Programação Orientada à Objetos (POO)

CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. João Paulo Aramuni

UNIVERSIDADE FUMEC

Sumário

* **Classes e Objetos**
  * Classes e Objetos: introdução e definição
  * Atributos
  * Métodos
  * Acesso aos atributos e métodos
  * Encapsulamento
  * Construtores
  * Pacotes

Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * Se a linguagem é orientada a objetos, precisamos de uma noção básica de objetos e classes para entender como ela funciona:
    * **Objeto**: É possível definir um objeto como algo que tenha atributos que serão manipulados e operações que serão invocadas.
      * Uma **caneta**, suas propriedades seriam a cor da tinta, peso e dimensões. Suas operações seriam escrever, rabiscar, etc.
      * Um **funcionário**, com as propriedades nomes, idade, endereço, etc. Suas operações seriam calcular bonificações, calcular horas extras, etc.
    * **Classe**: Podemos definir uma classe como sendo as especificações (molde) de um objeto. Dentro da classe teremos todas as operações. Todas as especificações: comportamentos (comandos) e atributos (declarações) dos objetos estão na definição da classe.

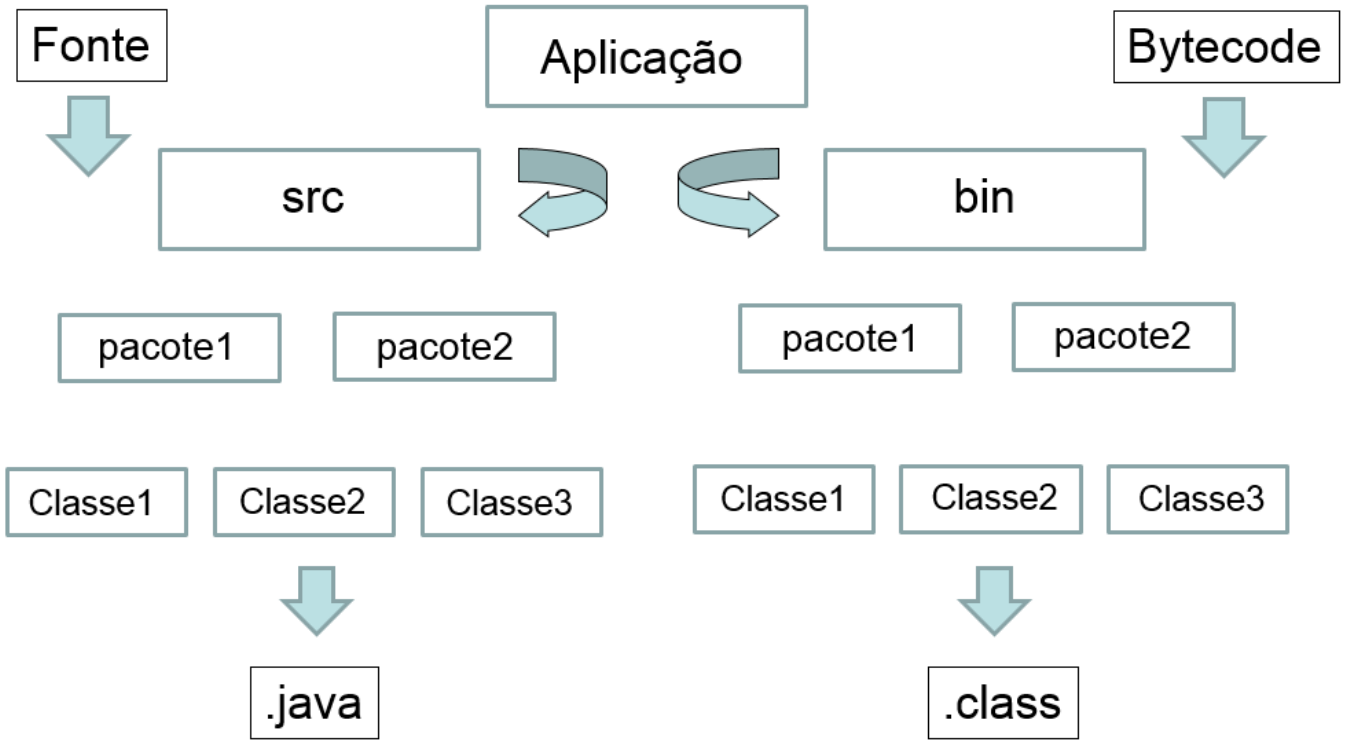
Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * A classe é fundamental na programação orientada a objetos.
  * Exemplo:

```
class Classe1 {
...
}
```

  * Nome do arquivo: **Classe1.java**

* **Classes e Objetos: introdução e definição**
  * **Padrão de Organização da Aplicação**



Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * **Programas em Java**
    * Um arquivo fonte é constituído por um conjunto de classes
      * Normalmente um arquivo contém apenas uma classe
    * Classes definem Tipos Abstratos de Dados
      * Composta de variáveis (**atributos**), funções e procedimentos (**métodos**).

Classes e Objetos

- **Classes e Objetos: introdução e definição**
  - **Definição de Classe em Java**
    - Um arquivo fonte é constituído por um conjunto de classes
      - Normalmente um arquivo contém apenas uma classe
    - Classes definem Tipos Abstratos de Dados
      - Composta de variáveis (**atributos**), funções e procedimentos (**métodos**).
    - A classe principal em um arquivo fonte é qualificada pela cláusula ***public*** (que a torna **visível** para as outras classes)
      - Somente uma classe é public em um arquivo.
      - A classe public deve ter o mesmo nome do arquivo.

Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * **Definição de Classe em Java**

```
public class  Nome_da_Classe
{
        // Declaração dos Atributos
         tipo  nome_do_atributo  [= valor];
        // ...  outros atributos
        // Implementação dos Métodos
        tipo_de_retorno  nome_do_método
        (lista_de_parâmetros)
        {
                comandos;
        }
        // ...  outros métodos
}
```

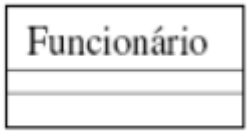
Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * **Definição de Classe em Java**
    * Uma classe é a descrição de um grupo de objetos com propriedades similares (atributos), comportamentos comuns (operações), relacionamentos comuns com outros objetos e semânticas idênticas.
    * Um objeto é uma instância de uma classe.
    * Exemplo:

Classe	Atributos	Operações
Pedido	número	adicionaItem
	data	cancelar
	vendedor	confirmaVenda

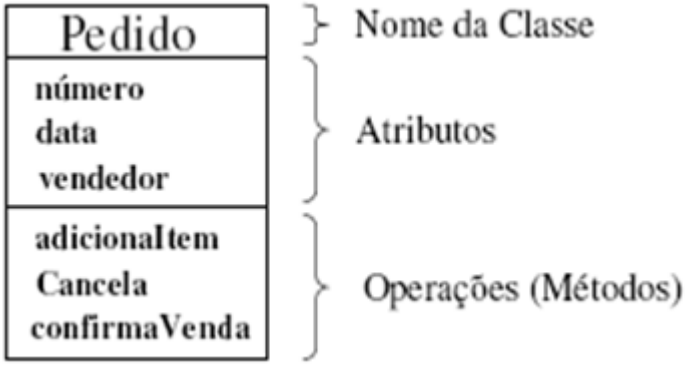
Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * **Definição de Classe em Java**
    * Uma classe é uma definição abstrata de um objeto.
      * Define a estrutura e o comportamento de qualquer objeto da classe.
      * Serve como um padrão para criação de objetos.
      * Exemplo:

Objetos	**Classe**
José Silva Maria Helena João Barros	

Classes e Objetos

* **Classes e Objetos: introdução e definição**
  * **Definição de Classe em Java**
    * Uma classe possui atributos e operações (métodos).
    * Em UML (Unified Modeling Language ou Linguagem Unificada de Modelagem), uma classe é representada utilizando-se um retângulo dividido em três seções:
      * A primeira seção contém o nome da classe.
      * A segunda seção mostra a estrutura (atributos).
      * A terceira seção mostra o comportamento (operações).



Classes e Objetos

* **Atributos**
  * Os atributos definem o conjunto de propriedades de uma classe.
  * Nome de atributos são substantivos simples ou frases substantivas.
  * Cada atributo deve ter uma definição clara e concisa.
  * Cada objeto tem um valor para cada atributo definido na sua classe.
  * Para **definir atributos**, liste as propriedades de uma classe que sejam relevantes para o domínio em questão. Deve-se procurar um compromisso entre **objetividade** (procurar atender a determinado projeto, com o mínimo custo) e **generalidade** (permitir a reutilização da classe em outros projetos).

Classes e Objetos

- **Atributos**
  - Um atributo é definido por:
    - **nome**: um identificador para o atributo.
    - **tipo**: o tipo do atributo (inteiro, real, caractere, etc.)
    - **valor_default**: opcionalmente, pode-se especificar um valor inicial para o atributo.
    - visibilidade: opcionalmente, pode-se especificar o quão acessível é um atributo de um objeto a partir de outros objetos. Valores possíveis são:
      - privativo – nenhuma visibilidade externa;
      - público – visibilidade externa total;
      - protegido – visibilidade externa limitada.

Classes e Objetos

* **Métodos**
  * Uma classe incorpora um conjunto de responsabilidade que **definem** o **comportamento dos objetos na classe.**
  * As responsabilidades de uma classe são executadas por suas **operações.**
  * Uma **operação** é um **serviço** que pode ser **requisitado** por um **objeto** para **obter** um **dado comportamento.**
  * **Operações** também são chamadas de **métodos** (Java) ou funções-membro (C++).

- **Métodos**
  - **Identificação das Operações**
    - Siga os seguintes procedimentos para identificar operações:
      - Liste os papéis e as responsabilidades de cada classe.
      - Defina o conjunto de operações necessário para satisfazer estas responsabilidades.
      - Garanta que cada operação seja primitiva.
        - Uma operação primitiva é uma operação que pode ser implementada apenas usando o que é intrínseco, interno da classe. Exemplo:
          - Adicione um item a um conjunto – operação primitiva.
          - Adicione quatro itens a um conjunto – não primitiva (pode ser implementada com múltiplas chamadas à operação anterior).
      - Garanta a completeza do conjunto de operações.

Classes e Objetos

* **Métodos**
  * **Diretrizes para Escolha de Operações**
    * Cada operação deve realizar uma função simples.
    * O nome deve refletir o resultado da operação, e não as suas etapas.
      * Ex: Use obterSaldo() ao invés de calcularSaldo(). Esta última indica que o saldo deve ser calculado , o que é uma decisão de implementação.
    * Evite excesso de argumentos de entrada e saída, o que geralmente indica a necessidade de partir as operações em outras mais simples.

Classes e Objetos

* **Métodos**
  * Variáveis declaradas dentro de um método são locais ao método.
    * O espaço de memória alocado a elas é liberado automaticamente ao término do método.
  * O método retorna para o ponto onde foi chamado por:
    * **return;**
    * **return** expressão;
  - Os parâmetros dos métodos são sempre passados por **valor** para tipos primitivos e por **referência** para objetos.
    - Arranjos e Objetos passam uma referência. O conteúdo apontado pela referência pode ser modificado pelo método.

Classes e Objetos

* **Métodos**
  * A chamada a um método é dada pelo **nome da classe ou do objeto** seguido pelo operador . (**ponto**) e pelo nome do método.
    * Exemplos:
      * **Math**.sqrt(9);
      * System.out.println(“Total : “ + objTexto.length());
  * Chamadas a métodos locais a uma classe não precisam conter o nome da classe.
  * Métodos podem ser **recursivos** (chamar a si próprio, direta ou indiretamente).

Método Recursivo

```
import utilitarios.Console;

public class FatorialRecursivo {
    public static void main(String[ ] args) {
        while ( true ) {
            int fat = Console.readInt("Fatorial de : “);
                if ( fat == 0 ) {
                    System.out.print("Programa Encerrado.");
                    System.exit(0); // Saída do aplicativo
                }
                    System.out.println("Fatorial de " + fat + "! = " + fat(fat) );
        }
    }

    //  Método recursivo
    private static double fat ( int n ) {
            return  (n > 1)  ?  n * fat ( n-1 ) : 1;
    }
}
```

Classes e Objetos

* **Acesso aos atributos e métodos**
  * Para acessarmos ou modificarmos o conteúdo de um atributo é necessário um método assessor ou um método modificador, respectivamente.

  * Vejamos um exemplo:

Acesso aos atributos e métodos

```
public  int  get ( int campo )
        // Recuperar o campo especificado de uma data.
        YEAR , MONTH , DAY_OF_MONTH , DAY_OF_WEEK,
        HOUR, HOUR_OF_DAY, MINUTE, SECOND, MILLISECOND
        public static void main(String[ ] args) {
        GregorianCalendar hr = new GregorianCalendar(2006,8-1,20,10,20,30);  // 20/08/2006 10:20:30
        System.out.println("Data e hora : " + new SimpleDateFormat("dd/MM/yyyy
        HH:mm:ss").format(hr.getTime()));
        System.out.println("Ano     : " + hr.get (GregorianCalendar.YEAR)); // Ano  : 2006
        System.out.println("Mês     : " + ( hr.get(GregorianCalendar.MONTH) + 1) ); // Mês 8
        System.out.println("Dia     : " + hr.get(GregorianCalendar.DAY_OF_MONTH)); // 20
        System.out.println("Hora    : " + hr.get(GregorianCalendar.HOUR)); // Hora  : 10
        System.out.println("Minuto  : " + hr.get(GregorianCalendar.MINUTE)); // Minuto : 20
        System.out.println("Segundo : " + hr.get(GregorianCalendar.SECOND)); // Seg : 30
        System.out.println("Dia da Semana : " + hr.get(GregorianCalendar.DAY_OF_WEEK));
        System.out.println("Data e hora : " + new SimpleDateFormat("EEEE dd/MM/yyyy
        HH:mm:ss").format(hr.getTime())); // Data e hora : Domingo 20/08/2006 10:20:30
}
```

Acesso aos atributos e métodos

```
public  void  set (int field, int value)
        // Modifica o campo especificado de uma data.
        YEAR , MONTH , DAY_OF_MONTH , DAY_OF_WEEK,
        HOUR, HOUR_OF_DAY, MINUTE, SECOND, MILLISECOND
        public static void main(String[ ] args) {
        GregorianCalendar hr = new GregorianCalendar();
        hr.set(2006,8-1,20,10,20,30); // 20/08/2006 10:20:30
        System.out.println("Data e hora : " + new SimpleDateFormat("dd/MM/yyyy
        HH:mm:ss").format(hr.getTime()));
        hr.set(GregorianCalendar.YEAR,2007);
        System.out.println("Ano: " + hr.get(GregorianCalendar.YEAR)); //Ano 2007
        hr.set(GregorianCalendar.MONTH,9); // Mês de Outubro
        System.out.println("Mês: " + (hr.get(GregorianCalendar.MONTH) + 1) );
        hr.set(GregorianCalendar.DAY_OF_MONTH,21);
        System.out.println("Dia: " + hr.get(GregorianCalendar.DAY_OF_MONTH));
        System.out.println("Data e hora : " + new SimpleDateFormat("EEEE dd/MM/yyyy
        HH:mm:ss").format(hr.getTime())); //Domingo 21/10/2007 10:20:30
}
```

Classes e Objetos

* **Encapsulamento**
  * Uma classe pode ser visualizada através de duas perspectivas: interface e implementação.
    * A **interface** pode ser vista e usada por outros objetos (clientes).
    * A **implementação** é escondida dos clientes.
  * Esconder os detalhes da implementação de um objeto é chamado **encapsulamento**.
  * Encapsulamento oferece dois tipos de proteção:
    * Protege o estado interno de um objeto de ser corrompido por seus clientes.
    * Protege o código cliente de mudanças na implementação dos objetos.

* **Benefícios do Encapsulamento**
  * O código cliente pode usar a interface para uma operação.
  * O código cliente não pode tirar vantagem da implementação de uma operação.
  * A implementação pode mudar, por exemplo para:
    * Corrigir um erro (bug).
    * Aumentar a performance.
    * Refletir uma mudança no plano de ação.
  * O código cliente não será afetado pelas mudanças na implementação, assim reduzindo o "efeito ondulação" no qual uma correção em uma operação força correções correspondentes numa operação cliente, o qual por sua vez, causa mudanças em um cliente do cliente...
  * A manutenção é mais fácil e menos custosa.

Classes e Objetos

- **Visibilidade e Encapsulamento**
  - O controle de acesso (visibilidade) é usado para garantir o encapsulamento.
    - A visibilidade é especificada para atributos e operações.
  - Atributos e Métodos devem ser tão privativos quanto possíveis.
    - Recomendação: os atributos devem ser sempre privados.
      - Para acessar os atributos, use operações get() e set().
    - Para um método, basta conhecer sua especificação. Não há necessidade de saber detalhes de sua implementação.

Classes e Objetos

* **Restrições de Acesso**
  * Restrições de acesso especificam o quão acessível é um membro (atributo ou método) de um objeto a partir de outros objetos.
  * Existem 4 formas de restrição de acesso em Java:
    * **public**: o membro é visível (pode ser acessado) a todos os objetos;
    * **protected**: o membro é visível dentro da própria classe, dentro de suas subclasses e dentro do pacote ao qual a classe pertence;
    * **package**: o membro é visível dentro da própria classe e dentro do pacote ao qual a classe pertence. Não se usa a palavra package: restrição **default**;
    * **private**: o membro é visível somente dentro da própria classe.

Classes e Objetos

* **Restrições de Acesso**

Especificador	Classe	Subclasse	Pacote	Aplicação(s)
public	X	X	X	X
protected	X	X	X	
package	X		X	
private	X			

  * Dicas:
    * Sempre mantenha os atributos privados.
    * Crie métodos assessores – **get** e modificadores - **set** para os atributos visíveis fora da classe
      * Nem todos os atributos necessitam de métodos assessores e modificadores.

Classes e Objetos

* **Criação de Objetos**
  * No paradigma de orientação por objetos, tudo pode ser potencialmente representado como um objeto. Um objeto não é muito diferente de uma variável normal.
  * Quando se cria um objeto, esse objeto adquire um espaço em memória para armazenar seu estado (os valores de seu conjunto de atributos, definidos pela classe) e um conjunto de operações que podem ser aplicadas ao objeto (o conjunto de métodos definidos pela classe).
  * Um programa orientado por objetos é composto por um conjunto de objetos que interagem através de "trocas de mensagens". Na prática, essa troca de mensagem traduz-se na aplicação de métodos a objetos.

Classes e Objetos

* **Criação de Objetos**
  * A criação de um objeto é feita pelo operador **new**.

    new NomeDaClasse();

  * Essa expressão é uma invocação do construtor, um método especial que toda a classe oferece, que indica o que deve ser feito na inicialização de um objeto.

Classes e Objetos

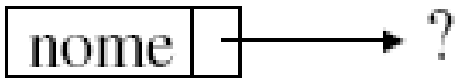
* **Criação de Objetos**
  * A aplicação do operador **new** ao construtor da classe retorna uma referência para o objeto. Para que o objeto possa ser efetivamente manipulado, essa referência deve ser armazenada por quem determinou a criação do objeto:
  * **NomeDaClasse** minhaRef = **new NomeDaClasse();**
  * Nesse exemplo, minhaRef é uma variável que guarda uma referência para um objeto do tipo **NomeDaClasse**.

Classes e Objetos

- **Construtores**
  - Construtor é um método especial da classe chamado pelo operador **new** quando um novo objeto da classe é criado.
  - Características dos métodos construtores :
    - normalmente são métodos públicos;
    - possuem o mesmo nome da classe;
    - não possuem valor de retorno;
    - podem ter parâmetros;
    - pode haver sobrecarga de construtores (overloading) ;
    - um construtor default sem parâmetros é gerado se nenhum construtor é fornecido pelo implementador da classe.
      - O construtor default inicializa todas os atributos da classe, não inicializados explicitamente, com seus valores padrão (números com zero, objetos com nulo e booleanos com falso).

Classes e Objetos

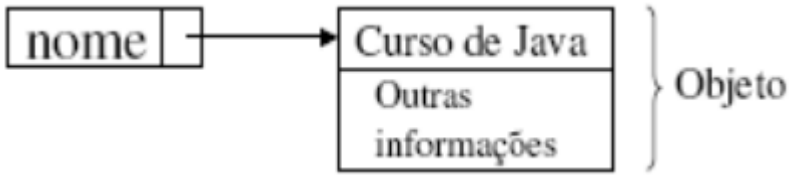
* **Manipulação de Objetos**
  * A declaração de uma variável cujo tipo é uma classe não cria um objeto. Cria-se uma referência para um objeto da classe, a qual inicialmente não faz referência a nenhum objeto válido.
  * Exemplo:
    * **String** nome;



Classes e Objetos

* **Manipulação de Objetos**
  * Ao se criar um objeto, usando o operador **new**, obtém-se uma referência válida, que é armazenada na variável do tipo da classe.
  * Exemplo (continuação):

    StringBuffer nome = **new StringBuffer**("Curso de Java");

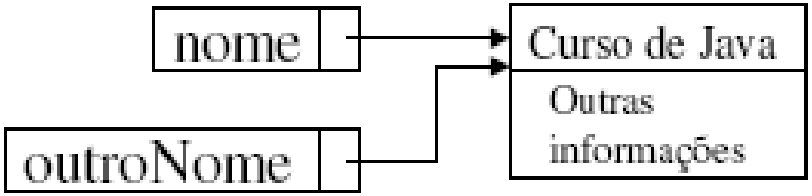


  * A variável nome armazena uma referência para o objeto cujo conteúdo é "Curso de Java".

Classes e Objetos

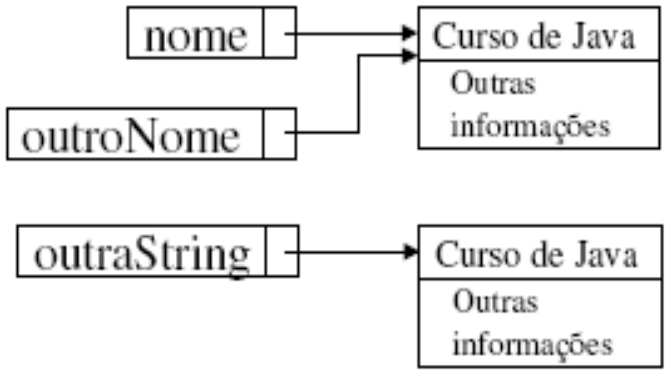
* **Manipulação de Objetos**
  * A variável nome do exemplo anterior mantém apenas a referência para o objeto e não o objeto em si.
  * O operador = (atribuição) não cria outro objeto. Ele simplesmente atribui a **referência** para o objeto.
  * Exemplo (continuação):

    StringBuffer outroNome = **nome;**



* **Manipulação de Objetos**
  * Para efetuar uma cópia de um objeto, criando um novo objeto com o mesmo conteúdo de um objeto já existente, é necessário usar o método **clone()** da classe **Object**, da qual **todos os objetos descendem**.
  * Exemplo (continuação):

```
StringBuffer outraString = nome.clone();
```



Classes e Objetos

- **Manipulação de Objetos**
  - O operador == para objetos compara apenas se os dois objetos têm a mesma **referência** (apontam para o mesmo local).
  - Exemplo (continuação):
    - nome == outroNome // resulta true
    - nome == outraString // resulta false
    - outroNome == outraString // resulta false
  - O método **equals()** da classe **StringBuffer** compara se o conteúdo de dois objetos são iguais.
  - Exemplo (continuação):
    - nome.equals(outroNome) // resulta true
    - outroNome.equals(nome) // resulta true
    - nome.equals(outraString) // resulta true
    - outraString.equals(nome) // resulta true

Classes e Objetos

* **Classes e Objetos**
  * **Pacotes**
    * Pacote é um recurso para agrupar física e logicamente classes e interfaces relacionadas.
    * Um pacote consiste de um ou mais arquivos.
    * Um arquivo pode ter no máximo uma classe pública.
    * Pacote é uma coletânea de arquivos de classes individuais.
    * O nome de um pacote corresponde a um nome de diretório.
    * Como pacote é um diretório, pode haver hierarquia de pacotes.

- **Classes e Objetos**
  - **Pacotes**
    - Um arquivo pertencente a um pacote inicia-se com a instrução **package**.
    - Se um arquivo importa (usa) classes de um pacote, em seguida, vem a instrução **import**.
      - O pacote **java.lang** é importado automaticamente.
    - Exemplo:

```
package cursojava; //  define um pacote de nome cursojava
import java.util.Arrays; //  importa classe Arrays do pacote java.util
import java.io.*; //  importa todas as classes do pacote java.io
public class Ordena { //  define uma classe pertencente ao pacote cursojava
// comandos
}
```

Classes e Objetos

* **Classes e Objetos**
  * **Convenção de Nomes em Java**
    * **Pacote** (recurso usado para agrupar física e logicamente classes e interfaces relacionadas.)
      * Letras minúsculas: Ex.: meupacote.
    * **Classe**
      * Primeira letra maiúscula, demais minúsculas. Nomes compostos iniciando com letras maiúsculas: Ex.: **M**inha**C**lasse.
    * **Métodos e Atributos**
      * Primeira letra minúscula, demais minúsculas. Nomes compostos iniciando com letras maiúsculas.
      * Ex.: **m**eu**M**etodo. Ex2.: **m**eu**A**tributo.
    * **Constantes**
      * Letras maiúsculas: Ex.: PI, LIMITE_DIAS.

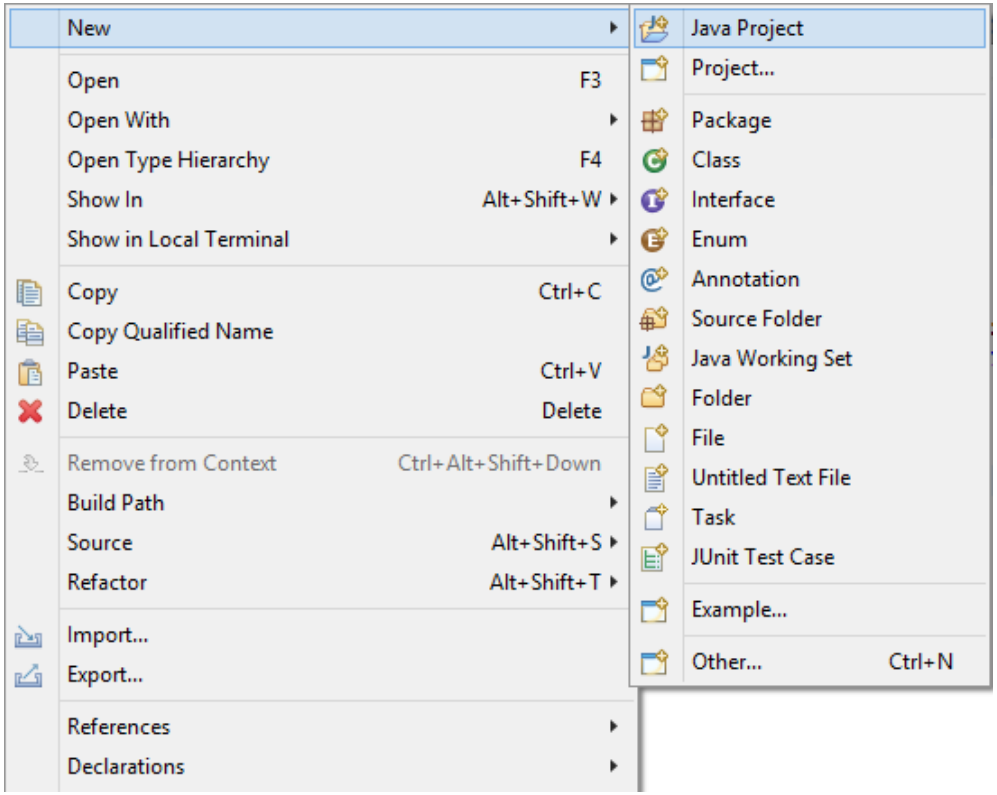
Classes e Objetos

* **Classes e Objetos**
  * **Convenção de Nomes em Java**
    * Exemplos:

Classes	Métodos	Variáveis	Constantes
Carro	desligar	motor	COMBUSTIVEL
CursoJavaIniciante	iniciarModulo	quantidadeModulos	NOME_CURSO
Hotel	reservarSuiteMaster	nomeReservaSuite	TAXA_SERVICO

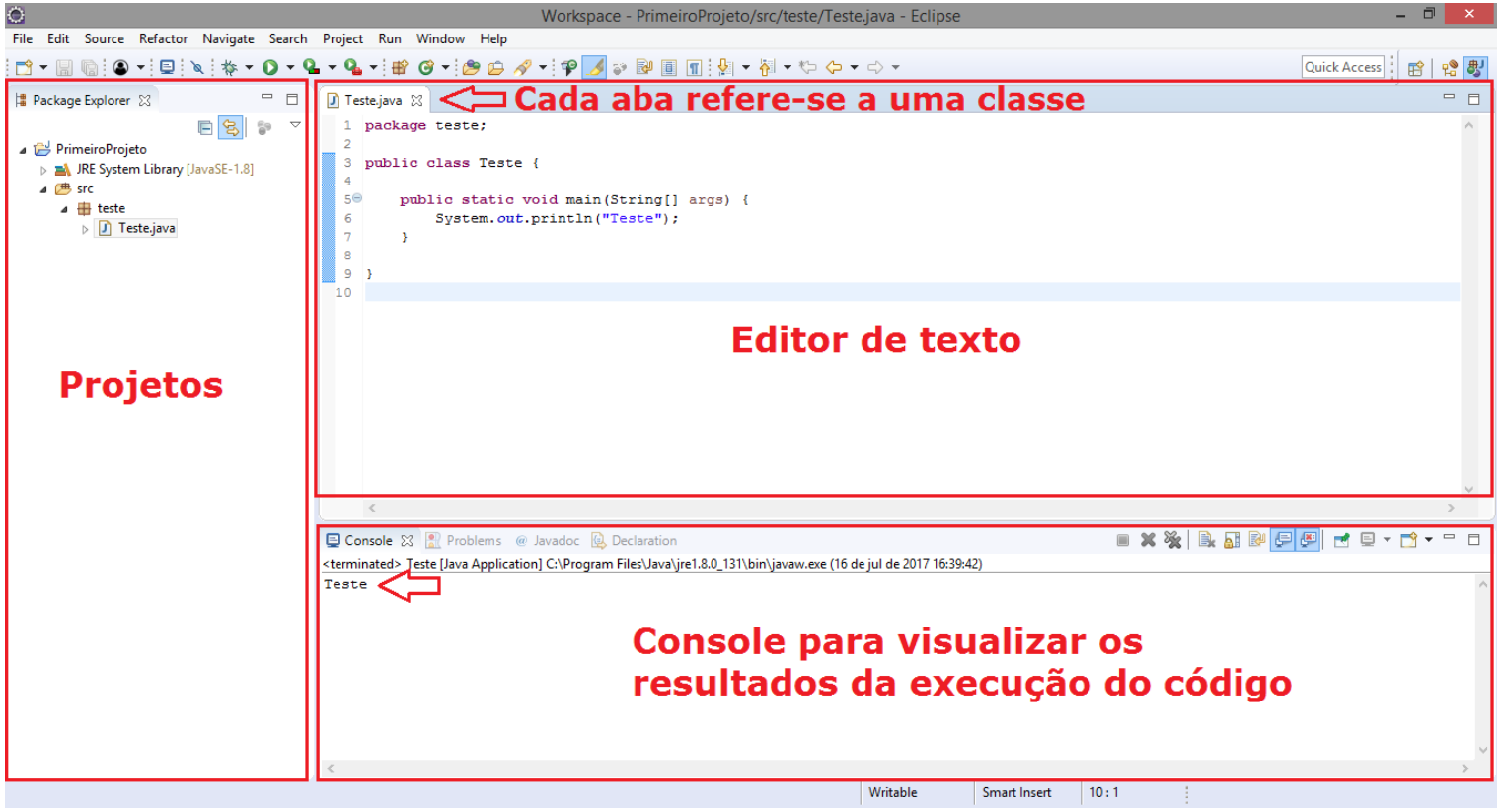
Classes e Objetos

* **Classes e Objetos**
  * **Abra o eclipse e crie um novo projeto Java:**



Classes e Objetos

Visualização básica do projeto:



Classes e Objetos

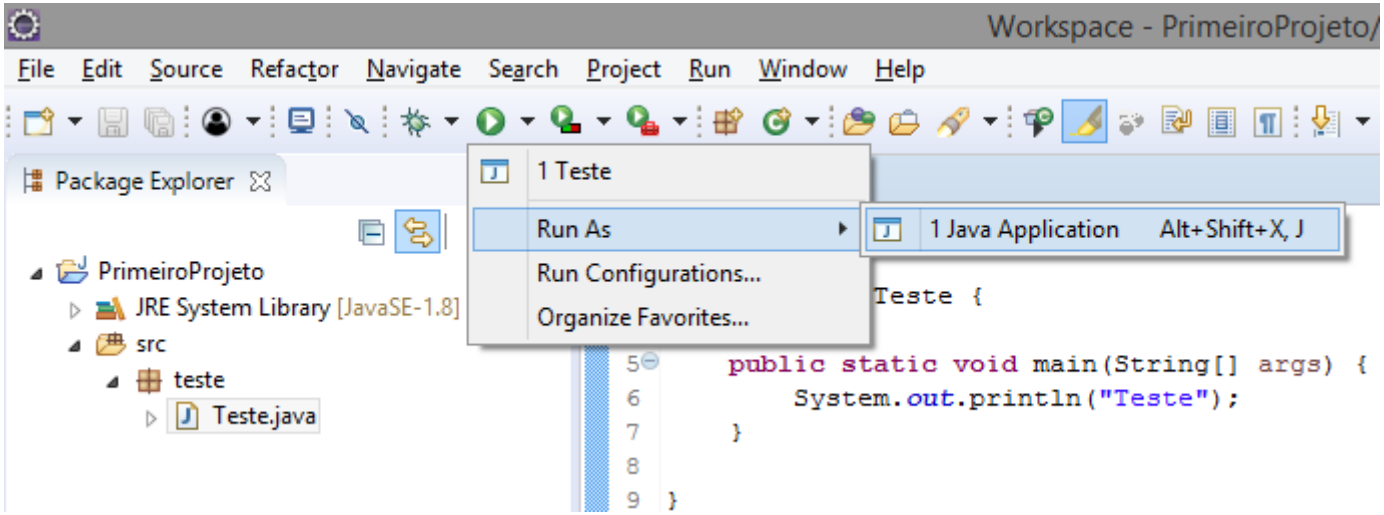
* **Classes e Objetos**
  * **Primeiro Programa**

```
package teste;
public class PrimeiroPrograma {
    public static void main ( String[ ] args ) {
        System.out.println ("Primeiro Programa Java!");
        System.exit(0);   // termino (Opcional)
    }
}
```

  * O nome do arquivo deve ser **PrimeiroPrograma.java**
  * Java é sensível a letras maiúsculas e minúsculas.
  * O método **main()** é o início da execução de uma aplicação Java.
  * A classe compilada é armazenada em um arquivo .class.

Classes e Objetos

* **Classes e Objetos**
  * **Para executar o código:**



Classes e Objetos

- **Classes e Objetos**
  - **Primeiro Programa**
    - O método **main** é o ponto de início de execução de uma aplicação Java.
    - A primeira classe a ser chamada em uma aplicação tem que possuir o método **main** para iniciar a aplicação.
    - Assinatura do método **main:**
      - **public static void** main (String[] args)
      - O nome do parâmetro (args) poderia ser diferente, mas os demais termos da assinatura devem obedecer ao formato especificado.

- **Classes e Objetos**
  - **Primeiro Programa**
    - O método **main** recebe como argumento um parâmetro do tipo arranjo de objetos da classe String.
    - Cada elemento desse arranjo corresponde a um argumento passado para o interpretador Java na linha de comando que o invocou.
    - Ex: **java** Teste aaaa 22 zzz
    - O método main (String[] args) da classe Teste vai receber, nessa execução, um arranjo (Vetor) de três elementos na variável args com os seguintes conteúdos:
      - args[0] – objeto String com conteúdo “aaaa”
      - args[1] – objeto String com conteúdo “22”
      - args[2] – objeto String com conteúdo “zzz”

Classes e Objetos

* **Classes e Objetos**
  * **Primeiro Programa**
    * O método **main** é do tipo **void**. Ele não tem valor de retorno.
    * Se for necessário retornar um valor deve-se usar o método System.exit(int).
      * A invocação desse método provoca o fim imediato da execução do interpretador Java.
      * Tipicamente, o argumento de exit () obedece à convenção de que "0" indica execução com sucesso, enquanto um valor diferente de o indica a ocorrência de algum problema.

* **Classes e Objetos**
  * **Primeiro Programa**
    * Observe no exemplo do **PrimeiroPrograma** o uso dos pares de chaves {}. As chaves delimitam blocos de códigos. Equivalente ao **begin end** do Pascal.
    * O método main possui somente uma instrução System.out.println ("Primeiro Programa Java!");
      * Por enquanto, saiba apenas que o método **println** imprime uma string na saída padrão.
      * Formalmente: a classe **System** possui um atributo estático **out** do tipo **PrintStream**, que por sua vez, possui o método **println** que imprime uma mensagem na saída padrão.
      * Uma string é delimitada por um **par de aspas**.
      * Toda instrução termina com um ponto-e-vírgula (;).

* **Classes e Objetos**
  * **Primeiro Programa**
    * Java possui três tipos de comentários:
      * **Comentário até o final da linha usando //**

        // isso é um exemplo de comentário até o final da linha
      * **Comentário em blocos usando delimitadores /* (início do comentário) e */ (fim do comentário)**

        /* o comentário começa aqui, continua aqui e

        termina aqui */
      * **Comentário para documentação usando javadoc. O comentário começa com /** e termina com */**

        / Comentário para a ferramenta javadoc**

        @version 1.0

        @author xyz

        ***/**

Classes e Objetos

* **Exemplo OO**
  * Crie uma classe chamada "**Principal**" dentro de um pacote **teste.**

```
package teste;
public class Principal {
    public static void main(String[] args) {
            // Objeto exemploOO, instância da classe ExemploOO
            ExemploOO exemploOO = new ExemploOO();
            exemploOO.setValor(15);
            System.out.println("Resultado: " + exemploOO.getValor());
    }
}
```

* **Exemplo OO**
  * Crie uma classe chamada "**ExemploOO**" dentro do pacote **teste.**

```
package teste;
public class ExemploOO {
    private int valor;

    public void setValor(int valor) {
            this.valor = valor;
    }
    public int getValor() {
            return valor;
    }
}
```

Exemplo Construtor

- Crie uma classe chamada **"Principal"** dentro de um pacote **teste.**

```
package teste;
public class Principal {
    public static void main(String[] args) {
    Caixa c1 = new Caixa();
    System.out.println("Volume da caixa 1 = " + c1.volume()); // volume = 1000
    Caixa c2 = new Caixa(10,5,3);
    System.out.println("Volume da caixa 2 = " + c2.volume()); // volume = 150
    c2.setComp(8);
    c2.setLarg(c2.getLarg()-3);
    System.out.println("Novo volume da caixa 2 = " + c2.volume()); // volume = 48
    }
}
```

Exemplo Construtor

- Crie uma classe chamada "**Caixa**" dentro do pacote **teste.**

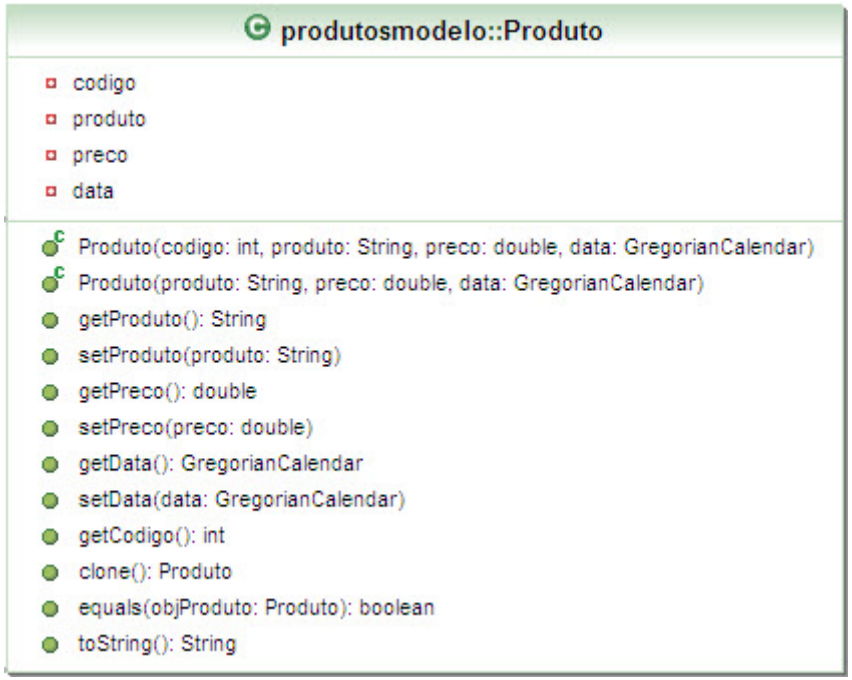
```
package teste;
public class Caixa {
    private double comp, larg, alt;
    public Caixa() { // Construtor
        this(10, 10, 10);
    }
    public Caixa(double comp, double larg, double alt) { // Construtor
        this.comp = comp;
        this.larg = larg;
        this.alt = alt;
    }
    public double volume() {
        return (comp * larg * alt);
    }
}
```

* **Exemplo Construtor**
  * Implemente os get's e set's.
    na classe **Caixa**

```
public double getComp() {
return comp;
}

public void setComp(double comp) {
this.comp = comp;
}

public double getLarg() {
return larg;
}

public void setLarg(double larg) {
this.larg = larg;
}

public double getAlt() {
return alt;
}

public void setAlt(double alt) {
this.alt = alt;
}
```

Diagramas de Classe

* **Exemplo de Diagrama de Classe:**

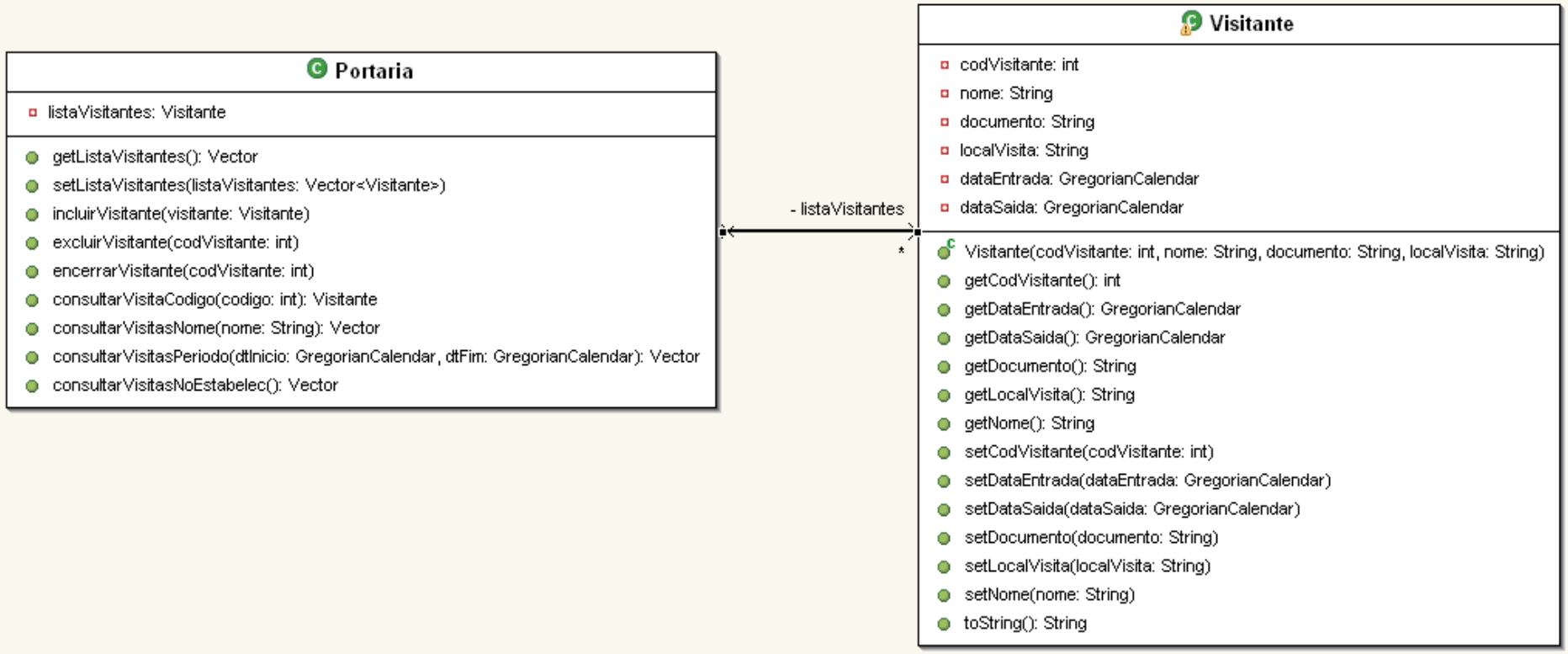


```java
public class Produto {
    //Atributos
    private int codigo;
    private String produto;
    private double preco;
    private GregorianCalendar data;

    public Produto(int codigo, String produto, double preco, GregorianCalendar data) {
        this.codigo = codigo;
        this.produto = produto;
        this.preco = preco;
        this.data = data;
    }
    public Produto(String produto, double preco, GregorianCalendar data) {
        this.produto = produto;
        this.preco = preco;
        this.data = data;
    }

    public String getProduto() { return produto; }
    public void setProduto(String produto) { this.produto = produto; }
    public double getPreco() { return preco; }
    public void setPreco(double preco) { this.preco = preco; }
    public GregorianCalendar getData() { return data; }
    public void setData(GregorianCalendar data) { this.data = data; }
    public int getCodigo() { return codigo; }

    public Produto clone () {
        return new Produto(codigo,produto,preco,data);
    }
    public boolean equals (Produto objProduto) {
        if (codigo==objProduto.getCodigo() && produto.equalsIgnoreCase(objProduto.getProduto()) &&
            preco==objProduto.getPreco() && data.equals(objProduto.getData()))
               return true;  else  return false;
    }
    public String toString() {
        return
            "Código  : " + codigo + "\n" +
            "Produto : " + produto + "\n" +
            "Preço   : " + LtpUtil.formatarValor(preco, "R$#,##0.00") + "\n" +
            "Data    : " + LtpUtil.formatarData(data, "dd/MM/yyyy") + "\n";
    }
}
```

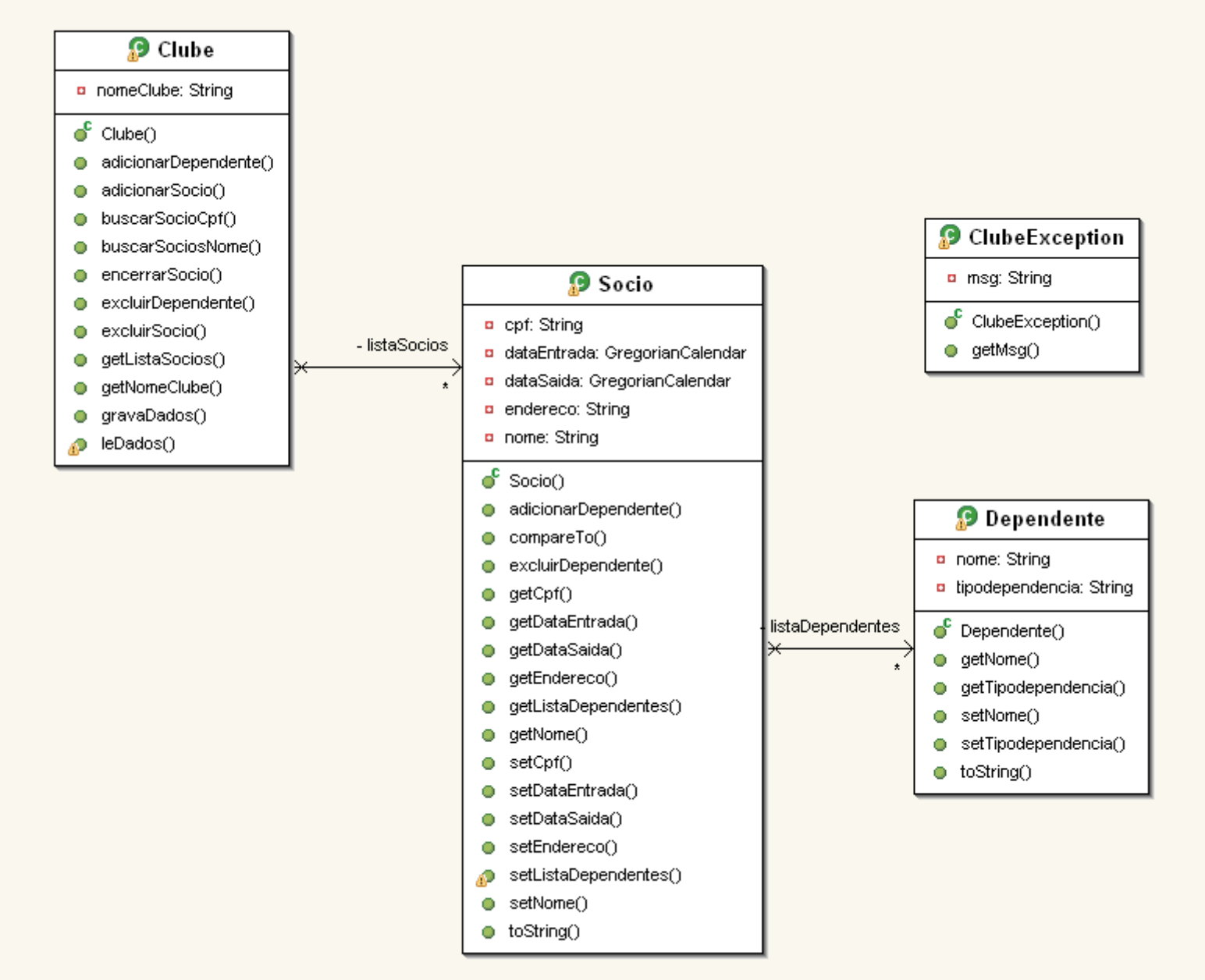
Construtores

get´s e set´s

* Outros exemplos: SisPortaria



* Outros exemplos: SisClube



Obrigado.

joaopauloaramuni@gmail.com
joaopauloaramuni@fumec.br

UNIVERSIDADE FUMEC
50 Anos