33. Implemente em Java um sistema para controle de entrada de pessoas em um prédio. Cada pessoa apresenta um documento na portaria. Um documento pode ser uma carteira de identidade, um cpf ou um título de eleitor. O sistema é formado pelas seguintes classes:

Classe: **Documento**

- Método abstrato para obter o documento. As informações sobre o documento devem ser retornadas como uma única string.

Classe: **DocumentoEstadual** (subclasse de Documento)

- Classe abstrata sem atributos e métodos.

Classe: **DocumentoFederal** (subclasse de Documento)

- Classe abstrata sem atributos e métodos.

Classe: **Identidade** (subclasse de DocumentoEstadual)

- Atributos: numero (tipo String) e estado (tipo String). Todos atributos privados;
- Método construtor para inicializar os atributos com valores passados por parâmetros;
- Método para obter o documento. Concretização do método abstrato de Documento. Retorna a identidade (numero e estado) como uma única string.

Classe: **Cpf** (subclasse de DocumentoFederal)

- Atributo: numero (tipo String). Atributo privado;
- Método construtor para inicializar o atributo com valor passado por parâmetro;
- Método para obter o documento. Concretização do método abstrato de Documento. Retorna o número do cpf como uma string.

Classe: **TituloEleitor** (subclasse de DocumentoFederal)

- Atributos: numero (tipo String), zona (tipo String) e seção (tipo String). Todos atributos privados;
- Método construtor para inicializar os atributos com valores passados por parâmetros;
- Método para obter o documento. Concretização do método abstrato de Documento. Retorna o título de eleitor (numero, zona e seção) como uma única string.

Classe: **Pessoa**

- Atributos: nome (tipo String) e documento (tipo Documento);
- Método construtor para inicializar os atributos. Parâmetros: nome e documento;
- Método get para obter o nome e o documento (em forma de String);

Classe: **Portaria**

- Método main para executar as seguintes ações:

- Criar uma lista, usando ArrayList, e inserir três pessoas, cada uma com um tipo de documento diferente (identidade, cpf e título de eleitor);
- Imprimir a relação das pessoas (nome e documento) que passaram pela portaria.

34. Considere o sistema de uma empresa com vários funcionários. Os funcionários ou são horistas ou são comissionados, não existe outro tipo de funcionário. As únicas informações necessárias no sistema são o nome do funcionário e os dados para cálculo de seu salário. O salário mensal de um funcionário horista é o valor da sua hora de trabalho vezes o número de horas trabalhadas por ele no mês. O salário mensal de um funcionário comissionado é o seu percentual de comissão vezes o valor de suas vendas no mês. A empresa mantém uma lista de todos os seus funcionários e necessita de opções para adicionar e excluir funcionários nessa lista e para retornar a lista.

    Faça as seguintes ações, usando polimorfismo sempre que possível:

    (a) O diagrama de classes para esse sistema.

    (b) O código de todas as classes em Java.

    (c) Uma classe de teste com um trecho de código para criar uma empresa com dois funcionários (um horista e um comissionado) e imprimir o nome e o salário mensal de todos os funcionários, o número de horas trabalhadas pelos horistas e o valor das vendas dos comissionados (suponha que você não sabe quantos funcionários existem na empresa). Use os métodos das classes definidos no item (b).

35. Considere a seguinte descrição de um sistema de controle bancário:

    Uma agência bancária é caracterizada por um número de identificação, um nome e um endereço. Uma conta bancária pertence a uma agência, possui um número de identificação, um saldo e está ligada a um ou mais clientes. Uma conta bancária pode ser do tipo conta corrente ou do tipo conta poupança. Uma conta corrente possui um limite de crédito que o cliente pode usar além do valor do saldo em conta. Uma conta poupança possui um controle por dia base, permitindo saldos e rendimentos por dia base. Por exemplo, a conta poupança de número 102 possui um saldo de R$800,00 com rendimentos no dia 10 e um saldo de R$1500,00 com rendimentos no dia 15, totalizando um saldo final de R$2300,00. De cada conta (independente do tipo), o sistema controla todas as movimentações. Movimentações são depósito, saque, transferência, cobrança de tarifa, crédito de juros etc. De cada movimentação é importante identificar a data/hora de ocorrência, o tipo de movimentação, o valor movimentado e uma indicação de débito ou crédito. Dos clientes é importante identificar CPF ou CNPJ, nome e endereço. Um cliente pode ter mais de uma conta.

O sistema deve ter opções para cadastrar (inserir, alterar e excluir) agências, clientes e contas, incluir e excluir clientes de contas, efetuar depósitos, saques e transferências, creditar juros em conta poupança, debitar tarifas de contas, obter o saldo e obter o extrato de uma conta (relação de todas as movimentações em um determinado período). Na inserção, o número de uma conta deve ser gerado automaticamente (use um atributo estático para fazer esse controle). Todas as operações que efetuam crédito ou débito em conta devem gerar registros de movimentação, com data/hora obtida do sistema. Ao projetar as operações, pense que o ponto de partida deve ser sempre o banco ou a agência. Por exemplo, você solicita a agência para efetuar um depósito em uma conta.

Você deve implementar todas as classes do sistema em Java, mostrando todos os atributos e as assinaturas de todos os métodos. Implemente completamente somente os métodos necessários para inserir uma nova conta corrente com limite de crédito, excluir um cliente de um banco e efetuar um depósito em uma conta poupança. Todas as listas devem ser implementadas usando a classe ArrayList. Crie também uma classe Banco com um método *main* mostrando como inserir uma nova conta, excluir um cliente, efetuar um depósito em uma conta poupança e imprimir o extrato de uma conta.

Observações:

1. O único local onde você pode imprimir mensagens ao usuário é no método *main* da classe Banco;
2. Outros métodos ou atributos podem ser adicionados ao seu programa, desde que não descaracterizem a definição feita acima.

36. Analise o programa abaixo e mostre as mensagens que serão impressas se ele for executado.

```
/**
  Classe de excecao para divisao por zero.
*/
class DivideByZeroException extends ArithmeticException {

        public DivideByZeroException() {
                super("O denominador na divisao tem valor zero!");
        }
}

/**
  Classe de teste de excecao.
  O que o programa abaixo imprime?
*/
public class TestaExcecao {
        public static void divisao1 (int num, int den) {
                try {
                        int div = num / den;
                        System.out.println("Divisao = "+ div);
```

```
        }
        catch (ArithmeticException e) {
                System.out.println("Erro1 : Divisao por Zero!");
        }
        catch (Exception e) {
                System.out.println("Erro qualquer!");
        }
}

public static void divisao2 (int num, int den) {
        try {
                int div = num / den;
                System.out.println("Divisao = "+ div);
        }
        catch (ArithmeticException e) {
                System.out.println("Erro2: "+ e.getMessage());
        }
        System.out.println("Divisao Encerrada!");
}

public static void divisao3 (int num, int den) throws Exception {
        try {
                int div = num / den;
        }
        catch (ArithmeticException e) {
                throw new Exception("Outra Divisao por Zero");
        }
        System.out.println("Divisao Encerrada Novamente!");
}

public static void divisao4 (int num, int den) throws DivideByZeroException {
        try {
                int div = num / den;
        }
        catch (ArithmeticException e) {
                throw new DivideByZeroException();
        }
        finally {
                System.out.println("Divisao Finalizada!");
        }
        System.out.println("Mais uma Divisao Encerrada!");
}

public static void divisao5 (int num, int den) {
        int div = num / den;
        System.out.println("Divisao = "+ div);
}

public static void main (String[] args) {
        divisao1 (2, 0);
        divisao2 (2, 0);
        try {
                divisao3 (2, 0);
        }
        catch (ArithmeticException e) {
                System.out.println("Erro3a: " + e.getMessage());
```

```
            }
            catch (Exception e) {
                System.out.println("Erro3b: " + e.getMessage());
            }
            try {
                divisao4 (2, 0);
            }
            catch (DivideByZeroException e) {
                System.out.println("Erro4: "+ e.getMessage());
            }
            try {
                divisao5 (2, 0);
            }
            catch (ArithmeticException e) {
                System.out.println("Erro5: " + e.getMessage());
            }
            divisao5 (2, 0);
        }
}
```

37. Considere a classe Cliente mostrada abaixo:

```
public class Cliente {
    private String nome;  // nome do cliente
    private String cpf;     // número de CPF do cliente

    public Cliente (String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }
    public String getNome () { return nome;}
    public String getCpf() {return cpf;}
}
```

Para manter o cadastro de clientes de uma empresa, foi criada a seguinte classe:

```
public class Empresa {
    private static final int MAXCLI = 1000;
    private Cliente[] clientes = new Cliente[MAXCLI];
    private int numCli = 0;

    public void adicionarCliente (String nome, String cpf) {
        if (numCli == MAXCLI) return;
        clientes[numCli] = new Cliente (nome, cpf);
        numCli++;
    }

    // . . . outros métodos ...

    public static void main (String[] args) {
        Empresa emp = new Empresa();
        String nome, cpf;
        nome = Console.readLine ("Entre com o nome do cliente:");
        while (!nome.equals("FIM")) {
            cpf = Console.readLine ("Entre com o CPF do cliente:");
            emp.adicionarCliente (nome, cpf);
            nome = Console.readLine ("Entre com o nome do cliente:");
```

```
        }
    }
}
```

Esse programa apresenta pelo menos dois problemas. O primeiro é que se lista de clientes estiver cheia, nenhuma informação é retornada. O segundo, é que ele não valida o CPF do cliente, aceitando qualquer string. Não é necessário considerar o fato de se poder adicionar o mesmo cliente mais de uma vez!

Você deve corrigir esses problemas criando um esquema de tratamento de exceção. Para o primeiro caso, retorne a mensagem "Lista cheia" através da classe ***Exception***. Para o segundo caso, crie uma classe de exceção própria com a mensagem "CPF Inválido". Para validar o CPF, considere que o mesmo deve ser formado por 11 dígitos, sendo os dois últimos os dígitos verificadores. O resto da divisão da soma dos 9 primeiros dígitos dividido por 11 deve ser igual ao dígito verificador.

Reescreva o programa acima de forma a incluir o tratamento de exceção e a emitir as mensagens de erro para o usuário. As mensagens só podem ser emitidas no método main da classe Empresa.

Consulte a API Java.

API Java

Classe: **String**
Métodos:

public char charAt (int posicao)
retorna o caractere na posição especificada (entre *0* e *length()-1*).

public int length ()
retorna o tamanho (em quantidade de caracteres) da string.

Classe: **Integer**
Métodos:

public static int parseInt (String s) throws NumberFormatException
retorna um número inteiro correspondente ao valor passado pela string s.
Levanta uma exceção se a string não corresponde a um número inteiro.

Classe: **Character**
Métodos:

public Character (char c)
constrói um objeto com conteúdo dado por c.

public String toString ()
retorna o caractere convertido para String.

38. Implemente um editor gráfico bidimensional simples. O editor deve permitir a criação de desenhos usando figuras geométricas básicas como retas, triângulos, retângulos e círculos. O programa deve ter uma estrutura de dados para armazenar as figuras e opção para desenhá-las em uma tela gráfica. Para isso, implemente as seguintes classes:

Classe: **FiguraGeometrica**

- Classe abstrata;
- Atributo cor (cor da figura, tipo Color). Atributo privado;
- Método abstrato para desenhar a figura geométrica na cor especificada;
- Métodos *get* e *set* para obter e definir a cor da figura.

Classe: **Reta** (subclasse de FiguraGeometrica)

- Atributos: p1 e p2 (ponto inicial e ponto final da reta, tipo *Point*). Todos atributos privados;
- Método construtor para inicializar os atributos com valores passados por parâmetros. Deve gerar uma exceção se os pontos não formarem reta;
- Método para desenhar a reta.

Classe: **Triângulo** (subclasse de FiguraGeometrica)

- Atributos: p1, p2 e p3 (pontos que formam o triângulo, tipo *Point*). Todos atributos privados;
- Método construtor para inicializar os atributos com valores passados por parâmetros. Deve gerar uma exceção se os pontos não formarem um triângulo;
- Método para desenhar o triângulo.

Classe: **Retângulo** (subclasse de FiguraGeometrica)

- Atributos: p1, p2, p3 e p4 (pontos que formam o retângulo, tipo *Point*). Todos atributos privados;
- Método construtor para inicializar os atributos com valores passados por parâmetros. Deve gerar uma exceção se os pontos não formarem retângulo;
- Método para desenhar o retângulo.

Classe: **Círculo** (subclasse de FiguraGeometrica)

- Atributos: p1 e p2 (ponto central e ponto sobre a circunferência que forma o círculo, tipo *Point*). Todos atributos privados;
- Método construtor para inicializar os atributos com valores passados por parâmetros. Deve gerar uma exceção se os pontos não formarem círculo;
- Método para desenhar o círculo.

Classe: **Desenho**

- Atributo: lista de figuras geométricas que formam o desenho (use a classe ArrayList). Atributo privado;
- Método para adicionar uma figura geométrica à lista que forma o desenho;
- Método para excluir uma figura geométrica da lista que forma o desenho;

- Método para desenhar as figuras que formam o desenho.

Classe: **Tela**

- Atributo: lista de desenhos (use a classe ArrayList). Atributo privado;
- Métodos para executar as seguintes operações:
  - Criar vários desenhos usando as diversas figuras geométricas, inserindo-os na lista de desenhos;
  - Exibir os desenhos da lista em uma tela gráfica;
  - Remover algumas figuras de alguns desenhos;
  - Redesenhar a tela gráfica.

Observações:

- Consulte a documentação java sobre as classes Point, ArrayList, Graphics e Color;
- O tratamento de exceção deve mostrar exemplos de uso de classes de exceção prontas (pré-definidas) e de criação de classes próprias;
- A classe Tela deve apresentar uma janela (classe JFrame) contendo um painel (JPanel) sobre o qual serão feitos os desenhos;

Esse projeto permite que você crie novas figuras geométricas apenas adicionando novas classes (exemplo: classes elípse, pentágono), sem a necessidade de mudar as classes existentes.

Você pode incrementá-lo para que o usuário escolha a figura a ser desenhada de uma paleta de componentes e, com o mouse, defina o desenho na tela.