

Compilando código Java pela aplicação em tempo de execução

Sebastian David C. de O. Galvão

Crie você mesmo o seu editor de Java. Aprenda a compilar código Java em tempo real direto pela sua aplicação sem usar o javac e depois executá-lo.

Introdução

Você já pensou em criar o seu próprio editor Java? Ou já necessitou alguma vez de compilar código Java diretamente da sua aplicação? Se a resposta para alguma dessas perguntas for sim, este tutorial pode ajudar bastante.

Para a criação da parte visual de um simples editor de código bastam alguns componentes em Swing para a interface com o usuário. Isso não será abordado neste artigo. Caso seja de interesse, consulte o tutorial de Swing no site da própria Sun (<http://java.sun.com>) .

O foco principal deste artigo será em como compilar seu código Java de dentro da sua aplicação, em tempo de execução, sem precisar usar o javac. Alguns poderiam tentar usar a classe *Runtime* e daí chamar o javac para compilar, porém podemos usar uma das ferramentas embutidas na J2SE1.4.

Para demonstrar os conceitos será apresentado um programa que é um mini-editor de Java: o usuário entra com um nome para a classe, entra com o código Java num JTextArea e pode compilar e rodar o código digitado.

Reflection

É interessante que o leitor tenha algum conhecimento em *Reflection* para aproveitar melhor o exemplo que será dado. Também conhecido como introspecção, serve basicamente para obter informações sobre as classes em tempo de execução, como, por exemplo, invocar um método cujo nome é desconhecido ou listar todas as propriedades de uma classe.

Para mais informações acesse:
<http://java.sun.com/docs/books/tutorial/reflect/index.html>

O compilador Java

O comando *javac* é simplesmente um empacotador da classe *com.java.tools.javac.Main*. Esta classe possui um método estático chamado *compile* que recebe como parâmetro um array de Strings contendo os nomes dos arquivos a serem compilados. Assim não é necessário usar uma linha de comando para compilar as classes.

Por exemplo:

```
com.sun.tools.javac.Main.compile(new String[]{"Exemplo.java"});
```

O código acima irá compilar o arquivo Exemplo.java. A saída, como os erros por exemplo, é a padrão neste caso em que não passamos o segundo parâmetro. Este parâmetro é um *PrintWriter* e serve caso seja desejado capturar a saída para outro destino.

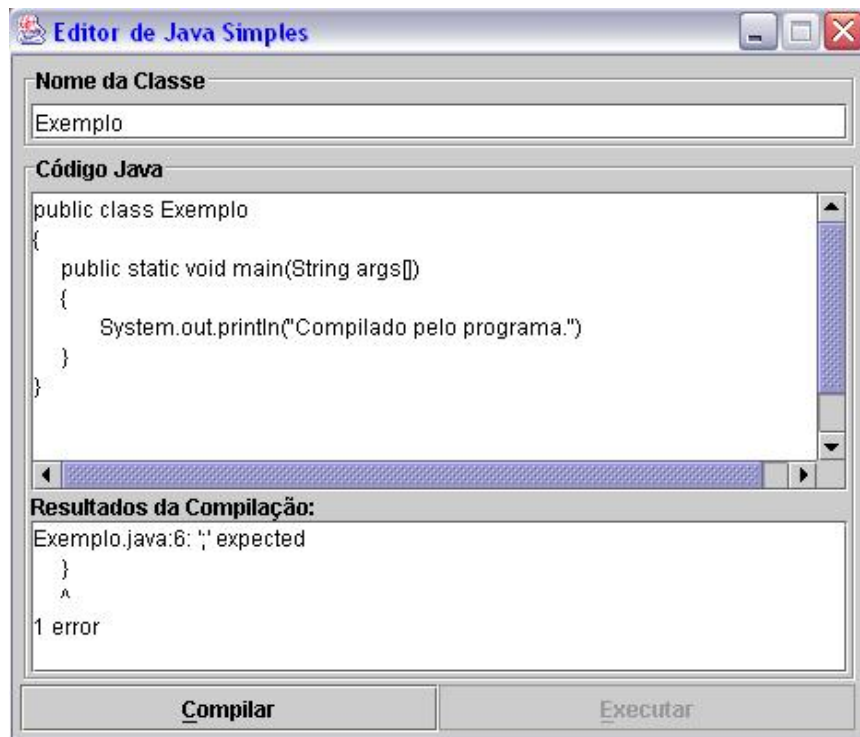
Para este código funcionar é necessário que se inclua no *classpath* o **tools.jar** que vem no diretório lib da instalação do J2SE1.4.

Vale lembrar que a API demonstrada aqui é dependente da instalação. É recomendado que se use a padrão recomendada pela Sun. Nos computadores Apple, por exemplo, essa organização é diferente. Na versão 1.5 (Tiger) da J2SE, está planejada a introdução de uma API mais portátil.

Programa de exemplo

O melhor jeito de se entender um novo conceito costuma ser apresentando um exemplo. Não será viável apresentar aqui todo o código do programa, mas mostraremos as partes principais. Caso deseje a versão completa baixe aqui do site.

A figura abaixo ilustra a interface gráfica do programa:



O programa de exemplo.

Basicamente o processo de compilar na implementação deste programa consiste nos seguintes passos:

1. gravar o código digitado pelo usuário no disco. Atribui como nome do arquivo o Nome da Classe entrado pelo usuário concatenado com ".java".
2. Compila o arquivo. Aqui que entra a classe *com.sun.tools.javac.Main*.
3. Executa o novo programa caso tenha um método Main(). Nessa parte que entram os conceitos de *Reflection*.

Em seguida são apresentados os códigos de cada uma dessas etapas. Para ter acesso ao código completo original, faça o download a partir daqui do site.

Para compilar o programa de exemplo é necessário adicionar o *tools.jar* ao classpath. Este arquivo contém as classes do compilador. Pode-se usar o argumento `-classpath` no momento de compilar:

Windows: `javac -classpath c:\j2se1.4.2\lib\tools.jar Editor.java`

Unix: `javac -classpath /homedir/jdk14/j2sdk1.4.2/lib/tools.jar Editor.java`

Como o compilador será usado em tempo de execução, também será necessário adicionar o *tools.jar* ao *classpath* no momento de executar o programa:

Windows: `java -classpath c:\jdk1.4.2\lib\tools.jar;. Editor`

Unix: `java -classpath /homedir/jdk14/j2sdk1.4.1/lib/tools.jar:. Editor`

Depois digite algum código em Java no JTextArea e clique em compilar. Caso o programa esteja correto, será exibido "Compilado corretamente" e então clique em Executar. O seu programa, que foi compilado em tempo de execução por outro programa Java, será executado. Se o novo programa for baseado em Swing uma nova janela será aberta.

Para efeitos de demonstração serão apresentados os códigos-fontes dos principais métodos da classe Editor, que é o programa de exemplo.

O método `Compilar()` descrito nos itens 1 e 2 acima:

```
import java.lang.reflect.*;

private void Compilar()
{
    try
    {
        /* deixa um log da compilacao num arquivo chamado logCompilacao.txt */
        PrintWriter saida = new PrintWriter(new FileWriter("logCompilacao.txt"));

        /* grava o codigo-fonte no disco */
        String arquivoFonte = (nomeClasse = txtNomeClasse.getText()) + ".java";
        FileWriter arq = new FileWriter(arquivoFonte);
        arq.write(txtCodigo.getText()); //grava no arquivo o codigo
        arq.close();

        /* agora sim compila, onde entra a grande sacada
         * de utilizar o compilador Java da Sun.
         * saida é onde sera gravada a saida do compilador (erros por exemplo),
         * em vez de usar a saída padrão System.out
         */
        int resultadoCompilacao = com.sun.tools.javac.Main.compile(
            new String[]{arquivoFonte},saida);
        if (resultadoCompilacao == 0)
        {
            txtResultados.append("Compilado com sucesso.");
            btnExecutar.setEnabled(true); //ativa o botao executar
        }
        else
        {
            btnExecutar.setEnabled(false); //desativa o botao executar

            /* le o arquivo de resultados e imprime na tela */
            BufferedReader result = new BufferedReader(
                new FileReader("logCompilacao.txt"));
            String linha;
            while((linha=result.readLine())!=null)
            {
                txtResultados.append(linha+"\n");
            }
        }
        saida.close();
    }
    catch(IOException ioe)
    {
        System.out.println("Erros ao gravar arquivo: \n"+ioe.getMessage());
        ioe.printStackTrace();
    }
}
```

O método `Executar()` descrito no item 3:

```
import java.lang.reflect.*;

private void Executar()
{
    try
    {
        Object parametrosObjeto[] = {new String[]{}}; //parametros do metodo
        Class parametrosClasse[] = {parametrosObjeto[0].getClass()}; //classe dos parametros
        Class classe = Class.forName(nomeClasse); //referencia a classe
        Object instancia = classe.newInstance(); //instancia a classe
        Method metodoMain = classe.getDeclaredMethod("main", parametrosClasse);

        metodoMain.invoke(instancia, parametrosObjeto); //executa o metodo
    }
    catch (Exception e)
    {
        /* varias excecoes podem ser lancadas ao usar os recursos de reflection, mas
         * nao as trataremos a fundo aqui. Apenas verificamos se alguma delas foi lançada.
         */
        e.printStackTrace();
    }
}
```

Variáveis de fora do escopo dos métodos `Compilar()` e `Executar()`:

- `txtNomeClasse`: `JTextField` onde o usuário entra com o nome da classe
- `txtCodigo`: `JTextArea` onde o usuário digita o código Java a ser compilado.
- `txtResultados`: `JTextArea` onde são mostradas as saídas do compilador.

Conclusão

Algumas vezes podemos nos encontrar numa situação em que é necessário compilar códigos de dentro do nosso próprio programa. Seja para criar o nosso próprio editor em Java, para apenas a verificação da consistência de algumas classes ou para qualquer outro motivo. De qualquer maneira é uma carta na manga do bom programador Java.

Sebastian David C. de O. Galvão (sebastian.david@terra.com.br) é desenvolvedor Java há mais de 2 anos e trabalha numa empresa de referência nacional em desenvolvimento de softwares educacionais.