1) (**Sobrecarga**) Escreva uma classe Ponto2D para representar um ponto no espaço cartesiano de duas dimensões. Esta classe deve fornecer dois construtores, sendo o primeiro um construtor padrão, para inicializar as coordenadas x e y do ponto em questão, e um outro, que permita criar pontos com coordenadas na origem. Esta classe deve ainda implementar os seguintes métodos:
Um método chamado distância,que recebe uma outra instância da classe Ponto2D e retorna um valor do tipo double correspondente à distância euclidiana entre o Ponto2D encapsulado e o passado como argumento.
Um método distância sobrecarregado, que não recebe nenhum argumento, mas calcula a distância euclidiana entre as coordenadas encapsuladas e a origem do sistema de coordenadas. Para isso, dentro do método, crie uma instância de Ponto2D correspondente à origem e passe-a como argumento para o método distância, implementado no item anterior. Crie uma classe teste, que permita testar as funcionalidades da classe definida.

Ponto2D.java

```
package exercicio01;
public class Ponto2D {
	private double x, y;
	public Ponto2D(double x1, double y1){
		x = x1;   y = y1;		}

	public Ponto2D(){
		x = 0.0;  y = 0.0;		}
	public double getX() {
		return x;		}
	public void setX(double x) {
		this.x = x;		}
	public double getY() {
		return y;		}
	public void setY(double y) {
		this.y = y;		}

	public double distanciaDoisPontos(Ponto2D ponto){
		double dist = Math.sqrt((Math.pow((ponto.getX()-x),2)) +
					(Math.pow((ponto.getY()-y),2)));
		return dist;			}

	public double distanciaDoisPontos(){
		double dist = Math.sqrt((Math.pow((x-0.0),2)) +
					(Math.pow((y-0.0),2)));
		return dist;				}

	public String toString(){
		String resultado= "Ponto= ("+x+","+y+")";
		return resultado;   }
}
```

Main.java

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

```
                Ponto2D ponto1 = new Ponto2D(10,20);
                Ponto2D ponto2 = new Ponto2D(50.2,30.7);
                Ponto2D ponto3 = new Ponto2D(10,20);

                System.out.println(ponto1.toString());
                System.out.println(ponto2.toString());
                System.out.println(ponto3.toString());

                double distancia = ponto1.distanciaDoisPontos(ponto2);
                System.out.println("Distancia entre os pontos 1 e 2: "+ distancia);

                distancia = ponto1.distanciaDoisPontos(ponto3);
                System.out.println("Distancia entre os pontos 1 e 3: "+ distancia);
        }
}
```

2) (**Sobrecarga**) Utilizando a classe definida no exercício anterior, crie uma classe Reta, para representar uma reta, unida por dois pontos no espaço cartesiano de duas dimensões. Implemente quatro construtores para esta classe: um sem argumentos que considere que a reta comece e termine no ponto (0,0); um que receba um argumento do tipo Ponto2D e que considere que a reta comece na origem e termine no ponto passado como argumento; um que receba duas instâncias da classe Ponto2D como argumentos e um que receba quatro valores de ponto flutuante, correspondentes às duas coordenadas. Crie uma classe teste, que permita testar as funcionalidades da classe Reta definida.

Reta.java

```
package exercicio02;
public class Reta {
        private Ponto2D p1, p2;

        public Reta() {
                p1 = new Ponto2D();
                p2 = new Ponto2D();    }

        public Reta(Ponto2D p2) {
                p1 = new Ponto2D();
                this.p2 = p2;     }

        public Reta(Ponto2D p1, Ponto2D p2) {
                this.p1 = p1;
                this.p2 = p2;     }

        public Reta(double x1, double y1, double x2, double y2) {
                p1 = new Ponto2D(x1, y1);
                p2 = new Ponto2D(x2, y2); }
```

```
        public String toString(){
                String resultado = p1.toString() + "  " + p2.toString();
                return resultado;       }
}
```

Main.java

```
package exercicio02;
public class Main {
        public static void main(String[] args) {

                Ponto2D ponto1 = new Ponto2D(10,20);
                Ponto2D ponto2 = new Ponto2D(50.2,30.7);
                Ponto2D ponto3 = new Ponto2D(10,20);

          Reta r1 = new Reta (ponto1, ponto2);
          Reta r2 = new Reta (ponto1);
           System.out.println(r1.toString());
           System.out.println(r2.toString());
        }

}
```

3) (**Sobrecarga**) Escreva três construtores para a classe NumeroComplexo. Um construtor deverá receber os dois valores (real e imaginário) como argumentos, o outro somente o valor real, considerando o imaginário como sendo zero, e o terceiro construtor não recebe argumentos, considerando as partes real e imaginária do número complexo como sendo iguais a zero. Implemente também uma classe teste, para testar os construtores definidos na classe NumeroComplexo.

NumeroComplexo.java

```
public class NumeroComplexo {
        private double real, imaginario;

        public NumeroComplexo(double r, double i){
                real = r;
                imaginario = i;   }

        public NumeroComplexo(double r){
                real = r;
                imaginario = 0.0;  }

        public NumeroComplexo(){
                real = 0.0;
                imaginario = 0.0;  }

        public String toString() {
                return real + " + " + imaginario + "i";
        }
}
```

Main.java

```
package exercicio03;
public class Main {
	public static void main(String[] args) {
		NumeroComplexo num1 = new NumeroComplexo(2.0, 3.0);
		NumeroComplexo num2 = new NumeroComplexo(2.0);
		NumeroComplexo num3 = new NumeroComplexo();
		System.out.println(num1.toString());
		System.out.println(num2.toString());
		System.out.println(num3.toString());
	}
}
```

4) (**Interface**) Escreva uma interface ObjetoGeometrico que representa um objeto geométrico. Essa interface deve ter métodos para mostrar os dados do objeto geométrico e para calcular e retornar sua área e perímetro. Usando essa interface como base, escreva as classes Circulo (contendo um raio), Retangulo (contendo dois valores para os lados) e Triangulo (contendo três valores para os lados).
Dicas: A área de um círculo pode ser calculada com Math.PI*r*r, onde r é o raio do círculo. O perímetro de um círculo é dado por 2*Math.PI*r. A área do retângulo é dada por b*h, onde b é um dos lados e h é o outro lado. Seu perímetro é dado por 2*b+2*h. A área de um triângulo é dada por Math.sqrt(s*(s-a)*(s-b)*(s-c)), onde Math.sqrt é a função que calcula a raiz quadrada, a, b e c são os lados do triângulo, e s é a metade do perímetro do triângulo. O perímetro do triângulo é calculado como (a+b+c).

FormasGeometricas.java

```
package exercicio04;
public interface FormasGeometricas {

  public abstract double area();
  public abstract double perimetro();
  public abstract String getNome();
}
```

Circulo.java

```
package exercicio04;
public class Circulo implements FormasGeometricas{
  private double raio;
  private double x, y; //centro do círculo

  public Circulo (double x, double y, double raio) {
	this.x = x; this.y = y; this.raio = raio;
  }
	@Override
	public double area() {
		return Math.PI * raio * raio;
	}
	@Override
	public double perimetro() {
```

```
                return 2*Math.PI*raio;
        }
        @Override
        public String getNome() {
                return "Circulo";
        }
        public String toString() {
                return "(" + x + "," + y + "), raio =" + raio;
        }
}
```

Retangulo.java

```
package exercicio04;
public class Retangulo implements FormasGeometricas{
  private double b, h; //lados

  public Retangulo (double b, double h) {
        this.b = b; this.h= h;
  }

        @Override
        public double area() {
                return b*h;
        }
        @Override
        public double perimetro() {
                return 2*b + 2*h;
        }
        @Override
        public String getNome() {
                return "Retangulo";
        }

        public String toString() {
          return "Lados:"+ b + ", " + h;
        }
}
```

Triangulo.java

```
package exercicio04;
public class Triangulo implements FormasGeometricas {
  private double a, b, c;

  public Triangulo(double a, double b, double c) {
        this.a = a; this.b = b; this.c = c;
  }

        @Override
        public double area() {
```

```
                double s = perimetro() /2;
                return Math.sqrt( s*(s-a)*(s-b)*(s-c) );
        }
        @Override
        public double perimetro() {
                return (a+b+c);
        }
        @Override
        public String getNome() {
                return "Triangulo";
        }
        public String toString() {
                return "Lados: " + a + "," + b + "," + c;
        }
}
```

Main.java

```
package exercicio04;
public class Main {
        public static void main(String[] args) {
                Circulo c = new Circulo(5,10,1);
                System.out.println(c.getNome());
                System.out.println(c.area());
                System.out.println(c.perimetro());

                Retangulo r = new Retangulo(5,10);
                System.out.println(r.getNome());
                System.out.println(r.area());
                System.out.println(r.perimetro());
        }
}
```

5) (**Herança**) Usando o exercício anterior, escreva a classe Quadrado, que herda da classe Retangulo, mas somente precisa inicializar um dos lados, e as classes TrianguloEquilatero, TrianguloIsosceles e TrianguloEscaleno, que precisam inicializar somente um, dois ou três lados do triângulo. Para cada uma dessas classes, quais métodos devem ser sobrepostos e quais podem ser aproveitados?

Quadrado.java

```
package exercicio04;
public class Quadrado extends Retangulo{

        public Quadrado(int a) {
                super(a,a);
        }
        //Sobreposição
        public String getNome() {
                return "Quadrado";
```

```
        }
}
```

TrianguloEquilatero.java

```
package exercicio04;
public class TrianguloEquilatero extends Triangulo{
        public TrianguloEquilatero(double a) {
                super(a, a, a);
        }
        //Sobreposição
        public String getNome() {
                return "Triangulo Equilatero";
        }
}
```

TrianguloIsoceles.java

```
package exercicio04;
public class TrianguloIsoceles extends Triangulo{
        public TrianguloIsoceles(double a, double b) {
                super(a, b, b);
        }

        //Sobreposição
                public String getNome() {
                        return "Triangulo Isoceles";
                }
}
```

TrianguloEscaleno.java

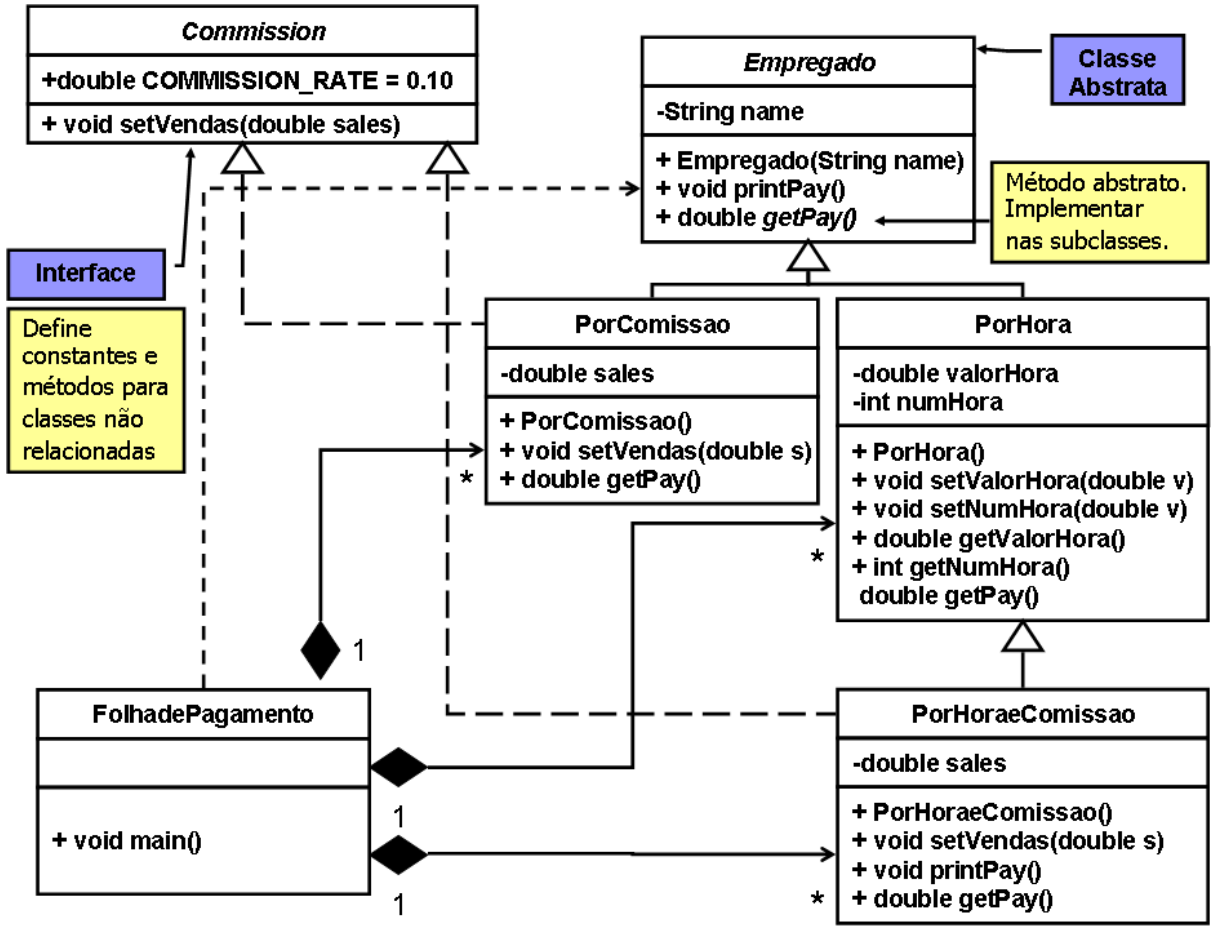
```
package exercicio04;
public class TrianguloEscaleno extends Triangulo {
        public TrianguloEscaleno(double a, double b, double c) {
                super(a, b, c);
        }
        //Sobreposição
                public String getNome() {
                        return "Triangulo Escaleno";
                }
}
```

Main.java

```
package exercicio04;
public class Main {
        public static void main(String[] args) {
                Circulo c = new Circulo(5,10,1);
                System.out.println(c.getNome());
                System.out.println(c.area());
                System.out.println(c.perimetro());

                Retangulo r = new Retangulo(5,10);
```

```
                System.out.println(r.getNome());
                System.out.println(r.area());
                System.out.println(r.perimetro());

                Quadrado q = new Quadrado(5);
                System.out.println(q.getNome());
                System.out.println(q.area());
                System.out.println(q.perimetro());

                TrianguloEquilatero ti = new TrianguloEquilatero(5);
                System.out.println(ti.getNome());
                System.out.println(ti.area());
                System.out.println(ti.perimetro());
        }
}
```

6)(**Classe Abstrata, Interface**) Dado o diagrama UML abaixo, construir um programa capaz de simular o funcionamento de folha de pagamento com quatro classes de trabalhadores: Empregado, PorHora, PorComissao e PorHoraeComissao. A classe Empregado deve ser abstrata, pois o método getPay(), que retorna o quanto cada tipo de empregado deve ganhar, só poderá ser definido nas subclasses. Desse modo, a classe Empregado deve ser declarada abstrata. Para todas as classes cujo ganho dos trabalhadores está relacionado com a comissão relativa ao montante de vendas (PorComissao e PorHoraeComissao), deve-se empregar o método setVendas e a informação contida no campo COMMISSION_RATE. Por último, a classe FolhadePagamento emprega objetos de todas as classes.



Empregado.java

```
package exercicio06;
public abstract class Empregado {
	private String nome;

	public Empregado(String nome) {
		this.nome = nome;
	}

	public void printPay() {

	}

	public abstract double getPay();
}
```

PorHora.java

```
package exercicio06;
public class PorHora extends Empregado{
	private double valorHora;
	private int numHora;

	public PorHora(String nome) {
```

```
                super(nome);
        }
        public PorHora(String nome, int numHora, double valorHora) {
                super(nome);
                this.numHora = numHora;
                this.valorHora = valorHora;
        }

        public double getValorHora() {
                return valorHora;
        }
        public void setValorHora(double valorHora) {
                this.valorHora = valorHora;
        }
        public int getNumHora() {
                return numHora;
        }
        public void setNumHora(int numHora) {
                this.numHora = numHora;
        }
        @Override
        public double getPay() {
                return numHora*valorHora;
        }

        public void printPay() {
                System.out.println("Pagamento:" + getPay());
        }
}
```

PorComissao.java

```
package exercicio06;
public class PorComissao extends Empregado implements Comission{
        private double sales;

        public PorComissao(String nome) {
                super(nome);
        }

        @Override
        public double getPay() {
                return sales*COMISSION_RATE;
        }
        @Override
        public void setVendas(double sales) {
                this.sales = sales;
        }

        public void printPay() {
```

```
                System.out.println("Pagamento:" + getPay());
        }
}
```

PorHoraeComissao.java

```
public class PorHoraeComissao extends PorHora implements Comission {
        private double sales;

        public PorHoraeComissao(String nome, int numHora, double valorHora) {
                super(nome, numHora, valorHora);
        }

        public double getPay() {
                return super.getNumHora()*super.getValorHora() + sales*COMISSION_RATE;
        }
        @Override
        public void setVendas(double sales) {
                this.sales = sales;
        }
        public void printPay() {
                System.out.println("Pagamento:" + getPay());
        }
}
```

Commission.java

```
package exercicio06;
public interface Comission {
        double COMISSION_RATE = 0.10;

        void setVendas(double sales);
}
```

Main.java

```
package exercicio06;
public class Main {
        public static void main(String[] args) {
                PorHora e1 = new PorHora("Ana", 160, 10.00);
                e1.printPay();

                PorComissao e2 = new PorComissao ("Joao");
                e2.setVendas(10000.00);
                e2.printPay();

                PorHoraeComissao e3 = new PorHoraeComissao("Carlos", 160, 10.00);
                e3.setVendas(5000.00);
                e3.printPay();

        }
}
```

7) (**Herança**) Utilize o conceito de herança para implementar as classes descritas abaixo.
Seja a classe Pessoa (atributos nome, cpf e email). A classe Fornecedor é subclasse de Pessoa e cada instância da classe Fornecedor tem, além dos atributos da classe Pessoa, os atributos cred correspondente ao crédito máximo e valorEmDivida que é o montante da dívida com o fornecedor. A classe Fornecedor deve ter um método obterSaldo que devolve a diferença entre os valores dos atributos cred e valorEmDivida.
Seja a classe Empregado, subclasse da classe Pessoa. Cada instância da classe Empregado tem, além dos atributos de Pessoa, os atributos numeroSeccao, salarioBase (vencimento base) e INSS (percentagem retida para INSS). Escreva a classe Empregado com métodos get e set e um método calcularSalario.
Seja a classe Administrador como subclasse da classe Empregado. Um determinado administrador tem como atributos, além dos atributos da classe Pessoa e da classe Empregado, o atributo ajudasDeCusto (ajudas referentes a viagens, estadias, ...). O salário de um administrador é equivalente ao salário de um empregado usual acrescido das ajudas de custo.
Escreva a classe Operario como subclasse da classe Empregado. Um determinado operário tem como atributos, além dos atributos da classe Pessoa e da classe Empregado, o atributo valorProducao (que corresponde ao valor monetário dos artigos produzidos pelo operário) e comissão (que corresponde à percentagem do valorProducao que será adicionado ao salário do operário). O salário de um operário é equivalente ao salário de um empregado usual acrescido da referida comissão.

8) (**Herança, Classe e Métodos Abstratos**) Implemente a hierarquia de classes a seguir.
Classe ContaBancaria. Uma conta bancária possui como atributos um número e um saldo. Como métodos, devem estar presentes os métodos abstratos sacar(double valor), depositar(doble valor) e tirarExtrato().
Classe ContaCorrente. Uma conta corrente possui como atributo um limite de cheque especial, além dos atributos número e saldo, definidos na classe ContaBancaria. O método sacar deve ser implementado nesta classe considerando-se o atributo limite de cheque especial. O método tirarExtrato deve retornar o valor do saldo em conta, acrescido do limite de cheque especial.
Classe ContaPoupanca. Uma conta poupança possui como atributo um valor que indica a taxa de rendimento da conta, além dos atributos número e saldo, definidos na classe ContaBancaria. O método tirarExtrato deve retornar o valor de saldo em conta multiplicado pela taxa de rendimento da conta.
Classe Agencia. Esta classe deve permitir armazenar contas em um array, sejam elas contas corrente ou contas poupança. O construtor da classe Agencia deve receber como parâmetro a quantidade máxima de contas que a agência pode armazenar. Crie, para esta classe, um método addConta, que permita incluir contas na agência; um método montanteContaCorrente, que retorna o montante de saldo acumulado em todas as contas correntes da agência; um método montanteContaPoupança, que retorna o montante de saldo acumulado em todas as contas poupanças da agência e um método imprimeContas, que permita visualizar as informações referentes às contas armazenadas.

ClasseAgenciaTeste. Instanciar nesta classe um objeto da classe Agencia, dois objetos da classe ContaCorrente e um objeto da classe ContaPoupanca. Adicionar à agência as contas criadas. Em seguida, imprimir as informações referentes às contas, bem como o montante total de saldo das contas correntes e o montante total de saldo das contas poupanças.

9) (**Classes e Métodos Abstratos**) Escreva uma classe abstrata chamada CartaoWeb. Essa classe representa todos os tipos de cartões web e conterá apenas um atributo: destinatário (tipo String). Nessa classe você deverá também declarar o método public abstract void showMessage(). Crie classes filhas da classe CartaoWeb: DiaDosNamorados, Natal, Aniversario. Cada uma dessas classes deve conter um método construtor que receba o nome do destinatário do cartão. Cada classe também deve implementar o método showMessage(), mostrando uma mensagem ao usuário com seu nome e que seja específica para a data comemorativa do cartão. Escreva um programa e no método main crie um array de CartaoWeb. Insira instâncias dos 3 tipos de cartões neste array. Após, use um laço para exibir as mensagens deste cartão chamando o método showMessage().

10) (**Exceção**) Dada a classe Quadrado a seguir:

```
public class Quadrado{
        private double lado;
        public setLado(double l){
                lado = l; }
public double área() {
        return lado*lado; }
}
```

Modifique esta classe de forma que uma exceção específica seja lançada pelo método setLado caso o valor informado para o lado do quadrado seja negativo. Essa exceção deve ser tratada no método main, permitindo que o usuário tente novamente fazer a entrada de dados, digitando um número válido (inteiro e positivo).

MinhaExcecao.java

```
package exercicio10;
public class MinhaExcecao extends Exception {

        public MinhaExcecao(String mensagem){
                super(mensagem);
        }
}
```

Quadrado.java

```
package exercicio10;
public class Quadrado {
        private double lado;

        public void setLado(double l) throws MinhaExcecao{
                if (l <= 0) {
```

```
                    MinhaExcecao e = new MinhaExcecao("O Valor do lado deve ser positivo");
                    throw e;//lança a exceção
                }
                lado = l;
        }

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

Main.java

```
package exercicio10;
import java.util.Scanner;
public class Main {
        public static void main(String[] args) {
                Quadrado q = new Quadrado();
                Scanner ler;
                boolean loop = true;
                while (loop) {
                        try {
                                ler = new Scanner (System.in);
                                int l = ler.nextInt();
                                q.setLado(l);
                                loop = false;
                        }
                   catch (MinhaExcecao e) {
                        System.out.println(e.getMessage());
                   }
                }
        }
}
```

11) (**Exceção**) Desenvolva um programa Java que crie um vetor de 5 posições de Strings. O programa deve solicitar ao usuário que preencha cada posição do vetor com valores que serão lidos. Em seguida, o programa deve permitir ao usuário realizar uma busca no vetor citado, informando a posição do vetor que deseja pesquisar. O programa então deve retornar a String localizada na posição informada e, também, qual o caracter que se encontra na quinta posição dessa String. Os resultados também devem ser informados utilizando-se caixas de diálogos. O programa deve ainda tratar todos os tipos de exceções que possam ser geradas em tempo de execução do mesmo, informando ao usuário qual o erro cometido.

Main.java

```
package exercicio11;
import java.util.InputMismatchException;
import java.util.Scanner;
public class Main {
        public static void main(String[] args) {
```

```
                String v[] = new String[5];

                Scanner ler = new Scanner (System.in);
                for (int i = 0; i < v.length; i++) {
                        String x = ler.nextLine();
                        v[i] = x;
                }
                try {
                        System.out.println("Digite a posição de consulta:");
                        int i = ler.nextInt();
                        System.out.println(v[i]);
                        try {
                                System.out.println("5o Caracter:" + v[i].charAt(4));
                        }
                        catch (IndexOutOfBoundsException e) {
                                System.out.println("String tem menos de 5 caracteres");
                        }
                }
                catch (InputMismatchException e) {
                        System.out.println("Posição deve ser número de 0 a 4");
                }
                catch (IndexOutOfBoundsException e) {
                        System.out.println("Posição inválida do vetor [0..4]");
                }
        }
}
```

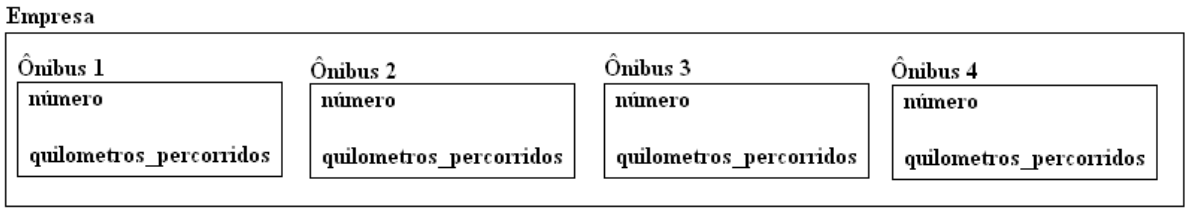
12) (**Exceção**) Desenvolva um programa em Java que contenha um método que recebe uma String como parâmetro e verifica se a mesma é composta apenas por caracteres maiúsculas. O método deve gerar dois tipos de exceções específicas: uma para indicar se existe algum caracter que não é uma letra e outra para indicar que uma das letras não é maiúscula. Para verificar o tipo dos caracteres use os métodos isLetter e isUpperCase, da classe Character, ambos static. Crie também uma classe teste para validar o método descrito acima e tratar as exceções que eventualmente são geradas pelo mesmo.

Main.java

```
package exercicio12;
import java.util.Scanner;
public class Main {
        public static void main(String[] args) {
                Scanner ler = new Scanner(System.in);
                String s = ler.nextLine();
                try {
                        Maiuscula(s);
                }
    catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
```

```
    }
        }
        public static void Maiuscula(String s) {
                        for (int i =0; i <s.length(); i++) {
                                if (!Character.isLetter(s.charAt(i))) {
                                        IllegalArgumentException e = new
IllegalArgumentException("Não é letra");
                                        throw e; //lança a exceção
                                 }
                                if (!Character.isUpperCase(s.charAt(i))) {
                                        IllegalArgumentException e = new
IllegalArgumentException("Não é maiúscula");
                                        throw e; //lança a exceção
                                        }
                        }
        }
}
```

13) (**ArrayList**) Considere uma empresa de ônibus,onde, para todos os ônibus deseja-se armazenar informações como número do ônibus e quilometragem total percorrida durante uma semana.



Escreva um projeto em Java que implemente a(s) classe(s) necessária(s) para modelar a situação descrita acima. Considere ainda que a empresa deve ser capaz de cadastrar novos ônibus em sua frota, bem como verificar a quilometragem total percorrida por todos os ônibus, juntamente com o maior e menor percurso realizado. Em seguida, crie uma classe para testar a(s) classe(s) implementadas.

Onibus.java

```
package exercicio13;
public class Onibus {
        private int num;
        private double km;

        public double getKm() {
                return km;
        }
        public Onibus(int num, double km) {
                this.num= num;
                this.km = km;}
}
```

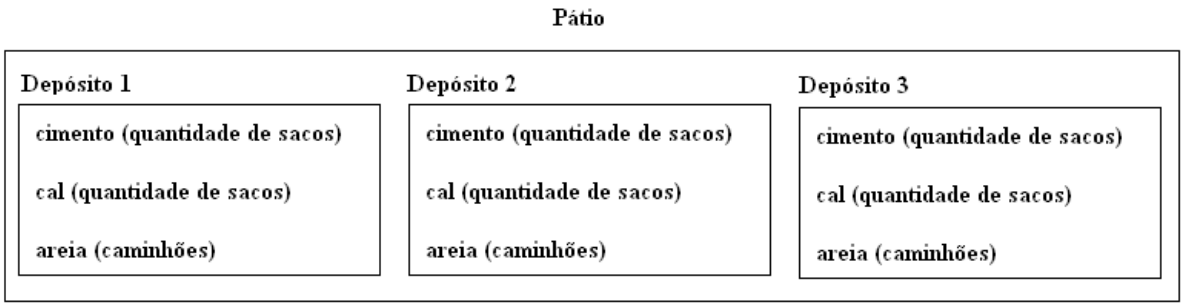
Empresa.java

```
package exercicio13;
import java.util.ArrayList;
public class Empresa {
	private ArrayList <Onibus> frota;

	public Empresa() {
		frota = new ArrayList<Onibus> ();
	}

	public void addOnibus(Onibus o) {
		frota.add(o);
	}

	public double getKmTotal() {
		double soma = 0.0;
		for (Onibus o: frota) {
			soma += o.getKm();
		}
		return soma;
	}
}
```

Main.java

```
package exercicio13;
public class Main {
	public static void main(String[] args) {
		Empresa emp = new Empresa();
		Onibus o1 = new Onibus(1, 450.7);
		Onibus o2 = new Onibus(2, 788.55);
		emp.addOnibus(o1);
		emp.addOnibus(o2);
		System.out.println(emp.getKmTotal());
	}
}
```

14) (**ArrayList**) Considere uma loja de materiais de construção especializada na venda de três tipos de produtos: cimento, cal e areia. Esta loja possui um pátio, composto por vários depósitos, nos quais estão armazenados seus estoques disponíveis para venda, conforme ilustrado em figura abaixo:



Deseja-se desenvolver um projeto em Java capaz de modelar a situação descrita acima. Considere na implementação que mais depósitos possam ser cadastrados no pátio. Deseja-se ainda verificar o total de cada material (areia, cal e cimento) disponível em estoque, considerando-se os depósitos disponíveis. Uma classe Teste deve estar presente para testar as funcionalidades das demais classes.

15) (**Padrões de projeto**) Escreva, compile e execute o programa abaixo. Em seguida, troque sua implementação para que a classe Incremental seja Singleton. Execute novamente e veja os resultados.

```
class Incremental {
	private static int count = 0;
	private int numero;
	public Incremental() {
		numero = ++count;
	}
	public String toString() {
	return "Incremental " + numero;
	}
}
```

```
public class TesteIncremental {
	public static void main(String[] args) {
		for (int i = 0; i < 10; i++) {
			Incremental inc = new Incremental();
			System.out.println(inc);
		}
	}
}
```

IncrementalSingleton.java

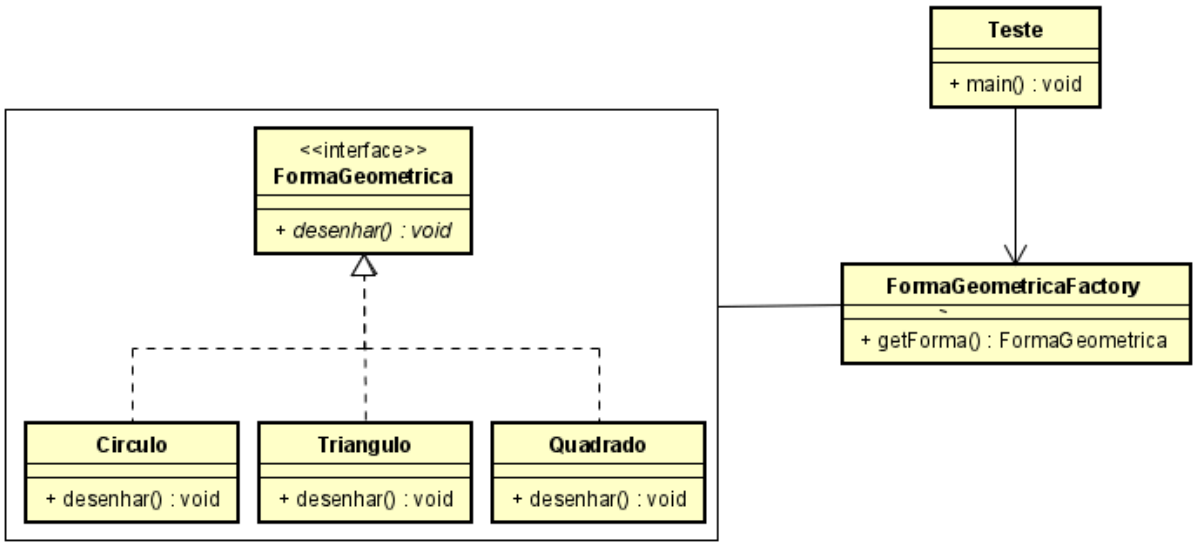
```
package exercicio15;
public class IncrementalSingleton {
	private static int count = 0;
	private int numero;

	private static IncrementalSingleton instancia = null;

	private IncrementalSingleton() {
```

```
                numero = ++count;
        }

        public static IncrementalSingleton getInstancia() {
                if (instancia == null)
                        instancia = new IncrementalSingleton();
                return instancia; }

        public String toString() {
                return "IncrementalSingleton" + numero;
        }
}
```

TesteIncrementalSingleton.java

```
package exercicio15;
public class TesteIncrementalSingleton {
        public static void main(String[] args) {
                for (int i = 0; i < 10; i++) {
                        IncrementalSingleton inc = IncrementalSingleton.getInstancia();
                        System.out.println(inc);
                }
        }
}
```

16) (**Padrões de projeto**) Utilize o padrão Factory Method para a criação de formas geométricas. O método desenhar deve imprimir na tela o nome da forma geométrica.



FormaFactory.java

```
package exercicio16;
public class FormaFactory {
        public Forma getForma(String tipo) { //factory method
```

```
            if(tipo == null)
                return null;
            if(tipo.equalsIgnoreCase("Circulo"))
                return new Circulo();
            else if(tipo.equalsIgnoreCase("Retangulo"))
                return new Retangulo();
            else if(tipo.equalsIgnoreCase("Quadrado"))
                return new Quadrado();
            return null;
        }
}
```

Forma.java

```
package exercicio16;
public interface Forma {
        void draw();
}
```

Circulo.java

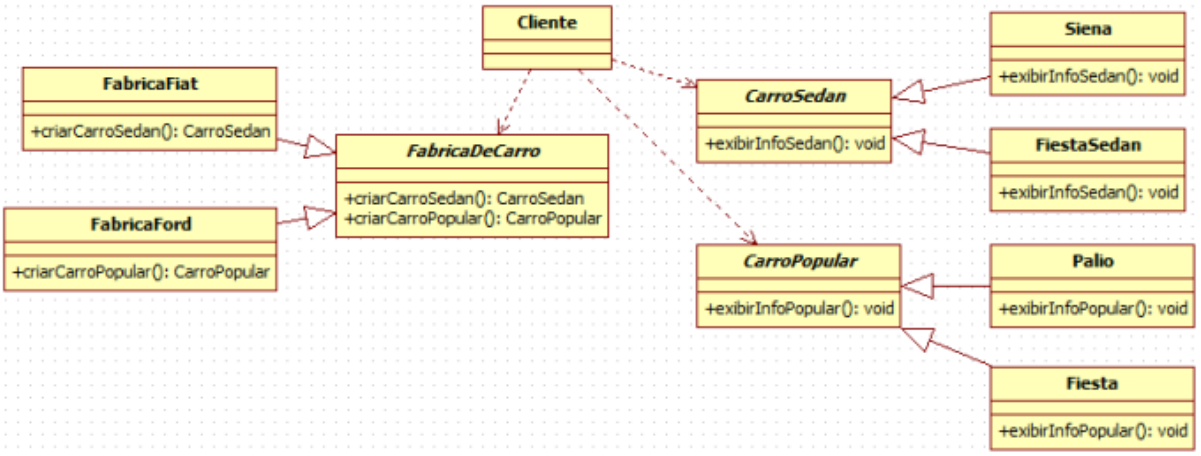
```
package exercicio16;
public class Circulo implements Forma {
        @Override
        public void draw() {
                System.out.println("Circulo::draw()");

        }
}
```

Main.java

```
package exercicio16;
public class Main {
        public static void main(String[] args) {
                FormaFactory formaFactory = new FormaFactory();
                Forma shape1 = formaFactory.getForma("Circulo");
                shape1.draw();
                Forma shape2 = formaFactory.getForma("Retangulo");
                shape2.draw();
                Forma shape3 = formaFactory.getForma("Quadrado");
                shape3.draw();
        }
}
```

17) (**Padrões de projeto**) Utilize o padrão Abstract Factory para a criação de uma fábrica de carros. O sistema pode construir carros de dois tipos: Sedan ou Popular.
A) Sedan
i - Siena - Fiat;
ii - Fiesta Sedan - Ford;

A) Popular
i - Palio - Fiat;
ii - Fiesta - Ford;
Dica: Utilize o seguinte diagrama de classes como referência.



FabricaDeCarro.java

```
package exercicio17;
public abstract class FabricaDeCarro { //abstract factory (super fábrica)
	public abstract CarroSedan criarCarroSedan();
	public abstract CarroPopular criarCarroPopular();
}
```

FabricaFiat.java

```
package exercicio17;
public class FabricaFiat extends FabricaDeCarro{
	@Override
	public CarroSedan criarCarroSedan() {
		return new Siena();
	}
	@Override
	public CarroPopular criarCarroPopular() {
		return new Palio();
	}
}
```

FabricaFord.java

```
package exercicio17;
public class FabricaFord extends FabricaDeCarro{
	@Override
	public CarroSedan criarCarroSedan() {
		return new FiestaSedan();
	}
	@Override
```

```
        public CarroPopular criarCarroPopular() {
                return new Fiesta();
        }
}
```

CarroPopular.java

```
package exercicio17;
public interface CarroPopular {
        void exibirInfoPopular();
}
```

CarroSedan.java

```
package exercicio17;
public interface CarroSedan {
        void exibirInfoSedan();
}
```

Fiesta.java

```
package exercicio17;
public class Fiesta implements CarroPopular {
        @Override
        public void exibirInfoPopular() {
                System.out.println("Modelo: Fiesta\nFábrica: Ford\nCategoria:Popular");
        }
}
```

Palio.java

```
package exercicio17;
public class Palio implements CarroPopular {
        @Override
        public void exibirInfoPopular() {
                System.out.println("Modelo: Palio\nFábrica: Fiat\nCategoria:Popular");
        }
}
```

Siena.java

```
package exercicio17;
public class Siena implements CarroSedan{
        @Override
        public void exibirInfoSedan() {
                System.out.println("Modelo: Siena\nFábrica: Fiat\nCategoria:Sedan");
        }
}
```

FiestaSedan.java

```
package exercicio17;
public class FiestaSedan implements CarroSedan{
        @Override
        public void exibirInfoSedan() {
```

```
                System.out.println("Modelo: Fiesta\nFábrica:Ford\nCategoria:Sedan");
        }
}
```

Cliente.java

```
package exercicio17;
public class Cliente {
        public static void main(String[] args) {
                FabricaDeCarro fabrica = new FabricaFiat();
                CarroSedan sedan = fabrica.criarCarroSedan();
                CarroPopular popular = fabrica.criarCarroPopular();
                sedan.exibirInfoSedan();
                System.out.println();
                popular.exibirInfoPopular();
                System.out.println();
                FabricaDeCarro fabrica2 = new FabricaFord();
                sedan = fabrica2.criarCarroSedan();
                popular = fabrica2.criarCarroPopular();
                sedan.exibirInfoSedan();
                System.out.println();
                popular.exibirInfoPopular();
        }
}
```