Capacitação técnica e pedagógica para professores de informática

Analise e desenvolvimento de aplicações orientadas a objeto com Java SE

1º Módulo

Objetivo

- Capacitar docentes do Centro Paula Souza a ministrarem as disciplinas DSII e DSIII
- Conteúdo:
  - Desenvolvimento de softwares orientado a objetos
  - Linguagem de apoio: Java SE (JDK 6 Update 13)
  - IDE: Eclipse 3.4.2
  - Sistema operacional: Microsoft Windows XP/Vista

Classes abstratas e concretas

- Uma classe abstrata é desenvolvida para representar entidades e conceitos abstratos.
- A classe abstrata é sempre uma superclasse que não possui instâncias.
- Ela define um modelo para uma funcionalidade e fornece uma implementação incompleta (a parte genérica dessa funcionalidade) que é compartilhada por um grupo de classes derivadas (subclasses).
- Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.

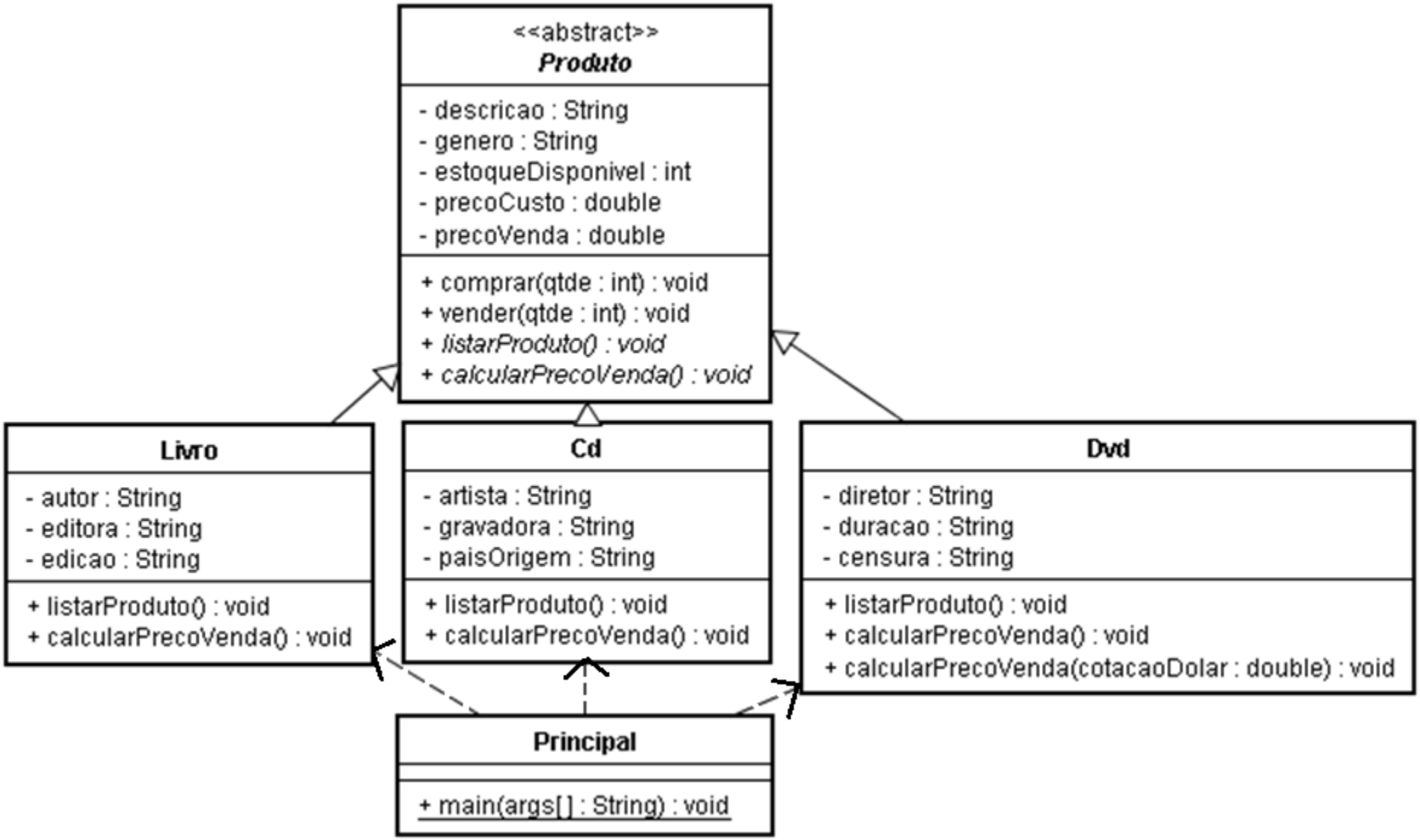
Métodos abstratos

- Uma classe abstrata pode conter métodos concretos, porém, um método abstrato só pode ser definido em uma classe abstrata.
- Esses métodos são implementados nas suas subclasses (concretas) com o objetivo de definir um comportamento (regras) específico.
- Um método abstrato define apenas a assinatura do método e, portanto, não contém código.

Métodos abstratos

- No exemplo da livraria, a classe Produto nunca será utilizada para instanciar um objeto porque os produtos efetivamente comercializados são livros, cds e dvds.
- A finalidade da classe Produto é somente a generalização dos produtos, portanto, conceitualmente ela deve ser definida como uma classe abstrata.
- Partindo desse princípio, o método calcularPrecoVenda também é um forte candidato a ser um método abstrato, porque, nesse exemplo, cada produto tem sua própria regra de calculo.

Projeto: Livraria



superclasse: Produto

```
abstract class Produto {

    abstract void calcularPrecoVenda();

    abstract void listarProduto();
```

subclasse: Dvd

```
public void calcularPrecoVenda(){
    // Como o método é definido como abstrato na superclasse, a inclusao da assinatura
    // do metodo é obrigatoria, mesmo que nao for utilizada
}
```

```
public void calcularPrecoVenda(double cotacaoDolar){
    // Apresenta o estoque atual e o estoque ja reajustado
    JOptionPane.showMessageDialog(null, "Preco venda em Reais:" + this.getPrecoVenda() + "\nCotacao do dolar: "
                + cotacaoDolar + "\nPreco de venda em Dolar:" + (this.getPrecoVenda() / cotacaoDolar));
}
```

Polimorfismo

- O termo polimorfismo é originário do grego e significa "muitas formas" (poli = muitas, morphos = formas).
- O polimorfismo permite que objetos de diferentes subclasses sejam tratados como objetos de uma única superclasse.
- Polimorfismo é a capacidade de um objeto poder ser referenciado de várias formas.
- É a possibilidade de manipular um objeto como **sendo** outro.
- O polimorfismo não quer dizer que o objeto se transforma em outro. Um objeto sempre será do tipo que foi instanciado o que pode mudar é a maneira como nos referimos a ele.

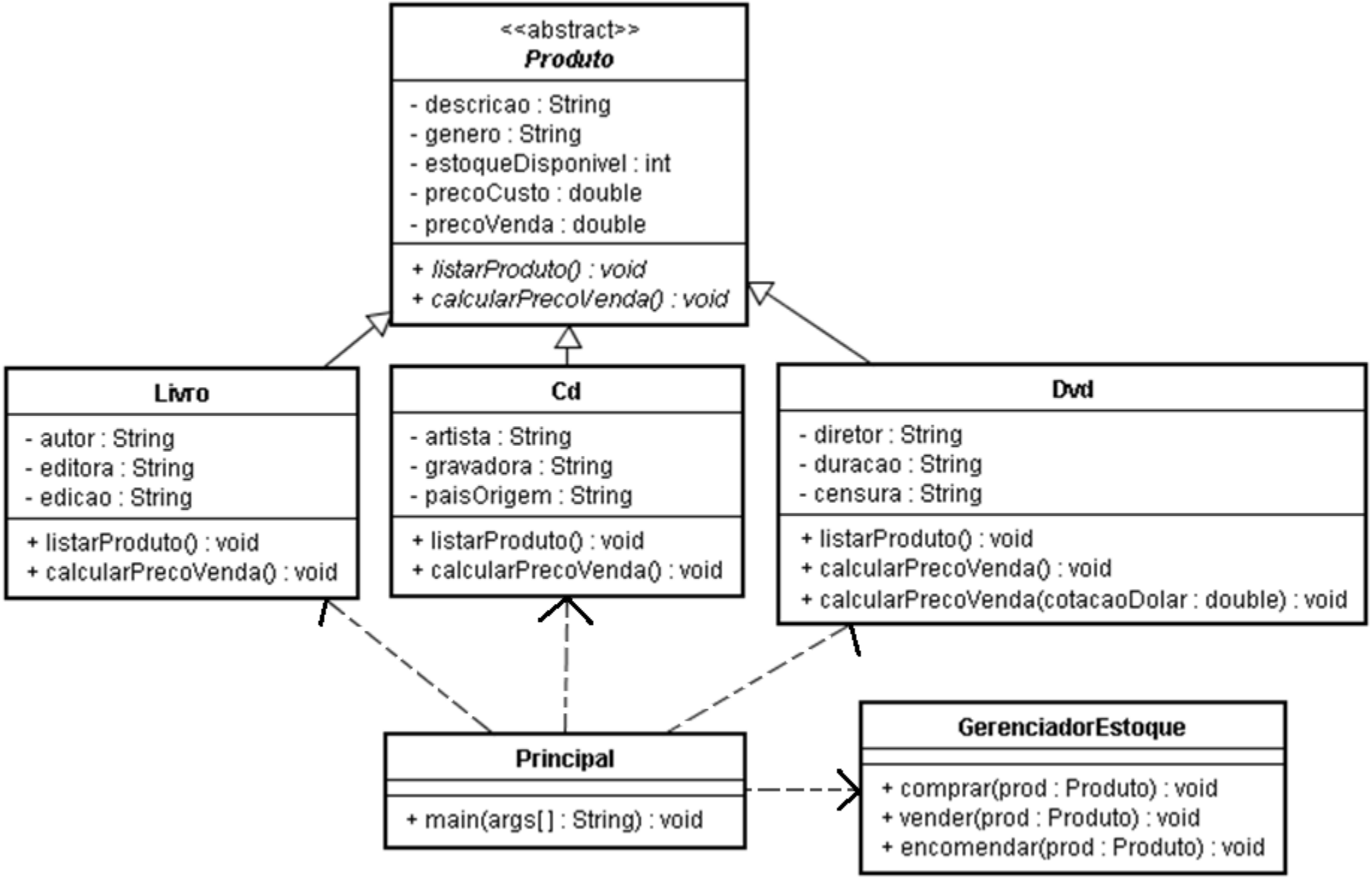
Polimorfismo

- Voltemos ao nosso projeto Livraria.
- Os produtos efetivamente comercializados, e que devem ser gerenciados, são livro, cd e dvd.
- Contudo, a afirmação: “Livro, cd e dvd **são** produtos” esta correta.
- O conceito de polimorfismo possibilita que tratemos um livro, cd ou dvd **como um** produto (pois, afinal de contas, eles **são** produtos), mas sem perdermos as suas características específicas porque apesar de serem produtos eles **não deixam de ser** um livro, um cd ou um dvd (especializações).

Polimorfismo

- Partindo desse raciocínio podemos definir métodos que reconheçam objetos através de suas superclasses mas que mantenham seus atributos e métodos implementados nas subclasses de origem.

Projeto: Livraria



classe: GerenciadorEstoque

```
// Recebe um objeto do tipo Produto por parametro
public void comprar(Produto prod){
    int quantidade = Integer.parseInt(JOptionPane.showInputDialog("Digite a quantidade comprada:"));
    // Apresenta o estoque atual e o estoque ja reajustado
    JOptionPane.showMessageDialog(null, "Estoque anterior de " + prod.getClass().getName() + ": " +
                            prod.getEstoqueDisponivel() + "\nQuantidade comprada:" + quantidade +
                              "\nEstoque atual:" + (prod.getEstoqueDisponivel() + quantidade));
    // Atribui o resultado da soma ao atributo estoqueDisponivel de prod
    prod.setEstoqueDisponivel(prod.getEstoqueDisponivel() + quantidade);
}

// Recebe um objeto do tipo Produto por parametro
public void vender(Produto prod){
    int quantidade = Integer.parseInt(JOptionPane.showInputDialog("Digite a quantidade vendida:"));
    // Apresenta o estoque atual e o estoque ja reajustado
    JOptionPane.showMessageDialog(null, "Estoque anterior " + prod.getClass().getName() + ": " +
                           prod.getEstoqueDisponivel() + "\nQuantidade vendida:" + quantidade +
                            "\nEstoque atual:" + (prod.getEstoqueDisponivel() - quantidade));
    // Atribui o resultado da subtracao do atributo estoqueDisponivel de prod
    prod.setEstoqueDisponivel(prod.getEstoqueDisponivel() - quantidade);
}
```

classe: GerenciadorEstoque

```
// Recebe um objeto do tipo Produto por parametro
public void encomendar(Produto prod){
    // Lê a quantidade desejada
    int qtde = Integer.parseInt(JOptionPane.showInputDialog("Digite a quantidade de " +
                                        prod.getClass().getName() + " desejada: "));
    // Verifica se há estoque disponivel para a encomenda do objeto desejado
    if(qtde <= prod.getEstoqueDisponivel()){
        JOptionPane.showMessageDialog(null, "Encomenda do " + prod.getClass().getName() + " - " +
                                        prod.getDescricao() + " realizada com pronta entrega!");
    }else{
        JOptionPane.showMessageDialog(null, "Encomenda do " + prod.getClass().getName() + " - " +
                        prod.getDescricao() + " em analise, realizando pedido com fornecedores!");
    }
}
```

classe: Principal

```
case 1: // Livro

    // Desvio condicional para definicao da operacao escolhida
    if(opOperacao == 1){ // Consultar
        l1.listarProduto();
    }

    if(opOperacao == 2){ // Comprar
        // Realiza chamada ao metodo comprar e passa o objeto l1 por parametro
        controle.comprar(l1);
    }

    if(opOperacao == 3){ // Vender
        // Realiza chamada ao metodo vender e passa o objeto l1 por parametro
        controle.vender(l1);
    }

    if(opOperacao == 4){ // Reajuste
        l1.calcularPrecoVenda();
    }

    if(opOperacao == 5){ // Encomenda de livros
        // Realiza chamada ao metodo encomendar e passa o objeto l1 por parametro
        controle.encomendar(l1);
    }

    break;
```

classe: Principal

```
case 2: // CD

    if(opOperacao == 1){ // Consultar
        c1.listarProduto();
    }

    if(opOperacao == 2){ // Comprar
        // Realiza chamada ao metodo comprar e passa o objeto c1 por parametro
        controle.comprar(c1);
    }

    if(opOperacao == 3){ // Vender
        // Realiza chamada ao metodo vender e passa o objeto c1 por parametro
        controle.vender(c1);
    }

    if(opOperacao == 4){ // Reajuste
        c1.calcularPrecoVenda();
    }

    if(opOperacao == 5){ // Encomenda de livros
        // Realiza chamada ao metodo encomendar e passa o objeto c1 por parametro
        controle.encomendar(c1);
    }

    break;
```

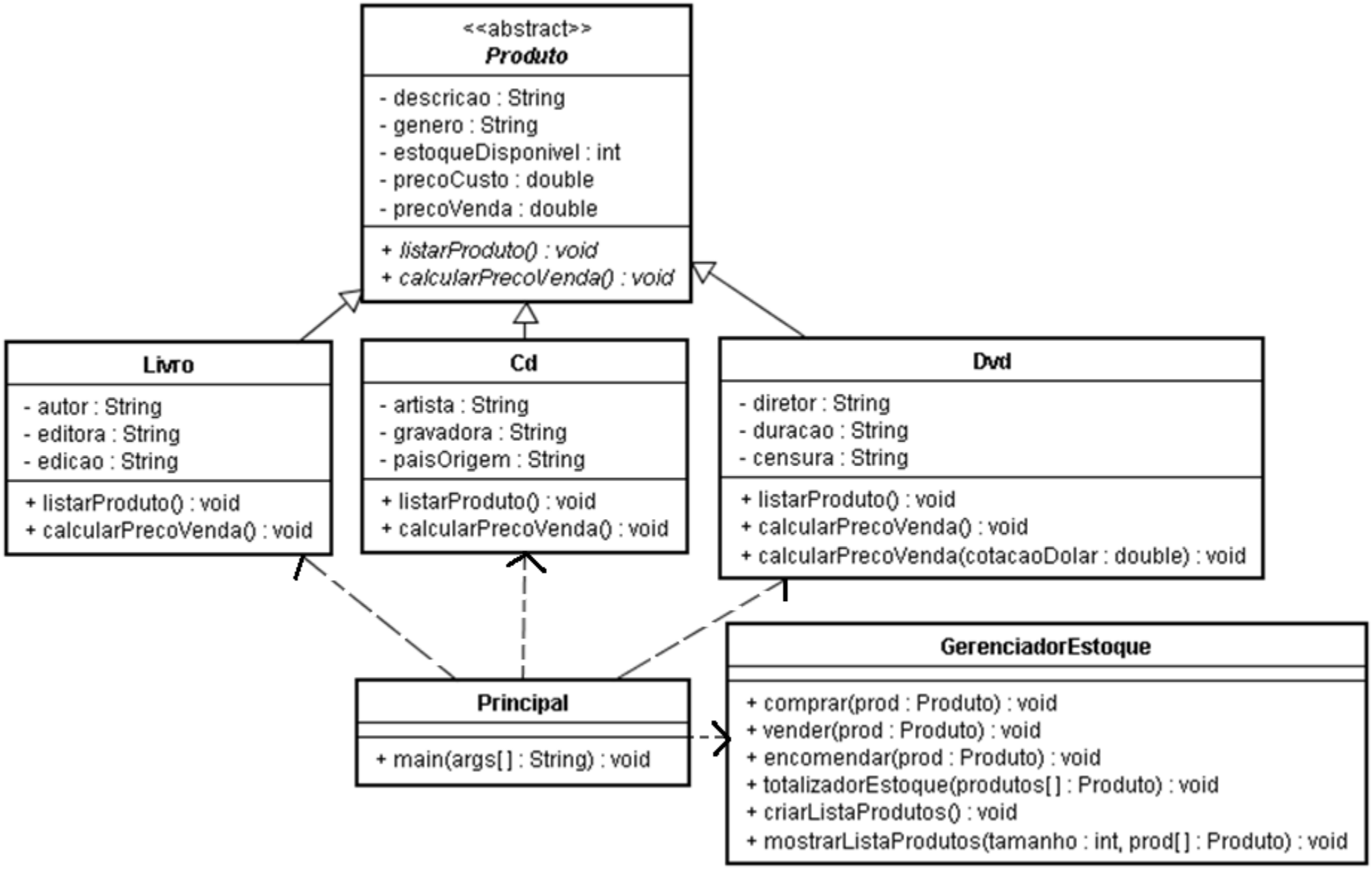
classe: Principal

```
case 3: // DVD

    if(opOperacao == 1){ // Consultar
        d1.listarProduto();
    }

    if(opOperacao == 2){ // Comprar
        // Realiza chamada ao metodo comprar e passa o objeto d1 por parametro
        controle.comprar(d1);
    }

    if(opOperacao == 3){ // Vender
        // Realiza chamada ao metodo vender e passa o objeto d1 por parametro
        controle.vender(d1);
    }

    if(opOperacao == 4){ // Reajuste
        double dolar = Double.parseDouble(JOptionPane.showInputDialog("Digite a cotacao do dolar:"));
        d1.calcularPrecoVenda(dolar);
    }

    if(opOperacao == 5){ // Encomenda de livros
        // Realiza chamada ao metodo encomendar e passa o objeto d1 por parametro
        controle.encomendar(d1);
    }

    break;
```

Arrays polimórficos

- O conceito de polimorfismo pode ser utilizado também na manipulação de arrays de objetos.
- Podemos armazenar objetos em arrays de tipos referenciados pela superclasse e armazenar objetos especializados (instanciados a partir de subclasses) tendo acesso sua estrutura de origem.

Projeto: Livraria



classe: GerenciadorEstoque

```
// Metodo implementado para testar passagem de array por parametro
public void totalizadorEstoque(Produto produtos[]){
    // Zera o valor total de estoque (para nao acumular com a totalizacao anterior)
    this.setValorTotalEstoque(0);
    // Percorre o array recebido por parametro do indice zero até seu tamanho total
    for (int i = 0; i <= produtos.length - 1; i++) {
        // Apresenta o produto e o preco de custo analizado
        JOptionPane.showMessageDialog(null, "Produto: " + produtos[i].getClass().getName() +
                    "\nPreço de custo: " + produtos[i].getPrecoCusto() + "\nQuantidade: " +
                                                produtos[i].getEstoqueDisponivel());
        // Soma em valorTotalEstoque
        this.setValorTotalEstoque(this.valorTotalEstoque += ( produtos[i].getPrecoCusto() *
                                                    produtos[i].getEstoqueDisponivel() ));
    }
    // Apresenta o total
    JOptionPane.showMessageDialog(null, "Total em estoque: " + this.getValorTotalEstoque());
}
```

classe: Principal

```
case 4: // Total em estoque

    // Declaracao de um array do tipo Produto
    Produto prod[] = new Produto[3];
    // Atribuicao dos objetos no array
    prod[0] = l1; // Objeto do tipo Livro
    prod[1] = c1; // Objeto do tipo Cd
    prod[2] = d1; // Objeto do tipo Dvd
    // Chamada do metodo e passagem do array por parametro
    gerenciador.totalizadorEstoque(prod);

    break;
```

classe: GerenciadorEstoque

```
public void criarListaProdutos(){
    // Declaracao e inicializacao de variaveis de controle de menu e indice do array
    int opProd = 0, indice = 0;
    // Declara e inicializa como nulo um objeto (uma referencia) do tipo Produto
    Produto prod = null;
    // Intancia um array do tipo Produto de 10 posicoes chamado produtos
    Produto produtos[] = new Produto[10];

    do{
        // Apresenta menu de produtos
        opProd = Integer.parseInt(JOptionPane.showInputDialog("Digite o produto desejado: " +
                        "\n1 - Livro \n2 - CD \n3 - DVD \n0 - Listar produtos cadastrados"));

        // Verifica o produto escolhido, instancia um objeto de acordo com a opcao
        // e atribui sua referencia ao objeto prod
        if(opProd == 1){
            prod = new Livro();
        }
        if(opProd == 2){
            prod = new Cd();
        }
        if(opProd == 3){
            prod = new Dvd();
        }
```

classe: GerenciadorEstoque

```
        // Se não foi escolhida a opcao Listar produtos cadastrados
        if(opProd !=0){
            // Lê a descricao o estoque disponivel e o preco de custo do objeto instanciado
            prod.setDescricao(JOptionPane.showInputDialog("Digite a descrição: "));
            prod.setEstoqueDisponivel(Integer.parseInt(JOptionPane.showInputDialog("Digite o estoque disponivel:
            prod.setPrecoCusto(Double.parseDouble(JOptionPane.showInputDialog("Digite o preço de custo: ")));
            // Atribui o objeto instanciado no array produtos
            produtos[indice] = prod;
            // Incrementa do indice
            indice++;
            // Destroi o objeto (limpa a referencia )
            prod = null;
        }
    // Enquanto a opcao Lista produtos cadastrados não for escolhida
    // repete as operacoes do looping
    }while(opProd != 0);

    // Apresenta os produtos armazenados no array produtos utilizando o metodo mostrarListaProdutos
    JOptionPane.showMessageDialog(null, mostrarListaProdutos(indice,produtos) + "\n");

}
```

classe: GerenciadorEstoque

```
// Metodo privado para leitura do array produtos
// recebe a quantidade de itens lidos (tamanho) e o array produtos
private String mostrarListaProdutos(int tamanho,Produto[] prod){
    // Monta uma string comecando com um cabecalho
    String relacao="Relação de Produtos Cadastrados\n";
    // e concatenando cada um dos produtos armazenados no array
    for(int i = 0;i < tamanho; i++){
        relacao = relacao + "\n" + String.valueOf(i) + " - Tipo: " + prod[i].getClass().getName() +
        " - Descricao: " + prod[i].getDescricao() + " - Estoque: " + prod[i].getEstoqueDisponivel() +
        " - Preco custo: " + prod[i].getPrecoCusto();
    }
    // Retorna uma string com todos os produtos contidos no array
    return relacao;
}
```

classe: Principal

```
case 5: // Criar relacao de produtos
    gerenciador.criarListaProdutos();
break;
```