|  |  |  |
| --- | --- | --- |
| **Recurso** | **Comando** | **Objetivo** |
| Construtor | Colocar o mesmo nome da Classe. Não colocar void | Inicializar |
| Encapsulamento | private | Proteger |
| Herança | extends | Reaproveitar |
| Polimorfismo | Colocar atributos e métodos com o mesmo nome | Generalizar |
| Classe Abstrata | abstract | Padronizar |
| Interface | Implements | Utilizar heranças multiplas |

Construtores

-Metodos chamados automaticamente no momento da criação do objeto (na instância).

No java: no ‘’new’’, ou seja, na hora que o objeto é instanciado, se tiver um construtor ali ele dispara.

- Não possuem valor de retorno (return);

- Oportunidade de inicializar dados

- Tem sempre o nome da classe

Construtores: Usando

Formato geral:

Na classe ->

**public** Classe ([parâmetro]){}

no programa ->

Classe objeto = **new** Classe([parâmetros]);

O construtor ele é chamado no momento da instancia e somente neste momento.

É possível ter sobrecarga de construtores?

Sim, por ser método, podem ter diferentes tipos de parâmetros, possibilitando a sobrecarga

No java, ele inicia os atributos numéricos com 0, os booleanos com false e string com vazio

Encapsulamento

O objetivo é ocultar atributos/métodos da classe  
Portanto, restringe e controla o acesso da classe  
Fornece apenas os métodos necessários para o uso da classe  
Os detalhes da implementação são omitidos  
Conhecidos também como data hiding, information hiding, dissimulação de informações  
Dados e programas isolados como uma capsula ou **caixa preta**

Encapsulamento: Vantagens

Clareza do código  
Minimização de erros  
Facilidade de extensão  
Mas a **Facilidade na manutenção** é a **maior** vantagem  
Por que a classe é conhecida apenas pela interface  
Pode-se mudar internamente sua estrutura sem que o usuário tome conhecimento disto

Encapsulamento: private

Formato geral:

Na classe->

**private** **tipo** atributo

**private** tipo método(){}

private é uma palavra reservada, qualificador, modificador ou “specifiers”  
De forma contrária a public, permite que os atributos e métodos sejam acessíveis somente nos métodos da própria classe  
É o nível mais rígido de encapsulamento.  
Pronome possessivos

-public nosso(publico)  
 -private meu(exclusivo)

private é o nível mais rígido que encapsulamento tem

Para alterar(assign) ou ler(access) o atributo encapsulado pode-se criar dois métodos  
Formato geral:

public void setAtributo (tipo parâmetro)

{this.atributo = parametro;}

Public tipo getAtributo()

{return(this.atributo);}

Outras formas de encapsular são os modificadores :

- protected (Herança)

- default : “package” / “friendly”  
(pacotes)

Herança

Criar novas classes (extender) a partir de classes já existentes sem duplicação de código  
Os atributos e comportamentos da super-classe são herdados na sub-classe  
A nova classe pode

- Acrescentar novas funcionalidades  
- Mudar a funcionalidade redefinindo métodos com o mesmo nome(override)

Possibilita a reutilização do código

|  |  |
| --- | --- |
| Sobrecarga Overloading | Dois ou mais métodos, com o mesmo nome, parâmetros diferentes, dentro de uma mesma classe |
| Override Sobre escrever | Dois métodos, com mesmo nome, mesmos parâmetros (mesma assinatura), numa relação de herança (Um está da classe Pai e o outro na classe Filha) |
|  |  |
|  |  |

Cancelamento é uma prática questionável  
Por definição todas as características de um tipo aplicam-se a seus subtipos

Protected (protegido) – qualificador que permite que atributos e métodos de uma classe sejam visíveis também em suas classes filhas, mantendo o encapsulamento para as demais  
Extends – Define de qual classe se deseja herdar  
Formato geral:

Classe subclasse extends superclasse

Super – Referência a atributos ou métodos da classe pai (super classe)

Polimorfismo

Visto como o conceito mais simples de OOP  
Do grego (polys e morfé) : muitas formas  
Diferentes classes podem definir atributos e métodos como o mesmo nome  
Quando uma operação assume **muitas formas** de implementação, dependendo do tipo do objeto  
Mensagem do mesmo nome em classes diferentes  
Permite escrever rotinas e métodos que operam com qualquer classe

class Casa{  
void abrirTudo(){  
para cada componente (objeto) da casa{  
objeto.abrir();}  
}...  
O método “abrir()” é chamado para cada componente por meio da variável polimórfica “objeto”.

Este processo realizado em tempo de execução é chamado **vinculação dinâmica**

((classe) objeto).mensgem – Faz com que um objeto genérico (declarado com sendo de uma superclasse) se comporte como uma instancia de uma classe especifica(subclasse)

Exemplo: ((Circulo)fp).diametro()

Objeto instanceof Classe – operador que checa qual é a classe do referido objeto (objeto é uma instancia de classe? )

Exemplo: if(fp instanceof Circulo)...

//A pergunta acima: “fp é um circulo?”

Classe Abstrata

São classes que não podem ser instanciadas;  
São classes que servem apenas de base a partir das quais outras se originarão através da herança  
Conhecidas também como Superclasses Abstratas  
Os métodos sem corpo podem ser definidos como abstract

Abstract – Palavra chave que cria uma classe abstrata.

Exemplo: abstract class Pessoa

abstract tipo método();  
Define um método abstrato, ou seja, sem corpo { }.

Exemplo: abstract double perímetro ();

O código abaixo tenta instanciar uma classe abstrata e recebe uma mensagem de erro:  
FiguraPlana fp = new FiguraPlana();  
FiguraPlana is abstract; cannot be instantiated

Interface

A maneira que o java utiliza para implementar a herança múltipla, ou seja, quando:

Uma subclasse herda características de duas hierarquias distintas.

Herança múltipla pode gerar conflitos de nomes de atributos ou métodos

Java, para simplificar, abandona a ideia de herança múltipla, disponibilizando o uso de **interfaces**

Interfaces não contem atributos

Os métodos da interface não contêm nenhuma implementação, ou corpo (comandos)

Pacotes

“Pacote é um conjunto de classes” (Passos, Jocélio)

Biblioteca que contém classes encapsuladas  
São estruturas de diretórios utilizadas para organizar classes  
Possibilita reutilização do software

Applets

|  |  |  |
| --- | --- | --- |
| Cliente | Web | Servidor |
| Applet – browse | Internet | Servlets |

|  |
| --- |
| Midlets |

É um programa Java com recursos gráficos que não é executado com o “java.exe” mas num browser (internet explorer / Netscape / Chrome) ou no “appletviewer.exe”

O applet view vai chamar o HTML e o HTML é o que chama o applet(class).

1. O appletviewer executa um documento .HTML (HyperText Markup Language)
2. Que contém referência a um arquivo .CLASS (um applet)
3. A máquina virtual (JVM) está embutida no navegador

Ciclo do Applet

Há métodos chaves que são chamados automaticamente:

1. public void **init ()** – Chamado na 1 execução do applet
2. public void **start () – Se existir,** chamado após o método **init** ou quando o usuário retorna a página do applet
3. public void **paint () –** Chamado após o método **init** e o método **start**. Chamado também automaticamente quando o applet é repintado, não pode faltar o paint no applet
4. public void **stop**() – Chamado quando o applet para a execução, podendo voltar ao applet ou ir ao destroy.
5. public void **destroy() –** Chamado quando o applet está sendo removido da memória.

Exceções

Erro que ocorre no decorrer do programa, ou seja, em tempo de execução

Além das exceções predefinidas pelo Java, podemos criar as nossas próprias exceções

Com exceções os programas se tornam mais:

- Claros  
- Robustos  
- Tolerantes a falhas

O tratamento de exceções em Java é semelhante àquele utilizado em C++

Throws Exception – Deixa o bloco de sobre aviso que o código seguinte pode gerar uma exceção

Try – Envolve o bloco que pode gerar uma exceção

Catch – Captura e trata uma exceção

Finally – Código sempre executado independentemente de uma exceção ocorrer

throw – lança uma exceção

Formato geral:  
throw new ClassException();

Exemplo:

throw new RuntimeException();

throw new ArrayIndexOutOfBoundException();

Exceção de run time derruba o sistema

O controle de exceção serve justamente para evitar que o programa seja desligado, abortado, e se tornar robusto.

Concorrência

Conhecida também como Multithreading.

Multiprocessamento: computador como mais de um processador

Multitarefa: Processador executa tarefas (programas) diferentes sincronizando seu tempo de processamento. Exemplo: Formatando, baixando arquivo, compilando programa.

**Quando é executado um programa em java é feito uma tarefa**

Thread: Execução simultânea (paralela) de blocos de código (subprogramas) do mesmo programa.

Multithreading: processamento de mais de um thread.

Não é um conceito simples

Vantagem: Métodos concorrentes aumentam a flexibilidade de programação

Aplicações

- Problemas Particulares em Sistemas Operacionais

- Simulação de sistemas físicos reais, múltiplos subsistemas concorrentes:

**- Voo de aeronaves em uma área controlada**

**- Corpo humano**

Sincronização → Mecanismo que controla a ordem de execução das tarefas

Cooperação → Quando a tarefa A precisa aguardar que a tarefa B conclua alguma atividade. Exemplo: produtor x consumidor.

Competição → Tarefa A e B requerem o uso de algum tipo de recurso que não pode ser usado simultaneamente.

Deadlock → Impasse que ocorre quando dois elementos de um processo estão aguardando que o outro responda. Nenhuma das tarefas renuncia ao recurso

Métodos de manipulação de threads

Thread – Classe pacote java.lang

void run () – Método que contem os comandos que definem o que a thread fará

void start () – Método que dispara a thread, chamando o método run ()

void sleep (int miliseconds) – Método que especifica quanto tempo (em milissegundo) a thread deve “dormir”

boolean interrupt () – Método que interrompe a thread

boolean isInterrupted () – Método que retorna se a thread está ou não interrompida

Interface Gráfica AWT

AWT

Abstract Windowing ToolKit

Kit de Ferramentas Abstrato para janelas

Componentes GUI originais

Pacote java.awt

GUI – Graphical User Interface

Componentes associados com as capacidades gráficas da plataforma local

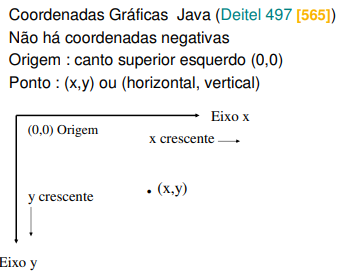
Aparência gráfica e interação com o usuário muda com a plataforma

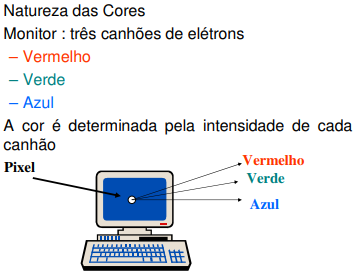
Java oferece muitos recursos gráficos

Applets

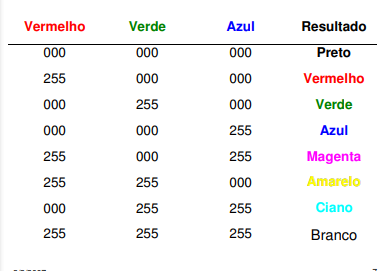
O pacote AWT contém um amplo conjunto de classes e interfaces dedicadas à construção de gráficos em applets e aplicativos Java

Windowing: os recursos gráficos estão sempre “desenhados” em uma janela









A classe Color nos permite manipular cores

Cores pré-definidas (Exemplo: Color.blue)

Color.

-black -blue -cyan -darkGray -gray -green

-lightGray -magenta -orange -pink -red -white

-yellow

Outros sistemas de cores:

CMYK, HSB

A classe graphics

Usada para executar várias funções gráficas

Representa um contexto gráfico

Representação abstrata de uma superfície gráfica(um papel)

public void paint (Graphics G) { }

O objeto Graphics é transmitido automaticamente ao método paint () de um applet e será utilizado nas operações gráficas

Tipos de imagem:

GIF – Graphics Interchange Format

-Formato de troca de gráficos

-Conjunto de imagens

-Eficiente na redução dos arquivos

JPEG/JPG – Joint Photographic Experts Group

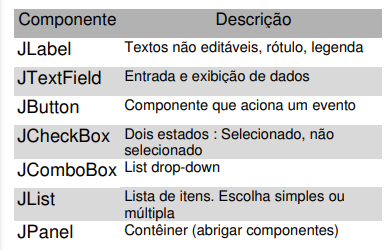
* Grupo reunido de especialistas em fotografia
* Meio altamente eficiente de armazenar imagens fotográficas.

Interface gráfica swing

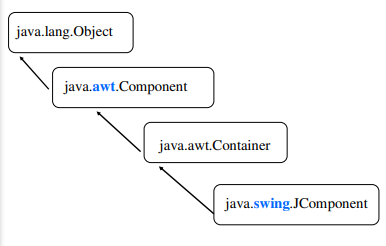
GUI – Graphical User Interface

Pacote javax.swing da plataforma Java 2

São componentes “Java puros”



Ao contrário dos componentes AWT os componentes swing permitem que o aplicativo mantenha a mesma aparência e o comportamento em qualquer plataforma



A classe component define métodos que podem ser aplicados a um objeto de qualquer subclasse de component

Exemplo: paint e repaint

As operações comuns para a maioria dos componentes GUI estão na classe Component

Container é uma coleção de componentes relacionados

Em frames e Applets anexa-se componentes a um painel, ou seja, a um container.

Evento gerado quando o usuário fechar a janla(frame)

O método addWindowListener registra o ouvinte de evento da janela

O argumento é uma referência a um objeto WindowsListener (java.awt.event).

Utilizamos o método windowClosing

Cria-se uma classe interna anônima e o objeto desta classe (new WindowAdapter()) é passado para addWindowListener.

Frame: Janelas

getContentPane() : Método da classe Frame que obtem o container relacionado ao frame;

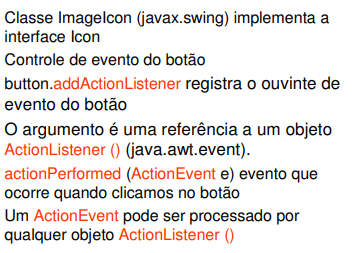
Método da classe container>

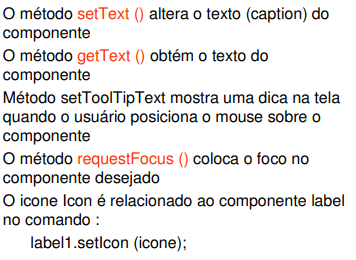
-add adiciona componentes

-SetLayout altera o gerenciador de layout

Classes de layout

- FlowLayout  
-BorderLayout  
-GridLayout





Métodos do Frame

-setSize (width, height) //largura,altura  
-show() //Mostrar o frame  
-dispose() //Fechar o frame

