Capítulo 1

# Empacotamento, compilação e interpretação de código Java

## Compreender Pacotes

### Projetando Pacotes

* Pacotes definem onde as classes estarão localizadas na estrutura hierárquica do diretório.
* Empacotamento evita colisão no mesmo espaço de nomes (*namespace*).
* Nome do pacote + nome da classe = *nome de classe totalmente qualificado* (*fully qualifed class name*).
* Empacotamento promove:
* reutilização de código;
* facilidade de manutenção;
* encapsulamento;
* modularidade.

Considerações sobre os atributos dos pacotes:

|  |  |
| --- | --- |
| **Atributo dos Pacotes** | **Benefícios** |
| Acoplamento de Classes | As dependências dos pacotes são reduzidas com o acoplamento de classes |
| Acoplamento de Sistema | As dependências dos pacotes são reduzidas com o acoplamento do sistema |
| Tamanho do Pacote | Pacotes maiores facilitam a reutilização e pacotes menores facilitam a manutenção |
| Capacidade de Manutenção | Alterações frequentemente restritas a um único pacote quando este contém funcionalidades específicas e relacionadas |
| Nomenclatura | Utilizar convenções. Usar nome de domínio invertido para a estrutura do pacote. Usar caracteres minúsculos delimitados por sublinhados para separar palavras em nomes de pacotes |

### A Instrução *package*

Instruções *package*:

* São opcionais;
* Somente uma por arquivo-fonte;
* Nome de domínio invertido: *com.ocajexam.utils*;
* Nomes de pacotes são estruturas de diretório: *com.ocajexam.utils = com/ocajexam/utils*;
* Nomes de pacote que comecem com *java.\** e *javax.\** são reservados;
* Grafia em letra minúscula, palavras individuais que componham o nome do pacote devem ser separadas por sublinhados.

|  |  |
| --- | --- |
| **Instrução** | **Diretório** |
| package java.net; | [caminho]\java\net\ |
| package com.ocajexam.utils; | [caminho]\com\ocajexam\utils\ |
| package nome\_pacote; | [caminho]\nome\_pacote\ |

### Instrução *import*

* entre *package* (opcional) e antes da definição da classe;
* um \**import*\* por pacote;
* recomendado importar explicitamente.

|  |  |
| --- | --- |
| **import** | **Definição** |
| import java.net.\*; | Importa todas as classes do pacote. |
| import java.net.URL; | Importa somente a classe *URL*. |
| import static java.awt.Color.\*; | Importa todos os membros estáticos da classe *Color* |
| import static java.awt.Color.ColorSpace.CS\_GRAY; | Importa o membro estático *CS\_GRAY* |

|Cenário|Solução|

|-------|-------|

|Qual pacote para gerar gráficos e imagens?|import java.awt.\*;|

|Qual pacote para fluxo de dados?|import java.io.\*;|

|Qual pacote para um app de rede?|import java.net.\*;|

|Qual pacote para framework de coleções, com o modelo de eventos e com recursos de data/hora?|import java.util.\*;|

|Qual pacote para interfaces básicas de Java?|import java.lang.\*; (Importado por padrão)|

\* \*import static\*: permite a importação de membros estáticos

#### Compreender Classes Derivadas de Pacotes

##### API Java de Utilitários

\* API de Utilitários -> \*java.util\*

Java Collecions Framework:

|Interface|Implementação|Descrição|

|---------|-------------|---------|

|List|ArrayList, LinkedList, Vector| Estruturas de dados baseadas em acesso posicional|

|Map|HashMap, Hashtable, LinkedHashMap, TreeMap|Estruturas de dados que mapeiam chaves para valores|

|Set|HashSet, LinkedHashSet, TreeSet| Estruturas de dados baseadas na exclusividade de elementos|

Queue|PriorityQueue|Normalmente FIFO. As filas de prioridade ordenam os elementos usando comparador fornecido|

\* \*Comparator\*: Classifica objetos por sua classe natural

\* Recursos legados de data e hora:

\* \*java.util.Date\*

\* \*java.util.Calendar\*

\* \*java.util.TimeZone\*

\* \*Locale\*: regiões geográficas

\* \*Currency\*: moedas -> ISO 4217

\* \*Random\*: gerador de números aleatórios

\* \*StringTokenizer\*: divide string em tokens

\* \*Timer\*: agendamento de tarefas

##### API Java de Entrada/Saída

![Reader e Writer](https://github.com/alvarengaricardo/Java/blob/main/Certificacao\_1Z0-808/StudyGuide/src/01/fig-1-3.jpeg?raw=true)

\* \*InputStream\* e \*OutputStream\*: Fluxo de bytes

\* \*Reader\* e \*Writer\*: Fluxos de caracteres

\* \*File\*: Representação de nomes de caminho de arquivos e diretórios

\* \*FileDescriptor\*: handle para abertura de arquivos e sockets

\* \*FilenameFilter\*: Filtragem por nome de arquivo

\* \*RandomAccesFile\*: Permite ler e gravar arquivos

##### API Java de Rede

\* Pacote \*java.net\*

\* Funcionalidades para aplicativos de rede

![java.net](https://github.com/alvarengaricardo/Java/blob/main/Certificacao\_1Z0-808/StudyGuide/src/01/fig-1-4.jpeg?raw=true)

##### API Java do Abstract Window Toolkit

\* Pacote \*java.awt\*

\* Geração de componentes pesados para a criação de interfaces de usuário e exibição de elementos gráficos e imagens associados

\* \*java.awt\*:

\* API de componentes pesados do AWT

\* Subsistema Focus

##### API Java Swing

\* Pacote \*javax.swing\*

\* Criação de contêineres e componentes leves (puramente java)

# Estudar melhor com outras fontes, verificar DEITEL

##### API do Java FX

# Estudar melhor com outras fontes, verificar DEITEL

![javax.swing](https://github.com/alvarengaricardo/Java/blob/main/Certificacao\_1Z0-808/StudyGuide/src/01/fig-1-6.jpeg?raw=true)

|Cenário|Solução (Importação)|

|-------|-------|

|Criar componentes Java básicos com Swing(botões, painéis e caixas de diálogo)|\*import javax.swing.\*\*;|

|Dar suporte a aspectos relacionados a texto de componentes Swing|\*import javax.swing.text.\*\*;|

|Implementação e suporte básico plugável a mais de um \*look-and-feel\*|\*import javax.swing.plaf.\*\*;|

|Usar adaptadores e receptores de eventos Swing (listeners)|\*import javax.swing.event.\*\*;|

\*

~~~java

static void metodoComVarargs(String a, boolean b, int... c) {

System.out.println(a);

System.out.println(b);

System.out.println(c.length);

for (int d : c)

System.out.println(d);

System.out.println();

}

~~~