Apêndice A

Utilizando o Dr. Java

Neste apêndice introduzimos a ferramanta `DrJava`, um ambiente de desenvolvimento para a linguagem `Java`. Por ser ele mesmo escrito em `Java`, pode ser usado em diversos ambientes, como, por exemplo, `Linux`, `Windows` e `Mac OS`. Um ambiente de desenvolvimento (também conhecido por IDE, de *Integrated Development Environment*) é um conjunto de ferramentas integradas que auxiliam a construção de programas.

O `DrJava` se encontra em desenvolvimento e, por isso, alguns recursos desejáveis ainda não estão disponíveis. Entretanto, os recursos mais simples que ele fornece já são apropriados para os objetivos deste livro. Planejamos então utilizar o `DrJava` para escrevermos nossos programas `Java`.

Objetivos

Neste apêndice forneceremos uma breve introdução ao uso do `DrJava`. O conteúdo é limitado e específico, suficiente para que você posteriormente seja capaz de conhecer melhor esta ferramenta por conta própria. Recomendamos para isso a consulta de manuais e outros documentos, não necessariamente sobre `DrJava` apenas. As descobertas pelo próprio uso e através de dicas de colegas também são incentivadas.

Neste apêndice você aprenderá a utilizar o `DrJava` para:

- escrever, compilar, manipular e depurar classes/objetos simples;
- gravar e reutilizar os arquivos que descrevem seus objetos.

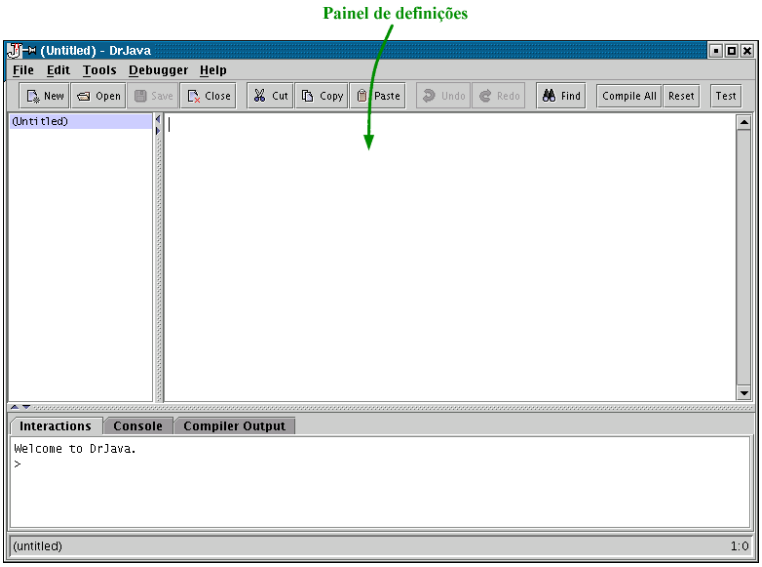
O `DrJava` pode ser obtido no endereço `http://drjava.org/`. Neste sítio é possível ainda encontrar documentação detalhada sobre como instalar e utilizar o `DrJava`.

A.1 Conversor de Temperatura simples

Como exemplo inicial, vamos construir a classe `Conversor` descrita no Capítulo 3. Lembrando, cada objeto dessa classe converte apenas a temperatura de 40 graus Celsius para a correspondente em graus Fahrenheit. Ao receber a mensagem `celsiusParaFahrenheit`, a classe `Conversor` devolve a temperatura em Fahrenheit equivalente a 40 graus Celsius.

Editando o código-fonte num arquivo

Vejamos o que o `DrJava` nos oferece para criarmos a classe. Ao iniciar o ambiente `DrJava`, abre-se uma janela parecida com a seguinte.



O painel de definições, indicado na figura acima, é um editor de textos. É nele que digitaremos o código `Java` que define a classe `Conversor`. Ele se parece muito com um editor de textos comum, exceto que possui alguns recursos para facilitar a digitação de código. Em particular, o comportamento das teclas <Enter> e <Tab> favorece a indentação do código. Outro recurso útil é a coloração e o destaque do texto.

Precisamos criar um arquivo novo para conter o código da nossa classe. Para isso, bastaria escolher a opção **New** do menu **File**. Porém, quando abrimos o `DrJava`, um novo arquivo sem nome já foi criado, então podemos usá-lo nesse momento (em vez de criar um novo). Sendo assim, digite o seguinte código no painel de definições.

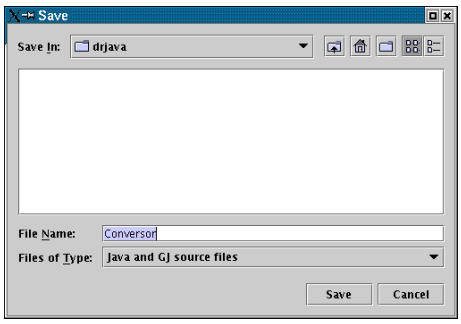
```
class Conversor
{
  int celsiusParaFahrenheit ()
  {
    return 9 * 40 / 5 + 32;
  }
}
```

Ao digitar, note os recursos mencionados que ajudam a programação (principalmente o comportamento das teclas <Enter> e <Tab>).

Gravando e reutilizando o arquivo

Vamos agora gravar o arquivo no disco. Escolha o menu **File** (clicando com o *mouse* ou usando as teclas de atalho, que nesse caso é a combinação <Alt>+<F>) e escolha o item **Save**. Como o arquivo ainda não tem

nome, estaremos na verdade acionando a opção **Save as...**. Por isso, surgirá um *diálogo* para definirmos a localização e o nome do arquivo, como mostra a seguinte figura.



Podemos determinar a localização, manipulando a *combo box* com rótulo **Save In:** (no topo do diálogo) e escolhendo um diretório na caixa abaixo dela, ou podemos deixar a localização como está. Desse modo, provavelmente o arquivo será gravado no diretório de onde o `DrJava` foi chamado.

Também precisamos escolher o nome do arquivo. Em algumas ocasiões, este deve ser *obrigatoriamente* idêntico ao nome da classe, mais o sufixo .java. Não é o nosso caso, mas mesmo assim vamos chamar o arquivo de Conversor.java. Como já digitamos o código da classe, o `DrJava` preencheu o nome do arquivo no *input field* de rótulo **File Name:** com o nome da classe. Note também que ele não acrescentou o sufixo .java no nome, o que será feito implicitamente quando finalizarmos (mas não há problema em digitar o sufixo mesmo assim).

Para confirmar a gravação, basta clicar no botão **Save**. As modificações futuras podem ser gravadas com o comando **Save**, sem precisar escolher o nome do arquivo novamente.

Com os arquivos das classes gravados em disco, podemos querer reutilizá-los no futuro. No `DrJava`, basta escolhermos a opção **Open** do menu **File** e um diálogo permitirá que você escolha o arquivo que deseja abrir novamente (é semelhante ao processo do **Save as...**).

Compilando a classe

Acabamos de definir a nossa classe, precisamos agora compilar para que possamos usá-la. No menu **Tools**, temos duas opções para fazer isso, **Compile All Documents** e **Compile Current Document**. Como só temos um documento aberto (Conversor.java), qualquer uma das opções serve. Escolha então **Compile Current Document**.

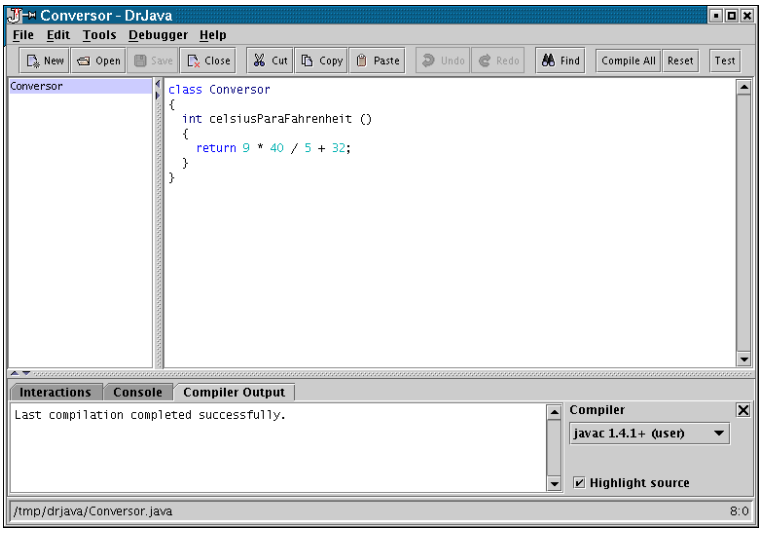
Com isso compilaremos a nossa classe. Note que, no painel inferior da janela, a *guia* **Compiler Output** se abre mostrando algumas mensagens. Se tudo der certo, a nossa janela se parecerá com a seguinte.

Usando a classe

Podemos finalmente usar a guia **Interactions**, que chamaremos de *janela do interpretador*, para criar e usar objetos da classe `Conversor`. Essa janela recebe comandos num *prompt* e a sintaxe desses comandos é muito parecida com a da linguagem `Java`.

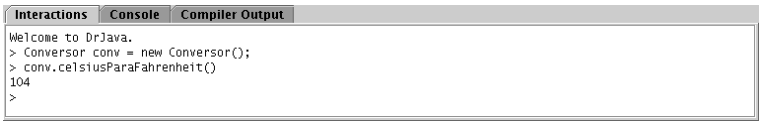
Clique então em **Interactions** e digite o seguinte.

```
Conversor conv = new Conversor();
conv.celsiusParaFahrenheit()
```



A ausência de `;` no final da linha do comando faz com que o interpretador imprima o valor devolvido pelo comando.

A janela do interpretador deverá se parecer com a figura abaixo.



A.2 Tratando erros

Utilizaremos agora a classe `Conversor4` descrita no Capítulo 3 para mostrarmos alguns outros recursos do `DrJava`. Para quem não lembra, objetos dessa classe convertem temperaturas entre graus Celsius e Fahrenheit fornecendo os seguintes métodos.

* `double celsiusParaFahrenheit (double c)`: recebe uma temperatura `c` em graus celsius e devolve a temperatura correspondente em graus fahrenheit. * `double fahrenheitParaCelsius (double f)`: recebe uma temperatura `f` em graus fahrenheit e devolve a temperatura correspondente em graus celsius.

Neste apêndice, porém, usaremos um código um pouco diferente em relação ao do Capítulo 3.

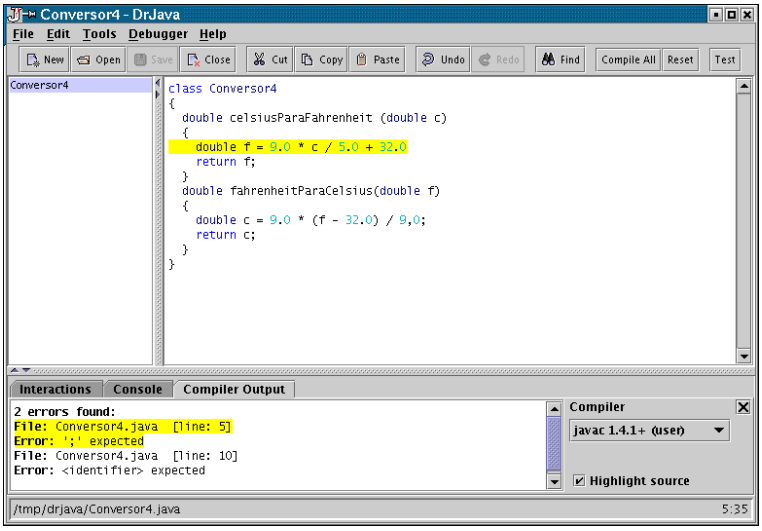
Erros no código

Feche o arquivo Conversor.java usando a opção **Close** do menu **File** e crie um novo arquivo (na verdade, como havia apenas um arquivo aberto, um novo é criado automaticamente).

Digite nesse arquivo o código abaixo (há erros intencionais nele). Você pode também usar o recurso de *copiar* e *colar* (*copy and paste*).

```
class Conversor4
{
  double celsiusParaFahrenheit (double c)
  {
    double f = 9.0 * c / 5.0 + 32.0
    return f;
  }
  double fahrenheitParaCelsius(double f)
  {
    double c = 9.0 * (f - 32.0) / 9,0;
    return c;
  }
}
```

Grave o arquivo e compile. Como o código contém erros, o compilador não terá sucesso e imprimirá algumas mensagens. Algo como mostra a figura abaixo.



O primeiro erro pode ser eliminado acrescentando-se um `;` no final da linha 5. Veja que o próprio compilador sugere isso.

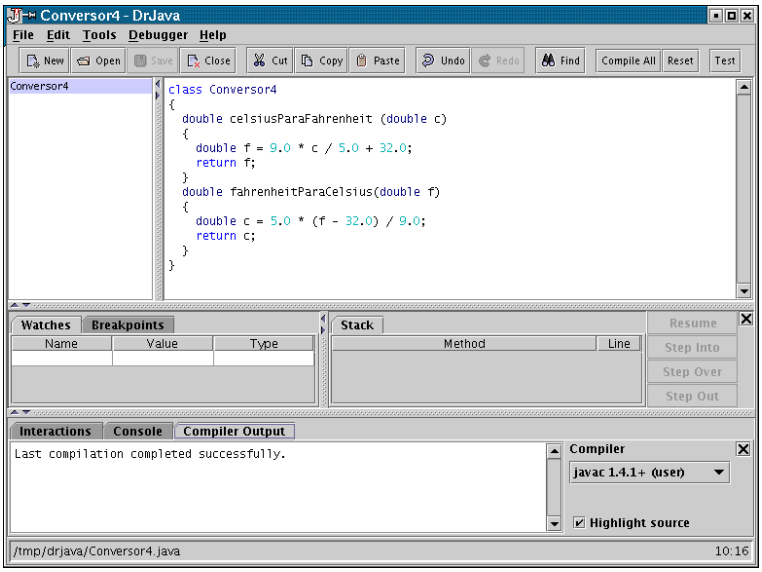
Já a descrição do segundo erro pode ser um pouco confusa. É importante saber que o compilador é capaz de encontrar erros no código, mas nem sempre pode determinar a causa exata. Nesses casos, as mensagens de erro apenas dão pistas para descobrirmos o que está errado (algumas vezes pistas falsas). O segundo erro é uma vírgula no lugar de um ponto, no final da linha 10.

Há outros tipos de erro os quais o compilador não tem condições de detectar. Um exemplo é o erro na fórmula de conversão da linha 10. Há um `9.0` onde deveria estar um `5.0`. No Capítulo 4 apresentamos uma maneira de detectarmos tais erros através de testes. Mas depois de detectarmos, precisamos descobrir a causa deles. Veremos a seguir uma ferramenta útil para essa tarefa. Mas antes corrija os erros e compile.

Depurador

Depurador (*debugger*) é uma ferramenta que nos ajuda a corrigir erros (*bugs*) de programas. O depurador do `DrJava` oferece apenas recursos básicos: pontos de parada (*breakpoints*), execução passo a passo (*step*) e inspeção simples de variáveis (*watch*). Mesmo assim, a classe `Conversor4` não é complexa o suficiente para justificar a aplicação do depurador, vamos utilizá-la apenas para demonstrar rapidamente cada um desses recursos.

Para ativar o depurador, escolha a opção **Debug Mode** do menu **Debugger**. Ao fazer isso, o painel de depuração é exibido na janela principal.



Começaremos selecionando um ponto de parada no código do `Conversor4`. Isso é feito no painel de definições, posicionando o cursor na linha desejada e escolhendo a opção **Toggle Breakpoint on Current Line** do menu **Debugger**.

O ponto de parada faz com que a execução do programa seja temporariamente interrompida exatamente antes da linha ser executada. A partir daí, para continuar a execução, deve-se usar os comandos de "passo a passo"(*step into*, *step over*, *step out*, *resume*). O que cada comando faz pode ser encontrado na seção *Documentation* da página do `DrJava` <http://drjava.sourceforge.net>

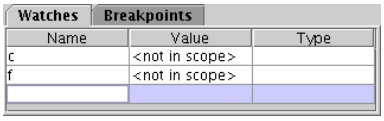
Posicione então o cursor do painel de definições na linha 10 do `Conversor4` e insira um ponto de parada (a linha ficará com um destaque vermelho). Para que o ponto de parada seja usado, temos que fazer com que o método seja executado. Para isso digite o seguinte na janela do interpretador.

```
Conversor4 c4 = new Conversor4();
c4.fahrenheitParaCelsius(42)
```

A execução do método será interrompida antes da linha 10 ser executada. O destaque azul da linha indica isso. Para continuar a execução passo a passo (linha a linha), execute o comando *step over* algumas vezes (pressionando a tecla <F11>). Dessa forma, é possível constatar quais trechos de código são usados numa execução em particular.

Um outro recurso bastante útil, que deve ser usado em conjunto com esses que acabamos ver, é a inspeção de valores de variáveis (*watch*). Esse recurso nos informa o valor de certas variáveis durante a execução passo a passo. Para isso, é necessário preencher a coluna *Name* da tabela da guia *Watches* com os nomes das variáveis que se deseja inspecionar (uma variável em cada linha da tabela). Basta clicar numa célula da coluna *Name*, digitar o nome da variável e pressionar <Enter>.

Faça este procedimento para as variáveis `c` e `f`. Teremos algo como a próxima figura.



Repita a execução passo a passo descrita anteriormente e observe o que ocorre com a tabela.

Dica: o interpretador armazena os últimos comandos digitados na janela. Para acessá-los, pressione as setas para cima e para baixo do teclado. Esse recurso se chama *History* e possui algumas opções úteis no menu **Tools**.