TGPSI

PSI - Módulo 10: Programação orientada a objetos

Objetivos / conteúdos

- Definir relações entre objectos.
- Conceito de Herança e Polimorfismo;
- Métodos Virtuais e Virtuais Puros;
- Representar esquematicamente diagramas de classes.

- Herança e Polimorfismo
- Mensagens entre Objectos
- Redefinição de Métodos. Redefinição de Comportamento
- Métodos Virtuais e não Virtuais
- Diagramas de Classes

Ferramentas

- Programação em JAVA / Android
- Eclipse

- Avaliação: (Ver os critérios de avaliação na página)
  - 2 testes
  - Projetos do eclipse feitos nas aulas (PortFolio)

Classes, Atributos e Métodos

- Uma **classe** é um objeto especial que serve de molde ou padrão para a criação de objectos similares designados por ***instâncias*** da classe.
- Estes objectos possuem a mesma estrutura interna(atributos) e a mesma interface(respondem às mesmas mensagens), pelo que possuem igual comportamento.

Classes, Atributos e Métodos

- Uma classe é composta por:
  - **Identificador** que indica o nome da classe.
  - **Atributos**(ou variáveis de instância), valores que cada objeto contém e cujos domínios podem ser de:
    - Tipos primitivos: inteiros, reais, caracteres,...
    - Referências a outros objectos (ou tipos classe): identificando relações entre objetos.
  - **Métodos** que são as operações que podem alterar os valores do objeto.
- Os atributos e os métodos são designados por **membros da classe**.

Classes, Atributos e Métodos

- Um **método** é uma sequência de ações, executada por um objeto, que pode alterar ou dar a conhecer os seu **estado** (valor dos atributos).
- Enquanto que o valor dos atributos reside no objeto, o método reside na classe.
- **Assinatura** dum método:
  - identificador do método;
  - identificador e tipo dos parâmetros;
  - valor de retorno.

Classes, Atributos e Métodos

- Os métodos são catalogados em:
- **Construtor**: executado na criação do objeto.
  - Têm usualmente o mesmo identificador da classe.
  - Não podem ser evocados.
  - Normalmente usados para inicializar os atributos.
- **Destrutor**: executado na destruição do objeto.
- **Modificador**: altera o valor dos atributos.
- **Seletor**: dá a conhecer o valor dos atributos, sem os alterar.

Classes, Atributos e Métodos

```
class Nome_Classe {
    //variáveis de instância
    ........
   //construtores
    .......
    //métodos de instância
    ........
}
```

Convenções:

- O nome de uma classe começa por Maiúsculas
- O nome de uma variável de instância começa por minúsculas e restantes palavras por Maiúscula - **formaGeométrica**
- O nome de um método de instância começa por Maiúsculas

Classes, Atributos e Métodos

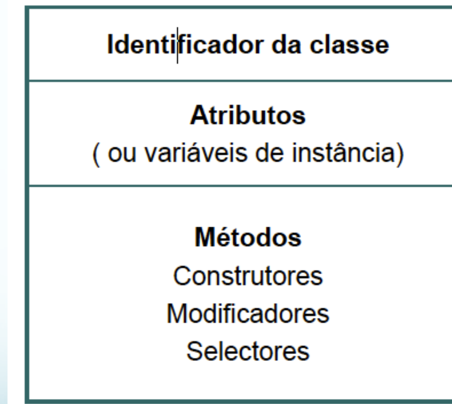
- Exemplos de métodos
- **Método Soma**
  - Recebe dois inteiro do tipo inteiro
  - Devolve um inteiro

```
int Soma(int a, int b){
  return(a + b);
}
```

- **Método Mensagem**
  - Sem argumentos
  - Não retorna void

```
void Mensagem(){
  System.out.println("Ola");
}
```


Classes, Atributos e Métodos




```
public class pessoa {

    private String nome;
    private String apelido;
    private int anoNasc;

    //construtor
    public pessoa(String n, String ap, int an)
    {
        nome = n;
        apelido = ap;
        anoNasc = an;
    }

    //Modificadores
    public void SetNome(String novoNome)
    {
        nome = novoNome;
    }

    public void SetApelido(String novoApelido)
    {
        apelido = novoApelido;
    }

    public void SetAnoNasc(int novoAnoNasc)
    {
        anoNasc = novoAnoNasc;
    }

    //Selectores
    public String GetNome()
    {
        return nome;
    }
}
```


Criação de objectos – usando construtores

- **Declaração da classe ParXY**

```
public class ParXY {
  int x,y;
  ParXY(int a,int b){
        x=a; y=b;
  }
  int soma(){
    return (x+y)
  }
}
```

- **Criação de instâncias da classe ParXY**

```
ParXY par4 = new ParXY(3,10);
ParXY p = new ParXY(2, 4);
p.soma();
```

Calcula a soma do objeto p

Manipulação de Objectos

Para aceder aos elementos de um objecto temos as seguintes situações:

- **O acesso ao objecto é feito através de uma variável que contenha a referência do objecto**

```
ParXY par4 = new ParXY(3,10);
```

- **Invocar métodos do objecto:** `referencia_obj.nome_do_método(parâmetros)`

```
par4.valorX();
```

- **Aceder aos atributos do objecto:** `referencia_obj.nome_do_atributo`

```
par4.x;
```

Proibido

Regra violada frequentemente na programação orientada por objectos

Violação do encapsulamento

Manipulação de objectos - Encapsulamento

Regra para uma correcta programação em POO:

"Qualquer que seja a linguagem de POO usada, a única forma de se poder garantir, mesmo usando os mecanismos típicos da POO, que estamos a programar entidades independentes do contexto e, assim, reutilizáveis, é garantir que nenhuma cápsula faz acesso directo às variáveis de instância de outra cápsula, mas que apenas invoca, por mensagens, os métodos de acesso a tais valores, que para tal devem ser programados em cada uma das cápsulas." [Mário Martins]

Exemplos de violação da regra, possíveis em C# e JAVA:

```
Contador ct1 = new Contador();
int c = ct1.conta;        /* leitura indevida => viola
ct1.conta = 22; /* alteração indevida => viola
```

- Esta forma de aceder externamente a uma variável de instância (*conta* é uma variável de instância de *ct1*) não é normalmente verificada em tempo de compilação

Referência *this*

- Auto-referência ao objecto que executa a mensagem
  - esta referência é utilizada por um objecto, no seu interior, para enviar uma mensagem a si próprio
- Exemplo

```
public class test{
  private int x;
  public test(int x){
    this.x = x
  }
}
```

This.x diz respeito à variável de instância
x diz respeito ao argumento

Modificadores de acesso(visibilidade)

- O Java fornece mecanismos de controlo de acessibilidade(visibilidade) tanto para classes como para os respectivos membros.
- public
- private
- protected

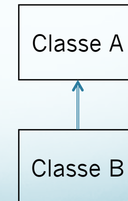
Herança

- É um mecanismo das linguagens de programação orientado a objetos **que permite que uma nova classe seja descrita a partir de uma classe já existente.**
- Permite que uma classe herde todo o comportamento e os atributos de outra classe.
- **Classe mãe**: superclasse, classe base;
- **Classe filha/filho**: subclasse, classe derivada;
- Classe filha (mais específica) herda atributos e métodos da classe mãe (mais geral);
- Classe filha possui atributos e métodos próprios.

Herança

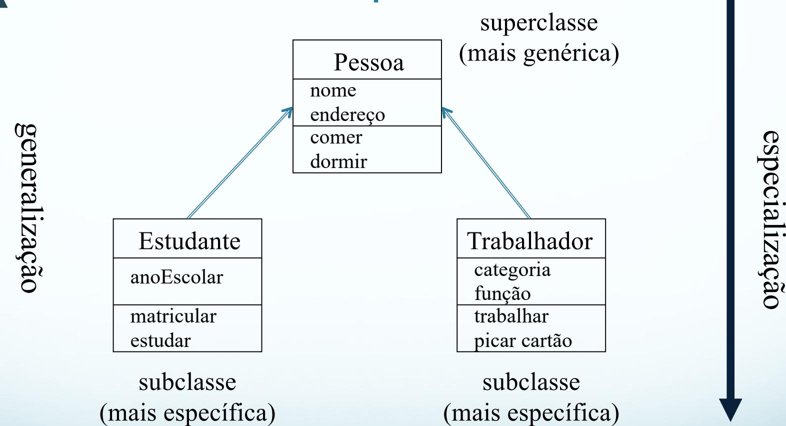
- Entre uma classe e a sua **superclasse**, é estabelecida uma relação de **especialização** que é automaticamente implementada através de um mecanismo de herança.
- Este mecanismo automático de herança estabelece as seguintes propriedades entre uma **subclasse B** e a sua **superclasse A**:

1. **B** herda de **A** todas as variáveis e métodos de instância (os atributos da classe B, declarados como `private` em A , só podem ter acesso pelos métodos `public` de A, e não diretamente)
2. **B** pode definir novas variáveis e novos métodos próprios.
3. **B** pode redefinir variáveis e métodos **herdados**.

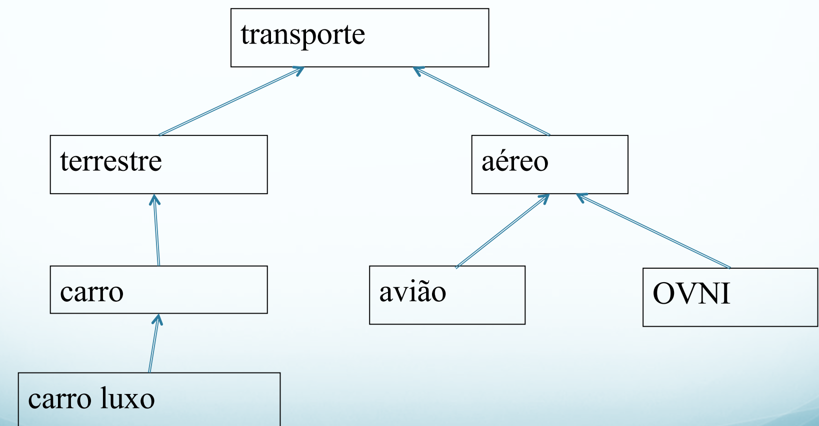


a herança não se aplica a variáveis e métodos de classe

Exemplo 1



Herança



Herança

- Implementação da herança em java
  - através da palavra reservada extends
  - o super serve para chamar o construtor da classe mãe (ou outros métodos da classe mãe).

  - Exercício prático
  - Classe pessoa – identificado por nome e morada
    - Para esta classe implementar o construtor, os seletores e os modificadores
    - Classe estudante que herda da classe pessoa. Contem o ano de escolaridade
  - Implementar uma classe para testar as funcionalidades das duas classes.

Modificadores de acesso

Private	**Visível somente dentro da classe onde está definida**
Public	Visível por todas as classes que a utilizem
Protected	Visível somente na classe que herda

Modificadores de acesso

```
class Circulo
{
    private float raio;
    Circulo(float r)
     {
       raio = r;
     }
    void setRaio( float novoraio )
     {
       raio = novoraio;
     }
}
```

```
class Pneu extends Circulo
{
    Pneu p = new Pneu();
    p.raio = 10.0; //Erro de compilação. O
Atributo raio é privado da classe Circulo
    p.setRaio(10.0); //Correto, pois a classe Pneu
está utilizando os métodos definidos na classe
Circulo para fazer acesso ao atributo privado raio
}
```


Manipulação de objectos - Encapsulamento

Regra para uma correcta programação em POO:

"Qualquer que seja a linguagem de POO usada, a única forma de se poder garantir, mesmo usando os mecanismos típicos da POO, que estamos a programar entidades independentes do contexto e, assim, reutilizáveis, é garantir que nenhuma cápsula faz acesso directo às variáveis de instância de outra cápsula, mas que apenas invoca, por mensagens, os métodos de acesso a tais valores, que para tal devem ser programados em cada uma das cápsulas." [Mário Martins]

Exemplos de violação da regra, possíveis em C# e JAVA:

```
Contador ct1 = new Contador();
int c = ct1.conta;  /* leitura indevida => viola
ct1.conta = 22;     /* alteração indevida => viola
```

- Esta forma de aceder externamente a uma variável de instância (*conta* é uma variável de instância de *ct1*) não é normalmente verificada em tempo de compilação

Regras de acessibilidade - encapsulamento

Temos 5 tipos possíveis de acesso: modificadores utilizados nas declarações

- ***private*** apenas visíveis dentro da mesma classe
- ***protected*** apenas visíveis dentro da mesma classe ou derivadas
- ***public*** visíveis de todo o lado

Criação de objectos – usando construtores

- **podemos criar instâncias de classes de várias formas** (diferentes iniciações)

Declaração da classe ParXY

```
public class ParXY {
    //construtores
    ParXY(){
        x=0;y=0;}
    ParXY(int val){
        x=val; y=val;}
    ParXY(int a,int b){
        x=a; y=b;}
    //variáveis de instância
    int x,y;
}
```

Criação de instâncias da classe ParXY

```
ParXY par1 = new ParXY();
ParXY par2 = new ParXY(10);
ParXY par3 = new ParXY(5);
ParXY par4 = new ParXY(3,10);
```

Sobrecarga de nomes de métodos (*overloadding*)

***Method overloadding*:**

numa mesma classe podem ser declarados métodos diferentes mas com o mesmo nome

- **É possível*:***
  - numa classe declarar métodos com o mesmo nome mas parâmetros diferentes
  - métodos com o mesmo nome podem ter tipos de resultados distintos desde que as suas assinaturas sejam distintas

Assinatura = *<identificador>* (<parâmetros>)

- o tipo de resultado de um método não faz parte da sua assinatura
- a ordem dos parâmetros faz parte da sua assinatura

Polimorfismo

- O **polimorfismo** surge quando uma classe derivada **reescreve** um método da classe base.
- Está ligado à herança.
- Exemplo:
  - Classe 1 possui 2 métodos: métodoA() e métodoB().
  - Classe 2 herda a classe 1.
  - Classe 2 reescreve todo o métodoA() que pertence a classe 1.
- Métodos declarados como ***Final***
  - Significa que não podem ser reescritos.

Polimorfismo - Exemplo

```
Public class Quadrilatero{
private int lado1, lado2, lado3, lado4
Public Quadrilatero(int l1, int l2, int l3, int l4)
{ lado1=l1; lado2=l2;lado3=l3;lado4=l4
}
public int area(){
  return 0;
}
//slectores e modificadores
}
```

```
public class quadrado extends Quadrilatero{

public quadrado(int lado){
  super(lado, lado,lado,lado)
}
public int area(){
  int lado = super.GetLado1();
  int c_area = lado * lado
  return c_area;
}
}
```


Polimorfismo - Exemplo

```
public class retangulo extends Quadrilatero{

public retangulo(int base, altura){
  super(base, altura,base,altura)
}
public int area(){
  int base = super.GetLado1();
  int altura = super.GetLado2();
  int c_area = base * altura
  return c_area;
}
}
```

```
public class Teste {
  public static void main(String[] args) {
    Quadrado q = new Quadrado (2);
    int a = q. area();
    System.out.println("Area do quadrado de lado 2 =" + a);
    //Criar um objeto retangulo
    //apresentar a area do objeto criado
  }

}
```


Polimorfismo - Final

```
Public class A{
  private int valor1, valor2;
  public A(int v1, int v2)
  { valor1= v1; valor2=v2; }
  //Seletores e modificadores
  final void ImprimeOi()
  {System.out.println("Oii");}
}
```

```
Public class B extends A{
  public B(int v1, v2){
    super(v1, v2);
  }
  public void ImprimeOi{
   System.out.println("Oi do B")
  }
}
```

Erro Porque?

Diagrama de classes

Operacoes
+ valor1: int + valor2: int
+ Operacoes() + Operacoes(x: int, y: int) + getValor1: int + getValor2: int + setValor1(v1: int): void + setValor2(v2: int): void + soma(): void + soma(v1: int): void + soma(v1: double): void