Algorithmen und Datenstrukturen

Der Experimentierkasten

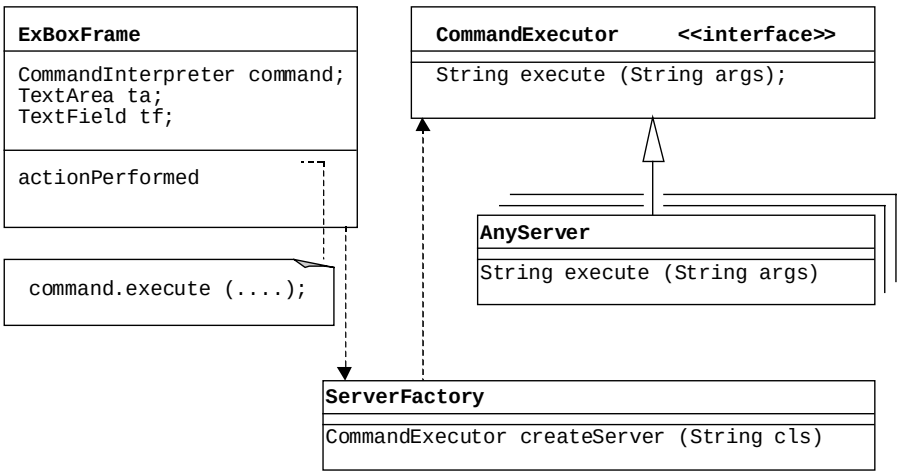
In diesem Semester werden wir eine Reihe von Algorithmen kennenlernen und mit diesen Experimente durchführen.

Wir benötigen dafür einen geeigneten "Experimentierkasten", für die Eingabe der Aufrufparameter und für textuelle und graphische Ausgaben. Dieser wird in einem sogenannten Client-Server-Arrangement als Plug-in Architektur realisiert. Der Client stellt dabei die Benützerschnittstelle zur Verfügung und der Server enthält die Experimente, d.h. die Implementierung der Lösung der Aufgaben. Der Client ist so entworfen, dass beliebige Server nachgeladen werden können (wir werden in den folgenden Aufgaben immer wieder neue Server schreiben). Der Client muss dafür:

- Die Schnittstelle kennen d.h. wie der Server aufgerufen werden kann. Die Schnittstelle wird als *Interface* `CommandExecutor` realisiert, welches vom Server implementiert wird, wie z.B. in `AnyServer`.
- Beliebige Server nachträglich nachladen, bzw. Klassen instanzieren können, von denen lediglich der Namen (als String) bekannt ist. Diese Funktionalität wird von `ServerFactory` zur Verfügung gestellt.

Für die Programmierung selber empfiehlt Ihnen der Dozent IntelliJ oder NetBeans. Sie können auch jede andere Ihnen gewohnte IDE/Editor verwenden (Notepad++, vi, emacs, Eclipse aber *NICHT* BlueJ[1]). Sie können die Projekte und Tests (JUnit4) auch in Ihrer IDE ausführen, oder ein Build Tool Ihrer Wahl[2] verwenden: z.B. ANT, Maven, Gradle, …Um die vorbereiteten JUnit4-Tests vor der Abgabe lokal durchführen zu können, werden `hamcrest-core.*.jar` und `JUnit4.*.jar` benötigt.

Klassendiagramm



[1] Falls Sie den *unbedingten Wunsch* verspüren, BlueJ verwenden *zu müssen*, dann konsultieren Sie bitte vorgängig folgende ZHAW Stelle: https://www.zhaw.ch/de/studium/waehrend-des-studiums/beratung/coaching-und-psychologische-beratung/

[2] Top IDE Index https://pypl.github.io/IDE.html

Beschreibung der Funktionalität

AnyServer stellt einen Beispiel-Server dar, welcher das Interface *CommandExecutor* implementiert. Letzteres ist das Interface, das von allen Servern implementiert werden muss. In jedem Praktikum werden Sie einen oder mehrere solcher Server implementieren.


```
public interface CommandExecutor {
  public String execute(String args);
}

public class AnyServer implements CommandExecutor {
   public String execute (String args) {
     // Experiement
   }
}
```

Bedienung/Befehle

Das Hauptfenster des GUI-Client enthält zwei Bereiche: 1) eine Textzeile für die Eingabe der Befehle und Argumente und 2) ein Textfenster, das die Resultate der ausgeführten Befehle anzeigt.

Menu-File

- *Open …:* öffnet ein File und übergibt den Inhalt der execute Methode
- *Exit:* beendet die ExBox

Menu-Server

- *Connect:* Verbindungsaufnahme zu einem Server, der das Interface CommandExecutor implementiert.
  - In der Klasse ServerFactory wird ein Objekt der Klasse deren Namen im Argument der Methode createServer übergeben wird instanziert:
- *Test…:* lädt einen JUnit Test und führt diesen aus.
- *Test:* führt den letzten Test nochmals aus

Menu-View

- *Clear:* löscht den Inhalt der Textbox
- *Text*: schaltet in die Textansicht
- *Graphic*: schaltet in die Graphikansicht

Knopf-Enter

- Ruft die *execute* Methode des Servers auf. Die von dieser Methode zurückgegebenen Daten übernimmt der Client und stellt sie im Text- oder Graphikfenster dar.

Knopf-Run

- Ruft die *execute* Methode des Servers kontinuierlich 10 mal pro Sekunde auf.

Dropdown-History

- Die Eingaben können so einfach wiederholt werden