


Compartilhar o seu link com: [luciorocha @ professores.utfpr.edu.br](mailto:luciorocha@professores.utfpr.edu.br)

Nome:


Link:

Pedro Reis: Cópia de POCO4A - Aula 11 - Exercícios

Vitor Luiz de Castro Viana: Cópia de POCO4A - Aula 11 - Exercícios

Alexandre Olah de Almeida Lima:  Cópia de POCO4A - Aula 11 - Exercícios

Bruno Keller Margaritelli: Bruno - POCO4A - Aula 11 - Exercícios

luis mendes: <  Cópia de POCO4A - Aula 11 - Exercícios >


Rafael Kendy Naramoto Lopes < Cópia de POCO4A - Aula 11 - Exercícios >

Marcos Tadao Shoji < Cópia de POCO4A - Aula 11 - Exercícios >


Filipe Augusto Parreira Almeida Cópia de POCO4A - Aula 11 - Exercícios

João Vitor N. Yoshida < Cópia de POCO4A - Aula 11 - Exercícios >

Deivid da Silva Galvao < Cópia de POCO4A - Aula 11 - Exercícios >

Julio Farias:  Cópia de POCO4A - Aula 11 - Exercícios

Daniel Martins de Carvalho < Cópia de POCO4A - Aula 11 - Exercícios >

Matheus Hirata <  Cópia de POCO4A - Aula 11 - Exercícios

```
2rep2rep2rep2rep2rep2rep2rep2rep2rep2rep2rep2rep2rep>
```

Vitor Hugo Leite A. de Oliveira < Aula 11 - Exercícios > LIBERADASSO


Isabella Melo Almeida Cópia de POCO4A - Aula 11 - Exercícios

Séfora Davanso de Assis POO - Aula 11

Rodrigo Leandro Benedito: Cópia de POCO4A - Aula 11 - Exercícios

Victor Ramos Bernardes: Cópia de POCO4A - Aula 11 - Exercícios

Lucas dos Reis Viana: Cópia de POCO4A - Aula 11 - Exercícios

Gustavo Nunes:  Cópia de POCO4A - Aula 11 - Exercícios

Felipe Antonio Magro: Cópia de POCO4A - Aula 11 - Exercícios

Exemplo1:

Interface: é similar a uma classe. A interface é um tipo abstrato de dados. Ela define quais métodos a classe que a implementa deve utilizar. -A interface não implementa os métodos declarados.

Classe Abstrata: Generalizar os métodos das subclasses. A classe abstrata não implementa seus métodos abstratos, mas pode definir a implementação de métodos, inclusive de métodos da interface. Caso a classe não implemente os métodos abstratos, ela se torna abstrata. A classe abstrata não precisa ter todos os métodos abstratos. Pelo menos 1 (um) método deve ser declarado abstrato.

Exemplo:

```
public interface IMamifero {

    public abstract void setTipo( String tipo );

    public abstract String getTipo();

}

public abstract class Animal {
    private String tipo;

    public abstract void setTipo(String tipo);

    public String getTipo() {
        return this.tipo;
    }
    public void setTipoPadrao(){ this.tipo = tipo; }

}

public class Leao extends Animal
    implements IMamifero {

    public void setTipo(String tipo){ //Veio por heranca (abstract)
        if ( tipo.equals("LEAO")
            super.setTipoPadrao("LEAO");
        else
            super.setTipoPadrao("OUTROS");

    }

}

public class Principal {

    public Principal(){

        Leao leao = new Leao();
        leao.setTipo("LEAO");
        System.out.println( leao.getTipo() );

    }

}
```

```

    public static void main(String [ ] args){
        new Principal();
    }
}

```

Exercícios propostos:

1) Observe a Figura 1 a seguir:



Figura 1: Diagrama UML de Classes.

Classes: Empregado, Chefe, Balconista, Estagiario

```

public abstract class Empregado {
    private Data dataNascimento; //Composicao

    public Empregado( Data dataNascimento ){

        setDataNascimento( dataNascimento );
    }

    public final Data getDataNascimento(){ return this.dataNascimento; }
    public final void setDataNascimento(Data dataNascimento){
        this.dataNascimento = dataNascimento;
    }
}

```

```

        public abstract void folhaPagamento();
    }
    public class Chefe extends Empregado {
        public Chefe( Data dataNascimento ){
            super( dataNascimento );
        }
        public void folhaPagamento(){ System.out.println(100.0); }
    }
    public class Balconista extends Empregado
        implements IEmpregado {
        public Balconista( Data dataNascimento ){
            super( dataNascimento );
        }
        public void folhaPagamento(){ System.out.println(10.0); }
        public void imprimir(){ System.out.println("BALCONISTA"); }
    }
    public class Estagiario extends Empregado {
        public Estagiario( Data dataNascimento ){
            super( dataNascimento );
        }
        public void folhaPagamento(){ System.out.println(90.0); }
    }

    public class Data {
        private int dia;
        private int mes;
        private int ano;

        public Data(int dia, int mes, int ano){
            setDia( dia );
            setMes( mes );
            setAno( ano );
        }

        public int getDia(){ return this.dia; }
        public int getMes(){ return this.mes; }
        public int getAno(){ return this.ano; }

        private void setDia(int dia){ this.dia = dia; }
        private void setMes(int mes){ this.mes = mes; }
        private void setAno(int ano){ this.ano = ano; }
    }

```

```

public interface IEmpregado {

    public abstract void imprimir();

}

public class Principal {

    public Principal(){

        ArrayList<Empregado> lista = new ArrayList<>();
        Data data = new Data( 29, 4, 1982);
        Chefe joao = new Chefe( data );
        lista.add( joao );

        Chefe maria = new Chefe( data );
        lista.add( maria );

        //for(int i=0; i<lista.length; i++)
        for( Chefe chefe : lista ) //Para cada elemento da lista
            System.out.println( chefe.folhaPagamento() );

    }

    public static void main(String [ ] args){
        new Principal();
    }
}

```

a) (DONE) Inclua no projeto a classe Data.

b)(DONE) Modifique o código para incluir a variável de instância 'private Data dataNascimento' na classe Empregado. Inclua um método acessor e um mutador para essa nova variável de instância.

c) (DONE) Não devem ser criados novos métodos nas classes Chefe, Balconista e Estagiario, porém, modifique o construtor de cada uma dessas Classes para incluir a Data de nascimento do funcionário.

- d) (DONE) Adicione apenas métodos acessores na Classe Data para cada uma das variáveis de instância.
- e) (DONE) Suponha que a folha de pagamento seja processada uma vez por mês. Crie um vetor de objetos Empregado para armazenar referências a vários objetos de funcionários.
- f) (DONE) Crie a interface IEmpregado que será implementada pela Classe Balconista.

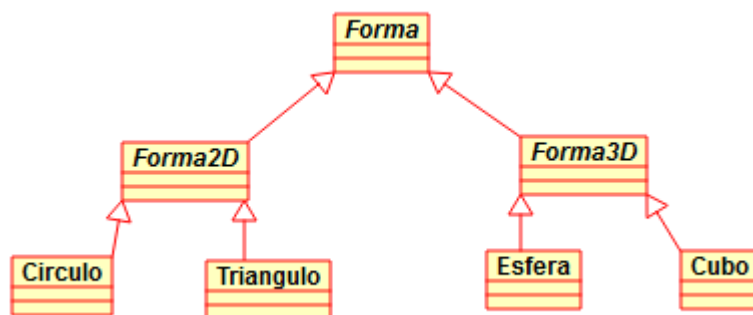


2) Faça a implementação Orientada a Objetos do problema anunciado a seguir:

- a) Crie 3 (três) classes não relacionadas por herança: Construção, Carro e Bicicleta.
- b) Dê a cada Classe atributos e comportamentos únicos que não estão presentes em outras classes.
- c) Crie a Interface EmissaoCarbono com um método getEmissaoCarbono.
- d) Cada Classe deve implementar a Interface EmissaoCarbono.
- e) Invoque o método getEmissaoCarbono de cada objeto.



3) Observe a Figura 2 a seguir:



- a) Implemente a hierarquia de Classes mostrada na Figura. Apenas as Classes folha são Classes concretas, as demais são classes abstratas.
- b) A Classe Forma2D deve conter o método `getArea`.
- c) A Classe Forma3D deve conter os métodos `getArea` e `getVolume`.
- d) Crie uma Classe Principal que tenha um vetor de Formas com objetos de cada Classe concreta.
- e) O programa deve imprimir o tipo de cada objeto instanciado.