



Programação Orientada a Objetos em Java

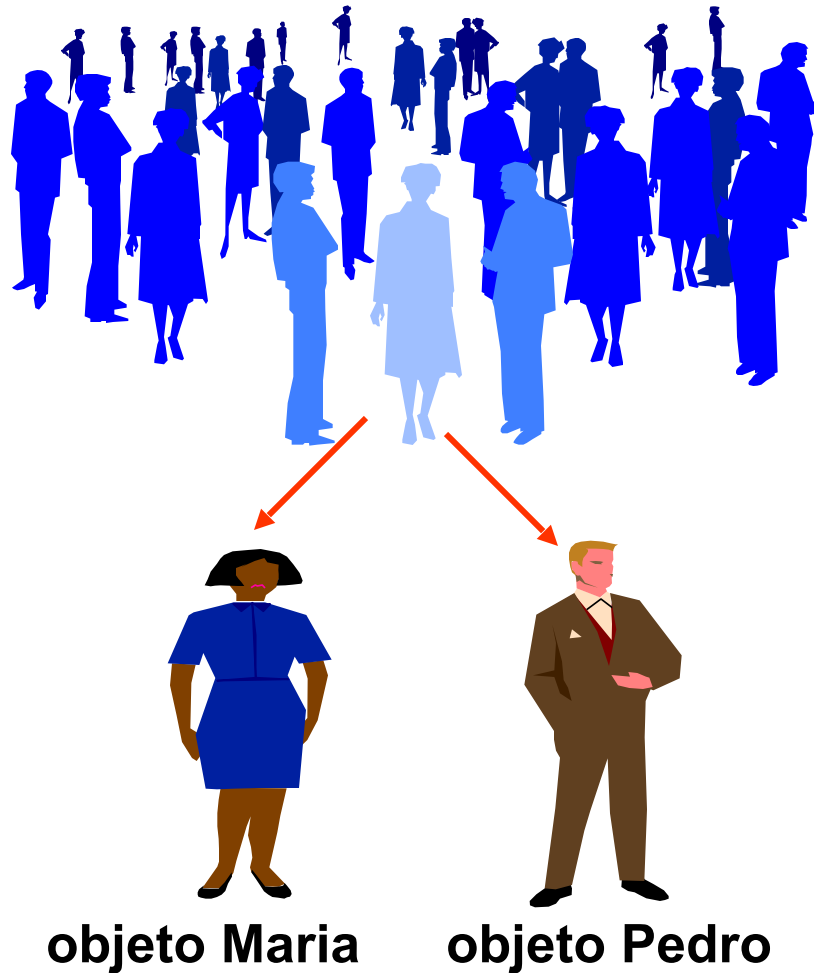
Aula 3 - Classes e Objetos





Classes e Objetos

Classe Pessoa





Classes e Objetos

Classe

Pessoa

nome
endereco
telefone
idade
altura

registrar()
matricular()
estudar()
cadastrar()
pagar()

Objetos



A
T
R
I
B
U
T
O
S
M
É
T
O
D
O
S

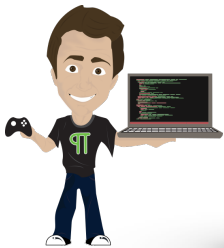
Maria Rua A, 23 3226-5689 15 1.6

Pedro Rua B, 45 3205-2365 21 1.8



Televisao

- Tamanho Tela (int)
- Marca (string)
- Voltagem (int)
- Tipo (smart/4k) (string)
- Tecnologia (string)
- Estado (boolean)
- Ligada (boolean)
- Canal (int)
- Volume (int)
- Ligar
- Desligar
- Avancar Canal
- Voltar Canal
- Mudar Canal
- Aumentar volume
- Diminuir volume
- Mutar



Classes e Objetos



Automóvel
Marca
Placa

Palio JWO-4567

Parati KLJ-0978

Celta JDK-6543

CLASSE



OBJETOS

(Instâncias da classe Automóvel)





Classes em Java

- Um dos principais resultados das fases de análise e projeto Orientado a Objetos:
 - Definição de um modelo conceitual para o domínio da aplicação, contemplando as classes relevantes e suas associações.
- O uso de uma linguagem de modelagem (exemplo, diagrama de classes UML) permite expressar esse resultado de maneira organizada e padronizada.
- Uma classe é um gabarito para a definição de objetos.
 - propriedades (**atributos**) e
 - comportamentos (**métodos**)





Definição de Classes em Java

- Diagrama de Classes: a representação de classes contempla três tipos básicos de informação:

Automóvel
nomeProprietario: String placa: String modelo: String
setNomeProprietario(novoProprietario: String): void setPlaca(novaPlaca: String): void imprimirInfo(): void

- Nome da classe: Um identificador para a classe que permite referenciá-la posteriormente (por exemplo, no momento da criação de um objeto).
- Atributos:
 - nome: um identificador para o atributo.
 - tipo: o tipo do atributo (inteiro, real, caracter, outra classe, etc.)
 - valor_default: opcionalmente, pode-se especificar um valor inicial para o atributo.



Definição de Classes em Java

- Métodos:
 - nome: um identificador para o método.
 - tipo: quando o método tem um valor de retorno, o tipo desse valor.
 - lista de argumentos: quando o método recebe parâmetros para sua execução, o tipo e um identificador para cada parâmetro.
- Modificadores de acesso a membros
 - Pode-se especificar o quão acessível é um atributo ou método de um objeto a partir de outros objetos.
 - Os valores possíveis são:
 - + public : visibilidade externa total
 - # protected : visibilidade externa limitada.
 - private : nenhuma visibilidade externa





Exemplo: Classe Automovel

```
//declaração de pacotes
//definição da classe Automovel
public class Automovel {
    //declaração dos atributos da classe
    private String nomeProprietario;
    private String modelo;
    private String placa;
    private int ano;

    //método construtor
    public Automovel (String nomeProprietario, String modelo, String
    placa, int ano) {
        this.nomeProprietario = nomeProprietario;
        this.modelo = modelo;
        this.placa = placa;
        this.ano = ano;
    }
}
```





Exemplo: Classe Automovel

```
//declaração dos demais métodos da classe
public void imprimirInfo (){
    System.out.println(nomeProprietario+ " possui um(a) "+modelo+
        " com placa "+placa+ " e ano "+ano);
}
public void setNomeProprietario (String nome){
    this.nomeProprietario = nome;
}
public void setPlaca (String nPlaca){
    this.placa = nPlaca;
}
```



Exemplo: Classe AutomovelTeste

```
public class AutomovelTeste{  
    public static void main (String args []) {  
        //instanciando um objeto da classe Automóvel  
        Automovel a = new Automovel ("Eduardo", "Palio", "JW02125",  
                                     2002);  
  
        //troca de mensagens (chamada ao método imprimir())  
        a.imprimir();  
        System.out.println ("***Transferencia de Proprietario***");  
        a.setNomeProprietario("Rosa");  
        a.imprimir();  
        Automovel b = new Automovel ("Rodrigo", "Parati",  
                                     "JSX6481", 1999);  
  
        b.imprimir();  
        System.out.println ("***Mudanca de Placa***");  
        b.setPlaca("SDK2581");  
        b.imprimir();  
    } //fim do método main  
} //fim da classe AutomovelTeste
```





Definição de Classes em Java

- Em Java, classes são definidas através do uso da palavra-chave **class**.
- Sintaxe para definir uma classe:

```
[modificador] class NomeDaClasse {  
    // corpo da classe...  
}
```

- Após a palavra-chave class, segue-se o nome da classe, que deve ser um identificador válido para a linguagem.
- [modificador] é opcional; se presente, pode ser uma combinação de public e abstract ou final.
- O modificador abstract indica que nenhum objeto dessa classe pode ser instanciado.
- O modificador final indica que a classe não pode ser uma superclasse (uma classe não pode herdar de uma classe final)





Definição de Classes em Java

- A definição da classe propriamente dita está entre { e }, que delimitam blocos na linguagem Java. Usualmente, o **corpo de uma classe** obedece à seguinte sequência de definição:
 - As **variáveis de classe**, iniciando pelas public, seguido pelas protected, e finalmente pelas private.
 - Os **atributos** (ou variáveis de instância) dos objetos dessa classe, seguindo a mesma ordenação definida para as variáveis de classe.
 - Os **construtores** de objetos dessa classe.
 - Os **métodos** da classe, geralmente agrupados por funcionalidade.
- Toda classe pode também ter um método main associado
 - Esse método como já mencionado é utilizado pelo interpretador Java para dar início à execução de uma aplicação.
- Propriedades de uma classe podem ser obtidas através das funcionalidades oferecidas na classe java.lang.Class



Modificadores de acesso a membros em Java

- Os modificadores de acesso controlam o acesso às variáveis de instância e aos métodos de uma classe
- Quando nenhum membro modificador de acesso é oferecido para um método ou variável quando estão definidos em uma classe – **acesso de pacote**
 - Se o programa consiste em uma definição de classe, isso não tem nenhum efeito específico no programa
 - Se o programa utilizar várias classes do mesmo pacote (um grupo de classes relacionadas), essas classes podem acessar todos os métodos de acesso e dados com acesso de pacote umas das outras diretamente, através de uma referência a um objeto



Modificadores de acesso a membros em Java

- public

- Amplia a visibilidade padrão (está restrita a todos os membros que fazem parte de um mesmo pacote) deixando-a sem restrições.
- Uma classe definida como pública pode ser utilizada por qualquer objeto de qualquer pacote.
- Um arquivo fonte (`.java`) pode ter no máximo uma classe pública, cujo nome deve ser o mesmo do arquivo.
- As demais classes num arquivo fonte, não públicas, são consideradas classes de suporte para a classe pública e têm a visibilidade padrão.





Modificadores de acesso a membros em Java

- protected

- Nível intermediário de proteção entre o acesso

public e private

- Restringe a visibilidade do membro modificado, atributo ou método, apenas à própria classe, àquelas derivada desta (herança) e também a outras classes no mesmo pacote (acesso de pacote).



Modificadores de acesso a membros em Java

- private

- Restringe o acesso a membros de uma classe (atributo ou método) a objetos da própria classe que contém sua definição - encapsulamento.
- Tornar private as variáveis de instância de uma classe e public os métodos da classe facilita a depuração, uma vez que os problemas com as manipulações dos dados estão localizados nos métodos da classe



Variáveis de Classe

- Cada objeto de uma classe tem sua própria cópia de todas as variáveis de instância da classe
- Em certos casos, apenas uma cópia de uma variável particular deve ser compartilhada por todos os objetos de uma classe
- Uma variável de classe tem sua declaração precedida pela palavra-chave static.



Variáveis de Classe

- Várias constantes são definidas em Java como

`public static final`

- a classe Math (java.lang) de Java, por exemplo, define as constantes E (2.71828...) e PI (3.14159...). Para ter acesso a esses valores, basta precedê-los com o nome da classe e um ponto, como em

```
double pi2 = Math.PI/2;
```

- Outro exemplo de variável `public static final` é a variável `out` da classe `System`. Essa variável, `System.out`, está associada a um objeto que representa a saída padrão.





Atributos ou Variáveis de Instância

- Sintaxe para definir um atributo em uma classe é:

`[modificador] tipo nome [= default];` onde

- modificador (opcional), uma combinação de
 - `public`, `protected` ou `private`;
 - `final`;
- tipo: deve ser um dos tipos primitivos da linguagem Java ou o nome de uma classe.
- nome: deve ser um identificador válido.
- default: (opcional) é a especificação de um valor inicial para a variável.
- O modificador `final` especifica que uma variável de instância não é modificável (**constante**).



Métodos em Java

- A definição de métodos reflete de forma quase direta a informação que estaria presente em um diagrama de classes UML, a não ser por uma diferença vital: o corpo do método.
- Dica: uma boa prática de programação é manter a funcionalidade de um método simples, desempenhando uma única tarefa.
 - O nome do método deve refletir de modo adequado a tarefa realizada.
 - Se a funcionalidade do método for simples, será fácil encontrar um nome adequado para o método.
- Os métodos são essencialmente procedimentos que podem:
 - Manipular atributos de objetos para os quais o método foi definido.
 - Definir e manipular variáveis locais
 - Receber parâmetros por valor através da lista de argumentos





Métodos em Java

- Sintaxe para definir um método em uma classe é:

```
[modificador] tipo nome(argumentos)
{ corpo do método } onde:
```

- modificador (opcional), uma combinação de:
 - `public`, `protected` ou `private`;
 - `abstract` ou `final`; e
 - `static`.
- tipo: indicador do valor de retorno, sendo `void` se o método não tiver um valor de retorno;
- nome: deve ser um identificador válido
- argumentos: são representados por uma lista de parâmetros separados por vírgulas, onde cada parâmetro obedece à forma `tipo nome`.



Métodos em Java

- O modificador abstract define apenas a assinatura do método que deverá ser implementado na subclasse
- O modificador final associado ao método indica que o mesmo não pode ser sobrescrito em uma subclasse
- O modificador static indica o uso do método independentemente de qualquer objeto de uma classe
- **Sintaxe** de chamada ou acesso a métodos:

`objeto.nomeDoMetodo (argumentos)`

`classe.nomeDoMetodo (argumentos) – para métodos static`



Métodos em Java

- Usualmente, métodos definidos em uma classe são aplicados a objetos daquela classe.
- Há situações nas quais um método pode fazer uso dos recursos de uma classe para realizar sua tarefa sem necessariamente ter de estar associado a um objeto individualmente.
 - Java define os métodos da classe, cuja declaração deve conter o modificador `static`.
 - Um método estático pode ser aplicado à classe e não necessariamente a um objeto.



Métodos em Java

- Exemplos de alguns métodos estáticos em Java incluem os métodos definidos nas classes:
 - java.lang.Character
 - java.lang.Integer
 - java.lang.Double
 - java.lang.Math
- Exemplo: para atribuir a raiz quadrada de 4 a uma variável raiz:

```
double raiz = Math.sqrt(4.0);
```



Métodos `get` e `set`

- As variáveis de instância privadas podem ser manipuladas somente por métodos da classe
- Métodos `set` (métodos modificadores)
 - Atribuir valores a variáveis de instância `private`
- Métodos `get` (métodos de acesso)
 - Obter valores de variáveis de instância `private`
- Embora os métodos `get` e `set` possam fornecer acesso a dados `private`, o acesso é restringido pela implementação dos métodos feita pelo programador



Métodos get e set

- Os métodos que configuram valores de dados `private` devem verificar se os novos valores pretendidos são adequados.
- Se eles não forem, os métodos `set` devem colocar variáveis de instância `private` em um estado consistente apropriado
- Exemplos:

```
public String getNome ()
{
    return nome;
}
```

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





Objetos em Java

- **O que é um objeto?**

- Sob o ponto de vista da programação, um objeto não é muito diferente de uma variável no paradigma de programação convencional.
 - **Exemplo**: quando se define uma variável do tipo `int` em C ou em Java, essa variável tem:
 - um espaço em memória para registrar o seu estado atual (um valor);
 - um conjunto de operações associadas que podem ser aplicadas a ela, através dos operadores definidos na linguagem que podem ser aplicados a valores inteiros (soma, subtração, inversão de sinal, multiplicação, divisão inteira, resto da divisão inteira, incremento, decremento).



Objetos em Java

- Da mesma forma, 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).
- É através de objetos que (praticamente) todo o processamento ocorre em aplicações desenvolvidas com linguagens de programação OO.
- Objetos são instâncias de classes:
 - Precisam ser criados a fim de que, através de sua manipulação, possam realizar seu trabalho.
 - Após a conclusão de suas atividades, objetos podem ser removidos.
 - Arranjos de tipos primitivos ou de objetos são criados e manipulados de forma análoga a objetos.



Como criar um objeto em Java

- A criação do objeto é feita através da aplicação do operador new (uma vez que a classe a partir da qual deseja-se criar o objeto exista):

NomeDaClasse umaRef = new NomeDaClasse(argumentos);

- essa expressão invoca o construtor da classe
- umaRef é uma variável que guarda uma referência para um objeto do tipo NomeDaClasse.
- new indica que um novo objeto está sendo criado
- argumentos especificam os valores utilizados pelo construtor da classe para inicializar o objeto



Como criar um objeto em Java

- Objetos nunca são manipulados diretamente, mas sempre através de uma variável que contém uma referência para o objeto.
- Internamente, uma referência conterá:
 - O endereço para a área de memória que contém o objeto, mas isso é irrelevante sob o ponto de vista do programador.

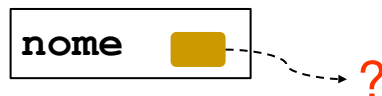


Manipulação de objetos

- Quando se declara uma variável cujo tipo é o nome de uma classe, como em:

`String nome;`

- não está se criando um objeto dessa classe, mas simplesmente uma referência para um objeto da classe `String`, a qual inicialmente não faz referência a nenhum objeto válido:



- Quando um objeto dessa classe é criado, obtém-se uma referência válida, que é armazenada na variável cujo tipo é o nome da classe do objeto.

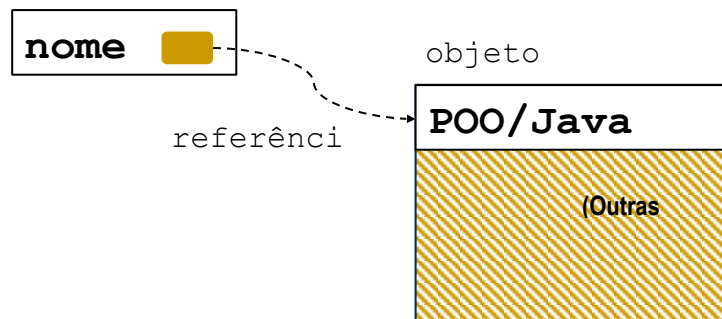


Manipulação de objetos

- Quando se cria uma *string* como em

```
nome = new String("POO/Java");
```

- nome é uma variável que armazena uma referência para um objeto específico da classe String - o objeto cujo conteúdo é "POO/Java":



- É importante ressaltar que a variável *nome* mantém apenas a referência para o objeto e não o objeto em si.

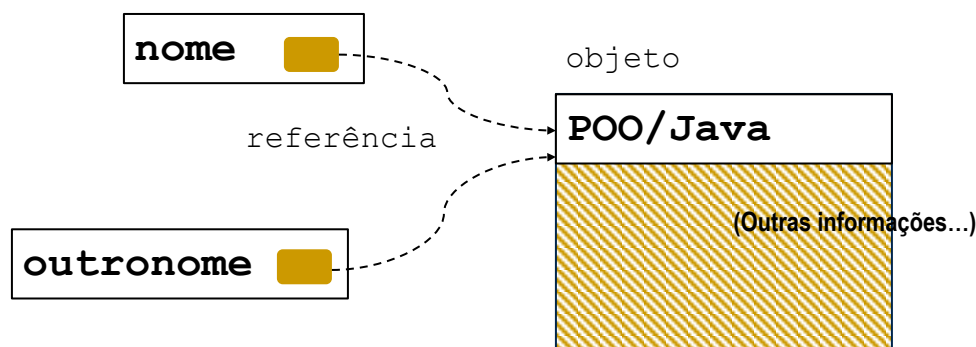


Manipulação de objetos

- Assim, uma atribuição como

```
String outroNome = nome;
```

- não cria outro objeto, mas simplesmente uma outra referência para o mesmo objeto:



- O único modo de aplicar os métodos a um objeto é através de uma referência ao objeto.



Construtores

- Um construtor é um método especial, definido para cada classe.
 - Determina as ações associadas à inicialização de cada objeto criado.
 - É invocado toda vez que o programa instancia um objeto dessa classe.
 - A assinatura de um construtor diferencia-se das assinaturas dos outros métodos por não ter nenhum tipo de retorno (nem mesmo `void`).
 - O nome do construtor deve ser o próprio nome da classe.
 - O construtor pode receber argumentos, como qualquer método.
 - Toda classe tem pelo menos um construtor sempre definido.



Construtores

- Se nenhum construtor for explicitamente definido pelo programador da classe, um construtor default (0 para tipos numéricos primitivos, `false` para `boolean` e `null` para referências), que não recebe argumentos, é criado pelo compilador Java.
- Se o programador da classe criar pelo menos um método construtor, o construtor default **não será** criado automaticamente - se ele o desejar, deverá criar um construtor sem argumentos explicitamente.
- Usando o mecanismo de sobrecarga (lista de parâmetro diferentes), mais de um construtor pode ser definido para oferecer diversas maneiras de inicializar os objetos dessa classe.



Referência this

- É uma referência a um objeto
- Quando um método de uma classe faz referência a outro membro dessa classe para um objeto específico dessa classe, como Java assegura que o objeto adequado recebe a referência?
 - Cada objeto tem uma referência a ele próprio - chamada de referência this
 - Utiliza-se a referência this implicitamente para fazer referências às variáveis de instância e aos métodos de um objeto



Referência this

- **Exemplos de uso de this**
 - A palavra-chave this é utilizada principalmente em dois contextos:
 1. Diferenciar atributos de objetos, de parâmetros ou variáveis locais de mesmo nome;
 2. Acessar o método construtor a partir de outros construtores.
 - Utilizar this explicitamente pode aumentar a clareza do programa em alguns contextos em que this é opcional



Referência this

- Esse exemplo ilustra esses dois usos:

```
public class EsteExemplo
{
    int x;
    int y;
    // exemplo do primeiro caso:
    public EsteExemplo(int x, int y) {
        this.x = x;
        this.y = y;
    }
    // exemplo do segundo caso:
    public EsteExemplo() {
        this(1, 1);
    }
}
```



Remoção de objetos

- Em linguagens de programação OO, a criação de objetos é um processo que determina a ocupação dinâmica de muitos pequenos segmentos de memória.
- Se esse espaço não fosse devolvido para reutilização pelo sistema, a dimensão de aplicações desenvolvidas com linguagens de programação OO estaria muito limitada.
- As duas abordagens possíveis são:
 - Dedicar essa tarefa ao programador ou
 - Deixar que o sistema seja o responsável por esta retomada de recursos.

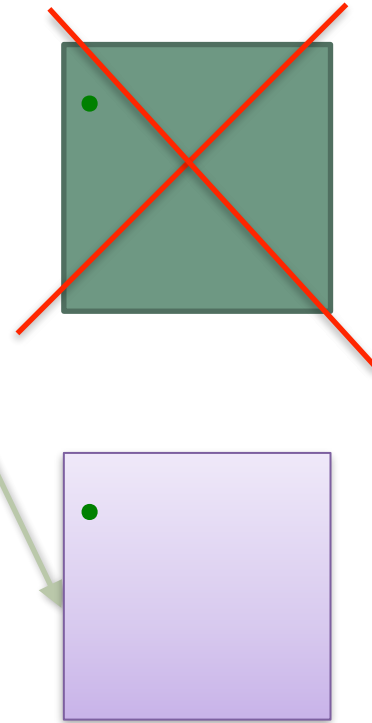


Pessoa p;

1. p = new Pessoa();

2. p = new Pessoa();

p





Remoção de objetos

- O problema da primeira abordagem é que o programador pode não considerar todas as possibilidades
- Na segunda abordagem, recursos adicionais do sistema são necessários para a manutenção da informação necessária para saber quando um recurso pode ser retomado e para a execução do processo que retoma os recursos.
- C++ é uma linguagem que adota a primeira estratégia.
 - A remoção de objetos é efetivada explicitamente pelo programador (através do operador delete), que pode também especificar o procedimento que deve ser executado para a liberação de outros recursos alocados pelo objeto através de um método **destructor**.



Remoção de objetos

- Java adota a abordagem de ter um coletor de lixo automático
 - verifica que objetos não possuem nenhuma referência válida, retomando o espaço dispensado para cada um desses objetos.
- O programador não precisa se preocupar com a remoção explícita de objetos.
- Pode ser necessário liberar outros recursos que tenham sido alocados para o objeto.
- Cada classe em Java pode ter um método finalizador que retorna recursos ao sistema
- É garantido que o método finalizador para um objeto é chamado para realizar uma limpeza de terminação no objeto imediatamente antes do coletor de lixo reivindicar a memória para o objeto



Remoção de objetos

- O método finalizador de uma classe sempre tem o nome `finalize()`, não recebe parâmetros e não retorna nenhum valor
- Esse método é definido na classe `Object` como protected. Portanto, se for necessário redefiní-lo o programador deve também declará-lo como protected.
- Chamada explícita ao coletor de lixo
 - Método `gc()` da classe `System`
 - Lembre-se de que não há garantia de que o coletor de lixo será executado quando `System.gc()` é invocado e nem há garantia de que ele colete objetos em uma ordem específica.



Exercícios

- Implementar uma Classe Pessoa (Pessoa.java e PessoaTeste.java)

Pessoa

nome: String

sobrenome: String

idade: int

sexo: char

imprimir(): String

Assinatura do método

public String imprimir()



Exercícios

- Implementar um Tipo de Dado Empregado como classe (Empregado.java e EmpregadoTeste.java)

Empregado
nome: String cargo: String salario: double
imprimir(): void aumentarSalario(...):void

Assinaturas dos métodos

public String imprimir()

public void aumentarSalario (double percentual)



Exercícios

Implementar um Tipo de Dado Time como classe (Time.java e TimeTeste.java)

Time

hora: int
minuto: int
segundo: int

setTime (...): void
exibirHoraUniversal(): String
exibirHoraPadrao(): String

Hora Universal

00:00:00 – 13:30:07

Hora Padrão

12:00:00 e um
indicador de AM PM

1:27:06 PM

Assinaturas dos métodos

```
public void setTime (int h, int m, int s)
public String exibirHoraUniversal(): String
public String exibirHoraPadrao(): String
```