

Programação Orientada por Objetos

Herança

Prof. José Cordeiro,

Prof. Cédric Grueau,

Prof. Laercio Junior

Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

2019/2020

- ❑ Sessão 1: Protótipo de uma rede social
- ❑ Sessão 2: Herança de classes
- ❑ Sessão 3: Construtores em herança
- ❑ Sessão 4: Herança por extensão
- ❑ Sessão 5: Princípio da substituição



Módulo 1 – Herança de classes

SESSÃO 1 – PROTÓTIPO DE UMA REDE SOCIAL

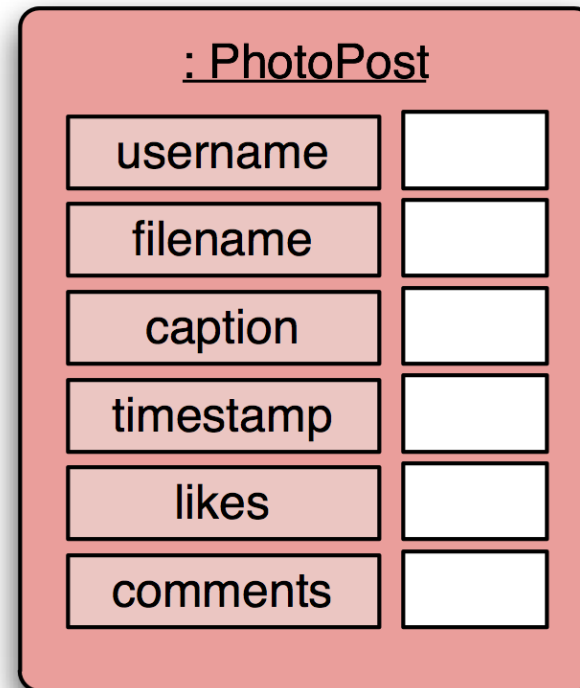
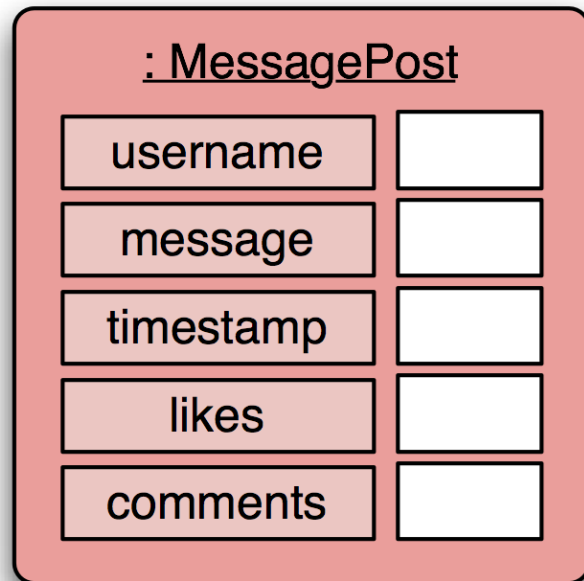
Exemplo — Rede Social

- ❑ 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.



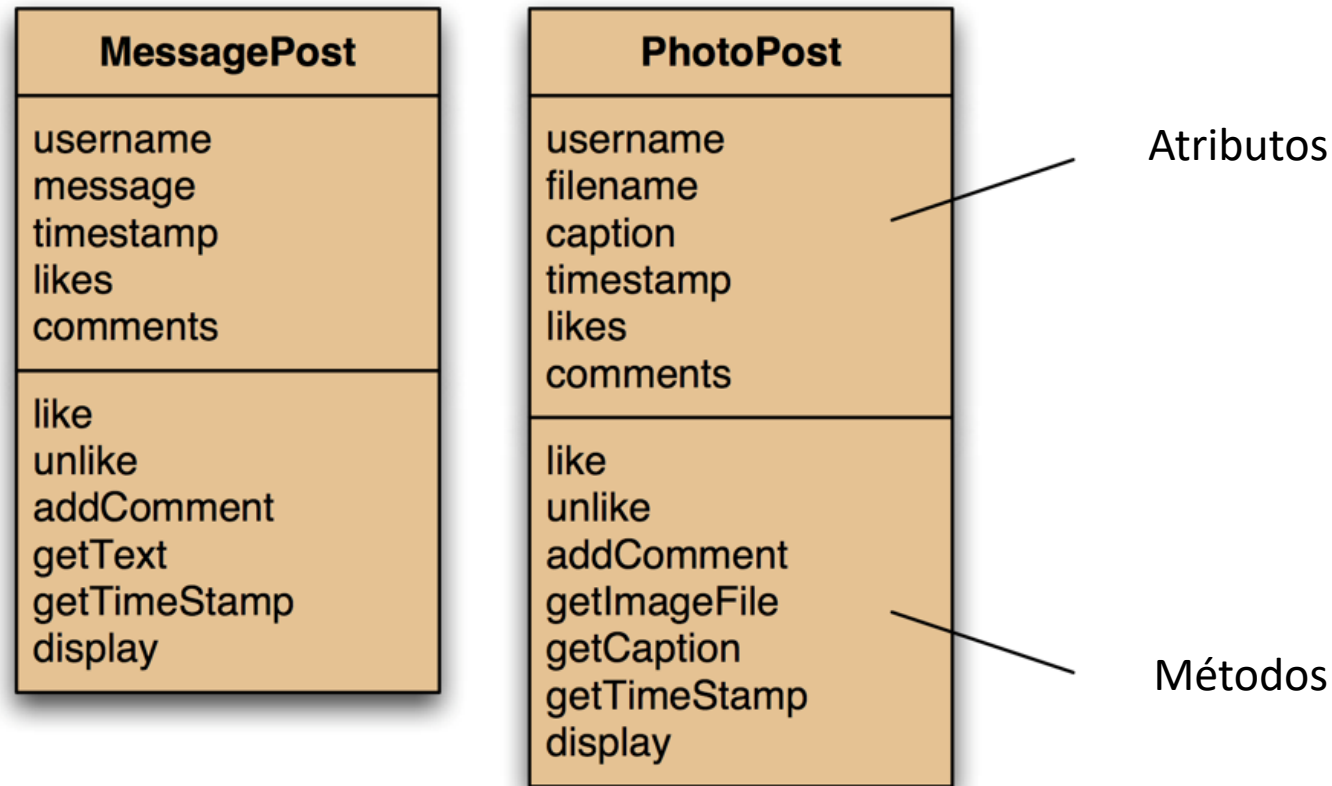
Exemplo — Rede Social

- Objetos que pretendemos representar: **MessagePost** e **PhotoPost**



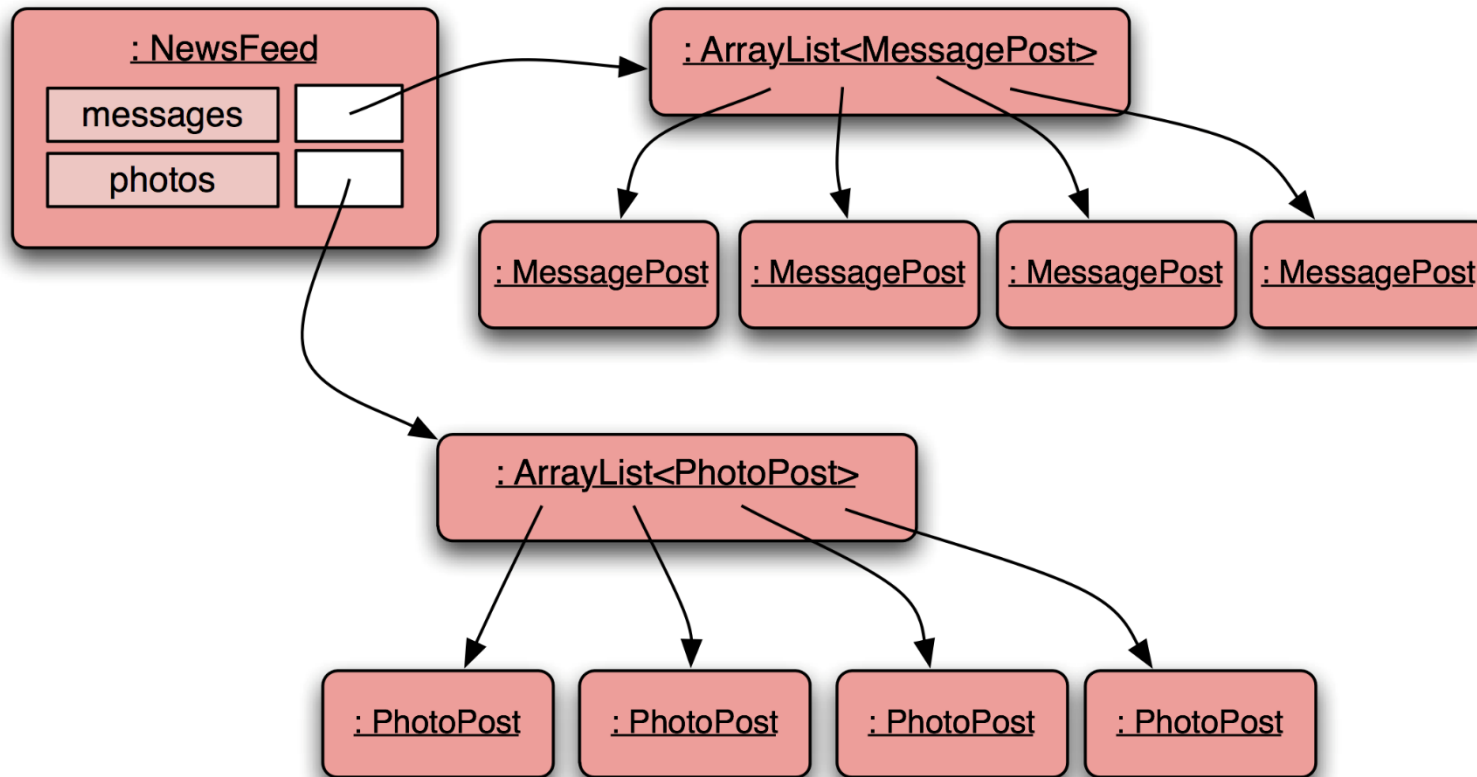
Exemplo — Rede Social

- Classes associadas: **MessagePost** e **PhotoPost**



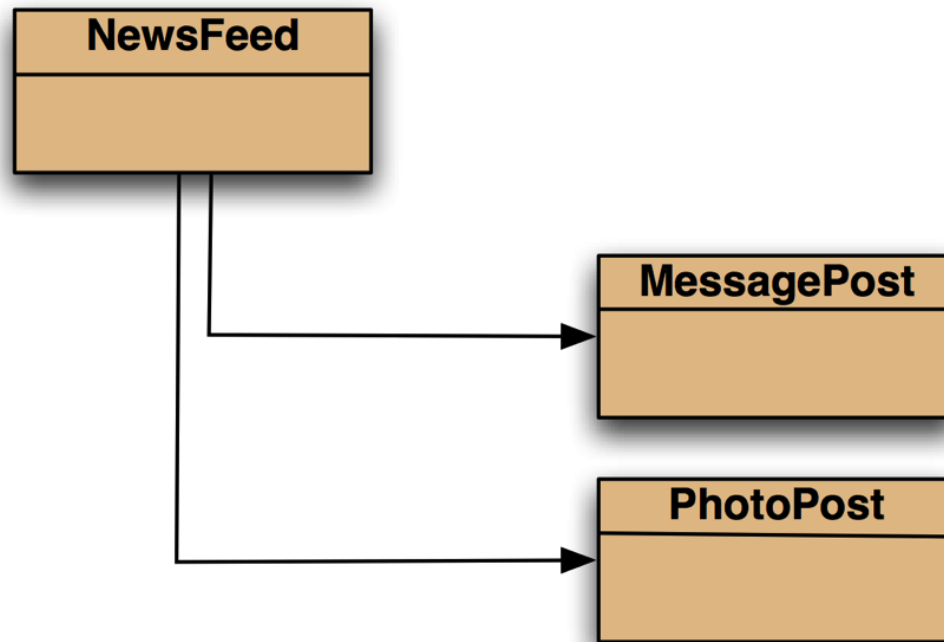
Exemplo — Rede Social

❑ Diagrama de objetos da rede social



Exemplo — Rede Social

□ Diagrama de classes da rede social



Exemplo — Rede Social

□ Classe **MessagePost**

```
public class MessagePost {  
  
    private String username;  
    private String message;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    public MessagePost(String author, String text) {  
        username = author;  
        message = text;  
        timestamp = System.currentTimeMillis();  
        likes = 0;  
        comments = new ArrayList<String>();  
    }  
  
    // continua...
```



currentTimeMillis ???

(System.currentTimeMillis — das bibliotecas do Java)

currentTimeMillis

```
public static long currentTimeMillis()
```

Returns the current time in milliseconds. Note that while the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger. For example, many operating systems measure time in units of tens of milliseconds.

See the description of the class `Date` for a discussion of slight discrepancies that may arise between "computer time" and coordinated universal time (UTC).

Returns:

the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

See Also:

`Date`

Exemplo — Rede Social

□ Métodos da classe **MessagePost** (1/3)

```
public void like() {  
    likes++;  
}  
  
public void unlike() {  
    if (likes > 0) {  
        likes--;  
    }  
}  
  
public void addComment(String text) {  
    comments.add(text);  
}  
  
public String getText() {  
    return message;  
}  
  
public long getTimeStamp() {  
    return timestamp;  
}
```

Exemplo — Rede Social

□ Métodos da classe **MessagePost** (2/3)

```
public void display() {  
    System.out.println(username);  
    System.out.println(message);  
    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.");  
    }  
}
```

Exemplo — Rede Social

□ Métodos da classe **MessagePost** (3/3)

```
private String timeString(long time) {  
    long current = System.currentTimeMillis();  
    long pastMillis = current - time;  
    long seconds = pastMillis/1000;  
    long minutes = seconds/60;  
    if(minutes > 0) {  
        return minutes + " minutes ago";  
    }  
    else {  
        return seconds + " seconds ago";  
    }  
}
```

Tempo passado
em milissegundos

Exemplo — Rede Social

□ Classe **PhotoPost**

```
public class PhotoPost {  
    private String username;  
    private String filename;  
    private String caption;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
}
```

Nome do ficheiro com a imagem

Legenda da imagem

```
public PhotoPost(String author, String filename, String caption)  
{  
    username = author;  
    this.filename = filename;  
    this.caption = caption;  
    timestamp = System.currentTimeMillis();  
    likes = 0;  
    comments = new ArrayList<String>();  
}
```

// continua...

Exemplo — Rede Social

□ Métodos da classe **PhotoPost** (1/3)

```
public void like() {  
    likes++;  
}  
  
public void unlike() {  
    if (likes > 0) {  
        likes--;  
    }  
}  
  
public void addComment(String text) {  
    comments.add(text);  
}  
  
public String getImageFile() {  
    return filename;  
}  
  
public String getCaption() {  
    return caption;  
}
```

Exemplo — Rede Social

□ Métodos da classe **PhotoPost** (2/3)

```
public long getTimeStamp() {
    return timestamp;
}

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


Exemplo — Rede Social

□ Métodos da classe **PhotoPost** (3/3)

```
private String timeString(long time) {  
    long current = System.currentTimeMillis();  
    long pastMillis = current - time;  
    long seconds = pastMillis/1000;  
    long minutes = seconds/60;  
    if(minutes > 0) {  
        return minutes + " minutes ago";  
    }  
    else {  
        return seconds + " seconds ago";  
    }  
}
```

Exemplo — Rede Social

□ Classe **NewsFeed**

```
public class NewsFeed {  
    private ArrayList<MessagePost> messages;  
    private ArrayList<PhotoPost> photos;  
  
    public NewsFeed() {  
        messages = new ArrayList<MessagePost>();  
        photos = new ArrayList<PhotoPost>();  
    }  
  
    public void addMessagePost(MessagePost message) {  
        messages.add(message);  
    }  
  
    public void addPhotoPost(PhotoPost photo) {  
        photos.add(photo);  
    }  
  
    // continua...
```

Exemplo — Rede Social

□ Classe **NewsFeed**

```
// continuação da classe Newsfeed...

public void show() {
    // display all text posts
    for(MessagePost message : messages) {
        message.display();
        System.out.println();    // empty line between posts
    }

    // display all photos
    for(PhotoPost photo : photos) {
        photo.display();
        System.out.println();    // empty line between posts
    }
}
```

Exemplo — Rede Social

❑ Problemas do protótipo criado:

■ Duplicação de código

- ❑ As classes **MessagePost** e **PhotoPost** são bastante parecidas (grande parte do código é idêntico).
- ❑ A manutenção do código dá mais trabalho.
- ❑ Corre-se o risco de se criarem *bugs* se não se alterar em todos os locais onde o código está em duplicado.
- ❑ A classe **NewsFeed** também tem duplicação de código.

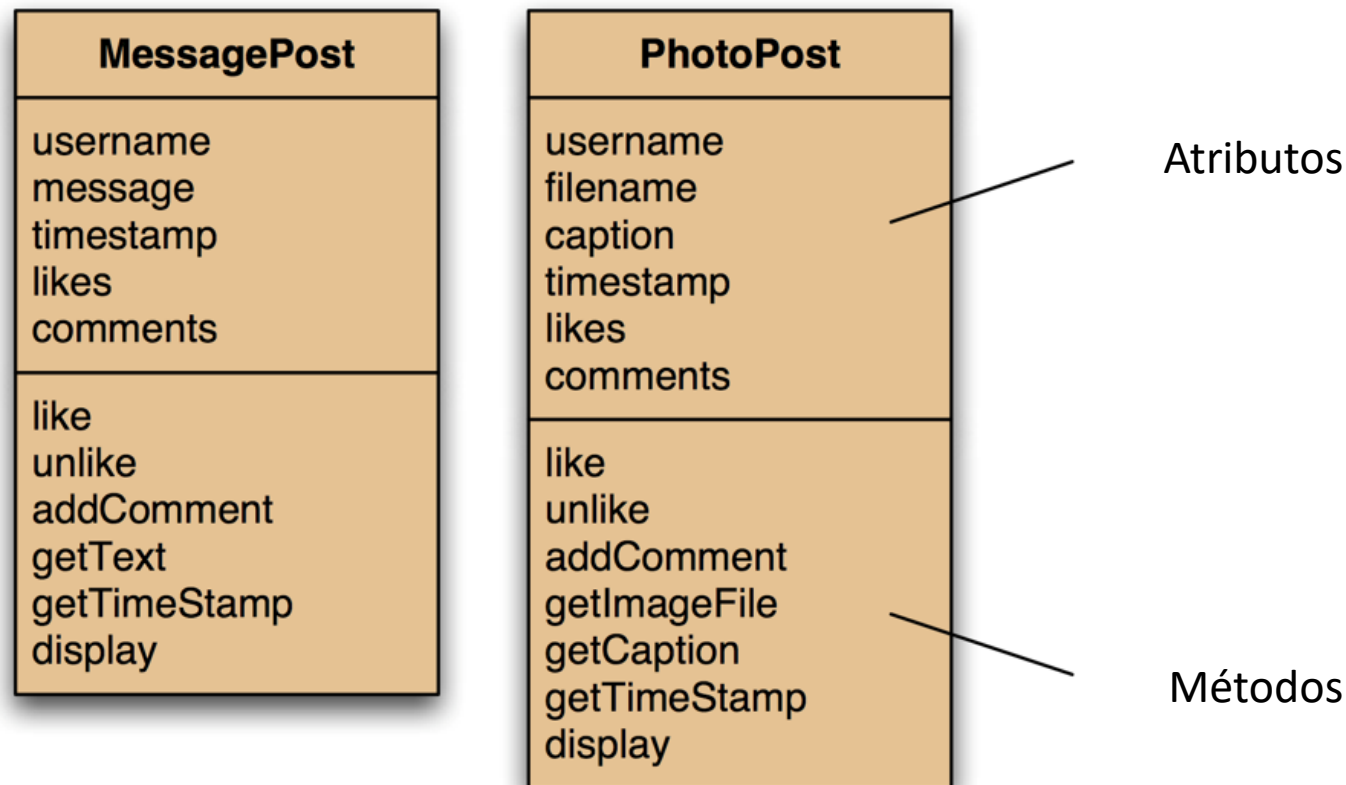


Módulo 1 – Herança de Classes

SESSÃO 2 – HERANÇA DE CLASSES

Exemplo — Rede Social

- Classes **MessagePost** e **PhotoPost**
 - Problema: **Código duplicado**



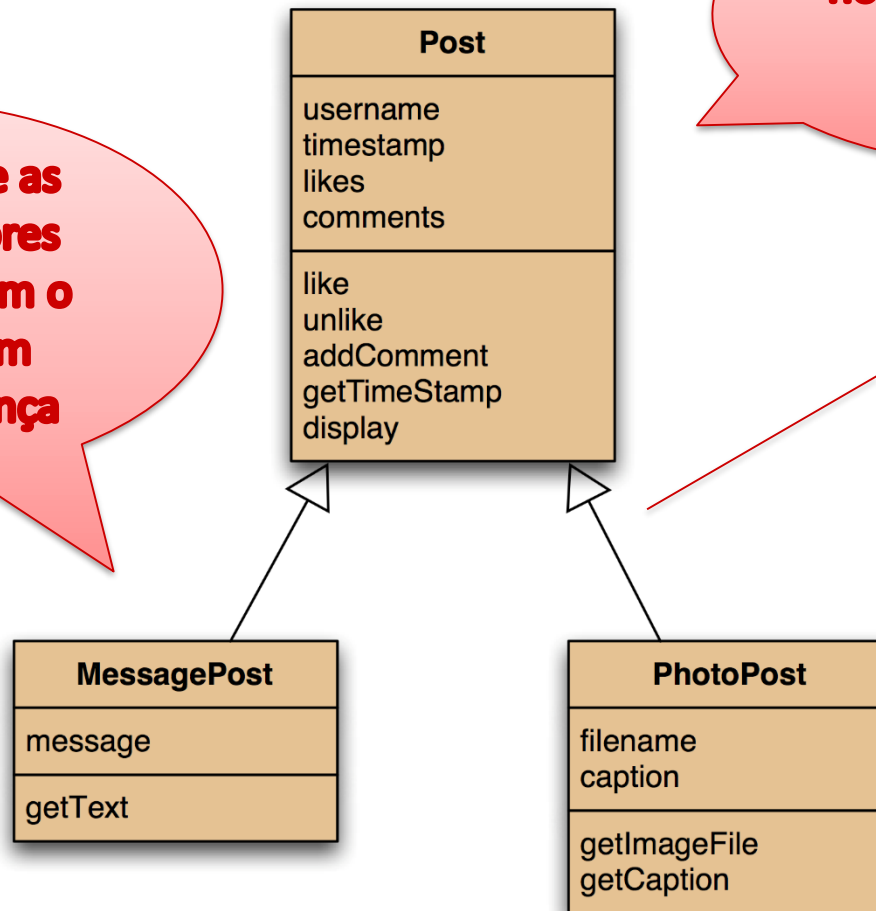
Exemplo — Rede Social

- Solução: **Herança de classes**

Reescrevem-se as classes anteriores para reutilizarem o código comum usando a herança

Constroi-se uma nova classe (Post) com o código comum.

Representa a relação de herança



□ Herança de classes

- É uma técnica usada em Programação Orientada por Objetos que vai permitir a reutilização de código.
 - Define-se uma classe com o código comum.
 - Criam-se outras classes com base na classe anterior que reutilizam esse código.
 - A classe criada inicialmente é a **superclasse** e as classes que vão reutilizar essa classe são as **subclasses**.
 - Ao processo de reutilização de uma classe dá-se o nome de **Herança**
 - As subclasses vão **herdar** o código da superclasse

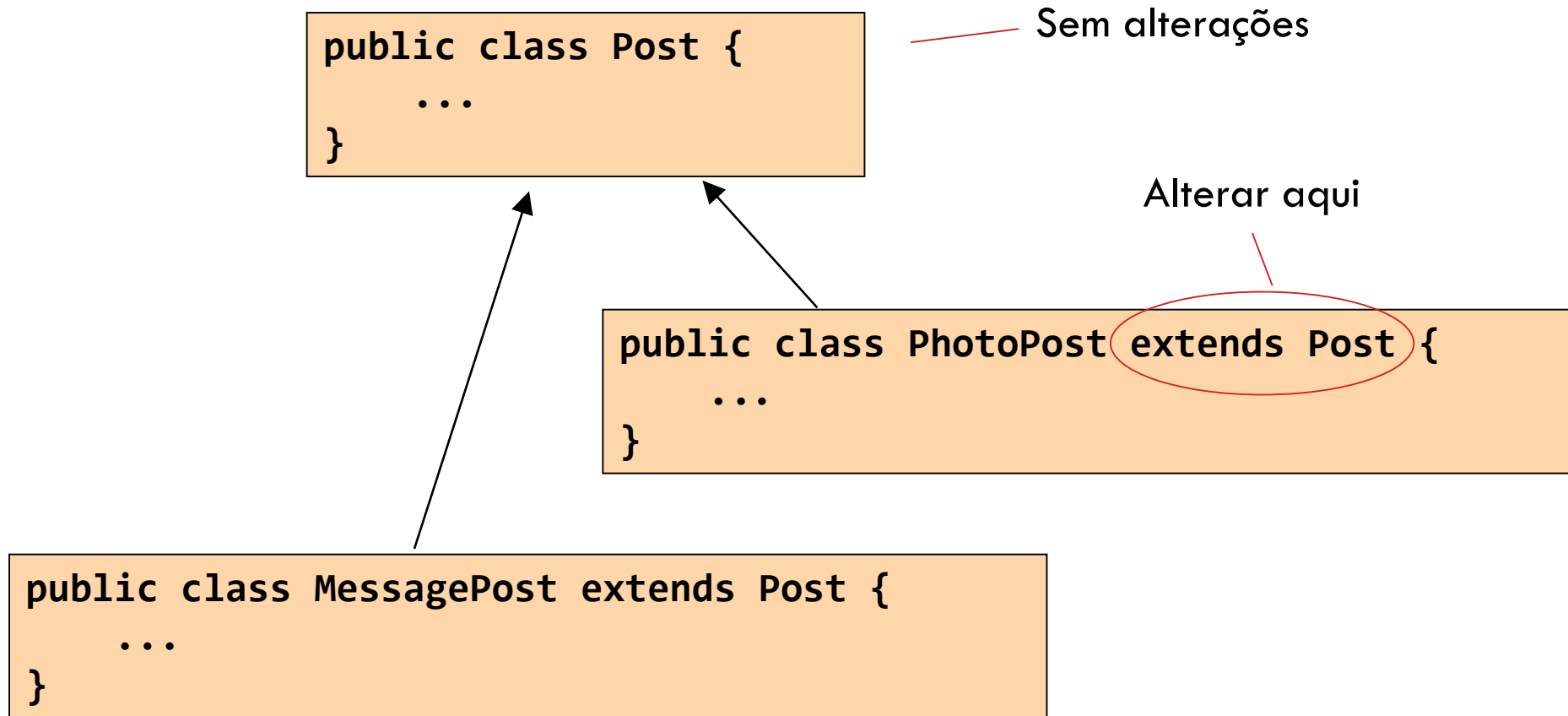
- ❑ Solução: usar a **Herança de classes**

Herança de classes (receita simples)

- Define-se uma **superclasse** : **Post**
- Define-se **subclasses** para **MessagePost** e **PhotoPost**
- Na superclasse definem-se os atributos comuns
- As subclasses **herdam** os atributos da superclasse
- As subclasses adicionam outros atributos

Exemplo — Rede Social

□ Herança de classes em Java



□ Superclasse — **Post**

```
public class Post {  
    private String username;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    // construtores e métodos omitidos  
}
```

Exemplo — Rede Social

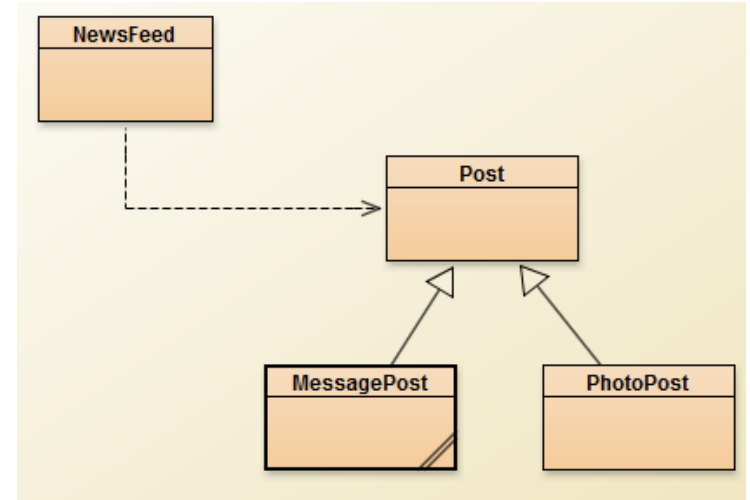
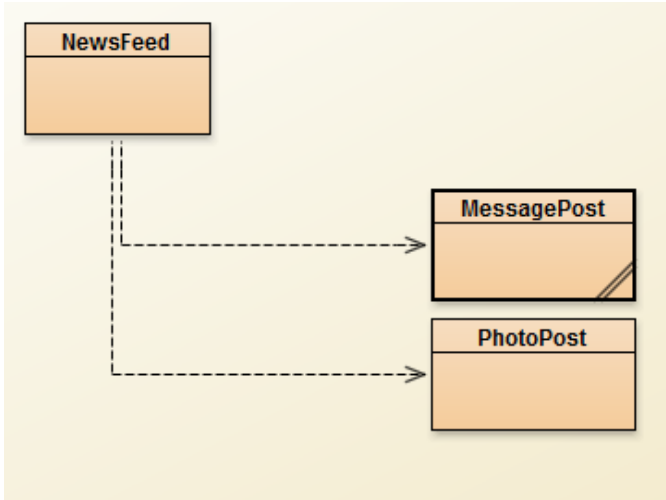
□ Subclasses — **MessagePost** e **PhotoPost**

```
public class MessagePost extends Post {  
    private String message;  
  
    // construtores e métodos omitidos  
}
```

```
public class PhotoPost extends Post {  
    private String filename;  
    private String caption;  
  
    // construtores e métodos omitidos  
}
```

Exemplo — Rede Social

□ Herança de classes em Java



messageP1 : MessagePost	
private String username	"Zé"
private String message	"Sem Herança"
private long timestamp	1456916495696
private int likes	0
private ArrayList<String> comments	
<input type="button" value="Mostrar campos estáticos"/>	
<input type="button" value="Fechar"/>	

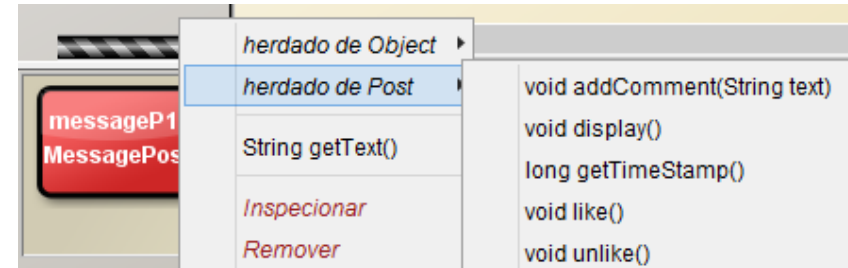
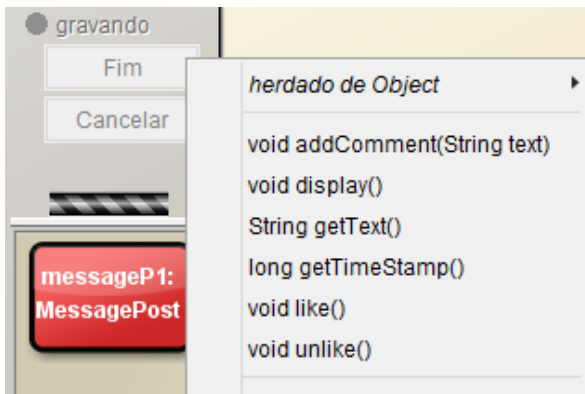
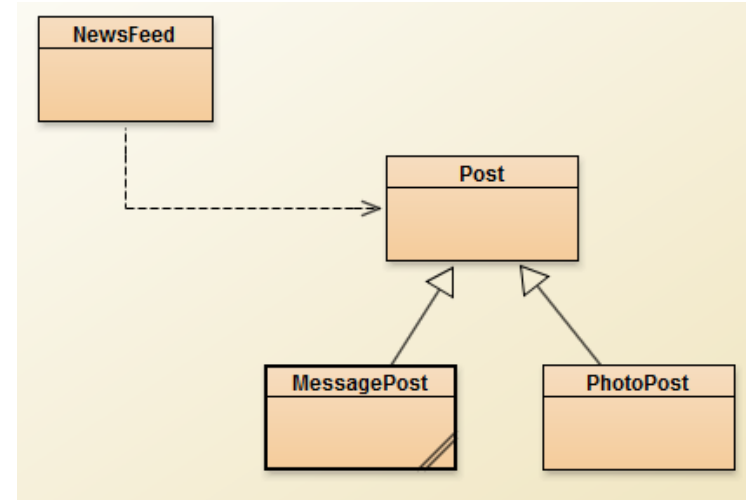
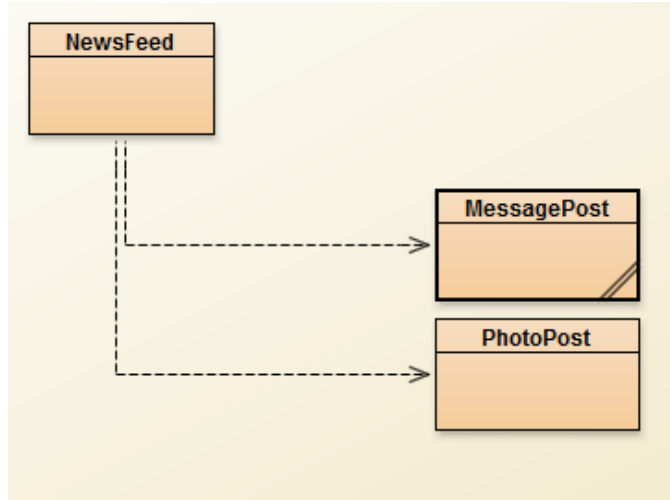


messageP1 : MessagePost	
private String message	"Olá Herança"
private String username	"Zé"
private long timestamp	1456916222369
private int likes	0
private ArrayList<String> comments	
<input type="button" value="Mostrar campos estáticos"/>	
<input type="button" value="Fechar"/>	



Exemplo — Rede Social

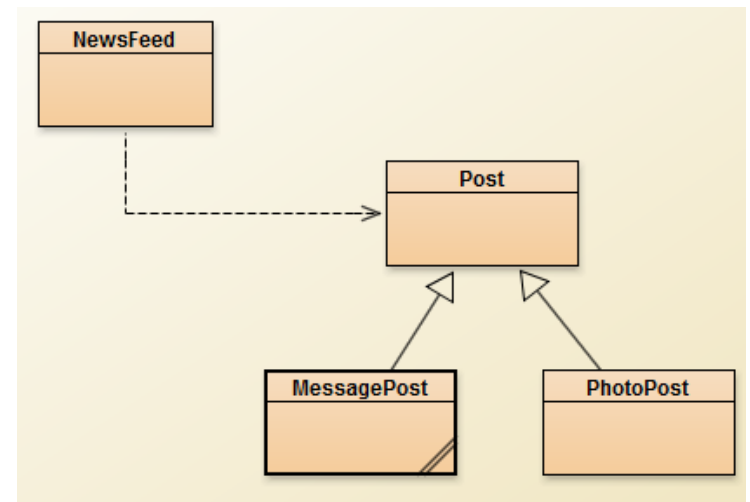
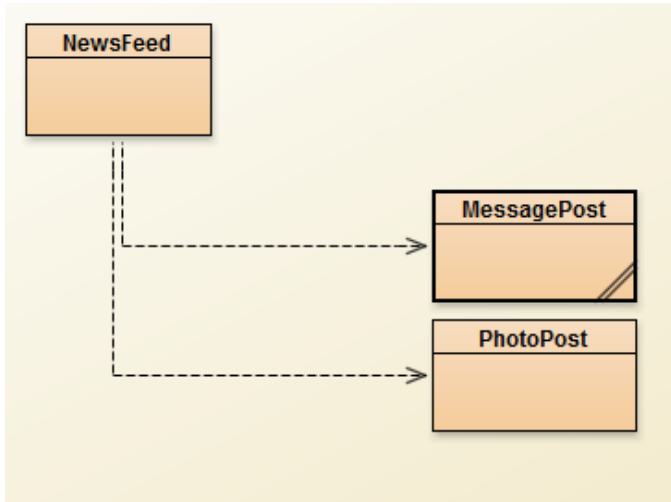
□ Herança de classes em Java



O BlueJ agrupa os métodos nas classes a que pertencem no menu mas a utilização em código não tem diferenças.

Exemplo — Rede Social

□ Herança de classes em Java



```
MessagePost post = new MessagePost("Zé", "Sem Herança");
post.like();
post.display();
String text = post.getText();
System.out.println(text);
```

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

Mesma
utilização

```
BlueJ: BlueJ: Janela de Terminal - network-v1
Opções
Zé
Sem Herança
47 seconds ago - 1 people like this.
    No comments.
Sem Herança
```

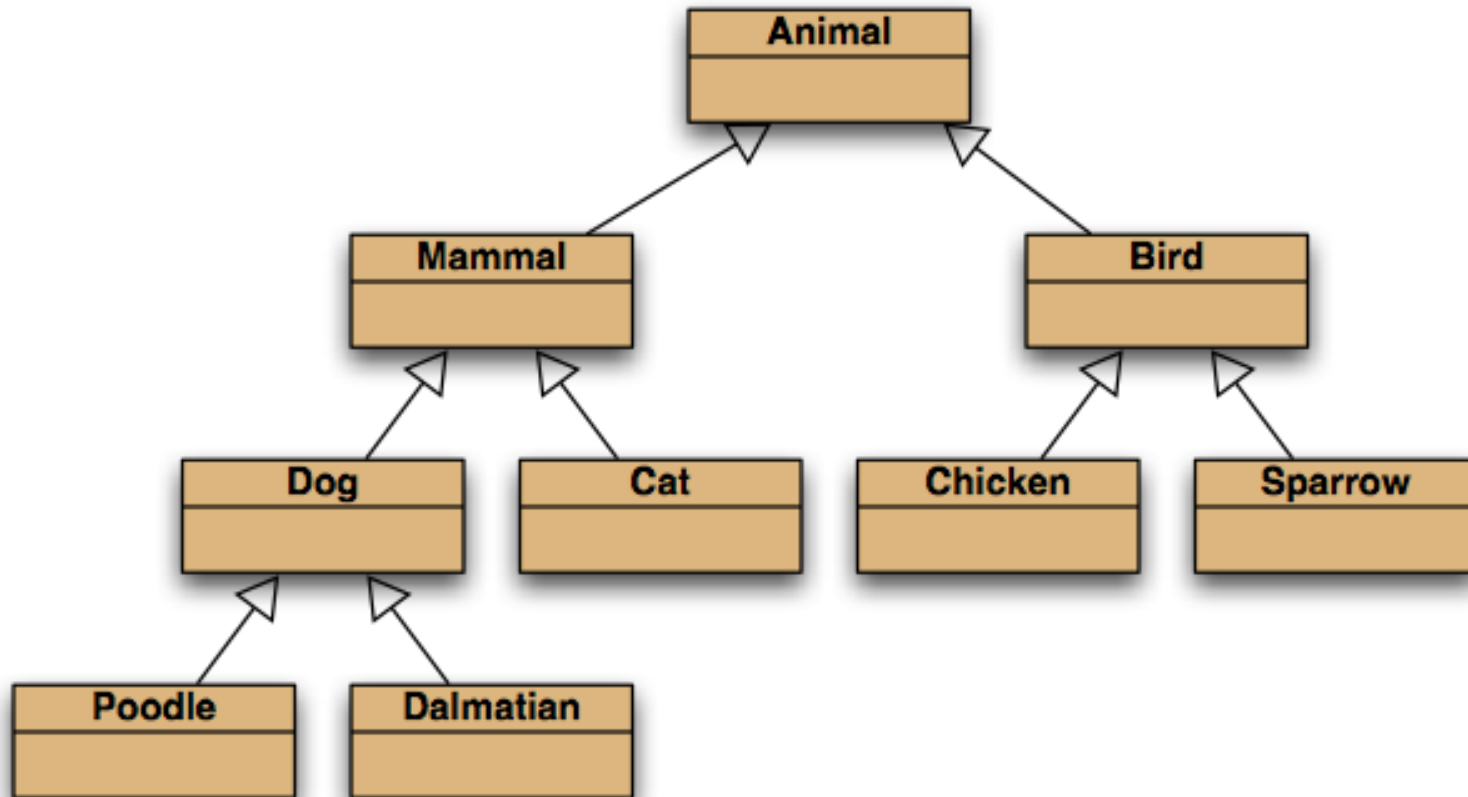
```
BlueJ: BlueJ: Janela de Terminal - network-v2
Opções
Zé
24 seconds ago - 1 people like this.
    No comments.
Olá Herança
```

□ Herança de classes

- A herança de classe define uma relação “**is-a**” entre classes.
 - Ex: A classe **MessagePost** herda da classe **Post**:
Uma mensagem de texto (**MessagePost**) é uma (Is-a) mensagem (**Post**)
 - Ex: A classe **Carro** herda da classe **Veiculo**:
Um carro é um (Is-a) veiculo
- Enquanto a composição define uma relação “**has-a**” entre classes
 - Ex: A classe **Carro** contém (has-a) um objeto da classe **Motor**:
Um carro contém (has-a) motor

Hierarquias de classes

- A herança de classes leva à formação de **hierarquias de classes**

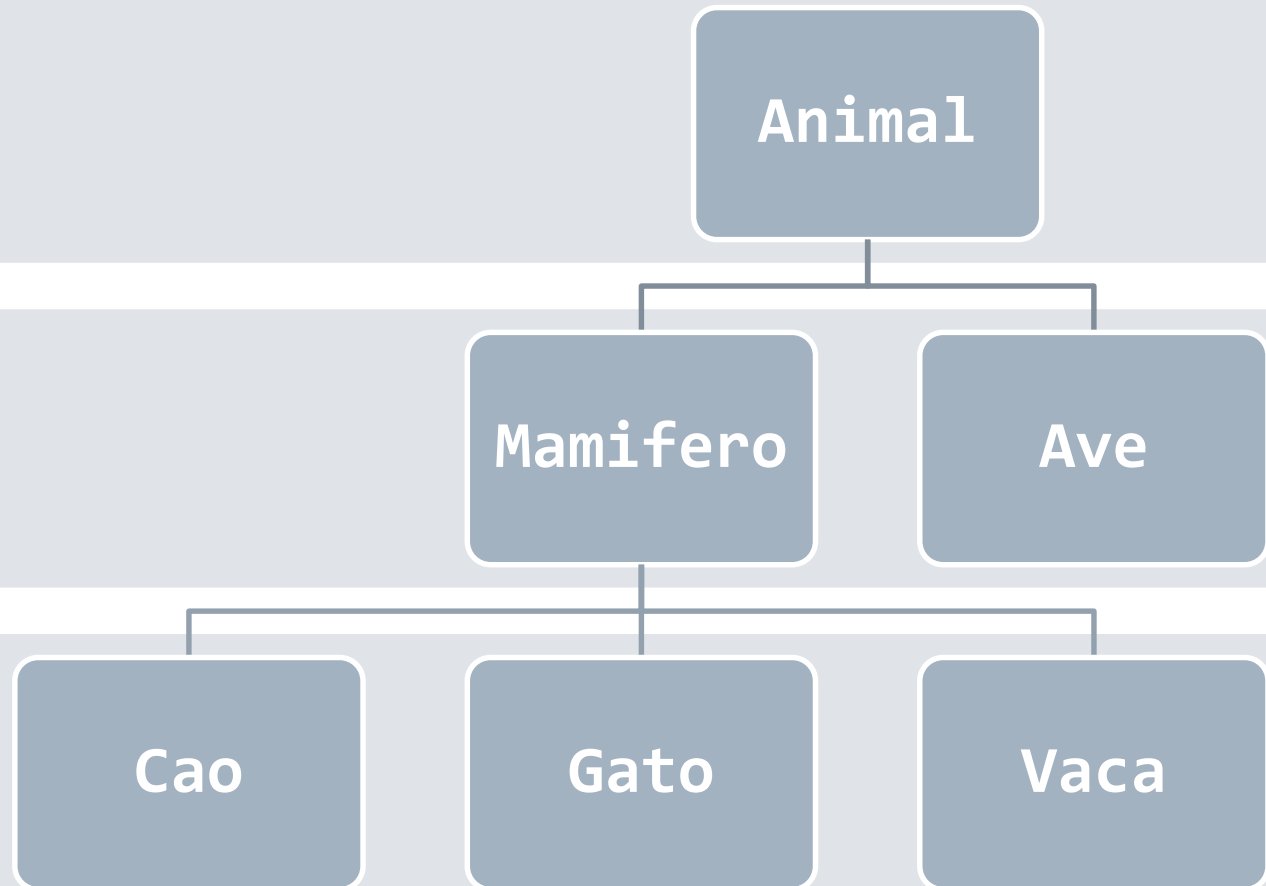


Herança — Hierarquia de Classes - vocabulário

Animal é **superclasse** direta (classe **base** ou classe **pai**) de **Mamifero** e **Ave**, e é super classe indireta de **Cao**, **Gato** e **Vaca**

Mamifero é **subclasse** direta (classe **derivada** ou classe **filha**) de **Animal** e super classe direta (base ou pai) de **Cao**, **Gato** e **Vaca**

Cao é subclasse direta (classe derivada ou classe filha) de **Mamifero** e indirecta de **Animal**



Herança

- ❑ As **subclasses** ou classes derivadas vão **herdar todos os atributos e métodos** da superclasse ou classe base.
- ❑ As subclasses **não herdam os construtores** da superclasse



Módulo 1 – Herança de Classes

SESSÃO 3 – CONSTRUTORES EM HERANÇA

Herança

- ❑ As **subclasses** ou classes derivadas vão **herdar todos os atributos e métodos** da superclasse ou classe base.
- ❑ As subclasses **não herdam os construtores** da superclasse

Herança

- As subclasses **não herdam os construtores** da superclasse

```
public class Post {  
    private String username;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    public Post(String author) {  
        username = author;  
        timestamp = System.currentTimeMillis();  
        likes = 0;  
        comments = new ArrayList<String>();  
    }  
  
    public void like() {  
        likes++;  
    }  
}
```

```
MessagePost message2 = new MessagePost("Ana");  
Error: constructor MessagePost in class MessagePost cannot be applied to give  
required: java.lang.String,java.lang.String  
found: java.lang.String  
reason: actual and formal argument lists differ in length
```

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

```
MessagePost message = new MessagePost("Zé", "Olá");  
message.like();
```

Herança

- ❑ As **subclasses** ou classes derivadas vão **herdar todos os atributos e métodos** da superclasse ou classe base.
- ❑ As subclasses **não herdam os construtores** da superclasse
- ❑ Neste caso é necessário lidar com os construtores de uma forma diferente:
 - Cada subclasse *escolhe* o construtor que vai usar da superclasse
 - A escolha é feita no construtor da subclasse utilizando-se o método **super()** que representa uma chamada ao construtor da **super**classe
 - A escolha do construtor é feita dependendo dos argumentos do método **super()**
Neste caso o número e tipo dos argumentos deve corresponder ao número e tipo de argumentos do construtor que se pretende utilizar da superclasse
 - O método **super()** deve ser o primeiro método a ser chamado dentro do construtor da subclasse

Exemplo — Rede Social

□ Construtores em herança — classe **Post**

```
public class Post {  
    private String username;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    public Post(String author) {  
        username = author;  
        timestamp = System.currentTimeMillis();  
        likes = 0;  
        comments = new ArrayList<String>();  
    }  
  
    // métodos omitidos  
}
```


Exemplo — Rede Social

❑ Construtores em herança — classe **MessagePost**

```
public class MessagePost extends Post {  
    private String message;  
  
    public MessagePost(String author, String text) {  
        super(author);  
        message = text;  
    }  
  
    // métodos omitidos  
}
```

**Chamada ao
construtor da
superclasse (Post)**

Construtores em herança

❑ **Construtores** em herança

- O **construtor da subclasse** deve incluir sempre uma chamada ao construtor da superclasse
 - ❑ Se não for incluída o compilador coloca automaticamente uma chamada ao construtor sem argumentos da superclasse: **super()**;
 - Apenas resulta se a superclasse tiver um construtor sem argumentos
- A chamada ao construtor da superclasse **deve ser a primeira instrução** do construtor da subclasse.

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

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

    // métodos omitidos
}
```

A classe **Post** é a superclasse da classe **MessagePost**

Chamada ao construtor da classe **Post**

Primeira instrução do construtor

Herança

- As subclasses **não herdam os construtores** da superclasse

```
public class Post {  
    private String username;  
    private long timestamp;  
    private int likes;  
    private ArrayList<String> comments;  
  
    public Post(String author) {  
        username = author;  
        timestamp = System.currentTimeMillis();  
        likes = 0;  
        comments = new ArrayList<String>();  
    }  
}
```

```
MessagePost message2 = new MessagePost("Ana");
```

```
public class MessagePost extends Post {  
    private String message;  
  
    public MessagePost(String author) {  
        super(author);  
        message = "";  
    }  
}
```

Novo construtor

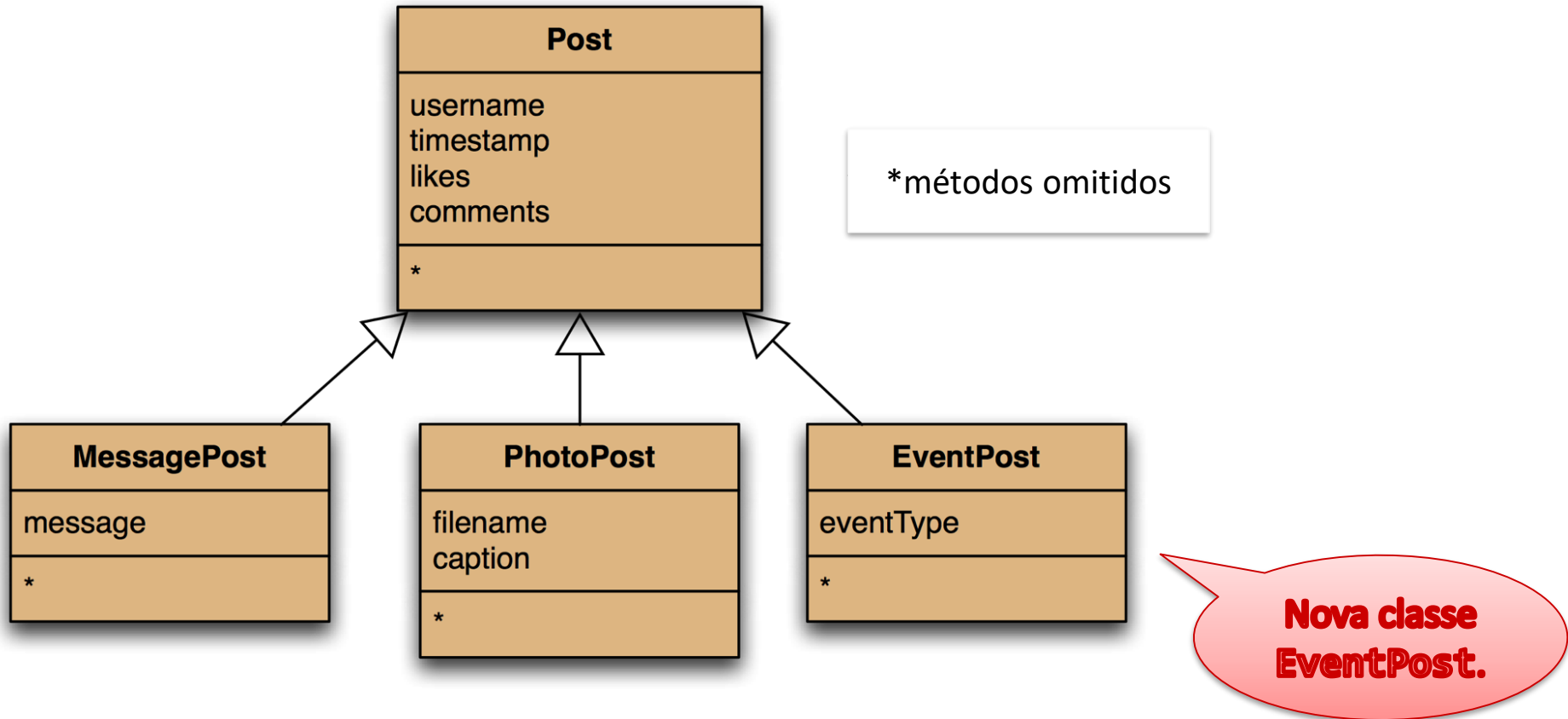


Módulo 1 – Herança de Classes

SESSÃO 4 – HERANÇA POR EXTENSÃO

Exemplo — Rede Social

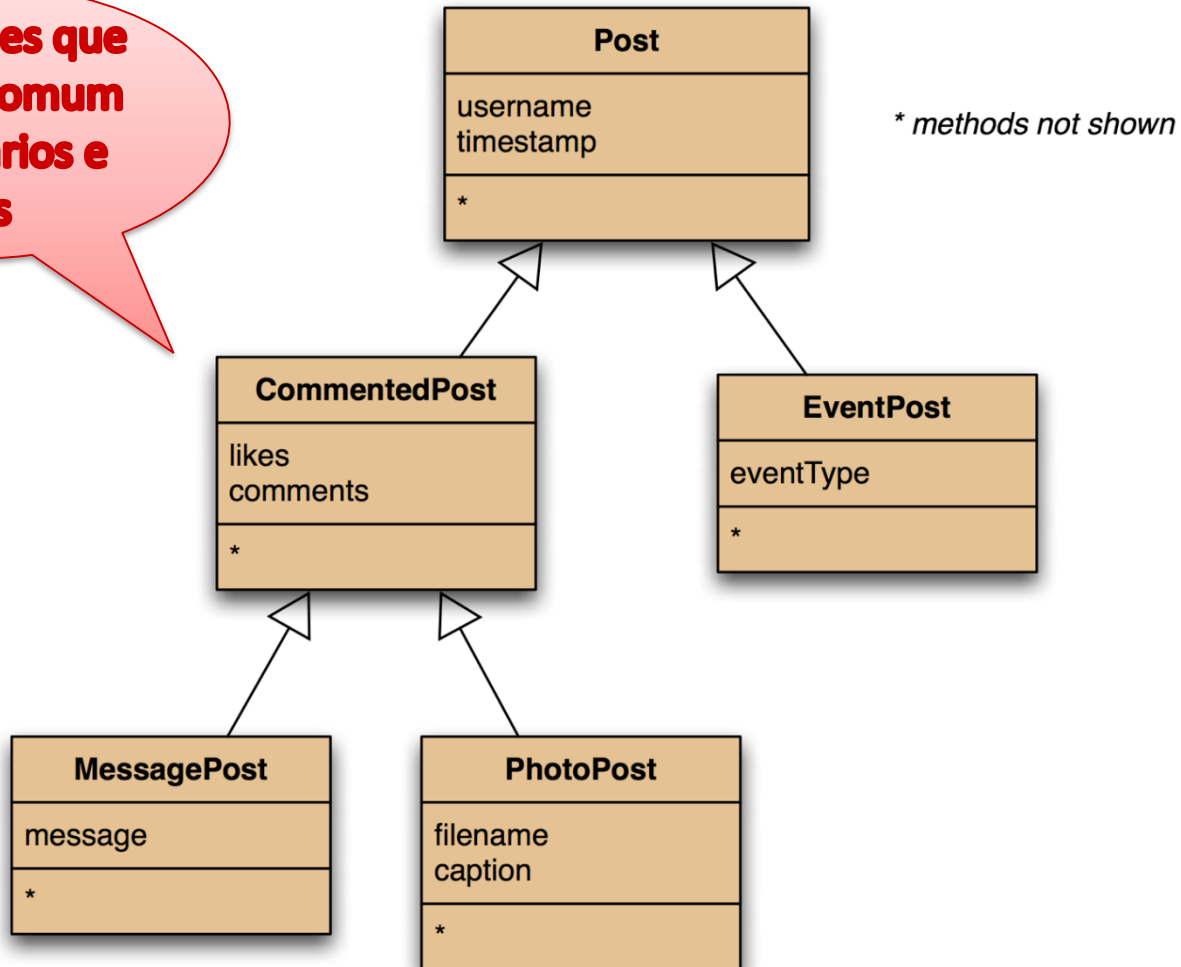
- Adição de outro tipo de Posts – **EventPost**



Exemplo — Rede Social

- Hierarquias mais complexas

Para classes que têm em comum comentários e likes



□ Benefícios da **Herança de classes**

- Evita a duplicação de código
- Permite a reutilização de código
- Simplifica a manutenção
- Promove a extensão de classes



Módulo 1 – Herança de Classes

SESSÃO 5 – PRINCIPIO DA SUBSTITUIÇÃO

Princípio 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 de uma subclasse:
 - Chama-se **princípio da substituição**
 - ❑ Exemplo

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



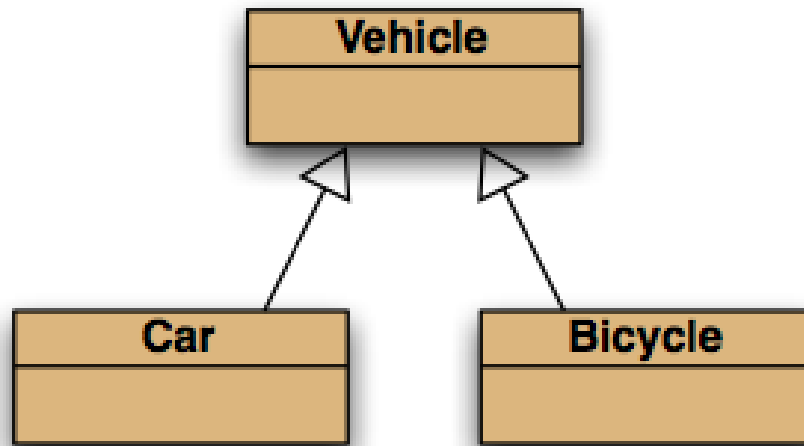
Guarda um objeto da classe **Post**



Atribui-se um objeto da subclasse
MessagePost

Princípio da Substituição

- ❑ **Princípio da substituição** aplicado à **atribuição de valores**



```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

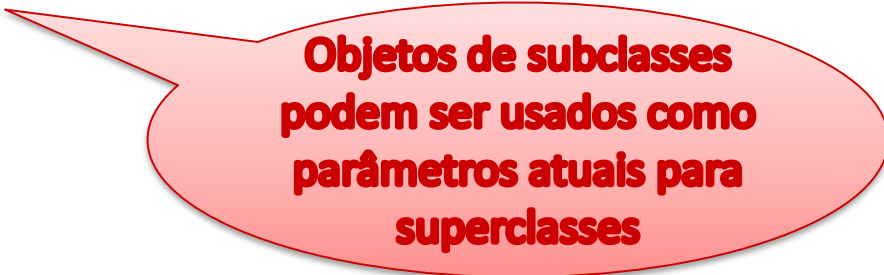
Princípio da Substituição

□ Princípio da substituição aplicado à passagem de parâmetros

```
public class NewsFeed {  
  
    public void addPost(Post post)  
    {  
        ...  
    }  
}
```

```
PhotoPost photo = new PhotoPost(...);  
MessagePost message = new MessagePost(...);
```

```
feed.addPost(photo);  
feed.addPost(message);
```



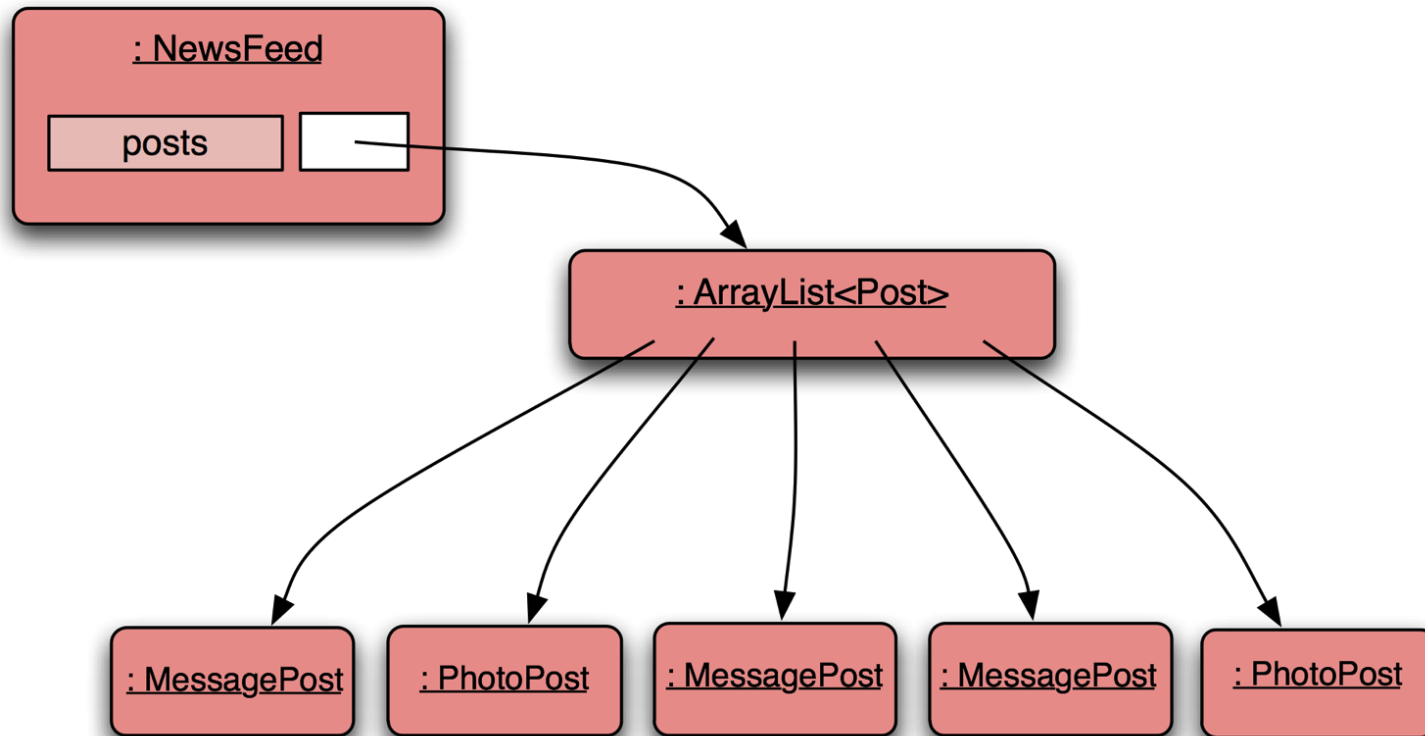
**Objetos de subclasses
podem ser usados como
parâmetros atuais para
superclasses**

□ Nova classe **NewsFeed**

```
public class NewsFeed {  
    private ArrayList<Post> posts;  
  
    public NewsFeed() {  
        posts = new ArrayList<Post>();  
    }  
  
    public void addPost(Post post) {  
        posts.add(post);  
    }  
  
    public void show() {  
        for(Post post : posts) {  
            post.display();  
            System.out.println();  
        }  
    }  
}
```

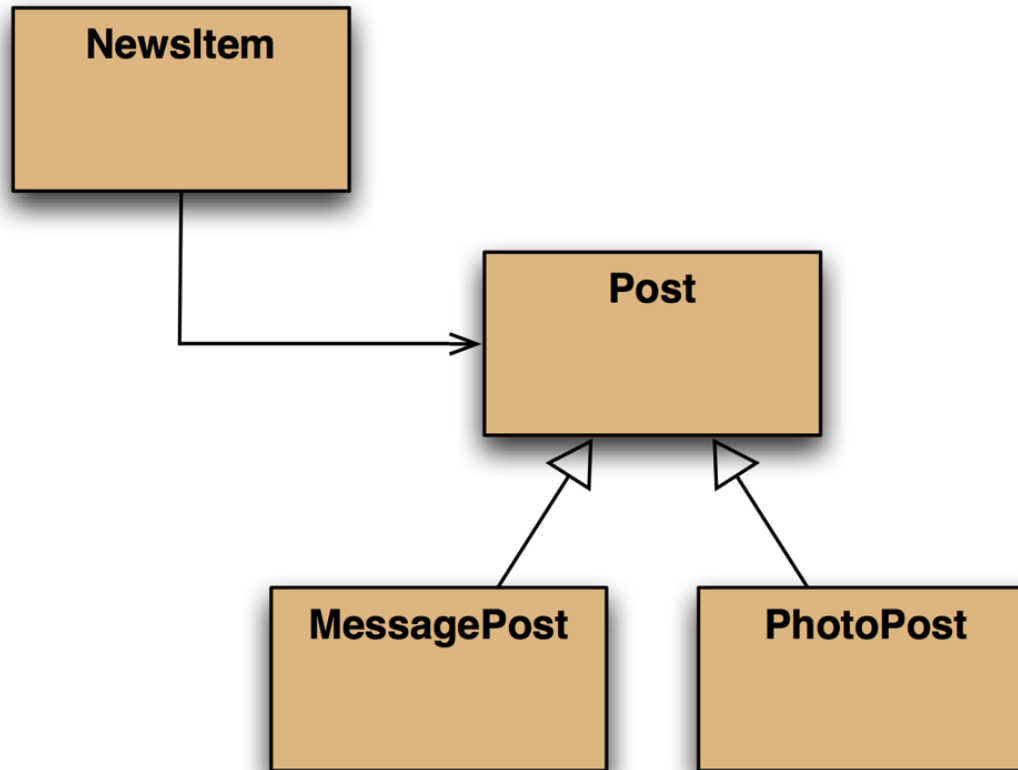
Exemplo — Rede Social

□ Diagrama de objetos



Exemplo — Rede Social

□ Diagrama de classes



Exemplo — Rede Social

❑ Antiga classe versus nova classe **NewsFeed**

```
public class NewsFeed {  
    private ArrayList<MessagePost> messages;  
    private ArrayList<PhotoPost> photos;  
    public NewsFeed() {  
        messages = new ArrayList<MessagePost>();  
        photos = new ArrayList<PhotoPost>();  
    }  
    public void addMessagePost(MessagePost message) {  
        messages.add(message);  
    }  
    public void addPhotoPost(PhotoPost photo) {  
        photos.add(photo);  
    }  
    public void show() {  
        for(MessagePost message : messages) {  
            message.display();  
            System.out.println();  
        }  
        for(PhotoPost photo : photos) {  
            photo.display();  
            System.out.println();  
        }  
    }  
}
```

```
public class NewsFeed {  
    private ArrayList<Post> posts;  
  
    public NewsFeed() {  
        posts = new ArrayList<Post>();  
    }  
  
    public void addPost(Post post) {  
        posts.add(post);  
    }  
  
    public void show() {  
        for(Post post : posts) {  
            post.display();  
            System.out.println();  
        }  
    }  
}
```

Bibliografia

- Objects First with Java (6th Edition), David Barnes & Michael Kölling, Pearson Education Limited, 2016
 - Capítulo 10

