UNIP

UNIVERSIDADE PAULISTA

Linguagem de Programação Orientada a Objeto

Polimorfismo, Classes Abstractas e Interfaces

Professora Sheila Cáceres

Polimorfismo

Polimorfismo

"Polimorfismo é a característica única de linguagens orientadas a objetos que permite que diferentes objetos respondam a mesma chamada de métodos cada um a sua maneira."

Polimorfismo

- É a capacidade de objetos instanciados de diferentes classes (com uma superclasse comum) responderem à chamada de métodos, de forma diferente (particular)
- Capacidade de um objeto decidir que método aplicar a si mesmo.
- Capacidade de assumir formas diferentes.
- Permite programar de forma genérica para manipular de uma grande variedade de classes.

Exemplo de Polimorfismo

```
public class Figura {
        public double calcularArea( ) {
                return 0;
        }
}

public class Quadrado extends Figura {
        double lado;
        public Quadrado(double lado) {
                this.lado = lado;
        }

        public double calcularArea( ) {
                double area = 0;
                area = lado * lado;
                return area;
        }
}
```

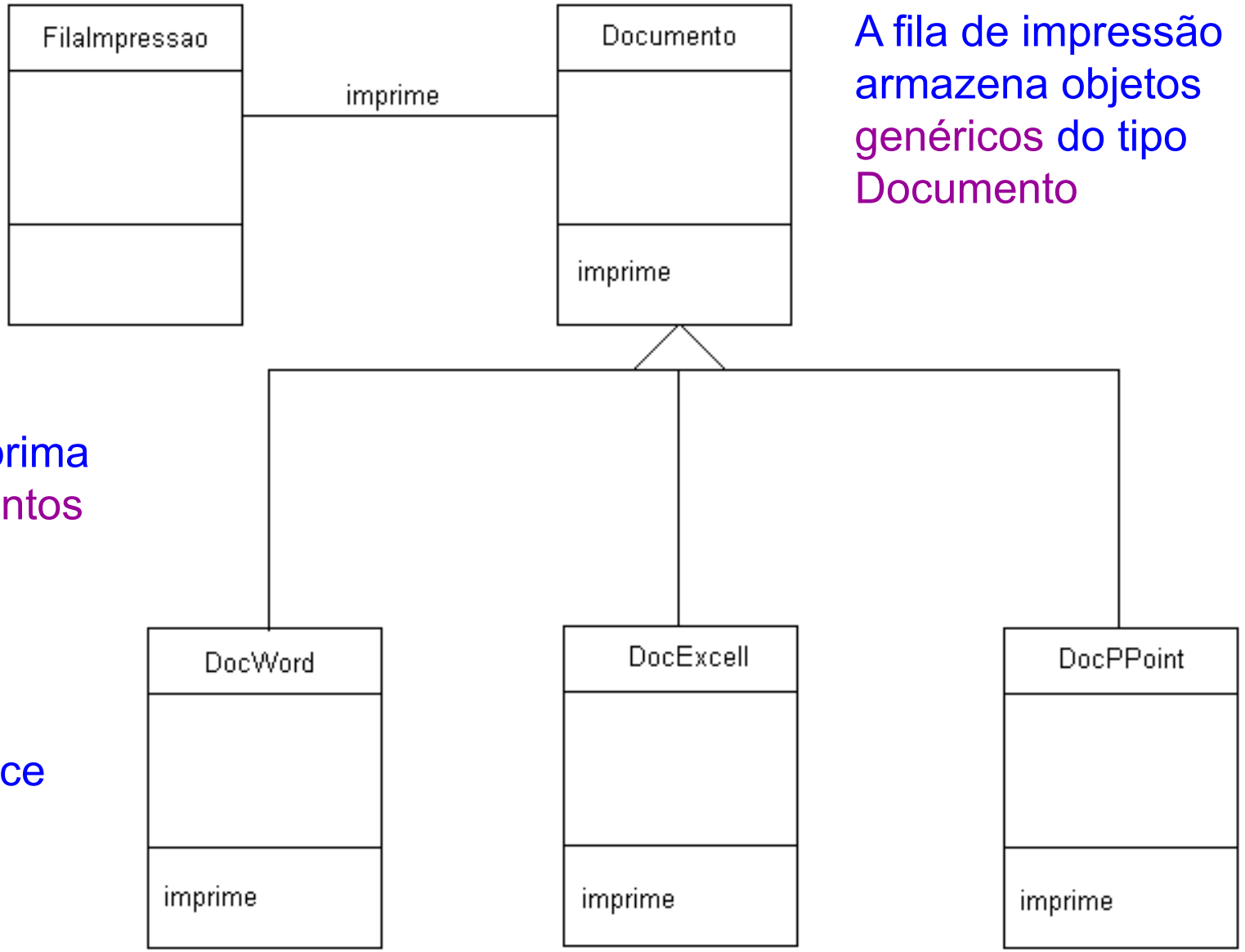
Exemplo de Polimorfismo

```
public class Circulo extends Figura {

        double raio;

        public Circulo (double raio) {
                this.raio = raio;
        }

        public double calcularArea( ) {
                double area = 0;
                area = 3.14 * raio * raio;
                return area;
        }
}
```

Exemplo de Polimorfismo

```
public class Main {

     public static void main(String[] args) {

          Figura f1 = new Quadrado(4);

         Figura f2 = new Circulo(2);

          System.out.println("Área da Figura 1 é: "
                        + f1.calcularArea( ) + "\n"
                      + "Área da Figura 2 é: "
                        + f2.calcularArea( ));
          }
}
```

Outro Exemplo

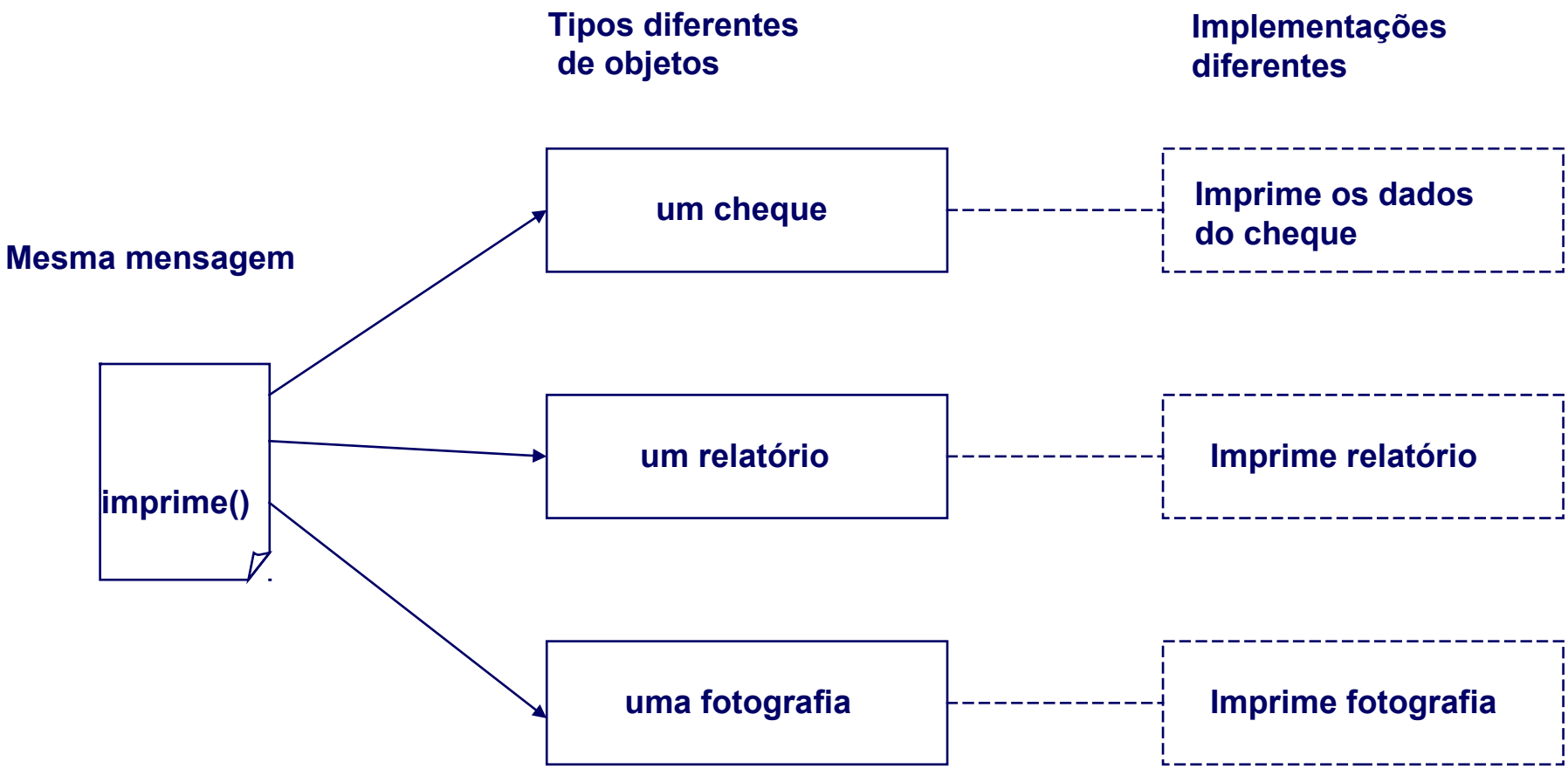


A fila de impressão armazena objetos genéricos do tipo Documento

A mensagem transmitida é: imprima todos os Documentos da fila

Cada tipo de documento conhece seus detalhes de impressão

Polimorfismo



Vantagens do Polimorfismo

- **Permitir que vários objetos de um mesmo tipo (classe pai) sejam tratados da mesma maneira e possam ter comportamentos diferentes (a traves da implementação nas classes filhas)**.
- Uma outra vantagem é **permitir aumentar um software de maneira mais controlada, mais localizada**.
- Considere o exemplo da fila de impressão. Se quisermos incrementar o software e permitir que novos tipos de documentos sejam impressos, a classe *FilaImpressao* não precisa ser alterada. Somente novas classes precisam ser criadas para implementar os novos tipos de documentos. Assim, o trabalho é menor e mais localizado, evitando que erros de programação sejam inseridos na classe *FilaImpressao*.

Aspectos a considerar

- Usa-se uma **variável de um tipo único** (normalmente do tipo da superclasse) para armazenar objetos variados do tipo das subclasses.
- O tipo declarado de uma variável é seu tipo estático (Compiladores verificam os tipos estáticos), o tipo de um objeto é seu tipo dinâmico (Tipos dinâmicos são utilizados em tempo de execução).
- Usa-se uma instancia de um objeto da super-classe (tipo genérico) para chamar a um método que foi reescrito em uma das sub-classes. O tipo específico do objeto não é conhecido até a execução do programa. A escolha do método a ser executado é feita em tempo de execução.
- É uma das ferramentas mais poderosas e um dos pilares do paradigma orientado a objetos.

Os métodos da classe `Object`

- Métodos em `Object` são herdados por todas as classes.
- Qualquer um desses pode ser sobrescrito.
- O método `toString` é comumente sobrescrito:
  - `public String toString()`
    - Retorna uma representação de string do objeto.

Modificador *final*

- Uma variável ou atributo pode ser marcado como final para se tornar uma constante

  final double PI = 3.14;

- Um método pode ser marcado como final para impedir que seja sobrescrito.

  public final void meuMetodo(){}

- Uma classe pode ser marcada como final para impedir que possa ser estendida com subclasses.

  public final class Color{}

Classes Abstratas

Classes Abstratas

- Ao subir na hierarquia de heranças, as classes se tornam mais genéricas e, provavelmente mais abstratas
- Em algum ponto, a classe ancestral se torna tão geral que acaba sendo vista mais como um **modelo** para outras classes do que uma classe com instâncias específicas que são usadas
- Uma classe abstrata **não pode ser instanciada**, ou seja, não há objetos que possam ser construídos diretamente de sua definição. Classes abstratas correspondem a especificações genéricas, que deverão ser concretizadas em classes derivadas (subclasses).
- Uma classe abstrata serve apenas para definir um comportamento comum que todas as classes derivadas devem seguir.

Classes abstratas

- Classes abstratas são classes que não podem ser instanciadas, mas é possível **declarar** uma variável (referência) deste tipo.
- São utilizadas apenas para permitir a derivação de novas classes.
- Sintaxe:

```
abstract class NomeDaSuperclasse
{ // corpo da classe abstrata... }
```

- Portanto:

  NomeDaSuperclasse f = new NomeDaSuperclasse( ); –> Erro

Classes Abstratas X Classes Concretas

- Uma classe abstrata é uma classe que não tem instâncias diretas.
- Uma classe concreta é uma classe que pode ser instanciada.
- As classes abstratas podem possuir **métodos abstratos.**

Métodos Abstratos

- No exemplo da Figura, se nenhum objeto da classe *Figura* poderá ser criado, não importa o que esteja implementado no método *calcularArea*, pois esse **código nunca será executado.**
- Pelo polimorfismo **somente os métodos das classes derivadas serão executados**.
- O Java permite que **métodos assim sejam definidos como abstratos** e, portanto, **sem nenhuma implementação**.
- Para criar um método abstrato, usamos a palavra-chave *abstract* na assinatura do método e omitimos o seu corpo (já que não há implementação):

```
public abstract class Figura {
      public abstract double calcularArea( );
}
```

Classes e Métodos Abstratos

- Um **método abstrato** promete que todos os descendentes não abstratos dessa classe abstrata irão implementar esse método abstrato
- Os **métodos abstratos** funcionam como uma espécie de guardador de lugar para métodos que serão posteriormente implementados nas **subclasses**
- Uma classe pode ser declarada como abstrata mesmo sem ter métodos abstratos

Regras sobre Classes Abstratas

- Toda **classe derivada** de uma classe abstrata **deve obrigatoriamente implementar os métodos** abstratos da superclasse, **caso contrário um erro de compilação é gerado**.

- **Uma classe que tenha um ou mais métodos abstratos deve ser obrigatoriamente definida como abstrata**, caso contrário um erro de compilação é gerado.

- Uma classe abstrata pode conter métodos não abstratos, isto é, com implementação.

- Se esses métodos não abstratos não forem definidos (sobrepostos) nas subclasses, então, quando um objeto da subclasse realizar a chamada a um desses métodos, o código contido na classe abstrata será executado (devido à herança).

Exemplo

```
public abstract class Figura {
    public abstract double calcularArea( );

    public void imprimeArea( ){
        System.out.println(calcularArea( ));
    }
}

public class Main {
     public static void main(String[] args) {
          Quadrado q = new Quadrado(4);
          Circulo c = new Circulo(2);
         System.out.print("Área da Figura 1 é: ");
          q.imprimeArea( );
         System.out.print("Área da Figura 2 é: ");
          c.imprimeArea( );
     }
}
```

Exemplo

```
public class Circulo extends Figura {

        double raio;

        public Circulo (double raio) {
                this.raio = raio;
        }

        public double calcularArea( ) {
                double area = 0;
                area = 3.14 * raio * raio;
                return area;
        }
}
```

Interfaces

Interface de um Objeto/Classe

- Interface em Java é uma palavra-chave usada para definir uma coleção de definições de métodos e de valores de constantes.
- São semelhantes as clases abstratas, mas todos os métodos comportam-se como abstratos.
- Os métodos são qualificados como public por default.
- Não definem atributos comuns
  - Só definem constantes, ("atributos" qualificados como `public`, `static` e `final`).
- `Não definem construtores`
- `Não podem ser instanciadas.`

Interfaces

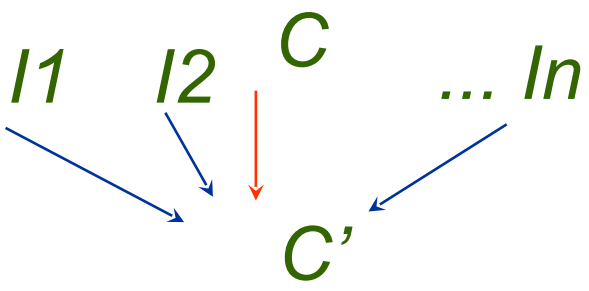
- Uma interface não pode ser instanciada (Não se pode criar objetos)
- Definem tipo de forma abstrata, apenas indicando a assinatura dos métodos
- Os métodos são implementados por classes e, para isso, é utilizada a palavra-chave *implements*.
- Mecanismo de projeto
  - podemos projetar sistemas utilizando interfaces
  - projetar serviços sem se preocupar com a sua implementação (abstração)

`implements`

- Classe que implementa uma interface deve *definir* os métodos da interface:
  - classes concretas têm que implementar os métodos
  - classes abstratas podem simplesmente conter métodos abstratos correspondentes aos métodos da interface

Definição de Classes: Forma Geral

```
class C'
   extends C
   implements I1, I2, ..., In {
      /* ... */
   }
```



Exemplo de Uso de Interface

```
public interface Figura {
   public double calcularArea( );
}

public class Quadrado implements Figura {
    double lado;
    public Quadrado(double lado) {
      this.lado = lado;
    }

    public double calcularArea( ) {
       double area = 0;
       area = lado * lado;
       return area;
    }
}
```

Como a classe Quadrado implementa a interface Figura, então, o método calcularArea( ) deve obrigatoriamente ser implementado.

Exemplo de Uso de Interface

```
public class Circulo implements Figura {

    double raio;

    public Circulo (double raio) {
        this.raio = raio;
    }

    public double calcularArea( ) {
        double area = 0;
        area = 3.14 * raio * raio;
        return area;
    }
}
```

Exemplo de Uso de Interface

```
public class Main {

    public static void main(String[] args) {

        Figura f1 = new Quadrado(4);

        Figura f2 = new Circulo(2);

       System.out.println("Área da Figura 1 é: "
                    + f1.calcularArea( ) + "\n"
                   + "Área da Figura 2 é: "
                    + f2.calcularArea( ));
       }
}
```

Observe que uma interface não pode ser instanciada mas é possível um objeto, declarado como sendo do tipo definido por uma interface, receber objetos de classes que implementam tal interface.

Multiplos supertipos

- Java permite somente a herança simples (uma única classe) com a palavra extends para herdar de uma classe (classe concreta ou abstrata).
- Interfaces são, portanto, um mecanismo simplificado de implementação de herança múltipla em Java, que possibilita que mais de uma interface determine os métodos que uma classe herdeira deve implementar.

```
classe Base extends Derivada implements
    interface1, interface2, ...
```

- Uma classe pode implementar mais de uma interface (contraste com classes abstratas...)

Interfaces e Sub-interfaces

- Uma interface também pode implementar outras interfaces (não classes).
- Assim como uma classe B pode estender outra classe A, uma interface I2 pode estender outra interface I1. Desse modo, quando uma classe C implementar I2, terá também obrigatoriamente que implementar os métodos definidos na interface I1.

```
interface I
  extends I1, I2, ..., In {
  /*... assinaturas de novos
        métodos ...
  */
}
```

Implementação de Múltiplas Interfaces

Arquivo: Printable.java

```
public interface Printable {
      public byte[ ] getBytesToPrint( );
}
```

Arquivo: Printer.java

```
public interface Printer {
      public void print(Printable p);
}
```

Arquivo: SendFax.java

```
public interface SendFax {
      public void transmit(Printable p);
}

public class FaxAndPrinter implements Printer, SendFax {
      ...
}
```

Uma classe pode implementar mais de uma interface, assumindo assim vários comportamentos.

O que usar? Quando?

Classes (abstratas)

- Agrupa objetos com implementações compartilhadas
- Define novas classes através de herança simples (herda de uma única classe abstrata ou não)
- Só uma pode ser supertipo de outra classe
- Podem conter métodos não-abstratos (com implementação)

Interfaces

- Agrupa objetos com implementações diferentes
- Define novas interfaces através de herança **múltipla** (implementa várias interfaces)
- Várias podem ser supertipo do mesmo tipo.
- Até java 7, não podiam conter métodos com implementação. Desde Java 8, se permite "Default Methods"

```
public interface oldInterface {
   public void existingMethod();
      default public void newDefaultMethod() {
      System.out.println("New default method"
           " is added in interface");
   }
}
```

Bibliografia

- Programação orientada a objetos com Java.
  Autores David J. Barnes e Michael Kolling.
- Java: Como programar.
  Autores: H. M. Deitel e P. J. Deitel
  Editora: Pearson – 8a Edição
- Nota: Alguns exemplos desta apresentação foram extraídos das fontes aqui apresentadas.