



Jail++

Rail to JBC Compiler in C++

Präsentiert von Jail Constructions Ltd. (auch bekannt als C++ Gruppe)

Jail Constructions Ltd.
Freie Universität Berlin

Softwareprojekt Übersetzerbau, 2010





Organisation



Figure : Scrum Master und Product Owner





Übersicht:



Übersicht:

- ▶ Anforderungen, Ziele und Prozesse



Organisation

Übersicht:

- ▶ Anforderungen, Ziele und Prozesse
- ▶ Team-Aufteilung



Organisation

Übersicht:

- ▶ Anforderungen, Ziele und Prozesse
- ▶ Team-Aufteilung
- ▶ Aufgetretene Probleme und gefundene Lösungen



Organisation

Übersicht:

- ▶ Anforderungen, Ziele und Prozesse
- ▶ Team-Aufteilung
- ▶ Aufgetretene Probleme und gefundene Lösungen
- ▶ Verwendete Tools

Organisation

Übersicht:

- ▶ Anforderungen, Ziele und Prozesse
- ▶ Team-Aufteilung
- ▶ Aufgetretene Probleme und gefundene Lösungen
- ▶ Verwendete Tools
- ▶ Vorstellung der Compiler-Pipeline





- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode
(selbstgewähltes Target)

- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode
(selbstgewähltes Target)
- ▶ Rail ist eine zweidimensionale, esoterische Programmiersprache



- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode
(selbstgewähltes Target)
- ▶ Rail ist eine zweidimensionale, esoterische Programmiersprache
- ▶ Der komplette Sprachumfang soll umgesetzt werden



- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode (selbstgewähltes Target)
- ▶ Rail ist eine zweidimensionale, esoterische Programmiersprache
- ▶ Der komplette Sprachumfang soll umgesetzt werden
- ▶ Dieses übergeordnete Ziel wurde unterteilt in 3 Meilensteine:

- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode (selbstgewähltes Target)
- ▶ Rail ist eine zweidimensionale, esoterische Programmiersprache
- ▶ Der komplette Sprachumfang soll umgesetzt werden
- ▶ Dieses übergeordnete Ziel wurde unterteilt in 3 Meilensteine:
 - ▶ **MS 1** grundlegendes Parsen der Schienen, strings und Konsolenausgabe

- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode (selbstgewähltes Target)
- ▶ Rail ist eine zweidimensionale, esoterische Programmiersprache
- ▶ Der komplette Sprachumfang soll umgesetzt werden
- ▶ Dieses übergeordnete Ziel wurde unterteilt in 3 Meilensteine:
 - ▶ **MS 1** grundlegendes Parsen der Schienen, strings und Konsolenausgabe
 - ▶ **MS 2** arithmetische Operationen, Verzweigungen (If-Else) und Variablen

- ▶ Bau eines Compilers für Rail in C++ zu Java Bytecode (selbstgewähltes Target)
- ▶ Rail ist eine zweidimensionale, esoterische Programmiersprache
- ▶ Der komplette Sprachumfang soll umgesetzt werden
- ▶ Dieses übergeordnete Ziel wurde unterteilt in 3 Meilensteine:
 - ▶ **MS 1** grundlegendes Parsen der Schienen, strings und Konsolenausgabe
 - ▶ **MS 2** arithmetische Operationen, Verzweigungen (If-Else) und Variablen
 - ▶ **MS 3** kompletter Sprachumfang Funktionsaufrufe, Rekursion, Schleifen



- #### ► Zusätzliche Anforderungen:



- ▶ Zusätzliche Anforderungen:
 - ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST

- ▶ Zusätzliche Anforderungen:

- ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
- ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe



- ▶ Zusätzliche Anforderungen:

- ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
- ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe
- ▶ **IDE** Eine für Rail spezialisierte Entwicklungsumgebung soll umgesetzt werden



- ▶ Zusätzliche Anforderungen:
 - ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
 - ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe
 - ▶ **IDE** Eine für Rail spezialisierte Entwicklungsumgebung soll umgesetzt werden
- ▶ Prozess ist Scrum-ähnlich:



- ▶ Zusätzliche Anforderungen:

- ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
- ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe
- ▶ **IDE** Eine für Rail spezialisierte Entwicklungsumgebung soll umgesetzt werden

- ▶ Prozess ist Scrum-ähnlich:

- ▶ Meilensteine (Sprints)



- ▶ Zusätzliche Anforderungen:
 - ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
 - ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe
 - ▶ **IDE** Eine für Rail spezialisierte Entwicklungsumgebung soll umgesetzt werden
- ▶ Prozess ist Scrum-ähnlich:
 - ▶ Meilensteine (Sprints)
 - ▶ Standup-Meetings



- ▶ Zusätzliche Anforderungen:
 - ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
 - ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe
 - ▶ **IDE** Eine für Rail spezialisierte Entwicklungsumgebung soll umgesetzt werden
- ▶ Prozess ist Scrum-ähnlich:
 - ▶ Meilensteine (Sprints)
 - ▶ Standup-Meetings
 - ▶ Retrospektiven (am Ende der Meilensteine)



- ▶ Zusätzliche Anforderungen:
 - ▶ **Cross-Kompatibilität** Erstellung eines austauschbaren Dateiformats für den AST
 - ▶ **Cross-Testing** Testen der Compiler der zwei Gruppen mit Ast-Dateien der jeweils anderen Gruppe
 - ▶ **IDE** Eine für Rail spezialisierte Entwicklungsumgebung soll umgesetzt werden
- ▶ Prozess ist Scrum-ähnlich:
 - ▶ Meilensteine (Sprints)
 - ▶ Standup-Meetings
 - ▶ Retrospektiven (am Ende der Meilensteine)
 - ▶ Rollen wie Scrum Master und Product Owner



Verwendete Tools



Verwendete Tools

- ▶ Teamspeak



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber
- ▶ Mail



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber
- ▶ Mail
- ▶ Github als Codebase



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber
- ▶ Mail
- ▶ Github als Codebase
- ▶ Keine festgelegte Entwicklungsumgebung für C++



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber
- ▶ Mail
- ▶ Github als Codebase
- ▶ Keine festgelegte Entwicklungsumgebung für C++
- ▶ Jenkins und Gradle für Continuous Integration



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber
- ▶ Mail
- ▶ Github als Codebase
- ▶ Keine festgelegte Entwicklungsumgebung für C++
- ▶ Jenkins und Gradle für Continuous Integration
- ▶ Qt für die Entwicklung des Editors



Verwendete Tools

- ▶ Teamspeak
- ▶ Jabber
- ▶ Mail
- ▶ Github als Codebase
- ▶ Keine festgelegte Entwicklungsumgebung für C++
- ▶ Jenkins und Gradle für Continuous Integration
- ▶ Qt für die Entwicklung des Editors
- ▶ Doxygen für automatisierte Dokumentationserzeugung

Organisation: Team-Aufteilung

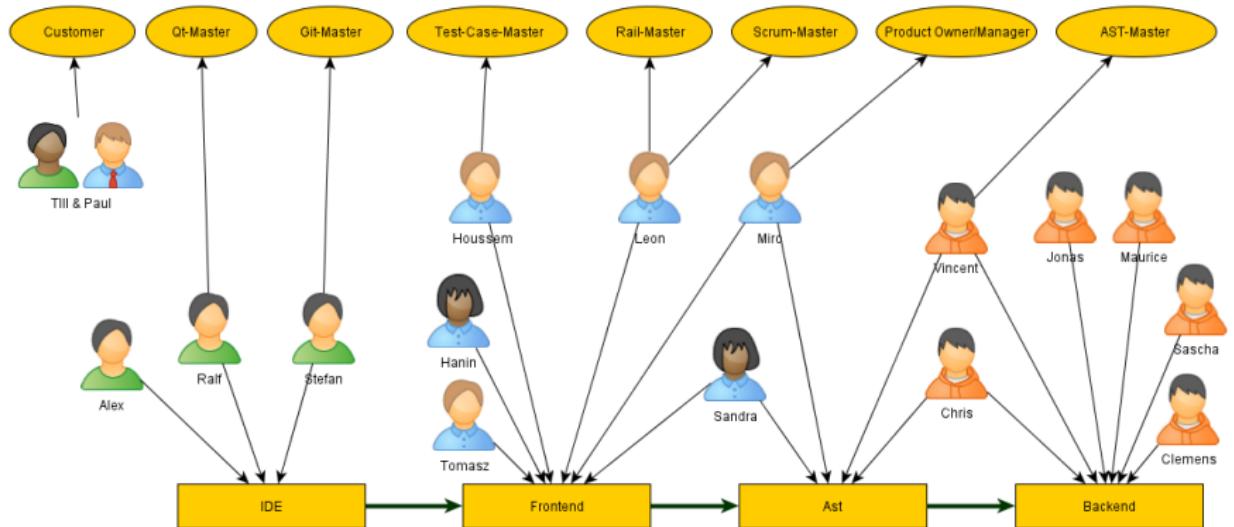


Figure : Organigramm

Organisation: Aufgetretene Probleme und gefundene Lösungen





- ▶ Probleme in MS1:



- ▶ Probleme in MS1:
 - ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)



- ▶ Probleme in MS1:

- ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)
- ▶ unklare bzw. ungleiche Arbeitsaufteilung



- ▶ Probleme in MS1:
 - ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)
 - ▶ unklare bzw. ungleiche Arbeitsaufteilung
- ▶ Verbessert durch:

- ▶ Probleme in MS1:

- ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)
- ▶ unklare bzw. ungleiche Arbeitsaufteilung

- ▶ Verbessert durch:

- ▶ Einführung von Daily-Scrums → 3 Treffen pro Subteam wöchentlich (Voice-Chat)



- ▶ Probleme in MS1:
 - ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)
 - ▶ unklare bzw. ungleiche Arbeitsaufteilung
- ▶ Verbessert durch:
 - ▶ Einführung von Daily-Scrums → 3 Treffen pro Subteam wöchentlich (Voice-Chat)
 - ▶ Scrum-Master und Product Owner sind bei allen Treffen anwesend → besserer Gesamtüberblick

- ▶ Probleme in MS1:
 - ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)
 - ▶ unklare bzw. ungleiche Arbeitsaufteilung
- ▶ Verbessert durch:
 - ▶ Einführung von Daily-Scrums → 3 Treffen pro Subteam wöchentlich (Voice-Chat)
 - ▶ Scrum-Master und Product Owner sind bei allen Treffen anwesend → besserer Gesamtüberblick
 - ▶ Nutzung der Github-Issues zur Verbesserung und Klarstellung der Aufgabenverteilung

- ▶ Probleme in MS1:
 - ▶ Kommunikation (kein einheitliches Tool, seltene Treffen)
 - ▶ unklare bzw. ungleiche Arbeitsaufteilung
- ▶ Verbessert durch:
 - ▶ Einführung von Daily-Scrums → 3 Treffen pro Subteam wöchentlich (Voice-Chat)
 - ▶ Scrum-Master und Product Owner sind bei allen Treffen anwesend → besserer Gesamtüberblick
 - ▶ Nutzung der Github-Issues zur Verbesserung und Klarstellung der Aufgabenverteilung
 - ▶ Jabber für die spontane Kommunikation

Vorstellung der Compiler-Pipeline



Figure : Compiler-Pipeline und Subteams





Figure : Team Frontend





Übersicht:



Übersicht:

- ▶ Softwaretechnik



Übersicht:

- ▶ Softwaretechnik
- ▶ Technische Aspekte



Übersicht:

- ▶ Softwaretechnik
- ▶ Technische Aspekte





Frontend: Softwaretechnik

- ▶ Teammitglieder: 6



Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei



Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt

Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt
- ▶ Tätigkeitsfelder:

Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt
- ▶ Tätigkeitsfelder:
 - ▶ Lexer

Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt
- ▶ Tätigkeitsfelder:
 - ▶ Lexer
 - ▶ Parser



Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt
- ▶ Tätigkeitsfelder:
 - ▶ Lexer
 - ▶ Parser
 - ▶ AST Serialisierung/Deserialisierung



Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt
- ▶ Tätigkeitsfelder:
 - ▶ Lexer
 - ▶ Parser
 - ▶ AST Serialisierung/Deserialisierung
 - ▶ Rail-Testcases



Frontend: Softwaretechnik

- ▶ Teammitglieder: 6
- ▶ Austausch mit Haskell-Gruppe: Design der AST-Datei
- ▶ Kommunikation wie im Organisationsteil erwähnt
- ▶ Tätigkeitsfelder:
 - ▶ Lexer
 - ▶ Parser
 - ▶ AST Serialisierung/Deserialisierung
 - ▶ Rail-Testcases
 - ▶ Testing-Framework



- ▶ Wichtige Erkenntnisse



- ▶ Wichtige Erkenntnisse
 - ▶ Gelernte Theorie lässt sich nur begrenzt anwenden



- ▶ Wichtige Erkenntnisse
 - ▶ Gelernte Theorie lässt sich nur begrenzt anwenden
 - ▶ Praktische Umstände verlangen individuelle Lösungen



- ▶ Wichtige Erkenntnisse

- ▶ Gelernte Theorie lässt sich nur begrenzt anwenden
- ▶ Praktische Umstände verlangen individuelle Lösungen
- ▶ Ein guter Entwurf spart enorm Aufwand



```
§ 'main' (--) :  
\\  
\\  
\\----[a]-----\\  
|  
|  
# |  
\\|  
\\|  
\\ / -----\\  
\\| / -----\\  
/-----*-----/ /  
| / \\ /  
\\ @ / -----  
-----
```

Figure : Grammatik hierfür?

Frontend: Technisches

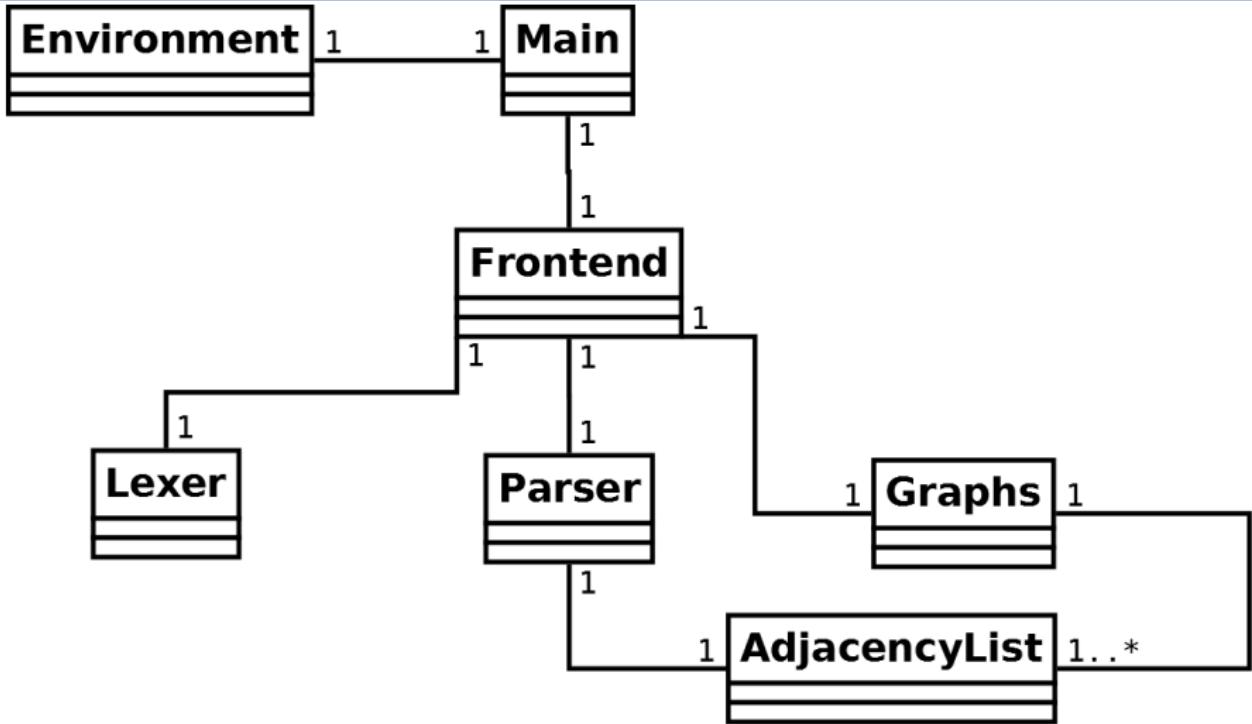


Figure : Klassendiagramm Frontend



Frontend: Technisches

Hauptarbeit geschieht im Parser:

Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen

Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:
 - ▶ Position



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:
 - ▶ Position
 - ▶ Richtung



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:
 - ▶ Position
 - ▶ Richtung
- ▶ Verknüpfung dieser Informationen mit dem Knoten verhindert mehrfachparsen



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:
 - ▶ Position
 - ▶ Richtung
- ▶ Verknüpfung dieser Informationen mit dem Knoten verhindert mehrfachparsen
- ▶ Verzweigungen durch rekursive Selbstaufrufe



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:
 - ▶ Position
 - ▶ Richtung
- ▶ Verknüpfung dieser Informationen mit dem Knoten verhindert mehrfachparsen
- ▶ Verzweigungen durch rekursive Selbstaufrufe
- ▶ Für anonyme Funktionen(Lambdas) werden neue Funktionsnamen generiert und die Funktionen geparst



Frontend: Technisches

Hauptarbeit geschieht im Parser:

- ▶ Durchlaufen der Funktion anhand der Schienen
- ▶ Aufbau einer Graph-Struktur durch den Parser
- ▶ Für jeden gesehenen Rail-Befehl wird gespeichert:
 - ▶ Position
 - ▶ Richtung
- ▶ Verknüpfung dieser Informationen mit dem Knoten verhindert mehrfachparsen
- ▶ Verzweigungen durch rekursive Selbstaufrufe
- ▶ Für anonyme Funktionen(Lambdas) werden neue Funktionsnamen generiert und die Funktionen geparst
- ▶ Erzeugt für jede Rail-Funktion einen Graphen



Backend

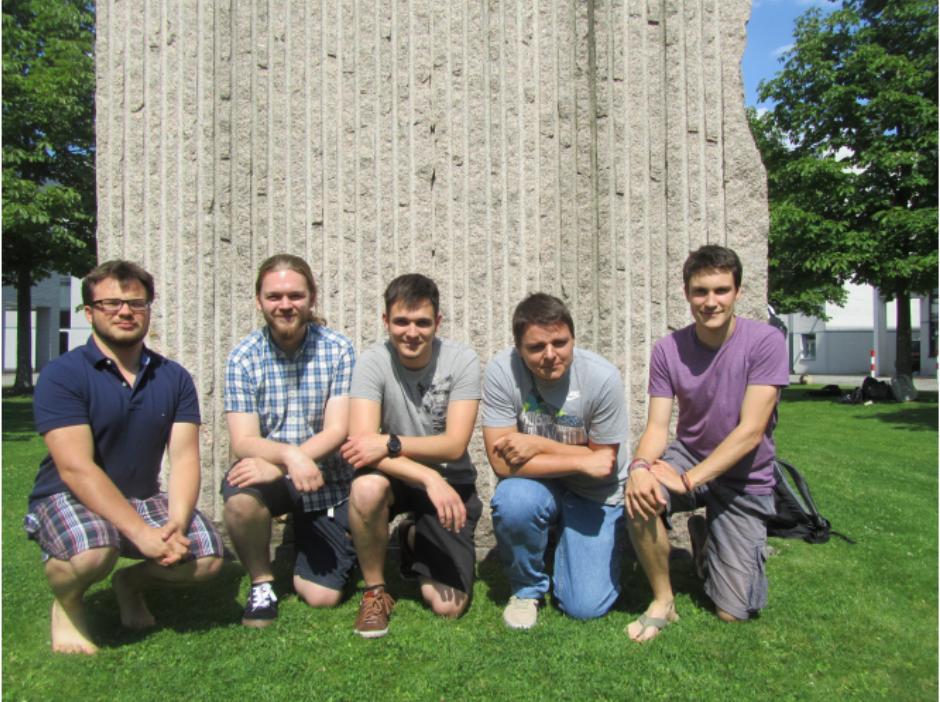


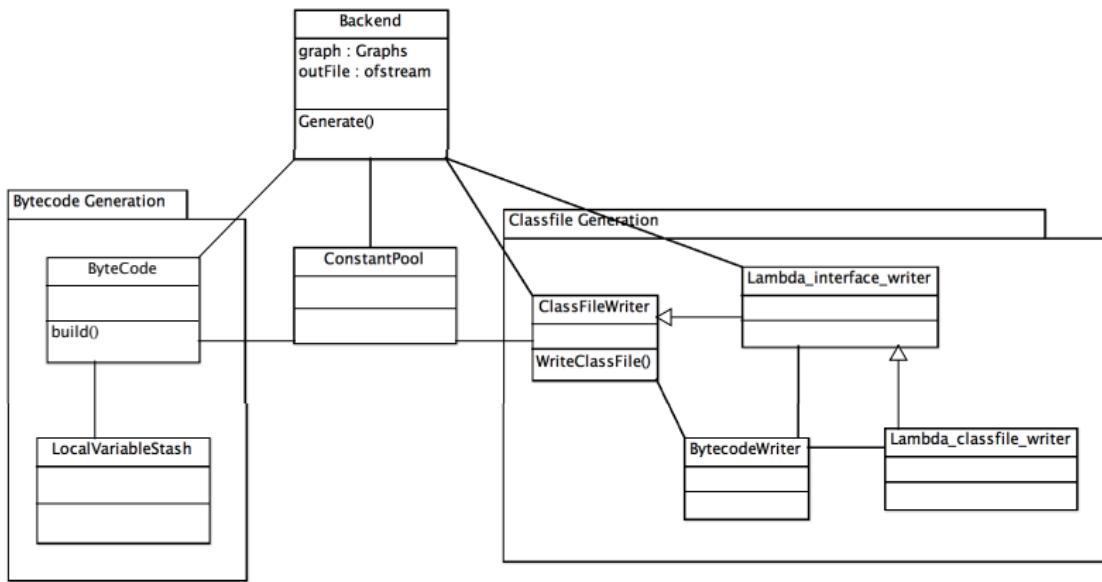
Figure : Team Backend

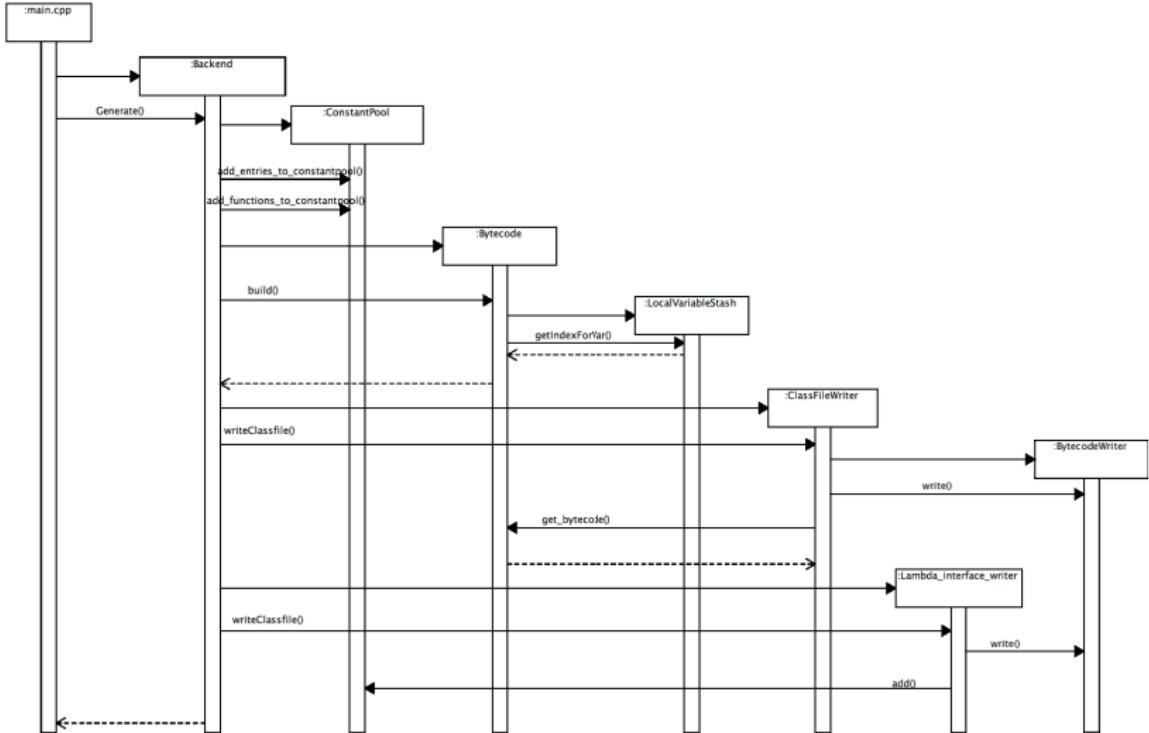


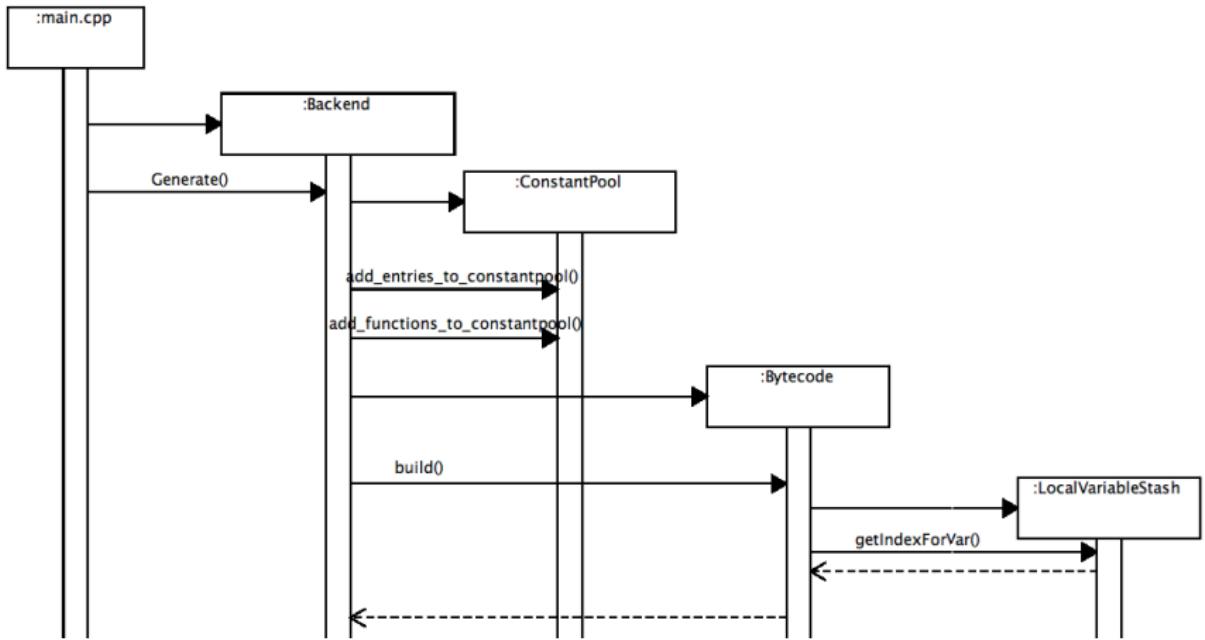
JVM Bytecode und Classfile

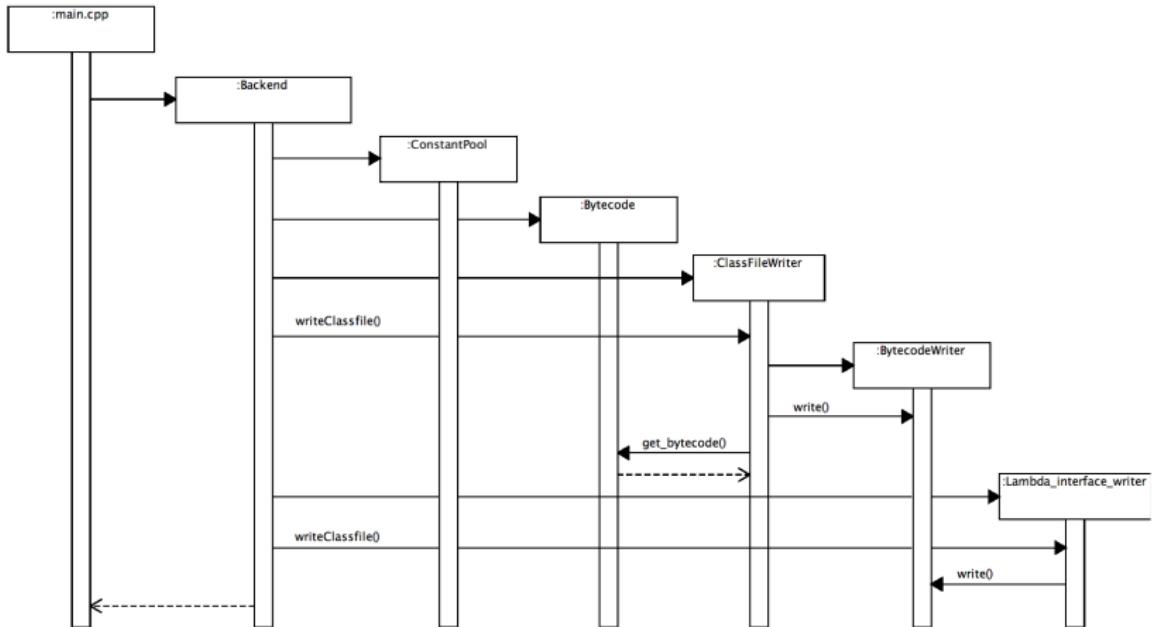
```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello World");
    }
}
```

Backend Klassendiagramm













Organisation



Organisation

- ▶ hauptsächlich Teamspeak und Git Kommentarbereich



Organisation

- ▶ hauptsächlich Teamspeak und Git Kommentarbereich
- ▶ Außerdem XMPP und E-Mail



Organisation und Probleme

Organisation

- ▶ hauptsächlich Teamspeak und Git Kommentarbereich
- ▶ Außerdem XMPP und E-Mail

Probleme



Organisation und Probleme

Organisation

- ▶ hauptsächlich Teamspeak und Git Kommentarbereich
- ▶ Außerdem XMPP und E-Mail

Probleme

- ▶ Classfiles waren anfangs kompliziert zu durchblicken



Organisation und Probleme

Organisation

- ▶ hauptsächlich Teamspeak und Git Kommentarbereich
- ▶ Außerdem XMPP und E-Mail

Probleme

- ▶ Classfiles waren anfangs kompliziert zu durchblicken
- ▶ Das Classfile-Gerüst war schwerer zu realisieren als die grundlegenden Funktionen



Organisation und Probleme

Organisation

- ▶ hauptsächlich Teamspeak und Git Kommentarbereich
- ▶ Außerdem XMPP und E-Mail

Probleme

- ▶ Classfiles waren anfangs kompliziert zu durchblicken
- ▶ Das Classfile-Gerüst war schwerer zu realisieren als die grundlegenden Funktionen
- ▶ Einteilung der Milestones war für uns kontraproduktiv



Beispiel

Zum Abschluss:

- ▶ Eine von uns generierte Classfile





Figure : Team Rail-Editor



Übersicht:

- ▶ Softwaretechnik
- ▶ Technische Aspekte
- ▶ Live-Demo



Rail-Editor: Softwaretechnik

- ▶ kleines Team → Kommunikation per Email
- ▶ direkte Reaktionen auf Emails (an alle geschickt)
- ▶ zuerst zwei, nach dem ersten Milestone drei Mitglieder
 - ▶ bessere Arbeitsverteilung
- ▶ Teilnahme an Daily Scrums (montags und mittwochs)
- ▶ zusätzliche Team-Meetings außer donnerstags
 - ▶ produktives Arbeiten durch Pair-Programming
- ▶ gute Kommunikation innerhalb des Teams
- ▶ anfangs spärliche Kommunikation mit anderen Teams



Rail-Editor: Technische Aspekte I

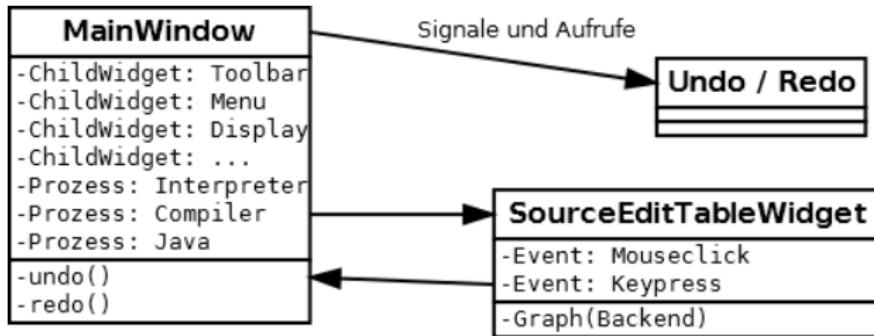
- ▶ QT für Programmierung der grafischen Benutzeroberfläche
- ▶ Signal- und Slottechnik
- ▶ Eventverarbeitung für Maus- und Tastendrücke
- ▶ Basisklassen abgeleitet und Funktionalitäten erweitert
- ▶ Graphstruktur für Syntax-Highlighting
 - ▶ internes Backend
- ▶ Smart-Cursor und Grab-Modus
 - ▶ für intuitives Schreiben von Quellcode
 - ▶ siehe Live-Demo



Rail-Editor: Technische Aspekte II

- ▶ Main-Window als *Brain*
 - ▶ Weiterleitung an Child-Widgets
- ▶ Undo-Redo-Funktionalität
 - ▶ abstrakte Klasse
 - ▶ wird durch konkrete Aktionen implementiert
- ▶ Compiler-Einbindung
 - ▶ Funktionen: Build, Run, Stop
 - ▶ auch der Rail-Interpreter kann verwendet werden
- ▶ *Preferences* (Editor-Einstellungen)
 - ▶ persistent gespeichert

Rail-Editor: Technische Aspekte III





Fazit





Fazit

Wir hatten



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons
- ▶ Mindestens 5 Nervenzusammenbrüche wegen des c++ 11 Standards



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons
- ▶ Mindestens 5 Nervenzusammenbrüche wegen des c++ 11 Standards
- ▶ Mindestens 20 Nervenzusammenbrüche weil jemand auf die Idee gekommen ist Eclipse zu verwenden



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons
- ▶ Mindestens 5 Nervenzusammenbrüche wegen des c++ 11 Standards
- ▶ Mindestens 20 Nervenzusammenbrüche weil jemand auf die Idee gekommen ist Eclipse zu verwenden
- ▶ Noch ein paar Nervenzusammenbrüche mehr wegen der Fehlermeldungen des GCC



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons
- ▶ Mindestens 5 Nervenzusammenbrüche wegen des c++ 11 Standards
- ▶ Mindestens 20 Nervenzusammenbrüche weil jemand auf die Idee gekommen ist Eclipse zu verwenden
- ▶ Noch ein paar Nervenzusammenbrüche mehr wegen der Fehlermeldungen des GCC
- ▶ Über 20 Mal "Oh Nein, das Projekt kompiliert nicht mehr!"



Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons
- ▶ Mindestens 5 Nervenzusammenbrüche wegen des c++ 11 Standards
- ▶ Mindestens 20 Nervenzusammenbrüche weil jemand auf die Idee gekommen ist Eclipse zu verwenden
- ▶ Noch ein paar Nervenzusammenbrüche mehr wegen der Fehlermeldungen des GCC
- ▶ Über 20 Mal "Oh Nein, das Projekt kompiliert nicht mehr!"
- ▶ Mindestens 4 Wochen lang eine kaputte .class-Datei generiert

Fazit

Wir hatten

- ▶ Ziemlich viel Arbeit
- ▶ Einige Coding-Marathons
- ▶ Mindestens 5 Nervenzusammenbrüche wegen des c++ 11 Standards
- ▶ Mindestens 20 Nervenzusammenbrüche weil jemand auf die Idee gekommen ist Eclipse zu verwenden
- ▶ Noch ein paar Nervenzusammenbrüche mehr wegen der Fehlermeldungen des GCC
- ▶ Über 20 Mal "Oh Nein, das Projekt kompiliert nicht mehr!"
- ▶ Mindestens 4 Wochen lang eine kaputte .class-Datei generiert
- ▶ Zahllose Wtf-Momente



Fazit



Fazit

Außerdem haben/hatten wir:



Fazit

Außerdem haben/hatten wir:

- ▶ Viel gelernt



Fazit

Außerdem haben/hatten wir:

- ▶ Viel gelernt
- ▶ Ziemlich viel Spaß gehabt



Fazit

Außerdem haben/hatten wir:

- ▶ Viel gelernt
- ▶ Ziemlich viel Spaß gehabt
- ▶ Super Teamgeist bewiesen



Fazit

Außerdem haben/hatten wir:

- ▶ Viel gelernt
- ▶ Ziemlich viel Spaß gehabt
- ▶ Super Teamgeist bewiesen
- ▶ Regelmäßig Kekse gegessen



Fazit

Außerdem haben/hatten wir:

- ▶ Viel gelernt
- ▶ Ziemlich viel Spaß gehabt
- ▶ Super Teamgeist bewiesen
- ▶ Regelmäßig Kekse gegessen
- ▶ Also kurz: Spiel, Spaß und Schokolade

Fazit

Außerdem haben/hatten wir:

- ▶ Viel gelernt
- ▶ Ziemlich viel Spaß gehabt
- ▶ Super Teamgeist bewiesen
- ▶ Regelmäßig Kekse gegessen
- ▶ Also kurz: Spiel, Spaß und Schokolade

Also auch wenn wir das Projekt Jail++ getauft haben kam uns das Projekt nicht wie ein Gefängnisaufenthalt vor.



Fazit



Figure : Jail Constructions Ltd. (auch bekannt als C++ Gruppe)