Instituto Tecnológico de Aeronáutica - ITA

Disciplina: Engenharia de Software (CES-28)

3̊ Laboratório: Mokito

Professor: Inaldo Capistrano Costa

Equipe:

1- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Regras:**

- Em grupo de dois ou três alunos;

- Postar solução no TIDIA até o dia 21/09/2018 às 22Hs.

**PARTE I - ORIENTAÇÕES**

1. Sobre o uso do **Mockito**, veja material postado em conjunto com este Lab.
2. Deve ser submetido no TIDIA (pode ser um link para o GITLAB) com os seguintes entregáveis:
   * 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.

**PARTE II - IMPLEMENTAÇÃO**

# **QUESTÃO 1: REFATORAÇÃO**

# **UM BAR COM MAU CHEIRO.**

1. 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!**

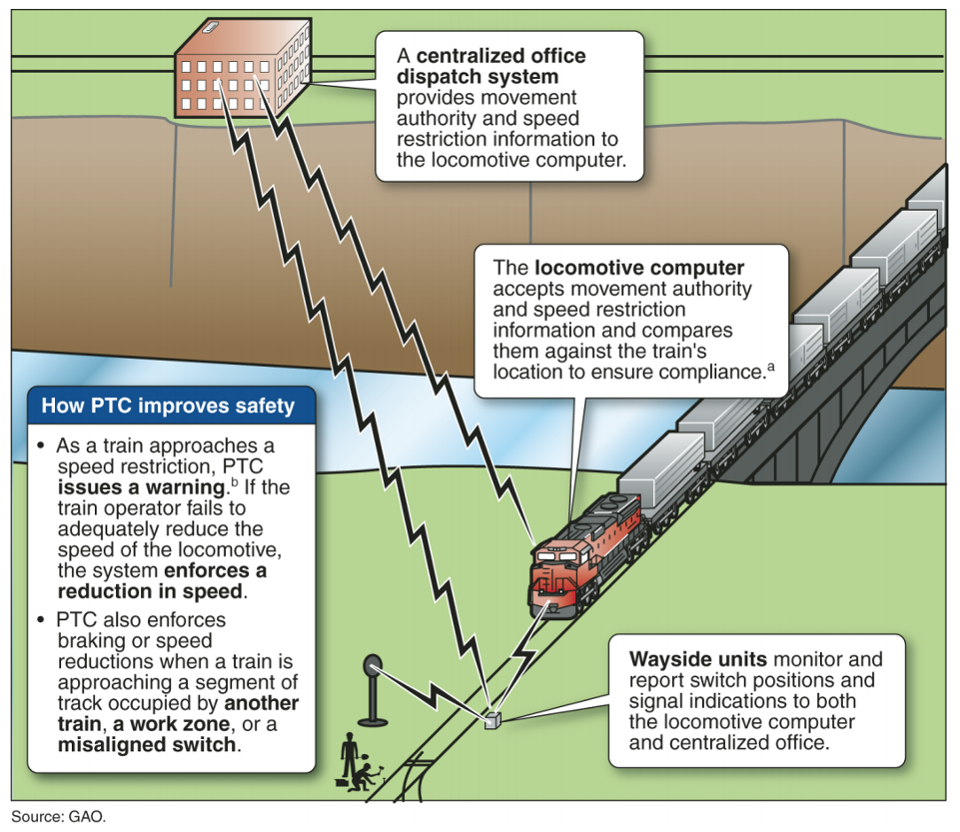
1. Refatore o código, criando novas classes de forma a dividir melhor as responsabilidades segundo a Lei de Deméter.
   * Uma tarefa comum de manutenção deste código seria *incluir e remover ingredientes e drinks no modelo, ou modificar as regras em relação aos já existentes*.

# **QUESTÃO 2: TESTES com MOKITO**

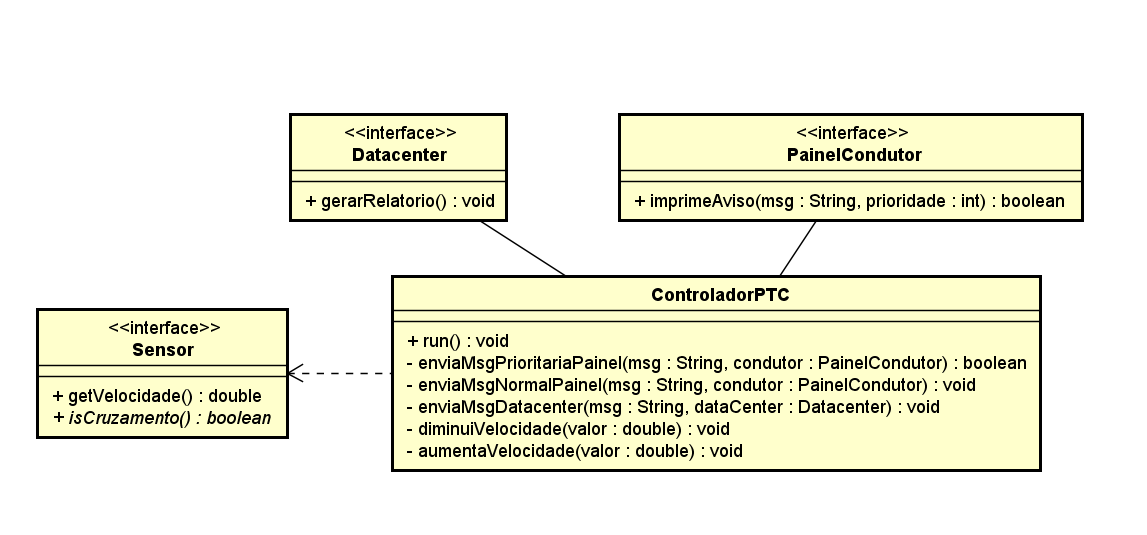
# **CONTROLE POSITIVO DE TRENS.**

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:

1. Dados urgentes que devem ser acionados imediatamente; e
2. 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.



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();  };  **public** **void** diminuiVelocidade(**double** valor) {  **this**.painelCond.diminuiVelocidadeTrem(valor);  };  **public** **void** aumentaVelocidade(**double** valor) {  **this**.painelCond.aceleraVelocidadeTrem(valor);  };  } |

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

1. Teste a inicialização do objeto **ControladorPTC**.
2. 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.
3. 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.
4. 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.

**Observação A**: Deve ser usar Test Double nas classes não relacionadas ao comportamento do Controlador.

**Observação B**: 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