Höhere Programmierkonzepte
Praktikum II

Prof. Dr. Nikolaus Wulff

10. – 19. Nov 2020

1 *Axela* - Engine

```
/**
 * Axela solves problems with help of registered Iris solver instances.
 */
public interface Axela {
  /**
   * Solve the problem with help of the given fact(s), using a registered
   * Iris solver implementation.
   */
  default <Problem,Facts,Solution>
  Solution solve(Problem problem, Facts facts) {
    if(hasSolverFor(problem,facts)) {
      Iris <Problem,Facts,Solution> solver = findSolverFor(problem,facts);
      Solution solution = solver.solve(problem,facts);
      return solution;
    }
    throw new IllegalArgumentException("no solver found");
  }
  /**
   * Register a Iris solver within the Axela engine.
   */
  <P,F,S> void registerSolver(P problem, F facts, Iris<P,F,S> solver);
  /**
   * Check if a solver for the given problem and fact(s) is registered.
   */
  <P,F,S> boolean hasSolverFor(P p, F f);
  /**
   * Find a pre-registered solver for the given problem and fact(s).
   */
  <P,F,S> Iris<P,F,S> findSolverFor(P p, F f);
}
```

Listing 1: Auszug der wichtigsten *Axela* Methoden.

Im ersten Praktikum haben Sie sich mit der *Iris* Schnittstelle beschäftigt und Zeichenketten konkateniert. Nun geht es darum die *Axela* Schnittstelle zu implementieren und höherwertige *Iris* Solver in einer `AxelaEngine` als Kontainer zu registrieren und ablaufen zu lassen. Die Schnittstelle (1), ein kleiner Auszug aus der zum Praktikum I in der Zip-Datei mit zur Verfügung gestellten *Axela* - Schnittstelle, zeigt die wichtigsten Methoden, die hierfür notwendig sind. *Iris* Solver lassen sich registrieren und wieder auffinden an Hand des Problems und der Faktenlage. Die `solve` Methode zeigt eine Blaupause für eine mögliche Implementierung, sicherlich fehlen noch NullPointer Checks oder ähnliche Details. In der obersten `solve` Methode sind die generischen Platzhalter `<Problem, Facts, Solution>` vollkommen analog zur *Iris* - Schnittstelle kodiert, in den weiteren Funktionen habe ich diese der Übersicht halber und aus typischer "Programmierer Schreibfaulheit" nur noch als `<P, F, S>` abgekürzt – dies hat aber vollkommen dieselbe Bedeutung und Funktionalität. Sie wissen Namen und Bezeichner sind Schall und Rauch - sollten aber dennoch möglichst sprechend und selbsterklärend gewählt werden. Bei *java.util.Map*`<K,V>` "K" für Key und "V" für Value und hier `<P,F,S>` für `<Problem, Facts, Solution>`. Nach der Einführung in Java Generics sollten Sie nun mit diesem Vorgehen und der Syntax vertraut sein – oder zumindest bald werden.

Aufgabe

Es gilt die *Axela* - Engine in einem weiteren Kindprojekt **Axela.Engine** zu entwickeln. Diese Separation hat den Vorteil, dass sie jetzt arbeitsteilig in separaten Teilprojekten vorgehen können und sich nicht gegenseitig im Git behindern. Einer entwickelt die Engine der/die Andere weitere *Iris* Implementierungen.

1. Legen Sie im Elternprojekt „*Axela* " das Teilprojekt `Axela.Engine` an. In diesem Projekt wird im Paket *de.lab4inf.axela.engine* die Klasse `AxelaEngine` entwickelt, welche die `Axela` Schnittstelle aus dem Axlea.Core Projekt implementiert. D.h. die Methoden der Schnittstelle müssen ausprogrammiert und mittels eines von **Ihnen zu erstellenden JUnit** `AxelaEngineTest` sinnvoll(!) getestet werden.

2. Innerhalb eines weiteren Teilprojekts `Axlea.Math` entwickeln Sie fünf *Iris* Solver, welche die "beiden Probleme" PLUS und MULT für Vektoren $x, y \in \mathbb{R}^n$ und Matrizen $A, B \in \mathbb{R}^{n \times m}$ als Operationen $x + y$, $x * y$, $A + B$ und $A * B$ sowie $A * x$ lösen[1].

   Spannend ist hier die Frage nach der Schnittstelle, was sind geeignete Belegungen/Bindungen für `<P,F,S>`? Wie findet *Axela* den richtigen

[1] beachten Sie den Hinweis zur Modellierung von Vektoren und Matrizen!

Solver für PLUS oder MULT wenn mal Vektoren, ein anderes Mal Matrizen oder auch ein Mix aus beidem übergeben wird? Wir werden uns in der Übung am Mittwoch ausführlich darüber austauschen müssen, um den Umgang mit Generics weiter zu vertiefen.

3. Für all diese relativ kleinen separaten *Iris* Solver sind sinnvolle JUnit Tests erforderlich.

4. Ein besonderes Augenmerk ist darauf zu richten, dass die `AxelaEngine` in der Lage ist den richtigen Solver zu finden - natürlich am besten mit JUnit Test unterfüttert.

Hinweise

Vektoren $x \in \mathbb{R}^n$ können am einfachsten als `double[]` Feld und Matrizen $A \in \mathbb{R}^{n \times m}$ als doppelt indiziertes Feld `double[][]` modelliert werden,d.h. ohne weitere, neue Klassen Vektor oder Matrix, was sehr aufwändig wäre. Mit diesem einfachen Ansatz sind sogar primitive Java Datentypen als Bindung für Generics möglich.

Auf die Vektor und Matrizen Solver sollten Sie entsprechend viel Energie und Sorgfalt verwenden und diese ausreichend testen und debuggen. Wir werden diese in späteren Praktika parallelisieren und benötigen dann die seriellen Solver Varianten, um die Parallelimplementierung mit ihnen zu testen und den Geschwindigkeitszuwachs vergleichen zu können.

Bereiten Sie sich entsprechend auf die Mittwochs-Übung und das Praktikum vor und scheuen Sie sich nicht **rechtzeitig** Fragen im Forum zu stellen, dann können wir all die vielen Unwägbarkeiten dieser Aufgabenstellung im Vorwege klären. Zu einem "echten Informatikprojekt" gehört die Anforderungsanalyse gemeinsam mit dem Auftragsgeber abzusprechen, um nicht am Thema vorbei zu entwickeln. Dies ist neben dem Programmieren eine der Kernkompetenzen, die eine Informatikausbildung vermitteln sollte.