Giovani Pieri

Projeto e implementação de uma linguagem de programação

Florianópolis – SC

2007/2

Giovani Pieri

Projeto e implementação de uma linguagem de programação

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador:
Luiz Fernando Bier Melgarejo

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis – SC

2007/2

Projeto e implementação de uma linguagem de programação

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Prof. Luiz Fernando Bier Melgarejo
Orientador

Banca Examinadora

Prof. Dr. Rosvelter Coelho da Costa

Prof. Dr. José Mazzucco Jr.

Resumo

O estudo e desenvolvimento de linguagens de programação é um ramo de estudo em Ciência da Computação. Linguagens de programação formam uma camada de abstração sobre a linguagem de máquina, o conjunto de instruções que a máquina oferece. Houveram diversas gerações de linguagens de programação, em cada uma mais abstrações e conceitos foram intro duzidas de forma a tornar a tarefa de programar menos propensa a erros e mais escalável, isto é, mais simples lidar com problemas grandes e complexos.

Atualmente, a indústria de hardware está abandonando a abordagem que vinha sendo utilizada de exploração de Instruction Level Parallelism e alta freqüência, e vem adotando a arquitetura multi-núcleo com freqüências mais baixas. Com isso, o software terá que ser paralelo para utilizar eficientemente o hardware.

Segundo Armstrong (2003) grande parte das linguagens de programação atuais são linguagens seqüenciais. O que torna a descrição de algoritmos e programas paralelos, difícil e propenso a erros nestas linguagens. Assim será desenvolvida uma linguagem de programação que visa estimular o uso de concorrência de forma natural, simples e de forma mais segura possível.

Palavras-chave: linguagens de programação, tipagem dinâmica, máquina de pilha

Abstract

Programming languages' study and development is a branch of Computer Science. Programming languages are an abstraction layer developed on top of machine language, the instruction set offered by the machine. There were several programming languages generations, in each one more abstraction and concepts were introduced in order to make programming tasks less error-prone and more scalable, that is, capable of dealing with larger and more complex problems.

Nowadays, the hardware industry is abandoning the approach that has been used until now, explore instruction level paralelism and high clock frequencies to extreme. Instead, it?s adopting a multicore architecture with lower clock frequency. So, software must be parallel to efficiently use all hardware resources and capacity.

According to erlang:concurrencyOriented majority of nowadays programming languages are sequential languages. This way, writing parallel algorithms and programs in those languages are difficult and error-prone. So will be developped a programming language whose main point is to stimulate use of concurrency in a natural, simple and more safety way possible.

Keywords: programming languages, dynamic typing, stack machine, parallelism

Sumário

Lista de Figuras

Lista de Listagens

1 Introdução

Linguagens de programação são linguagens artificiais criadas com o intuito de descrever algoritmos e programas passíveis de execução em máquinas reais (diretamente ou com algum pré-processamento). Elas formam uma camada de abstração sobre a chamada linguagem de máquina, que nada mais é que o conjunto de instruções que a máquina que se está programando oferece.

A existência de linguagens de programação data desde os primórdios da informática quando os primeiros programadores desenvolverem linguagens, mapeando as palavras binárias dos computadores em mnemônicos. Estas linguagens receberam o nome de *assembly*. Com o passar do tempo as linguagens evoluíram, novos paradigmas foram criados.

Atualmente existem diversas linguagens de programação, cada uma com suas particularidades, características, vantagens e desvantagens. Entretanto, estas linguagens possuem uma sintaxe e semântica relativamente complexa. Além disso alguns conceitos tal como paralelismo, são tratados marginalmente tornando difícil e muitas vezes não-intuitivo tratá-los.

Com isso em mente, existe uma brecha na qual poucas linguagens se encaixam surgindo a necessidade de uma linguagem simples que possa rapidamente levar programadores iniciantes a produzirem. Entretanto não deverá dificultar o desenvolvimento de programas por programadores experientes. Com esta proposta, uma linguagem de programação será criada.

Esta linguagem deverá suportar um modelo de paralelismo de alto nível nativamente, deverá ter sintaxe e semântica simples mas ser poderosa de modo a não limitar desenvolvedores experientes. As demais características e construções da linguagem serão estudadas e debatidas sempre levando em conta esta filosofia.

1.1 Objetivo Geral

Desenvolver uma linguagem de programação orientada a objetos, dinamicamente tipada. Ela deve ser sintática e semanticamente simples, suportar concorrência de forma natural e in-

tuitiva.

1.2 Objetivo Específico

- Desenvolver uma linguagem de programação orientada a objetos.
- Desenvolver uma linguagem interpretada intermediária, que rodará o programa compilado.
- Criar a sintaxe das duas linguagens
- Dar semântica às duas linguagens
- Estudar características e construções para adicionar às linguagens.

2 *Definição da Linguagem Alvo*

A linguagem alvo será uma linguagem de programação interpretada baseada na linguagem Telis. Linguagens interpretadas são aquelas que rodam sobre um interpretador, que é um programa cuja entrada é o código fonte de um programa escrito na linguagem interpretada e após realizar as análises léxica, sintática e semântica o executa.

Telis é uma linguagem madura, desenvolvida no laboratório Edugraf, que vem sendo desenvolvida desde 1998. Ela é fruto do estudo e desenvolvimento de várias outras linguagens anteriormente desenvolvidas pelo mesmo laboratório. O objetivo desta linguagem é ser sintática e semanticamente simples de tal forma que mesmo programadores iniciantes consigam expressar-se de forma natural, mas ao mesmo tempo expor conceitos avançados como paralelismo, comunicação via Internet (sistemas distribuídos) e princípios de orientação a objetos.

Telis vem sendo utilizado diariamente durante anos por alunos da Universidade Federal de Santa Catarina nos cursos de Ciência da Computação e Engenharia de Automação e Sistemas. Por este motivo, as idéia e conceitos desta linguagem serão utilizadas como base para a linguagem alvo.

2.1 Sintaxe de Telis

Telis possui uma sintaxe bastante simples, descrita pela gramática EBNF (Extended Backus Normal Form) mostrada na listagem 2.1.

```
programa ::= texto listaDeAtores listaDeModelosOuMoldes incluirAplique
giria ::= ( palavra | lista )*
lista ::= "[" ( palavra | lista )* "]"
listaDeAtores ::= "[" ( identificador )* "]"
listaDeModelosOuMoldes ::= "[" ( chamadaParaInclusaoDeModeloOuMolde )* "]"
chamadaParaInclusaoDeModeloOuMolde ::= lista texto lista
                                       primitivaDeInclusaoDeModeloOuMolde
primitivaDeInclusaoDeModeloOuMolde ::= <ID>
palavra ::= atribuicao | ( identificador | identificadorValorAtual |
```

```
            numero | operador | alcance | texto | operadorPonto | booleano
                )
atribuicao ::= ( <ID> ( <ASSOCIAR> ) )
identificador ::= <ID>
incluirAplique ::= <INCLUIR_APLIQUE>
identificadorValorAtual ::= ( <ID_VALORATUAL> )
numero ::= <NUMERO>
operadorPonto ::= "." <ID>
operador ::= ( "+" | "-" | "*" | "/" | "^" | "\\" | "<" | ">" | "<=" |
             ">=" | "<>" | "=" | <ASSOCIAR> )
texto ::= <TEXTO>
booleano ::= <BOOLEANO>
alcance ::= <ALCANCE>
```

Listagem 2.1: Gramática BNF de Telis

Os valores entre parênteses angulares são elementos provenientes da análise léxica e definidos através de expressões regulares. A descrição destes elementos serão omitidas aqui, por seus nomes serem em sua maioria auto-explicativos. Elementos que possam gerar dúvidas serão explanados adiante.

Como é possível verificar um programa Telis é composto basicamente por textos, números, símbolos e listas. Todo programa em Telis possui um texto com o nome do aplique, uma lista contendo o nome dos modelos que serão instanciados, uma segunda lista contendo em seu interior as definições de modelos e moldes, e por último o símbolo *incluirAplique* responsável pela criação de Apliques. Toda esta definição de um programa Telis é escondida do programador pelo Ambiente Telis, um ambiente de desenvolvimento desenvolvido também pelo laboratório Edugraf para facilitar (ainda mais) a programação em Telis.

Também é possível ver que praticamente qualquer código (lista) é sintaticamente válido, muitas poucas restrições são impostas pela gramática Telis.

2.2 Semântica de Telis

2.2.1 Conceitos básicos

Telis é semanticamente simples também, possuindo forte influência de Forth. Segundo Brodie (1986), em Forth, existe um local central no qual números são temporariamente armazenados antes de serem operados. Este local é uma pilha. Números são empilhados e então operados. De forma análoga, as operações em Telis são definidas sobre uma pilha, sendo que a

passagem de parâmetros e resultados são todas realizadas através da mesma.

```
10 20 +
```

Listagem 2.2: Exemplo de soma em Telis

O código mostrado na listagem 2.2 mostra um exemplo de avaliação de expressões em Telis, no caso, uma soma. O interpretador Telis primeiramente executa o número 10, coloca-o na pilha. O mesmo ocorre com o número 20. Após a execução destes comandos a pilha está como mostrado na figura 2.1.

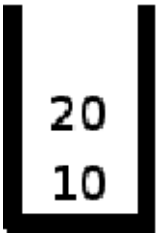

Figura 2.1: Pilha após os comandos `10 20`

Quando o interpretador Telis encontra o operador + ele o executa. Para isso, dois parâmetros da pilha são consumidos, somados e o resultado é empilhado.

Símbolos em Telis são uma cadeia de caracteres separados por espaços. De forma análoga aos operadores, sempre que a máquina Telis encontra um símbolo ela tenta executá-lo, caso não consiga um erro de tempo de execução é gerado.

A máquina Telis possui um conjunto de símbolos, inerente à máquina, sendo cada um dos elementos denominado *agenda primitiva* e cada uma destas agendas primitivas possui associada a si um determinado comportamento. Toda vez que um símbolo associado à uma determinada agenda é encontrado o comportamento relacionado a esta agenda é executado. Este comportamento pode ser afetado por parâmetros, que são obtidos da pilha e possíveis resultados serão empilhados. As primitivas são utilizadas para modelar as estruturas de controle de fluxo e repetição, operações sobre textos, listas e números entre outras funcionalidades.

Listas desempenham um papel muito importante em Telis. Além de poderem ser utilizadas como estruturas de dados para o armazenamento e processamento de informações, elas podem agrupar um conjunto de instruções. Telis se aproveita deste fato para modelar, por exemplo, expressões condicionais. Expressões condicionais recebem como parâmetro uma lista contendo os comandos que deverão ser realizados dependendo do valor de um segundo parâmetro booleano, como mostrado na listagem 2.3.

```
verdadeiro
```

```
[
    10 mostrar
]
seVerdade
```

Listagem 2.3: Exemplo de estrutura de controle de fluxo em Telis

No código da listagem 2.3, um valor booleano e uma lista contendo comandos são empilhados. A primitiva *seVerdade* recebe estes dois parâmetros e caso o valor booleano seja verdadeiro, o que realmente ocorre neste caso, a lista de comandos é executada. Neste caso, o número 10 é mostrado na tela.

A atribuição de variáveis é um caso particular e viola a política do interpretador de sempre executar um símbolo assim que este é encontrado. No jargão de Telis, a atribuição de um determinado valor à uma variável é chamado de associação. Variáveis são associadas através da primitiva *associar*, como mostrado na listagem 2.4. A diferença em relação a outras primitivas é que a primitiva associar pode receber um símbolo como parâmetro. Na gramática BNF de Telis, mostrada na listagem 2.1 na página 13, pode-se notar que na produção *palavra* há um tratamento especial em relação a primitiva *associar*, para o caso no qual um símbolo seja seguido de um associar.

```
"um texto qualquer" umaVariavel associar
```

Listagem 2.4: Exemplo de criação de variáveis em Telis

Durante a execução do código mostrado na listagem 2.4, quando a máquina Telis se depara com o símbolo *umaVariável*, é olhado um *token* a frente e caso este token seja um *associar*, o que realmente ocorre, o valor que está na pilha é associado ao símbolo atual, no caso umaVariável. Para empilhar o valor anteriormente atribuído à uma variável, basta executar o símbolo ao qual a variável foi associada.

2.2.2 Concorrência

Telis é uma linguagem inerentemente concorrente. A concorrência, a exemplo de linguagens como Erlang (ARMSTRONG, 1997), é tratada diretamente por mecanismos providos pela linguagem, diferentemente de outras linguagens nas quais são oferecidos mecanismos para a criação de *threads* e a sincronização é realizada através de semáforos e/ou monitores utilizando-se memória compartilhada.

Telis adota uma abordagem baseada em Atores. Atores são parecidos com processos no

que diz respeito a compartilhamento de recursos. Não há memória compartilhada, conseqüentemente a necessidade de sincronização de áreas de memória compartilhadas é eliminada.

A comunicação entre atores se dá através de troca de mensagens. O modelo de troca de mensagens entre atores originalmente proposto diz que para um ator se comunicar com outro ator ele deve poder identificá-lo. Após identificar com quem se deseja comunicar o envio da mensagem pode ser realizado.

Telis adotou uma abordagem distinta. Um ator não necessita nenhum tipo de informação a respeito de quem receberá a mensagem. A mensagem é enviada a todos os outros atores que tenham se cadastrado para recebê-la.

Uma tupla é uma seqüência finita de objetos, cada um de um determinado tipo. Uma tupla em Telis é representada por uma lista, e é a base para o sistema de troca de mensagens do Telis. Mensagens em Telis são chamadas de estímulos ou eventos. Ao enviar uma mensagem para um ou mais atores diz-se que um estímulo foi emitido. Ao recepcionar este estímulo diz-se que o estímulo foi tratado.

São necessários dois mecanismos básicos para que atores possam se comunicar: eles devem poder emitir estímulos e poder se cadastrar com o intuito de receber um determinado estímulo. Logo, duas primitivas são utilizadas para realizarem a comunicação entre dois atores distintos: *dizer* e *seDito*.

A agenda primitiva *dizer* é responsável pela emissão de estímulos. Seu funcionamento é bastante simples: uma tupla (lista) é buscada na pilha e enviada a todos os atores com exceção de si mesmo.

A agenda primitiva *seDito* é responsável pelo cadastro de um tratador de estímulos, sendo este composto por duas partes: um filtro e uma lista de instruções. Como resultado desta operação o tratador de estímulos é cadastrado e sempre que um estímulo compatível com o filtro for recebido o ator é interrompido e o respectivo filtro passa a executar. Após o término do processamento do estímulo, o ator retoma a sua execução normal sendo que qualquer alteração feita na pilha pelo tratador é desfeita, isto é, a pilha irá possuir os mesmos dados de quando fora interrompida.

Filtros são tuplas. Diz-se que um filtro casa com um estímulo se as condições abaixo forem satisfeitas:

1. Filtro e estímulo possuem mesmo tamanho
2. Para todo elemento do estímulo, o elemento de mesma posição no filtro deve: ser igual

ou indicar o tipo do elemento do estímulo através dos símbolos **texto**, **número** ou **lista**.

O funcionamento do mecanismo de recepção de mensagens é muito similar ao funcionamento de interrupções em processadores. Ao se ativar uma linha de interrupção o processador automaticamente desvia para o endereço onde o tratador de interrupção se encontra e logo após o término do código de tratamento de interrupções, todos os registradores são restaurados e o processador devia para a instrução que seria executada caso não houvesse uma interrupção, como se nada houvesse ocorrido.

De maneira similar o ator Telis executa os comandos até que um tratador cujo filtro case com um estímulo que tenha sido emitido. Neste momento, o fluxo normal de execução é interrompido, é empilhado o estímulo que está sendo tratado e as instruções anexadas ao tratador pela execução da primitiva seDito é executado. Após o término da execução do tratador, a pilha é restaurada ao estado inicial, isto é, os elementos contidos na pilha serão iguais ao que estava na mesma antes do estímulo ser iniciado e o fluxo de execução normal é retomado.

Uma característica importante do sistema de notificação do Telis é que durante o tratamento de um estímulo dizer, novos estímulo são perdidos. Isto é, o ator não é capaz de tratar mais de um estímulo emitido pelo dizer simultaneamente.

2.2.3 Atores, modelos e agendas

Atores são entidades autônomas, isto é, vários atores podem rodar simultaneamente, de forma independente.

Quando um ator é criado ele executa a agenda de nome *iniciar*. Agendas são análogos à métodos em linguagens orientadas a objetos. Diferentemente de agendas primitivas, que são providas diretamente pela linguagem, agendas contém código definido pelo programador. Agendas são executadas quando o símbolo associado ao nome da agenda for executado pelo interpretador Telis.

Para se criar um ator é necessário uma espécie de fábrica de atores contendo a definição dos mesmos, isto é, a definição das agendas que ele possuirá. Esta fábrica de atores é chamada de modelo, e funciona de maneira similar a classes em linguagens orientada a objetos. Uma vez definido um modelo, e conseqüentemente suas agendas, um ator pode ser instanciado. Para se instanciar um ator a partir de um modelo basta executar o símbolo correspondente ao nome do modelo que fora definido. Neste momento, o interpretador Telis toma as seguintes ações:

1. Cria um novo ator. Este ator possui acesso a todas as agendas definidas pelo modelo.

2. Copia o topo da pilha do ator que está criando o novo ator para a pilha do ator recém criado.

3. A agenda iniciar do ator original é executada.

4. Uma referência ao ator criado é empilhada no ator que criou o novo ator.

A referência a um ator pode ser utilizada posteriormente para realizar comunicação direta com o ator, isto é, enviar uma mensagem direcionada pedindo que o mesmo execute uma agenda. Todo ator possui uma identidade, e há garantia que esta identidade é única. É possível acessar a própria identidade através da primitiva *obterIdentidade*, e obter uma referência para si próprio através da primitiva *euMesmo*.

2.2.4 Herança, moldes e princípios de orientação a objetos

Telis permite uma noção de herança com a utilização de moldes. A herança de Telis é diferente da herança em linguagens como Java e C++. O intuito é fornecer um mecanismo simples de ser entendido e explicado, embora seja menos flexível. Isto ocorre porque o ponto onde a superclasse será chamada é pré-definida ao invés de ser indicada pelo programador.

Classes abstratas em Telis são denominadas moldes. Diz-se que um molde molda um modelo, e este por sua vez é moldado pelo molde. Um modelo pode ser moldado por zero ou mais moldes, como é possível que um modelo seja moldado por mais de um molde há em Telis a noção de herança múltipla. Além disso, um molde também pode ser moldado por zero ou mais moldes. Um modelo não pode ser utilizado como molde.

A partir dessa definição de moldado/molde é possível criar um grafo de herança G(V, A) definido da seguinte forma:

$V = \{ x \mid x \text{ é um molde ou modelo} \}$

$A = \{ (x, y) \mid x \text{ molda } y \wedge x \text{ é um molde} \}$

A figura 2.2 mostra um grafo de herança montado supondo a existência de um modelo denominado ModeloUm moldado por dois moldes, chamados de MoldeUm e MoldeDois.

Esse grafo deve possuir as seguintes propriedades:

1. Não deve possuir ciclos.

2. Todos os modelos devem ser nodos folha, isto é, seu grau de emissão[1] é igual a 0.

[1] O grau de emissão de um vértice v corresponde ao número de arestas que partem de v.

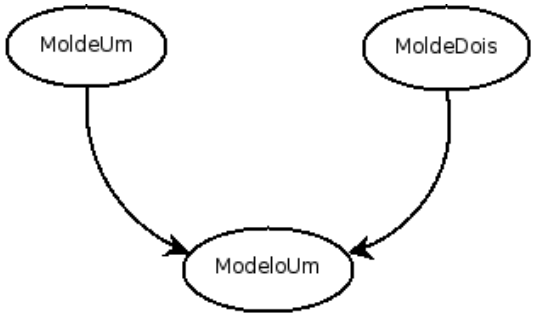

Figura 2.2: Grafo de herança com um modelo e dois moldes.

Os moldes definem agendas, de forma análoga a modelos. Estas agendas podem ser utilizadas por quem for moldado por este molde. Caso um modelo ou molde defina uma agenda com mesmo nome de uma agenda definida em um dos moldes que o está moldando, este irá estender a agenda do molde. Do ponto de vista do programador, o código definido pelo modelo é anexado ao fim das agendas do molde cujos nomes sejam idênticos.

É importante notar que os moldes podem referenciar agendas inexistentes no molde. Isto é útil para realizar implementação de alguns padrões de projetos comuns tais como *Template Method*. Como pode ser visto no diagrama mostrado na figura 2.3, a descrição geral de um algoritmo é definido em uma superclasse em um método *template*, devendo este delegar tarefas para métodos abstratos. Cada subclasse deve preencher as lacunas no algoritmo com os detalhes pertinentes.

Em Telis, agendas abstratas não são explicitamente definidas mas podem então ser simuladas através deste mecanismo de delegação para agendas inexistentes.

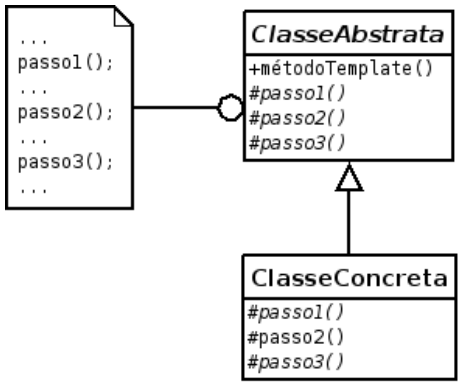

Figura 2.3: Diagrama de classes do padrão de projeto *Template Method*.

Por definição, durante a invocação de uma agenda, as agendas definidas nos moldes são executados anteriormente as agendas de quem está sendo moldado. Dado um grafo de herança G e um modelo M pertencente a este grafo, é definido o subgrafo X composto pelos vértices

pertencentes ao fecho transitivo inverso[2] de M e toda aresta definida entre dois vértices pertencentes a este fecho transitivo inverso. Isto é:

V = { x | x ∈ FTI(G, M)}, onde FTI(G, M) é o conjunto de vértices do fecho transitivo inverso do vértice M do grafo G.

A = { (x, y) | (x, y) ∈ E(G) }, onde E(G) é o conjunto de arestas de G.

A ordem de invocação das agendas é um ordenamento topológico do grafo X. Em teoria de grafos, o ordenamento topológico de um grafo direcionado acíclico é um ordenamento linear de seus vértices que é compatível com a ordem parcial R induzida nos vértices onde x vem antes de y (xRy) se existe um caminho de x para y no grafo. Por esta definição é possível perceber que há diversas formas de se montar um ordenamento topológico de X.

A ordem em que os moldes se encontram na lista de moldes é utilizado para determinar a ordem na qual as agendas dos moldes serão invocadas, isto é, o ordenamento topológico de X. As agendas dos moldes são invocados preferencialmente na ordem em que foram listados na lista de moldes do moldado. Caso haja necessidade, é possível que o a ordem da lista de moldes seja desrespeitada para que o ordenamento linear de X que será gerado continue sendo um ordenamento topológico, mas o algoritmo tentará manter a ordem listada.

Outro ponto importante é que durante a invocação da agenda de um modelo, todas as agendas de mesmo nome dos moldes contidos no fecho transitivo inverso do modelo são invocadas uma e somente uma vez.

2.2.5 Comunicação direta e polimorfismo

A comunicação entre atores em Telis se dá através de estímulos. A forma mais simples de realizar comunicação entre atores diferentes é através de estímulos assíncronos utilizando-se primitivas *dizer* e *seDito*. Entretanto, esta forma de comunicação pode se tornar um tanto incomoda de ser utilizada. Devido a sua natureza intrínseca de *broadcast*, isto é, um estímulo emitido pode influenciar um número qualquer de atores. Este comportamento pode ser uma vantagem, como na implementação de arquiteturas *Model-View-Controler* (MVC) e do padrão *Observer* por exemplo. Entretanto pode se tornar complexo realizar comunicação entre dois atores previamente conhecidos, principalmente se for necessário uma troca de dados ocorrendo nos dois sentidos, isto é, os dois devem trocar dados de forma sincronizada. Isto ocorre porque será necessário criar um protocolo de comunicação entre as duas partes, além de construir um sistema que garanta que as mensagens afetarão apenas as partes envolvidas.

[2] O fecho transitivo inverso (fti) de um vértice v é o conjunto de todos os vértices a partir dos quais se pode atingir v por algum caminho (levando em conta a orientação das arestas).

De forma a simplificar este tipo de comunicação, existe em Telis um tipo diferenciado de estímulo chamado estímulo de comunicação direta. A comunicação direta encapsula toda a complexidade de fazer dois atores se comunicarem, sincronizando-os. Para realizar a comunicação direta, um ator de posse de uma referência ao ator alvo envia um pedido, e se necessário um parâmetro. O ator que realizou a comunicação direta é então bloqueado até que o ator alvo termine o processamento do pedido, que ocorre utilizando o mesmo princípio de interrupção utilizado pelo seDito. Quando o processamento do pedido é finalizado caso a pilha não esteja vazia o seu topo é retornado ao ator que iniciou a comunicação direta e o ator alvo retoma o que estava realizando, nos mesmo moldes que o tratamento convencional do seDito.

A sintaxe da comunicação direta foi inspirada em linguagens como Java e Ruby nos quais mensagens são enviadas a objetos utilizando-se a notação do ponto, como mostrado na listagem 2.5

```
public class Xpto {
        public void metodo() {
                ...
        }
}

public class Main {
        public static void main(String[] args) {
                Xpto xpto = new Xpto();
                xpto.metodo();
        }
}
```

Listagem 2.5: Envio de mensagens em Java

Na notação de ponto, coloca-se a referência ao objeto seguido do nome do método que se deseja invocar precedido por um ponto. Normalmente a referência é guardada em uma variável, mas isto não é obrigatório. Em Telis ocorre algo parecido, primeiro se empilha a referência do ator com quem se deseja comunicar, em seguida se escreve-se o nome da agenda que se deseja pedir que o ator alvo execute. Caso a agenda invocada necessite de um parâmetro ele é buscado na pilha. Apenas um parâmetro está disponível ao ator alvo, caso seja necessário um número maior de dados, é possível enviá-los colocando-os em uma lista.

No código da listagem 2.6 são mostrados dois modelos de atores. O primeiro, chamado ModeloAlvo possui uma agenda chamada somarDois que pega um número na pilha soma o número dois e retorna o resultado. Um segundo modelo, denominado MeuModelo possui um código mais interessante em sua agenda iniciar, primeiramente o modelo ModeloAlvo é instan-

ciado e sua identidade guardada na variável somador. Em seguida o número 5 e o conteúdo da variável somador (identidade do ator alvo) é empilhado e, utilizando-se a notação de ponto a agenda somarDois do ator alvo é invocada. Depois que o ator alvo termina o processamento e calcula o resultado da soma (7) o retorna. Neste ponto, o retorno (7) é empilhado na pilha do ator que iniciou a comunicação direta e este ator é liberado, executando assim a primitiva mostrar, que imprime o que está na pilha na tela, que é o próprio sete.

```
"umAplique" [MeuModelo]
[
        [] "ModeloAlvo"
        [
                "somarDois"
                [
                        2 +
                ]
                incluirAgenda
        ]
        incluirModelo

        [] "MeuModelo"
        [
                "iniciar"
                [
                        [] ModeloAlvo somador associar
                        5 somador.somarDois
                        mostrar
                ]
                incluirAgenda
        ]
        incluirModelo
]
incluirAplique
```

Listagem 2.6: Envio de mensagens síncronas em Telis

Permitir que um ator peça para que outro execute uma de suas agendas e retorne o resultado pode ser bastante útil, entretanto permitir que ele execute qualquer outra agenda não é uma boa pratica. Grande parte das linguagens possuem um sistema de restrição de visibilidade de métodos. Por exemplo Java define quatro níveis de visibilidade: *public*, *protected*, *package*, *private*. Em Telis também existe níveis de visibilidade das agendas, dois mais especificamente: público e privado. Agendas públicas podem ser alvo de uma comunicação direta, enquanto agendas privadas não. Caso tente-se fazer comunicação direta com uma agenda privada é gerado

um erro em tempo de execução. A diferenciação entre agendas públicas e privadas diz respeito ao nome. Foi convencionado que agendas cujo nome inicia com "_" são privadas, qualquer outra agenda é pública.

Existem algumas diferenças entre os estímulos utilizados através da primitiva dizer/seDito e a comunicação direta. A comunicação direta, diferentemente dos estímulos emitidos pelo dizer, não é perdida caso um novo pedido de comunicação direta chegue enquanto um outro estímulo está sendo tratado, ao invés disso, o pedido é colocado em uma fila para ser tratado assim que possível. Outra diferença é que atores podem tratar estímulos de dizer assim que um tratador seja instalado utilizando a primitiva seDito, mas estímulos de comunicação direta são enfileirados e serão tratados apenas quando o ator finalizar a execução da *agenda iniciar*. Esta última restrição foi imposta a fim de prover um mecanismo com o qual o programador possa inicializar as suas variáveis internas, deixando o ator em um estado consistente, antes de fornecer serviços a outros atores. Entretanto, como resultado disto, existe a possibilidade de ocorrer *deadlocks* caso a agenda iniciar não retorne nunca.

Em Telis há uma hierarquia de prioridades entre os dois tipos de estímulos. Os estímulos oriundos de comunicação direta são os menos prioritários, enquanto os estímulos oriundos da primitiva dizer são os mais prioritários. Quando um ator está tratando um estímulo dizer nenhum outro estímulo pode ser tratado. Caso um outro estímulo chegue ele é descartado, caso chegue um estímulo de comunicação direta este é enfileirado. Estímulos de comunicação direta são menos prioritários. Enquanto um ator está tratando um estímulo de comunicação direta ele pode ser interrompida para o tratamento de um estímulo dizer, entretanto se outro pedido de comunicação direta chegar, ele é enfileirado para tratamento posterior.

Telis é uma linguagem dinamicamente tipada. Devido a isto, nenhum tipo de verificação quanto a existência de uma agenda é realizada em tempo de compilação, sendo a validade da comunicação direta verificada apenas em tempo de execução. Devido a isto, o polimorfismo é algo natural e inerente. Não é necessário, para utilizar polimorfismo, herança, interfaces ou *casts* ao contrário de linguagens estaticamente tipadas. Para utilizar polimorfismo em Telis basta que os atores (e seus modelos) possuam o subconjunto das agendas utilizadas definidas. Ruby, Python e outras linguagens dinamicamente tipadas possuem o mesmo comportamento. Este tipo de polimorfismo é denominado *duck typing*. Este termo tem origens na frase *"If it quacks like a duck, walks like a duck, it must be a duck!"*. A origem deste termo é desconhecida mas suspeita-se que foi cunhada por Alex Martelli em uma mensagem ao *newsgroup* comp.lang.python.

2.2.6 Escopo de váriaveis

O escopo de variáveis em Telis é diferente do escopo de variáveis em linguagens como Java. Em Java, o escopo de váriaveis é dito ser estático ou léxico. Neste tipo de escopo, varáveis livres são amarradas a definição da variável no escopo léxico ou sintático mais próximo. Por exemplo no código Java da listagem 2.7, a variável x é uma variável livre em relação ao bloco if, então ela é amarrada a variável x encontrada no bloco (elemento sintático) no qual o bloco do if está incluído que é a variável x declarada pelo método umMetodo.

```
public class ExemploEscopo {
        int x = 20;
        public void umMetodo() {
                int x = 10;
                if ("a" == "a") {
                        x = 20;
                }
        }
}
```

Listagem 2.7: Exemplo de escopo estático em Java

Em Telis é utilizado o que é chamado de escopo dinâmico. No escopo dinâmico variáveis livres são associadas à variável de nome idêntico encontrada primeiro na pilha de execução. As variáveis associadas em uma agenda são mantidas até o momento no qual a agenda retorna. Uma exceção a esta regra é a agenda iniciar. As variáveis associadas na agenda iniciar são consideradas variáveis de instância e podem ser acessadas por tratadores de estímulos ou chamadas de agendas feitas através de comunicação direta.

Algumas linguagens conhecidas também implementam escopo dinâmico, dentre elas pode-se citar Perl e Common Lisp.

Para descobrir a quem uma variável deve ser amarrada, é necessário realizar uma análise da pilha de execução. A primeira variável encontrada com mesmo nome analisando o *stack-trace*, isto é, as variáveis associadas na agenda na ordem em que as agendas irão retornar.

O fato de que utilizando-se escopo estático as variáveis livres de um método são amarradas sempre da mesma forma, é a principal diferença em relação ao escopo dinâmico no qual este fato não é necessariamente verdade. Caso uma mesma agenda seja chamada de duas agendas distintas as variáveis da última serão amarradas as variáveis de mesmo nome na agenda que a chamou sendo diferente em cada uma das duas chamadas.

O código Telis na listagem 2.8 ilustra um exemplo onde escopo dinâmico difere do escopo estático. Neste código, apenas um ator do modelo ModeloExemplo é instanciado. Sua agenda iniciar invoca primeiro a agenda1 em seguida a agenda2. A agenda1 associa à variável x o valor 10 e invoca a agenda mostrarX. A agenda mostrarX neste momento acessa a variável associada pela agenda agenda1, já que foi esta quem a invocou, mostrando na tela o valor 10. A agenda2 funciona do mesmo modo. Primeiro associa à variável x o valor 20, em seguida invoca a agenda mostrarX. Mas desta vez a agenda mostrarX irá acessar a variável associada pela agenda2 que é 20, conseqüentemente imprimindo o valor 20 na tela. Caso uma linguagem com escopo estático fosse utilizado neste caso, um erro em tempo de execução ou compilação ocorreria já que a variável x não está definida lexicamente. Apenas em tempo de execução é possível encontrar esta variável.

```
"ExemploEscopo" [ModeloExemplo]
[
        [] "ModeloExemplo"
        [
                "iniciar"
                [
                        agenda1
                        agenda2
                ]
                incluirAgenda

                "agenda1"
                [
                        10 x associar
                        mostrarX
                ]
                incluirAgenda

                "agenda2"
                [
                        20 x associar
                        mostrarX
                ]
                incluirAgenda

                "mostrarX"
                [
                        x mostrar
                ]
```

```
                incluirAgenda
        ]
        incluirModelo
]
incluirAplique
```

Listagem 2.8: Exemplo escopo dinâmico em Telis

A primitiva associar além de associar um valor a uma variável, cria uma nova variável dentro do escopo da agenda atual caso esta variável não consiga ser amarrada a nenhuma outra. Entretanto com este comportamento do associar, pode-se tornar incomodo por exemplo para associar variáveis em agendas recursivas. Como as variáveis possuem o mesmo nome, a primeira chamada a agenda recursiva cria uma variável no escopo desta agenda. A partir daí todas as variáveis de todas as agendas chamadas recursivamente acabam sendo amarradas a agenda da primeira chamada fazendo com que as agendas compartilhem indesejavelmente a variável. Para resolver este problema, existe a primitiva declarar. Esta primitiva cria uma variável na agenda atual, sem tentar amarrá-la a uma variável. Desta forma é possível utilizar agendas recursivas sem problemas. O código da listagem 2.9 mostra o exemplo do fatorial escrito recursivamente.

```
"Fatorial" [Fatorial]
[
        [] "Fatorial"
        [
                "iniciar"
                [
                        20 fatorial
                        mostrar
                ]
                incluirAgenda

                "fatorial"
                [
                        [valor] declarar
                        valor associar
                        valor 0 =
                        [
                                1
                        ]
                        [
                                valor 1 - fatorial
                                valor *
                        ]
```

```
                    entãoSenão
                ]
                incluirAgenda
        ]
        incluirModelo
]
incluirAplique
```

Listagem 2.9: Fatorial escrito recursivamente

2.2.7 Programas Telis e o *bootstrap*

Programas em Telis são denominados apliques. Os atores são criados dentro de apliques. Apliques possuem interface gráfica e podem rodar tanto de forma independente quando dentro de navegadores *web*.

Um aplique é definido pelos seguintes parâmetros:

- nome
- tamanho
- definição dos moldes
- definição de modelos
- lista de modelos que serão instanciados

Ao abrir um aplique o processo de *bootstrap* é iniciado. O interpretador é inicializado, os modelos e moldes são lidos e compilados em seguida a lista de modelos que deve ser instanciada é percorrida e atores são criados. A ordem em que os atores são instanciados na lista não é definida, podendo ser aleatória. A lista pode conter elementos repetidos, caso isto ocorre, um mesmo modelo é instanciado tantas vezes quantas o seu nome estiver na lista.

Apliques podem ser definidos através da própria linguagem, como mostrado no código da listagem 2.6 na página 23. Mas existe uma representação alternativa também utilizada que se baseia em arquivos xml (extensible mark-up language). A definição deste xml não será mostrado ou discutido, mas vale comentar que várias transformações são realizadas sobre este xml. Desde a geração do código em linguagem Telis pura até geração de páginas html (hyper-text mark-up language) descrevendo o código Telis.

2.3 A nova linguagem

A nova linguagem será baseada em Telis. Telis foi utilizado por muito tempo, por muitos programadores e provou ser uma linguagem simples e poderosa. Os conceitos na linguagem permitem que sejam escritos programas básicos, inclusive sem o uso de variáveis, até programas complexos, concorrentes e orientados a objetos.

Entretanto existem alguns pontos em Telis aos quais não foi dado muita atenção, algumas exceções que podem ser resolvidas ou funcionalidades e conceitos que ainda não foram implementadas. Então será realizada uma análise destes pontos nos quais Telis deixa a desejar e propor alterações semânticas e/ou sintáticas com o objetivo de melhorar a linguagem.

2.3.1 Associação de variáveis

A execução de programas Telis pelo interpretador é descrito de forma simples. Primeiramente o interpretador Telis busca a instrução, caso ela seja um símbolo a executa, caso não seja empilha o valor na pilha de dados. Este procedimento é um processo simples, mecânico e de fácil compreensão. Entretanto a primitiva associar viola este processo.

A primitiva *associar* é uma exceção. Quando um símbolo é sucedido de um associar ele não é executado, ao invés disso, é utilizado pela primitiva associar para nomear a variável e o topo da pilha é o conteúdo desta. É como se a primitiva associar causasse uma distorção espaço-temporal, executando antes dela ser buscada pelo interpretador.

Como resultado deste comportamento, no mínimo estranho, da primitiva associar há uma série de confusões causadas entre programadores Telis. Em situações simples esta exceção funciona muito bem e torna a leitura do código muito boa, entretanto em situações mais complexas, como por exemplo montagem dinâmica de nome de variáveis, causa vários problemas.

O fragmento de código mostrado na listagem 2.10 ilustra uma situação na qual o programador Telis normalmente espera um comportamento mas ocorre algo um tanto quanto inesperado. O código, quando lido de forma simplista, se comportaria da seguinte maneira: empilha o 20, “variavel” e “Um”, em seguida concatena “variavel” e “Um” resultando em “variavelUm”. Agora a variavelUm seria associada ao número 20. Este pensamento, apesar de parecer coerente não é o que ocorre. Esta linha de raciocínio ignora o comportamento excepcional do associar. O correto seria o seguinte: empilha o 20, “variavel” e “Um”, agora o símbolo concatenar seguido de um associar, logo o concatenar é o nome da variável e não uma agenda primitiva como esperado. Conseqüentemente, a variável concatenar é associada com o texto “Um”

```
20 "variavel" "Um" concatenar associar
```

Listagem 2.10: Comportamento não usual na associação de váriaveis

Além disso, este comportamento do associar causa um impacto na gramática de Telis (mostrada na listagem 2.1 na página 13). Telis é uma gramática livre de contexto, mais especificamente é uma gramática LL. Gramáticas LL são um subconjunto das linguagens livres de contexto na qual as seguintes restrições são impostas:

- Se $< X > \Rightarrow < V1 > | < V2 > |...| < Vn >$, então $FIRST(V_i) \bigcap FIRST(V_j) = \emptyset$
- Se $< Z > \Rightarrow X$, então $FIRST(X) \bigcap FOLLOW(Z) = \emptyset$
- Recursão a esquerda não é permitida

FIRST(X) é definido como o conjunto de todos os terminais que podem aparecer no início de uma sentença derivada a partir de um não terminal X. FOLLOW(Z) é definido como a união do FIRST(V) para todas as produções na forma: $X \Rightarrow \alpha ZV$.

Caso exista um parser LL que com um número k de *tokens* de *lookahead*[3] possa reconhecer todas as palavras geradas por uma gramática sem realizar *backtracking*, então diz-se que esta gramática é LL(k).

As restrições feitas sobre gramáticas BNF tem o objetivo de simplificar o *parsing*. Existem vários algoritmos para realizar o reconhecimento de sentenças de forma eficiente e quanto menor o valor de k da gramática, mais eficiente este algoritmo se torna.

A produção *palavra* na gramática do Telis, torna a gramática do Telis uma gramática LL(2). Pois há um conflito entre as produções atribuicao e identificador. Se a gramática for alterada e a produção *atribuicao* deixar de existir a gramática poderá ser reduzido a uma gramática LL(1).

Para resolver o problema e tentar tornar o associar uma primitiva normal, como outra qualquer, foi proposto que continue se utilizando um símbolo para indicar o nome de uma variável, entretanto este simbolo deverá ser buscado na pilha. Em Telis, símbolos não podem ir para a pilha de modo simples pois quando o interpretador Telis os encontram ele imediatamente executa. A única forma de fazer isso no Telis antigo é colocando o símbolo dentro de uma lista e em seguida retirá-lo. A proposta para facilitar o empilhamento de símbolos é utilizar um operador, que pode ser utilizado em símbolos e quando este operador é executado ele empilha o

[3]Lookahead define quantos dados de entradas serão levados em consideração em um passo do algoritmo. No caso de *parsers*, o número de tokens que serão levados em conta para realizar uma transição da máquina de reconhecimento.

símbolo que ele está marcando, de forma similar a notação de ponto. Foi proposta também a utilização do caracter $ para realizar tal operação. O fragmento de código mostrado na listagem 2.11 ilustra a associação de variáveis utilizando-se este operador. No exemplo, o símbolo nomeDaVariável é marcado com o operador $ e conseqüentemente não será executado, mas sim empilhado e consumido pelo associar da pilha.

```
20 $nomeDaVariável associar
```

Listagem 2.11: Exemplo de utilização do operador $

Essa abordagem, apesar de tornar o código um pouco menos legível trás importantes vantagens. A utilização deste operador obriga o programador a explicitar quando o símbolo que precede o associar deve ser executado ou é realmente o nome da variável. Outro ponto é o impacto que trás a gramática, tornando-a LL(1) ao invés de LL(2), o que trás vantagens já que parsers para gramáticas LL(1) são muito mais otimizadas e simples do que gramáticas LL com um número maior de *lookaheads*. Além disso, com símbolos sendo facilmente empilhados os torna mais propensos a usos tais como para enumerações e definição de nomes não só de variáveis mas de agendas, modelos e moldes também tornando Telis mais padronizada.

2.3.2 Blocos de comandos

Blocos de comandos são um grupo de instruções que poderão ser executados futuramente através de alguma das primitivas de controle. Em Telis, estes blocos de comandos e listas de dados não são diferenciados. Ambos são tratados da mesma forma, utilizam a mesma construção sintática (listas) e podem ser utilizados intercambiavelmente.

Esta não-diferenciação pode vir a ser útil, permitindo a construção dinâmica de comandos por exemplo. Entretanto, traz alguns problemas, principalmente quando utilizadas em conjunto com o operador valor atual (@) de Telis.

O operador valor atual é utilizado dentro de listas. Sua função é realizar a avaliação de um símbolo, colocando no lugar do mesmo o que restou no topo da pilha após esta avaliação ocorrer. É importante salientar que a pilha é limpa logo após a avaliação ser realizada, isto é, após cada avaliação utilizando-se o operador valor atual a pilha é restaurada para o estado anterior, na iminência da lista ser empilhada. Os símbolo são resolvidos no escopo atual, e nenhuma restrição é posta em relação ao que este símbolo está atrelado. As duas únicas restrições são:

- Não há passagem de parâmetros;
- A avaliação do símbolo deverá retornar pelo menos um valor.

O fragmento de código mostrado na listagem 2.12 mostra como o operador valor atual pode ser utilizado no Telis, e como obter o mesmo resultado sem utilizá-lo. Primeiramente, uma variável é criada e um valor à ela associado, logo em seguida a lista é empilhada mas como esta lista possui um operador valor atual, o símbolo será substituído pela avaliação do mesmo. Como o símbolo é a variável previamente criada, o valor desta variável será colocada no lugar do operador e do símbolo, resultando na lista ["valor da variável"] na pilha, que posteriormente é mostrada na tela. Na próxima linha, é mostrada uma forma alternativa de obter o mesmo resultado sem utilizar o operador valor atual.

```
"valor da variável" aVariavel associar
[@aVariavel] mostrar
aVariavel 1 gerarLista mostrar
```

Listagem 2.12: Exemplo de utilização do operador @

O problema deste operador é que se espera comportamentos distintos quando encontrados em listas de dados e em blocos de comandos. Em listas de dados, espera-se que os valores das variáveis sejam substituídos imediatamente, utilizando-se os valores das variáveis no momento em que a lista é empilhada para realizar as substituições. Já em blocos de comandos, espera-se que os operadores valor atual não atuem, mas que sejam ignorados e utilizados quando o bloco for executado. Afinal, bloco de comandos definem código para ser executados posteriormente, e o operador valor atual é um elemento do código Telis e portanto deve poder ser expressado para execução futura em blocos.

Como não existe nenhum tipo de distinção sintática entre blocos de comandos e listas, deve ser realizada uma analise semântica para descobrir o que uma lista representa e quais ações deverão ser tomadas. Este tipo de análise é complexa de ser feita devido a própria estrutura de Telis, no momento em que a lista vai para a pilha não há nenhum indício do objetivo desta pilha estar sendo empilhada. Este conflito de comportamento causado pelo operador valor atual foi resolvido no interpretador Telis utilizando-se um algoritmo baseado em *backtrack*. Assume-se que uma lista é uma lista de dados, caso mais tarde se descubra que na realidade é um bloco de comandos o *backtrack* é realizado e o valor correto da lista utilizado. Esta abordagem funciona para grande parte dos casos, entretanto causa alguns problemas. Como é utilizada uma heurística para decidir qual das duas abordagens deve ser tomada há uma chance de se errar e caso se erre há a possibilidade de que um valor que não deveria ser avaliado seja podendo resultando em *loops* infinitos e alguns outros comportamentos inesperados.

Para resolver este problema foi proposta a distinção sintática entre listas de dados e blocos de comandos. Na nova linguagem que está sendo desenvolvida, as listas de dados serão deli-

mitadas por colchetes pareados, da mesma forma que listas em Telis. Blocos por sua vez serão representados através de chaves, também pareadas.

A listagem 2.13 contém um fragmento de código da nova linguagem que mostra como listas de dados e comandos serão representados.

```
[0 4] gerarAleatório umValor associar
umValor 2 >=
        {
                "valor maior ou igual a 2" mostrar
        }
        {
                "valor menor que 2" mostrar
        }
entãoSenão
```

Listagem 2.13: Listas de dados e blocos de comandos

Além da resolução de problemas relativo ao operador de valor atual, a distinção entre estruturas de dados e blocos de comandos é forte em outras linguagens. Conseqüentemente a adoção desta distinção na nova linguagem favorece a transição para estas outras linguagens e vice-versa.

2.3.3 Escopo de variáveis e closures

Como foi exposto anteriormente, o escopo das variáveis em Telis são do tipo dinâmico. Apesar de relativamente simples, alguns de seus efeitos não são muito intuitivos quando comparados ao escopo estático (léxico). O fato da avaliação de uma variável depender do contexto em que uma chamada a agenda for feita pode levar a erros difíceis de detectar. Por este motivo, resolveu-se utilizar o sistema de escopo de variável estático para a nova linguagem.

Em escopos estáticos a definição de qual variável está sendo referenciada pode ser realizado em tempo de compilação, diferentemente do escopo dinâmico onde só em tempo de execução isto será decidido.

Em uma linguagem com escopo estático, uma variável A declarada em um bloco B é acessível a um bloco C declarados dentro de B após a declaração de A. Note que caso exista um bloco D declarado dentro de C, D também poderá acessar a variável A. Caso B e C possuam duas variáveis diferentes de mesmo nome, o bloco D irá utilizar a variável C pois está é a que está lexicamente mais próxima a D.

Em Telis, os blocos de comandos (que não passam de listas comuns) são tratadas como elementos de primeira classe, isto é, podem ser passadas como parâmetros ou retornadas (através da pilha). Nenhuma restrição é imposta neste sentido. Seria interessante manter blocos de comandos elementos de primeira classe na nova linguagem também.

Em Telis o escopo dinâmico facilita que blocos de comandos sejam elementos de primeira classe. Como a variável é amarrada no momento em que é executada não há problemas quanto as variáveis, elas serão amarradas quando o bloco for executado e fim. Já com escopo estático não funciona deste modo. Com escopo estático a variável é amarrada no momento em que o bloco é definido. Se o bloco for executado imediatamente, não ocorrem problemas. Entretanto, caso o bloco seja passado para uma outra agenda ou for retornado, o escopo no qual o bloco foi definido deve ser mantido para que as variáveis definidas no bloco possam se referenciar ao ambiente no qual se encontravam quando necessário.

Closures são fragmentos de código que mantém uma referência ao ambiente no qual foram criados. Desta forma, o retorno de um método não necessariamente implica na destruição das variáveis locais deste já que um closure pode estar referenciando estas variáveis locais. Isto torna a pilha de execução não mais uma pilha, mas sim uma árvore.

Closures são muito utilizados em linguagens como Ruby, Python, Lisp, entre outras. Além disso, closures provém uma forma elegante para a implementação de alguns padrões de projetos tais como Visitantes. O código da listagem 2.14 mostra um código escrito em Ruby que utiliza um visitante para calcular o somatório de uma lista de valores. Ao final da execução, o valor 45 é mostrado na tela.

```
valores = [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
somatorio = 0
valores.each do |x|
        somatorio = somatorio + x
end
puts somatorio
```

Listagem 2.14: Exemplo uso de closures em Ruby

Há linguagens nas quais não existem closures, apesar do escopo estático, como Java[4]. Há quem confunda as classes anônimas de Java com closures, entretanto não são iguais. Classes anônimas, apesar de poderem ser utilizadas para algumas coisas da mesma forma que closures, não são closures completos pois não mantém uma cópia para o contexto no qual fora criada.

[4]Java ainda não possui closures, há um grupo de especialistas estudando a implementação, sintaxe e semântica de closures na linguagem. Estima-se que closures serão incorporados a linguagens na versão 7 do Java.

Java permite apenas que variáveis finais do escopo no qual a classe foi criada sejam acessadas, permitindo que uma cópia da variável externa seja feita para a classe anônima.

A introdução de closures entretanto acarretam um problema ao modelo de concorrência dos atores. O modelo de atores, por definição, não possuem memória compartilhada sendo toda a comunicação entre eles realizada via troca de mensagens. Closures carregam consigo o contexto no qual foram definidos, logo se um closure é definido em um ator A e posteriormente é enviado a um ator B e lá executado, o ator B passa a executar operações concorrentemente com o ator A na área de memória deste, causando uma condição de corrida (*race condition*).

Para evitar que este problema ocorra, há uma restrição em quem pode executar os closures. Apenas o ator que criou um closure poderá executá-lo. Desta forma, a condição de corrida será evitada, em contrapartida o potencial de utilização de closures será um pouco reduzido.

2.3.4 Objetos

Em Telis a unidade de execução básica é um ator. Todo ator, por definição, roda independentemente de qualquer outro ator, isto é, possui uma linha de execução própria. Além disso, ele só é destruído quando executa a primitiva suicidar. Mesmo que termine sua agenda iniciar, não pode simplesmente ser destruído já que ele pode receber estímulos e ter de agir de acordo com estes. Após o término de execução da agenda iniciar diz-se que ator entra em um estado de hibernação, do qual pode ser acordado para tratar algum estímulo.

Atores é uma boa forma de organizar o programa, mas não é sempre isso que se deseja. Muitas vezes não se quer uma entidade autônoma para resolver uma tarefa, mas sim algo mais simples. Neste sentido, se propõe o conceito de objeto.

A nova linguagem deverá possuir o conceito de objeto. Objetos assim como atores são definidos através de modelos que serão chamados de modelos de objetos. Modelos de objetos são exatamente iguais à modelos de atores, com a exceção de que quando instanciados não criam um ator, mas sim um objeto. Um modelo de objetos, a exemplo de modelos de atores, poderão ser moldados por moldes e possuem uma coleção de agendas.

Objetos são passivos, ao contrário de atores que são ativos. Objetos não possuem atrelados a si uma linha de execução, portanto dependem de um ator para poderem executar o seu código. Por este motivo toda e qualquer ação realizada com um objeto será síncrona com o ator que ocasionou a reação do objeto. Devido a isso não lhes será permitido instalar tratadores de estímulo dizer, já que estes possuem natureza assíncrona. Note que não foram feitas restrições quanto a emissão de estímulos, estes poderão ser emitidos sem problemas. A comunicação

direta, ao contrário dos estímulos dizer, são inerentemente síncronos, logo esta abordagem será utilizada para permitir que atores possam interagir com objetos, e estes com outros objetos.

Para a criação de modelos de objetos será utiliza a primitiva incluirModeloDeObjeto que possuirá os mesmos parâmetros da primitiva incluirModelo. Isto é, uma lista de moldes e um bloco de comandos contendo a definição das agendas. Um exemplo de utilização do incluirModeloDeObjeto é mostrado na listagem 2.15.

```
[Molde1 Molde2] $NomeDoModeloDeObjeto
{
        $iniciar
        {
                [parametro] associar
        }
        incluirAgenda

        $somar
        {
                parametro +
        }
        incluirAgenda
}
incluirModeloDeObjeto
```

Listagem 2.15: Exemplo da criação de modelos de objetos.

A instanciação de objetos também se dá de forma síncrona. De forma análoga à atores, quando instanciados objetos executam sua agenda iniciar, mas devido à restrição imposta sobre todas as operações realizadas sobre objetos, é garantido que a agenda iniciar será executada até o final antes do controle de fluxo retornar a quem o instanciou, ficando este bloqueado. Da mesma forma que na instanciação de atores, o primeiro elemento da lista de quem está instanciando um objeto é passado como parâmetro para a agenda iniciar do objeto.

2.3.5 Inicialização de apliques

Linguagens de programação como C++ e Java utilizam-se de uma marcação para definir as classes que por sua vez contém métodos que contém o código. Em C/C++ o ponto de entrada de programa é descoberto através de uma convenção na nomenclatura de método. O método chamado *main* será o ponto de entrada do programa. Em Java a definição de pontos de entradas são definidos por métodos estáticos chamados de *main* contidos em classes. O nome de uma classe

contendo este método é então passada para a Máquina Virtual Java que inicializa o programa invocando-o.

Este modo de inicialização no qual o ponto de entrada é dado por uma convenção de nomenclatura possui vantagens como simplicidade.

A proposta de Telis é um pouco mais complexa. A inicialização de apliques em Telis se dá especificando um aplique e suas propriedades. Isto se dá através da declaração dos modelos e moldes e suas respectivas agendas que por sua vez contém o código da aplicação. Além desta definição, ainda são declarados alguns meta-dados que definem alguns outros parâmetros do aplique tal como seu nome, dimensões e atores que deverão ser instanciados.

Claro que ninguém escreve um aplique na mão. Programadores Telis se valem do ambiente de desenvolvimento que torna todo o processo de definição de atributos gráfico, deixando ao programador o foco apenas no que é importante.

Visando simplificar este processo, a nova linguagem deverá possuir um novo sistema de inicialização mais simples e menos verboso, isto é, aplicações simples devem poder ser descritas de forma mínima. Linguagens como Python e Ruby conseguiram isto. Não é necessário especificar nenhum ponto de entrada no programa. O código inteiro pode ser visto como a definição de um método que será invocado, podendo-se escrever instruções diretamente no arquivo fonte sem a necessidade de marcar trechos de código. Com a evolução do programa, pode-se então criar novas classes, métodos, entre outras coisas demarcando trechos de código com palavras chaves, mas a princípio isto não é necessário.

A nova linguagem adotará esta idéia presente nestas linguagens. Um código fonte nada mais é do que a definição da agenda iniciar de um modelo que será instanciado automaticamente, sendo criado o primeiro ator do sistema. Desta forma, o programa mostrado na listagem 2.16 será válido e faria exatamente a mesma coisa que o programa mostrado na listagem 2.17. Como pode-se notar há uma grande simplificação do código na linguagem nova em relação ao Telis nos casos mais simples.

```
10 $x associar
20 $y associar
x y + mostrar
suicidar
```

Listagem 2.16: Exemplo programa linguagem nova.

```
[50 50] "ProgramaEquivalente" [ModeloInicial]
[
        [] "ModeloInicial"
```

```
        [
                "iniciar"
                [
                        10 x associar
                        20 y associar
                        x y + mostrar
                        suicidar
                ]
                incluirAgenda
        ]
        incluirModelo
]
incluirAplique
```

Listagem 2.17: Programa equivalente ao da listagem 2.16 em Telis.

2.3.6 Ambientes de execução

A linguagem que será desenvolvida deverá poder rodar em ambientes distintos, com peculiaridades diferentes. Um ambiente de execução é o local no qual o aplique rodará. A princípio serão definidos três ambientes básicos, mas não são limitados apenas a estes:

- Aplicação textual (console)
- Aplicação gráfica baseada em turtle graphics
- Aplicação servidor

Um aplique deverá ser capaz de rodar em um destes três ambientes. Entretanto há diferenças muito grandes com relação ao que deverá ser oferecido ao programador em cada um destes ambientes. Em ambientes gráficos deve-se oferecer formas de manipular diferentes objetos gráficos, coisa que não deverá existir em aplicações textuais e de servidor.

Linguagens como Java resolvem este tipo de problema, fazendo uma linguagem absolutamente neutra e entrega APIs (*Application programmers interface*) e *frameworks* que encapsulam as peculiaridades destes ambientes. Esta é uma abordagem simples, que transfere a responsabilidade da linguagem para a biblioteca desta mesma linguagem. Entretanto, Telis assim como a nova linguagem tem como principal objetivo ser simples. Toda a funcionalidade inerente a um dado ambiente deve estar disponível de forma simples sem a necessidade de utilizar

frameworks ou APIs complexas. Telis resolveu o problema se limitando a um ambiente apenas, o gráfico.

A nova linguagem deverá ser capaz de rodar independentemente do ambiente. Note que não foi mencionado que um mesmo aplique deverá ser capaz de rodar em mais de um ambiente simultaneamente.

O paradigma de orientação a objeto possui um princípio chamado princípio de separação de preocupações (*separation of concerns*). Este princípio diz uma classe deve possuir uma e apenas uma responsabilidade, quando uma classe começa a ter mais responsabilidades do que deveria, então ela deve ser dividida em outras classes.

Aplicando este princípio, podemos dizer que a nova linguagem na verdade será composta por um framework comum e especializada para cada um destes diferentes ambientes de execução. Quando o núcleo da linguagem é especializado para um determinado ambiente de execução, agendas primitivas especializadas poderão ser incluídas, modelos/moldes definidos, a inicialização personalizada, integração com o ambiente em questão realizada e qualquer outra ação que se achar necessária.

O inconveniente desta abordagem é a existência de vários dialetos da linguagem, desta forma um aplique escrito em um ambiente não rodará em outro e o programador deverá estar ciente disto e escolher o dialeto apropriado para o ambiente de execução desejado. Entretanto, em linguagens comuns, programas que funcionam em vários ambientes de execução sem alterações são raros, gerando erros em ambientes não suportados.

2.3.7 Definição sintática e léxica formal da nova linguagem

A definição léxica da nova linguagem intermediária é mostrada na listagem 2.18. A definição léxica é bastante simples, sendo definidos os terminais para comentários, números, textos, símbolos e operadores. O operador de pré-avaliação de Telis não será implementado na nova linguagem, mas poderá ser integrado futuramente.

```
<COMENTARIO> = "(" (~["}"])* ")"
<TEXTO> = "\"" ( ~["\\","\n","\r","\""] | ("\\" ["n","t","\\","\""]) )*
    "\""
<NUMERO> = ("-")? (<FLOAT> | <FLOAT> ( ["e","E"] (["-","+"])? <INTEIRO> )?)
<FLOAT> = <INTEIRO> | <INTEIRO> ( ["."] <INTEIRO> )?
<INTEIRO> = ["0" - "9"]+
<SIMBOLO> = ( <LETRA> | "_" )+ ( <DIGITO> | <LETRA> | "_")*
<LETRA> = ["a"-"z", "A"-"Z"] | "á" | "à" | "â" | "ä" | "ã" |
                  "Á" | "À" | "Â" | "Ä" | "Ã" | "é" | "è" | "ê" | "ë" |
```

```
                "É" | "È" | "Ê" | "Ë" | "í" | "ì" | "î" | "ï" |
                "Í" | "Ì" | "Î" | "Ï" | "ó" | "ò" | "ô" | "ö" | "õ" |
                "Ó" | "Ò" | "Ô" | "Ö" | "Õ" | "ú" | "ù" | "û" | "ü" |
                "Ú" | "Ù" | "Û" | "Ü" | "ç" | "Ç"
<OPERADOR> = ("+" | "-" | "*" | "/" | "\\" | "^" | "=" | "~=" | "<" | ">" |
    "<=" | ">=")
```

Listagem 2.18: Definição léxica da nova linguagem

A gramática da nova linguagem é mostrada na listagem 2.19. Como é possível notar, em comparação com a gramática do Telis mostrada na listagem 2.1 na página 13 a gramática da nova linguagem está muito mais simples. Grande parte da simplificação ocorre devido a nova abordagem de lançamento adotada na linguagem nova.

```
instrucoes ::= ( instrucao )* <EOF>
instrucao ::= texto | operador | operadorDeEmpilhamento |
   operadorDeComunicacaoDireta
               | numero | simbolo | lista | bloco
texto ::= <TEXTO>
numero ::= <NUMERO>
simbolo ::= <SIMBOLO>
operador ::= <OPERADOR>
operadorDeEmpilhamento ::= "$" <SIMBOLO>
operadorDeComunicacaoDireta ::= "." <SIMBOLO>
lista ::= "[" ( instrucao )* "]"
bloco ::= "{" ( instrucao )* "}"
```

Listagem 2.19: Gramática BNF da nova linguagem

3 *Implementação do Interpretador*

A implementação do interpretador foi realizado utilizando-se a linguagem de programação Java. Java é uma linguagem moderna, surgida em meados da década de 90 muito utilizada nos dias de hoje para programação profissional. É uma linguagem orientada a objetos, fortemente tipada.

3.1 Visão geral

O interpretador foi desenvolvido utilizando-se o paradigma de orientação a objetos. E foi subdividido em diversos módulos, tendo cada um uma responsabilidade dentro do sistema.

O diagrama de classes completo será omitido devido ao seu tamanho. O sistema completo é composto por algumas centenas de classes e se torna impraticável colocá-lo na integra. Entretanto os pontos mais interessantes serão abordados.

O interpretador é dividido em três grandes áreas como ilustrado na figura 3.1. Os *wrappers* são uma biblioteca que implementa uma hierarquia de tipos responsáveis por encapsular os tipos de dados primitivos de Java, como por exemplo *Strings*, números(*double*) nome, além de representar os tipos de dados únicos em Telis, como os símbolos, e fornecer operações sobre estes que ainda não existem nativamente na linguagem. Além disso, possuem uma superclasse comum permitindo que sejam criadas estruturas de dados parametrizadas. O coração da máquina é responsável pela maior parte da funcionalidade do interpretador, ele implementa a estrutura básica da linguagem, as ações semânticas independentes do ambiente, o parser da linguagem, a estrutura de concorrência e sincronização, etc.. Por fim, o ambiente será responsável em adaptar o coração da máquina para cada um dos ambientes suportados, implementando primitivas dependente do ambiente, realizando a inicialização da máquina e do ambiente no qual se encontra entre outras funções. Para este trabalho serão desenvolvidos dois ambientes de execução distintos. O primeiro é baseado em console, o segundo será gráfico sendo disponibilizadas agendas primitivas para efetuar operações típicas de *Turtle Graphics*, de Logo.

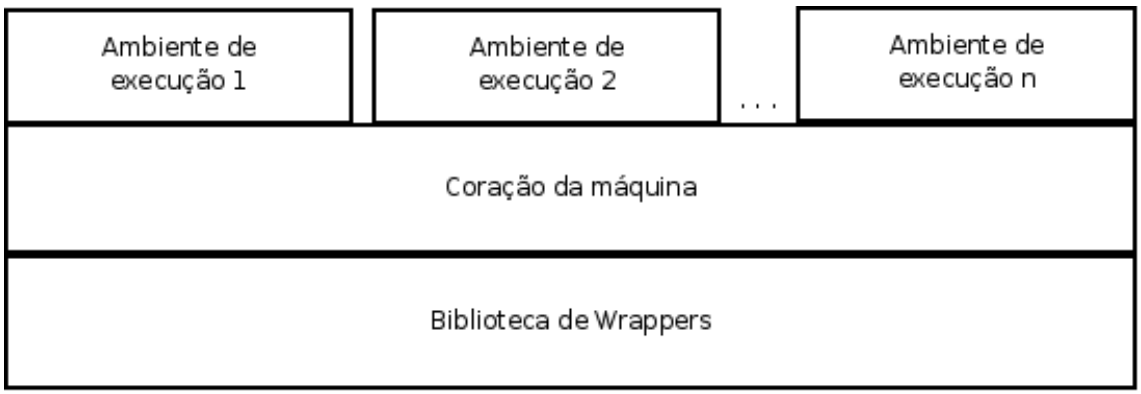

Figura 3.1: Diagrama de camadas do sistema.

3.2 Visão geral da biblioteca de *wrappers*

A biblioteca de wrappers, como já foi comentado, encapsula os tipos de dados primitivos de Java com o objetivo de:

1. Diferentes tipos de dados pertencerem a uma mesma hierarquia de classes
2. Definir operações comuns sobre tipos fora do coração da máquina

O primeiro objetivo se deve ao fato de melhor utilizar os tipos genéricos introduzidos na versão 5 de Java. Além disso, permite a definição de métodos abstratos na raiz da hierarquia de classes e a utilização de polimorfismo.

O segundo objetivo visa aumentar a coesão, removendo código de manipulação dos dados das classes pertencentes ao coração da máquina e movendo-o para classes especializadas.

Os módulos e classes de um sistema orientados devem apresentar alta coesão, isto é, cada classe e método deve possuir uma única responsabilidade. Módulos e classes não coesas acarretam problemas de manutenibilidade de código, já que se torna difícil localizar onde determinada funcionalidade deverá ser encontrada e quando encontrada, ela normalmente está misturada com código responsável por outras funcionalidades fazendo com que mudanças possam acarretar conseqüências não previstas.

A figura 3.2 mostra o diagrama de classes da hierarquia de *wrappers*. Nota-se que há algumas classes como OperadorDeComunicacaoDireta, ListaDeComandos e OperadorDeEmpilhamento que representam estruturas e particularidades do próprio código Telis. Estas classes existem pois o interpretador mantém internamente os dados e o código da mesma forma, utilizando listas de palavras. A primeira linguagem a possuir esta característica foi Lisp, esta propriedade é chamada de *homoiconic*.

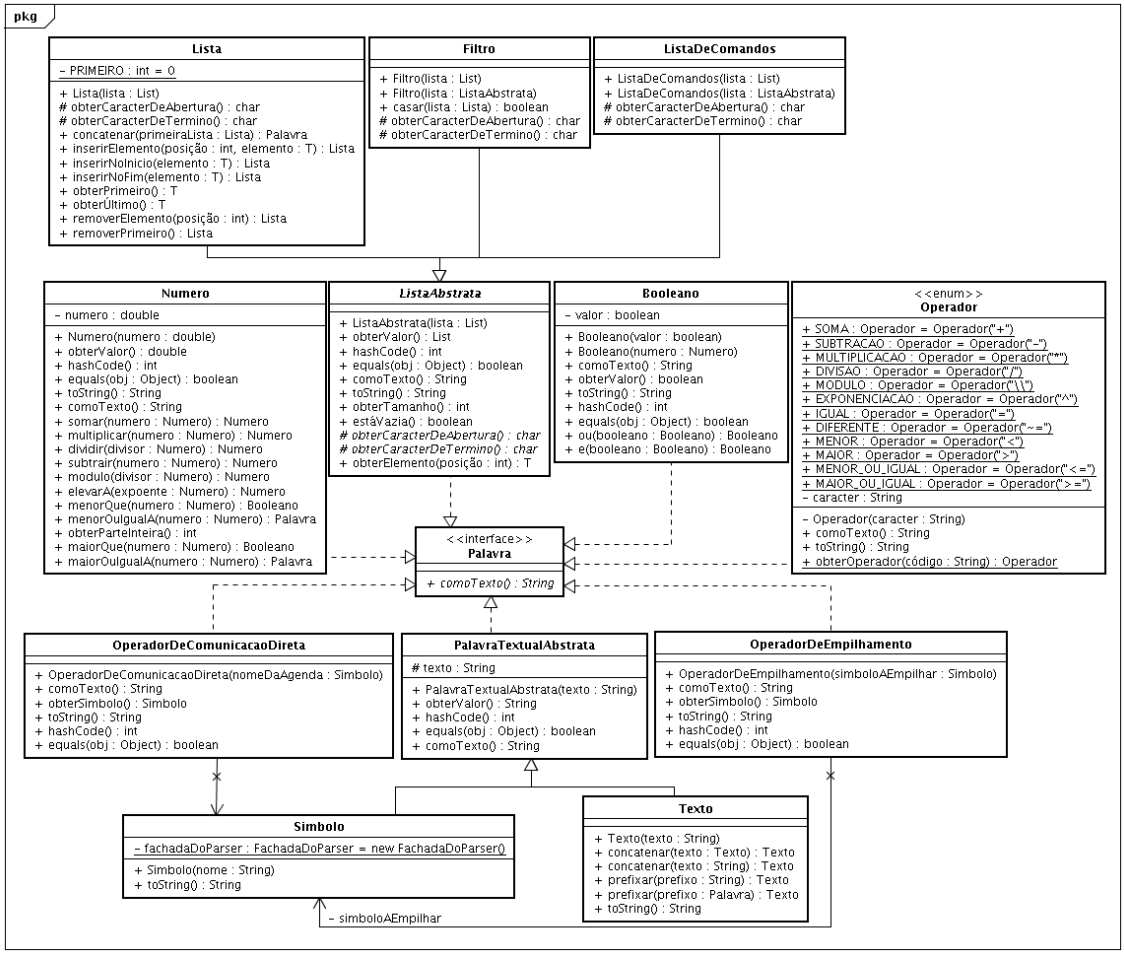

Figura 3.2: Diagrama de classe mostrando hierarquia dos *wrappers*.

Além destas classes, existe ainda mais uma classe adicional junto à esta estrutura. É uma instância do padrão de projeto *Builder*, cujo nome é ConstrutorDeListas. O objetivo do padrão de projeto *Builder* é separar a construção de um objeto complexo de sua representação. A figura 3.3 mostra a estrutura típica de um *Builder*.

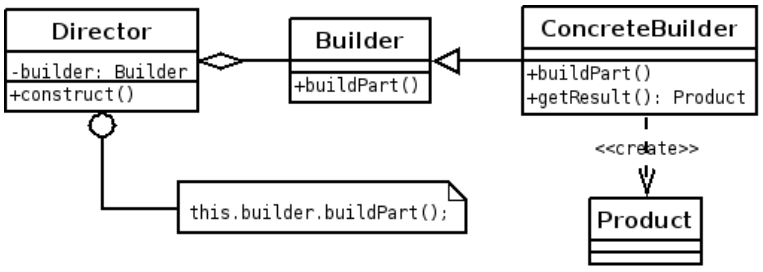

Figura 3.3: Diagrama de classe mostrando estrutura geral do padrão *builder*.

O padrão builder, conforme descrito por Gamma et al. (1995), possui os seguintes elementos com suas respectivas responsabilidades:

- *director*: constrói um objeto utilizando a interface do builder

- *builder*: especifica uma interface para um construtor de partes do objeto product
- *concrete builder*: define uma implementação da interface builder
- *product*: o objeto complexo em construção

Optou-se em utilizar o padrão *Builder* para permitir a criação de listas e a definição de seu conteúdo sem a necessidade de conhecimento da hierarquia mostrada na figura 3.2. A listagem 3.1 mostra um fragmento de código Java criando uma lista com e sem a utilização do *Builder*. Nota-se que o código que utiliza o builder não precisa ter conhecimento sobre os detalhes de construção dos objetos, pois estes detalhes estão encapsuladas no ConstrutorDeListas.

```
// Utilizando Builder
ConstrutorDeListas construtor = new ConstrutorDeListas();
Lista<Palavra> lista = construtor.adicionarTexto("umTexto")
                                        .iniciarLista()
                                                .adicionarNúmero(10)
                                        .terminarLista()
                                        .obterSaidaComoLista();
// Sem utilizar Builder
List<Palavra> conteudo = new ArrayList<Palavra>();
conteudo.add(new Texto("umTexto"));
List<Palavra> primeiroElemento = new ArrayList<Palavra>();
primeiroElemento.add(new Numero(10));
conteudo.add(new Lista<Palavra>(primeiroElemento));
Lista<Palavra> listaEquivalente = new Lista<Palavra>(conteudo);
```

Listagem 3.1: Ilustrando criação de listas. A lista nas variáveis lista e listaEquivalente são iguais.

O ConstrutorDeListas possui um ciclo de vida representado pela máquina de estados mostrado na figura 3.4. Quando criado, o ConstrutorDeListas se encontra no estado Aberto e espera que a descrição da lista seja feita. Após a lista (que faz o papel de *product*) ser retirada do ConstrutorDeListas através dos métodos obterLista ou obterListaDeDados ocorre a transição para o estado Fechado. Neste estado o objeto para de responder as mensagens, devendo ser descartado. Caso alguma mensagem seja enviada à este objeto uma exceção será lançada. Para implementar este comportamento utilizou-se o padrão de projeto Estado.

O diagrama da figura 3.5 mostra a classe ConstrutorDeListas e o padrão Estado responsável pela gerência do ciclo de vida do objeto.

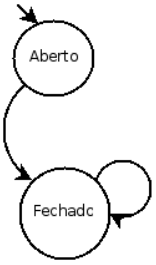

Figura 3.4: Diagrama de estados mostrando ciclo de vida de um ConstrutorDeListas.

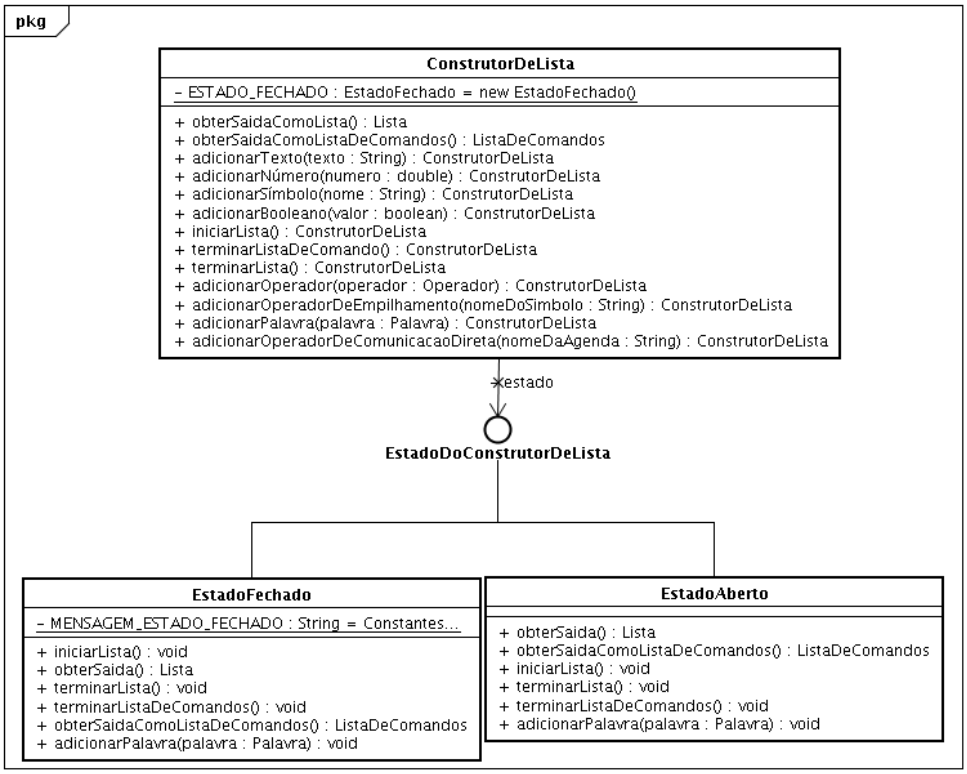

Figura 3.5: Diagrama mostrando estrutura do ConstrutorDeListas e de seus Estados.

3.3 O coração da máquina

O coração da máquina é o cerne do interpretador. Ela é um framework cuja responsabilidade é oferecer a infreestrutura básica da linguagem de forma que cada um dos ambientes de execução possa personalizar este framework da forma que for necessária.

O coração da máquina é responsável por:

- Realizar o *parsing* da entrada
- Possuir infraestrutura para a interpretação
- Oferecer primitivas básicas como, por exemplo, primitivas de controle de fluxo

- Oferecer mecanismos de herança, isto é, moldes e modelos
- Implementar closures
- Implementar detalhes básicos da linguagem como váriaveis e escopos
- Implementar Atores, incluindo os seus mecanismos de comunicação

Além destes detalhes que o coração da máquina deve implementar há uma série de outras características desejáveis:

- Oferecer mecanismo simples para inclusão de novas primitivas
- Oferecer mecanismos para simplificar a configuração e criação de novos ambientes

Como pode-se notar o coração da máquina é a parte mais complexa da implementação e ainda deve ser flexível o bastante para que possa atuar em diversos ambientes. Para explicar o funcionamento e organização do interpretador será explicado o que ocorre a partir da classe AmbienteAbstrato. A classe AmbienteAbstrato é uma classe abstrata que deve ser estendida pelos ambientes. Ela é responsável por simplificar o desenvolvimento de novos ambientes realizando tarefas comuns a todos os ambientes de execução como a inicialização dos subsistemas, configuração da máquina (interpretador) e a criação do ator inicial. Esta classe é uma instância do padrão *Method Template*, descrevendo o algoritmo de inicialização da máquina de forma geral, deixando detalhes específicos para implementação nas subclasses através de métodos abstratos. Neste caso, as subclasses são implementadas pelos utilizadores do framework.

A classe AmbienteAbstrato possui os métodos mostrados no diagrama de classe mostrado na figura 3.6. Seu construtor recebe um Reader, abstração de um fluxo de caracteres implementado na biblioteca padrão de Java, com o código fonte que deverá ser executado pelo interpretador. Então o método iniciarAplique deve ser invocado pelo ambiente de execução. Este método tem quatro passos básicos:

- configurar ambiente de execução
- enviar código do programa para o parser
- criar um ator para executar o programa
- iniciar a execução do ator

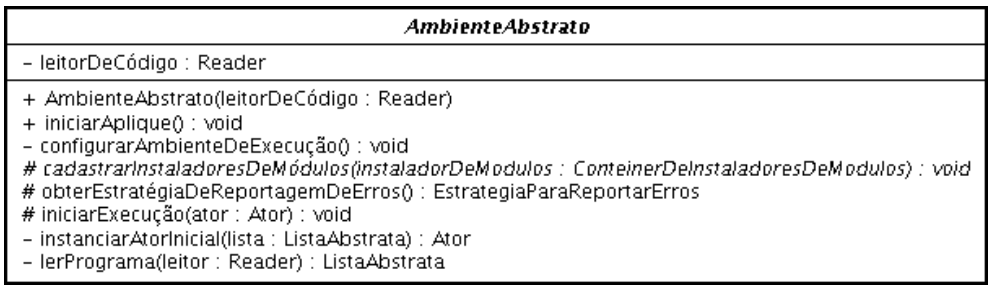

Figura 3.6: Diagrama de classes de AmbienteAbstrato.

Daqui pode-se perceber como ocorre o início da interpretação. É responsabilidade do ambiente de execução obter um *Reader* com o código fonte e passá-lo ao AmbienteAbstrato. O AmbienteAbstrato irá redirecionar ao *parser*. O parser irá por sua vez gerar uma estrutura de dados representando o programa e o texto pode então ser descartado. Depois disso, os subsistemas necessários são inicializados e um ator é criado e lhe é passado o programa para execução. O diagrama da figura 3.7 mostra esquematicamente o que cada módulo recebe como entrada e produz até que o ator possa ser iniciado.

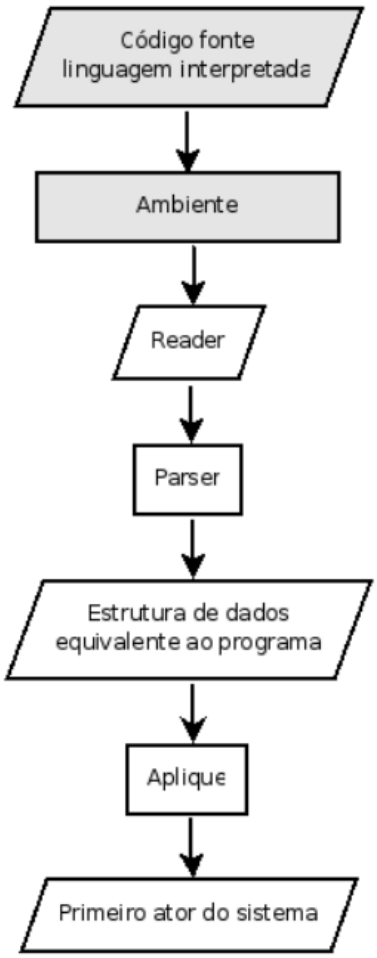

Figura 3.7: Diagrama esquemático da entrada e saída até criação do primeiro ator.

3.3.1 O Parser

Para criar o parser do interpretador foi utilizado um *compiler-compiler* chamado JavaCC. Um *compiler-compiler* é um compilador que recebe como entrada uma especificação formal, normalmente uma gramática baseada em EBNF, e gera um parser em uma linguagem de programação convencional. JavaCC gera um parser LL(k) na linguagem Java.

Existem vários outros *compiler-compilers* para Java, mas uma das principais vantagens do JavaCC é que o código gerado por ele não depende de nenhuma outra biblioteca como é comum em outras aplicações. As classes geradas pelo JavaCC são mostradas no diagrama da figura 3.8.

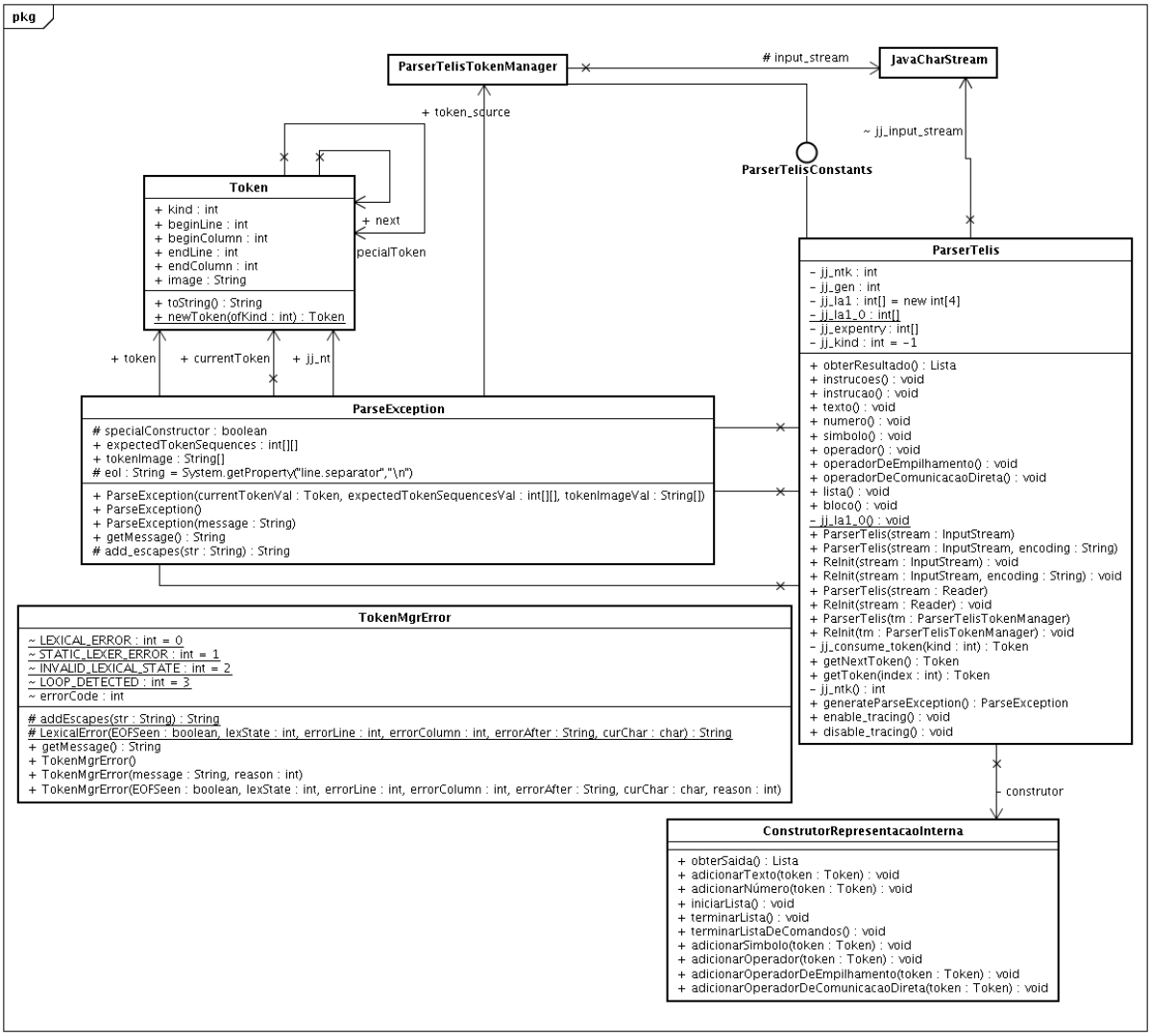

Figura 3.8: Diagrama das classes geradas pelo JavaCC.

O JavaCC permite que ações sejam realizadas quando uma sentença da linguagem é reconhecida. Estas ações são fragmentos de código que podem realizar ações arbitrárias. No caso do interpretador da nossa linguagem, estas ações são utilizadas para gerar uma estrutura de

dados intermediária representando o programa. A estrutura gerada durante o *parsing* pelo interpretador é uma Lista, e esta lista pode possuir qualquer elemento do tipo Palavra (hierarquia mostrada na figura 3.2).

Para facilitar a geração e diminuir a quantidade de código dentro do parser gerado pelo JavaCC é utilizado sistema de *callback*, isto é, toda vez que uma determinada seqüência é reconhecida pelo interpretador uma mensagem é enviada a um objeto previamente cadastrado. Este objeto é uma instância da classe ConstrutorRepresentacaoInterna. Esta classe é uma instancia do padrão de projeto *Adapter*. Um *adapter* adapta a interface de um objeto para uma segunda interface esperada pelo cliente, sendo responsabilidade dele transformar os parametros recebidos para os esperados. A classe ConstrutorRepresentacaoInterna adapta a classe ConstrutorDeListas para a interface de notificação esperada pelo parser.

Várias classes são geradas pelo JavaCC, cada uma responsável por uma parte do processo de parsing. A classe ParserTelisTokenManager é responsável por realizar a analise léxica, gerando tokens para a classe ParserTelis que implementa o analisador sintático. As demais classes são classes auxiliares utilizadas como estruturas de dados (Token por exemplo), classes utilitárias (como JavaCharStream) ou classes para reportar erros.

3.3.2 Classes Aplique, Ator e Modelo

Após o *parsing* ter sido concluído, e a representação interna do programa gerada o próximo passo tomado pelo algoritmo de inicialização implementado em AmbienteAbstrato é a criação de um ator e o início de sua execução.

A classe Ator é a abstração de atores no interpretador. O diagrama de classes da figura 3.10 mostra a classe Ator e a sua relação com as classes Modelo e Aplique.

A classe Aplique é uma instância do padrão de projeto *Method Factory*. Ela é responsável pela criação de novos atores, durante a criação de um ator, o método fábrica além de criá-lo deve passar para o ator o seu parâmetro inicial, o modelo a qual o Ator pertence e fornecer à ele uma identidade única. Além destas responsabilidades, a classe Aplique deve manter uma coleção contendo todos os atores criados no sistema, para que, por exemplo, atores possam ser notificados de estímulos emitidos. Além de usar o padrão *Method Factory*, o padrão *Singleton* também é usado nesta classe, pois o interpretador deve possui um e apenas um Aplique criando e mantendo referencias para os Atores.

A classe Aplique ainda oferece a possibilidade de uso do padrão de projeto *Visitor* para que um visitante visite todos os atores instanciados, de forma que durante o processamento nenhum

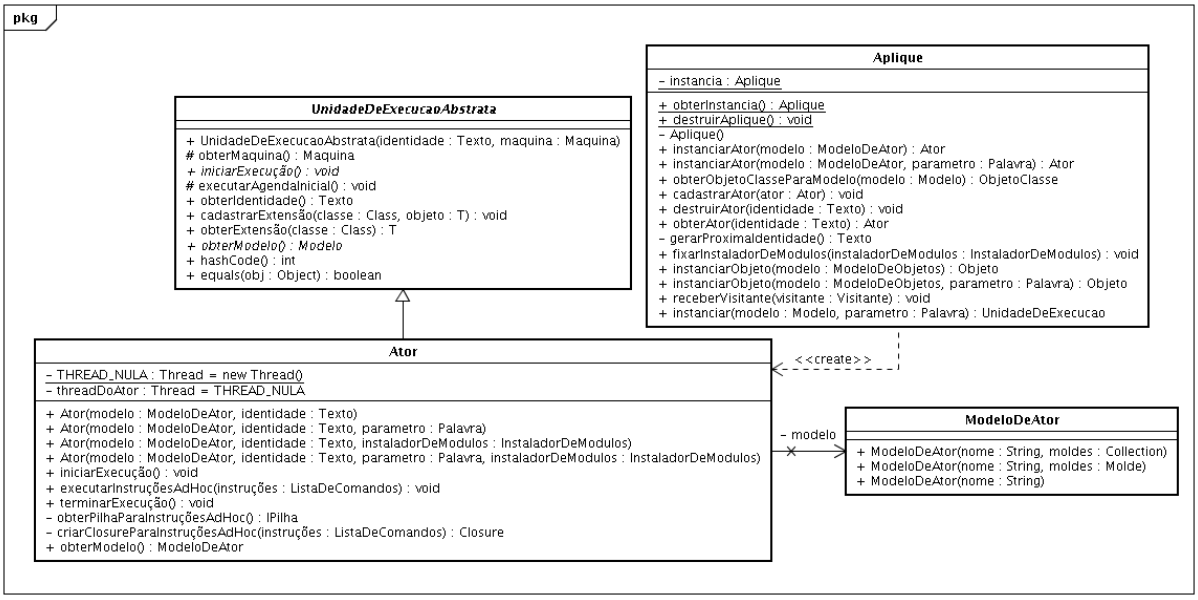

Figura 3.9: Diagrama de classes de Ator, Aplique e Modelo.

ator novo seja criado ou destruído, evitando problemas de concorrência.

A classe Modelo guarda a descrição do ator, isto é, a descrição de suas agendas (nome e código) e moldes. Oferece métodos para acesso a decrição e para a alteração da mesma. Todo Ator está atrelado a um Modelo, e este deve ser informado durante a construção do Ator.

O ator possui o ciclo de vida mostrado no diagrama de estados da figura 3.10. Quando o ator é instanciado ele se encontra no estado Criado. Após o método iniciarExecução da classe Ator ser invocado, o ator em questão transita para o estado Executando. Após a transição, a execução de sua agenda iniciar é realizada em uma linha de execução (*Thread*) própria. Após a agenda iniciar ter sido concluída e retornado, o ator entra no estado Hibernando. Neste estado o ator apenas espera por eventos. Um evento pode ser a chegada de um estímulo emitido por um dizer ou uma comunicação direta, e em decorrência destes estímulos o ator pode realizar algum processamento novamente. O ator fica neste estado indefinidamente até que ele seja destruído, transitando para o estado Morto. Neste estado, o ator está se preparando para ser destruído tomando as medidas necessárias antes disso tais como: desregistrar-se de qualquer lugar que tenha se incrito para *callback*, sair da lista de atores do Aplique, finalizar a sua linha de execução.

Um objeto Ator é uma instância do padrão de projeto *Observer*. Em cada uma das transições do seu ciclo de vida, eventos são gerados e os observadores cadastrados no ator são notificados. Desta forma é oferecido ganchos para subsistemas realizarem ações dependendo do estado do ator. Por exemplo, o subsistema de estímulos observa o ator e só o deixa receber estímulos de

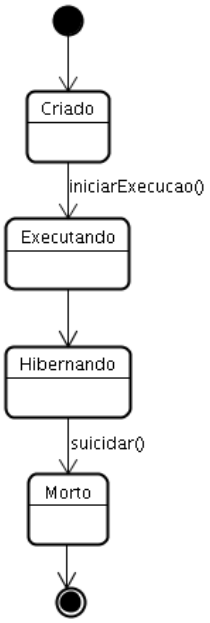

Figura 3.10: Diagrama de estados do ciclo de vida de um Ator.

comunicação direta após ele ter entrado no estado de hibernação.

3.3.3 Máquina de Pilha

A classe Ator mantém as referências para o modelo e oferece mecanismos para o controle de ciclo de vida de um ator, entretanto não é responsável diretamente pela execução. Este trabalho é delegado para a classe MaquinaDePilha. A classe máquina de pilha é quem realmente executa o código.

O ciclo de execução de um código é bastante simples, e é ilustrado no diagrama da figura 3.12. A MaquinaDePilha irá receber uma lista de Palavras para executar. Com a lista em mãos, tudo que a MaquinaDePilha faz é iterar sobre esta lista e tomar as medidas cabíveis para cada um dos elementos. Existem três possíveis caminhos para um dado elemento. Caso o elemento a ser executado seja uma ListaDeComandos, a máquina de pilha cria um novo Closure e o empilha. Caso o elemento seja uma simbolo Resolvivel, isto é, implemente a interface Resolvivel, o símbolo será submetido a uma pesquisa em busca de um Executavel (instância do padrão Command) e então o Executavel será executado. Caso nenhum dos casos acima ocorra, o elemento é empilhado. As palavras que implementam a interface Resolvível são:

- Simbolo
- Operador

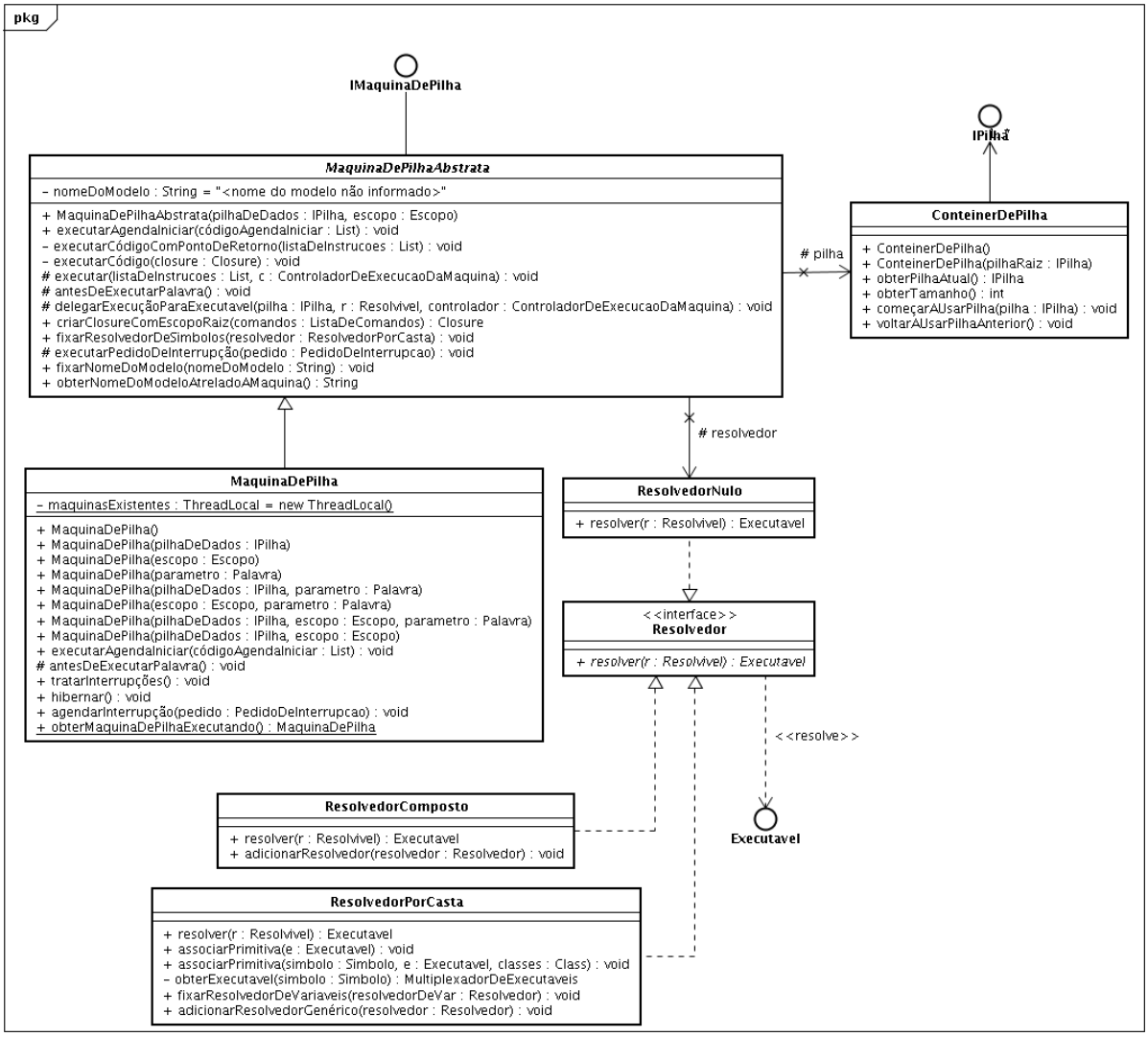

Figura 3.11: Diagrama de classes da MaquinaDePilha e seus colaboradores.

- OperadorComunicacaoDireta
- OperadorEmpilhamento

A separação da busca de um Executavel para um Resolvivel do algoritmo de execução é o que traz a flexibilidade ao interpretador permitindo que primitivas diferentes sejam mapeadas para ações diferentes dependendo do ambiente de execução. Quando a MaquinaDePilha encontra uma Palavra que implementa a interface Resolvivel, ela delega para um Resolvedor. O Resolvedor é um interface contendo um único método que recebe como parametro um Resolvivel e retornando um Executavel. Caso nenhum Executavel seja encontrado é retornado *null*, e a MaquinaDePilha gera um erro em tempo de execução caso contrário o método executar do Executavel retornado é invocado.

A MaquinaDePilha ao ser criada possui um ResolvedorNulo. Objetos nulos são um padrão

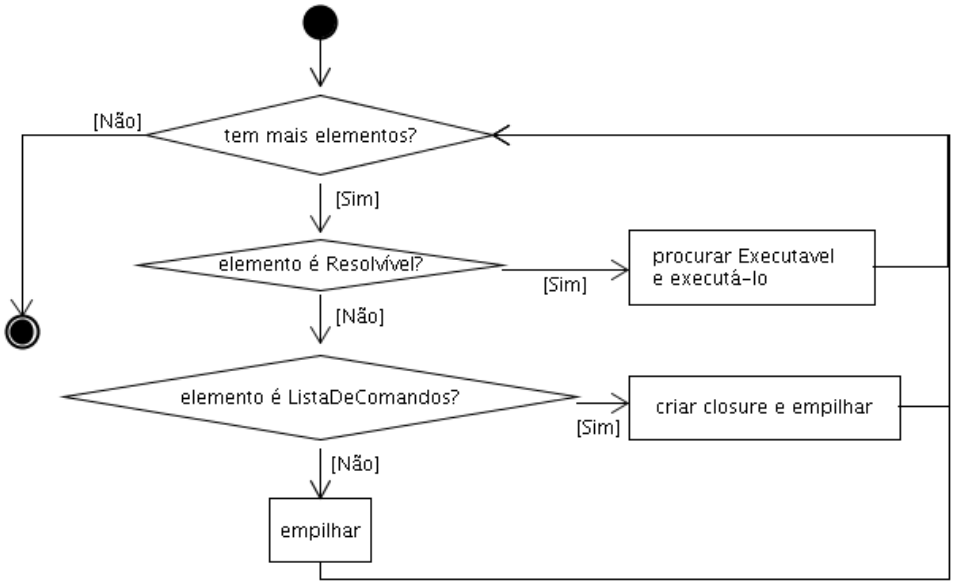

Figura 3.12: Diagrama da execução na MaquinaDePilha.

de projeto, apesar de não estarem entre os definidos por Gamma et al. (1995), e fornecem um comportamento padrão quando um objeto de determinado tipo não existe, substituindo assim o *null*. O ResolvedorNulo não é capaz de realizar nenhuma transformação de Resolvivel em Executavel, portanto a MaquinaDePilha irá gerar erros de execução para todo e qualquer Resolvivel que for encontrado. O resolvedor pode ser alterado utilizando-se IoC, sigla para *Inversion of control.* Neste estilo de programação os colaboradores de um objeto são injetados no mesmo por alguém responsável por criar e gerenciar as dependencias entre estes objetos. No caso da máquina de pilha, quem é responsável por gerenciar estas dependências é a classe Ator. Ela cria, configura e injeta os colaboradores da MaquinaDePilha.

Para garantir flexibilidade e tornar simples e configurável o Resolvedor, que é quem realmente conhece as primitivas e como executá-las, utilizou-se o padrão *Composite.* A classe que implementa o padrão composite é chamada de ResolvedorComposto. O padrão de projeto Composite cria uma interface única para coleções de objetos e objetos propriamente dito. No caso do Resolvedor, este comportamento se torna bastante útil pois a MaquinaDePilha não precisa de lógica adicional para tratar com coleções de Resolvedores, e torna os resolvedores extremamente flexíveis pois cada subsistema pode criar um Resolvedor próprio contendo apenas os seus próprios mapeamentos e uma entidade separada ser responsável por coletar todos os resolvedores desejados. E é isto mesmo que é feito. Cada subsistema fornece um resolvedor (que por sua vez pode ser um outro Composite) e um objeto separado configura um ResolvedorComposto contendo referencia para todos os outros resolvedores. Este último objeto implementa a interface InstaladorDeModulos que deve ser implementado pelos ambientes de execução, ele é

armazenado junto à classe Aplique e é passado para as instancias de Ator que a utilizam para configurar um resolvedor composto antes de injetá-lo na MaquinaDePilha.

Uma instância do padrão Composite, como o caso do ResolvedorComposto, traz uma série de benifícios mas traz uma definiência muito importante para o interpretador: a ordem. A ordem em que os resolvedores filhos serão invocados para a busca de um Executavel não é definida pelo padrão. Isto é um problema pois caso dois subsistemas definam dois executaveis diferentes para um mesmo resolvivel, cria-se um problema. A linguagem deve fornecer um esquema de prioridades para resolver este tipo de questão. Por este motivo, criou-se uma versão um pouco alterada do ResolvedorComposto, chamado de ResolvedorPorCastas. A idéia por trás do ResolvedorPorCastas é criar um Composite que leve em consideração os tipos essenciais de resolvedores estipulando um ordem entre eles. Desta forma, o problema de prioridades é resolvido.

3.3.4 Escopo variáveis

O escopo de variáveis é a área de código na qual as váriaveis declaradas são visíveis. No interpretador, foi criada uma classe Escopo. A classe Escopo possui um mapeamento entre nomes de variáveis e valores. Além disso, esta classe é uma instância do padrão de projeto *Chain of Responsability*. Possuindo assim uma referência a um segundo Escopo que é utilizado se o escopo atual não possuir a variável declarada. Neste caso, a ação é delegada ao segundo escopo. Com isso se torna fácil implementar cadeias de escopos. Por exemplo, um Ator possui váriaveis de instâncias que são visíveis em todo código do ator. Ao realizar a chamada a uma agenda qualquer, é criado um novo escopo e o escopo do ator é colocado no *Chain of Responsability*. Desta forma as variáveis do ator continuarão a ser visíveis na agenda mas a agenda ainda poderá referenciar as variáveis do escopo do ator.

O escopo de variáveis esta atrelado à chamadas de agendas. Toda máquina de pilha possui atrelada a si um GerenteDeEscopo. Toda vez que uma agenda é invocada, o gerente de escopo cria um novo escopo, e sempre que a agenda retorna, o escopo anterior é restaurado.

Quem fará uma chamada de agenda é o subsistema que cuida de agendas, modelos e moldes. Mas a máquina é quem deve oferecer a infra-estrutura básica para que este subsistema possa controlar a máquina de pilha fazendo com que ela altere seu fluxo de execução. Para isso é utilizada uma classe interna a máquina de pilha chamada ControladorDaMaquinaDePilha. Ela é classe interna privada, mas implementa uma interface pública e é passada como parametro para o Executavel. A classe ControladorDaMaquinaDePilha oferece métodos para a execução de uma agenda, quando este método é invocado o ControladorDaMaquinaDePilha irá criar um

novo escopo com o escopo do ator como sendo o próximo escopo na cadeia de responsabilidade, como explicado anteriormente. Então é realizada uma chamada à MaquinaDePilha para que ela execute o código da agenda, como o escopo atual foi trocado no GerenteDeEscopo, a agenda executará no novo escopo. Após terminar a execução da agenda, o fluxo de execução retorna ao ControladorDaMaquinaDePilha e o escopo que estava sendo utilizado antes da agenda ser invocada é restaurado e a MaquinaDePilha irá continuar a executar do ponto onde havia parado.

3.3.5 Criação e execução de closures

Closures são uma lista de comandos mais as variáveis e primitivas disponíveis no escopo no qual ele foi declarado. Desta forma, variáveis e primitivas encontradas em uma lista de comandos continuam sendo acessíveis mesmo depois que o escopo onde o closure foi definido tenha sido destruído.

Para a implementação de closures, foi criada no interpretador a classe Closure. A classe Closure é uma classe contendo referências para uma ListaDeComandos, que define os comandos a serem executados, para uma MaquinaDePilha, indicando o local onde foi declarado e para o escopo onde o Closure foi criado. O Closure é criado no momento em qua a MaquinaDePilha encontra uma ListaDeComandos, neste momento um Closure é criado e empilhado.

Closures podem ser executados posteriormente a sua declaração pela primitiva executar. Esta primitiva envia uma mensagem ao ControladorDeMaquinaDePilha informando que se deseja executar o closure. Neste momento, a MaquinaDePilha troca o escopo que ela está utilizando para o escopo referênciado pelo Closure, executa a ListaDeComandos atrelado ao closure e em seguida restaura o escopo que estav sendo utilizado anteriormente. O diagrama de classes da figura 3.13 mostra o Closure, a interface ControladorDeExecucaoDaMaquina e sua implementação Controlador. Como pode-se ver, o Controlador possui os métodos executar(Closure), para realizar a execuçao de um closure, e o método avaliarExpressão(Closure, Palavras) que permite que um closure seja avaliada e o que permanecer no topo da sua pilha irá ser retornado. Importante ressaltar que avaliarExpressão não modifica a pilha que a MaquinaDePilha está utilizando, uma nova pilha é criada e as palavras passadas como parametros empilhadas nesta pilha temporária. Após o termino de execução do closure a pilha que estava sendo utilizada é restaurada e o fluxo de execução retomado.

Existem outras primitivas que podem implicar na execução de um closure como as primitivas de decisão (seVerdade, seFalso) e iteração (enquantoVerdadeiro, vezesRepetir). Outra utilidade importante do closure é que ele é utilizado para definir o que deve ser feito em resposta a um determinado evento, isto é, definir as ações a serem tomadas mediante a chegada de um

evento.

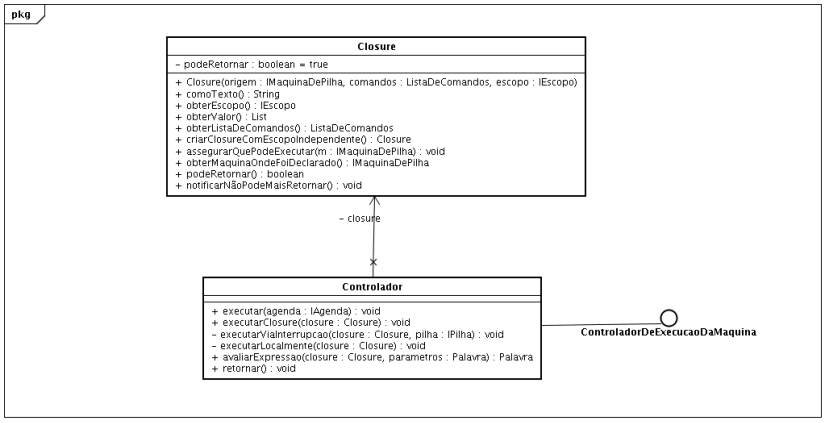

Figura 3.13: Diagrama mostrando Closure e sua relação com o ControladorDeExecucaoDaMaquina.

3.3.6 Classe Programa

A classe Programa é responsável por armazenar e fornecer um ponto de acesso a estruturas que descrevam modelos e moldes definidos pelos usuários. Esta classe é uma instância do padrão *Singleton*, da mesma forma como Aplique. Todos os modelos que podem ser instânciados e moldes que podem ser referênciados durante a execução do programa deve estar armazenado aqui.

A classe Programa descreve estaticamente a estrutura do programa, e está intimamente relacionada a classe Aplique. O subsistema da classe Programa é responsável por fornecer primitivas de criação de modelos, moldes e agendas. Além disso, deve trabalhar em conjunto com a classe Aplique para permitir a instânciação de Modelos previamente criados. O diagrama de classes da figura 3.14 mostra a classe Programa e seus métodos.

3.3.7 Modelo, moldes e herança

Como visto anteriormente, o programa armazena os modelos e moldes e fornece um ponto de acesso global aos mesmo através do padrão de projeto *Singleton*. Os moldes e modelos por sua vez são um conjunto de agendas e lista de moldes que o moldam. Modelos e moldes são representadas pelas classes ModeloDeAtor e Molde respectivamente. O diagrama de classes da figura 3.15 mostra as classes ModeloDeAtor e Moldes e sua hierarquia de classes.

A classe Molde herda de ConteinerAbstrato, responsável por implementar funções básicas comuns a moldes e modelos, como manutenção dos moldes e agendas, com métodos para

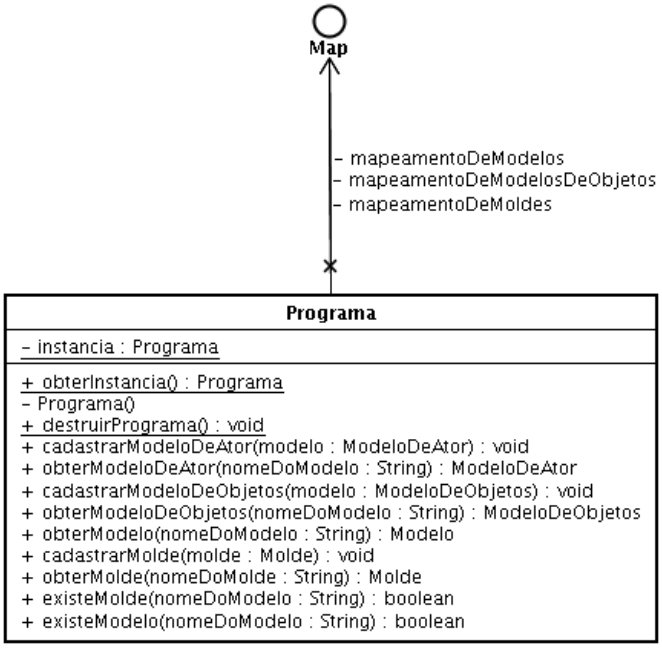

Figura 3.14: Diagrama da classe Programa e seus métodos.

adicionar, remover e acessar as estes elementos. Além disso, é implementada nesta classe o algoritmo para o cálculo do ordenamento topológico sobre o grafo de herança, que define a ordem em que as agendas herdadas serão invocadas.

Para percorrer o grafo de herança na ordem em que as agendas serão invocadas é utilizado o padrão de projeto *Visitor*. O padrão *Visitor* é utilizado para separar o algoritmo utilizado para percorrer uma estrutura de dados do algoritmo que operará sobre os dados contidos nesta estrutura. A principal vantagem da utilização deste padrão é que se torna simples adicionar ou trocar os algoritmos que irão operar sobre os dados. Outro padrão que normalmente é utilizado nestes casos é o padrão *Iterator*. A principal diferença entre os dois padrões é que é responsabilidade de quem utiliza o *Iterator* verificar se há mais dados e pegá-los do *Iterator*. No caso do *Visitor*, quem faz isso é o própria classe da estruturas de dados. Logo o padrão *Visitor* é melhor, neste caso, pois encapsula mais detalhes do que o *Iterator* tornando o código do usuário mais simples o que o iterator.

Quando uma agenda é invocada deve ser calculada todas as agendas que deverão ser invocadas, isto é, as agendas de toda a hierarquia do modelo. Para isto, é utilizado um visitante que percorre a hierarquia de moldes em ordem e cria um objeto da classe AgendaComposta, que é uma instância do padrão de projeto *Composite*. O diagrama de classes pode ser visto no diagrama da figura 3.16. A classe Agenda é, na nomenclatura do padrão *Composite* é um *Leaf* enquanto a classe AgendaComposta é um *Composite* e a interface IAgenda é um *Component*.

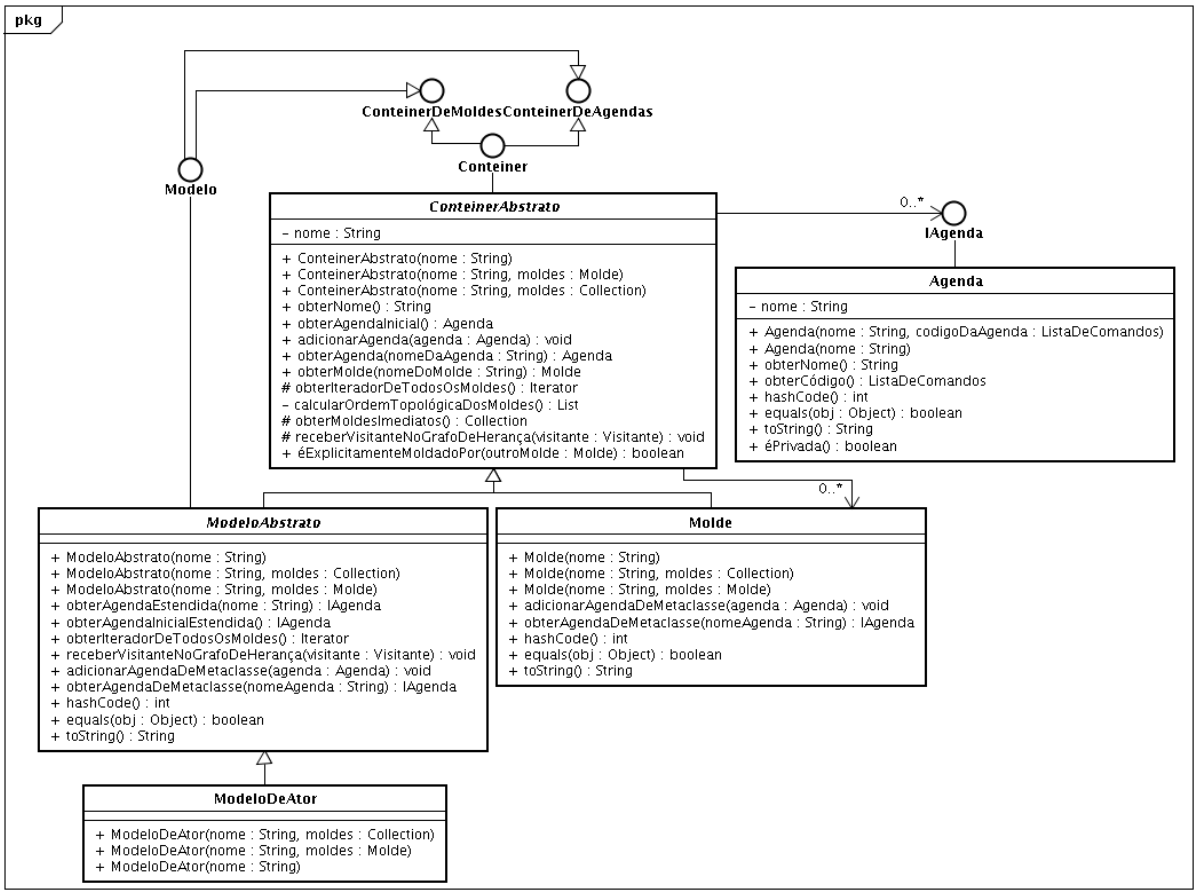

Figura 3.15: Diagrama das classes ModeloDeAtor e Molde.

3.3.8 Estímulos e comunicação direta

Estímulos e comunicação direta são os mecanismos que são oferecidos pela linguagem para permitir que dois ou mais atores se sincronizem, troquem dados, se comuniquem. Assim sendo, são importantíssimos. A diferença entre ambos, como já explanado, é que um é síncrono e direcionado - comunicação direta - enquanto outro é assíncrono e segue um sistema de *broadcast*.

Estes mecanismos foram implementadados por um subsistema de estímulos que fornece um Executavel às primitivas dizer e seDito, além de fornecer um Executavel para o Resolvivel OperadorDeComunicacaoDireta. A classe responsável por armazenar pedidos pendentes de comunicação direta, verificar a disponibilidade de um tratatador e verificar se é possível tratar interrupções (prioridades entre atores) é a classe MaquinaDeEstimulos. As máquinas de estímulos necessitam realizar comunicação entre si, para isto foi utilizado o padrão de projeto *Mediator*. O objetivo do padrão *Mediator* é encapsular a iteração entre objetos em um único objeto. A classe que usa este padrão é o MediadorDeMaquinasDeEstimulos.

A MaquinaDePilha possui um sistema básico para suportar tratamento de estímulos. Diz-

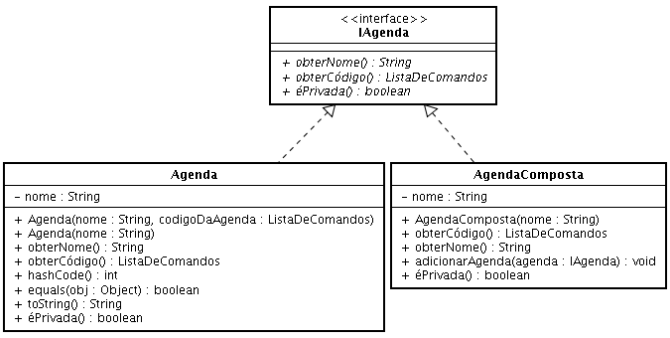

Figura 3.16: Diagrama da hierquia Agenda.

se que uma MaquinaDePilha sofre uma interrupção quando deve parar o que está executando para executar um tratador. A MaquinaDePilha permite o agendamento de uma interrupção e um sistema de *callback* é utilizado para notificar à MaquinaDeEstimulos quando a interrupção finalizou o seu tratamento. Na MaquinaDePilha não há nenhum esquema de prioridade, o estímulo agendado na MaquinaDePilha irá interromper a máquina independente do que a MaquinaDePilha está fazendo no momento. A responsabilidade de implementar prioridades é da MaquinaDeEstímulos.

3.4 Ambientes de execução

Os Ambientes de Execução são responsáveis pela especialização do coração da máquina para cada ambiente específico que a linguagem irá suportar. O coração da máquina fornece ao ambiente de execução vários pontos nos quais o ambiente poderá modificar, contribuir e personalizar, sendo os mais importantes a criação de novas primitivas, alteração do comportamento de primitivas já existentes baseado no conteúdo da pilha. Além disso é possível criar novos tipos de dados, adicionando novas palavras a hierarquia Palavra tornando simples por exemplo que um ambiente gráfico por exemplo crie dados do tipo Imagem e Som que em um ambiente outro ambiente não fazem muito sentido.

Como já mencionando anteriormente o coração da máquina oferece aos ambientes de execução a classe abstrata AmbienteAbstrato que o ambiente de execução deve implementar. A classe AmbienteAbstrato é responsável por realizar a inicialização do coração da máquina, mas deixa alguns métodos abstratos em aberto para o ambiente realizar a personalização da coração da máquina.

Ao iniciar o interpretador, o fluxo de execução vai para o ambiente de execução. Então o ambiente de execução deverá realizar qualquer tipo de inicialização e configuração do ambiente

de execução e em seguida iniciar o coração da máquina. A inicialização do coração da máquina é realizado criando-se a classe que estende AmbienteAbstrato do ambiente em questão e invocando o método inicializar, passando um fluxo de caracteres com o código fonte. A obtenção deste fluxo é dependente da plataforma de execução.

Foram implementados dois ambientes de execução. O ambiente console e ambiente gráfico. O ambiente console oferece primitivas para leitura e escrita no console e não há nenhuma dependência com ambiente gráficos. O ambiente gráfico oferece algumas primitivas de desenho, sendo estas primitivas baseadas nas idéia de *TurtleGraphics*. Neste ambiente, cada ator se comporta como um pintor e responde a primitivas típicas de TurtleGraphics para movimentos relativos para movimentar o ator para frente, giros entre outros.

4 Definição da Linguagem Compilada

Linguagens compilada são aquelas na qual há um passo intermediário entre o código fonte e a execução, este passo é justamente a geração de um arquivo intermediário em alguma outra linguagem - desde linguagem de máquina até linguagens de alto nível - que poderá ser executada.

A sintaxe desta linguagem é fortemente inspirada em Smalltalk e na idéia de linguagens orientada a objetos puras, isto é, tudo é objeto incluindo as classes e tipos como números inteiros e caracteres que em outras linguagens não são. A sintaxe de Smalltalk é muito simples e praticamente não há exceções na linguagem, além disso a legibilidade dela é muito grande por isso foi escolhida como base para esta linguagem.

As principais características da linguagem interpretada estarão presentes na linguagem compilada, como por exemplo *closure*, atores, estímulos, comunicação direta, entre outras.

4.1 Sintaxe da linguagem

A gramática da linguagem compilada, como dita anteriormente, é baseada em Smalltalk. A gramática é bastante simples, a exemplo de Smalltalk, quando comparada a outras linguagens utilizadas atualmente como Ruby, Java ou Python. A gramática EBNF é mostrada na listagem 4.1.

```
Agora ::= Sentencas <EOF>
Sentencas ::= ((Retorno | Expressao | DeclaracaoDeModeloOuMolde) ".")*
DeclaracaoDeModeloOuMolde ::= <PALAVRA_CHAVE> Simbolo (<PALAVRA_CHAVE>
    Lista)? ListaDeAgendas
ListaDeAgendas ::= "[" (DeclaracaoUnaria | DeclaracaoBinaria |
                        DeclaracaoPorPalavraChave)* "]"
DeclaracaoUnaria ::= Identificador "[" Sentencas "]"
DeclaracaoBinaria ::= Seletor Identificador "[" Sentencas "]"
DeclaracaoPorPalavraChave ::= (PalavraChave Identificador)+ "[" Sentencas
    "]"
```

```
Retorno ::= "^" Expressao
Expressao ::= Atribuicao | ExpressaoBasica
Atribuicao ::= Identificador ":=" Expressao
ExpressaoBasica ::= PrimitivaOuExpressao ( Mensagem MensagemEmCascata )?
PrimitivaOuExpressao ::= Primitiva | "(" Expressao ")"
Primitiva ::= Identificador
                | Numero
                | String
                | Booleano
                | Simbolo
                | Bloco
                | Lista
Bloco ::= "[" Parametros Sentencas "]"
Parametros ::= ((":" Identificador)+ "|")?
Lista ::= "#[" (Primitiva)* "]"
Mensagem ::= (MensagemUnaria+ MensagemBinaria* MensagemPorPalavraChave?
                | MensagemBinaria+ MensagemPorPalavraChave?
                | MensagemPorPalavraChave
MensagemUnaria ::= Identificador
MensagemBinaria ::= Seletor PrimitivaOuExpressao MensagemUnaria*
MensagemPorPalavraChave ::= (PalavraChave Argumento)+
Argumento ::= PrimitivaOuExpressao MensagemUnaria* MensagemBinaria*
MensagemEmCascata ::= (";" Mensagem)*
Identificador ::= <IDENTIFICADOR>
Numero ::= <NUMERO>
Booleano ::= <BOOLEANO>
String ::= <STRING>
Simbolo ::= "#" <IDENTIFICADOR>
Seletor ::= <SELETOR>
PalavraChave ::= <PALAVRA_CHAVE>
```

Listagem 4.1: Gramática EBNF da linguagem compilada

Os valores entre parênteses angulares são elementos provenientes da análise léxica e definidos através de expressões regulares, e estão descritos na listagem 4.2.

```
<COMENTARIO_UMA_LINHA> = "//" (~["\n","\r"])* ("\n"|"\r"|"\r\n")
<COMENTARIO_VARIAS_LINHAS> = "/*" (~["*"])* "*" (~["/"] (~["*"])* "*")* "/"
<STRING> = "\"" ( ~["\\","\n","\r","\""] | ("\\" ["n","t","\\","\""]) )*
    "\""
<NUMERO> = ("+" | "-")? (<FLOAT> | <FLOAT> ( ["e","E"] (["-","+"])? <
    INTEIRO> )?)
<PALAVRA_CHAVE> = <IDENTIFICADOR> ":"
<FLOAT> = <INTEIRO> | <INTEIRO> ( ["."] <INTEIRO> )?
```

```
<INTEIRO> = ["0" - "9"]+
<SIMBOLO> = ( <LETRA> | "_" )+ ( <DIGITO> | <LETRA> | "_")*
<LETRA> = ["a"-"z", "A"-"Z"] | "á" | "à" | "â" | "ä" | "ã" |
                  "Á" | "À" | "Â" | "Ä" | "Ã" | "é" | "è" | "ê" | "ë" |
                  "É" | "È" | "Ê" | "Ë" | "í" | "ì" | "î" | "ï" |
                  "Í" | "Ì" | "Î" | "Ï" | "ó" | "ò" | "ô" | "ö" | "õ" |
                  "Ó" | "Ò" | "Ô" | "Ö" | "Õ" | "ú" | "ù" | "û" | "ü" |
                  "Ú" | "Ù" | "Û" | "Ü" | "ç" | "Ç"
<SELETOR>: ("-" | "," | "+" | "/" | "\\" | "*" | "~" | "<" | ">" | "=" |
                  "@" | "%" | "&" | "?" | "!")
```

Listagem 4.2: Definição léxica da linguagem compilada

Como é possível observar, um programa nesta linguagem é um conjunto de setenças sendo que uma sentença é ou uma atribuição de variável ou envio de uma mensagem ou declaração de modelo/molde. O operador de atribuição é ":=".

O terminador de sentenças nesta linguagem será o ponto final (".”). O ponto final deverá ser colocado ao final de toda a sentença escrita nesta linguagem. Outro operador interessante é o operador ponto-e-vírgula. O operador ponto e vírgula possui o intuito de enviar diversas mensagens a um mesmo objeto. Este operador é utilizado após uma mensagem ser enviada para que a próxima mensagem opere sobre o mesmo objeto que havia recebido a mensagem anterior.

O que é interessante notar na gramática é a precedência dos seletores, que em outras linguagens são chamados de operadores. Nota-se que não há uma precedência, tudo é avaliado da esquerda para a direita. O motivo disso ficará mais claro quando a semântica da linguagem for explicada.

4.2 Semântica da linguagem compilada

O objetivo da linguagem compilada é que ela seja semanticamente simples, com o mínimo de exceções possível. Na linguagem basicamente o que há são trocas de mensagens. Um ator envia mensagens a outros atores e objetos para que a computação se realize.

Ao encontrar um elemento sintático descrevendo um tipo de dado, como por exemplo uma *String*, um número ou uma lista, o compilador irá gerar código para a criação de um modelo de objeto previamente definido. Este modelo de objeto é como qualquer outro e possui agendas previamente definidas, sendo portanto capaz de responder as mensagens à ele enviado.

As variáveis não precisam ser previamente declaradas. De forma análoga a linguagem

alvo, as variáveis são criadas automaticamente e são sempre locais, com exceção das variáveis declaradas na agenda iniciar de um modelo ou molde. Estas últimas serão variáveis de instância do modelo. É possível utilizar encademeamento de atribuições, como mostrado na listagem 4.3. Como tudo na linguagems atribuições também retornam um valor, que é o valor que acabara de ser atribuido a ele. Assim sendo, todas as variáveis em uma atribuição em cascata irão receber o mesmo valor.

```
umNumero := numeroVinte := 20.
```

Listagem 4.3: Exemplo de atribuição encadeada

Existem três tipos de mensagens diferentes como é possível ver na gramática EBNF da listagem 4.1, são elas:

- Mensagem unária – é utilizado para enviar uma mensagem à um destino sem realizar passagem de parâmetros. É a mensagem mais prioritária.
- Mensagem binária – é utilizado para enviar uma mensagem à um destino passando um único parâmetro utilizando-se os seletores (operadores, em outras linguagens).
- Mensagem por palavra chave – é utilizado para enviar uma mensagem à um destino passando no mínimo um parâmetro.

Há pequenas diferenças sintáticas entre cada uma dessas mensagens. Na listagem 4.4 é mostrado um exemplo contendo todos os tipos de mensagens.

Na primeira linha do exemplo um objeto do modelo Texto é criado e enviado a mensagem comoNúmero, que é uma mensagem unária. O resultado desta mensagem é armazenado na variável numeroVinte. Nota-se que o nome desta mensagem é composto somente por uma seqüência de letras, isto é, um *token SIMBOLO* descrito na listagem 4.2.

Na segunda linha do exemplo é enviada a mensagem "+" com o objeto que está armazenado na variável *numeroVinte* e o objeto "10" é enviado como parâmetro. O valor retornado pelo método será armazenado na variável "somatorio". Este exemplo é da mensagem binária primeiro é colocado o objeto que irá receber a mensagem, assim como em todos os tipos de mensagens, sendo seguido pelo seletor, um *token SELETOR* descrito na listagem 4.2 e por fim o parâmetro.

Na terceira linha do exemplo é enviada a mensagem "mostrar:" com o objeto que está armazenado na variável *devo* e o objeto armazenado na variável "somatorio" é enviado como parâmetro. Este exemplo é da mensagem por palavra chave, primeiro é colocado o objeto que irá receber a mensagem seguido pelo primeiro fragmento do nome do método (até o dois pontos)

seguido pelo parametro, sendo seguido pelo segundo fragmento do nome do método, seguido por outro parametro e assim por diante. O número de parâmetros é determinado pelo número de fragmentos. Para cada fragmento há um parâmetro. A palavra-chave *devo* é utilizada para dizer que o próprio ator deve receber a mensagem, isto é, é uma referência para o próprio ator que está executado.

```
numeroVinte := "20" comoNúmero.
somatorio := numeroVinte + 10.
devo mostrar: somatorio.
```

Listagem 4.4: Exemplo de mensagens

É interessante notar que estruturas de controle de fluxo pode ser explicado pelo ótica de mensagens e troca de mensagens. No fragmento de código listagem 4.5 é mostrado um exemplo de estrutura de escolha, no qual é realizado um teste para verificar a igualdade entre dois valores e mostrar um valor qualquer em cada um dos casos. A comparação entre umNumero e outroNumero não é nada mais do que a mensagem binária "=" enviada à umNumero. Ao valor retornado por esta mensagem é enviado a mensagem "seVerdade:seFalso:". Espera-se, por convenção, que o valor retornado pela mensagem "=" seja um valor booleano, ou instância do modelo Verdadeiro ou do modelo Falso. Por polimorfismo então a escolha é feita. Caso seja uma instância de Verdadeiro, o primeiro parâmetro é avaliado. Caso seja uma instância de Falso, o segundo parâmetro é avaliado. Resumindo, é possível implementar estrutura de decisão baseando-se somente as ferramentas oferecidas pela linguagem, ou seja, mensagens e closures.

Closures, como explanado na definição da linguagem alvo, são fragmentos de código atrelados ao contexto no qual fora criado. Closures na linguagem compilada possuem comportamento idêntico a closures na linguagem interpretada, mas a sintaxe é diferente. Ao invés da utilização de chaves para denotar o ínicio e fim da declaração de um closure, utiliza-se colchetes. Assim como na linguagem alvo, closures são elementos de primeira classe, podendo por tanto serem passados como parâmetros para outras agendas, serem retornados de agendas, resumindo é um objeto como outro qualquer. Pelas mesmas razões já expostas na descrição da linguagem alvo, closures criados em um ator só podem ser executados pelo próprio para se evitar condições de corrida.

```
umNumero := 10.
outroNumero := 20.

umNumero = outroNumero
        seVerdade: [ devo mostrar: "é igual". ]
        seFalso: [ devo mostrar: "é diferente". ].
```

Listagem 4.5: Exemplo com estrutura de decisão

A concorrência na linguagem compilada utilizará a semântica de atores da linguagem alvo. Todas as características dos atores e as formas de comunicação serão mantidas. A troca de mensagem entre atores na linguagem compilada irá utilizar a mesma semântica da comunicação direta da linguagem interpretada. Todas as catacterísticas, vantagens e desvantagens discutidas anteriormente irão continuar a existir nesta linguagem.

Assim como na linguagem interpretada atores irão rodar simultaneamente de forma independente, sendo que quando criados eles executam a agenda de nome *iniciar*.

A nomenclatura da linguagem original em respeito a orientação a objetos será mantida. Logo, a linguagem compilada também possuirá os conceitos de modelo, molde e herança extamente da mesma forma que a linguagem interpretada. A sintaxe será adaptada para refletir as características da nova linguagem. O fragmento de código da listagem 4.6 mostra como ocorre a declaração de um molde, com nome UmMolde, e um modelo, com nome UmModelo, que é modelado pelo molde UmMolde. Para se criar um molde basicamente se escreve “molde:”, seguido pelo nome e por uma lista com as definições das agendas. O nome é precedido por cerquilha. A cerquilha denota um símbolo, cuja semântica é idêntica a semântica da linguagem alvo. Para se instanciar um ator, ao invés de “molde:” utiliza-se “modelo”. Ambas as declarações podem possuir, opcionalmente a lista de moldes que o moldam, colocando-se logo após o nome a palavra-chave “moldadoPor:” seguida por uma lista de dados, isto é, uma lista precedida por cerquilha contendo os nomes dos moldes.

Na listagem 4.6, vê-se também como ocorre a declaração das agendas que respondem a cada um dos três tipos de mensagens. A primeira, e mais simples, é a agenda da mensagem unária, exemplificada pela agenda iniciar, que é declarada simplesmente escrevendo-se o nome da agenda seguida pela lista contendo a declaração da agenda. Após temos a agenda que respondem a mensagens por palavra chave, exemplificada pela mensagem “somar: valor”. As mensagens por palavra chave são declaradas colocando-se as partes do nome da agenda entrelaçadas com os nomes dos parâmetros. No caso da agenda “somar:”, valor é o nome do parâmetro formal da agenda. Pode-se declarar agendas com maior número de parâmetros. Por exemplo, pode-se criar uma agenda com a assinatura: “somar: valor com: valor2”, neste caso o nome da agenda é “somar:com:” e os parâmetros formais são valor e valor2. Por último, a declaração de uma agenda que responde a agendas binárias, exemplificado pela agenda “+”. Essas agendas são criadas colocando-se o seletor, no caso “+”, seguido do nome do parâmetro formal.

```
molde: #UmMolde
```

```
[
        iniciar
        [
                var := 10.
        ]

        somar: valor
        [
                var := var + valor.
        ]
].

modelo: #MeuModelo moldadoPor: #[UmMolde]
[
        iniciar
        [
                devo mostrar: "ok".
        ]
        + algo
        [
                ^ var + algo.
        ]
].
xpto := MeuModelo novo.
xpto somar: 10.
devo mostrar: xpto + 20;
        suicidar.
```

Listagem 4.6: Exemplo Modelo e Molde

As últimas duas linhas do exemplo mostrado na listagem 4.6 mostram um exemplo do uso do operador ponto-e-vírgula. Primeiramente, a mensagem "mostrar:" é enviada ao próprio objeto (através da palavra chave devo). Em seguida, o operador é utilizado para que a próxima mensagem, que no caso é "suicidar", seja enviada para o mesmo objeto que a mensagem anterior, isto é, o próprio objeto.

Um outro aspecto importante é que nesta linguagem tudo retorna algum valor. Caso uma agenda não possua nenhum retorno, automaticamente será retornado uma referência ao objeto ao qual a agenda pertence. No caso de closures, o valor retornado pela avaliação do mesmo é igual ao valor da última expressão do closure.

Linguagem orientada à domínio ou DSL (*Domain Specific Language*) é uma abordagem

antiga em desenvolvimento de software que está ganhando muita atenção. DSLs são basicamente linguagens específicas particularizadas para um determinado domínio de aplicação. Como exemplo de uma DSL pode-se citar JavaCC, compiler-compiler aqui utilizado. Existem dois tipos de DSLs, as chamadas externas são escritas em uma linguagem diferente da linguagem de programação principal utilizada para desenvolver o programa, as internas são escritas na mesma linguagem de programação. Existem diversas vantagens e desvantagens em cada um destes tipos de DSLs.

Devido a sua sintaxe, esta linguagem favorece muito o desenvolvimento de DSLs internas. Como DSLs internas são limitadas pela linguagem e devem ser sintaticamente e semanticamente válidas, quanto menor o impacto causado pela sintaxe melhor será para desenvolver DSL. A ausência de parênteses para a passagem de parâmetros, o entrelaçamento do nome do método e parâmetros, a ausência de pontos para realizar a troca de mensagens entre objetos. Tudo isso faz com que a estrutura de uma sentença se assemelhe muito a uma frase escrita em uma linguagem natural, facilitando assim o desenvolvimento destes tipos de linguagens.

5 *Implementação do compilador*

Como opção de implementação, optou-se por tornar o compilador o mais simples possível sendo a maior parte da complexidade referente à semântica da linguagem implementada pelo interpretador. O sistema de herança, concorrência, comunicação entre atores, entre outros detalhes, são todos implementados pelo interpretador.

O compilador é divido em duas partes básicas: o sistema de runtime e o tradutor. É responsabilidade do tradutor ler o código fonte e gerar código na linguagem alvo semanticamente equivalente. O sistema de runtime é um sistema de apoio escrito na linguagem alvo que o código gerado pelo tradutor depende. Estas funcionalidades são modelos, atores, classes de biblioteca que são expostas ao programador entre outras coisas que necessitam ser escritas na linguagem alvo.

Quando o compilador é executado lhe é fornecido o arquivo no qual o código fonte que se deseja compilar está escrito e o arquivo no qual a saída do compilador deverá ser escrita. Então, como ilustrado esquematicamente na figura 5.1, o compilador lê o código fonte e o envia ao parser, responsável por realizar a análise léxica e sintática gerando uma Abstract Syntax Tree (AST) representando hierarquicamente a entrada. A AST gerada é então processada, sendo realizada aqui a analise semântica (que é muito limitada, não é realizada quase nenhuma analise semântica) e a geração de código. Por último o código gerado é mesclado com o código do sistema de runtime e escrito no arquivo de saída.

5.1 Parser

O parser do compilador não foi escrito manualmente, à exemplo do que foi feito no interpretador. Foi utilizado também o compiler-compiler JavaCC para gerar o parser. Entretanto, não foi utilizado apenas o JavaCC, foi utilizado uma ferramenta que acompanha a distribuição do JavaCC chamada JJTree.

O JJTree é uma ferramenta com um nível de abstração mais alto quando comparada ao

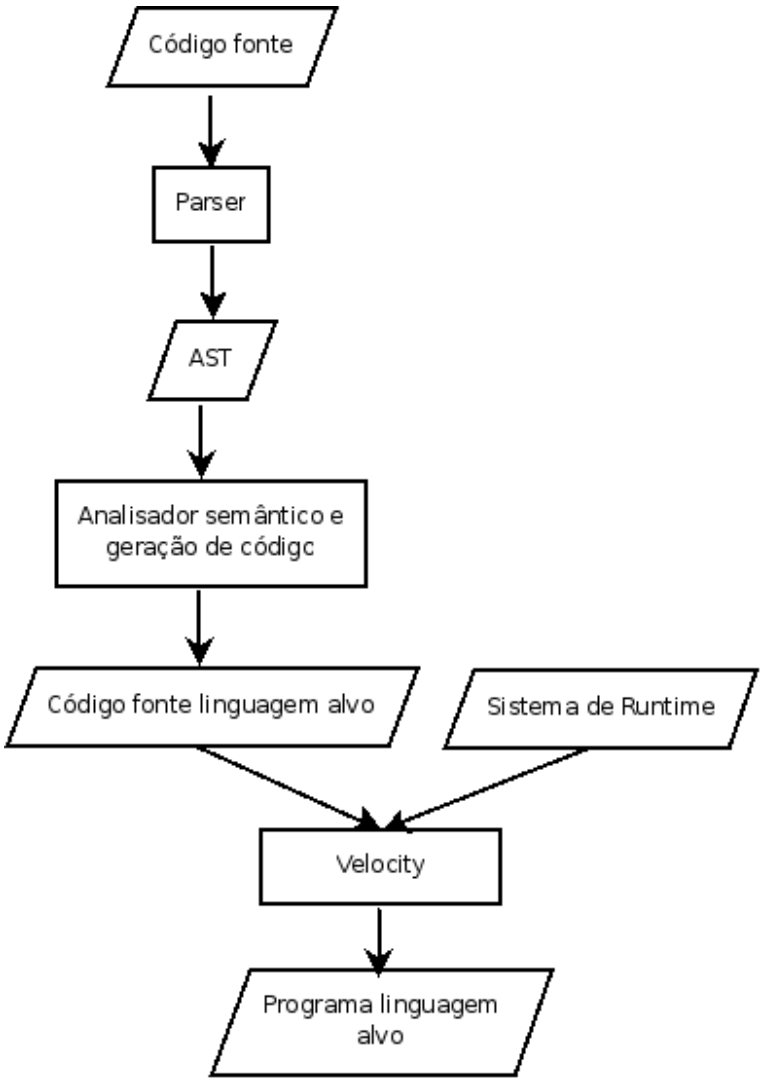

Figura 5.1: Diagrama dos passos do compilador até geração do programa alvo.

JavaCC. Com JavaCC, é possível descrever o parser através de uma sintaxe semelhante a EBNF, com uma extensão para descrever ações semânticas a serem tomadas. O JJTree é um pré-processador para o JavaCC que adiciona mais uma extensão a gramática original do JavaCC na qual é possível descrever a forma da AST que se quer gerar, sendo o código responsável pela montagem da AST gerado automaticamente. Além disto, uma classe é gerada para cada tipo de nó diferente dentro da árvore.

As classes geradas pelo JJTree são muito semelhantes as que foram geradas pelo JavaCC para o interpretador, e por este motivo será omitida. A principal diferença entre as duas é que o JJTree além de gerar o parser, gera uma interface Node e uma classe que implementa Node para cada produção que irá gerar um nó na AST.

5.2 Geração de código

Após o *parsing* estar completo e a AST ter sido completamente gerada o compilador começa a fase de análise semântica e geração de código. Como dito anteriormente, a análise semântica é bastante limitada sendo que poucas verificações e testes são realizados, como é comum em linguagens dinamicamente tipadas.

Toda a análise e geração de código é realizada utilizando-se o padrão de projeto Visitor. O padrão de projeto Visitor é muito utilizado para percorrer uma estrutura de dados, no caso uma árvore, tornando possível separar a forma como se percorre os dados do algoritmo. Além disto, visitor simula um sistema de double-dispatching em uma linguagem single-dispatching. Uma linguagem que utiliza single-dispatching permite que vários métodos possuam mesmo nome e difiram quanto aos parâmetros, mas a escolha de qual o método que será invocado é decidido em tempo de compilação. Em linguagens que utilizam double-dispatching o método que será invocado será decidido em tempo de execução, baseando-se nos tipos em tempo de execução.

Foram criados vários visitantes, cada um responsável pela análise e geração do código em um determinado contexto. Isto ocorre porque um mesmo nodo, dependendo do local no qual é utilizado, pode requerer ações semânticas e/ou geração de códigos diferentes. Quando se está processando o nodo e sabe-se que o próximo nodo exigirá um novo visitante, um é criado e a troca de visitante é realizada.

O tradutor, isto é, a parte do compilador responsável pela geração de código é subdivida em três partes: parser, os visitantes e a descrição de código. Como já explanado, o parser irá gerar uma AST que irá ser processada pelos visitantes. Os visitantes então realizam a análise semântica, preenchem estruturas de dados com informações necessárias para o seu próprio processamento e informam que algum código deve ser gerado. Esta notificação é feita ao subsistema responsável pela descrição do código em linguagem alvo. Este subsistema recebe notificações e armazena o código que deverá ser escrito em memória serializando-o no fim do processo de tradução.

Após o processo de geração ter sido concluído, o compilador pegará o código fonte equivalente escrito na linguagem alvo e o mesclará ao código do sistema de runtime. Isto é realizado utilizando um sistema de templates chamado Velocity. Um sistema de template é caracterizado por processar um documento escrito em uma linguagem própria definido pelo sistema de template e gerar uma saída alterando váriaveis e dados do template por valores fornecidos pelo usuário. O sistema de runtime é escrito em forma de template, demarcando como e onde o código gerado deverá ser inserido.

6 Conclusão

Como resultado deste trabalho de conclusão de curso foi especificada uma linguagem de programação orientada a objetos, dinamicamente tipada, sintática e semanticamente simples, naturalmente concorrente. Esta linguagem herdou a sintaxe limpa de *Smalltalk* e apresentou conceitos como herança múltipla e polimorfismo. Sempre levando em consideração e tentando incluir características mais interessantes das diversas linguagens estudadas.

Além da especificação da linguagem, foi criado um compilador que gera código em uma linguagem intermediária. A linguagem intermediária também foi especificada e um interpretador criado para a mesma. A linguagem é baseada em pilha, e possui forte influencia sintática e semântica da linguagem de programação Telis.

Referências Bibliográficas

ARMSTRONG, J. The development of erlang. *SIGPLAN Not.*, ACM Press, New York, NY, USA, v. 32, n. 8, p. 196–203, 1997. ISSN 0362-1340.

ARMSTRONG, J. Concurrency oriented programming in erlang. Swedish Institute of Computer Science, 2003.

BRODIE, L. *Starting FORTH*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986. ISBN 0-13-843087-X.

GAMMA, E. et al. *Design Patterns*. Boston: Addison-Wesley, 1995. ISBN 0201633612.

Apêndice A – Artigo

Projeto e implementação de uma linguagem de programação

Giovani Pieri

Resumo. *Grande parte das linguagens de programação atuais são linguagens seqüenciais, o que torna a descrição de algoritmos e programas paralelos, difícil e propenso a erros. Com a adoção de arquiteturas multi-núcleo com freqüências mais baixas, o software terá que expor paralelismo de forma explicita para utilizar eficientemente o hardware. Então será desenvolvida uma linguagem de programação que visa estimular o uso de concorrência de forma natural, simples e de forma mais segura possível.*

1. Introdução

Grande parte das linguagens de programação atuais são linguagens seqüenciais, o que torna a descrição de algoritmos e programas paralelos, difícil e propenso a erros. Com a adoção de arquiteturas multi-núcleo com freqüências mais baixas, o software terá que expor paralelismo de forma explicita para utilizar eficientemente o hardware. Então será desenvolvida uma linguagem de programação que visa estimular o uso de concorrência de forma natural, simples e de forma mais segura possível.

Esta linguagem apresentará uma série de elementos que visam o aumento do poder de expressividade por parte do programador e, principalmente, mecanismos que incentivem a criação de programas paralelos. Deseja-se que a linguagem possua o maior poder de expressividade possível devido a DSL (*Domain Specific Languages*). Um dos objetivos da linguagem é facilitar a utilização e desenvolvimento de DSLs internas. DSLs internas são aquelas definidas dentro das regras sintáticas e semânticas da própria linguagem na qual a DSL foi escrita, expondo ao utilizador da DSL a possibilidade de utilizar todos os recurso da linguagem pai. O outro tipo de DSL é chamada de DSL externa, na qual não existe nenhuma relação entre a linguagem em que a DSL foi desenvolvida e a própria DSL, isto é, uma linguagem totalmente nova é criada.

A sintaxe da linguagem é fortemente influênciada por Smalltalk e na idéia de linguagens orientada a objetos puras, nas quais tudo é objeto incluindo as classes e tipos como números inteiros e caracteres que em outras linguagens não são. A sintaxe de Smalltalk é muito simples e praticamente não há exceções na linguagem, além disso a legibilidade dela é muito grande por isso foi escolhida como base para esta linguagem.

2. Modelo de concorrência: Atores

A linguagem será inerentemente concorrente. A concorrência, a exemplo de linguagens como Erlang, será tratada diretamente por mecanismos providos pela linguagem, diferentemente de outras linguagens nas quais são oferecidos mecanismos para a criação de *threads* e a sincronização é realizada através de semáforos e/ou monitores utilizando-se memória compartilhada.

Para isso será utilizada uma abordagem baseada em Atores. Atores são semelhantes a processos no que diz respeito a compartilhamento de recursos. Não há memória compartilhada, conseqüentemente a necessidade de sincronização de áreas de memória compartilhadas é eliminada.

A comunicação entre atores se dá através de troca de mensagens. Existem dois tipos de trocas de mensagem na linguagem. O primeiro é síncrono e o segundo assíncrono. A comunicação síncrona se da de um para um, um ator envia uma mensagem a um segundo ator que processa e retorna um resultado, só então o ator que iniciou a comunicação segue com o seu processamento. A segunda abordagem é assíncrona e se da de um para muitos. Vários atores podem receber uma mensagem e enviada de forma assíncrona e tratá-la, isto é, a mensagem é enviada a todos os outros atores que tenham interesse em recebê-la.

Uma tupla é uma lista e é imutável, e é em tuplas que se baseia o sistema de troca de mensagens, mensagens são tuplas. São necessários dois mecanismos básicos para que atores possam se comunicar de forma assíncrona: eles devem poder enviar mensagens e poder se cadastrar com o intuito de receber uma mensagem. Para enviar uma mensagem assíncrona especifica-se uma tupla. Para demonstar interesse na recepção de uma mensagem assíncrona determina-se um filtro e uma lista de instruções. Um filtro nada mais é do que uma especificação do formato da tupla que o ator está interessado em tratar. Um filtro casa com uma tupla, isto é, é compatível com uma tupla se as condições abaixo forem satisfeitas:

1. Filtro e estímulo possuem mesmo tamanho
2. Para todo elemento do estímulo, o elemento de mesma posição no filtro deve: ser igual ou indicar o tipo do elemento do estímulo.

Quando uma mensagem assíncrona é enviada e um ator recebe esta tupla e possui um comandos a executar o fluxo normal de execução é interrompido, os comandos são executados e a computação volta da onde estava anteriormente.

3. Herança, moldes e princípios de orientação a objetos

A herança nesta linguagem é diferente da implementada em linguagens como Java e C++. O intuito é fornecer um mecanismo simples de ser entendido e explicado, embora seja menos flexível. Isto ocorre porque o ponto onde a superclasse será chamada é pré-definida ao invés de ser indicada pelo programador.

Classes concretas são denominadas modelos. Um modelo quando instânciado gera um novo ator. Um modelo possui diversos métodos, chamados de agendas. Estas agendas podem ser públicas ou privadas, as agendas privadas iniciam com o símbolo "_" e só podem ser invocadas pelos atores instânciados pelo modelo. As agendas públicas podem ser alvo de comunicação síncrona.

Classes abstratas são denominadas moldes. Diz-se que um molde molda um modelo, e este por sua vez é moldado pelo molde. Um modelo pode ser moldado por zero ou mais moldes, como é possível que um modelo seja moldado por mais de um molde há a noção de herança múltipla. Além disso, um molde também pode ser moldado por zero ou mais moldes. Um modelo não pode ser utilizado como molde.

A partir dessa definição de moldado/molde é possível criar um grafo de herança G(V, A) definido da seguinte forma:

$V = \{ x \mid x \text{ é um molde ou modelo} \}$

$A = \{ (x, y) \mid x \text{ molda } y \wedge x \text{ é um molde} \}$

Esse grafo deve possuir as seguintes propriedades:

1. Não deve possuir ciclos.
2. Todos os modelos devem ser nodos folha, isto é, seu grau de emissão[1] é igual a 0.

Os moldes definem agendas, de forma análoga a modelos. Estas agendas podem ser utilizadas por quem for moldado por este molde. Caso um modelo ou molde defina uma agenda com mesmo nome de uma agenda definida em um dos moldes que o está moldando, este irá estender a agenda do molde. Do ponto de vista do programador, o código definido pelo modelo é anexado ao fim das agendas do molde cujos nomes sejam idênticos.

A linguagem é dinamicamente tipada. Devido a isto, nenhum tipo de verificação quanto a existência de uma agenda é realizada em tempo de compilação, sendo a viabilidade de um ator em responder uma determinada mensagem síncrona verificada apenas em tempo de execução. Devido a isto, o polimorfismo é algo natural e inerente. Não é necessário, para utilizar polimorfismo, herança, interfaces ao contrário de linguagens estaticamente tipadas. Para utilizar polimorfismo basta que os atores (e seus modelos) possuam o subconjunto das agendas utilizadas definidas. Ruby, Python e outras linguagens dinamicamente tipadas possuem o mesmo comportamento. Este tipo de polimorfismo é denominado *duck typing*. Este termo tem origens na frase *"If it quacks like a duck, walks like a duck, it must be a duck!"*. A origem deste termo é desconhecida mas suspeita-se que foi cunhada por Alex Martelli em uma mensagem ao *newsgroup* comp.lang.python.

4. Troca de mensagens

A sintaxe da linguagem, como já foi dito, foi fortemente inspirado em Smalltalk e deste herda a sintaxe para o envio de mensagens. Diferentemente da maioria das linguagens de programação que utiliza um ponto para denotar que uma mensagem deve ser enviada a um determinado objeto anteriormente referenciado, esta seguirá Smalltalk, na qual um objeto é especificado e nenhum símbolo especial deve ser colocado antes do nome da mensagem a ser executada, apenas um espaço.

Existem três tipos distintos de troca de mensagens. O primeiro é mensagem unária e corresponde a mensagens que não possuem nenhum parâmetro. O nome da mensagem é composta apenas por uma seqüência alfa-numérica de caracteres. Em seguida temos as mensagens binárias que correspondem a mensagens que possuem apenas um parâmetro e o nome é um operador. Por fim temos as mensagens por palavra-chave que possuem um ou mais parâmetros. O nome deste último tipo de mensagem é composto por uma ou mais seqüências alfa-numérica de caracteres seguida por dois pontos. Os dois pontos indicam onde os parâmetros serão colocados. A exemplo de Smalltalk os parâmetros são entrelaçados ao nome do método.

O operador ponto e vírgula é responsável pelo cascateamento de mensagens, isto é, a mensagem colocada após o ponto e vírgula será enviada ao objeto que recebeu a última mensagem enviada.

Assim como Smalltalk, não há precedencia entre operadores, como soma e multiplicação por exemplo, os operadores são mensagens e todas as mensagens são avaliadas da direita para a esquerda.

[1] O grau de emissão de um vértice v corresponde ao número de arestas que partem de v.

4.1. Closures

Blocos de comandos são fragmentos de códigos cuja execução não se dá imediatamente, mas é postergada a algum momento futuro. Na lingaugem, estes blocos são tratados como elementos de primeira classe, isto é, podem ser passadas como parâmetros, retornadas como resultado de uma mensagem, salvo em váriaveis, campos entre outras coisas.

Closures são fragmentos de código que mantém uma referência ao ambiente no qual foram criados. Desta forma, o retorno de um método não necessariamente implica na destruição das variáveis locais deste já que um closure pode estar referenciando estas variáveis locais. Isto torna a pilha de execução não mais uma pilha, mas sim uma árvore.

Closures são muito utilizados em linguagens como Ruby, Python, Lisp, entre outras, e dão uma ferramenta muito expressiva ao programador. Há linguagens nas quais não existem closures como Java e C/C++. Há quem confunda as classes anônimas de Java com closures, entretanto não são iguais. Classes anônimas, apesar de poderem ser utilizadas para algumas coisas da mesma forma que closures, não são closures completos pois não mantém uma cópia para o contexto no qual fora criada. Java permite apenas que variáveis finais do escopo no qual a classe foi criada sejam acessadas, permitindo que uma cópia da variável externa seja feita para a classe anônima.

A introdução de closures entretanto acarretam um problema ao modelo de concorrência dos atores. O modelo de atores, por definição, não possuem memória compartilhada sendo toda a comunicação entre eles realizada via troca de mensagens. Closures carregam consigo o contexto no qual foram definidos, logo se um closure é definido em um ator A e posteriormente é enviado a um ator B e lá executado, o ator B passa a executar operações concorrentemente com o ator A na área de memória deste, causando uma condição de corrida (*race condition*).

Para evitar que este problema ocorra, há uma restrição em quem pode executar os closures. Apenas o ator que criou um closure poderá executá-lo. Desta forma, a condição de corrida será evitada, em contrapartida o potencial de utilização de closures será um pouco reduzido.

5. Ambientes de execução

A linguagem que será desenvolvida deverá poder rodar em ambientes distintos, com peculiaridades diferentes. Um ambiente de execução é o local no qual o aplique rodará. A princípio serão definidos três ambientes básicos, mas não são limitados apenas a estes:

- Aplicação textual (console)
- Aplicação gráfica baseada em turtle graphics
- Aplicação servidor

Um programa deverá ser capaz de rodar em um destes três ambientes. Entretanto há diferenças muito grandes com relação ao que deverá ser oferecido ao programador em cada um destes ambientes. Em ambientes gráficos deve-se oferecer formas de manipular diferentes objetos gráficos, coisa que não deverá existir em aplicações textuais e de servidor.

Linguagens como Java resolvem este tipo de problema, fazendo uma linguagem absolutamente neutra e entrega APIs (*Application programmers interface*) e *frameworks*

que encapsulam as peculiaridades destes ambientes. Esta é uma abordagem simples, que transfere a responsabilidade da linguagem para a biblioteca desta mesma linguagem. Entretanto, Telis assim como a nova linguagem tem como principal objetivo ser simples. Toda a funcionalidade inerente a um dado ambiente deve estar disponível de forma simples sem a necessidade de utilizar frameworks ou APIs complexas. Telis resolveu o problema se limitando a um ambiente apenas, o gráfico.

A deverá ser capaz de rodar independentemente do ambiente. Note que não foi mencionado que um mesmo aplique deverá ser capaz de rodar em mais de um ambiente simultaneamente.

O paradigma de orientação a objeto possui um princípio chamado princípio de separação de preocupações (*separation of concerns*). Este princípio diz uma classe deve possuir uma e apenas uma responsabilidade, quando uma classe começa a ter mais responsabilidades do que deveria, então ela deve ser dividida em outras classes.

Aplicando este princípio, podemos dizer que a nova linguagem na verdade será composta por um framework comum e especializada para cada um destes diferentes ambientes de execução. Quando o núcleo da linguagem é especializado para um determinado ambiente de execução, agendas primitivas especializadas poderão ser incluídas, modelos/moldes definidos, a inicialização personalizada, integração com o ambiente em questão realizada e qualquer outra ação que se achar necessária.

O inconveniente desta abordagem é a existência de vários dialetos da linguagem, desta forma um aplique escrito em um ambiente não rodará em outro e o programador deverá estar ciente disto e escolher o dialeto apropriado para o ambiente de execução desejado. Entretanto, em linguagens comuns, programas que funcionam em vários ambientes de execução sem alterações são raros, gerando erros em ambientes não suportados.

6. Implementação

Para implementar esta linguagem decidiu-se utilizar uma abordagem híbrida entre compilação e interpretação. A linguagem é, em um primeiro passo, compilada para uma segunda linguagem mais simples e em um segundo momento esta linguagem mais simples é submetida a um interpretador que a executa. Como opção de implementação, optou-se por tornar o compilador o mais simples possível sendo a maior parte da complexidade referente à semântica da linguagem implementada pelo interpretador. O sistema de herança, concorrência, comunicação entre atores, entre outros detalhes, são todos implementados pelo interpretador.

O compilador é divido em duas partes básicas: o sistema de runtime e o tradutor. É responsabilidade do tradutor ler o código fonte e gerar código na linguagem alvo semanticamente equivalente. O sistema de runtime é um sistema de apoio escrito na linguagem alvo que o código gerado pelo tradutor depende. Estas funcionalidades são modelos, atores, classes de biblioteca que são expostas ao programador entre outras coisas que necessitam ser escritas na linguagem alvo.

Quando o compilador é executado lhe é fornecido o arquivo no qual o código fonte que se deseja compilar está escrito e o arquivo no qual a saída do compilador deverá ser escrita. Então, como ilustrado esquematicamente na figura 1, o compilador lê o código fonte e o envia ao parser, responsável por realizar a análise léxica e sintática gerando uma

Abstract Syntax Tree (AST) representando hierarquicamente a entrada. A AST gerada é então processada, sendo realizada aqui a analise semântica (que é muito limitada, não é realizada quase nenhuma analise semântica) e a geração de código. Por último o código gerado é mesclado com o código do sistema de runtime e escrito no arquivo de saída.

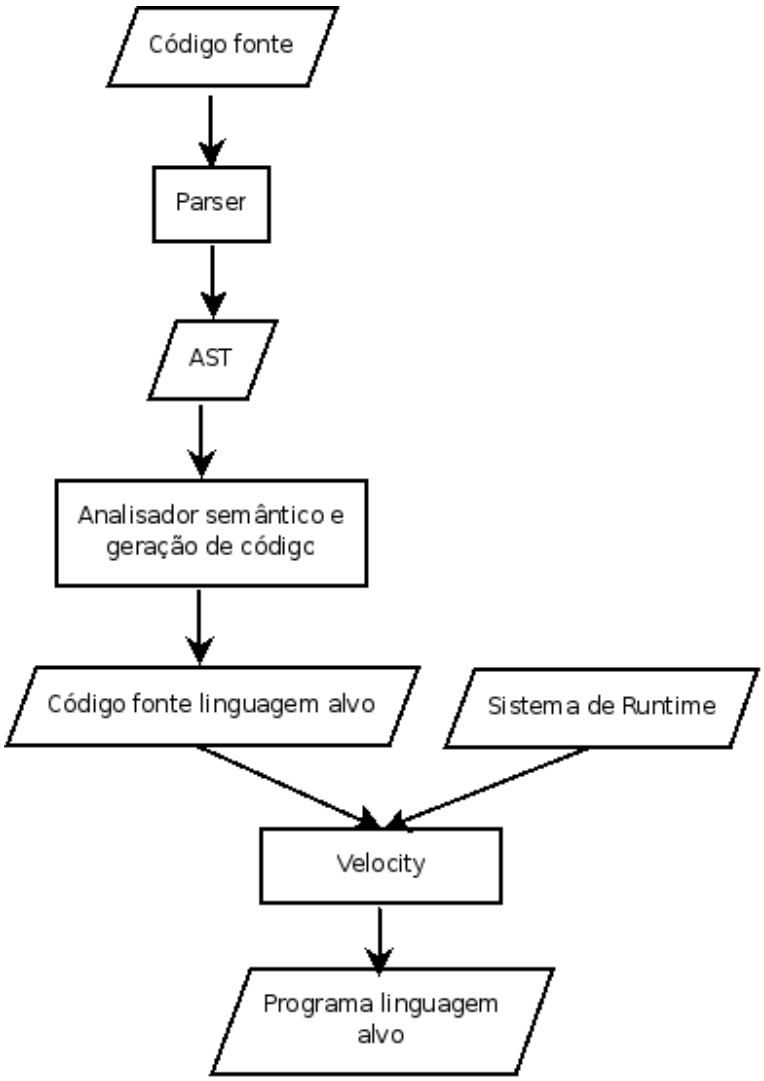

Figura 1. Diagrama dos passos do compilador até geração do programa alvo.

7. Conclusão

Como resultado deste trabalho foi especificada uma linguagem de programação orientada a objetos, dinamicamente tipada, sintática e semanticamente simples, naturalmente concorrente. Esta linguagem herdou a sintaxe limpa de *Smalltalk* e apresentou conceitos como herança múltipla e polimorfismo. Sempre levando em consideração e tentando incluir as características mais interessantes das diversas linguagens estudadas.

Além da especificação da linguagem, foi criado um compilador que gera código em uma linguagem intermediária. A linguagem intermediária também foi especificada e um interpretador criado para a mesma. A linguagem é baseada em pilha, e possui forte influencia sintática e semântica da linguagem de programação Telis.

Apêndice B – Codigo fonte

Classe: br.ufsc.edugraf.agora.excecoes.ExcecaoAgora

```
package br.ufsc.edugraf.agora.excecoes;

public class ExcecaoAgora extends Exception {

  private static final long serialVersionUID = 8116798043123075856L;

  public ExcecaoAgora() {
    super();
  }

  public ExcecaoAgora(String message, Throwable cause) {
    super(message, cause);
  }

  public ExcecaoAgora(String message) {
    super(message);
  }

  public ExcecaoAgora(Throwable cause) {
    super(cause);
  }

}
```

Classe: br.ufsc.edugraf.agora.excecoes.ExcecaoTemplateMalFormado

```
package br.ufsc.edugraf.agora.excecoes;

public class ExcecaoTemplateMalFormado extends
    ExcecaoAgora {

  public ExcecaoTemplateMalFormado() {
    super();
  }

  public ExcecaoTemplateMalFormado(String message) {
    super(message);
  }

}
```

Classe: br.ufsc.edugraf.agora.publicacao.Publicador

```
package br.ufsc.edugraf.agora.publicacao;

import java.io.Writer;

import org.apache.velocity.Template;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.app.Velocity;

public class Publicador {

  private static final String CAMINHO_DO_TEMPLATE = "br/ufsc/edugraf/agora/publicac

  private final String código;
  private final Writer saída;

  public Publicador(String código, Writer saída) {
    this.código = código;
    this.saída = saída;
```

```
}

public void publicar() {
  try {
    VelocityContext contexto = new VelocityContext();
    contexto.put("codigo", código);
    Template template = Velocity.getTemplate(
        CAMINHO_DO_TEMPLATE, "UTF8");
    template.merge(contexto, saída);
    saída.flush();
  } catch (Exception e) {
    e.printStackTrace(); // TODO: tratar erro
  }
}

static {
  Velocity.setProperty("resource.loader",
      "class");
  Velocity.setProperty("resource.loader",
      "class");
  Velocity.setProperty(
      "class.resource.loader.description",
      "Velocity Classpath Resource Loader");
  Velocity
      .setProperty(
          "class.resource.loader.class",
          "org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader");

  try {
    Velocity.init();
  } catch (Exception e) {
    System.out
        .println("Não foi possível inicializar velocity.");
  }
}
```

```
}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Instrucao

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;

public class Instrucao implements ProdutorDeCodigo {

  private final String texto;

  public Instrucao(String texto) {
    this.texto = texto;
  }

  public void gerarCódigo(StringBuilder codigo) {
    codigo.append(texto + "\n");
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ContainerDeComandos

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;

public interface ContainerDeComandos extends
    ProdutorDeCodigo {

  public abstract ContainerDeComandos empilharLista(
      Lista lista);

  public abstract ContainerDeComandos empilharBloco(
      Bloco bloco);

  public abstract ContainerDeComandos chamarAgenda(
```

```
        String nomeDaAgenda);

    public abstract ContainerDeComandos simboloExecutável(
        String simbolo);

    public abstract ContainerDeComandos associarVariável(
        String variável);

    public abstract ContainerDeComandos empilharNúmero(
        String numero);

    public abstract ContainerDeComandos empilharTexto(
        String texto);

    public abstract void empilharSimbolo(
        String obterNomeDoModelo);

    public abstract ContainerDeComandos instanciarModelo(
        String nomeDoModelo);

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ProdutorDeCodigo

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;

public interface ProdutorDeCodigo {

    public void gerarCódigo(StringBuilder codigo);

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Bloco

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;
```

```
public class Bloco extends ContainerDeComandosAbstrato {

  public Bloco() {
  }

  public void gerarCódigo(StringBuilder codigo) { // TODO: extrair interface
    codigo.append("{\n");
    for (ProdutorDeCodigo instrucao : listaDeInstrucoes)
      instrucao.gerarCódigo(codigo);
    codigo.append("}\n");
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ProgramaTelis

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;

public class ProgramaTelis extends
    ContainerDeComandosAbstrato {

  public void gerarCódigo(StringBuilder codigo) {
    for (ProdutorDeCodigo instrucao : listaDeInstrucoes)
      instrucao.gerarCódigo(codigo);
  }

  public String gerarCódigo() {
    StringBuilder stringBuilder = new StringBuilder();
    gerarCódigo(stringBuilder);
    return stringBuilder.toString();
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Lista

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;

public class Lista extends ContainerDeComandosAbstrato {

  public Lista() {
  }

  public void gerarCódigo(StringBuilder codigo) { // TODO: extrair interface
    codigo.append("[\n");
    for (ProdutorDeCodigo instrucao : listaDeInstrucoes)
      instrucao.gerarCódigo(codigo);
    codigo.append("]\n");
  }

  public boolean estáVazia() {
    return listaDeInstrucoes.isEmpty();
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ContainerDeComandosA

```
package br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis;

import java.util.ArrayList;
import java.util.List;

public abstract class ContainerDeComandosAbstrato
    implements ContainerDeComandos {

  protected List<ProdutorDeCodigo> listaDeInstrucoes = new ArrayList<ProdutorDeCodi

  public ContainerDeComandos empilharTexto(
```

```
    String texto) {
  listaDeInstrucoes.add(new Instrucao(texto));
  return this;
}

public ContainerDeComandos empilharBloco(
    Bloco bloco) {
  listaDeInstrucoes.add(bloco);
  return this;
}

public ContainerDeComandos empilharNúmero(
    String numero) {
  listaDeInstrucoes.add(new Instrucao(numero));
  return this;
}

public ContainerDeComandos associarVariável(
    String variável) {
  listaDeInstrucoes.add(new Instrucao("$"
      + variável + " associar"));
  return this;
}

public void empilharSimbolo(String simbolo) {
  listaDeInstrucoes.add(new Instrucao("$"
      + simbolo));
}

public ContainerDeComandos simboloExecutável(
    String simbolo) {
  listaDeInstrucoes.add(new Instrucao(simbolo));
  return this;
}
```

```
  public ContainerDeComandos chamarAgenda(
      String nomeDaAgenda) {
    listaDeInstrucoes.add(new Instrucao("."
        + nomeDaAgenda));
    return this;
  }

  public ContainerDeComandos instanciarModelo(
      String nomeDoModelo) {
    listaDeInstrucoes.add(new Instrucao(
        nomeDoModelo + ".novo"));
    return this;
  }

  public ContainerDeComandos empilharLista(
      Lista lista) {
    listaDeInstrucoes.add(lista);
    return this;
  }

  public boolean estáVazia() {
    return listaDeInstrucoes.isEmpty();
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.VisitanteArmazenadorEmContainerDeC

```
package br.ufsc.edugraf.agora.geracaoDeCodigo;

import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ContainerDeComandosAbst
import br.ufsc.edugraf.agora.parser.javacc.ASTAtribuicao;
import br.ufsc.edugraf.agora.parser.javacc.ASTLista;

public class VisitanteArmazenadorEmContainerDeComandos
```

```
    extends VisitanteNulo {

  private final ContainerDeComandosAbstrato comandos;

  public VisitanteArmazenadorEmContainerDeComandos(
      ContainerDeComandosAbstrato comandos) {
    this.comandos = comandos;
  }

  @Override
  public Object visit(ASTAtribuicao node, Object data) {
    return super.visit(node, data);
  }

  @Override
  public Object visit(ASTLista node, Object data) {
    return super.visit(node, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.VisitanteNulo

```
package br.ufsc.edugraf.agora.geracaoDeCodigo;

import br.ufsc.edugraf.agora.parser.javacc.ASTAgora;
import br.ufsc.edugraf.agora.parser.javacc.ASTAtribuicao;
import br.ufsc.edugraf.agora.parser.javacc.ASTBloco;
import br.ufsc.edugraf.agora.parser.javacc.ASTBooleano;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoBinaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoDeModeloOuMolde;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoPorPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoUnaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTExpressao;
import br.ufsc.edugraf.agora.parser.javacc.ASTIdentificador;
import br.ufsc.edugraf.agora.parser.javacc.ASTLista;
```

```
import br.ufsc.edugraf.agora.parser.javacc.ASTListaDeAgendas;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagem;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemBinaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemEmCascata;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemPorPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemUnaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTNumero;
import br.ufsc.edugraf.agora.parser.javacc.ASTPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTParametros;
import br.ufsc.edugraf.agora.parser.javacc.ASTRetorno;
import br.ufsc.edugraf.agora.parser.javacc.ASTSeletor;
import br.ufsc.edugraf.agora.parser.javacc.ASTSentencas;
import br.ufsc.edugraf.agora.parser.javacc.ASTSimbolo;
import br.ufsc.edugraf.agora.parser.javacc.ASTString;
import br.ufsc.edugraf.agora.parser.javacc.AgoraVisitor;
import br.ufsc.edugraf.agora.parser.javacc.SimpleNode;

public class VisitanteNulo implements AgoraVisitor {

  public Object visit(SimpleNode node, Object data) {
    return null;
  }

  public Object visit(ASTAgora node, Object data) {
    return null;
  }

  public Object visit(ASTSentencas node, Object data) {
    return null;
  }

  public Object visit(ASTRetorno node, Object data) {
    return null;
  }
```

```
public Object visit(ASTExpressao node, Object data) {
  return null;
}

public Object visit(ASTAtribuicao node, Object data) {
  return null;
}

public Object visit(ASTBloco node, Object data) {
  return null;
}

public Object visit(ASTParametros node, Object data) {
  return null;
}

public Object visit(ASTLista node, Object data) {
  return null;
}

public Object visit(ASTMensagemUnaria node,
    Object data) {
  return null;
}

public Object visit(ASTMensagemBinaria node,
    Object data) {
  return null;
}

public Object visit(
    ASTMensagemPorPalavraChave node,
    Object data) {
  return null;
}
```

```
public Object visit(ASTMensagemEmCascata node,
    Object data) {
  return null;
}

public Object visit(ASTIdentificador node,
    Object data) {
  return null;
}

public Object visit(ASTNumero node, Object data) {
  return null;
}

public Object visit(ASTString node, Object data) {
  return null;
}

public Object visit(ASTSimbolo node, Object data) {
  return null;
}

public Object visit(ASTSeletor node, Object data) {
  return null;
}

public Object visit(ASTPalavraChave node,
    Object data) {
  return null;
}

public Object visit(ASTBooleano node, Object data) {
  return null;
}
```

```
  public Object visit(
      ASTDeclaracaoDeModeloOuMolde node,
      Object data) {
    return null;
  }

  public Object visit(ASTListaDeAgendas node,
      Object data) {
    return null;
  }

  public Object visit(ASTDeclaracaoUnaria node,
      Object data) {
    return null;
  }

  public Object visit(ASTDeclaracaoBinaria node,
      Object data) {
    return null;
  }

  public Object visit(
      ASTDeclaracaoPorPalavraChave node,
      Object data) {
    return null;
  }

  public Object visit(ASTMensagem node, Object data) {
    return null;
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.VisitanteGeradorDeBlocoDeComandos

```
package br.ufsc.edugraf.agora.geracaoDeCodigo;

import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Bloco;
import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ContainerDeComandos;
import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Lista;
import br.ufsc.edugraf.agora.parser.javacc.ASTAgora;
import br.ufsc.edugraf.agora.parser.javacc.ASTAtribuicao;
import br.ufsc.edugraf.agora.parser.javacc.ASTBloco;
import br.ufsc.edugraf.agora.parser.javacc.ASTBooleano;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoBinaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoDeModeloOuMolde;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoPorPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoUnaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTExpressao;
import br.ufsc.edugraf.agora.parser.javacc.ASTIdentificador;
import br.ufsc.edugraf.agora.parser.javacc.ASTLista;
import br.ufsc.edugraf.agora.parser.javacc.ASTListaDeAgendas;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagem;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemBinaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemEmCascata;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemPorPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemUnaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTNumero;
import br.ufsc.edugraf.agora.parser.javacc.ASTParametros;
import br.ufsc.edugraf.agora.parser.javacc.ASTRetorno;
import br.ufsc.edugraf.agora.parser.javacc.ASTSeletor;
import br.ufsc.edugraf.agora.parser.javacc.ASTSentencas;
import br.ufsc.edugraf.agora.parser.javacc.ASTSimbolo;
import br.ufsc.edugraf.agora.parser.javacc.ASTString;
import br.ufsc.edugraf.agora.parser.javacc.Node;

public class VisitanteGeradorDeBlocoDeComandos extends
    VisitanteNulo {

  private ContainerDeComandos container;
```

```
public VisitanteGeradorDeBlocoDeComandos(
    ContainerDeComandos container) {
  this.container = container;
}

public Object visit(ASTAgora node, Object data) {
  return node.childrenAccept(this, data);
}

@Override
public Object visit(ASTSentencas node, Object data) {
  for (int i = 0; i < node.jjtGetNumChildren(); i++) {
    Node filho = node.jjtGetChild(i);
    filho.jjtAccept(this, data);
    if (i < node.jjtGetNumChildren() - 1
        && filho instanceof ASTExpressao) {
      container.simboloExecutável("tirar");
    }
  }
  return null;
}

@Override
public Object visit(ASTExpressao node, Object data) {
  if (!node.éMensagem()) {
    node.childrenAccept(this, data);
    return null;
  }

  node.jjtGetChild(0).jjtAccept(this, data);
  for (int i = 0; i < node
      .númeroDeMensagensEncadeadas(); i++)
    container.simboloExecutável("copiar");
```

```
  for (int i = 1; i < node.jjtGetNumChildren(); i++) {
    node.jjtGetChild(i).jjtAccept(this, data);
  }

  return null;
}

@Override
public Object visit(ASTRetorno node, Object data) {
  node.childrenAccept(this, data);
  container.simboloExecutável("retornar");
  return null;
}

@Override
public Object visit(ASTString node, Object data) {
  container.empilharTexto(node.obterValor());
  container.instanciarModelo("Texto");
  return null;
}

@Override
public Object visit(ASTIdentificador node,
    Object data) {
  container.simboloExecutável(node.obterValor());
  return null;
}

@Override
public Object visit(ASTNumero node, Object data) {
  container.empilharNúmero(node.obterValor());
  container.instanciarModelo("Número");
  return null;
}
```

```
@Override
public Object visit(
    ASTMensagemPorPalavraChave node,
    Object data) {
  node.aceitarVisitanteNosParametros(this, null);

  container.empilharNúmero(node
      .obterNúmeroDeParametros());
  container.simboloExecutável("gerarLista");

  container.simboloExecutável("trocar");
  container.chamarAgenda(node
      .obterNomeDoMétodo());
  return null;
}

@Override
public Object visit(ASTMensagemUnaria node,
    Object data) {
  container.empilharLista(new Lista());
  container.simboloExecutável("trocar");
  container.chamarAgenda(node
      .obterNomeDoMétodo());
  return null;
}

@Override
public Object visit(ASTMensagemEmCascata node,
    Object data) {
  container.simboloExecutável("tirar");
  node.childrenAccept(this, null);
  return null;
}

@Override
```

```
public Object visit(ASTAtribuicao node, Object data) {
  node.visitarExpressao(this, data);
  container.simboloExecutável("copiar");
  container.associarVariável(node
      .obterNomeDaVariavel());
  return null;
}

@Override
public Object visit(ASTSeletor node, Object data) {
  container.chamarAgenda(node
      .obterNomeDaAgenda());
  return null;
}

@Override
public Object visit(ASTMensagemBinaria node,
    Object data) {
  node.visitarExpressao(this, data);
  container.simboloExecutável("trocar");
  node.visitarSeletor(this, data);
  return null;
}

@Override
public Object visit(ASTBloco node, Object data) {
  Bloco bloco = new Bloco();
  VisitanteGeradorDeBlocoDeComandos visitante = new VisitanteGeradorDeBlocoDeComa
      bloco);

  node.childrenAccept(visitante, data);

  container.empilharBloco(bloco);
  return super.visit(node, data);
}
```

```
@Override
public Object visit(ASTParametros node, Object data) {
  Lista lista = new Lista();
  node.childrenAccept(
      new VisitanteGeradorDeBlocoDeComandos(
          lista), data);
  container.empilharLista(lista);
  container.simboloExecutável("associar");
  return super.visit(node, data);
}

@Override
public Object visit(ASTLista node,
    Object deveEncapsular) {
  Lista lista = new Lista();
  VisitanteGeradorDeLista visitante = new VisitanteGeradorDeLista(
      lista);

  node.childrenAccept(visitante, deveEncapsular);

  container.empilharLista(lista);
  if (!(deveEncapsular instanceof Boolean)
      || ((Boolean) deveEncapsular)
          .booleanValue() == true)
    container.instanciarModelo("Lista");
  return super.visit(node, deveEncapsular);
}

@Override
public Object visit(ASTBooleano node, Object data) {
  container.simboloExecutável(node.obterValor());
  container.instanciarModelo("Booleano");
  return super.visit(node, data);
}
```

```
@Override
public Object visit(ASTSimbolo node, Object data) {
  container.empilharSimbolo(node.obterValor());
  return super.visit(node, data);
}

@Override
public Object visit(
    ASTDeclaracaoDeModeloOuMolde node,
    Object data) {
  Lista listaDeMoldes = new Lista();
  listaDeMoldes
      .simboloExecutável("MoldePrimitivo");
  if (node.possuiListaDeMoldes())
    node.obterListaDeMoldes().childrenAccept(
        new VisitanteGeradorDeLista(
            listaDeMoldes), null);
  container.empilharLista(listaDeMoldes);
  container.empilharSimbolo(node
      .obterNomeDoModelo());
  node.obterDefinicaoDeAgendas().jjtAccept(this,
      null);
  if (node.éModelo())
    container
        .simboloExecutável("incluirModelo");
  else
    container
        .simboloExecutável("incluirMolde");
  return null;
}

@Override
public Object visit(ASTListaDeAgendas node,
    Object data) {
```

```
  Bloco bloco = new Bloco();
  VisitanteGeradorDeBlocoDeComandos visitante = new VisitanteGeradorDeBlocoDeComa
      bloco);
  for (int i = 0; i < node.jjtGetNumChildren(); i++) {
    node.jjtGetChild(i).jjtAccept(visitante,
        null);
    bloco.simboloExecutável("incluirAgenda");
  }
  container.empilharBloco(bloco);
  return super.visit(node, data);
}

@Override
public Object visit(ASTDeclaracaoBinaria node,
    Object data) {
  container.simboloExecutável(node
      .obterSeletor().obterNomeDaAgenda());
  Bloco bloco = new Bloco();

  bloco.empilharSimbolo(node
      .obterNomeDoParametro());
  bloco.simboloExecutável("associar");

  node.obterDefinicao().jjtAccept(
      new VisitanteGeradorDeBlocoDeComandos(
          bloco), null);

  bloco.simboloExecutável("euMesmo");
  container.empilharBloco(bloco);
  return super.visit(node, data);
}

@Override
public Object visit(
    ASTDeclaracaoPorPalavraChave node,
```

```
    Object data) {
  container.simboloExecutável(node
      .obterNomeDaMensagem());
  Bloco bloco = new Bloco();

  Lista lista = new Lista();
  for (String nome : node
      .obterNomeDosParametros())
    lista.simboloExecutável(nome);
  bloco.empilharLista(lista);
  bloco.simboloExecutável("associar");

  node.obterDefinicao().jjtAccept(
      new VisitanteGeradorDeBlocoDeComandos(
          bloco), null);

  bloco.simboloExecutável("euMesmo");
  container.empilharBloco(bloco);
  return super.visit(node, data);
}

@Override
public Object visit(ASTDeclaracaoUnaria node,
    Object data) {
  container.simboloExecutável(node
      .obterNomeDaMensagem());
  Bloco bloco = new Bloco();
  bloco.empilharLista(new Lista());
  bloco.simboloExecutável("associar");
  node.obterDefinicao().jjtAccept(
      new VisitanteGeradorDeBlocoDeComandos(
          bloco), null);
  bloco.simboloExecutável("euMesmo");
  container.empilharBloco(bloco);
  return super.visit(node, data);
```

```
  }

  @Override
  public Object visit(ASTMensagem node, Object data) {
    node.childrenAccept(this, data);
    return super.visit(node, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.VisitanteGeradorDeLista

```
package br.ufsc.edugraf.agora.geracaoDeCodigo;

import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Bloco;
import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ContainerDeComandos;
import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.Lista;
import br.ufsc.edugraf.agora.parser.javacc.ASTAgora;
import br.ufsc.edugraf.agora.parser.javacc.ASTAtribuicao;
import br.ufsc.edugraf.agora.parser.javacc.ASTBloco;
import br.ufsc.edugraf.agora.parser.javacc.ASTBooleano;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoBinaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoDeModeloOuMolde;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoPorPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoUnaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTExpressao;
import br.ufsc.edugraf.agora.parser.javacc.ASTIdentificador;
import br.ufsc.edugraf.agora.parser.javacc.ASTLista;
import br.ufsc.edugraf.agora.parser.javacc.ASTListaDeAgendas;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagem;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemBinaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemEmCascata;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemPorPalavraChave;
import br.ufsc.edugraf.agora.parser.javacc.ASTMensagemUnaria;
import br.ufsc.edugraf.agora.parser.javacc.ASTNumero;
import br.ufsc.edugraf.agora.parser.javacc.ASTPalavraChave;
```

```
import br.ufsc.edugraf.agora.parser.javacc.ASTParametros;
import br.ufsc.edugraf.agora.parser.javacc.ASTRetorno;
import br.ufsc.edugraf.agora.parser.javacc.ASTSeletor;
import br.ufsc.edugraf.agora.parser.javacc.ASTSentencas;
import br.ufsc.edugraf.agora.parser.javacc.ASTSimbolo;
import br.ufsc.edugraf.agora.parser.javacc.ASTString;
import br.ufsc.edugraf.agora.parser.javacc.AgoraVisitor;
import br.ufsc.edugraf.agora.parser.javacc.SimpleNode;

public class VisitanteGeradorDeLista implements
    AgoraVisitor {

  private ContainerDeComandos container;

  public VisitanteGeradorDeLista(Lista container) {
    this.container = container;
  }

  public Object visit(ASTBloco node, Object data) {
    Bloco bloco = new Bloco();
    node.childrenAccept(
        new VisitanteGeradorDeBlocoDeComandos(
            bloco), null);
    container.empilharBloco(bloco);
    throw new UnsupportedOperationException(
        "Erro semantico.");
  }

  public Object visit(ASTLista node, Object data) {
    Lista lista = new Lista();
    node.childrenAccept(
        new VisitanteGeradorDeBlocoDeComandos(
            lista), null);
    container.empilharLista(lista);
    return null;
```

```
}

public Object visit(ASTIdentificador node,
    Object data) {
  container.simboloExecutável(node.obterValor());
  return null;
}

public Object visit(ASTNumero node, Object data) {
  container.empilharNúmero(node.obterValor());
  return null;
}

public Object visit(ASTBooleano node, Object data) {
  container.simboloExecutável(node.obterValor());
  return null;
}

public Object visit(ASTString node, Object data) {
  container.empilharTexto(node.obterValor());
  return null;
}

public Object visit(ASTSimbolo node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTSeletor node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTPalavraChave node,
    Object data) {
```

```
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTParametros node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTMensagemUnaria node,
    Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTMensagemBinaria node,
    Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(
    ASTMensagemPorPalavraChave node,
    Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTMensagemEmCascata node,
    Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(SimpleNode node, Object data) {
```

```
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTAgora node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTSentencas node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTRetorno node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTExpressao node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(ASTAtribuicao node, Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}

public Object visit(
    ASTDeclaracaoDeModeloOuMolde node,
    Object data) {
  throw new UnsupportedOperationException(
      "Erro semantico.");
}
```

```
  public Object visit(ASTListaDeAgendas node,
      Object data) {
    throw new UnsupportedOperationException(
        "Erro semantico.");
  }

  public Object visit(ASTDeclaracaoUnaria node,
      Object data) {
    throw new UnsupportedOperationException(
        "Erro semantico.");
  }

  public Object visit(ASTDeclaracaoBinaria node,
      Object data) {
    throw new UnsupportedOperationException(
        "Erro semantico.");
  }

  public Object visit(
      ASTDeclaracaoPorPalavraChave node,
      Object data) {
    throw new UnsupportedOperationException(
        "Erro semantico.");
  }

  public Object visit(ASTMensagem node, Object data) {
    node.childrenAccept(this, data);
    return null;
  }

}
```

Classe: br.ufsc.edugraf.agora.geracaoDeCodigo.FachadaDaGeracaoDeCodigo

```
package br.ufsc.edugraf.agora.geracaoDeCodigo;

import java.io.IOException;
import java.io.Writer;

import br.ufsc.edugraf.agora.geracaoDeCodigo.descricaoTelis.ProgramaTelis;
import br.ufsc.edugraf.agora.parser.javacc.ASTAgora;

public class FachadaDaGeracaoDeCodigo {

  public void gerarCodigo(ASTAgora ast, Writer saída) {
    ProgramaTelis programa;
    programa = new ProgramaTelis();

    VisitanteGeradorDeBlocoDeComandos visitanteGerador = new VisitanteGeradorDeBloc
        programa);
    ast.jjtAccept(visitanteGerador, null);

    try {
      saída.write(programa.gerarCódigo());
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTString

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTString extends ASTPrimitivaAbstrata {
  public ASTString(int id) {
    super(id);
  }
```

```
  public ASTString(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.TokenMgrError

```
package br.ufsc.edugraf.agora.parser.javacc;

public class TokenMgrError extends Error {
  /*
   * Ordinals for various reasons why an Error of this type can be thrown.
   */

  /**
   * Lexical error occurred.
   */
  static final int LEXICAL_ERROR = 0;

  /**
   * An attempt was made to create a second instance of a static token
   * manager.
   */
  static final int STATIC_LEXER_ERROR = 1;

  /**
   * Tried to change to an invalid lexical state.
   */
```

```
static final int INVALID_LEXICAL_STATE = 2;

/**
 * Detected (and bailed out of) an infinite loop in the token manager.
 */
static final int LOOP_DETECTED = 3;

/**
 * Indicates the reason why the exception is thrown. It will have one of the
 * above 4 values.
 */
int errorCode;

/**
 * Replaces unprintable characters by their escaped (or unicode escaped)
 * equivalents in the given string
 */
protected static final String addEscapes(String str) {
  StringBuffer retval = new StringBuffer();
  char ch;
  for (int i = 0; i < str.length(); i++) {
    switch (str.charAt(i)) {
    case 0:
      continue;
    case '\b':
      retval.append("\\b");
      continue;
    case '\t':
      retval.append("\\t");
      continue;
    case '\n':
      retval.append("\\n");
      continue;
    case '\f':
      retval.append("\\f");
```

```
      continue;
    case '\r':
      retval.append("\\r");
      continue;
    case '\"':
      retval.append("\\\"");
      continue;
    case '\'':
      retval.append("\\\'");
      continue;
    case '\\':
      retval.append("\\\\");
      continue;
    default:
      if ((ch = str.charAt(i)) < 0x20
          || ch > 0x7e) {
        String s = "0000"
            + Integer.toString(ch, 16);
        retval
            .append("\\u"
                + s
                    .substring(
                        s
                            .length() - 4,
                        s
                            .length()));
      } else {
        retval.append(ch);
      }
      continue;
    }
  }
  return retval.toString();
}
```

```
/**
 * Returns a detailed message for the Error when it is thrown by the token
 * manager to indicate a lexical error. Parameters : EOFSeen : indicates if
 * EOF caused the lexical error curLexState : lexical state in which this
 * error occurred errorLine : line number when the error occurred
 * errorColumn : column number when the error occurred errorAfter : prefix
 * that was seen before this error occurred curchar : the offending
 * character Note: You can customize the lexical error message by modifying
 * this method.
 */
protected static String LexicalError(
    boolean EOFSeen, int lexState,
    int errorLine, int errorColumn,
    String errorAfter, char curChar) {
  return ("Lexical error at line "
      + errorLine
      + ", column "
      + errorColumn
      + ".  Encountered: "
      + (EOFSeen ? "<EOF> "
          : ("\""
              + addEscapes(String
                  .valueOf(curChar)) + "\"")
              + " ("
              + (int) curChar
              + "), ")
      + "after : \""
      + addEscapes(errorAfter) + "\"");
}

/**
 * You can also modify the body of this method to customize your error
 * messages. For example, cases like LOOP_DETECTED and INVALID_LEXICAL_STATE
 * are not of end-users concern, so you can return something like :
 *
```

```
   * "Internal Error : Please file a bug report .... "
   *
   * from this method for such cases in the release version of your parser.
   */
  public String getMessage() {
    return super.getMessage();
  }

  /*
   * Constructors of various flavors follow.
   */

  public TokenMgrError() {
  }

  public TokenMgrError(String message, int reason) {
    super(message);
    errorCode = reason;
  }

  public TokenMgrError(boolean EOFSeen,
      int lexState, int errorLine,
      int errorColumn, String errorAfter,
      char curChar, int reason) {
    this(LexicalError(EOFSeen, lexState,
        errorLine, errorColumn, errorAfter,
        curChar), reason);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.JJTAgoraState

```
package br.ufsc.edugraf.agora.parser.javacc;

public class JJTAgoraState {
```

```
private java.util.Stack<Node> nodes;
private java.util.Stack<Integer> marks;

private int sp; // number of nodes on stack
private int mk; // current mark
private boolean node_created;

public JJTAgoraState() {
  nodes = new java.util.Stack<Node>();
  marks = new java.util.Stack<Integer>();
  sp = 0;
  mk = 0;
}

/*
 * Determines whether the current node was actually closed and pushed. This
 * should only be called in the final user action of a node scope.
 */
boolean nodeCreated() {
  return node_created;
}

/*
 * Call this to reinitialize the node stack. It is called automatically by
 * the parser's ReInit() method.
 */
void reset() {
  nodes.removeAllElements();
  marks.removeAllElements();
  sp = 0;
  mk = 0;
}

/*
 * Returns the root node of the AST. It only makes sense to call this after
```

```
 * a successful parse.
 */
Node rootNode() {
  return (Node) nodes.elementAt(0);
}

/* Pushes a node on to the stack. */
void pushNode(Node n) {
  nodes.push(n);
  ++sp;
}

/*
 * Returns the node on the top of the stack, and remove it from the stack.
 */
Node popNode() {
  if (--sp < mk) {
    mk = ((Integer) marks.pop()).intValue();
  }
  return (Node) nodes.pop();
}

/* Returns the node currently on the top of the stack. */
Node peekNode() {
  return (Node) nodes.peek();
}

/*
 * Returns the number of children on the stack in the current node scope.
 */
int nodeArity() {
  return sp - mk;
}

void clearNodeScope(Node n) {
```

```
  while (sp > mk) {
    popNode();
  }
  mk = ((Integer) marks.pop()).intValue();
}

void openNodeScope(Node n) {
  marks.push(new Integer(mk));
  mk = sp;
  n.jjtOpen();
}

/*
 * A definite node is constructed from a specified number of children. That
 * number of nodes are popped from the stack and made the children of the
 * definite node. Then the definite node is pushed on to the stack.
 */
void closeNodeScope(Node n, int num) {
  mk = ((Integer) marks.pop()).intValue();
  while (num-- > 0) {
    Node c = popNode();
    c.jjtSetParent(n);
    n.jjtAddChild(c, num);
  }
  n.jjtClose();
  pushNode(n);
  node_created = true;
}

/*
 * A conditional node is constructed if its condition is true. All the nodes
 * that have been pushed since the node was opened are made children of the
 * the conditional node, which is then pushed on to the stack. If the
 * condition is false the node is not constructed and they are left on the
 * stack.
```

```
   */
  void closeNodeScope(Node n, boolean condition) {
    if (condition) {
      int a = nodeArity();
      mk = ((Integer) marks.pop()).intValue();
      while (a-- > 0) {
        Node c = popNode();
        c.jjtSetParent(n);
        n.jjtAddChild(c, a);
      }
      n.jjtClose();
      pushNode(n);
      node_created = true;
    } else {
      mk = ((Integer) marks.pop()).intValue();
      node_created = false;
    }
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTArgumento

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTArgumento extends SimpleNode {
  public ASTArgumento(int id) {
    super(id);
  }

  public ASTArgumento(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
```

```
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ParseException

```
package br.ufsc.edugraf.agora.parser.javacc;

/**
 * This exception is thrown when parse errors are encountered. You can
 * explicitly create objects of this exception type by calling the method
 * generateParseException in the generated parser.
 *
 * You can modify this class to customize your error reporting mechanisms so
 * long as you retain the public fields.
 */
public class ParseException extends Exception {

  /**
   * This constructor is used by the method "generateParseException" in the
   * generated parser. Calling this constructor generates a new object of this
   * type with the fields "currentToken", "expectedTokenSequences", and
   * "tokenImage" set. The boolean flag "specialConstructor" is also set to
   * true to indicate that this constructor was used to create this object.
   * This constructor calls its super class with the empty string to force the
   * "toString" method of parent class "Throwable" to print the error message
   * in the form: ParseException: <result of getMessage>
   */
  public ParseException(Token currentTokenVal,
      int[][] expectedTokenSequencesVal,
      String[] tokenImageVal) {
    super("");
    specialConstructor = true;
    currentToken = currentTokenVal;
```

```
  expectedTokenSequences = expectedTokenSequencesVal;
  tokenImage = tokenImageVal;
}

/**
 * The following constructors are for use by you for whatever purpose you
 * can think of. Constructing the exception in this manner makes the
 * exception behave in the normal way - i.e., as documented in the class
 * "Throwable". The fields "errorToken", "expectedTokenSequences", and
 * "tokenImage" do not contain relevant information. The JavaCC generated
 * code does not use these constructors.
 */

public ParseException() {
  super();
  specialConstructor = false;
}

public ParseException(String message) {
  super(message);
  specialConstructor = false;
}

/**
 * This variable determines which constructor was used to create this object
 * and thereby affects the semantics of the "getMessage" method (see below).
 */
protected boolean specialConstructor;

/**
 * This is the last token that has been consumed successfully. If this
 * object has been created due to a parse error, the token followng this
 * token will (therefore) be the first error token.
 */
public Token currentToken;
```

```
/**
 * Each entry in this array is an array of integers. Each array of integers
 * represents a sequence of tokens (by their ordinal values) that is
 * expected at this point of the parse.
 */
public int[][] expectedTokenSequences;

/**
 * This is a reference to the "tokenImage" array of the generated parser
 * within which the parse error occurred. This array is defined in the
 * generated ...Constants interface.
 */
public String[] tokenImage;

/**
 * This method has the standard behavior when this object has been created
 * using the standard constructors. Otherwise, it uses "currentToken" and
 * "expectedTokenSequences" to generate a parse error message and returns
 * it. If this object has been created due to a parse error, and you do not
 * catch it (it gets thrown from the parser), then this method is called
 * during the printing of the final stack trace, and hence the correct error
 * message gets displayed.
 */
public String getMessage() {
  if (!specialConstructor) {
    return super.getMessage();
  }
  StringBuffer expected = new StringBuffer();
  int maxSize = 0;
  for (int i = 0; i < expectedTokenSequences.length; i++) {
    if (maxSize < expectedTokenSequences[i].length) {
      maxSize = expectedTokenSequences[i].length;
    }
    for (int j = 0; j < expectedTokenSequences[i].length; j++) {
```

```
    expected
        .append(
            tokenImage[expectedTokenSequences[i][j]])
        .append(' ');
  }
  if (expectedTokenSequences[i][expectedTokenSequences[i].length - 1] != 0) {
    expected.append("...");
  }
  expected.append(eol).append("    ");
}
String retval = "Encountered \"";
Token tok = currentToken.next;
for (int i = 0; i < maxSize; i++) {
  if (i != 0)
    retval += " ";
  if (tok.kind == 0) {
    retval += tokenImage[0];
    break;
  }
  retval += add_escapes(tok.image);
  tok = tok.next;
}
retval += "\" at line "
    + currentToken.next.beginLine
    + ", column "
    + currentToken.next.beginColumn;
retval += "." + eol;
if (expectedTokenSequences.length == 1) {
  retval += "Was expecting:" + eol + "    ";
} else {
  retval += "Was expecting one of:" + eol
      + "    ";
}
retval += expected.toString();
return retval;
```

```
}

/**
 * The end of line string for this machine.
 */
protected String eol = System.getProperty(
    "line.separator", "\n");

/**
 * Used to convert raw characters to their escaped version when these raw
 * version cannot be used as part of an ASCII string literal.
 */
protected String add_escapes(String str) {
  StringBuffer retval = new StringBuffer();
  char ch;
  for (int i = 0; i < str.length(); i++) {
    switch (str.charAt(i)) {
    case 0:
      continue;
    case '\b':
      retval.append("\\b");
      continue;
    case '\t':
      retval.append("\\t");
      continue;
    case '\n':
      retval.append("\\n");
      continue;
    case '\f':
      retval.append("\\f");
      continue;
    case '\r':
      retval.append("\\r");
      continue;
    case '\"':
```

```
        retval.append("\\\"");
        continue;
      case '\'':
        retval.append("\\\'");
        continue;
      case '\\':
        retval.append("\\\\");
        continue;
      default:
        if ((ch = str.charAt(i)) < 0x20
            || ch > 0x7e) {
          String s = "0000"
              + Integer.toString(ch, 16);
          retval
              .append("\\u"
                  + s
                      .substring(
                          s
                              .length() - 4,
                          s
                              .length()));
        } else {
          retval.append(ch);
        }
        continue;
      }
    }
    return retval.toString();
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTIdentificador

```
package br.ufsc.edugraf.agora.parser.javacc;
```

```
public class ASTIdentificador extends
    ASTPrimitivaAbstrata {
  public ASTIdentificador(int id) {
    super(id);
  }

  public ASTIdentificador(Agora p, int id) {
    super(p, id);
  }

  @Override
  public String obterValor() {
    String obterValor = super.obterValor();
    if (obterValor.equals("devo"))
      return "euMesmo";
    return obterValor;
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTMensagemUnaria

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTMensagemUnaria extends SimpleNode {
  public ASTMensagemUnaria(int id) {
    super(id);
  }
```

```
  public ASTMensagemUnaria(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public String obterNomeDoMétodo() {
    return ((ASTIdentificador) jjtGetChild(0))
        .obterValor();
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTAgora

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTAgora extends SimpleNode {
  public ASTAgora(int id) {
    super(id);
  }

  public ASTAgora(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
```

```
  public ASTSentencas obterSentenças() {
    return (ASTSentencas) jjtGetChild(0);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoBinaria

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTDeclaracaoBinaria extends SimpleNode {
  private static final int POSIÇÃO_DO_SELETOR = 0;
  private static final int POSIÇÃO_DO_PARAMETRO = 1;
  private static final int POSIÇÃO_DA_DEFINIÇÃO = 2;

  public ASTDeclaracaoBinaria(int id) {
    super(id);
  }

  public ASTDeclaracaoBinaria(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public ASTSeletor obterSeletor() {
    return (ASTSeletor) jjtGetChild(POSIÇÃO_DO_SELETOR);
  }

  public String obterNomeDoParametro() {
    return ((ASTIdentificador) jjtGetChild(POSIÇÃO_DO_PARAMETRO))
        .obterValor();
```

```
  }

  public ASTSentencas obterDefinicao() {
    return (ASTSentencas) jjtGetChild(POSIÇÃO_DA_DEFINIÇÃO);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.SimpleCharStream

```
package br.ufsc.edugraf.agora.parser.javacc;

/**
 * An implementation of interface CharStream, where the stream is assumed to
 * contain only ASCII characters (without unicode processing).
 */

public class SimpleCharStream {
  public static final boolean staticFlag = false;
  int bufsize;
  int available;
  int tokenBegin;
  public int bufpos = -1;
  protected int bufline[];
  protected int bufcolumn[];

  protected int column = 0;
  protected int line = 1;

  protected boolean prevCharIsCR = false;
  protected boolean prevCharIsLF = false;

  protected java.io.Reader inputStream;

  protected char[] buffer;
  protected int maxNextCharInd = 0;
```

```
protected int inBuf = 0;
protected int tabSize = 8;

protected void setTabSize(int i) {
  tabSize = i;
}

protected int getTabSize(int i) {
  return tabSize;
}

protected void ExpandBuff(boolean wrapAround) {
  char[] newbuffer = new char[bufsize + 2048];
  int newbufline[] = new int[bufsize + 2048];
  int newbufcolumn[] = new int[bufsize + 2048];

  try {
    if (wrapAround) {
      System.arraycopy(buffer, tokenBegin,
          newbuffer, 0, bufsize
              - tokenBegin);
      System.arraycopy(buffer, 0, newbuffer,
          bufsize - tokenBegin, bufpos);
      buffer = newbuffer;

      System.arraycopy(bufline, tokenBegin,
          newbufline, 0, bufsize
              - tokenBegin);
      System.arraycopy(bufline, 0,
          newbufline, bufsize
              - tokenBegin, bufpos);
      bufline = newbufline;

      System.arraycopy(bufcolumn,
          tokenBegin, newbufcolumn, 0,
```

```
            bufsize - tokenBegin);
        System.arraycopy(bufcolumn, 0,
            newbufcolumn, bufsize
                - tokenBegin, bufpos);
        bufcolumn = newbufcolumn;

        maxNextCharInd = (bufpos += (bufsize - tokenBegin));
      } else {
        System.arraycopy(buffer, tokenBegin,
            newbuffer, 0, bufsize
                - tokenBegin);
        buffer = newbuffer;

        System.arraycopy(bufline, tokenBegin,
            newbufline, 0, bufsize
                - tokenBegin);
        bufline = newbufline;

        System.arraycopy(bufcolumn,
            tokenBegin, newbufcolumn, 0,
            bufsize - tokenBegin);
        bufcolumn = newbufcolumn;

        maxNextCharInd = (bufpos -= tokenBegin);
      }
    } catch (Throwable t) {
      throw new Error(t.getMessage());
    }

    bufsize += 2048;
    available = bufsize;
    tokenBegin = 0;
  }

  protected void FillBuff()
```

```
  throws java.io.IOException {
if (maxNextCharInd == available) {
  if (available == bufsize) {
    if (tokenBegin > 2048) {
      bufpos = maxNextCharInd = 0;
      available = tokenBegin;
    } else if (tokenBegin < 0)
      bufpos = maxNextCharInd = 0;
    else
      ExpandBuff(false);
  } else if (available > tokenBegin)
    available = bufsize;
  else if ((tokenBegin - available) < 2048)
    ExpandBuff(true);
  else
    available = tokenBegin;
}

int i;
try {
  if ((i = inputStream.read(buffer,
      maxNextCharInd, available
          - maxNextCharInd)) == -1) {
    inputStream.close();
    throw new java.io.IOException();
  } else
    maxNextCharInd += i;
  return;
} catch (java.io.IOException e) {
  --bufpos;
  backup(0);
  if (tokenBegin == -1)
    tokenBegin = bufpos;
  throw e;
}
```

```
}

public char BeginToken()
    throws java.io.IOException {
  tokenBegin = -1;
  char c = readChar();
  tokenBegin = bufpos;

  return c;
}

protected void UpdateLineColumn(char c) {
  column++;

  if (prevCharIsLF) {
    prevCharIsLF = false;
    line += (column = 1);
  } else if (prevCharIsCR) {
    prevCharIsCR = false;
    if (c == '\n') {
      prevCharIsLF = true;
    } else
      line += (column = 1);
  }

  switch (c) {
  case '\r':
    prevCharIsCR = true;
    break;
  case '\n':
    prevCharIsLF = true;
    break;
  case '\t':
    column--;
    column += (tabSize - (column % tabSize));
```

```
    break;
  default:
    break;
  }

  bufline[bufpos] = line;
  bufcolumn[bufpos] = column;
}

public char readChar() throws java.io.IOException {
  if (inBuf > 0) {
    --inBuf;

    if (++bufpos == bufsize)
      bufpos = 0;

    return buffer[bufpos];
  }

  if (++bufpos >= maxNextCharInd)
    FillBuff();

  char c = buffer[bufpos];

  UpdateLineColumn(c);
  return c;
}

/**
 * @deprecated
 * @see #getEndColumn
 */

public int getColumn() {
  return bufcolumn[bufpos];
```

```
}

/**
 * @deprecated
 * @see #getEndLine
 */

public int getLine() {
  return bufline[bufpos];
}

public int getEndColumn() {
  return bufcolumn[bufpos];
}

public int getEndLine() {
  return bufline[bufpos];
}

public int getBeginColumn() {
  return bufcolumn[tokenBegin];
}

public int getBeginLine() {
  return bufline[tokenBegin];
}

public void backup(int amount) {

  inBuf += amount;
  if ((bufpos -= amount) < 0)
    bufpos += bufsize;
}

public SimpleCharStream(java.io.Reader dstream,
```

```
    int startline, int startcolumn,
    int buffersize) {
  inputStream = dstream;
  line = startline;
  column = startcolumn - 1;

  available = bufsize = buffersize;
  buffer = new char[buffersize];
  bufline = new int[buffersize];
  bufcolumn = new int[buffersize];
}

public SimpleCharStream(java.io.Reader dstream,
    int startline, int startcolumn) {
  this(dstream, startline, startcolumn, 4096);
}

public SimpleCharStream(java.io.Reader dstream) {
  this(dstream, 1, 1, 4096);
}

public void ReInit(java.io.Reader dstream,
    int startline, int startcolumn,
    int buffersize) {
  inputStream = dstream;
  line = startline;
  column = startcolumn - 1;

  if (buffer == null
      || buffersize != buffer.length) {
    available = bufsize = buffersize;
    buffer = new char[buffersize];
    bufline = new int[buffersize];
    bufcolumn = new int[buffersize];
  }
```

```
  prevCharIsLF = prevCharIsCR = false;
  tokenBegin = inBuf = maxNextCharInd = 0;
  bufpos = -1;
}

public void ReInit(java.io.Reader dstream,
    int startline, int startcolumn) {
  ReInit(dstream, startline, startcolumn, 4096);
}

public void ReInit(java.io.Reader dstream) {
  ReInit(dstream, 1, 1, 4096);
}

public SimpleCharStream(
    java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn, int buffersize)
    throws java.io.UnsupportedEncodingException {
  this(
      encoding == null ? new java.io.InputStreamReader(
          dstream)
          : new java.io.InputStreamReader(
              dstream, encoding),
      startline, startcolumn, buffersize);
}

public SimpleCharStream(
    java.io.InputStream dstream,
    int startline, int startcolumn,
    int buffersize) {
  this(new java.io.InputStreamReader(dstream),
      startline, startcolumn, buffersize);
}
```

```
public SimpleCharStream(
    java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn)
    throws java.io.UnsupportedEncodingException {
  this(dstream, encoding, startline,
      startcolumn, 4096);
}

public SimpleCharStream(
    java.io.InputStream dstream,
    int startline, int startcolumn) {
  this(dstream, startline, startcolumn, 4096);
}

public SimpleCharStream(
    java.io.InputStream dstream,
    String encoding)
    throws java.io.UnsupportedEncodingException {
  this(dstream, encoding, 1, 1, 4096);
}

public SimpleCharStream(java.io.InputStream dstream) {
  this(dstream, 1, 1, 4096);
}

public void ReInit(java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn, int buffersize)
    throws java.io.UnsupportedEncodingException {
  ReInit(
      encoding == null ? new java.io.InputStreamReader(
          dstream)
          : new java.io.InputStreamReader(
              dstream, encoding),
```

```
      startline, startcolumn, buffersize);
}

public void ReInit(java.io.InputStream dstream,
    int startline, int startcolumn,
    int buffersize) {
  ReInit(new java.io.InputStreamReader(dstream),
      startline, startcolumn, buffersize);
}

public void ReInit(java.io.InputStream dstream,
    String encoding)
    throws java.io.UnsupportedEncodingException {
  ReInit(dstream, encoding, 1, 1, 4096);
}

public void ReInit(java.io.InputStream dstream) {
  ReInit(dstream, 1, 1, 4096);
}

public void ReInit(java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn)
    throws java.io.UnsupportedEncodingException {
  ReInit(dstream, encoding, startline,
      startcolumn, 4096);
}

public void ReInit(java.io.InputStream dstream,
    int startline, int startcolumn) {
  ReInit(dstream, startline, startcolumn, 4096);
}

public String GetImage() {
  if (bufpos >= tokenBegin)
```

```
    return new String(buffer, tokenBegin,
        bufpos - tokenBegin + 1);
  else
    return new String(buffer, tokenBegin,
        bufsize - tokenBegin)
        + new String(buffer, 0, bufpos + 1);
}

public char[] GetSuffix(int len) {
  char[] ret = new char[len];

  if ((bufpos + 1) >= len)
    System.arraycopy(buffer, bufpos - len + 1,
        ret, 0, len);
  else {
    System.arraycopy(buffer, bufsize
        - (len - bufpos - 1), ret, 0, len
        - bufpos - 1);
    System.arraycopy(buffer, 0, ret, len
        - bufpos - 1, bufpos + 1);
  }

  return ret;
}

public void Done() {
  buffer = null;
  bufline = null;
  bufcolumn = null;
}

/**
 * Method to adjust line and column numbers for the start of a token.
 */
public void adjustBeginLineColumn(int newLine,
```

```
  int newCol) {
int start = tokenBegin;
int len;

if (bufpos >= tokenBegin) {
  len = bufpos - tokenBegin + inBuf + 1;
} else {
  len = bufsize - tokenBegin + bufpos + 1
      + inBuf;
}

int i = 0, j = 0, k = 0;
int nextColDiff = 0, columnDiff = 0;

while (i < len
    && bufline[j = start % bufsize] == bufline[k = ++start
        % bufsize]) {
  bufline[j] = newLine;
  nextColDiff = columnDiff + bufcolumn[k]
      - bufcolumn[j];
  bufcolumn[j] = newCol + columnDiff;
  columnDiff = nextColDiff;
  i++;
}

if (i < len) {
  bufline[j] = newLine++;
  bufcolumn[j] = newCol + columnDiff;

  while (i++ < len) {
    if (bufline[j = start % bufsize] != bufline[++start
        % bufsize])
      bufline[j] = newLine++;
    else
      bufline[j] = newLine;
```

```
      }
    }

    line = bufline[j];
    column = bufcolumn[j];
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTBooleano

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTBooleano extends ASTPrimitivaAbstrata {
  public ASTBooleano(int id) {
    super(id);
  }

  public ASTBooleano(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTBloco

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTBloco extends SimpleNode {
```

```
  public ASTBloco(int id) {
    super(id);
  }

  public ASTBloco(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.SimpleNode

```
package br.ufsc.edugraf.agora.parser.javacc;

public class SimpleNode implements Node {
  protected Node parent;
  protected Node[] children;
  protected int id;
  protected Agora parser;

  public SimpleNode(int i) {
    id = i;
  }

  public SimpleNode(Agora p, int i) {
    this(i);
    parser = p;
  }

  public void jjtOpen() {
```

```
}

public void jjtClose() {
}

public void jjtSetParent(Node n) {
  parent = n;
}

public Node jjtGetParent() {
  return parent;
}

public void jjtAddChild(Node n, int i) {
  if (children == null) {
    children = new Node[i + 1];
  } else if (i >= children.length) {
    Node c[] = new Node[i + 1];
    System.arraycopy(children, 0, c, 0,
        children.length);
    children = c;
  }
  children[i] = n;
}

public Node jjtGetChild(int i) {
  return children[i];
}

public int jjtGetNumChildren() {
  return (children == null) ? 0
      : children.length;
}

/** Accept the visitor. * */
```

```
public Object jjtAccept(AgoraVisitor visitor,
    Object data) {
  return visitor.visit(this, data);
}

/** Accept the visitor. * */
public Object childrenAccept(AgoraVisitor visitor,
    Object data) {
  if (children != null) {
    for (int i = 0; i < children.length; ++i) {
      children[i].jjtAccept(visitor, data);
    }
  }
  return data;
}

/*
 * You can override these two methods in subclasses of SimpleNode to
 * customize the way the node appears when the tree is dumped. If your
 * output uses more than one line you should override toString(String),
 * otherwise overriding toString() is probably all you need to do.
 */

public String toString() {
  return AgoraTreeConstants.jjtNodeName[id];
}

public String toString(String prefix) {
  return prefix + toString();
}

/*
 * Override this method if you want to customize how the node dumps out its
 * children.
 */
```

```
  public void dump(String prefix) {
    System.out.println(toString(prefix));
    if (children != null) {
      for (int i = 0; i < children.length; ++i) {
        SimpleNode n = (SimpleNode) children[i];
        if (n != null) {
          n.dump(prefix + " ");
        }
      }
    }
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTExpressao

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTExpressao extends SimpleNode {
  public ASTExpressao(int id) {
    super(id);
  }

  public ASTExpressao(Agora p, int id) {
    super(p, id);
  }

  public boolean éAtribuição() {
    return jjtGetChild(0) instanceof ASTAtribuicao;
  }

  public boolean éMensagem() {
    return !(éAtribuição())
        && jjtGetNumChildren() > 1;
  }
```

```
  public int númeroDeMensagensEncadeadas() {
    if (éMensagem())
      return jjtGetNumChildren() - 2;
    return 0;
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTNumero

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTNumero extends ASTPrimitivaAbstrata {
  public ASTNumero(int id) {
    super(id);
  }

  public ASTNumero(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTLista

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTLista extends SimpleNode {
  public ASTLista(int id) {
    super(id);
  }

  public ASTLista(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.AgoraTreeConstants

```
package br.ufsc.edugraf.agora.parser.javacc;

public interface AgoraTreeConstants {
  public int JJTAGORA = 0;
  public int JJTSENTENCAS = 1;
  public int JJTDECLARACAODEMODELOOUMOLDE = 2;
  public int JJTLISTADEAGENDAS = 3;
  public int JJTDECLARACAOUNARIA = 4;
  public int JJTDECLARACAOBINARIA = 5;
  public int JJTDECLARACAOPORPALAVRACHAVE = 6;
  public int JJTRETORNO = 7;
  public int JJTEXPRESSAO = 8;
  public int JJTATRIBUICAO = 9;
```

```
  public int JJTVOID = 10;
  public int JJTBLOCO = 11;
  public int JJTPARAMETROS = 12;
  public int JJTLISTA = 13;
  public int JJTMENSAGEM = 14;
  public int JJTMENSAGEMUNARIA = 15;
  public int JJTMENSAGEMBINARIA = 16;
  public int JJTMENSAGEMPORPALAVRACHAVE = 17;
  public int JJTMENSAGEMEMCASCATA = 18;
  public int JJTIDENTIFICADOR = 19;
  public int JJTNUMERO = 20;
  public int JJTBOOLEANO = 21;
  public int JJTSTRING = 22;
  public int JJTSIMBOLO = 23;
  public int JJTSELETOR = 24;
  public int JJTPALAVRACHAVE = 25;

  public String[] jjtNodeName = { "Agora",
      "Sentencas", "DeclaracaoDeModeloOuMolde",
      "ListaDeAgendas", "DeclaracaoUnaria",
      "DeclaracaoBinaria",
      "DeclaracaoPorPalavraChave", "Retorno",
      "Expressao", "Atribuicao", "void",
      "Bloco", "Parametros", "Lista",
      "Mensagem", "MensagemUnaria",
      "MensagemBinaria",
      "MensagemPorPalavraChave",
      "MensagemEmCascata", "Identificador",
      "Numero", "Booleano", "String", "Simbolo",
      "Seletor", "PalavraChave", };
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTMensagem

```
package br.ufsc.edugraf.agora.parser.javacc;
```

```
public class ASTMensagem extends SimpleNode {
  public ASTMensagem(int id) {
    super(id);
  }

  public ASTMensagem(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTSimbolo

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTSimbolo extends ASTPrimitivaAbstrata {
  public ASTSimbolo(int id) {
    super(id);
  }

  public ASTSimbolo(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
```

```
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTSeletor

```
package br.ufsc.edugraf.agora.parser.javacc;

import java.util.HashMap;
import java.util.Map;

public class ASTSeletor extends ASTPrimitivaAbstrata {
  private static Map<String, String> nomesDasAgendas = new HashMap<String, String>(

  static {
    nomesDasAgendas.put("+", "somar!");
    nomesDasAgendas.put("-", "subtrair!");
    nomesDasAgendas.put("*", "multiplicar!");
    nomesDasAgendas.put("/", "dividir!");
    nomesDasAgendas.put("\\", "obterMódulo!");
    nomesDasAgendas.put("<", "éMenorQue!");
    nomesDasAgendas.put(">", "éMaiorQue!");
    nomesDasAgendas.put("=", "éIgual!");
    // "-","+","/","\\","*","~","<",">","=","@","%","&","?","!"
  }

  public ASTSeletor(int id) {
    super(id);
  }

  public ASTSeletor(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
```

```
      Object data) {
    return visitor.visit(this, data);
  }

  public String obterNomeDaAgenda() {
    return nomesDasAgendas.get(this.obterValor());
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoUnaria

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTDeclaracaoUnaria extends SimpleNode {
  private static final int POSIÇÃO_NOME_AGENDA = 0;
  private static final int POSIÇÃO_DEFINIÇÃO = 1;

  public ASTDeclaracaoUnaria(int id) {
    super(id);
  }

  public ASTDeclaracaoUnaria(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public String obterNomeDaMensagem() {
    return ((ASTIdentificador) jjtGetChild(POSIÇÃO_NOME_AGENDA))
        .obterValor();
  }
```

```
  public ASTSentencas obterDefinicao() {
    return (ASTSentencas) jjtGetChild(POSIÇÃO_DEFINIÇÃO);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTPrimitivaAbstrata

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTPrimitivaAbstrata extends SimpleNode {

  private String name;

  public ASTPrimitivaAbstrata(Agora p, int i) {
    super(p, i);
  }

  public ASTPrimitivaAbstrata(int i) {
    super(i);
  }

  public String obterValor() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  @Override
  public String toString() {
    return super.toString() + ": " + obterValor();
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTMensagemPorPalavraChave

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTMensagemPorPalavraChave extends
    SimpleNode {
  public ASTMensagemPorPalavraChave(int id) {
    super(id);
  }

  public ASTMensagemPorPalavraChave(Agora p, int id) {
    super(p, id);
  }

  public String obterNomeDoMétodo() {
    StringBuilder nome = new StringBuilder();
    for (int i = 0; i < jjtGetNumChildren(); i = i + 2) {
      String palavraChave = ((ASTPalavraChave) jjtGetChild(i))
          .obterValor();
      nome.append(palavraChave);
    }
    return nome.toString();
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public Object aceitarVisitanteNosParametros(
      AgoraVisitor visitante, Object data) {
    if (children != null) {
      for (int i = 1; i < jjtGetNumChildren(); i = i + 2) {
```

```
        jjtGetChild(i).jjtAccept(visitante,
            data);
      }
    }
    return data;
  }

  public String obterNúmeroDeParametros() {
    int num = 0;
    for (int i = 0; i < jjtGetNumChildren(); i = i + 2) {
      num++;
    }
    return Integer.toString(num);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.Agora

```
package br.ufsc.edugraf.agora.parser.javacc;

import java.util.*;

public class Agora/* @bgen(jjtree) */implements
    AgoraTreeConstants, AgoraConstants {/* @bgen(jjtree) */
  protected JJTAgoraState jjtree = new JJTAgoraState();

  final public ASTAgora Agora()
      throws ParseException {
    /* @bgen(jjtree) Agora */
    ASTAgora jjtn000 = new ASTAgora(JJTAGORA);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);
    try {
      Sentencas();
      jj_consume_token(0);
```

```
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    {
      if (true)
        return jjtn000;
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
  throw new Error(
```

```
      "Missing return statement in function");
}

final public void Sentencas()
    throws ParseException {
  /* @bgen(jjtree) Sentencas */
  ASTSentencas jjtn000 = new ASTSentencas(
      JJTSENTENCAS);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    label_1: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case NUMERO:
      case STRING:
      case PALAVRA_CHAVE:
      case BOOLEANO:
      case IDENTIFICADOR:
      case 17:
      case 19:
      case 21:
      case 25:
      case 27:
        ;
        break;
      default:
        jj_la1[0] = jj_gen;
        break label_1;
      }
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case 19:
        Retorno();
        break;
```

```
      case NUMERO:
      case STRING:
      case BOOLEANO:
      case IDENTIFICADOR:
      case 17:
      case 21:
      case 25:
      case 27:
        Expressao();
        break;
      case PALAVRA_CHAVE:
        DeclaracaoDeModeloOuMolde();
        break;
      default:
        jj_la1[1] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
      jj_consume_token(16);
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
```

```
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void DeclaracaoDeModeloOuMolde()
    throws ParseException {
  /* @bgen(jjtree) DeclaracaoDeModeloOuMolde */
  ASTDeclaracaoDeModeloOuMolde jjtn000 = new ASTDeclaracaoDeModeloOuMolde(
      JJTDECLARACAODEMODELOOUMOLDE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  Token tipo, p = null;
  try {
    tipo = jj_consume_token(PALAVRA_CHAVE);
    Simbolo();
    switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
    case PALAVRA_CHAVE:
      p = jj_consume_token(PALAVRA_CHAVE);
      Lista();
      break;
    default:
      jj_la1[2] = jj_gen;
      ;
    }
```

```
    ListaDeAgendas();
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    if ("molde:".equals(tipo.image))
      jjtn000.fixarÉModelo(false);
    else if ("modelo:".equals(tipo.image))
      jjtn000.fixarÉModelo(true);
    else {
      if (true)
        throw new ParseException(
            "Esperado molde:, modelo:");
    }
    if (p != null
        && !"moldadoPor:".equals(p.image)) {
      if (true)
        throw new ParseException(
            "Esperado moldadoPor:");
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
```

```
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void ListaDeAgendas()
    throws ParseException {
  /* @bgen(jjtree) ListaDeAgendas */
  ASTListaDeAgendas jjtn000 = new ASTListaDeAgendas(
      JJTLISTADEAGENDAS);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(17);
    label_2: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case PALAVRA_CHAVE:
      case SELETOR:
      case IDENTIFICADOR:
        ;
        break;
      default:
        jj_la1[3] = jj_gen;
        break label_2;
      }
      switch ((jj_ntk == -1) ? jj_ntk()
```

```
        : jj_ntk) {
      case IDENTIFICADOR:
        DeclaracaoUnaria();
        break;
      case SELETOR:
        DeclaracaoBinaria();
        break;
      case PALAVRA_CHAVE:
        DeclaracaoPorPalavraChave();
        break;
      default:
        jj_la1[4] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
    jj_consume_token(18);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
```

```
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void DeclaracaoUnaria()
    throws ParseException {
  /* @bgen(jjtree) DeclaracaoUnaria */
  ASTDeclaracaoUnaria jjtn000 = new ASTDeclaracaoUnaria(
      JJTDECLARACAOUNARIA);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    Identificador();
    jj_consume_token(17);
    Sentencas();
    jj_consume_token(18);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
```

```
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void DeclaracaoBinaria()
    throws ParseException {
  /* @bgen(jjtree) DeclaracaoBinaria */
  ASTDeclaracaoBinaria jjtn000 = new ASTDeclaracaoBinaria(
      JJTDECLARACAOBINARIA);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    Seletor();
    Identificador();
    jj_consume_token(17);
    Sentencas();
    jj_consume_token(18);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
```

```
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void DeclaracaoPorPalavraChave()
    throws ParseException {
  /* @bgen(jjtree) DeclaracaoPorPalavraChave */
  ASTDeclaracaoPorPalavraChave jjtn000 = new ASTDeclaracaoPorPalavraChave(
      JJTDECLARACAOPORPALAVRACHAVE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
```

```
    label_3: while (true) {
      PalavraChave();
      Identificador();
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case PALAVRA_CHAVE:
        ;
        break;
      default:
        jj_la1[5] = jj_gen;
        break label_3;
      }
    }
    jj_consume_token(17);
    Sentencas();
    jj_consume_token(18);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
```

```
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Retorno() throws ParseException {
  /* @bgen(jjtree) Retorno */
  ASTRetorno jjtn000 = new ASTRetorno(JJTRETORNO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(19);
    Expressao();
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
```

```
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Expressao()
    throws ParseException {
  /* @bgen(jjtree) Expressao */
  ASTExpressao jjtn000 = new ASTExpressao(
      JJTEXPRESSAO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    if (jj_2_1(2)) {
      Atribuicao();
    } else {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case NUMERO:
      case STRING:
      case BOOLEANO:
      case IDENTIFICADOR:
      case 17:
      case 21:
      case 25:
      case 27:
```

```
      ExpressaoBasica();
      break;
    default:
      jj_la1[6] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
  }
} catch (Throwable jjte000) {
  if (jjtc000) {
    jjtree.clearNodeScope(jjtn000);
    jjtc000 = false;
  } else {
    jjtree.popNode();
  }
  if (jjte000 instanceof RuntimeException) {
    {
      if (true)
        throw (RuntimeException) jjte000;
    }
  }
  if (jjte000 instanceof ParseException) {
    {
      if (true)
        throw (ParseException) jjte000;
    }
  }
  {
    if (true)
      throw (Error) jjte000;
  }
} finally {
  if (jjtc000) {
    jjtree.closeNodeScope(jjtn000, true);
  }
```

```
  }
}

final public void Atribuicao()
    throws ParseException {
  /* @bgen(jjtree) Atribuicao */
  ASTAtribuicao jjtn000 = new ASTAtribuicao(
      JJTATRIBUICAO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    Identificador();
    jj_consume_token(20);
    Expressao();
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
```

```
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void ExpressaoBasica()
    throws ParseException {
  PrimitivaOuExpressao();
  switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
  case PALAVRA_CHAVE:
  case SELETOR:
  case IDENTIFICADOR:
    Mensagem();
    MensagemEmCascata();
    break;
  default:
    jj_la1[7] = jj_gen;
    ;
  }
}

final public void PrimitivaOuExpressao()
    throws ParseException {
  switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
  case NUMERO:
  case STRING:
  case BOOLEANO:
  case IDENTIFICADOR:
  case 17:
  case 25:
  case 27:
```

```
    Primitiva();
    break;
  case 21:
    jj_consume_token(21);
    Expressao();
    jj_consume_token(22);
    break;
  default:
    jj_la1[8] = jj_gen;
    jj_consume_token(-1);
    throw new ParseException();
  }
}

final public void Primitiva()
    throws ParseException {
  switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
  case IDENTIFICADOR:
    Identificador();
    break;
  case NUMERO:
    Numero();
    break;
  case STRING:
    String();
    break;
  case BOOLEANO:
    Booleano();
    break;
  case 27:
    Simbolo();
    break;
  case 17:
    Bloco();
    break;
```

```
  case 25:
    Lista();
    break;
  default:
    jj_la1[9] = jj_gen;
    jj_consume_token(-1);
    throw new ParseException();
  }
}

final public void Bloco() throws ParseException {
  /* @bgen(jjtree) Bloco */
  ASTBloco jjtn000 = new ASTBloco(JJTBLOCO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(17);
    Parametros();
    Sentencas();
    jj_consume_token(18);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
```

```
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Parametros()
    throws ParseException {
  /* @bgen(jjtree) Parametros */
  ASTParametros jjtn000 = new ASTParametros(
      JJTPARAMETROS);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
    case 23:
      label_4: while (true) {
        jj_consume_token(23);
        Identificador();
        switch ((jj_ntk == -1) ? jj_ntk()
            : jj_ntk) {
        case 23:
          ;
          break;
        default:
          jj_la1[10] = jj_gen;
```

```
        break label_4;
      }
    }
    jj_consume_token(24);
    break;
  default:
    jj_la1[11] = jj_gen;
    ;
  }
} catch (Throwable jjte000) {
  if (jjtc000) {
    jjtree.clearNodeScope(jjtn000);
    jjtc000 = false;
  } else {
    jjtree.popNode();
  }
  if (jjte000 instanceof RuntimeException) {
    {
      if (true)
        throw (RuntimeException) jjte000;
    }
  }
  if (jjte000 instanceof ParseException) {
    {
      if (true)
        throw (ParseException) jjte000;
    }
  }
  {
    if (true)
      throw (Error) jjte000;
  }
} finally {
  if (jjtc000) {
    jjtree.closeNodeScope(jjtn000, true);
```

```
    }
  }
}

final public void Lista() throws ParseException {
  /* @bgen(jjtree) Lista */
  ASTLista jjtn000 = new ASTLista(JJTLISTA);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(25);
    label_5: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case NUMERO:
      case STRING:
      case BOOLEANO:
      case IDENTIFICADOR:
      case 17:
      case 25:
      case 27:
        ;
        break;
      default:
        jj_la1[12] = jj_gen;
        break label_5;
      }
      Primitiva();
    }
    jj_consume_token(18);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
```

```
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Mensagem() throws ParseException {
  /* @bgen(jjtree) Mensagem */
  ASTMensagem jjtn000 = new ASTMensagem(
      JJTMENSAGEM);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
    case IDENTIFICADOR:
      label_6: while (true) {
```

```
      MensagemUnaria();
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case IDENTIFICADOR:
        ;
        break;
      default:
        jj_la1[13] = jj_gen;
        break label_6;
      }
    }
    label_7: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case SELETOR:
        ;
        break;
      default:
        jj_la1[14] = jj_gen;
        break label_7;
      }
      MensagemBinaria();
    }
    switch ((jj_ntk == -1) ? jj_ntk()
        : jj_ntk) {
    case PALAVRA_CHAVE:
      MensagemPorPalavraChave();
      break;
    default:
      jj_la1[15] = jj_gen;
      ;
    }
    break;
  case SELETOR:
    label_8: while (true) {
```

```
        MensagemBinaria();
        switch ((jj_ntk == -1) ? jj_ntk()
            : jj_ntk) {
        case SELETOR:
          ;
          break;
        default:
          jj_la1[16] = jj_gen;
          break label_8;
        }
      }
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case PALAVRA_CHAVE:
        MensagemPorPalavraChave();
        break;
      default:
        jj_la1[17] = jj_gen;
        ;
      }
      break;
    case PALAVRA_CHAVE:
      MensagemPorPalavraChave();
      break;
    default:
      jj_la1[18] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
```

```
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void MensagemUnaria()
    throws ParseException {
  /* @bgen(jjtree) MensagemUnaria */
  ASTMensagemUnaria jjtn000 = new ASTMensagemUnaria(
      JJTMENSAGEMUNARIA);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    Identificador();
  } catch (Throwable jjte000) {
    if (jjtc000) {
```

```
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void MensagemBinaria()
    throws ParseException {
  /* @bgen(jjtree) MensagemBinaria */
  ASTMensagemBinaria jjtn000 = new ASTMensagemBinaria(
      JJTMENSAGEMBINARIA);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
```

```
try {
  Seletor();
  ASTExpressao jjtn001 = new ASTExpressao(
      JJTEXPRESSAO);
  boolean jjtc001 = true;
  jjtree.openNodeScope(jjtn001);
  try {
    PrimitivaOuExpressao();
    label_9: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case IDENTIFICADOR:
        ;
        break;
      default:
        jj_la1[19] = jj_gen;
        break label_9;
      }
      MensagemUnaria();
    }
  } catch (Throwable jjte001) {
    if (jjtc001) {
      jjtree.clearNodeScope(jjtn001);
      jjtc001 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte001 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte001;
      }
    }
    if (jjte001 instanceof ParseException) {
      {
```

```
        if (true)
          throw (ParseException) jjte001;
      }
    }
    {
      if (true)
        throw (Error) jjte001;
    }
  } finally {
    if (jjtc001) {
      jjtree.closeNodeScope(jjtn001,
          true);
    }
  }
} catch (Throwable jjte000) {
  if (jjtc000) {
    jjtree.clearNodeScope(jjtn000);
    jjtc000 = false;
  } else {
    jjtree.popNode();
  }
  if (jjte000 instanceof RuntimeException) {
    {
      if (true)
        throw (RuntimeException) jjte000;
    }
  }
  if (jjte000 instanceof ParseException) {
    {
      if (true)
        throw (ParseException) jjte000;
    }
  }
  {
    if (true)
```

```
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void MensagemPorPalavraChave()
    throws ParseException {
  /* @bgen(jjtree) MensagemPorPalavraChave */
  ASTMensagemPorPalavraChave jjtn000 = new ASTMensagemPorPalavraChave(
      JJTMENSAGEMPORPALAVRACHAVE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    label_10: while (true) {
      PalavraChave();
      Argumento();
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case PALAVRA_CHAVE:
        ;
        break;
      default:
        jj_la1[20] = jj_gen;
        break label_10;
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
```

```
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Argumento()
    throws ParseException {
  /* @bgen(jjtree) Expressao */
  ASTExpressao jjtn000 = new ASTExpressao(
      JJTEXPRESSAO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    PrimitivaOuExpressao();
    label_11: while (true) {
```

```
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case IDENTIFICADOR:
        ;
        break;
      default:
        jj_la1[21] = jj_gen;
        break label_11;
      }
      MensagemUnaria();
    }
    label_12: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk()
          : jj_ntk) {
      case SELETOR:
        ;
        break;
      default:
        jj_la1[22] = jj_gen;
        break label_12;
      }
      MensagemBinaria();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {
        if (true)
          throw (RuntimeException) jjte000;
      }
```

```
    }
    if (jjte000 instanceof ParseException) {
      {
        if (true)
          throw (ParseException) jjte000;
      }
    }
    {
      if (true)
        throw (Error) jjte000;
    }
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void MensagemEmCascata()
    throws ParseException {
  label_13: while (true) {
    switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
    case 26:
      ;
      break;
    default:
      jj_la1[23] = jj_gen;
      break label_13;
    }
    ASTMensagemEmCascata jjtn001 = new ASTMensagemEmCascata(
        JJTMENSAGEMEMCASCATA);
    boolean jjtc001 = true;
    jjtree.openNodeScope(jjtn001);
    try {
      jj_consume_token(26);
```

```
      Mensagem();
    } catch (Throwable jjte001) {
      if (jjtc001) {
        jjtree.clearNodeScope(jjtn001);
        jjtc001 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte001 instanceof RuntimeException) {
        {
          if (true)
            throw (RuntimeException) jjte001;
        }
      }
      if (jjte001 instanceof ParseException) {
        {
          if (true)
            throw (ParseException) jjte001;
        }
      }
      {
        if (true)
          throw (Error) jjte001;
      }
    } finally {
      if (jjtc001) {
        jjtree.closeNodeScope(jjtn001,
            true);
      }
    }
  }
}

final public void Identificador()
    throws ParseException {
```

```
  /* @bgen(jjtree) Identificador */
  ASTIdentificador jjtn000 = new ASTIdentificador(
      JJTIDENTIFICADOR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(IDENTIFICADOR);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Numero() throws ParseException {
  /* @bgen(jjtree) Numero */
  ASTNumero jjtn000 = new ASTNumero(JJTNUMERO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(NUMERO);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Booleano() throws ParseException {
```

```
  /* @bgen(jjtree) Booleano */
  ASTBooleano jjtn000 = new ASTBooleano(
      JJTBOOLEANO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(BOOLEANO);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void String() throws ParseException {
  /* @bgen(jjtree) String */
  ASTString jjtn000 = new ASTString(JJTSTRING);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(STRING);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Simbolo() throws ParseException {
```

```
  /* @bgen(jjtree) Simbolo */
  ASTSimbolo jjtn000 = new ASTSimbolo(JJTSIMBOLO);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(27);
    jj_consume_token(IDENTIFICADOR);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void Seletor() throws ParseException {
  /* @bgen(jjtree) Seletor */
  ASTSeletor jjtn000 = new ASTSeletor(JJTSELETOR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(SELETOR);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void PalavraChave()
```

```
    throws ParseException {
  /* @bgen(jjtree) PalavraChave */
  ASTPalavraChave jjtn000 = new ASTPalavraChave(
      JJTPALAVRACHAVE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(PALAVRA_CHAVE);
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
    jjtn000.setName(token.image);
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final private boolean jj_2_1(int xla) {
  jj_la = xla;
  jj_lastpos = jj_scanpos = token;
  try {
    return !jj_3_1();
  } catch (LookaheadSuccess ls) {
    return true;
  } finally {
    jj_save(0, xla);
  }
}

final private boolean jj_3_1() {
  if (jj_3R_14())
    return true;
  return false;
}
```

```
final private boolean jj_3R_15() {
  if (jj_scan_token(IDENTIFICADOR))
    return true;
  return false;
}

final private boolean jj_3R_14() {
  if (jj_3R_15())
    return true;
  if (jj_scan_token(20))
    return true;
  return false;
}

public AgoraTokenManager token_source;
SimpleCharStream jj_input_stream;
public Token token, jj_nt;
private int jj_ntk;
private Token jj_scanpos, jj_lastpos;
private int jj_la;
public boolean lookingAhead = false;
private boolean jj_semLA;
private int jj_gen;
final private int[] jj_la1 = new int[24];
static private int[] jj_la1_0;
static {
  jj_la1_0();
}

private static void jj_la1_0() {
  jj_la1_0 = new int[] { 0xa2a1780, 0xa2a1780,
      0x200, 0x1a00, 0x1a00, 0x200,
      0xa221580, 0x1a00, 0xa221580,
      0xa021580, 0x800000, 0x800000,
```

```
      0xa021580, 0x1000, 0x800, 0x200,
      0x800, 0x200, 0x1a00, 0x1000, 0x200,
      0x1000, 0x800, 0x4000000, };
}

final private JJCalls[] jj_2_rtns = new JJCalls[1];
private boolean jj_rescan = false;
private int jj_gc = 0;

public Agora(java.io.InputStream stream) {
  this(stream, null);
}

public Agora(java.io.InputStream stream,
    String encoding) {
  try {
    jj_input_stream = new SimpleCharStream(
        stream, encoding, 1, 1);
  } catch (java.io.UnsupportedEncodingException e) {
    throw new RuntimeException(e);
  }
  token_source = new AgoraTokenManager(
      jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 24; i++)
    jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++)
    jj_2_rtns[i] = new JJCalls();
}

public void ReInit(java.io.InputStream stream) {
  ReInit(stream, null);
}
```

```
public void ReInit(java.io.InputStream stream,
    String encoding) {
  try {
    jj_input_stream.ReInit(stream, encoding,
        1, 1);
  } catch (java.io.UnsupportedEncodingException e) {
    throw new RuntimeException(e);
  }
  token_source.ReInit(jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jjtree.reset();
  jj_gen = 0;
  for (int i = 0; i < 24; i++)
    jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++)
    jj_2_rtns[i] = new JJCalls();
}

public Agora(java.io.Reader stream) {
  jj_input_stream = new SimpleCharStream(stream,
      1, 1);
  token_source = new AgoraTokenManager(
      jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 24; i++)
    jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++)
    jj_2_rtns[i] = new JJCalls();
}

public void ReInit(java.io.Reader stream) {
```

```
    jj_input_stream.ReInit(stream, 1, 1);
    token_source.ReInit(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jjtree.reset();
    jj_gen = 0;
    for (int i = 0; i < 24; i++)
      jj_la1[i] = -1;
    for (int i = 0; i < jj_2_rtns.length; i++)
      jj_2_rtns[i] = new JJCalls();
  }

  public Agora(AgoraTokenManager tm) {
    token_source = tm;
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 24; i++)
      jj_la1[i] = -1;
    for (int i = 0; i < jj_2_rtns.length; i++)
      jj_2_rtns[i] = new JJCalls();
  }

  public void ReInit(AgoraTokenManager tm) {
    token_source = tm;
    token = new Token();
    jj_ntk = -1;
    jjtree.reset();
    jj_gen = 0;
    for (int i = 0; i < 24; i++)
      jj_la1[i] = -1;
    for (int i = 0; i < jj_2_rtns.length; i++)
      jj_2_rtns[i] = new JJCalls();
  }
```

```
final private Token jj_consume_token(int kind)
    throws ParseException {
  Token oldToken;
  if ((oldToken = token).next != null)
    token = token.next;
  else
    token = token.next = token_source
        .getNextToken();
  jj_ntk = -1;
  if (token.kind == kind) {
    jj_gen++;
    if (++jj_gc > 100) {
      jj_gc = 0;
      for (int i = 0; i < jj_2_rtns.length; i++) {
        JJCalls c = jj_2_rtns[i];
        while (c != null) {
          if (c.gen < jj_gen)
            c.first = null;
          c = c.next;
        }
      }
    }
    return token;
  }
  token = oldToken;
  jj_kind = kind;
  throw generateParseException();
}

static private final class LookaheadSuccess extends
    java.lang.Error {
}

final private LookaheadSuccess jj_ls = new LookaheadSuccess();
```

```
final private boolean jj_scan_token(int kind) {
  if (jj_scanpos == jj_lastpos) {
    jj_la--;
    if (jj_scanpos.next == null) {
      jj_lastpos = jj_scanpos = jj_scanpos.next = token_source
          .getNextToken();
    } else {
      jj_lastpos = jj_scanpos = jj_scanpos.next;
    }
  } else {
    jj_scanpos = jj_scanpos.next;
  }
  if (jj_rescan) {
    int i = 0;
    Token tok = token;
    while (tok != null && tok != jj_scanpos) {
      i++;
      tok = tok.next;
    }
    if (tok != null)
      jj_add_error_token(kind, i);
  }
  if (jj_scanpos.kind != kind)
    return true;
  if (jj_la == 0 && jj_scanpos == jj_lastpos)
    throw jj_ls;
  return false;
}

final public Token getNextToken() {
  if (token.next != null)
    token = token.next;
  else
    token = token.next = token_source
        .getNextToken();
```

```
  jj_ntk = -1;
  jj_gen++;
  return token;
}

final public Token getToken(int index) {
  Token t = lookingAhead ? jj_scanpos : token;
  for (int i = 0; i < index; i++) {
    if (t.next != null)
      t = t.next;
    else
      t = t.next = token_source
          .getNextToken();
  }
  return t;
}

final private int jj_ntk() {
  if ((jj_nt = token.next) == null)
    return (jj_ntk = (token.next = token_source
        .getNextToken()).kind);
  else
    return (jj_ntk = jj_nt.kind);
}

private java.util.Vector<int[]> jj_expentries = new java.util.Vector<int[]>();
private int[] jj_expentry;
private int jj_kind = -1;
private int[] jj_lasttokens = new int[100];
private int jj_endpos;

private void jj_add_error_token(int kind, int pos) {
  if (pos >= 100)
    return;
  if (pos == jj_endpos + 1) {
```

```
    jj_lasttokens[jj_endpos++] = kind;
  } else if (jj_endpos != 0) {
    jj_expentry = new int[jj_endpos];
    for (int i = 0; i < jj_endpos; i++) {
      jj_expentry[i] = jj_lasttokens[i];
    }
    boolean exists = false;
    for (java.util.Enumeration e = jj_expentries
        .elements(); e.hasMoreElements();) {
      int[] oldentry = (int[]) (e
          .nextElement());
      if (oldentry.length == jj_expentry.length) {
        exists = true;
        for (int i = 0; i < jj_expentry.length; i++) {
          if (oldentry[i] != jj_expentry[i]) {
            exists = false;
            break;
          }
        }
        if (exists)
          break;
      }
    }
    if (!exists)
      jj_expentries.addElement(jj_expentry);
    if (pos != 0)
      jj_lasttokens[(jj_endpos = pos) - 1] = kind;
  }
}

public ParseException generateParseException() {
  jj_expentries.removeAllElements();
  boolean[] la1tokens = new boolean[28];
  if (jj_kind >= 0) {
    la1tokens[jj_kind] = true;
```

```
    jj_kind = -1;
  }
  for (int i = 0; i < 24; i++) {
    if (jj_la1[i] == jj_gen) {
      for (int j = 0; j < 32; j++) {
        if ((jj_la1_0[i] & (1 << j)) != 0) {
          la1tokens[j] = true;
        }
      }
    }
  }
  for (int i = 0; i < 28; i++) {
    if (la1tokens[i]) {
      jj_expentry = new int[1];
      jj_expentry[0] = i;
      jj_expentries.addElement(jj_expentry);
    }
  }
  jj_endpos = 0;
  jj_rescan_token();
  jj_add_error_token(0, 0);
  int[][] exptokseq = new int[jj_expentries
      .size()][];
  for (int i = 0; i < jj_expentries.size(); i++) {
    exptokseq[i] = jj_expentries.elementAt(i);
  }
  return new ParseException(token, exptokseq,
      tokenImage);
}

final public void enable_tracing() {
}

final public void disable_tracing() {
}
```

```
final private void jj_rescan_token() {
  jj_rescan = true;
  for (int i = 0; i < 1; i++) {
    try {
      JJCalls p = jj_2_rtns[i];
      do {
        if (p.gen > jj_gen) {
          jj_la = p.arg;
          jj_lastpos = jj_scanpos = p.first;
          switch (i) {
          case 0:
            jj_3_1();
            break;
          }
        }
        p = p.next;
      } while (p != null);
    } catch (LookaheadSuccess ls) {
    }
  }
  jj_rescan = false;
}

final private void jj_save(int index, int xla) {
  JJCalls p = jj_2_rtns[index];
  while (p.gen > jj_gen) {
    if (p.next == null) {
      p = p.next = new JJCalls();
      break;
    }
    p = p.next;
  }
  p.gen = jj_gen + xla - jj_la;
  p.first = token;
```

```
    p.arg = xla;
  }

  static final class JJCalls {
    int gen;
    Token first;
    int arg;
    JJCalls next;
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTPrimitiva

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTPrimitiva extends SimpleNode {
  public ASTPrimitiva(int id) {
    super(id);
  }

  public ASTPrimitiva(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTMensagemBinaria

```
package br.ufsc.edugraf.agora.parser.javacc;
```

```
import br.ufsc.edugraf.agora.geracaoDeCodigo.VisitanteGeradorDeBlocoDeComandos;

public class ASTMensagemBinaria extends SimpleNode {
  public ASTMensagemBinaria(int id) {
    super(id);
  }

  public ASTMensagemBinaria(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public Object visitarExpressao(
      AgoraVisitor visitante, Object data) {
    return jjtGetChild(1).jjtAccept(visitante,
        data);
  }

  public Object visitarSeletor(
      AgoraVisitor visitante, Object data) {
    return jjtGetChild(0).jjtAccept(visitante,
        data);
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTParametros

```
package br.ufsc.edugraf.agora.parser.javacc;
```

```
public class ASTParametros extends SimpleNode {
  public ASTParametros(int id) {
    super(id);
  }

  public ASTParametros(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.AgoraConstants

```
package br.ufsc.edugraf.agora.parser.javacc;

public interface AgoraConstants {

  int EOF = 0;
  int NUMERO = 7;
  int STRING = 8;
  int PALAVRA_CHAVE = 9;
  int BOOLEANO = 10;
  int SELETOR = 11;
  int IDENTIFICADOR = 12;
  int DIGITOS = 13;
  int LETRA = 14;
  int DIGITO = 15;

  int DEFAULT = 0;
```

```
  String[] tokenImage = { "<EOF>", "\" \"",
      "\"\\t\"", "\"\\n\"", "\"\\r\"",
      "<token of kind 5>", "<token of kind 6>",
      "<NUMERO>", "<STRING>", "<PALAVRA_CHAVE>",
      "<BOOLEANO>", "<SELETOR>",
      "<IDENTIFICADOR>", "<DIGITOS>", "<LETRA>",
      "<DIGITO>", "\".\"", "\"[\"", "\"]\"",
      "\"^\"", "\":=\"", "\"(\"", "\")\"",
      "\":\"", "\"|\"", "\"#[\"", "\";\"",
      "\"#\"", };

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.Token

```
package br.ufsc.edugraf.agora.parser.javacc;

/**
 * Describes the input token stream.
 */

public class Token {

  /**
   * An integer that describes the kind of this token. This numbering system
   * is determined by JavaCCParser, and a table of these numbers is stored in
   * the file ...Constants.java.
   */
  public int kind;

  /**
   * beginLine and beginColumn describe the position of the first character of
   * this token; endLine and endColumn describe the position of the last
   * character of this token.
```

```
 */
public int beginLine, beginColumn, endLine,
    endColumn;

/**
 * The string image of the token.
 */
public String image;

/**
 * A reference to the next regular (non-special) token from the input
 * stream. If this is the last token from the input stream, or if the token
 * manager has not read tokens beyond this one, this field is set to null.
 * This is true only if this token is also a regular token. Otherwise, see
 * below for a description of the contents of this field.
 */
public Token next;

/**
 * This field is used to access special tokens that occur prior to this
 * token, but after the immediately preceding regular (non-special) token.
 * If there are no such special tokens, this field is set to null. When
 * there are more than one such special token, this field refers to the last
 * of these special tokens, which in turn refers to the next previous
 * special token through its specialToken field, and so on until the first
 * special token (whose specialToken field is null). The next fields of
 * special tokens refer to other special tokens that immediately follow it
 * (without an intervening regular token). If there is no such token, this
 * field is null.
 */
public Token specialToken;

/**
 * Returns the image.
 */
```

```
  public String toString() {
    return image;
  }

  /**
   * Returns a new Token object, by default. However, if you want, you can
   * create and return subclass objects based on the value of ofKind. Simply
   * add the cases to the switch for all those special cases. For example, if
   * you have a subclass of Token called IDToken that you want to create if
   * ofKind is ID, simlpy add something like :
   *
   * case MyParserConstants.ID : return new IDToken();
   *
   * to the following switch statement. Then you can cast matchedToken
   * variable to the appropriate type and use it in your lexical actions.
   */
  public static final Token newToken(int ofKind) {
    switch (ofKind) {
    default:
      return new Token();
    }
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.Node

```
package br.ufsc.edugraf.agora.parser.javacc;

/*
 * All AST nodes must implement this interface. It provides basic machinery for
 * constructing the parent and child relationships between nodes.
 */

public interface Node {
```

```
/**
 * This method is called after the node has been made the current node. It
 * indicates that child nodes can now be added to it.
 */
public void jjtOpen();

/**
 * This method is called after all the child nodes have been added.
 */
public void jjtClose();

/**
 * This pair of methods are used to inform the node of its parent.
 */
public void jjtSetParent(Node n);

public Node jjtGetParent();

/**
 * This method tells the node to add its argument to the node's list of
 * children.
 */
public void jjtAddChild(Node n, int i);

/**
 * This method returns a child node. The children are numbered from zero,
 * left to right.
 */
public Node jjtGetChild(int i);

/** Return the number of children the node has. */
public int jjtGetNumChildren();

/** Accept the visitor. * */
```

```
  public Object jjtAccept(AgoraVisitor visitor,
      Object data);
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.AgoraTokenManager

```
package br.ufsc.edugraf.agora.parser.javacc;

import java.util.*;

public class AgoraTokenManager implements
    AgoraConstants {
  public java.io.PrintStream debugStream = System.out;

  public void setDebugStream(java.io.PrintStream ds) {
    debugStream = ds;
  }

  private final int jjStopStringLiteralDfa_0(
      int pos, long active0) {
    switch (pos) {
    default:
      return -1;
    }
  }

  private final int jjStartNfa_0(int pos,
      long active0) {
    return jjMoveNfa_0(jjStopStringLiteralDfa_0(
        pos, active0), pos + 1);
  }

  private final int jjStopAtPos(int pos, int kind) {
    jjmatchedKind = kind;
    jjmatchedPos = pos;
```

```
  return pos + 1;
}

private final int jjStartNfaWithStates_0(int pos,
    int kind, int state) {
  jjmatchedKind = kind;
  jjmatchedPos = pos;
  try {
    curChar = input_stream.readChar();
  } catch (java.io.IOException e) {
    return pos + 1;
  }
  return jjMoveNfa_0(state, pos + 1);
}

private final int jjMoveStringLiteralDfa0_0() {
  switch (curChar) {
  case 35:
    jjmatchedKind = 27;
    return jjMoveStringLiteralDfa1_0(0x2000000L);
  case 40:
    return jjStopAtPos(0, 21);
  case 41:
    return jjStopAtPos(0, 22);
  case 46:
    return jjStopAtPos(0, 16);
  case 58:
    jjmatchedKind = 23;
    return jjMoveStringLiteralDfa1_0(0x100000L);
  case 59:
    return jjStopAtPos(0, 26);
  case 91:
    return jjStopAtPos(0, 17);
  case 93:
    return jjStopAtPos(0, 18);
```

```
  case 94:
    return jjStopAtPos(0, 19);
  case 124:
    return jjStopAtPos(0, 24);
  default:
    return jjMoveNfa_0(0, 0);
  }
}

private final int jjMoveStringLiteralDfa1_0(
    long active0) {
  try {
    curChar = input_stream.readChar();
  } catch (java.io.IOException e) {
    jjStopStringLiteralDfa_0(0, active0);
    return 1;
  }
  switch (curChar) {
  case 61:
    if ((active0 & 0x100000L) != 0L)
      return jjStopAtPos(1, 20);
    break;
  case 91:
    if ((active0 & 0x2000000L) != 0L)
      return jjStopAtPos(1, 25);
    break;
  default:
    break;
  }
  return jjStartNfa_0(0, active0);
}

private final void jjCheckNAdd(int state) {
  if (jjrounds[state] != jjround) {
    jjstateSet[jjnewStateCnt++] = state;
```

```
    jjrounds[state] = jjround;
  }
}

private final void jjAddStates(int start, int end) {
  do {
    jjstateSet[jjnewStateCnt++] = jjnextStates[start];
  } while (start++ != end);
}

private final void jjCheckNAddTwoStates(
    int state1, int state2) {
  jjCheckNAdd(state1);
  jjCheckNAdd(state2);
}

private final void jjCheckNAddStates(int start,
    int end) {
  do {
    jjCheckNAdd(jjnextStates[start]);
  } while (start++ != end);
}

private final void jjCheckNAddStates(int start) {
  jjCheckNAdd(jjnextStates[start]);
  jjCheckNAdd(jjnextStates[start + 1]);
}

static final long[] jjbitVec0 = { 0x0L, 0x0L,
    0xffffffffffffffffL, 0xffffffffffffffffL };
static final long[] jjbitVec1 = { 0x0L, 0x0L,
    0x0L, 0x1e7cff9f1e7cff9fL };

private final int jjMoveNfa_0(int startState,
    int curPos) {
```

```
int[] nextStates;
int startsAt = 0;
jjnewStateCnt = 43;
int i = 1;
jjstateSet[0] = startState;
int j, kind = 0x7fffffff;
for (;;) {
  if (++jjround == 0x7fffffff)
    ReInitRounds();
  if (curChar < 64) {
    long l = 1L << curChar;
    MatchLoop: do {
      switch (jjstateSet[--i]) {
      case 0:
        if ((0xf000bc6200000000L & l) != 0L) {
          if (kind > 11)
            kind = 11;
        } else if ((0x3ff000000000000L & l) != 0L) {
          if (kind > 7)
            kind = 7;
          jjCheckNAddStates(0, 2);
        } else if (curChar == 34)
          jjCheckNAddStates(3, 5);
        if ((0x280000000000L & l) != 0L)
          jjCheckNAdd(1);
        else if (curChar == 47)
          jjAddStates(6, 7);
        break;
      case 1:
        if ((0x3ff000000000000L & l) == 0L)
          break;
        if (kind > 7)
          kind = 7;
        jjCheckNAddStates(0, 2);
        break;
```

```
case 2:
  if (curChar == 44)
    jjCheckNAdd(3);
  break;
case 3:
  if ((0x3ff000000000000L & l) == 0L)
    break;
  if (kind > 7)
    kind = 7;
  jjCheckNAddTwoStates(3, 4);
  break;
case 5:
  if ((0x280000000000L & l) != 0L)
    jjCheckNAdd(6);
  break;
case 6:
  if ((0x3ff000000000000L & l) == 0L)
    break;
  if (kind > 7)
    kind = 7;
  jjCheckNAdd(6);
  break;
case 7:
  if (curChar == 34)
    jjCheckNAddStates(3, 5);
  break;
case 8:
  if ((0xfffffffbffffffffL & l) != 0L)
    jjCheckNAddTwoStates(8, 9);
  break;
case 9:
  if (curChar == 34 && kind > 8)
    kind = 8;
  break;
case 11:
```

```
  if (curChar == 34)
    jjCheckNAdd(9);
  break;
case 26:
  if ((0xf000bc6200000000L & l) != 0L
      && kind > 11)
    kind = 11;
  break;
case 27:
  if (curChar == 47)
    jjAddStates(6, 7);
  break;
case 28:
  if (curChar == 47)
    jjCheckNAddStates(8, 10);
  break;
case 29:
  if ((0xfffffffffffdbffL & l) != 0L)
    jjCheckNAddStates(8, 10);
  break;
case 30:
  if ((0x2400L & l) != 0L
      && kind > 5)
    kind = 5;
  break;
case 31:
  if (curChar == 10 && kind > 5)
    kind = 5;
  break;
case 32:
  if (curChar == 13)
    jjstateSet[jjnewStateCnt++] = 31;
  break;
case 33:
  if (curChar == 42)
```

```
    jjCheckNAddTwoStates(34,
        35);
  break;
case 34:
  if ((0xfffffbffffffffffL & l) != 0L)
    jjCheckNAddTwoStates(34,
        35);
  break;
case 35:
  if (curChar == 42)
    jjAddStates(11, 12);
  break;
case 36:
  if ((0xffff7fffffffffffL & l) != 0L)
    jjCheckNAddTwoStates(37,
        35);
  break;
case 37:
  if ((0xfffffbffffffffffL & l) != 0L)
    jjCheckNAddTwoStates(37,
        35);
  break;
case 38:
  if (curChar == 47 && kind > 6)
    kind = 6;
  break;
case 40:
  if ((0x3ff000000000000L & l) != 0L)
    jjAddStates(13, 14);
  break;
case 41:
  if (curChar == 58 && kind > 9)
    kind = 9;
  break;
case 42:
```

```
      if ((0x3ff000000000000L & l) == 0L)
        break;
      if (kind > 12)
        kind = 12;
      jjstateSet[jjnewStateCnt++] = 42;
      break;
    default:
      break;
    }
  } while (i != startsAt);
} else if (curChar < 128) {
  long l = 1L << (curChar & 077);
  MatchLoop: do {
    switch (jjstateSet[--i]) {
    case 0:
      if ((0x7fffffe87fffffeL & l) != 0L) {
        if (kind > 12)
          kind = 12;
        jjCheckNAddStates(15, 17);
      } else if ((0x4000000010000001L & l) != 0L) {
        if (kind > 11)
          kind = 11;
      }
      if (curChar == 102)
        jjstateSet[jjnewStateCnt++] = 24;
      else if (curChar == 118)
        jjstateSet[jjnewStateCnt++] = 20;
      break;
    case 4:
      if ((0x2000000020L & l) != 0L)
        jjAddStates(18, 19);
      break;
    case 8:
      if ((0xfffffffeffffffffL & l) != 0L)
        jjCheckNAddTwoStates(8, 9);
```

```
  break;
case 10:
  if (curChar == 92)
    jjstateSet[jjnewStateCnt++] = 11;
  break;
case 11:
  if ((0x14404410000000L & l) != 0L)
    jjCheckNAdd(9);
  break;
case 12:
  if (curChar == 111
      && kind > 10)
    kind = 10;
  break;
case 13:
  if (curChar == 114)
    jjCheckNAdd(12);
  break;
case 14:
  if (curChar == 105)
    jjstateSet[jjnewStateCnt++] = 13;
  break;
case 15:
  if (curChar == 101)
    jjstateSet[jjnewStateCnt++] = 14;
  break;
case 16:
  if (curChar == 100)
    jjstateSet[jjnewStateCnt++] = 15;
  break;
case 17:
  if (curChar == 97)
    jjstateSet[jjnewStateCnt++] = 16;
  break;
case 18:
```

```
  if (curChar == 100)
    jjstateSet[jjnewStateCnt++] = 17;
  break;
case 19:
  if (curChar == 114)
    jjstateSet[jjnewStateCnt++] = 18;
  break;
case 20:
  if (curChar == 101)
    jjstateSet[jjnewStateCnt++] = 19;
  break;
case 21:
  if (curChar == 118)
    jjstateSet[jjnewStateCnt++] = 20;
  break;
case 22:
  if (curChar == 115)
    jjCheckNAdd(12);
  break;
case 23:
  if (curChar == 108)
    jjstateSet[jjnewStateCnt++] = 22;
  break;
case 24:
  if (curChar == 97)
    jjstateSet[jjnewStateCnt++] = 23;
  break;
case 25:
  if (curChar == 102)
    jjstateSet[jjnewStateCnt++] = 24;
  break;
case 26:
  if ((0x4000000010000001L & l) != 0L
      && kind > 11)
    kind = 11;
```

```
      break;
    case 29:
      jjAddStates(8, 10);
      break;
    case 34:
      jjCheckNAddTwoStates(34, 35);
      break;
    case 36:
    case 37:
      jjCheckNAddTwoStates(37, 35);
      break;
    case 39:
      if ((0x7fffffe87fffffeL & l) == 0L)
        break;
      if (kind > 12)
        kind = 12;
      jjCheckNAddStates(15, 17);
      break;
    case 40:
      if ((0x7fffffe87fffffeL & l) != 0L)
        jjCheckNAddTwoStates(40,
            41);
      break;
    case 42:
      if ((0x7fffffe87fffffeL & l) == 0L)
        break;
      if (kind > 12)
        kind = 12;
      jjCheckNAdd(42);
      break;
    default:
      break;
    }
  } while (i != startsAt);
} else {
```

```
int i2 = (curChar & 0xff) >> 6;
long l2 = 1L << (curChar & 077);
MatchLoop: do {
  switch (jjstateSet[--i]) {
  case 0:
    if ((jjbitVec1[i2] & l2) == 0L)
      break;
    if (kind > 12)
      kind = 12;
    jjCheckNAddStates(15, 17);
    break;
  case 8:
    if ((jjbitVec0[i2] & l2) != 0L)
      jjAddStates(20, 21);
    break;
  case 29:
    if ((jjbitVec0[i2] & l2) != 0L)
      jjAddStates(8, 10);
    break;
  case 34:
    if ((jjbitVec0[i2] & l2) != 0L)
      jjCheckNAddTwoStates(34,
          35);
    break;
  case 36:
  case 37:
    if ((jjbitVec0[i2] & l2) != 0L)
      jjCheckNAddTwoStates(37,
          35);
    break;
  case 40:
    if ((jjbitVec1[i2] & l2) != 0L)
      jjCheckNAddTwoStates(40,
          41);
    break;
```

```
        case 42:
          if ((jjbitVec1[i2] & l2) == 0L)
            break;
          if (kind > 12)
            kind = 12;
          jjCheckNAdd(42);
          break;
        default:
          break;
        }
      } while (i != startsAt);
    }
    if (kind != 0x7fffffff) {
      jjmatchedKind = kind;
      jjmatchedPos = curPos;
      kind = 0x7fffffff;
    }
    ++curPos;
    if ((i = jjnewStateCnt) == (startsAt = 43 - (jjnewStateCnt = startsAt)))
      return curPos;
    try {
      curChar = input_stream.readChar();
    } catch (java.io.IOException e) {
      return curPos;
    }
  }
}

static final int[] jjnextStates = { 1, 2, 4, 8,
    10, 9, 28, 33, 29, 30, 32, 36, 38, 40, 41,
    40, 41, 42, 5, 6, 8, 9, };
public static final String[] jjstrLiteralImages = {
    "", null, null, null, null, null, null,
    null, null, null, null, null, null, null,
    null, null, "\56", "\133", "\135", "\136",
```

```
    "\72\75", "\50", "\51", "\72", "\174",
    "\43\133", "\73", "\43", };
public static final String[] lexStateNames = { "DEFAULT", };
static final long[] jjtoToken = { 0xfff1f81L, };
static final long[] jjtoSkip = { 0x7eL, };
protected SimpleCharStream input_stream;
private final int[] jjrounds = new int[43];
private final int[] jjstateSet = new int[86];
protected char curChar;

public AgoraTokenManager(SimpleCharStream stream) {
  if (SimpleCharStream.staticFlag)
    throw new Error(
        "ERROR: Cannot use a static CharStream class with a non-static lexical an
  input_stream = stream;
}

public AgoraTokenManager(SimpleCharStream stream,
    int lexState) {
  this(stream);
  SwitchTo(lexState);
}

public void ReInit(SimpleCharStream stream) {
  jjmatchedPos = jjnewStateCnt = 0;
  curLexState = defaultLexState;
  input_stream = stream;
  ReInitRounds();
}

private final void ReInitRounds() {
  int i;
  jjround = 0x80000001;
  for (i = 43; i-- > 0;)
    jjrounds[i] = 0x80000000;
```

```
}

public void ReInit(SimpleCharStream stream,
    int lexState) {
  ReInit(stream);
  SwitchTo(lexState);
}

public void SwitchTo(int lexState) {
  if (lexState >= 1 || lexState < 0)
    throw new TokenMgrError(
        "Error: Ignoring invalid lexical state : "
            + lexState
            + ". State unchanged.",
        TokenMgrError.INVALID_LEXICAL_STATE);
  else
    curLexState = lexState;
}

protected Token jjFillToken() {
  Token t = Token.newToken(jjmatchedKind);
  t.kind = jjmatchedKind;
  String im = jjstrLiteralImages[jjmatchedKind];
  t.image = (im == null) ? input_stream
      .GetImage() : im;
  t.beginLine = input_stream.getBeginLine();
  t.beginColumn = input_stream.getBeginColumn();
  t.endLine = input_stream.getEndLine();
  t.endColumn = input_stream.getEndColumn();
  return t;
}

int curLexState = 0;
int defaultLexState = 0;
int jjnewStateCnt;
```

```
int jjround;
int jjmatchedPos;
int jjmatchedKind;

public Token getNextToken() {
  int kind;
  Token specialToken = null;
  Token matchedToken;
  int curPos = 0;

  EOFLoop: for (;;) {
    try {
      curChar = input_stream.BeginToken();
    } catch (java.io.IOException e) {
      jjmatchedKind = 0;
      matchedToken = jjFillToken();
      return matchedToken;
    }

    try {
      input_stream.backup(0);
      while (curChar <= 32
          && (0x100002600L & (1L << curChar)) != 0L)
        curChar = input_stream
            .BeginToken();
    } catch (java.io.IOException e1) {
      continue EOFLoop;
    }
    jjmatchedKind = 0x7fffffff;
    jjmatchedPos = 0;
    curPos = jjMoveStringLiteralDfa0_0();
    if (jjmatchedKind != 0x7fffffff) {
      if (jjmatchedPos + 1 < curPos)
        input_stream.backup(curPos
            - jjmatchedPos - 1);
```

```
  if ((jjtoToken[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
    matchedToken = jjFillToken();
    return matchedToken;
  } else {
    continue EOFLoop;
  }
}
int error_line = input_stream.getEndLine();
int error_column = input_stream
    .getEndColumn();
String error_after = null;
boolean EOFSeen = false;
try {
  input_stream.readChar();
  input_stream.backup(1);
} catch (java.io.IOException e1) {
  EOFSeen = true;
  error_after = curPos <= 1 ? ""
      : input_stream.GetImage();
  if (curChar == '\n' || curChar == '\r') {
    error_line++;
    error_column = 0;
  } else
    error_column++;
}
if (!EOFSeen) {
  input_stream.backup(1);
  error_after = curPos <= 1 ? ""
      : input_stream.GetImage();
}
throw new TokenMgrError(EOFSeen,
    curLexState, error_line,
    error_column, error_after,
    curChar,
    TokenMgrError.LEXICAL_ERROR);
```

```
    }
  }

}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTRetorno

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTRetorno extends SimpleNode {
  public ASTRetorno(int id) {
    super(id);
  }

  public ASTRetorno(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTListaDeAgendas

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTListaDeAgendas extends SimpleNode {
  public ASTListaDeAgendas(int id) {
    super(id);
  }

  public ASTListaDeAgendas(Agora p, int id) {
```

```
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTAtribuicao

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTAtribuicao extends SimpleNode {
  public ASTAtribuicao(int id) {
    super(id);
  }

  public ASTAtribuicao(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public Object visitarExpressao(
      AgoraVisitor visitante, Object data) {
    return jjtGetChild(1).jjtAccept(visitante,
        data);
  }
```

```
  public String obterNomeDaVariavel() {
    return ((ASTIdentificador) jjtGetChild(0))
        .obterValor();
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTSentencas

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTSentencas extends SimpleNode {
  public ASTSentencas(int id) {
    super(id);
  }

  public ASTSentencas(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTPalavraChave

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTPalavraChave extends
    ASTPrimitivaAbstrata {
  public ASTPalavraChave(int id) {
    super(id);
  }
```

```
  public ASTPalavraChave(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTMensagemEmCascata

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTMensagemEmCascata extends SimpleNode {
  public ASTMensagemEmCascata(int id) {
    super(id);
  }

  public ASTMensagemEmCascata(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoPorPalavraChave

```
package br.ufsc.edugraf.agora.parser.javacc;
```

```
import java.util.ArrayList;

public class ASTDeclaracaoPorPalavraChave extends
    SimpleNode {
  public ASTDeclaracaoPorPalavraChave(int id) {
    super(id);
  }

  public ASTDeclaracaoPorPalavraChave(Agora p, int id) {
    super(p, id);
  }

  /** Accept the visitor. * */
  public Object jjtAccept(AgoraVisitor visitor,
      Object data) {
    return visitor.visit(this, data);
  }

  public String obterNomeDaMensagem() {
    StringBuilder nome = new StringBuilder();
    for (int i = 0; i < jjtGetNumChildren() - 1; i = i + 2) {
      String palavraChave = ((ASTPalavraChave) jjtGetChild(i))
          .obterValor();
      nome.append(palavraChave);
    }
    return nome.toString();
  }

  public ArrayList<String> obterNomeDosParametros() {
    ArrayList<String> nomes = new ArrayList<String>();
    for (int i = 1; i < jjtGetNumChildren() - 1; i = i + 2) {
      String palavraChave = ((ASTIdentificador) jjtGetChild(i))
          .obterValor();
      nomes.add(palavraChave);
```

```
    }
    return nomes;
  }

  public ASTSentencas obterDefinicao() {
    return (ASTSentencas) jjtGetChild(jjtGetNumChildren() - 1);
  }

  public Object aceitarVisitanteNasSentencas(
      AgoraVisitor visitor, Object data) {
    return obterDefinicao().jjtAccept(visitor,
        data);
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.AgoraVisitor

```
package br.ufsc.edugraf.agora.parser.javacc;

public interface AgoraVisitor {
  public Object visit(SimpleNode node, Object data);

  public Object visit(ASTAgora node, Object data);

  public Object visit(ASTSentencas node, Object data);

  public Object visit(
      ASTDeclaracaoDeModeloOuMolde node,
      Object data);

  public Object visit(ASTListaDeAgendas node,
      Object data);

  public Object visit(ASTDeclaracaoUnaria node,
      Object data);
```

```
public Object visit(ASTDeclaracaoBinaria node,
    Object data);

public Object visit(
    ASTDeclaracaoPorPalavraChave node,
    Object data);

public Object visit(ASTRetorno node, Object data);

public Object visit(ASTExpressao node, Object data);

public Object visit(ASTAtribuicao node, Object data);

public Object visit(ASTBloco node, Object data);

public Object visit(ASTParametros node, Object data);

public Object visit(ASTLista node, Object data);

public Object visit(ASTMensagem node, Object data);

public Object visit(ASTMensagemUnaria node,
    Object data);

public Object visit(ASTMensagemBinaria node,
    Object data);

public Object visit(
    ASTMensagemPorPalavraChave node,
    Object data);

public Object visit(ASTMensagemEmCascata node,
    Object data);
```

```
  public Object visit(ASTIdentificador node,
      Object data);

  public Object visit(ASTNumero node, Object data);

  public Object visit(ASTBooleano node, Object data);

  public Object visit(ASTString node, Object data);

  public Object visit(ASTSimbolo node, Object data);

  public Object visit(ASTSeletor node, Object data);

  public Object visit(ASTPalavraChave node,
      Object data);
}
```

Classe: br.ufsc.edugraf.agora.parser.javacc.ASTDeclaracaoDeModeloOuMolde

```
package br.ufsc.edugraf.agora.parser.javacc;

public class ASTDeclaracaoDeModeloOuMolde extends
    SimpleNode {
  private static final int POSICAO_LISTA_DE_MOLDES = 1;
  private static final int POSICAO_NOME = 0;

  private boolean éModelo;

  public ASTDeclaracaoDeModeloOuMolde(int id) {
    super(id);
  }

  public ASTDeclaracaoDeModeloOuMolde(Agora p, int id) {
    super(p, id);
  }
```

```
/** Accept the visitor. * */
public Object jjtAccept(AgoraVisitor visitor,
    Object data) {
  return visitor.visit(this, data);
}

public String obterNomeDoModelo() {
  return ((ASTSimbolo) jjtGetChild(POSICAO_NOME))
      .obterValor();
}

public boolean possuiListaDeMoldes() {
  return jjtGetNumChildren() == 3;
}

public ASTLista obterListaDeMoldes() {
  if (possuiListaDeMoldes())
    return (ASTLista) jjtGetChild(POSICAO_LISTA_DE_MOLDES);
  return null;
}

public ASTListaDeAgendas obterDefinicaoDeAgendas() {
  int offset = 0;
  if (possuiListaDeMoldes())
    offset = 1;
  return (ASTListaDeAgendas) jjtGetChild(POSICAO_LISTA_DE_MOLDES
      + offset);
}

public void fixarÉModelo(boolean éModelo) {
  this.éModelo = éModelo;
}

public boolean éModelo() {
```

```
    return éModelo;
  }
}
```

Classe: br.ufsc.edugraf.agora.parser.FachadaDoParser

```
package br.ufsc.edugraf.agora.parser;

import java.io.Reader;

import br.ufsc.edugraf.agora.parser.javacc.ASTAgora;
import br.ufsc.edugraf.agora.parser.javacc.Agora;
import br.ufsc.edugraf.agora.parser.javacc.ParseException;

public class FachadaDoParser {

  public ASTAgora parsear(Reader leitor) {
    Agora agora = new Agora(leitor);
    try {
      return agora.Agora();
    } catch (ParseException e) {
      e.printStackTrace();
      throw new RuntimeException(
          "Relançar exceção descente", e);
    }
  }

}
```

Classe: br.ufsc.edugraf.agora.Main

```
package br.ufsc.edugraf.agora;

import java.io.BufferedReader;
import java.io.BufferedWriter;
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.io.Writer;

public class Main {

  public static void main(String[] args) {
    if (args.length != 2)
      gerarErro("Uso: agorac fonte destino.");
    try {
      Reader leitor = new BufferedReader(
          new FileReader(args[0]));
      Writer escritor = new BufferedWriter(
          new FileWriter(args[1]));
      CompiladorAgora compilador = new CompiladorAgora();
      compilador.compilar(leitor, escritor);
    } catch (FileNotFoundException e) {
      System.out
          .println("Arquivo de entrada não encontrado");
    } catch (IOException e) {
      System.out.println("Erro de I/O.");
    } catch (Exception e) {
      System.out.println("Erro: "
          + e.getMessage());
    }
  }

  private static void gerarErro(String mensagem) {
    System.out.println(mensagem);
    System.exit(0);
  }
```

```
}
```

Classe: br.ufsc.edugraf.agora.CompiladorAgora

```
package br.ufsc.edugraf.agora;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.io.StringWriter;
import java.io.Writer;

import br.ufsc.edugraf.agora.geracaoDeCodigo.FachadaDaGeracaoDeCodigo;
import br.ufsc.edugraf.agora.parser.FachadaDoParser;
import br.ufsc.edugraf.agora.parser.javacc.ASTAgora;
import br.ufsc.edugraf.agora.publicacao.Publicador;

public class CompiladorAgora {

  public void compilar(Reader entrada, Writer saída) {
    try {
      FachadaDoParser parser = new FachadaDoParser();
      ASTAgora ast = parser
          .parsear(new BufferedReader(
              entrada));

      StringWriter saídaTemporaria = new StringWriter();
      FachadaDaGeracaoDeCodigo geracao = new FachadaDaGeracaoDeCodigo();
      geracao.gerarCodigo(ast, saídaTemporaria);

      Publicador publicador = new Publicador(
          saídaTemporaria.toString(), saída);
      publicador.publicar();

    } finally {
```

```
      try {
        entrada.close();
        saída.close();
      } catch (IOException e) {
      }
    }
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErro

```
package br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public class ReportadorDeErros {

  private static EstrategiaParaReportarErros estrategia = new EstrategiaDeReportage

  public static void reportarErro(ExcecaoTelis e) {
    estrategia.reportarErro(e);
  }

  public static void fixarEstrategiaParaReportarErros(
      EstrategiaParaReportarErros e) {
    estrategia = e;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeRepo

```
package br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem;
```

```
import java.io.OutputStream;
import java.io.PrintStream;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public class EstrategiaDeReportagemParaStream
    implements EstrategiaParaReportarErros {

  private final PrintStream os;

  public EstrategiaDeReportagemParaStream(
      OutputStream os) {
    this.os = new PrintStream(os);
  }

  public EstrategiaDeReportagemParaStream(
      PrintStream ps) {
    this.os = ps;
  }

  public void reportarErro(ExcecaoTelis e) {
    os.println(e.obterMensagemComStackTrace());
    os.flush();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeRepo

```
package br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public class EstrategiaDeReportagemNula implements
    EstrategiaParaReportarErros {
```

```
  public void reportarErro(ExcecaoTelis e) {
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeRepo

```
package br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem;

import java.util.ArrayList;
import java.util.List;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public class EstrategiaDeReportagemDeBufferizacao
    implements EstrategiaParaReportarErros {

  private final List<ExcecaoTelis> exceções = new ArrayList<ExcecaoTelis>();

  public void reportarErro(ExcecaoTelis e) {
    exceções.add(e);
  }

  public boolean ocorreuErro() {
    return !exceções.isEmpty();
  }

  public void limparExceções() {
    exceções.clear();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaParaRep

```
package br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public interface EstrategiaParaReportarErros {

  void reportarErro(ExcecaoTelis e);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis

```
package br.ufsc.edugraf.maquinaTelis.excecoes;

import br.ufsc.edugraf.telis.Resolvivel;

public class ExcecaoTelis extends RuntimeException {

  private static final long serialVersionUID = 84738469824782073 45L;

  private StringBuilder stackTrace = new StringBuilder();

  private Resolvivel instruçãoCausadoraDeErro;

  public ExcecaoTelis() {
    super("Erro durante execução.");
  }

  public ExcecaoTelis(String message, Throwable cause) {
    super(message, cause);
  }

  public ExcecaoTelis(String message) {
    super(message);
  }
```

```
public ExcecaoTelis(Throwable cause) {
  super(cause.getMessage(), cause);
}

public void notificarQuePassouPelaAgenda(
    String nomeDaAgenda, String nomeDoModelo) {
  stackTrace.append("\n\t");
  stackTrace.append("em: " + nomeDoModelo + "."
      + nomeDaAgenda);
}

public String obterMensagemComStackTrace() {
  String mensagem = super.getMessage() + "\n";
  if (instruçãoCausadoraDeErro != null)
    mensagem += "Causada por: "
        + instruçãoCausadoraDeErro + "\n";
  if (stackTrace.length() == 0)
    return mensagem;
  return mensagem + "Pilha de rastreamento:"
      + stackTrace;
}

public String obterStackTrace() {
  return stackTrace.toString();
}

public void fixarInstruçãoCausadora(Resolvivel r) {
  if (instruçãoCausadoraDeErro == null)
    this.instruçãoCausadoraDeErro = r;
}

public Resolvivel obterInstruçãoCausadora() {
  return instruçãoCausadoraDeErro;
}
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTipoErrado

```
package br.ufsc.edugraf.maquinaTelis.excecoes;

public class ExcecaoTipoErrado extends ExcecaoTelis {

  private static final long serialVersionUID = 6296959512149661763L;

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoPalavraNaoEncontrada

```
package br.ufsc.edugraf.maquinaTelis.excecoes;

import br.ufsc.edugraf.telis.Resolvivel;

public class ExcecaoPalavraNaoEncontrada extends
    ExcecaoTelis {

  private static final long serialVersionUID = -6756266925533246930L;

  public ExcecaoPalavraNaoEncontrada(
      Resolvivel itemNaoResolvido) {
    super("Palavra \"" + itemNaoResolvido
        + "\" não pode ser resolvida.");
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoPilhaVazia

```
package br.ufsc.edugraf.maquinaTelis.excecoes;
```

```
public class ExcecaoPilhaVazia extends ExcecaoTelis {

  private static final long serialVersionUID = 4639685120438232538L;

  public ExcecaoPilhaVazia() {
    super("Pilha está vazia.");
  }

}
```

Classe: *.Dormir

```
 *
 */
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.MaquinaDeEstimulosDeAtor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

@Primitiva(nome = "dormir", parametros = { Numero.class })
public class Dormir implements Executavel {
  private final MaquinaDeEstimulosDeAtor maquinaDeEstimulos;

  public Dormir(
      MaquinaDeEstimulosDeAtor maquinaDeEstimulos) {
    this.maquinaDeEstimulos = maquinaDeEstimulos;
  }

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
```

```
    double tempoDeDescanso = pilha
        .desempilharComoNúmero().obterValor();
    maquinaDeEstimulos
        .bloquearRecepçãoDeEstimulos();
    try {
      Thread
          .sleep((long) (tempoDeDescanso * 1000));
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
    maquinaDeEstimulos
        .liberarRecepçãoDeEstimulos();
  }
}
```

Classe: *.Descansar

```
 *
 */
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.telis.palavras.Numero;

@Primitiva(nome = "descansar", parametros = { Numero.class })
public class Descansar implements Executavel {
  private final MaquinaDePilha maquina;

  public Descansar(MaquinaDePilha maquina) {
    this.maquina = maquina;
  }
```

```
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    double tempoDeDescanso = pilha
        .desempilharComoNúmero().obterValor();
    double horaParaAcordar = System
        .currentTimeMillis()
        + tempoDeDescanso * 1000;
    try {
      while (horaParaAcordar > System
          .currentTimeMillis()) {
        Thread.sleep(10);
        maquina.tratarInterrupções();
      }
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
  }
}
```

Classe: *.Dizer

```
 *
 */
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.IMaquinaDeEstimulos;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "dizer", parametros = { Lista.class })
public class Dizer implements Executavel {
  IMaquinaDeEstimulos maquinaDeEstimulos;
```

```
  public Dizer(IMaquinaDeEstimulos maquinaDeEstimulos) {
    this.maquinaDeEstimulos = maquinaDeEstimulos;
  }

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Lista<Palavra> tupla = pilha
        .desempilharComoLista();
    maquinaDeEstimulos.dizer(tupla);
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.ComunicacaoDiret

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.IMaquinaDeEstimulos;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.ChamadaDeAgenda
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class ComunicacaoDireta implements Executavel {
  private final Simbolo nomeDaAgenda;
  private final UnidadeDeExecucao unidade;
  private final IMaquinaDeEstimulos maquinaDeEstimulos;

  public ComunicacaoDireta(Simbolo nomeDaAgenda,
      UnidadeDeExecucao unidade,
      IMaquinaDeEstimulos maquina) {
```

```
  this.nomeDaAgenda = nomeDaAgenda;
  this.unidade = unidade;
  maquinaDeEstimulos = maquina;
}

public void executar(IPilha pilha,
    ControladorDeExecucaoDaMaquina controlador) {
  Referencia referencia = pilha
      .desempilharComoReferencia();
  if (referencia.obterUnidadeReferenciada()
      .equals(unidade)) {
    IAgenda agenda = unidade.obterModelo()
        .obterAgendaEstendida(
            nomeDaAgenda.obterValor());
    if (agenda == null) {
      String nomeDoModelo = referencia
          .obterUnidadeReferenciada()
          .obterModelo().obterNome();
      throw new ExcecaoTelis("Agenda "
          + nomeDaAgenda
          + " inexistente no modelo "
          + nomeDoModelo + ".");
    }
    ChamadaDeAgenda chamadaDeAgenda = new ChamadaDeAgenda(
        agenda);
    chamadaDeAgenda.executar(pilha,
        controlador);
  } else {
    UnidadeDeExecucao unidadeDeExecução = referencia
        .obterUnidadeReferenciada();
    if (unidadeDeExecução == null)
      throw new ExcecaoTelis("Ator "
          + referencia + " inexistente.");

    maquinaDeEstimulos.comunicarDiretamente(
```

```
          unidadeDeExecução, nomeDaAgenda
              .obterValor(), pilha);
    }
  }
}
```

Classe: *.SeDito

```
 *
 */
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.IMaquinaDeEstimulos;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Filtro;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "seDito", parametros = {
    Lista.class, Closure.class })
public class SeDito implements Executavel {
  private final IMaquinaDeEstimulos maquinaDeEstimulos;

  public SeDito(
      IMaquinaDeEstimulos maquinaDeEstimulos) {
    this.maquinaDeEstimulos = maquinaDeEstimulos;
  }

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure closure = pilha.desempilharClosure();
    Lista<Palavra> tupla = pilha
```

```
        .desempilharComoLista();
    maquinaDeEstimulos.instalarTratadorDizer(
        new Filtro<Palavra>(tupla), closure);
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.InstaladorDePrimitivasDeEsti

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.ComunicacaoDiret
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Descansar;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Dizer;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Dormir;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.SeDito;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.EventosDeCicloDeVidaDoA
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.util.Observador;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.OperadorDeComunicacaoDireta;

public class InstaladorDePrimitivasDeEstimuloParaAtor
    implements InstalavelEmResolvedorPorCastas {

  private MaquinaDeEstimulosDeAtor maquinaDeEstimulos;
  private MaquinaDePilha maquina;
  private Ator ator;

  public InstaladorDePrimitivasDeEstimuloParaAtor(
      MaquinaDePilha maquina, Ator ator) {
    this.maquina = maquina;
```

```
  this.ator = ator;
  associarComAtorEMaquina();
}

private void associarComAtorEMaquina() {
  ator
      .cadastrarObservador(new Observador<EventosDeCicloDeVidaDoAtor>() {
        public void notificar(
            EventosDeCicloDeVidaDoAtor evento) {
          if (evento == EventosDeCicloDeVidaDoAtor.HIBERNOU)
            maquinaDeEstimulos
                .liberarRecepçãoDeComunicaçãoDireta();
          else if (evento == EventosDeCicloDeVidaDoAtor.SUICIDOU) {
            maquinaDeEstimulos
                .prepararParaDestruição();
          }
        }
      });

  maquinaDeEstimulos = MediadorDeMaquinasDeEstimulos
      .obterInstancia()
      .criarMaquinaDeRecepcaoDeEstimulos(
          ator, maquina);
  maquinaDeEstimulos
      .bloquearRecepçãoDeComunicaçãoDireta();
}

public void instalar(ResolvedorPorCasta resolvedor) {
  resolvedor.associarPrimitiva(new Dizer(
      maquinaDeEstimulos));
  resolvedor.associarPrimitiva(new SeDito(
      maquinaDeEstimulos));
  resolvedor.associarPrimitiva(new Dormir(
      maquinaDeEstimulos));
  resolvedor.associarPrimitiva(new Descansar(
```

```
        maquina));
    Resolvedor resolvedorDeComunicacaoDireta = new Resolvedor() {
      public Executavel resolver(Resolvivel r) {
        if (r instanceof OperadorDeComunicacaoDireta) {
          OperadorDeComunicacaoDireta operador = (OperadorDeComunicacaoDireta) r;
          return new ComunicacaoDireta(
              operador.obterSimbolo(),
              ator, maquinaDeEstimulos);
        }
        return null;
      }
    };
    resolvedor
        .adicionarResolvedorGenérico(resolvedorDeComunicacaoDireta);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.MaquinaDeEstimulosDeObje

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import java.util.ArrayList;
import java.util.EnumSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Agenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.PedidoDeInterrupcao;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Pilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamentoDeI
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Filtro;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MaquinaDeEstimulosDeObjetos
    extends
    MaquinaDeEstimulosAbstrata<MaquinaDePilhaSimples> {

  public MaquinaDeEstimulosDeObjetos(
      MaquinaDePilhaSimples maquinaDePilha) {
    super(maquinaDePilha);
  }

  public void instalarTratadorDizer(
      Filtro<? extends Palavra> filtro,
      Closure closure) {
    throw new UnsupportedOperationException(
        "Objetos não podem instalar tratadores de estímulos.");
  }

  public synchronized void notificarRecepcaoDeEstimuloDizer(
      Lista<? extends Palavra> tupla) {
  }

  public String obterNomeDoModelo() {
    return obterMaquinaDePilha()
        .obterNomeDoModeloAtreladoAMaquina();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.GerenteDeEstados

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import java.util.EnumSet;

public class GerenteDeEstados {

  private EnumSet<EstadosDaMaquinaDeEstimulo> estadosAtuais = EnumSet
      .noneOf(EstadosDaMaquinaDeEstimulo.class);

  protected boolean máquinaEstáAptaATratarComunicaçãoDireta() {
    return estadosAtuais.isEmpty();
  }

  protected void entrarNoEstado(
      EstadosDaMaquinaDeEstimulo novoEstado) {
    estadosAtuais.add(novoEstado);
  }

  protected void sairDoEstado(
      EstadosDaMaquinaDeEstimulo estado) {
    boolean removeu = estadosAtuais.remove(estado);
    assert removeu : "Ator não está no estado "
        + estado
        + ", logo não pode ter sair deste.";
  }

  protected boolean estáNosEstados(
      EstadosDaMaquinaDeEstimulo... estados) {
    for (EstadosDaMaquinaDeEstimulo e : estados) {
      if (!estadosAtuais.contains(e))
        return false;
    }
    return true;
  }
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.IMaquinaDeEstimulos

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Filtro;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public interface IMaquinaDeEstimulos {

  public abstract void dizer(
      Lista<? extends Palavra> tupla);

  public abstract void comunicarDiretamente(
      UnidadeDeExecucao unidade,
      String nomeDaAgenda, IPilha pilha);

  public abstract void instalarTratadorDizer(
      Filtro<? extends Palavra> filtro,
      Closure closure);

  public abstract void notificarRecepcaoDeEstimuloDizer(
      Lista<? extends Palavra> tupla);

  public abstract void notificarRecepcaoDeEstimuloComunicaçãoDireta(
      PedidoDeComunicacaoDireta pedido);

  public abstract String obterNomeDoModelo();
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.TerminoDeTratamentoDeDiz

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErros;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamentoDeI

class TerminoDeTratamentoDeDizerCallback implements
    TratadorDeEventosDoTratamentoDeInterrupcao {

  private final MaquinaDeEstimulosDeAtor maquinaDeEstimulo;

  public TerminoDeTratamentoDeDizerCallback(
      MaquinaDeEstimulosDeAtor maquinaDeEstimulo) {
    this.maquinaDeEstimulo = maquinaDeEstimulo;
  }

  public void aoTerminarDeTratarInterrupcao() {
    maquinaDeEstimulo.aoTerminarTratamentoDizer();
  }

  public void aoTerminarDevidoAExcecao(ExcecaoTelis e) {
    e.notificarQuePassouPelaAgenda(
        "<tratador seDito>", maquinaDeEstimulo
            .obterNomeDoModelo());
    ReportadorDeErros.reportarErro(e);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.MaquinaDeEstimulosDeObje

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;
```

```
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MaquinaDeEstimulosDeObjetosClasse extends
    MaquinaDeEstimulosDeObjetos {

  public MaquinaDeEstimulosDeObjetosClasse(
      MaquinaDePilhaSimples maquinaDePilha) {
    super(maquinaDePilha);
  }

  protected void realizarComunicaçãoDireta(
      PedidoDeComunicacaoDireta pedido) {
    String nomeAgenda = pedido.obterNomeDaAgenda();
    Modelo modelo = pedido.obterModelo();

    if ("novo".equals(nomeAgenda)) {
      invocarAgendaNovo(pedido);
    } else {
      TerminoDeTratamentoDeComunicacaoDiretaCallback callbackDeTerminoDeComunicação
          this, pedido);
      IAgenda agenda = modelo
          .obterAgendaDeMetaclasse(nomeAgenda);
      pedido.verificarValidadeDaAgenda(agenda);
      agendarInterrupçãoNaMaquinaDePilha(agenda,
          pedido.obterPilha(),
          callbackDeTerminoDeComunicação);
    }
```

```
  }

  private void invocarAgendaNovo(
      PedidoDeComunicacaoDireta pedido) {
    assert "novo".equals(pedido
        .obterNomeDaAgenda());
    Modelo modelo = pedido.obterModelo();
    TerminoDeTratamentoDeComunicacaoDiretaCallback callbackDeTerminoDeComunicação =
        this, pedido);
    try {
      Palavra parametro = pedido.obterPilha()
          .desempilhar();
      UnidadeDeExecucao unidade = Aplique
          .obterInstancia().instanciar(
              modelo, parametro);
      pedido.obterPilha().empilhar(
          new Referencia(unidade));
    } catch (ExcecaoTelis e) {
      e.notificarQuePassouPelaAgenda("novo",
          modelo.obterNome());
      throw e;
    }
    IAgenda agenda = modelo
        .obterAgendaDeMetaclasse("novo");
    if (agenda != null) {
      pedido.verificarValidadeDaAgenda(agenda);
      agendarInterrupçãoNaMaquinaDePilha(agenda,
          pedido.obterPilha(),
          callbackDeTerminoDeComunicação);
    }
    callbackDeTerminoDeComunicação
        .aoTerminarDeTratarInterrupcao();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.MaquinaDeEstimulosDeAtor

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.PedidoDeInterrupcao;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Pilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamentoDeI
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Filtro;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MaquinaDeEstimulosDeAtor extends
    MaquinaDeEstimulosAbstrata<MaquinaDePilha> {

  private List<Tratador> tratadores = new ArrayList<Tratador>();
  private Queue<PedidoDeComunicacaoDireta> filaDeComunicaçãoDireta = new LinkedList
  protected GerenteDeEstados gerenteDeEstados = new GerenteDeEstados();

  public MaquinaDeEstimulosDeAtor(
      MaquinaDePilha maquinaDePilha) {
    super(maquinaDePilha);
  }

  @Override
```

```
public void notificarRecepcaoDeEstimuloComunicaçãoDireta(
    PedidoDeComunicacaoDireta pedido) {
  if (máquinaEstáAptaATratarComunicaçãoDireta()) {
    super
        .notificarRecepcaoDeEstimuloComunicaçãoDireta(pedido);
    gerenteDeEstados
        .entrarNoEstado(EstadosDaMaquinaDeEstimulo.TRATANDO_COMUNICAÇÃO_DIRETA);
  } else {
    filaDeComunicaçãoDireta.offer(pedido);
  }
}

public synchronized void instalarTratadorDizer(
    Filtro<? extends Palavra> filtro,
    Closure closure) {
  Tratador tratador = new Tratador(filtro,
      closure);
  tratadores.add(tratador);
}

public synchronized void notificarRecepcaoDeEstimuloDizer(
    Lista<? extends Palavra> tupla) {
  if (máquinaEstáAptaATratarDizer()) {
    for (Tratador tratador : tratadores) {
      if (tratador.casar(tupla)) {
        IPilha pilha = new Pilha();
        pilha.empilhar(tupla);
        agendarInterrupçãoNaMaquinaDePilha(
            tratador, pilha);
        gerenteDeEstados
            .entrarNoEstado(EstadosDaMaquinaDeEstimulo.TRATANDO_DIZER);
      }
    }
  }
}
```

```
public String obterNomeDoModelo() {
  return obterMaquinaDePilha()
      .obterNomeDoModeloAtreladoAMaquina();
}

public void prepararParaDestruição() {
  for (PedidoDeComunicacaoDireta pedido : filaDeComunicaçãoDireta) {
    ExcecaoTelis excecao = new ExcecaoTelis(
        "Ator do modelo \""
            + obterNomeDoModelo()
            + "\" com quem comunicação direta estava sendo feita se suicidou.");
    excecao.fillInStackTrace();
    pedido.notificarOcorrenciaDeErro(excecao);
  }
}

private void agendarInterrupçãoNaMaquinaDePilha(
    Tratador tratador, IPilha pilha) {
  PedidoDeInterrupcao pedidoDeInterrupcao = new PedidoDeInterrupcao(
      tratador.obterClosure(),
      pilha,
      new TerminoDeTratamentoDeDizerCallback(
          this));
  obterMaquinaDePilha().agendarInterrupção(
      pedidoDeInterrupcao);
}

private boolean máquinaEstáAptaATratarDizer() {
  return !gerenteDeEstados
      .estáNosEstados(EstadosDaMaquinaDeEstimulo.TRATANDO_DIZER)
      && !gerenteDeEstados
          .estáNosEstados(EstadosDaMaquinaDeEstimulo.ESTIMULOS_BLOQUEADOS);
}
```

```
private boolean temComunicaçãoDiretaPendente() {
  return !filaDeComunicaçãoDireta.isEmpty();
}

protected boolean máquinaEstáAptaATratarComunicaçãoDireta() {
  return gerenteDeEstados
      .máquinaEstáAptaATratarComunicaçãoDireta();
}

public synchronized void bloquearRecepçãoDeComunicaçãoDireta() {
  gerenteDeEstados
      .entrarNoEstado(EstadosDaMaquinaDeEstimulo.COMUNICAÇÃO_DIRETA_BLOQUEADA);
}

public synchronized void bloquearRecepçãoDeEstimulos() {
  gerenteDeEstados
      .entrarNoEstado(EstadosDaMaquinaDeEstimulo.ESTIMULOS_BLOQUEADOS);
}

public void liberarRecepçãoDeComunicaçãoDireta() {
  sairDoEstado(EstadosDaMaquinaDeEstimulo.COMUNICAÇÃO_DIRETA_BLOQUEADA);
}

public void liberarRecepçãoDeEstimulos() {
  sairDoEstado(EstadosDaMaquinaDeEstimulo.ESTIMULOS_BLOQUEADOS);
}

synchronized void aoTerminarTratamentoDizer() {
  sairDoEstado(EstadosDaMaquinaDeEstimulo.TRATANDO_DIZER);
}

@Override
public void aoTerminarTratamentoDeComunicacaoDireta() {
  sairDoEstado(EstadosDaMaquinaDeEstimulo.TRATANDO_COMUNICAÇÃO_DIRETA);
}
```

```
  private synchronized void sairDoEstado(
      EstadosDaMaquinaDeEstimulo estado) {
    gerenteDeEstados.sairDoEstado(estado);
    if (máquinaEstáAptaATratarComunicaçãoDireta()
        && temComunicaçãoDiretaPendente())
      if (máquinaEstáAptaATratarComunicaçãoDireta()) {
        realizarComunicaçãoDireta(filaDeComunicaçãoDireta
            .poll());
        gerenteDeEstados
            .entrarNoEstado(EstadosDaMaquinaDeEstimulo.TRATANDO_COMUNICAÇÃO_DIRETA)
      }
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.InstaladorDePrimitivasDeEsti

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.ComunicacaoDiret
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Descansar;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Dizer;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Dormir;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.SeDito;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.EventosDeCicloDeVidaDoA
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.util.Observador;
```

```
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.OperadorDeComunicacaoDireta;

public class InstaladorDePrimitivasDeEstimuloParaObjeto
    implements InstalavelEmResolvedorPorCastas {

  private MaquinaDeEstimulosDeObjetos maquinaDeEstimulos;
  private Objeto ator;

  public InstaladorDePrimitivasDeEstimuloParaObjeto(
      MaquinaDePilhaSimples maquina,
      Objeto objeto) {
    this.ator = objeto;
    this.maquinaDeEstimulos = MediadorDeMaquinasDeEstimulos
        .obterInstancia()
        .criarMaquinaDeRecepcaoDeEstimulosDeObjetos(
            objeto, maquina);
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new Dizer(
        maquinaDeEstimulos));
    resolvedor.associarPrimitiva(new SeDito(
        maquinaDeEstimulos));
    Resolvedor resolvedorDeComunicacaoDireta = new Resolvedor() {
      public Executavel resolver(Resolvivel r) {
        if (r instanceof OperadorDeComunicacaoDireta) {
          OperadorDeComunicacaoDireta operador = (OperadorDeComunicacaoDireta) r;
          return new ComunicacaoDireta(
              operador.obterSimbolo(),
              ator, maquinaDeEstimulos);
        }
        return null;
      }
    };
```

```
    resolvedor
        .adicionarResolvedorGenérico(resolvedorDeComunicacaoDireta);
    ator = null;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.EstadosDaMaquinaDeEstimu

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

enum EstadosDaMaquinaDeEstimulo {

  TRATANDO_COMUNICAÇÃO_DIRETA, TRATANDO_DIZER, COMUNICAÇÃO_DIRETA_BLOQUEADA, ESTIMU

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.MaquinaDeEstimulosAbstrata

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IMaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.PedidoDeInterrupcao;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamentoDeI
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public abstract class MaquinaDeEstimulosAbstrata<Maquina extends IMaquinaDePilha>
    implements IMaquinaDeEstimulos {

  private final Maquina maquinaDePilha;
```

```
public MaquinaDeEstimulosAbstrata(
    Maquina maquinaDePilha) {
  this.maquinaDePilha = maquinaDePilha;
}

protected Maquina obterMaquinaDePilha() {
  return maquinaDePilha;
}

public abstract void notificarRecepcaoDeEstimuloDizer(
    Lista<? extends Palavra> tupla);

public void notificarRecepcaoDeEstimuloComunicaçãoDireta(
    PedidoDeComunicacaoDireta pedido) {
  realizarComunicaçãoDireta(pedido);
}

protected void agendarInterrupçãoNaMaquinaDePilha(
    IAgenda agenda,
    IPilha pilha,
    TratadorDeEventosDoTratamentoDeInterrupcao aoCompletar) {
  Closure closure = obterMaquinaDePilha()
      .criarClosureComEscopoRaiz(
          agenda.obterCódigo());
  PedidoDeInterrupcao pedidoDeInterrupcao = new PedidoDeInterrupcao(
      closure, pilha, aoCompletar);
  obterMaquinaDePilha().agendarInterrupção(
      pedidoDeInterrupcao);
}

public void dizer(Lista<? extends Palavra> tupla) {
  MediadorDeMaquinasDeEstimulos.obterInstancia()
      .realizarBroadcast(null, tupla);
}
```

```
  public void comunicarDiretamente(
      UnidadeDeExecucao unidade,
      String nomeDaAgenda, IPilha pilha) {
    MediadorDeMaquinasDeEstimulos.obterInstancia()
        .fazerComunicaçãoDireta(unidade,
            nomeDaAgenda, pilha);
  }

  protected void realizarComunicaçãoDireta(
      PedidoDeComunicacaoDireta pedido) {
    agendarInterrupçãoNaMaquinaDePilha(
        pedido.obterAgenda(),
        pedido.obterPilha(),
        new TerminoDeTratamentoDeComunicacaoDiretaCallback(
            this, pedido));
  }

  public void aoTerminarTratamentoDeComunicacaoDireta() {
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.TerminoDeTratamentoDeCon

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamentoDeI

class TerminoDeTratamentoDeComunicacaoDiretaCallback
    implements
    TratadorDeEventosDoTratamentoDeInterrupcao {

  private final MaquinaDeEstimulosAbstrata maquinaDeEstimulo;
```

```
  private final PedidoDeComunicacaoDireta pedido;

  public TerminoDeTratamentoDeComunicacaoDiretaCallback(
      MaquinaDeEstimulosAbstrata maquinaDeEstimulosAbstrata,
      PedidoDeComunicacaoDireta pedido) {
    this.maquinaDeEstimulo = maquinaDeEstimulosAbstrata;
    this.pedido = pedido;
  }

  public void aoTerminarDeTratarInterrupcao() {
    pedido.notificarTerminoDeTratamento();
    maquinaDeEstimulo
        .aoTerminarTratamentoDeComunicacaoDireta();
  }

  public void aoTerminarDevidoAExcecao(ExcecaoTelis e) {
    pedido.notificarOcorrenciaDeErro(e);
    maquinaDeEstimulo
        .aoTerminarTratamentoDeComunicacaoDireta();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.MediadorDeMaquinasDeEsti

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import java.util.HashMap;
import java.util.Map;
import java.util.WeakHashMap;
import java.util.concurrent.Semaphore;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
```

```
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Molde;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.util.Visitante;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MediadorDeMaquinasDeEstimulos {

  private static MediadorDeMaquinasDeEstimulos instancia = new MediadorDeMaquinasDe

  public static MediadorDeMaquinasDeEstimulos obterInstancia() {
    return instancia;
  }

  public static void limpar() {
    instancia = new MediadorDeMaquinasDeEstimulos();
  }

  public MaquinaDeEstimulosDeAtor criarMaquinaDeRecepcaoDeEstimulos(
      Ator ator, MaquinaDePilha maquinaDePilha) {
    MaquinaDeEstimulosDeAtor maquinaDeEstimulos = new MaquinaDeEstimulosDeAtor(
        maquinaDePilha);
    ator.cadastrarExtensão(
        IMaquinaDeEstimulos.class,
        maquinaDeEstimulos);
    return maquinaDeEstimulos;
  }
```

```
public MaquinaDeEstimulosDeObjetos criarMaquinaDeRecepcaoDeEstimulosDeObjetos(
    Objeto objeto,
    MaquinaDePilhaSimples maquinaDePilha) {
  MaquinaDeEstimulosDeObjetos maquinaDeEstimulos = new MaquinaDeEstimulosDeObjeto
      maquinaDePilha);
  objeto.cadastrarExtensão(
      IMaquinaDeEstimulos.class,
      maquinaDeEstimulos);
  return maquinaDeEstimulos;
}

public MaquinaDeEstimulosDeObjetosClasse criarMaquinaDeRecepcaoDeEstimulosDeObjet
    ObjetoClasse objeto,
    MaquinaDePilhaSimples maquina) {
  MaquinaDeEstimulosDeObjetosClasse maquinaDeEstimulos = new MaquinaDeEstimulosDe
      maquina);
  objeto.cadastrarExtensão(
      IMaquinaDeEstimulos.class,
      maquinaDeEstimulos);
  return maquinaDeEstimulos;
}

public void realizarBroadcast(
    Lista<Palavra> estimulo) {
  realizarBroadcast(null, estimulo);
}

public void realizarBroadcast(
    final IMaquinaDeEstimulos origem,
    final Lista<? extends Palavra> estimulo) {
  Aplique aplique = Aplique.obterInstancia();
  aplique
      .receberVisitante(new Visitante<Ator>() {
        public void visitar(Ator dado) {
          IMaquinaDeEstimulos destino = dado
```

```
              .obterExtensão(IMaquinaDeEstimulos.class);
          if (destino != origem)
            destino
                .notificarRecepcaoDeEstimuloDizer(estimulo);
        }
      });
}

public void fazerComunicaçãoDireta(
    UnidadeDeExecucao unidade,
    String nomeDaAgenda, IPilha pilha) {
  IMaquinaDeEstimulos maquinaDeEstimulos = unidade
      .obterExtensão(IMaquinaDeEstimulos.class);
  assegurarQueNãoÉNulo(maquinaDeEstimulos);
  Semaphore semaforo = new Semaphore(0);
  PedidoDeComunicacaoDireta pedido = new PedidoDeComunicacaoDireta(
      nomeDaAgenda, unidade, pilha, semaforo);
  maquinaDeEstimulos
      .notificarRecepcaoDeEstimuloComunicaçãoDireta(pedido);
  try {
    semaforo.acquire(); // TODO: estabelecer time-out (via tryAcquire)
    ExcecaoTelis erro = pedido
        .obterErroOcorrido();
    if (erro != null) {
      erro.notificarQuePassouPelaAgenda(
          nomeDaAgenda,
          maquinaDeEstimulos
              .obterNomeDoModelo());
      throw erro;
    }
  } catch (InterruptedException e) {
    Thread.currentThread().interrupt();
  }
}
```

```
  private void assegurarQueNãoÉNulo(
      IMaquinaDeEstimulos maquinaDeEstimulos) {
    if (maquinaDeEstimulos == null)
      throw new ExcecaoTelis(
          "Ator/objeto com quem se está tentando fazer comunicação direta não foi e
  }

  boolean possuiMaquinaDeEstimulosAssociada(
      UnidadeDeExecucao unidade) {
    return unidade
        .obterExtensão(IMaquinaDeEstimulos.class) != null;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.PedidoDeComunicacaoDireta

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import java.util.concurrent.Semaphore;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Agenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ProxyDaPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

class PedidoDeComunicacaoDireta {

  private final String nomeDaAgenda;
  private final UnidadeDeExecucao unidade;
  private final ProxyDaPilha pilha;
```

```
private final Semaphore semaforo;
private ExcecaoTelis erro;

public PedidoDeComunicacaoDireta(
    String nomeDaAgenda,
    UnidadeDeExecucao unidade, IPilha pilha,
    Semaphore semaforo) {
  this.nomeDaAgenda = nomeDaAgenda;
  this.unidade = unidade;
  this.pilha = new ProxyDaPilha(pilha, 1);
  this.semaforo = semaforo;
}

public IAgenda obterAgenda() {
  IAgenda agenda = obterModelo()
      .obterAgendaEstendida(nomeDaAgenda);
  verificarValidadeDaAgenda(agenda);
  return agenda;
}

public Modelo obterModelo() {
  return unidade.obterModelo();
}

public void verificarValidadeDaAgenda(
    IAgenda agenda) {
  if (agenda == null)
    throw new ExcecaoTelis("Agenda "
        + nomeDaAgenda
        + " inexistente no modelo \""
        + unidade.obterModelo()
            .obterNome() + "\".");
  if (agenda.éPrivada())
    throw new ExcecaoTelis("Agenda \""
        + agenda.obterNome()
```

```
          + "\" é privada.");
  }

  public IPilha obterPilha() {
    return pilha;
  }

  public void notificarTerminoDeTratamento() {
    Palavra retorno = null;
    if (!pilha.estaVazia())
      retorno = pilha.desempilhar();
    pilha.limparPilha();
    if (retorno != null)
      pilha.empilhar(retorno);
    semaforo.release(); // IMPORTANTE: pilha deve ser limpar e retorno
    // empilhado *antes* do semaforo ser liberado
  }

  public void notificarOcorrenciaDeErro(
      ExcecaoTelis e) {
    pilha.limparPilha();
    this.erro = e;
    semaforo.release(); // IMPORTANTE: pilha deve ser limpar e retorno
    // empilhado *antes* do semaforo ser liberado
  }

  public ExcecaoTelis obterErroOcorrido() {
    return erro;
  }

  public String obterNomeDaAgenda() {
    return nomeDaAgenda;
  }

}
```

Classe: *.Tratador

```
 *
 */
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;

import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Filtro;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

class Tratador {
  private Filtro<? extends Palavra> filtro;
  private Closure closure;

  public Tratador(Filtro<? extends Palavra> filtro,
      Closure closure) {
    this.filtro = filtro;
    this.closure = closure;
  }

  public boolean casar(
      Lista<? extends Palavra> estimulo) {
    return filtro.casar(estimulo);
  }

  public Closure obterClosure() {
    return closure;
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.InstaladorDePrimitivasDeEsti

```
package br.ufsc.edugraf.maquinaTelis.linguagem.estimulos;
```

```
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.ComunicacaoDiret
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.Dizer;
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.primitivas.SeDito;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.OperadorDeComunicacaoDireta;

public class InstaladorDePrimitivasDeEstimuloParaObjetoClasse {

  private MaquinaDeEstimulosDeObjetos maquinaDeEstimulos;
  private ObjetoClasse objeto;

  public InstaladorDePrimitivasDeEstimuloParaObjetoClasse(
      MaquinaDePilhaSimples maquina,
      ObjetoClasse objeto) {
    this.objeto = objeto;
    this.maquinaDeEstimulos = MediadorDeMaquinasDeEstimulos
        .obterInstancia()
        .criarMaquinaDeRecepcaoDeEstimulosDeObjetosClasse(
            objeto, maquina);
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new Dizer(
        maquinaDeEstimulos));
    resolvedor.associarPrimitiva(new SeDito(
        maquinaDeEstimulos));
    Resolvedor resolvedorDeComunicacaoDireta = new Resolvedor() {
      public Executavel resolver(Resolvivel r) {
        if (r instanceof OperadorDeComunicacaoDireta) {
```

```
            OperadorDeComunicacaoDireta operador = (OperadorDeComunicacaoDireta) r;
            return new ComunicacaoDireta(
                operador.obterSimbolo(),
                objeto, maquinaDeEstimulos);
          }
          return null;
        }
      };
      resolvedor
          .adicionarResolvedorGenérico(resolvedorDeComunicacaoDireta);
    }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeMo

```
package br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public class InstaladorDeModulosNulo implements
    InstaladorDeModulos {

  public void instalarMódulos(Ator ator,
      MaquinaDePilha maquina,
      ResolvedorPorCasta resolvedor) {

  }

  public void instalarMódulos(Objeto objeto,
```

```
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
  }

  public void instalarMódulos(ObjetoClasse objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeM

```
package br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.InstaladorDePrimitivasDeEst
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.InstaladorDePrimitivasDeEst
import br.ufsc.edugraf.maquinaTelis.linguagem.estimulos.InstaladorDePrimitivasDeEst
import br.ufsc.edugraf.maquinaTelis.linguagem.operadores.ResolvedorDeOperadores;
import br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.Instalado
import br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha.Instala
import br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.Instalado
import br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Insta
import br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDePropriedadesDoAtor.Instal
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.ResolvedorDeAge
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.ResolvedorDeAge
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida.InstaladorD
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida.ResolvedorD
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas.InstaladorDe
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
```

```
public class InstaladorDeModulosBasicos implements
    InstaladorDeModulos {

  public void instalarMódulos(Ator ator,
      MaquinaDePilha maquina,
      ResolvedorPorCasta resolvedor) {
    instalarPrimitivasGerais(resolvedor);

    new InstaladorDePrimitivasDePropriedadeDeUnidadesDeExecucao(
        ator).instalar(resolvedor);
    new InstaladorDePrimitivasDeEstimuloParaAtor(
        maquina, ator).instalar(resolvedor);

    new InstaladorDePrimitivasDeCicloDeVida(ator)
        .instalar(resolvedor);
    new ResolvedorDeAgendas(ator.obterModelo())
        .instalar(resolvedor);
    new ResolvedorDeModelos().instalar(resolvedor);
  }

  public void instalarMódulos(Objeto objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
    instalarPrimitivasGerais(resolvedor);

    new InstaladorDePrimitivasDePropriedadeDeUnidadesDeExecucao(
        objeto).instalar(resolvedor);
    new InstaladorDePrimitivasDeEstimuloParaObjeto(
        maquina, objeto).instalar(resolvedor);

    new ResolvedorDeAgendas(objeto.obterModelo())
        .instalar(resolvedor);
    new ResolvedorDeModelos().instalar(resolvedor);
  }
```

```
  public void instalarMódulos(ObjetoClasse objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
    instalarPrimitivasGerais(resolvedor);

    new InstaladorDePrimitivasDePropriedadeDeUnidadesDeExecucao(
        objeto).instalar(resolvedor);
    new InstaladorDePrimitivasDeEstimuloParaObjetoClasse(
        maquina, objeto).instalar(resolvedor);

    new ResolvedorDeAgendasDeMetaclasse(objeto
        .obterModelo()).instalar(resolvedor);
    new ResolvedorDeModelos().instalar(resolvedor);
  }

  private void instalarPrimitivasGerais(
      ResolvedorPorCasta resolvedor) {
    new ResolvedorDeOperadores()
        .instalar(resolvedor);
    new InstaladorDePrimitivasDeControle()
        .instalar(resolvedor);
    new InstaladorDePrimitivasDeCriacao()
        .instalar(resolvedor);
    new InstaladorDePrimitivasDeManipulacaoDeListas()
        .instalar(resolvedor);
    new InstaladorDePrimitivasDeManipulacaoDePilha()
        .instalar(resolvedor);
    new InstaladorDePrimitivasDeConversaoDeTipo()
        .instalar(resolvedor);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.ConteinerDeIns

```
package br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos;

import java.util.ArrayList;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public class ConteinerDeInstaladoresDeModulos
    implements InstaladorDeModulos {

  private ArrayList<InstaladorDeModulos> listaDeModulos = new ArrayList<InstaladorD

  public void instalarMódulos(Ator ator,
      MaquinaDePilha maquina,
      ResolvedorPorCasta resolvedor) {
    for (InstaladorDeModulos instalador : listaDeModulos)
      instalador.instalarMódulos(ator, maquina,
          resolvedor);
  }

  public void instalarMódulos(Objeto objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
    for (InstaladorDeModulos instalador : listaDeModulos)
      instalador.instalarMódulos(objeto,
          maquina, resolvedor);
  }

  public ConteinerDeInstaladoresDeModulos adicionarInstalador(
      InstaladorDeModulos instalador) {
    listaDeModulos.add(instalador);
```

```
    return this;
  }

  public void instalarMódulos(ObjetoClasse objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
    for (InstaladorDeModulos instalador : listaDeModulos)
      instalador.instalarMódulos(objeto,
          maquina, resolvedor);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeMo

```
package br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public interface InstaladorDeModulos {

  public abstract void instalarMódulos(Ator ator,
      MaquinaDePilha maquina,
      ResolvedorPorCasta resolvedor);

  public abstract void instalarMódulos(
      Objeto objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor);
```

```
  public abstract void instalarMódulos(
      ObjetoClasse objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDePropriedadesDoAtor.Instal

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDePropriedadesDoAtor;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.utilitarios.ResolvedorAssociativoAbst
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.Resolvivel;

public class InstaladorDePrimitivasDePropriedadeDeUnidadesDeExecucao
    extends
    ResolvedorAssociativoAbstrato<Resolvivel>
    implements InstalavelEmResolvedorPorCastas {

  private final UnidadeDeExecucao unidade;

  @Primitiva(nome = "euMesmo")
  private class EuMesmo implements Executavel {
    public void executar(
        IPilha pilha,
        ControladorDeExecucaoDaMaquina controlador) {
      pilha.empilhar(new Referencia(unidade));
```

```
    }
  }

  public InstaladorDePrimitivasDePropriedadeDeUnidadesDeExecucao(
      UnidadeDeExecucao u) {
    this.unidade = u;
  }

  public void instalar(ResolvedorPorCasta r) {
    r.associarPrimitiva(new EuMesmo());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Subtracao

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Subtracao extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.subtrair(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Divisao

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Divisao extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    if (numero2.obterValor() == 0)
      throw new ExcecaoTelis(
          "Não é possível dividir um número por zero.");
    pilha.empilhar(numero1.dividir(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Exponenciacao

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Exponenciacao extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    if (numero1.obterValor() == 0
        && numero2.obterValor() == 0)
      throw new ExcecaoTelis(
          "Não é possível elevar um número a zero.");
    pilha.empilhar(numero1.elevarA(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Maior

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Maior extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.maiorQue(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Modulo

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Modulo extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.modulo(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Igualdade

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import java.util.concurrent.Executor;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class Igualdade implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra palavra1 = pilha.desempilhar();
    Palavra palavra2 = pilha.desempilhar();
    pilha.empilhar(new Booleano(palavra1
        .equals(palavra2)));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Menor

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Menor extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.menorQue(numero2));
  }
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Multiplicacao

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Numero;

public class Multiplicacao extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.multiplicar(numero2));
  }

  public void executar(IPilha pilha,
      Booleano booleano1, Booleano booleano2) {
    pilha.empilhar(booleano1.e(booleano2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.MaiorOuIgual

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class MaiorOuIgual extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
```

```
    pilha.empilhar(numero1.maiorOuIgualA(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Empilhador

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class Empilhador implements Executavel {

  private final Simbolo simbolo;

  public Empilhador(Simbolo simbolo) {
    this.simbolo = simbolo;
  }

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    pilha.empilhar(simbolo);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Soma

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Numero;
```

```
import br.ufsc.edugraf.telis.palavras.Texto;

public class Soma extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Texto texto1,
      Texto texto2) {
    pilha.empilhar(texto1.concatenar(texto2));
  }

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.somar(numero2));
  }

  public void executar(IPilha pilha, Texto texto,
      Numero numero) {
    pilha.empilhar(texto.concatenar(numero
        .toString()));
  }

  public void executar(IPilha pilha, Numero numero,
      Texto texto) {
    pilha.empilhar(texto.prefixar(numero
        .toString()));
  }

  public void executar(IPilha pilha,
      Booleano booleano1, Booleano booleano2) {
    pilha.empilhar(booleano1.ou(booleano2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.ExecutorDeOperacao

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;
```

```
import java.lang.reflect.Method;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTipoErrado;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

// TODO: substituir nome executar por uma anotação
public class ExecutorDeOperacao implements Executavel {
  protected ExecutorDeOperacao() {
  }

  public final void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra palavra2 = pilha.desempilhar();
    Palavra palavra1 = pilha.desempilhar();
    Class<? extends Palavra> tipo1 = palavra1
        .getClass();
    Class<? extends Palavra> tipo2 = palavra2
        .getClass();
    try {
      Method metodo = this.getClass().getMethod(
          "executar", IPilha.class, tipo1,
          tipo2);
      metodo.invoke(this, pilha, palavra1,
          palavra2);
    } catch (NoSuchMethodException e) {
      throw new ExcecaoTipoErrado(); // TODO: colocar maiores informações
      // nessa exceção e reempilhar
    } catch (Exception e) {
      throw new ExcecaoTelis(e);
    }
```

```
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.ResolvedorDeOperadores

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.linguagem.utilitarios.ResolvedorAssociativoAbst
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorComposto;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.Operador;
import br.ufsc.edugraf.telis.palavras.OperadorDeEmpilhamento;

public class ResolvedorDeOperadores extends
    ResolvedorComposto implements
    InstalavelEmResolvedorPorCastas {

  private class ResolvedorDeOperadoresSimples extends
      ResolvedorAssociativoAbstrato<Operador> {
    public ResolvedorDeOperadoresSimples() {
      associar(Operador.SOMA, new Soma());
      associar(Operador.SUBTRACAO,
          new Subtracao());
      associar(Operador.MULTIPLICACAO,
          new Multiplicacao());
      associar(Operador.DIVISAO, new Divisao());
      associar(Operador.MODULO, new Modulo());
      associar(Operador.IGUAL, new Igualdade());
      associar(Operador.DIFERENTE,
```

```
            new Diferente());
        associar(Operador.MENOR, new Menor());
        associar(Operador.MENOR_OU_IGUAL,
            new MenorOuIgual());
        associar(Operador.MAIOR, new Maior());
        associar(Operador.MAIOR_OU_IGUAL,
            new MaiorOuIgual());
        associar(Operador.EXPONENCIACAO,
            new Exponenciacao());
      }
    }

    private class ResolvedorDeOperadorDeEmpilhamento
        implements Resolvedor {
      public Executavel resolver(Resolvivel r) {
        if (r instanceof OperadorDeEmpilhamento) {
          OperadorDeEmpilhamento operadorDeEmpilhamento = (OperadorDeEmpilhamento) r;
          return new Empilhador(
              operadorDeEmpilhamento
                  .obterSimbolo());
        }
        return null;
      }
    }

    public ResolvedorDeOperadores() {
      adicionarResolvedor(new ResolvedorDeOperadoresSimples());
      adicionarResolvedor(new ResolvedorDeOperadorDeEmpilhamento());
    }

    public void instalar(ResolvedorPorCasta resolvedor) {
      resolvedor.adicionarResolvedorGenérico(this);
    }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.MenorOuIgual

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Numero;

public class MenorOuIgual extends ExecutorDeOperacao {

  public void executar(IPilha pilha, Numero numero1,
      Numero numero2) {
    pilha.empilhar(numero1.menorOuIgualA(numero2));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.operadores.Diferente

```
package br.ufsc.edugraf.maquinaTelis.linguagem.operadores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class Diferente implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra palavra1 = pilha.desempilhar();
    Palavra palavra2 = pilha.desempilhar();
    pilha.empilhar(new Booleano(!palavra1
        .equals(palavra2)));
  }
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha.Tirar

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;

@Primitiva(nome = "tirar")
public class Tirar implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    pilha.desempilhar();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha.Copia

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "copiar")
public class Copiar implements Executavel {
```

```
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra palavra = pilha.desempilhar();
    pilha.empilhar(palavra);
    pilha.empilhar(palavra);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha.Instala

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class InstaladorDePrimitivasDeManipulacaoDePilha
    implements InstalavelEmResolvedorPorCastas {

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new Tirar());
    resolvedor.associarPrimitiva(new Copiar());
    resolvedor.associarPrimitiva(new Trocar());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha.Troca

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeManipulacaoDaPilha;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "trocar")
public class Trocar implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    Palavra subtopo = pilha.desempilhar();
    pilha.empilhar(topo);
    pilha.empilhar(subtopo);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Con

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.ListaAbstrata;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

@Primitiva(nome = "concatenar", parametros = { Lista.class })
public class Concatenar implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
```

```
    Palavra resultado;
    if (pilha.topoDaPilhaÉ(Lista.class))
      resultado = concatenarLista(pilha
          .desempilharComoLista(), pilha
          .desempilharComoLista());
    else
      resultado = concatenarTexto(pilha
          .desempilharComoTexto(), pilha
          .desempilharComoTexto());
    pilha.empilhar(resultado);
  }

  private Palavra concatenarTexto(
      Texto segundoTexto, Texto primeiroTexto) {
    return primeiroTexto.concatenar(segundoTexto);
  }

  private Palavra concatenarLista(
      Lista<Palavra> segundaLista,
      Lista<Palavra> primeiraLista) {
    return primeiraLista.concatenar(segundaLista);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Inse

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;
```

```
@Primitiva(nome = "inserirNoInicio", parametros = {
    Lista.class, Palavra.class })
public class InserirNoInicio implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra elemento = pilha.desempilhar();
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    pilha
        .empilhar(lista
            .inserirNoInicio(elemento));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Ulti

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "ultimo", parametros = { Lista.class })
public class Ultimo implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
```

```
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    if (lista.obterTamanho() == 0)
      throw new ExcecaoTelis(
          "Lista não deve ser vazia.");

    Palavra palavra = lista.obterÚltimo();

    pilha.empilhar(palavra);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Gera

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import java.util.Arrays;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "gerarLista", parametros = { Numero.class })
public class GerarLista implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    int tamanho = pilha.desempilharComoNúmero()
```

```
        .obterParteInteira();
    if (tamanho < 0)
      throw new ExcecaoTelis(
          "Não é possível gerar uma lista com tamanho negativo.");

    Palavra resultado[] = new Palavra[tamanho];
    for (int i = tamanho - 1; i >= 0; i--)
      resultado[i] = pilha.desempilhar();
    pilha.empilhar(new Lista<Palavra>(Arrays
        .asList(resultado)));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Sem

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "semPrimeiro", parametros = { Lista.class })
public class SemPrimeiro implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Lista<Palavra> lista = pilha
        .desempilharComoLista();
```

```
    if (lista.estáVazia())
      throw new ExcecaoTelis(
          "Lista não pode estar vazia.");

    pilha.empilhar(lista.removerPrimeiro());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Obte

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "obterElemento", parametros = {
    Lista.class, Numero.class })
public class ObterElemento implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    int posição = pilha.desempilharComoNúmero()
        .obterParteInteira() - 1;
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    if (posição < 0)
      throw new ExcecaoTelis(
```

```
          "Posição deve ser maior ou igual a um.");
    if (posição >= lista.obterTamanho())
      throw new ExcecaoTelis(
          "Posição deve ser menor ou igual ao tamanho da lista");

    Palavra palavra = lista.obterElemento(posição);

    pilha.empilhar(palavra);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Obte

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "obterTamanho", parametros = { Lista.class })
public class ObterTamanho implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    pilha
        .empilhar(new Numero(lista
            .obterTamanho()));
```

```
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Inse

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "inserirNoFim", parametros = {
    Lista.class, Palavra.class })
public class InserirNoFim implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra elemento = pilha.desempilhar();
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    pilha.empilhar(lista.inserirNoFim(elemento));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Inse

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import java.util.List;
```

```
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "inserirElemento", parametros = {
    Lista.class, Palavra.class, Numero.class })
public class InserirElemento implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    int posição = pilha.desempilharComoNúmero()
        .obterParteInteira() - 1;
    Palavra elemento = pilha.desempilhar();
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    if (posição < 0)
      throw new ExcecaoTelis(
          "Posição deve ser maior ou igual a um.");
    if (posição > lista.obterTamanho())
      throw new ExcecaoTelis(
          "Posição deve ser menor ou igual ao tamanho da lista");

    pilha.empilhar(lista.inserirElemento(posição,
        elemento));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Insta

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public class InstaladorDePrimitivasDeManipulacaoDeListas
    implements InstalavelEmResolvedorPorCastas {

  public InstaladorDePrimitivasDeManipulacaoDeListas() {
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new Concatenar());
    resolvedor.associarPrimitiva(new GerarLista());
    resolvedor
        .associarPrimitiva(new InserirElemento());
    resolvedor
        .associarPrimitiva(new InserirNoInicio());
    resolvedor
        .associarPrimitiva(new InserirNoFim());
    resolvedor
        .associarPrimitiva(new ObterElemento());
    resolvedor
        .associarPrimitiva(new ObterTamanho());
    resolvedor.associarPrimitiva(new Primeiro());
    resolvedor.associarPrimitiva(new Ultimo());
    resolvedor
        .associarPrimitiva(new RemoverElemento());
    resolvedor
        .associarPrimitiva(new SemPrimeiro());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Prim

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "primeiro", parametros = { Lista.class })
public class Primeiro implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    if (lista.obterTamanho() == 0)
      throw new ExcecaoTelis(
          "Lista não deve ser vazia.");

    Palavra palavra = lista.obterPrimeiro();

    pilha.empilhar(palavra);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas.Rem

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeManipulacaoDeListas;
```

```
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "removerElemento", parametros = {
    Lista.class, Numero.class })
public class RemoverElemento implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    int posição = pilha.desempilharComoNúmero()
        .obterParteInteira() - 1;
    Lista<Palavra> lista = pilha
        .desempilharComoLista();

    if (posição < 0)
      throw new ExcecaoTelis(
          "Posição deve ser maior ou igual a um.");
    if (posição > lista.obterTamanho())
      throw new ExcecaoTelis(
          "Posição deve ser menor ou igual ao tamanho da lista");

    pilha.empilhar(lista.removerElemento(posição));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.ParaSem

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;
```

```
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;

@Primitiva(nome = "paraSempre", parametros = { Closure.class })
public class ParaSempre implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure comandos = pilha.desempilharClosure();
    while (true) {
      controlador.executarClosure(comandos);
    }
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.EntaoSe

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;

@Primitiva(nome = "entaoSenao", parametros = {
    Booleano.class, Closure.class, Closure.class })
public class EntaoSenao implements Executavel {
```

```
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure comandosRamoSenão = pilha
        .desempilharClosure();
    Closure comandosRamoEntão = pilha
        .desempilharClosure();
    Booleano condicao = pilha
        .desempilharComoBooleano();
    if (condicao.obterValor())
      controlador
          .executarClosure(comandosRamoEntão);
    else
      controlador
          .executarClosure(comandosRamoSenão);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.Instalado

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class InstaladorDePrimitivasDeControle
    implements InstalavelEmResolvedorPorCastas {

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new SeVerdade());
    resolvedor.associarPrimitiva(new SeFalso());
    resolvedor.associarPrimitiva(new EntaoSenao());
```

```
    resolvedor.associarPrimitiva(new Executar());
    resolvedor
        .associarPrimitiva(new VezesRepetir());
    resolvedor.associarPrimitiva(new Retornar());
    resolvedor
        .associarPrimitiva(new EnquantoVerdadeiro());
    resolvedor.associarPrimitiva(new Tentar());
    resolvedor.associarPrimitiva(new ParaSempre());
    resolvedor.associarPrimitiva(new ParaCada());
    resolvedor.associarPrimitiva(new GerarErro());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.SeVerda

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;

@Primitiva(nome = "seVerdade", parametros = {
    Booleano.class, Closure.class })
public class SeVerdade implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure closure = pilha.desempilharClosure();
    Booleano condicao = pilha
        .desempilharComoBooleano();
    if (condicao.obterValor())
```

```
      controlador.executarClosure(closure);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.Tentar

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.ConstrutorDeLista;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "tentar", parametros = { Closure.class })
public class Tentar implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure closure = pilha.desempilharClosure();
    try {
      controlador.executarClosure(closure);
      if (!Thread.currentThread()
          .isInterrupted())
        pilha.empilhar(new Booleano(true));
    } catch (ExcecaoTelis e) {
      Lista<Palavra> listaDescrição = new ConstrutorDeLista()
          .adicionarSímbolo("exceção")
          .adicionarTexto(e.getMessage())
```

```
            .adicionarTexto(
                e
                    .obterInstruçãoCausadora()
                    .toString())
            .obterSaidaComoLista();
        pilha.empilhar(listaDescrição);
        pilha.empilhar(new Booleano(true));
      }
    }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.Enquanto

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import static br.ufsc.edugraf.maquinaTelis.palavras.Conversor.converter;
import static br.ufsc.edugraf.maquinaTelis.palavras.Conversor.éPossívelConversão;
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.IEscopo;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "enquantoVerdadeiro", parametros = {
    Closure.class, Closure.class })
public class EnquantoVerdadeiro implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
```

```
  Closure closureCódigo = pilha
      .desempilharClosure()
      .criarClosureComEscopoIndependente();
  Closure closureCondição = pilha
      .desempilharClosure()
      .criarClosureComEscopoIndependente();

  IEscopo escopoCódigo = closureCódigo
      .obterEscopo();
  IEscopo escopoCondição = closureCondição
      .obterEscopo();
  escopoCódigo.declararVariavel("contarGiro");
  escopoCondição.declararVariavel("contarGiro");
  escopoCódigo.associarVariavel("contarGiro",
      new Numero(1));
  escopoCondição.associarVariavel("contarGiro",
      new Numero(1));

  int i = 1;
  while (!Thread.currentThread().isInterrupted()
      && deveContinuar(closureCondição,
          controlador)) {
    controlador.executarClosure(closureCódigo);

    i++;
    escopoCódigo.associarVariavel(
        "contarGiro", new Numero(i));
    escopoCondição.associarVariavel(
        "contarGiro", new Numero(i));
  }
}

private boolean deveContinuar(
    Closure closureCondição,
    ControladorDeExecucaoDaMaquina controlador) {
```

```
    Palavra resultado = controlador
        .avaliarExpressao(closureCondição);
    if (resultado == null
        || !éPossívelConversão(resultado,
            Booleano.class))
      throw new ExcecaoTelis(
          "Condição deve retornar um valor Booleano.");
    return converter(resultado, Booleano.class)
        .obterValor();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.ParaCada

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.IEscopo;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.ConstrutorDeLista;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "paraCada", parametros = {
    Lista.class, Closure.class })
public class ParaCada implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure closure = pilha.desempilharClosure()
```

```
        .criarClosureComEscopoIndependente();
    Lista<Palavra> lista = pilha
        .desempilharComoLista();
    IEscopo escopo = closure.obterEscopo();

    int contarGiro = 1;
    escopo.declararVariavel("contarGiro");
    escopo.associarVariavel("contarGiro",
        new Numero(contarGiro));

    ConstrutorDeLista construtorDeResposta = new ConstrutorDeLista();
    for (Palavra p : lista.obterValor()) {
      if (Thread.currentThread().isInterrupted())
        return;
      Palavra resultado = controlador
          .avaliarExpressao(closure, p);
      if (resultado != null)
        construtorDeResposta
            .adicionarPalavra(resultado);

      contarGiro = contarGiro + 1;
      escopo.associarVariavel("contarGiro",
          new Numero(contarGiro));
    }
    pilha.empilhar(construtorDeResposta
        .obterSaidaComoLista());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.SeFalso

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;

@Primitiva(nome = "seFalso", parametros = {
    Booleano.class, Closure.class })
public class SeFalso implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure closure = pilha.desempilharClosure();
    Booleano condicao = pilha
        .desempilharComoBooleano();
    if (!condicao.obterValor())
      controlador.executarClosure(closure);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.VezesRe

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Escopo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.IEscopo;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Numero;
```

```
@Primitiva(nome = "vezesRepetir", parametros = {
    Closure.class, Numero.class })
public class VezesRepetir implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Numero numeroDeVezes = pilha
        .desempilharComoNúmero();
    Closure closureOriginal = pilha
        .desempilharClosure()
        .criarClosureComEscopoIndependente();

    if (numeroDeVezes.obterValor() < 0) // TODO: utilizar/criar método
      // menorQue de Numero
      throw new ExcecaoTelis(
          "Número de iterações do vezesRepetir deve ser maior que zero.");

    IEscopo escopo = closureOriginal.obterEscopo();
    escopo.declararVariavel("contarGiro");

    for (int i = 1; i <= numeroDeVezes
        .obterParteInteira(); i++) {
      if (Thread.currentThread().isInterrupted())
        return;

      escopo.associarVariavel("contarGiro",
          new Numero(i));
      controlador
          .executarClosure(closureOriginal);
    }
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.Executar

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;

@Primitiva(nome = "executar", parametros = { Closure.class })
public class Executar implements Executavel {
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Closure closure = pilha.desempilharClosure();
    controlador.executarClosure(closure);
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.Retornar

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;

@Primitiva(nome = "retornar")
public class Retornar implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    controlador.retornar();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo.GerarErr

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivasDeControleDeFluxo;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Texto;

@Primitiva(nome = "gerarErro", parametros = { Texto.class })
public class GerarErro implements Executavel {
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Texto texto = pilha.desempilharComoTexto();
    throw new ExcecaoTelis(texto.comoTexto());
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.EhTexto

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

@Primitiva(nome = "éTexto")
```

```
public class EhTexto implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    pilha.empilhar(new Booleano(
        topo instanceof Texto));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.ComoB

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "trocar")
public class ComoBooleano implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    Palavra subtopo = pilha.desempilhar();
    pilha.empilhar(topo);
    pilha.empilhar(subtopo);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.ComoN

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;

@Primitiva(nome = "tirar")
public class ComoNumero implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    pilha.desempilhar();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.EhLista

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

@Primitiva(nome = "éLista")
public class EhLista implements Executavel {

  public void executar(IPilha pilha,
```

```
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    pilha.empilhar(new Booleano(
        topo instanceof Lista));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.Instalad

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class InstaladorDePrimitivasDeConversaoDeTipo
    implements InstalavelEmResolvedorPorCastas {

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new EhBooleano());
    resolvedor.associarPrimitiva(new EhTexto());
    resolvedor.associarPrimitiva(new EhLista());
    resolvedor.associarPrimitiva(new EhNumero());
    resolvedor.associarPrimitiva(new EhClosure());
    resolvedor.associarPrimitiva(new EhSimbolo());

    resolvedor.associarPrimitiva(new ComoTexto());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.ComoTe

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

@Primitiva(nome = "comoTexto")
public class ComoTexto implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra palavra = pilha.desempilhar();
    pilha.empilhar(new Texto(palavra.comoTexto()));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.EhNume

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

@Primitiva(nome = "éNúmero")
public class EhNumero implements Executavel {
```

```
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    pilha.empilhar(new Booleano(
        topo instanceof Numero));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.EhBoole

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "éBooleano")
public class EhBooleano implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    pilha.empilhar(new Booleano(
        topo instanceof Booleano));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.EhClosu

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

@Primitiva(nome = "éClosure")
public class EhClosure implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    Palavra subtopo = pilha.desempilhar();
    pilha.empilhar(topo);
    pilha.empilhar(subtopo);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos.EhSimb

```
package br.ufsc.edugraf.maquinaTelis.linguagem.primitivaDeConversaoDeTipos;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;

@Primitiva(nome = "éSímbolo")
public class EhSimbolo implements Executavel {
```

```
  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra topo = pilha.desempilhar();
    pilha.empilhar(new Booleano(
        topo instanceof Simbolo));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva

```
package br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import br.ufsc.edugraf.telis.palavras.Palavra;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Primitiva {

  String nome();

  Class<? extends Palavra>[] parametros() default {};

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas.IncluirModelo

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas;
```

```
import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeAtor;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;

@Primitiva(nome = "incluirModelo", parametros = {
    Lista.class, Simbolo.class, Closure.class })
public class IncluirModelo extends
    IncluirConteiner<ModeloDeAtor> {

  protected void cadastrarConteiner(
      ModeloDeAtor modelo) {
    Programa.obterInstancia()
        .cadastrarModeloDeAtor(modelo);
  }

  protected ModeloDeAtor criarConteiner(
      Simbolo nomeDoModelo,
      Lista<Palavra> listaDeMoldes) {
    return new ModeloDeAtor(nomeDoModelo
        .comoTexto(),
        obterListaDeMoldes(listaDeMoldes));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas.IncluirMolde

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Molde;
```

```
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;

@Primitiva(nome = "incluirMolde", parametros = {
    Lista.class, Simbolo.class, Closure.class })
public class IncluirMolde extends
    IncluirConteiner<Molde> {

  @Override
  protected void cadastrarConteiner(Molde molde) {
    Programa.obterInstancia()
        .cadastrarMolde(molde);
  }

  @Override
  protected Molde criarConteiner(
      Simbolo nomeDoMolde,
      Lista<Palavra> listaDeMoldes) {
    return new Molde(nomeDoMolde.comoTexto(),
        obterListaDeMoldes(listaDeMoldes));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas.IncluirModeloI

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeObjetos;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
```

```
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;

@Primitiva(nome = "incluirModeloDeObjeto", parametros = {
    Lista.class, Simbolo.class, Closure.class })
public class IncluirModeloDeObjeto extends
    IncluirConteiner<ModeloDeObjetos> {

  protected void cadastrarConteiner(
      ModeloDeObjetos modelo) {
    Programa.obterInstancia()
        .cadastrarModeloDeObjetos(modelo);
  }

  protected ModeloDeObjetos criarConteiner(
      Simbolo nomeDoModelo,
      Lista<Palavra> listaDeMoldes) {
    return new ModeloDeObjetos(nomeDoModelo
        .comoTexto(),
        obterListaDeMoldes(listaDeMoldes));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas.InstaladorDePr

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public class InstaladorDePrimitivasDeCriacao implements
    InstalavelEmResolvedorPorCastas {
```

```
  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor
        .associarPrimitiva(new IncluirModeloDeObjeto());
    resolvedor
        .associarPrimitiva(new IncluirModelo());
    resolvedor
        .associarPrimitiva(new IncluirMolde());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas.IncluirConteine

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.primitivas;

import java.util.ArrayList;
import java.util.Collection;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Agenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Conteiner;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ConteinerDeAgendas;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Molde;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.ListaAbstrata;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;
import br.ufsc.edugraf.telis.palavras.Texto;
```

```
abstract class IncluirConteiner<T extends Conteiner>
    implements Executavel {

  protected abstract T criarConteiner(
      Simbolo nomeDoMolde,
      Lista<Palavra> listaDeMoldes);

  protected abstract void cadastrarConteiner(
      T conteiner);

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    ListaDeComandos definicaoDasAgendas = pilha
        .desempilharClosure()
        .obterListaDeComandos();
    Simbolo nomeDoMolde = pilha
        .desempilharComoSímbolo();
    Lista<Palavra> listaDeMoldes = pilha
        .desempilharComoLista();
    T conteiner = criarConteiner(nomeDoMolde,
        listaDeMoldes);
    incluirAgendas(conteiner, definicaoDasAgendas);
    cadastrarConteiner(conteiner);
  }

  protected Collection<Molde> obterListaDeMoldes(
      Lista<Palavra> listaDeMoldes) {
    ArrayList<Molde> retorno = new ArrayList<Molde>();
    for (Palavra nomeDoMolde : listaDeMoldes
        .obterValor()) {
      if (!(nomeDoMolde instanceof Simbolo))
        throw new ExcecaoTelis();
      Molde molde = Programa.obterInstancia()
          .obterMolde(
              nomeDoMolde.comoTexto());
```

```
    if (molde == null)
      throw new ExcecaoTelis();
    retorno.add(molde);
  }
  return retorno;
}

private void incluirAgendas(
    ConteinerDeAgendas conteinerDeAgendas,
    ListaAbstrata<Palavra> definicaoDasAgendas) {
  if (definicaoDasAgendas.obterTamanho() % 3 != 0)
    throw new ExcecaoTelis();
  for (int i = 0; i < definicaoDasAgendas
      .obterTamanho(); i = i + 3) {
    Palavra palavraNome = definicaoDasAgendas
        .obterElemento(i);
    Palavra palavraCodigo = definicaoDasAgendas
        .obterElemento(i + 1);
    Palavra palavraTipo = definicaoDasAgendas
        .obterElemento(i + 2);
    if (palavraNome instanceof Texto)
      palavraNome = convertarParaSimbolo((Texto) palavraNome);
    if (!(palavraNome instanceof Simbolo
        && palavraCodigo instanceof ListaDeComandos && palavraTipo instanceof Sim
      throw new ExcecaoTelis(
          "Lista de agendas deve possuir um símbolo, uma lista de comandos e uma

    Simbolo nome = (Simbolo) palavraNome;
    ListaDeComandos códigoDaAgenda = (ListaDeComandos) palavraCodigo;
    Simbolo tipo = (Simbolo) palavraTipo;

    if (tipo
        .obterValor()
        .equals(
            Constantes
```

```
                .obterString("Primitivas.incluirAgenda"))) //$NON-NLS-1$
        conteinerDeAgendas
            .adicionarAgenda(new Agenda(
                nome.obterValor(),
                códigoDaAgenda));
      else if (tipo
          .obterValor()
          .equals(
              Constantes
                  .obterString("Primitivas.incluirAgendaDeModelo"))) //$NON-NLS-1$
        conteinerDeAgendas
            .adicionarAgendaDeMetaclasse(new Agenda(
                nome.obterValor(),
                códigoDaAgenda));
      else
        throw new ExcecaoTelis(
            "Lista de agendas está utilizando primitiva de inclusão de agendas invá

    }
  }

  private Simbolo convertarParaSimbolo(Texto simbolo) {
    return new Simbolo(simbolo.obterValor());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import br.ufsc.edugraf.telis.palavras.ListaDeComandos;

public interface IAgenda {
```

```
  public abstract String obterNome();

  public abstract ListaDeComandos obterCódigo();

  public abstract boolean éPrivada();

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Molde

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Collection;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public class Molde extends ConteinerAbstrato {

  public Molde(String nome) {
    super(nome);
  }

  public Molde(String nome, Collection<Molde> moldes) {
    super(nome, moldes);
  }

  public Molde(String nome, Molde... moldes) {
    super(nome, moldes);
  }

  public void adicionarAgendaDeMetaclasse(
      Agenda agenda) {
    throw new ExcecaoTelis(
        "Não é possível adicionar agenda de modelo em Moldes.");
  }
```

```
public IAgenda obterAgendaDeMetaclasse(
    String nomeAgenda) {
  throw new ExcecaoTelis(
      "Moldes não possuem agendas de Modelo.");
}

@Override
public int hashCode() {
  final int PRIME = 31;
  int result = 1;
  result = PRIME
      * result
      + ((obterNome() == null) ? 0
          : obterNome().hashCode());
  return result;
}

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  final Molde other = (Molde) obj;
  if (obterNome() == null) {
    if (other.obterNome() != null)
      return false;
  } else if (!obterNome().equals(
      other.obterNome()))
    return false;
  return true;
}
```

```
  @Override
  public String toString() {
    return "Molde: " + obterNome(); //$NON-NLS-1$
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ConteinerDeMoldes

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

public interface ConteinerDeMoldes {

  public abstract Molde obterMolde(String nomeDoMolde);

  boolean éExplicitamenteMoldadoPor(Molde outroMolde);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Iterator;

import br.ufsc.edugraf.maquinaTelis.util.Visitante;

public interface Modelo extends ConteinerDeAgendas,
    ConteinerDeMoldes {

  public abstract IAgenda obterAgendaEstendida(
      String nome);

  public abstract IAgenda obterAgendaDeMetaclasse(
      String nome);
```

```
  public abstract IAgenda obterAgendaInicialEstendida();

  public abstract Iterator<Molde> obterIteradorDeTodosOsMoldes();

  public abstract void receberVisitanteNoGrafoDeHerança(
      Visitante<Conteiner> visitante);

  public abstract String obterNome();

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ConteinerAbstrato

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.maquinaTelis.util.OperacoesSobreColecoes;
import br.ufsc.edugraf.maquinaTelis.util.Visitante;

abstract class ConteinerAbstrato implements Conteiner {

  private final String nome;
  private final Map<String, Agenda> agendas = Collections
      .synchronizedMap(new HashMap<String, Agenda>());
  private final Map<String, Molde> moldes = Collections
      .synchronizedMap(new LinkedHashMap<String, Molde>());
```

```
public ConteinerAbstrato(String nome) {
  this.nome = nome;
  agendas.put(NOME_AGENDA_INICIAR, new Agenda(
      NOME_AGENDA_INICIAR));
}

public ConteinerAbstrato(String nome,
    Molde... moldes) {
  this(nome);
  for (Molde molde : moldes) {
    if (this.moldes.containsKey(molde
        .obterNome()))
      throw new ExcecaoTelis(
          "Lista de moldes não pode conter entradas duplicatas.");
    this.moldes.put(molde.obterNome(), molde);
  }
}

public ConteinerAbstrato(String nome,
    Collection<Molde> moldes) {
  this(nome, moldes.toArray(new Molde[] {}));
}

public String obterNome() {
  return nome;
}

public Agenda obterAgendaInicial() {
  return obterAgenda(Constantes
      .obterString("Modelo.nomeAgendaIniciar")); //$NON-NLS-1$
}

public void adicionarAgenda(Agenda agenda) {
  agendas.put(agenda.obterNome(), agenda);
```

```
}

public Agenda obterAgenda(String nomeDaAgenda) {
  return agendas.get(nomeDaAgenda);
}

public Molde obterMolde(String nomeDoMolde) {
  return moldes.get(nomeDoMolde);
}

protected Iterator<Molde> obterIteradorDeTodosOsMoldes() {
  List<Molde> moldes = calcularOrdemTopológicaDosMoldes();
  return moldes.iterator();
}

private List<Molde> calcularOrdemTopológicaDosMoldes() {
  LinkedList<Molde> moldesAProcessar = new LinkedList<Molde>();
  OperacoesSobreColecoes
      .adicionarListaInvertida(
          moldesAProcessar, moldes
              .values());
  LinkedList<Molde> resultado = new LinkedList<Molde>();
  while (!moldesAProcessar.isEmpty()) {
    Molde molde = moldesAProcessar.poll();
    resultado.remove(molde);
    resultado.add(molde);
    OperacoesSobreColecoes
        .adicionarListaInvertida(
            moldesAProcessar,
            molde
                .obterMoldesImediatos());
  }
  Collections.reverse(resultado);
  return resultado;
}
```

```
  protected Collection<Molde> obterMoldesImediatos() {
    return moldes.values();
  }

  protected void receberVisitanteNoGrafoDeHerança(
      Visitante<Conteiner> visitante) {
    Iterator<Molde> iterador = obterIteradorDeTodosOsMoldes();
    while (iterador.hasNext())
      visitante.visitar(iterador.next());
    visitante.visitar(this);
  }

  public boolean éExplicitamenteMoldadoPor(
      Molde outroMolde) {
    return obterMoldesImediatos().contains(
        outroMolde);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class Programa {

  private static Programa instancia;

  private final Map<String, ModeloDeAtor> mapeamentoDeModelos = Collections
      .synchronizedMap(new HashMap<String, ModeloDeAtor>());
```

```
private final Map<String, Molde> mapeamentoDeMoldes = Collections
    .synchronizedMap(new HashMap<String, Molde>());

private final Map<String, ModeloDeObjetos> mapeamentoDeModelosDeObjetos = Collect
    .synchronizedMap(new HashMap<String, ModeloDeObjetos>());

public synchronized static Programa obterInstancia() {
  if (instancia == null)
    instancia = new Programa();
  return instancia;
}

private Programa() {
}

public synchronized static void destruirPrograma() {
  instancia = null;
}

public void cadastrarModeloDeAtor(
    ModeloDeAtor modelo) {
  mapeamentoDeModelos.put(modelo.obterNome(),
      modelo);
}

public ModeloDeAtor obterModeloDeAtor(
    String nomeDoModelo) {
  return mapeamentoDeModelos.get(nomeDoModelo);
}

public void cadastrarModeloDeObjetos(
    ModeloDeObjetos modelo) {
  mapeamentoDeModelosDeObjetos.put(modelo
      .obterNome(), modelo);
```

```
}

public ModeloDeObjetos obterModeloDeObjetos(
    String nomeDoModelo) {
  return mapeamentoDeModelosDeObjetos
      .get(nomeDoModelo);
}

public Modelo obterModelo(String nomeDoModelo) {
  Modelo modelo = obterModeloDeAtor(nomeDoModelo);
  if (modelo == null)
    modelo = obterModeloDeObjetos(nomeDoModelo);
  return modelo;
}

public void cadastrarMolde(Molde molde) {
  mapeamentoDeMoldes.put(molde.obterNome(),
      molde);
}

public Molde obterMolde(String nomeDoMolde) {
  return mapeamentoDeMoldes.get(nomeDoMolde);
}

public boolean existeMolde(String nomeDoModelo) {
  return mapeamentoDeMoldes
      .containsKey(nomeDoModelo);
}

public boolean existeModelo(String nomeDoModelo) {
  return mapeamentoDeModelos
      .containsKey(nomeDoModelo)
      || mapeamentoDeModelosDeObjetos
          .containsKey(nomeDoModelo);
}
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloAbstrato

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.AgendaComposta;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.maquinaTelis.util.Visitante;

abstract class ModeloAbstrato extends
    ConteinerAbstrato implements Modelo {
  private final Map<String, Agenda> agendasDaMetaclasse = Collections
      .synchronizedMap(new HashMap<String, Agenda>());

  public static class MontadorDeAgendaComposta
      implements Visitante<Conteiner> {
    private final String nomeDaAgenda;
    private AgendaComposta agendaComposta;

    public MontadorDeAgendaComposta(
        String nomeDaAgenda) {
      this.nomeDaAgenda = nomeDaAgenda;
    }

    public void visitar(Conteiner conteiner) {
      Agenda agenda = conteiner
          .obterAgenda(nomeDaAgenda);
      if (agenda != null)
```

```
      obterAgendaComposta().adicionarAgenda(
          agenda);
  }

  private AgendaComposta obterAgendaComposta() {
    if (agendaComposta == null)
      agendaComposta = new AgendaComposta(
          nomeDaAgenda);
    return agendaComposta;
  }

  public IAgenda obterResultado() {
    return agendaComposta;
  }
}

public ModeloAbstrato(String nome) {
  super(nome);
}

public ModeloAbstrato(String nome,
    Collection<Molde> moldes) {
  super(nome, moldes);
}

public ModeloAbstrato(String nome, Molde... moldes) {
  super(nome, moldes);
}

public IAgenda obterAgendaEstendida(String nome) {
  ModeloAbstrato.MontadorDeAgendaComposta montador = new ModeloAbstrato.MontadorD
      nome);
  receberVisitanteNoGrafoDeHerança(montador);
  IAgenda agenda = montador.obterResultado();
  return agenda;
```

```
}

public IAgenda obterAgendaInicialEstendida() {
  return obterAgendaEstendida(Constantes
      .obterString("Modelo.nomeAgendaIniciar")); //$NON-NLS-1$
}

@Override
public Iterator<Molde> obterIteradorDeTodosOsMoldes() {
  return super.obterIteradorDeTodosOsMoldes();
}

@Override
public void receberVisitanteNoGrafoDeHerança(
    Visitante<Conteiner> visitante) {
  super
      .receberVisitanteNoGrafoDeHerança(visitante);
}

public void adicionarAgendaDeMetaclasse(
    Agenda agenda) {
  assert agenda != null;
  agendasDaMetaclasse.put(agenda.obterNome(),
      agenda);
}

public IAgenda obterAgendaDeMetaclasse(
    String nomeAgenda) {
  return agendasDaMetaclasse.get(nomeAgenda);
}

@Override
public int hashCode() {
  final int PRIME = 31;
  int result = 1;
```

```
    result = PRIME
        * result
        + ((obterNome() == null) ? 0
            : obterNome().hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    final ModeloAbstrato other = (ModeloAbstrato) obj;
    if (obterNome() == null) {
      if (other.obterNome() != null)
        return false;
    } else if (!obterNome().equals(
        other.obterNome()))
      return false;
    return true;
  }

  @Override
  public String toString() {
    return "Modelo: " + obterNome(); //$NON-NLS-1$
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeObjetos

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;
```

```
import java.util.Collection;
import java.util.Iterator;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.AgendaComposta;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.maquinaTelis.util.Visitante;

public class ModeloDeObjetos extends ModeloAbstrato {

  public ModeloDeObjetos(String nome,
      Collection<Molde> moldes) {
    super(nome, moldes);
  }

  public ModeloDeObjetos(String nome,
      Molde... moldes) {
    super(nome, moldes);
  }

  public ModeloDeObjetos(String nome) {
    super(nome);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ConteinerDeAgendas

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;

public interface ConteinerDeAgendas {

  public static final String NOME_AGENDA_INICIAR = Constantes
```

```
      .obterString("Modelo.nomeAgendaIniciar");

  public abstract Agenda obterAgendaInicial();

  public abstract void adicionarAgenda(Agenda agenda);

  public abstract Agenda obterAgenda(
      String nomeDaAgenda);

  public void adicionarAgendaDeMetaclasse(
      Agenda agenda);

  public IAgenda obterAgendaDeMetaclasse(
      String nomeAgenda);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Agenda

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Collections;

import br.ufsc.edugraf.telis.palavras.ListaDeComandos;

public class Agenda implements IAgenda {

  private final String nome;
  private final ListaDeComandos codigoDaAgenda;

  @SuppressWarnings("unchecked")//$NON-NLS-1$
  public Agenda(String nome,
      ListaDeComandos codigoDaAgenda) {
    this.nome = nome;
    if (codigoDaAgenda == null)
      this.codigoDaAgenda = new ListaDeComandos(
```

```
        Collections.EMPTY_LIST);
  else
    this.codigoDaAgenda = codigoDaAgenda;
}

public Agenda(String nome) {
  this(nome, null);
}

public String obterNome() {
  return nome;
}

public ListaDeComandos obterCódigo() {
  return codigoDaAgenda;
}

@Override
public int hashCode() {
  final int PRIME = 31;
  int result = 1;
  result = PRIME
      * result
      + ((nome == null) ? 0 : nome
          .hashCode());
  return result;
}

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
```

```
      return false;
    final Agenda other = (Agenda) obj;
    if (nome == null) {
      if (other.nome != null)
        return false;
    } else if (!nome.equals(other.nome))
      return false;
    return true;
  }

  @Override
  public String toString() {
    return "Agenda \"" + nome + "\""; //$NON-NLS-1$ //$NON-NLS-2$
  }

  public boolean éPrivada() {
    return nome.startsWith("_");
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeAtor

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

import java.util.Collection;
import java.util.Iterator;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.AgendaComposta;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.maquinaTelis.util.Visitante;

public class ModeloDeAtor extends ModeloAbstrato {

  public ModeloDeAtor(String nome,
      Collection<Molde> moldes) {
```

```
    super(nome, moldes);
  }

  public ModeloDeAtor(String nome, Molde... moldes) {
    super(nome, moldes);
  }

  public ModeloDeAtor(String nome) {
    super(nome);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Conteiner

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica;

public interface Conteiner extends ConteinerDeMoldes,
    ConteinerDeAgendas {

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.utilitarios.ResolvedorAssociativo

```
package br.ufsc.edugraf.maquinaTelis.linguagem.utilitarios;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.telis.Resolvivel;

public class ResolvedorAssociativo<T extends Resolvivel>
    extends ResolvedorAssociativoAbstrato<T> {

  @Override
  public void associar(T resolvivel,
      Executavel executavel) {
```

```
    super.associar(resolvivel, executavel);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.utilitarios.ResolvedorAssociativoAbstra

```
package br.ufsc.edugraf.maquinaTelis.linguagem.utilitarios;

import java.util.HashMap;
import java.util.Map;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.telis.Resolvivel;

public class ResolvedorAssociativoAbstrato<T extends Resolvivel>
    implements Resolvedor {

  private Map<T, Executavel> mapeamento = new HashMap<T, Executavel>();

  protected ResolvedorAssociativoAbstrato() {
  }

  public final Executavel resolver(Resolvivel simbolo) {
    return mapeamento.get(simbolo);
  }

  protected void associar(T resolvivel,
      Executavel executavel) {
    mapeamento.put(resolvivel, executavel);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida.Suicidar

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;

@Primitiva(nome = "suicidar")
public class Suicidar implements Executavel {

  private final Ator ator;

  public Suicidar(Ator ator) {
    this.ator = ator;
  }

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    ator.terminarExecução();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida.InstaladorD

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.palavras.Simbolo;
```

```
public class InstaladorDePrimitivasDeCicloDeVida
    implements InstalavelEmResolvedorPorCastas {

  private final Ator ator;

  public InstaladorDePrimitivasDeCicloDeVida(
      Ator ator) {
    this.ator = ator;
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor
        .associarPrimitiva(new Suicidar(ator));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida.Resolvedor

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.cicloDeVida;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeAtor;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeObjetos;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
```

```
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class ResolvedorDeModelos implements
    Resolvedor, InstalavelEmResolvedorPorCastas {

  private class EmpilhadorDeReferencia implements
      Executavel {
    private final UnidadeDeExecucao u;

    public EmpilhadorDeReferencia(
        UnidadeDeExecucao u) {
      this.u = u;
    }

    public void executar(
        IPilha pilha,
        ControladorDeExecucaoDaMaquina controlador) {
      pilha.empilhar(new Referencia(u));
    }
  }

  public Executavel resolver(Resolvivel r) {
    if (r instanceof Simbolo) {
      Programa programa = Programa
          .obterInstancia();
      Simbolo simbolo = (Simbolo) r;
      String nomeDoSimbolo = simbolo
          .obterValor();
      Modelo modelo = programa
          .obterModelo(nomeDoSimbolo);
      if (modelo != null) {
        ObjetoClasse objeto = Aplique
            .obterInstancia()
```

```
            .obterObjetoClasseParaModelo(
                modelo);
        return new EmpilhadorDeReferencia(
            objeto);
      }
      if (programa.existeMolde(nomeDoSimbolo))
        throw new ExcecaoTelis(
            "Moldes não podem ser instanciado.");
    }
    return null;
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.adicionarResolvedorGenérico(this);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.ChamadaDeAg

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;

public class ChamadaDeAgenda implements Executavel {

  private final IAgenda agenda;

  public ChamadaDeAgenda(IAgenda agenda) {
    this.agenda = agenda;
  }

  public void executar(IPilha pilha,
```

```
      ControladorDeExecucaoDaMaquina controlador) {
    controlador.executar(agenda);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.AgendaCompos

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas;

import java.util.ArrayList;
import java.util.List;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class AgendaComposta implements IAgenda {

  private final String nome;

  private List<IAgenda> listaDeAgendas = new ArrayList<IAgenda>();

  public AgendaComposta(String nome) {
    this.nome = nome;
  }

  public ListaDeComandos obterCódigo() {
    ArrayList<Palavra> códigoComposto = new ArrayList<Palavra>();
    for (IAgenda agenda : listaDeAgendas)
      códigoComposto.addAll(agenda.obterCódigo()
          .obterValor());
    return new ListaDeComandos(códigoComposto);
  }
```

```
  public String obterNome() {
    return nome;
  }

  public void adicionarAgenda(IAgenda agenda) {
    listaDeAgendas.add(agenda);
  }

  public boolean éPrivada() {
    return nome.startsWith("_");
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.ResolvedorDeA

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class ResolvedorDeAgendasDeMetaclasse implements
    Resolvedor, InstalavelEmResolvedorPorCastas {

  private Modelo modelo;

  public ResolvedorDeAgendasDeMetaclasse(
      Modelo modelo) {
    this.modelo = modelo;
```

```
  }

  public Executavel resolver(Resolvivel r) {
    if (r instanceof Simbolo) {
      Simbolo simbolo = (Simbolo) r;
      IAgenda agenda = modelo
          .obterAgendaDeMetaclasse(simbolo
              .obterValor());
      if (agenda != null)
        return new ChamadaDeAgenda(agenda);
    }
    return null;
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.adicionarResolvedorGenérico(this);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas.ResolvedorDeA

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.agendas;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class ResolvedorDeAgendas implements
    Resolvedor, InstalavelEmResolvedorPorCastas {
```

```
  private Modelo modelo;

  public ResolvedorDeAgendas(Modelo modelo) {
    this.modelo = modelo;
  }

  public Executavel resolver(Resolvivel r) {
    if (r instanceof Simbolo) {
      Simbolo simbolo = (Simbolo) r;
      IAgenda agenda = modelo
          .obterAgendaEstendida(simbolo
              .obterValor());
      if (agenda != null)
        return new ChamadaDeAgenda(agenda);
    }
    return null;
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.adicionarResolvedorGenérico(this);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucaoAb

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IMaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.maquinaTelis.util.Observavel;
import br.ufsc.edugraf.telis.palavras.Texto;

abstract class UnidadeDeExecucaoAbstrata<Maquina extends IMaquinaDePilha>
    extends Observavel<EventosDeCicloDeVidaDoAtor>
    implements UnidadeDeExecucao {

  private final Maquina maquina;
  private Texto identidade;
  private Map<Class<?>, Object> extensoes = new HashMap<Class<?>, Object>();

  public UnidadeDeExecucaoAbstrata(Texto identidade,
      Maquina maquina) {
    this.maquina = maquina;
    this.identidade = identidade;
  }

  protected Maquina obterMaquina() {
    return maquina;
  }

  public abstract void iniciarExecução();

  protected void executarAgendaInicial() {
    try {
      maquina
          .executarAgendaIniciar(obterModelo()
              .obterAgendaInicialEstendida()
              .obterCódigo()
              .obterValor());
    } catch (ExcecaoTelis e) {
      e
          .notificarQuePassouPelaAgenda(
```

```
            Constantes
                .obterString("Modelo.nomeAgendaIniciar"),
            obterModelo().obterNome());
    throw e;
  }
}

public Texto obterIdentidade() {
  return identidade;
}

public <T> void cadastrarExtensão(Class<T> classe,
    T objeto) {
  extensoes.put(classe, objeto);
}

@SuppressWarnings("unchecked")
public <T> T obterExtensão(Class<T> classe) {
  Object objeto = extensoes.get(classe);
  return (T) objeto;
}

public abstract Modelo obterModelo();

@Override
public int hashCode() {
  return (identidade == null) ? 0 : identidade
      .hashCode();
}

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
```

```
      return false;
    if (getClass() != obj.getClass())
      return false;
    final UnidadeDeExecucaoAbstrata<?> other = (UnidadeDeExecucaoAbstrata<?>) obj;
    if (identidade == null) {
      if (other.identidade != null)
        return false;
    } else if (!identidade
        .equals(other.identidade))
      return false;
    return true;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;

import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeAtor;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeObjetos;
import br.ufsc.edugraf.maquinaTelis.util.Visitante;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;
```

```
public class Aplique {

  private static Aplique instancia;
  private volatile AtomicInteger proximaIdentidade;
  private volatile ConcurrentHashMap<Texto, Ator> atores = new ConcurrentHashMap<Te
  private Map<Modelo, ObjetoClasse> objetosClasse = new HashMap<Modelo, ObjetoClass
  private InstaladorDeModulos instaladorDeModulos = new InstaladorDeModulosBasicos(

  public synchronized static Aplique obterInstancia() {
    if (instancia == null) {
      instancia = new Aplique();
    }
    return instancia;
  }

  public synchronized static void destruirAplique() {
    if (instancia != null)
      for (Ator ator : instancia.atores.values())
        ator.terminarExecução();
    instancia = null;
  }

  private Aplique() {
    proximaIdentidade = new AtomicInteger(
        new Random().nextInt());
  }

  public Ator instanciarAtor(ModeloDeAtor modelo) {
    Ator ator = new Ator(modelo,
        gerarProximaIdentidade(),
        instaladorDeModulos);
    cadastrarAtor(ator);
    return ator;
  }
```

```
public Ator instanciarAtor(ModeloDeAtor modelo,
    Palavra parametro) {
  Ator ator = new Ator(modelo,
      gerarProximaIdentidade(), parametro,
      instaladorDeModulos);
  cadastrarAtor(ator);
  return ator;
}

public synchronized ObjetoClasse obterObjetoClasseParaModelo(
    Modelo modelo) {
  ObjetoClasse objeto = objetosClasse
      .get(modelo);
  if (objeto == null) {
    objeto = new ObjetoClasse(modelo,
        gerarProximaIdentidade(),
        instaladorDeModulos);
    objetosClasse.put(modelo, objeto);
  }
  return objeto;
}

public void cadastrarAtor(Ator ator) {
  atores.put(ator.obterIdentidade(), ator);
}

public void destruirAtor(Texto identidade) {
  Ator ator = atores.remove(identidade);
  if (ator != null)
    ator.terminarExecução();
}

public Ator obterAtor(Texto identidade) {
  return atores.get(identidade);
}
```

```
private Texto gerarProximaIdentidade() {
  int id = proximaIdentidade.incrementAndGet();
  return new Texto(Integer.toString(id));
}

public void fixarInstaladorDeModulos(
    InstaladorDeModulos instaladorDeModulos) {
  this.instaladorDeModulos = instaladorDeModulos;
}

public Objeto instanciarObjeto(
    ModeloDeObjetos modelo) {
  return new Objeto(modelo,
      gerarProximaIdentidade(),
      instaladorDeModulos);
}

public Objeto instanciarObjeto(
    ModeloDeObjetos modelo, Palavra parametro) {
  return new Objeto(modelo,
      gerarProximaIdentidade(), parametro,
      instaladorDeModulos);
}

public synchronized void receberVisitante(
    Visitante<Ator> visitante) {
  for (Ator ator : atores.values())
    visitante.visitar(ator);
}

public UnidadeDeExecucao instanciar(Modelo modelo,
    Palavra parametro) {
  UnidadeDeExecucao unidade;
  if (modelo instanceof ModeloDeObjetos) {
```

```
      Objeto o = instanciarObjeto(
          (ModeloDeObjetos) modelo,
          parametro);
      o.iniciarExecução();
      unidade = o;
    } else {
      Ator a = instanciarAtor(
          (ModeloDeAtor) modelo, parametro);
      a.iniciarExecução();
      unidade = a;
    }
    return unidade;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.EventosDeCicloDeVida

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

public enum EventosDeCicloDeVidaDoAtor {

  INICIOU_EXECUÇÃO, HIBERNOU, SUICIDOU

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.telis.palavras.Texto;

public interface UnidadeDeExecucao {
```

```
  public abstract Texto obterIdentidade();

  public abstract Modelo obterModelo();

  <T> void cadastrarExtensão(Class<T> classe,
      T objeto);

  <T> T obterExtensão(Class<T> classe);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

import java.util.concurrent.Semaphore;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErros;
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeAtor;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IMaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.PedidoDeInterrupcao;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Pilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamentoDeI
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Escopo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.IEscopo;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.maquinaTelis.util.Observavel;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
```

```
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

public class Ator extends
    UnidadeDeExecucaoAbstrata<MaquinaDePilha> {

  private final class ThreadDoAtor extends Thread {
    public void run() {
      dispararEvento(EventosDeCicloDeVidaDoAtor.INICIOU_EXECUÇÃO);
      try {
        executarAgendaInicial();
        if (!threadDoAtor.isInterrupted())
          dispararEvento(EventosDeCicloDeVidaDoAtor.HIBERNOU);
        obterMaquina().hibernar();
      } catch (ExcecaoTelis e) {
        ReportadorDeErros.reportarErro(e);
      } catch (Throwable e) {
        // TODO: logar exceção, nunca deve ocorrer
        throw new AssertionError(
            "Nunca deveria acontecer");
      } finally {
        Aplique.obterInstancia().destruirAtor(
            obterIdentidade());
        dispararEvento(EventosDeCicloDeVidaDoAtor.SUICIDOU);
      }
    }

    public boolean maquinaDoAtorExecutandoÉ(
        IMaquinaDePilha maquina) {
      return obterMaquina() == maquina;
    }
  }

  private static final Thread THREAD_NULA = new Thread();
  private Thread threadDoAtor = THREAD_NULA;
```

```
private IEscopo escopoParaInstruçõesAdHoc;
private IPilha pilhaParaInstruçõesAdHoc;
private ModeloDeAtor modelo;

public Ator(ModeloDeAtor modelo, Texto identidade) {
  this(modelo, identidade, null,
      new InstaladorDeModulosBasicos());
}

public Ator(ModeloDeAtor modelo, Texto identidade,
    Palavra parametro) {
  this(modelo, identidade, parametro,
      new InstaladorDeModulosBasicos());
}

public Ator(ModeloDeAtor modelo, Texto identidade,
    InstaladorDeModulos instaladorDeModulos) {
  this(modelo, identidade, null,
      instaladorDeModulos);
}

public Ator(ModeloDeAtor modelo, Texto identidade,
    Palavra parametro,
    InstaladorDeModulos instaladorDeModulos) {
  super(identidade,
      new MaquinaDePilha(parametro));

  this.modelo = modelo;
  obterMaquina().fixarNomeDoModelo(
      modelo.obterNome());

  ResolvedorPorCasta r = new ResolvedorPorCasta();
  obterMaquina().fixarResolvedorDeSimbolos(r);
  instaladorDeModulos.instalarMódulos(this,
      obterMaquina(), r);
```

```
}

public void iniciarExecução() {
  threadDoAtor = new ThreadDoAtor();
  threadDoAtor.start();
}

public void executarInstruçõesAdHoc(
    ListaDeComandos instruções) {
  final Semaphore semaforo = new Semaphore(0);
  Closure closure = criarClosureParaInstruçõesAdHoc(instruções);
  PedidoDeInterrupcao pedidoDeInterrupcao = new PedidoDeInterrupcao(
      closure,
      obterPilhaParaInstruçõesAdHoc(),
      new TratadorDeEventosDoTratamentoDeInterrupcao() {
        public void aoTerminarDeTratarInterrupcao() {
          semaforo.release();
        }

        public void aoTerminarDevidoAExcecao(
            ExcecaoTelis e) {
          e
              .notificarQuePassouPelaAgenda(
                  "<instruções ad-hoc>",
                  obterModelo()
                      .obterNome());
          ReportadorDeErros
              .reportarErro(e);
          semaforo.release();
        }
      });
  obterMaquina().agendarInterrupção(
      pedidoDeInterrupcao);
  try {
    semaforo.acquire();
```

```
  } catch (InterruptedException e) {
    Thread.currentThread().interrupt();
  }
}

public void terminarExecução() {
  threadDoAtor.interrupt();
  Aplique.obterInstancia().destruirAtor(
      obterIdentidade());
}

private IPilha obterPilhaParaInstruçõesAdHoc() {
  if (pilhaParaInstruçõesAdHoc == null)
    pilhaParaInstruçõesAdHoc = new Pilha();
  return pilhaParaInstruçõesAdHoc;
}

private Closure criarClosureParaInstruçõesAdHoc(
    ListaDeComandos instruções) {
  if (escopoParaInstruçõesAdHoc == null) {
    Closure closure = obterMaquina()
        .criarClosureComEscopoRaiz(
            instruções);
    escopoParaInstruçõesAdHoc = new Escopo(
        closure.obterEscopo()) {
      private Escopo escopoCompartilhado = new Escopo(
          this);

      @Override
      public Escopo criarSubescopo() {
        return escopoCompartilhado;
      }
    };
  }
  return new Closure(obterMaquina(), instruções,
```

```
        escopoParaInstruçõesAdHoc);
  }

  @Override
  public ModeloDeAtor obterModelo() {
    return modelo;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeObjetos;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

public class ObjetoClasse
    extends
    UnidadeDeExecucaoAbstrata<MaquinaDePilhaSimples> {

  private final Modelo modelo;

  public ObjetoClasse(Modelo modelo, Texto identidade) {
    this(modelo, identidade,
        new InstaladorDeModulosBasicos());
  }

  public ObjetoClasse(Modelo modelo,
```

```
      Texto identidade,
      InstaladorDeModulos instaladorDeModulos) {
    super(identidade, new MaquinaDePilhaSimples());

    this.modelo = modelo;

    ResolvedorPorCasta r = new ResolvedorPorCasta();
    obterMaquina().fixarResolvedorDeSimbolos(r);
    obterMaquina().fixarNomeDoModelo(
        "Classe " + modelo.obterNome());
    instaladorDeModulos.instalarMódulos(this,
        obterMaquina(), r);
  }

  @Override
  public void iniciarExecução() {
    executarAgendaInicial();
  }

  @Override
  public Modelo obterModelo() {
    return modelo;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto

```
package br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica;

import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeObjetos;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
```

```
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

public class Objeto
    extends
    UnidadeDeExecucaoAbstrata<MaquinaDePilhaSimples> {

  private final ModeloDeObjetos modelo;

  public Objeto(ModeloDeObjetos modelo,
      Texto identidade) {
    this(modelo, identidade, null,
        new InstaladorDeModulosBasicos());
  }

  public Objeto(ModeloDeObjetos modelo,
      Texto identidade, Palavra parametro) {
    this(modelo, identidade, parametro,
        new InstaladorDeModulosBasicos());
  }

  public Objeto(ModeloDeObjetos modelo,
      Texto identidade, Palavra parametro,
      InstaladorDeModulos instaladorDeModulos) {
    super(identidade, new MaquinaDePilhaSimples(
        parametro));

    this.modelo = modelo;

    ResolvedorPorCasta r = new ResolvedorPorCasta();
    obterMaquina().fixarResolvedorDeSimbolos(r);
    obterMaquina().fixarNomeDoModelo(
        modelo.obterNome());
    instaladorDeModulos.instalarMódulos(this,
        obterMaquina(), r);
```

```
  }

  public Objeto(ModeloDeObjetos modelo,
      Texto identidade,
      InstaladorDeModulos instaladorDeModulos) {
    this(modelo, identidade, null,
        instaladorDeModulos);
  }

  @Override
  public void iniciarExecução() {
    executarAgendaInicial();
  }

  @Override
  public ModeloDeObjetos obterModelo() {
    return modelo;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.util.OperacoesSobreColecoes

```
package br.ufsc.edugraf.maquinaTelis.util;

import java.util.Collection;
import java.util.Iterator;
import java.util.List;

public class OperacoesSobreColecoes {

  public static <T> void adicionarListaInvertida(
      List<T> destino, Collection<T> origem) {
    Iterator<T> e1 = origem.iterator();
    int posição = destino.size();
```

```
    while (e1.hasNext())
      destino.add(posição, e1.next());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.util.Observavel

```
package br.ufsc.edugraf.maquinaTelis.util;

import java.util.ArrayList;
import java.util.List;

public class Observavel<T> {

  private List<Observador<T>> listaDeObservadores = new ArrayList<Observador<T>>();

  protected Observavel() {
  }

  public void cadastrarObservador(
      Observador<T> observador) {
    listaDeObservadores.add(observador);
  }

  public void removerObservador(
      Observador<T> observador) {
    listaDeObservadores.remove(observador);
  }

  protected void dispararEvento(T parametro) {
    for (Observador<T> observador : listaDeObservadores) {
      observador.notificar(parametro);
    }
  }
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.util.Visitante

```
package br.ufsc.edugraf.maquinaTelis.util;

public interface Visitante<T> {

  void visitar(T dado);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.util.Observador

```
package br.ufsc.edugraf.maquinaTelis.util;

public interface Observador<T> {

  public abstract void notificar(T descriçãoDoEvento);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.traducoes.Constantes

```
package br.ufsc.edugraf.maquinaTelis.traducoes;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Constantes {
  private static final String BUNDLE_NAME = "br.ufsc.edugraf.maquinaTelis.traducoes

  private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
      .getBundle(BUNDLE_NAME);
```

```
  private Constantes() {
  }

  public static String obterString(String key) {
    try {
      return RESOURCE_BUNDLE.getString(key);
    } catch (MissingResourceException e) {
      return '!' + key + '!';
    }
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.traducoes.Mensagens

```
package br.ufsc.edugraf.maquinaTelis.traducoes;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Mensagens {
  private static final String BUNDLE_NAME = "br.ufsc.edugraf.maquinaTelis.traducoes

  private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
      .getBundle(BUNDLE_NAME);

  private Mensagens() {
  }

  public static String getString(String key) {
    try {
      return RESOURCE_BUNDLE.getString(key);
    } catch (MissingResourceException e) {
      return '!' + key + '!';
    }
```

```
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambiente.utils.AmbienteAbstrato

```
package br.ufsc.edugraf.maquinaTelis.ambiente.utils;

import java.io.Reader;

import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeReport
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaParaRepo
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErros;
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.ConteinerDeInsta
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Agenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.ModeloDeAtor;
import br.ufsc.edugraf.telis.palavras.ListaAbstrata;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.parser.FachadaDoParser;

public abstract class AmbienteAbstrato {

  protected Reader leitorDeCódigo;

  public AmbienteAbstrato(Reader leitorDeCódigo) {
    this.leitorDeCódigo = leitorDeCódigo;
  }

  public final void iniciarAplique()
      throws Exception {
    configurarAmbienteDeExecução();
    ListaAbstrata<Palavra> lista = lerPrograma(leitorDeCódigo);
```

```
  Ator ator = instanciarAtorInicial(lista);
  iniciarExecução(ator);
}

private void configurarAmbienteDeExecução() {
  ReportadorDeErros
      .fixarEstrategiaParaReportarErros(obterEstratégiaDeReportagemDeErros());
  ConteinerDeInstaladoresDeModulos instaladorDeModulos = new ConteinerDeInstalado
  cadastrarInstaladoresDeMódulos(instaladorDeModulos);
  Aplique.obterInstancia()
      .fixarInstaladorDeModulos(
          instaladorDeModulos);
}

protected abstract void cadastrarInstaladoresDeMódulos(
    ConteinerDeInstaladoresDeModulos instaladorDeModulos);

protected EstrategiaParaReportarErros obterEstratégiaDeReportagemDeErros() {
  return new EstrategiaDeReportagemNula();
}

protected void iniciarExecução(Ator ator) {
  ator.iniciarExecução();
}

private Ator instanciarAtorInicial(
    ListaAbstrata<Palavra> lista) {
  ModeloDeAtor modelo = new ModeloDeAtor(
      "modeloAnonimo");
  modelo.adicionarAgenda(new Agenda(
      ModeloDeAtor.NOME_AGENDA_INICIAR,
      new ListaDeComandos(lista)));
  Ator ator = Aplique.obterInstancia()
      .instanciarAtor(modelo);
  return ator;
```

```
  }

  private ListaAbstrata<Palavra> lerPrograma(
      Reader leitor) throws Exception {
    FachadaDoParser fachada = new FachadaDoParser();
    ListaAbstrata<Palavra> lista = fachada
        .parsearInstruções(leitor);
    return lista;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Associador

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis;

import java.util.Iterator;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.PalavraTextualAbstrata;

class Associador implements Executavel {
  private final MantenedorDeEscopos mantenedorDeEscopos;

  public Associador(
      MantenedorDeEscopos mantenedorDeEscopos) {
    this.mantenedorDeEscopos = mantenedorDeEscopos;
  }

  public void executar(IPilha pilha,
```

```
    ControladorDeExecucaoDaMaquina controlador) {
  if (pilha.topoDaPilhaÉ(Lista.class)) {
    Lista<Palavra> nome = pilha
        .desempilharComoLista();
    Lista<Palavra> valor = pilha
        .desempilharComoLista();
    associarListas(nome, valor);
  } else {
    PalavraTextualAbstrata nome = pilha
        .desempilharComoPalavraTextualAbstrata();
    Palavra valor = pilha.desempilhar();
    mantenedorDeEscopos.associarVariavel(nome
        .obterValor(), valor);
  }
}

private void associarListas(Lista<Palavra> nome,
    Lista<Palavra> valor) {
  if (nome.obterTamanho() != valor
      .obterTamanho())
    throw new ExcecaoTelis(
        "Número de váriaveis e valores são diferentes.");
  Iterator<Palavra> iteradorDeNomes = nome
      .obterValor().iterator();
  Iterator<Palavra> iteradorDeValores = valor
      .obterValor().iterator();
  while (iteradorDeNomes.hasNext()) {
    Palavra next = iteradorDeNomes.next();
    if (!(next instanceof PalavraTextualAbstrata))
      throw new ExcecaoTelis(
          "Lista de nomes de variáveis deve ser composta por símbolos e/ou textos
    PalavraTextualAbstrata n = (PalavraTextualAbstrata) next;
    mantenedorDeEscopos.associarVariavel(n
        .obterValor(), iteradorDeValores
        .next());
```

```
    }
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.LeitorDeVariavel

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Simbolo;

class LeitorDeVariavel implements Executavel {
  private final MantenedorDeEscopos mantenedorDeEscopos;
  private final Simbolo nomeVariavel;

  public LeitorDeVariavel(
      MantenedorDeEscopos mantenedorDeEscopos,
      Simbolo nomeVariavel) {
    this.mantenedorDeEscopos = mantenedorDeEscopos;
    this.nomeVariavel = nomeVariavel;
  }

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    pilha
        .empilhar(mantenedorDeEscopos
            .lerVariavel(nomeVariavel
                .obterValor()));
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.MantenedorDeEscopos

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis;
```

```
import java.util.LinkedList;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class MantenedorDeEscopos {

  private Escopo escopoAtual = new Escopo();

  private Escopo escopoBase = escopoAtual;

  private LinkedList<Escopo> escoposAnteriores = new LinkedList<Escopo>();

  private class AdaptadorParaResolvedor implements
      Resolvedor {

    public Executavel resolver(Resolvivel r) {
      if (r instanceof Simbolo) {
        Simbolo simbolo = (Simbolo) r;
        if (simbolo
            .obterValor()
            .equals(
                Constantes
                    .obterString("Primitivas.associar"))) //$NON-NLS-1$
          return new Associador(
              MantenedorDeEscopos.this);
        if (escopoAtual.variavelExiste(simbolo
            .obterValor()))
          return new LeitorDeVariavel(
              MantenedorDeEscopos.this,
```

```
            simbolo);
    }
    return null;
  }

}

public MantenedorDeEscopos() {
  escopoAtual = escopoBase = new Escopo();
}

public MantenedorDeEscopos(Escopo escopo) {
  escopoAtual = escopoBase = escopo;
}

public void associarVariavel(String nome,
    Palavra conteudo) {
  escopoAtual.associarVariavel(nome, conteudo);
}

public void iniciarEscopo() {
  escoposAnteriores.addFirst(escopoAtual);
  escopoAtual = escopoBase.criarSubescopo();
}

public void terminarEscopo() {
  escopoAtual = escoposAnteriores.removeFirst();
}

public IEscopo obterEscopo() {
  return escopoAtual;
}

public Palavra lerVariavel(String nome) {
  Palavra valorVariavel = escopoAtual
```

```
        .lerVariavel(nome);
    if (valorVariavel == null)
      throw new ExcecaoTelis();
    return valorVariavel;
  }

  public void iniciarSubescopo(IEscopo escopoSuperior) {
    escoposAnteriores.addFirst(escopoAtual);
    escopoAtual = escopoSuperior.criarSubescopo();
  }

  public Resolvedor adaptarParaResolvedor() {
    return new AdaptadorParaResolvedor();
  }

  public IEscopo obterEscopoBase() {
    return escopoBase;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.IEscopo

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis;

import br.ufsc.edugraf.telis.palavras.Palavra;

public interface IEscopo {

  public abstract Palavra lerVariavel(String nome);

  public abstract void associarVariavel(String nome,
      Palavra conteudo);

  public abstract boolean variavelExiste(String nome);
```

```
  public abstract Escopo criarSubescopo();

  public abstract void declararVariavel(String nome);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Escopo

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis;

import java.util.HashMap;
import java.util.Map;

import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Texto;

public class Escopo implements IEscopo {
  private Map<String, Palavra> variaveis = new HashMap<String, Palavra>();
  private IEscopo escopoSuperior;

  public Escopo() {
  }

  public Escopo(IEscopo superior) {
    escopoSuperior = superior;
  }

  public Palavra lerVariavel(String nome) {
    Palavra valor = variaveis.get(nome);
    if (valor == null && escopoSuperior != null)
      return escopoSuperior.lerVariavel(nome);
    return valor;
  }

  public void declararVariavel(String nome) {
```

```
    variaveis.put(nome, new Texto(""));
  }

  public void associarVariavel(String nome,
      Palavra conteudo) {
    if (!variaveis.containsKey(nome)
        && escopoSuperior != null
        && escopoSuperior.variavelExiste(nome)) {
      escopoSuperior.associarVariavel(nome,
          conteudo);
    } else {
      variaveis.put(nome, conteudo);
    }
  }

  public boolean variavelExiste(String nome) {
    return variaveis.containsKey(nome)
        || (escopoSuperior != null && escopoSuperior
            .variavelExiste(nome));
  }

  public Escopo criarSubescopo() {
    return new Escopo(this);
  }

  @Override
  public String toString() {
    return "Escopo: " + variaveis;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorCompost

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores;
```

```
import java.util.ArrayList;
import java.util.List;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.telis.Resolvivel;

public class ResolvedorComposto implements Resolvedor {

  private List<Resolvedor> resolvedores = new ArrayList<Resolvedor>();

  public Executavel resolver(Resolvivel r) {
    for (Resolvedor resolvedor : resolvedores) {
      Executavel possivelResposta = resolvedor
          .resolver(r);
      if (possivelResposta != null)
        return possivelResposta;
    }
    return null;
  }

  public void adicionarResolvedor(
      Resolvedor resolvedor) {
    resolvedores.add(resolvedor);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorNulo

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.telis.Resolvivel;
```

```
public class ResolvedorNulo implements Resolvedor {

  public Executavel resolver(Resolvivel r) {
    return null;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores;

import java.util.HashMap;
import java.util.Map;

import br.ufsc.edugraf.maquinaTelis.linguagem.anotacoes.Primitiva;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MultiplexadorDeExecutaveis;
import br.ufsc.edugraf.maquinaTelis.traducoes.Constantes;
import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class ResolvedorPorCasta implements Resolvedor {

  private Map<Simbolo, MultiplexadorDeExecutaveis> mapeamento = new HashMap<Simbolo
  private Resolvedor resolvedorDeVar = new ResolvedorNulo();
  private ResolvedorComposto resolvedoresGenéricos = new ResolvedorComposto();

  public Executavel resolver(Resolvivel r) {
    Executavel e = resolvedorDeVar.resolver(r);
    if (e != null)
      return e;
    e = mapeamento.get(r);
    if (e != null)
```

```
    return e;
  return resolvedoresGenéricos.resolver(r);
}

public void associarPrimitiva(Executavel e) {
  Primitiva anotaçãoPrimitiva = e.getClass()
      .getAnnotation(Primitiva.class);
  assert anotaçãoPrimitiva != null : "Executavel deve ser anotado com meta-dados
  Simbolo simbolo = new Simbolo(Constantes
      .obterString("Primitivas."
          + anotaçãoPrimitiva.nome()));
  associarPrimitiva(simbolo, e,
      anotaçãoPrimitiva.parametros());
}

public void associarPrimitiva(Simbolo simbolo,
    Executavel e,
    Class<? extends Palavra>... classes) {
  obterExecutavel(simbolo).incluirExecutavel(e,
      classes);
}

private MultiplexadorDeExecutaveis obterExecutavel(
    Simbolo simbolo) {
  MultiplexadorDeExecutaveis multiplexador = mapeamento
      .get(simbolo);
  if (multiplexador == null) {
    multiplexador = new MultiplexadorDeExecutaveis();
    mapeamento.put(simbolo, multiplexador);
  }
  return multiplexador;
}

public void fixarResolvedorDeVariaveis(
    Resolvedor resolvedorDeVar) {
```

```
    this.resolvedorDeVar = resolvedorDeVar;
  }

  public void adicionarResolvedorGenérico(
      Resolvedor resolvedor) {
    resolvedoresGenéricos
        .adicionarResolvedor(resolvedor);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.Resolvedor

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.telis.Resolvivel;

public interface Resolvedor {

  public abstract Executavel resolver(Resolvivel r);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import java.util.LinkedList;
import java.util.List;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Escopo;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MaquinaDePilha extends
```

```
  MaquinaDePilhaAbstrata {

private LinkedList<PedidoDeInterrupcao> pedidosDeInterrupção = new LinkedList<Ped
private static final ThreadLocal<MaquinaDePilha> maquinasExistentes = new ThreadL

public MaquinaDePilha() {
  this(new Pilha(), new Escopo());
}

public MaquinaDePilha(IPilha pilhaDeDados) {
  this(pilhaDeDados, new Escopo());
}

public MaquinaDePilha(Escopo escopo) {
  this(new Pilha(), escopo);
}

public MaquinaDePilha(Palavra parametro) {
  this(new Pilha(), new Escopo(), parametro);
}

public MaquinaDePilha(IPilha pilhaDeDados,
    Palavra parametro) {
  this(pilhaDeDados, new Escopo(), parametro);
}

public MaquinaDePilha(Escopo escopo,
    Palavra parametro) {
  this(new Pilha(), escopo, parametro);
}

public MaquinaDePilha(IPilha pilhaDeDados,
    Escopo escopo, Palavra parametro) {
  this(pilhaDeDados, escopo);
  if (parametro != null)
```

```
    pilha.obterPilhaAtual()
        .empilhar(parametro);
}

public MaquinaDePilha(IPilha pilhaDeDados,
    Escopo escopo) {
  super(pilhaDeDados, escopo);
}

@Override
public void executarAgendaIniciar(
    List<Palavra> códigoAgendaIniciar) {
  maquinasExistentes.set(this);
  super
      .executarAgendaIniciar(códigoAgendaIniciar);
}

@Override
protected void antesDeExecutarPalavra() {
  tratarInterrupções();
}

public void tratarInterrupções() {
  PedidoDeInterrupcao pedido = null;
  synchronized (this) {
    if (!pedidosDeInterrupção.isEmpty())
      pedido = pedidosDeInterrupção
          .removeLast();
  }
  executarPedidoDeInterrupção(pedido);
}

public void hibernar() {
  try {
    while (true) {
```

```
        tratarInterrupções();
        synchronized (this) {
          wait();
        }
      }
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
  }

  public synchronized void agendarInterrupção(
      PedidoDeInterrupcao pedido) {
    assert pedido != null;
    pedidosDeInterrupção.addFirst(pedido);
    notify();
  }

  public static MaquinaDePilha obterMaquinaDePilhaExecutando() {
    return maquinasExistentes.get();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.PedidoDeInterrupcao

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErros;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;

public class PedidoDeInterrupcao {
  private static final TratadorDeEventosDoTratamentoDeInterrupcao TRATADOR_NULO = n
    public void aoTerminarDeTratarInterrupcao() {
    }
```

```
  public void aoTerminarDevidoAExcecao(
      ExcecaoTelis e) {
    ReportadorDeErros.reportarErro(e);
  }
};

private final Closure tratador;
private final IPilha pilha;
private final TratadorDeEventosDoTratamentoDeInterrupcao tratadorDeEventos;

public PedidoDeInterrupcao(Closure tratador,
    IPilha pilha) {
  this(tratador, pilha, TRATADOR_NULO);
}

public PedidoDeInterrupcao(
    Closure tratador,
    IPilha pilha,
    TratadorDeEventosDoTratamentoDeInterrupcao tratadorDeEventos) {
  this.tratador = tratador;
  this.pilha = pilha;
  this.tratadorDeEventos = tratadorDeEventos;
}

public Closure obterTratador() {
  return tratador;
}

public IPilha obterPilha() {
  return pilha;
}

public void notificarTerminoDeExecucaoDaInterrupcao() {
  tratadorDeEventos
```

```
        .aoTerminarDeTratarInterrupcao();
  }

  public void notificarTerminoDeExecucaoDevidoAExcecao(
      ExcecaoTelis e) {
    tratadorDeEventos.aoTerminarDevidoAExcecao(e);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ConteinerDePilha

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import java.util.LinkedList;
import java.util.ListIterator;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

class ConteinerDePilha {

  private LinkedList<IPilha> pilhaDePilhas = new LinkedList<IPilha>();

  public ConteinerDePilha() {
    pilhaDePilhas.addLast(new Pilha());
  }

  public ConteinerDePilha(IPilha pilhaRaiz) {
    pilhaDePilhas.addLast(pilhaRaiz);
  }

  public IPilha obterPilhaAtual() {
    ListIterator<IPilha> iterador = pilhaDePilhas
        .listIterator();
    if (!iterador.hasNext())
      throw new AssertionError(
```

```
          "Nunca deve chegar aqui");
    return iterador.next();
  }

  public int obterTamanho() {
    return obterPilhaAtual().obterTamanho();
  }

  public void começarAUsarPilha(IPilha pilha) {
    pilhaDePilhas.addFirst(pilha);
  }

  public void voltarAUsarPilhaAnterior() {
    if (pilhaDePilhas.size() <= 1)
      throw new ExcecaoTelis();
    pilhaDePilhas.removeFirst();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaAbstrata

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import java.util.List;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoPalavraNaoEncontrada;
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorComposto;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Escopo;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.MantenedorDeEscopos;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.Resolvivel;
```

```
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;

public abstract class MaquinaDePilhaAbstrata implements
    IMaquinaDePilha {

  protected final ResolvedorComposto resolvedor = new ResolvedorComposto();
  protected final ConteinerDePilha pilha;
  private MantenedorDeEscopos mantenedorDeEscopos = new MantenedorDeEscopos();
  private String nomeDoModelo = "<nome do modelo não informado>";
  private GerenteDeRetorno gerenteDeRetorno = new GerenteDeRetorno();

  private class Controlador implements
      ControladorDeExecucaoDaMaquina {
    private Closure closure;

    public void executar(IAgenda agenda) {
      mantenedorDeEscopos.iniciarEscopo();
      try {
        MaquinaDePilhaAbstrata.this
            .executarCódigoComPontoDeRetorno(agenda
                .obterCódigo()
                .obterValor());
      } catch (ExcecaoTelis e) {
        e.notificarQuePassouPelaAgenda(agenda
            .obterNome(), nomeDoModelo);
        throw e;
      }
      mantenedorDeEscopos.terminarEscopo();
    }

    public void executarClosure(Closure closure) {
      closure
          .assegurarQuePodeExecutar(MaquinaDePilhaAbstrata.this);
      if (closure.obterMaquinaOndeFoiDeclarado() == MaquinaDePilhaAbstrata.this) {
```

```
    executarLocalmente(closure);
  } else {
    executarViaInterrupcao(closure, pilha
        .obterPilhaAtual());
  }
}

private void executarViaInterrupcao(
    Closure closure, IPilha pilha) {
  assert (MaquinaDePilhaAbstrata.this instanceof MaquinaDePilhaSimples) : "Exec
  PedidoDeInterrupcao pedido = new PedidoDeInterrupcao(
      closure,
      pilha,
      new TratadorDeEventosDoTratamentoDeInterrupcao() {
        public void aoTerminarDeTratarInterrupcao() {
        }

        public void aoTerminarDevidoAExcecao(
            ExcecaoTelis e) {
          throw e;
        }
      });
  IMaquinaDePilha maquinaDePilhaExecutando = closure
      .obterMaquinaOndeFoiDeclarado();
  maquinaDePilhaExecutando
      .agendarInterrupção(pedido);
  if (maquinaDePilhaExecutando instanceof MaquinaDePilha)
    ((MaquinaDePilha) maquinaDePilhaExecutando)
        .tratarInterrupções();
}

private void executarLocalmente(Closure closure) {
  mantenedorDeEscopos
      .iniciarSubescopo(closure
          .obterEscopo());
```

```
    MaquinaDePilhaAbstrata.this
        .executarCódigo(closure);
    mantenedorDeEscopos.terminarEscopo();
  }

  public Palavra avaliarExpressao(
      Closure closure, Palavra... parametros) {
    closure
        .assegurarQuePodeExecutar(MaquinaDePilhaAbstrata.this);
    Pilha pilhaDaExpressão = new Pilha(
        parametros);
    if (closure.obterMaquinaOndeFoiDeclarado() == MaquinaDePilhaAbstrata.this) {
      pilha
          .começarAUsarPilha(pilhaDaExpressão);
      executarLocalmente(closure);
      pilha.voltarAUsarPilhaAnterior();
    } else {
      executarViaInterrupcao(closure,
          pilhaDaExpressão);
    }
    if (pilhaDaExpressão.estaVazia())
      return null;
    return pilhaDaExpressão.desempilhar();
  }

  public void retornar() {
    gerenteDeRetorno.retornar(closure);
    gerenteDeRetorno
        .interromperExecuçãoSeEstiverRetornando();
  }
}

public MaquinaDePilhaAbstrata(IPilha pilhaDeDados,
    Escopo escopo) {
  mantenedorDeEscopos = new MantenedorDeEscopos(
```

```
      escopo);
  pilha = new ConteinerDePilha(pilhaDeDados);
  resolvedor
      .adicionarResolvedor(mantenedorDeEscopos
          .adaptarParaResolvedor());
}

public void executarAgendaIniciar(
    List<Palavra> códigoAgendaIniciar) {
  executarCódigoComPontoDeRetorno(códigoAgendaIniciar);
}

private void executarCódigoComPontoDeRetorno(
    List<Palavra> listaDeInstrucoes) {
  gerenteDeRetorno
      .notificarInicioDeCódigoRetornavel();
  executar(listaDeInstrucoes, new Controlador());
  gerenteDeRetorno.passouPorPontoDeRetorno();
}

private void executarCódigo(Closure closure) {
  Controlador controlador = new Controlador();
  controlador.closure = closure;
  executar(closure.obterValor(), controlador);
}

protected final void executar(
    List<Palavra> listaDeInstrucoes,
    ControladorDeExecucaoDaMaquina c) {
  for (Palavra palavra : listaDeInstrucoes) {
    if (Thread.currentThread().isInterrupted())
      break;

    antesDeExecutarPalavra();
    if (palavra instanceof Resolvivel) {
```

```
      Resolvivel r = (Resolvivel) palavra;
      delegarExecuçãoParaExecutavel(pilha
          .obterPilhaAtual(), r, c);
    } else if (palavra instanceof ListaDeComandos) {
      Closure closure = new Closure(this,
          (ListaDeComandos) palavra,
          mantenedorDeEscopos
              .obterEscopo());
      gerenteDeRetorno
          .notificarCriaçãoDeClosure(closure);
      pilha.obterPilhaAtual().empilhar(
          closure);
    } else {
      pilha.obterPilhaAtual().empilhar(
          palavra);
    }
  }
}

protected void antesDeExecutarPalavra() {
};

protected void delegarExecuçãoParaExecutavel(
    IPilha pilha, Resolvivel r,
    ControladorDeExecucaoDaMaquina controlador) {
  try {
    Executavel executavel = resolvedor
        .resolver(r);
    if (executavel == null)
      throw new ExcecaoPalavraNaoEncontrada(
          r);
    executavel.executar(pilha, controlador);
  } catch (ExcecaoTelis e) {
    e.fixarInstruçãoCausadora(r);
    throw e;
```

```
  } catch (Throwable e) {
    ExcecaoTelis excecaoTelis = new ExcecaoTelis(
        e);
    excecaoTelis.fixarInstruçãoCausadora(r);
    throw excecaoTelis;
  }
}

public Closure criarClosureComEscopoRaiz(
    ListaDeComandos comandos) {
  return new Closure(this, comandos,
      mantenedorDeEscopos.obterEscopoBase());
}

public void fixarResolvedorDeSimbolos(
    ResolvedorPorCasta resolvedor) {
  resolvedor
      .fixarResolvedorDeVariaveis(mantenedorDeEscopos
          .adaptarParaResolvedor());
  this.resolvedor
      .adicionarResolvedor(resolvedor);
}

protected final void executarPedidoDeInterrupção(
    PedidoDeInterrupcao pedido) {
  if (pedido != null) {
    Closure closure = pedido.obterTratador();
    mantenedorDeEscopos
        .iniciarSubescopo(closure
            .obterEscopo());
    try {
      pilha.começarAUsarPilha(pedido
          .obterPilha());
      this
          .executarCódigoComPontoDeRetorno(closure
```

```
                .obterValor());
        pilha.voltarAUsarPilhaAnterior();
      } catch (ExcecaoTelis e) {
        pedido
            .notificarTerminoDeExecucaoDevidoAExcecao(e);
        return;
      } finally {
        mantenedorDeEscopos.terminarEscopo();
      }
      pedido
          .notificarTerminoDeExecucaoDaInterrupcao();
    }
  }

  public void fixarNomeDoModelo(String nomeDoModelo) {
    this.nomeDoModelo = nomeDoModelo;
  }

  public String obterNomeDoModeloAtreladoAMaquina() {
    return nomeDoModelo;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCasta

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public interface InstalavelEmResolvedorPorCastas {

  public void instalar(ResolvedorPorCasta resolvedor);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.GerenteDeRetorno

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import java.util.LinkedList;
import java.util.WeakHashMap;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;

public class GerenteDeRetorno {

  private boolean retornar;

  private LinkedList<WeakHashMap<Closure, Object>> pilhaDeClosures = new LinkedList

  private Object closureRetornando;

  public boolean estáRetornando() {
    return retornar;
  }

  public void retornar() {
    retornar(null);
  }

  public void passouPorPontoDeRetorno() {
    WeakHashMap<Closure, Object> ultimo = invalidarClosures();
    if (estáRetornando()
        && (closureRetornando == null || ultimo
            .containsKey(closureRetornando)))
      limparEstadoDeRetorno();
  }
```

```
private void limparEstadoDeRetorno() {
  retornar = false;
  Thread.interrupted();
  closureRetornando = null;
}

private WeakHashMap<Closure, Object> invalidarClosures() {
  WeakHashMap<Closure, Object> ultimo = pilhaDeClosures
      .removeLast();
  for (Closure c : ultimo.keySet())
    if (c != null)
      c.notificarNãoPodeMaisRetornar();
  return ultimo;
}

public void notificarInicioDeCódigoRetornavel() {
  pilhaDeClosures
      .add(new WeakHashMap<Closure, Object>());
}

public void notificarCriaçãoDeClosure(
    Closure closure) {
  pilhaDeClosures.getLast().put(closure, null);
}

public void retornar(Closure closure) {
  if (closure != null && !closure.podeRetornar())
    throw new ExcecaoTelis(
        "Closure não pode mais retornar.");

  closureRetornando = closure;
  retornar = true;
}

public void passouPorPontoDeRetorno(
```

```
      GerenteDeRetorno próximoGerente) {
    invalidarClosures();
    if (estáRetornando()) {
      próximoGerente.retornar = true;
      próximoGerente.closureRetornando = closureRetornando;
      limparEstadoDeRetorno();
    }
  }

  public void interromperExecuçãoSeEstiverRetornando() {
    if (estáRetornando())
      Thread.currentThread().interrupt();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

public interface Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IMaquinaDePilha

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import java.util.List;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
```

```
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;

public interface IMaquinaDePilha {

  public abstract void executarAgendaIniciar(
      List<Palavra> códigoAgendaIniciar);

  public abstract Closure criarClosureComEscopoRaiz(
      ListaDeComandos comandos);

  public abstract void fixarResolvedorDeSimbolos(
      ResolvedorPorCasta resolvedor);

  public abstract void agendarInterrupção(
      PedidoDeInterrupcao pedido);

  public abstract void fixarNomeDoModelo(
      String nomeDoModelo);

  public abstract String obterNomeDoModeloAtreladoAMaquina();

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.PalavraTextualAbstrata;
import br.ufsc.edugraf.telis.palavras.Simbolo;
```

```
import br.ufsc.edugraf.telis.palavras.Texto;

public interface IPilha {

  public abstract void empilhar(Palavra palavra);

  public abstract Palavra desempilhar();

  public abstract Texto desempilharComoTexto();

  public abstract Numero desempilharComoNúmero();

  public abstract Referencia desempilharComoReferencia();

  public abstract Lista<Palavra> desempilharComoLista();

  public abstract Simbolo desempilharComoSímbolo();

  public abstract PalavraTextualAbstrata desempilharComoPalavraTextualAbstrata();

  public abstract boolean estaVazia();

  public abstract Booleano desempilharComoBooleano();

  public abstract int obterTamanho();

  public abstract Closure desempilharClosure();

  public abstract boolean topoDaPilhaÉ(
      Class<? extends Palavra>... classes);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ProxyDaPilha

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;
```

```
import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoPilhaVazia;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.PalavraTextualAbstrata;
import br.ufsc.edugraf.telis.palavras.Simbolo;
import br.ufsc.edugraf.telis.palavras.Texto;

import static java.lang.Math.min;

public class ProxyDaPilha implements IPilha {

  private IPilha pilha;
  private int tamanhoOriginal;
  private int numeroDeElementosCompartilhados;

  public ProxyDaPilha(IPilha pilha) {
    this(pilha, 0);
  }

  public ProxyDaPilha(IPilha pilha,
      int numeroDeElementosCompartilhados) {
    this.pilha = pilha;
    tamanhoOriginal = pilha.obterTamanho();
    this.numeroDeElementosCompartilhados = min(
        tamanhoOriginal,
        numeroDeElementosCompartilhados);
  }

  public Palavra desempilhar() {
    antesDeDesempilhar();
```

```
  return pilha.desempilhar();
}

public Closure desempilharClosure() {
  antesDeDesempilhar();
  return pilha.desempilharClosure();
}

public Referencia desempilharComoReferencia() {
  antesDeDesempilhar();
  return pilha.desempilharComoReferencia();
}

public Booleano desempilharComoBooleano() {
  antesDeDesempilhar();
  return pilha.desempilharComoBooleano();
}

public Lista<Palavra> desempilharComoLista() {
  antesDeDesempilhar();
  return pilha.desempilharComoLista();
}

public Numero desempilharComoNúmero() {
  antesDeDesempilhar();
  return pilha.desempilharComoNúmero();
}

public PalavraTextualAbstrata desempilharComoPalavraTextualAbstrata() {
  antesDeDesempilhar();
  return pilha
      .desempilharComoPalavraTextualAbstrata();
}

public Simbolo desempilharComoSímbolo() {
```

```
  antesDeDesempilhar();
  return pilha.desempilharComoSímbolo();
}

public Texto desempilharComoTexto() {
  antesDeDesempilhar();
  return pilha.desempilharComoTexto();
}

public boolean topoDaPilhaÉ(
    Class<? extends Palavra>... classes) {
  if (classes.length > obterTamanho())
    return false;
  return pilha.topoDaPilhaÉ(classes);
}

public void empilhar(Palavra palavra) {
  pilha.empilhar(palavra);
}

public boolean estaVazia() {
  return obterTamanho() == 0;
}

public int obterTamanho() {
  return pilha.obterTamanho()
      - obterNumeroDeDadosCompletamenteCongelados();
}

@Override
public String toString() {
  return pilha.toString();
}

private void antesDeDesempilhar() {
```

```
    if (pilha.obterTamanho() == obterNumeroDeDadosCompletamenteCongelados())
      throw new ExcecaoPilhaVazia();

    if (pilha.obterTamanho() == tamanhoOriginal) {
      numeroDeElementosCompartilhados--;
      tamanhoOriginal--;
    }
  }

  public void limparPilha() {
    while (pilha.obterTamanho() != tamanhoOriginal)
      pilha.desempilhar();
  }

  private int obterNumeroDeDadosCompletamenteCongelados() {
    return tamanhoOriginal
        - numeroDeElementosCompartilhados;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MultiplexadorDeExecutaveis

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import java.util.ArrayList;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTipoErrado;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MultiplexadorDeExecutaveis implements
    Executavel {
  private ArrayList<Executavel> executaveis = new ArrayList<Executavel>(
      1);
  private ArrayList<Class<? extends Palavra>[]> parametros = new ArrayList<Class<?
```

```
      1);

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Executavel e = obterExecutavelParaOsParametros(pilha);
    if (e != null)
      e.executar(pilha, controlador);
    else
      throw new ExcecaoTipoErrado();
  }

  public Executavel obterExecutavelParaOsParametros(
      IPilha pilha) {
    int i = 0;
    for (Class<? extends Palavra>[] p : parametros) {
      if (pilha.topoDaPilhaÉ(p))
        return executaveis.get(i);
      i++;
    }
    return null;
  }

  public void incluirExecutavel(Executavel e,
      Class<? extends Palavra>... classes) {
    executaveis.add(e);
    parametros.add(classes);
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaqu

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.IAgenda;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.telis.palavras.Booleano;
```

```
import br.ufsc.edugraf.telis.palavras.Palavra;

public interface ControladorDeExecucaoDaMaquina {

  void executar(IAgenda agenda);

  void executarClosure(Closure closure);

  Palavra avaliarExpressao(Closure closure,
      Palavra... parametros);

  void retornar();

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.Escopo;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class MaquinaDePilhaSimples extends
    MaquinaDePilhaAbstrata {

  public MaquinaDePilhaSimples() {
    this(new Pilha(), new Escopo());
  }

  public MaquinaDePilhaSimples(IPilha pilhaDeDados) {
    this(pilhaDeDados, new Escopo());
  }

  public MaquinaDePilhaSimples(Escopo escopo) {
    this(new Pilha(), escopo);
```

```
}

public MaquinaDePilhaSimples(Palavra parametro) {
  this(new Pilha(), new Escopo(), parametro);
}

public MaquinaDePilhaSimples(IPilha pilhaDeDados,
    Palavra parametro) {
  this(pilhaDeDados, new Escopo(), parametro);
}

public MaquinaDePilhaSimples(Escopo escopo,
    Palavra parametro) {
  this(new Pilha(), escopo, parametro);
}

public MaquinaDePilhaSimples(IPilha pilhaDeDados,
    Escopo escopo, Palavra parametro) {
  this(pilhaDeDados, escopo);
  if (parametro != null)
    pilha.obterPilhaAtual()
        .empilhar(parametro);
}

public MaquinaDePilhaSimples(IPilha pilhaDeDados,
    Escopo escopo) {
  super(pilhaDeDados, escopo);
}

public synchronized void agendarInterrupção(
    PedidoDeInterrupcao pedido) {
  assert pedido != null;
  executarPedidoDeInterrupção(pedido);
}
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.TratadorDeEventosDoTratamento

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;

public interface TratadorDeEventosDoTratamentoDeInterrupcao {

  void aoTerminarDeTratarInterrupcao();

  void aoTerminarDevidoAExcecao(ExcecaoTelis e);

}
```

Classe: br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Pilha

```
package br.ufsc.edugraf.maquinaTelis.maquinaDePilha;

import static br.ufsc.edugraf.maquinaTelis.palavras.Conversor.converter;

import java.util.LinkedList;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoPilhaVazia;
import br.ufsc.edugraf.maquinaTelis.palavras.Closure;
import br.ufsc.edugraf.maquinaTelis.palavras.Conversor;
import br.ufsc.edugraf.maquinaTelis.palavras.Referencia;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.PalavraTextualAbstrata;
import br.ufsc.edugraf.telis.palavras.Simbolo;
```

```
import br.ufsc.edugraf.telis.palavras.Texto;

public class Pilha implements IPilha {

  private LinkedList<Palavra> pilha = new LinkedList<Palavra>();

  public Pilha(Palavra... palavrasIniciais) {
    for (Palavra p : palavrasIniciais)
      this.empilhar(p);
  }

  public void empilhar(Palavra palavra) {
    if (palavra instanceof ListaDeComandos)
      throw new AssertionError(
          "Lista de comandos devem ser transformadas em Closures antes de serem emp
    pilha.addLast(palavra);
  }

  public Palavra desempilhar() {
    if (pilha.isEmpty())
      throw new ExcecaoPilhaVazia();
    return pilha.removeLast();
  }

  public Texto desempilharComoTexto() {
    return converter(desempilhar(), Texto.class);
  }

  public Numero desempilharComoNúmero() {
    return converter(desempilhar(), Numero.class);
  }

  @SuppressWarnings("unchecked")//$NON-NLS-1$
  public Lista<Palavra> desempilharComoLista() {
    return converter(desempilhar(), Lista.class);
```

```
}

public Simbolo desempilharComoSímbolo() {
  return converter(desempilhar(), Simbolo.class);
}

public PalavraTextualAbstrata desempilharComoPalavraTextualAbstrata() {
  return converter(desempilhar(),
      PalavraTextualAbstrata.class);
}

public Booleano desempilharComoBooleano() {
  return converter(desempilhar(), Booleano.class);
}

public Closure desempilharClosure() {
  return converter(desempilhar(), Closure.class);
}

public boolean estaVazia() {
  return pilha.isEmpty();
}

public int obterTamanho() {
  return pilha.size();
}

private Palavra espiarElemento(int n) {
  if (pilha.isEmpty())
    throw new ExcecaoPilhaVazia();
  return pilha.get(obterTamanho() - n - 1);
}

@Override
public String toString() {
```

```
    return "Pilha: " + pilha;
  }

  public boolean topoDaPilhaÉ(
      Class<? extends Palavra>... classes) {
    if (classes.length > obterTamanho())
      return false;

    boolean resposta = true;
    int i = classes.length - 1;
    for (Class<? extends Palavra> c : classes) {
      resposta &= Conversor.éPossívelConversão(
          espiarElemento(i), c);
      i--;
    }
    return resposta;
  }

  public Referencia desempilharComoReferencia() {
    return converter(desempilhar(),
        Referencia.class);
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.palavras.Closure

```
package br.ufsc.edugraf.maquinaTelis.palavras;

import java.util.List;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IMaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaAbstrata;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.variaveis.IEscopo;
```

```
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class Closure implements Palavra {

  private final IMaquinaDePilha maquinaDePilha;
  private final ListaDeComandos comandos;
  private final IEscopo escopo;
  private boolean podeRetornar = true;

  public Closure(IMaquinaDePilha origem,
      ListaDeComandos comandos, IEscopo escopo) {
    this.maquinaDePilha = origem;
    this.comandos = comandos;
    this.escopo = escopo;
  }

  public String comoTexto() {
    return comandos.comoTexto();
  }

  public IEscopo obterEscopo() {
    return escopo;
  }

  public List<Palavra> obterValor() {
    return comandos.obterValor();
  }

  public ListaDeComandos obterListaDeComandos() {
    return comandos;
  }

  public Closure criarClosureComEscopoIndependente() {
    IEscopo escopo = obterEscopo()
```

```
        .criarSubescopo();
    return new Closure(maquinaDePilha,
        obterListaDeComandos(), escopo);
  }

  public void assegurarQuePodeExecutar(
      IMaquinaDePilha m) {
    if (MaquinaDePilha
        .obterMaquinaDePilhaExecutando() == maquinaDePilha)
      return;
    if (maquinaDePilha != m)
      throw new ExcecaoTelis(
          "Listas de comando não podem ser executadas por outros atores além daquel
  }

  public IMaquinaDePilha obterMaquinaOndeFoiDeclarado() {
    return maquinaDePilha;
  }

  public boolean podeRetornar() {
    return podeRetornar;
  }

  public void notificarNãoPodeMaisRetornar() {
    podeRetornar = false;
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.palavras.Referencia

```
package br.ufsc.edugraf.maquinaTelis.palavras;

import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.UnidadeDeExecucao;
import br.ufsc.edugraf.telis.palavras.Palavra;
```

```
public class Referencia implements Palavra {

  private UnidadeDeExecucao u;

  public Referencia(UnidadeDeExecucao u) {
    this.u = u;
  }

  public UnidadeDeExecucao obterUnidadeReferenciada() {
    return u;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result
        + ((u == null) ? 0 : u.hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    final Referencia other = (Referencia) obj;
    if (u == null) {
      if (other.u != null)
        return false;
    } else if (!u.equals(other.u))
```

```
      return false;
    return true;
  }

  public String comoTexto() {
    return u.obterIdentidade().comoTexto();
  }

  @Override
  public String toString() {
    return comoTexto();
  }
}
```

Classe: br.ufsc.edugraf.maquinaTelis.palavras.Conversor

```
package br.ufsc.edugraf.maquinaTelis.palavras;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTipoErrado;
import br.ufsc.edugraf.telis.palavras.Booleano;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;
import br.ufsc.edugraf.telis.palavras.Texto;

public class Conversor {

  public static <T extends Palavra> boolean éPossívelConversão(
      Palavra palavra, Class<T> classe) {
    if (palavra instanceof Numero
        && Booleano.class.equals(classe))
      return true;

    if (palavra instanceof Texto
        && Simbolo.class.equals(classe))
```

```
      return true;

    return classe.isInstance(palavra);
  }

  public static <T extends Palavra> T converter(
      Palavra palavra, Class<T> classe) {
    if (palavra instanceof Numero
        && Booleano.class.equals(classe))
      return (T) new Booleano((Numero) palavra);

    if (palavra instanceof Texto
        && Simbolo.class.equals(classe))
      return (T) new Simbolo(((Texto) palavra)
          .obterValor());
    ;

    if (!classe.isInstance(palavra))
      throw new ExcecaoTipoErrado();
    return classe.cast(palavra);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.instaladorDeModulos.Instalador

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.instaladorDeModulos;

import br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas.ResolvedorDePrimit
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public class InstaladorDeModulosDoConsole implements
    InstaladorDeModulos {

  public void instalarMódulos(Ator ator,
      MaquinaDePilha maquina,
      ResolvedorPorCasta resolvedor) {
    new ResolvedorDePrimitivasDoConsole()
        .instalar(resolvedor);
  }

  public void instalarMódulos(Objeto objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
    new ResolvedorDePrimitivasDoConsole()
        .instalar(resolvedor);
  }

  public void instalarMódulos(ObjetoClasse objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
    new ResolvedorDePrimitivasDoConsole()
        .instalar(resolvedor);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.batch.Console

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.batch;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.Reader;
```

```
import br.ufsc.edugraf.maquinaTelis.ambiente.utils.AmbienteAbstrato;
import br.ufsc.edugraf.maquinaTelis.ambientes.console.instaladorDeModulos.Instalado
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeReport
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.ConteinerDeInsta
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu

public class Console extends AmbienteAbstrato {

  public Console(String arquivo) throws Exception {
    super(new BufferedReader(new FileReader(
        arquivo)));
  }

  protected Console(Reader leitorDeCódigo)
      throws Exception {
    super(leitorDeCódigo);
  }

  @Override
  protected void cadastrarInstaladoresDeMódulos(
      ConteinerDeInstaladoresDeModulos instaladorDeModulos) {
    instaladorDeModulos
        .adicionarInstalador(
            new InstaladorDeModulosBasicos())
        .adicionarInstalador(
            new InstaladorDeModulosDoConsole());
  }

  @Override
  protected EstrategiaDeReportagemParaStream obterEstratégiaDeReportagemDeErros() {
    return new EstrategiaDeReportagemParaStream(
        System.out);
  }
```

```
  public static void main(String[] args)
      throws Exception {
    if (args.length != 1) {
      System.out.println("Parametro inválido");
      System.exit(0);
    }
    Console console = new Console(args[0]);
    console.iniciarAplique();
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.linhaDeComando.LinhaDeCom

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.linhaDeComando;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.StringReader;
import java.util.Locale;
import java.util.Scanner;

import br.ufsc.edugraf.maquinaTelis.ambientes.console.batch.Console;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;

public class LinhaDeComando extends Console {

  public LinhaDeComando() throws Exception {
    super(new StringReader(""));
  }

  public static void main(String[] args)
      throws Exception {
    if (args.length == 1)
```

```
    Locale.setDefault(new Locale(args[0]));
  LinhaDeComando linhaDeComando = new LinhaDeComando();
  linhaDeComando.iniciarAplique();
}

@Override
protected void iniciarExecução(Ator ator) {
  Executor executor = new Executor(ator);
  Scanner leitor = new Scanner(System.in,
      "UTF-8");
  leitor.useDelimiter("\n");
  while (true) {
    System.out.print("> ");
    String comando = leitor.next();
    if (comando.startsWith("\\lerDeArquivo ")) {
      String caminho = comando
          .substring("\\lerDeArquivo "
              .length());
      executarOArquivo(caminho, executor);
    } else
      executor.executar(comando);
  }
}

private void executarOArquivo(String caminho,
    Executor executor) {
  try {
    BufferedReader leitor = new BufferedReader(
        new FileReader(caminho));
    String linhaAtual = leitor.readLine();
    while (linhaAtual != null) {
      System.out.println("> " + linhaAtual);
      executor.executar(linhaAtual);
      linhaAtual = leitor.readLine();
    }
```

```
    } catch (IOException e) {
      System.out.println("Erro ao ler arquivo.");
    }
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.linhaDeComando.Executor

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.linhaDeComando;

import java.io.StringReader;

import br.ufsc.edugraf.maquinaTelis.excecoes.ExcecaoTelis;
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErros;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.EventosDeCicloDeVidaDoA
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Programa;
import br.ufsc.edugraf.maquinaTelis.util.Observador;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.parser.FachadaDoParser;
import br.ufsc.edugraf.telis.parser.ParseException;
import br.ufsc.edugraf.telis.parser.TokenMgrError;

public class Executor {

  private FachadaDoParser fachada = new FachadaDoParser();
  private Ator ator;

  public Executor(Ator ator) {
    this.ator = ator;
    ator
        .cadastrarObservador(new Observador<EventosDeCicloDeVidaDoAtor>() {
```

```
          public void notificar(
              EventosDeCicloDeVidaDoAtor evento) {
            if (evento == EventosDeCicloDeVidaDoAtor.SUICIDOU) {
              System.out
                  .println("Saindo do interpretador Telis");
              System.exit(0);
            }
          }
        });
    ator.iniciarExecução();
  }

  public void executar(String comando) {
    try {
      ListaDeComandos comandos = fachada
          .parsearInstruções(new StringReader(
              comando));
      ator.executarInstruçõesAdHoc(comandos);
    } catch (ParseException e) {
      ReportadorDeErros
          .reportarErro(new ExcecaoTelis(
              "Comando inválido", e));
    } catch (TokenMgrError e) {
      ReportadorDeErros
          .reportarErro(new ExcecaoTelis(
              "Comando inválido", e));
    }
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas.ResolvedorDePrimiti

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas;
```

```
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.palavras.Simbolo;

public class ResolvedorDePrimitivasDoConsole implements
    InstalavelEmResolvedorPorCastas {

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new Simbolo(
        "mostrar"), new Mostrar());
    resolvedor.associarPrimitiva(
        new Simbolo("ler"), new Ler());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas.Mostrar

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class Mostrar implements Executavel {

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    Palavra palavra = pilha.desempilhar();
    System.out.println(palavra.comoTexto());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas.Ler

```
package br.ufsc.edugraf.maquinaTelis.ambientes.console.primitivas;

import java.util.Scanner;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.telis.palavras.Texto;

public class Ler implements Executavel {

  private Scanner scanner = new Scanner(System.in);

  public void executar(IPilha pilha,
      ControladorDeExecucaoDaMaquina controlador) {
    String texto = scanner.next();
    pilha.empilhar(new Texto(texto));
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.grafico.primitivas.ResolvedorDePrimitivasGraficas

```
package br.ufsc.edugraf.maquinaTelis.grafico.primitivas;

import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.ControladorDeExecucaoDaMaquina;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.Executavel;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.IPilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.InstalavelEmResolvedorPorCastas;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;
import br.ufsc.edugraf.telis.palavras.Numero;
import br.ufsc.edugraf.telis.palavras.Simbolo;
import edugraf.pichador.IPichador;
```

```
import edugraf.pichador.PainelDoPichador;

public class ResolvedorDePrimitivasGraficas implements
    InstalavelEmResolvedorPorCastas {

  private PainelDoPichador painelPichador;
  private IPichador pichador;

  private class ComRastros implements Executavel {
    public void executar(
        IPilha pilha,
        ControladorDeExecucaoDaMaquina controlador) {
      obterPichador().comRastros();
    }
  }

  private class Frente implements Executavel {
    public void executar(
        IPilha pilha,
        ControladorDeExecucaoDaMaquina controlador) {
      Numero número = pilha
          .desempilharComoNúmero();
      obterPichador().frente(
          número.obterParteInteira());
    }
  }

  private class Direita implements Executavel {
    public void executar(
        IPilha pilha,
        ControladorDeExecucaoDaMaquina controlador) {
      Numero número = pilha
          .desempilharComoNúmero();
      obterPichador().direita(
          número.obterParteInteira());
```

```
    }
  }

  public ResolvedorDePrimitivasGraficas(
      PainelDoPichador painelPichador) {
    this.painelPichador = painelPichador;
  }

  private IPichador obterPichador() {
    if (pichador == null)
      pichador = painelPichador.criarPichador();
    return pichador;
  }

  public void instalar(ResolvedorPorCasta resolvedor) {
    resolvedor.associarPrimitiva(new Simbolo(
        "comRastros"), new ComRastros());
    resolvedor.associarPrimitiva(new Simbolo(
        "frente"), new Frente());
    resolvedor.associarPrimitiva(new Simbolo(
        "direita"), new Direita());
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.grafico.ConfiguradorAmbiente

```
package br.ufsc.edugraf.maquinaTelis.grafico;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;

import edugraf.pichador.PainelDoPichador;
```

```
import br.ufsc.edugraf.maquinaTelis.ambiente.utils.AmbienteAbstrato;
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeReport
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaParaRepo
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.ConteinerDeInsta
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu

public class ConfiguradorAmbiente extends
    AmbienteAbstrato {

  private final PainelDoPichador painelPichador;

  public ConfiguradorAmbiente(String arquivo,
      PainelDoPichador painelPichador)
      throws FileNotFoundException {
    super(new BufferedReader(new FileReader(
        arquivo)));
    this.painelPichador = painelPichador;
  }

  @Override
  protected void cadastrarInstaladoresDeMódulos(
      ConteinerDeInstaladoresDeModulos instaladorDeModulos) {
    instaladorDeModulos
        .adicionarInstalador(
            new InstaladorDeModulosBasicos())
        .adicionarInstalador(
            new InstaladorDoModuloPichador(
                painelPichador));
  }

  @Override
  protected EstrategiaParaReportarErros obterEstratégiaDeReportagemDeErros() {
    return new EstrategiaDeReportagemParaStream(
        System.out);
  }
```

```
}
```

Classe: br.ufsc.edugraf.maquinaTelis.grafico.ConteinerSwing

```
package br.ufsc.edugraf.maquinaTelis.grafico;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;

import javax.swing.JFrame;

import edugraf.pichador.IPichador;
import edugraf.pichador.PainelDoPichador;
import edugraf.pichador.Pichador;

import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.EstrategiaDeReport
import br.ufsc.edugraf.maquinaTelis.excecoes.sistemaDeReportagem.ReportadorDeErros;
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.ConteinerDeInsta
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Aplique;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Agenda;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoEstatica.Modelo;
import br.ufsc.edugraf.telis.palavras.ListaAbstrata;
import br.ufsc.edugraf.telis.palavras.ListaDeComandos;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.parser.FachadaDoParser;
```

```
import br.ufsc.edugraf.telis.parser.ParseException;

public class ConteinerSwing extends JFrame {

  public ConteinerSwing(String arquivo) {
    setSize(500, 500);
    PainelDoPichador p = new PainelDoPichador(500,
        500, obterURLDoDiretórioAtual());
    this.add(p, BorderLayout.CENTER);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    ConfiguradorAmbiente configuradorAmbiente = null;
    try {
      configuradorAmbiente = new ConfiguradorAmbiente(
          arquivo, p);
    } catch (FileNotFoundException e) {
      throw new RuntimeException(e);
    }

    try {
      configuradorAmbiente.iniciarAplique();
    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }

  private URL obterURLDoDiretórioAtual() {
    String diretorioCorrente = System
        .getProperty("user.dir");
    File f = new File(diretorioCorrente);
    try {
      return f.toURL();
    } catch (MalformedURLException e1) {
      throw new AssertionError(
          "nunca deverá ocorrer");
```

```
    }
  }

  public static void main(String[] args)
      throws Exception {
    if (args.length != 1) {
      System.out.println("Parametro inválido");
      System.exit(0);
    }

    new ConteinerSwing(args[0]).setVisible(true);
  }

}
```

Classe: br.ufsc.edugraf.maquinaTelis.grafico.InstaladorDoModuloPichador

```
package br.ufsc.edugraf.maquinaTelis.grafico;

import edugraf.pichador.PainelDoPichador;
import br.ufsc.edugraf.maquinaTelis.grafico.primitivas.ResolvedorDePrimitivasGrafic
import br.ufsc.edugraf.maquinaTelis.linguagem.gerenciadorDeModulos.InstaladorDeModu
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Ator;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.Objeto;
import br.ufsc.edugraf.maquinaTelis.linguagem.visaoDinamica.ObjetoClasse;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilha;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.MaquinaDePilhaSimples;
import br.ufsc.edugraf.maquinaTelis.maquinaDePilha.resolvedores.ResolvedorPorCasta;

public class InstaladorDoModuloPichador implements
    InstaladorDeModulos {

  private PainelDoPichador painelPichador;

  public InstaladorDoModuloPichador(
```

```
      PainelDoPichador painelPichador) {
    this.painelPichador = painelPichador;
  }

  public void instalarMódulos(Ator ator,
      MaquinaDePilha maquina,
      ResolvedorPorCasta resolvedor) {
    new ResolvedorDePrimitivasGraficas(
        painelPichador).instalar(resolvedor);
  }

  public void instalarMódulos(Objeto objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
  }

  public void instalarMódulos(ObjetoClasse objeto,
      MaquinaDePilhaSimples maquina,
      ResolvedorPorCasta resolvedor) {
  }

}
```

Classe: br.ufsc.edugraf.telis.traducoes.Constantes

```
package br.ufsc.edugraf.telis.traducoes;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Constantes {
  private static final String BUNDLE_NAME = "br.ufsc.edugraf.telis.traducoes.lingua

  private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
      .getBundle(BUNDLE_NAME);
```

```
  private Constantes() {
  }

  public static String obterString(String key) {
    try {
      return RESOURCE_BUNDLE.getString(key);
    } catch (MissingResourceException e) {
      return '!' + key + '!';
    }
  }
}
```

Classe: br.ufsc.edugraf.telis.traducoes.Mensagens

```
package br.ufsc.edugraf.telis.traducoes;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Mensagens {
  private static final String BUNDLE_NAME = "br.ufsc.edugraf.telis.traducoes.erros"

  private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
      .getBundle(BUNDLE_NAME);

  private Mensagens() {
  }

  public static String getString(String key) {
    try {
      return RESOURCE_BUNDLE.getString(key);
    } catch (MissingResourceException e) {
      return '!' + key + '!';
    }
```

```
  }
}
```

Classe: br.ufsc.edugraf.telis.parser.TokenMgrError

```
package br.ufsc.edugraf.telis.parser;

public class TokenMgrError extends Error {
  /*
   * Ordinals for various reasons why an Error of this type can be thrown.
   */

  /**
   * Lexical error occured.
   */
  static final int LEXICAL_ERROR = 0;

  /**
   * An attempt wass made to create a second instance of a static token
   * manager.
   */
  static final int STATIC_LEXER_ERROR = 1;

  /**
   * Tried to change to an invalid lexical state.
   */
  static final int INVALID_LEXICAL_STATE = 2;

  /**
   * Detected (and bailed out of) an infinite loop in the token manager.
   */
  static final int LOOP_DETECTED = 3;

  /**
   * Indicates the reason why the exception is thrown. It will have one of the
```

```
 * above 4 values.
 */
int errorCode;

/**
 * Replaces unprintable characters by their espaced (or unicode escaped)
 * equivalents in the given string
 */
protected static final String addEscapes(String str) {
  StringBuffer retval = new StringBuffer();
  char ch;
  for (int i = 0; i < str.length(); i++) {
    switch (str.charAt(i)) {
    case 0:
      continue;
    case '\b':
      retval.append("\\b");
      continue;
    case '\t':
      retval.append("\\t");
      continue;
    case '\n':
      retval.append("\\n");
      continue;
    case '\f':
      retval.append("\\f");
      continue;
    case '\r':
      retval.append("\\r");
      continue;
    case '\"':
      retval.append("\\\"");
      continue;
    case '\'':
      retval.append("\\\'");
```

```
      continue;
    case '\\':
      retval.append("\\\\");
      continue;
    default:
      if ((ch = str.charAt(i)) < 0x20
          || ch > 0x7e) {
        String s = "0000"
            + Integer.toString(ch, 16);
        retval
            .append("\\u"
                + s
                    .substring(
                        s
                            .length() - 4,
                        s
                            .length()));
      } else {
        retval.append(ch);
      }
      continue;
    }
  }
  return retval.toString();
}

/**
 * Returns a detailed message for the Error when it is thrown by the token
 * manager to indicate a lexical error. Parameters : EOFSeen : indicates if
 * EOF caused the lexicl error curLexState : lexical state in which this
 * error occured errorLine : line number when the error occured errorColumn :
 * column number when the error occured errorAfter : prefix that was seen
 * before this error occured curchar : the offending character Note: You can
 * customize the lexical error message by modifying this method.
 */
```

```
protected static String LexicalError(
    boolean EOFSeen, int lexState,
    int errorLine, int errorColumn,
    String errorAfter, char curChar) {
  return ("Lexical error at line "
      + errorLine
      + ", column "
      + errorColumn
      + ".  Encountered: "
      + (EOFSeen ? "<EOF> "
          : ("\""
              + addEscapes(String
                  .valueOf(curChar)) + "\"")
              + " ("
              + (int) curChar
              + "), ")
      + "after : \""
      + addEscapes(errorAfter) + "\"");
}

/**
 * You can also modify the body of this method to customize your error
 * messages. For example, cases like LOOP_DETECTED and INVALID_LEXICAL_STATE
 * are not of end-users concern, so you can return something like :
 *
 * "Internal Error : Please file a bug report .... "
 *
 * from this method for such cases in the release version of your parser.
 */
public String getMessage() {
  return super.getMessage();
}

/*
 * Constructors of various flavors follow.
```

```
   */

  public TokenMgrError() {
  }

  public TokenMgrError(String message, int reason) {
    super(message);
    errorCode = reason;
  }

  public TokenMgrError(boolean EOFSeen,
      int lexState, int errorLine,
      int errorColumn, String errorAfter,
      char curChar, int reason) {
    this(LexicalError(EOFSeen, lexState,
        errorLine, errorColumn, errorAfter,
        curChar), reason);
  }
}
```

Classe: br.ufsc.edugraf.telis.parser.ConstrutorRepresentacaoInterna

```
package br.ufsc.edugraf.telis.parser;

import br.ufsc.edugraf.telis.palavras.ConstrutorDeLista;
import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Operador;
import br.ufsc.edugraf.telis.palavras.Palavra;
import br.ufsc.edugraf.telis.palavras.Simbolo;
import br.ufsc.edugraf.telis.traducoes.Constantes;

public class ConstrutorRepresentacaoInterna {

  private ConstrutorDeLista construtor = new ConstrutorDeLista();
```

```
public Lista<Palavra> obterSaida() {
  return construtor.obterSaidaComoLista();
}

public void adicionarTexto(Token token) {
  assert token.kind == ParserTelisConstants.TEXTO : "Tentando adicionar um token
  String texto = token.image;
  texto = texto.substring(1, texto.length() - 1);
  texto = texto.replaceAll("\\\\\"", "\""); //$NON-NLS-1$ //$NON-NLS-2$
  texto = texto.replaceAll("\\\\n", "\n"); //$NON-NLS-1$ //$NON-NLS-2$
  texto = texto.replaceAll("\\\\t", "\t"); //$NON-NLS-1$ //$NON-NLS-2$
  texto = texto.replaceAll("\\\\\\\\", "\\\\"); //$NON-NLS-1$ //$NON-NLS-2$
  construtor.adicionarTexto(texto);
}

public void adicionarNúmero(Token token) {
  assert token.kind == ParserTelisConstants.NUMERO;
  String numeroComoTexto = token.image
      .replaceAll(",", "."); //$NON-NLS-1$ //$NON-NLS-2$
  double numero = Double
      .parseDouble(numeroComoTexto);
  construtor.adicionarNúmero(numero);
}

public void iniciarLista() {
  construtor.iniciarLista();
}

public void terminarLista() {
  construtor.terminarLista();
}

public void terminarListaDeComandos() {
  construtor.terminarListaDeComando();
}
```

```
public void adicionarSimbolo(Token token) {
  assert token.kind == ParserTelisConstants.SIMBOLO;
  if (token.image
      .equals(Constantes
          .obterString("PalavraReservada.verdadeiro"))) //$NON-NLS-1$
    construtor.adicionarBooleano(true);
  else if (token.image
      .equals(Constantes
          .obterString("PalavraReservada.falso"))) //$NON-NLS-1$
    construtor.adicionarBooleano(false);
  else
    construtor.adicionarSímbolo(token.image);
}

public void adicionarOperador(Token token) {
  assert token.kind == ParserTelisConstants.OPERADOR;
  construtor.adicionarOperador(Operador
      .obterOperador(token.image));
}

public void adicionarOperadorDeEmpilhamento(
    Token token) {
  assert token.kind == ParserTelisConstants.SIMBOLO;
  construtor
      .adicionarOperadorDeEmpilhamento(token.image);
}

public void adicionarOperadorDeComunicacaoDireta(
    Token token) {
  assert token.kind == ParserTelisConstants.SIMBOLO;
  construtor
      .adicionarOperadorDeComunicacaoDireta(token.image);
}
```

```
}
```

Classe: br.ufsc.edugraf.telis.parser.ParseException

```
package br.ufsc.edugraf.telis.parser;

/**
 * This exception is thrown when parse errors are encountered. You can
 * explicitly create objects of this exception type by calling the method
 * generateParseException in the generated parser.
 *
 * You can modify this class to customize your error reporting mechanisms so
 * long as you retain the public fields.
 */
public class ParseException extends Exception {

  /**
   * This constructor is used by the method "generateParseException" in the
   * generated parser. Calling this constructor generates a new object of this
   * type with the fields "currentToken", "expectedTokenSequences", and
   * "tokenImage" set. The boolean flag "specialConstructor" is also set to
   * true to indicate that this constructor was used to create this object.
   * This constructor calls its super class with the empty string to force the
   * "toString" method of parent class "Throwable" to print the error message
   * in the form: ParseException: <result of getMessage>
   */
  public ParseException(Token currentTokenVal,
      int[][] expectedTokenSequencesVal,
      String[] tokenImageVal) {
    super("");
    specialConstructor = true;
    currentToken = currentTokenVal;
    expectedTokenSequences = expectedTokenSequencesVal;
    tokenImage = tokenImageVal;
  }
```

```
/**
 * The following constructors are for use by you for whatever purpose you
 * can think of. Constructing the exception in this manner makes the
 * exception behave in the normal way - i.e., as documented in the class
 * "Throwable". The fields "errorToken", "expectedTokenSequences", and
 * "tokenImage" do not contain relevant information. The JavaCC generated
 * code does not use these constructors.
 */

public ParseException() {
  super();
  specialConstructor = false;
}

public ParseException(String message) {
  super(message);
  specialConstructor = false;
}

/**
 * This variable determines which constructor was used to create this object
 * and thereby affects the semantics of the "getMessage" method (see below).
 */
protected boolean specialConstructor;

/**
 * This is the last token that has been consumed successfully. If this
 * object has been created due to a parse error, the token followng this
 * token will (therefore) be the first error token.
 */
public Token currentToken;

/**
 * Each entry in this array is an array of integers. Each array of integers
```

```
 * represents a sequence of tokens (by their ordinal values) that is
 * expected at this point of the parse.
 */
public int[][] expectedTokenSequences;

/**
 * This is a reference to the "tokenImage" array of the generated parser
 * within which the parse error occurred. This array is defined in the
 * generated ...Constants interface.
 */
public String[] tokenImage;

/**
 * This method has the standard behavior when this object has been created
 * using the standard constructors. Otherwise, it uses "currentToken" and
 * "expectedTokenSequences" to generate a parse error message and returns
 * it. If this object has been created due to a parse error, and you do not
 * catch it (it gets thrown from the parser), then this method is called
 * during the printing of the final stack trace, and hence the correct error
 * message gets displayed.
 */
public String getMessage() {
  if (!specialConstructor) {
    return super.getMessage();
  }
  StringBuffer expected = new StringBuffer();
  int maxSize = 0;
  for (int i = 0; i < expectedTokenSequences.length; i++) {
    if (maxSize < expectedTokenSequences[i].length) {
      maxSize = expectedTokenSequences[i].length;
    }
    for (int j = 0; j < expectedTokenSequences[i].length; j++) {
      expected
          .append(
              tokenImage[expectedTokenSequences[i][j]])
```

```
          .append(" ");
    }
    if (expectedTokenSequences[i][expectedTokenSequences[i].length - 1] != 0) {
      expected.append("...");
    }
    expected.append(eol).append("    ");
  }
  String retval = "Encountered \"";
  Token tok = currentToken.next;
  for (int i = 0; i < maxSize; i++) {
    if (i != 0)
      retval += " ";
    if (tok.kind == 0) {
      retval += tokenImage[0];
      break;
    }
    retval += add_escapes(tok.image);
    tok = tok.next;
  }
  retval += "\" at line "
      + currentToken.next.beginLine
      + ", column "
      + currentToken.next.beginColumn;
  retval += "." + eol;
  if (expectedTokenSequences.length == 1) {
    retval += "Was expecting:" + eol + "    ";
  } else {
    retval += "Was expecting one of:" + eol
        + "    ";
  }
  retval += expected.toString();
  return retval;
}

/**
```

```
 * The end of line string for this machine.
 */
protected String eol = System.getProperty(
    "line.separator", "\n");

/**
 * Used to convert raw characters to their escaped version when these raw
 * version cannot be used as part of an ASCII string literal.
 */
protected String add_escapes(String str) {
  StringBuffer retval = new StringBuffer();
  char ch;
  for (int i = 0; i < str.length(); i++) {
    switch (str.charAt(i)) {
    case 0:
      continue;
    case '\b':
      retval.append("\\b");
      continue;
    case '\t':
      retval.append("\\t");
      continue;
    case '\n':
      retval.append("\\n");
      continue;
    case '\f':
      retval.append("\\f");
      continue;
    case '\r':
      retval.append("\\r");
      continue;
    case '\"':
      retval.append("\\\"");
      continue;
    case '\'':
```

```
        retval.append("\\\'");
        continue;
      case '\\':
        retval.append("\\\\");
        continue;
      default:
        if ((ch = str.charAt(i)) < 0x20
            || ch > 0x7e) {
          String s = "0000"
              + Integer.toString(ch, 16);
          retval
              .append("\\u"
                  + s
                      .substring(
                          s
                              .length() - 4,
                          s
                              .length()));
        } else {
          retval.append(ch);
        }
        continue;
      }
    }
    return retval.toString();
  }

}
```

Classe: br.ufsc.edugraf.telis.parser.ParserTelisConstants

```
package br.ufsc.edugraf.telis.parser;

public interface ParserTelisConstants {
```

```
  int EOF = 0;
  int COMENTARIO = 5;
  int NUMERO = 6;
  int FLOAT = 7;
  int INTEIRO = 8;
  int DIGITO = 9;
  int TEXTO = 10;
  int CARACTER_ACEITO = 11;
  int OPERADOR = 12;
  int SIMBOLO = 13;
  int LETRA = 14;

  int DEFAULT = 0;

  String[] tokenImage = { "<EOF>", "\" \"",
      "\"\\t\"", "\"\\n\"", "\"\\r\"",
      "<COMENTARIO>", "<NUMERO>", "<FLOAT>",
      "<INTEIRO>", "<DIGITO>", "<TEXTO>",
      "<CARACTER_ACEITO>", "<OPERADOR>",
      "<SIMBOLO>", "<LETRA>", "\"$\"", "\".\"",
      "\"[\"", "\"]\"", "\"{\"", "\"}\"", };

}
```

Classe: br.ufsc.edugraf.telis.parser.JavaCharStream

```
package br.ufsc.edugraf.telis.parser;

/**
 * An implementation of interface CharStream, where the stream is assumed to
 * contain only ASCII characters (with java-like unicode escape processing).
 */

public class JavaCharStream {
  public static final boolean staticFlag = false;
```

```
static final int hexval(char c)
    throws java.io.IOException {
  switch (c) {
  case '0':
    return 0;
  case '1':
    return 1;
  case '2':
    return 2;
  case '3':
    return 3;
  case '4':
    return 4;
  case '5':
    return 5;
  case '6':
    return 6;
  case '7':
    return 7;
  case '8':
    return 8;
  case '9':
    return 9;

  case 'a':
  case 'A':
    return 10;
  case 'b':
  case 'B':
    return 11;
  case 'c':
  case 'C':
    return 12;
  case 'd':
```

```
  case 'D':
    return 13;
  case 'e':
  case 'E':
    return 14;
  case 'f':
  case 'F':
    return 15;
  }

  throw new java.io.IOException(); // Should never come here
}

public int bufpos = -1;
int bufsize;
int available;
int tokenBegin;
protected int bufline[];
protected int bufcolumn[];

protected int column = 0;
protected int line = 1;

protected boolean prevCharIsCR = false;
protected boolean prevCharIsLF = false;

protected java.io.Reader inputStream;

protected char[] nextCharBuf;
protected char[] buffer;
protected int maxNextCharInd = 0;
protected int nextCharInd = -1;
protected int inBuf = 0;
protected int tabSize = 8;
```

```
protected void setTabSize(int i) {
  tabSize = i;
}

protected int getTabSize(int i) {
  return tabSize;
}

protected void ExpandBuff(boolean wrapAround) {
  char[] newbuffer = new char[bufsize + 2048];
  int newbufline[] = new int[bufsize + 2048];
  int newbufcolumn[] = new int[bufsize + 2048];

  try {
    if (wrapAround) {
      System.arraycopy(buffer, tokenBegin,
          newbuffer, 0, bufsize
              - tokenBegin);
      System.arraycopy(buffer, 0, newbuffer,
          bufsize - tokenBegin, bufpos);
      buffer = newbuffer;

      System.arraycopy(bufline, tokenBegin,
          newbufline, 0, bufsize
              - tokenBegin);
      System.arraycopy(bufline, 0,
          newbufline, bufsize
              - tokenBegin, bufpos);
      bufline = newbufline;

      System.arraycopy(bufcolumn,
          tokenBegin, newbufcolumn, 0,
          bufsize - tokenBegin);
      System.arraycopy(bufcolumn, 0,
          newbufcolumn, bufsize
```

```
                - tokenBegin, bufpos);
      bufcolumn = newbufcolumn;

      bufpos += (bufsize - tokenBegin);
    } else {
      System.arraycopy(buffer, tokenBegin,
          newbuffer, 0, bufsize
              - tokenBegin);
      buffer = newbuffer;

      System.arraycopy(bufline, tokenBegin,
          newbufline, 0, bufsize
              - tokenBegin);
      bufline = newbufline;

      System.arraycopy(bufcolumn,
          tokenBegin, newbufcolumn, 0,
          bufsize - tokenBegin);
      bufcolumn = newbufcolumn;

      bufpos -= tokenBegin;
    }
  } catch (Throwable t) {
    throw new Error(t.getMessage());
  }

  available = (bufsize += 2048);
  tokenBegin = 0;
}

protected void FillBuff()
    throws java.io.IOException {
  int i;
  if (maxNextCharInd == 4096)
    maxNextCharInd = nextCharInd = 0;
```

```
  try {
    if ((i = inputStream.read(nextCharBuf,
        maxNextCharInd,
        4096 - maxNextCharInd)) == -1) {
      inputStream.close();
      throw new java.io.IOException();
    } else
      maxNextCharInd += i;
    return;
  } catch (java.io.IOException e) {
    if (bufpos != 0) {
      --bufpos;
      backup(0);
    } else {
      bufline[bufpos] = line;
      bufcolumn[bufpos] = column;
    }
    throw e;
  }
}

protected char ReadByte()
    throws java.io.IOException {
  if (++nextCharInd >= maxNextCharInd)
    FillBuff();

  return nextCharBuf[nextCharInd];
}

public char BeginToken()
    throws java.io.IOException {
  if (inBuf > 0) {
    --inBuf;
```

```
    if (++bufpos == bufsize)
      bufpos = 0;

    tokenBegin = bufpos;
    return buffer[bufpos];
  }

  tokenBegin = 0;
  bufpos = -1;

  return readChar();
}

protected void AdjustBuffSize() {
  if (available == bufsize) {
    if (tokenBegin > 2048) {
      bufpos = 0;
      available = tokenBegin;
    } else
      ExpandBuff(false);
  } else if (available > tokenBegin)
    available = bufsize;
  else if ((tokenBegin - available) < 2048)
    ExpandBuff(true);
  else
    available = tokenBegin;
}

protected void UpdateLineColumn(char c) {
  column++;

  if (prevCharIsLF) {
    prevCharIsLF = false;
    line += (column = 1);
  } else if (prevCharIsCR) {
```

```
    prevCharIsCR = false;
    if (c == '\n') {
      prevCharIsLF = true;
    } else
      line += (column = 1);
  }

  switch (c) {
  case '\r':
    prevCharIsCR = true;
    break;
  case '\n':
    prevCharIsLF = true;
    break;
  case '\t':
    column--;
    column += (tabSize - (column % tabSize));
    break;
  default:
    break;
  }

  bufline[bufpos] = line;
  bufcolumn[bufpos] = column;
}

public char readChar() throws java.io.IOException {
  if (inBuf > 0) {
    --inBuf;

    if (++bufpos == bufsize)
      bufpos = 0;

    return buffer[bufpos];
  }
```

```
char c;

if (++bufpos == available)
  AdjustBuffSize();

if ((buffer[bufpos] = c = ReadByte()) == '\\') {
  UpdateLineColumn(c);

  int backSlashCnt = 1;

  for (;;) // Read all the backslashes
  {
    if (++bufpos == available)
      AdjustBuffSize();

    try {
      if ((buffer[bufpos] = c = ReadByte()) != '\\') {
        UpdateLineColumn(c);
        // found a non-backslash char.
        if ((c == 'u')
            && ((backSlashCnt & 1) == 1)) {
          if (--bufpos < 0)
            bufpos = bufsize - 1;

          break;
        }

        backup(backSlashCnt);
        return '\\';
      }
    } catch (java.io.IOException e) {
      if (backSlashCnt > 1)
        backup(backSlashCnt);
```

```
        return '\\';
      }

      UpdateLineColumn(c);
      backSlashCnt++;
    }

    // Here, we have seen an odd number of backslash's followed by a 'u'
    try {
      while ((c = ReadByte()) == 'u')
        ++column;

      buffer[bufpos] = c = (char) (hexval(c) << 12
          | hexval(ReadByte()) << 8
          | hexval(ReadByte()) << 4 | hexval(ReadByte()));

      column += 4;
    } catch (java.io.IOException e) {
      throw new Error(
          "Invalid escape character at line "
              + line + " column "
              + column + ".");
    }

    if (backSlashCnt == 1)
      return c;
    else {
      backup(backSlashCnt - 1);
      return '\\';
    }
  } else {
    UpdateLineColumn(c);
    return (c);
  }
}
```

```
/**
 * @deprecated
 * @see #getEndColumn
 */

public int getColumn() {
  return bufcolumn[bufpos];
}

/**
 * @deprecated
 * @see #getEndLine
 */

public int getLine() {
  return bufline[bufpos];
}

public int getEndColumn() {
  return bufcolumn[bufpos];
}

public int getEndLine() {
  return bufline[bufpos];
}

public int getBeginColumn() {
  return bufcolumn[tokenBegin];
}

public int getBeginLine() {
  return bufline[tokenBegin];
}
```

```
public void backup(int amount) {

  inBuf += amount;
  if ((bufpos -= amount) < 0)
    bufpos += bufsize;
}

public JavaCharStream(java.io.Reader dstream,
    int startline, int startcolumn,
    int buffersize) {
  inputStream = dstream;
  line = startline;
  column = startcolumn - 1;

  available = bufsize = buffersize;
  buffer = new char[buffersize];
  bufline = new int[buffersize];
  bufcolumn = new int[buffersize];
  nextCharBuf = new char[4096];
}

public JavaCharStream(java.io.Reader dstream,
    int startline, int startcolumn) {
  this(dstream, startline, startcolumn, 4096);
}

public JavaCharStream(java.io.Reader dstream) {
  this(dstream, 1, 1, 4096);
}

public void ReInit(java.io.Reader dstream,
    int startline, int startcolumn,
    int buffersize) {
  inputStream = dstream;
  line = startline;
```

```
  column = startcolumn - 1;

  if (buffer == null
      || buffersize != buffer.length) {
    available = bufsize = buffersize;
    buffer = new char[buffersize];
    bufline = new int[buffersize];
    bufcolumn = new int[buffersize];
    nextCharBuf = new char[4096];
  }
  prevCharIsLF = prevCharIsCR = false;
  tokenBegin = inBuf = maxNextCharInd = 0;
  nextCharInd = bufpos = -1;
}

public void ReInit(java.io.Reader dstream,
    int startline, int startcolumn) {
  ReInit(dstream, startline, startcolumn, 4096);
}

public void ReInit(java.io.Reader dstream) {
  ReInit(dstream, 1, 1, 4096);
}

public JavaCharStream(java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn, int buffersize)
    throws java.io.UnsupportedEncodingException {
  this(
      encoding == null ? new java.io.InputStreamReader(
          dstream)
          : new java.io.InputStreamReader(
              dstream, encoding),
      startline, startcolumn, buffersize);
}
```

```
public JavaCharStream(java.io.InputStream dstream,
    int startline, int startcolumn,
    int buffersize) {
  this(new java.io.InputStreamReader(dstream),
      startline, startcolumn, 4096);
}

public JavaCharStream(java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn)
    throws java.io.UnsupportedEncodingException {
  this(dstream, encoding, startline,
      startcolumn, 4096);
}

public JavaCharStream(java.io.InputStream dstream,
    int startline, int startcolumn) {
  this(dstream, startline, startcolumn, 4096);
}

public JavaCharStream(java.io.InputStream dstream,
    String encoding)
    throws java.io.UnsupportedEncodingException {
  this(dstream, encoding, 1, 1, 4096);
}

public JavaCharStream(java.io.InputStream dstream) {
  this(dstream, 1, 1, 4096);
}

public void ReInit(java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn, int buffersize)
    throws java.io.UnsupportedEncodingException {
```

```
  ReInit(
      encoding == null ? new java.io.InputStreamReader(
          dstream)
          : new java.io.InputStreamReader(
              dstream, encoding),
      startline, startcolumn, buffersize);
}

public void ReInit(java.io.InputStream dstream,
    int startline, int startcolumn,
    int buffersize) {
  ReInit(new java.io.InputStreamReader(dstream),
      startline, startcolumn, buffersize);
}

public void ReInit(java.io.InputStream dstream,
    String encoding, int startline,
    int startcolumn)
    throws java.io.UnsupportedEncodingException {
  ReInit(dstream, encoding, startline,
      startcolumn, 4096);
}

public void ReInit(java.io.InputStream dstream,
    int startline, int startcolumn) {
  ReInit(dstream, startline, startcolumn, 4096);
}

public void ReInit(java.io.InputStream dstream,
    String encoding)
    throws java.io.UnsupportedEncodingException {
  ReInit(dstream, encoding, 1, 1, 4096);
}

public void ReInit(java.io.InputStream dstream) {
```

```
  ReInit(dstream, 1, 1, 4096);
}

public String GetImage() {
  if (bufpos >= tokenBegin)
    return new String(buffer, tokenBegin,
        bufpos - tokenBegin + 1);
  else
    return new String(buffer, tokenBegin,
        bufsize - tokenBegin)
        + new String(buffer, 0, bufpos + 1);
}

public char[] GetSuffix(int len) {
  char[] ret = new char[len];

  if ((bufpos + 1) >= len)
    System.arraycopy(buffer, bufpos - len + 1,
        ret, 0, len);
  else {
    System.arraycopy(buffer, bufsize
        - (len - bufpos - 1), ret, 0, len
        - bufpos - 1);
    System.arraycopy(buffer, 0, ret, len
        - bufpos - 1, bufpos + 1);
  }

  return ret;
}

public void Done() {
  nextCharBuf = null;
  buffer = null;
  bufline = null;
  bufcolumn = null;
```

```
}

/**
 * Method to adjust line and column numbers for the start of a token.
 */
public void adjustBeginLineColumn(int newLine,
    int newCol) {
  int start = tokenBegin;
  int len;

  if (bufpos >= tokenBegin) {
    len = bufpos - tokenBegin + inBuf + 1;
  } else {
    len = bufsize - tokenBegin + bufpos + 1
        + inBuf;
  }

  int i = 0, j = 0, k = 0;
  int nextColDiff = 0, columnDiff = 0;

  while (i < len
      && bufline[j = start % bufsize] == bufline[k = ++start
          % bufsize]) {
    bufline[j] = newLine;
    nextColDiff = columnDiff + bufcolumn[k]
        - bufcolumn[j];
    bufcolumn[j] = newCol + columnDiff;
    columnDiff = nextColDiff;
    i++;
  }

  if (i < len) {
    bufline[j] = newLine++;
    bufcolumn[j] = newCol + columnDiff;
```

```
      while (i++ < len) {
        if (bufline[j = start % bufsize] != bufline[++start
            % bufsize])
          bufline[j] = newLine++;
        else
          bufline[j] = newLine;
      }
    }

    line = bufline[j];
    column = bufcolumn[j];
  }

}
```

Classe: br.ufsc.edugraf.telis.parser.ParserTelis

```
package br.ufsc.edugraf.telis.parser;

import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class ParserTelis implements
    ParserTelisConstants {
  private ConstrutorRepresentacaoInterna construtor = new ConstrutorRepresentacaoIn

  public Lista<Palavra> obterResultado() {
    return construtor.obterSaida();
  }

  final public void instrucoes()
      throws ParseException {
    label_1: while (true) {
      switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
      case NUMERO:
```

```
    case TEXTO:
    case OPERADOR:
    case SIMBOLO:
    case 15:
    case 16:
    case 17:
    case 19:
      ;
      break;
    default:
      jj_la1[0] = jj_gen;
      break label_1;
    }
    instrucao();
  }
  jj_consume_token(0);
}

final public void instrucao()
    throws ParseException {
  switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
  case TEXTO:
    texto();
    break;
  case OPERADOR:
    operador();
    break;
  case 15:
    operadorDeEmpilhamento();
    break;
  case 16:
    operadorDeComunicacaoDireta();
    break;
  case NUMERO:
    numero();
```

```
    break;
  case SIMBOLO:
    simbolo();
    break;
  case 17:
    lista();
    break;
  case 19:
    bloco();
    break;
  default:
    jj_la1[1] = jj_gen;
    jj_consume_token(-1);
    throw new ParseException();
  }
}

final public void texto() throws ParseException {
  jj_consume_token(TEXTO);
  construtor.adicionarTexto(token);
}

final public void numero() throws ParseException {
  jj_consume_token(NUMERO);
  construtor.adicionarNúmero(token);
}

final public void simbolo() throws ParseException {
  jj_consume_token(SIMBOLO);
  construtor.adicionarSimbolo(token);
}

final public void operador() throws ParseException {
  jj_consume_token(OPERADOR);
  construtor.adicionarOperador(token);
```

```
}

final public void operadorDeEmpilhamento()
    throws ParseException {
  jj_consume_token(15);
  jj_consume_token(SIMBOLO);
  construtor
      .adicionarOperadorDeEmpilhamento(token);
}

final public void operadorDeComunicacaoDireta()
    throws ParseException {
  jj_consume_token(16);
  jj_consume_token(SIMBOLO);
  construtor
      .adicionarOperadorDeComunicacaoDireta(token);
}

final public void lista() throws ParseException {
  construtor.iniciarLista();
  jj_consume_token(17);
  label_2: while (true) {
    switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
    case NUMERO:
    case TEXTO:
    case OPERADOR:
    case SIMBOLO:
    case 15:
    case 16:
    case 17:
    case 19:
      ;
      break;
    default:
      jj_la1[2] = jj_gen;
```

```
      break label_2;
    }
    instrucao();
  }
  jj_consume_token(18);
  construtor.terminarLista();
}

final public void bloco() throws ParseException {
  construtor.iniciarLista();
  jj_consume_token(19);
  label_3: while (true) {
    switch ((jj_ntk == -1) ? jj_ntk() : jj_ntk) {
    case NUMERO:
    case TEXTO:
    case OPERADOR:
    case SIMBOLO:
    case 15:
    case 16:
    case 17:
    case 19:
      ;
      break;
    default:
      jj_la1[3] = jj_gen;
      break label_3;
    }
    instrucao();
  }
  jj_consume_token(20);
  construtor.terminarListaDeComandos();
}

public ParserTelisTokenManager token_source;
JavaCharStream jj_input_stream;
```

```
public Token token, jj_nt;
private int jj_ntk;
private int jj_gen;
final private int[] jj_la1 = new int[4];
static private int[] jj_la1_0;
static {
  jj_la1_0();
}

private static void jj_la1_0() {
  jj_la1_0 = new int[] { 0xbb440, 0xbb440,
      0xbb440, 0xbb440, };
}

public ParserTelis(java.io.InputStream stream) {
  this(stream, null);
}

public ParserTelis(java.io.InputStream stream,
    String encoding) {
  try {
    jj_input_stream = new JavaCharStream(
        stream, encoding, 1, 1);
  } catch (java.io.UnsupportedEncodingException e) {
    throw new RuntimeException(e);
  }
  token_source = new ParserTelisTokenManager(
      jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 4; i++)
    jj_la1[i] = -1;
}
```

```
public void ReInit(java.io.InputStream stream) {
  ReInit(stream, null);
}

public void ReInit(java.io.InputStream stream,
    String encoding) {
  try {
    jj_input_stream.ReInit(stream, encoding,
        1, 1);
  } catch (java.io.UnsupportedEncodingException e) {
    throw new RuntimeException(e);
  }
  token_source.ReInit(jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 4; i++)
    jj_la1[i] = -1;
}

public ParserTelis(java.io.Reader stream) {
  jj_input_stream = new JavaCharStream(stream,
      1, 1);
  token_source = new ParserTelisTokenManager(
      jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 4; i++)
    jj_la1[i] = -1;
}

public void ReInit(java.io.Reader stream) {
  jj_input_stream.ReInit(stream, 1, 1);
  token_source.ReInit(jj_input_stream);
```

```
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 4; i++)
    jj_la1[i] = -1;
}

public ParserTelis(ParserTelisTokenManager tm) {
  token_source = tm;
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 4; i++)
    jj_la1[i] = -1;
}

public void ReInit(ParserTelisTokenManager tm) {
  token_source = tm;
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 4; i++)
    jj_la1[i] = -1;
}

final private Token jj_consume_token(int kind)
    throws ParseException {
  Token oldToken;
  if ((oldToken = token).next != null)
    token = token.next;
  else
    token = token.next = token_source
        .getNextToken();
  jj_ntk = -1;
  if (token.kind == kind) {
```

```
    jj_gen++;
    return token;
  }
  token = oldToken;
  jj_kind = kind;
  throw generateParseException();
}

final public Token getNextToken() {
  if (token.next != null)
    token = token.next;
  else
    token = token.next = token_source
        .getNextToken();
  jj_ntk = -1;
  jj_gen++;
  return token;
}

final public Token getToken(int index) {
  Token t = token;
  for (int i = 0; i < index; i++) {
    if (t.next != null)
      t = t.next;
    else
      t = t.next = token_source
          .getNextToken();
  }
  return t;
}

final private int jj_ntk() {
  if ((jj_nt = token.next) == null)
    return (jj_ntk = (token.next = token_source
        .getNextToken()).kind);
```

```
  else
    return (jj_ntk = jj_nt.kind);
}

private java.util.Vector<int[]> jj_expentries = new java.util.Vector<int[]>();
private int[] jj_expentry;
private int jj_kind = -1;

public ParseException generateParseException() {
  jj_expentries.removeAllElements();
  boolean[] la1tokens = new boolean[21];
  if (jj_kind >= 0) {
    la1tokens[jj_kind] = true;
    jj_kind = -1;
  }
  for (int i = 0; i < 4; i++) {
    if (jj_la1[i] == jj_gen) {
      for (int j = 0; j < 32; j++) {
        if ((jj_la1_0[i] & (1 << j)) != 0) {
          la1tokens[j] = true;
        }
      }
    }
  }
  for (int i = 0; i < 21; i++) {
    if (la1tokens[i]) {
      jj_expentry = new int[1];
      jj_expentry[0] = i;
      jj_expentries.addElement(jj_expentry);
    }
  }
  int[][] exptokseq = new int[jj_expentries
      .size()][];
  for (int i = 0; i < jj_expentries.size(); i++) {
    exptokseq[i] = jj_expentries.elementAt(i);
```

```
    }
    return new ParseException(token, exptokseq,
        tokenImage);
  }

  final public void enable_tracing() {
  }

  final public void disable_tracing() {
  }

}
```

Classe: br.ufsc.edugraf.telis.parser.ParserTelisTokenManager

```
package br.ufsc.edugraf.telis.parser;

import br.ufsc.edugraf.telis.palavras.Lista;
import br.ufsc.edugraf.telis.palavras.Palavra;

public class ParserTelisTokenManager implements
    ParserTelisConstants {
  public java.io.PrintStream debugStream = System.out;

  public void setDebugStream(java.io.PrintStream ds) {
    debugStream = ds;
  }

  private final int jjStopStringLiteralDfa_0(
      int pos, long active0) {
    switch (pos) {
    default:
      return -1;
    }
  }
```

```
private final int jjStartNfa_0(int pos,
    long active0) {
  return jjMoveNfa_0(jjStopStringLiteralDfa_0(
      pos, active0), pos + 1);
}

private final int jjStopAtPos(int pos, int kind) {
  jjmatchedKind = kind;
  jjmatchedPos = pos;
  return pos + 1;
}

private final int jjStartNfaWithStates_0(int pos,
    int kind, int state) {
  jjmatchedKind = kind;
  jjmatchedPos = pos;
  try {
    curChar = input_stream.readChar();
  } catch (java.io.IOException e) {
    return pos + 1;
  }
  return jjMoveNfa_0(state, pos + 1);
}

private final int jjMoveStringLiteralDfa0_0() {
  switch (curChar) {
  case 36:
    return jjStopAtPos(0, 15);
  case 46:
    return jjStopAtPos(0, 16);
  case 91:
    return jjStopAtPos(0, 17);
  case 93:
    return jjStopAtPos(0, 18);
```

```
  case 123:
    return jjStopAtPos(0, 19);
  case 125:
    return jjStopAtPos(0, 20);
  default:
    return jjMoveNfa_0(0, 0);
  }
}

private final void jjCheckNAdd(int state) {
  if (jjrounds[state] != jjround) {
    jjstateSet[jjnewStateCnt++] = state;
    jjrounds[state] = jjround;
  }
}

private final void jjAddStates(int start, int end) {
  do {
    jjstateSet[jjnewStateCnt++] = jjnextStates[start];
  } while (start++ != end);
}

private final void jjCheckNAddTwoStates(
    int state1, int state2) {
  jjCheckNAdd(state1);
  jjCheckNAdd(state2);
}

private final void jjCheckNAddStates(int start,
    int end) {
  do {
    jjCheckNAdd(jjnextStates[start]);
  } while (start++ != end);
}
```

```
private final void jjCheckNAddStates(int start) {
  jjCheckNAdd(jjnextStates[start]);
  jjCheckNAdd(jjnextStates[start + 1]);
}

static final long[] jjbitVec0 = {
    0xfffffffffffffffeL, 0xffffffffffffffffL,
    0xffffffffffffffffL, 0xffffffffffffffffL };
static final long[] jjbitVec2 = { 0x0L, 0x0L,
    0xffffffffffffffffL, 0xffffffffffffffffL };
static final long[] jjbitVec3 = { 0x0L, 0x0L,
    0x0L, 0x1e7cff9f1e7cff9fL };

private final int jjMoveNfa_0(int startState,
    int curPos) {
  int[] nextStates;
  int startsAt = 0;
  jjnewStateCnt = 28;
  int i = 1;
  jjstateSet[0] = startState;
  int j, kind = 0x7fffffff;
  for (;;) {
    if (++jjround == 0x7fffffff)
      ReInitRounds();
    if (curChar < 64) {
      long l = 1L << curChar;
      MatchLoop: do {
        switch (jjstateSet[--i]) {
        case 0:
          if ((0x3ff000000000000L & l) != 0L) {
            if (kind > 6)
              kind = 6;
            jjCheckNAddStates(0, 6);
          } else if ((0x7000ac0000000000L & l) != 0L) {
            if (kind > 12)
```

```
      kind = 12;
  } else if (curChar == 34)
    jjCheckNAddStates(7, 9);
  else if (curChar == 40)
    jjCheckNAddTwoStates(1, 2);
  if (curChar == 62)
    jjCheckNAdd(22);
  else if (curChar == 60)
    jjCheckNAdd(22);
  else if (curChar == 45)
    jjstateSet[jjnewStateCnt++] = 4;
  break;
case 1:
  jjCheckNAddTwoStates(1, 2);
  break;
case 2:
  if (curChar == 41 && kind > 5)
    kind = 5;
  break;
case 3:
  if (curChar == 45)
    jjstateSet[jjnewStateCnt++] = 4;
  break;
case 4:
  if ((0x3ff000000000000L & l) == 0L)
    break;
  if (kind > 6)
    kind = 6;
  jjCheckNAddStates(0, 6);
  break;
case 5:
  if ((0x3ff000000000000L & l) == 0L)
    break;
  if (kind > 6)
    kind = 6;
```

```
    jjCheckNAddTwoStates(5, 6);
    break;
  case 7:
    if ((0x280000000000L & l) != 0L)
      jjCheckNAdd(8);
    break;
  case 8:
    if ((0x3ff000000000000L & l) == 0L)
      break;
    if (kind > 6)
      kind = 6;
    jjCheckNAdd(8);
    break;
  case 9:
    if ((0x3ff000000000000L & l) == 0L)
      break;
    if (kind > 6)
      kind = 6;
    jjCheckNAddStates(10, 12);
    break;
  case 10:
    if (curChar == 44)
      jjCheckNAdd(11);
    break;
  case 11:
    if ((0x3ff000000000000L & l) == 0L)
      break;
    if (kind > 6)
      kind = 6;
    jjCheckNAddTwoStates(11, 6);
    break;
  case 12:
    if ((0x3ff000000000000L & l) == 0L)
      break;
    if (kind > 6)
```

```
    kind = 6;
  jjCheckNAddTwoStates(12, 13);
  break;
case 13:
  if (curChar == 44)
    jjCheckNAdd(14);
  break;
case 14:
  if ((0x3ff000000000000L & l) == 0L)
    break;
  if (kind > 6)
    kind = 6;
  jjCheckNAdd(14);
  break;
case 15:
  if ((0x3ff000000000000L & l) == 0L)
    break;
  if (kind > 6)
    kind = 6;
  jjCheckNAdd(15);
  break;
case 16:
case 19:
  if (curChar == 34)
    jjCheckNAddStates(7, 9);
  break;
case 17:
  if ((0xfffffffbffffdbffL & l) != 0L)
    jjCheckNAddStates(7, 9);
  break;
case 20:
  if (curChar == 34 && kind > 10)
    kind = 10;
  break;
case 21:
```

```
        if ((0x7000ac0000000000L & l) != 0L
            && kind > 12)
          kind = 12;
        break;
      case 22:
        if (curChar == 61 && kind > 12)
          kind = 12;
        break;
      case 24:
        if (curChar == 60)
          jjCheckNAdd(22);
        break;
      case 25:
        if (curChar == 62)
          jjCheckNAdd(22);
        break;
      case 27:
        if ((0x87ff000200000000L & l) == 0L)
          break;
        if (kind > 13)
          kind = 13;
        jjstateSet[jjnewStateCnt++] = 27;
        break;
      default:
        break;
      }
    } while (i != startsAt);
  } else if (curChar < 128) {
    long l = 1L << (curChar & 077);
    MatchLoop: do {
      switch (jjstateSet[--i]) {
      case 0:
        if ((0x7fffffe87fffffeL & l) != 0L) {
          if (kind > 13)
            kind = 13;
```

```
    jjCheckNAdd(27);
  } else if ((0x50000000L & l) != 0L) {
    if (kind > 12)
      kind = 12;
  } else if (curChar == 126)
    jjstateSet[jjnewStateCnt++] = 22;
  break;
case 1:
  if ((0xdfffffffffffffffL & l) != 0L)
    jjAddStates(13, 14);
  break;
case 6:
  if ((0x2000000020L & l) != 0L)
    jjAddStates(15, 16);
  break;
case 17:
  if ((0xfffffffffeffffffL & l) != 0L)
    jjCheckNAddStates(7, 9);
  break;
case 18:
  if (curChar == 92)
    jjstateSet[jjnewStateCnt++] = 19;
  break;
case 19:
  if ((0x104000100000000L & l) != 0L)
    jjCheckNAddStates(7, 9);
  break;
case 21:
  if ((0x50000000L & l) != 0L
      && kind > 12)
    kind = 12;
  break;
case 23:
  if (curChar == 126)
    jjstateSet[jjnewStateCnt++] = 22;
```

```
      break;
    case 26:
    case 27:
      if ((0x7fffffe87fffffeL & l) == 0L)
        break;
      if (kind > 13)
        kind = 13;
      jjCheckNAdd(27);
      break;
    default:
      break;
    }
  } while (i != startsAt);
} else {
  int hiByte = (int) (curChar >> 8);
  int i1 = hiByte >> 6;
  long l1 = 1L << (hiByte & 077);
  int i2 = (curChar & 0xff) >> 6;
  long l2 = 1L << (curChar & 077);
  MatchLoop: do {
    switch (jjstateSet[--i]) {
    case 0:
    case 27:
      if (!jjCanMove_1(hiByte, i1,
          i2, l1, l2))
        break;
      if (kind > 13)
        kind = 13;
      jjCheckNAdd(27);
      break;
    case 1:
      if (jjCanMove_0(hiByte, i1,
          i2, l1, l2))
        jjAddStates(13, 14);
      break;
```

```
        case 17:
          if (jjCanMove_0(hiByte, i1,
              i2, l1, l2))
            jjAddStates(7, 9);
          break;
        default:
          break;
        }
      } while (i != startsAt);
    }
    if (kind != 0x7fffffff) {
      jjmatchedKind = kind;
      jjmatchedPos = curPos;
      kind = 0x7fffffff;
    }
    ++curPos;
    if ((i = jjnewStateCnt) == (startsAt = 28 - (jjnewStateCnt = startsAt)))
      return curPos;
    try {
      curChar = input_stream.readChar();
    } catch (java.io.IOException e) {
      return curPos;
    }
  }
}

static final int[] jjnextStates = { 5, 9, 10, 6,
    12, 13, 15, 17, 18, 20, 9, 10, 6, 1, 2, 7,
    8, };

private static final boolean jjCanMove_0(
    int hiByte, int i1, int i2, long l1,
    long l2) {
  switch (hiByte) {
  case 0:
```

```
    return ((jjbitVec2[i2] & l2) != 0L);
  default:
    if ((jjbitVec0[i1] & l1) != 0L)
      return true;
    return false;
  }
}

private static final boolean jjCanMove_1(
    int hiByte, int i1, int i2, long l1,
    long l2) {
  switch (hiByte) {
  case 0:
    return ((jjbitVec3[i2] & l2) != 0L);
  default:
    return false;
  }
}

public static final String[] jjstrLiteralImages = {
    "", null, null, null, null, null, null,
    null, null, null, null, null, null, null,
    null, "\44", "\56", "\133", "\135",
    "\173", "\175", };
public static final String[] lexStateNames = { "DEFAULT", };
static final long[] jjtoToken = { 0x1fb441L, };
static final long[] jjtoSkip = { 0x3eL, };
protected JavaCharStream input_stream;
private final int[] jjrounds = new int[28];
private final int[] jjstateSet = new int[56];
protected char curChar;

public ParserTelisTokenManager(
    JavaCharStream stream) {
  if (JavaCharStream.staticFlag)
```

```
    throw new Error(
        "ERROR: Cannot use a static CharStream class with a non-static lexical an
  input_stream = stream;
}

public ParserTelisTokenManager(
    JavaCharStream stream, int lexState) {
  this(stream);
  SwitchTo(lexState);
}

public void ReInit(JavaCharStream stream) {
  jjmatchedPos = jjnewStateCnt = 0;
  curLexState = defaultLexState;
  input_stream = stream;
  ReInitRounds();
}

private final void ReInitRounds() {
  int i;
  jjround = 0x80000001;
  for (i = 28; i-- > 0;)
    jjrounds[i] = 0x80000000;
}

public void ReInit(JavaCharStream stream,
    int lexState) {
  ReInit(stream);
  SwitchTo(lexState);
}

public void SwitchTo(int lexState) {
  if (lexState >= 1 || lexState < 0)
    throw new TokenMgrError(
        "Error: Ignoring invalid lexical state : "
```

```
            + lexState
            + ". State unchanged.",
        TokenMgrError.INVALID_LEXICAL_STATE);
  else
    curLexState = lexState;
}

protected Token jjFillToken() {
  Token t = Token.newToken(jjmatchedKind);
  t.kind = jjmatchedKind;
  String im = jjstrLiteralImages[jjmatchedKind];
  t.image = (im == null) ? input_stream
      .GetImage() : im;
  t.beginLine = input_stream.getBeginLine();
  t.beginColumn = input_stream.getBeginColumn();
  t.endLine = input_stream.getEndLine();
  t.endColumn = input_stream.getEndColumn();
  return t;
}

int curLexState = 0;
int defaultLexState = 0;
int jjnewStateCnt;
int jjround;
int jjmatchedPos;
int jjmatchedKind;

public Token getNextToken() {
  int kind;
  Token specialToken = null;
  Token matchedToken;
  int curPos = 0;

  EOFLoop: for (;;) {
    try {
```

```
  curChar = input_stream.BeginToken();
} catch (java.io.IOException e) {
  jjmatchedKind = 0;
  matchedToken = jjFillToken();
  return matchedToken;
}

try {
  input_stream.backup(0);
  while (curChar <= 32
      && (0x100002600L & (1L << curChar)) != 0L)
    curChar = input_stream
        .BeginToken();
} catch (java.io.IOException e1) {
  continue EOFLoop;
}
jjmatchedKind = 0x7fffffff;
jjmatchedPos = 0;
curPos = jjMoveStringLiteralDfa0_0();
if (jjmatchedKind != 0x7fffffff) {
  if (jjmatchedPos + 1 < curPos)
    input_stream.backup(curPos
        - jjmatchedPos - 1);
  if ((jjtoToken[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
    matchedToken = jjFillToken();
    return matchedToken;
  } else {
    continue EOFLoop;
  }
}
int error_line = input_stream.getEndLine();
int error_column = input_stream
    .getEndColumn();
String error_after = null;
boolean EOFSeen = false;
```

```
      try {
        input_stream.readChar();
        input_stream.backup(1);
      } catch (java.io.IOException e1) {
        EOFSeen = true;
        error_after = curPos <= 1 ? ""
            : input_stream.GetImage();
        if (curChar == '\n' || curChar == '\r') {
          error_line++;
          error_column = 0;
        } else
          error_column++;
      }
      if (!EOFSeen) {
        input_stream.backup(1);
        error_after = curPos <= 1 ? ""
            : input_stream.GetImage();
      }
      throw new TokenMgrError(EOFSeen,
          curLexState, error_line,
          error_column, error_after,
          curChar,
          TokenMgrError.LEXICAL_ERROR);
    }
  }

}
```

Classe: br.ufsc.edugraf.telis.parser.Token

```
package br.ufsc.edugraf.telis.parser;

/**
 * Describes the input token stream.
 */
```

```
public class Token {

  /**
   * An integer that describes the kind of this token. This numbering system
   * is determined by JavaCCParser, and a table of these numbers is stored in
   * the file ...Constants.java.
   */
  public int kind;

  /**
   * beginLine and beginColumn describe the position of the first character of
   * this token; endLine and endColumn describe the position of the last
   * character of this token.
   */
  public int beginLine, beginColumn, endLine,
      endColumn;

  /**
   * The string image of the token.
   */
  public String image;

  /**
   * A reference to the next regular (non-special) token from the input
   * stream. If this is the last token from the input stream, or if the token
   * manager has not read tokens beyond this one, this field is set to null.
   * This is true only if this token is also a regular token. Otherwise, see
   * below for a description of the contents of this field.
   */
  public Token next;

  /**
   * This field is used to access special tokens that occur prior to this
   * token, but after the immediately preceding regular (non-special) token.
```

```
 * If there are no such special tokens, this field is set to null. When
 * there are more than one such special token, this field refers to the last
 * of these special tokens, which in turn refers to the next previous
 * special token through its specialToken field, and so on until the first
 * special token (whose specialToken field is null). The next fields of
 * special tokens refer to other special tokens that immediately follow it
 * (without an intervening regular token). If there is no such token, this
 * field is null.
 */
public Token specialToken;

/**
 * Returns the image.
 */
public String toString() {
  return image;
}

/**
 * Returns a new Token object, by default. However, if you want, you can
 * create and return subclass objects based on the value of ofKind. Simply
 * add the cases to the switch for all those special cases. For example, if
 * you have a subclass of Token called IDToken that you want to create if
 * ofKind is ID, simlpy add something like :
 *
 * case MyParserConstants.ID : return new IDToken();
 *
 * to the following switch statement. Then you can cast matchedToken
 * variable to the appropriate type and use it in your lexical actions.
 */
public static final Token newToken(int ofKind) {
  switch (ofKind) {
  default:
    return new Token();
  }
```

```
  }

}
```

Classe: br.ufsc.edugraf.telis.parser.FachadaDoParser

```
package br.ufsc.edugraf.telis.parser;

import java.io.Reader;
import java.io.StringReader;
import java.util.regex.Pattern;

import br.ufsc.edugraf.telis.palavras.ListaDeComandos;

public class FachadaDoParser {

  private static Pattern caracteresIgnorados = Pattern
      .compile("\\s");

  public ListaDeComandos parsearInstruções(
      Reader leitor) throws ParseException {
    ParserTelis parser = new ParserTelis(leitor);
    parser.instrucoes();
    return new ListaDeComandos(parser
        .obterResultado());
  }

  public boolean símboloÉValido(String simbolo) {
    try {
      ParserTelisTokenManager telis = new ParserTelisTokenManager(
          new JavaCharStream(
              new StringReader(simbolo)));
      if (telis.getNextToken().kind != ParserTelisConstants.SIMBOLO)
        return false;
      if (telis.getNextToken().kind != ParserTelisConstants.EOF)
```

```
        return false;
    } catch (TokenMgrError e) {
      return false;
    }
    if (caracteresIgnorados.matcher(simbolo)
        .find())
      return false;
    return true;
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Palavra

```
package br.ufsc.edugraf.telis.palavras;

public interface Palavra {

  public abstract String comoTexto();

}
```

Classe: br.ufsc.edugraf.telis.palavras.ListaDeComandos

```
package br.ufsc.edugraf.telis.palavras;

import java.util.List;

public class ListaDeComandos extends
    ListaAbstrata<Palavra> {

  public ListaDeComandos(List<Palavra> lista) {
    super(lista);
  }
```

```
  public ListaDeComandos(ListaAbstrata<Palavra> lista) {
    super(lista.obterValor());
  }

  @Override
  protected char obterCaracterDeAbertura() {
    return '{';
  }

  @Override
  protected char obterCaracterDeTermino() {
    return '}';
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.ConstrutorDeLista

```
package br.ufsc.edugraf.telis.palavras;

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

import br.ufsc.edugraf.telis.traducoes.Constantes;
import br.ufsc.edugraf.telis.traducoes.Mensagens;

public class ConstrutorDeLista {
  private static final EstadoFechado ESTADO_FECHADO = new EstadoFechado();

  private interface EstadoDoConstrutorDeLista {

    public abstract Lista<Palavra> obterSaida();

    public abstract void iniciarLista();
```

```
  public abstract void terminarLista();

  public abstract void terminarListaDeComandos();

  public abstract void adicionarPalavra(
      Palavra palavra);

  public abstract ListaDeComandos obterSaidaComoListaDeComandos();

}

private class EstadoAberto implements
    EstadoDoConstrutorDeLista {

  private List<Palavra> listaDeTrabalho = new ArrayList<Palavra>();
  private Stack<List<Palavra>> listasEnvolventesDaListaDeTrabalho = new Stack<Lis

  public Lista<Palavra> obterSaida() {
    if (!listasEnvolventesDaListaDeTrabalho
        .isEmpty())
      throw new IllegalStateException(
          Mensagens
              .getString("ConstrutorDeLista.listaNaoConcluida")); //$NON-NLS-1$
    return new Lista<Palavra>(listaDeTrabalho);
  }

  public ListaDeComandos obterSaidaComoListaDeComandos() {
    if (!listasEnvolventesDaListaDeTrabalho
        .isEmpty())
      throw new IllegalStateException(
          Mensagens
              .getString("ConstrutorDeLista.listaNaoConcluida")); //$NON-NLS-1$
    return new ListaDeComandos(listaDeTrabalho);
  }
```

```
public void iniciarLista() {
  listasEnvolventesDaListaDeTrabalho
      .push(listaDeTrabalho);
  listaDeTrabalho = new ArrayList<Palavra>();
}

public void terminarLista() {
  if (listasEnvolventesDaListaDeTrabalho
      .isEmpty())
    throw new IllegalStateException(
        Mensagens
            .getString("ConstrutorDeLista.listaConcluida")); //$NON-NLS-1$
  List<Palavra> lista = listasEnvolventesDaListaDeTrabalho
      .pop();
  lista.add(new Lista<Palavra>(
      listaDeTrabalho));
  listaDeTrabalho = lista;
}

public void terminarListaDeComandos() {
  if (listasEnvolventesDaListaDeTrabalho
      .isEmpty())
    throw new IllegalStateException(
        Mensagens
            .getString("ConstrutorDeLista.listaConcluida")); //$NON-NLS-1$
  List<Palavra> lista = listasEnvolventesDaListaDeTrabalho
      .pop();
  lista.add(new ListaDeComandos(
      listaDeTrabalho));
  listaDeTrabalho = lista;
}

public void adicionarPalavra(Palavra palavra) {
  listaDeTrabalho.add(palavra);
```

```
  }

}

private static class EstadoFechado implements
    EstadoDoConstrutorDeLista {
  private static final String MENSAGEM_ESTADO_FECHADO = Constantes
      .obterString("ConstrutorDeLista.fechado"); //$NON-NLS-1$

  public void iniciarLista() {
    throw new IllegalStateException(
        MENSAGEM_ESTADO_FECHADO);
  }

  public Lista<Palavra> obterSaida() {
    throw new IllegalStateException(
        MENSAGEM_ESTADO_FECHADO);
  }

  public void terminarLista() {
    throw new IllegalStateException(
        MENSAGEM_ESTADO_FECHADO);
  }

  public void terminarListaDeComandos() {
    throw new IllegalStateException(
        MENSAGEM_ESTADO_FECHADO);
  }

  public ListaDeComandos obterSaidaComoListaDeComandos() {
    throw new IllegalStateException(
        MENSAGEM_ESTADO_FECHADO);
  }

  public void adicionarPalavra(Palavra palavra) {
```

```
    throw new IllegalStateException(
        MENSAGEM_ESTADO_FECHADO);
  }
}

private EstadoDoConstrutorDeLista estado = new EstadoAberto();

public Lista<Palavra> obterSaidaComoLista() {
  Lista<Palavra> obterSaida = estado
      .obterSaida();
  estado = ESTADO_FECHADO;
  return obterSaida;
}

public ListaDeComandos obterSaidaComoListaDeComandos() {
  ListaDeComandos saida = estado
      .obterSaidaComoListaDeComandos();
  estado = ESTADO_FECHADO;
  return saida;
}

public ConstrutorDeLista adicionarTexto(
    String texto) {
  estado.adicionarPalavra(new Texto(texto));
  return this;
}

public ConstrutorDeLista adicionarNúmero(
    double numero) {
  estado.adicionarPalavra(new Numero(numero));
  return this;
}

public ConstrutorDeLista adicionarSímbolo(
    String nome) {
```

```
  estado.adicionarPalavra(new Simbolo(nome));
  return this;
}

public ConstrutorDeLista adicionarBooleano(
    boolean valor) {
  estado.adicionarPalavra(new Booleano(valor));
  return this;
}

public ConstrutorDeLista iniciarLista() {
  estado.iniciarLista();
  return this;
}

public ConstrutorDeLista terminarListaDeComando() {
  estado.terminarListaDeComandos();
  return this;
}

public ConstrutorDeLista terminarLista() {
  estado.terminarLista();
  return this;
}

public ConstrutorDeLista adicionarOperador(
    Operador operador) {
  estado.adicionarPalavra(operador);
  return this;
}

public ConstrutorDeLista adicionarOperadorDeEmpilhamento(
    String nomeDoSimbolo) {
  estado
      .adicionarPalavra(new OperadorDeEmpilhamento(
```

```
            new Simbolo(nomeDoSimbolo)));
    return this;
  }

  public ConstrutorDeLista adicionarPalavra(
      Palavra palavra) {
    estado.adicionarPalavra(palavra);
    return this;
  }

  public ConstrutorDeLista adicionarOperadorDeComunicacaoDireta(
      String nomeDaAgenda) {
    estado
        .adicionarPalavra(new OperadorDeComunicacaoDireta(
            new Simbolo(nomeDaAgenda)));
    return this;
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Operador

```
package br.ufsc.edugraf.telis.palavras;

import br.ufsc.edugraf.telis.Resolvivel;

public enum Operador implements Palavra, Resolvivel {

  SOMA("+"), //$NON-NLS-1$
  SUBTRACAO("-"), //$NON-NLS-1$
  MULTIPLICACAO("*"), //$NON-NLS-1$
  DIVISAO("/"), //$NON-NLS-1$
  MODULO("\\"), //$NON-NLS-1$
  EXPONENCIACAO("^"), //$NON-NLS-1$
  IGUAL("="), //$NON-NLS-1$
  DIFERENTE("~="), //$NON-NLS-1$
```

```
MENOR("<"), //$NON-NLS-1$
MAIOR(">"), //$NON-NLS-1$
MENOR_OU_IGUAL("<="), //$NON-NLS-1$
MAIOR_OU_IGUAL(">="); //$NON-NLS-1$

private final String caracter;

private Operador(String caracter) {
  this.caracter = caracter;
}

public String comoTexto() {
  return caracter;
}

@Override
public String toString() {
  return comoTexto();
}

public static Operador obterOperador(String código) {
  assert código != null : "código nunca deve ser nulo"; //$NON-NLS-1$
  if ("+".equals(código)) //$NON-NLS-1$
    return SOMA;
  else if ("-".equals(código)) //$NON-NLS-1$
    return SUBTRACAO;
  else if ("*".equals(código)) //$NON-NLS-1$
    return MULTIPLICACAO;
  else if ("/".equals(código)) //$NON-NLS-1$
    return DIVISAO;
  else if ("\\".equals(código)) //$NON-NLS-1$
    return MODULO;
  else if ("=".equals(código)) //$NON-NLS-1$
    return IGUAL;
  else if ("~=".equals(código)) //$NON-NLS-1$
```

```
      return DIFERENTE;
    else if ("<".equals(código)) //$NON-NLS-1$
      return MENOR;
    else if ("<=".equals(código)) //$NON-NLS-1$
      return MENOR_OU_IGUAL;
    else if (">".equals(código)) //$NON-NLS-1$
      return MAIOR;
    else if (">=".equals(código)) //$NON-NLS-1$
      return MAIOR_OU_IGUAL;
    else if ("^".equals(código)) //$NON-NLS-1$
      return EXPONENCIACAO;
    throw new AssertionError(
        "\""  + código + "\" não pode ser transformado em um operador."); //$NON-NL
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Simbolo

```
package br.ufsc.edugraf.telis.palavras;

import java.security.InvalidParameterException;

import br.ufsc.edugraf.telis.Resolvivel;
import br.ufsc.edugraf.telis.parser.FachadaDoParser;

public class Simbolo extends PalavraTextualAbstrata
    implements Resolvivel {

  private static final FachadaDoParser fachadaDoParser = new FachadaDoParser();

  public Simbolo(String nome) {
    super(nome);
    if (!fachadaDoParser.símboloÉValido(nome))
      throw new InvalidParameterException(
```

```
            "Não é possível utilizar: \""
                + nome
                + "\" como um símbolo.");
    }

    @Override
    public String toString() {
      return obterValor();
    }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Texto

```
package br.ufsc.edugraf.telis.palavras;

public class Texto extends PalavraTextualAbstrata {

    public Texto(String texto) {
      super(texto);
    }

    public Texto concatenar(Texto texto) {
      return new Texto(obterValor()
          + texto.obterValor());
    }

    public Texto concatenar(String texto) {
      return new Texto(obterValor() + texto);
    }

    public Texto prefixar(String prefixo) {
      return new Texto(prefixo + obterValor());
    }
```

```
  public Texto prefixar(Palavra prefixo) {
    return prefixar(prefixo);
  }

  @Override
  public String toString() {
    return "\"" + obterValor() + "\""; //$NON-NLS-1$ //$NON-NLS-2$
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.ListaAbstrata

```
package br.ufsc.edugraf.telis.palavras;

import java.util.Collections;
import java.util.List;

public abstract class ListaAbstrata<T extends Palavra>
    implements Palavra {

  protected final List<T> lista;

  public ListaAbstrata(List<T> lista) {
    this.lista = lista;
  }

  public List<T> obterValor() {
    return Collections.unmodifiableList(lista);
  }

  @Override
  public int hashCode() {
    final int PRIME = 31;
    int result = 1;
```

```
  result = PRIME
      * result
      + ((lista == null) ? 0 : lista
          .hashCode());
  return result;
}

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  final ListaAbstrata<?> other = (ListaAbstrata<?>) obj;
  if (lista == null) {
    if (other.lista != null)
      return false;
  } else if (!lista.equals(other.lista))
    return false;
  return true;
}

public String comoTexto() {
  StringBuilder valor = new StringBuilder();
  valor.append(obterCaracterDeAbertura());
  for (T elemento : lista) {
    valor.append(' ');
    valor.append(elemento.comoTexto());
  }
  valor.append(" " + obterCaracterDeTermino()); //$NON-NLS-1$
  return valor.toString();
}
```

```
  @Override
  public String toString() {
    StringBuilder valor = new StringBuilder();
    valor.append(obterCaracterDeAbertura());
    for (T elemento : lista) {
      valor.append(' ');
      valor.append(elemento.toString());
    }
    valor.append(" " + obterCaracterDeTermino()); //$NON-NLS-1$
    return valor.toString();
  }

  public int obterTamanho() {
    return lista.size();
  }

  public boolean estáVazia() {
    return obterTamanho() == 0;
  }

  protected abstract char obterCaracterDeAbertura();

  protected abstract char obterCaracterDeTermino();

  public T obterElemento(int posição) {
    return lista.get(posição);
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Numero

```
package br.ufsc.edugraf.telis.palavras;

import java.text.NumberFormat;
import static java.lang.Math.*;
```

```
public class Numero implements Palavra {

  private final double numero;

  public Numero(double numero) {
    this.numero = numero;
  }

  public double obterValor() {
    return numero;
  }

  @Override
  public int hashCode() {
    final int PRIME = 31;
    int result = 1;
    long temp;
    temp = Double.doubleToLongBits(numero);
    result = PRIME * result
        + (int) (temp ^ (temp >>> 32));
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (!(obj instanceof Numero))
      return false;
    final Numero other = (Numero) obj;
    if (Double.doubleToLongBits(numero) != Double
        .doubleToLongBits(other.numero))
```

```
    return false;
  return true;
}

@Override
public String toString() {
  NumberFormat formatador = NumberFormat
      .getInstance();
  return formatador.format(numero);
}

public String comoTexto() {
  return toString();
}

public Numero somar(Numero numero) {
  return new Numero(numero.obterValor()
      + obterValor());
}

public Numero multiplicar(Numero numero) {
  return new Numero(numero.obterValor()
      * obterValor());
}

public Numero dividir(Numero divisor) {
  return new Numero(obterValor()
      / divisor.obterValor());
}

public Numero subtrair(Numero numero) {
  return new Numero(obterValor()
      - numero.obterValor());
}
```

```
  public Numero modulo(Numero divisor) {
    return new Numero(obterValor()
        % divisor.obterValor());
  }

  public Numero elevarA(Numero expoente) {
    return new Numero(pow(obterValor(), expoente
        .obterValor()));
  }

  public Booleano menorQue(Numero numero) {
    return new Booleano(obterValor() < numero
        .obterValor());
  }

  public Palavra menorOuIgualA(Numero numero) {
    return new Booleano(obterValor() <= numero
        .obterValor());
  }

  public int obterParteInteira() {
    return (int) obterValor();
  }

  public Booleano maiorQue(Numero numero) {
    return new Booleano(obterValor() > numero
        .obterValor());
  }

  public Palavra maiorOuIgualA(Numero numero) {
    return new Booleano(obterValor() >= numero
        .obterValor());
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Booleano

```
package br.ufsc.edugraf.telis.palavras;

import br.ufsc.edugraf.telis.traducoes.Constantes;

public class Booleano implements Palavra {

  private final boolean valor;

  public Booleano(boolean valor) {
    this.valor = valor;
  }

  public Booleano(Numero numero) {
    this.valor = numero.obterValor() != 0;
  }

  public String comoTexto() {
    return toString();
  }

  public boolean obterValor() {
    return valor;
  }

  @Override
  public String toString() {
    return valor ? Constantes
        .obterString("PalavraReservada.verdadeiro") : Constantes.obterString("Palav
  }

  @Override
  public int hashCode() {
```

```
    final int PRIME = 31;
    int result = 1;
    result = PRIME * result
        + (valor ? 1231 : 1237);
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    final Booleano other = (Booleano) obj;
    if (valor != other.valor)
      return false;
    return true;
  }

  public Booleano ou(Booleano booleano) {
    return new Booleano(valor
        || booleano.obterValor());
  }

  public Booleano e(Booleano booleano) {
    return new Booleano(valor
        && booleano.obterValor());
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Filtro

```
package br.ufsc.edugraf.telis.palavras;

import java.util.Iterator;
import java.util.List;

import br.ufsc.edugraf.telis.traducoes.Constantes;

public class Filtro<T extends Palavra> extends
    ListaAbstrata<T> {
  public Filtro(List<T> lista) {
    super(lista);
  }

  public Filtro(ListaAbstrata<T> lista) {
    super(lista.obterValor());
  }

  public boolean casar(Lista<? extends Palavra> lista) {
    if (lista.obterTamanho() != this
        .obterTamanho())
      return false;

    Iterator<? extends Palavra> iteradorDaTupla = lista
        .obterValor().iterator();
    for (Palavra valor : obterValor()) {
      Palavra valorDaTupla = iteradorDaTupla
          .next();
      if (valor instanceof Simbolo) {
        String valorComoTexto = ((Simbolo) valor)
            .obterValor();
        if ((valorComoTexto.equals(Constantes
            .obterString("Filtros.texto")) && valorDaTupla instanceof Texto) || //$
            (valorComoTexto
                .equals(Constantes
                    .obterString("Filtros.numero")) && valorDaTupla instanceof Nume
```

```
            (valorComoTexto
                .equals(Constantes
                    .obterString("Filtros.lista")) && valorDaTupla instanceof Lista
          return true;
      }
      if (!valor.equals(valorDaTupla))
        return false;
    }
    return true;
  }

  @Override
  protected char obterCaracterDeAbertura() {
    return '[';
  }

  @Override
  protected char obterCaracterDeTermino() {
    return ']';
  }
}
```

Classe: br.ufsc.edugraf.telis.palavras.OperadorDeComunicacaoDireta

```
package br.ufsc.edugraf.telis.palavras;

import br.ufsc.edugraf.telis.Resolvivel;

// TODO: extrair superClasse com OperadorDeEmpilhamento.
public class OperadorDeComunicacaoDireta implements
    Palavra, Resolvivel {

  private final Simbolo simboloDaAgenda;

  public OperadorDeComunicacaoDireta(
```

```
    Simbolo nomeDaAgenda) {
  this.simboloDaAgenda = nomeDaAgenda;
}

public String comoTexto() {
  return toString();
}

public Simbolo obterSimbolo() {
  return simboloDaAgenda;
}

@Override
public String toString() {
  return "." + simboloDaAgenda.toString(); //$NON-NLS-1$
}

@Override
public int hashCode() {
  final int PRIME = 31;
  int result = 1;
  result = PRIME
      * result
      + ((simboloDaAgenda == null) ? 0
          : simboloDaAgenda.hashCode());
  return result;
}

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
```

```
      return false;
    final OperadorDeComunicacaoDireta other = (OperadorDeComunicacaoDireta) obj;
    if (simboloDaAgenda == null) {
      if (other.simboloDaAgenda != null)
        return false;
    } else if (!simboloDaAgenda
        .equals(other.simboloDaAgenda))
      return false;
    return true;
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.Lista

```
package br.ufsc.edugraf.telis.palavras;

import java.util.ArrayList;
import java.util.List;

public class Lista<T extends Palavra> extends
    ListaAbstrata<T> {

  private static final int PRIMEIRO = 0;

  public Lista(List<T> lista) {
    super(lista);
  }

  @Override
  protected char obterCaracterDeAbertura() {
    return '[';
  }

  @Override
```

```
protected char obterCaracterDeTermino() {
  return ']';
}

public Palavra concatenar(
    Lista<Palavra> primeiraLista) {
  ArrayList<Palavra> lista = new ArrayList<Palavra>();
  lista.addAll(obterValor());
  lista.addAll(primeiraLista.obterValor());
  return new Lista<Palavra>(lista);
}

public Lista<T> inserirElemento(int posição,
    T elemento) {
  List<T> listaNova = new ArrayList<T>(
      obterValor());
  listaNova.add(posição, elemento);
  return new Lista<T>(listaNova);
}

public Lista<T> inserirNoInicio(T elemento) {
  return inserirElemento(PRIMEIRO, elemento);
}

public Lista<T> inserirNoFim(T elemento) {
  return inserirElemento(obterTamanho(),
      elemento);
}

public T obterPrimeiro() {
  return obterElemento(PRIMEIRO);
}

public T obterÚltimo() {
  return obterElemento(obterTamanho() - 1);
```

```
  }

  public Lista<T> removerElemento(int posição) {
    ArrayList<T> listaRetorno = new ArrayList<T>();
    List<T> lista = obterValor();
    listaRetorno.addAll(lista);
    listaRetorno.remove(posição);
    return new Lista<T>(listaRetorno);
  }

  public Lista<T> removerPrimeiro() {
    return removerElemento(0);
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.OperadorDeEmpilhamento

```
package br.ufsc.edugraf.telis.palavras;

import br.ufsc.edugraf.telis.Resolvivel;

public class OperadorDeEmpilhamento implements
    Palavra, Resolvivel {

  private final Simbolo simboloAEmpilhar;

  public OperadorDeEmpilhamento(
      Simbolo simboloAEmpilhar) {
    this.simboloAEmpilhar = simboloAEmpilhar;
  }

  public String comoTexto() {
    return toString();
  }
```

```
public Simbolo obterSimbolo() {
  return simboloAEmpilhar;
}

@Override
public String toString() {
  return "$" + simboloAEmpilhar.toString(); //$NON-NLS-1$
}

@Override
public int hashCode() {
  final int PRIME = 31;
  int result = 1;
  result = PRIME
      * result
      + ((simboloAEmpilhar == null) ? 0
          : simboloAEmpilhar.hashCode());
  return result;
}

@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  final OperadorDeEmpilhamento other = (OperadorDeEmpilhamento) obj;
  if (simboloAEmpilhar == null) {
    if (other.simboloAEmpilhar != null)
      return false;
  } else if (!simboloAEmpilhar
      .equals(other.simboloAEmpilhar))
```

```
      return false;
    return true;
  }

}
```

Classe: br.ufsc.edugraf.telis.palavras.PalavraTextualAbstrata

```
package br.ufsc.edugraf.telis.palavras;

public class PalavraTextualAbstrata implements Palavra {

  protected final String texto;

  public PalavraTextualAbstrata(String texto) {
    this.texto = texto;
  }

  public String obterValor() {
    return texto;
  }

  @Override
  public int hashCode() {
    final int PRIME = 31;
    int result = 1;
    result = PRIME
        * result
        + ((texto == null) ? 0 : texto
            .hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
```

```
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    final PalavraTextualAbstrata other = (PalavraTextualAbstrata) obj;
    if (texto == null) {
      if (other.texto != null)
        return false;
    } else if (!texto.equals(other.texto))
      return false;
    return true;
  }

  public String comoTexto() {
    return obterValor();
  }

}
```

Classe: br.ufsc.edugraf.telis.Resolvivel

```
package br.ufsc.edugraf.telis;

public interface Resolvivel {

}
```