Pacotes

Palavra-chave import

Um pacote é uma coleção de classes e interfaces Java, agrupadas

- *O pacote faz parte do nome da classe: Uma classe NovaYork do pacote simulador.cidades pode ser usada por outra classe (de pacote diferente) apenas se for usado o nome completo simulador.cidades.NovaYork*
- *Toda classe pertence a um pacote: Se a classe não tiver declaração package, ela pertence ao pacote default, que, durante a execução, corresponde à raiz do Classpath.*

import, pode ser usado, para compartilhar os espaços de nomes de pacotes diferentes (assim, não será preciso usar nomes completos)

Este módulo explora pacotes em Java, iniciando pelos pacotes da própria API Java e terminando por mostrar como construir pacotes e guardá-los em um arquivo JAR

Assuntos

- *API do Java 2 - visão geral dos principais pacotes*
- *Classes do java.lang: visão geral das principais classes*
- *Métodos de java.lang.Object que devem ser implementados*
- *Pacotes em Java: como criar e como usar*
- *Arquivos JAR para bibliotecas e executáveis*

A API do Java 2 consiste de classes distribuídas e organizadas em pacotes e subpacotes

Pacotes básicos

- java.lang: classes fundamentais - importado automaticamente
- java.util: classes utilitárias
- java.io: classes para entrada e saída
- java.net: classes para uso em rede (TCP/IP)
- java.sql: classes para acesso via JDBC
- java.awt: interface gráfica universal nativa
- java.text: internacionalização, transformação e formatação de texto

Pacote java.lang

É importante conhecer bem as classes deste pacote

Interfaces

- *Cloneable*
- *Runnable*

Classes

- *Boolean, Number (e subclasses), Character, Void*
- *Class*
- *Math*
- *Object*
- *Process, Thread, System e Runtime*
- *String e StringBuffer*
- *Throwable e Exception (e subclasses)*

java.lang.Object

Raiz da hierarquia de classes da API Java

Toda classe estende Object, direta ou indiretamente

- *Classes que não declaram estender ninguém, estendem Object diretamente*

```
class Ponto {}
```

é o mesmo que

```
class Ponto extends Object {}
```

- *Classes que declaram estender outra classe, herdam de Object pela outra classe cuja hierarquia começa em Object*

Todos os métodos de Object estão automaticamente disponíveis para qualquer objeto

- *Porém, as implementações são default, e geralmente inúteis para objetos específicos*

Estendendo Object

Há vários métodos em Object que devem ser sobrepostos pelas subclasses

- *A subclasse que você está estendendo talvez já tenha sobreposto esses métodos mas, alguns deles, talvez precisem ser redefinidos para que sua classe possa ser usada de forma correta*

Métodos que devem ser sobrepostos

- `boolean equals(Object o)`: *Defina o critério de igualdade para seu objeto*
- `int hashCode()`: *Para que seu objeto possa ser localizado em Hashtables*
- `String toString()`: *Sobreponha com informações específicas do seu objeto*

equals()

Determine quais os critérios (que propriedades do objeto) que podem ser usados para dizer que um objeto é igual a outro

- *O raio, em um objeto Círculo*
- *O número de série, em um objeto genérico*
- *O nome, sobrenome e departamento, para um empregado*
- *A chave primária, para um objeto de negócio*

Implemente o equals(), testando essas condições e retornando true apenas se forem verdadeiras (false, caso contrário)

- *Verifique que a assinatura seja igual à definida em Object*

java.lang.Math

A classe Math é uma classe final (não pode ser estendida) com construtor private (não permite a criação de objetos)

Serve como repositório de funções e constantes matemáticas

Para usar, chame a constante ou função precedida do nome da classe:

```
double distancia = Math.sin(0.566);
int sorte = (int) (Math.random() * 1000);
double area = 4 * Math.PI * raio;
```

Consulte a documentação sobre a classe Math e explore as funções disponíveis

Acesso ao Sistema

Classe System dá acesso a objetos do sistema operacional

- `System.out` - *saída padrão (java.io.PrintStream)*
- `System.err` - *saída padrão de erro (java.io.PrintStream)*
- `System.in` - *entrada padrão (java.io.InputStream)*

Runtime e Process permitem controlar processos externos (processos do S.O.)

```
Runtime r = System.getRuntime();
Process p =
  r.exec("c:\program~1\micros~1\msie.exe");
```

java.util.Date

- *Não representa uma data de calendário (para isto existe java.util.Calendar - pesquise da documentação!)*
- *Não representa data ou hora do calendário ocidental (para isto existe java.util.GregorianCalendar)*

- *Use Date para obter o momento atual*

```
Date agora = new Date();
```

- *Ou para criar momentos com base na contagem de milissegundos*

```
Date ontem =
     new Date(agora.getTime() - 86400000);
Date intervalo =
     agora.getTime() - inicio.getTime();
```