Universidade Federal do Rio Grande do Norte
Instituto Metropole Digital

IMD0040 - Linguagem de Programação 2

Aula03 - Lista de exercícios

- Esta lista de exercício é composta por 4 questões. Sendo uma fácil, uma média, uma difícil e uma de desafio.
- O aluno poderá escolher entre realizar as 3 primeiras e somar 100% ou apenas o desafio para conseguir os 100%.
- NÃO HAVERÁ ACUMULO, portanto, se decidir fazer as 4, não ganhará mais por isso (mas terá aprendido muito mais).
- Não se assustem com o tamanho dos textos.

(Fácil - 20%) Celular

Imagine que você é parte de uma equipe trabalhando em um sistema de software para uma empresa de telefone celular. Você foi convidado para escrever uma classe cujas instâncias podem armazenar detalhes de valores de crédito de um cliente de telefone pré-pago. A classe vai precisar de um campo para armazenar uma quantidade de dinheiro (em números inteiros de centavos) e métodos que permitem verificar e alterar o saldo de um cliente - tanto a adição de dinheiro quando um cliente compra créditos, e a retirada de dinheiro quando o cliente usa o seu telefone. Você vai perceber que a Classe TicketMachine no capítulo 2 do livro texto tem um monte de funcionalidades semelhantes, para que você possa usá-la para ajudar a escrever sua própria classe.

É importante acrescentar alguma verificação no construtor e métodos da classe para se assegurar de que as operações ilegais no saldo não possam ser realizadas, por exemplo, definindo o saldo inicial para um valor negativo, carregando créditos com um valor negativo; gastar mais dinheiro do que é permitido, e assim por diante. Detalhes completos dos requisitos são dadas nas fases seguintes.

Ao escrever software é essencial garantir que seu código atenda aos requisitos do mesmo. Esta garantia só pode ser adquirida através de testes exaustivos no código. Você deve, portanto, verificar o seu código extensivamente durante o desenvolvimento, assim como quando você tiver terminado. Só porque o seu código compila não significa que ele funciona corretamente. Teste seu código, portanto, através da criação de objetos e de chamadas aos seus métodos com uma vasta gama de valores de parâmetros. Embora isso pode parecer tedioso às vezes, muitas vezes você vai revelar erros que você não tinha imaginado que existia.

Criando uma nova classe no BlueJ

Inicie um novo projeto no BlueJ usando a opção Project/New project. Escolha um nome e localização para o projeto. Selecione o botão New Class para começar a criar uma classe. Dê a sua classe o nome Credit. O BlueJ criará um esboço de definição de classe para você na área principal do projeto.

O BlueJ lhe deu exemplos de campo de amostra, construtor e método, mas estes não são relevantes para este trabalho e é provavelmente mais fácil eliminar tudo dentro do corpo da classe.

Adicionando um campo

Objetos da classe Credit precisam de um campo inteiro para armazenar o número de créditos do cliente em centavos. Adicione um campo com um nome apropriado a sua escolha (mas evite nomes sem sentido, uma única letra como c, X, Y, etc.). Não se esqueça de incluir um modificador de visibilidade adequada para o campo. Se você não tem certeza o que é isso, olhe para os campos em alguns dos exemplos de classes que você já viu e modele seu campo baseado nesses exemplos.

Compile seu código e corrija os erros que você encontrar, como ponto e vírgula esquecidos ou nomes de tipo escritos de maneira errada. Uma boa regra a lembrar é que todas as palavras-chave de Java (como classe, int, boolean, público, privado, etc) consistem inteiramente em letras minúsculas. Mesmo que a classe não tenha métodos, você ainda pode criar objetos a partir dela. Faça isso, e use o inspetor para mostrar o estado de um objeto. Você deverá ver o campo que você definiu com um valor de zero. Neste ponto, você não será capaz de mudar isso (default) valor porque você não tem métodos para alterá-los (métodos modificadores).

Construtor

Adicione um construtor para a classe. O construtor não deve ter nenhum parâmetro. Este construtor deve inicializar o campo para ter um valor de 1000 (equivalente a 10 reais de crédito). Compile sua classe, corrija os erros e, em seguida, crie um objeto e verifique se o campo tem um valor de 1000 logo que o objeto é criado.

Adicione um segundo construtor que receba um único parâmetro inteiro, cujo valor deve ser utilizado para inicializar o campo. Mais uma vez, compile a classe, corrija os erros, crie alguns objetos usando o novo construtor e valores de parâmetros diferentes para verificar se as coisas estão funcionando como deveriam. A primeira coisa a se olhar é se o valor do crédito que você vê quando você inspeciona um objeto corresponde ao valor do parâmetro que você passou como parâmetro para o construtor.

O tipo de ciclo que temos seguido até este ponto - adicionar um pouco de código, compilar; corrigir erros, compilar, criar objetos e teste - é uma prática fundamental que você deve seguir sempre que você está escrevendo software, não importa se você é um novato ou um especialista. Uma das vantagens de fazer as coisas desta maneira é que você consegue pegar erros muito rapidamente, e vai saber mais ou menos onde procurar o

problema, porque ele provavelmente vai ter algo a ver com o código mais recentemente adicionado ou alterado. Assumiremos que você vai seguir este ciclo de agora em diante, e não ficar repetindo que deve fazê-lo.

Métodos de Acesso

Adicione um método de acesso à sua classe que devolverá o montante de crédito na conta do usuário. Métodos de acesso, muitas vezes têm nomes começando com 'get' seguido pelo nome do campo que estão associadas, por exemplo, por um getTotal para um campo chamado total, getAmount para um campo chamado amount, getDate, etc. Dê ao seu método uma visibilidade pública e um tipo de retorno, que é o mesmo que o tipo do campo. O método não deve receber nenhum parâmetro. No corpo do método deve haver uma instrução de retorno para retornar o valor do campo.

Métodos modificadores

Escreva um método modificador que receba um único parâmetro que é utilizado para se alterar o valor do campo. Nomes de métodos modificadores são frequentemente chamados de 'set' alguma coisa, com o resto do nome correspondendo ao nome do campo (por exemplo, setTime para um campo chamado time, setVisible, etc.). Dê a visibilidade pública a seu método e um tipo de retorno void. O corpo do método deve simplesmente atribuir o valor do parâmetro para a variável do campo. Nesta fase, não se preocupe em verificar se o valor do parâmetro é sensato, basta fazer a atribuição ao campo.

Um método para colocar créditos

Adicione um método chamado de topUp (recarga em inglês) que irá ser muito semelhante ao método modificador. No entanto, em vez de designar o valor do parâmetro para o campo, este método deve adicionar o valor do parâmetro para o campo, como um processo de recarga normal. Mais uma vez, não se preocupe sobre a verificação de um valor de parâmetro sensível, basta fazer a adição. Verifique cuidadosamente se você implementou este método corretamente, chamando-o várias vezes com valores de parâmetros diferentes em um objeto. Utilize o inspetor de objeto para verificar.

Um método para consumir créditos

Escreva um método que subtrai o valor no seu parâmetro do campo. Este método vai fazer o oposto do método topUp. Embora, eventualmente, exigiremos que o valor do parâmetro seja não-negativo, não se preocupe com isto nesse momento.

Revisão

Neste ponto, você deve ter uma classe que define um campo para armazenar uma quantidade de centavos, dois construtores, um método de acesso para esse campo, e três métodos que são capazes de alterar o valor do campo (por atribuição direta, a adição e

subtração). Na prática, isso vai ser muito semelhante em funcionalidade para a classe TicketMachine no projeto naive-ticket-machine.

Usando instruções condicionais

Agora é hora de adicionar algumas funcionalidades de verificação para que o campo seja protegido contra alterações de maneiras inapropriada ser fixados ou alterados de forma inapropriada. Para isso, você precisará usar-se declarações. A sintaxe básica de uma instrução if é:

```
if(testa-alguma-condição) {
        executa essa parte se o resultado do teste for true
}
else {
        executa essa parte se o resultado do teste for false
}
```

Por exemplo:

```
// Verifica se tem créditos suficiente para enviar uma mensagem de texto.
if(credit < 10) {
        System.out.println("Você não tem créditos suficientes.");
}
else {
        // Reduz do crédito o custo da mensagem de texto.
        credit = credit – 10;
}
```

Deixando a Classe mais Robusta

Adicione testes (instruções condicionais) para todos os construtores e métodos que alteram o campo devido a um valor de parâmetro. Estes só devem mudar o campo, se o valor do parâmetro não for negativo. Se um valor de parâmetro negativo é utilizado, esses métodos devem imprimir uma mensagem de erro apropriada e não alterar o campo. A mensagem de erro deve identificar tanto que o método foi chamado incorretamente, e o valor do parâmetro incorreto que foi usado. Por exemplo:

```
topUp foi chamado com um valor negativo : -1000
```

Se o construtor for chamado com um valor de parâmetro negativo, imprima uma mensagem de erro e atribua zero ao campo.

Quando você estiver testando, se certifique de testar todos os métodos com parâmetros positivos e negativos, e tenha certeza de que o campo não é alterado se um valor inválido de parâmetro for utilizado.

(Médio - 30%) Heater

Imagine que você precisa criar o software de um aquecedor que, por segurança, trabalhe com limites de temperatura.

Abra o Bluej e crie um projeto chamado "Aquecedor", o objetivo deste projeto é construir uma aplicação Java que consiga gerenciar a temperatura de um determinado ambiente.

A classe Heater

Crie a classe Heater, contendo um único campo inteiro chamado temperature e um construtor que não recebe parâmetro e inicia a temperatura com o valor 15. Defina os métodos Warmer e Cooler. Cujo o efeito é, respectivamente, aumentar e diminuir o valor da temperatura em 5 graus por vez. Defina um método de acesso para retornar o valor de temperature.

Melhorando a classe

Certifique-se de que todos os passos anteriores estão funcionando perfeitamente. Agora, defina três novos campos do tipo inteiro: min, max e increment. Os valores de min e de max devem ser configurados por parâmetros passados para o construtor. O valor de increment deve ser iniciado como 5. Modifique as definições dos métodos Warmer e Cooler de modo que utilizem o valor de increment em vez de um valor fixo. Verifique se tudo funciona.

Modifique o método Warmer de modo que ele não permita que temperature seja elevada à valores maiores do que a max. De maneira semelhante, modifique Cooler. Faça de tal modo que não seja possível diminuir temperature para valores menores do que min. Verifique se a classe funciona adequadamente.

Finalizando a classe

Acrescente um método setIncrement, que recebe um único parâmetro inteiro e o utiliza para configurar o valor de increment. Teste sua classe e, em seguida, descubra se sua classe ainda funciona caso você passe um valor negativo para setIncrement. Modifique o método para impedir tal possibilidade.

(Díficil - 50%) Mostrador de relógio

Usando como base o projeto Clock-display do capítulo 3, mude um relógio de 24 horas para um de 12 horas. Em um relógio de 12 horas as horas depois da meia-noite e depois do meio-dia não são mostradas como 00:30, mas como 12:30. Assim, o mostrador do minuto mostra valores de 0 a 59, enquanto o mostrador da hora mostra valores de 1 a 12. Lembre-se que é necessário ainda, incluir a informação AM/PM.

Dica: Há (pelo menos) duas maneiras de fazer um relógio de 12 horas.

(Desafio - 100%) Simulando um Carro

A tarefa

Você foi contratado por uma montadora de carros e precisa especificar um novo modelo computacional para a realização de experimentos.

Modelando

No BlueJ, crie um novo projeto (como o nome que você desejar). O carro contém motor, 4 rodas, 2 portas, 1 tanque de combustível, e etc (você é livre para adicionar quaisquer outras classes que desejar). Esses itens correspondem aos campos da classe Carro. Assim para cada um desses itens, crie uma classe, que será utilizada como tipo para os campos da classe Carro.

Os campos

Seu carro e suas classes auxiliares deverão ser especificados. Defina campos para cada classe de acordo com as instruções a seguir:

O motor contém campos para representar as seguintes informações: Tipo de combustível: álcool, gasolina ou ambos (flex); Cilindradas: 1.0, 1.4, 1.6; Se está ligado ou desligado;

Uma roda armazena a informação de tamanho do aro (um inteiro) e a pressão de seu pneu: um inteiro que vai de 0 a 50.

Cada porta armazena a informação se está aberta ou fechada.

O tanque possui um tamanho máximo (em litros), assim como um valor que indica a quantidade de combustível no momento.

Além das informações contidas nas classes que compõem um carro, a classe Carro contém também campos para armazenar o Número de marchas (sem contar a ré), sua cor, e sua velocidade atual.

Os Métodos

O construtor deve inicializar todos os campos (definidos pelo usuário/projetista). É preciso criar métodos de acesso (gets) e métodos de alteração (sets) para possíveis alterações no veículo.

O carro deve ligar, desligar, acelerar e frear. O usuário deve setar a velocidade do veículo através do método acelerar e quanto cada freada vai reduzir dessa velocidade no método frear (ou seja, ambos os métodos tem valores como parâmetros).

Seu carro também pode ser abastecido com uma determinada quantidade de combustível. Cuidado para não permitir que o tanque transborde.

Melhorando a classe

Um carro tem um painel, por que não criar um método de visualizar painel, que imprima na tela as principais informações do veículo (velocidade, km rodados, km/L na atual velocidade, farol ligado/desligado etc.)

Certifique-se de que você não pode desligar seu carro enquanto ele estiver em movimento e nem de que ele se movimente com velocidade negativa.

Limite as velocidades de acordo com a marcha (pense na criação de um método passar marcha e reduzir marcha, bem como suas limitações).

A parte interessante: criando uma simulação simples.

Pense em sua modelagem como uma representação de um carro como na vida real: carros andam, fream, mudam de marcha, engatam a ré, buzinam etc. Existem diversas limitações que precisam ser levadas em consideração em um carro em movimento. Logo, alguns métodos precisam ter verificações importantes para o funcionamento perfeito do carro.

Tente acrescentar uma funcionalidade que calcule consumo de combustível de acordo com aceleração colocada no seu carro (a tantos km/h o carro faz X km/L). Por enquanto, não represente a passagem de tempo em sua simulação. Desse modo, isso pode ser feito, por exemplo, sempre que o carro for acelerado (sempre que se acelerar a X km/h, retira-se y L do tanque de combustível).

Faça as verificações necessárias e deixe sua classe segura. Acrescente campos e métodos, se necessário, para aproximar seu objeto da realidade.

*Dica: Lembre-se de criar métodos gets para ter acesso aos campos e ao estado atual do carro.