ADS – IFPB – Campus Monteiro

LISTA SEMANAL – PROGRAMAÇÃO II - PROGRAMAÇÃO ORIENTADA A OBJETOS  
Prof. Cleyton Caetano de Souza

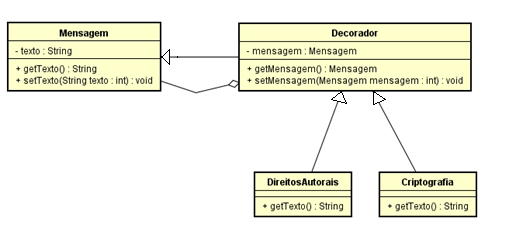
**Observação:** Essa lista pode ser feita em dupla e, fazendo todas as questões, ela pode entregar ela impressa.

Semana 11

**Projeto Mensageiro**

1. **[Autoria: Yaggo (Monitor)]** Considere um sistema de uma empresa para envio de mensagens. Crie a classe Mensageiro, com o método prepararMensagem, que recebe uma String, com a mensagem­ a ser enviada e outra com destino (“interno” ou “externo”) e retorna um objeto Mensagem. A classe Mensagem possuíra um único atributo chamado texto e seus respetivos gets e sets.

2. **[Autoria: Yaggo (Monitor)]** Todas as mensagens que são enviadas para fora da empresa devem conter os direitos autorais da empresa e serem criptografadas. Para mensagens enviadas internamente, não é necessário incluir os direitos autorais, nem criptografia. A decisão de adicionar ou não essas informações (direitos autorais e criptografia) devem ser “tomadas” em tempo de execução. *Pra isso, vamos fazer uso do PADRAO DECORATOR, como ilustrado no diagrama de classe abaixo. Se o destino for externo, decore (isso significa combine) a mensagem com criptografia e com direitos autorais. O método getTexto de mensagem retorna o texto sem mudanças.* O método getTexto de DireitosAutorais, adiciona “... [Direitos Reservados].” ao final da mensagem (dica: você pode usar o método concat() de String). O método getTexto de Criptografia, adicione ‘\*’ entre as palavras que compõem a mensagem. (dica: você pode usar o método replace() de String). Altere o método do Mensageiro para decorar a mensagem, de acordo com o destino passado como parâmetro.



3. **[Autoria: Yaggo (Monitor)]** Crie um programa em que você testará o Mensageiro. Leia a mensagem do usuário, bem como o destino da mensagem (interno ou externo). Em seguida, use o Mensageiro para preparar um objeto mensagem. Exiba no console o objeto mensagem criado.

**Uso de Interfaces**

4. Crie uma interface chamada ClasseComparavel; essa interface possuirá o método comparacao, que receberá um Object como entrada e retornará um valor do tipo inteiro. Crie uma classe chamada MinhaData, com os atributos: dia, mês e ano (todos inteiros) – essa classe possuirá um construtor para receber o valor desses três parâmetros. Faça a classe MinhaData implementar a interface ClasseComparavel. Escreva o método comparacao de MinhaData: a primeira coisa que ele faz é pegar o parâmetro de entrada (object) e “convertê-lo” em um objeto MinhaData – fazendo downcasting. O método então compara o parâmetro de entrada com as propriedades da própria classe: (1) se a data passada como parâmetro for igual a do objeto que chamada o método, ele retorna zero – significando que as duas datas são iguais; (2) se a data passada como parâmetro for anterior, ele retorna um – significando que a data que chama o método vem depois da passada como parâmetro; (3) se a data passada como parâmetro for posterior; ele retorna menos um – significando que a data que chama o método é anterior a que foi passada como parâmetro. Crie um programa em que você lerá duas datas no formato DD/MM/YYYY. Use o método Split de String para partir as entradas e, em seguida, instancie dois objetos MinhaData, compare os dois e exiba-os em ordem crescente.

**Projeto Novos e Usados**

5. Crie a classe abstrata Carro, que possui os atributos: modelo e preço. Crie uma classe CarroNovo, que herda de **Carro** e possui um atributo chamado “*percentual de acréscimo*”. Crie os métodos de acesso e sobrescreva o getPreço para retornar o preço do carro acrescido desse percentual (o percentual é fixo, deve ser setado no construtor com o valor 10%). Crie uma classe CarroUsado, que herda de **Carro** e possui um atributo chamado “percentual de desconto”. Crie métodos de acesso e sobrescreva o getPreço para retornar o preço decrescido desse percentual de desconto **(o percentual é fixo, deve ser setado no construtor com o valor 10%)**. Adicione a classe abstrata Carro um construtor que recebe o preço **e o modelo** do Carro (isso vai impactar nas subclasses). Sobreescreva o toString das subclasses para retornar o tipo do carro seguido do modelo e do preço original sem o acréscimo/desconto. Crie um programa em que você pergunta ao usuário qual o preço do carro que ele está comprando e que tipo de carro será (novo ou usado). Ao final, exiba o preço que o usuário pagará no carro (**importante: não deve haver outros métodos para calcular o preço ou retornar ele, apenas o getPreço**).

6. Na questão anterior, a classe Carro poderia ser uma interface? Explique a sua resposta.

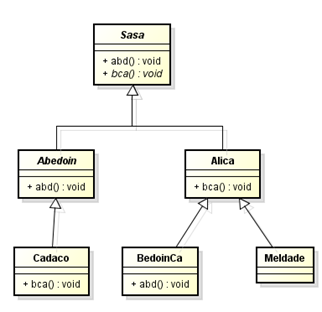
7. Em geral, os programadores costumam privilegiar o uso de interfaces sobre o uso de classes, quando se refere à herança. Pesquise e disserte por que isso acontece (10 linhas no mínimo). Inclua na sua resposta as referências para os conteúdos que você leu que inspiraram sua resposta (caso queira, escolha alguma comunidade de desenvolvedores – pode ser algum grupo do Facebook ou o Stackoverflow ou o GUJ – e pergunte nessa comunidade por que, em Java, se deve priorizar o uso de interfaces, ao invés de herança com classes).

8. Pesquise e explique como outras linguagens orientadas a objetos resolvem o problema do diamante para utilizar herança múltipla.

9. O que acontecerá quando você tentar compilar e executar o seguinte código:

|  |  |  |
| --- | --- | --- |
| I) |  | |
| public class A {  A() {  System.out.println(“A”);  }  }  public class B extends A {  B() {  this(4);  super();  System.out.println(“B”);  }  B(int i) {  System.out.println(“C”);  } } | | Selecione uma alternativa:   * 1. Não compila   2. Imprime “ABC”   3. Imprime “ACB”   4. Imprime “CAB” |
| II) |  | |
| public class Base {  private void aMethod(int iBase) {  System.out.println(“base”);  }  }  public class Over extends Base {  public static void main(String[] args) {  Base o = new Over();  int iBase = 0;  o.aMethod(iBase);  }  public void aMethod(int Over) {  System.out.println(“Over”);  }  } | | Selecione uma alternativa:   * 1. Erro de compilação   2. Imprime “Base”   3. Imprime “Over”   4. ClassCastException |
| III) |  | |
| public class A {  public A(int a) {  System.out.print(“a”);  }  }  public class B {  public B(int b) {  System.out.print(“b”);  }  public static void main(String[] args) {  A a = new B(10);  }  } | | Selecione uma alternativa:   1. ab 2. b 3. ba 4. Erro de compilação |

10. Dado o diagrama de classes abaixo, identifique no método fazerAlgo, onde está a implementado cada método chamado em cada caso.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a) | public void fazerAlgo() {     Sasa as = new Meldade();     as.bca();  } | b) | public void fazerAlgo() {     Sasa as = new BedoinCa();     as.abd();  } | c) | public void fazerAlgo() {     Abedoin jj = new Cadaco();     jj.abd();  } |
| d) | public void fazerAlgo() {     Abedoin jj = new Cadaco();     jj.bca();  } | e) | public void fazerAlgo() {     Meldade m = new Meldade();     m.abd();  } | f) | public void fazerAlgo() {     Alica a = new Alica();     a.bca();  } |
| g) | public void fazerAlgo() {     Sasa sasa = new Meldade();     sasa.abd();  } | h) | public void fazerAlgo() {  BedoinCa b = new BedoinCa();  b.bca();  } | i) | public void fazerAlgo() {     Sasa c = new Cadaco();     c.abd();  } |