Programação I

Pacotes

Jorge Roberto Trento
Bacharel em Ciências da Computação - UNOESC
Especialização em Ciências da Computação - UFSC
Formação Pedagógica - Formadores de Educação Profissional – UNISUL
Especialização em Ensino Superior – FIE
Especialização em Gestão de Tecnologia da Informação – FIE

Introdução

- Aplicação pequena em Java possui poucas classes. Em contra partida, uma aplicação mais complexa pode apresentar dezenas, até centenas de classes;
- Com o reaproveitamento de classes entre uma aplicação e outra, podem haver conflitos nos nomes das classes, sem mencionar que encontrar uma classe específica num mar de centenas delas não é uma tarefa fácil.

Introdução

- Diante destas situações, programadores podem agregar grupos de classes com tipos relacionados em pacotes;
- Em uma visão prática, pacotes funcionam como bibliotecas: um pacote chamado **grafico** provavelmente contenha diversas classes relacionadas com gráficos; um pacote chamado **matematica** provavelmente contenha classes para cálculos matemáticos diversos.

Para que servem?

- Organizar as classes por afinidade e hierarquia, tornando fácil o trabalho de localização e compartilhamento de código entre programadores.
- Uma classe pertencente a um pacote tem seu nome prefixado pelo nome do pacote; isso quer dizer que uma classe **Retangulo**, por exemplo, pertencente ao pacote **geometria** na verdade se chama **geometria.Retangulo** (é a classe **Retangulo** do pacote **geometria**).
- É muito provável que outro programador já tenha criado uma classe chamada Retangulo, porém se ela pertencer a outro pacote, ambas as classes podem coexistir, visto que seus nomes são prefixados pelo nome de seus pacotes (**geometria.Retangulo** é uma classe, enquanto **formas.Retangulo** é outra).

Organização de classes em pacotes

- A nomenclatura de pacotes geralmente segue alguns padrões. Um deles é que um pacote geralmente é formado por diversos subpacotes, com o objetivo de organizar ainda melhor as classes.
- Imagine o exemplo do pacote geometria: Retangulo é uma forma geométrica, porém um pacote chamado geometria possui diversas formas. Podemos agrupá-las em **formas retas** (triângulos, quadrados, etc) e **formas curvas** (círculos, elipses, etc).
- Dessa forma, nosso pacote geometria poderia ter dois subpacotes: retos e curvos. A hierarquia de pacotes utiliza a notação do ponto para indicar nível, assim como fazemos com classes e membros (ex.: System.out.println()).
- Nossos pacotes seriam, então, **geometria.retos** e **geometria.curvos**.

Organização de classes em pacotes

- Quando uma classe faz parte de um pacote, a primeira linha de código dessa classe deve informar isso.
- O exemplo abaixo mostra a classe Retangulo, do pacote **geometria.retos**:

```
package geometria.retos;
    public class Retangulo {
        // código da classe
    }
```

- A primeira linha do arquivo Retangulo.java é a indicação de qual pacote a classe faz parte. Em relação à declaração do pacote, utilizamos a palavra-chave **package** seguida do nome do pacote.
- No caso do exemplo, a classe **Retangulo** faz parte do sub pacote retos do pacote geometria , por isso o nome do pacote é **geometria.retos**.
- Se a classe fizesse parte apenas do pacote geometria , a primeira linha do código seria **package geometria**.

Utilização de classes de pacotes

- Conforme citado, classes que pertencem a um pacote tem seu nome prefixado pelo nome do pacote. Utilizando-se a classe Retangulo do exemplo anterior, o nome dessa classe não é simplesmente Retangulo , é geometria.retos.Retangulo . Uma classe nomeada simplesmente como Retangulo não pertence a pacote algum.
- Classes que fazem parte de um mesmo pacote não precisam utilizar seu nome completo qualificado quando se referenciarem, visto que fazem parte do mesmo pacote.
- Se uma classe do pacote geometria.retos instanciar a classe Retangulo (também desse pacote), ela pode fazê-lo através de **Retangulo r = new Retangulo()**.
- Se uma classe de outro pacote instanciar a classe Retangulo , porém, ela não pode simplesmente utilizar o nome Retangulo , porque ela estaria se referindo à classe Retangulo sem pacote, porém o desejado é a classe Retangulo do pacote **geometria.retos**.

Usar o nome qualificado completo da classe

- A classe Retangulo do exemplo anterior, por exemplo, faz parte do pacote geometria.retos , logo seu nome qualificado completo é geometria.retos.Retangulo . Esse é o nome que deve ser utilizado para se instanciar a classe, conforme o exemplo abaixo.

```
public class Main {
    public static void main(String args[]) {
        geometria.retos.Retangulo r = new geometria.retos.Retangulo();
        r.metodo1();
        r.metodo2();
    }
}
```

- Essa forma é rápida e eficaz, porém se for utilizada algumas vezes no mesmo código tornará o programa difícil de ser lido e entendido. Para contornar esse problema, pode-se importar uma classe específica de um pacote, assim seu nome completo não é necessário.

Importar uma classe específica de um pacote

- Ao importar uma classe específica de um pacote, pode-se referenciar a classe apenas pelo seu primeiro nome ao invés do nome qualificado completo. A importação de uma classe é feita através do comando import, como no exemplo abaixo.

```
import geometria.retos.Retangulo;
public class Main {
    public static void main(String args[]) {
        Retangulo r = new Retangulo();
        r.metodo1();
        r.metodo2();
    }
}
```

- O comando import indica qual a classe a ser importada. É importante notar que o comando import exige que o nome qualificado completo da classe seja fornecido. Depois que a importação é feita, a classe pode ser utilizada pelo seu primeiro nome, sem necessidade de referenciar seu pacote.
- Quando faz-se necessário importar mais de uma classe de um determinado pacote, o código tente a ter diversos imports (um para cada classe a ser importada). Para evitar esse problema, pode-se importar todas as classes de um determinado pacote.

Importar um pacote inteiro

- Para realizar a importação de todas as classes de um pacote, utiliza-se o comando import, porém informando-se um asterisco (*) no nome da classe (o que indica algo como "todas as classes"). O código abaixo exemplifica a importação de todas as classes do pacote geometria.retos.

```
import geometria.retos.*;

public class Main {
    public static void main(String args[]) {
        Retangulo r = new Retangulo();
        Triangulo t = new Triangulo();
        Quadrado q = new Quadrado();
        r.metodo1();
        r.metodo2();
    }
}
```

Importar um pacote inteiro

- Uma característica muito importante é que o operador asterisco não importa classes pertencentes a subpacotes, ou seja, o comando **import geometria.*** NÃO importa as classes do pacote retos e curvos. Embora o pacote retos, por exemplo, esteja dentro do pacote geometria, eles não possuem relação em termos de código. O pacote retos está dentro do pacote geometria apenas para demonstrar uma hierarquia e ajudar a fazer a organização das classes mais explícita.
- **O comando import geometria.* faz a importação de todas as classes que estão no pacote geometria .** O pacote geometria.retos é um pacote diferente, assim como geometria.cursos . Para importar todas as classes do pacote geometria , o programador deveria fazer o seguinte (assumindo que não existem classes dentro do pacote geometria e que somente retos e curvos são os subpacotes de geometria):

```
import geometria.retos.*;

import geometria.curvos.*;

public class Main {
    public static void main(String args[]) {
        // codigo...
    }
}
```

Organização Física dos Arquivos

- Em termos de organização física de arquivos, um pacote é descrito como uma hierarquia de diretórios. Dessa forma, as classes do pacote geometria.retos estariam dentro da pasta retos , que estaria dentro da pasta geometria.
- Ex:
  - <caminho do projeto>\geometria\retos\ no Windows, e
  - <caminho do projeto>/geometria/retos no Linux.

Organização Física dos Arquivos

- Imaginando-se as classes geometria.retos.Retangulo , geometria.retos.Quadrado e geometria.curvos.Circulo, a seguinte estrutura seria encontrada no disco (imaginando o sistema Windows):
- <caminho do projeto>\geometria\retos\Retangulo.java
- <caminho do projeto>\geometria\retos\Quadrado.java
- <caminho do projeto>\geometria\curvos\Circulo.java

  O local onde residem os diretórios dos pacotes e suas classes chama-se classpath (do inglês "class path", caminho das classes). Em um projeto pequeno ou em um programa de testes, geralmente não se faz uso de pacotes. Com isso, todas as classes são criadas e colocadas na pasta do projeto, como o exemplo abaixo
- <caminho do projeto>\Main.java
- <caminho do projeto>\Classe1.java
- Todas as classes da aplicação estão na pasta do projeto, então o classpath é essa própria pasta. Por essa razão, a compilação das classes é feita sem que nenhum diretório seja informado: <caminho do projeto>\javac *.java

  E após: <caminho do projeto>\java Main

Organização Física dos Arquivos

- O comando anterior funciona corretamente, porque a JVM fará uma busca pela classe Main compilada (Main.class) no diretório corrente (que é <caminho do projeto>) e o arquivo será encontrado.
- Quando pacotes são utilizados, as classes estão em diretórios (e subdiretórios), então o compilador java precisa ser informado disso. Imaginando-se a seguinte estrutura de pastas e arquivos:

  <caminho do projeto>\Main.java

  <caminho projeto>\geometria\retos\Retangulo.java

  <caminho projeto>\geometria\retos\Quadrado.java

  <caminho projeto>\geometria\curvos\Circulo.java

Organização Física dos Arquivos

- nota-se que a classe Main não faz parte de qualquer pacote, porém as demais classes fazem parte de outros pacotes (geometria.retos e geometria.curvos).
- Para compilar todas as classes, é necessário informar ao compilador java os arquivos (e pastas) a serem compilados:

  <caminho>\javac *.java .\geometria\retos*.java .\geometria\curvos*.java

Organização Física dos Arquivos

- Imaginando-se o código da classe Main como o seguinte:

```
import geometria.retos.*;
import geometria.curvos.*;
public class Main {
    public static void main(String args[]) {
        Retangulo t = new Retangulo();
        Circulo c = new Circulo();
    }
}
```

- a execução do programa seria também dada pelo seguinte comando

```
<caminho do projeto>\java Main
```

Outros casos

- Porém, pode ocorrer de termos estruturas diferentes em nossos projetos. Nesse exemplo, todas as classes do pacote foram colocadas em outro local, na pasta “classes”.

```
<caminho do projeto>\Main.java

<caminho do projeto>\classes\geometria\retos\Retangulo.java

<caminho do projeto>\classes\geometria\retos\Quadrado.java

<caminho do projeto>\classes\geometria\curvos\Circulo.java
```

Outros casos

Depois de compilar, os arquivos .class ficaram organizados dessa forma:

```
<caminho do projeto>\Main.java
<caminho do projeto>\Main.class
<caminho do projeto>\classes\geometria\retos\Retangulo.java
<caminho do projeto>\classes\geometria\retos\Retangulo.class
<caminho do projeto>\classes\geometria\retos\Quadrado.java
<caminho do projeto>\classes\geometria\retos\Quadrado.class
<caminho do projeto>\classes\geometria\curvos\Circulo.java
<caminho do projeto>\classes\geometria\curvos\Circulo.class
<caminho do projeto>\classes\geometria\curvos\Elipse.java
<caminho do projeto>\classes\geometria\curvos\Elipse.class
```

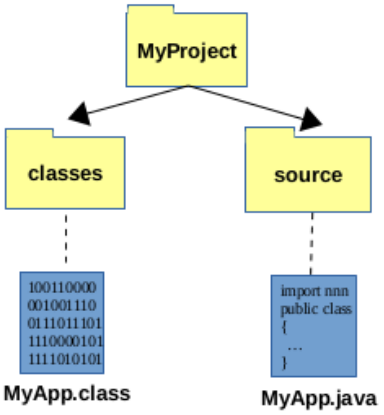
Para que o programa funcione corretamente, é necessário informar a JVM que existem **dois classpaths** diferentes: um deles é o diretório atual e o outro é a pasta classes.

```
<caminho do projeto>\java -cp <classpath> Main
<caminho do projeto>\java -cp .;./classes Main
```

Compilando com a flag -d

- Para facilitar o trabalho de organização de nossos arquivos, além de organizar as classes em pacotes, podemos separar os arquivos-fonte dos arquivos compilados usando a flag -d do compilador javac.
  - $ javac arquivo.java -d caminho

- Vamos separar as classes de nosso projeto em duas pastas: source, para arquivos-fonte e classes para classes compiladas. Teremos a seguinte estrutura:



Compilando com a flag -d

- Coloque todos os seus arquivos-fonte dentro da pasta source.
- No terminal, entre nesta pasta e compile-os, direcionando o resultado para a pasta classes. Note que para chegarmos a classes a partir de source , temos que voltar um nível na estrutura de pastas (.. significa "a pasta acima"). Pode-se também compilar todos os arquivos em um mesmo comando usando *.java .

  $ cd MyProject/source

  $ javac *.java -d ../classes
- O aplicativo compilado estará na pasta classes:

  $ cd MyProject/classes

  $ java MyApp

Arquivos JAR

- Após concluir o desenvolvimento de um aplicativo, temos 3 opções de implantação/distribuição:
  - **Local:** o aplicativo inteiro será executado no computador do usuário final, como um programa autônomo, implantado como um arquivo jar executável;
  - **Combinação de local e remota:** o aplicativo é distribuído com uma parte cliente sendo executada no sistema local do usuário, conectada a um servidor onde outras partes do aplicativo são executadas. Isso pode ser implementado através das tecnologias Java Web Start ou por um aplicativo RMI (Remote Method Invocation);
  - **Remota:** o aplicativo Java inteiro é executado em um sistema servidor, com o cliente acessando o sistema através de algum meio não relacionado à Java, provavelmene um navegador Web. Temos como exemplo o uso de Servlets.

Arquivos JAR

- Examinaremos a seguir a primeira opção: arquivos JAR. Um arquivo JAR é um Java ARchive. Ele se baseia no formato de arquivo pkzip e permitirá que possamos empacotar todas as classes pertencentes a um aplicativo em um arquivo único e executável pela JVM.
- Por executável, queremos dizer que o usuário final não precisará extrair os arquivos das classes antes de executar o programa. Ele poderá executar o aplicativo com os arquivos de classes ainda no formato JAR. Para isso, é necessário incluir no arquivo JAR um arquivo de declaração (manifesto) que informe à JVM qual das classes tem o método main().

Para criar um arquivo JAR, proceda da seguinte forma:

1. Crie um arquivo chamado manifest.txt com o seguinte conteúdo: **Main-Class: MyApp**
2. Pressione ENTER depois de digitar a linha acima, ou seu arquivo pode não funcionar corretamente. Salve-o na pasta onde estão as classes do seu aplicativo.
3. Execute a ferramenta jar para criar um arquivo JAR que contenha suas classes, mais o arquivo manifest.txt:

   $ jar -cmvf manifest.txt MyApp.jar *.class

4. Se as classes de seu aplicativo estão organizadas em pacotes, inclua também os nomes dos mesmos:

   $ jar -cmvf manifest.txt MyApp.jar *.class pack1 pack2

5. Para executar um arquivo JAR:

   $ java -jar MyApp.jar

Ferramentas automáticas

Existem diversas ferramentas que servem para automatizar o processo de **deploy**, que consiste em compilar, gerar documentação, bibliotecas etc. As duas mais famosas são o **ANT** e o **MAVEN**, ambos são projetos do grupo Apache.

O Eclipse pode gerar facilmente um jar, porém, se o seu build é complexo e precisa preparar e copiar uma série de recursos, as ferramentas indicadas acima possuem mais recursos.

Gerando o JAR pelo Eclipse

Escolha um dos projetos para criar o JAR.

1. Clique com o botão direito em cima do nome do seu projeto e selecione a opção Export.
2. Na tela Export, selecione a opção "JAR file" e aperte o botão "Next".
3. Na opção "JAR file:", selecione o local que você deseja salvar o arquivo JAR. E aperte "Next“.
4. Next.
5. Na tela **JAR Manifest specification**, na opção "select the class of the application entry point", você deve escolher qual classe será a classe que vai rodar automaticamente quando você executar o JAR.
6. Entre na linha de comando: **`java –jar nomeescolhido.jar`**

   OBS: É comum dar um nome mais significativo aos JARs, incluindo nome da empresa, do projeto e versão, como doglas-aplicattion-1.0.jar

Pacotes

Autor
Prof. Doglas André Finco
doglas.andref@uffs.edu.br

Jorge.trento@uffs.edu.br