CES-28 Prova 1 - 2017

Sem consulta - individual - com computador - 3h

PARTE I - ORIENTAÇÕES

1. Qualquer dúvida de codificação Java só pode ser sanada com textos/sites oficiais da **Oracle** ou **JUnit**.
    a. Exceção são idiomas (ou 'macacos') da linguagem como sintaxe do método .equals(), ou sintaxe de set para percorrer collections, não relacionados ao exercício sendo resolvido. Nesse caso, podem procurar exemplos da sintaxe na web.
2. Sobre o uso do **Mockito**, com a finalidade de procurar exemplos da sintaxe para os testes, podem ser usados sites de ajuda online, o próprio material da aula com (pdf´s, exemplos de código e labs), assim como o seu próprio código, **mas sem usar código de outros alunos**.
    a. Lembre-se de configurar seu build com os jar(s) existentes em **bibliotecas.zip**.
3. Questões com itens diversos, favor identificar claramente pela letra que representa o item, para que seja possível saber precisamente a que item corresponde a resposta dada!
    a. **NO CASO DE NÃO IDENTIFICAÇÃO, A QUESTÃO SERÁ ZERADA,**
4. Se necessário realizar implementação, somente serão aceitos códigos implementados no Eclipse. Essas questões ou itens serão indicados com o rótulo **[IMPLEMENTAÇÃO]**! Para as outras questões, pode ser usado o Eclipse, caso for mais confortável, digitando os exemplos, mas não é necessário um código completo, executando. Basta incluir trechos do código no texto da resposta.
5. Deve ser submetido tanto no TIDIA, como no GITLAB os seguintes entregáveis:
    a. **QUESTÕES DE IMPLEMENTAÇÃO:** Código completo e funcional da questão, bem como todas as bibliotecas devidamente configuradas nos seus respectivos diretórios. O projeto deve SEMPRE ter um "source folder" **src** (onde estarão os códigos fontes) e outro **test**, onde estarão as classes de testes, caso seja o caso. Cada questão de implementação deve ser um projeto à parte.
    b. **DEMAIS QUESTÕES:** Deve haver ser submetido um arquivo PDF com as devidas respostas. Use os números das questões para identificá-las.
6. No caso de diagramas, pode ser usado qualquer editor de diagrama UML, assim como desenhar no papel, tirar a foto, e **incluí-la no pdf dentro da resposta, NÃO como anexo separado**. **Atenção: use linhas grossas, garanta que a foto é legível!!!!**

PARTE II - CONCEITUAL

QUESTÃO 1 - CIRCLE X ELLIPSE

Dado que a classe **ELLIPSE** é pai da classe **CIRCLE** (*faz sentido, porque círculos são elipses*), a classe **CIRCLE** pode reusar todo o conteúdo da classe **ELLIPSE**, bastando para isso apenas sobrescrever os métodos, visando garantir que os eixos maior e menor permaneçam iguais. No entanto, o método

void Ellipse.stretchMaior() // "estica" a elipse na direcao do eixo maior

não funciona com **CIRCLE**, pois o resultado deixa de ser um círculo.

Não é possível fazer **CIRCLE** pai de **ELLIPSE**, pois seria conceitualmente errado, já que nem toda **ELLIPSE** é um **CIRCLE**. Analise, o que aconteceria se uma função espera um círculo e recebe uma elipse?
Além disso, o método *double Circle.getRadius()* não faz sentido com uma elipse.

a) Explique este dilema conceitualmente, usando para isso apenas os conceitos e vocabulários constantes de POO, especialmente àqueles relacionados a responsabilidade e herança. **(1.0 PT)**

Uma subclasse possui, a princípio, todos os métodos da superclasse disponíveis. Assim, faz sentido que a classe filha seja uma especialização da classe pai, que é o que acontece com CIRCLE, que é uma especialização de ELLIPSE.

Se uma função espera um círculo e recebe uma elipse, haverá erro e não será possível compilar o código. Conceitualmente está errado pois embora exista uma hierarquia entre as classes, elas são diferentes. Mais especificamente, ELLIPSE e CIRCLE não se comportam da mesma maneira, e é possível que haja métodos diferentes para cada uma delas. Isso significa que passar uma elipse no lugar de círculo pode gerar

mais erros na medida em que se tenta acessar métodos que estão implementados apenas em CIRCLE e não em ELLIPSE.

Para o caso de se criar uma hierarquia entre ELLIPSE e CIRCLE, surgirá um problema na medida em que a função stretchMaior() não pode ser implementada em CIRCLE, e sendo esta filha de ELLIPSE estaria implementada.

b) Forneça uma solução que ainda promova o reuso de código. A sua solução pode ter uma desvantagem, no ponto de vista do programador que usa as suas classes. Explique-a conceitualmente a solução e a desvantagem, usando o vocabulário de POO, do ponto de vista do programador que usa as suas classes. **(1.0 PT)**

Uma solução para o problema é criar uma classe (abstrata) que será pai tanto de ELLIPSE como de CIRCLE. As vantagens disso é que se pode implementar todos os métodos comuns às duas classes. Assim o método stretchMaior() estaria implementado apenas em ELLIPSE e a função getRadius(), apenas em CIRCLE. A desvantagem de tal abordagem é que um círculo não poderá ser utilizado no lugar de uma elipse, uma vez que não há uma hierarquia direta entre elas.

Para essa solução, temos:

```
public abstract class RoundedShape {
        …
}

public class Circle extends RoundedShape {

        …
        public double getRadius() {
        …
        }
}

public class Ellipse extends RoundedShape {

        …
        public void stretchMaior() {
        …
        }
}
```

Outra solução seria continuar com a hierarquia – ELLIPSE pai de CIRCLE – mas fazendo um override da função stretchMaior() dentro da classe CIRCLE, para que retorne uma exceção em vez de realizar uma ação. A desvantagem nesse caso fica sendo que a classe CIRCLE possui o método stretchMaior(), que não faz sentido para ela.

Nessa solução, teríamos:

```
public class Ellipse {

        …
        public void stretchMaior() {
        …
        }
}


public class Circle extends Ellipse {

        …
        public double getRadius() {
        …
        }

        @Override
        public void stretchMaior() {
        …
        }

}
```

OBS: NÃO SÃO NECESSÁRIAS IMPLEMENTAÇÕES COMPLETAS, APENAS DEVE-SE USUAR TRECHOS DE CÓDIGO NA RESPOSTA QUANDO RELEVANTE.

QUESTÃO 2 - TDD

Dado as sentenças abaixo, marque V para àquelas que são verdadeiras, ou F para as falsas. **(1.0 PT)**

[V] Podemos dizer que o exemplo a seguir é um bom exemplo de TDD?

Recebemos um código legado bastante grande de um projeto anterior, desenvolvido sem nenhum teste, e refatoramos o mesmo, criando testes. È iniciado pelo desenvolvimento de testes triviais, passando por testes simples,

testes de unidade, até chegar em testes maiores, com o objetivo de nos certificarmos de que o código funciona e posteriormente permitir a evolução e manutenção desse código.

[V] TDD supõe uma serie de ferramentas de desenvolvimento. A comparação do TDD versus um desenvolvimento não-TDD seria muito menos favorável se não existissem ferramentas e IDEs "bonitinhas" para automatizar testes, inclusive facilitar a leitura dos resultados dos mesmos, verificar rapidamente o que passou e não passou, facilitar inclusive varias refatoracoes comuns, e ferramentas de diff e controle de versão para reverter eventuais erros e/ou encontrar as últimas mudanças com data e responsável. Inclusive podemos considerar isso como uma das razoes porque o TDD demorou algumas décadas para aparecer, e não apareceu nos primórdios da computação.

[F] Refatorações no TDD são relativamente infrequentes, acontecem apenas quando é detectado algum erro que deve ser corrigido. Uma refatoração é sempre retrabalho e o resultado de algum erro humano.

[V] Há alguns casos limite tão comuns que praticamente sempre devemos testar pelo menos vários deles, especialmente quando se usam estruturas de dados. Caso vazio, cheio, apenas um elemento, ultimo e primeiro, usar o índice zero versus índice 1, etc. Para algumas estruturas de dados, pode também ser importante testar os casos de número de elementos par e ímpar, ou entrada ordenada e desordenada. Quando se implementa uma pilha, por exemplo, testar pelo menos algumas dessas condições deve ser um reflexo automático para o programador TDD.

PARTE III - IMPLEMENTAÇÃO

[IMPLEMENTAÇÃO] – QUESTÃO 3 UM BAR COM MAU CHEIRO.

Abra o projeto Pub.java, e execute os testes. Nesse projeto existe uma série de mal cheiros e problemas de responsabilidades.

IMPORTANTE - NUNCA MUDE OS TESTES! ELES DEVEM CONTINUAR FUNCIONANDO!

a) Refatore o código, criando novas classes de forma a dividir melhor as responsabilidades. **(3.0 PT)**
   i) Uma tarefa comum de manutenção deste código seria *incluir e remover ingredientes e drinks no modelo, ou modificar as regras em relação aos já existentes.*

b) A sua solução deve facilitar a tarefa de manutenção descrita em *a-i* e ainda continuar provendo o reuso de código. Considerando o requisito apresentado no item ***a-i***, explique como a manutenibilidade e reuso são promovidos pela sua solução. **(0.5 PT)**

Na solução apresentada, implementou-se uma interface Drink com os métodos getPrice(), isLimited() e hasStudentDiscount(). Elas fornecem o preço da bebida, se há limite na sua compra e se possui desconto para estudante, respectivamente. A partir dessa interface, implementou-se uma classe para cada bebida (Beer, Cider, BacardeSpecial, etc), em que implementa a interface Drink retornando os devidos valores para a dada bebida. Vale notar que nem todas as bebidas possuem um preço independente, algumas delas dependem do preço dos ingredientes. Para isso, essas bebidas instanciam as classes dos ingredientes necessários e assim obtém o preço pela soma dos preços dos ingredientes.

Ingredient é uma classe abstrata que possui métodos para retornar (e settar) o preço do ingrediente. Suas classes filho são os próprios ingredientes, que settam em sua construção o preço do ingrediente.

Por fim, criou-se uma classe DrinkFactory, que é a responsável por retornar uma bebida dado um pedido. Ela faz a tradução de uma String para um objeto de Drink (“hansa”, por exemplo, retorna um objeto Beer).

O que resta para a classe Pub é fabricar a bebida a partir do pedido do cliente com a classe DrinkFactory, depois checar se a bebida possui limite de vendas ou se o preço deve ter desconto para estudantes e, assim, retornar o preço final multiplicando pela quantidade de bebidas pedidas.

O código é de fácil manutenção devido ao baixo acoplamento e alta coesão. Cada classe possui uma tarefa específica: DrinkFactory retorna uma bebida dado um pedido; Drink retorna os dados de uma dada bebida; e Ingredient fornece o preço dele próprio.

Vale notar que Ingredient é package-private, o que significa que só pode ser acessada por classes dentro do DrinkPackage, e por isso não pode ser modificado pelo Pub, isso dá mais segurança e robustez à implementação.

Para se acrescentar uma nova bebida ao cardápio, por exemplo, basta criar a classe correspondente à bebida implementando a interface Drink e associá-la a um pedido em DrinkFactory.

Mudar os ingrediente de uma bebida também é simples, basta mudar as classes instanciadas no método que calcula o preço de uma bebida.

c) Escreva um diagrama UML representando a sua solução, considerando as classes, associações e tipos de associações (agregação/associação/composição), bem como multiplicidades. **(0.5 PT)**

A seguir é apresentada a imagem com o UML do código implementado.

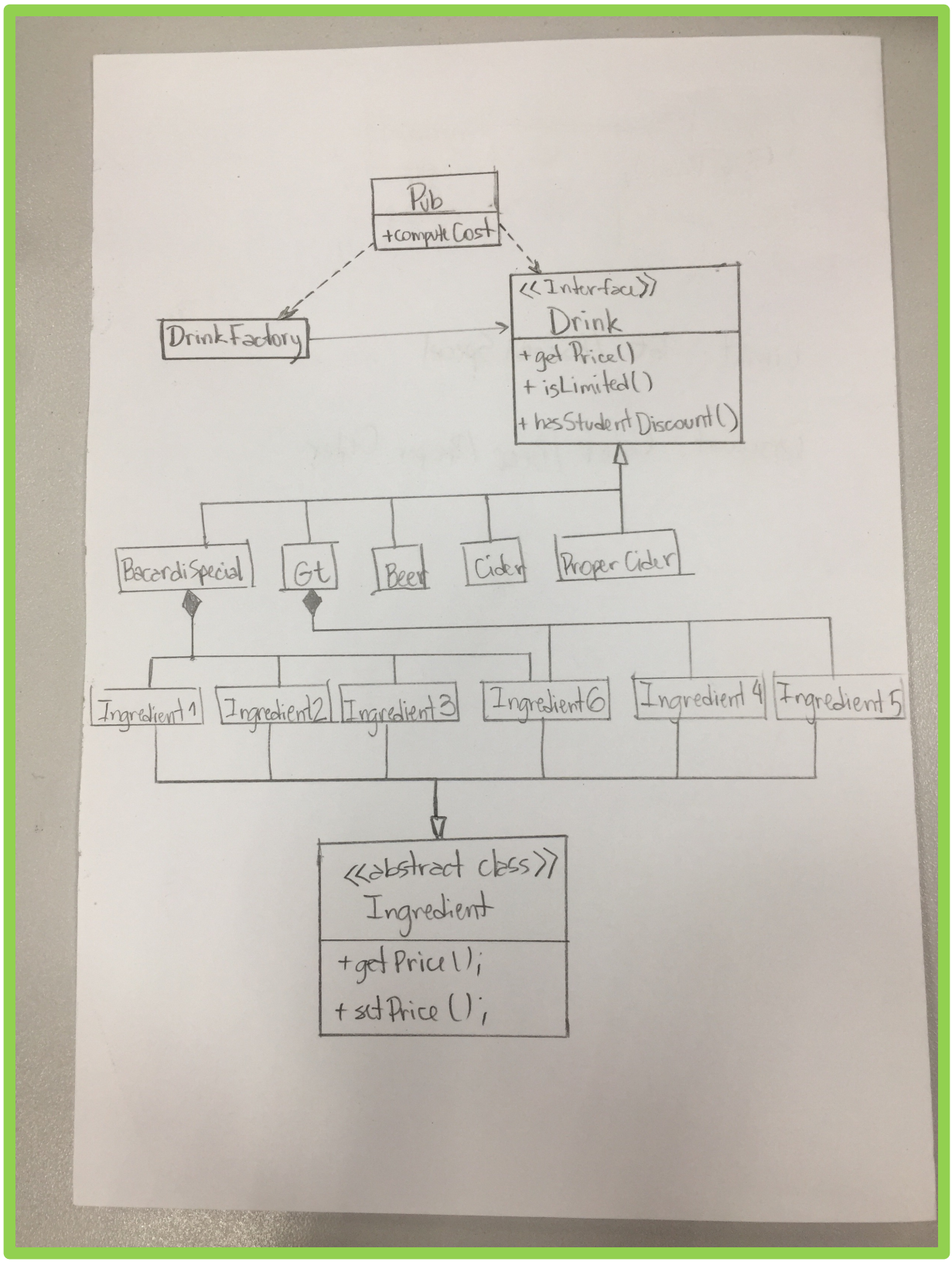


Figura 1. UML do código implementado na questão 3.

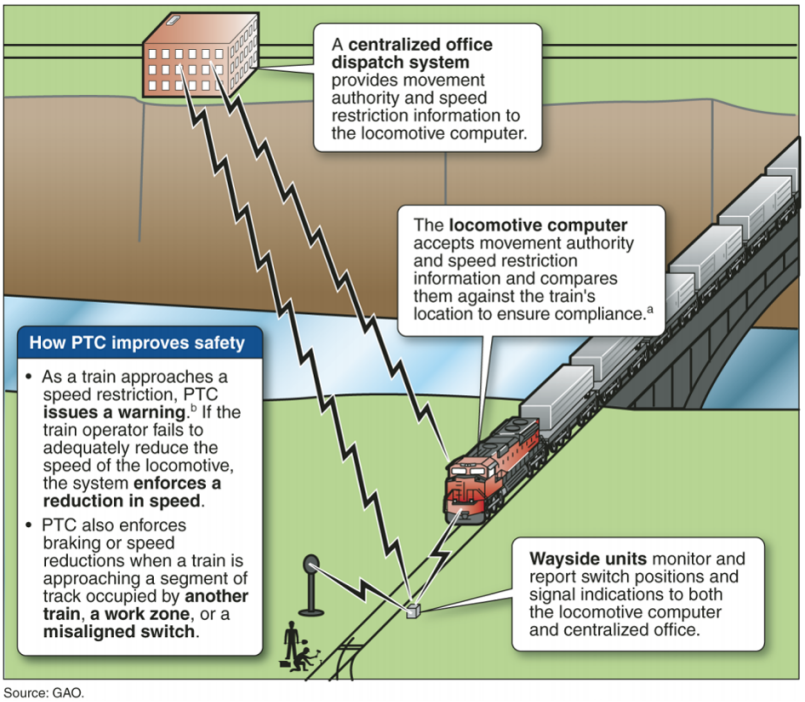
PS: descontos são arredondados para o próximo inteiro maior - *Math.ceil()* resolve o arredondamento.

[IMPLEMENTAÇÃO] – QUESTÃO 4 –CONTROLE POSITIVO DE TRENS.

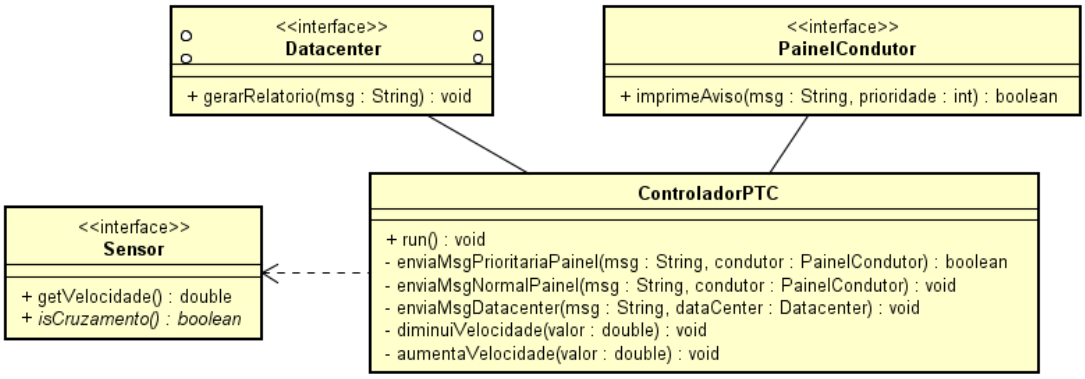
Considere um sistema de Controle positivo de trens (*Positive Train Control - PTC*). Um PTC requer a coleta e a ação em 2 tipos de informações:

a) Dados urgentes que devem ser acionados imediatamente; e
b) Dados enviados para o datacenter para serem mensurados e utilizados posteriormente.

Para esse desenvolvimento, **sensores de trilhos** coletam e registram dados sobre a *rota*, a *velocidade* e as *características de carga dos trens*. Todos os dados passam pela **camada de controle**, onde o software de mensagens e regras de negócios determina o que fazer com os dados. Conforme o trem se aproxima de um cruzamento, as *mensagens para alterar a velocidade* são transmitidas para o **painel do condutor** *com alta prioridade*. *Informações com menos urgência*, sobre velocidade, eficiência de combustível, peso e outras são armazenadas no **datacenter** para serem analisadas. **Caso essas direções urgentes sejam ignoradas**, *ações automáticas entram em ação* no **sistema de bordo do trem** para parar, diminuir ou acelerar a velocidade do mesmo. Colisões são evitadas e os dados são armazenados de maneira segura.



Source: GAO.



O diagrama de classe que implementa o supracitado sistema é apresentado na figura acima. Abaixo é apresentado o código que implementa o ControladorPTC.

```
package Q4.ptc;

import java.util.concurrent.TimeUnit;

public class ControladorPTC {
	private Sensor sensor;
	private Datacenter dataCenter;
	private PainelCondutor painelCond;

	public ControladorPTC(Sensor sensor, Datacenter dataCenter, PainelCondutor painelCond) {
		super();
		this.sensor = sensor;
		this.dataCenter = dataCenter;
		this.painelCond = painelCond;
	}

	public void run() {

		double velocidade = sensor.getVelocidade();
		boolean isCruzamento = sensor.isCruzamento();

		// checa se o trem esta com velocidade acima do permitido no cruzamento
		if (isCruzamento && (velocidade > 100)) {
			boolean result = enviaMsgPrioritariaPainel("Velocidade alta", painelCond);
			if (result == false) {
				diminuiVelocidade(20);
			}
		}

		// checa se o trem esta lento demais no cruzamento
		if (isCruzamento && (velocidade < 20)) {
			boolean result = enviaMsgPrioritariaPainel("Velocidade Baixa", painelCond);
			if (result == false) {
				aumentaVelocidade(20);
			}
		}

		else {
			enviaMsgDatacenter(new Double(velocidade), dataCenter);
			enviaMsgNormalPainel(new Double(velocidade), painelCond);
```

```
                }

        }

        public boolean enviaMsgPrioritariaPainel(String msg, PainelCondutor condutor) {
                boolean result = condutor.imprimirAviso(msg, 1);
                if (result == false) {
                        try {
                                TimeUnit.SECONDS.sleep(10);
                                result = condutor.imprimirAviso(msg, 1);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
                return result;
        }

        public void enviaMsgNormalPainel(Object msg, PainelCondutor condutor) {
                condutor.imprimirAviso(msg.toString(), 1);
        }

        public void enviaMsgDatacenter(Object msg, Datacenter datacenter) {
                datacenter.gerarRelatorio(msg.toString());
        };

        public void diminuiVelocidade(double valor) {
                this.painelCond.diminuiVelocidadeTrem(valor);
        };

        public void aumentaVelocidade(double valor) {
                this.painelCond.aceleraVelocidadeTrem(valor);
        };

}
```

Através do uso de Test Double e do uso do Framework Mokito, resolva as questões abaixo:

a) Teste a inicialização do objeto **ControladorPTC**. **(1.0 PT).**
b) Construa um caso de teste, quando o trem não se encontra em um cruzamento, ou seja, o método ***isCruzamento()*** de **Sensor** retorna falso. Verifique o comportamento se deu certo. **(1.0 PT).**
c) Construa um caso de teste, quando o trem se encontra em um cruzamento e a velocidade é superior 100Km/h, ou seja, o método ***isCruzamento()*** de **Sensor** retorna verdadeiro. Além disso, o usuário localizado no Painel do Condutor deve informar que leu a mensagem, ou seja, o retorno do método `enviaMsgPrioritariaPainel()` deve ser verdadeiro. Verifique o comportamento se deu certo. **(1.0 PT).**
d) Construa um caso de teste, quando o trem se encontra em um cruzamento e a velocidade é inferior a 20Km/h, ou seja, o método ***isCruzamento()*** de **Sensor** retorna verdadeiro. Além disso, o usuário localizado no Painel do Condutor nãp deve confirmar a leitura da mensagem, ou seja, o retorno do método `enviaMsgPrioritariaPainel()` deve ser falso. Verifique o comportamento se deu certo. **(2.0 PT).**

a. **Observação A**: Deve ser usar Test Double nas classes não relacionadas ao comportamento do Controlador.
b. **Observação B**: Na correção será considerado que aderência e pertinência do Test Double selecionado.
c. **Observação C**: Será avaliado a pertinência e cobertura dos casos de testes realizados, ou seja, devem ser construídos casos de testes para cada uma das funcionalidades do CDS apresentadas no cenário acima apresentado.
d. **Observação D**: Um melhor detalhamento do cenário pode ser encontrado em: https://www.forbes.com/sites/hilarybrueck/2015/05/20/how-positive-train-control-works-how-it-could-make-rail-travel-safer/#722452ac7e9d