



Leistung
Konsumenten



Erneuerbare
Energien



Umwelt-
einflüsse



In der Bauphase können zwei Arten von Aktionen ausgeführt werden

1. **Ausbau von Umspannwerken zu Kraftwerken**
2. **Ausbau von Leitungen**

- Für jeden Ausbau fallen bestimmte Kosten an
- Es können beliebige Baukosten in der Bauphase verursacht werden („unendlicher Kredit“)
- Die Baukosten müssen in der Ausführungsphase amortisiert werden



Baukosten und maximale Produktion von Kraftwerken

Die Baukosten für Kraftwerke hängen von mehreren Faktoren ab

- Basiskosten je Kraftwerktyp (→ *Spielanleitung: Tab. 8*)
- Baukostenfaktor je nach Art des Bauortes (→ *Spielanleitung: Tab. 4,5*)

Der Typ und Bauort eines Kraftwerkes hat auch Auswirkungen auf dessen maximal mögliche Produktion

- (→ *Spielanleitung: Tab. 6*)



Beispiel: Bau von Kraftwerken



Basiskosten
Kohlekraftwerk

1500

×



Multiplikator
„Medium“

1

=



Gesamtbaukosten

1500



Ausbaukosten für Leitungen

- Alle Leitungen haben zu Beginn die Kapazität **LowVoltage** und können zu **MediumVoltage** und **HighVoltage** ausgebaut werden
- Die Basiskosten für den Ausbau ergeben sich aus dem Leitungstyp (→ *Spielanleitung: Tab. 7*)
- Der Ausbau hängt weiter von der *Manhattan-Distanz* d der Leitung auf dem Spielfeld ab. Seien (x_1, y_1) und (x_2, y_2) die Start- und Endkoordinaten der Leitung, dann gilt:

$$d = |x_1 - x_2| + |y_1 - y_2|$$



Ausführungsschritte pro Runde

1. Aktualisierung der Umwelteinflüsse:

- Je nach Tageszeit wird der Wert der Sonnenintensität aktualisiert.
- Für die aktuelle Windstärke wird ein Zufallswert ermittelt.
- Für Konsumenten mit schwankendem Bedarf wird die angeforderte Leistung aktualisiert

2. Aufruf der Regelungskommandos:

- Zugewiesene Regelungskommandos werden ausgeführt
- Verfügbarkeit regelbarer Knoten wird aktualisiert

3. Aktualisierung der Leistung

- Bestimmung der aktuellen Leistung der Netzknoten

4. Punkteberechnung

- auf Basis der angeforderten und bereitgestellten Leistung
- Anteil erneuerbarer Energieformen



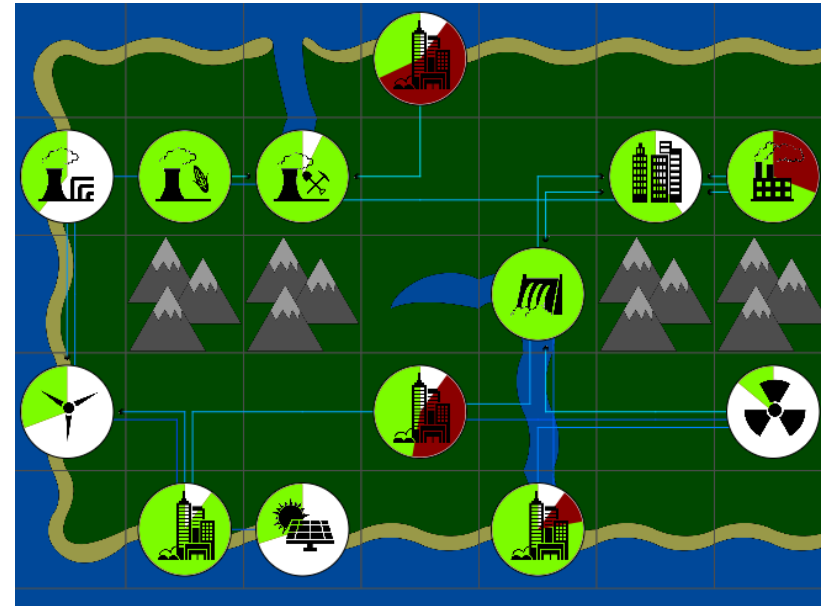
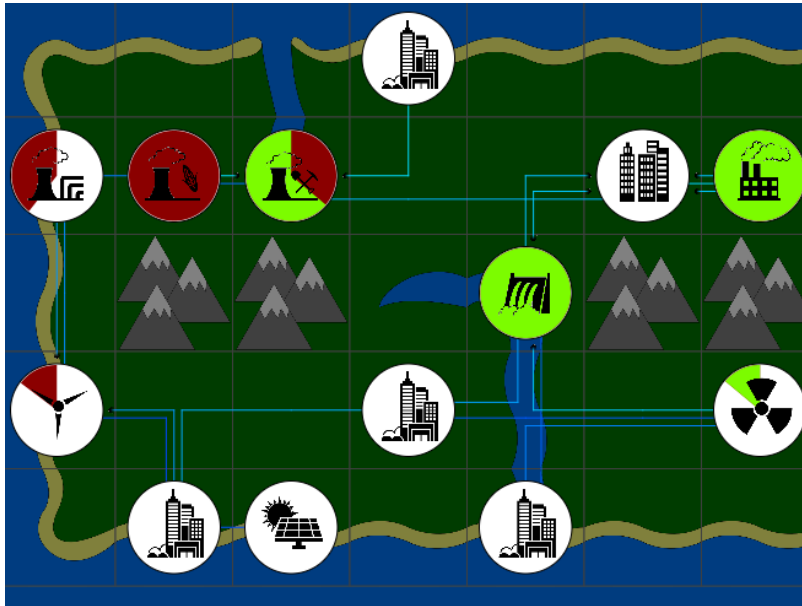
Das Auslösen eines **Regelungskommandos** (**AdjustmentCommand**) bewirkt die Veränderung

- der angeforderten Leistung von Konsumenten des Typs **IndustrialPark** oder
- der bereitgestellten Leistung von Produzenten der Typen **BioGasFiredPowerPlant**, **CoalFiredPowerPlant**, **GasFiredPowerPlant**, und **NuclerarPowerPlant**

Die **Regelungsdauer** und **-kosten** (→ *Spielanleitung: Tab. 8*)

- bemessen sich in der Anzahl der benötigten Runden (inklusive der aktuellen Runde) in Abhängigkeit des Knotentyps
- In der ersten Runde weisen alle Regelungskommandos eine Regelungsdauer von 1 auf
- Die Zuweisung von Regelungskommandos verursacht typspezifische Kosten

Auslastung von Produzenten und Konsumenten



- Keine Auslastung (weiß)
- abgegebene / erhaltene Leistung (grün)
- Differenz zu bereitgestellter/angeforderter Leistung (rot)

Punkteberechnung (Scoring)

Die erzielten Punkte werden pro Runde ermittelt und setzen sich aus folgenden Faktoren zusammen (→ *Spielanleitung: Tab. 8*):

- + Einnahmen für konsumierte Leistungs-Einheiten
- Kosten für produzierte Leistungs-Einheiten
- Kosten für Überkapazitäten bei Produzenten
- Kosten für Unterversorgung von Konsumenten
- Kosten für Regelungskommandos



Ende des Spiels

Die Ausführungsphase ist rundenbasiert und endet

1. nach Ablauf einer vorgegebenen Rundenzahl, oder
2. vorzeitig, falls weder in der Bauphase, noch in den ersten 24 Runden der Ausführungsphase eine Ausbau- bzw. Regelungsaktivität ausgeführt wurde.

Gewinner des Spiels ist dasjenige Team, das

- in der Summe (Bauphase, Ausführungsphase) die höchste Punktzahl verbuchen kann
- und (bei Gleichstand) zum Erreichen dieser Punktzahl weniger Rechenzeit benötigt.



- Einordnung des Praktikums
- Vorstellung des Frameworks EVS
- Zeitplan, Organisatorisches
- Erste Schritte
- Aufgabenblock 1

An der Durchführung des Praktikums sind folgende Personen beteiligt.

Mitarbeiter des FG Echtzeitsysteme:

- Malte Lochau (Frontalveranstaltungen und Sprechstunde)
- Markus Weckesser (Übungsbetrieb)

Tutoren:

- Fabian Rösch
- Ivan Kliasheu
- Jonas Schulz
- Peter Förster
- Puria Izady
- Ralf Mäder
- Richard Säuberlich
- Tobias Neumann

Organisatorische Fragen an: sopra@es.tu-darmstadt.de

Ablauf:

- 6 Aufgabenblöcke (AB) mit unterschiedlichen Aufgaben
- rechtzeitige Abgabe Ihrer Lösung in Moodle
- wöchentliche Frontalveranstaltung
 - Haupttermine: Vorstellung der Aufgaben
 - Zusatztermine: Wiederholung von Java-Grundlagen

Aufgaben:

- Entwicklungsumgebung: Eclipse
- Bearbeitung im Team
- Eine gemeinsame Abgabe für das gesamte Team per Upload auf Moodle



Softwarepraktikum WS17/18

Richtlinien



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Abgabe der Übungen:

- Per Upload in Moodle bis zum angegebenen Datum
- Lösung ist als **ZIP**-Datei hochzuladen
- Formate: Erlaubte Dateien in der zip-Datei sind: **.txt, .java, .xml**
 - Andere Formate werden **nicht** bewertet
- Dateiname enthält **Gruppe, Team** und **Aufgabenblock** in der Form **GnnTmABk.zip** (z.B. **G02T6AB1.zip**)



Abgabetermine:

AB1: 02.11.2017, 23:59

AB2: 16.11.2017, 23:59

AB3: 30.11.2017, 23:59

AB4: 14.12.2017, 23:59

AB5: 11.01.2018, 23:59

AB6: 25.01.2018, 23:59



- Die Prüfungsanmeldung für das Softwarepraktikum ist bereits geöffnet und die Anmeldung muss **jetzt** erfolgen
- Nur wenn die Anmeldung erfolgt ist, können die Abgaben der Übungen angenommen werden, da es sich um einen Teil der Prüfungsleistung handelt

- **Praktikumsaufgaben**

- 50% der Note
- 240 Punkte maximal erreichbar
- → 200 Punkte maximal zur Notenbildung berücksichtigt

- **Klausur**

- 50% der Note
- 50 Punkte insgesamt
- Klausur muss bestanden werden

- **Gesamtpunktzahl =**
$$\boxed{\text{Übungspunkte} \cdot 1/4 \text{ (maximal } 200 / 4 = 50 \text{ Pkt)}} + \boxed{\text{Klausurpunkte (maximal 50 Pkt)}}$$

Beispiel: 212 Übungspunkte, 45 Klausurpunkte

→ $200 \text{ Punkte} / 4 + 45 \text{ Punkte} = 95 \text{ Punkte}$ in der Veranstaltung



Voraussetzung:

- Account im ET-Pool

Einteilung:

- ca. 12 Übungsgruppen
- max. 6 Teams je Übungsgruppe
- 3 oder 4 Studierende je Team

Betreuung:

- wöchentlich in den Gruppenterminen durch die Tutoren
- Anwesenheit zum jeweils ersten Termin nach Ausgabe eines neuen Aufgabenblocks ist **verpflichtend** (max. eine Ausnahme) und wird durch die Tutoren kontrolliert
- Sprechstunde: nach Vereinbarung

Gruppentermine

	Dienstag		Mittwoch		Freitag	
	S3 21 1		S3 21 1		S3 21 1	
14-16 Uhr	Gruppe 1 Tobias	Gruppe 3 Fabian	Gruppe 5 Richard/Jonas	Gruppe 7 Ivan	Gruppe 9 Puria	Gruppe 11 Peter/Ralf
16-18 Uhr	Gruppe 2 Tobias	Gruppe 4 Fabian	Gruppe 6 Richard/Jonas	Gruppe 8 Ivan	Gruppe 10 Puria	(Gruppe 12)

Erklärungen zum Zeitplan (siehe nächste Folie)

- AB: Aufgabenblock (Frontaltermin)
- RE: Repetitorium (Frontaltermin)
- PT: Pflichttermin (Pool)
- FT: Freiwilliger Termin (Pool)
- DL: Abgabetermin (Moodle)



Zeitplan

	Dienstag	Mittwoch	Donnerstag	Freitag
16.10. – 20.10.				AB1, PT1
23. 10.– 27.10.	PT1	PT1		RE1, FT1
30.10. – 03.11.	FT1	FT1	DL1	AB2, PT2
06.11. – 10.11.	PT2	PT2		RE2, FT2
13.11. – 17.11.	FT2	FT2	DL2	AB3, PT3
20.11. – 24.11.	PT3	PT3		RE3, FT3
27.11. – 01.12.	FT3	FT3	DL3	AB4, PT4
04.12. – 08.12.	PT4	PT4		FT4
11.12. – 15.12.	FT4	FT4	DL4	AB5, PT5
18.12. – 22.12.	PT5	PT5		FT5
08.01. – 12.01.	FT5	FT5	DL5	AB6, PT6
15.01.- 19.01.	PT6	PT6		FT6
22.01. – 26.01.	FT6	FT6	DL6	
29.01. – 02.02.				Abschluss



Aufgabenblöcke

20.10.2017	EVS Framework, Eclipse, Util-Methoden, Javadoc, JUnit
03.11.2017	Kosten ermitteln, Comparator, Sortieren
17.11.2017	Flussgraphen, Residualgraphen
01.12.2017	Wegesuche, Ford-Fulkerson Algorithmus
15.12.2017	Bauphase, Energienetz-Analyse
12.01.2018	Gesamtalgorithmus
25.01.2018	Abgabe Gesamtlösung
02.02.2018	Abschlusswettbewerb
Termin folgt	Klausur



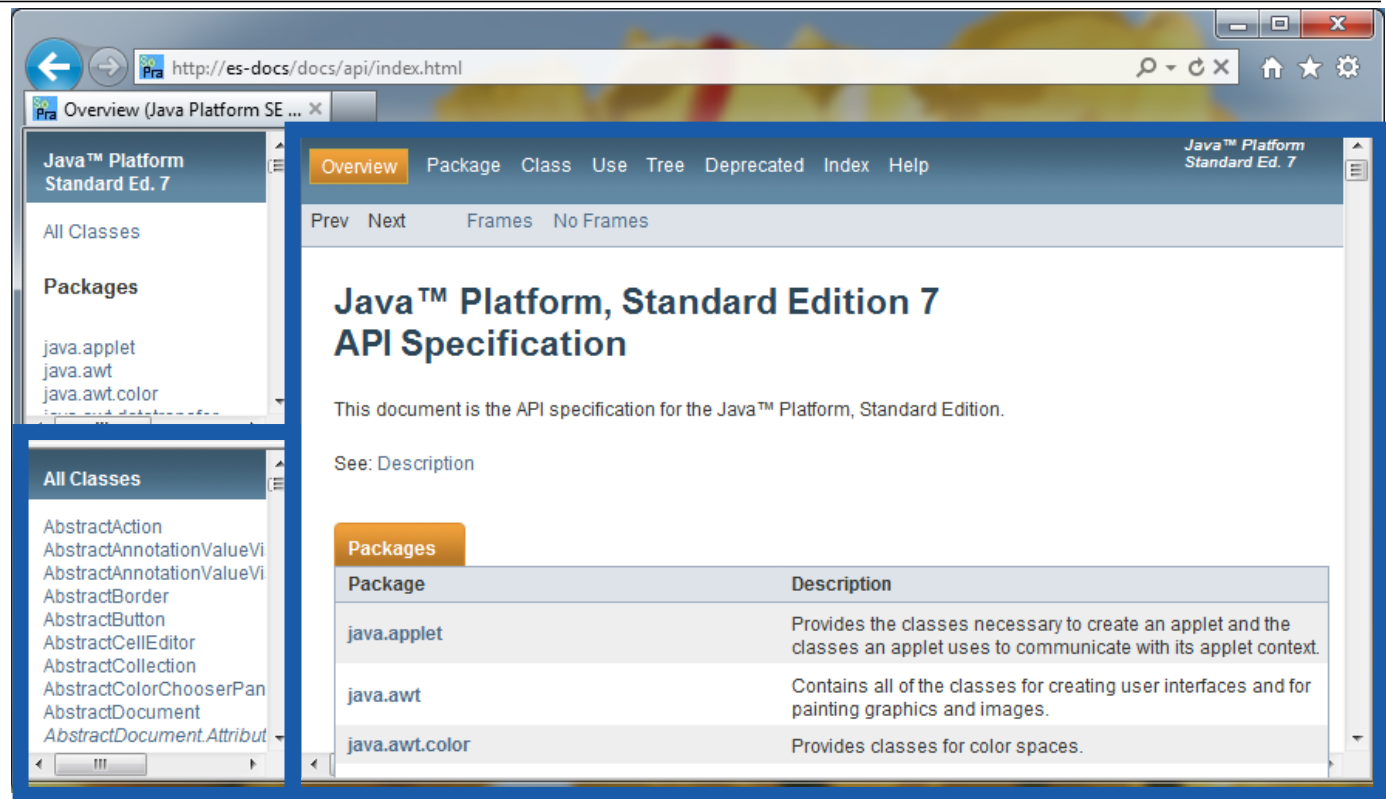
- Einordnung des Praktikums
- Vorstellung des Frameworks EVS
- Zeitplan, Organisatorisches
- Erste Schritte
- Aufgabenblock 1

Voraussetzungen und Hinweise

- Kenntnis und Verständnis der Inhalte aus AI1 wird vorausgesetzt
- Das Buch „[Java ist auch eine Insel](#)“, ist auf der Website des Rheinwerk-Verlags frei einsehbar und ist sehr nützlich als Nachschlagewerk
- Die Dokumente „inoffizielle Tipps“ und „Tutorials“ auf Moodle geben weitere Hinweise und erklären die verwendeten Grundlagen



Java Dokumentation



Java™ Platform Standard Ed. 7

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.

Liste aller
Klassen und
Schnittstellen

<https://docs.oracle.com/javase/8/docs/api/>

Details der aktuellen Auswahl
(Pakete, Klassen, Schnittstellen)



Javadocs zum Rahmenwerk

JavaScript is disabled on your browser.

org.sopra 1.0.1 API

OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES ALL CLASSES

org.sopra 1.0.1 API

Packages

Package	Description
org.sopra.api	
org.sopra.api.command	
org.sopra.api.exercises	
org.sopra.api.exercises.exercise1	
org.sopra.api.exercises.exercise2	
org.sopra.api.exercises.exercise3	
org.sopra.api.exercises.exercise4	
org.sopra.api.exercises.exercise5	
org.sopra.api.model	
org.sopra.api.model.consumer	
org.sopra.api.model.producer	
org.sopra.api.util.scenarioloader	

OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES ALL CLASSES

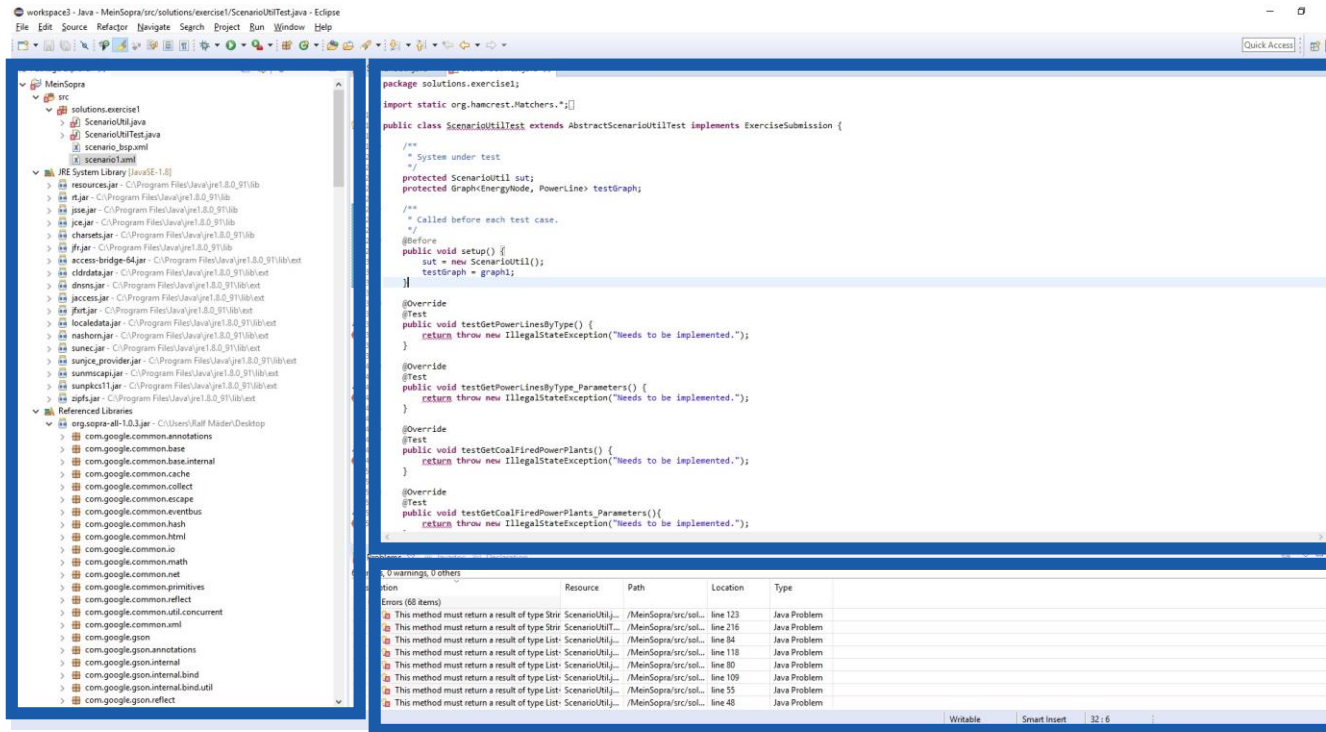
All Classes

- AbstractEnergyNetworkAnalyzer
- AbstractFlowGraphTest
- AbstractFordFulkersonTest
- AbstractQuickSortTest
- AbstractQuickSortTest.IntegerComparator
- AbstractScenarioUtilTest
- AbstractSunEnergyBroker
- AdjustConsumerCommand
- AdjustmentCommand
- AdjustProducerCommand
- BioGasFiredPowerPlant
- BuildPowerPlantCommand
- CannotAssignCommandException
- CannotExecuteCommandException
- City
- CoalFiredPowerPlant
- Command
- CommandFactory
- CommercialPark
- ConstructionCostCalculator
- Consumer
- ConsumerType
- Controllable
- ControllableConsumer
- ControllableProducer
- CostCalculatorFactory
- Edge
- EnergyNode
- EnergyNodeConfig
- ExerciseSubmission
- FlowEdge
- FlowGraph
- FordFulkerson
- Game
- GasFiredPowerPlant
- Graph
- HydroPowerPlant
- IndustrialPark
- Multiplicator
- NuclearPowerPlant
- PlantLocation

→ auf Moodle
zu finden



Entwicklungsumgebung: Eclipse



Editor mit
Quelltexten

Projekt-Dateien: verschiedene Ansichten: Errors, Javadoc, etc.
z.B. Klassen → konfigurierbar



Eclipse bietet viele **Möglichkeiten:**

- Debugging
- übersichtliche Dateistruktur
- integrierte JavaDocs
- Auto-Vervollständigung (z.B. Methodennamen)
- ...und vieles mehr



Machen Sie sich mit Eclipse vertraut!

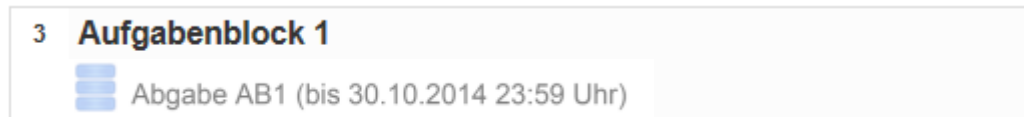
→komfortableres Programmieren, Zeitersparnis



Moodle: Abgabe der Lösungen (1/3)

Link zur Moodle-Website

1) Passende *Abgabeaktivität* wählen:



2) Eigene *Teamnummer* ist ausgewählt! (Ansonsten Lösung nicht hochladen!)

Abgabe AB1 (bis 30.10.2014 23:59 Uhr)

Abgabe bis 30.10.2014 23:59 Uhr

Hier bitte eine zip-Datei mit der Lösung zu AB1 mit dem Dateinament **GxxTyABz.zip** (xx = Gruppennummer, y = Teamnummer, z = Arbeitsblattnummer) hochladen.

Erlaubte Dateien in der zip-Datei sind: **.txt, .java, .xml**

Es wird die letzte hochgeladene Version bewertet.

Getrennte Gruppen

3) *Eintrag hinzufügen* wählen.



Moodle: Abgabe der Lösungen (2/3)

4) *Datei reinziehen* (Format: GxxTyyABzz):





Getrennte Gruppen Gruppe 03: Team 1

[Listenansicht](#) [Einzelansicht](#) [Suche](#) [Eintrag hinzufügen](#) [Export](#) [Vorlagen](#) [Felder](#) [Vorlagensätze](#)


Neuer Eintrag

Datei:

Maximale Größe für neue Dateien: 2MB, Maximale Zahl von Anhängen: 1



► Dateien



Bewegen Sie Dateien in dieses Feld (Drag&Drop)

Kommentar:

[Sichern und anzeigen](#) [Sichern und weitere hinzufügen](#)

5) *Kommentar einfügen*
(optional).

6) *Speichern* der Datei in Moodle.

Moodle: Abgabe der Lösungen (3/3)

6) Abgabe überprüfen (stimmen Gruppe und Team?, ist die zip-Datei die richtige?)

Abgabe AB1 (bis 03.11.2016 23:59 Uhr)

Abgabe bis 03.11.2016 23:59 Uhr

Hier bitte eine zip-Datei mit dem gesamten Projekt ihrer Lösung zu AB1 mit dem Dateinamen **GxxTyABz.zip** (xx = Gruppennummer, y = Teamnummer, z = Arbeitsblattnummer) hochladen.

Es wird die letzte hochgeladene Version bewertet.

Getrennte Gruppen

[Listenansicht](#) [Einzelansicht](#) [Suche](#) [Eintrag hinzufügen](#) [Export](#) [Vorlagen](#) [Felder](#) [Vorlagensätze](#)

Seite: ([Zurück](#)) [1](#) [2](#)

Titel: Abgabe von Ralf Siegmund Mäder am Freitag, 21. Oktober 2016, 08:00 eingereicht

Datei:  G15T6AB1.zip

Kommentar:

Moodle: Bewertungen einsehen

7) Bewertungen auswählen

EINSTELLUNGEN

▼ Kurs-Administration

Abmelden aus

'Softwarepraktikum

18-su-1020-pr WiSe

2014/15'

Bewertungen



SoPa-Übungen					
Arbeitsblatt 1					
1.1	1.2	1.3	1.4	Summe für Arbeitsblatt 1	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	
-	-	-	-	-	



Szenario-Beschreibung – XML Format

```
<scenario>
<playfield>
  <elements>
    <element type="beach" xpos="0" ypos="0" />
    <element type="beach" xpos="1" ypos="0" />
    <element type="beach" xpos="2" ypos="0" />
    <element type="beach" xpos="0" ypos="1" />
    <element type="grassland" xpos="1" ypos="1" />
    <element type="beach" xpos="2" ypos="1" />
    <element type="beach" xpos="0" ypos="2" />
    <element type="beach" xpos="1" ypos="2" />
    <element type="beach" xpos="2" ypos="2" />
  </elements>
</playfield>

<graph>
  <nodes>
    <node id="0" type="hydropowerplant"
name="Wasserkraftwerk1" xpos="1" ypos="0" />
    <node id="1" type="city" name="Stadt1" xpos="1" |
ypos="1" />
  </nodes>
  <edges>
    <edge type="lowVoltage">
      <first reference="0"/>
      <second reference="1"/>
    </edge>
  </edges>
</graph>

</scenario>
```

Kacheln auf dem Spielfeld

Producer und Consumer

Leitungen



Aufbau einer Java Datei

```
package solutions.exercisel;  
  
import org.sopra.api.Scenario;  
import org.sopra.api.model.Graph;  
  
/**  
 * This class implements methods to get basic data from a scenario.  
 * @since 1.0  
 */  
public class ScenarioUtil implements ExerciseSubmission {  
  
    /**  
     * Returns information of the team.<p>  
     * @return text with team information  
     * @since 1.0  
     */  
    public String getTeamIdentifier() {  
        return „G19T9“;  
    }  
}
```

} Paket

} Import von
Klassen

} Kurze Doku.
der Klasse

} Start der
Klasse

} Kurze Doku.
einer Methode

} Vollständige
Methode

Dokumentation im Quelltext

Kommentare

- separate Kommentarzeilen mit //
- nicht jede Codezeile kommentieren
- kleinere Codeblöcke kommentieren

JavaDoc

- Informationen in `/** */` Blöcken
- Klasse: Beschreibung, Autor(en), Version
- Variable: Beschreibung
- Konstante: Beschreibung
- Methode: Beschreibung, Parameter, Resultat

Alle von Ihnen selbstgeschriebenen Quelltexte müssen Javadoc-Informationen enthalten.

Im Praktikum werden nur die folgenden Tags verwendet:

@author	<i>name-text</i>
@param	<i>parameter-name description</i>
@return	<i>description</i>
@version	<i>version-text</i>

Dokumentation einer Klasse:

```
/**
 * This is a class to practice debugging.
 * @author   Markus Schmidt
 * @version  1.01
 */
public class MyDebug extends Debug {
    :
}
}
```

Dokumentation eines Attributs:

```
/**  
 * The String name of the team.  
 */  
protected String name;
```

Dokumentation einer Methode:

```
/**  
 * Returns information of the team.<p>  
 * @return text with team information  
 * @since 1.0  
 */  
public String getTeamIdentifier() {  
    return „G19T9“;  
}
```

Quelltext:

```
/**
 * Computes the maximum flow in a flow network graph from
 * source s to target t.
 *
 * @param graph
 *         The graph representing the flow network. Flows along edges in
 *         this graph will be set to max flows.
 * @param start
 *         The source node.
 * @param target
 *         The target node.
 */

public void findMaxFlow(FlowGraph<V, FlowEdge<V>> graph, V start, V target)
```

HTML Browser:

findMaxFlow

```
void findMaxFlow(FlowGraph<V,FlowEdge<V>> graph,
                 V start,
                 V target)
```

Computes the maximum flow in a flow network graph from source s to target t.

Given a flow network and a pair of nodes s and t, produces a maximum s-t in that network.

Parameters:

graph - The graph representing the flow network. Flows along edges in this graph will be set to max flows.

start - The source node.

target - The target node.

Throws:

java.lang.IllegalArgumentException - If any parameter is null.

java.util.NoSuchElementException - If s or t are not nodes in the graph.

Wieso Testen?

Typische “Gegenargumente“

- ~~keine Zeit zum Testen~~ ⇒ Teufelskreis
- ~~langweilig~~ ⇒ kreativer Prozess
- ~~Code sowieso fehlerfrei~~ ⇒ Code ist (fast) nie fehlerfrei

Welcher Test?

- Performanztest
- Lasttest
- Unit Test
- Integrationstest

Im Praktikum werden Unit Tests verwendet

Was ist Testen?

- Erstellen und Ausführen von **Unit-Testfällen**
- Ein Unit-Testfall ist eine experimentelle Ausführung einer Methode unter kontrollierten Bedingungen und erwartetem/überprüfbarem Ergebnis

Testen mit JUnit

1. Importieren von `org.junit.Test`;
2. Jede beliebige Methode kann als Testfall markiert werden
3. Markierung durch Annotation `@Test` vor der Methode
4. Methoden sind immer `public void` und haben **keine Parameter**
5. Vorgabe: Namenskonvention für Testfall-Methoden `test_<Beschreibung>`.

Testfall



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
package solutions.exercise3;
import org.junit.Test;
import static org.junit.Assert.*;

public class FlowGraphTest extends AbstractFlowGraphTest implements
ExerciseSubmission {

    @Test
    public void test_edgesFrom(){

        // initialize SUT (attribute of superclass) with test data
        initSutWithTestData();

        // Check structure
        assertEquals(sut.edgesFrom("s").size(), 2);
    }
}
```

→ Tutorial auf Moodle!



Testfallsammlung in Testklassen

- Testfallmethoden können beliebig auf Klassen verteilt werden
- Konvention: Sammeln von Testfallmethoden in Testklassen
<Name>Test.java
- Spezielle Methoden zum Vor-/Nachbereiten von Testfallausführungen
 - Setup-Aufgabe: Initialisieren von zu testenden Datenstrukturen
 - Teardown-Aufgabe: Freigabe von Ressourcen
- 1. **@Before** und **@After** markieren Setup- und Teardown-Methoden, die vor/nach jeder Testfallausführung ausgeführt werden.
- 2. **@BeforeClass** und **@AfterClass** markieren Setup- und Teardown-Methoden, die einmalig für eine Testklasse ausgeführt werden.
Diese müssen **public static void** sein.

Namenskonvention:

```
@Before public void setUp()
```

```
@After public void tearDown()
```



Testklasse



```
package solutions;
import org.junit.Test;
import static org.junit.Assert.*;
public class SolutionNameTest {
    private SolutionName implSolutionName1;
    private SolutionName implSolutionName2;

    @Before public void setUp() {
        implSolutionName1 = new SolutionName();
        implSolutionName2 = Null;
    }

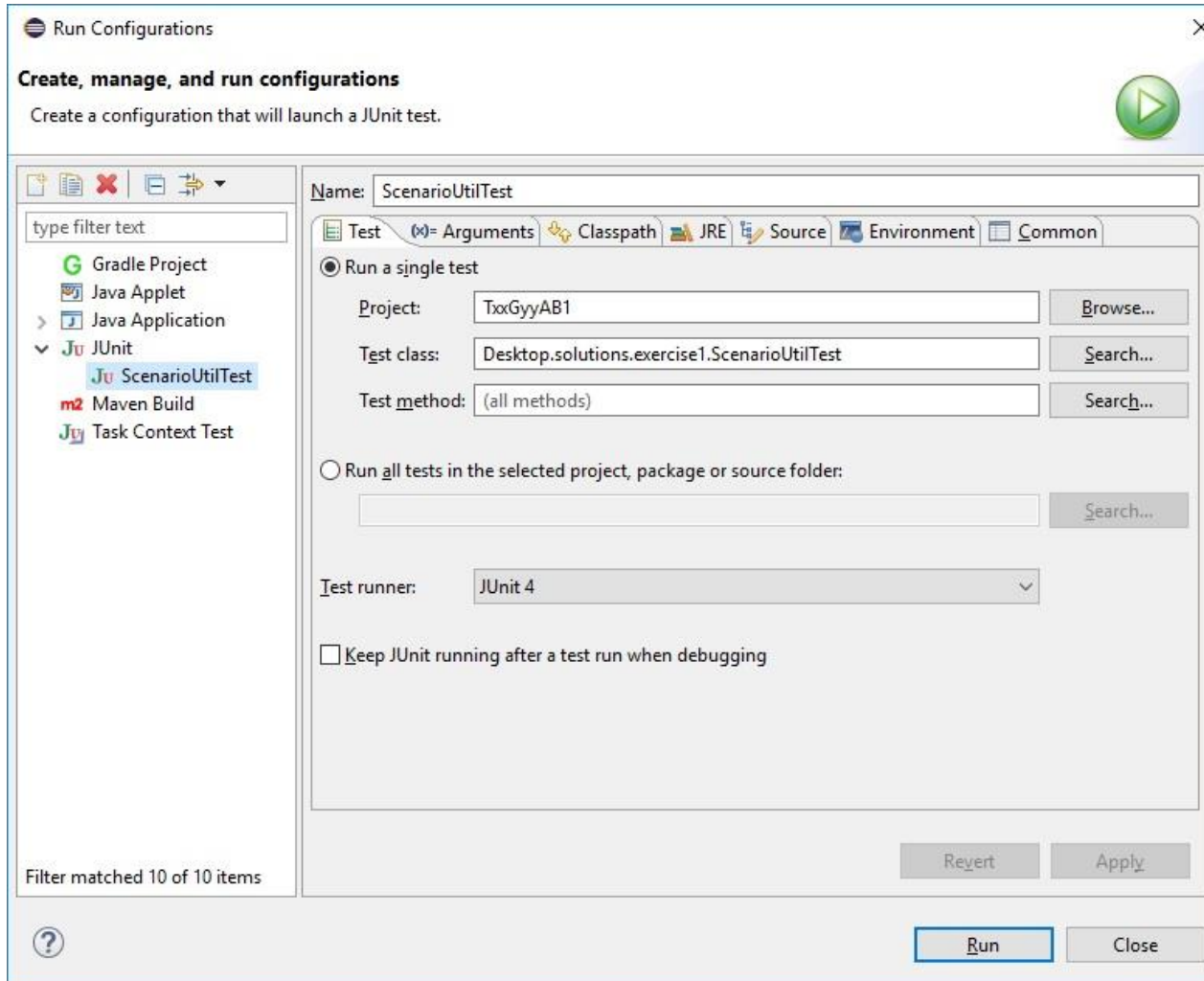
    @After public void tearDown() {
    }

    @Test public void test_Scenario0_null() {
        assertTrue(implSolutionName1.getTeamIdentifier() == null);
    }




    @Test public void test_Scenario1_null() {
        assertNotNull(implSolutionName2);
    }
}
```

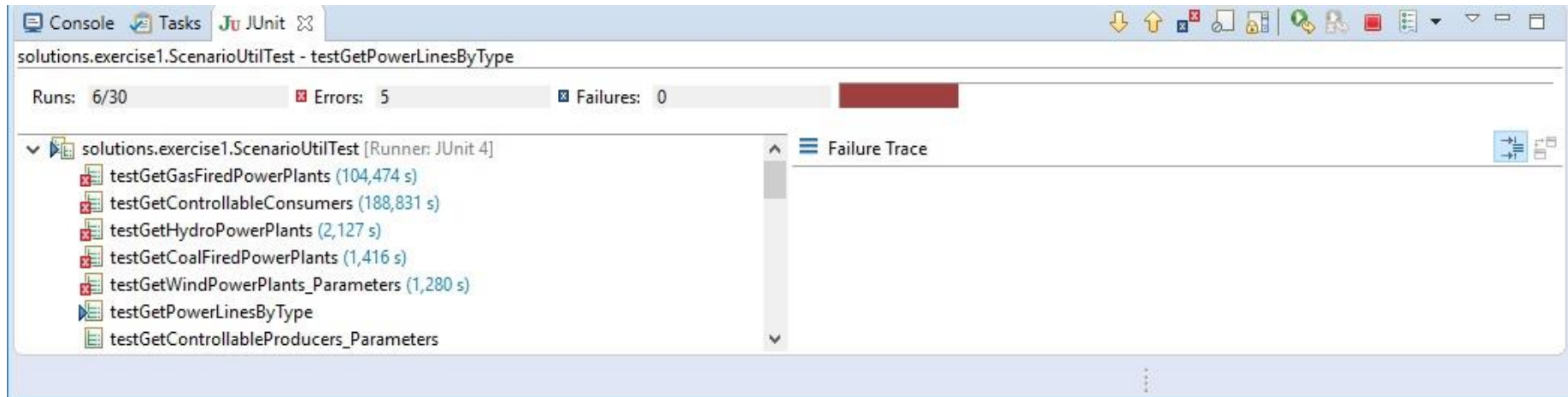


Testen mit JUnit




Testen mit JUnit

-  Success – Assertion liefert true
-  Failure – Assertion liefert false
-  Error – unerwarteter Fehler im Code (z.B. unerwartete Exception)



Testen von Lösungen

 Exercise 1

Execute Test Cases

Exercise 1

Aufgabe 1.1 - Szenario erstellen

passed

Aufgabe 1.2 - Szenario Hilfsmethoden

failed

- FAILED: Expected exception type

Expected: an instance of java.lang.IllegalArgumentException
but: <java.lang.Error: Unresolved compilation problem:
Unreachable code
> is a java.lang.Error

- FAILED: null

Aufgabe 1.3 - Szenario Hilfsmethoden testen

passed



- Die Tests dienen als Bewertungsgrundlage, allerdings wird der Code zusätzlich von den Tutoren kontrolliert
- Eine Anleitung zum Ausführen der internen Tests in JUnit ist auf Moodle verfügbar

- Einordnung des Praktikums
- Vorstellung des Frameworks EVS
- Zeitplan, Organisatorisches
- Erste Schritte
- Aufgabenblock 1

Aufgabenblock 1

- 1.0. Entwicklungsumgebung einrichten
- 1.1. Szenario erstellen
- 1.2. Hilfsmethoden implementieren
- 1.3. Testen der Szenario Hilfsmethoden
- 1.4. Erstellen einer EnergyNode-Hierarchie