1) Defina os conceitos abaixo e exemplifique com código Java.

A. Classe;

```java
//Java
class Personagem {

}
```

- Estrutura que abstrai(**serve de base**) um conjunto de objetos com características similares;
- Define o comportamento de seus objetos através de métodos;
- Define os estados possíveis destes objetos, através de atributos.
- A partir das classes que se pode criar objetos.

B. Objeto;

```java
//Java

Personagem personagem = new Personagem();
```

- É a instanciação de uma classe;
- representação de um conceito/entidade do mundo real, com significado bem definido para determinado software.

C. Atributo;

```java
//Java
class Personagem {

  String nome;
  String cor;
  int quantidadeDeCogumelos;
  float altura;
  String tipoFisico;
  boolean possuiBigode;

}
```

- É o elemento de uma classe, responsável por definir sua estrutura de dados.
- O conjunto de atributos representa as características da classe;
- Fará parte dos objetos criados a partir da classe.
- Deve ser de algum tipo (ex. boolean, byte, short, int, long, float, double, char, String ou Classe;

D. Método;

```java
String getNome() {
  return nome;
}

void setNome(String nome) {
  this.nome = nome;
}

// get/set para os demais

void pular() {
  // implementação aqui
}
```

- É uma porção do código (sub-rotina) que é disponibilizada pela classe;
- Executado quando é feita uma requisição a ele;
- Serve para identificar quais serviços, ações que a classe oferece;
- Define e realiza determinado comportamento.
- Retornam: tipos primitivos, classes ou void(sem retorno)

E. Método Construtor;

```java
//Java
class Personagem {

  // Atributos definidos anteriormente

  //Construtor
  Personagem () {
    // implementação desejada
  }
```

- Método especial, responsável por criar objetos a partir da classe em questão.
- Instancia objetos, para que se manipule de forma efetiva seus atributos e métodos;
- Possui o mesmo nome da Classe;

F. Método Acessor e Método Modificador;

```java
String getNome() {
  return nome;
}

void setNome(String nome) {
  this.nome = nome;
}
```

- Métodos que fornecem atributos privados(assessores), os chamados Getters;
- Métodos que modificam os atributos privados(modificadores, os chamados setters.

G. Método main;

```
public class ExemploLista {

    public static void main(String[] args) {

        ArrayList<Aluno> listaAlunos = new ArrayList<>();
```

- Método principal;
- Ponto de início de execução de aplicação java
- Associado à classe;
- método estático

H. Método static;

```
static void Andar()
{
  // implementação desejada
}
```

- Método que pertence à classe e não ao objeto;
- usa a palavra reservada static;
- Acessados através da classe.

I. Visibilidade de Atributo e Método

- Modificadores de acesso;
- Controla até que ponto uma classe, um método ou atributo pode ser usado;
- Fundamental para o uso efetivo de OO;

a)- private

mais restritiva;
atributos e métodos só podem ser manipulados no local de sua definição(dentro da classe em que foram definidos;

b)- protected

visibilidade intermediária;
atributos e métodos só podem ser manipulados no mesmo pacote, no local de sua definição e nas classes que estendam(herdam) a classe na qual foram definidos;
ou seja, nas classes e subclasses

c)- public

menos restritiva;
todos os membros definidos são acessíveis de qualquer lugar, em qualquer pacote

d)- default ou package private(sem modificador)

os membros da classe só serão visíveis dentro do mesmo pacote

J. Mensagem;

```
//Java

Pessoa pessoa = new Pessoa();

pessoa.falar();

Pessoa.andar();
```

- Mensagem é o processo de ativação de um método de um objeto;
- Ocorre com a chamada do método;
- Dispara a execução do comportamento descrito pela classe;
- pode ser direcionada diretamente à classe(método estático)

K. Abstração;

- Processo pelo qual se isolam características de um objeto, considerando os que tenham em comum certos grupos de objetos.

L. Encapsulamento;

- Esconder como algo foi feito, dando a quem precisa apenas o resultado gerado;
- ocultação de informação;
- evita acessos indevidos;

M. Classe Abstrata;

```
//Java
abstract class Funcionario extends Pessoa {

    Date dataAdmissao;
    String matricula;

    // get/set e métodos afins
}

abstract class Medico extends Funcionario {

    int CRM;
```

- Respresentam conceitos genéricos;
- Apenas abstraem;
- Devem ser completados pelas classes que gerdarem dela(subtipos);
- não podem ser instanciadas(não se pode criar objetos diretamente a partir delas;

N. Herança;

```java
//Java
class Pessoa {

    String nome;
    String sexo;

    // get/set e métodos afins
}

class Paciente extends Pessoa {

    Date dataInternacao;

    // get/set e métodos afins
}

class Funcionario extends Pessoa {

    Date dataAdmissao;
    String matricula;
```

- Herança é o relacionamento entre classes em que uma classe chamada de subclasse (classe filha, classe derivada) é uma extensão, um subtipo, de outra classe chamada de superclasse (classe pai, classe mãe, classe base).;
- herda atributos e métodos possíveis da classe mãe;
- a subclasse pode definir seus próprios membros(atributos e métodos***nova nomenclatura).

O. Interface;

```java
interface IDemonstrativoOperacional {

    double disponibilizarFaturamentoMensal();

    Procedimento[] informarProcedimentoExecutados();
}

class TransmissaoDadosMinisterio implemets IDemonstrativoOperacio
nal {

    @Override
    public double disponibilizarFaturamentoMensal() {
        // implementação específica para o hospital
        // conseguir informar seu faturamento mensal
    }

    @Override
    public Procedimento[] informarProcedimentoExecutados() {
        // implementação específica para o hospital
        // conseguir informar os procedimentos executados
    }

}
```

- Interface define um contrato que deve ser seguido pela classe que
a implementa;
- Quando uma classe implementa uma interface, ela se
compromete a realizar todos os comportamentos que a interface
disponibiliza;
- Boa prática começar o nome da interface com a letra I

P. Polimorfismo;

```java
class Obstetra extends Medico {

    @Override
    void operar() {
        // passos a serem seguidos para realizar o parto em si
    }
}

class Pediatra extends Medico {

    @Override
    void operar() {
        // passos a serem seguidos para averiguar a saúde do recé
m-nascido
    }
}
```

- Utilizado quando precisamos que um mesmo método se comporte de forma diferente dependendo do objeto instanciado a partid de uma classe de uma hierarquia qualquer;

Q. Sobrecarga de Método;

```java
//Java
class Quadrilatero {

  // área do quadrado
  double calcularArea(double lado) {
    return lado * lado;
  }

  // área do retângulo
  double calcularArea(double baseMaior, double baseMenor) {
    return baseMaior * baseMenor;
  }
```

- métodos com entradas(parâmetros) diferentes;
- deve manter o mesmo nome do método;

R. Variável Estática;

```java
//Java
class Pessoa {

  String nome;
  static int quantidaDeOlhos;

}
```

- Variável da classe, não do objeto;
- Podem ser acessados diretamente da classe;
- não precisa instanciar objeto para acessá-lo
- pode ser acessado/usado via objeto(não é boa prática)

S. Args

```java
// parameter: array
public static void main (String[] args) {}

// parameter: varargs
public static void main (String... args) {}

// static public/public static are ok
static public void main(String[] args) {}

// parameter name does not matter
public static void main (String...
listOfArgumentsOurUserSentUs){}

// parameter: array variation
public static void main (String args[]) {}
```

- objeto, parâmetro do método main, que pode ser um array ou varargs de String;
- Pode ser substituído por qualquer outro nome;
- Essencial para passar valores ao executar programa java por linha de comando;

T. Pacote

```java
package project.model;

public class Person {}
```

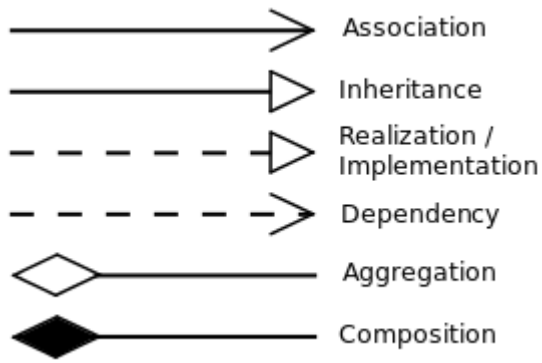
```java
import java.util.Calendar;

class Person {
    Calendar birthday;
}
```

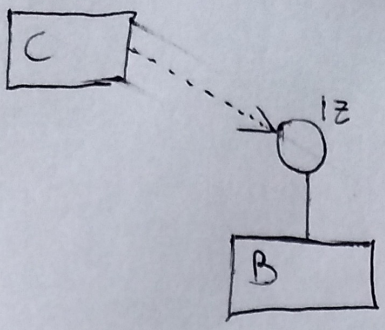
- Pacotes são usados pela JVM como uma maneira de encontrar as classes no sistema de arquivos, logo a estrutura de diretórios do projeto deve ser a mesma da estrutura de pacotes
- O nome do pacote deve ser todo em letras minúsculas;
• Um pacote deve começar com o site da empresa, ao contrário;
• Após o site, deve vir o projeto;
• Após o projeto, a estrutura é livre.
- Podem ser importados

2) Quanto às relações entre classes listadas abaixo, pede-se: Desenhe um diagrama de classes correspondente e o código que o implementa.

A. use;
B. é-um herança;
C. é-um interface;
D. tem-um associação;
E. todo-parte agregação;
F. todo-parte composição.



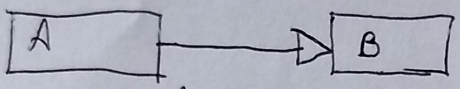
A_ USE



C usa Interface IZ para acessar B

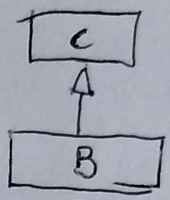
C depends Iz

B_ É-uma herança



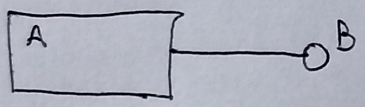
A é um B
A extends B

ou



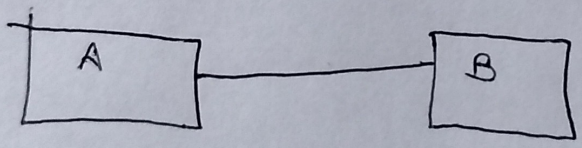
B é filho de C

C_ É-um interface



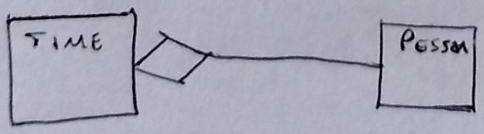
B é-um interface de A
A implements B

D_ Tem uma associação



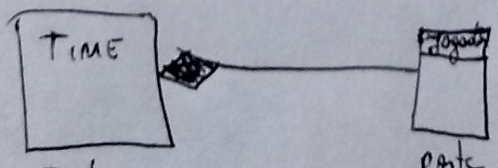
A tem associação com B

E_ AGREGAÇÃO



Time é todo
Pessoa é parte

F_ COMPOSIÇÃO



Jogador tem que pertencer ao time

Time é todo feito de jogadores um não existe sem o outro