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 24/10/2022 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] (2,0) Implemente uma classe **Robot**, a qual representa uma abstração de um robô em um campo que contém nome (String), número da camisa (int) e localização como a posição no campo. Considere que existam robôs reservas e que para esses, a instanciação do objeto deverá conter somente o atributo de nome e todos os outros devem ser internamente definidos para valores *default*: -1 para camisa e (-1, -1) para posição, indicando que o robô está atualmente fora do campo. Assuma que os robôs não possam ter o mesmo nome.

Adicione uma classe **Team** que contém dois vetores de robôs (Robot[ ]), um compondo o time titular e o outro o time reserva, cada um sendo de no máximo seis robôs. Um objeto de time pode ser instanciado considerando a adição de todos os jogadores dos dois times ou sem nenhuma atribuição inicial. Logo, a classe deve possuir dois métodos de adição, onde um adiciona jogadores no time titular e o outro, no time reserva. Além disso, a classe deverá possuir um método de substituição, que remove um robô do campo e coloca no seu lugar um robô substituto, atualizando o time reserva e o titular. O método poderá variar seus parâmetros, recebendo dois objetos do tipo robô (um titular e um reserva) e trocando seus ids e posições ou recebendo somente um robô reserva. Nesse último caso, o mesmo deverá substituir o robô titular que tem mais vida em campo, ou seja, o mais antigo (inserido primeiro).

Na **Main**, instancie dois times, um começando vazio e outro com todos os jogadores já atribuídos. Para o primeiro, adicione dois titulares e dois reservas e substitua o reserva de sua preferência com o segundo jogador inserido. Para o segundo time, substitua o titular mais antigo por algum reserva. Por fim, mostre o nome de todos os jogadores titulares do primeiro time e de todos os reservas do segundo time.

[Q2] (3,0) Em um futuro distante, humanos (classe **Human**) e robôs (classe **Robot**) vivem em harmonia em uma cidade, ambos sendo considerados cidadãos. Um cidadão é representado pela classe **Citizen** e possui nome (String), endereço (String) e CEP (int), além de um método **greet(Citizen)** do tipo void, o qual tem a função de fazer um morador cumprimentar outro e será diferente para robôs e humanos. O humano possuirá também um vetor de amigos (Citizen[ ]) que sempre será inicializado internamente com todas as posições vazias, mas com tamanho máximo de cinco.

Os robôs da cidade não são muito simpáticos e interagem apenas com seus vizinhos. Por isso, em seu método de cumprimento, deve ser exibida uma String de saudação apenas se o robô e o morador recebido por parâmetro tiverem o mesmo CEP.

Para o humano, ao realizar o cumprimento, deve ser exibida uma String de saudação, em que o humano cumprimenta o outro cidadão por seu nome e, em seguida, o adiciona a seu vetor de amigos, caso já não seja.

Um humano pode ser uma criança ou um adulto. Uma criança (classe **Child**) tem um brinquedo preferido (String) e, toda vez que cumprimentar alguém, além do comportamento padrão, também exibirá um “Quer brincar com meu <brinquedo>?”. Já um adulto (classe **Adult**), tem uma profissão (String) e pode ter filhos (Child [ ]). O vetor de filhos de cada adulto poderá conter, no máximo, quatro crianças e deve ser inicializado internamente como vazio. Por isso, nessa classe, há um método **addKid(Child)**, o qual deve adicionar uma criança ao vetor de filhos do adulto.

Na classe **Main**, faça um robô cumprimentar um outro robô e um adulto, com o primeiro sendo do mesmo CEP. Adicione um filho ao adulto criado anteriormente e faça o mesmo cumprimentar algum dos robôs já criado. Por fim, exiba o nome de todos os amigos da criança e mude seu brinquedo preferido.

[Q3] (1,5) Escreva um programa que implementa classes relacionadas a pelo menos três figuras geométricas, como por exemplo, quadrado, triângulo e trapézio, onde todas possuem um padrão de serviço relacionados ao cálculo de área e perímetro. Logo, todas deverão possuir os métodos **calculateArea()** e **calculatePerimeter()**, além de atributos internos associados às suas arestas e respectivos valores.

Na **Main**, instancie três figuras geométricas diferentes e calcule a área e o perímetro de todas, exibindo que valores foram usados nos cálculos a partir dos atributos da classe.

[Q4] (3,5) A loja de magia dos Irmãos Weasley começou a expandir em várias sedes e precisa de um sistema para cadastro dos seus clientes, que são bruxos. Todo bruxo (classe **Wizard**) contém um nome (String) e uma varinha (String). Um cliente (classe **Client**) deve possuir um saldo (float) e uma pontuação (int), que será atualizada a cada compra realizada na loja e deve ser internamente definida como zero na criação do objeto.

A loja (classe **Store**) possui um banco de dados (classe **DataBaseWeasleys**) que tem a função de gerenciar os produtos de seu estoque, com tamanho máximo de 50 produtos e sendo inicialmente vazio.

Um banco de dados (interface **DataBase**) possui três métodos: inserir, remover e atualizar algum item. O banco de dados dos Wesleys (classe **DataBaseWesleys**) armazena, em um vetor de uma tupla, o nome de cada produto da loja (String), a quantidade (int) e o preço (float) de cada um deles. Assim, deve ser possível inserir e remover produtos desse banco de dados, bem como atualizar suas quantidades conforme as vendas ocorram na loja.

Implemente um programa para lidar com a compra e venda do cliente com a loja. Ou seja, o cliente deverá comprar algum item em uma loja fornecendo o item, mas, caso o seu saldo seja negativo ou insuficiente para o valor do produto, uma exceção deverá ser lançada. Caso a compra seja bem sucedida, o saldo e a pontuação deverão ser atualizados, com a segunda sendo acrescida de 10% do valor da compra.

Implemente um método auxiliar que verifica a possibilidade de receber um brinde - o que ocorre quando a pontuação do cliente atinge o valor 500 -, solicitando o brinde quando possível. O método deve ser chamado após toda compra e a loja deverá fornecer o item a de menor valor quando o brinde for solicitado.

Na **Main**, instancie uma loja com um determinado banco de itens e um cliente. Faça um caso de teste que cause uma exceção na compra. Além disso, faça uma compra bem sucedida de forma que o cliente consiga um brinde. Por fim, considere que a loja não irá mais comercializar algum dos produtos de seu estoque e, por isso, remova o mesmo do banco de dados.