

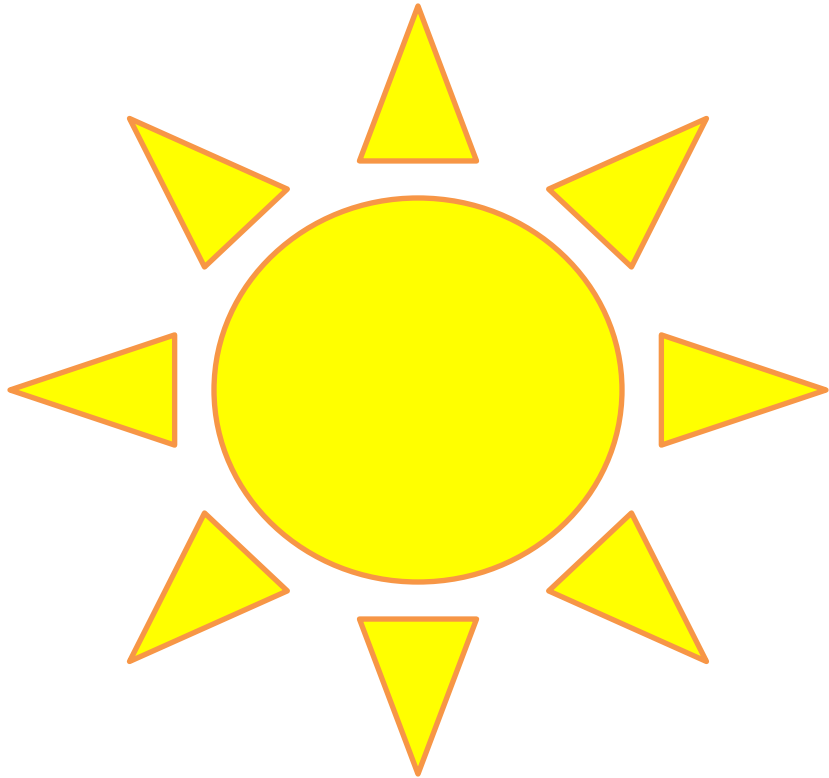


Universidade Federal do ABC  
Centro de Matemática, Computação e Cognição

# Programação Orientada a Objetos

Monael Pinheiro Ribeiro, D.Sc.

O que é isso?



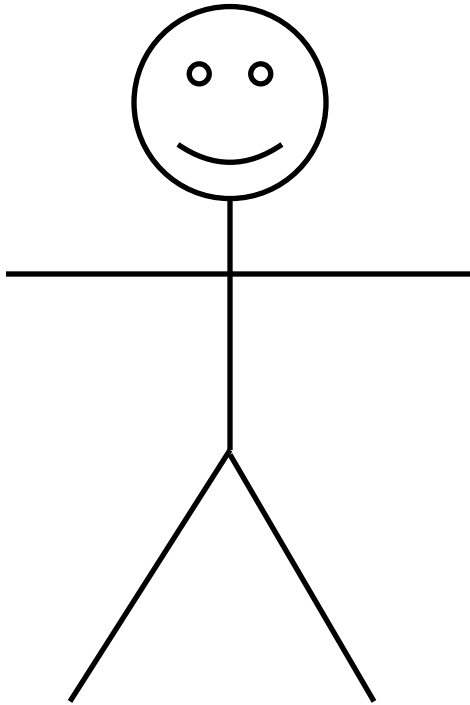
O que é isso?



O que é isso?



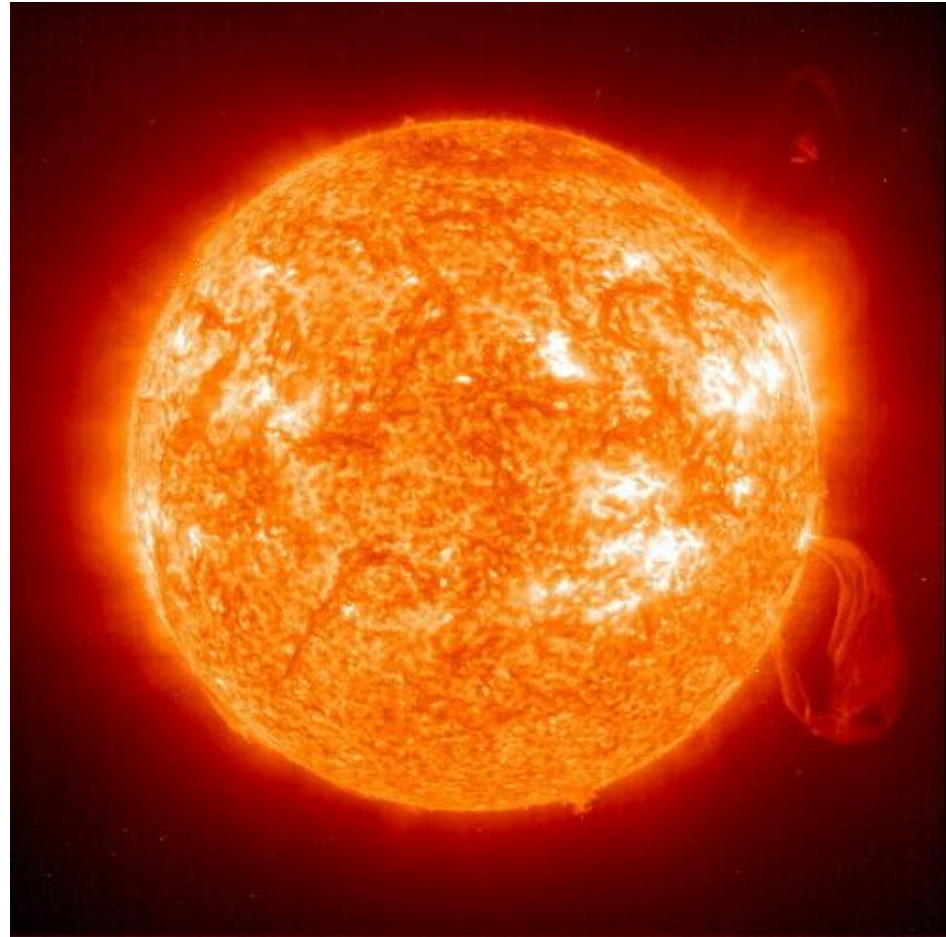
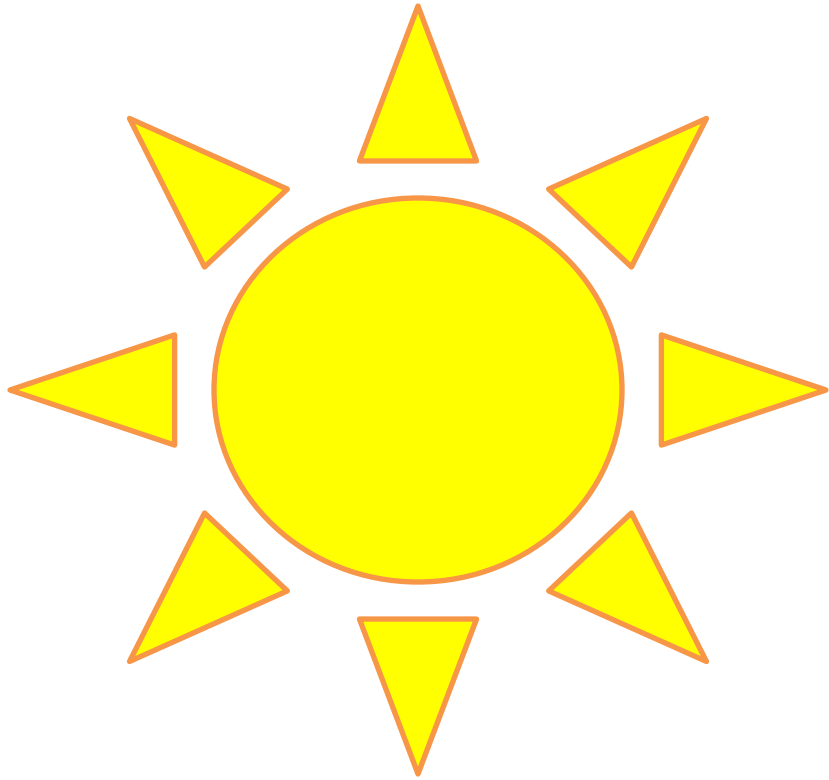
O que é isso?



# Abstração

- Abstrair é:
  - Capacidade e habilidade em ater-se aos aspectos essenciais em um contexto.
  - Isolar o que importa e desprezar as características menos importantes.

# Abstração



# Abstração

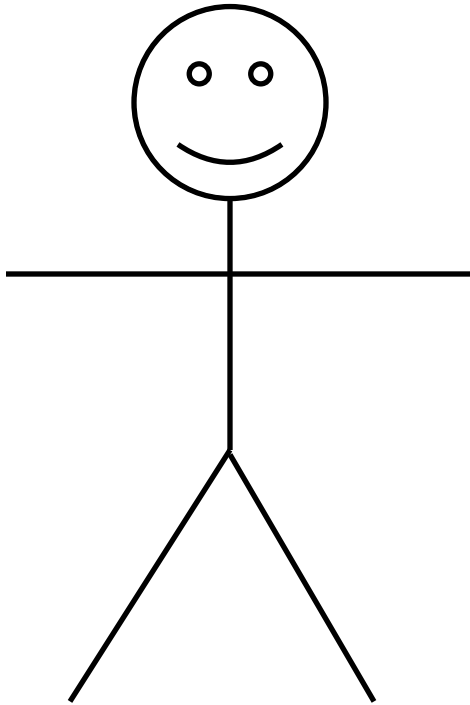




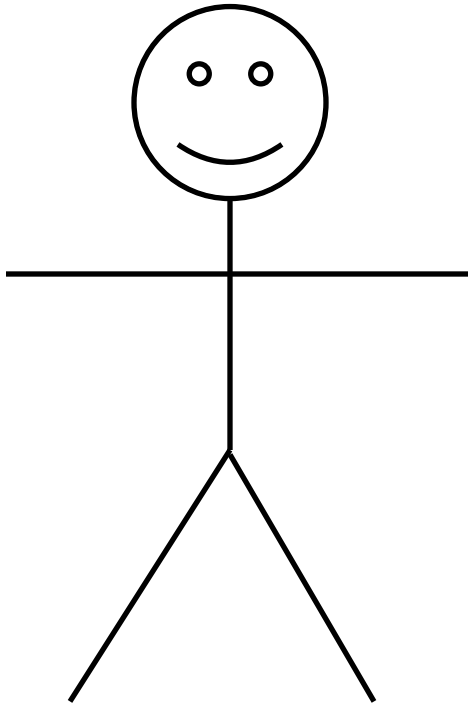
# Abstração



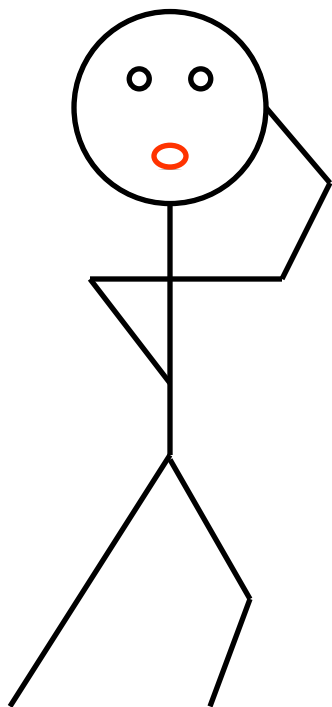
# Abstração



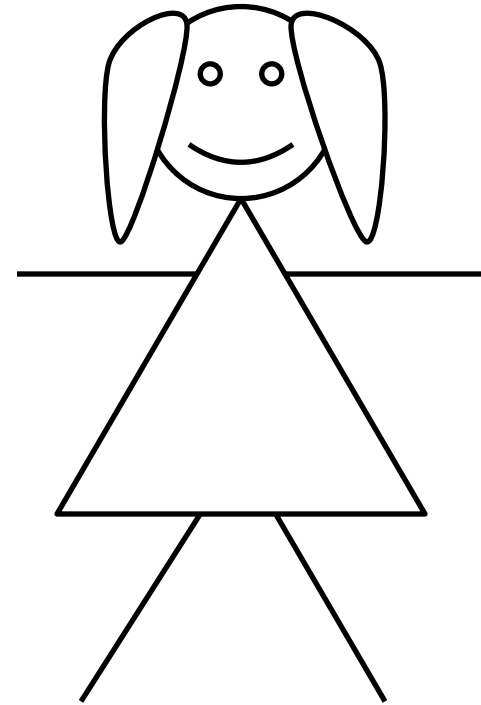
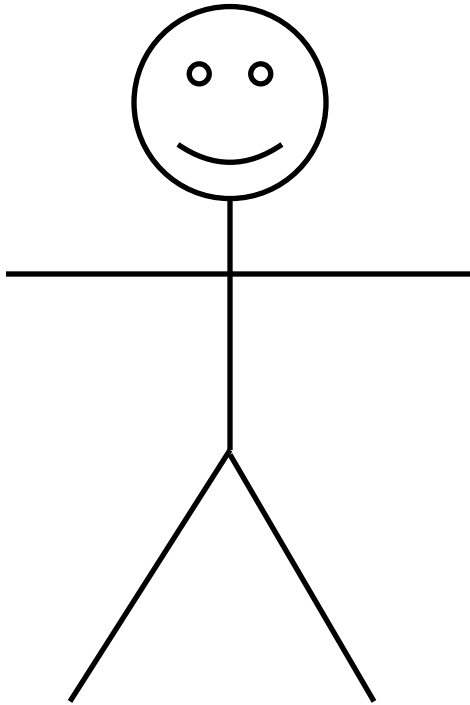
# Abstração ?



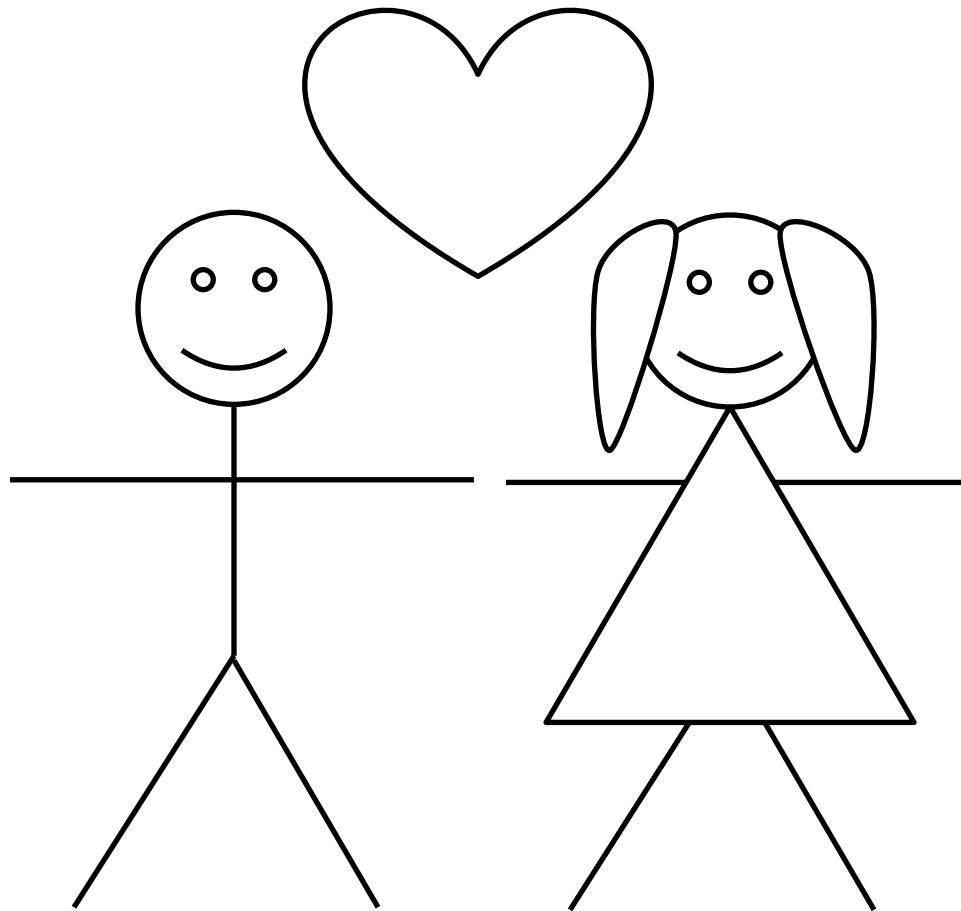
# Abstração



# Abstração



# Abstração

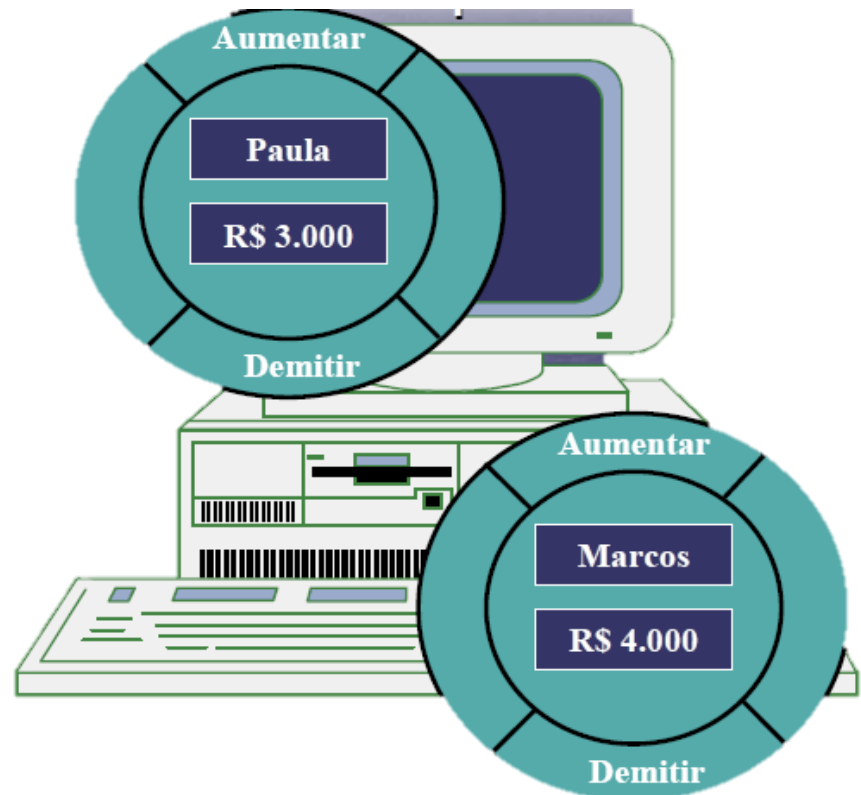


# Abstração em POO

- Abstração em **AOO**, uma **classe** é uma abstração de uma entidade existente no domínio de um problema que está sendo modelado através de um sistema computacional:

# Abstração em POO

- Abstração em **AOO**, uma **classe** é uma abstração de uma entidade existente no domínio de um problema que está sendo modelado através de um sistema computacional:



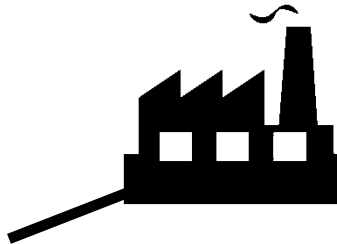


# Introdução à POO

- POO é uma técnica para montagens de programas que intenta imitar o mundo real;
- Assim os programas se tornam mais reutilizáveis, confiáveis e inteligíveis;
- Um programa desenvolvido de acordo com a OO incorpora os seguintes princípios:
  - Organização dos programas em elementos chamados **CLASSES**;
  - Aprendizado de como as **CLASSES** são usados para criar **OBJETOS**;
  - Definição de uma **CLASSE** por dois aspectos:
    - Como ela deve se **comportar (o que ela faz)**;
    - Quais são seus **atributos (qualidades, características)**.
  - Conexão entre as **CLASSES** de modo que uma **CLASSE herde** características de outras **CLASSES**.

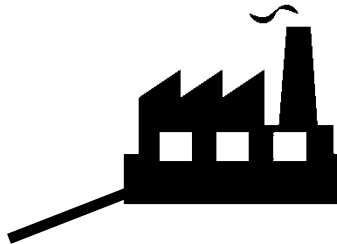
# Classes

- O que é uma Classe?
  - É um modelo (uma matriz) usada para criar um objeto. Cada objeto criado a partir de uma mesma classe terá características semelhantes ou mesmo idênticas.
  - Incorpora todas as características de um conjunto de objetos específicos, que são os **ATRIBUTOS** ou Dados Membro.
  - Também é incorporada na classe as funções que operam sobre seus atributos, que são os **MÉTODOS** ou Funções Membro.
  - Exemplos de Classes:



# Classes

- O que é uma Classe?
  - É um modelo (uma matriz) usada para criar um objeto. Cada objeto criado a partir de uma mesma classe terá características semelhantes ou mesmo idênticas.
  - Incorpora todas as características de um conjunto de objetos específicos, que são os **ATRIBUTOS** ou Dados Membro.
  - Também é incorporada na classe as funções que operam sobre seus atributos, que são os **MÉTODOS** ou Funções Membro.
  - Exemplos de Classes:
    - Veículo
    - Árvore
    - Aluno
    - Move1
    - Queue (Fila)
    - Stack (Pilha)
    - LinkedList (Lista)
    - Panel
    - Button
    - ComboBox
    - TextField



# Classes

- Definição de uma Classe em C++

```
class PrimeiraClasse
{

};

int main()
{
    return 0;
}
```

# Classes

- Definição de uma Classe em JAVA

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

# Classes

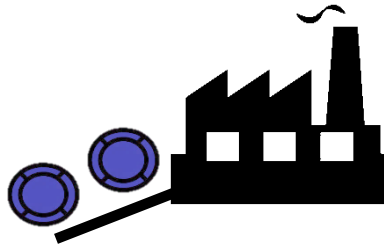
- Definição de uma Classe em JAVA

```
class PrimeiraClasse  
{  
}
```

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

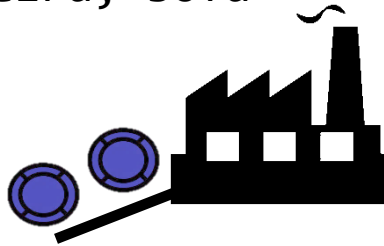
# Objetos

- O que são Objetos?
  - É um elemento autocontido de um programa, que representa um grupo relacionado de recursos e está projetado para realizar tarefas específicas;
  - Possui um contexto estático e um contexto dinâmico;
  - Exemplos de Objetos:



# Objetos

- O que são Objetos?
  - É um elemento autocontido de um programa, que representa um grupo relacionado de recursos e está projetado para realizar tarefas específicas;
  - Possui um contexto estático e um contexto dinâmico;
  - Exemplos de Objetos:
    - carro, navio, avião
    - pinheiro, cajuzeiro, laranjeira
    - maria, jose, pedro
    - mesa, cadeira, sofá

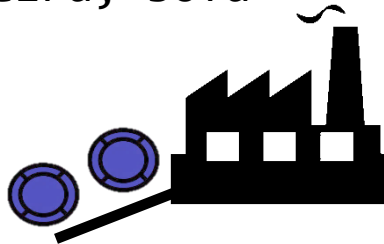
A screenshot of a Windows-style dialog box titled "Conta - Alterar Conta". It contains several input fields and buttons. The fields are: "Número:" with the value "3483090", "Títular:" with the value "Fernando da Silva e Silva", "CPF:" with the value "75963214566", "Tipo:" with a dropdown menu showing "Poupança", and "Saldo: R\$" with the value "9000". At the bottom, there are two buttons: "Ok" and "Cancelar".

Número:	3483090
Títular:	Fernando da Silva e Silva
CPF:	75963214566
Tipo:	Poupança
Saldo: R\$	9000



# Objetos

- O que são Objetos?
  - É um elemento autocontido de um programa, que representa um grupo relacionado de recursos e está projetado para realizar tarefas específicas;
  - Possui um contexto estático e um contexto dinâmico;
  - Exemplos de Objetos:
    - carro, navio, avião
    - pinheiro, cajuzeiro, laranjeira
    - maria, jose, pedro
    - mesa, cadeira, sofá



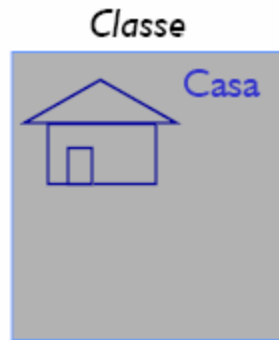
Um **objeto** é uma instância de uma classe.

A screenshot of a Windows-style dialog box titled "Conta - Alterar Conta". It contains several input fields and a dropdown menu. The fields are: "Número:" with the value "3483090", "Títular:" with the value "Fernando da Silva e Silva", "CPF:" with the value "75963214566", "Tipo:" with a dropdown menu showing "Poupança", and "Saldo: R\$" with the value "9000". At the bottom are two buttons: "Ok" and "Cancelar".

Número:	3483090
Títular:	Fernando da Silva e Silva
CPF:	75963214566
Tipo:	Poupança
Saldo: R\$	9000

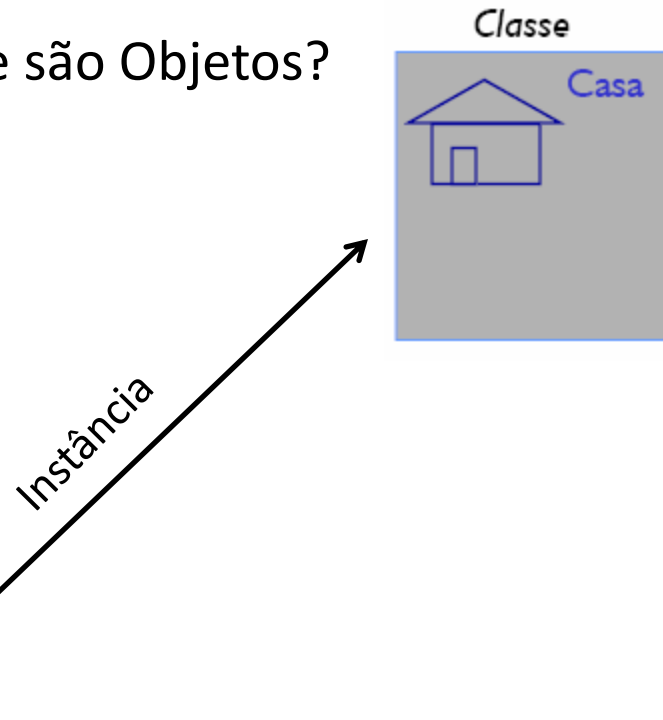
# Objetos

- O que são Objetos?



# Objetos

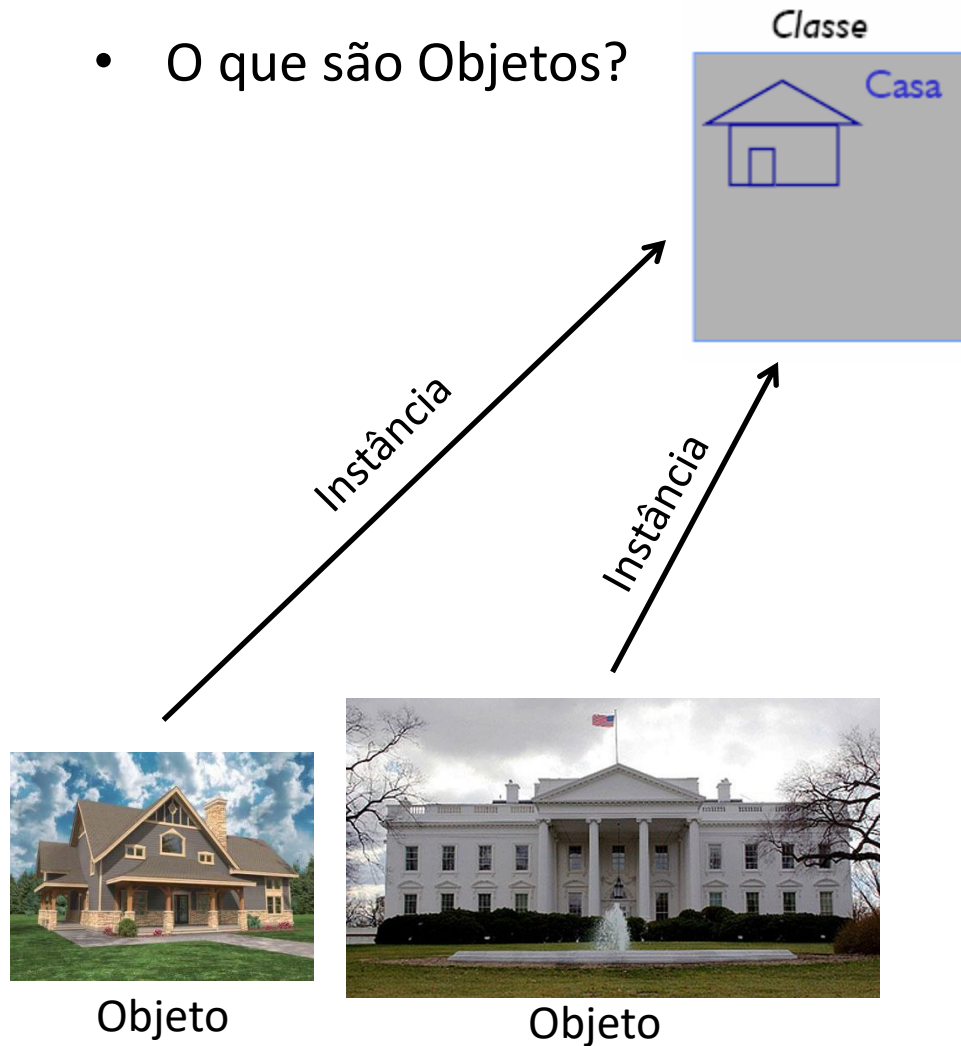
- O que são Objetos?



Objeto

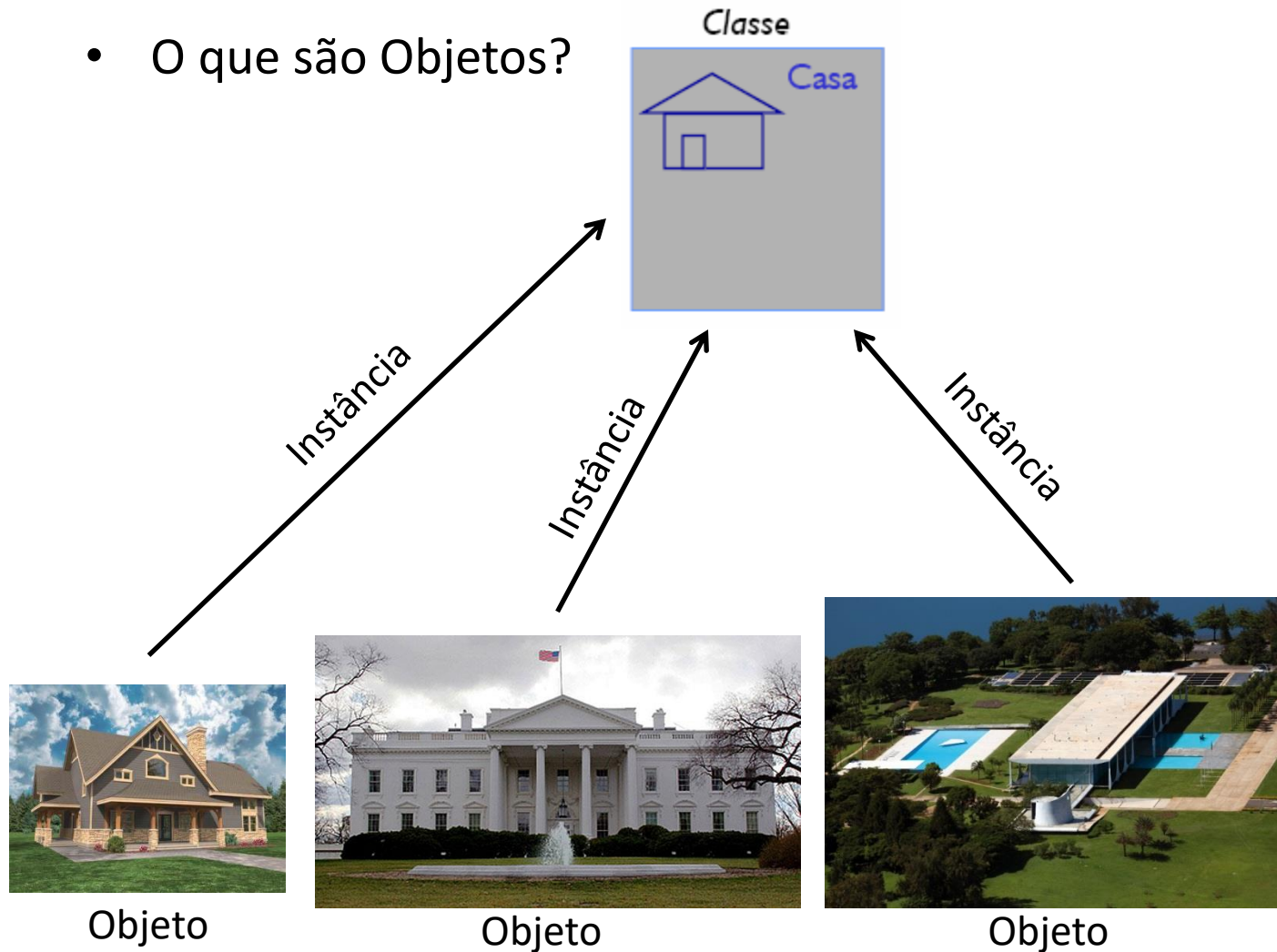
# Objetos

- O que são Objetos?



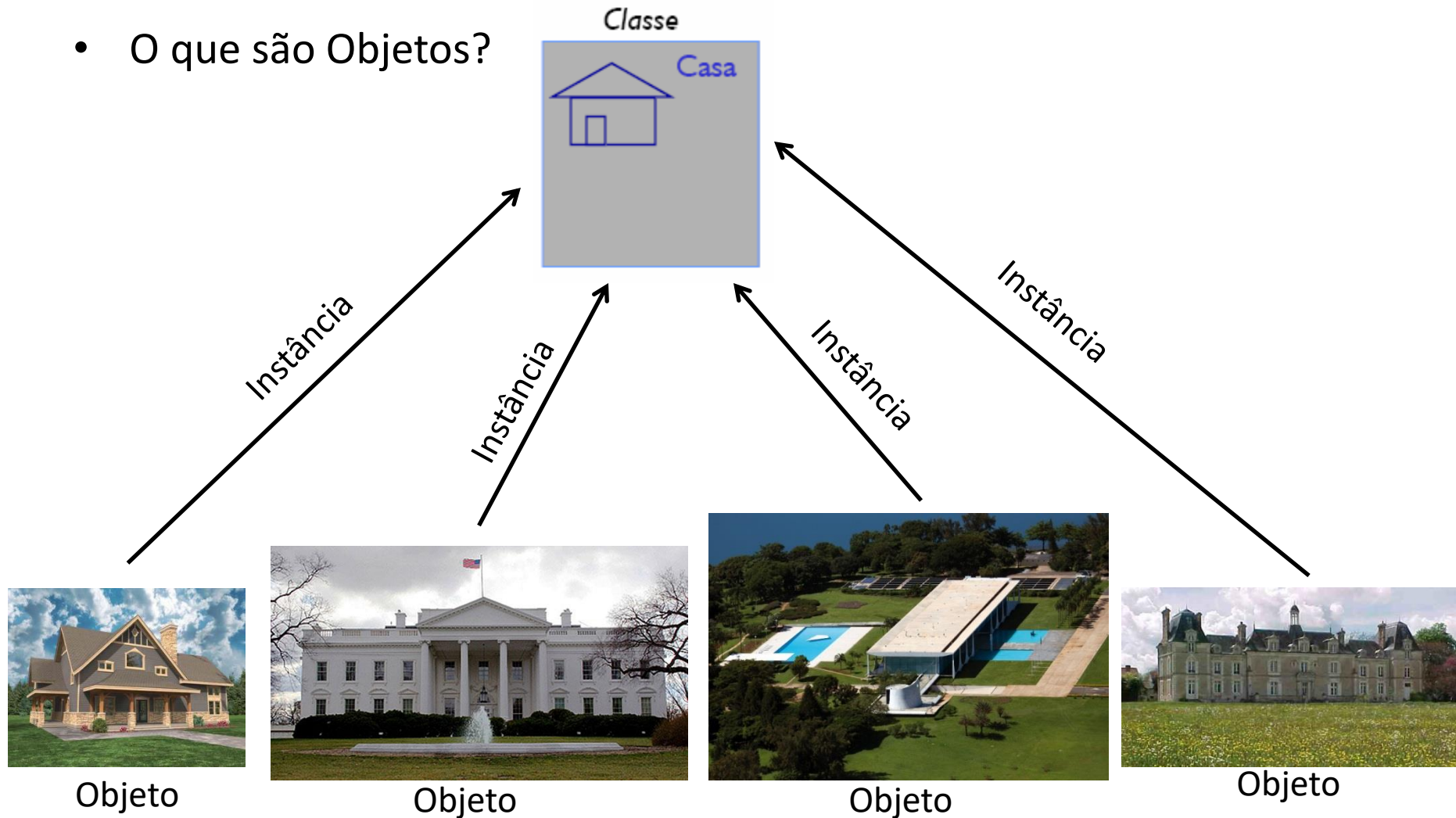
# Objetos

- O que são Objetos?



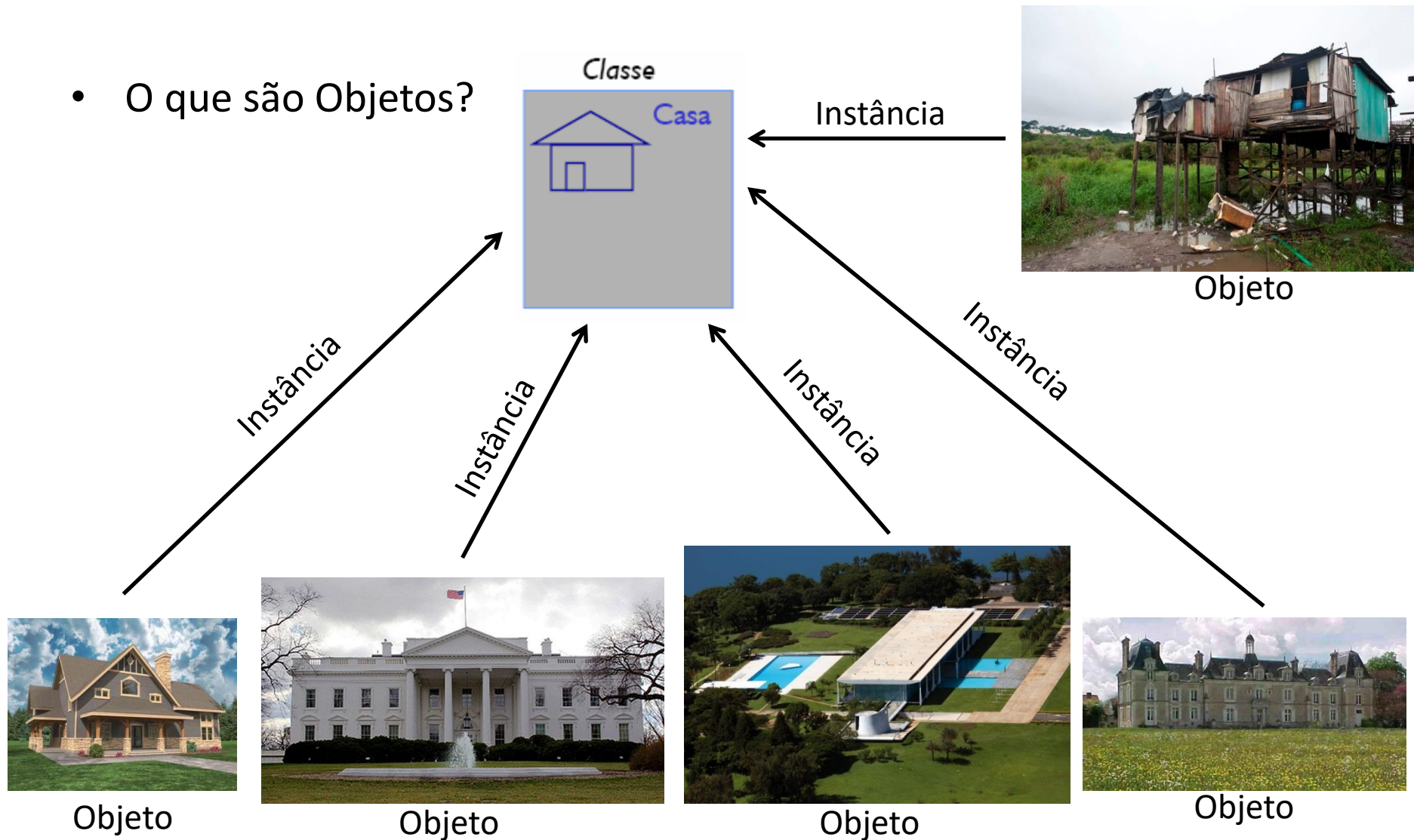
# Objetos

- O que são Objetos?



# Objetos

- O que são Objetos?



# Objetos

- Criação de um objeto em C++

```
class PrimeiraClasse  
{  
  
};
```

```
int main()  
{  
    PrimeiraClasse primeiroObjeto;  
    PrimeiraClasse segundoObjeto, terceiroObjeto;  
    return 0;  
}
```



# Objetos

- Operador new

- É usado para criar um novo objeto.

- Sintaxe:

```
<identificador da classe> <identificador do objeto> = new <identificador da classe>([lista de argumentos]);
```

Em C++ o operador **new** retorna um ponteiro para o objeto criado.

- Exemplo:

```
Data *aniversario = new Data(28, 2, 1951);
```

```
Casa *casa1 = new Casa();
```

```
Queue *fila = new Queue();
```

```
Queue *fila2 = new Queue(it);
```

# Objetos

- Operador new

- É usado para criar um novo objeto.

- Sintaxe:

`<identificador da classe> <identificador do objeto> = new <identificador da classe>([lista de argumentos]);`

Em JAVA não existe ponteiro. Mas perceba que todo objeto é uma referência.

- Exemplo:

```
Data aniversario = new Data(28, 2, 1951);
```

```
Casa casa1 = new Casa();
```

```
Queue fila = new Queue();
```

```
Queue fila2 = new Queue(it);
```

# Objetos

- **Criação de um objeto em C++ (com ponteiros)**

```
class PrimeiraClasse  
{  
  
};
```

```
int main()  
{  
    PrimeiraClasse *primeiroObjeto = new PrimeiraClasse();  
    PrimeiraClasse *segundoObjeto, *terceiroObjeto;  
    segundoObjeto = new PrimeiraClasse();  
    terceiroObjeto = new PrimeiraClasse();  
    return 0;  
}
```

# Objetos

- **Criação de um objeto em JAVA**

```
public class PrimeiraClasse
{
    public static void main(String[] args)
    {
        PrimeiraClasse primeiroObjeto = new PrimeiraClasse();
        PrimeiraClasse segundoObjeto, terceiroObjeto;

        segundoObjeto = new PrimeiraClasse();
        terceiroObjeto = new PrimeiraClasse();
    }
}
```

# Objetos

- **Criação de um objeto em JAVA**

```
class PrimeiraClasse  
{  
}
```

```
public class Principal  
{  
    public static void main(String[] args)  
    {  
        PrimeiraClasse primeiroObjeto = new PrimeiraClasse();  
        PrimeiraClasse segundoObjeto, terceiroObjeto;  
  
        segundoObjeto = new PrimeiraClasse();  
        terceiroObjeto = new PrimeiraClasse();  
    }  
}
```

# Objetos

- Isso não é tão novo ... Lembre-se

# Objetos

- Isso não é tão novo ... Lembre-se

```
import java.util.Scanner;

public class Leitor
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
    }
}
```

# Objetos

- Isso não é tão novo ... Lembre-se

```
import java.util.Scanner;

public class Vetor
{
    public static void main(String[] args)
    {
        int vetor = new int[100];
    }
}
```



# Atributos

- **Atributos** (*Dados-Membro*)
  - São os dados que diferenciam um objeto do outro. Eles pode ser usados para determinar a aparência, o estado e outras qualidades dos objetos que pertencem a essa classe.

# Atributos

- **Definição de Atributos em C++**

```
class Data
{
    int dia, mes, ano;
};
```

```
int main()
{
    Data aniversario;
    return 0;
}
```

# Atributos

- **Definição de Atributos em JAVA**

```
class Data
{
    int dia, mes, ano;
}
```

```
public class Principal
{
    public static void main(String[] args)
    {
        Data aniversario = new Data();
    }
}
```

# Atributos

- **Acessando Atributos de um objeto (C++ e JAVA)**

- Para se acessar as variáveis de um objeto, usa-se o operador (.) ponto.

- Sintaxe:

`<identificador do objeto>.<identificador do atributo>`

- Exemplo:  

	<code>q.first;</code>
	<code>q.last;</code>
<code>aniversario.ano;</code>	<code>s.top;</code>
<code>casa1.cor;</code>	<code>it.next;</code>
<code>obj.nome;</code>	<code>node.left;</code>
	<code>t.root;</code>

# Atributos

- **Acessando Atributos de um objeto (C++)**
  - Para se acessar as variáveis de um objeto, usa-se o operador (.) ponto.
  - Quando temos o ponteiro para o objeto, lembre-se de primeiro resolver o endereço, antes de acessar o atributo.

- Exemplo:

```
(*q).first;  
(*q).last;  
(*s).top;  
(*aniversario).ano;  
(*it).next;  
(*casa1).cor;  
(*node).left;  
(*obj).nome;  
(*t).root;
```

# Atributos

- **Acessando Atributos de um objeto (C++)**
  - Para se acessar as variáveis de um objeto, usa-se o operador (.) ponto.
  - Quando temos o ponteiro para o objeto, lembre-se de primeiro resolver o endereço, antes de acessar o atributo.
  - Ou use o operador de resolução de endereço e acesso a atributo (->) seta.
  - Exemplo:

	q->first;
	q->last;
aniversario->ano;	s->top;
casa1->cor;	it->next;
obj->nome;	node->left;
	t->root;

# Atributos

- **Alterando / Atribuindo valor para um atributo (C++ e JAVA)**

- Acessa-se o atributo com operador (.) ponto e usa-se o operador de atribuição (=) para alterar o valor.

- Sintaxe:

`<identificador do objeto>.<identificador do atributo> = [valor];`

- Exemplo:

```
aniversario.ano = 2016;  
casa1.cor = "verde";  
obj.nome = "Luiz";
```

```
q.first = it;  
q.last = it;  
s.top = novo;  
it.next = it;  
node.left = novo;  
t.root = node;
```

# Métodos

- **Métodos** (*Funções-Membro*)
  - São grupos de instruções relacionados em uma classe de objetos que tratam de uma tarefa. Eles são usados para realizar tarefas específicas em seus próprios objetos.



# Métodos

- **Definição de um método em C++**

```
#include <iostream>
class Porta
{
    bool aberta;
    public:
        void abrir();
        void fechar();
        void situacao();
};
void Porta::abrir()
{
    aberta = true;
}
void Porta::fechar()
{
    aberta = false;
}
void Porta::situacao()
{
    std::cout << "A porta estah " << (aberta?"aberta":"fechada") << std::endl;
}
```

# Métodos

- **Acessando Métodos de um objeto (C++ e JAVA)**

- Para se acessar as variáveis de um objeto, também usa-se o operador (.) ponto.

- Sintaxe:

`<identificador do objeto>.<identificador do método>([lista de argumentos]);`

- Exemplo:

```
aniversario.getAno();  
casa1.setCor("rosa");  
obj.calculaSalario();
```

```
q.enqueue(it);  
it = q.dequeue();  
s.setTop(it);  
it = s.push();  
it.hasNext();  
node.left;  
t.root;
```

# Métodos

- **Acessando Métodos de um objeto (C++)**

- Para se acessar as variáveis de um objeto, também usa-se o operador (.) ponto.
- Caso tenha o ponteiro para o objeto resolva o endereço previamente ou use o operador (->).

- Exemplos:

```
aniversario->getAno();  
(*casa1).setCor("rosa");  
ibj->calculaSalario();
```

```
q->enqueue(it);  
it = (*q).dequeue();  
s->setTop(it);  
it = (*s).push();  
it->hasNext();  
(*node).left;  
t->root;
```

# Métodos

- Definição de um método em C++

```
#include <iostream>
class Porta
{
    bool aberta;
    public:
        void abrir();
        void fechar();
        void situacao();
};
void Porta::abrir()
{
    aberta = true;
}
void Porta::fechar()
{
    aberta = false;
}
void Porta::situacao()
{
    std::cout << "A porta estah " << (aberta?"aberta":"fechada") << std::endl;
}

int main()
{
    Porta sala523;

    sala523.situacao();
    sala523.abrir();
    sala523.situacao();
    sala523.fechar();
    sala523.situacao();

    return 0;
}
```



# Métodos

- Definição de um método em JAVA

```
class Porta
{
    boolean aberta;

    void abrir()
    {
        aberta = true;
    }
    void fechar()
    {
        aberta = false;
    }
    void situacao()
    {
        System.out.println("A porta estah " + (aberta?"aberta":"fechada"));
    }
}
```

```
public class Principal
{
    public static void main(String[] args)
    {
        Porta sala523 = new Porta();
        sala523.situacao();
        sala523.abrir();
        sala523.situacao();
        sala523.fechar();
        sala523.situacao();
    }
}
```



# Auto Referência `this`

- **this**
  - Trata-se de um ponteiro para o próprio objeto.
  - Quaisquer métodos **não estático** do objeto podem acessá-la.

# Auto Referência `this`

- **this**
  - Trata-se de um ponteiro para o próprio objeto.
  - Quaisquer métodos **não estático** do objeto podem acessá-la.
  - Exemplo em C++:

```
#include <iostream>
class Porta
{
    bool aberta;
    public:
        void abrir();
        void fechar();
        void situacao();
};
```

```
void Porta::abrir()
{
    this->aberta = true;
}
```

```
void Porta::fechar()
{
    this->aberta = false;
}
```

```
void Porta::situacao()
{
    std::cout << "A porta estah " << (this->aberta?"aberta":"fechada") << std::endl;
}
```

# Auto Referência this

- **this**
  - Trata-se de um ponteiro para o próprio objeto.
  - Quaisquer métodos **não estático** do objeto podem acessá-la.
  - Exemplo em JAVA:

```
class Porta
{
    boolean aberta;

    void abrir()
    {
        this.aberta = true;
    }
    void fechar()
    {
        this.aberta = false;
    }
    void situacao()
    {
        System.out.println("A porta estah " + (this.aberta?"aberta":"fechada"));
    }
}
```



# Auto Referência `this`

- **`this`**
  - Entendendo melhor ...

# Auto Referência this

- **this**
  - Entendendo melhor ... Em C++

```
#include <iostream>
class Classe
{
    int att1;
    float att2;
    std::string att3;
    public:
        void printEu()
        {
            std::cout <<"Eu: " << this << std::endl;
        }
        void printEle(Classe *that)
        {
            std::cout <<"Ele: " << that << std::endl;
        }
        bool souEu(Classe *that)
        {
            this->printEu();
            this->printEle(that);
            return this == that;
        }
};
```

```
int main()
{
    Classe obj1, obj2;
    std::cout << obj1.souEu(&obj2) << std::endl;
    std::cout << obj1.souEu(&obj1) << std::endl;
    std::cout << obj2.souEu(&obj1) << std::endl;
    std::cout << obj2.souEu(&obj2) << std::endl;
    return 0;
}
```



# Auto Referência this

- **this**
  - Entendendo melhor ... Em JAVA

```
class Classe
{
    private int att1;
    private float att2;
    private String att3;

    public void printEu()
    {
        System.out.println("Eu" + this);
    }
    public void printEle(ExemploThis that)
    {
        System.out.println("Ele" + that);
    }
    public boolean souEu(ExemploThis that)
    {
        this.printEu();
        this.printEle(that);
        return this == that;
    }
}
```

```
public class Principal
{
    public static void main(String[] args)
    {
        Classe obj1, obj2;
        obj1 = new Classe();
        obj2 = new Classe();
        obj1.souEu(obj2);
        obj1.souEu(obj1);
        obj2.souEu(obj1);
        obj2.souEu(obj2);
    }
}
```



# Classes e Objetos

- **Encapsulamento**
  - Restringir a visibilidade dos atributos de um objeto.
  - Transforma os atributos em uma caixa preta, onde somente os métodos do objeto terão acesso.
  - Vantagens:
    - Melhora a Legibilidade do Código
    - Facilita a Manutenção
    - Favorece a Reutilização

# Classes e Objetos

- Encapsulamento

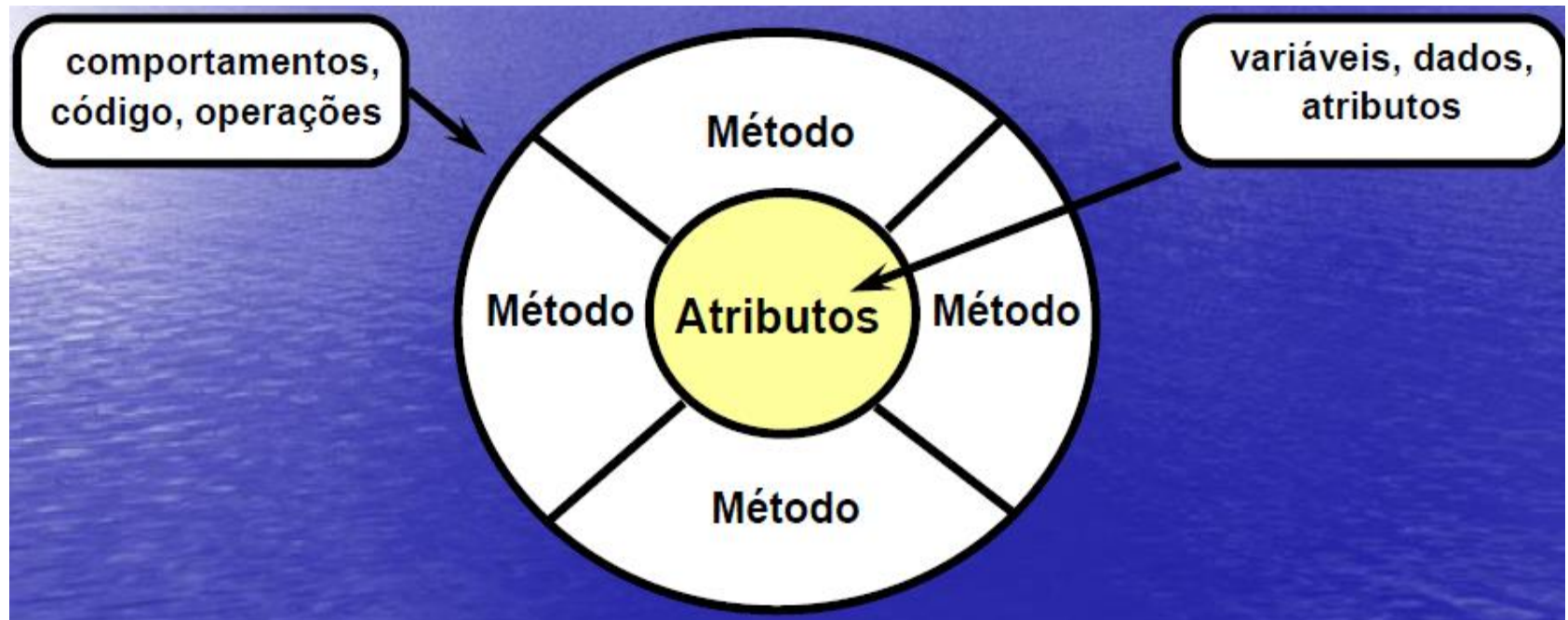
Modelo da Rosquinha



# Classes e Objetos

- Encapsulamento

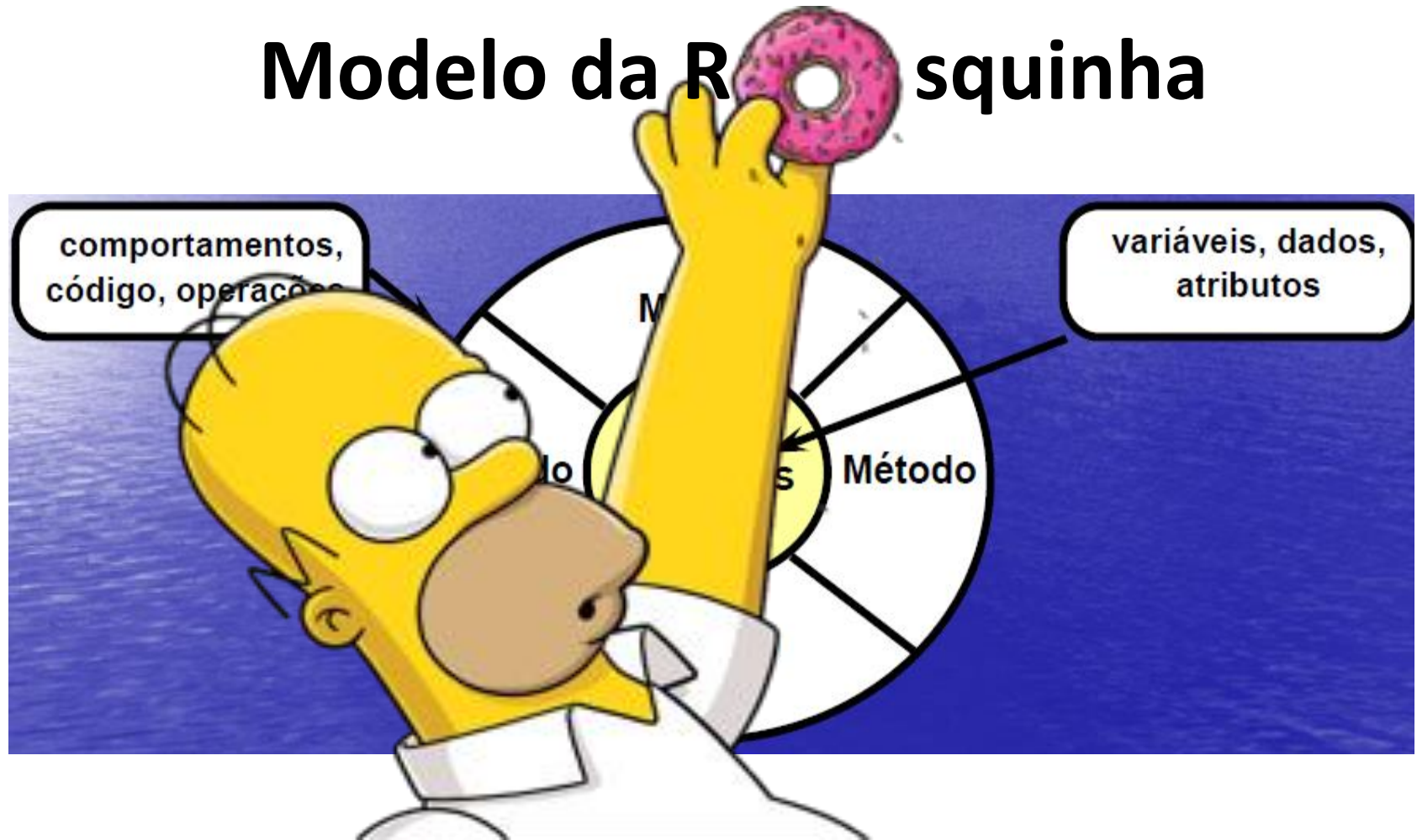
## Modelo da Rosquinha



# Classes e Objetos

- Encapsulamento

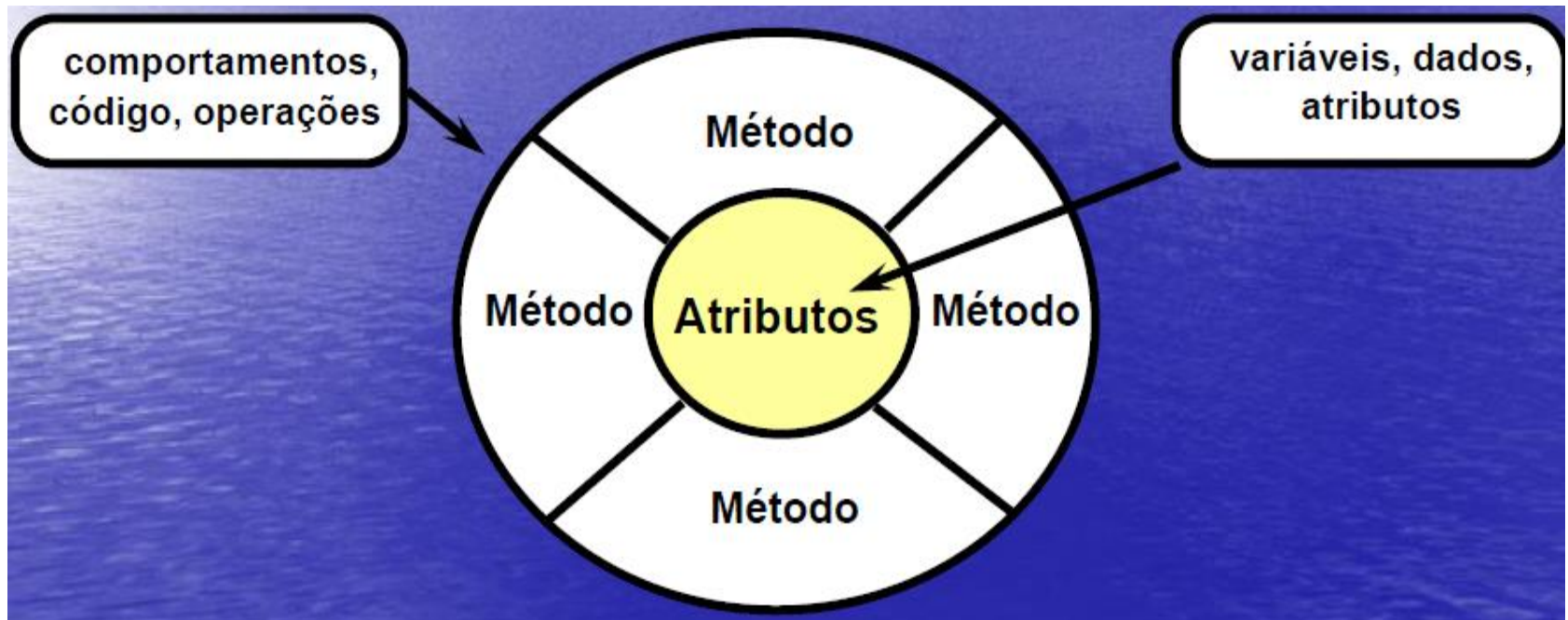
## Modelo da Rosquinha



# Classes e Objetos

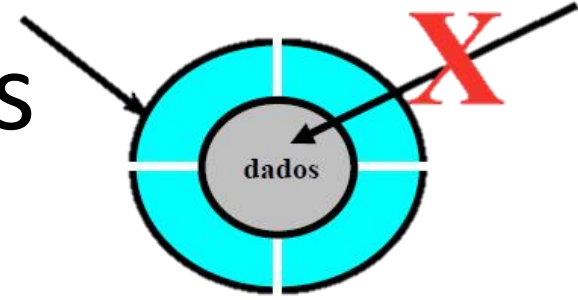
- Encapsulamento

## Modelo da Roubinha





# Classes e Objetos



- **Encapsulamento**

- Promove-se através das palavras reservadas:
  - **public**: Pode ser acessado de qualquer lugar e por qualquer entidade que visualiza a classe.
  - **protected**: Torna o membro acessível às classes do mesmo pacote e às subclasses.
  - **private**: Não são acessados por nenhuma outra classe.

# Classes e Objetos

- Encapsulamento em C++

```
class Obra
{
    private:
        std::string autor;
        Data publicacao;
    protected:
        int paginas;
        float peso;
    public:
        void vender(int);
        void repor(int);
};
```

# Classes e Objetos

- Encapsulamento em C++

```
class Obra
{
    private:
        std::string autor;
        Data publicacao;

    protected:
        int paginas;
        float peso;

    public:
        void vender(int);
        void repor(int);
};

int main()
{
    Obra tst;
    tst.vender(10);
    tst.repor(10);
    std::cout << tst.autor << std::endl;
    std::cout << tst.paginas << std::endl;
    std::cout << tst.peso << std::endl;
    return 0;
}
```

# Classes e Objetos

- Encapsulamento em C++


```
class Obra
{
    private:
        std::string autor;
        Data publicacao;

    protected:
        int paginas;
        float peso;

    public:
        void vender(int);
        void repor(int);
};
```

Encapsula.cpp:8:29: error: 'std::string Obra::autor' is private std::string autor;

```
int main()
{
    Obra tst;
    tst.vender(10);
    tst.repor(10);
    std::cout << tst.autor << std::endl;
    std::cout << tst.paginas << std::endl;
    std::cout << tst.peso << std::endl;
    return 0;
}
```



# Classes e Objetos

- Encapsulamento em C++

```
class Obra
{
    private:
        std::string autor;
        Data publicacao;

    protected:
        int paginas;
        float peso;

    public:
        void vender(int);
        void repor(int);
};
```

Encapsula.cpp:8:29: error: 'std::string Obra::autor' is private std::string autor;

```
int main()
{
    Obra tst;
    tst.vender(10);
    tst.repor(10);
    std::cout << tst.autor << std::endl;
    std::cout << tst.paginas << std::endl;
    std::cout << tst.peso << std::endl;
    return 0;
}
```

Encapsula.cpp:12:23: error: 'float Obra::peso' is protected float peso;  
Encapsula.cpp:11:21: error: 'int Obra::paginas' is protected int paginas;



# Métodos Modificadores

## Métodos de Acesso

- Perceba que ao tornar os atributos de uma classe privados, você impede que qualquer membro externo acesse ou modifique os valores dos atributos da classe.
- Apenas métodos da própria classe terão esse privilégio.
- Porém, é uma boa prática de programação que sejam disponibilizados métodos públicos que acessem e modifiquem cada atributo privado, são eles os métodos modificadores e os métodos de acesso.

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos Modificadores:
  - São também conhecidos como métodos **set**.
  - Não precisam obrigatoriamente ter esse nome, mas é uma boa prática de programação manter o padrão **set + o nome do Atributo**.
  - São **métodos públicos** de **retorno vazio** que recebem como argumento **um valor** e **o atribui** diretamente **ao atributo** correspondente.
  - É uma boa prática que eles sejam os **únicos** métodos a fazer essa atribuição, **inclusive entre os demais métodos** da própria classe.

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos Modificadores:

- Exemplo em C++

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        void setDia(int);
        void setMes(int);
        void setAno(int);
};
```

```
void Data::setDia(int d)
{
    this->dia = d;
}
```

```
void Data::setMes(int m)
{
    this->mes = m;
}
```

```
void Data::setAno(int a)
{
    this->ano = a;
}
```



# Métodos Modificadores

## Métodos de Acesso

- Os Métodos Modificadores:

- Exemplo em C++

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        void setDia(int);
        void setMes(int);
        void setAno(int);
};

int main()
{
    Data aniversario;
    aniversario.setDia(20);
    aniversario.setMes(1);
    aniversario.setAno(1994);
    return 0;
}
```

```
void Data::setDia(int d)
{
    this->dia = d;
}
```

```
void Data::setMes(int m)
{
    this->mes = m;
}
```

```
void Data::setAno(int a)
{
    this->ano = a;
}
```



com get

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos Modificadores:
  - Exemplo em JAVA

```
class Data
{
    private int dia, mes, ano;

    public void setDia(int d)
    {
        this.dia = d;
    }

    public void setMes(int m)
    {
        this.mes = m;
    }

    public void setAno(int a)
    {
        this.ano = a;
    }
}
```

```
public class Principal
{
    public static void main(String[] args)
    {
        Data aniversario;
        aniversario.setDia(20);
        aniversario.setMes(1);
        aniversario.setAno(1994);
    }
}
```



com get

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos de Acesso:
  - São também conhecidos como métodos **get**.
  - Não precisam obrigatoriamente ter esse nome, mas é uma boa prática de programação manter o padrão **get + o nome do Atributo**.
  - São **métodos públicos** de **retorno do mesmo tipo do atributo** que não recebem argumento e **retornam o valor do atributo** correspondente.
  - É uma boa prática que eles sejam os **únicos** métodos a fazer esse acesso, **inclusive entre os demais métodos** da própria classe.

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos de Acesso:

- Exemplo em C++

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        int getDia();
        int getMes();
        int getAno();
};
```

```
int Data::getDia()
{
    return this->dia;
}
```

```
int Data::getMes()
{
    return this->mes;
}
```

```
int Data::getAno()
{
    return this->ano;
}
```

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos de Acesso:

- Exemplo em C++

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        int getDia();
        int getMes();
        int getAno();
};

int main()
{
    Data aniversario;
    std::cout << "Dia: " << aniversario.getDia() << std::endl;
    std::cout << "Mes: " << aniversario.getMes() << std::endl;
    std::cout << "Ano: " << aniversario.getAno() << std::endl;
    return 0;
}
```

```
int Data::getDia()
{
    return this->dia;
}

int Data::getMes()
{
    return this->mes;
}

int Data::getAno()
{
    return this->ano;
}
```



+ set

# Métodos Modificadores

## Métodos de Acesso

- Os Métodos de Acesso:
  - Exemplo em JAVA

```
class Data
{
    private int dia, mes, ano;
```

```
    public int getDia()
    {
        return this.dia;
    }
```

```
    public int getMes()
    {
        return this.mes;
    }
```

```
    public int getAno()
    {
        return this.ano;
    }
```

```
}
```

```
public class Principal
{
```

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

```
        Data aniversario;
        System.out.println("Dia: " + aniversario.getDia());
        System.out.println("Mes: " + aniversario.getMes());
        System.out.println("Ano: " + aniversario.getAno());
```

```
    }
```

```
}
```



+ set

# Método Construtor

- **Métodos Construtores**
  - Trata-se de um método especial que é executado automaticamente quando um objeto da classe é instanciado.
  - Características dos métodos Construtores:
    - Os métodos construtores não possuem tipo de retorno.
    - Os métodos construtores não retornam nenhum tipo de valor.
    - Os métodos construtores possuem exatamente o mesmo nome da Classe.

# Método Construtor

- **Métodos Construtores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é instanciado.
- Exemplos em C++:

```
#include <iostream>
class Construtor
{
    private:
        int att1;
    public:
        Construtor();
};
Construtor::Construtor()
{
    std::cout << "Oi estou no construtor" << std::endl;
}

int main()
{
    Construtor obj;
    return 0;
}
```



# Método Construtor

- **Métodos Construtores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é instanciado.
- Exemplos em C++:

```
#include <iostream>
class Construtor
{
    private:
        int att1;
    public:
        Construtor();
};
Construtor::Construtor()
{
    std::cout << "Oi estou no construtor" << std::endl;
}

int main()
{
    Construtor obj, *obj2;
    std::cout << "vou instanciar o obj2" << std::endl;
    obj2 = new Construtor();
    std::cout << "vou desalocar o obj2" << std::endl;
    delete obj2;
    std::cout << "vou terminar" << std::endl;
    return 0;
}
```

# Método Construtor

- **Métodos Construtores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é instanciado.
- Exemplos em JAVA:

```
class Construtor
{
    private int att1;
    public Construtor()
    {
        System.out.println("Estou no construtor");
    }
}

public class Principal
{
    public static void main(String[] args)
    {
        Construtor obj;
        System.out.println("Vou instanciar");
        obj = new Construtor();
    }
}
```

# Método Construtor

- **Métodos Construtores**
  - Métodos Construtores são especialmente úteis para inicializar ou alocar atributos do objeto.

# Método Construtor

- **Métodos Construtores**

- Métodos Construtores são especialmente úteis para inicializar ou alocar atributos do objeto.

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        Data(int, int, int);
        void setDia(int);
        void setMes(int);
        void setAno(int);
        int getDia();
        int getMes();
        int getAno();
        void showData();
};
```

```
Data::Data(int d, int m, int a)
{
    this->setDia(d);
    this->setMes(m);
    this->setAno(a);
}
```

```
int main()
{
    Data * aniversario = new Data(15,2,1994);
    aniversario->showData();
    return 0;
}
```



# Método Construtor

- **Métodos Construtores**

- Métodos Construtores são especialmente úteis para inicializar ou alocar atributos do objeto.

```
class Fila
{
    private:
        int *itens, tamanho, inicio, fim;
    public:
        Fila(int);
        void setItens(int, int);
        void setTamanho(int);
        void setInicio(int);
        void setFim(int);
        int getItens(int);
        int getTamanho();
        int getInicio();
        int getFim();
        void enqueue();
        int dequeue();
        bool vazia();
        bool cheia();
};
```

```
Fila::Fila(int t)
{
    this->setTamanho(t);
    this->itens = new int[this->getTamanho()];
    this->setInicio(-1);
    this->setFim(-1);
}

int main()
{
    Fila * minhaFila = new Fila(100);
    ...
}
```

# Método Destrutor

- **Métodos Destrutores**
  - Trata-se de um método especial que é executado automaticamente quando um objeto da classe é desalocado.
  - Características dos métodos Destrutores:
    - Os métodos destrutores não possuem tipo de retorno.
    - Os métodos destrutores não retornam nenhum tipo de valor.
    - Os métodos destrutores possuem exatamente o mesmo nome da Classe.

# Método Destrutor

- **Métodos Destrutores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é desalocado.
- Exemplos em C++:

```
#include <iostream>
class Destrutor
{
    private:
        int att1;
    public:
        ~Destrutor();
};
Destrutor::~Destrutor()
{
    std::cout << "Oi estou no destrutor" << std::endl;
}

int main()
{
    Destrutor obj;
    return 0;
}
```

# Método Destrutor

- **Métodos Destrutores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é desalocado.

- Exemplos em C++:

```
#include <iostream>
class Destrutor
{
    private:
        int att1;
    public:
        ~Destrutor();
};
Destrutor::~Destrutor()
{
    std::cout << "Oi estou no destrutor" << std::endl;
}
```

```
int main()
{
    Destrutor *obj;
    obj = new Destrutor();
    delete obj;
    obj = new Destrutor();
    delete obj;
    return 0;
}
```



# Método Destrutor

- **Métodos Destrutores**
  - Trata-se de um método especial que é executado automaticamente quando um objeto da classe é desalocado.
  - Exemplo em JAVA:

# Método Destrutor

- **Métodos Destrutores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é desalocado.
- Exemplo em JAVA:
- Não há método destrutor na Linguagem JAVA

C++



# Método Destrutor

- **Métodos Destrutores**

- Trata-se de um método especial que é executado automaticamente quando um objeto da classe é desalocado.
- Exemplo em JAVA:
- Não há método destrutor na Linguagem JAVA
  - O que pode ser feito é implementar o método void finalize() da classe Object, uma vez que esse método é invocado quando o Garbage Colector desaloca o objeto.
  - Porém, não se tem controle quando o Garbage Colector o fará.

# Método Destrutor

- **Métodos Destrutores**
  - Métodos Destrutores são especialmente úteis para desalocar recursos que foram alocados pelo objeto.

# Método Destrutor

- **Métodos Destrutores**

- Métodos Destrutores são especialmente úteis para desalocar recursos que foram alocados pelo objeto.

```
class Fila
{
    private:
        int *itens, tamanho, inicio, fim;
    public:
        Fila(int);
        ~Fila();
        void setItens(int, int);
        void setTamanho(int);
        void setInicio(int);
        void setFim(int);
        int getItens(int);
        int getTamanho();
        int getInicio();
        int getFim();
        void enqueue();
        int dequeue();
        bool vazia();
        bool cheia();
};
```

```
Fila::~~Fila()
{
    delete this->itens;
}
```

```
int main()
{
    Fila * minhaFila = new Fila(100);
    ...
    delete minhaFila;
    ...
}
```