



FATEC

Faculdade de Tecnologia do Estado de São Paulo

Programação Orientada à Objetos

Apresentação da Disciplina

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Prof. ME. Eng. Comp. Gerson Neto

gersonpenhaneto@gmail.com

gerson.penha@fatec.sp.gov.br

São José dos Campos - SP

Roteiro



Apresentação do Professor

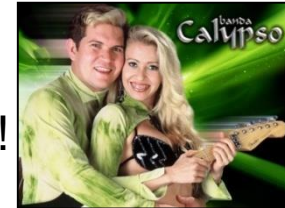


Belém do Pará

google.com



O melhor time do
Brasil, na série B !



google.com

Não, eu não sou Carioca !! Não, eu não sou Baiano !!

Graduado em Engenharia da Computação:

Universidade Federal do Pará - UFPA

Mestre em Computação Aplicada:

Instituto Nacional de Pesquisas Espaciais - INPE

Pesquisador da Microsoft Research – **INPE/FAPESP**

No mercado: **Analista Desenvolvedor**

Empresário: **Ramo de desenvolvimentos de sistemas (ERP) para MEI, ME e EPP.**

Apresentação da Disciplina

Programação Orientada à Objetos

Curso	{	Tecnologia em Análise e Desenvolvimento de Sistemas.
Campi	{	Fatec São José dos Campos
Turno	{	Matutino
Carga Horária	{	40h/Teóricas 40h/Práticas
Laboratórios	{	A definir
Ferramentas	{	Linguagem de Programação Java. IDE Eclipse. Servidor de Aplicação Apache Tomcat.

Apresentação da Disciplina

Objetivos

Apresentar o conceito do paradigma orientado à objeto. Conhecer as principais características de uma linguagem de programação orientada à objetos.

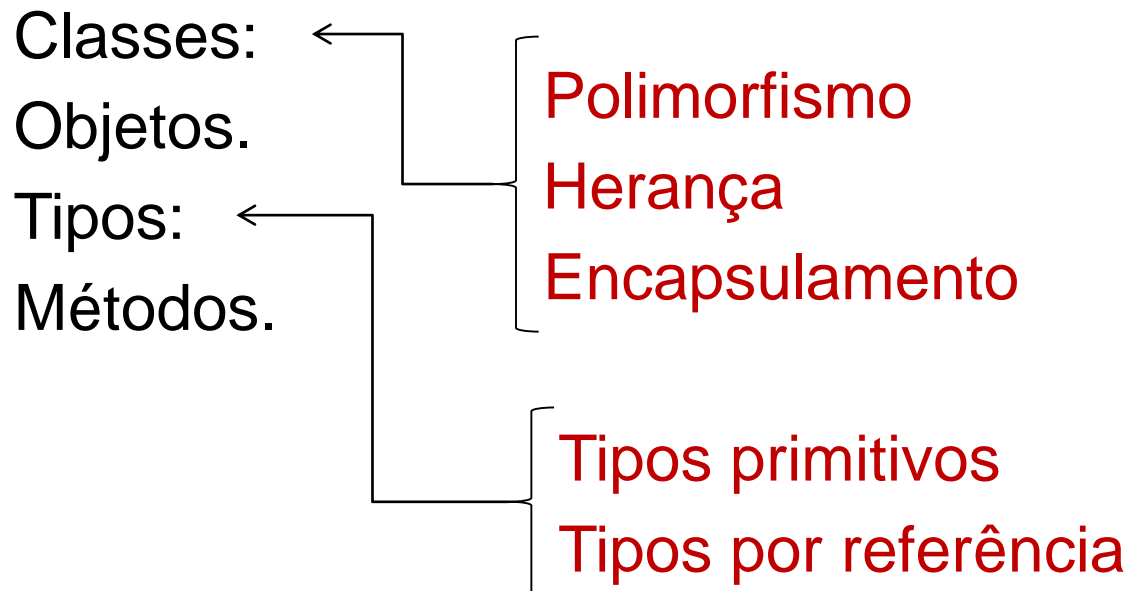
Conhecer a linguagem de programação Java e seus principais atributos relacionados a programação orientada à objetos.

Conhecer o processo de desenvolvimento de software seguindo o padrão de desenvolvimento orientado a objeto. Conhecer a implementação básica de aplicações web através do paradigma orientado à objeto.

Apresentação da Disciplina

Ementa

Paradigma Orientado à Objeto:



Apresentação da Disciplina

Ementa

Linguagem Java:

Classes, interfaces, Membros de Classes, Declaração de Variáveis de Arrays, Convenções de Código, Métodos, Wrapper, AutoBoxing, Coleta de Lixo.

Operadores, Controle de Fluxo, Conversões, Threads, Collections, Generics, Tratamento de Exceções e Arquivos.

Apresentação da Disciplina

Ementa

J2EE:

Java Sever Pages, Expression Language, Tag Library, Erros, Gerenciamento de Sessão.

Servlets, estrutura de diretorios, mapeamento de servlers, ciclo de vida e comunicação entre servlets.

Formas de Avaliação

Avaliações

Prova 1: P1

Prova 2: P2

Atividades e Exercícios: T1 e T2

Nota Final: $P1 \cdot 0.3 + P2 \cdot 0.3 + (T1 + T2) \cdot 0.4$

Nota Final $\geq 6,0$ - Aluno aprovado.

Frequência $< 75\%$ - Aluno reprovado.

Haverá uma prova substitutiva para o aluno que deixar de comparecer na avaliação P1 ou P2. Contudo será somente por ausência justificada e sob análise.

Bibliografia Base

- SIERRA, K. ,BATES, B., SCJP: Certificação Sun para Programador Java 5 – Guia de Estudo (Exame 310-055). Rio de Janeiro, Editora Alta Books, 2006.
- SIERRA, K. ,BATES, B., Use a Cabeça! Java, ALTA BOOKS, 2007.
- DEITEL, H. M., DEITEL, P.J. Java Como Programar, 6a. edição, Porto Alegre: Bookman, 2007.

Bibliografia Complementar

BASHAN, B., SIERRA, K. Use a cabeça! Servlets e JSP. São Paulo: Alta Books, 2005.

KURNIAWAN, B. Java para Web com Servlets, JSP e EJB. Ciência Moderna, 2002.

MCLAUGHLIN, Brett. Java and XML. O'Reilly & Assoc, 2006.

GONÇALVES, Edson. Desenvolvendo Aplicações Web com Servlets, JavaServer Faces, Hibernate, EJB3 Persistence e AJAX. Ciência Moderna, 2007.

Recomendações

Da coordenação:

Evitar barulho nos corredores

Prezar pela limpeza.

Do professor:

Evitar faltas desnecessárias.

Participar das aulas e atividades.

Em caso de dúvida pergunte.

Aula cooperativa.

Revisão antes da prova e correção depois da prova.

Pontualidade na entrega de trabalhos.

Dúvidas?





FATEC

Faculdade de Tecnologia do Estado de São Paulo

Programação Orientada à Objetos

Introdução ao paradigma orientado à objetos

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Prof. ME. Eng. Comp. Gerson Neto

São José dos Campos - SP

O que caracteriza uma Linguagem de Programação

Gramática e significado bem definidos.

Implementável (executável) com eficiência “aceitável”.

Universal: deve ser possível expressar todo problema computável.

Requisitos mínimos obrigatórios:

Estrutura de atribuição. Ex: `int x = 0;`

Estruturas de seleção. Ex: `if(false){...}`

Estruturas de repetição. Ex: `for(...)`

Compilar não é obrigatório, existem linguagens que não são compiladas, são interpretadas. Ex: JavaScript

Características e Aspectos da uma linguagem de programação.

Sintaxe:

A sintaxe refere-se a gramática da linguagem de programação. A forma de como se escreve o código.

```
String string = "String em Java";  
char string[] = "String em C";
```

Diferentes formas de declarar uma String em C e em Java.

Características e Aspectos da uma linguagem de programação.

Semântica:

A semântica refere-se ao significado do bloco de código. A forma de como se escreve o código para que o compilador entenda o que se deseja exatamente.

```
const char *string = "String em C";  
  
char string[] = "String em C";
```

Diferentes formas de declarar uma String em C.

Características e Aspectos da uma linguagem de programação.

Pragmática (Onde se aplica):

Algumas linguagens são mais aplicadas em fins determinados.

A linguagem C é muito usada para a confecção de aplicações científicas.

A linguagem PHP é usada no desenvolvimento de aplicações WEB.

Processadores (Como é executado):

Algumas linguagens de programação são compiladas como o C. Outras são interpretadas como o PHP e Java.

Por que existem tantas linguagens?

Propósitos diferentes:

Dependendo da aplicação a ser desenvolvida algumas linguagens podem se destacar. **Ex: PHP e Java para aplicações WEB.**

Avanços tecnológicos:

A avanço tecnologico pode resultar em melhorias para uma linguagem de programação ou até mesmo em uma nova linguagem. **Ex: VisualBasic e C# para desenvolvimento de aplicações Desktop.**

Por que existem tantas linguagens?

Interesses comerciais:

Algumas linguagens são criadas para dar suporte ao desenvolvimento de aplicações comerciais. Ex: Objective-C para desenvolvimento de aplicações para iPhone.

Cultura e arcabouço científico:

Algumas linguagens são mantidas ao longo do tempo por razões culturais. Ex: Delphi para aplicações desktop. Outras são desenvolvidas para comunidades específicas. Ex: Lua para aplicações geofísicas.

O que é um paradigma de programação?

Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns. Ex: Java, PHP e C# são orientados à objeto.

A classificação de linguagens em paradigmas pode ser uma consequência de decisões de projeto, que impactam radicalmente a forma na qual uma aplicação real é modelada do ponto de vista computacional. Ex: É possível desenvolver uma aplicação (pequena) em Java mas sem usar todos os conceitos de Orientação à objeto.

Orientação a Objeto: Conceito

Orientação a objeto é um conceito dentro do desenvolvimento de software que visa a modelagem de um ambiente real, com estruturas, coleções de objetos, com comportamento próprio.

Entende-se por comportamento próprio que uma estrutura, ou objeto, possui atributos e métodos (funções) que implementam a semelhança desse objeto em software com o seu respectivo modelo real.

Ex:

Atributos que uma pessoa real pode ter: nome, idade, peso e altura. Atributos que um objeto utilizado para modelar uma pessoa: String nome, Int idade, etc...

Linguagem Java: Origem

A linguagem de programação Java evolui a partir da linguagem C++ que já era orientada a objeto.

Java foi desenvolvida para atender a necessidade de desenvolvimento de software voltados para o mercado consumidor a partir da revolução causada pelos microprocessadores.

Em 1991 a Sun Microsystems, financiou um projeto com o codnome Green que resultou em uma linguagem baseada em C++, o criador da linguagem, James Gosling, que inicialmente se chamava Oak. O nome java foi atribuído posteriormente e em 1995 Linguagem foi oficialmente apresentada pela Sun.

Linguagem Java

Java é uma linguagem de programação interpretada e seu funcionamento depende de uma plataforma conhecida como Máquina Virtual Java – JVM (do inglês *Java Virtual Machine*).

Java é considerada multiplataforma pois o desenvolvimento de uma aplicação não depende do sistema operacional usado. Esse conceito é conhecido como: Write once, run anywhere. Foi a principal propaganda da Sun ao lançar a linguagem Java.

ByteCode



JVM



SO

Linguagem Java

Alguns, principalmente os concorrentes afirmam que Java é lento por ser interpretado (MITO). A lentidão não é uma consequência da interpretação.

Java é otimizado através do *JIT (Just inTime Compiler)*. Esse compilador é responsável por transforma o bytecode em código compilado para a plataforma, quando necessário.

Java possui várias versões: Java 1.0 e 1.1 são as versões antigas do Java, que já traziam bibliotecas. Com o Java 1.2 houve um aumento grande no tamanho da API, e foi nesse momento em que trocaram a nomenclatura de Java para Java2, com o objetivo de diminuir a confusão que havia entre Java e Javascript.

Linguagem Java

O que são JVM, JDK e JRE?

JVM: é a java virtual machine, não é possível fazer o download ela sozinha, a jvm sempre vem acompanhada.

JRE: *Java Runtime Environment*, ambiente de execução Java, formado pela JVM e bibliotecas, tudo que se precisa para executar uma aplicação Java. Mas não é suficiente para desenvolvimento de aplicações.


JDK: *Java Development Kit*. Kit necessário para o desenvolvimento de aplicações. Deve-se fazer o download do JDK do Java SE (Standard Edition). Ele é formado pela JRE somado a ferramentas, como o compilador de bytecode javac.

Linguagem Java

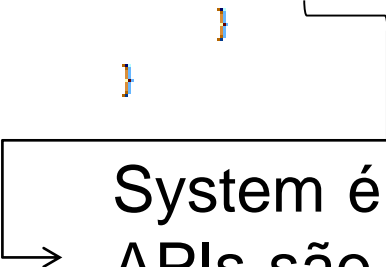
Exemplo de uma classe java:

O método main, é o primeiro método executado em uma aplicação Java. Ele deve existir dentro de uma classe da aplicação e deve ser modificado para o tipo static.

```
public class Primeira {  
    public static void main (String args[]) {  
        System.out.println("Meu primeiro programa em Java");  
    }  
}
```



System é uma API (*Application Programming Interface*).
APIs são as bibliotecas do java. Existe uma API padrão chamada *lang*.



Linguagem Java

Em java também é possível usar a saída formatada.

```
public class Primeira {  
    public static void main (String args[]){  
        /*  
        * Em Java também é possível usar printf  
        */  
        System.out.printf("%s", "Meu primeiro programa em Java");  
    }  
}
```

→ Exemplo de comentários em códigos.

Linguagem Java

Java possui cinco operadores aritméticos e existe uma ordem de precedência para execução.

```
public class Operadores {  
    public static void main (String args[]) {  
        int i = 10;  
        int j = 5;  
        int m = 2;  
        int resultado = (10*5)/2;  
        System.out.println("Valor: /n"+resultado);  
    }  
}
```

Operadores: +, -, *, /, %.

Precedência: *, /, %, +, -.

Linguagem Java

Java possui dois operadores de igualdade e quatro operadores relacionais.

Operadores
de igualdade:
== e !=

Operadores
relacionais: >,
<, >= e <=

```
import java.util.Scanner;

public class Comparar {
    public static void main (String args[]){
        Scanner entrada = new Scanner(System.in);
        System.out.println("Primeiro: ");
        int n1 = entrada.nextInt();
        System.out.println("Segundo: ");
        int n2 = entrada.nextInt();
        if(n1 == n2){
            System.out.println("São iguais");
        }else{
            System.out.println("Não são iguais");
        }
    }
}
```

Linguagem Java

Entende-se por classe a abstração de algum elemento real na forma de software. A classe é um modelo usado para se criar objetos.

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    private int peso;  
  
    public String getNome() {...}  
  
    public void setNome(String nome) {...}  
  
    public int getIdade() {...}  
  
    public void setIdade(int idade) {...}  
  
    public int getPeso() {...}  
  
    public void setPeso(int peso) {...}  
}
```

→ Variáveis de instância.

Métodos get e set.

Linguagem Java

Objetos são sempre tipos por referência.

```
import modelo.Pessoa;  
public class Main {  
    public static void main (String args[]){  
        Pessoa alonso = new Pessoa();  
        Pessoa ariosvaldo = alonso;  
    }  
}
```

Em Java ponteiros não são declarados de forma explícita, mas toda variável que recebe um objeto é um ponteiro, pois ela guarda a referência a esse objeto.

As variáveis alonso e ariosvaldo contém a referência do mesmo objeto.

Dúvidas?



Linguagem Java

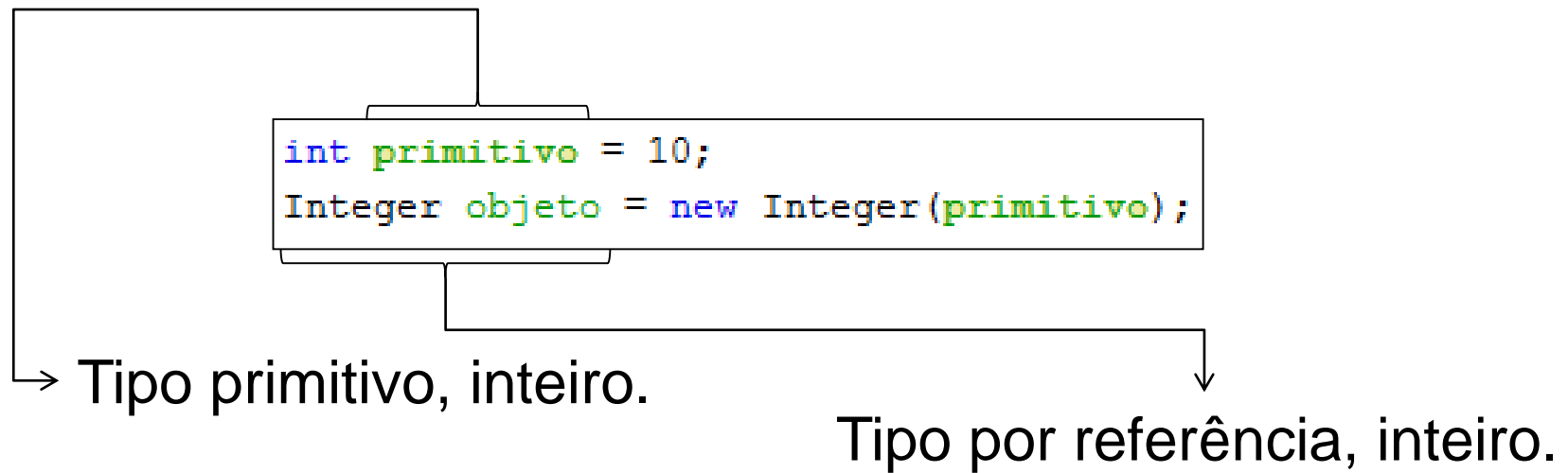
Java possui 8 tipos primitivos.

Tipo primitivo	Tamanho em Bytes
boolean	1bit
short	2
int	4
long	8
float	4
double	8
char	2
byte	1

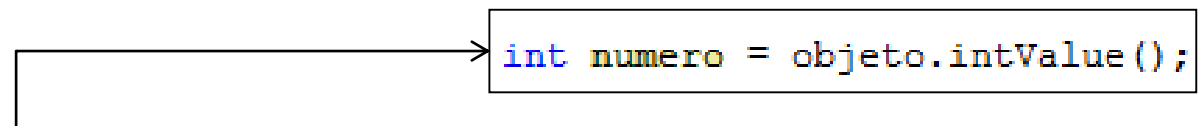
Em Java, tipos primitivos possuem classes associadas. Desse forma é possível converter um tipo primitivo para um objeto, isso é conhecido como empacotamento.

Linguagem Java

Empacotamento(*wrapping*) é utilizado quando se deseja manipular um tipo primitivo como um objeto.



A principal diferença em se usar um tipo por referência no lugar de um tipo primitivo é a capacidade de se ter métodos.



```
int numero = objeto.intValue();
```

Até antes da versão java 5.

Linguagem Java

A partir da versão Java 5, foi atribuída uma nova funcionalidade na linguagem Java para se trabalhar com empacotamento. Ela é chamada de *autoboxing*.

```
int primitivo = 10;  
Integer objeto = primitivo;  
  
int numero = objeto;
```

→ Tipo por referência sendo colocado em um tipo por valor diretamente.

→ Tipo por valor sendo colocado em um tipo por referência diretamente, sem o uso de métodos construtores.

Linguagem Java

Declaração de matrizes em Java:

```
int[] A;  
int[] B = new int[21];  
int[] C = {1,2,3,4,5,6};
```

Em Java é possível se ter matrizes de tipos por referência e tipos primitivos.

Uma matriz em Java pode guardar referências a vários objetos do mesmo tipo. Por esse motivo uma matriz também é um objeto.

É possível preencher a matriz com seus objetos ou valores primitivos no momento da declaração.

```
Pessoa[] P1;  
Pessoa[] P2 = new Pessoa[10];  
Pessoa[] P3 = {new Pessoa(), new Pessoa()};
```

Linguagem Java

Declaração de matrizes multidimensionais em Java:

```
Pessoa[][] A;  
Pessoa[][] B = new Pessoa[2][2];  
Pessoa[][] C = new Pessoa[2][];  
C[0] = new Pessoa[2];  
C[1] = new Pessoa[5];  
Pessoa[][][] D;
```

Uma matriz em Java pode ter quantas dimensões forem necessárias, a quantidade de dimensões é definida pela quantidade de [].

→ Quantidade de dimensões.

Não existe declaração de ponteiros em Java, como na linguagem C. Mas matrizes em Java também são ponteiros, portanto uma matriz multidimensional é do tipo ponteiro para ponteiro.

Linguagem Java

Iterando sobre uma matriz utilizando a estrutura de repetição for:

```
Pessoa[] pessoas = {new Pessoa(), new Pessoa()};  
for(int i = 0; i < pessoas.length; i++){  
    System.out.println(pessoas[i].getNome());  
}
```

Método getNome() utilizado para se recuperar o valor da variável de instância nome.

```
Pessoa[] pessoas = {new Pessoa(), new Pessoa()};  
for(Pessoa pessoa : pessoas){  
    System.out.println(pessoa.getNome());  
}
```

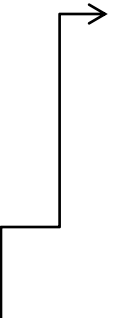
Iteração utilizando a estrutura for aprimorada, também conhecida como for-Each Loop.

Linguagem Java

Java possui quatro modificadores de acesso: public, private, protected e default.

Modificadores de acesso servem para definir de que modo deve-se acessar um atributo(variáveis de instância) de um objetos.

Por padrão todos os atributos de um objeto são do tipo default. O modificador de acesso default acontece quando não se usa nenhuma declaração de modificador antes da variável de instância.



```
public class Pessoa {  
    String nome;  
    int idade;  
}
```

A visibilidade provocada pelo modificador default é conhecida como packet-private (privado ao pacote).

Linguagem Java

O modificador `public` permite que uma variável de instância seja acessada por qualquer outra classe de qualquer outro lugar.

```
public class Pessoa {  
    public String nome;  
    public int idade;  
}
```

O modificador `private` permite que uma variável de instância seja acessada somente pela própria classe a qual pertence. Dessa forma, se for necessário ter acesso a uma variável `private`, um método deve ser disponibilizado.

```
private String nome;  
private int idade;
```

```
public int getIdade() {  
    return idade;  
}  
  
public void setIdade(int idade) {  
    this.idade = idade;  
}
```

```
public String getNome() {  
    return nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}
```

Linguagem Java

O modificador `protected` permite que uma variável de instância seja acessada somente por classes que estejam no mesmo pacote da classe a qual pertence, semelhante ao modificador `default`, e também a subclasses, mesmo que estejam em pacotes diferentes.

```
public class Pessoa {  
    protected String nome;  
    protected int idade;  
}
```

A classe `Pessoa` e a classe `Aluno` estão em pacotes diferentes.

```
package etep;  
import aula.Pessoa;  
public class Aluno extends Pessoa{  
}
```

```
Aluno aluno = new Aluno();  
aluno.nome = "Ariosvaldo";
```

O atributo `nome` fica visível a partir de um objeto `aluno`, para classes no mesmo pacote que `pessoa`.

Linguagem Java

Controlar a forma de acesso a uma variável de instância define o conceito de encapsulamento.

Variáveis de instância podem ser do tipo primitivo ou por referência. Em geral os métodos que permitem o acesso a essas variáveis são conhecidos por sets e gets.

```
private String nome;  
private int idade;
```

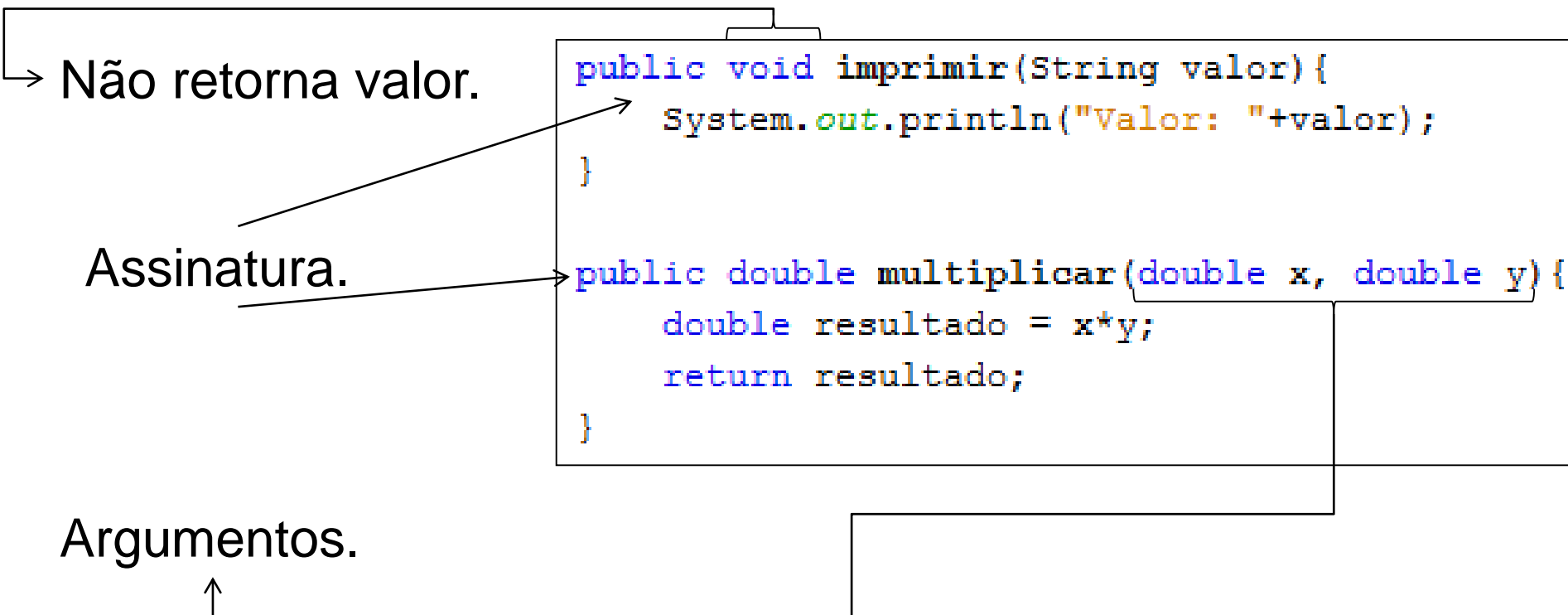
Métodos sets e gets são uma convenção entre os desenvolvedores Java, que mais tarde passarem a ser obrigatórios em alguns frameworks, como o hibernate e JSF.

```
public int getIdade() {  
    return idade;  
}  
  
public void setIdade(int idade) {  
    this.idade = idade;  
}
```

```
public String getNome() {  
    return nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}
```

Linguagem Java

Todo método possui uma assinatura. A assinatura define o nome do método, tipo de retorno, se necessário, e ainda os argumentos usados no método.



Linguagem Java

Existe um tipo de método utilizado quando se cria um objeto. Este método é chamado de método construtor.

O método construtor tem, em sua assinatura, o mesmo nome da classe e também pode ter argumentos, mas não retorna valor.

```
public class Pessoa {  
    String nome;  
    int idade;  
    public Pessoa(String nome){  
        this.nome = nome;  
    }  
  
    public Pessoa(String nome, int idade){  
        this.nome = nome;  
        this.idade = idade;  
    }  
}
```

É possível sobrecarregar métodos.

Várias implementações.

Argumentos diferentes.

Linguagem Java

Sobrecarregar métodos, significa declarar mais de um método com o mesmo nome, porém com a quantidade ou tipo de argumentos diferentes.

```
public int somar(int x, int z){  
    int soma = x+z;  
    return soma;  
}  
  
public double somar(double x, double y){  
    double soma = x+y;  
    return soma;  
}
```

A máquina virtual Java define qual método será usado de acordo com a quantidade e/ou tipo de argumentos usados na assinatura do método. O tipo de retorno deve ser o mesmo para todos os métodos sobrecarregados.

Linguagem Java

O acesso a um método também pode ser definido por um modificador de acesso.

As mesmas regras usadas para variáveis de instância também se aplicam aos métodos assim como os mesmos modificadores.

Contudo public, private, protected e default não são os únicos modificadores que existem. É possível definir outras formas de acesso a métodos e variáveis de instância utilizando os modificadores static e final.

Para o caso de métodos o modificador final impede que o método seja sobreescrito por subclasses. E para o caso de uso em variáveis de instância ele é usado para definir constantes.

Linguagem Java

→ Constante em Java são definidas através do modificador final. Em uma constante só se pode atribuir valor uma única vez.

```
final String nome = "Asdrubal";
```

→ O modificador static mantém o valor de uma variável de instância igual para todos os objetos da classe.

```
static String valor = "Esse valor será igual para todos";
```


Linguagem Java

O modificador `static` também pode ser usado em métodos. Neste caso ele provoca um comportamento diferenciado onde o método em questão pode ser usado sem a necessidade de se criar um objeto da classe.

```
public static void main(String[] args){  
    double valor = Math.pow(2, 4);  
    System.out.println("Valor: "+valor);  
}
```

O acesso a métodos estáticos acontece a partir do nome da classe. Por isso os métodos da classe `Math` são usados sem se criar objetos do tipo `Math`. Também é por esse motivo que o método `main` tem que ser estático.

Dúvidas?



Linguagem Java

Uma classe é uma abstração de algum objeto, é um modelo que deve ser seguido para criar objetos. Porém existem situações em que se deseja criar modelos para classes. Isso é possível através do conceito de classe abstrata.

```
public abstract class Pessoa {  
    protected String nome;  
    protected int peso;  
    protected int idade;  
}
```

Para se definir uma classe como sendo abstrata, na linguagem Java, deve-se na declaração da classe colocar a palavra reservada `abstract`.

Classes abstratas são modelos, para outras classes, portanto não podem ser instanciadas, ou seja, não se pode criar um objeto de uma classe abstrata.

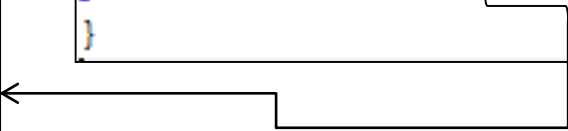
Linguagem Java

A aplicação de uma classe abstrata acontece quando alguma outra classe, herda da classe abstrada.

Entende-se por herança, em orientação a objeto, quando uma classe herda, as funcionalidades de outra classe. A classe usada como modelo é denominada de super classe, e a classe que herda as funcionalidades é denominada de subclasse.

```
public abstract class Pessoa {  
    protected String nome;  
    protected int peso;  
    protected int idade;  
}
```

```
public class Aluno extends Pessoa{  
}
```



```
graph LR; Aluno --> Pessoa;
```

Em java, usa-se a palavra reservada `extends`, para definir uma herança.

Linguagem Java

Uma super classe, pode usar usadas por mais de uma subclasse.

```
public abstract class Pessoa {  
    protected String nome;  
    protected int peso;  
    protected int idade;  
}
```

```
public class Aluno extends Pessoa{  
}
```

```
public class Professor extends Pessoa{  
}
```

Também é possível se fazer heranças múltiplas, ou seja, uma classe pode herdar de uma classe que também herdou de uma outra classe.

Herança é importante não somente pela abstração semelhante ao mundo real, mas também sobre a questão de reaproveitamento de código.

Linguagem Java

Através da herança também se chega ao conceito de polimorfismo. Polimorfismo significa várias formas, ou no caso da programação orientada a objetos, significa que você pode usar o comportamento de uma classe em várias formas

Fazendo uma analogia com o mundo real, pode-se exemplificar o conceito de polimorfismo, comparando o funcionamento de carros. Todo carro pode ser dirigido, guiado, mas seu funcionamento interno, ou seja, como o motor ou peças funcionam, não é igual para todos os tipos de carro.

```
Pessoa pessoa = new Aluno();
```


```
Pessoa pessoa = new Professor();
```

Linguagem Java

Uma das formas de se implementar o polimorfismo em Java é através do conceito de interfaces. Interfaces não são classes, interfaces são contratos ou ainda especificações que uma determinada classe precisa ter.

Usa-se interface quando é necessário definir um tipo de comportamento ou especificação para uma classe.

Uma interface é declarada de maneira semelhante a uma classe, mas com a palavra reservada interface.



```
public interface Coordenador {  
    public void corrigirNotas();  
}
```

Linguagem Java

Em herança, diz-se que uma classe herda de outra classe, no polimorfismo realizado através de interface, diz-se que uma classe implementa uma interface, ou seja, a classe implementa o comportamento da interface.

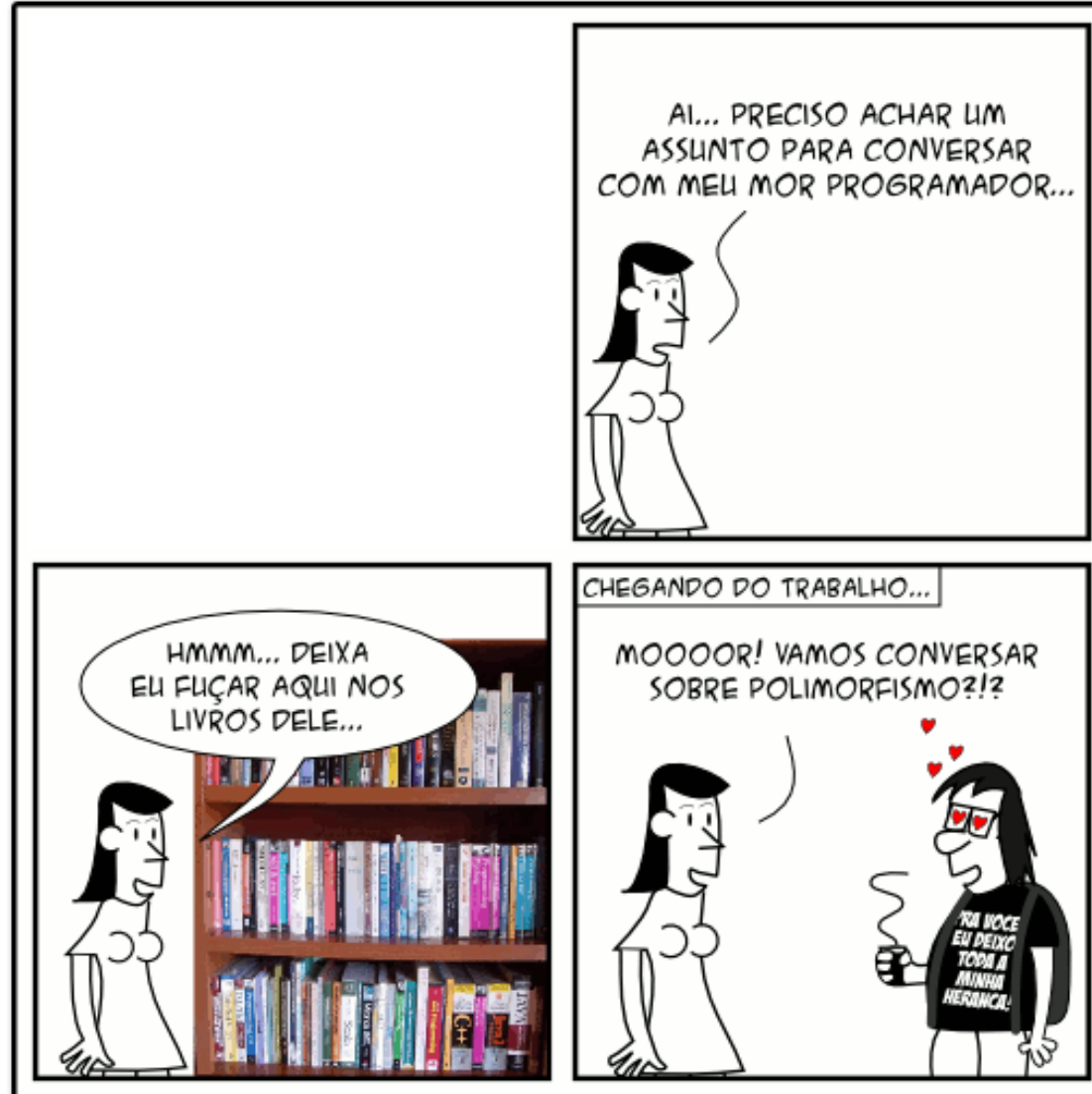
Usa-se interface quando é necessário definir um tipo de comportamento ou especificação para uma classe.

```
public interface Coordenador {  
    public void corrigirNotas();  
}
```

```
public class Professor implements Coordenador{  
    @Override  
    public void corrigirNotas() {  
        System.out.println("O coordenador "  
            + "pode corrigir notas");  
    }  
}
```

A classe professor, teve que implementar o comportamento da interface coordenador.

Perguntas?



Linguagem Java

Aplicação simples para agenda de contatos:

```
public class Pessoa {  
    private String nome;  
    private String telefone;  
    private String email;  
  
    public Pessoa(String nome, String telefone, String email){  
        this.nome = nome;  
        this.telefone = telefone;  
        this.email = email;  
    }  
  
    public Pessoa(){}  
  
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getTelefone() {  
        return telefone;  
    }  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
    public String getEmail() {  
        return email;  
    }  
    public void setEmail(String email) {  
        this.email = email;  
    }  
}
```

Linguagem Java

Aplicação simples para agenda de contatos:

```
public class AgendaTelefonica {
    private Pessoa[] pessoas;
    private Scanner scanner;

    public AgendaTelefonica(){
        pessoas = new Pessoa[5];
        scanner = new Scanner(System.in);
    }

    public void cadastrarPessoa(){
        for(int i = 0; i < pessoas.length; i++){
            if(pessoas[i] == null){
                System.out.println("\nInsira um nome para a pessoa: ");
                String nome = scanner.next();
                System.out.println("\nInsira um telefone para a pessoa: ");
                String telefone = scanner.next();
                System.out.println("\nInsira um email para a pessoa: ");
                String email = scanner.next();
                Pessoa nova = new Pessoa(nome, telefone, email);
                pessoas[i] = nova;
                System.out.println("\nNovo contato adicionado\n");
                break;
            }
        }
    }
}
```

Linguagem Java

Aplicação simples para agenda de contatos:

```
public void imprimirOpcoes(){
    System.out.println("\nEscoha uma das opções: \n");
    System.out.println("1 - Listar contatos. \n");
    System.out.println("2 - Incluir contatos. \n");
    System.out.println("3 - Sair \n");
}

public void imprimirPessoas(){
    for(Pessoa pessoa: pessoas){
        if(pessoa != null){
            System.out.println("\nNome: "+pessoa.getNome()+"\nTelefone: "
                               +pessoa.getTelefone()+"\nEmail: "+pessoa.getEmail());
        }
    }
}

public Pessoa[] getPessoas() {
    return pessoas;
}

public void setPessoas(Pessoa[] pessoas) {
    this.pessoas = pessoas;
}
}
```

Linguagem Java

Aplicação simples para agenda de contatos:

```
public class Aplicacao {  
    private static Scanner scanner;  
    public static void main(String args[]){  
        scanner = new Scanner(System.in);  
        AgendaTelefonica agenda = new AgendaTelefonica();  
        int sair = 0;  
        int opcao = 0;  
        while(sair == 0){  
            switch(opcao){  
                case 0:{  
                    agenda.imprimirOpcoes();  
                    opcao = scanner.nextInt();  
                    break;  
                }  
                case 1:{  
                    agenda.imprimirPessoas();  
                    opcao = 0;  
                    break;  
                }  
                case 2:{  
                    agenda.cadastrarPessoa();  
                    opcao = 0;  
                    break;  
                }  
            }  
        }  
    }  
}
```

```
                case 3:{  
                    sair = 1;  
                    System.out.println("\nAté mais!\n");  
                    break;  
                }  
                default:{  
                    agenda.imprimirOpcoes();  
                    opcao = scanner.nextInt();  
                    break;  
                }  
            }  
        }  
    }  
}
```

Linguagem Java

Mesma aplicação agora com usando Vector<T>:

```
java.util
```

Class Vector<E>

```
java.lang.Object
```

```
    java.util.AbstractCollection<E>
```

```
        java.util.AbstractList<E>
```

```
            java.util.Vector<E>
```

All Implemented Interfaces:

```
    Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess
```

Direct Known Subclasses:

```
    Stack
```

```
public class Vector<E>
```

```
    extends AbstractList<E>
```

```
    implements List<E>, RandomAccess, Cloneable, Serializable
```

Linguagem Java

Mesma aplicação agora com usando Vector<T>:

```
public class AgendaTelefonica {  
    private Vector<Pessoa> pessoas;  
    private Scanner scanner;  
  
    public AgendaTelefonica(){  
        pessoas = new Vector<Pessoa>();  
        scanner = new Scanner(System.in);  
    }  
  
    public void cadastrarPessoa(){  
        System.out.println("\nInsira um nome para a pessoa: ");  
        String nome = scanner.next();  
        System.out.println("\nInsira um telefone para a pessoa: ");  
        String telefone = scanner.next();  
        System.out.println("\nInsira um email para a pessoa: ");  
        String email = scanner.next();  
        Pessoa nova = new Pessoa(nome, telefone, email);  
        pessoas.add(nova);  
        System.out.println("\nNovo contato adicionado\n");  
    }  
}
```

Linguagem Java

Mesma aplicação agora com usando ArrayList<T>:

java.util

Class ArrayList<E>

java.lang.Object

java.util.AbstractCollection<E>

java.util.AbstractList<E>

java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
```

```
extends AbstractList<E>
```

```
implements List<E>, RandomAccess, Cloneable, Serializable
```

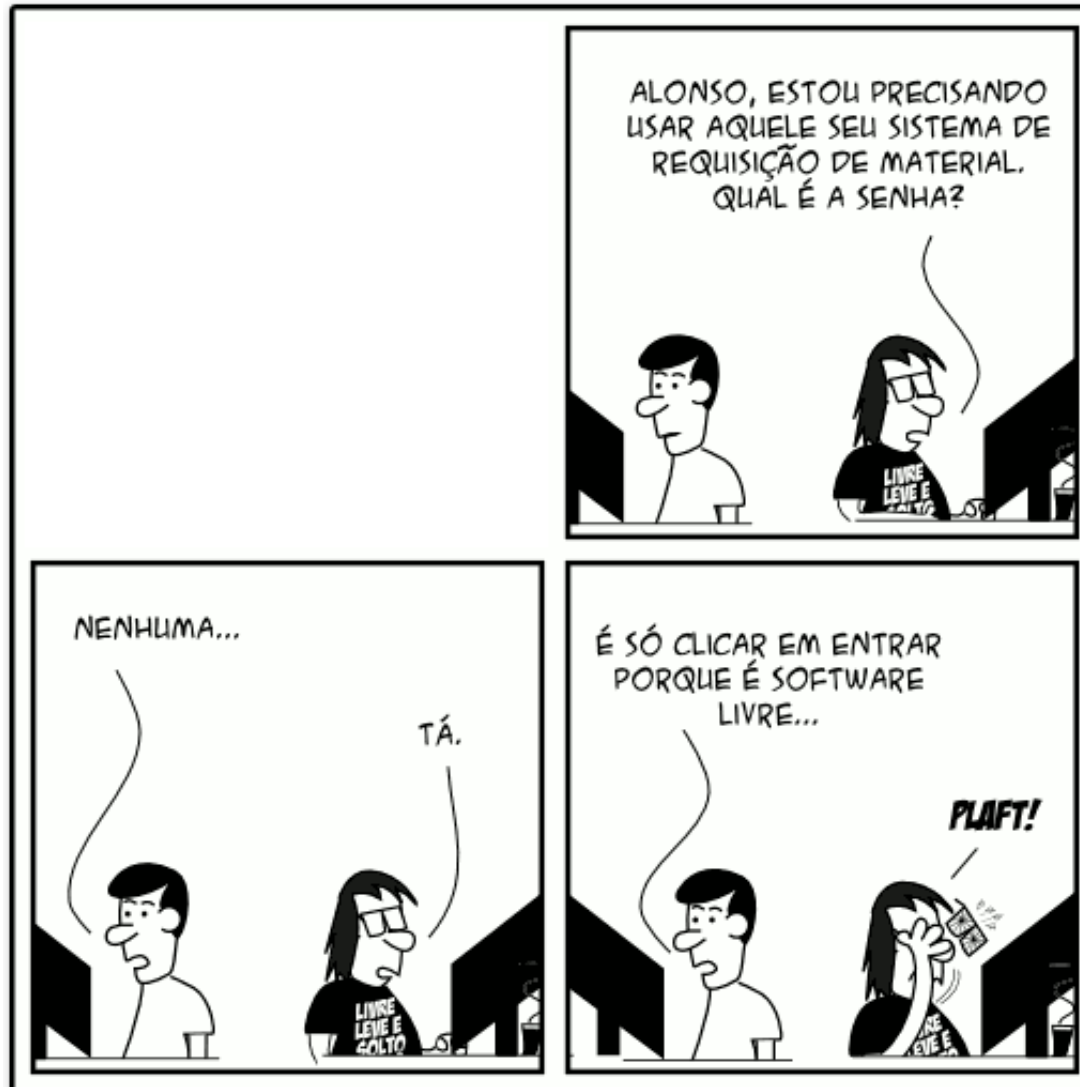
Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, i array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

Linguagem Java

Mesma aplicação agora com usando ArrayList<T>:

```
public class AgendaTelefonica {  
    private ArrayList<Pessoa> pessoas;  
    private Scanner scanner;  
  
    public AgendaTelefonica(){  
        pessoas = new ArrayList<Pessoa>();  
        scanner = new Scanner(System.in);  
    }  
  
    public void cadastrarPessoa(){  
        System.out.println("\nInsira um nome para a pessoa: ");  
        String nome = scanner.next();  
        System.out.println("\nInsira um telefone para a pessoa: ");  
        String telefone = scanner.next();  
        System.out.println("\nInsira um email para a pessoa: ");  
        String email = scanner.next();  
        Pessoa nova = new Pessoa(nome, telefone, email);  
        pessoas.add(nova);  
        System.out.println("\nNovo contato adicionado\n");  
    }  
}
```

Perguntas?

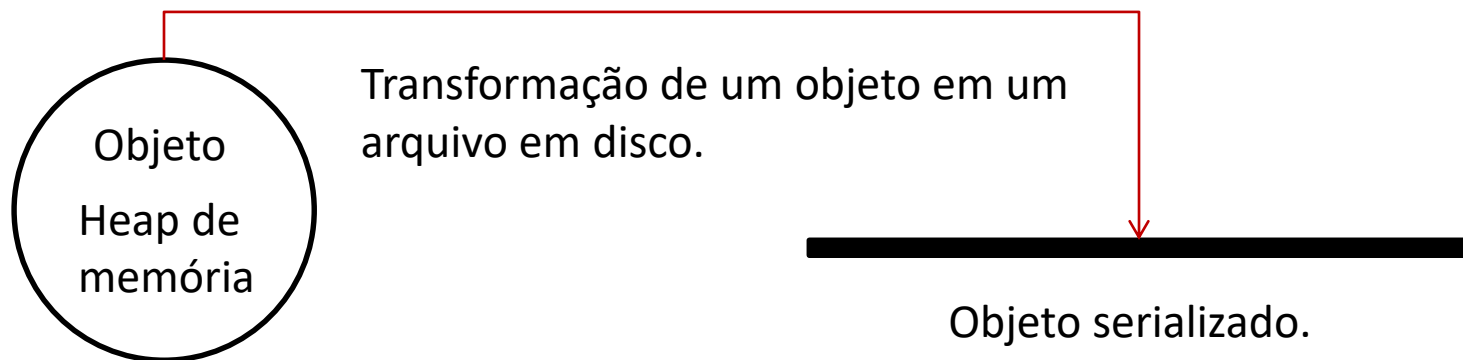


Linguagem Java

Serialização de um objeto.

Se os seus dados serão lidos e processados apenas por um programas feitos em Java, então é possível armazená-los de forma serializada (flat).

Após a serialização é possível transportas objetos em sua forma binária, como se fosse um arquivo.



Linguagem Java

Nos exemplos anteriores fizemos uma agenda eletrônica, mas não existia persistência das informações inseridas na agenda.

Agora a agenda pode ser serializada.



```
public class Agenda implements Serializable{  
}
```

Na linguagem Java tem-se uma API extremamente robusta para leitura e escrita em disco. O pacote `java.io` tem tudo que você precisa para serializar um objeto.

Para “avisar” a máquina virtual Java que um objeto deve ou pode se serializado usa-se a interface `Serializable`.

Linguagem Java

Mas vamos começar com um exemplo menor!

API para escrita de arquivos
na linguagem Java.

```
import java.io.Serializable;

@SuppressWarnings("serial")
public class Pessoa implements Serializable{
    private String nome;
    private String email;
    private String telefone;
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
Pessoa eu = new Pessoa();
eu.setNome("Gerson"); eu.setEmail("eu@gmail.com");
eu.setTelefone("12987453627");

FileOutputStream escritaEmArquivo = new FileOutputStream("pessoa.ser");
ObjectOutputStream escritaDoObjeto = new ObjectOutputStream(escritaEmArquivo);
escritaDoObjeto.writeObject(eu);
escritaDoObjeto.close();
```

Linguagem Java

O que de fato acontece ?

FileOutputStream é uma classe que tem métodos para escrita em disco.

ObjectOutputStream é uma classe que tem métodos para escrita de objetos. A escrita de objetos não acontece somente em disco, mas também em outros tipos de saídas como Rede, e Banco de Dados.

A serialização transforma um objeto na memória em um arquivo binário, nesse caso, são salvos as informações binárias do ByteCode gerado para o objeto.

Linguagem Java

E como se faz a leitura ?

```
FileInputStream leituraDoArquivo = new FileInputStream("pessoa.ser");  
ObjectInputStream leituraDoObjeto = new ObjectInputStream(leituraDoArquivo);  
Object meuObjeto = leituraDoObjeto.readObject();  
eu = (Pessoa)meuObjeto;  
leituraDoObjeto.close();
```

A API java.io fornece classes para escrita e leitura de arquivos em disco.

Detalhe importante, quando se necessita serializar objetos que tenham como variáveis de instância outros objetos, então as classes dos objetos usados como variáveis também devem implementar a interface serializable.

Linguagem Java

Um erro acontece aqui. A classe pessoa necessita implementar a interface de serialização.

```
public class Pessoa{  
    private String nome;  
    private String email;  
    private String telefone;
```

```
public class Agenda implements Serializable{  
    private List<Pessoa> pessoas = new ArrayList<Pessoa>();
```

```
Agenda agenda = new Agenda();  
Pessoa eu = new Pessoa();  
eu.setNome("Gerson"); eu.setEmail("eu@gmail.com");  
eu.setTelefone("12987453627");  
agenda.getPessoas().add(eu);  
  
FileOutputStream escritaEmArquivo = new FileOutputStream("agenda.ser");  
ObjectOutputStream escritaDoObjeto = new ObjectOutputStream(escritaEmArquivo);  
escritaDoObjeto.writeObject(agenda);  
escritaDoObjeto.close();
```


Linguagem Java

É possível escolher definir se alguma variável de instância não será serializada.

```
public class Pessoa{  
    private String nome;  
    private String email;  
    private String telefone;
```

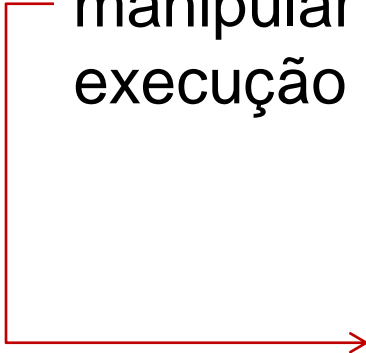
```
public class Agenda implements Serializable{  
    private transient List<Pessoa> pessoas = new ArrayList<Pessoa>();
```

```
Agenda agenda = new Agenda();  
Pessoa eu = new Pessoa();  
eu.setNome("Gerson"); eu.setEmail("eu@gmail.com");  
eu.setTelefone("12987453627");  
agenda.getPessoas().add(eu);  
  
FileOutputStream escritaEmArquivo = new FileOutputStream("agenda.ser");  
ObjectOutputStream escritaDoObjeto = new ObjectOutputStream(escritaEmArquivo);  
escritaDoObjeto.writeObject(agenda);  
escritaDoObjeto.close();
```

Linguagem Java

Tratamento de exceção.

Tratamento de exceção é o procedimento realizado para manipular problemas que podem acontecer durante a execução do programa.



```
public static void main(String[] args) {  
    System.out.println("Divisão: "+dividir(10,5));  
}  
public static int dividir(int x, int y){  
    return x/y;  
}
```

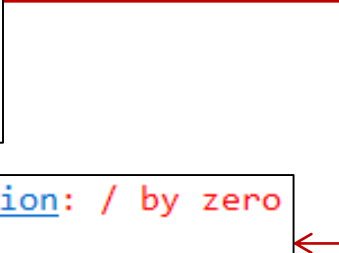
O que acontece quando a divisão for por Zero?

Uma exceção é exatamente isso, um problema que pode vir a acontecer e que tem tratamento, ou seja, tem alguma solução para isso.

Linguagem Java

```
public static void main(String[] args) {  
    System.out.println("Divisão: "+dividir(10,0));  
}  
public static int dividir(int x, int y){  
    return x/y;  
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at br.com.fatec.Inicio.dividir(Inicio.java:8)  
    at br.com.fatec.Inicio.main(Inicio.java:5)
```



Existem vários tipos de exceção e todas são heranças da classe Exception.

Ex:

Exception.

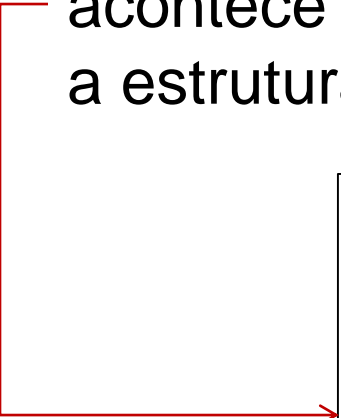
ArithmeticException.

NoSuchFileException.

...

Linguagem Java

É possível capturar a exceção no momento em que ela acontece a dar o tratamento adequado. Para isso usa-se a estrutura Try-Catch.




```
public static void main(String[] args) {  
    System.out.println("Divisão: "+dividir(10,0));  
}  
public static int dividir(int x, int y){  
    try{  
        return x/y;  
    }catch(ArithmeticException ex){  
        return 0;  
    }  
}
```

Um bloco do tipo Try pode ter vários blocos do tipo Catch associados a ele, isso serve para tratar em um método que pode disparar mais de uma exceção.

Linguagem Java

Também é possível lançar uma exceção para que ela seja tratada por um nível superior.



```
public static void main(String[] args) {  
    try{  
        System.out.println("Divisão: "+dividir(10,0));  
    }catch(ArithmeticException ex){  
        System.out.println("Divisão por zero não é possível");  
    }  
}  
public static int dividir(int x, int y) throws ArithmeticException{  
    return x/y;  
}
```

Desse modo pode-se chegar a conclusão de que é possível criar uma cascata de exceções. Se o último nível de processamento, ou seja o último método, não tratar a exceção então ela será executada pela máquina virtual.

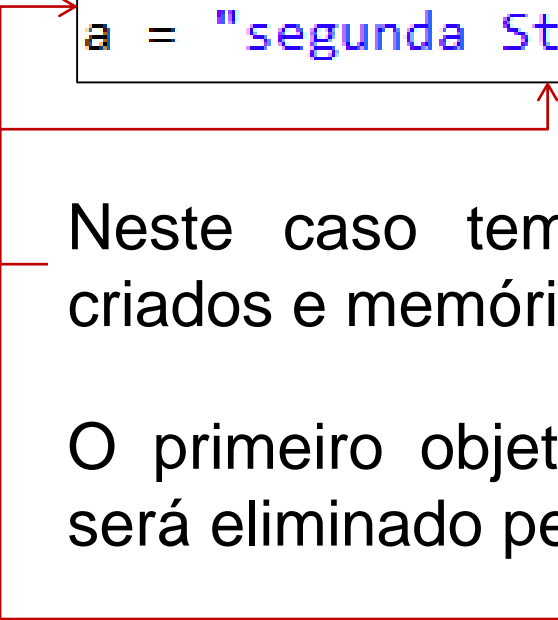
Perguntas?



Linguagem Java

String é um objeto imutável, ou seja, não é possível alterar o valor de um objeto do tipo String.

```
String a = "primeira String";  
a = "segunda String";
```



Neste caso tem-se uma variável mas dois objetos criados e memória.

O primeiro objeto criado fica “perdido” na memória e será eliminado pelo coletor de lixo.

Linguagem Java

Neste caso a cada execução do laço, um novo objeto do tipo String será criado.

```
String a = "";  
for(int i = 0; i < 100; i++){  
    a += "string";  
}
```

```
String str1 = "teste";  
str1.concat("imutável");  
str1.toUpperCase();  
System.out.println(str1);
```

Neste caso o que acontece com a str1?

Existem duas classes criadas para tratar o problema da imutabilidade em Strings. StringBuilder e StringBuffer. Recomenda-se o uso dessas classes em softwares que fazem o grande uso de concatenações.

Linguagem Java

Os operadores += e + usados para concatenação são lentos se comparados com os métodos das classes StringBuilder e StringBuffer.

```
StringBuffer buffer = new StringBuffer();  
for(int i = 0; i < 10; i++){  
    buffer.append("string");  
}  
String a = buffer.toString();
```

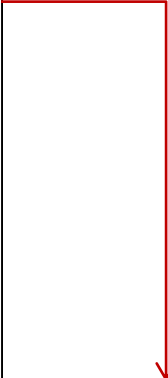
O procedimento para se usar o StringBuilder é análogo ao procedimento apresentado para o StringBuffer.

A diferença entre eles é que o StringBuffer suporta sincronismo. Por isso o StringBuilder é sensivelmente mais rápido.

```
public static void concatenarBuilder(int vezes){
    StringBuilder builder = new StringBuilder();
    for(int i = 0; i < vezes; i++){
        builder.append("String");
    }
    String a = builder.toString();
}

public static void concatenarBuffer(int vezes){
    StringBuffer buffer = new StringBuffer();
    for(int i = 0; i < vezes; i++){
        buffer.append("String");
    }
    String a = buffer.toString();
}

public static void concatenar(int vezes){
    String a = "";
    for(int i = 0; i < vezes; i++){
        a += "String";
    }
}
```



```
int vezes = 10000;
long tempo = System.currentTimeMillis();
concatenarBuilder(vezes);
System.out.println(System.currentTimeMillis() - tempo);
tempo = System.currentTimeMillis();
concatenarBuffer(vezes);
System.out.println(System.currentTimeMillis() - tempo);
tempo = System.currentTimeMillis();
concatenar(vezes);
System.out.println(System.currentTimeMillis() - tempo);
```

Linguagem Java

A comparação de tipos primitivos é feita através do operador ==.

Em objetos esse operador serve para comparar referências. Mas também é possível usar o método equals().


```
String a = "A";  
String b = "A";  
System.out.println(a.equals(b));
```

Na classe String o método equals() funciona comparando caractere à caractere.

Linguagem Java

```
public class Pessoa {  
    private String nome;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    @Override  
    public boolean equals(Object obj){  
        boolean igual = false;  
        Pessoa p = (Pessoa)obj;  
        if(this.nome.equals(p.getNome()))  
            igual = true;  
        return igual;  
    }  
}
```

```
Pessoa p1 = new Pessoa();  
Pessoa p2 = new Pessoa();  
p1.setNome("nome");  
p2.setNome("nome");  
System.out.println(p1.equals(p2));
```



Linguagem Java

Casos estranhos:

```
public class Main {  
    public static void main(String args[]) {  
        float a = 1.299f; float b = 2.899f;  
        int c = a+b; // perda de precisão  
        int d = 0;  
        d += a; d += b;  
        System.out.println(d); // 3!  
    }  
}
```

```
public static boolean éÍmpar(int x) {  
    return ((x % 2) == 1);  
}  
  
public static void main(String[] args) {  
    System.out.println(éÍmpar(0)); // false  
    System.out.println(éÍmpar(1)); // true  
    System.out.println(éÍmpar(2)); // false  
    System.out.println(éÍmpar(-2)); // false  
    System.out.println(éÍmpar(-1)); // false ??  
}
```

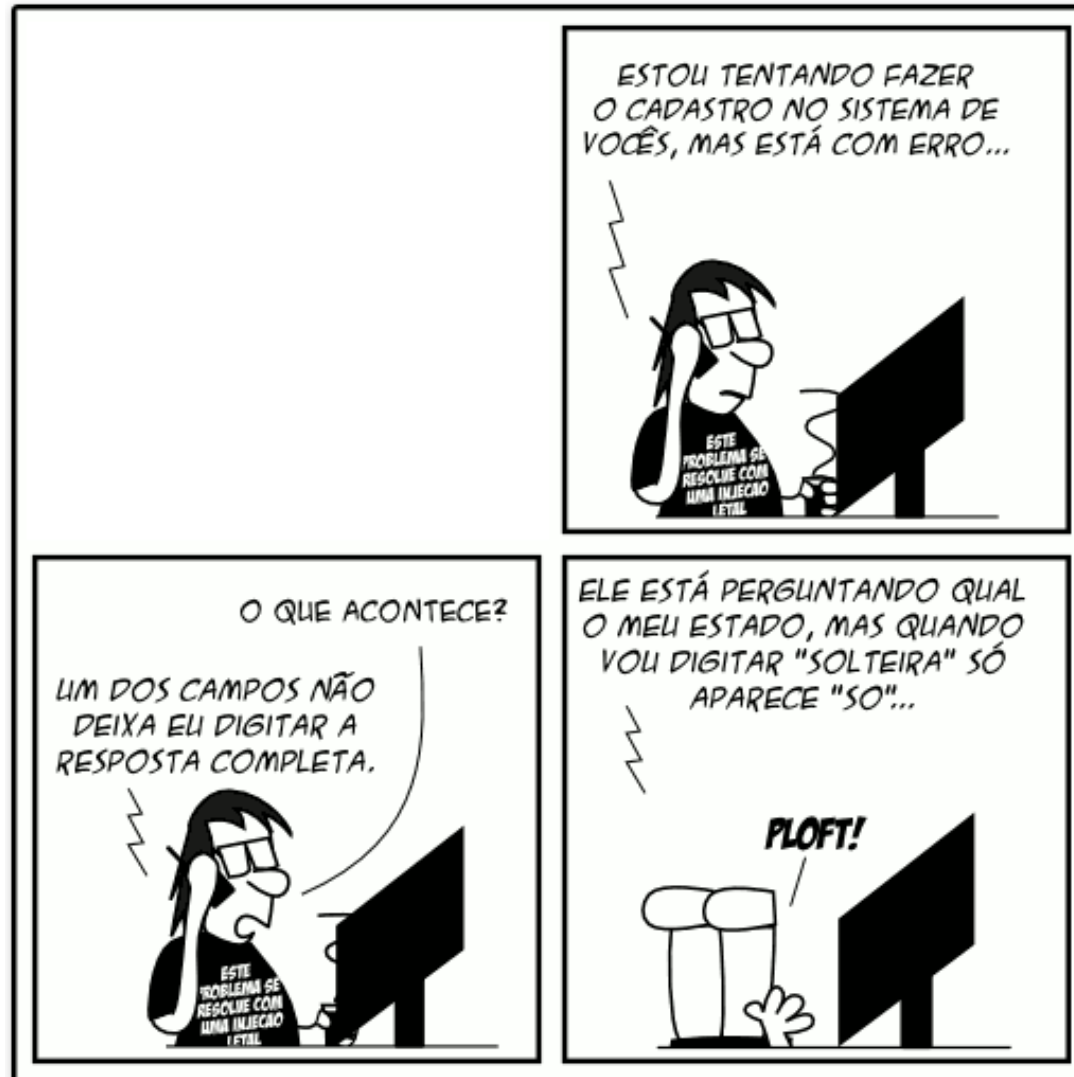
Linguagem Java

Casos estranhos:

```
public class Main {  
    public static void main(String args[]) {  
        float a = 1.299f; float b = 2.899f;  
        int c = a+b; // perda de precisão  
        int d = 0;  
        d += a; d += b;  
        System.out.println(d); // 3!  
    }  
}
```

```
public static boolean éÍmpar(int x) {  
    return ((x % 2) == 1);  
}  
  
public static void main(String[] args) {  
    System.out.println(éÍmpar(0)); // false  
    System.out.println(éÍmpar(1)); // true  
    System.out.println(éÍmpar(2)); // false  
    System.out.println(éÍmpar(-2)); // false  
    System.out.println(éÍmpar(-1)); // false ??  
}
```

Perguntas?



Linguagem Java

Leitura e escrita em arquivos de texto ASCII.

```
FileWriter escritor = new FileWriter("arquivo.txt");  
escritor.write("escrevendo no arquivo");  
escritor.close();
```

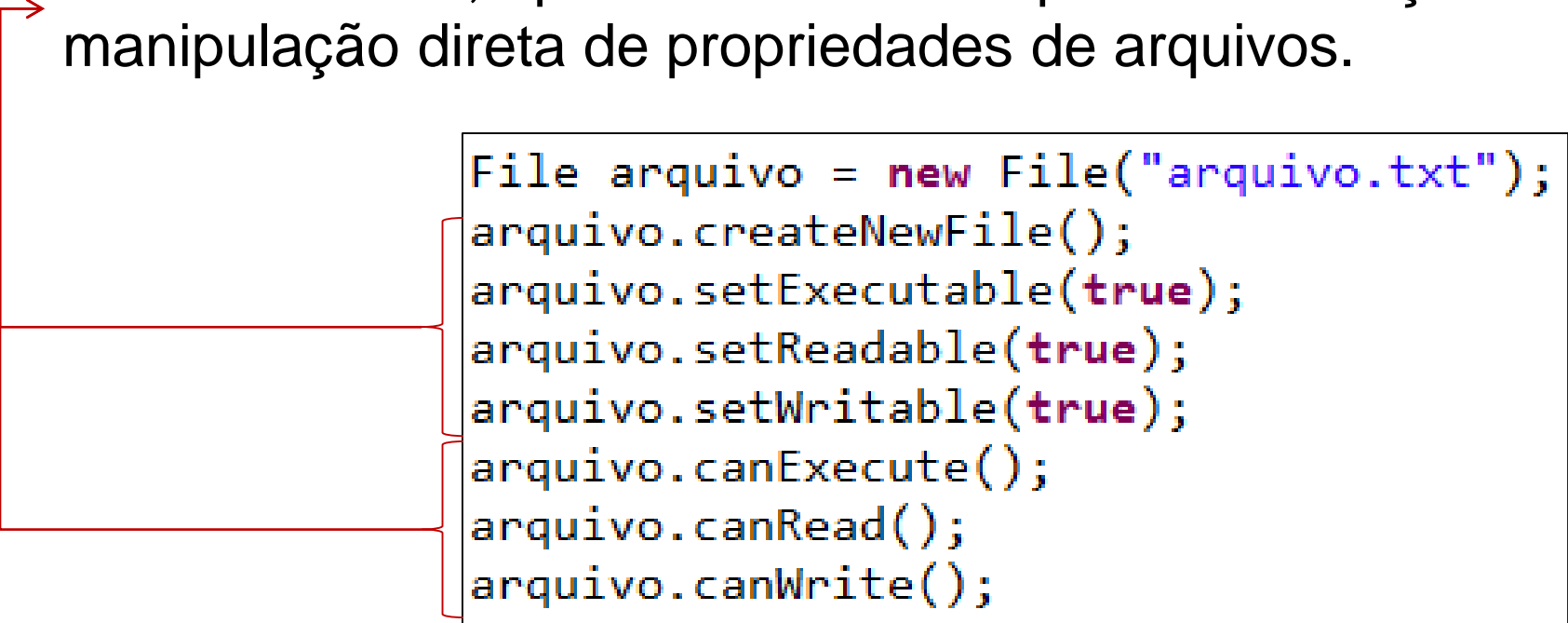
O procedimento de leitura e escrita em um arquivo de texto é semelhante ao procedimento de serialização de objetos.

O método write() escreve em um arquivo, cujo nome e endereço deve ser passado como argumento.

```
File arquivo = new File("arquivo.txt");  
FileWriter escritor = new FileWriter(arquivo);  
escritor.write("escrevendo no arquivo");  
escritor.close();
```


Linguagem Java

A classe File, possui métodos para verificação e manipulação direta de propriedades de arquivos.



```
File arquivo = new File("arquivo.txt");
arquivo.createNewFile();
arquivo.setExecutable(true);
arquivo.setReadable(true);
arquivo.setWritable(true);
arquivo.canExecute();
arquivo.canRead();
arquivo.canWrite();
```

Essas propriedades serão válidas para todo o sistema operacional incluindo seus usuários. É possível inclusive definir a acessibilidade de propriedades de um arquivo para um usuário específico.

Linguagem Java

A classe File, também pode criar pastas.

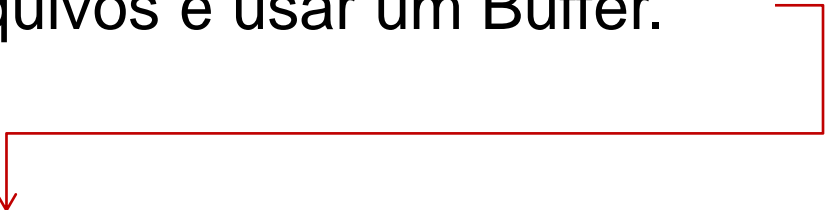
```
File arquivo = new File("pasta");
arquivo.mkdir();
arquivo.setExecutable(false);
arquivo.setReadable(false);
arquivo.setWritable(false);
arquivo.canExecute();
arquivo.canRead();
arquivo.canWrite();
```

Ainda é possível listar o conteúdo de uma pasta e com isso saber exatamente o tipo de cada arquivo.

```
File arquivo = new File("C:\\Users\\Gerson\\ebooks\\JAVA");
if(arquivo.isDirectory()){
    String[] arquivos = arquivo.list();
    for(String nome : arquivos){
        System.out.println("Nome: "+nome);
    }
}
```

Linguagem Java

Outra forma escrever em arquivos é usar um Buffer.



```
BufferedWriter escritor = new BufferedWriter(new FileWriter("arquivo.txt"));
escritor.write("escrevendo no arquivo");
escritor.close();
```

A vantagem em usar um Buffer está no desempenho durante o processo de escrita.

```
BufferedWriter escritor = new BufferedWriter(new FileWriter("arquivo.txt"));
for(int i = 0; i < 10; i++){
    escritor.write("escrevendo no arquivo");
    escritor.newLine();
}
escritor.close();
```

Linguagem Java

A leitura de informações em um arquivo de texto, segue um processo semelhante a escrita.

```
File arquivo = new File("C:\\Users\\Gerson\\Desktop\\arquivo.txt");
FileReader leitor = new FileReader(arquivo);
BufferedReader bufferLeitor = new BufferedReader(leitor);
String linha = bufferLeitor.readLine();
String[] elementos;
while(linha != null){
    elementos = linha.split(" ");
    for(String elemento:elementos ){
        System.out.println(elemento);
    }
    linha = bufferLeitor.readLine();
}
leitor.close();
```

A mesma hierarquia de classes pode ser aplicada para leitura em arquivos.

Linguagem Java

A classe String é uma das mais usadas em aplicações Java. Usa-se e muito a String para tratar informações lidas a partir de arquivos de texto.

```
String email = "gerson.neto@gmail.com";  
String[] pedacos = email.split("@");  
for(String pedaco : pedacos){  
    System.out.println(pedaco);  
}
```

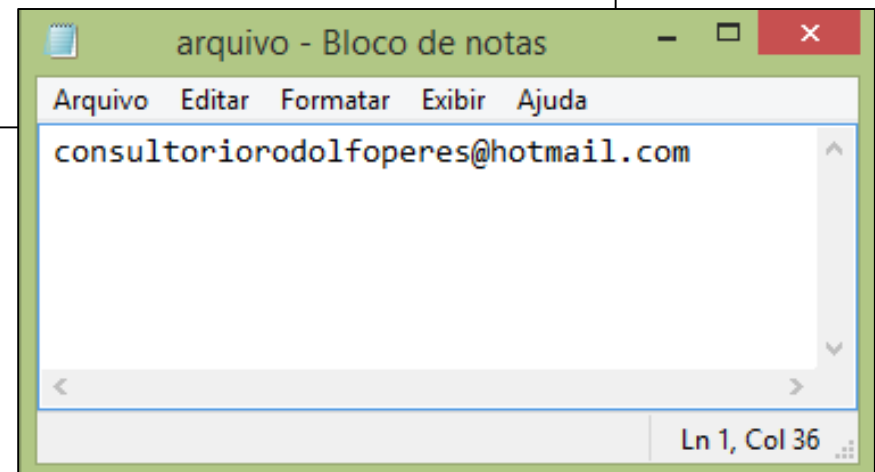
O método `split()` separa uma string em pedaços, sendo que o ponto de corte usado para separar as seqüências de caracteres é uma outra string. No exemplo, foram criadas duas novas strings. O ponto de corte é a string “@”.

Linguagem Java

Tratando informações de arquivos. Ex: separar o nome de usuário do domínio do email.

```
// Separar o domínio do usuário de email
File arquivo = new File("C:\\Users\\Gerson\\Desktop\\arquivo.txt");
FileReader leitor = new FileReader(arquivo);
BufferedReader bufferLeitor = new BufferedReader(leitor);
String linha = bufferLeitor.readLine();
String[] elementos;
while(linha != null){
    elementos = linha.split("@");
    System.out.println("Email: "+elementos[0]);
    System.out.println("Domínio: "+elementos[1]);
    linha = bufferLeitor.readLine();
}
leitor.close();
```

Email: consultoriorodolfoperes
Domínio: hotmail.com

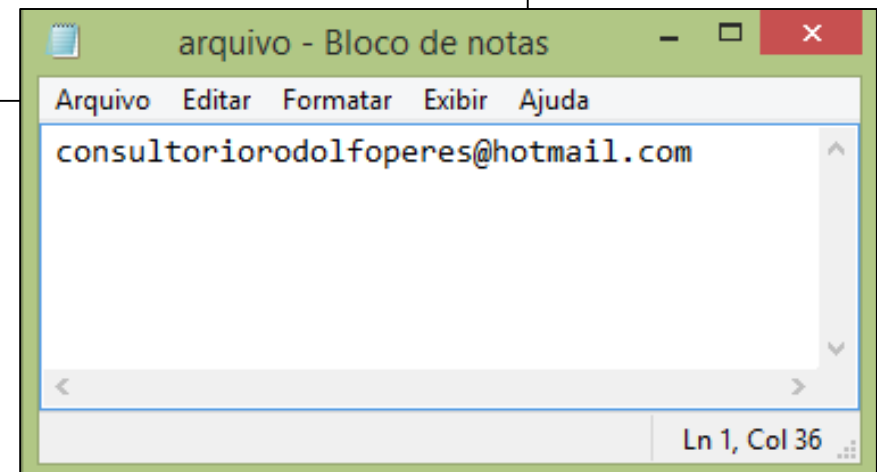


Linguagem Java

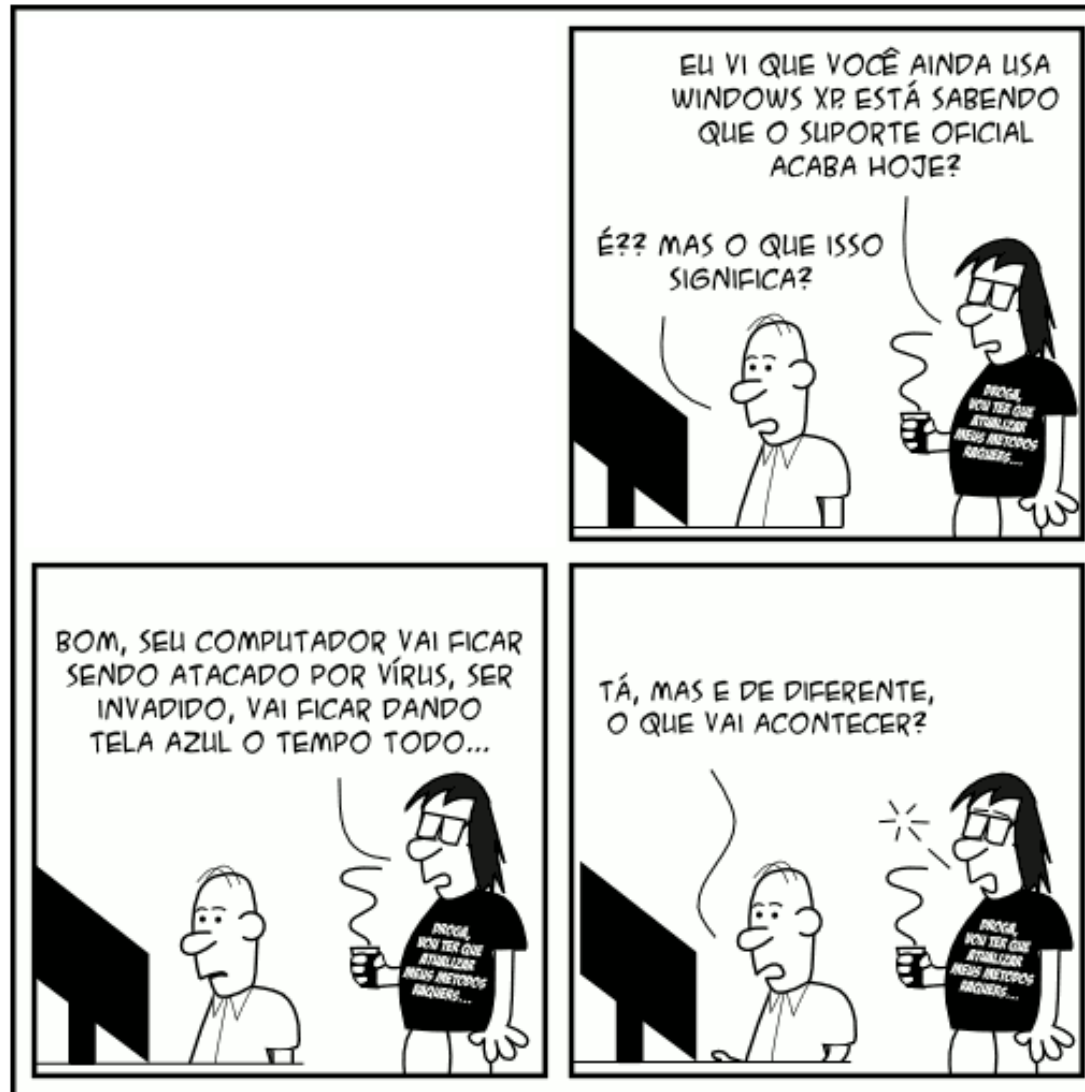
Tratando informações de arquivos. Ex: transformar para letras maiúsculas.

```
// Separar o domínio do usuário de email
File arquivo = new File("C:\\Users\\Gerson\\Desktop\\arquivo.txt");
FileReader leitor = new FileReader(arquivo);
BufferedReader bufferLeitor = new BufferedReader(leitor);
String linha = bufferLeitor.readLine();
String[] elementos;
while(linha != null){
    elementos = linha.split("@");
    System.out.println("Email: "+elementos[0].toUpperCase());
    System.out.println("Domínio: "+elementos[1].toUpperCase());
    linha = bufferLeitor.readLine();
}
leitor.close();
```

Email: consultoriorodolfoperes
Domínio: hotmail.com

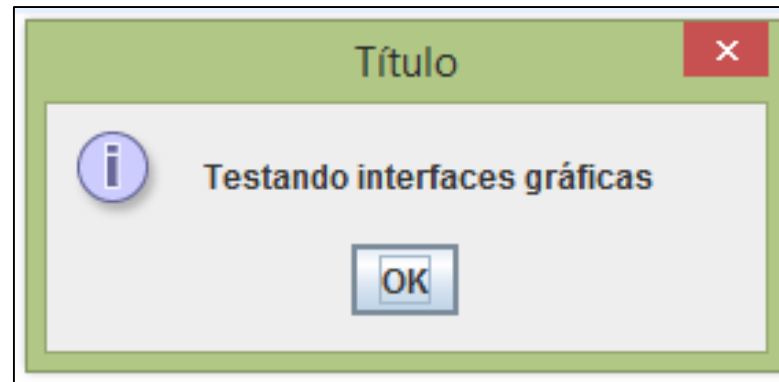


Perguntas?



Linguagem Java

Uma interface gráfica com o usuário (Graphical User Interface - GUI) apresenta um mecanismo amigável ao usuário para interagir com um aplicativo.



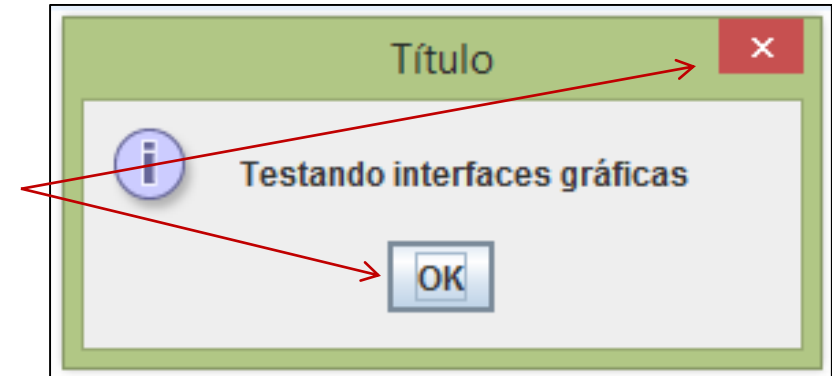
GUIs são muito úteis pois dão uma aparência e um comportamento peculiar aos aplicativos. Fornecer aos aplicativos diferentes componentes de interface com o usuário acelera a aprendizagem de uso do aplicativo.

Linguagem Java

A classe JOptionPane do Java (javax.swing) fornece caixas de diálogo pré-empacotadas tanto para entrada como para saída.

```
public static void main(String[] args){  
    JOptionPane.showMessageDialog(null, "Testando interfaces gráficas",  
        "Título", 1);  
}
```

Os botões OK e fechar são fornecidos.

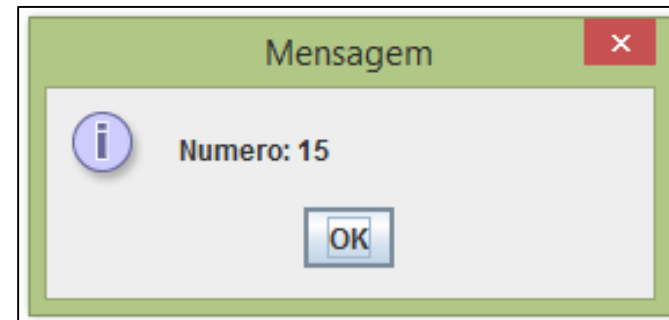
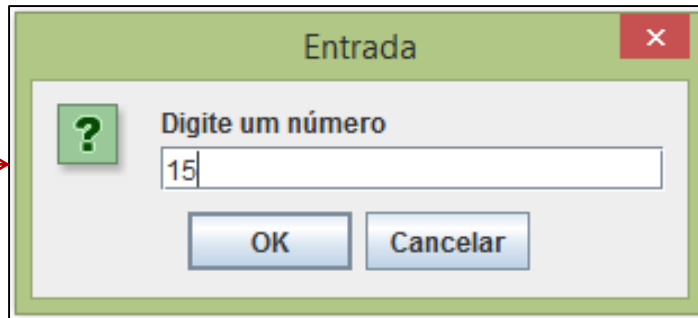


```
public static void main(String[] args){  
    JOptionPane.showMessageDialog(null, "Testando interfaces gráficas");  
}
```

Linguagem Java

É possível receber valores através dos métodos da classe JOptionPane.

```
String n1 = JOptionPane.showInputDialog("Digite um número");  
int numero = Integer.parseInt(n1);  
StringBuilder sb = new StringBuilder();  
sb.append("Numero: ");sb.append(numero);  
JOptionPane.showMessageDialog(null, sb.toString());
```



Os métodos usados para entrada e saída de valores são abstratos.

Linguagem Java

Toda aplicação com interface gráfica, começa com uma janela gráfica. Em Java para construção de Janelas Gráficas usa-se a classe JFrame.

```
JFrame janela = new JFrame("Titulo");  
janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
janela.setSize(400, 400);  
janela.setVisible(true);
```

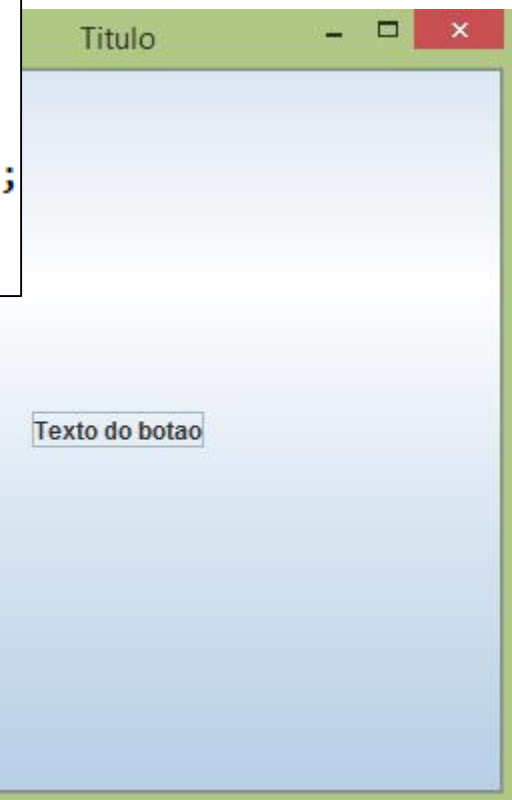
Informações básicas para construção de uma janela gráfica em Java.

Classe Java pertencente ao pacote javax.swing. O pacote swing contém vários componentes gráficos.

Linguagem Java

Adicionar componentes na janela gráfica, corresponde a criar e adicionar objetos como conteúdo do Objeto JFrame.

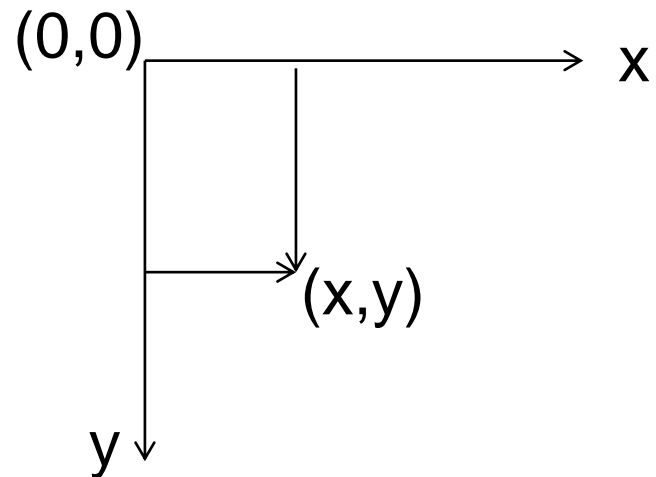
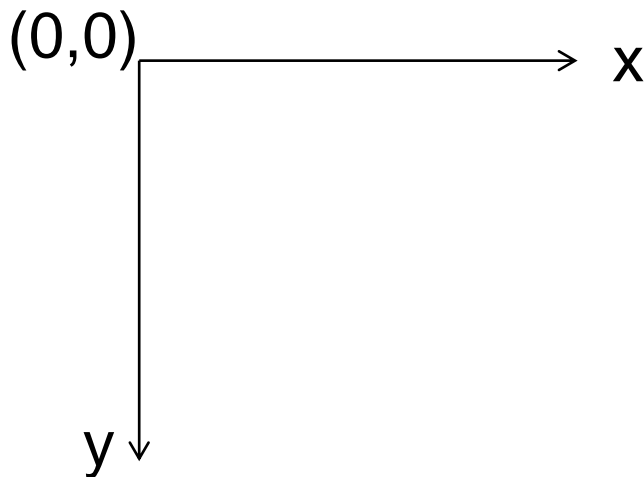
```
JFrame janela = new JFrame("Titulo");  
JButton botao = new JButton("Texto do botao");  
botao.setSize(50, 50);  
janela.getContentPane().add(botao);  
janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
janela.setSize(400, 400);  
janela.setVisible(true);
```



Linguagem Java

Um dos recursos mais interessantes em Java é seu suporte gráfico, que permite aos programadores aprimorar visualmente seus aplicativos.

Em Java todo desenho feito na tela, segue um sistema de coordenadas, que começa no canto superior esquerdo.



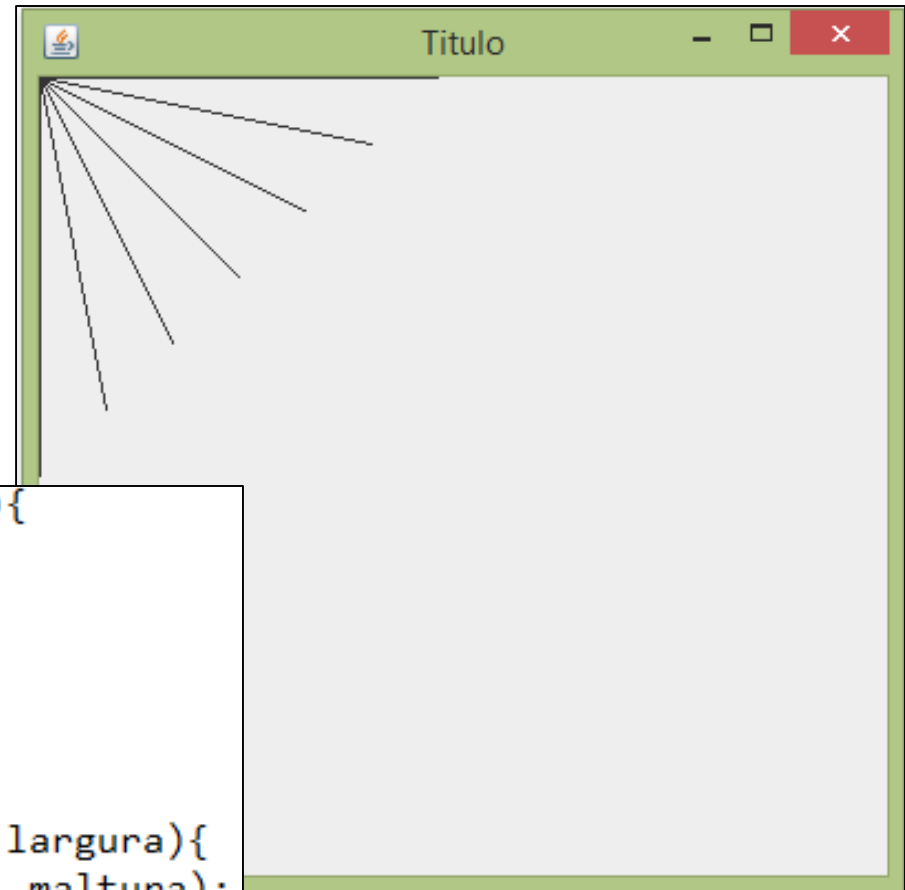
Linguagem Java

A linguagem Java possui uma biblioteca robusta para se fazer desenhos.

```
public class MeuPainel extends JPanel {  
    @Override  
    public void paintComponent(Graphics g){  
        super.paintComponent(g);  
        int largura = getWidth();  
        int altura = getHeight();  
        g.drawLine(0, 0, largura, altura);  
        g.drawLine(0, altura, largura, 0);  
    }  
}
```

A classe JPanel representa literalmente um painel onde pode-se desenhar formas de maneira livre. Pode-se também adicionar componentes a um painel. A classe Graphics possui métodos específicos para desenhar formas, como círculos, quadrados e outros.

Linguagem Java

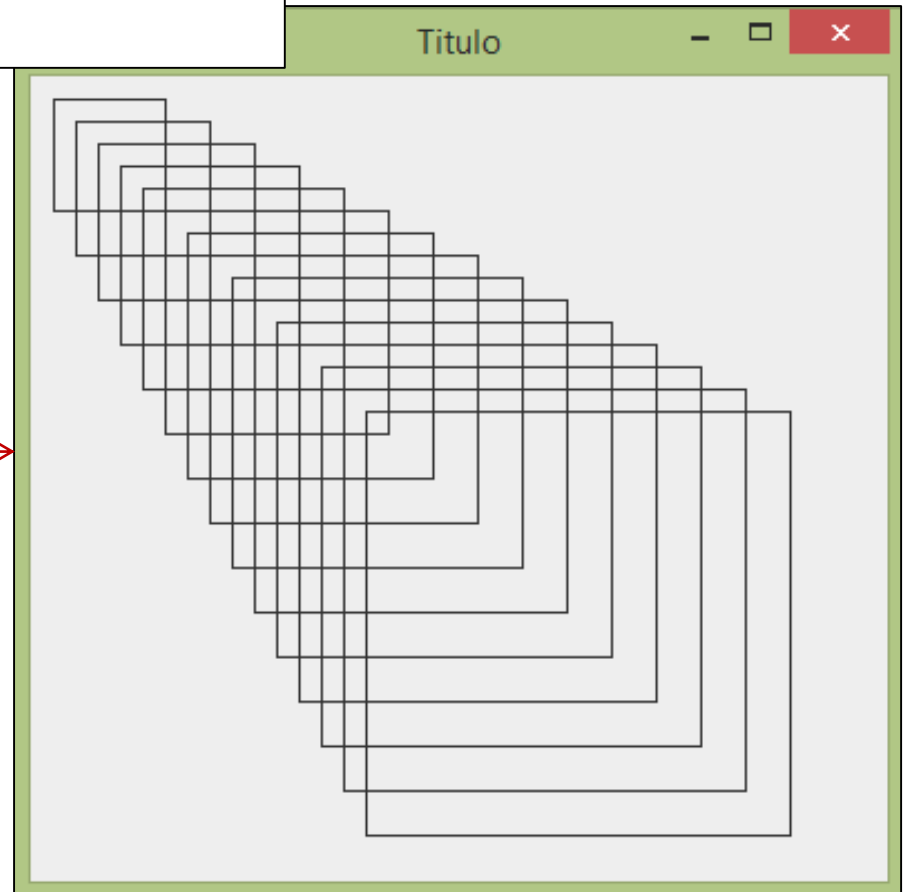


```
public void paintComponent(Graphics g){  
    super.paintComponent(g);  
    int largura = getWidth();  
    int altura = getHeight();  
    int mlargura = 0;  
    int maltura = altura/2;  
    for(int i = 0; i < 15; i++){  
        if(maltura >= 0 & mlargura <= largura){  
            g.drawLine(0, 0, mlargura, maltura);  
            mlargura+=30;  
            maltura-=30;  
        }  
    }  
}
```


Linguagem Java

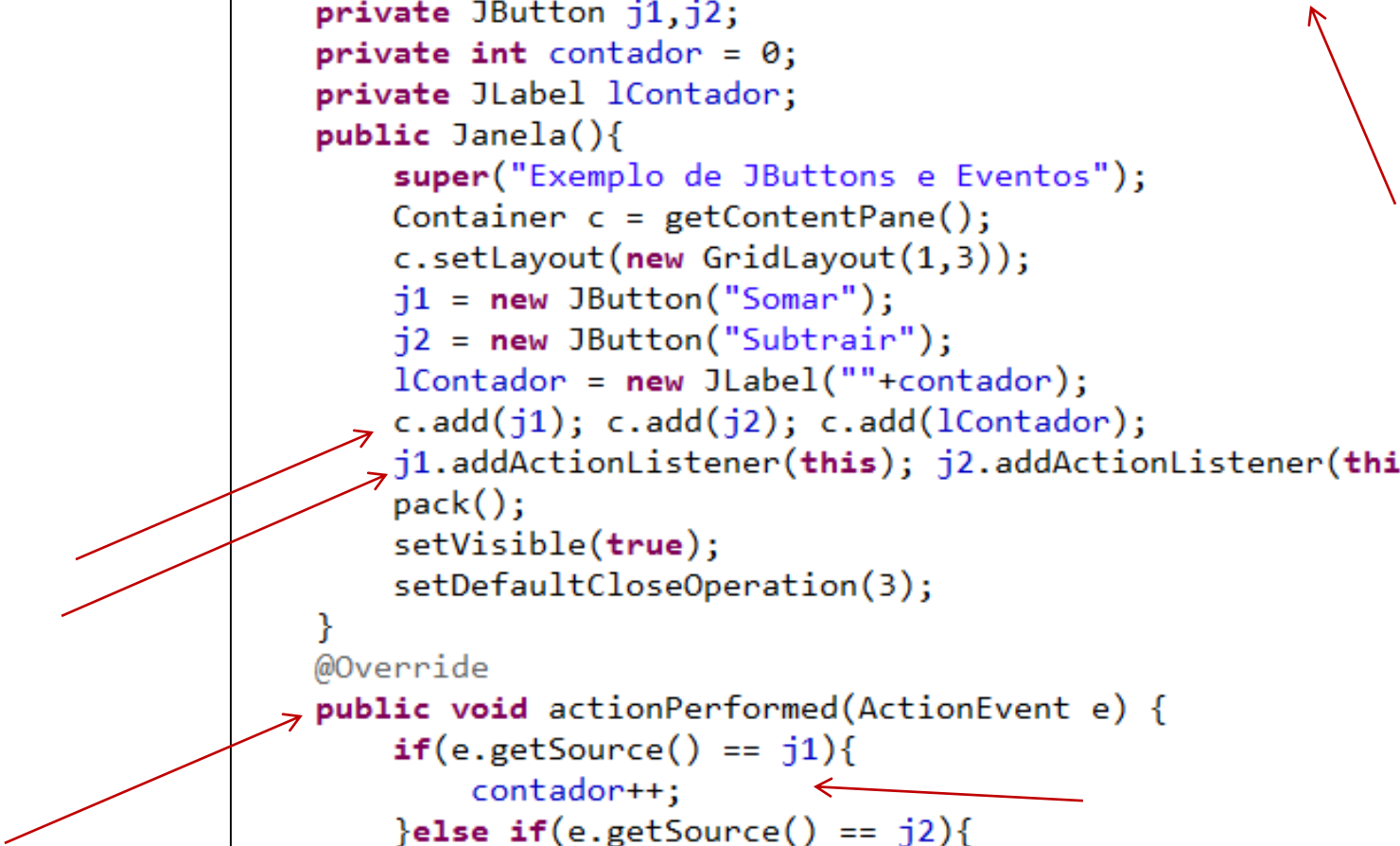
```
public void paintComponent(Graphics g){  
    super.paintComponent(g);  
    for(int i = 0; i < 15; i++){  
        g.drawRect(10+i*10, 10+i*10, 50+i*10, 50+i*10);  
    }  
}
```

A Classe Graphics possui métodos para desenhar forma geométricas simples.



Linguagem Java

```
public class Janela extends JFrame implements ActionListener{
    private JButton j1,j2;
    private int contador = 0;
    private JLabel lContador;
    public Janela(){
        super("Exemplo de JButtons e Eventos");
        Container c = getContentPane();
        c.setLayout(new GridLayout(1,3));
        j1 = new JButton("Somar");
        j2 = new JButton("Subtrair");
        lContador = new JLabel(""+contador);
        c.add(j1); c.add(j2); c.add(lContador);
        j1.addActionListener(this); j2.addActionListener(this);
        pack();
        setVisible(true);
        setDefaultCloseOperation(3);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == j1){
            contador++;
        }else if(e.getSource() == j2){
            contador--;
        }
        lContador.setText(""+contador);
    }
}
```

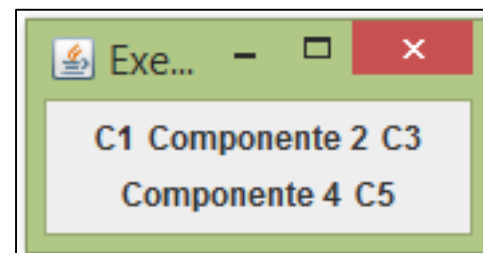
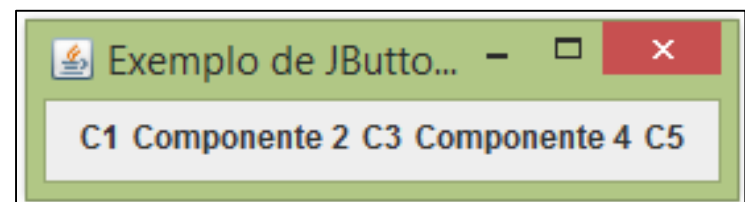


Linguagem Java

Os componentes em uma Janela gráfica são agrupados através de layouts.

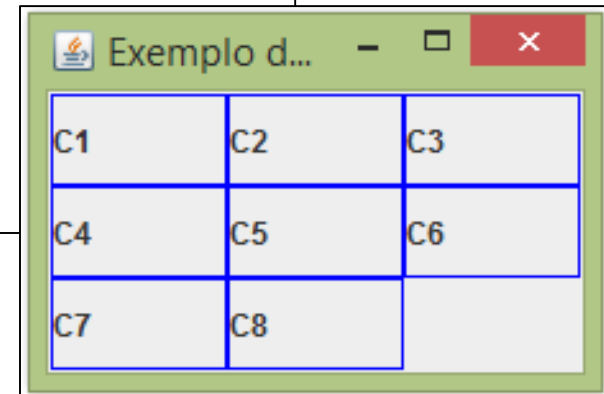
Para usar um layout, executamos um método indicar qual será o layout do painel de conteúdo da janela gráfica. O layout também indica como os componentes serão rearranjados se o tamanho da janela mudar.

```
public Janela(){  
    super("Exemplo de JButtons e Eventos");  
    Container c = getContentPane();  
    c.setLayout(new FlowLayout());  
    c.add(new JLabel("C1"));  
    c.add(new JLabel("Componente 2"));  
    c.add(new JLabel("C3"));  
    c.add(new JLabel("Componente 4"));  
    c.add(new JLabel("C5"));  
    pack();  
    setVisible(true);  
    setDefaultCloseOperation(3);  
}
```



Linguagem Java

```
public Janela(){  
    super("Exemplo de JButtons e Eventos");  
    Container c = getContentPane();  
    c.setLayout(new GridLayout(3,3));  
    for(int i = 1; i <= 8; i++){  
        JLabel l = new JLabel("C"+i);  
        l.setBorder(BorderFactory.createLineBorder(Color.blue));  
        c.add(l);  
    }  
    pack();  
    setVisible(true);  
    setDefaultCloseOperation(3);  
}
```



Com o GridLayout os componentes são organizados em uma grade. Existem outros layouts, como: BorderLayout, CardLayout, SpringLayout...

Linguagem Java

Todo componente tem uma super classe chamada JComponente.

```
public class Rabisco extends JComponent implements
    MouseListener, MouseMotionListener{
    private ArrayList<Point> pontos;
    private int tamanho = 8;
    private int metade = tamanho / 2;
    private Color cor;

    public Rabisco(Color cor){
        this.cor = cor;
        pontos = new ArrayList<Point>(1024);
        addMouseListener(this);
        addMouseMotionListener(this);
    }
}
```

@Override

```
protected void paintComponent(Graphics g){
    Graphics2D g2d = (Graphics2D)g;
    g2d.setColor(Color.WHITE);
    g2d.fillRect(0, 0, getWidth(), getHeight());
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
    g2d.setColor(cor);
    for(Point ponto : pontos){
        g2d.fillOval(ponto.x - metade, ponto.y - metade, tamanho, tamanho);
    }
}
```

Linguagem Java

Eventos de clique e arrastar com o mouse são tratados por métodos das interfaces.

Métodos sem uso, para a aplicação em questão não será necessário tratar os demais eventos.

```
@Override
public void mouseDragged(MouseEvent e) {
    pontos.add(e.getPoint()); repaint();
}
@Override
public void mousePressed(MouseEvent e) {
    pontos.add(e.getPoint()); repaint();
}
@Override
public void mouseMoved(MouseEvent e) {}
@Override
public void mouseClicked(MouseEvent e) {}
@Override
public void mouseEntered(MouseEvent e) {}
@Override
public void mouseExited(MouseEvent e) {}
@Override
public void mouseReleased(MouseEvent e) {}
```

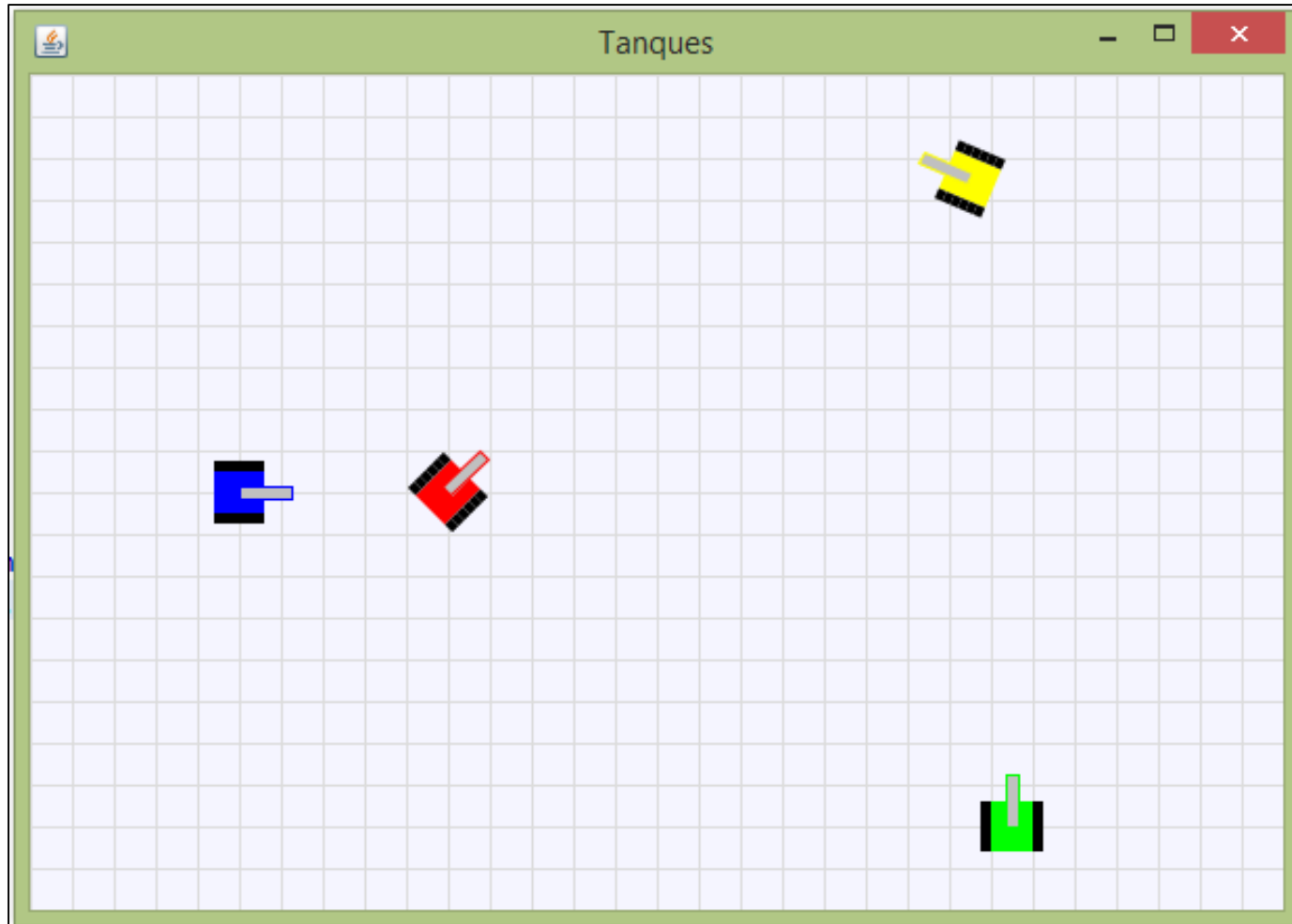
Linguagem Java

```
public class Janela extends JFrame{  
    public Janela(){  
        Rabisco rab1 = new Rabisco(Color.RED);  
        Rabisco rab2 = new Rabisco(Color.BLUE);  
        rab1.setBorder(BorderFactory.createLineBorder(Color.RED));  
        rab2.setBorder(BorderFactory.createLineBorder(Color.BLUE));  
        getContentPane().setLayout(new GridLayout(1,2));  
        getContentPane().add(rab1);  
        getContentPane().add(rab2);  
        pack();  
        setVisible(true);  
        setSize(600, 300);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
    public static void main(String[] args){  
        new Janela();  
    }  
}
```



Linguagem Java

Utilizando Janelas gráficas para criar jogos!



Linguagem Java

Vamos começar pela construção da classe tanque. Parte 1

```
public class Tanque {  
    private double x,y;  
    private double angulo;  
    private double velocidade;  
    private Color cor;  
    private boolean estaAtivo;  
    public Tanque(int x, int y, int a, Color cor){  
        this.x = x; this.y = y; this.angulo = 90-a;  
        this.cor = cor; velocidade = 0;  
        this.estaAtivo = false;  
    }  
}
```

As variáveis x,y armazenam a coordenada do tanque. A variável ângulo é responsável pela posição do tanque na tela. Já a Variável velocidade é quem dita a velocidade de movimento do tanque.

Linguagem Java

Ainda na construção do tanque. Parte 2

```
public void aumentarVelocidade(){
    velocidade++;
}
public void girarHorario(int a){
    angulo += a;
}
public void girarAntiHorario(int a){
    angulo -= a;
}
public void mover(){
    x = x + Math.sin(Math.toRadians(angulo)) * velocidade;
    y = y - Math.sin(Math.toRadians(angulo)) * velocidade;
}
public void setEstaAtivo(boolean estaAtivo){
    this.estaAtivo = estaAtivo;
}
```

Métodos responsáveis por tratar e/ou disparar os eventos de movimento do tanque.

Linguagem Java

Método de desenho da Figura Tanque. Parte 3

```
public void draw(Graphics2D g2d){
    //Armazenamos o sistema de coordenadas original.
    AffineTransform antes = g2d.getTransform();
    //Criamos um sistema de coordenadas para o tanque.
    AffineTransform depois = new AffineTransform();
    depois.translate(x, y);
    depois.rotate(Math.toRadians(angulo));
    //Aplicamos o sistema de coordenadas.
    g2d.transform(depois);
    //Desenhamos o tanque. Primeiro o corpo
    g2d.setColor(cor);
    g2d.fillRect(-10, -12, 20, 24);
    //Agora as esteiras
    for(int i = -12; i <= 8; i += 4){
        g2d.setColor(Color.LIGHT_GRAY);
        g2d.fillRect(-15, i, 5, 4);
        g2d.fillRect(10, i, 5, 4);
        g2d.setColor(Color.BLACK);
        g2d.fillRect(-15, i, 5, 4);
        g2d.fillRect(10, i, 5, 4);
    }
}
```

Linguagem Java

Método de desenho da Figura Tanque. Parte 5

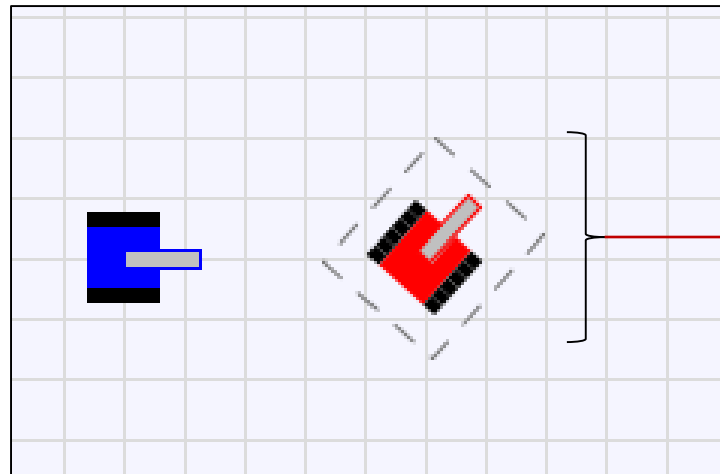
```
//O canhão  
g2d.setColor(Color.LIGHT_GRAY);  
g2d.fillRect(-3, -25, 6, 25);  
g2d.setColor(cor);  
g2d.drawRect(-3, -25, 6, 25);  
//Se o tanque estiver ativo  
//Desenhamos uma margem  
if(estaAtivo){  
    g2d.setColor(new Color(120,120,120));  
    Stroke linha = g2d.getStroke();  
    g2d.setStroke(new BasicStroke(1f,BasicStroke.CAP_ROUND,  
        BasicStroke.JOIN_ROUND,0,  
        new float[]{8},0));  
    g2d.drawRect(-24, -32, 48, 55);  
    g2d.setStroke(linha);  
}  
//Aplicamos o sistema de coordenadas  
g2d.setTransform(antes);  
}
```

Linguagem Java

Se o tanque estiver ativo (selecionado) desenhar uma linha pontilhada na margem em volta do tanque. Parte 5

```
public Shape getRectEnvolvente(){  
    AffineTransform at = new AffineTransform();  
    at.translate(x,y);  
    at.rotate(Math.toRadians(angulo));  
    Rectangle rect = new Rectangle(-24,-32,48,55);  
    return at.createTransformedShape(rect);  
}
```

Pronto. Agora vamos
criar a arena de
tanques!



Linguagem Java

A classe arena é um JComponente. A arena irá guardar a lista de tanques para o jogo. Parte 1

```
public class Arena extends JComponent
    implements MouseListener, ActionListener{
    private int largura, altura;
    private HashSet<Tanque> tanques;
    private Timer contador;
    public Arena(int largura, int altura){
        this.largura = largura;
        this.altura = altura;
        tanques = new HashSet<Tanque>();
        addMouseListener(this);
        contador = new Timer(500, this);
        contador.start();
    }
```

HashSet é uma estrutura de dados, usada para armazenar objetos. A classe Timer provoca uma atualização, um repaint.

Linguagem Java

Ainda na construção da Arena. Parte 2

```
public void adicionaTanque(Tanque t){  
    tanques.add(t);  
}  
public Dimension getMaximumSize(){  
    return getPreferredSize();  
}  
public Dimension getMinimumSize(){  
    return getPreferredSize();  
}  
public Dimension getPreferredSize(){  
    return new Dimension(largura, altura);  
}
```

Os métodos acima servem para adicionar um tanque a lista de tanques na arena e também para recuperar o tamanho máximo da arena, importante para na hora de desenhar cada tanque.

Linguagem Java

Todo componente precisa de um método que aponte como ele deve ser desenhado na janela gráfica. Parte 3

```
protected void paintComponent(Graphics g){  
    super.paintComponent(g);  
    Graphics2D g2d = (Graphics2D)g;  
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
        RenderingHints.VALUE_ANTIALIAS_ON);  
    g2d.setColor(new Color(245,245,255));  
    g2d.fillRect(0,0,largura,altura);  
    g2d.setColor(new Color(220,220,220));  
    for(int _largura=0;_largura<=largura;_largura+=20)  
        g2d.drawLine(_largura,0,_largura,altura);  
    for(int _altura=0;_altura<=altura;_altura+=20)  
        g2d.drawLine(0,_altura,largura,_altura);  
    // Desenhemos todos os tanques  
    for(Tanque t:tanques) t.draw(g2d);  
}
```

As linhas serão desenhadas de maneira a gerar um grid.

Linguagem Java

```
public void mouseClicked(MouseEvent e){
    for(Tanque t:tanques)
        t.setEstaAtivo(false);
    for(Tanque t:tanques){
        boolean clicado = t.getRectEnvolvente().contains(e.getX(),e.getY());
        if (clicado){
            t.setEstaAtivo(true);
            switch(e.getButton()){
                case MouseEvent.BUTTON1: t.girarAntiHorario(3); break;
                case MouseEvent.BUTTON2: t.aumentarVelocidade(); break;
                case MouseEvent.BUTTON3: t.girarHorario(3); break;
            }
            break;
        }
    }
    repaint();
}

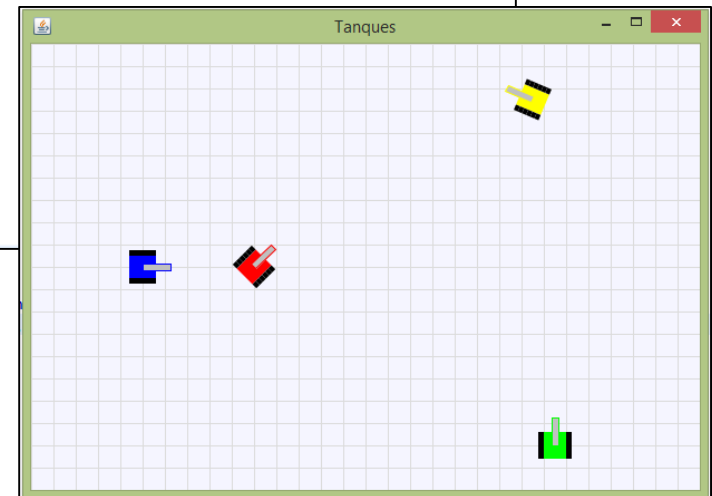
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void actionPerformed(ActionEvent e){
    for(Tanque t:tanques)
        t.mover();
    repaint();
}
```

É preciso definir quais eventos serão tratados no jogo. Parte 4

Linguagem Java

Para finalizar, deve-se lembrar que a Arena é um JComponent, logo ela precisa ser adicionada a uma Janela gráfica. Parte 5

```
public static void main(String args[]){  
    Arena arena = new Arena(600,400);  
    arena.adicionaTanque(new Tanque(100,200,0,Color.BLUE));  
    arena.adicionaTanque(new Tanque(200,200,45,Color.RED));  
    arena.adicionaTanque(new Tanque(470,360,90,Color.GREEN));  
    arena.adicionaTanque(new Tanque(450,50,157,Color.YELLOW));  
    JFrame janela = new JFrame("Tanques");  
    janela.getContentPane().add(arena);  
    janela.pack();  
    janela.setVisible(true);  
    janela.setDefaultCloseOperation(3);  
}
```



Perguntas?



Linguagem Java

Programação seqüencial:

A primeira forma de exceção de software a ser desenvolvida foi a forma seqüencial.

```
public static void main(String[] args) {  
    int contagem = 0;  
    for(int i = 0; i < 10; i++){  
        contagem++;  
        System.out.println(contagem);  
    }  
}
```

Na programação seqüencial a execução das instruções do programa acontecem de maneira contígua entre as linhas do programa, começando da primeira linha até a última.

Linguagem Java

O que é um Processo?

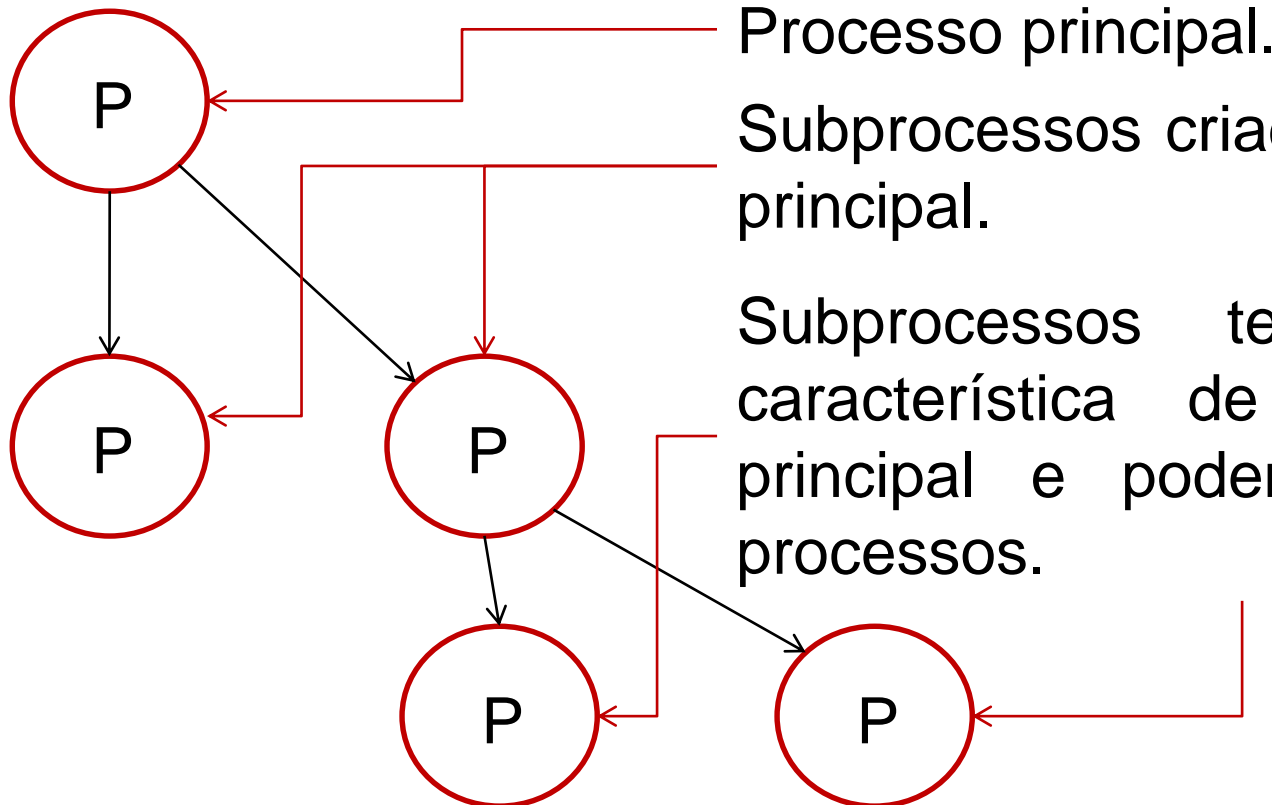
Considera-se um processo, a execução de determinado software pelo processador. O processo possui uma estrutura complexa que contém além do programa em seu formato executável todas as informações necessárias à execução e ao controle da execução.

O que é um Programa?

Um programa é uma entidade passiva. Deve ser visto como um conteúdo de um arquivo em disco. O programa contém as instruções que serão executadas por um processo, descritas em uma linguagem de programação.

Linguagem Java

Em sistemas operacionais modernos, juntamente com processadores com mais de um núcleo, é possível que um determinado processo gere um ou mais subprocessos que serão executados de maneira concorrente.



Processo principal.

Subprocessos criados do processo principal.

Subprocessos tem a mesma característica de um processo principal e podem gerar novos processos.

Linguagem Java

Thread (um processo leve ou simples): Uma unidade básica de utilização de CPU.

Considera-se que cada subprocesso de um processo pai, uma thread. O uso de thread pode reduzir o tempo total de execução de um programa.

Uma thread é executada por vez, contudo na programação concorrente existe a alternância de execuções de threads no processador ao longo do tempo, dessa forma tem-se a impressão que o processamento ocorre em paralelo.

Execução concorrente não significa execução simultânea, o que acontece é a distribuição de processos em mais de um núcleo do processador, ou entre CPUs.

Linguagem Java

Uma forma de se implementar uma Thread e fazer uma herança a classe Thread. Ou então deve-se implementar a interface Runnable.

```
public class Paralela extends Thread{  
    @Override  
    public void run(){  
        System.out.println("processamento...");  
    }  
}
```

```
public class Paralela implements Runnable{  
    @Override  
    public void run(){  
        System.out.println("processamento...");  
    }  
}
```


Linguagem Java

```
public static void main(String[] args) {  
    Paralela p1 = new Paralela("A");  
    Paralela p2 = new Paralela("B");  
    p1.contagem();  
    p2.contagem();  
}
```

Processamento seqüencial. O método contagem de p1 irá processar primeiro, e após ele o contagem de p2.

```
public class Paralela{  
    private String nome;  
    public Paralela(String nome){  
        this.nome = nome;  
    }  
    public void contagem(){  
        int contagem = 0;  
        for(int i = 1; i <= 10; i++){  
            contagem++;  
            System.out.println("Contagem "+nome+": "+contagem);  
        }  
    }  
}
```

Linguagem Java

```
public class Paralela extends Thread{  
    private String nome;  
    public Paralela(String nome){  
        this.nome = nome;  
    }  
    public void contagem(){  
        int contagem = 0;  
        for(int i = 1; i <= 10; i++){  
            contagem++;  
            System.out.println("Contagem "  
                               +nome+": "+contagem);  
        }  
    }  
    @Override  
    public void run(){  
        contagem();  
    }  
}
```

O método start()
inicia o
processamento
concorrente da
thread.

```
public static void main(String[] args) {  
    Paralela p1 = new Paralela("A");  
    Paralela p2 = new Paralela("B");  
    p1.start();  
    p2.start();  
}
```

Linguagem Java

```
public class Paralela implements Runnable{  
    private String nome;  
    public Paralela(String nome){  
        this.nome = nome;  
    }  
    public void contagem(){  
        int contagem = 0;  
        for(int i = 1; i <= 10; i++){  
            contagem++;  
            System.out.println("Contagem "  
                               +nome+": "+contagem);  
        }  
    }  
    @Override  
    public void run(){  
        contagem();  
    }  
}
```

O método start()
inicia o
processamento
concorrente da
thread.

```
public static void main(String[] args) {  
    Paralela p1 = new Paralela("A");  
    Paralela p2 = new Paralela("B");  
    Thread t1 = new Thread(p1);  
    Thread t2 = new Thread(p2);  
    t1.start();  
    t2.start();  
}
```

Linguagem Java

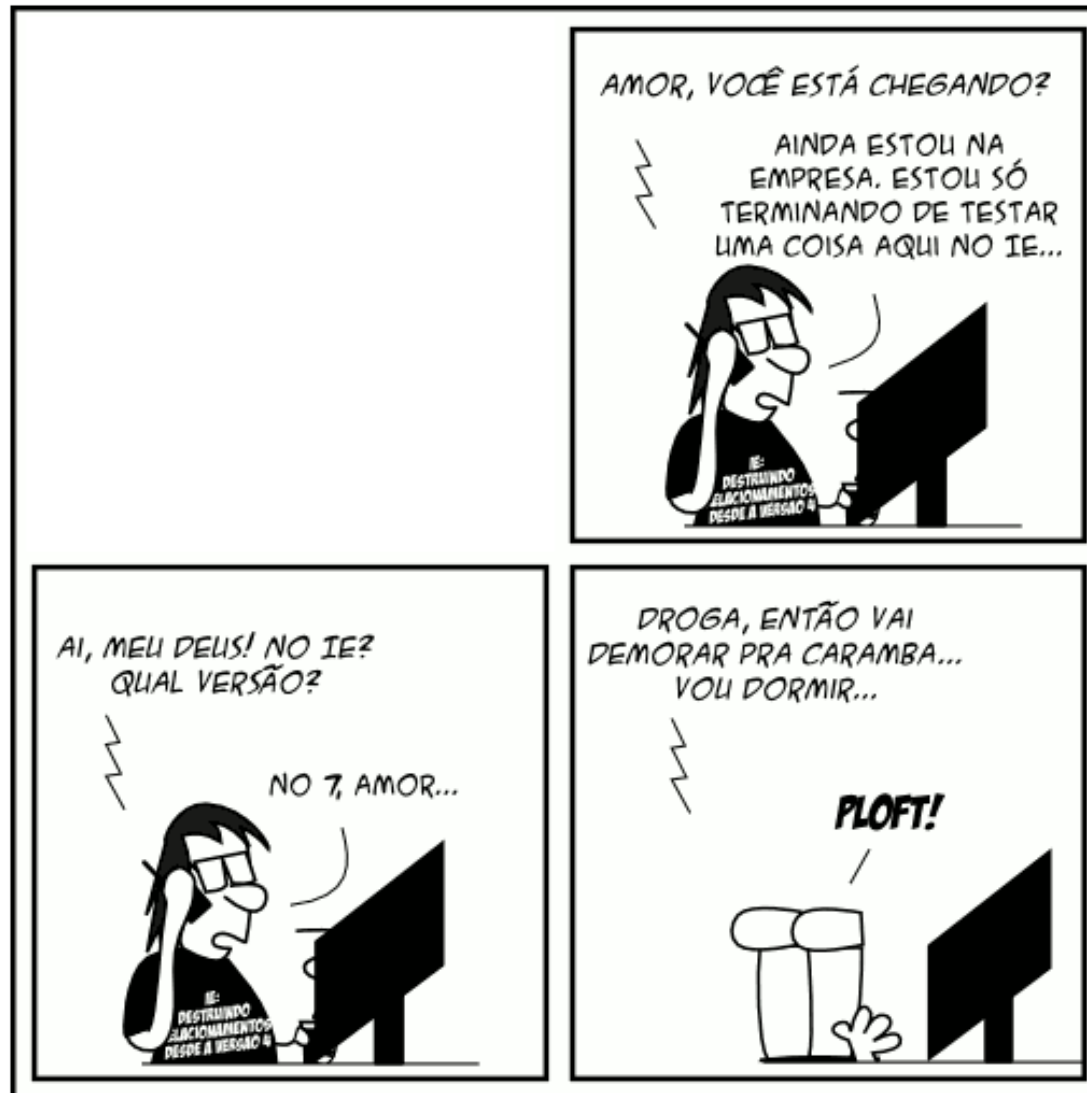
Alguns métodos interessantes da classe Thread.

- **setPriority** (int): Seta a prioridade de uma Thread;
- **setName** (String): Seta o nome de uma Thread;
- **sleep** (int milisegundos): Faz a Thread dormir por um certo tempo;

Em java a prioridade de execução de uma thread é um valor inteiro entre 1 e 10. O valor default para a prioridade de execução é 5.

Deve-se lembrar que para uma chamada ao método `sleep()` tem-se a interrupção da thread que está executando.

Perguntas?



Linguagem Java

Trabalhando com coleções de dados.

- Java tem uma API específica para manipulação de dados (objetos), denominada Collection.
- É comum usarmos um objeto que armazene vários outros.
- A API Collections provê interfaces e classes para coleções que fica no pacote: *java.util*
- Por meio destas coleções, é possível representar diferentes tipos de estruturas, como: **listas, conjuntos e mapas.**

Linguagem Java

Conceitos:

- Uma coleção **é um objeto** que agrupa vários outros objetos.
- Uma coleção também pode ser chamada de contêiner.
- É uma solução flexível para armazenar objetos, o que facilita bastante a vida do desenvolvedor.
- Numa coleção, **a quantidade armazenada de objetos não é fixa**, como ocorre com arrays.
- Além disto, seu tamanho aumenta automaticamente conforme se adiciona mais elementos.

Linguagem Java

- As operações que podem ser feitas em coleções variam mas normalmente incluem:
 - Adição de elementos.
 - Remoção de elementos.
 - Acesso aos elementos.
 - Pesquisa de elementos.
 - Indagar sobre atributos.
 - Percorrer os elementos.
 - Verificar o número de elementos.

```
List<String> lista = new ArrayList<String>();  
lista.add("Objeto String");  
lista.size();  
lista.remove(0);
```

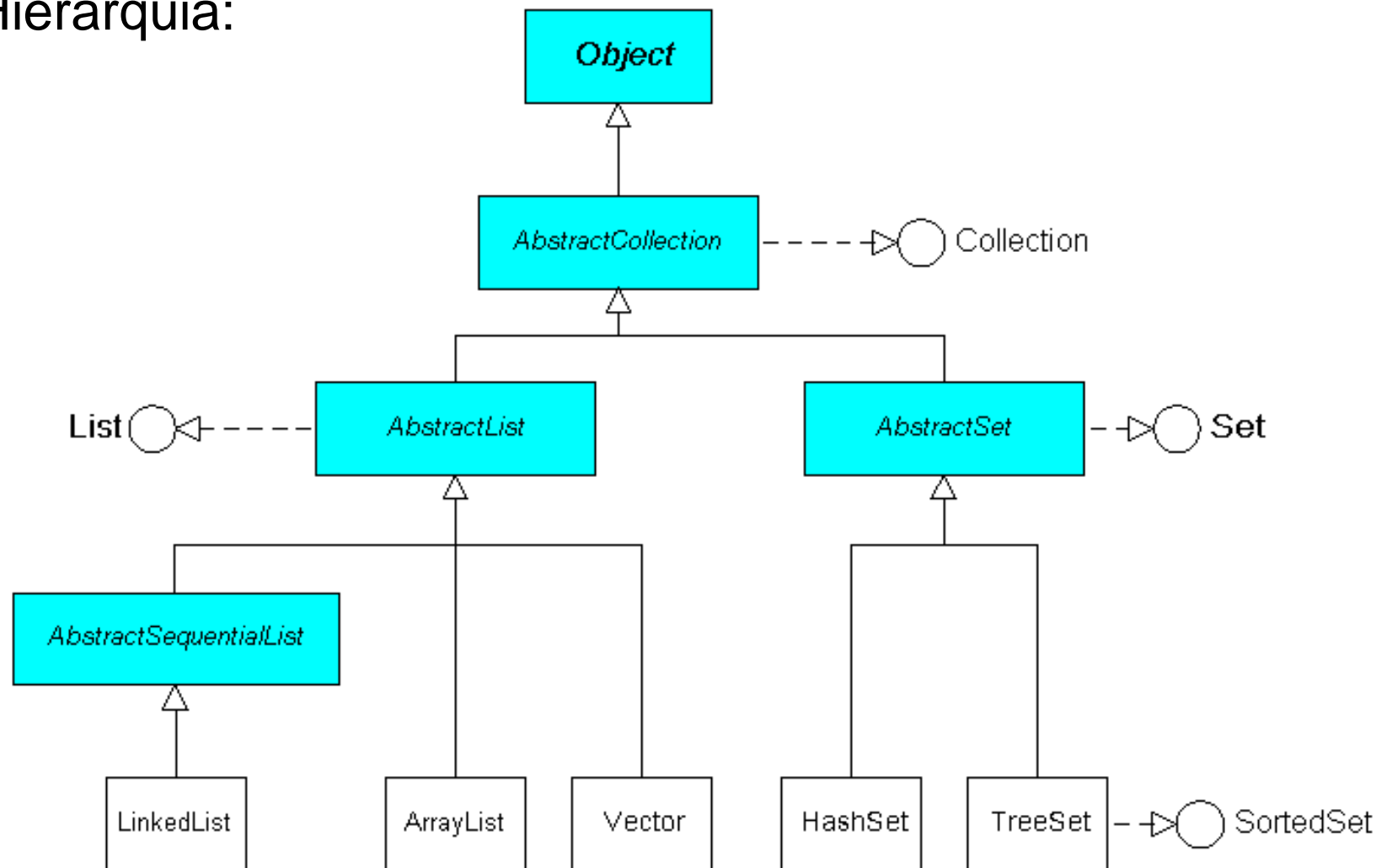

Linguagem Java

Coleções vs Arrays.

- Pode-se usar arrays em lugar de coleções quando:
 - Sabe-se de antemão o número máximo de elementos que será armazenado;
 - Objetiva-se armazenar tipos primitivos ao invés de objetos;
 - Preocupação com o desempenho e consumo de memória, posto que o array é mais econômico. Todavia, na prática isto pouco acontece;
 - Não haverá alterações significativas na estrutura, como inserção ou deleção de elementos.

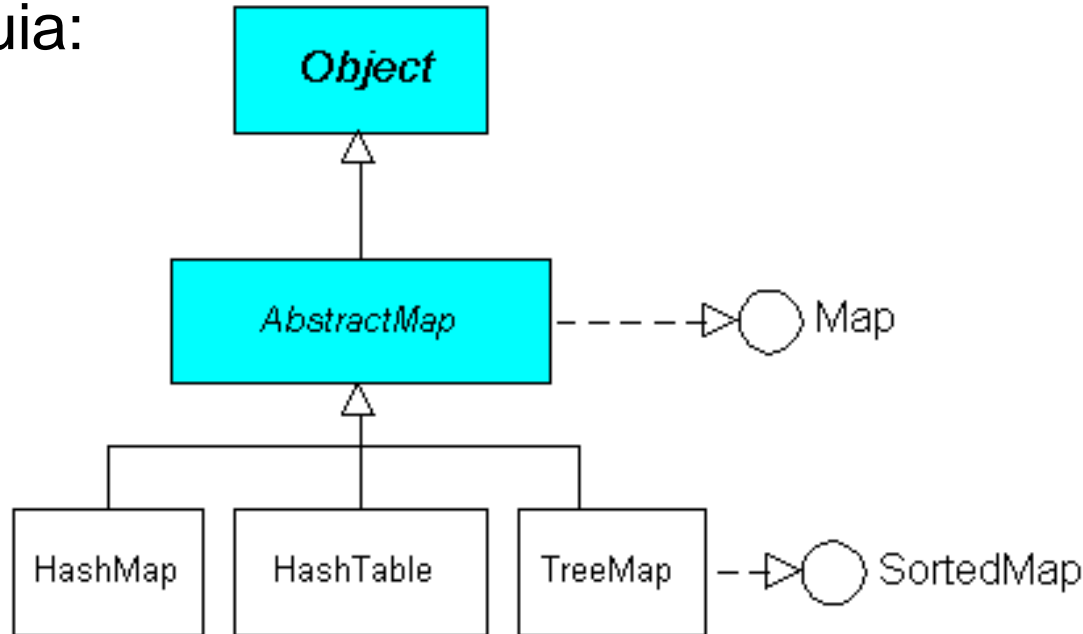
Linguagem Java

Hierarquia:



Linguagem Java

Hierarquia:



A interface **Map** não faz parte da mesma hierarquia das interfaces **set** e **list**. Contudo é considerada como parte da API de coleções da linguagem Java.

Linguagem Java

Sobre a interface List:

- Uma lista é uma coleção de elementos arrumados em uma ordem linear;
- Isto é, cada elemento tem um antecessor (exceto o primeiro) e um sucessor (exceto o último);
- Normalmente implementada como "Array" ou "Lista Encadeada" ;
- É uma coleção **em ordem** (algumas vezes chamada de sequência);
- Isto é, os elementos estão na ordem em que foram naturalmente adicionados à lista.

Linguagem Java

Sobre a interface List:

- A implementação mais utilizada da interface List é a classe ArrayList;
- ArrayList é ideal para acesso randômico, é, portanto, rápido para pesquisa;
- ArrayList não é sincronizada, isto é não tem segurança para programação concorrente (threads).
- Vector – ideal para acesso randômico, porém é sincronizado, logo, é mais lento;
- LinkedList – ideal para acesso sequencial; É apropriado para inserção de elementos ao seu final, ou seja, ideal para pilhas e filas, mas ruim para o acesso posicional.

Linguagem Java

Sobre a interface List:

- Principais métodos:
 - **void add (Object o):** adiciona objeto na posição ao final da lista;
 - **void add (int index , Object o):** adiciona objeto na posição e empurra os objetos para direita;
 - **void get (int index):** recupera objeto pelo índice;
 - **int indexOf (Object o):** procura objeto e retorna o índice da primeira ocorrência;
 - **void set (int index, Object o):** grava objeto na posição indicada, apagando o objeto que estava naquela posição;
 - **int size ():** retorna o número de objetos na lista;
 - **Iterator listIterator():** retorna um objeto Iterator, que possibilita iterar sob os objetos contidos na lista.

Linguagem Java

Sobre a interface Set:


- As classes que implementam esta interface **não aceitam elementos repetidos**;
- Observe que, no conjunto, **não há noção de “ordem natural da inserção dos elementos”**;
- Caso seja adicionado ao conjunto um elemento repetido, o elemento equivalente dará lugar ao novo elemento adicionado;
- Pode possuir um único elemento **“null”**.
- **Existem duas principais implementações:**
 - **HashSet**: os elementos não ficarão ordenados;
 - **TreeSet**: os elementos ficarão ordenados, independente da ordem que forem adicionados ao TreeSet.

Linguagem Java


Sobre a interface Set:

- Principais métodos da interface:

- add (Object o)
- contains (Object o)
- remove(Object o)
- isEmpty()
- iterator()
- size()
- toArray()



```
Set<String> conjunto = new HashSet<String>();  
conjunto.add("uma string");
```



```
Set<String> ordenados = new TreeSet<String>();  
ordenados.add("8"); ordenados.add("2");  
ordenados.add("5"); ordenados.add("3");  
for(String elemento : ordenados){  
    System.out.println(elemento);  
}
```


Linguagem Java

Sobre a interface Map:

- Objetos Map são semelhantes a arrays, **mas em vez de índices numéricos, usam objetos como chaves;**
- Cada elemento tem um par **(chave, valor);**
- As **chaves são únicas**, porém os valores podem ser duplicados;
- Caso haja uma repetição de chave, o elemento antigo é sobreposto pelo novo;
- A chave é utilizada para achar um elemento rapidamente .

Linguagem Java

- Existem três principais subclasses:
 - **HashMap**: não sincronizado e não ordenado, aceita **null** como chave;
 - É o mais difundido entre as estruturas Maps.
 - **HashTable**: igual ao HashMap, porém **sincronizado** e não aceita **null** como chave;
 - **TreeMap**: não sincronizado mas é ordenado.
- Principais métodos:
 - void put (Object key, Object value): adiciona um objeto ao mapa;
 - Object get (Object key): recupera um objeto;
 - Set keySet(): retorna um Set das chaves.

Linguagem Java

```
Map<String,String> mapa = new HashMap<String,String>();  
mapa.put("c1", "João");  
mapa.put("c2", "Maria");  
mapa.put(null, "Você");  
System.out.println(mapa.get("c1"));  
System.out.println(mapa.get(null));
```

```
Map<String,String> mapa = new Hashtable<String,String>();  
mapa.put("c1", "João");  
mapa.put("c2", "Maria");  
mapa.put(null, "Você");  
System.out.println(mapa.get("c1"));  
System.out.println(mapa.get(null));
```

```
Exception in thread "main" java.lang.NullPointerException  
    at java.util.Hashtable.hash(Unknown Source)  
    at java.util.Hashtable.put(Unknown Source)  
    at br.com.fatec.Inicio.main(Inicio.java:11)
```

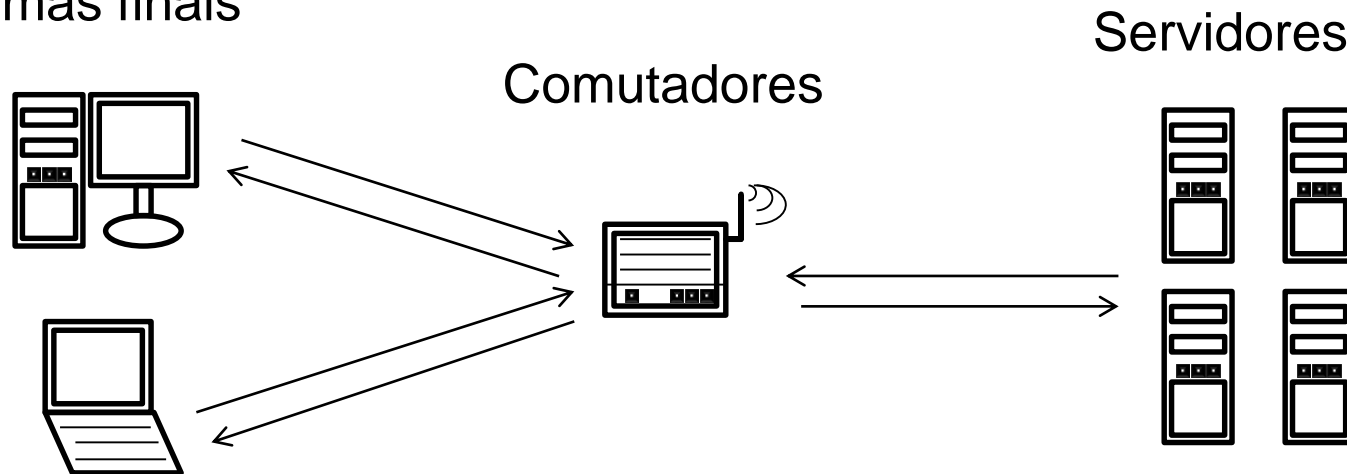


Linguagem Java

Sistemas finais

Hoje a Internet alcança aparelhos como computadores de mesa, computadores portáteis, telefones celulares, TVs, automóveis, equipamentos de sensoramento ambiental, sistemas domésticos elétricos, câmeras e sistemas de segurança.

Sistemas finais

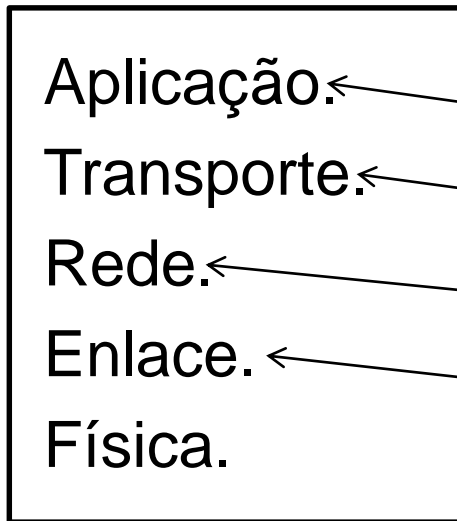


Linguagem Java

Arquitetura em camadas.

5 Camadas.

Cada protocolo pertence a uma camada.



Mensagem.

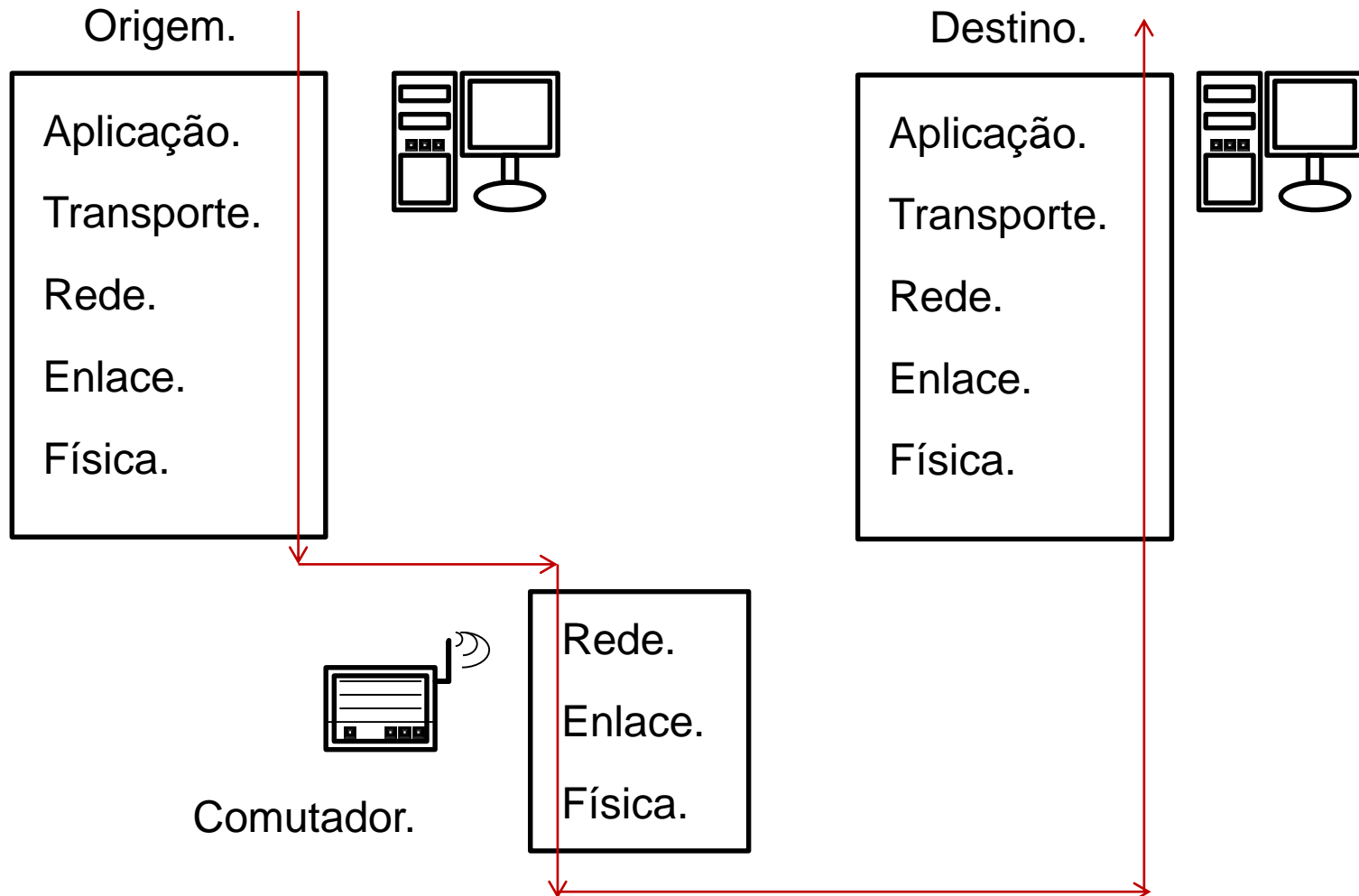
Segmento.

Datagrama.

Quadro.

Os projetistas de rede organizam software e hardware que implementam protocolos em camadas.

Linguagem Java



Linguagem Java

Classe responsável por fazer uma conexão. Existem outros atributos que podem ser passados para o método construtor da classe.

```
Socket conexao = new Socket("192.168.1.0", 21);
```

As informações essenciais para realizar a conexão é o IP e a porta de conexão.

```
ServerSocket socketServidor = new ServerSocket(1234);
```

A classe ServerSocket é usada para se iniciar uma conexão do lado do servidor. Por isso, único parâmetro passado é a porta.

Linguagem Java

```
public class Servidor {  
    private ServerSocket socketServidor;  
    public Servidor () throws IOException{  
        System.out.println("Iniciando servidor");  
        socketServidor = new ServerSocket(1234);  
    }  
    public void iniciar() throws Exception{  
        while(true){  
            Socket socketEscuta = socketServidor.accept();  
            InputStreamReader streamReader =  
                new InputStreamReader(socketEscuta.getInputStream());  
            BufferedReader reader = new BufferedReader (streamReader);  
            String textoEnviado = reader.readLine();  
            System.out.println(textoEnviado);  
            reader.close();  
        }  
    }  
}
```

Código servidor. Não é possível executar um programa servidor e outro cliente na mesma aplicação exceto se usar threads.

Linguagem Java

```
public class Cliente {  
    private Socket socketCliente;  
    public Cliente() throws Exception{  
        System.out.println("Fazendo conexão");  
        socketCliente = new Socket("127.0.0.1", 1234);  
    }  
    public void conectarEnviar() throws Exception{  
        PrintWriter escritor =  
            new PrintWriter(socketCliente.getOutputStream());  
        System.out.println("Enviando...");  
        escritor.write("Texto enviado para o servidor");  
        escritor.close();  
    }  
}
```

Código Cliente. Não é possível executar um programa servidor e outro cliente na mesma aplicação exceto se usar threads.

Linguagem Java

Internacionalização de projetos.

- Projetos podem ser internacionalizados para outras localidades.
- Java tem uma API específica para manipulação de datas, números e moeda.
- Com isso, pode-se resolver vários problemas.
- relacionados à formatação de datas, números e moedas de diferentes países sem ter que mudar o código fonte.

Principais classes envolvidas:

```
Date date = new Date();
```

```
Calendar calendar = Calendar.getInstance();
```

```
Locale locale = new Locale("pt","BR");
```

A classe Date

- A classe Date atualmente é pouco utilizada, pois a maioria dos seus métodos foram depreciados (caíram em desuso)
- Não fornece facilidades para manipular datas, bem como suporte à internacionalização
- Exemplo de utilização desta classe:
 - `Date data = new Date();`
 - `System.out.println(data);`
 - `System.out.println(data.getTime());`
 - Saída: Mon Nov 10 08:56:10 BR 2010
 - 1291034753805 // tempo em milisegundos
 - Esse tempo é contado a partir de 01/01/1970

Classe Calendar

- É uma classe mais poderosa do que a `Date`, criada recentemente para prover suporte à internacionalização
- Além disto, tem outros mecanismos que provêem facilidades para manipulação de datas.
- A classe **Calendar** sendo uma classe abstrata não pode ser instanciada com o operador *new*
- Por isso, deve ser criada utilizando um operador estático sobrecarregado **getInstance()**
 - Ex: `Calendar c = Calendar.getInstance();`

Classe Calendar

- A classe **Calendar** também provê várias constantes que são úteis para manipulação de datas, tais como:
- Ano: `Calendar.YEAR`
- Mês: `Calendar.MONTH`
- Dia:
 - `Calendar.DAY_OF_MONTH`
 - `Calendar.DAY_OF_WEEK`
 - `Calendar.DAY_OF_YEAR`
- Hora: `Calendar.HOUR`
- Segundos: `Calendar.SECOND`
- Milisegundos: `Calendar.MILLISECOND`

Classe Calendar

- Com a classe Calendar, podemos configurar uma data de duas formas:

```
Calendar ca = Calendar.getInstance();  
ca.set(Calendar.YEAR, 2011);  
ca.set(Calendar.MONTH, 0);  
ca.set(Calendar.DAY_OF_MONTH, 1);
```

Ou de uma forma mais simples:

```
Calendar c = Calendar.getInstance();  
c.setTime(new Date());
```

Classe Calendar

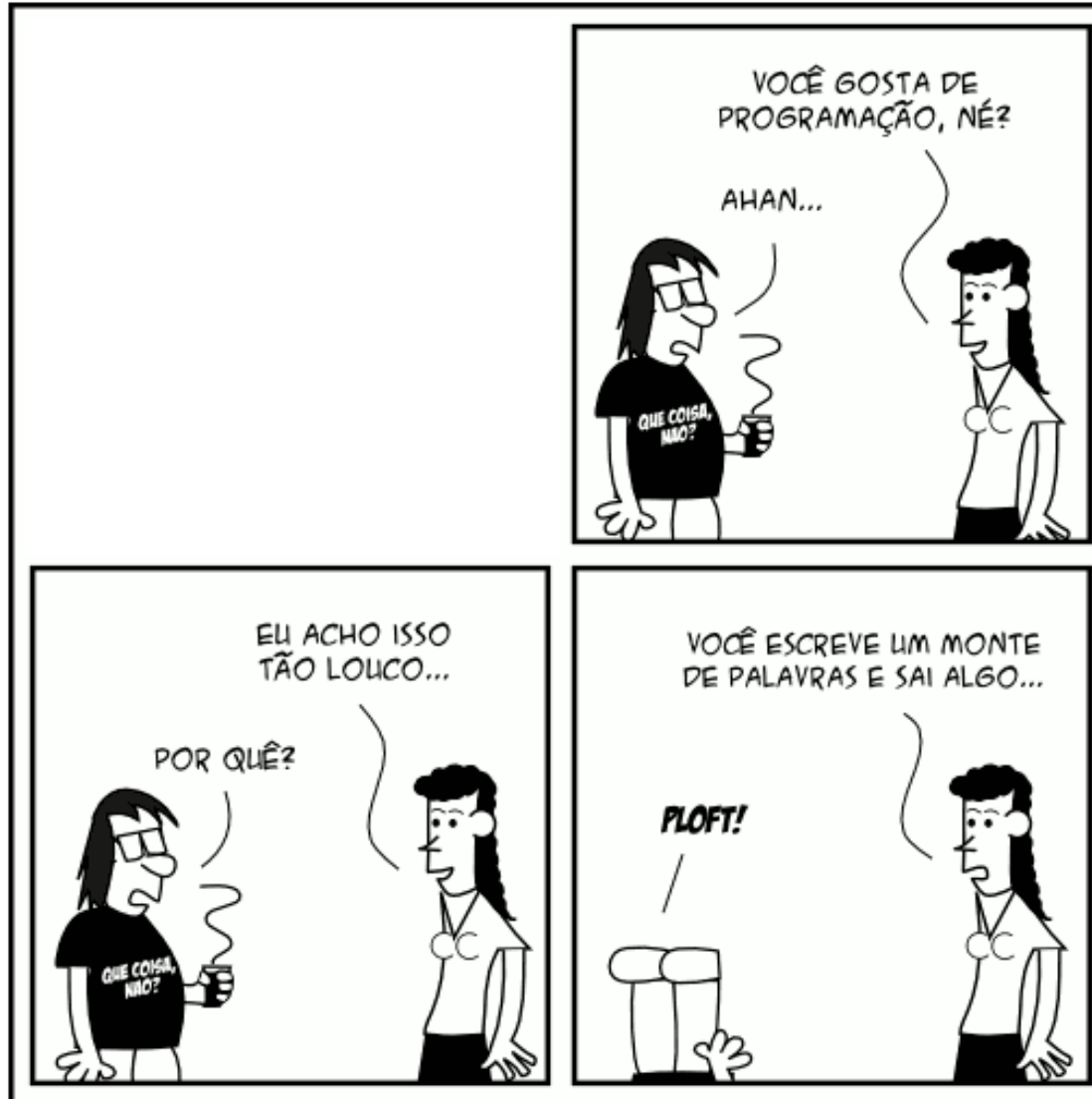
- Com a data devidamente configurada pode-se realizar diversas operações com a data
- Podemos adicionar ou subtrair um dia, hora, mês ou ano, utilizando o método **add()**

```
Calendar d = Calendar.getInstance(pt); // cria o calendario assim

c.setTime(data); // aqui vc atribui a data ao Calendar, e a partir daqui vc pode manipular
c.add(Calendar.MONTH, 3); // adiciona 3 meses na data atual
c.add(Calendar.YEAR, 2); // adiciona 2 anos na data atual
c.add(Calendar.HOUR, 4); // adiciona 4 horas
c.getTime() // retorna a data
```

Caso o número passado como parâmetro pelo método *add()* seja negativo, será subtraído ao invés de adicionar um determinado intervalo de tempo à data corrente

Perguntas?



Classe Locale

- A classe *Locale* é útil para configurar qual localidade seu projeto será executado: Brasil, EUA, China, Inglaterra,...
- Com isso, é possível obter formatações padrões destes países em relação à moeda, números e datas
- A definição de um objeto desta classe de forma isolada não contribui para formatação, e deve ser usada em conjunto com outras classes da API Java
 - Ex: `DateFormat` e `NumberFormat`
- Lembrando que a classe *Locale* fica no pacote `java.util`
- É possível instanciar um objeto da classe *Locale* da seguinte maneira:
 - `Locale brasil = new Locale("pt", "BR");` // língua e país
 - `Locale americano = new Locale("en", "US");` // língua e país

Classe Locale

- Existe uma tabela padrão com todos países e seus respectivos códigos
- Também há constantes padrão para representar *Locales* de países mais conhecidos, por exemplo:
 - Locale.US
 - Locale.UK
 - Locale.GERMAN
- Desta forma, evita-se criar um objeto do tipo Locale, tal como Locale.US
 - Ex: `Locale eua = new Locale("en", "US");`

Classe DateFormat

- A classe **DateFormat** nos fornece uma maneira simples de formatar datas
- Também permite a internacionalização a partir da classe **Locale**
- Detém as seguintes constantes que definem o estilo de formatação das datas:
 - `DateFormat.SHORT` // ex. *03/04/10*
 - `DateFormat.MEDIUM` // ex. *03/04/2010*
 - `DateFormat.LONG` // ex. *3 de Abril de 2010*
 - `DateFormat.FULL` // ex. *Sábado, 3 de Abril de 2010*

Classe DateFormat

Estilo
formatação



```
Locale brasil = new Locale("pt", "Br");  
DateFormat dfBrasil = DateFormat.getDateInstance(DateFormat.LONG, brasil);  
DateFormat dfBrasil2 = DateFormat.getDateInstance(DateFormat.SHORT, brasil);  
System.out.println("Hoje no Brasil: " + dfBrasil.format(hoje));  
System.out.println("Hoje no Brasil: " + dfBrasil2.format(hoje));
```

Locale



Date hoje = new Date();

Saída:

Hoje no Brasil: 29 de Novembro de 2010

Hoje no Brasil: 29/11/10

Classe DateFormat

Estilo
formatação

```
Locale us = new Locale("en", "US");  
DateFormat df2 = DateFormat.getDateInstance(DateFormat.SHORT, Locale.US);  
DateFormat dfusa2 = DateFormat.getDateInstance(DateFormat.LONG, us);  
System.out.println("Today in USA: " + df2.format(hoje));  
System.out.println("Today in USA: " + dfusa2.format(hoje));
```

Saída:

Today in USA: 11/20/10

Today in USA: November 20, 2010

Locale

Classe NumberFormat

- Esta classe foi criada para efetuar a manipulação de números, de acordo com a localidade (internacionalização)
- Com isso é possível obter um formato padrão quanto à estrutura da moeda de acordo com o país especificado pela classe Locale

```
public static void main(String[] args) {  
  
    Locale brasil = new Locale("pt", "BR");  
    double pagamento = Math.random() * 10000;  
    NumberFormat nf = NumberFormat.getCurrencyInstance(brasil);  
    System.out.println("Seu pagamento é " + nf.format(pagamento));  
    NumberFormat nfus = NumberFormat.getCurrencyInstance(Locale.US);  
    System.out.println("Seu pagamento é " + nfus.format(pagamento));  
  
}
```

Perguntas?

