



Classe e objeto

Prof^a. Rachel Reis
rachel@inf.ufpr.br



Problema

Uma empresa de tecnologia precisa de uma aplicação que armazene e imprima os dados de todos os seus funcionários.

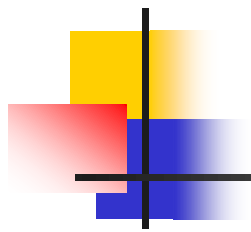




Problema

- Quais são as “classes” e “objetos” deste problema?

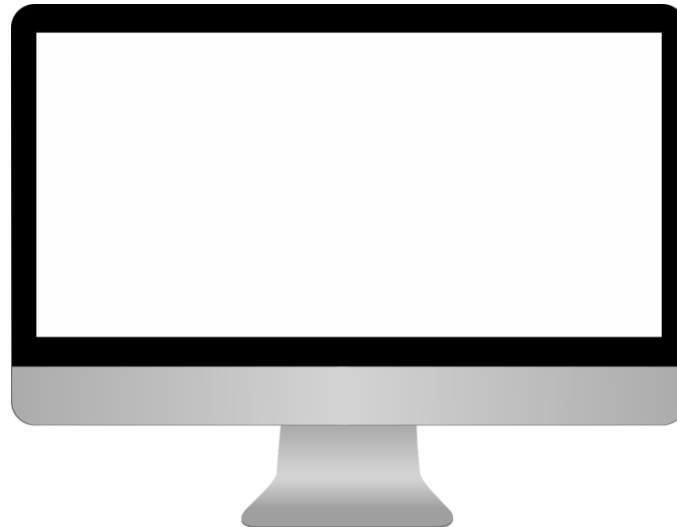
Uma empresa de tecnologia precisa de uma aplicação que armazene e imprima os dados de todos os seus funcionários.



Para entender os conceitos de **classe e objeto**
é necessário primeiro entender o conceito de
abstração



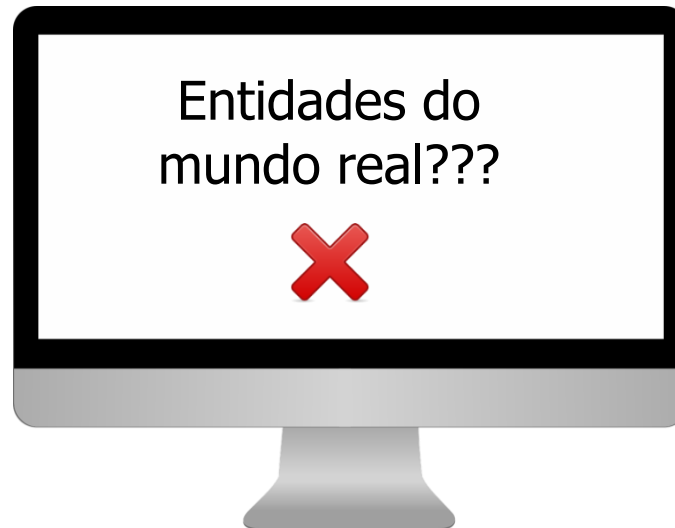
Abstração de Dados



- Necessária para resolver problemas no computador.



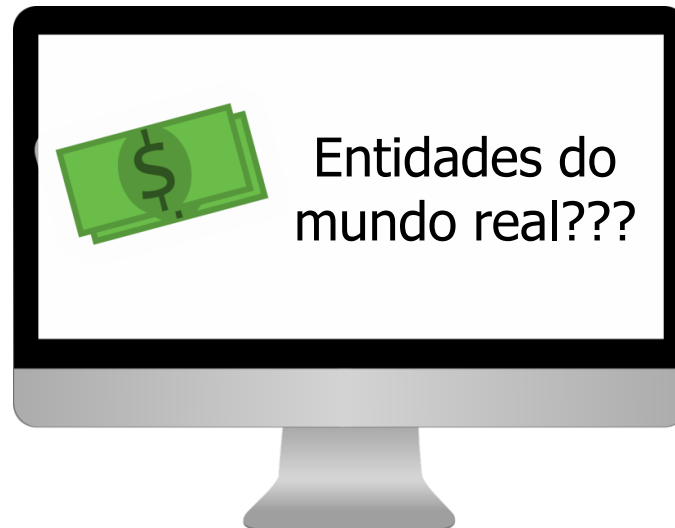
Abstração de Dados



- Não é possível manipular diretamente entidades do mundo real no computador.



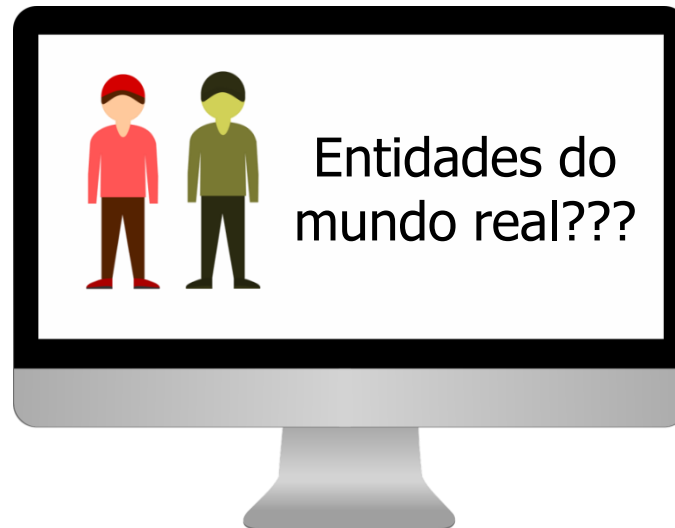
Abstração de Dados



- Exemplo: dinheiro em espécie.



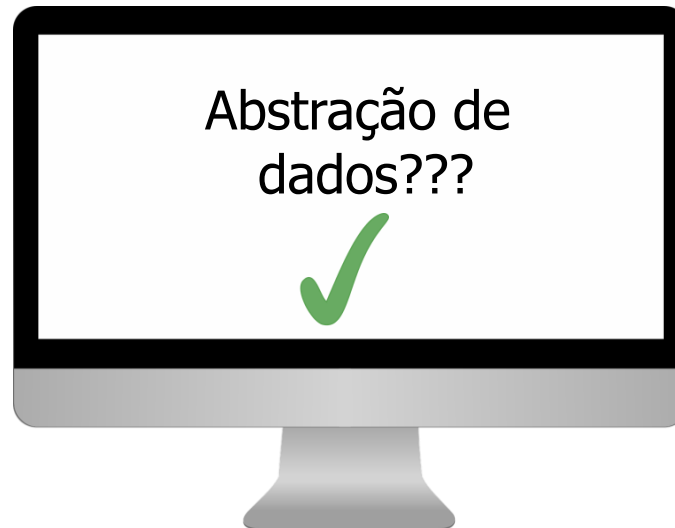
Abstração de Dados



- Exemplo: pessoas reais.



Abstração de Dados



- Solução: usar abstração de dados.



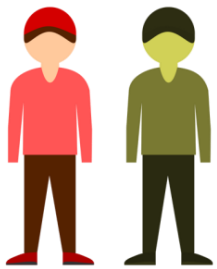
Abstração de Dados

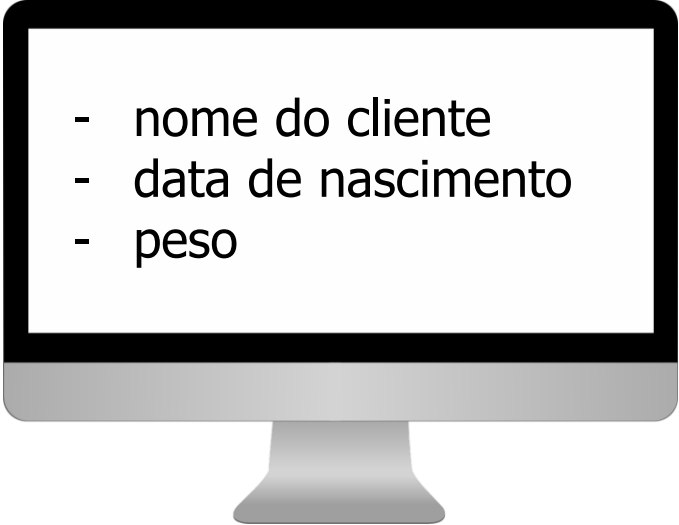
- Definição:

“Habilidade de concentrar nos **aspectos essenciais** de um contexto qualquer, ignorando características menos importantes.”



Abstração de Dados



- 
- nome do cliente
 - data de nascimento
 - peso

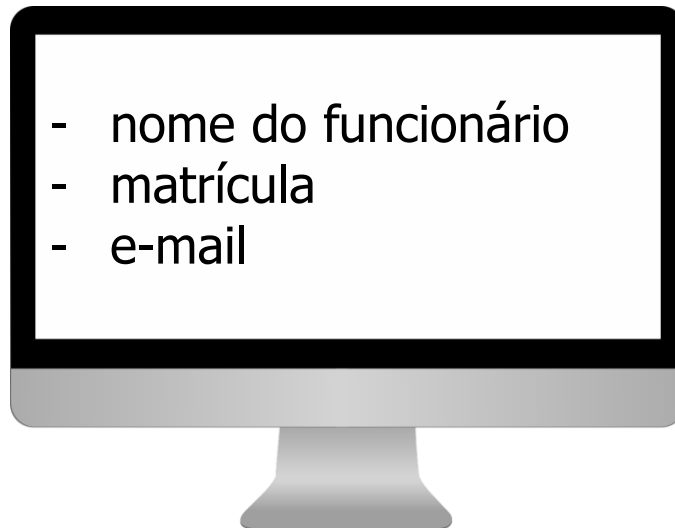
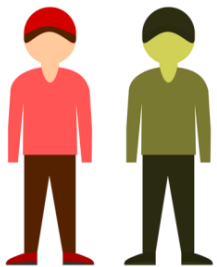
- Contexto: Academia



- Aplicando abstração de dados (exemplo 1).



Abstração de Dados



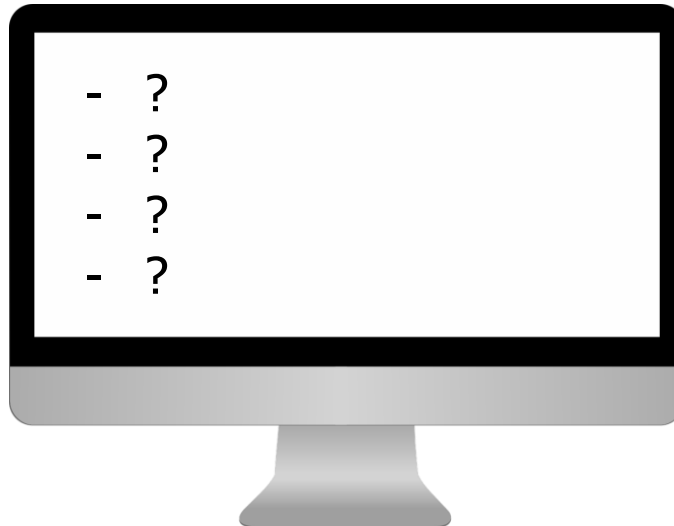
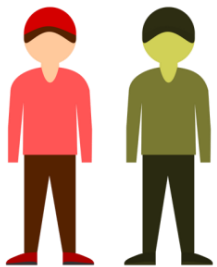
- Contexto: Universidade



- Aplicando abstração de dados (exemplo 2).



Abstração de Dados



- Contexto: Empresa



- Que informações do funcionário seriam necessárias para uma empresa de tecnologia?



O que é uma Classe?

- Representa um grupo de objetos com características (atributos) e comportamentos (métodos) semelhantes.
- Exemplo 1:





O que é uma Classe?

- Representa um grupo de objetos com características (atributos) e comportamentos (métodos) semelhantes.
- Exemplo 2:

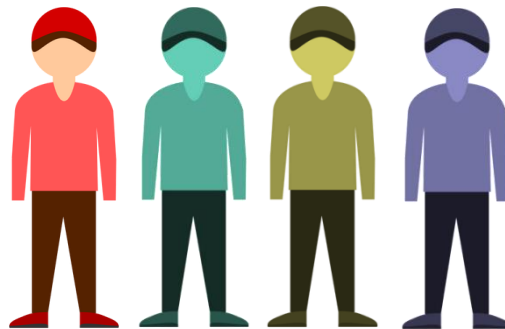




O que é uma Classe?

- Representa um grupo de objetos com características (atributos) e comportamentos (métodos) semelhantes.

Funcionário





Classe

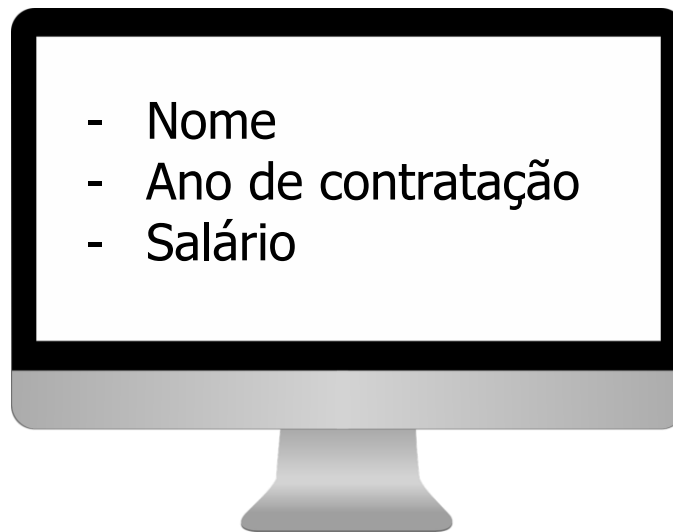
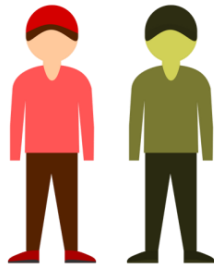
```
public class <Nome da classe>{  
  
    // Atributos  
    // Métodos  
  
}
```

- Convenção:
 - ✓ Nome simples: inicial em maiúsculo
 - ✓ Nome composto: inicial em maiúsculo e demais palavras também com inicial em maiúsculo
 - ✓ Exemplos: Carro, DataDeNascimento



Atributos da Classe

- Quais são as características dos funcionários de uma empresa de tecnologia?





Atributos da Classe

- Atributos: características do objeto.

```
public class <Nome da classe>{  
  
    // Atributos  
    ...  
  
}
```



Atributos da Classe

- Declaração de atributos:

<modif_acesso> <tipo> <identificador>;

- Modificador de acesso: visibilidade do atributo
 1. Público (*public*): visível a todas as classes
 2. Privado (*private*): visível somente na classe em que está
 3. Protegido (*protected*): visível na classe em que está e na classe pai (aula de Herança)

Obs: Por enquanto, **não** incluiremos o modificador de acesso na declaração dos atributos.



Atributos da Classe

- Declaração de atributos:

`<modif_acesso> <tipo> <identificador>;`

- Tipo: valor a ser armazenado no atributo
 - Tipo primitivo: *int, char, double, float*, etc.
 - Nome de classes:
 - ✓ Classes já existentes: *String, Math*, etc.
 - ✓ Classes novas: *Funcionario, Carro*, etc.



Atributos da Classe

- Declaração de atributos:

`<modif_acesso> <tipo> <identificador>;`

- Identificador: nome do atributo

- Convenção:

- ✓ Nome simples: inicial em minúsculo
- ✓ Nome composto: inicial em minúsculo e demais palavras com inicial em maiúsculo
- ✓ Exemplos: ano, anoC, anoDeContratacao



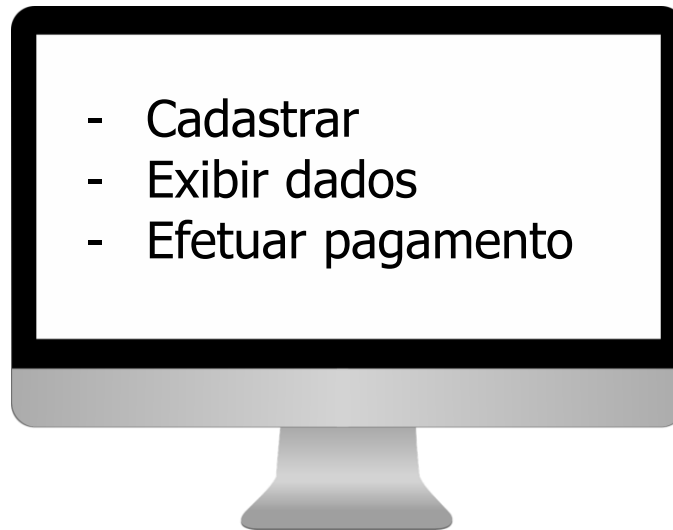
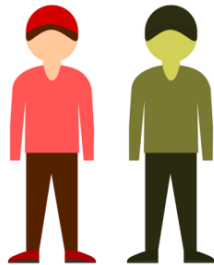
Atributos - classe Funcionário

```
public class Funcionario{  
    // Atributos  
    String nome;  
    int ano;  
    double salario;  
}
```



Métodos da Classe

- Quais são as comportamentos dos objetos da classe Funcionário?





Métodos da Classe

- Métodos: comportamentos do objeto.

```
public class <Nome da classe>{  
  
    // Atributos  
    // Métodos  
  
}
```



Métodos da Classe

- Declaração de métodos

<modif_acesso> <tipo> <identificador> (parâmetros) { }

- Modificador de acesso: visibilidade do método

- Em geral, *public*.



Métodos da Classe

- Declaração de métodos

`<modif_acesso> <tipo> <identificador> (parâmetros) { }`

- Tipo: valor retornado pelo método

- Quando o método não retorna nada → usar *void*
- Quando o método retorna um valor
 - ✓ Colocar o tipo do valor retornado (ex.: *int*, *char*, etc)
 - ✓ Incluir a palavra *return* no corpo do método



Métodos da Classe

- Declaração de métodos

`<modif_acesso> <tipo> <identificador> (parâmetros) { }`

- Identificador: nome do método

- Segue as mesmas regras para nome de atributos
- Exemplos: armazenarDados, imprimirValores, etc.



Métodos da Classe

- Declaração de métodos

`<modif_acesso> <tipo> <identificador> (parâmetros) { }`

- Parâmetros: informações passadas para o método



Métodos da Classe

- Declaração de métodos

`<modif_acesso> <tipo> <identificador> (parâmetros) {}`

- Abre e fecha chaves: corpo do método



Métodos da Classe

- Declaração de métodos

`<modif_acesso> <tipo> <identificador> (parâmetros) {}`

```
public void exibirDados()  
{  
    System.out.println("Nome: " + this.nome);  
    System.out.println("Ano: " + this.ano);  
    System.out.println("Salario: "+ this.salario);  
}
```



Classe Funcionário

```
public class Funcionario{  
    // Atributos  
    String nome;  
    int ano;  
    double salario;  
  
    // Métodos  
    public void exibirDados() {...}  
    ...  
}
```



Para que serve esse modelo?



Criar Objetos

- Objeto: representa uma entidade do mundo real.
- Exemplo:



- Nome: Robin
- Ano de contratação: 2015
- Salário: R\$2500,00

Classe x Objetos

Classe - Modelo



- Nome
- Ano de contratação
- Salário

Objetos - Real



- Robin
- 2015
- R\$2500,00



- Hulk
- 2020
- R\$1000,00



- Flash (Rev)
- 2010
- R\$5000,00



- Coringa
- 2020
- R\$1000,00



Exemplo 1

Uma empresa de tecnologia precisa de uma aplicação que **armazene e imprima** os dados de todos os seus funcionários.

- Implementar os métodos:
- Cadastrar
 - Exibir dados



Raciocinando a solução...

```
public class Funcionario
{
    ...
}
```

```
public class Principal
{
    public static void main(String[ ] args)
    {
        ...
    }
}
```

```
public class Funcionario{
```

- Definição da classe Funcionario
- Código fonte deve ser salvo como **Funcionario.java**

```
}
```

```
public class Funcionario{  
    String nome;  
    int ano;  
    double salario;
```

- Definição dos **atributos** da classe Funcionario

```
}
```

```
public class Funcionario{
```

```
    String nome;
```

```
    int ano;
```

```
    double salario;
```

```
    public void cadastrar(String nome, int ano, double salario)
```

```
{
```

```
        this.nome = nome;
```

```
        this.ano = ano;
```

```
        this.salario = salario;
```

```
}
```

```
...
```

- Definição do método cadastrar

```
}
```



Palavra-chave *this*

```
this.nome = nome;  
this.ano = ano;  
this.salario = salario;
```

- Usada para referenciar um atributo da classe.
 - Diferencia uma variável de escopo (parâmetro) de uma variável de classe (atributo).
- Melhora a legibilidade do código.


```
public class Funcionario{  
    String nome;  
    int ano;  
    double salario;
```

```
...
```

- Definição do método
exibirDados

```
public void exibirDados( )  
{  
    System.out.println("Nome: " + this.nome);  
    System.out.println("Ano: " + this.ano);  
    System.out.println("Salario: R$" + this.salario);  
}  
  
}
```



```
public class Funcionario{
    String nome;
    int ano;
    double salario;

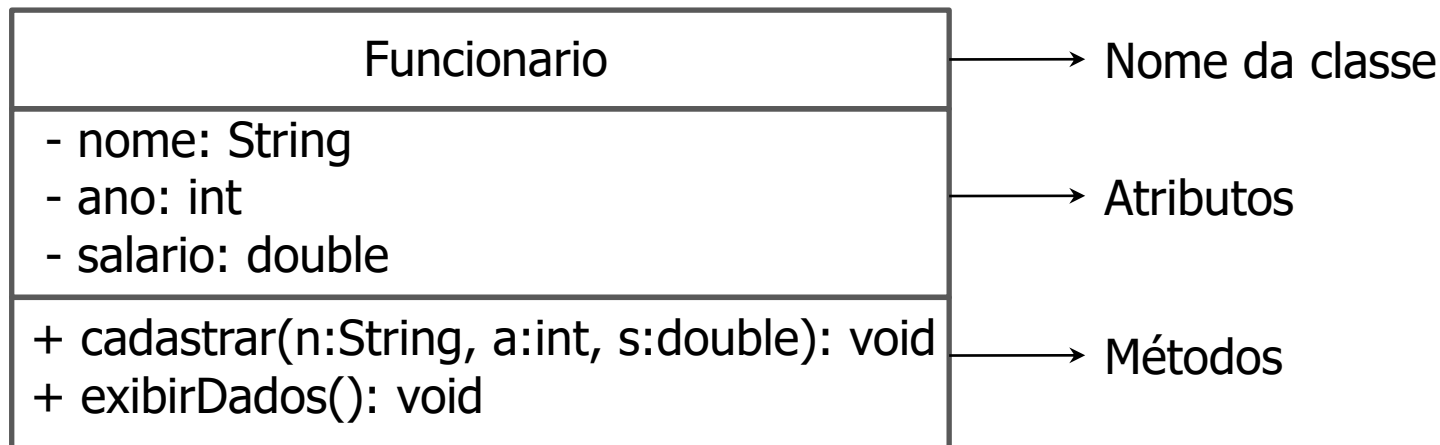
    public void cadastrar(String nome, int ano, double salario) {
        this.nome = nome;
        this.ano = ano;
        this.salario = salario;
    }

    public void exibirDados( ) {
        System.out.println("Nome: " + this.nome);
        System.out.println("Ano: " + this.ano);
        System.out.println("Salario: " + this.salario);
    }
}
```



Representação UML

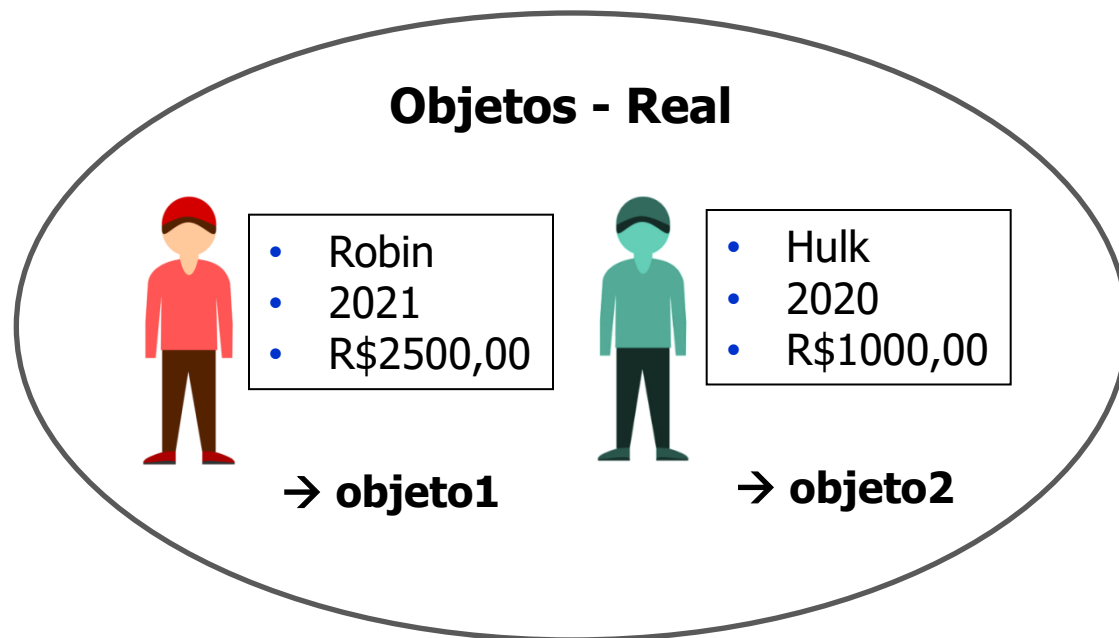
- Define a estrutura das classes de um sistema.
- Classe é representada por um retângulo com 3 divisões:





Classe Principal

- Crie duas instâncias (objetos) da classe funcionário e chame os métodos cadastrar() e exibirDados()



```
public class Principal
{
    public static void main(String[ ] args)
    {
```

- A classe Principal contém o método main() que é responsável por iniciar o programa Java.
- Código fonte deve ser salvo como **Principal.java**

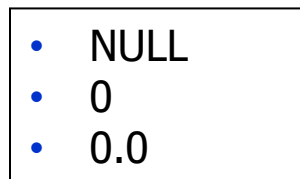
```
}
```

```
}
```

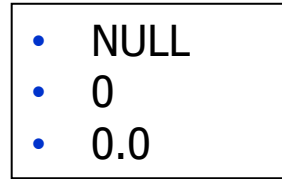
```
public class Principal
{
    public static void main(String[ ] args)
    {
        Funcionario objeto1 = new Funcionario( );
        Funcionario objeto2 = new Funcionario( );

        • Declaração de dois objetos (instâncias)
          da classe Funcionario.

        }
    }
}
```



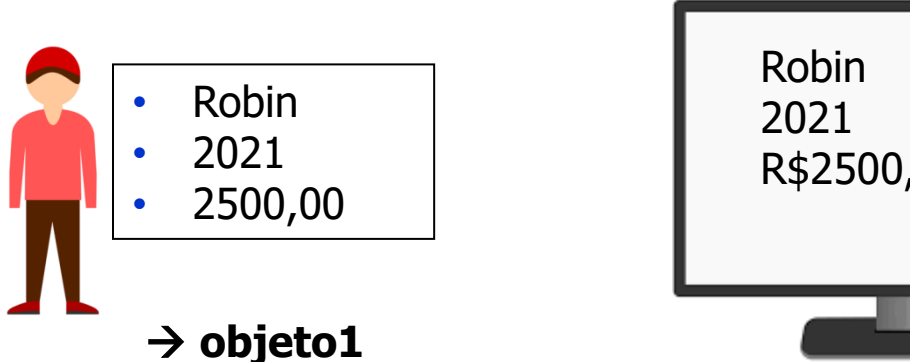
→ **objeto1**



→ **objeto2**

```
public class Principal
{
    public static void main(String[ ] args)
    {
        Funcionario objeto1 = new Funcionario( );
        Funcionario objeto2 = new Funcionario( );

        objeto1.cadastrar("Robin", 2021, 2500);
        objeto1.exibirDados( );
    }
}
```




Robin
2021
R\$2500,00

```
public class Principal
{
    public static void main(String[ ] args)
    {
        Funcionario objeto1 = new Funcionario( );
        Funcionario objeto2 = new Funcionario( );

        objeto1.cadastrar("Robin", 2021, 2500);
        objeto1.exibirDados( );

        objeto2.cadastrar("Hulk", 2020, 1000);
        objeto2.exibirDados();
    }
}
```



• Hulk
• 2020
• 1000

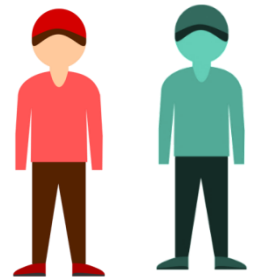
Hulk
2020
R\$1000,00

→ **objeto2**


```
public class Principal
{
    public static void main(String[ ] args)
    {
        Funcionario objeto1 = new Funcionario( );
        Funcionario objeto2 = new Funcionario( );

        objeto1.cadastrar("Robin", 2021, 2500);
        objeto1.exibirDados( );

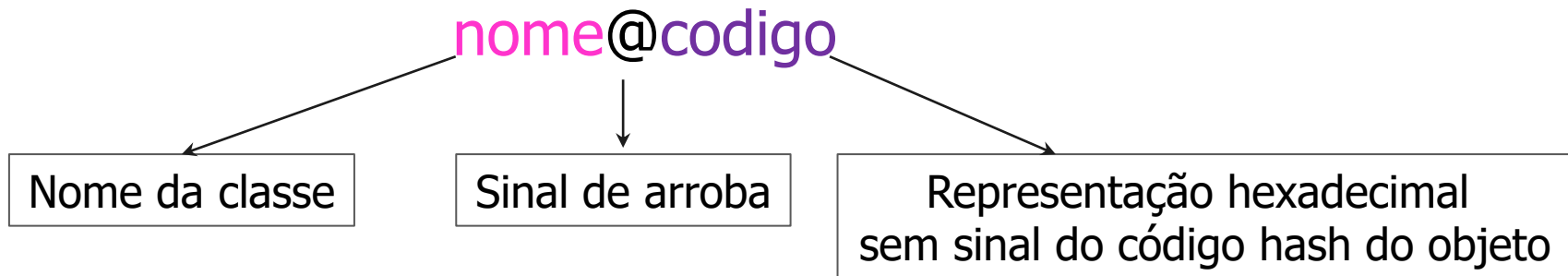
        objeto2.cadastrar("Hulk", 2020, 1000);
        objeto2.exibirDados( );
    }
}
```





Método toString()

- Retorna a representação textual de um objeto
- Formato:



```
public class Principal{  
    public static void main(String[ ] args){  
        Funcionario objeto1 = new Funcionario( );  
        Funcionario objeto2 = new Funcionario( );  
  
        objeto1.cadastrar("Robin", 2021, 2500);  
        objeto1.exibirDados( );  
        System.out.println(objeto1.toString());  
  
        objeto2.cadastrar("Hulk", 2020, 1000);  
        objeto2.exibirDados(objeto2.toString());  
    }  
}
```



Funcionario@53d8d10a



Funcionario@e9e54c2

Altere o código abaixo para **criar um vetor** com duas instâncias (objetos) da classe Funcionário e, em seguida, chame os métodos cadastrar e exibirDados.

```
public class Principal{
    public static void main(String[ ] args)    {
        Funcionario objeto1 = new Funcionario( );
        Funcionario objeto2 = new Funcionario( );

        objeto1.cadastrar("Robin", 2021, 2500);
        objeto1.exibirDados( );

        objeto2.cadastrar("Hulk", 2020, 1000);
        objeto2.exibirDados();
    }
}
```

```
public class Principal1
{
    public static void main(String[ ] args)
    {
```

- A classe Principal1 contém o método main() que é responsável por iniciar o programa Java.
- Código fonte deve ser salvo como **Principal1.java**

```
}
```

```
}
```

```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];
```

- Criando um vetor com duas posições

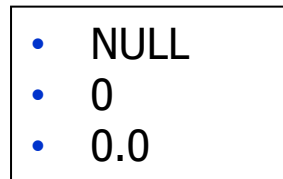
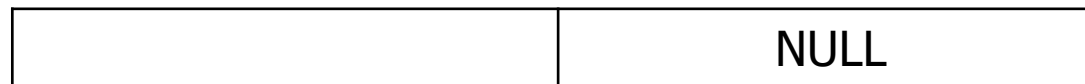
NULL	NULL
------	------

```
}
```

```
}
```

```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];  
        vetObjetos[0] = new Funcionario();  
    }  
}
```

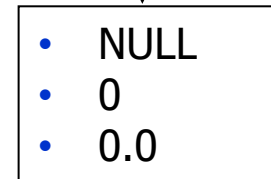
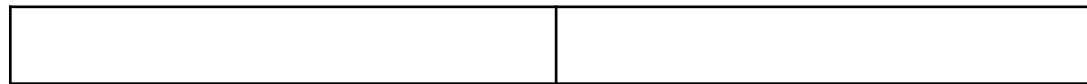
- Instanciando o primeiro objeto



vetObjetos[0]

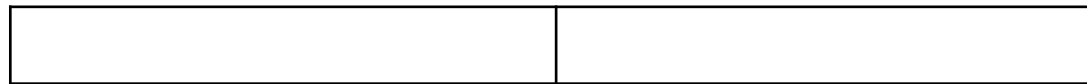
```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];  
        vetObjetos[0] = new Funcionario();  
        vetObjetos[1] = new Funcionario();  
    }  
}
```

- Instanciando o segundo objeto



vetObjetos[1]

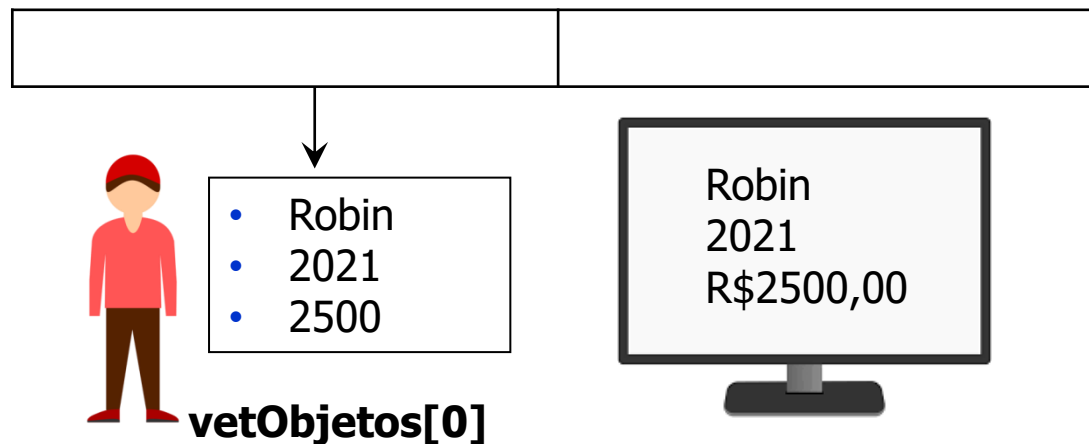

```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];  
        vetObjetos[0] = new Funcionario();  
        vetObjetos[1] = new Funcionario();  
        vetObjetos[0].cadastrar("Robin", 2021, 2500);  
    }  
}
```



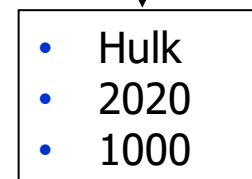
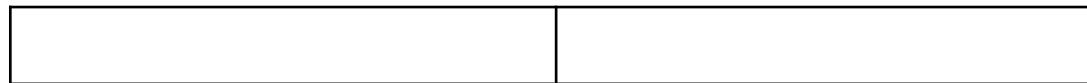
- Robin
- 2021
- 2500

vetObjetos[0]

```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];  
  
        vetObjetos[0] = new Funcionario();  
        vetObjetos[1] = new Funcionario();  
  
        vetObjetos[0].cadastrar("Robin", 2021, 2500);  
        vetObjetos[0].exibirDados();  
    }  
}
```

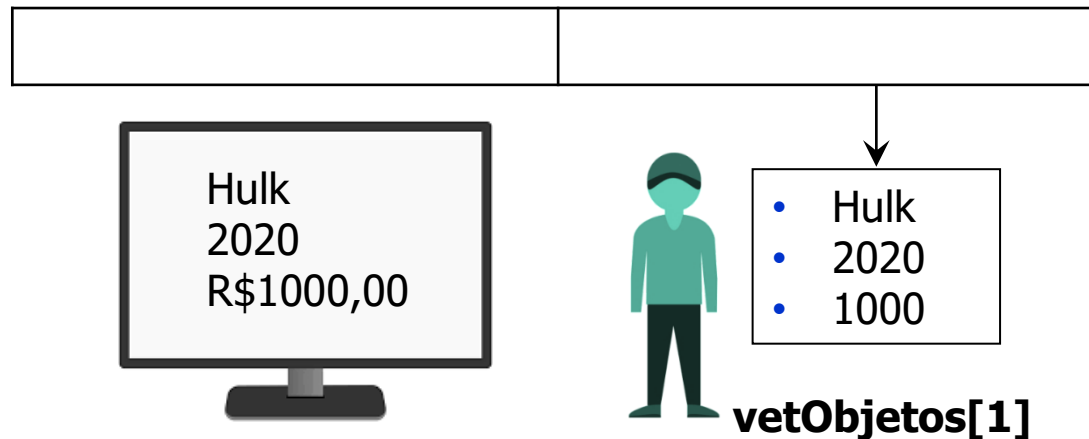


```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];  
  
        vetObjetos[0] = new Funcionario();  
        vetObjetos[1] = new Funcionario();  
  
        vetObjetos[0].cadastrar("Robin", 2021, 2500);  
        vetObjetos[0].exibirDados();  
  
        vetObjetos[1].cadastrar("Hulk", 2020, 1500);  
  
    }  
}
```



vetObjetos[1]

```
public class Principal{  
    public static void main(String[ ] args)  
    {  
        Funcionario[ ] vetObjetos = new Funcionario[2];  
  
        vetObjetos[0] = new Funcionario();  
        vetObjetos[1] = new Funcionario();  
  
        vetObjetos[0].cadastrar("Robin", 2021, 2500);  
        vetObjetos[0].exibirDados();  
  
        vetObjetos[1].cadastrar("Hulk", 2020, 1500);  
        vetObjetos[1].exibirDados();  
    }  
}
```





Praticar...

- Implemente uma `telefone` com os atributos: código do país (DDI), código de área (DDD), número e métodos para cadastrar e exibir.
- Implemente um programa que leia cinco número de telefones e armazene em um vetor. Lembrando que os códigos de país válidos são: 55 (Brasil), 1 (Estados Unidos) e 61 (Austrália). Na sequência, exiba os números de telefone no seguinte formato: +DDI (DDD) número.



Referências

- Deitel, P. J.; Deitel, H. M. (2017). Java como programar. 10a edição. São Paulo: Pearson Prentice Hall.
- Boratti, I. C. (2007). Programação orientada a objetos em Java. Florianópolis, SC: Visual Books.
- Bezerra, E. "Princípios de Análise e Projeto de Sistemas com UML". Editora Campus, 2002.