# IMD0040 Trabalho 2

## Introdução

Este trabalho foi projetado para demonstrar seu entendimento no material apresentado pelos capítulo 8 e 9. Você vai entregar o trabalho eletronicamente via SIGAA.

Qualquer método escrito por você deverá ser totalmente comentado utilizando tags javadoc.

## Plágio e Duplicação de Material

O trabalho que você submeter deverá ser somente seu. Iremos rodar programas verificadores em todos os trabalhos submetidos para identificar plágio, e adotar as medidas disciplinares cabíveis quando for o caso. Algumas dicas para evitar problemas de plágio:

- Um dos motivos mais comuns para problemas de plágio em trabalhos de programação é deixar para fazer o trabalho de última hora. Evite isso, e tenha certeza de descobrir o que você tem que fazer (que não significa necessariamente como fazer) o mais cedo o possível. Em seguida, decida o que você precisará fazer para completar o trabalho. Isto provavelmente envolverá alguma leitura e prática de programação. Se estiver em dúvida sobre o que foi pedido pelo trabalho, pergunte ao professor da disciplina.
- Outra razão muito comum é trabalhar em conjunto com outros alunos da disciplina. Não faça trabalhos de programação em conjunto, ou seja, não utilizem um único PC, ou sentem lado a lado, principalmente, digitando código ao mesmo tempo. Discutam as diversas partes do trabalho, mas não submetam o mesmo código.
- Não é aceitável a submissão de código com diferenças em comentários e nomes de variáveis, por exemplo. É muito fácil para nós detectar quando isso for feito, e verificaremos esse caso.
- Nunca deixe outra pessoa ter uma cópia de seu código, não importando o quão desesperado eles possam estar. Sempre aconselhe alguém nesta situação a buscar ajudar com o professor da disciplina.

## A Tarefa

Este trabalho consiste em diversas modificações que serão feitas um projeto existente.

O cenário é de um programa que está em desenvolvimento para uma empresa de transportes. No meio da implementação, o antigo programador teve de abandonar o desenvolvimento, e você foi escolhido para terminar o trabalho. Olhando para o código você perceberá que ele está basicamente OK, mas melhorias poderiam ser feitas através do uso de herança.

O funcionamento básico do programa é que um objeto `TaxiCo` é criado, e um nome fornecido para a empresa. A frota de táxis e ônibus é criada chamando os métodos `addTaxi` e `addShuttle`.

Uma empresa de amostra, com dois táxis e um ônibus, pode ser criada utilizando a classe `Helper`. Utilize esta classe para manipular os objetos criados.

## Taxis e ônibus

Um táxi (`Taxi`) tem uma identidade (ID), um local (inicialmente a base da empresa) e um destino opcional. Se o táxi está livre (não reservado), então ele não tem destino (é nulo). Se ele tiver sido reservado, então ele tem um destino. Se o método `arrived` é chamado em um táxi, então isso faz

com que ele mude seu estado de modo que seu local se torna o que era seu destino, e o táxi torna-se livre novamente.

Os ônibus (`Shuttle`) são diferentes de táxis em que eles se deslocam entre um conjunto fixo de destinos em uma rota circular que inclui a base da empresa. Cada vez que o método `arrived` de ônibus é chamado, sua localização torna-se o que era seu destino, e seu destino muda para o próximo lugar em sua lista de rotas. Cada ônibus tem um ID.

Há claramente algumas semelhanças nas implementações de `Taxi` e Ônibus (`Shuttle`) que sugerem o uso de herança para representá-los. Introduzir a herança é a principal tarefa da avaliação.

## A classe TaxiCo

A classe `TaxiCo` mantém listas separadas de táxis e ônibus. Tem um método chamado `lookup`, que procura um táxi com um determinado ID. O programador original deixou um método semelhante que pode ser usado para ônibus (`Shuttle`). Você perceberá que este método introduziria duplicação de código na classe `TaxiCo`.

## Introduzindo herança no projeto (35 pontos)

As classes `Taxi` e `Shuttle` compartilham alguns atributos - identificação, localização e destino. Elas também têm alguns métodos comuns - getId, getLocation, getDestination, getStatus e setDestination. Tente capturar estes elementos comuns da maneira que você achar apropriada em uma nova classe, `Vehicle`, que torna-se superclasse de tanto `Taxi` e `Shuttle`.

Esta mudança envolve a colocação dos campos e métodos comuns em `Vehicle` e removê-los de `Taxi` e `Shuttle`. Em vez de fazer as mudanças todas de uma vez, será mais seguro mover um campo de cada vez, talvez como descrito abaixo, mas esta é apenas uma sugestão. Cada vez que você fizer uma mudança significativa ao seu código, verifique se recompila e comporta-se como esperado. Você pode usar a classe `Helper` para isso, e adicionar outros testes para a aumentar a robustez de sua implementação.

**Movendo o campo id**

Para mover o campo id você vai precisar para se engajar em um processo chamado refatoração. O objetivo é melhorar as estruturas de classe através da revisão do código existente. Isso implicará em mover algum código de lugar, remover outros pedaços, e fazer pequenas mudanças para outros trechos. No entanto, não estamos buscando no introduzir uma nova funcionalidade por este processo.

Comece criando uma nova classe chamada `Vehicle`. Modifque a classe `Taxi` para indicar que é uma sub-classe de `Vehicle`.

Mova o campo id para `Vehicle`, retirando-o de `Taxi`. Como este campo também, eventualmente, ser herdada pela classe `Shuttle` refatorada, certifique-se de que o campo tem um comentário apropriado uma vez que ele tenha sido movido.

Organize o construtor de `Taxi` para chamar o construtor de `Vehicle` para que um valor de id seja passado de um para o outro.

Mova o método de acesso getID de `Taxi` para `Vehicle`. Certifique-se que o texto do comentário do método é apropriado para um método comum a várias classes.

Como id agora é um campo privado de `Vehicle`, subclasses não pode usar o campo diretamente em seus métodos. `Taxi` deve substituir acessos diretos com chamadas para o método público getID herdado de `Vehicle`. Faça essas alterações em `Taxi` e confira que todas as classes compilem corretamente.

Use a classe `Helper` para ajudar a verificar os resultados de suas alterações.

Uma vez que você moveu com sucesso o campo id para `Vehicle`, você pode fazer uso desse campo na classe `Shuttle`. Faça com que `Shuttle` seja uma subclasse de `Vehicle` e remova todos os elementos de que já não necessita, pois são herdados de Veículo. Você terá que fazer um conjunto de mudanças muito semelhante àquelas que você acabou de fazer a `Taxi`.

**Movendo o campo destination**

Assim que você tiver mudado com sucesso o campo id para a classe de veículos você pode fazer o mesmo com o campo de destino (`destination`). O processo será semelhante e você deverá provavelmente trabalhar em `Taxi` e `Shuttle` ao mesmo tempo.

Inicie de um modo semelhante como com o id - mova o campo de destino, juntamente de seus métodos de acesso e modificador para `Vehicle`. Substitua nas subclasses os acessos a estes campos com chamadas aos métodos de acesso e modificador.

Certifique-se de que tudo compila e funciona como seria de esperar. Melhore a classe `Helper` com métodos novos que apoiam estes testes de regressão.

**Movendo o campo location**

Usando a experiência adquirida com as tarefas acima, mova o campo `location` para a classe `Vehicle`.

Utilize a classe `Helper` para verificar se tudo está funcionando como antes. Adicione outros testes à classe `Helper` para aumentar a confiança de que tudo está funcionando como deveria.

## Introduzindo polimorfismo (25 pontos)

A classe `TaxiCo` mantém listas separadas para objetos `Taxi` e `Shuttle`. Substitua essas duas listas com uma única lista de objetos `Vehicle`. Perceba que com isso o método `lookup` retornará objetos `Vehicle` em vez de `Taxi`. Outra modificação será no método `showStatus`, que poderá ser implementado com um simples for-each em vez de dois.

## Informação de status melhorada (10 pontos)

O método `getStatus` original da classe `Taxi` imprime `null` se um taxi estiver livre. Se sua versão da classe `Vehicle` faz o mesmo, inclua um teste para verificar se o destino é `null` ou não, e imprima uma mensagem mais significativa caso seja.

## Desafio (30 pontos)

Um novo método foi solicitado para a classe TaxiCo. A ideia é que um cliente ligou para a empresa de táxi e quer pegar ou um ônibus ou um táxi para um determinado destino. O destino é passado como um parâmetro para o método.

Este método deve retornar um veículo que é:

- um Shuttle cujo próxima parada (ou seja, destino) é onde o cliente quer ir; ou
- um Taxi, que esteja livre (ou seja, tem um destino nulo).

Se não houver transporte disponível, e nenhum táxi estiver livre, então este método deve retornar `null`.

Um cliente vai sempre preferir um `Shuttle` porque eles são mais baratos, então se existir um `Shuttle` que atenda ao cliente, este deve ser devolvido.

Observe que há um problema aqui. É importante ser capaz de distinguir entre um ônibus cujo próximo destino é o necessário, e um táxi que já está reservado para ir lá. Um táxi reservado não é bom para o cliente, embora esteja indo para onde ele quer ir. A fim de ser capaz de identificar se um determinado veículo é um táxi ou um ônibus você vai precisar usar o operador instanceof.
Além disso, podemos usar um cast para se recuperar o subtipo e acessar a funcionalidade completa do objeto. O exemplo a seguir mostra como esses recursos são usados em uma hierarquia onde Cat e Dog são subclasses de Animal:

```
public void identify(Animal whatAmI)
{
     if(whatAmI instanceof Dog) {
          System.out.println("You are a dog.");
          // Use a cast to access its Dog characteristics.
          Dog fido = (Dog) whatAmI;
          fido.bark();
     }
     else if(whatAmI instanceof Cat) {
          System.out.println("You are a cat.");
          // Use a cast to access its Cat characteristics.
          Cat tiger = (Cat) whatAmI;
          tiger.miaow();
     }
     else {
          System.out.println("I don'tknow what you are.");
     }
}
```

Teste seu código considerando todas os casos possíveis.

## Finalmente

Antes de submeter seu trabalho, teste exaustivamente a classe inteira para se certificar de que nada do que foi acrescentado recentemente quebrou algo adicionado anteriormente. Se você não conseguir completar a classe - mesmo se você não conseguir compilar – submeta o que você tem, pois é provável que você vai ter pelo menos algum ponto por isso.