Programação Orientada por Objetos

Polimorfismo

Prof. José Cordeiro,

Prof. Cédric Grueau,

Prof. Laercio Júnior

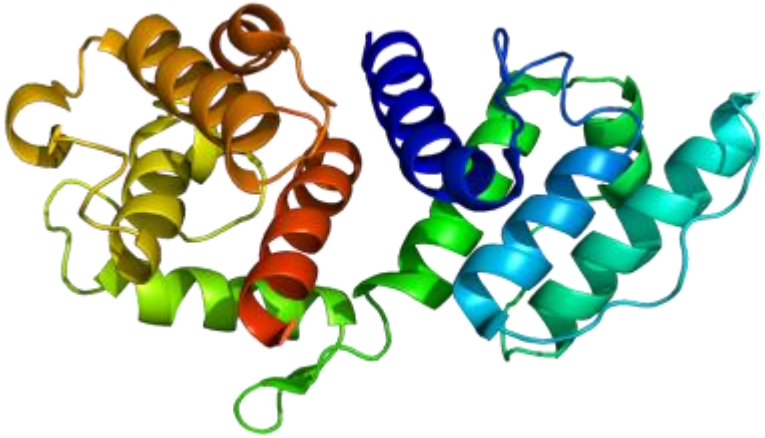
Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

2019/2020

Módulo Polimorfismo

- Sessão 1: Principio da Substituição
- Sessão 2: Tipos Estáticos e Tipos Dinâmicos
- Sessão 3: Polimorfismo
- Sessão 4: Exemplo Formas Geométricas



Módulo 2 – Polimorfismo

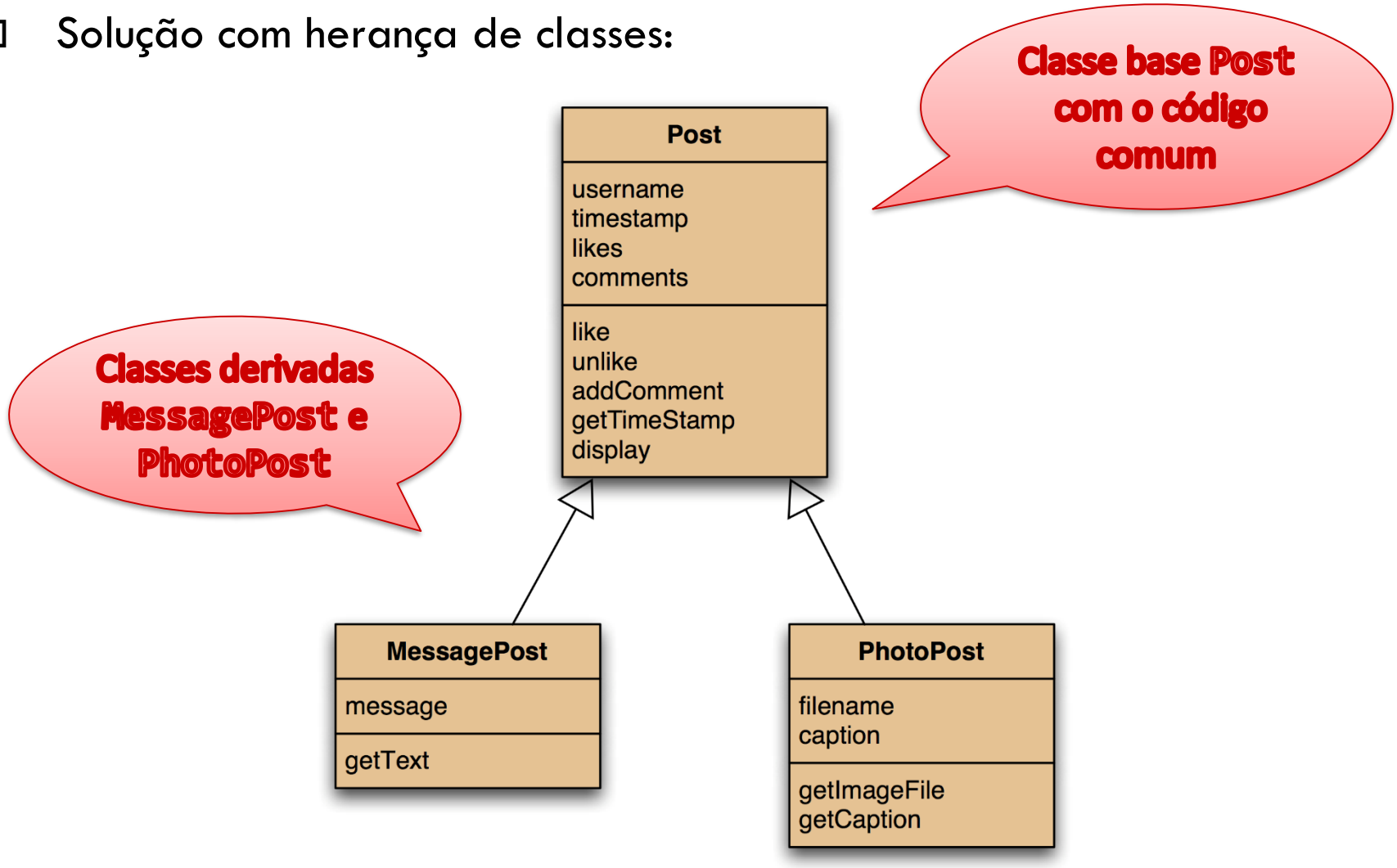
SESSÃO 1 – PRINCIPIO DA SUBSTITUIÇÃO

Exemplo – Rede Social

- Requisitos da rede social:
  - Um pequeno protótipo com a base para o armazenamento e apresentação de mensagens.
  - Faz o armazenamento de mensagens de texto e mensagens de imagem.
    - As mensagens de texto podem ter várias linhas;
    - as mensagens de imagem têm uma imagem e uma descrição.
  - Todas as mensagens devem incluir o seu autor, a altura em que foi enviada, o número de "gostos" e a lista de comentários.

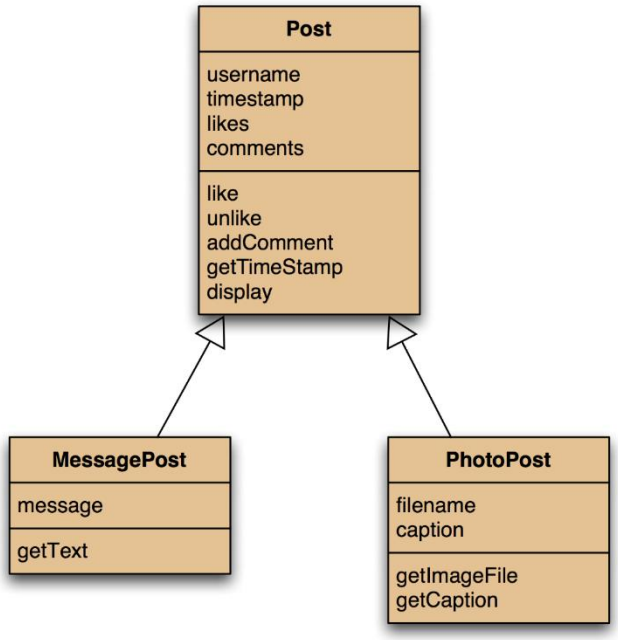
Exemplo – Rede Social

- Solução com herança de classes:

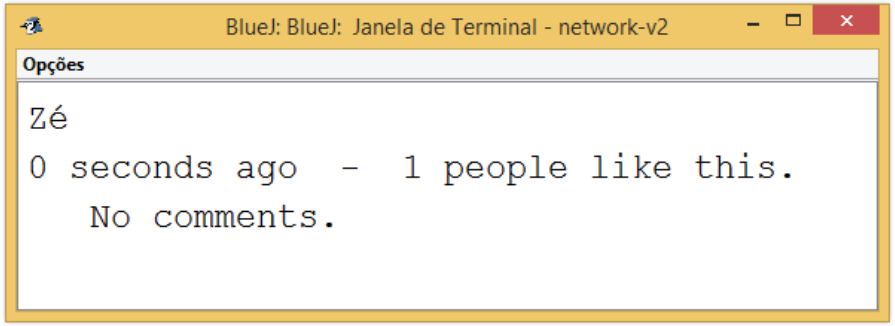


Exemplo – Rede Social

- Exemplo de utilização



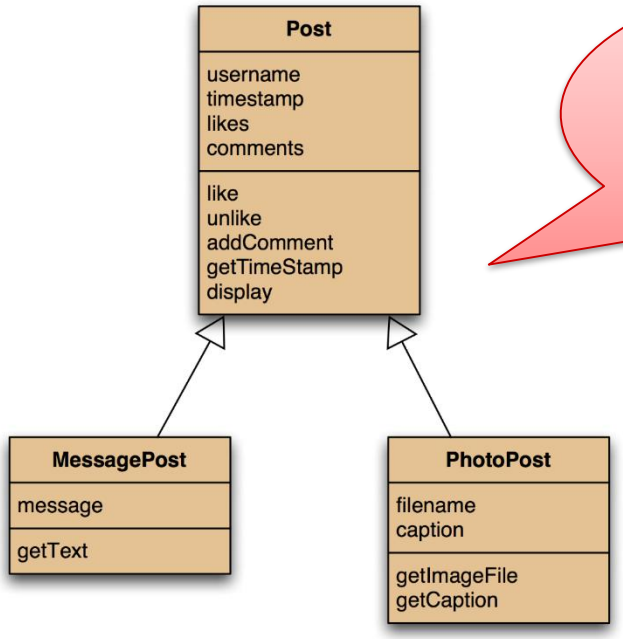

```
MessagePost post = new MessagePost("Zé", "Olá Herança");

post.like();
post.display();
```



O texto da mensagem não é mostrado

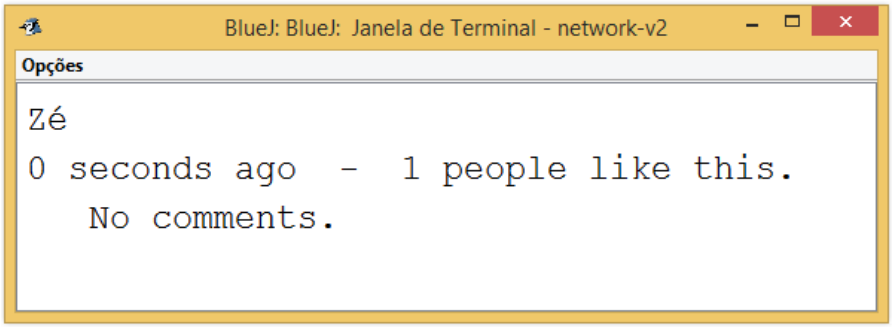
Exemplo – Rede Social

- Exemplo de utilização



O método display que devia mostrar a mensagem está na classe Post

```
MessagePost post = new MessagePost("Zé", "Olá Herança");

post.like();
post.display();
```



Exemplo – Rede Social

- Exemplo de utilização

```
public class Post {
    private String username;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    // restante código omitido
    public void display() {
        System.out.println(username);
        System.out.print(timeString(timestamp));
        if(likes > 0) {
            System.out.println("  -  " + likes + " people like this.");
        }
        else {
            System.out.println();
        }
        if(comments.isEmpty()) {
            System.out.println("   No comments.");
        }
        else {
            System.out.println("   " + comments.size() +
                               " comment(s). Click here to view.");
        }
    }
}
```

A informação mostrada está incompleta para as classes MessagePost e PhotoPost

Solução:
Redefinir o método
nas classes derivadas

Exemplo – Rede Social

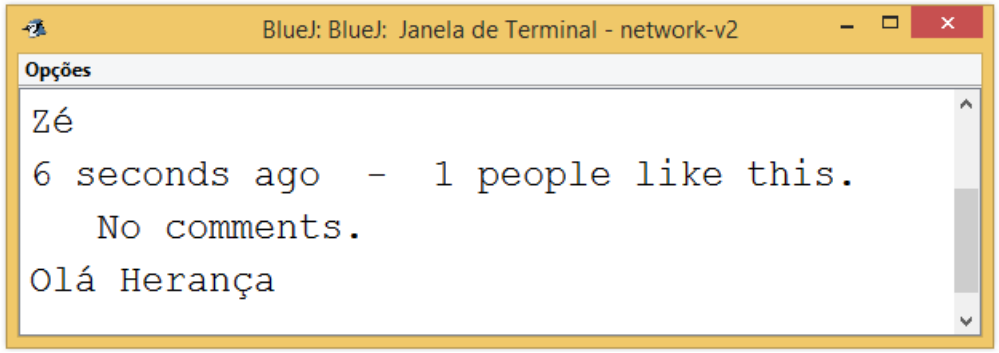
- Exemplo de utilização

```
public class MessagePost extends Post {
    private String message;
    // restantes métodos omitidos

    @Override
    public void display() {
        super.display();
        System.out.println(message);
    }
}

public class PhotoPost extends Post {
    private String filename;
    private String caption;
    // restantes métodos omitidos

    @Override
    public void display() {
        super.display();
        System.out.println("  [" + filename + "]");
        System.out.println("  " + caption);
    }
}
```

Exemplo – Rede Social

- Exemplo de utilização

```
MessagePost post = new MessagePost("Zé", "Olá Herança");

post.like();
post.display();
```



Acrescenta o texto da mensagem no caso de objetos MessagePost

Principio da Substituição

- **Subclasses** e **Subtipos**
  - Uma classe define um **tipo**
  - Uma subclasse define um **subtipo**
  - Sempre que é necessário um objeto de uma classe pode-se usar em vez disso um objeto duma subclasse:
    - Chama-se **principio da substituição**
      - Exemplo

```
Post post = new MessagePost("João", "Olá Mundo");
```

Guarda um objeto da classe **Post**

Atribui-se um objeto da subclasse **MessagePost**

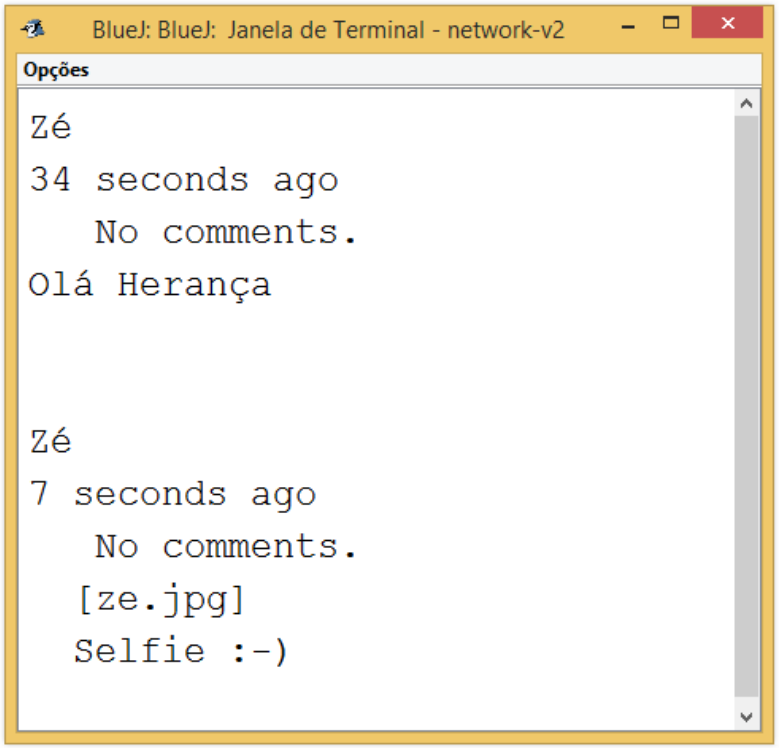
Principio da Substituição

- Exemplo de utilização

```
Post post = new MessagePost("Zé", "Olá Herança");
post.display();
post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.display();
```

display de **MessagePost**

display de **PhotoPost**

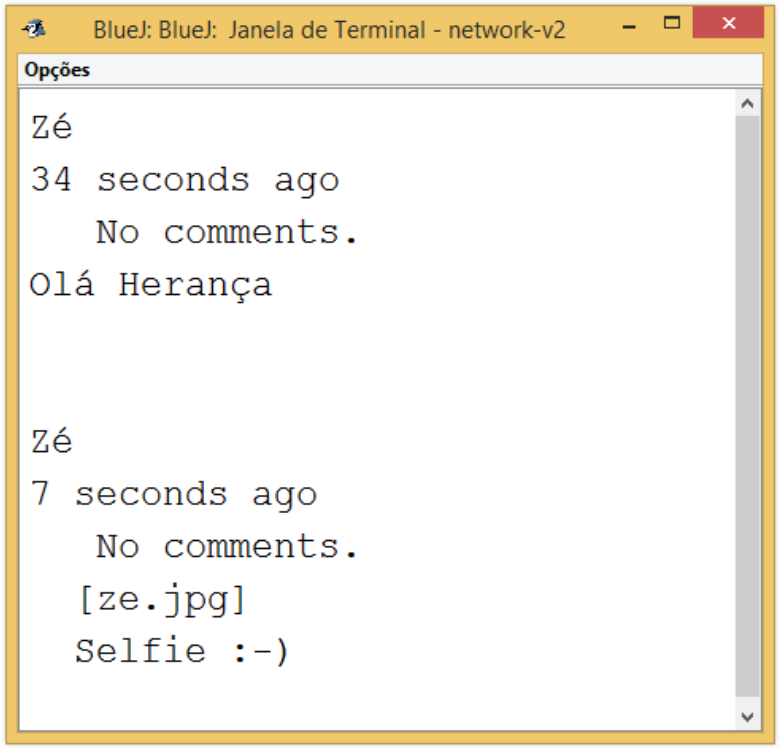


Principio da Substituição

- Exemplo de utilização

```
Post post = new MessagePost("Zé", "Olá Herança");
post.display();
post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.display();
```

A variável post é do tipo Post mas pode guardar objetos de tipos derivados como é o caso com objetos dos tipos MessagePost e PhotoPost

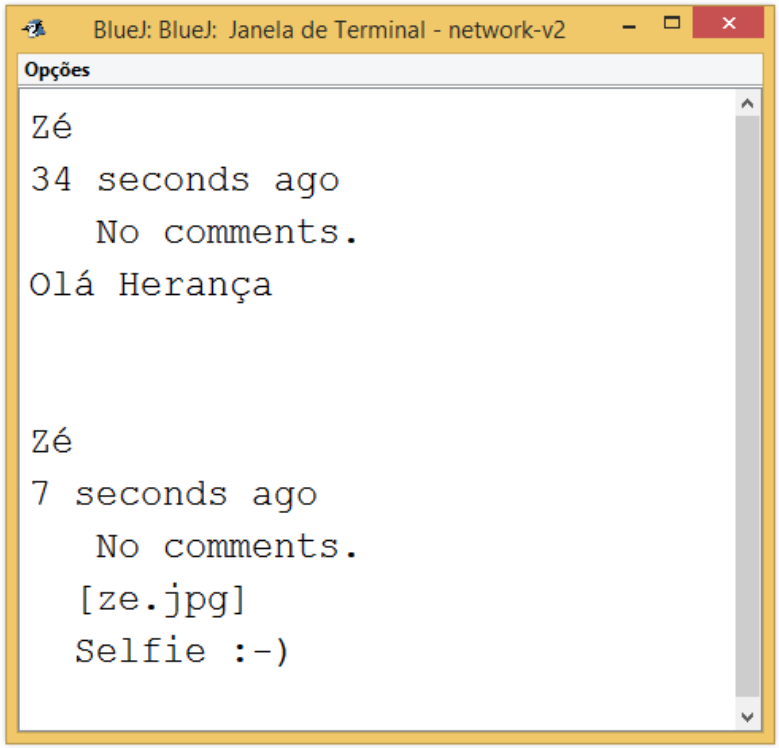


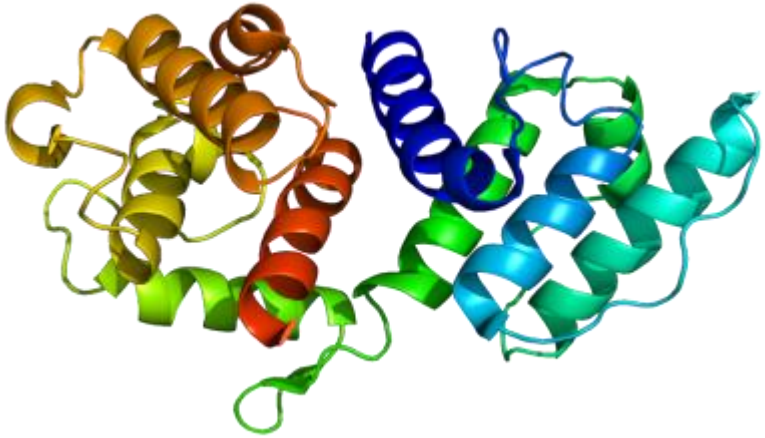
Principio da Substituição

- Exemplo de utilização

```
Post post = new MessagePost("Zé", "Olá Herança");
post.display();
post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.display();
```

- O método **display** que é **executado** é o que está no objeto guardado na variável
  - **display** do objeto **MessagePost** mostra o texto da mensagem
  - **display** do objeto **PhotoPost** mostra o ficheiro e a legenda da imagem





Módulo 2 – Polimorfismo

SESSÃO 2 – TIPOS ESTÁTICOS E DINÂMICOS

Exemplo – Rede Social

- **Novo** requisito da rede social adicional:
  - Os *posts* de imagem e de texto devem devolver o texto associado à mensagem.
    - No caso das mensagens de texto corresponde ao texto da mensagem.
    - No caso das mensagens de imagem corresponde à legenda da imagem.
    - Cada tipo de mensagem deve definir o texto que retorna.

Exemplo – Rede Social

- Classe **MessagePost**

```
public class MessagePost extends Post {
    private String message;

    public MessagePost(String author, String text) {
        super(author);
        message = text;
    }

    // métodos omitidos

    public String getText()  {
        return message;
    }

}
```

Exemplo – Rede Social

- Classe **PhotoPost**

```
public class PhotoPost extends Post {
    private String filename;
    private String caption;

    public PhotoPost(String author, String filename, String caption) {
        super(author);
        this.filename = filename;
        this.caption = caption;
    }
    // métodos omitidos

    public String getText()  {
        return caption;
    }

}
```

Exemplo – Rede Social

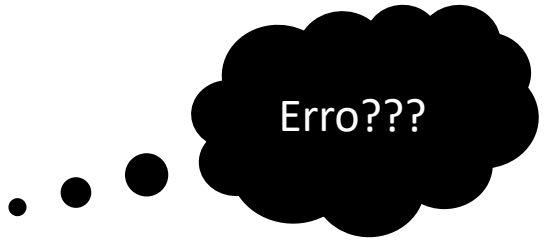
- Superclasse – **Post**

```
public class Post {
    private String username;
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    // construtores e métodos omitidos
}
```

Principio da Substituição

- Exemplo de utilização – método **getText**

```
Post post = new MessagePost("Zé", "Olá Herança");
post.getText();
Error: cannot find symbol -   method getText()
```




```
Post post = new MessagePost("Zé", "Olá Herança");
post.getText();
Error: cannot find symbol -   method getText()

post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.getText();
Error: cannot find symbol -   method getText()
```

Não funciona porque o método getText não existe na classe base Post

Tipos Estáticos e Tipos dinâmicos

```
MessagePost p1 = new MessagePost("José", "Olá pessoal");
```

Qual é o tipo de **p1**?

```
Post p2 = new MessagePost ("José", "Olá pessoal");
```

Qual é o tipo de **p2**?

Tipos Estáticos e Tipos dinâmicos

- O tipo declarado de uma variável é o seu **tipo estático** (*static type*)
- O tipo do objeto referido pela variável é o seu **tipo dinâmico** (*dynamic type*)
- O compilador verifica sempre se existem erros nos tipos estáticos.
  - Não é possível referir objetos que não sejam da classe do tipo estático ou de uma das classes derivadas do mesmo.
  - Não é possível chamar métodos para uma variável que não existam no seu tipo estático. Ou seja, que não existam na classe declarada.

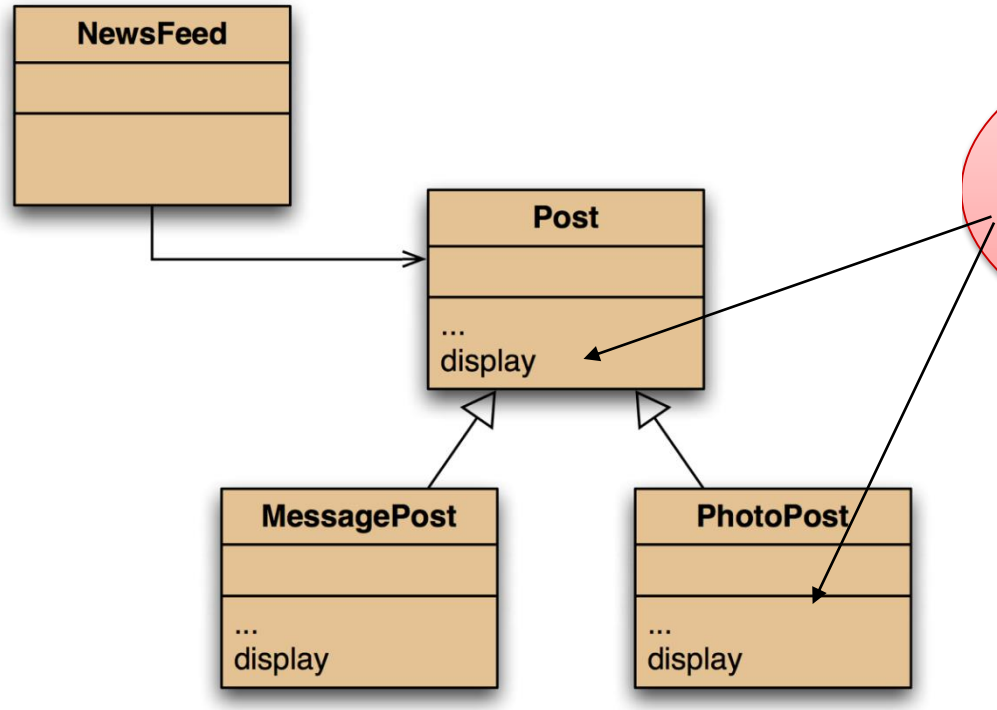
```
Post p2 = new MessagePost ("José", "Olá pessoal");
```

Tipos Estáticos e Tipos dinâmicos

```
Post post = new MessagePost("Zé", "Olá Herança");
post.getText();
Error: cannot find symbol -   method getText()

post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.getText();
Error: cannot find symbol -   method getText()
```

- A variável **`post`** é do tipo (estático) **`Post`**
- O método **`getText`** não existe na classe **`Post`** (o seu tipo estático)
- O compisider deteta que o método **`getText`** não existe para a classe **`Post`** e dá erro de compilação
  - No caso mostrado o BlueJ acusa o erro quando está a *interpretar* o código

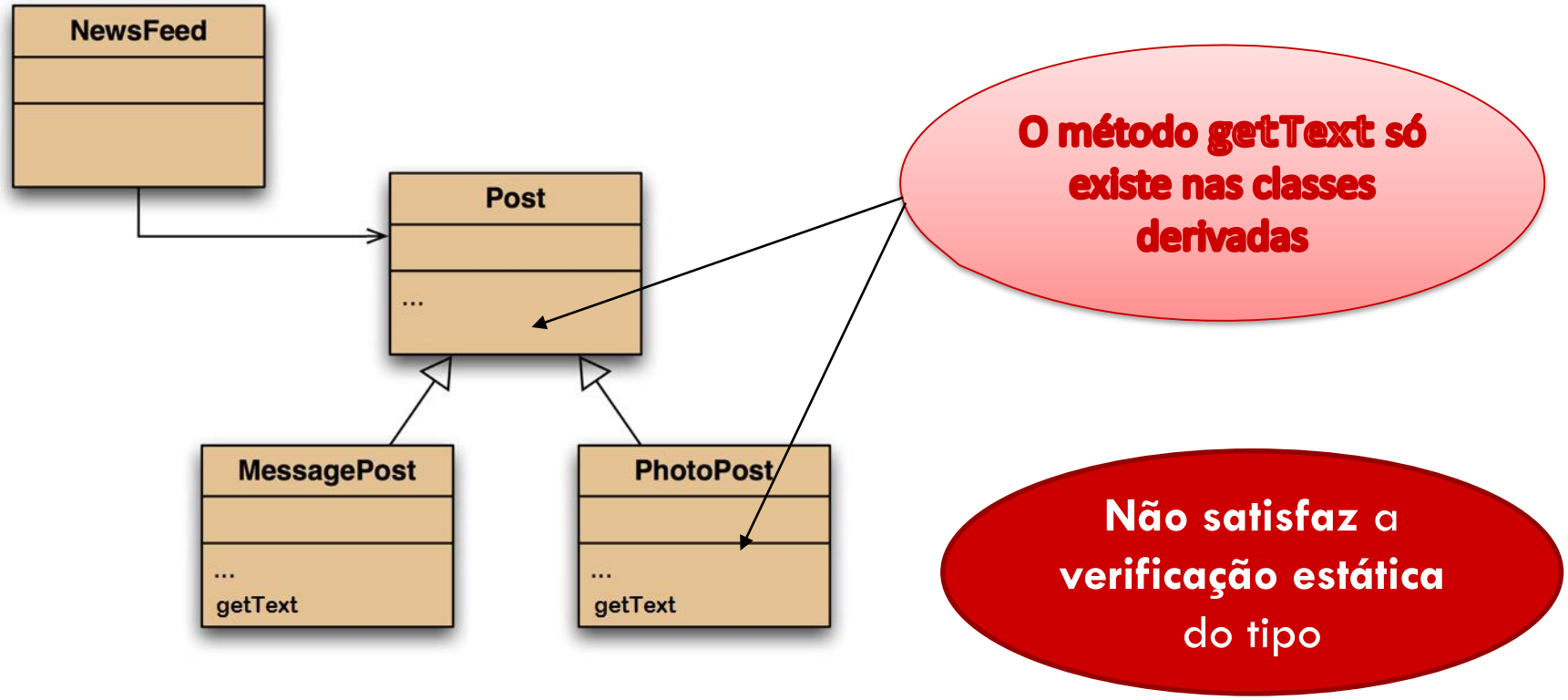
Tipos Estáticos e Tipos dinâmicos



No caso do método `display` não há problema porque existe na classe base e nas classes derivadas

Satisfaz a **verificação estática** (e dinâmica) do tipo

Tipos Estáticos e Tipos dinâmicos

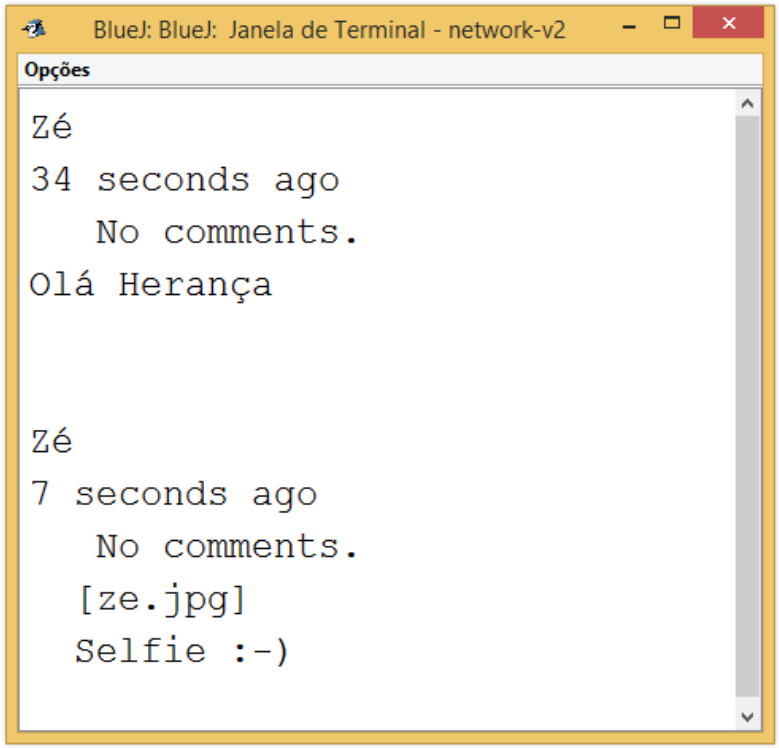


Tipos Estáticos e Tipos dinâmicos

- Em tempo de **execução** (runTime)

```
Post post = new MessagePost("Zé", "Olá Herança");
post.display();
post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.display();
```

- O método **display** que é **executado** é o que está no objeto guardado na variável (o seu **tipo dinâmico**)
  - **display** do objeto **MessagePost** mostra o texto da mensagem
  - **display** do objeto **PhotoPost** mostra o ficheiro e a legenda da imagem

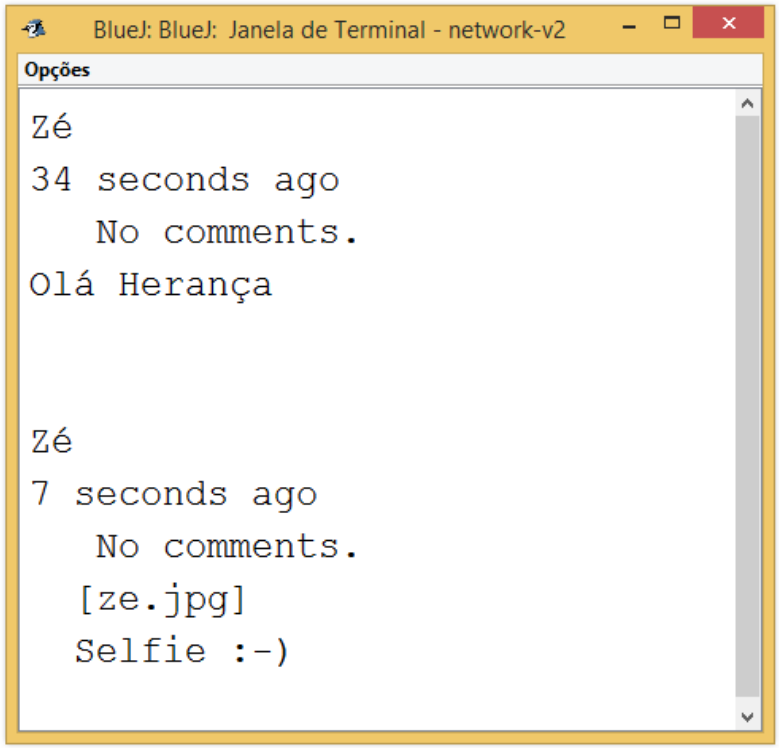


Tipos Estáticos e Tipos dinâmicos

- Em tempo de **execução** (runTime)

```
Post post = new MessagePost("Zé", "Olá Herança");
post.display();
post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.display();
```

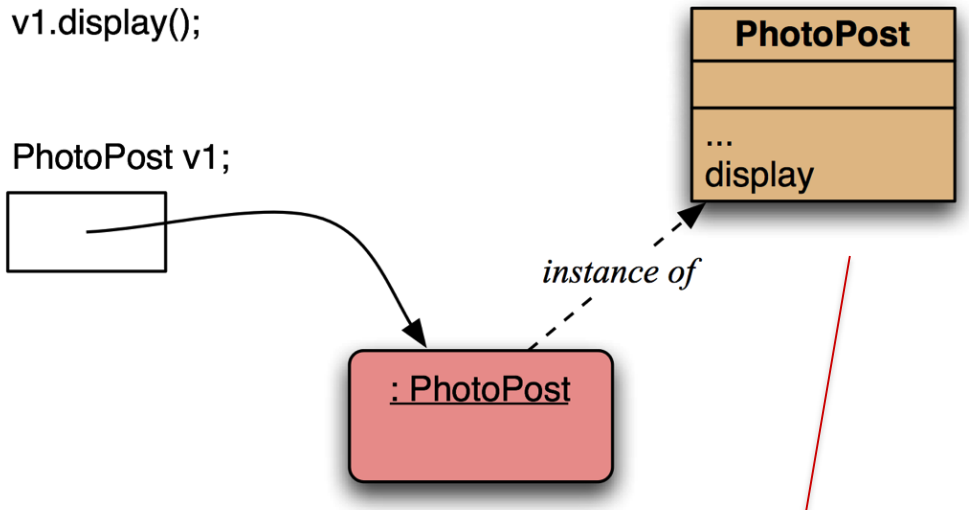
- O método **display** que é **executado** é o que está no objeto guardado na variável (o seu tipo dinâmico)
- Durante a execução é procurado o método a executar (*Method lookup*) de acordo com o objeto que está guardado na variável
  - … e não com o tipo da variável



Tipos Estáticos e Tipos dinâmicos

- Em tempo de **execução** (runTime) – procura do método a executar:

Tipo estático e tipo dinâmico são iguais: **PhotoPost**

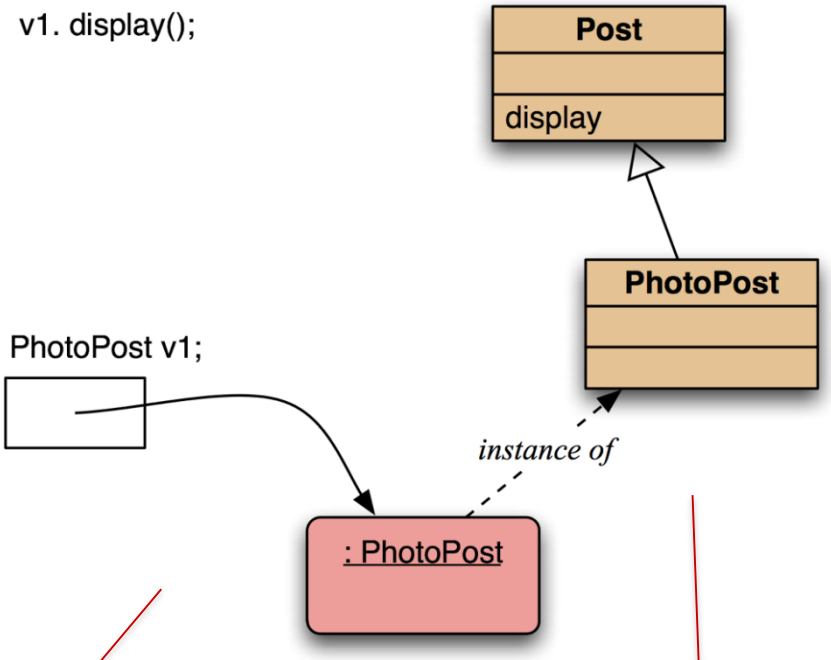


O método **`display`** a executar apenas pode ser o que está em **PhotoPost**

Neste exemplo **não existe herança**

Tipos Estáticos e Tipos dinâmicos

- Em tempo de **execução** (runTime) – procura do método a executar:



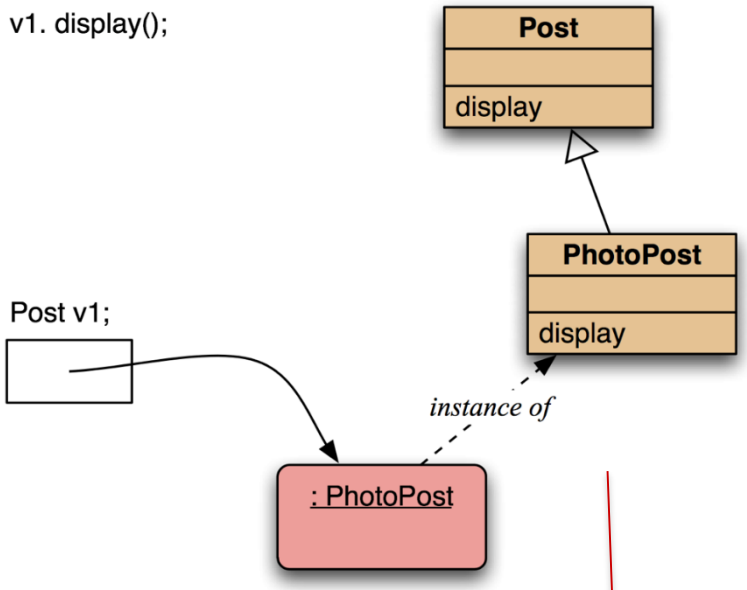
Tipo estático e tipo dinâmico são iguais: **PhotoPost**

O método **`display`** é procurado na classe do objeto **PhotoPost** mas como não é encontrado é procurado na classe pai **Post.** Ao ser encontrado é executado esse método

Neste exemplo **existe herança** mas **não existe redefinição** do método

Tipos Estáticos e Tipos dinâmicos

- Em tempo de **execução** (runTime) – procura do método a executar:



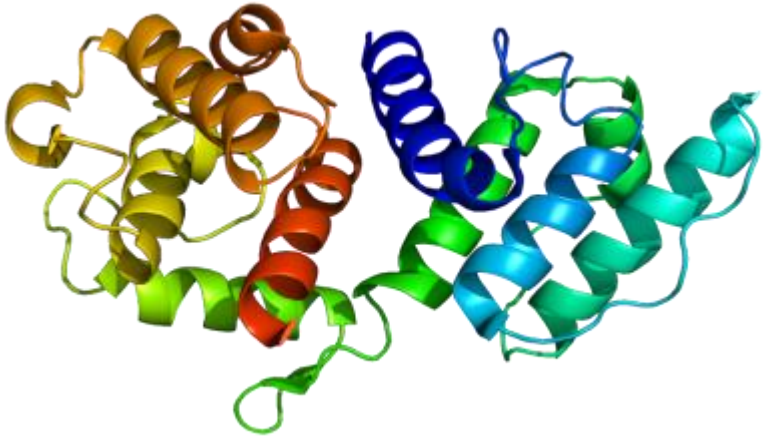
O tipo estático é **Post** e o tipo dinâmico é **PhotoPost**

O método display é procurado na classe do objeto **PhotoPost** e como é encontrado é executado

Neste exemplo **existe herança** e **existe redefinição** do método

Tipos Estáticos e Tipos dinâmicos

- Sumário da **procura do método** a executar (Method lookup) em tempo de execução (Run time):
  1. A variável é acedida
  2. É obtido o nome do método a executar
  3. É obtida a classe do objeto referido
  4. O método a executar é procurado na classe do objeto
  5. Se não for encontrado o método, é procurado na sua superclasse
  6. O passo anterior é repetido até que o método seja encontrado
  7. Os métodos redefinidos aparecem primeiro tendo assim precedência sobre os outros métodos



Módulo 2 – Polimorfismo

SESSÃO 3 – POLIMORFISMO

Polimorfismo

- Quando executamos um método, o método que é executado depende da classe do objeto que o executa. Neste caso dizemos que temos **polimorfismo** de métodos.
  - Um método **várias** (**poli**) **formas** (**Morfo**) de resposta
    - O método `display` tem várias formas de resposta, dentro da classe `MessagePost` tem uma forma, dentro da classe `PhotoPost` tem outra forma.
- Uma **variável polimórfica** pode guardar objetos de vários tipos
- A chamada de métodos em Java é polimórfica.
  - O método que realmente é chamado depende do tipo do objeto em tempo de execução (o tipo dinâmico)

Polimorfismo

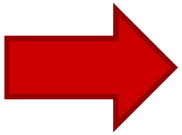
```
Post post = new MessagePost("Zé", "Olá Herança");
post.getText();
Error: cannot find symbol -   method getText()

post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
post.getText();
Error: cannot find symbol -   method getText()
```

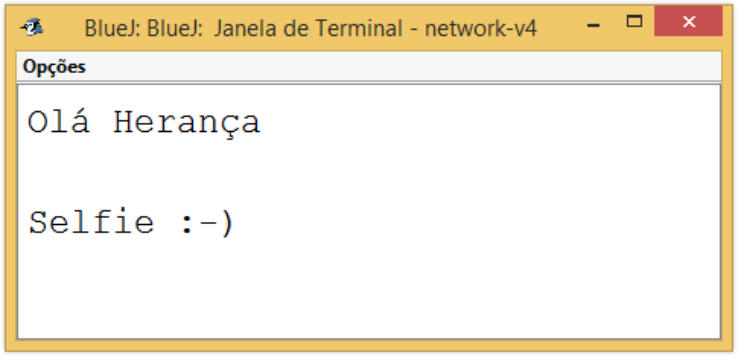
- Para solucionar este erro ter-se-ia que criar o método **getText** na classe **Post** mesmo que não fizesse nada

```
public class Post {

    // restante código omitido

    public String getText(){
        return "";
    }
}
```

Polimorfismo

```
Post post = new MessagePost("Zé", "Olá Herança");
String text = post.getText();
System.out.println(text);

System.out.println();

post = new PhotoPost("Zé", "ze.jpg", "Selfie :-)");
text = post.getText();
System.out.println(text);
```





- Na execução, o método **getText** executado vai corresponder ao método **getText da classe do objeto**

```
public class MessagePost extends Post {
    private String message;
    // código omitido

    @Override
    public String getText()    {
        return message;
    }
}
```

```
public class PhotoPost extends Post {
    private String filename;
    private String caption;
    // código omitido

    @Override
    public String getText()   {
        return caption;
    }
}
```

Polimorfismo

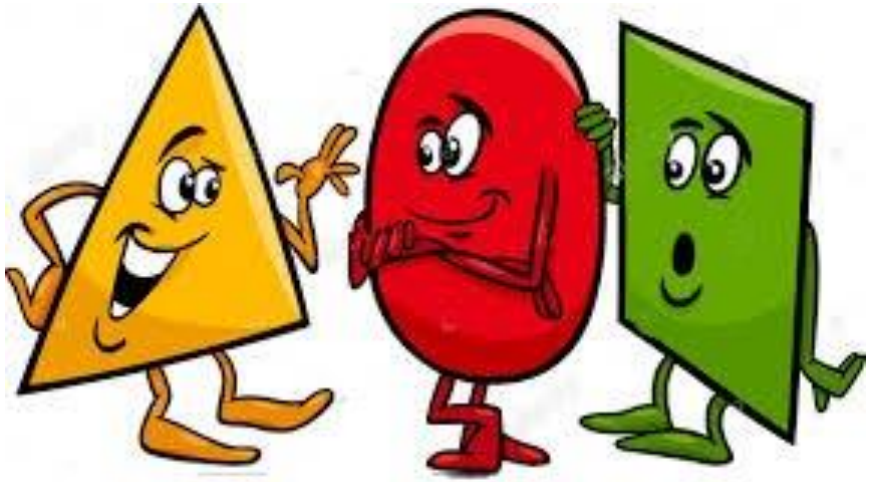
- Poliformismo

Módulo 2 – Polimorfismo

SESSÃO 4 – EXEMPLO FORMAS GEOMÉTRICAS

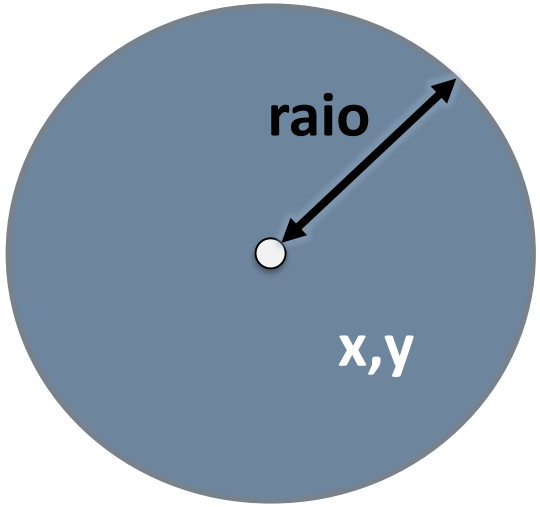
Exemplo – Formas Geométricas

- Requisitos do programa:
  - Desenho de formas geométricas.
  - Representar círculos, quadrados **e retângulos**.
    - Deve ser possível saber as dimensões e a posição de cada um deles.
    - Deve ser possível deslocá-los.

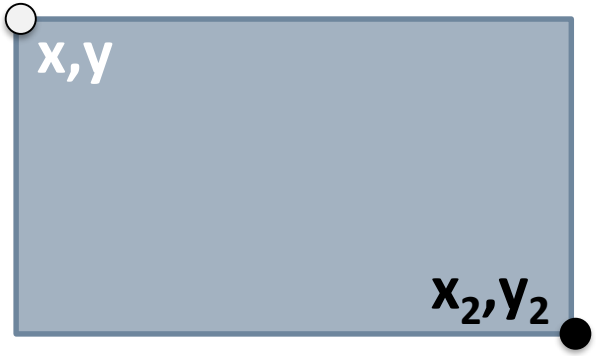


Polimorfismo - exemplo

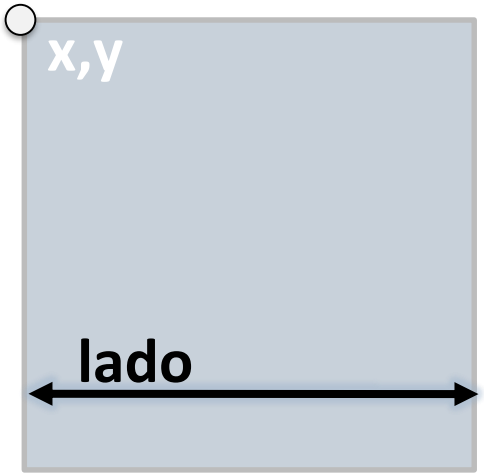
- Exemplo de formas geométricas



Círculo

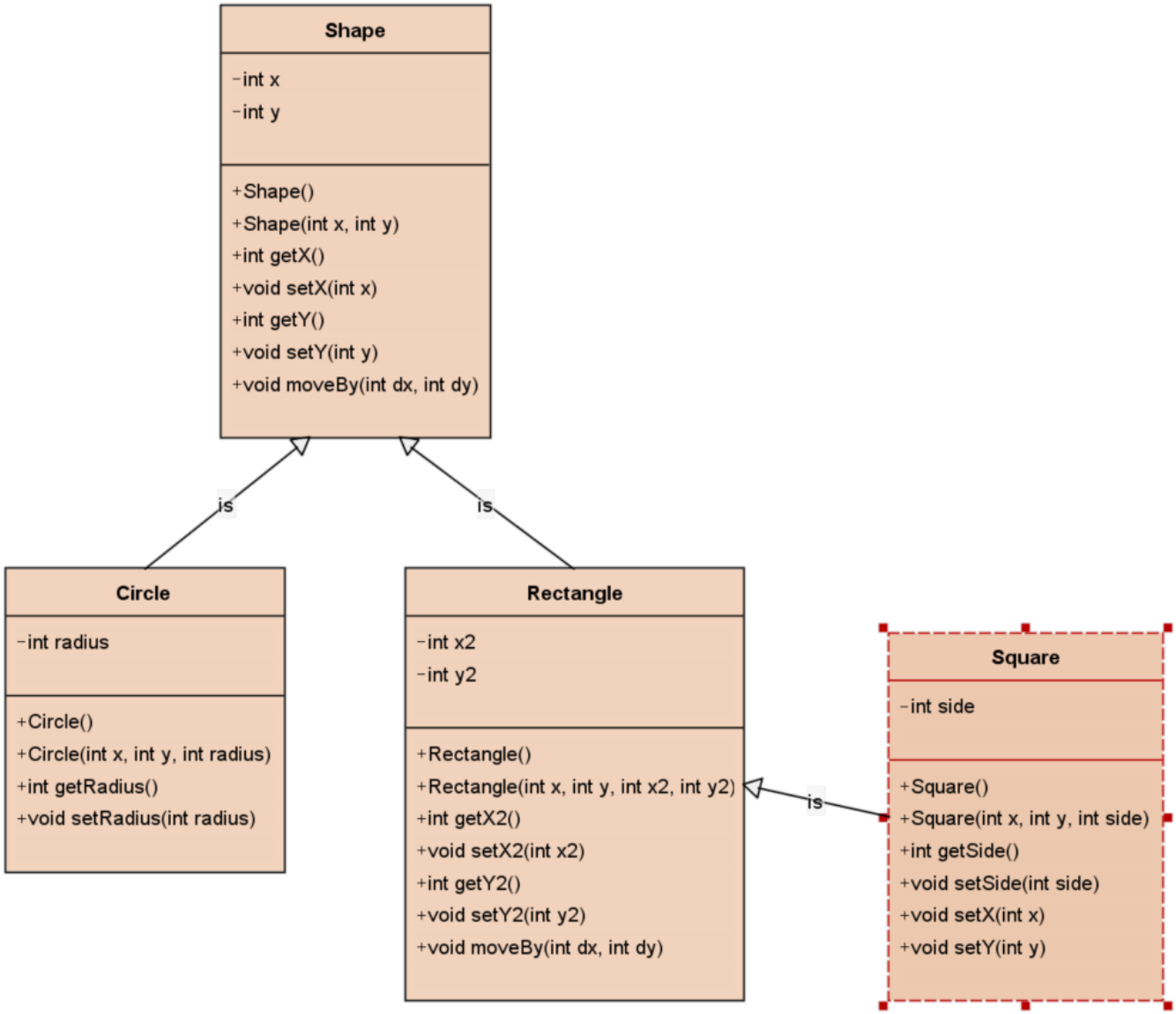


Retângulo



Quadrado

Polimorfismo - exemplo



Polimorfismo - exemplo

```
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

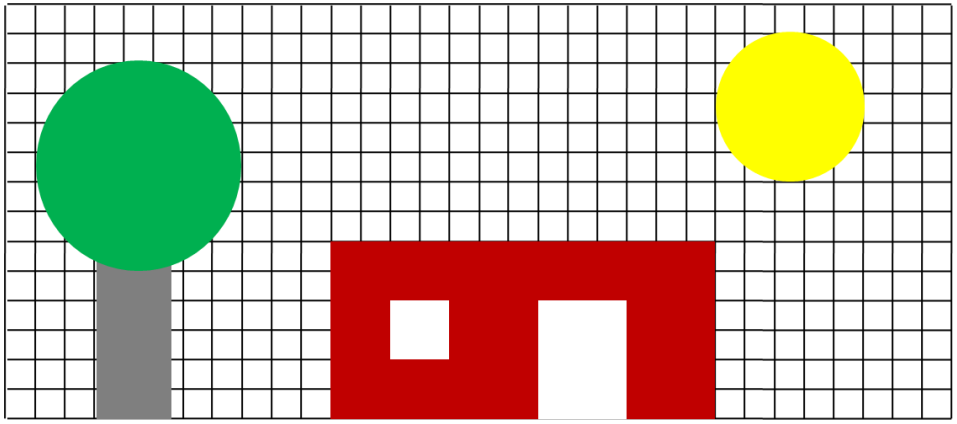
```
public class circle extends Shape {
    private int radius;
    public circle() { this.radius = 1; }
    public circle(int x, int y, int radius) {
        super(x, y); this.radius = radius; }
    public int getRadius() { return radius; }
    public void setRadius(int radius) { this.radius =
radius; }
```

```
public class Rectangle extends Shape {
    private int x2, y2;
    public Rectangle() {this.x2 = 0; this.y2 = 0; }
    public Rectangle(int x, int y, int x2, int y2) {
        super(x, y); this.x2 = x2;this.y2 = y2; }
    public int getX2() { return x2; }
    public void setX2(int x2) { this.x2 = x2; }
    public int getY2() { return y2; }
    public void setY2(int y2) { this.y2 = y2; }
}
```

```
public class Square extends Rectangle {
    private int side;
    public Square() {this.side = 1;}
    public Square(int x, int y, int side) {
        super(x, y, x+side, y+side); this.side = side;}
    public int getSide() {return side;}
    public void setSide(int side) { this.side = side;
        setX2(getX() + side); setY2(getY() - side); }
    @Override public void setX(int x) { super.setX(x);
        setX2(getX() + side); } ... }
```

Polimorfismo - exemplo

- Requisitos
  - Pretende-se criar um desenho com base em formas geométricas.
  - É necessário calcular a área ocupada pelas formas geométricas



Polimorfismo - exemplo

```
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

```
public class circle extends Shape {
    private int radius;
    public circle() { this.radius = 1; }
    public circle(int x, int y, int radius) {
        super(x, y); this.radius = radius; }
    public int getRadius() { return radius; }
    public void setRadius(int radius) { this.radius =
radius; }
```

```
public class Rectangle extends Shape {
    private int x2, y2;
    public Rectangle() {this.x2 = 0; this.y2 = 0; }
    public Rectangle(int x, int y, int x2, int y2) {
        ner(x, y); this.x2 = x2;this.y2 = y2; }
        getX2() { return x2; }
        setX2(int x2) { this.x2 = x2; }
        etY2() { return y2; }
        setY2(int y2) { this.y2 = y2; }
```

Podemos adicionar um método getArea em cada uma das formas geométricas!!!

```
public class Square extends Rectangle {
    private int side;
    public Square() {this.side = 1;}
    public Square(int x, int y, int side) {
        super(x, y, x+side, y+side); this.side = side;}
    public int getSide() {return side;}
    public void setSide(int side) { this.side = side;
        setX2(getX() + side);setY2(getY() - side); }
    @Override public void setX(int x) { super.setX(x);
        setX2(getX() + side); } ... }
```

Polimorfismo - exemplo

```
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, i
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```


```
public class circle extends Shape {
    private int radius;
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
} // Restante código omitido
```

```
public class Rectangle extends Shape {
    private int x2, y2;
    public Rectangle() {this.x2 = 0; this.y2 = 0; }
    public Rectangle(int x, int y, int x2, int y2) {
        super(x, y); this.x2 = x2;this.y2 = y2; }
    public int getX2() { return x2; }
    public void setX2(int x2) { this.x2 = x2; }
    public int getY2() { return y2; }
    public void setY2(int y2) { this.y2 = y2; }
}
```

```
public class Square extends Rectangle {
    private int side;
    public Square() {this.side = 1;}
    public Square(int x, int y, int side) {
        super(x, y, x+side, y+side); this.side = side;}
    public int getSide() {return side;}
    public void setSide(int side) { this.side = side;
        setX2(getX() + side);setY2(getY() - side); }
    @Override public void setX(int x) { super.setX(x);
        setX2(getX() + side); } ... }
```

Polimorfismo - exemplo

```
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```


```
public class circle extends Shape {
    private int radius;
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
} // Restante código omitido
```

```
public class Rectangle extends Shape {
    private int x2, y2;
    @Override
    public double getArea() {
        double dx = x2 - getX();
        double dy = y2 - getY();
        return Math.abs(dx * dy);
    }
} // Restante código omitido
```

```
public class Square extends Rectangle {
    private int side;
    public Square() {this.side = 1;}
    public Square(int x, int y, int side) {
        super(x, y, x+side, y+side); this.side = side;}
    public int getSide() {return side;}
    public void setSide(int side) { this.side = side;
        setX2(getX() + side);setY2(getY() - side); }
    @Override public void setX(int x) { super.setX(x);
        setX2(getX() + side); } ... }
```

Polimorfismo - exemplo

```
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, int y) {
        this.x = x;
        this.y = y;
    }
    publi
    pu
        this
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

getArea do quadrado não é necessária, o método herdado serve

```
public class circle extends Shape {
    private int radius;
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
} // Restante código omitido
```

```
public class Rectangle extends Shape {
    private int x2, y2;
    @Override
    public double getArea() {
        double dx = x2 - getX();
        double dy = y2 - getY();
        return Math.abs(dx * dy);
    }
} // Restante código omitido
```

```
public class Square extends Rectangle {
    private int side;



} // Restante código omitido
```

Polimorfismo - exemplo

```java
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
}
```

Teremos de adicionar o método getArea à classe Shape para o polimorfismo funcionar

```java
public class circle extends Shape {
    private int radius;
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
} // Restante código omitido
```

```java
extends Shape {

() {
- getX();
= y2 - getY();
return Math.abs(dx * dy);
    }
} // Restante código omitido
```

```java
public class Square extends Rectangle {
    private int side;



} // Restante código omitido
```

Polimorfismo - exemplo

```java
public class Shape {
    private int x, y;

    public Shape () {
        x = 0;
        y = 0;
    }
    public Shape (int x, int y) {
        this.x = x;
        this.y = y;
    }

    // Restante código omitido
    public double getArea() {
        return 0.0;
    }
}
```

O método getArea não faz sentido nesta classe mas tem de ser criado para o polimorfismo funcionar. Colocamo-lo a retornar 0.0!

```java
public class circle extends Shape {
    private int radius;
    @Override
    public double getArea() {
        return Math.PI*radius*radius;
    }
} // Restante código omitido
```

```java
public class Rectangle extends Shape {
    private int x2, y2;
    @Override
    public double getArea() {
        double dx = x2 - getX();
        double dy = y2 - getY();
        return Math.abs(dx * dy);
    }
} // Restante código omitido
```

```java
public class Square extends Rectangle {
    private int side;



} // Restante código omitido
```

Polimorfismo - exemplo

```
public class Main{
    public static void main(String[] args) {
        Shape[] shapes;
        shapes = new Shape[6];
        shapes[0] = new circle(4, 8, 3);
        shapes[1] = new circle(28, 10, 4);
        shapes[2] = new Rectangle(3, 0, 5, 6);
        shapes[3] = new Rectangle(11, 0, 23, 6);
        shapes[4] = new Rectangle(18, 0, 21, 4);
        shapes[5] = new Square(13, 2, 2);
        double totalArea = 0;

        for (int i=0; i < shapes.length; i++) {
                totalArea += shapes[i].getArea();
        }
       System.out.println("Area Total: " + totalArea);
    }
 }
```

Usamos o principio da substituição para guardar as diferentes formas geométricas no mesmo *array*

Calculamos a área aplicando o polimorfismo. Cada forma geométrica retorna a sua área

Polimorfismo - exemplo

```
public class Main{
    public static void main(String[] args) {
        ArrayList<Shape> shapes = new ArrayList<>();
         shapes.add(new circle(4, 8, 3));
         shapes.add(new circle(28, 10, 4));
         shapes.add(new Rectangle(3, 0, 5, 6));
         shapes.add(new Rectangle(11, 0, 23, 6));
         shapes.add(new Rectangle(18, 0, 21, 4));
         shapes.add(new Square(13, 2, 2));
         double totalArea = 0;

         for (Shape shape : shapes) {
             totalArea += shape.getArea();
         }
         System.out.println("Area Total: " + totalArea);
    }
 }
```

Bibliografia

- Objects First with Java (6th Edition), David Barnes & Michael Kölling, Pearson Education Limited, 2016
  - Capítulo 11