Computação Orientada a Objetos

Lista de Exercícios

1. Explique como cada um dos recursos abaixo disponíveis na linguagem Java possibilitam a criação de um código com melhor qualidade:

   a) Possibilidade de criação de pacotes.
   b) Mecanismo para tratamento de exceções.
   c) Tipos genéricos.

2. Considere o cenário em que uma classe qualquer define um atributo de instância, e é esperado que os "clientes/usuários" que manipulam instâncias desta classe sejam capazes tanto de obter o valor atual do atributo, quanto de alterá-lo. Por que, mesmo neste caso, é recomendável proteger o atributo, tornando-o privado, e definir *getters* e *setters* para o atributo em questão?

3. Explique como **composição** pode ser usada como uma alternativa à **herança**.

4. Suponha que você esteja trabalhando na implementação de uma certa classe, e você precise disponibilizar a informação de quantas instâncias desta classe foram criadas desde o momento em que o programa iniciou sua execução, até o instante atual. Qual seria uma boa forma de implementar tal funcionalidade?

5. Você está trabalhando com um projeto Java que possui os seguintes pacotes: **projeto** (do qual faz parte o programa principal Ex5.java), **projeto.mat** (ao qual pertencem as classes **Circulo**, **Triangulo** e **Retangulo**) e **projeto.graficos** (ao qual pertencem as classes **Poligono**, **Reta**, **Circulo**). Por que o código Ex5.java (o código fonte para esta e outras questões encontram-se a partir da página 5 deste documento) não compila? Como fazer para corrigi-lo?

6. Descreva as diversas maneiras de se utilizar, dentro do seu código, uma classe que pertence a um pacote diferente do pacote atual (isto é, aquele ao qual pertence a classe "usuária"). Escreva um trecho de código para ilustrar cada maneira.

7. Quais as saídas geradas pelo programa Ex7.java, para os seguintes pares de números digitados como entrada pelo usuário: "20 16", "8 8", "23 9.5" "10 20"?

8. Qual a utilidade de um bloco ***finally***?

9. Para que servem as instruções **throw** e **throws** relacionadas ao mecanismo para tratamento de exceções presente na linguagem Java?

10. Que característica uma classe deve ter para que suas instâncias possam ser lançadas?

11. Quais as diferenças que existem entre as exceções *verificadas* e *não verificadas*?

12. Imagine que você está trabalhando no desenvolvimento de uma biblioteca que encapsula um determinado conjunto de funcionalidades, a serem usadas por um outro grupo de desenvolvedores. Suponha que um dos métodos, de uma das classes desta biblioteca, só deve executar determinada ação se uma condição for satisfeita, e caso a condição não seja satisfeita uma exceção é lançada como forma de sinalizar ao usuário da classe/método que algo de errado aconteceu. Para cada possibilidade, listada nos itens (a), (b) e (c), referente ao tipo de exceção a ser lançada, discuta como ela afeta o lado dos desenvolvedores-usuários da biblioteca, e qual delas você escolheria usar:

    a) lançar uma instância do tipo Exception.
    b) lançar uma instância de uma classe desenvolvida por você que estenda Exception.
    c) lançar uma instância de uma classe desenvolvida por você que estenda RuntimeException.

13. Implemente uma **Fila** genérica que capaz de armazenar objetos de qualquer tipo. A classe **Fila** deve implementar o método **adiciona**, para a inserção de elementos, o método **remove**, para a remoção, e o método **tamanho** que devolve a quantidade de elementos armazenados. Internamente, sua classe deve usar um vetor (*array*) para armazenar os objetos contidos na fila.

14. Implemente um programa de teste para a classe **Fila** do exercício anterior, com os seguintes métodos genéricos: **criaFila** que recebe um vetor (*array*) de objetos genéricos e devolve uma **Fila** do tipo genérico correspondente que contenha os elementos do vetor; e **processaFila** que recebe uma **Fila** genérica e consome seus elementos, imprimindo os objetos nela guardados. Usando estes métodos genéricos, implemente um programa de teste que crie filas de diversos tipos e processe seus elementos.

15. Ainda considerando a classe de teste do exercício 14, modifique o método **processaFila** de modo que seja devolvido o elemento de menor valor presente na fila. Modifique seu programa de teste para que o elemento devolvido pelo novo método **processaFila** também seja impresso na saída do seu programa. Que tipo de modificação este novo funcionamento do método **processaFila** irá demandar?

16. Implemente o método estático `get` que recebe como parâmetros uma lista do tipo `List<T>`, um objeto do tipo `T` e, caso exista na lista um elemento considerado igual ao objeto, devolve a referência para o elemento da lista (caso contrário, o método deve devolver `null`).

17. Faça uma implementação diferente do método `get`, mas que funcionamento da mesma forma como descrito no exercício 16.

18. Faça uma terceira implementação do método `get`, com funcionamento igual ao descrito no exercício 16, mas recebendo agora um conjunto do tipo `SortedSet<T>` ao invés de uma lista do tipo `List<T>`.

19. Por que o código Ex19.java não compila, se `Number` é superclasse de `Double`? Como fazer para corrigi-lo?

20. Cite e explique as diferenças existentes entre as interfaces `Set` e `List`.

21. Cite e explique vantagens e desvantagens de cada uma das seguintes implementações da interface `List`: `ArrayList` e `LinkedList`.

22. Cite e explique vantagens e desvantagens de cada uma das seguintes implementações da interface `Set`: `HashSet` e `TreeSet`.

23. Descreva duas formas distintas de se percorrer os elementos armazenados em coleções do tipo `List`. Faça um código de exemplo ilustrando as duas formas, e comente eventuais limitações associadas a cada forma.

24. Considere o seguinte cenário: você possui uma lista de elementos ordenados e precisa realizar uma busca binária nesta lista através do método `binarySearch` da classe `Collections`. Qual implementação da interface `List` é mais indicada para esta situação? Justifique.

25. Considere o código Ex25.java. Por que o programa imprime a saída "`chave inexistente!`" mesmo que exista no mapa uma chave cujo campo id possui valor 5020? Como corrigir o programa, para que o objeto `Pessoa`, associado à chave de campo id igual a 5020, seja devolvido na chamada ao método `get`?

26. Escreve um método que recebe duas strings, e devolve um *boolean* que indica se o conjunto de caracteres únicos usados em ambas as strings é o mesmo.

27. Escreva um método que recebe uma string e determine a frequência de cada **caractere** único presente na mesma. Tente aproveitar ao máximo os recursos disponíveis na API de coleções para contabilizar a frequência dos caracteres.

28. Dados dois `Sets` **a** e **b** de elementos do tipo `T` (onde `T` é uma variável de tipo genérico), escreva métodos genéricos para determinar:

    a) A união de **a** com **b**.
    b) A intersecção de **a** com **b**.
    c) A diferença **(a – b)** (valores que existem apenas em **a**).

29. Considere o seguinte cenário: você está trabalhando em projeto no qual é definida a classe **FormaGeometrica** que possui, entre outros, um atributo privado chamado *cor*, do tipo **Cor**, acessível através de um *getter*. A classe **Cor**, por sua vez, possui três atributos privados *r*, *g* e *b*, definidos através do construtor da classe, que representam, respectivamente, as componentes vermelha, verde e azul de uma determinada cor (os três atributos são do tipo inteiro, podendo assumir valores entre 0 e 255). A partir deste cenário, escreva um método que recebe uma **Collection** de objetos do tipo **FormaGeometrica** os categorize de acordo com suas cores, ou seja, devolva uma coleção composta por subcoleções, onde cada subcoleção contem objetos de mesma cor (isto é, com os mesmos valores para as componentes *r*, *g* e *b*). Escreva seu método de modo a tirar o máximo proveito da API de coleções do Java, tanto em relação à facilidade de codificação, quanto em relação ao eficiência do método.

Código Ex5.java:

```
package projeto;

import projeto.mat.*;
import projeto.graficos.*;

public class Ex5 {

  public static void main(String [] args){

    Triangulo triangulo = new Triangulo();
    Retangulo retangulo = new Retangulo();
    Circulo circulo = new Circulo();
    Poligono poligono = new Poligono();
    Reta reta = new Reta();
  }
}
```

Código Ex7.java

```
import java.util.*;
public class Ex7 {

  public static void metodo(){
    try{
      int a, b, c;
      Scanner scanner = new Scanner(System.in);
      System.out.print("Entre com dois numeros: ");
      a = scanner.nextInt();
      b = scanner.nextInt();
      if(a - b < 0) return;
      c = a / (a - b);
      System.out.println("Resultado = " + c);
    }
    catch(ArithmeticException e){
      System.out.println("Divisao por zero!");
    }
    finally{
      System.out.println("Finally.");
    }
  }

  public static void main(String [] args){
    System.out.println("Ex8: inicio.");
    metodo();
    System.out.println("Ex8: fim.");
  }
}
```

Código Ex19.java:

```
import java.util.*;
public class Ex19 {

  public static void imprime(List<Number> lista){
    for(Number n : lista){
      double d = n.doubleValue();
      System.out.println(d);
    }
  }

  public static void main(String [] args){

    double [] valores = {10.1, 20.2, 30.3, 40.4, 50.5};
    List<Double> lista = new ArrayList<Double>();

    for(double d : valores){
      lista.add(d);
    }
    imprime(lista);
  }
}
```

Código Ex25.java:

```
import java.util.*;

public class Ex25 {

  public static void main(String [] args){

    Map<Chave, Pessoa> map = new HashMap<Chave, Pessoa>();

    int [] ids = {5000, 5010, 5020, 5030};
    String [] nomes = {"Joao", "Pedro", "Maria", "Ana"};

    for(int i = 0; i < ids.length; i++){
      Chave c = new Chave(ids[i]);
      Pessoa p = new Pessoa(c, nomes[i]);
      map.put(c, p);
    }

    Chave chave = new Chave(5020);
    if(map.containsKey(chave)){
      System.out.println(map.get(chave));
    }
    else{
      System.out.println("chave inexistente!");
    }
  }
}
```

Continuação Ex25.java:

```
class Chave {

  private int id;

  public Chave(int id){
    this.id = id;
  }

  public String toString(){
    return String.valueOf(id);
  }
}

class Pessoa {

  private Chave chave;
  private String nome;

  public Pessoa (Chave chave, String nome){
    this.chave = chave;
    this.nome = nome;
  }

  public String toString(){
    return "[Pessoa: chave = " + chave + " nome = " + nome + "]";
  }
}
```