Desafio de Programação do Elo7

Bem vindo candidato(a)!

Vamos explicar como funciona o nosso desafio:

Imagine que um desenvolvedor recebeu uma tarefa de uma pessoa da equipe de produto. A pessoa de produto queria poder controlar sondas em outros planetas por meio de comandos. Para explicar o funcionamento do produto, o seguinte exemplo foi escrito em um pedaço de papel:

Explicação da necessidade:

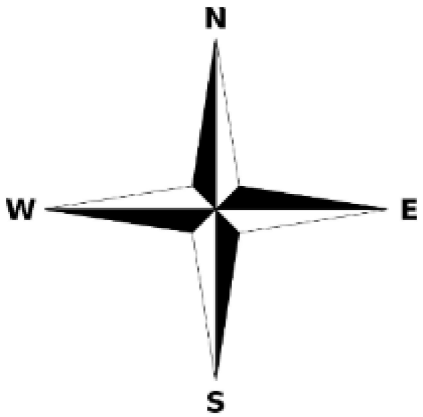
```
Tamanho da área do planeta : 5x5

Posição de pouso da sonda 1: x=1, y=2 apontando para Norte
Sequencia de comandos: LMLMLMLMM
Posição final da sonda: x=1 y=3 apontando para Norte

Posição de pouso da sonda 2: x=3, y=3 apontando para Leste
Sequencia de comandos: MMRMMRMRRML
Posição final da sonda: x=5 y=1 apontando para Norte
```

Detalhes sobre o funcionamento acima:

A sequência de comandos é um conjunto de instruções enviadas da terra para a sonda, onde :

- `M` -> Andar para a frente na direção que está 1 posição.
- `L` -> Virar a sonda para a esquerda (90 graus)
- `R` -> Virar a sonda para a direita (90 graus)

A orientação da sonda dentro do plano cartesiano usa uma rosa dos ventos como referência



O desafio

Certifique-se que você leu atenciosamente **todos** os checkbox para a sua vaga (estágio ou júnior). Os links que passamos são informativos, se você já domina os conteúdos não se sinta obrigado a lê-los! (mas se você está em dúvida provavelmente será melhor fazer a leitura)

Regra de negócios:

- [ ] Primeiramente você precisa verificar se a equipe de produto não pensou em todos detalhes da regra de negócios. Pense quais comportamentos fazem sentido para dar suporte em um cenário com **várias sondas pousando e se movimentando em um mesmo planeta com uma superfície limitada** (podendo haver vários planetas). Considere que as sondas possuem combustível infinito e **podemos passar comandos para uma sonda pousada a qualquer momento no futuro**.

- [ ] Para modelar a regra de negócio, pense nas principais palavras usadas pela pessoa de produto para descrever o problema, possivelmente fará sentido você criar classes para representar elas no seu código!

- [ ] No momento de decidir onde colocar as regras de negócio (lógica de movimentação da sonda etc) leia os links abaixo para entender nossos guidelines de como preferimos que seja isso feito:

  https://www.alura.com.br/artigos/nao-aprender-oo-getters-e-setters

  https://www.alura.com.br/artigos/o-que-e-modelo-anemico-e-por-que-fugir-dele

- [ ] Crie um repositório no github ou bitbucket e coloque a sua solução por lá, de preferência faça pequenos commits! **Favor enviar email com o repositório do teste assim que criá-lo**, vamos tentar acompanhar a solução. **Quando terminar sua solução por favor nos avise por email**.

Para vagas de ESTÁGIO:

Para vagas de estágio não esperamos que o candidato saiba como fazer uma API web. Se você está concorrendo a esse tipo de vaga sua tarefa é:

- [ ] Fazer a entrada de dados via terminal, lendo a entrada em um método main

Aspectos que vamos analisar:

- Separar conceitos por classe (ex: Carro, Motor..);
- Código precisa ser simples de entender;
- Conhecimentos básicos de modelagem na camada de modelo;
- Orientação a Objeto (encapsulamento, divisão de responsabilidades, acoplamentos saudáveis, imutabilidade etc).

- [ ] **Diferencial:** Se você conhecer o conceito de testes de unidade e quiser criar cobertura de testes automatizados para os comportamentos que você criou fique à vontade. Segue uma referência caso você queira encarar esse desafio.

- [ ] **Diferencial: (DIFÍCIL)** Para o teste de estágio não esperamos uma aplicação WEB, caso você já esteja familiarizado com APIs Web e queira tentar fazer esse upgrade na solução indicamos tentar ser aderente ao REST. Esse artigo pode ajudar.

Para vagas JUNIOR:

- [ ] Para vagas júnior é esperado algum conhecimento de como fazer aplicações web. A idéia é que você monte uma API Rest para poder pousar uma sonda em um planeta (ou várias sondas no mesmo planeta ou ainda várias sondas em planetas diferentes). Sugerimos usar o Spring Boot que possui um wizard para criar o projeto na estrutura correta. Segue uma referência que pode ser usada (tópico 5 do link).

- [ ] Caso você queira implementar uma camada REST pense carinhosamente em quais recursos você vai ter na sua API. Esse artigo pode ajudar.

Aspectos que vamos analisar:

- Separar conceitos por classe (ex: Carro, Motor..);
- Código precisa ser simples de entender;
- Conhecimentos básicos de modelagem na camada de modelo;
- Orientação a Objeto (encapsulamento, divisão de responsabilidades, acoplamentos saudáveis, imutabilidade etc);
- Básico de REST (Conhecimentos básicos de endpoint, métodos HTTP).

- [ ] Na solução você pode criar uma persistência em memória simples para armazenar seus objetos seguindo o modelo (não se preocupe com cenários de concorrência).

```
//imports omitidos

public class BancoEmMemoriaSonda {
    private static List<Sonda> sondas = new ArrayList<>();
    private static int proximoId = 1;

    public void salva(Sonda sonda) {
        sonda.setId(proximoId);
        proximoId += 1; //curiosidade: essa linha poderia ser usada acima com um
proximoId++
        sondas.add(sonda);
    }
    //outros métodos que você achar úteis
}
```

- [ ] **Diferencial:** em vez de usar o esquema acima de persistência em memória você pode escolher algum banco (relacional ou não relacional) para fazer a persistência dos dados.

- [ ] **Diferencial:** é altamente desejável (mas não obrigatório) que as regras de negócio mais importantes estejam cobertas por testes de unidade. (A cobertura não precisa ser 100%. Segue uma referência caso você queira encarar esse desafio).

Para vagas Pleno:

- [ ] Para vagas pleno vamos exigir todos os pontos do teste para junior, mas com mais maturidade na modelagem Orientada a Objeto, divisão de pacotes, responsabilidades das classes e uso mais maduro de framework (injeção de dependência etc).
- [ ] A cobertura com testes de unidade das regras de negócio mais relevantes é necessária para plenos.
- [ ] A persistência usando algum banco (relacional e/ou não relacional) é necessário para plenos.
- [ ] **Diferencial:** um desafio não obrigatório é pensar em uma persistência otimizada quando se fizer necessário no desafio.
- [ ] **Diferencial:** dar um bom suporte a concorrência para a API Rest.