

MANUAL DE INTEGRAÇÃO DO PORTUGOL

LUIZ FERNANDO NOSCHANG

Objetivo

Este documento é um manual de instruções que tem como objetivo guiar os desenvolvedores de software no processo de integração do Portugol e seus componentes com outras aplicações. O manual está organizado de forma sequencial em capítulos que descrevem cada fase necessária para realizar esta integração.

Todos os passos abordados neste manual são realizados exclusivamente no sistema operacional Windows 7, por ser um ambiente familiar à maioria dos usuários e por ser a versão mais atual do sistema operacional. No entanto, com algumas adaptações deverá ser possível realizar estes passos em outras versões do Windows e em outros sistemas como Linux e MacOS. Se possível, as versões futuras do manual incluirão também as instruções para estes sistemas.

Espera-se que os usuários deste manual possuam o domínio das linguagens de programação, bem como de seus ambientes (IDEs) e ferramentas, e que já as tenham instaladas no computador. Por isso, tais tecnologias não são detalhadas no manual e deverão ser pesquisadas em fontes externas caso haja dúvidas na sua utilização ou instalação.

Sumário

[1. Obtendo o código fonte 4](#_Toc327974963)

[2. Integrando com Java 7](#_Toc327974964)

[2.1 Compilando e incluindo os módulos necessários 7](#_Toc327974965)

[2.2 Utilizando os serviços do núcleo 8](#_Toc327974966)

[2.2.1 Análise e tratamento de erros 8](#_Toc327974967)

[2.2.2 Geração de código intermediário 10](#_Toc327974968)

[2.2.3 Execução de programas 13](#_Toc327974969)

[3. Integrando com C# 20](#_Toc327974970)

[3.1 Compilando os módulos necessários 20](#_Toc327974971)

[3.1.1 Módulo Java 20](#_Toc327974972)

[3.1.2 Módulo C# 20](#_Toc327974973)

[3.2 Incluindo e configurando os módulos necessários 21](#_Toc327974974)

[3.3 Gerenciando o serviço de integração 23](#_Toc327974975)

[3.4 Utilizando os serviços do núcleo 25](#_Toc327974976)

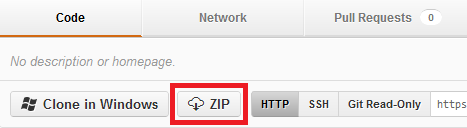
[3.5 Limitações 27](#_Toc327974977)

1. Obtendo o código fonte

O primeiro passo para realizar a integração é obter o código fonte do Portugol e seus componentes, bem como dos módulos de integração. O código fonte do Portugol está disponível em um repositório de projetos Open Source (GitHub) sob o sistema de versionamento git. A página do repositório é: <https://github.com/UNIVALI-L2S/Portugol>.

Existem duas formas de se obter o código fonte. A primeira forma (e a preferencial) é utilizar uma ferramenta de versionamento git para “clonar” o repositório do projeto. Utilizando esta abordagem será mais fácil obter as versões atualizadas do Portugol futuramente e permitirá que alterações feitas no código fonte possam ser submetidas aos desenvolvedores para contribuir com o projeto. A segunda forma é baixar o código fonte em um arquivo no formato zip. Isto pode ser feito clicando no link “ZIP” da página do repositório, como mostra o destaque em vermelho na Figura 1.

Figura 1. Link para baixar o código fonte no formato zip



Neste manual será utilizada a abordagem de clonagem do repositório. Para que seja possível clonar o repositório, é necessário possuir uma conta no GitHub e ter instalado no computador uma ferramenta de versionamento git.

A criação de uma conta grátis é bem rápida e simples e pode ser realizada no endereço: <https://github.com/signup/free>. Para a criação da conta, basta preencher o formulário e clicar em “Create an account”. Após a criação da conta o GitHub irá guiá-lo automaticamente na instalação da ferramenta git para o seu sistema operacional. Também é possível acessar as instruções diretamente no endereço: <https://help.github.com/articles/set-up-git>.

Assim que a instalação do git estiver concluída, siga as instruções disponíveis no endereço <https://help.github.com/articles/fork-a-repo> para clonar o repositório, o processo de clonagem leva algum tempo. Você deverá substituir a url do repositório que aparece nos exemplos com a url do repositório do Portugol: [git@github.com:UNIVALI-L2S/Portugol.git](mailto:git@github.com:UNIVALI-L2S/Portugol.git). Quando o processo estiver concluído, entre na pasta do Portugol utilizando o git e digite os seguintes comandos:

$ cd Portugol

# Entra no diretório raiz do projeto

$ git submodule init

# Inicializa os módulos do Portugol

$ git submodule update

# Baixa o código fonte atualizado dos módulos

Estes comandos instruem o git a inicializar e baixar o código fonte atualizado de cada projeto dentro do repositório. Novamente levará algum tempo até que o git baixe e atualize cada projeto. Quando este processo estiver concluído, digite o comando “git status”. Se tudo ocorreu corretamente, o resultado deve ser similar ao do quadro a seguir.

$ git status

# On branch master

Nothing to commit (working directory clean)

O repositório do Portugol está organizado em quatro pastas:

* **Documentacao:** contém toda a documentação gerada para o Portugol (incluindo este manual) e, os documentos consultados durante o desenvolvimento, como artigos e tutoriais das ferramentas utilizadas.
* **Exemplos:** contém diversos algoritmos escritos em Portugol 2.0, exemplificando todos os comandos e construções da linguagem.
* **Ferramentas:** contém o instalador/executável de algumas das ferramentas utilizadas no desenvolvimento do Portugol, como o ANTLR, por exemplo.
* **Projetos:** contém o código fonte de cada um dos módulos que compõem o Portugol, conforme descreve a Tabela 1.

Tabela 1. Módulos do Portugol

| **Módulo** | **Descrição** | **Pasta** |
| --- | --- | --- |
| Núcleo | Este é o módulo principal do Portugol. Ele é responsável por realizar o parsing, análise sintática e semântica, geração de código intermediário (ASA) e execução dos algoritmos em Portugol 2.0. | Portugol-Nucleo |
| Relator de Erros | Este módulo trabalha em conjunto com o núcleo, capturando exceções que ainda não foram devidamente tratadas e gerando relatórios detalhados destas exceções para posterior análise e correção. | Portugol-Relator-Erros |
| WebService | Este módulo é uma aplicação Web em Java que recebe via POST os relatórios de exceção gerados pelo relator de erros. Os relatórios são armazenados em um banco de dados MySQL. | Portugol-WebService |
| PortugolStudio | Este módulo é um ambiente de desenvolvimento (IDE) escrito em Java para permitir a criação e execução de algoritmos em Portugol 2.0. Ele possui os recursos básicos de toda IDE e utiliza os serviços disponibilizados pelo núcleo. | Portugol-Studio |
| Corretor automático | Este módulo é um corretor desenvolvido em Java que permite corrigir automaticamente um conjunto pré-definido de exercícios. O módulo utiliza arquivos XML com modelos de solução e pares de entrada e saída pré-definidos. | Portugol-Corretor |
| Módulo de Integração Java | Este módulo inicializa o serviço de integração do Portugol e faz a conversão dos objetos Java para objetos CORBA e vice versa. Este módulo é necessário para realizar a integração do núcleo com outras linguagens de programação. | Portugol-Integracao |
| Módulo de Integração C# | Este módulo inicializa e comunica-se com o módulo de integração Java e faz a conversão dos objetos C# para CORBA e vice versa. | Portugol-Integracao-CSharp |

1. Integrando com Java

Os passos descritos neste manual são específicos para o ambiente de programação NetBeans, no qual o Portugol foi desenvolvido. Outros ambientes poderão ser utilizados, mas neste caso os passos deverão ser adaptados para o novo ambiente. Caso você opte por utilizar outra ferramenta, consulte a documentação da mesma para saber como realizar os passos descritos no manual.

* 1. Compilando e incluindo os módulos necessários

O Portugol foi desenvolvido na linguagem Java e, portanto, oferece integração nativa com outras aplicações em Java. Para realizar a integração, basta incluir os arquivos jar do núcleo e suas bibliotecas na aplicação Java. Não há necessidade de incluir os módulos de integração listados no capítulo anterior, pois estes módulos são necessários apenas para a integração com aplicações em outras linguagens de programação.

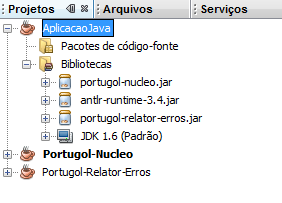
Para começar, abra o projeto “Portugol-Nucleo” no NetBeans, certificando-se de marcar as opções “Abrir como projeto principal” e “Abrir projetos requeridos”. Depois, clique com o botão direito do mouse sobre o projeto e selecione a opção “Limpar e construir”.

Ao selecionar esta opção, o NetBeans irá compilar o código fonte do núcleo e de todos os projetos dependentes. No final da compilação, será gerada uma pasta com o nome “dist” dentro da pasta do projeto, contendo os arquivos jar do núcleo e das bibliotecas/módulos utilizados por ele.

Para incluir os arquivos na aplicação Java (supondo que a mesma tenha sido criada no NetBeans), clique com o botão direito do mouse sobre o projeto e selecione a opção “Propriedades”. Na tela a seguir, selecione a opção “Bibliotecas”, localizada no painel “Categorias” à esquerda. Logo após, clique no botão “Adicionar JAR/pasta”, localizado no painel à direita e navegue até a pasta “dist”.

Dentro da pasta, selecione todos os arquivos jar e clique no botão “Abrir”, certificando-se de marcar a opção “Copiar para pasta de bibliotecas”. Você pode selecionar todos os arquivos de uma só vez segurando a tecla “Shift” ao clicar sobre cada arquivo. Se tudo der certo, você verá os arquivos listados no item “Bibliotecas” do projeto, conforme mostra a Figura 2.

Figura 2. Lista de bibliotecas



Feito isto, as classes e serviços do Portugol já estarão disponíveis para uso na aplicação. Todas as classes do Portugol estão no pacote “br.univali.portugol.nucleo”. O Capítulo 2.2 descreve os serviços do núcleo de forma geral, dando exemplos de sua utilização. Para obter uma descrição mais detalhada de cada uma das classes do Portugol, consulte o Javadoc na pasta “Documentacao” do repositório.

* 1. Utilizando os serviços do núcleo

O núcleo do Portugol disponibiliza três serviços para as aplicações que o utilizam: análise e tratamento de erros, geração de código intermediário e execução de programas. Estes serviços estão acessíveis a partir de um conjunto de classes de fachada (facade) que abstraem os detalhes da implementação. O objetivo deste capítulo é descrever cada um destes serviços e demonstrar como utilizá-los dentro de uma aplicação.

1. * 1. Análise e tratamento de erros

O núcleo do Portugol provê um mecanismo de análise e tratamento de erros para garantir a corretude dos algoritmos escritos em Portugol. Esta análise está internamente dividida em duas partes distintas: análise sintática e análise semântica e, ocorre nesta ordem.

A análise sintática do Portugol é responsável por identificar os erros sintáticos do programa, ou seja, os erros relacionados à má formação das construções gramaticais, como por exemplo, o não fechamento de um parêntese em uma expressão.

A análise semântica por sua vez, identifica os erros semânticos, ou seja, as construções gramaticais que não possuem um significado válido dentro do contexto da linguagem, como por exemplo, a soma entre tipos de dados incompatíveis. Além dos erros semânticos, são detectadas conversões entre tipos e arredondamentos, os quais são tratados pelo Portugol não como erros, mas como avisos. Por último, a análise semântica do Portugol só pode ser realizada em programas que estejam sintaticamente corretos, criando uma dependência com a análise sintática.

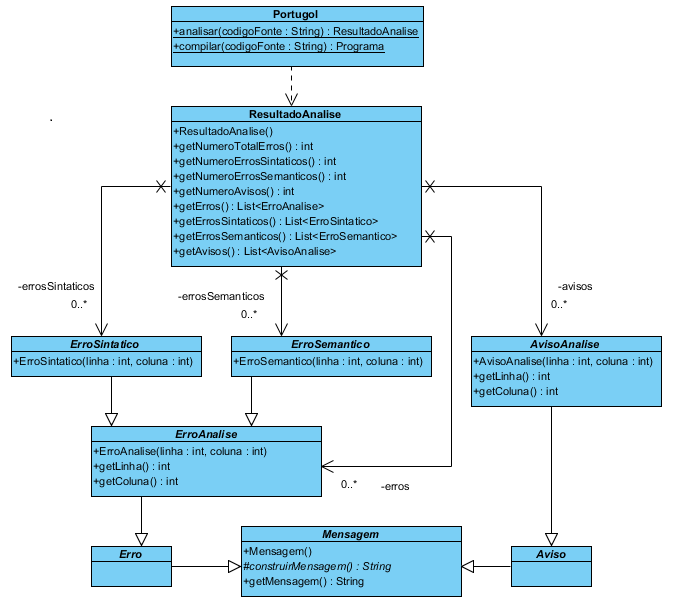
Para utilizar a análise de erros do Portugol, deve-se chamar o método estático “analisar” da classe “Portugol”, passando por parâmetro o código fonte a ser analisado. No final da análise, um objeto da classe “ResultadoAnalise” é retornado, o qual contém os erros e avisos encontrados durante a análise. A partir daí, os erros podem ser percorridos tomando as ações adequadas, como por exemplo, exibi-los ao usuário. O Quadro 1 apresenta um trecho de código que demonstra a utilização da análise de erros do Portugol.

Quadro 1. Exemplo de utilização do serviço de análise e tratamento de erros

|  |
| --- |
| **public final class** IDE  {  **public static void** ***main***(String[] args)  {  File arquivo = **new** File(**"C:\\exemplo\\algoritmo.por"**);  String algoritmo = CodigoFonte.*carregar*(arquivo);  ResultadoAnalise resultadoAnalise = Portugol.*analisar*(algoritmo);    **if** (resultadoAnalise.getNumeroTotalErros() > 0)  {  **int** numeroErros = resultadoAnalise.getNumeroTotalErros();  String mensagem = **"Seu algoritmo contém %d erros:"**;  System.out.println(String.format(mensagem, numeroErros));  **for** (ErroAnalise erro : resultadoAnalise.getErros())  {  System.out.println(erro.getMensagem());  }  }  }  } |

Internamente, várias classes estão envolvidas no serviço de análise e tratamento de erros do Portugol. No entanto, para facilitar a utilização do serviço, o núcleo disponibiliza um conjunto reduzido de classes de fachada (facade) que abstraem os detalhes da implementação. Para uma melhor compreensão, o diagrama da Figura 3 mostra este conjunto de classes e seus relacionamentos.

Figura 3. Diagrama de classes do serviço de análise e tratamento de erros



* + 1. Geração de código intermediário

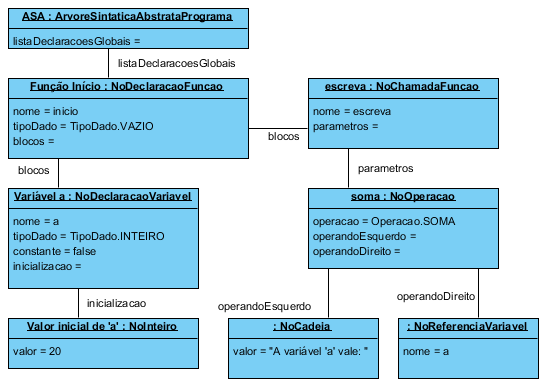
O núcleo do Portugol provê um mecanismo para a geração de código intermediário a partir do código fonte de qualquer algoritmo em Portugol 2.0. Este código intermediário é representado por uma estrutura em árvore chamada ASA (Árvore Sintática Abstrata) onde cada instrução do código-fonte corresponde a um nó da árvore e é representada por um objeto.

A principal vantagem da ASA é a abstração que ela proporciona em relação à gramática (sintaxe) do Portugol 2.0. Não importa, por exemplo, se na gramática uma multiplicação é representada pelo operador “\*” ou “x”, pois uma operação de multiplicação sempre será representada por um objeto da classe “NoOperacao”. Para uma melhor compreensão, o diagrama da Figura 4 ilustra a ASA gerada a partir do algoritmo do Quadro 2.

Quadro 2. Exemplo de algoritmo do Portugol 2.0

|  |
| --- |
| **programa**  {  **funcao** **inicio**()  {  **inteiro** a = **20**  **escreva**(**"A variável 'a' vale: "** + a)  }  } |

Figura 4. Exemplo de código intermediário do Portugol



O núcleo do Portugol utiliza a ASA para a execução dos programas. No entanto, ela pode ser manipulada para obter outros resultados, como por exemplo: otimização de código, análise de fluxo (ex.: detectar chamadas recursivas infinitas), detecção de código morto (Ex.: atribuir uma variável a si mesma) entre outros. Para facilitar a criação de novas funcionalidades utilizando a ASA, o Portugol dá suporte ao padrão de projeto Visitor através da interface “VisitanteASA”.

Para utilizar a geração de código intermediário do Portugol, a aplicação deve chamar o método estático “compilar” da classe “Portugol” passando o código fonte a ser compilado. No final da compilação, um objeto da classe “Programa” é retornado e o código intermediário pode então ser obtido através do método “getArvoreSintaticaAbstrata”.

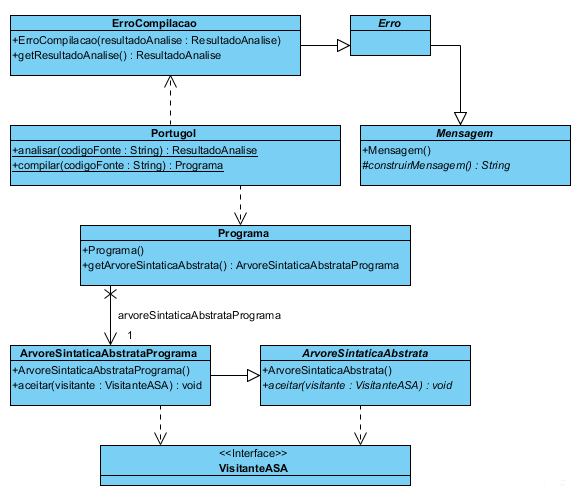
A geração de código intermediário só pode ser realizada em algoritmos que não possuem erros sintáticos ou semânticos. Desta forma, ao chamar o método “compilar” o núcleo do Portugol automaticamente realiza uma análise de erros. Caso o algoritmo contenha erros, será lançada uma exceção do tipo “ErroCompilacao”. A partir desta exceção é possível obter a lista dos erros ocorridos através do método “getResultadoAnalise”. O Quadro 3 apresenta um trecho de código que demonstra a utilização do serviço de geração de código intermediário.

Quadro 3. Exemplo de utilização do serviço de geração de código intermediário

|  |
| --- |
| **public final class** IDE  {  **public static void** ***main***(String[] args)  {  File arquivo = **new** File(**"C:\\exemplo\\algoritmo.por"**);  String algoritmo = CodigoFonte.*carregar*(arquivo);    Programa programa = **null**;  ArvoreSintaticaAbstrata asa = **null**;  List<NoDeclaracao> declaracoes = **null**;    **try**  {  programa = Portugol.*compilar*(algoritmo);  asa = programa.getArvoreSintaticaAbstrata();  declaracoes = asa.getListaDeclaracoesGlobais();    **for** (NoDeclaracao declaracao : declaracoes)  {  **if** (declaracao **instanceof** NoDeclaracaoVariavel)  {  String nome = declaracao.getNome();  String mensagem = **"Variável global encontrada: %s"**;  System.out.println(String.format(mensagem, nome));  }  }  }  **catch** (ErroCompilacao erro)  {  exibirErros(erro.getResultadoAnalise());  }  }  } |

Internamente, várias classes estão envolvidas no processo de geração de código intermediário do Portugol. Mas assim como na análise de erros, esta implementação está abstraída através de um conjunto reduzido de classes de fachada. Para uma melhor compreensão, o diagrama da Figura 5 mostra este conjunto de classes e seus relacionamentos.

Figura 5. Diagrama de classes do serviço de geração de código intermediário

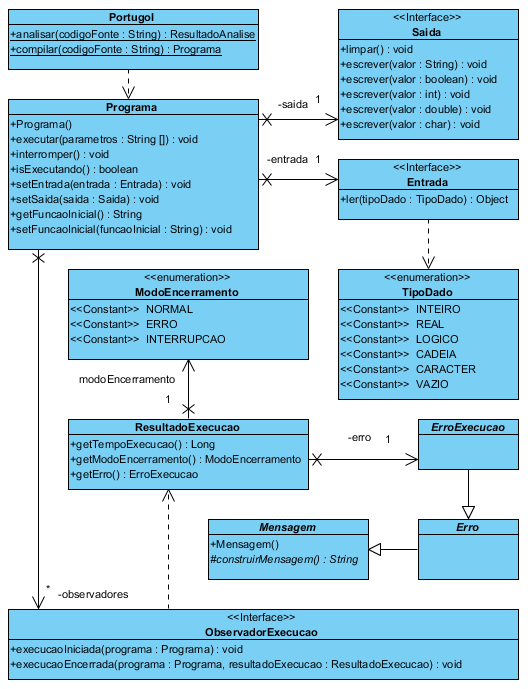


* + 1. Execução de programas

O núcleo do Portugol provê um mecanismo para a execução dos programas escritos em Portugol. A execução dos programas é feita percorrendo a ASA e interpretando cada nó no momento em que é encontrado. O núcleo permite que vários programas diferentes sejam executados ao mesmo tempo.

Da mesma forma que os demais serviços do Portugol, a execução de programas também é abstraída por um conjunto reduzido de classes de fachada. Para uma melhor compreensão, o diagrama da Figura 6 mostra este conjunto de classes e seus relacionamentos.

Figura 6. Diagrama de classes do serviço de execução de programas



Para que uma aplicação possa utilizar o serviço de execução de programas do Portugol, as interfaces “Entrada” e “Saída”, representadas no diagrama, devem ser obrigatoriamente implementadas.

A interface “Entrada” é responsável pela leitura dos valores de entrada do programa utilizando o comando “leia”. Para realizar esta leitura, a interface provê o método “ler” passando como argumento o tipo de dado esperado. A aplicação ao implementar esta interface deve retornar um objeto correspondente ao tipo de dado solicitado. A entrada dos dados pode ser feita de forma interativa com o usuário ou de forma automática. O Quadro 4 apresenta um trecho de código que demonstra uma implementação interativa:

Quadro 4. Exemplo de implementação da interface de entrada de dados

|  |
| --- |
| **import static** javax.swing.JOptionPane.showInputDialog;  **public final** **class** IDE **implements** Entrada  {  **@Override**  **public** Object **ler**(TipoDado tipoDado) **throws** Exception  {  String tipo = tipoDado.toString();  String mensagem = **"Digite um valor do tipo %s:"**;  String entrada = showInputDialog(String.format(mensagem, tipos));    **switch** (tipoDado)  {  **case** **INTEIRO**: **return** Integer.parseInt(entrada);  **case** **REAL**: **return** Double.parseDouble(entrada);  **case** **CARACTER**: **return** entrada.charAt(0);  **case** **CADEIA**: **return** entrada;  **case** **LOGICO**:  {  **if** (entrada.equals(**"verdadeiro"**)) **return** **true**;  **else**  **if** (entrada.equals(**"falso"**)) **return** **false**;  }  }  **return** **null**;  }  } |

A interface “Saída” é responsável pela escrita dos valores de saída do programa utilizando o comando “escreva”. Para realizar esta escrita, a interface provê cinco sobrecargas do método “escrever”, uma para cada tipo de dado, passando como argumento o valor a ser escrito. Esta interface provê ainda o método “limpar” utilizado para limpar a saída de dados.

A implementação da saída de dados pode ser feita de qualquer forma, ficando a critério da aplicação. É possível, por exemplo, redirecioná-la para alimentar um banco de dados ou a entrada de outros algoritmos. O Quadro 5 apresenta um trecho de código que demonstra uma saída de dados simples onde os valores são impressos no console da JVM (Java Virtual Machine).

Quadro 5. Exemplo de implementação da interface de saída de dados

|  |
| --- |
| **public** **final** **class** IDE **implements** Saida  {  **@Override**  **public** **void** **escrever**(String cadeia) **throws** Exception  {  System.out.print(cadeia);  }  **@Override**  **public** **void** **escrever**(**boolean** logico) **throws** Exception  {  System.out.print(logico? **"verdadeiro"** : **"falso"**);  }  **@Override**  **public** **void** **escrever**(**int** inteiro) **throws** Exception  {  System.out.print(inteiro);  }  **@Override**  **public** **void** **escrever**(**double** real) **throws** Exception  {  System.out.print(real);  }  **@Override**  **public** **void** **escrever**(**char** caracter) **throws** Exception  {  System.out.print(caracter);  }  **@Override**  **public** **void** **limpar**() **throws** Exception  {  **for** (**int** i = 0; i < 20; i++)  {  System.out.println();  }  }  } |

Ao implementar as interfaces “Entrada” e “Saída” deve-se tomar muito cuidado com a utilização de threads, pois a entrada e saída de dados do Portugol ocorre de forma síncrona. Em outras palavras, quando uma instrução de entrada ou saída é encontrada, o núcleo do Portugol fica aguardando até que a leitura/escrita termine de ser processada para então continuar a execução do programa. Se as instruções de entrada e saída forem realizadas dentro de threads, cabe à aplicação garantir que os métodos das interfaces não irão retornar até que os valores tenham sido lidos e escritos, caso contrário ocorrerá erros na execução dos programas.

Existe ainda uma terceira interface disponibilizada pelo núcleo do Portugol chamada “ObservadorExecucao”. Não é obrigatório implementá-la para utilizar o serviço de execução de programas, no entanto, ao fazê-lo é possível monitorar o estado da execução dos programas. Para permitir este monitoramento, esta interface provê dois métodos: “execucaoIniciada” e “execucaoEncerrada”.

O método “execucaoIniciada” é chamado pelo núcleo do Portugol antes de iniciar a execução de um programa e recebe por parâmetro a instância do programa que está sendo iniciado. Este método pode ser utilizado para realizar ações como, desabilitar a edição de código fonte na IDE durante a execução dos programas. O Quadro 6 ilustra um trecho de código que demonstra a utilização deste método imprimindo uma mensagem na saída de dados.

Quadro 6. Monitorando o início da execução de um programa

|  |
| --- |
| **public** **final** **class** IDE **implements** ObservadorExecucao  {  **@Override**  **public** **void** **execucaoIniciada**(Programa programa)  {  System.out.println(**"Iniciando a execução do programa..."**);  }  } |

O método “execucaoEncerrada” é chamado pelo núcleo do Portugol após o encerramento de um programa. Existem três modos diferentes de encerramento de um programa: encerramento normal, encerramento por interrupção e encerramento ocasionado por erro. O encerramento normal ocorre quando todas as instruções foram interpretadas e o programa chegou ao seu fim. O encerramento por interrupção ocorre quando o usuário interrompe manualmente o programa por algum motivo. O encerramento ocasionado por erro ocorre quando o programa encontra um erro em tempo de execução (Ex.: divisão por zero; acesso a um índice de vetor inválido).

O método “execucaoEncerrada” ao ser chamado recebe por parâmetro o programa que estava sendo executado e um objeto do tipo “ResultadoExecucao”, a partir do qual é possível obter três informações importantes: o tempo de execução do programa, o modo de encerramento do programa e, caso tenha ocorrido, o erro que ocasionou o encerramento. O Quadro 7 ilustra um trecho de código que demonstra a utilização deste método.

Quadro 7. Monitorando o encerramento de um programa

|  |
| --- |
| **public** **final** **class** IDE **implements** ObservadorExecucao  {  **@Override**  **public** void **execucaoEncerrada**(Programa p, ResultadoExecucao resultado)  {  Long tempo = resultado.getTempoExecucao();  String mensagem = **"Tempo de execução: %d milissegundos."**;  **switch** (resultado.getModoEncerramento())  {  **case** **NORMAL**:  System.out.println(**"Programa finalizado."**);  **break**;  **case** **INTERRUPCAO**:  System.out.println(**"Programa interrompido."**);  **break**;  **case** **ERRO**:  System.out.println(resultado.getErro().getMensagem());  **break**;  }    System.out.println(String.format(mensagem, tempo));  }  } |

Uma vez que estas interfaces tenham sido implementadas, é necessário registrá-las no núcleo do Portugol. Isto é feito através dos métodos “setEntrada”, “setSaida” e “adicionarObservadorExecucao” do objeto “Programa”. Por fim, a execução pode ser iniciada realizando-se uma chamada ao método “executar” do objeto “Programa”. Após iniciada, a execução do programa pode ser interrompida a qualquer momento chamando-se o método “interromper” do objeto “Programa”. Também é possível verificar se um programa está em execução através do método “isExecutando”.

O Portugol permite ainda, definir o ponto de partida para a execução. Isto é feito através do método “setFuncaoInicial” do objeto “Programa”, passando-se o nome de uma função. O programa a ser executado deverá obrigatoriamente conter esta função, caso contrário será lançado um erro e a execução não ocorrerá. Se a função inicial não for informada, o Portugol assume por padrão a função “inicio”. Além disso, é possível passar parâmetros para o programa através do método “executar”. Neste caso, a função inicial deve declarar um vetor do tipo cadeia. O Quadro 8 ilustra um trecho de código que demonstra a utilização do serviço de execução de programas.

Quadro 8. Exemplo de utilização do serviço de execução de programas

|  |
| --- |
| **public** **final** **class** IDE **implements** Entrada, Saida, ObservadorExecucao  {  **public** **static** **void** ***main***(String[] args)  {  IDE ide = **new** IDE();  ide.iniciar(args);  }  **public** **void** **iniciar**(String[] parametros)  {  File arquivo = **new** File(**"C:\\exemplo\\algoritmo.por"**);  String algoritmo = CodigoFonte.*carregar*(arquivo);  **try**  {  Programa programa = Portugol.*compilar*(algoritmo);  programa.setEntrada(**this**);  programa.setSaida(**this**);  programa.adicionarObservadorExecucao(**this**);  programa.setFuncaoInicial(**"main"**);  programa.executar(parametros);  **/\***  **\* No Portugol, os programas são executados em threads,**  **\* logo, os comandos abaixo são executados imediatamente**  **\* após a chamada ao método "executar".**  **\*/**  **if** (programa.isExecutando())  {  **try** { Thread.*sleep*(**2500**); }  **catch** (Exception excecao) { }  programa.interromper();  }  }  **catch** (ErroCompilacao erro)  {  exibirErros(erro.getResultadoAnalise());  }  }  } |

1. Integrando com C#

Os passos descritos neste manual são específicos para o ambiente de programação VisualStudio 2010, no qual o módulo de integração C# foi desenvolvido. Caso você opte por utilizar versões anteriores da ferramenta, consulte a documentação da mesma para saber como converter o código fonte para a versão utilizada e realizar os passos descritos no manual.

* 1. Compilando os módulos necessários

O mecanismo de integração do Portugol divide-se em dois módulos que precisam ser incluídos na aplicação para realizar a integração, são eles: módulo Java e módulo C#. Este capítulo descreve os passos necessários para compilar e incluir corretamente estes módulos.

* + 1. Módulo Java

O módulo Java é a parte do mecanismo de integração do Portugol responsável por inicializar os serviços do CORBA e converter os objetos Java em objetos CORBA e vice versa.

Para compilar este módulo, abra o projeto “Portugol-Integracao” no NetBeans, certificando-se de marcar as opções “Abrir como projeto principal” e “Abrir projetos requeridos”. Depois, clique com o botão direito do mouse sobre o projeto e selecione a opção “Limpar e construir”.

Ao selecionar esta opção, o NetBeans irá compilar o código fonte do módulo Java e de todos os projetos dependentes. No final da compilação, será gerada uma pasta com o nome “dist” dentro da pasta do projeto, contendo os arquivos jar do módulo Java e das bibliotecas/módulos utilizados por ele. Estes arquivos serão incluídos posteriormente na aplicação em C#.

* + 1. Módulo C#

O módulo C# é a parte do mecanismo de integração do Portugol responsável por inicializar e finalizar a biblioteca IIOP.NET e o módulo Java.

Para compilar este módulo, abra a solução “Portugol-Integracao-CSharp” no Visual Studio e verifique se a solução está com a configuração “Debug”. Se estiver, altere para a configuração “Release” e pressione a tecla “F6” para compilar a solução.

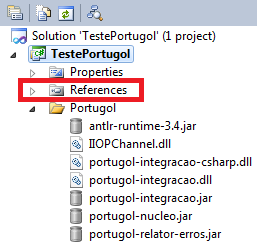
Se a compilação for bem sucedida, a mensagem “Build Succeeded” será exibida na barra de status do Visual Studio. Caso contrário, será exibida uma mensagem de erro na tela. No final da compilação, os arquivos executáveis do módulo terão sido gerados dentro da pasta do projeto: “Projetos\Portugol-Integracao-CSharp\Portugol-Integracao-CSharp\bin\Release”. Estes arquivos serão incluídos posteriormente na aplicação C#.

Para testar a compilação do módulo e verificar se tudo está funcionando corretamente, você pode executar a aplicação de testes que acompanha o módulo C#. Clique com o botão direito do mouse sobre o projeto “Portugol-Integracao-CSharp-Teste” e selecione a opção “Set as StartUp Project” para definí-lo como projeto principal. Depois pressione a tecla “F5” para executar o projeto. Agora basta seguir as instruções que vão aparecendo na tela.

* 1. Incluindo e configurando os módulos necessários

Para que o mecanismo de integração possa funcionar, os arquivos executáveis do módulo Java e do módulo C# devem ser incluídos e configurados corretamente na aplicação C#. Para começar, abra a solução de sua aplicação no Visual Studio e crie uma pasta para colocar os arquivos executáveis dos módulos de integração. Se a solução possuir mais de um projeto, crie a pasta no projeto principal. Você pode colocar qualquer nome na pasta. Para os exemplos do manual, será utilizada a pasta “Portugol”.

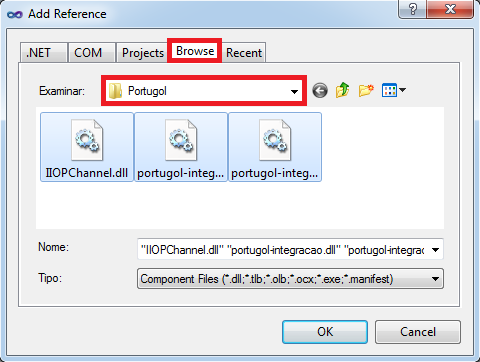
Figura 7. Lista de arquivos



Agora, copie para esta pasta todos os arquivos que foram gerados durante a compilação do módulo Java, realizada no Capítulo 3.1.1. Copie também todos os arquivos gerados durante a compilação do módulo C#, realizada no Capítulo 3.1.2. Ao final, você deverá ter algo parecido com a Figura 7.

Após incluir os arquivos, é necessário referenciá-los no projeto. Para isso, clique com o botão direito do mouse sobre o item “References”, em destaque na Figura 7, e selecione a opção “Add Reference...”. Na tela a seguir (Figura 8), clique na guia “Browse”, abra a pasta que contém os arquivos dos módulos e selecione todos os arquivos que aparecem na tela. Depois clique no botão “OK” para confirmar a operação.

Figura 8. Referenciando os arquivos dos módulos

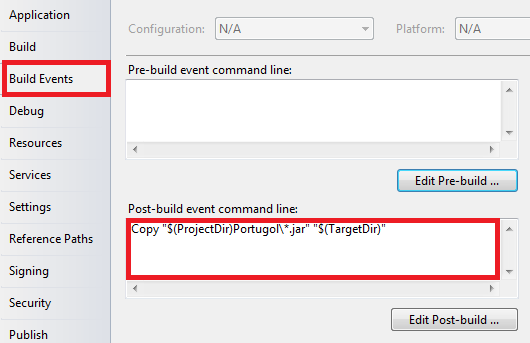


Para verificar se as referências foram adicionadas corretamente, abra o item “References” e certifique-se de que os seguintes itens são listados: “IIOPChannel”, “portugol-integracao” e “portugol-integracao-csharp”.

Por último, é preciso configurar o Visual Studio para que ele copie os arquivos para o diretório de saída no momento da compilação. Se isto não for feito, a aplicação não encontrará os arquivos durante a execução e não funcionará.

Para fazer isto, clique com o botão direito do mouse sobre o nome do seu projeto e selecione a opção “Propriedades”. Na tela seguinte (Figura 9), selecione a guia “Build Events” localizada à esquerda da janela e digite o comando ‘Copy “$(ProjectDir)\Portugol\\*.jar” “$(TargetDir)”’ no campo “Post-build event command line”. Depois basta salvar o projeto para concluir a alteração. Observação: você deve substituir a palavra “Portugol” pelo nome da pasta que você criou.

Figura 9. Configurando a cópia dos arquivos



* 1. Gerenciando o serviço de integração

O módulo Java e o módulo C# do mecanismo de integração do Portugol foram projetados de forma a exigir o mínimo de esforço para realizar a integração. Desta forma, estes módulos conseguem gerenciar sozinhos todos os aspectos da arquitetura CORBA, restando à aplicação apenas o trabalho de inicializar e finalizar o serviço de integração.

O serviço de integração deve ser inicializado antes que seja feita qualquer chamada aos serviços do núcleo. Da mesma forma, o serviço deve ser finalizado após os objetos do Portugol não estarem mais sendo utilizados. Para isto, o ideal é inicializar o serviço quando a aplicação inicia e finalizá-lo quando a aplicação encerra (Ex.: quando formulário principal é fechado).

A seguir será apresentada uma solução modelo que pode ser utilizada para gerenciar o serviço de integração. Para começar, localize e abra a classe da aplicação que possui o método “Main”. Este método inicializa a aplicação e geralmente está na classe “Program”. O código fonte é similar ao apresentado no Quadro 9.

Quadro 9. Exemplo de código da classe “Program”

|  |
| --- |
| namespace TestePortugol  {  static class Program  {  [STAThread]  static void Main()  {  Application.EnableVisualStyles();  Application.SetCompatibleTextRenderingDefault(false);  Application.Run(new Form1());  }  }  } |

Agora, modifique-o para que fique igual ao Quadro 10. Observação: as declarações de “usings” de sistema (Ex.: System.Windows.Forms) foram omitidas em ambos os quadros para facilitar a visualização das alterações.

Quadro 10. Inicializando o serviço de integração

|  |
| --- |
| using br.univali.portugol.integracao.csharp;  using br.univali.portugol.integracao;  namespace TestePortugol  {  static class Program  {  public static Portugol Portugol;  [STAThread]  public static void Main()  {  try  {  Portugol = ServicoIntegracao.getInstance().iniciar();  Application.EnableVisualStyles();  Application.SetCompatibleTextRenderingDefault(false);  Application.Run(new Form1());  }  catch (Exception excecao)  {  MessageBox.Show("Erro: " + excecao.Message);  }  }  }  } |

Feito isto, localize o formulário principal da aplicação e implemente a finalização do serviço no evento “FormClosed”. O código para finalização do serviço deve ficar similar ao Quadro 11.

Quadro 11. Finalizando o serviço de integração

|  |
| --- |
| namespace TestePortugol  {  public partial class Form1 : Form  {  public Form1()  {  InitializeComponent();  }  private void Form1\_FormClosed(object sender, FormClosedEventArgs e)  {  try  {  Program.Portugol.encerrar();  }  catch (Exception excecao)  {  // Se houver algum erro não faz nada e sai da aplicação  }  }  }  } |

Estes são os únicos passos necessários para inicializar e finalizar o serviço de integração. Uma vez que o serviço esteja sendo gerenciado corretamente, você poderá utilizar os serviços do núcleo do Portugol sem nenhuma preocupação.

* 1. Utilizando os serviços do núcleo

Para utilizar os serviços do núcleo no C#, você pode seguir as mesmas orientações descritas no Capítulo 2.2 O Capítulo 2.2 apresenta cada um dos serviços do núcleo, apresentando exemplos de utilização. Existem apenas algumas diferenças que devem ser observadas, as quais serão descritas neste Capítulo.

No C#, todos os objetos do Portugol estão organizados da mesma forma que no Java, porém estão no namespace “br.univali.portugol.integracao”. Então, se no Java o objeto é acessado como “br.univali.portugol.nucleo.Programa”, no C# ele será acessado como “br.univali.portugol.integracao.Programa”. Os trechos de código destacados em cinza no Quadro 12 e no Quadro 13 demonstram as diferenças entre as duas aplicações.

Quadro 12. Usando o Portugol em uma aplicação em Java

|  |
| --- |
| **package** MinhaAplicacao;  **import** br.univali.portugol.nucleo.\*;  **import** br.univali.portugol.nucleo.asa.\*;  **import** br.univali.portugol.nucleo.analise.\*;  **public** **final** **class** Aplicacao  {  **public** **static** **void** ***main***(String[] args) **throws** ErroCompilacao  {  String algoritmo = CodigoFonte.*carregar*(**"C:\\algoritmo.por"**);    Programa programa = Portugol.*compilar*(algoritmo);  programa.executar(**new** String[] {});  }  } |

Quadro 13. Usando o Portugol em uma aplicação C#

|  |
| --- |
| using br.univali.portugol.integracao;  using br.univali.portugol.integracao.asa;  using br.univali.portugol.integracao.analise;  namespace MinhaAplicacao  {  static class Aplicacao  {  [STAThread]  public static void Main()  {  string algoritmo = CodigoFonte.carregar"C:\\algoritmo.por");  Programa programa = Portugol.compilar(algoritmo);  programa.executar(new string[] {});  }  }  } |

No C#, as interfaces “Entrada”, “Saida”, “ObservadorExecucao” e “VisitanteASA” devem obrigatoriamente estender a classe “MarshalByRefObject” e possuir a anotação “[Serializable]”. Estas interfaces também NÃO podem ser implementadas por uma única classe, ao invés disso, cada uma das interfaces deve ser implementada por uma classe diferente.

Se estas observações não forem atendidas, ocorrerão exceções do CORBA na aplicação. Para uma melhor compreensão, o Quadro 14 demonstra uma implementação INCORRETA enquanto que o Quadro 15 demonstra uma implementação CORRETA.

Quadro 14. Exemplo de implementação incorreta

|  |
| --- |
| using System;  using br.univali.portugol.integracao.asa;  using br.univali.portugol.integracao.execucao;  namespace MinhaAplicacao  {  /\*  \* 1 - Não foi adicionada a anotação [Serializable]  \* 2 – A classe Aplicacao não estende a classe MarshalByRefObject  \* 3 - A classe Aplicacao implementa sozinha as 3 interfaces  \*/  class Aplicacao : Entrada, Saida, ObservadorExecucao, VisitanteASA  {  [...]  }  } |

Quadro 15. Exemplo de implementação correta

|  |
| --- |
| using System;  using br.univali.portugol.integracao.asa;  using br.univali.portugol.integracao.execucao;  namespace MinhaAplicacao  {  /\*  \* 1 - Cada interface foi implementada por uma classe diferente  \* 2 - Cada uma das classes foi anotada com a anotação [Serializable]  \* 3 - Cada uma das classes estende a classe MarshalByRefObject  \*/  public class Aplicacao  {  private SuporteEntrada SuporteEntrada = new SuporteEntrada();  private SuporteSaida SuporteSaida = new SuporteSaida();  private SuporteExecucao SuporteExecucao = new SuporteExecucao();  }  [Serializable]  class SuporteEntrada : MarshalByRefObject, Entrada {...}  [Serializable]  class SuporteSaida : MarshalByRefObject, Saida {...}  [Serializable]  class SuporteExecucao : MarshalByRefObject, ObservadorExecucao {...}  [Serializable]  class GeradorAssembly : MarshalByRefObject, VisitanteASA {...}  } |

* 1. Limitações

Além das observações apontadas no Capítulo 3.4, existem algumas limitações no módulo C# que devem ser levadas em consideração.

A primeira limitação do módulo C# está no uso da biblioteca IIOP.NET como implementação CORBA. A biblioteca foi escrita utilizando os recursos da versão 2.0 da plataforma .NET. Com isto, as aplicações que utilizam versões anteriores da plataforma, precisam ser migradas para a versão 2.0 para que possam utilizar o Portugol.

A segunda limitação também está relacionada à biblioteca IIOP.NET. O arquivo DLL gerado pelo compilador IDL da biblioteca, sempre utiliza a versão da plataforma .NET que está em uso no momento da compilação do arquivo IDL. Por exemplo, se o arquivo IDL for compilado em um computador com a versão 3.5 do framework, a DLL gerada utilizará a versão 3.5. O arquivo DLL que vem com o código fonte do módulo C#, foi compilado em um computador com a versão 4.0 da plataforma. Isto significa que as aplicações que quiserem utilizar o Portugol precisarão ser migradas para a versão 4.0.