Compilador Java 5.0 para gerar executável nativo para plataforma Palm OS

Júlio Vilmar Gesser

Orientado por Joyce Martins

Roteiro da apresentação

- Introdução
- Objetivos
- Fundamentação teórica
- Desenvolvimento
- Conclusão
- Extensões

Introdução

- Desenvolvimento de aplicativos para Palm OS em C/C++
- Por que não utilizar J2ME?
- Solução proposta

Objetivos

- Executar as análises léxica, sintática e semântica de programas Java
- Gerar código intermediário C++ e simular o coletor de lixo do Java
- Implementar em C++ algumas bibliotecas da API do Java

Fundamentação teórica

- Plataforma Java
- Compiladores
- Plataforma Palm OS
- Trabalhos correlatos

Fundamentação teórica
Plataforma Java

- Tecnologias Java se dividem em três plataformas:
  - Java Platform Micro Edition (Java ME)
    - Voltado para dispositivos móveis
  - Java Platform Standard Edition (Java SE)
    - Voltado para aplicações desktop
  - Java Platform Enterprise Edition (Java EE)
    - Voltado para aplicações corporativas e de grande porte

Fundamentação teórica
Plataforma Java

- Linguagem de programação orientada a objetos
- Programas interpretados e independentes de plataforma
- Possui coletor de lixo
- Possui API rica em utilitários e estruturas de dados

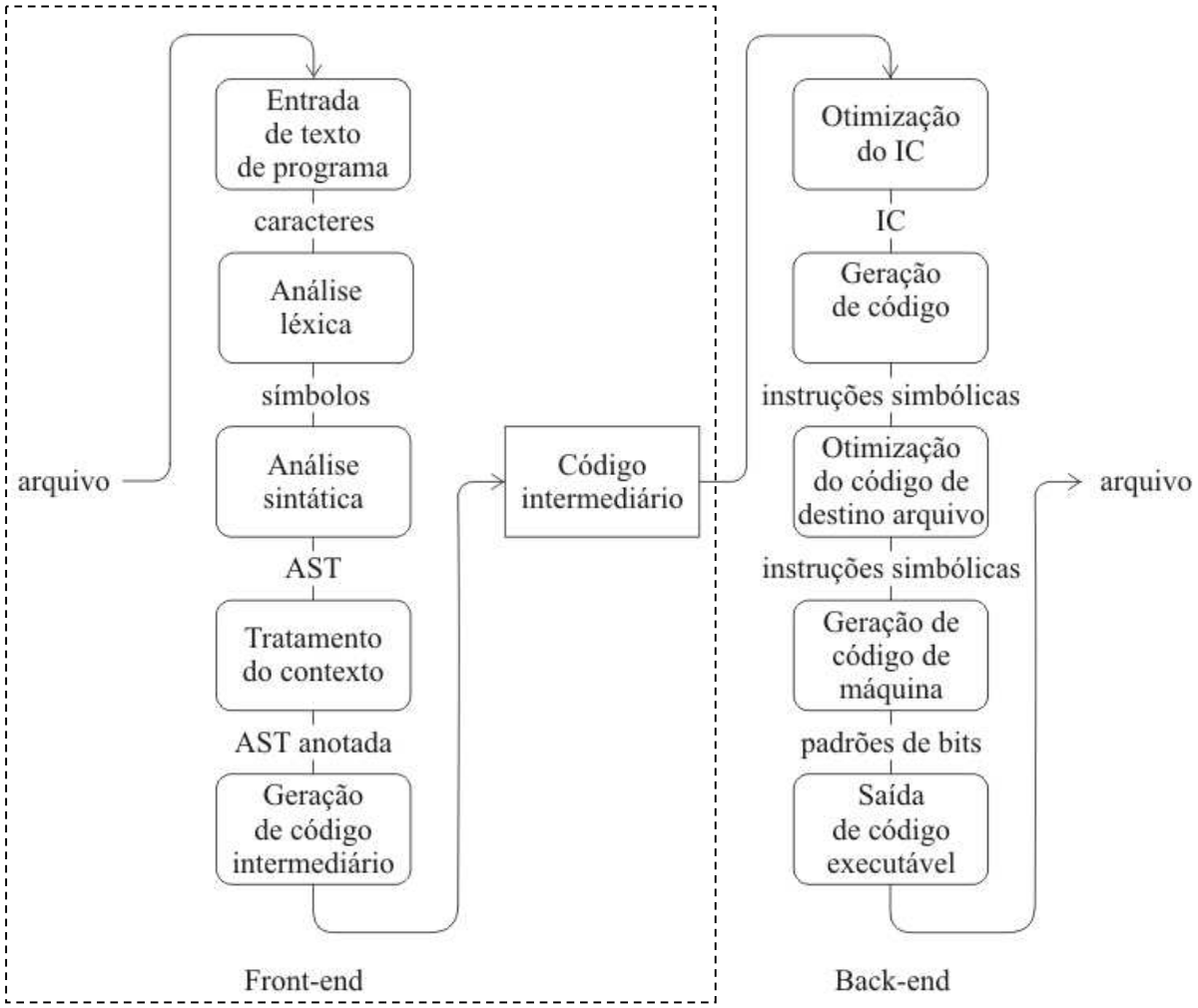
Fundamentação teórica
Compiladores

- Transformar um programa escrito em uma linguagem em um programa equivalente em outra linguagem
- Informar erros encontrados no programa fonte

Fundamentação teórica
Compiladores

- Divide-se em:
  - Front-end
    - Análise do programa fonte
    - Detecção de erros de compilação
  - Back-end
    - Síntese da linguagem alvo
    - Otimizações de código

Fundamentação teórica
Compiladores



Fundamentação teórica
Plataforma Palm OS

Fundamentação teórica
Plataforma Palm OS

- Projetada para gerenciar informação
- Facilidade de utilização e grande compatibilidade
- SO com núcleo multitarefas preemptivo, executa apenas uma aplicação por vez
- Aplicativos não podem ser *multithread*

Fundamentação teórica
Plataforma Palm OS

- Endereçamento de memória 16 bits
- Jumps relativos limitados em 32K
- Ambientes de programação
  - Palm OS Developer Suite
  - CodeWarrior for Palm OS
    - Permite endereçamento 32 bits

Fundamentação teórica
Trabalhos correlatos

- Linguagem de programação orientada a objetos para a plataforma Microsoft .NET (Leyendecker)
- Processo de desenvolvimento em palm usando C++ (Hildebrandt)
- Jump (Kleberhoff e Dickerson)
- Java2cpp (Programics.com)

Desenvolvimento

- Levantamento dos requisitos
- Especificação
- Implementação
- Estudo de caso

Desenvolvimento
Levantamento de requisitos

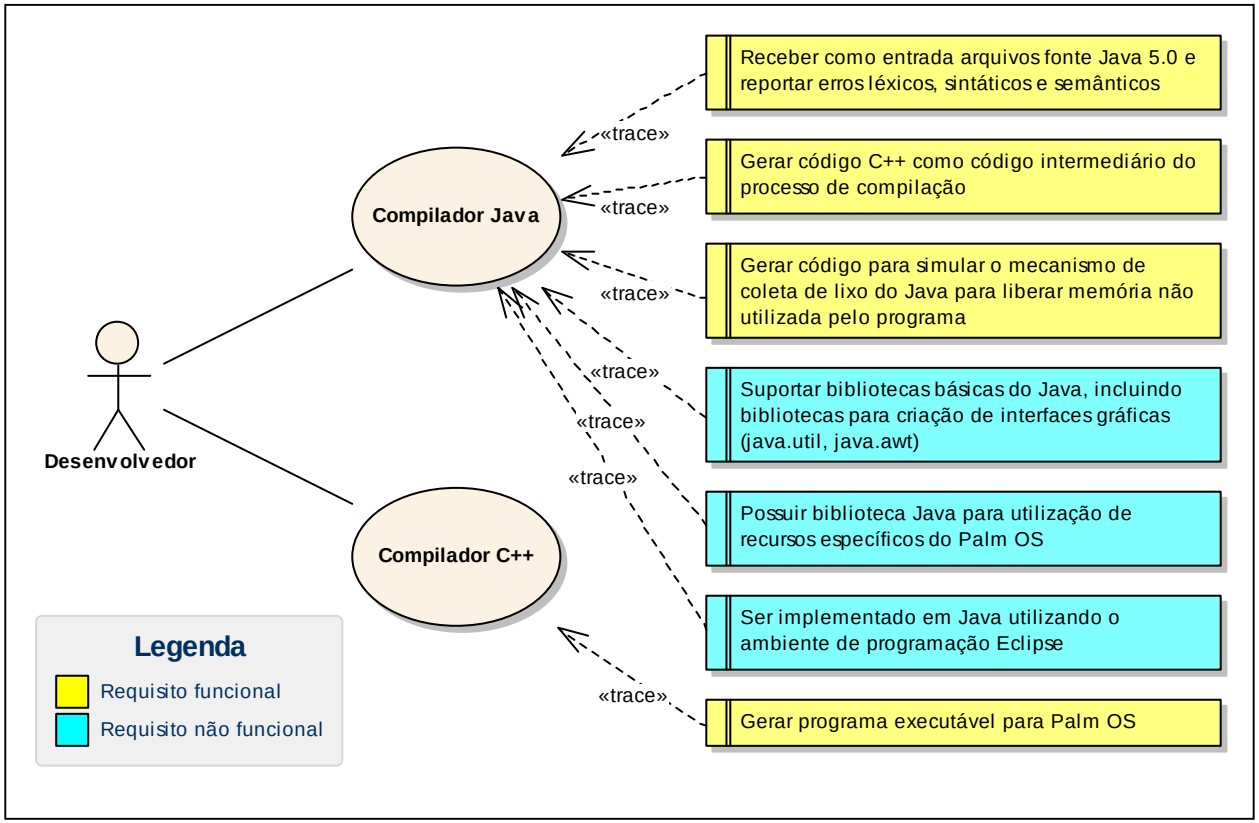
- Requisitos funcionais
  - receber como entrada arquivos fonte Java 5.0 e reportar erros léxicos, sintáticos e semânticos
  - gerar código C++ como código intermediário do processo de compilação
  - gerar código para simular o mecanismo de coleta de lixo do Java para liberar memória não utilizada pelo programa

Desenvolvimento
Levantamento de requisitos

- Requisitos não-funcionais
  - suportar bibliotecas básicas do Java, incluindo bibliotecas para criação de interfaces gráficas (java.util, java.awt)
  - possuir biblioteca Java para utilização de recursos específicos do Palm OS
  - ser implementado em Java utilizando o ambiente de programação Eclipse

Desenvolvimento
Especificação

- Casos de uso



Desenvolvimento
Especificação

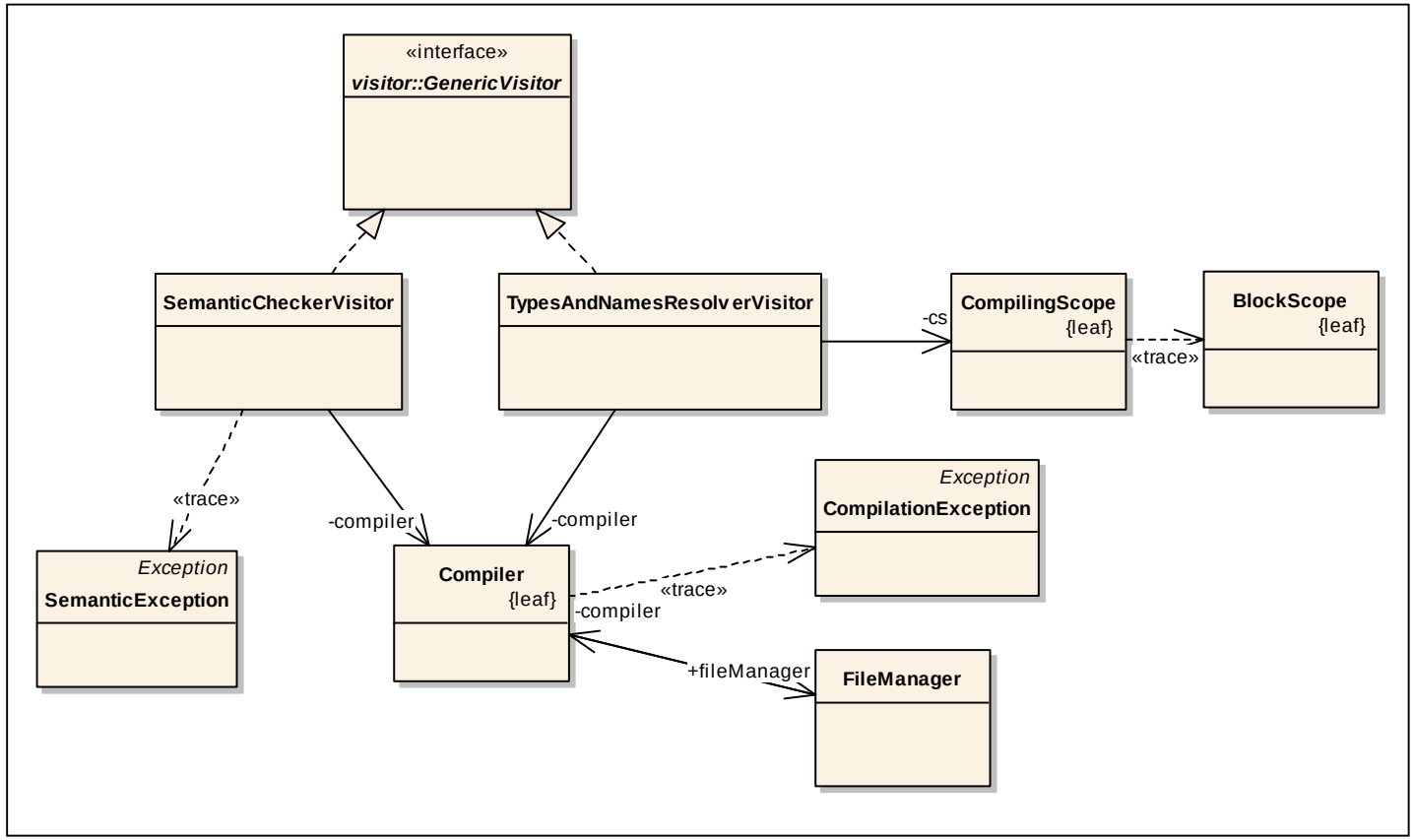
- Analisadores léxico e sintático
  - Gerados com o JavaCC a partir de gramática obtida no repositório de gramáticas do JavaCC
  - Gramática utilizada continha *bugs*
  - Construção da árvore sintática do programa (AST)
  - Árvore sintática especificada manualmente, composta por 85 classes

Desenvolvimento
Especificação

- Analisador semântico
  - Efetua duas passadas na AST
    - Resolução de nomes e tipos
    - Verificações semânticas
  - Anota a AST com informações contextuais utilizadas na geração de código
  - Utiliza o *pattern Visitor* para percorrer a AST

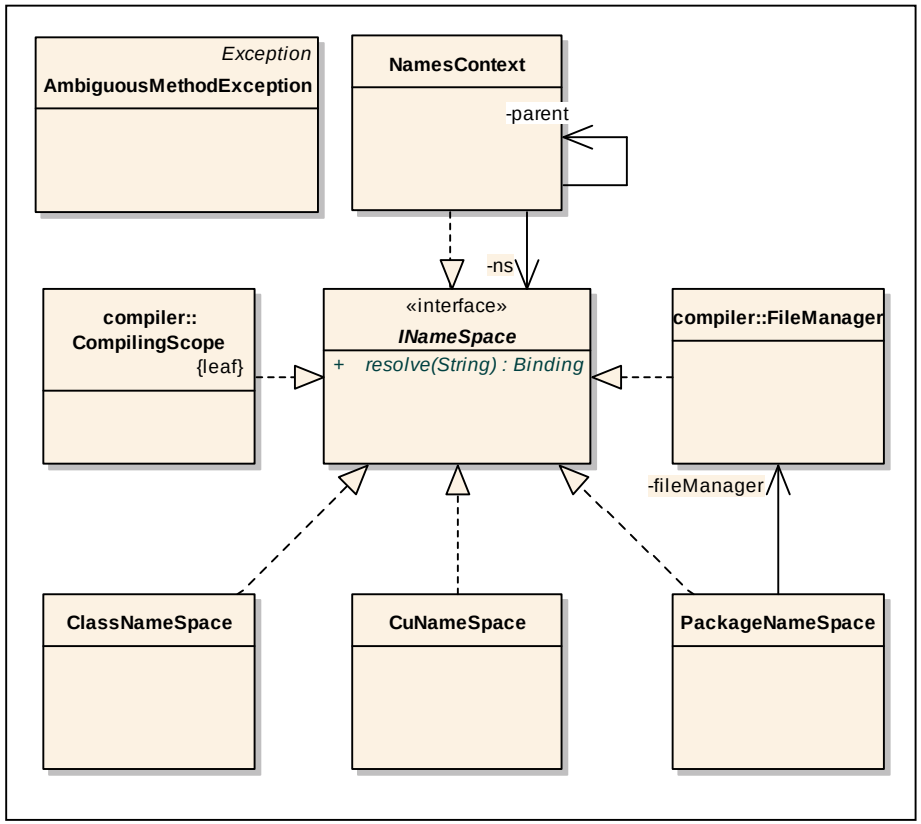
Desenvolvimento
Especificação

- Diagrama de classes – análise semântica



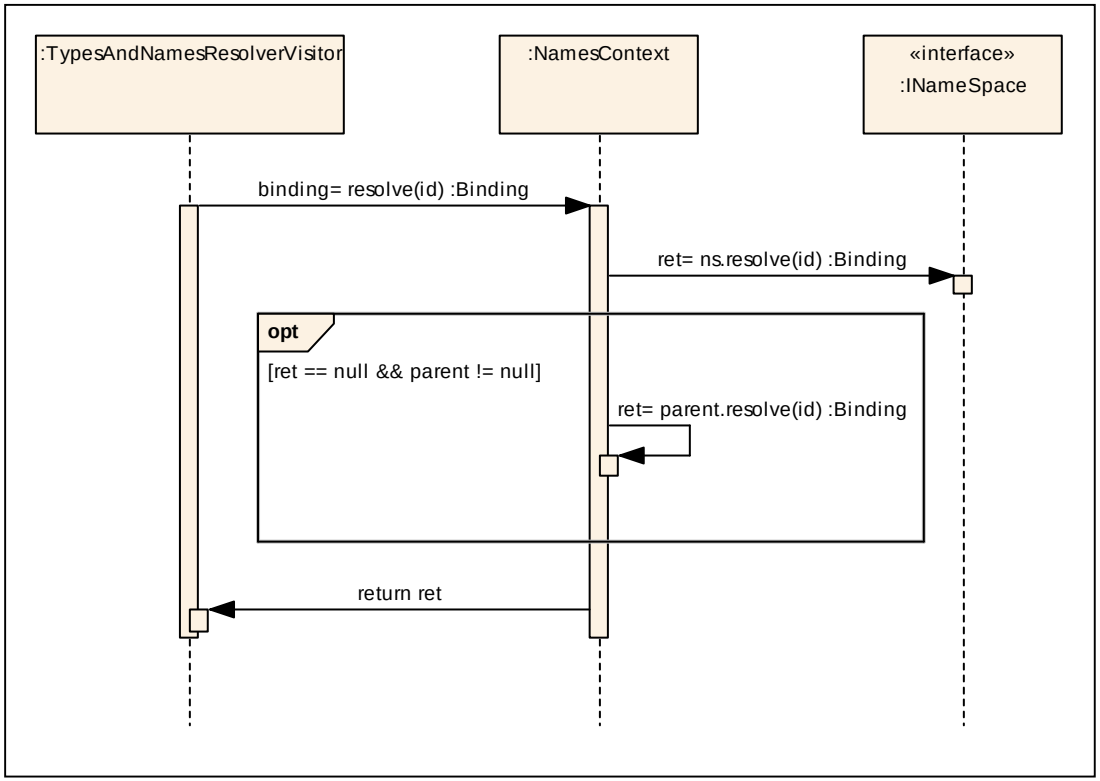
Desenvolvimento
Especificação

- Diagrama de classes – resolução de nomes



Desenvolvimento
Especificação

- Diagrama de seqüência – resolução de nomes

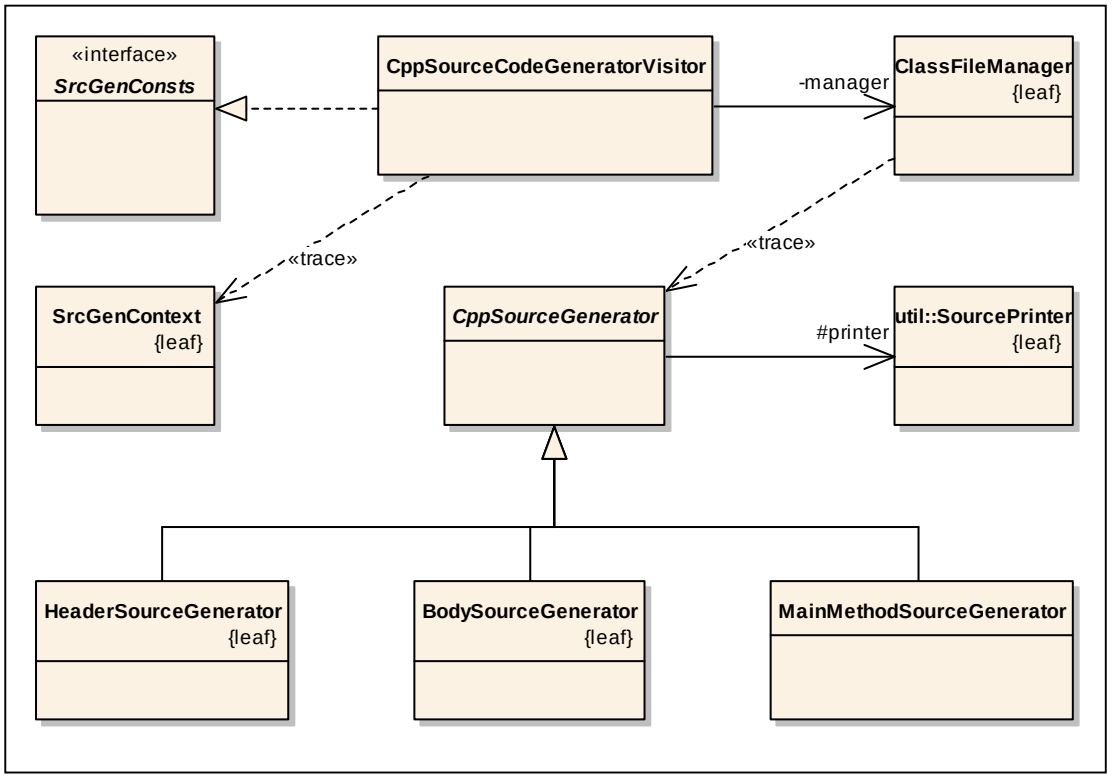


Desenvolvimento
Especificação

- Geração de código
  - Utiliza o *pattern Visitor* para percorrer a AST
  - Para cada classe Java, inclusive classes internas, é gerado uma classe C++ dividida em dois arquivos: `.h` e `.cpp`
  - Visibilidade dos métodos e atributos é respeitada
  - Visibilidade `protected` para classes que estão no mesmo pacote é simulada utilizando o recurso `friend class` do C++

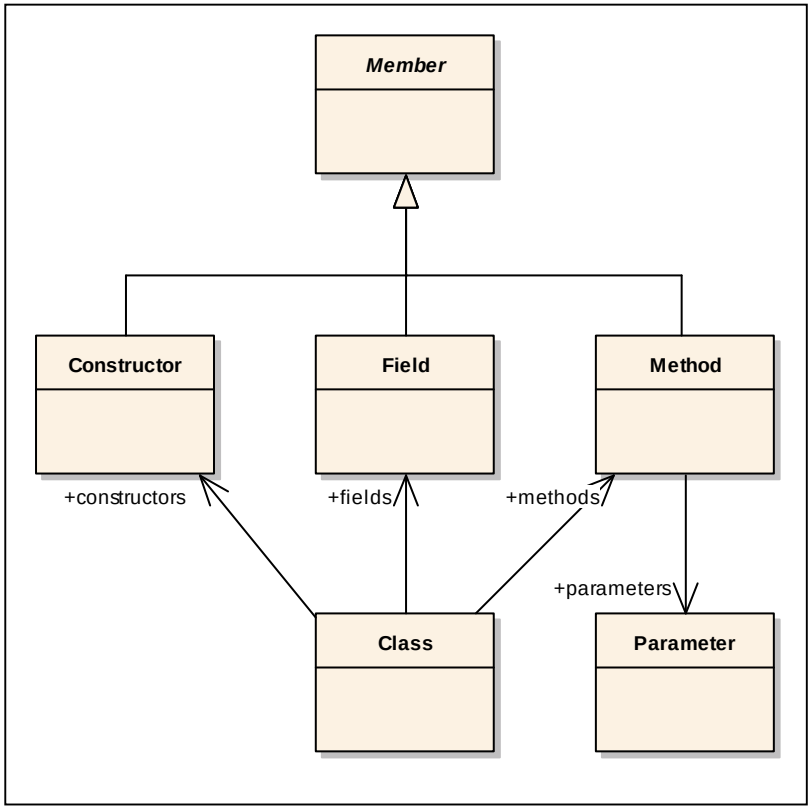
Desenvolvimento
Especificação

- Diagrama de classes – geração de código

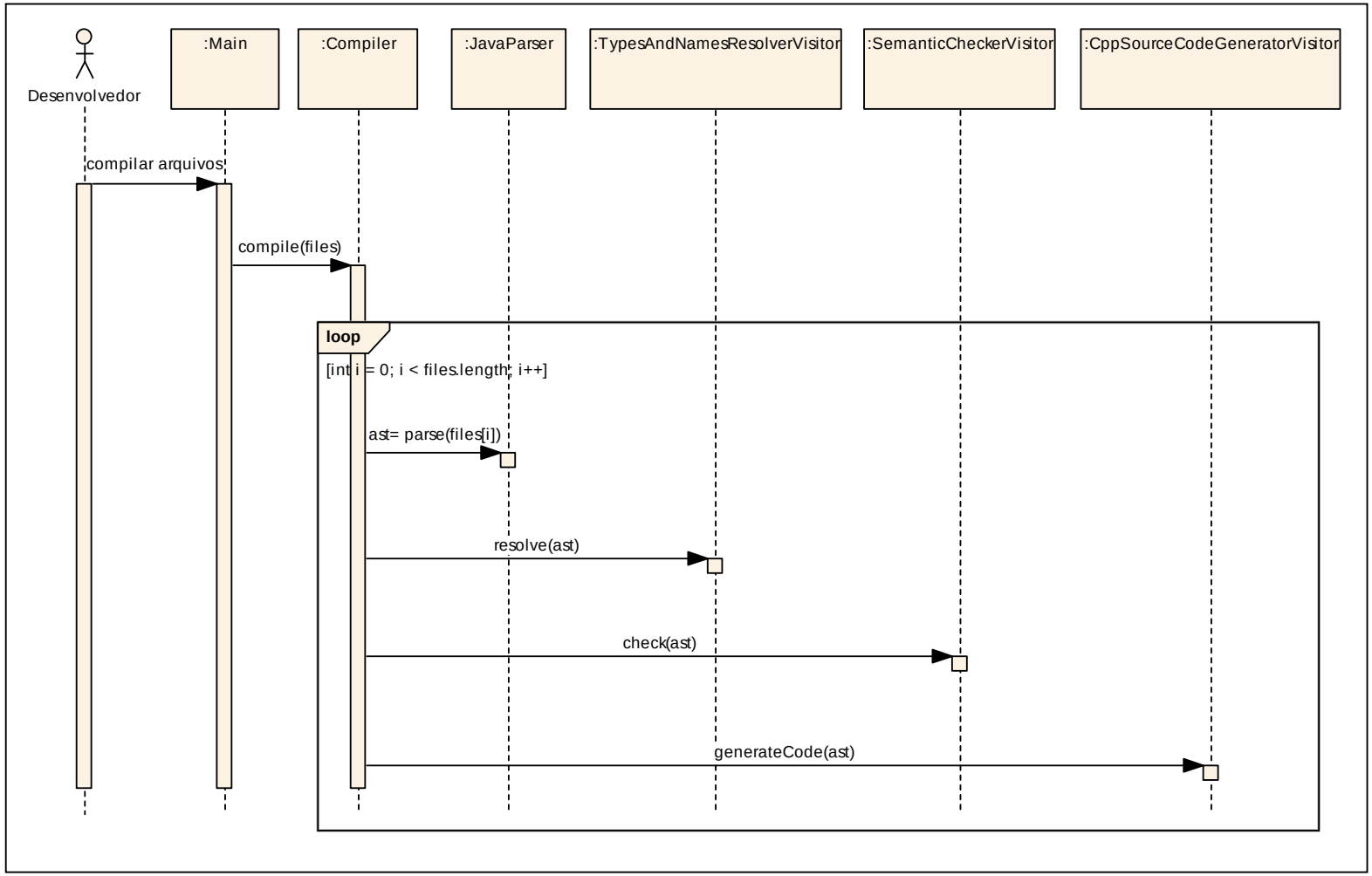


Desenvolvimento
Especificação

- Diagrama de classes – geração de código



Desenvolvimento
Especificação



Desenvolvimento
Implementação

- Implementação orientada ao estudo de caso
  - Análise semântica não foi totalmente implementada
  - Geração de código não foi totalmente implementada
  - Apenas bibliotecas Java necessárias para o estudo de caso foram implementadas em C++

Desenvolvimento
Implementação

- java.lang.Object
- **java.lang.Array**
- **java.lang.String**
- java.lang.StringBuffer
- java.lang.System
- java.lang.Math
- java.lang.Exception
- java.util.Random
- java.awt.Component
- java.awt.Container
- java.awt.Window
- **java.awt.Frame**
- **java.awt.Dialog**
- java.awt.Panel
- **java.awt.Button**
- **java.awt.Label**
- **java.awt.List**
- **java.awt.Choice**
- java.awt.MenuItem
- java.awt.MenuBar
- java.awt.Menu
- java.awt.MenuShortcut

- java.awt.LayoutManager
- java.awt.BorderLayout
- **java.awt.GridLayout**
- java.awt.FlowLayout
- **java.awt.Graphics**
- java.awt.Font
- java.awt.Color
- java.awt.Insets
- java.awt.FontMetrics
- java.awt.Dimension
- java.awt.event.ComponentListener
- java.awt.event.WindowListener
- java.awt.event.MouseListener
- **java.awt.event.ActionListener**
- java.awt.event.ComponentAdapter
- java.awt.event.WindowAdapter
- java.awt.event.MouseAdapter
- java.awt.event.MouseEvent
- java.awt.event.KeyEvent
- java.awt.event.WindowEvent
- **java.awt.event.ActionEvent**
- java.awt.event.ComponentEvent

Desenvolvimento
Implementação

- Coletor de lixo
  - *Smart pointer*
  - *Reference counting*
  - Ciclos

```
...
class java_awt_Color : public java_lang_Object {
   ...
};
typedef gc::ptr<java_awt_Color> java_awt_ColorPtr;
...
```

Desenvolvimento
Implementação

```java
//HelloWorld.java

public class HelloWorld {
    public static void main(String[] args) {
        HelloWorld helloWorld = new HelloWorld("Hello World!");
        System.out.println(helloWorld.getStr());
    }

    private final String str;

    public HelloWorld(String str) {
        this.str = str;
    }

    public String getStr() {
        return str;
    }
}
```

```java
//HelloWorld.java

public class HelloWorld {
    public static void main(String[] args) {
        HelloWorld helloWorld = new HelloWorld("Hello
    World!");
        System.out.println(helloWorld.getStr());
    }

    private final String str;

    public HelloWorld(String str) {
        this.str = str;
    }

    public String getStr() {
        return str;
    }
}
```

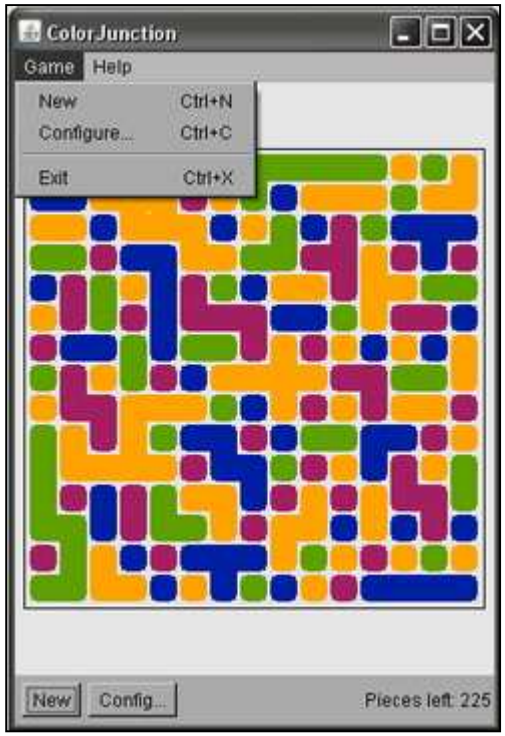
```java
//HelloWorld.h

#ifndef HELLOWORLD_H_
#define HELLOWORLD_H_
#include "java.lang.Array.h"
#include "java.lang.String.h"
#include "java.lang.System.h"
#include "japa.lang.GC.h"
class HelloWorld {

public:
    HelloWorld(java_lang_StringPtr str);
    static void main(java_lang_ArrayPtr<
java_lang_StringPtr > args);
    virtual java_lang_StringPtr getStr();

private:
    const java_lang_StringPtr str;
};

typedef gc::ptr<HelloWorld> HelloWorldPtr;

#endif //HELLOWORLD_H_
```

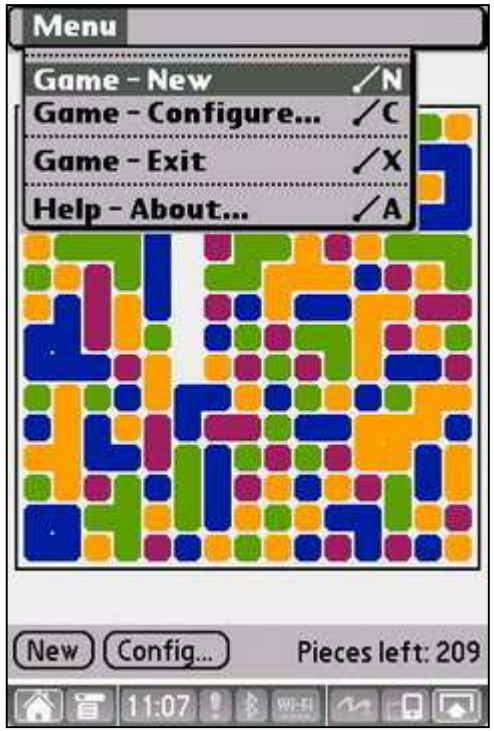
```java
//HelloWorld.java

public class HelloWorld {
    public static void main(String[] args) {
        HelloWorld helloWorld = new HelloWorld("Hello
    World!");
        System.out.println(helloWorld.getStr());
    }

    private final String str;

    public HelloWorld(String str) {
        this.str = str;
    }

    public String getStr() {
        return str;
    }
}
```

```cpp
//HelloWorld.cpp

#include "HelloWorld.h“

HelloWorld::HelloWorld(java_lang_StringPtr str):
    str(str) {
}

void HelloWorld::main(java_lang_ArrayPtr<
java_lang_StringPtr > args) {
    HelloWorldPtr helloWorld =
      HelloWorldPtr(new
HelloWorld(java_lang_StringPtr("Hello World!")));
    java_lang_System::out->println(helloWorld-
>getStr());
}

java_lang_StringPtr HelloWorld::getStr() {
    return str;
}
```

Desenvolvimento
Estudo de caso

- Jogo ColorJunction





Resultados e discussão

- Desenvolver aplicativos em Java é mais produtivo do que desenvolver em C++ nativo para Palm OS
- O coletor de lixo do Java facilita o desenvolvimento de programas e evita erros
- As bibliotecas Java implementadas em C++ conseguiram reproduzir o comportamento dos originais em Java

Conclusão

- Principais objetivos foram alcançados
- Técnicas utilizadas para simular o coletor de lixo se mostraram eficientes
- Gerador de código C++ obteve êxito em gerar código funcionalmente equivalente ao código Java
- Bibliotecas Java implementadas em C++ permitiram a execução de interfaces gráficas do AWT sem perder características em relação ao Java

Conclusão

- Compilar programas Java para executar nativamente em Palm OS é totalmente viável, facilitando o desenvolvimento de aplicativos para Palm OS

Extensões

- Concluir o analisador semântico e o gerador de código, dando suporte a todas as construções do Java 5.0
- Transcrever os comentários das classes Java para as classes C++ geradas, permitindo uma melhor leitura dos fontes gerados
- Implementar em C++ o restante das classes da biblioteca AWT, possibilitando a compilação de programas mais complexos

Extensões

- Codificar outras classes da API do Java em C++, tais como: Collections, Sockets, Remote Method Invocation (RMI), JDBC, Swing, entre outras
- Melhorar o GC desenvolvido para que detecte e colete ciclos
- Escrever as classes da API do Java para outras plataformas, permitindo que o compilador desenvolvido gere executável nativo para outras plataformas

Demonstração da implementação