Antes de começar a resolver as questões, leia atenciosamente as instruções a seguir:
1 - Os exercícios devem ser realizados **INDIVIDUALMENTE**.
2 - As resoluções dos exercícios devem estar em **ARQUIVOS DIFERENTES e uma pasta .zip com todos os arquivos deverá ser entregue até às 23:59 do dia 25/09/2023 pelo Google Classroom.**
3 - Nas descrições das questões, todos os atributos, métodos, classes e/ou interfaces em negrito devem ser explicitamente criados com o **MESMO NOME** nas soluções. Do contrário, você é livre para escolher qual a assinatura dos mesmos.
4 - Os tipos de entrada e saída explicitados devem ser respeitados.
5 - É permitido criar atributos, métodos, classes e/ou qualquer estrutura auxiliar que você considerar necessária para a resolução do problema. Porém, tudo que for pedido deverá **obrigatoriamente** ser implementado.
6 - Os métodos que forem sobrescritos devem ter a anotação **@Override**.
7 - **A correção da lista será feita através da análise individual de cada código, o principal aspecto não se baseia no *output* correto, mas em uma arquitetura de solução condizente com os princípios da programação orientada a objeto. Logo, utilizem *getters*, *setters*, modificações de visibilidade e todos os demais conceitos estudados em sala de aula na medida que julgarem necessário para resolução do problema.**
8 **-** Qualquer dúvida ou inconsistência em relação às questões, deve-se informar imediatamente aos monitores.
9 - Boa sorte!

[Q1] (3,0) Você está desenvolvendo um sistema de processamento de pagamento para uma loja online. Crie uma interface chamada **FormaDePagamento** que contenha os seguintes métodos:

- bool **autenticar**();
- void **processarPagamento**(double valor);

Agora, crie três classes que implementem a interface **FormaDePagamento** para representar diferentes formas de pagamento:

CartaoCredito: Esta classe representa um pagamento com cartão de crédito. O método **autenticar** deve pedir a senha do cartão e o código de segurança e retornar *true* caso seja sucesso, ou false caso o contrário, se esse método não retornar *true*, o método **procesarPagamento** deve retornar uma exceção avisando que o pagamento não foi autorizado. O método **procesarPagamento** deve exibir a mensagem "Pagamento com cartão de crédito no valor de [valor] processado com sucesso", onde [valor] é o valor passado como parâmetro.

BoletoBancario: Esta classe representa um pagamento com boleto bancário. O método **autenticar** deve pedir o código do boleto e retornar *true* caso seja sucesso, se esse método não retornar *true*, ou false caso o contrário, o método **procesarPagamento** deve retornar uma exceção avisando que o pagamento não foi autorizado. O método

procesarPagamento deve exibir a mensagem "Pagamento com boleto bancário no valor de [valor] processado com sucesso", onde [valor] é o valor passado como parâmetro.

PayPal: Esta classe representa um pagamento com PayPal. O método **autenticar** deve pedir o login e senha e retornar *true* caso seja sucesso, ou false caso o contrário, se esse método não retornar *true*, o método **procesarPagamento** deve retornar uma exceção avisando que o pagamento não foi autorizado. O método deve exibir a mensagem "Pagamento com PayPal no valor de [valor] processado com sucesso", onde [valor] é o valor passado como parâmetro.

Para a **autenticação**, cada implementação deverá salvar em um array de tuplas ou de algum objeto (escolha do aluno) que salve as informações necessárias para autenticação.
Exemplo para paypal:

```
class PayPalClient {
  1 usage
  public String login, senha;
  2 usages
  PayPalClient(String login, String senha) {
    this.login = login;
    this.senha = senha;
  }
}
no usages
public class PayPal {
  no usages
  PayPalClient[] clients = new PayPalClient[]{
      new PayPalClient( login: "pnc",  senha: "senha123"),
      new PayPalClient( login: "plc_master_123",  senha: "I_Love_Java")
  };
```

Ao autenticar, você deve conferir se as informações enviadas pelo usuário **(Pelo stdin, nesse caso, use a classe Scanner para receber o input**) batem com um das entradas salvas no seu array, se sim retorne *true*, se não, *false*.
Para a main, pergunte ao usuário o método de pagamento desejado e receba pelo stdin alguma string ou número que simboliza um dos métodos(fica a sua escolha), após isso, instancie a classe apropriada e chame os métodos, **autenticar** e **procesarPagamento**.

[Q2] (1,0) Imagine que você está desenvolvendo um sistema de gerenciamento de veículos para uma locadora de automóveis. Neste contexto, explique quando seria mais apropriado usar uma classe concreta, uma classe abstrata ou uma interface para representar os diferentes tipos de veículos disponíveis (por exemplo, carros, motos, bicicletas). Considere fatores como reutilização de código, extensibilidade e a necessidade de implementar comportamentos específicos para cada tipo de veículo. Justifique sua escolha em cada caso.

[Q3] (3,0) Em um sistema de gerenciamento de tarefas para uma equipe de desenvolvimento de software cada tarefa pode ser de diferentes tipos, como desenvolvimento de código, correção de bugs, revisão de código, etc. Além disso, cada tarefa é atribuída a um responsável.

Para lidar com essa diversidade de tarefas e responsáveis, você decide implementar uma classe chamada **Tarefa<T>**. A classe **Tarefa** deve ser capaz de armazenar informações comuns a todas as tarefas, como a descrição da tarefa e o responsável que pode ser

membro da equipe, líder da equipe, etc. No entanto, ela também deve ser flexível o suficiente para lidar com diferentes tipos de tarefas e membros da equipe de forma genérica.

Sua tarefa é criar a classe **Tarefa** permitindo que diferentes tipos de tarefas possam ser representados por objetos dessa classe. Além disso, você deve implementar um método que calcule a carga de trabalho total de um determinado membro da equipe, considerando todas as tarefas atribuídas a esse membro.

Na aplicação Main demonstre o uso da classe **Tarefa** e do método **calcularCargaDeTrabalho**. Crie instâncias de tarefas de diferentes tipos (por exemplo, desenvolvimento de código, correção de bugs, revisão de código) e atribua essas tarefas a diferentes membros da equipe. Em seguida, calcule a carga de trabalho total para cada membro da equipe. Segue um exemplo de como você pode testar o código na Main:

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

    MembroEquipe membro1 = new MembroEquipe("João");
    MembroEquipe membro2 = new MembroEquipe("Maria");

    LiderEquipe lider1 = new LiderEquipe("Pedro");

    Tarefa<MembroEquipe> tarefa1 = new Tarefa<>("Desenvolvimento de código", membro1);
    Tarefa<MembroEquipe> tarefa2 = new Tarefa<>("Correção de bugs", membro2);
    Tarefa<LiderEquipe> tarefa3 = new Tarefa<>("Revisão de código", lider1);

    List<Tarefa<MembroEquipe>> tarefas = new ArrayList<>();
    tarefas.add(tarefa1);
    tarefas.add(tarefa2);

    double cargaJoao = calcularCargaDeTrabalho(tarefas, membro1);
    double cargaMaria = calcularCargaDeTrabalho(tarefas, membro2);

    System.out.println("Carga de trabalho do João: " + cargaJoao + " tarefa(s).");
    System.out.println("Carga de trabalho da Maria: " + cargaMaria + " tarefa(s).");

  }
}
```

[Q4] (1,0)

CoffeeCin

Você tem a tarefa de implementar um programa de máquina de café que pode fazer diferentes tipos de bebidas de café, como "Espresso", "Latte" e "Cappuccino". Cada tipo de café tem ingredientes e etapas de preparação específicas. Utilize o **polimorfismo** para criar uma máquina de café flexível e extensível.

Instruções:

Crie uma classe **abstrata** chamada **Coffee** com os seguintes atributos e métodos:

Atributos:

- **name (String)**
- **waterRequired (double)**
- **milkRequired (double)**
- **coffeeRequired (double).**

Método:

- **prepare()**

que imprime as etapas necessárias para fazer o café.

Crie três classes que estendem Coffee:

- **Espresso**
- **Latte**
- **Cappuccino.**

Cada classe deve definir os valores específicos para waterRequired, milkRequired e coffeeRequired e implementar o método prepare() com instruções para fazer esse tipo de café.

Implemente uma classe **CoffeeMachine** com os seguintes métodos:

- **makeCoffee(Coffee coffee)**

Esse método recebe um objeto **Coffee** como parâmetro e simula a preparação do café especificado. Certifique-se de que ele deduza os ingredientes necessários dos recursos da máquina de café (por exemplo, água, leite, grãos de café).

Crie uma classe **Main** que simule uma cafeteria:

- Inicialize um **objeto CoffeeMachine**.
- Crie instâncias de diferentes tipos de café Espresso, Latte e Cappuccino.
- Use o método **makeCoffee()** para fazer cada tipo de café e exibir as etapas de preparação.

Fila de Cinema (1,0)

Você deve criar uma fila que faz com que as pessoas mais velhas fiquem sempre na frente de alguém mais jovem, e que existem 2 tipos de ingresso, um para os **adultos** e um para as **crianças**. Use o conceito de **polimorfismo** para diferenciar o tipo de ingresso que a pessoa terá e, usando os conceitos de **generics**, faça com que essa fila só aceite objetos que herdam do tipo **Pessoa** que você deve criar.

Instruções:

- Crie um **enum Ticket** que tenha os dois tipos de ingresso disponíveis: **ADULTO** e **CRIANÇA**;

- Crie uma classe abstrata chamada **Pessoa** que **implementa** a interface **Comparable**, essa classe deve ter um único método abstrato chamado **getTicketType()**, além desse método, a classe deve conter os outros atributos e métodos:

> Atributos:
- **idade(int)**
- **nome (String)**

Obs: use o construtor de Pessoa para inicializar ambos os atributos

> Métodos:
- **int compareTo(Person person) –** o método que deve ser implementado quando você cria uma classe que **implementa a interface Comparable** (use a idade para implementar esse método)
- **String getName()**
- **int getAge()**
- **toString() -** com o seguinte retorno: **name+": "+age+"["+getTicketType()+"]";**

- Crie duas classes que estendem a classe **Pessoa:**
- **Adulto**
- **Criança**

Obs: cada classe deve retornar o tipo de ingresso correspondente com a o seu tipo quando implementar a classe Pessoa.

- Crie a classe **Queue** que deve aceitar apenas objetos que implementam a classe Pessoa (use os conceitos de generics) e adicione os seguintes atributos e métodos na fila:

> Atributos:
- **pessoas (ArrayList<T>)**

Obs: use o construtor da classe para dizer a capacidade inicial da fila e se nenhum valor for informado, use **10** como um valor padrão.

> Métodos:
- **push(T pessoa) –** lembre-se de fazer com que as pessoas mais velhas sempre sejam inseridas antes das mais novas
- **T pop()**
- **boolena isEmpty()**

Obs: a intenção não é testar a melhor implementação quando fizer o push na fila, inclusive, recomendo que faça da seguinte forma: sempre insira no array e dê um sort pela ordem reversa, a classe **Collections** do java pode simplificar o processo!

- Implemente a função main de uma maneira semelhante como a imagem abaixo:

```java
public class Main {
    public static void main(String[] args) {
        Queue<Person> queue = new Queue<>( capacity: 5);

        // A linha 12 deve dar erro ao tentar ser compilada, com uma mensagem semelhante a:
        // Type parameter 'java.lang.Integer' is not within its bound; should extend 'person.Person'
        // Queue<Integer> q = new Queue<>();

        queue.push(new Child( age: 5,  name: "Child 1"));
        queue.push(new Adult( age: 30,  name: "Adult 1"));
        queue.push(new Child( age: 6,  name: "Child 2"));
        queue.push(new Adult( age: 20,  name: "Adult 2"));
        queue.push(new Child( age: 8,  name: "Child 3"));

        while (!queue.isEmpty()) {
            Person p = queue.pop();
            System.out.println(p);
        }
    }
}
```

E a saída deve ser semelhante a saída abaixo:

```
Adult 1: 30[ADULT]
Adult 2: 20[ADULT]
Child 3: 8[CHILD]
Child 2: 6[CHILD]
Child 1: 5[CHILD]
```

A saída acima foi obtida por causa do método **toString** da classe **Pessoa**, por isso, se você implementou de maneira diferente, mas que seja semelhante, não tem problema!

Sistema Eletrônico(1,0)

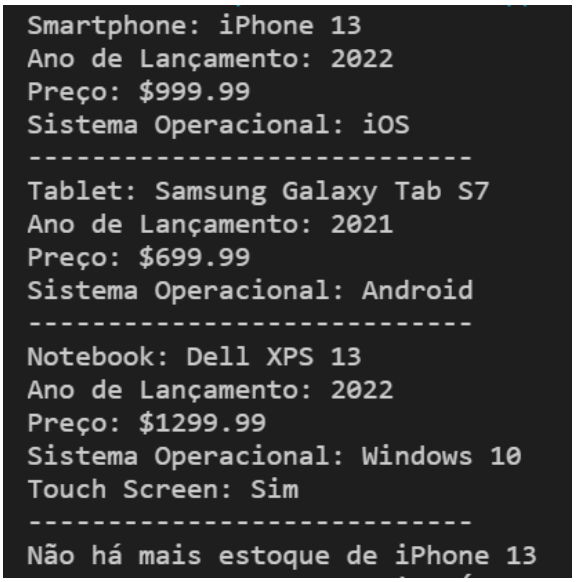
Imagine que você está desenvolvendo um sistema de venda de produtos eletrônicos. Crie uma classe abstrata **ProdutoEletronico** que contenha as informações:

- modelo → String
- anoLancamento → int;
- preco → double;
- sistemaOperacional → String;
- quantidadeDisponível → int;

Além de uma função *exibirInformacoes()*.

Em seguida, crie três classes que estendem **ProdutoEletronico**: Smartphone, Tablet e Notebook. Além dos atributos já citados, o notebook tem o boleano **isTouchScreen**, que indica se o modelo tem touch screen ou não. Em cada classe, implemente a função de exibirInformacoes() de acordo com a classe.

Crie uma classe **Loja** que contém a função *venderProduto(ProdutoEletronico produtoEletronico)*. Ela deve imprimir as informações relacionadas ao produto vendido. Caso não haja mais produto no estoque, a função deve imprimir que não há mais estoque daquele produto, conforme a figura abaixo.



Por fim, crie uma Main e inicialize a loja, crie instâncias dos diferentes tipos de produtos eletrônicos e simule a venda deles.