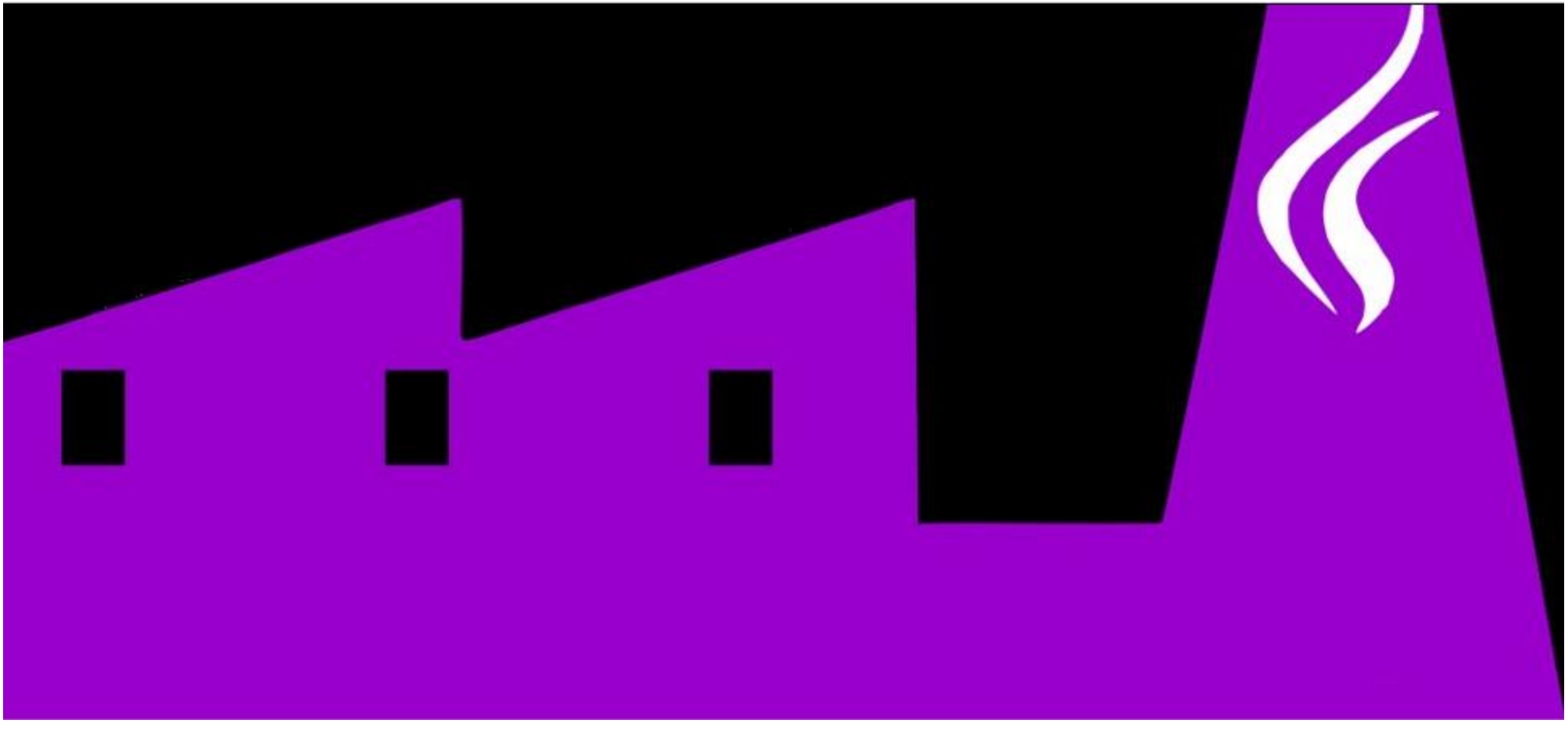
Fábrica de Software

Prof[es.] Ivan L. Süptitz e Evandro Franzen

Classes e métodos abstratos; Interfaces



Objetivo da aula

Compreender a diferença entre métodos concretos e **abstratos**. Entender porque se desenvolve **classes abstratas** e evoluir esse entendimento até as **Interfaces**

Métodos concretos X abstratos

```
/*é concreto pois tem uma implementação (um corpo que realmente faz algo)*/
public int fazerA(int p1, int p2)
{
     return p1 + p2*p2/100;
}


/*é abstrato pois tem somente a assinatura (sem um corpo que faz alguma coisa)*/
public int fazerB(int p1, int p2);


//Se não faz nada, então para o que serve?
//Serve para criar classes abstratas
```

Classes abstratas

O pol morf smo perm te que classes abstratas cons gam receber comportamentos atraves de classes concretas.

Por exemplo, um d spos t vo USB, podemos cons derar que o USB ser a uma classe abstrata enquanto os d spos t vos (Pen Dr ve, Ipad, Cameras, etc) ser am as classes concretas.

Ou se a, o USB e uma **espec f cacao** que pode ter var as mplementacoes com caracter st cas d ferentes.

Classes abstratas

Pode-se dizer que as classes abstratas servem como “modelo” para outras classes que dela herdem, **não podendo ser instanciada por si só**.

Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada herdando dela e então instanciar essa nova classe.

Os métodos da classe abstrata devem então serem sobrescritos nas classes filhas.

Classes abstratas

```
public abstract class Conta {

    private double saldo;

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public double getSaldo() {
        return saldo;
    }

    public abstract void imprimeExtrato();
}
```

Herança de classes abstratas

Na classe abstrata “Conta” os métodos que são abstratos têm um comportamento diferente, por isso não possuem corpo.

Ou seja, as subclasses que estão herdando precisam desse método mas não de forma genérica, aonde permite inserir as particularidades de cada subclasse.

Herança de classes abstratas

```
public class Poupanca extends Conta {

    @Override
    public void imprimeExtrato() {
        System.out.println("### Extrato da Conta ###");

        SimpleDateFormat sdf =
                new SimpleDateFormat("dd/MM/aaaa HH:mm:ss");
        Date date = new Date();

        System.out.println("Saldo: " + this.getSaldo());
        System.out.println("Data: " + sdf.format(date));

    }
}
```

Herança de classes abstratas

Usamos a palavra chave abstract para impedir que ela possa ser instanciada.

Conta cp = new Conta(); //não compila

Esse é o efeito direto de se usar o modificador abstract na declaração de uma classe:

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

    Conta cp = new Poupanca();
    cp.setSaldo(2121);
    cp.imprimeExtrato();

}
```

Ou seja, dá mais segurança e consistência ao sistema.

Interfaces

- As vezes chegamos ao ponto de escrever classes que tem todos os métodos definidos como abstract.
  - Em vez de fazer isso, criamos uma **interface**.

- As interfaces servem para especificar padrões que devem ser seguidos. Como um **contrato**.

- Esse “contrato” define um determinado conjunto de métodos que serão implementados nas classes que assinarem esse contrato.

- Uma **interface é 100% abstrata**, ou seja, os seus métodos são definidos como abstract, e as variáveis por padrão são sempre constantes (static final).

- Uma interface é definida através da palavra reservada “**interface**”. Para uma classe implementar uma interface é usada a palavra “**implements**”.

Interfaces

Como a linguagem Java não tem herança múltipla, as interfaces ajudam nessa questão.

Uma classe pode ser herdada apenas uma vez, mas pode implementar inúmeras interfaces.

As classes que forem implementar uma interface terão de adicionar todos os métodos da interface ou se transformar em uma classe abstrata

```
public interface Conta2 {

    void depositar(double valor);

    void sacar(double valor);

    double getSaldo();
}
```

Interface

```java
public class ContaCorrente implements Conta2 {

    private double saldo;
    private double taxaOperacao = 0.45;

    @Override
    public void depositar(double valor) {
        this.saldo += valor - taxaOperacao;
    }

    @Override
    public double getSaldo() {
        return this.saldo;
    }

    @Override
    public void sacar(double valor) {
        this.saldo -= valor + taxaOperacao;
    }

}
```

```java
public class ContaPoupanca implements Conta2 {

    private double saldo;

    @Override
    public void depositar(double valor) {
        this.saldo += valor;
    }

    @Override
    public double getSaldo() {
        return this.saldo;
    }

    @Override
    public void sacar(double valor) {
        this.saldo -= valor;

    }

}
```

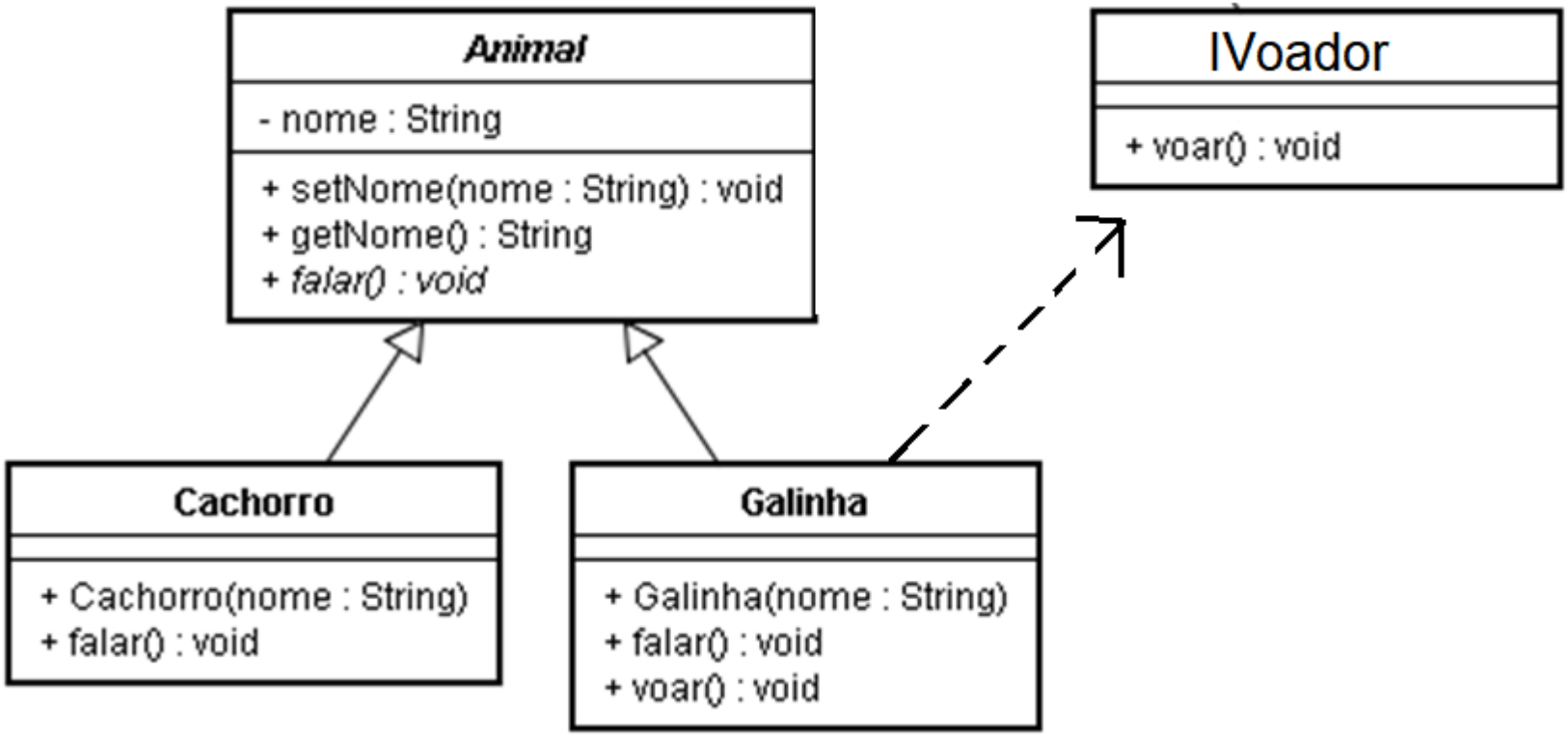
```java
public class GeradorExtratos {

    public void geradorConta(Conta2 conta) {
        System.out.println("Saldo Atual: " + conta.getSaldo());
    }
}
```

Interfaces

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

    ContaCorrente cc = new ContaCorrente();
    cc.depositar(1200.20);
    cc.sacar(300);

    ContaPoupanca cp = new ContaPoupanca();
    cp.depositar(500.50);
    cp.sacar(25);

    GeradorExtratos gec = new GeradorExtratos();

    gec.geradorConta(cc);

    gec.geradorConta(cp);
}
```

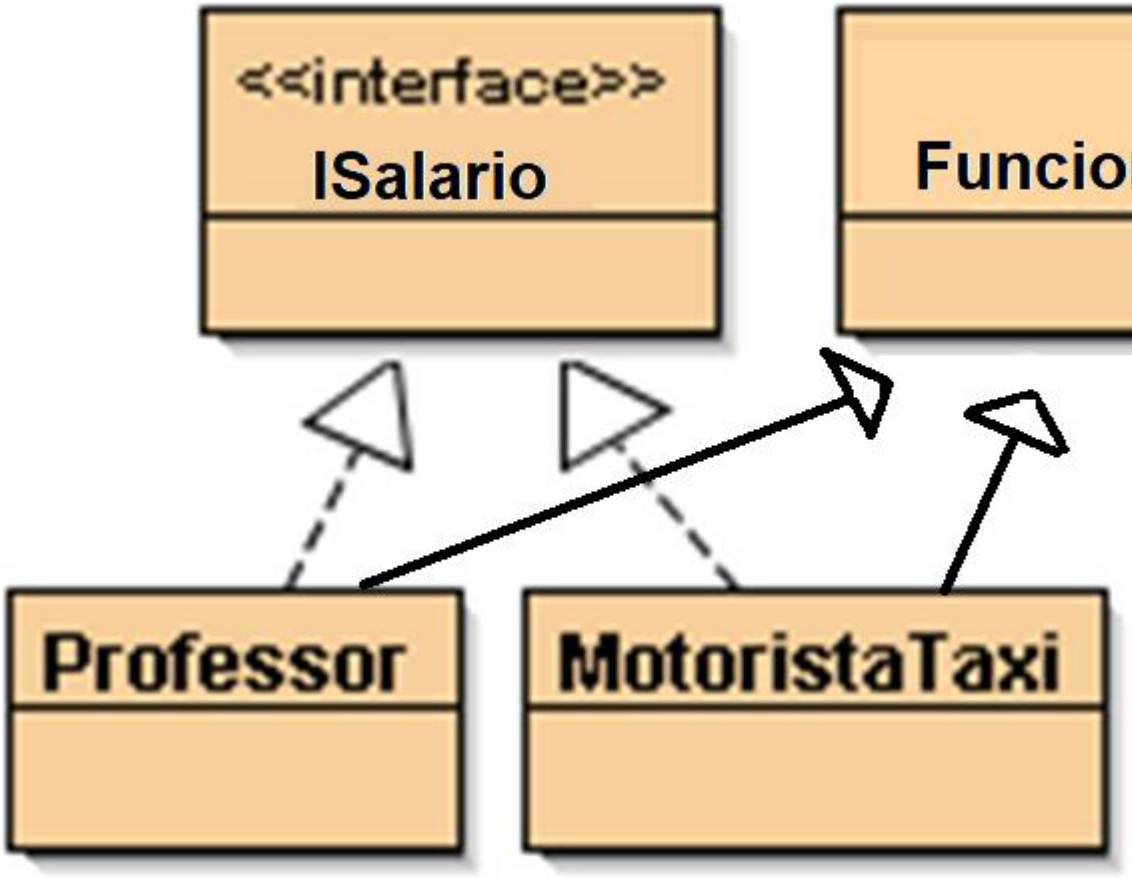
Exercícios

8.1. Simule o esquema abaixo:
Na classe principal, gerar uma lista de Animais aleatoriamente e passar para um método chamado testadorPolimorfico() que deve chamar todos os métodos do objeto passado.



Exercícios de Interface

8.2. Crie a estrutura abaixo. Tanto o Professor como o Motorista devem herdar da classe Funcionario, a qual tem como atributos: nome, valor por hora e dias trabalhados.




```
/** Interface Salario */
public interface ISalario {
    int SAL_MINIMO = 1320;
    float getSalarioLiquido();
    float getQtdSalMinimos();
}
```

Exercício 8.2 (continuação)

- Na classe Funcionario criar o método concreto calculaSalarioBruto() que retorna o valor da hora multiplicado pela carga horária multiplicado pelo número de dias. Para isso ele precisa implementar uma interface chamada IConstantes, a qual possui como constante a carga horaria = 8.8 horas por dia
- Veja que a classe Professor, estende Funcionario e implementa ISalario.
- O salário líquido do professor é o salário bruto – 20%. O salário líquido do motorista tem desconto de 15%.
- OBS: uma classe pode implementar **várias** interfaces simultaneamente

```
class Professor
    extends Funcionario
    implements ISalario,
               IConstantes {}
```

- Esta é uma diferença fundamental em Java: apenas **herança simples**, mas implementação de **múltiplas interfaces**
- **Nesse exemplo, não seria necessário, mas vamos testar o conceito**