MC302 – Programação Orientada a Objetos 12/04/2016
Instituto de Computação – UNICAMP
Profa.: Cecília Mary Fischer Rubira

Lab07: Classes Abstratas e Concretas

INSTRUÇÕES INICIAIS:

Nota: Os exercícios devem ser executados na ordem apresentada, pois o nível de dificuldade é crescente.

1. Obtenha o arquivo **lab07.zip** (disponível no ensino aberto);
2. Descompacte esse arquivo no seu diretório de trabalho (workspace do Eclipse);
3. Crie um novo projeto Java no Eclipse. Clicar em File > New > Java Project, manter o lugar padrão e escrever como nome do projeto exatamente o nome da pasta descompactada.
4. Ao final do lab, compacte a **pasta raiz do projeto**, em um arquivo com extensão .**zip**, e suba no ensino aberto no seu portfólio*, disponibilizando-o para os formadores. Suba apenas um arquivo. **Outros formatos não serão avaliados, implicando em nota 0 (zero) na atividade.**
5. Suba um arquivo de texto com a resposta para todas as questões conceituais **dentro** da pasta compactada. Serão aceitos os formatos .**txt** e **pdf**. **Coloque seu nome e RA neste arquivo.**

***obs:** criar um item no portfólio, com nome Lab07, com **somente** o arquivo lab07.zip. Não criem pastas no portfólio.*

As questões de 1 a 9 devem ser feitas em sala e entregues até às 19h. A questão 10 pode ser entregue até às 23h59 de quinta-feira. Para isso, basta editar o item atual e substituir o zip com a atividade completa.

Pacote *br.unicamp.ic.mc302.*`veiculos`:

1. Abra os arquivos `FilaVeiculo.java`, `Veiculo.java`, `Carro.java`, `Caminhao.Java` e `Inicial.java`. Compile e execute a classe `Inicial`.

(i) adicione no final da lista um objeto do tipo `Veiculo;`
(ii) adicione o modificador abstract à definição da classe `Veiculo`. Compile-a e execute novamente a classe `Inicial`. Você sabe explicar o porquê do erro? Desfaça o item (i) antes de passar para o próximo item.

2. Defina em `Veiculo` a seguinte operação: public abstract boolean ligar( );

Recompile as classes `Caminhao` e `Carro`. O que aconteceu?

3. Adicione o modificador abstract à definição da classe `Carro` e `Caminhao` e tente compilá-la. Por que o programa ainda não executa? Remova o modificador abstract das classes `Carro` e `Caminhao`, antes de passar para o próximo item.

4. Redefina a operação ligar( ) nas classes `Carro` e `Caminhao`. Essa operação deve imprimir na tela uma mensagem informando que o carro ou caminhão foi ligado corretamente. Abra a classe `ExemploLigar.java`, compile os arquivos e execute essa classe.

5. Implemente uma operação ligarTodos( ) na classe `FilaVeiculo`. Essa operação deve funcionar de maneira muito similar a mostraFila( ) mas, ao invés de enviar a mensagem mostra( ) de cada veículo, envia a mensagem ligar( ). Modifique a operação main( ) da classe `Inicial` para que envie a mensagem ligarTodos( ) para o objeto fila. Compile as classes e execute a classe `Inicial`.

6. Abra o arquivo `Fila.java`. Essa classe implementa o conceito de polimorfismo paramétrico. Modifique a classe `Inicial` de forma a utilizar a fila paramétrica com objetos `Veículo`. Imprima a fila, execute e comente os resultados. Para imprimir as informações de `Carro` e `Veículo` (e não somente o identificador do objeto), pode-se sobrescrever o método `toString()` da classe `Object`. Sobrescreva esse método nas classes `Carro` e `Caminhao` de forma que a impressão da fila paramétrica seja similar à impressão da fila original.

7. Abra o arquivo `Aviao.java`. Apesar de representar um tipo de veículo, a classe `Aviao` não apresenta todas as características definidas pela classe `Veiculo`. Por exemplo: não faz sentido um avião ter um atributo chamado placa. Use o conceito de classes abstratas para reestruturar a hierarquia `Veiculo`.

Pacote *br.unicamp.ic.mc302.figuras*:

8. Abra o Arquivo `ExemploFiguras.java`. Crie um objeto do tipo `Circulo` e use o recurso de auto-completar do Eclipse (digite o nome do objeto, em seguida um ponto e então *Ctrl+Space*). A interface do objeto é listada. Então explique a origem de todas as operações da interface pública do objeto e qual o tipo de polimorfismo aplicado e porquê.

9. Abra os arquivos `Figura.java`, `Circulo.java`, `Linha.java` e `ExemploFiguras.java`. Compile os arquivos, execute a classe `ExemploFiguras` e observe os resultados. Modifique as operações esconder( ) e mostrar( ) da classe `Figura` para que deixem de ser abstratas e informem uma mensagem com o nome da operação, seguida da mensagem "da classe Figura.". Recompile os arquivos e execute novamente a classe `ExemploFiguras`. Em seguida, comente a redefinição do método esconder() na classe `Linha` e comente a redefinição do método mostrar() na classe `Circulo`. Recompile os arquivos mais uma vez e execute novamente a classe `ExemploFiguras`. Explique o que aconteceu.

---

10. Considere o enunciado a seguir definido no Lab03:

"Uma empresa de ônibus contratou você para criar um programa Java para controlar reservas de assentos em um ônibus. Cada ônibus possui 40 lugares.

0000000000

0000000000

00000*0000

0000000000

A poltrona é referida pela coluna e pela linha. Por exemplo, a poltrona marcada com * é a poltrona da fileira 6 assento B. As fileiras são marcadas de 1, 2. 3 .. 10, e os assentos são A,B,C,D. O objetivo do seu programa é verificar se a poltrona desejada está ou não ocupada.

Se estiver ocupada, imprima a mensagem ``Poltrona ocupada! Tente outra." na tela, mostre a alocação atual do ônibus e peça outra poltrona. Se a poltrona estiver vaga imprima a mensagem ``Reserva efetuada!" na tela, mostre a alocação atual do ônibus e peça outra poltrona."

Use o conceito de classes abstratas para implementar um "refactoring" da sua solução incluindo os seguintes requisitos funcionais:

(a) a empresa tem 2 tipos de ônibus: ônibus leito e ônibus convencional. Um ônibus leito tem menos fileiras (5 ao invés de 10) e a passagem custa mais caro.

(b) um ônibus convencional possui assentos diferenciados nas 2 primeiras fileiras com mais espaço que custam mais caro para reservar.

(c) implemente uma operação que calcula o valor total das vendas de passagem para um determinado ônibus.

Crie o pacote *br.unicamp.ic.mc302.onibus* para implementar sua solução, todas as classes que julgar necessárias, e uma classe `ExemploOnibus`, com um método *main* para exemplificar sua solução, imprimindo na tela exemplos comprovando que os requisitos foram atendidos.