Resultados

undefined — 2a tentativa

10
De 10 pontos

01:48
Tempo para esta tentativa

Esta avaliação não tem limite de tentativas. Fazer agora

Histórico de tentativas

Resultados	Pontos	Pontuação	(pontuação Maior é mantida)
Tentativa 1	5 de 10	50%	
Tentativa 2	10 de 10	100%	(pontuação Maior)

Suas respostas:

1 1 / 1 ponto

Considere o seguinte código em Java.

```java
import java.sql.*;
public class DAO {
    private static final String URL = "jdbc:mysql://localhost:3306/questoes";
    public void consultaBanco(String consulta) {
        try {
            Connection conexao = DriverManager.getConnection(URL);
            PreparedStatement stmt = conexao.prepareStatement(consulta);
            ResultSet rs = stmt.executeQuery();
            imprimeResultados(rs);
            conexao.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void consultaBanco(String consulta, String nomeUsuario, String senha) {
        try {
            Connection conexao = DriverManager.getConnection(URL, nomeUsuario, senha);
            PreparedStatement stmt = conexao.prepareStatement(consulta);
            ResultSet rs = stmt.executeQuery();
            imprimeResultados(rs);
            conexao.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void imprimeResultados(ResultSet rs) throws Exception {
        while (rs.next()) {
            System.out.println(rs.getString(1) + "  " + rs.getInt(2));
        }
    }
}
```

Analise as seguintes proposições.
I. A existência do método imprimeResultados é o que caracteriza a sobrecarga de métodos feita pela classe DAO.
II. Ambos os métodos “consultaBanco” esperam que a string “consulta” contenha um comando SQL INSERT, já que a execução é feita utilizando-se o método executeQuery.
III. A classe DAO utiliza JDBC e sobrecarga de métodos.

É correto apenas o que se afirma em

- II e III
- II
- I
- I e II
- ✓ III

Feedback

Feedback geral

A proposição I está incorreta. A sobrecarga é caracterizada por métodos de nomes iguais. Há somente um método chamado imprimeResultados na classe DAO e, portanto, a sua existência nada tem a ver com a sobrecarga feita pela classe.

A proposição II está incorreta. O método executeQuery devolve um objeto do tipo

ResultSet, que representa dados de uma tabela produzida pelo SGBD, o que requer o uso de um comando SELECT.

A proposição III está correta. Connection, ResultSet e PreparedStatement fazem parte da especificação JDBC. A sobrecarga de métodos está caracterizada dado que há dois métodos de nomes iguais (consultaBanco) e listas de parâmetros diferentes.

2 1 / 1 ponto

Cristina é uma desenvolvedora de software que trabalha em um projeto grande e complexo em Java. Ela recebeu a tarefa de escrever uma classe para lidar com transações bancárias. Para manter a flexibilidade do sistema, ela decidiu empregar o princípio de encapsulamento. No entanto, Cristina está um pouco confusa sobre a melhor maneira de aplicar este princípio. Escolha a alternativa correta.

- ✓ Cristina deve decidir quais membros da classe encapsular, fazer isso aplicando o modificador de acesso private e não necessariamente criar métodos getters e/ou setters.
- Cristina deve declarar todas as variáveis e métodos da classe como privados. Isso impede o acesso externo, mas pode dificultar a integração com outras partes do sistema.
- Cristina deve declarar todas as variáveis da classe como públicas, permitindo que qualquer outra classe altere seus valores diretamente, e assim garantir a rapidez das operações.
- Cristina deve apenas aplicar o modificador de acesso private a todas as variáveis da classe e escrever getters e setters para cada uma delas.
- Cristina deve fazer uso do encapsulamento consciente de que não se pode aplicar o modificador de acesso private a métodos.

Feedback

Feedback geral

Cristina deve declarar todas as variáveis da classe como públicas, permitindo que qualquer outra classe altere seus valores diretamente, e assim garantir a rapidez das operações. Incorreta. Declarar as variáveis como públicas viola o princípio da alta coesão, o que faz com que as classes sejam mais altamente acopladas e torna as alterações futuras mais difíceis.

Cristina deve declarar todas as variáveis e métodos da classe como privados. Isso impede o acesso externo, mas pode dificultar a integração com outras partes do sistema. Incorreta. Cristina deve analisar as variáveis e métodos um a um para decidir quais devem ser privados. Somente aqueles que forem detalhes de implementação internos da classe devem ser privados. Além disso, a classe precisa oferecer membros (variáveis, métodos e/ou construtores) acessíveis para que possa ser utilizada pelas demais classes.

Cristina deve decidir quais membros da classe encapsular, fazer isso aplicando o modificador de acesso private e não necessariamente criar métodos getters e/ou setters.
Correta. Cristina deve analisar membro a membro quais devem ser privados, escondendo os detalhes de implementação de sua classe e, ao mesmo tempo, mantendo uma interface pública por meio da qual as demais classes possam interagir com a sua. Nem todas as variáveis devem ter métodos de acesso e modificadores. A sua criação deve ser feita de acordo com o princípio do menor privilégio (somente se cria algo quando a sua existência é requerida de fato).

Cristina deve fazer uso do encapsulamento consciente de que não se pode aplicar o modificador de acesso private a métodos.
Incorreta. Métodos também podem ser encapsulados. Isso acontece quando um método é um detalhe de implementação interno de uma classe e a sua existência é irrelevante para as demais classes.

Cristina deve apenas aplicar o modificador de acesso private a todas as variáveis da classe e escrever getters e setters para cada uma delas.
Incorreta. Nem todas as variáveis de uma classe devem, necessariamente, ser private. Além disso, a definição de métodos getters e setters deve ser feita de acordo com o princípio do menor privilégio.

3 1 / 1 ponto

Em Java, baixo acoplamento é o conceito de que uma classe deve saber o mínimo possível sobre outras classes.

- [x] Verdadeiro
- [ ] Falso

Feedback

Feedback geral

A afirmação é verdadeira. O baixo acoplamento se caracteriza quando duas ou mais classes sabem poucos detalhes de implementação umas das outras, resultando em código mais fácil de se manter e de se reutilizar.

4 1 / 1 ponto

Clara está desenvolvendo um sistema de gestão para uma fazenda de abelhas e, para isso, está criando uma classe em Java chamada Abelha. Ela decide que seria útil ter diferentes construtores para essa classe, dependendo das informações disponíveis no momento da criação de um objeto Abelha. Analise os trechos de código a seguir.

```java
public class Abelha {
    private String especie;
    private double peso;
    private boolean rainha;

    // Construtor com todos os campos
    public Abelha(String especie, double peso, boolean rainha) {
        this.especie = especie;
        this.peso = peso;
        this.rainha = rainha;
    }

    // Construtor sem o campo 'rainha'
    public Abelha(String especie, double peso) {
        this(especie, peso, false);
    }

    // Métodos getters e setters...
}
```

```java
public class TesteAbelha {
    public static void main(String[] args) {
        Abelha abelha1 = new Abelha("Apis mellifera", 0.1, false);
        Abelha abelha2 = new Abelha("Apis cerana", 0.08);
        // Código para testar as abelhas...
    }
}
```

Escolha a alternativa correta.

- A classe Abelha faz uso da sobrecarga de métodos.
- O construtor da classe Abelha que aceita três parâmetros é desnecessário e pode ser removido sem afetar a funcionalidade da classe TesteAbelha.
- O código está incorreto porque a sobrecarga de construtores não é permitida em Java.
- A classe TesteAbelha está incorreta porque não se pode criar um objeto do tipo Abelha sem especificar se é uma rainha ou não.
- ✓ **A classe Abelha ilustra a sobrecarga de construtores, permitindo a criação de um objeto Abelha com diferentes conjuntos de dados iniciais.**

Feedback

Feedback geral

A classe Abelha faz uso da sobrecarga de métodos.
Incorreta. A classe Abelha tem apenas os métodos getters e setters (os quais nunca estão envolvidos em sobrecarga) e os construtores.

O código está incorreto porque a sobrecarga de construtores não é permitida em Java.
Incorreta. A sobrecarga de construtores existe na linguagem e funciona da mesma forma como acontece com a sobrecarga de métodos.

A classe Abelha ilustra a sobrecarga de construtores, permitindo a criação de um objeto Abelha com diferentes conjuntos de dados iniciais.
Correta. A classe Abelha possui dois construtores válidos, diferentes nas listas de parâmetros (tipo, número e ordem). Cada um permite que uma Abelha seja construída com valores diferentes.

O construtor da classe Abelha que aceita três parâmetros é desnecessário e pode ser removido sem afetar a funcionalidade da classe TesteAbelha.
Incorreta. A linha 3 da classe TesteAbelha depende da existência de tal construtor.

A classe TesteAbelha está incorreta porque não se pode criar um objeto do tipo Abelha sem especificar se é uma rainha ou não.
Incorreta. Isso é possível graças à existência do construtor da linha 14 da classe Abelha.

5 1 / 1 ponto

Considere os seguintes trechos de código e comando SQL.

```sql
CREATE TABLE Animal (
    ID INT PRIMARY KEY,
    Nome VARCHAR(255),
    Especie VARCHAR(255)
);
```

```java
public class AnimalDAO {
    private Connection con;

    public AnimalDAO() {
        con = ConnectionFactory.obterConexao();
    }

    public void inserirAnimal(int id, String nome, String especie) {
        String sql = "INSERT INTO Animal (ID, Nome, Especie) VALUES (?, ?, ?)";

        try (PreparedStatement pstmt = con.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            pstmt.setString(2, nome);
            pstmt.setString(3, especie);
            pstmt.executeUpdate();
        }catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Analise as seguintes proposições.

I. O método inserirAnimal executa corretamente mesmo nos casos em que o método obterConexao, que é chamado pelo construtor padrão, lançar uma exceção.

II. O número de placeholders (símbolos "?") é incondizente com o número de colunas da tabela.

III. Embora use um try-with-resources, o método inserirAnimal não fecha a conexão com o banco.

É correto apenas o que se afirma em

- II e III
- I
- I e II
- II
- ✓ III

Feedback

Feedback geral

A proposição I está incorreta. Se o método obterConexao lançar uma exceção, "con" jamais será inicializada e o funcionamento do método inserirAnimal depende disso, dado que ele utiliza "con" na linha 11.

A proposição II está incorreta. Há três placeholders e três colunas. Isso está correto.

A proposição III está correta. o try-with-resources usado pelo método inserirAnimal fecha apenas o objeto PreparedStatement.

6 1 / 1 ponto

Pedro escreveu o seguinte método com a intenção de obter dados de uma base MySQL usando JDBC. Suponha que a tabela existe e possui as colunas mencionadas no código.

```java
import java.sql.*;
public class ExemplODAO {
    public void getData() {
        String query = "SELECT nome, descricao FROM tb_produto WHERE id = ?";
        try {
            Connection con = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/produtos", "user", "password");

            PreparedStatement pstmt = con.prepareStatement(query);
            pstmt.setInt(1, 10);
            ResultSet rs = pstmt.executeQuery();

            while (rs.next()) {
                System.out.println(rs.getString(1) + "  " + rs.getString(2));
            }
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Analise as seguintes proposições.
I. Há um erro em tempo de compilação já que Pedro tenta acessar um placeholder (símbolo ?) de número 10.
II. É possível que a linha 14 jamais execute, ainda que nenhuma exceção aconteça.
III. É possível que a linha 16 jamais execute.

É correto apenas o que se afirma em

- III
- ✓ II e III
- II
- I e II
- I

Feedback

Feedback geral

A proposição I está incorreta. Não há nenhuma linha tentando acessar um placeholder de número 10. O único acesso a placeholders é feito na linha 10 e o placeholder ali acessado é o de número 1. A ele estamos acessando o valor 10.

A proposição II está correta. Se a query executada não trouxer dado algum, o método next representará isso devolvendo false na primeira vez que for chamado, fazendo com que o corpo da instrução while jamais execute.

A proposição III está correta. Se uma exceção acontecer antes de a linha 16 executar, o

fluxo de execução desviará imediatamente para a linha 18, fazendo com que a 16 jamais execute.

7 1 / 1 ponto

Considere o seguinte código em Java e escolha a alternativa correta.

```
1   class Carro {
2     private String marca;
3     private String modelo;
4     private int ano;
5
6     Carro(String marca, String modelo, int ano) {
7       this.marca = marca;
8       this.modelo = modelo;
9       this.ano = ano;
10    }
11
12    public String getMarca() {
13      return this.marca;
14    }
15
16    public String getModelo() {
17      return this.modelo;
18    }
19
20    public int getAno() {
21      return this.ano;
22    }
23
24    public void setMarca(String marca) {
25      this.marca = marca;
26    }
27
28    public void setModelo(String modelo) {
29      this.modelo = modelo;
30    }
31
32    public void setAno(int ano) {
33      this.ano = ano;
34    }
35  }
```

```java
public class Main {
  public static void main(String[] args) {
    Carro carro = new Carro(
      "Ford",
      "Fiesta",
      2020
    );
    carro.setAno(2023);
    System.out.println(
      carro.getMarca() + " " +
      carro.getModelo() + " " +
      carro.getAno()
    );
  }
}
```

O que o programa exibe?

- Ford Fiesta 2020
- Erro em tempo de compilação
- ✓ Ford Fiesta 2023
- Ford Fiesta 0
- Erro em tempo de execução

Feedback

Feedback geral

Ford Fiesta 2020

Incorreta. A linha 8 da classe Main altera o ano para 2023.

Ford Fiesta 2023

Correta. Marca, nome e ano são configurados no momento em que o Carro é construído, com Ford, Fiesta e 2020. Entretanto, o ano é alterado na linha 8 da classe Main.

Ford Fiesta 0

Incorreta. Não há nenhum fluxo de execução pelo qual o programa passe que faça a atribuição do valor 0 ao ano.

Erro em tempo de compilação

Incorreta. O programa não apresenta erros que impeçam a sua compilação.

e) Erro em tempo de execução

Incorreta. O programa compila e executa normalmente.

8 1 / 1 ponto

João, um programador Java, está trabalhando em um projeto que envolve modelar guitarras. Ele decide criar uma classe que descreve o que são guitarras. Veja.

```
1   class Guitarra {
2       String tipo;
3       String tamanho;
4       String cor;
5   }
```

```
1   public class TesteGuitarra {
2     public static void main(String[] args) {
3       Guitarra minhaGuitarra = new Guitarra();
4       minhaGuitarra.tipo = "elétrica";
5       minhaGuitarra.tamanho = "grande";
6       minhaGuitarra.cor = "azul";
7     }
8   }
```

Escolha a alternativa correta.

- Guitarra é um objeto e minhaGuitarra é uma classe.
- tipo, tamanho e cor são classes definidas pela classe Guitarra.
- Erro em tempo de compilação pois a classe Guitarra não tem construtor.
- minhaGuitarra é um método na classe Guitarra.

minhaGuitarra referencia uma instância da classe Guitarra.

Feedback

Feedback geral

Guitarra é um objeto e minhaGuitarra é uma classe.
Incorreta. Guitarra é uma classe e minhaGuitarra é uma variável de referência que referencia um objeto do tipo Guitarra.

minhaGuitarra referencia uma instância da classe Guitarra.
Correta. minhaGuitarra referencia o objeto do tipo Guitarra construído com o operador new.

minhaGuitarra é um método na classe Guitarra.
Incorreta. minhaGuitarra é uma variável de referência definida pela classe TesteGuitarra.

Erro em tempo de compilação pois a classe Guitarra não tem construtor.
Incorreta. A classe Guitarra tem o construtor padrão. Ele foi incluído implicitamente pelo compilador.

tipo, tamanho e cor são classes definidas pela classe Guitarra.
Incorreta. tipo, tamanho e cor são atributos (ou variáveis de instância) que a classe Guitarra define.

9 1 / 1 ponto

No desenvolvimento de software, existem vários princípios e práticas recomendadas para aumentar a reusabilidade, a flexibilidade e a facilidade de manutenção do código. Considere a classe Celular.

```
1  class Celular {
2      String marca;
3      String modelo;
4      String cor;
5  }
```

Escolha a alternativa correta.

- Atribuir múltiplas responsabilidades à classe Celular facilita a manutenção do código, já que todas as mudanças podem ser feitas em um único lugar.
- Baixa coesão, como a incorporação de várias funcionalidades não relacionadas à classe Celular, resultará em maior flexibilidade, pois a classe pode realizar diversas funções.

A flexibilidade do código é melhorada quando deixamos de encapsular os atributos, como no exemplo da classe Celular, para permitir acesso de qualquer lugar.

A classe Celular é altamente coesa, embora não possua um método main.

A reutilização de código em Java é desencorajada, pois pode aumentar a complexidade do projeto.

Feedback

Feedback geral

A classe Celular é altamente coesa, embora não possua um método main.
Correta. A classe Celular tem uma única finalidade, que é explicar o que é um Celular. O fato de ela não ter um método main apenas contribui para esse fato.

A reutilização de código em Java é desencorajada, pois pode aumentar a complexidade do projeto.
Incorreta. A reutilização de código simplifica o projeto. Por exemplo, quando definimos um método e o reutilizamos, temos um único ponto para dar manutenção caso uma regra de negócio seja alterada no futuro.

A flexibilidade do código é melhorada quando deixamos de encapsular os atributos, como no exemplo da classe Celular, para permitir acesso de qualquer lugar.
Incorreta. Deixar de utilizar o encapsulamento torna as classes altamente acopladas, o que torna o código menos flexível, já que uma simples alteração em uma classe tem o potencial de causa impacto em várias outras.

Atribuir múltiplas responsabilidades à classe Celular facilita a manutenção do código, já que todas as mudanças podem ser feitas em um único lugar.
Incorreta. Uma classe com múltiplas responsabilidades viola o princípio da alta coesão. Quando tem muitas responsabilidades, uma classe tende a ter muitas linhas de código e tende a misturar funcionalidades, como acessar o banco de dados e exibir interfaces gráficas, por exemplo, o que dificulta a manutenção e a compreensão do código.

Baixa coesão, como a incorporação de várias funcionalidades não relacionadas à classe Celular, resultará em maior flexibilidade, pois a classe pode realizar diversas funções.
Incorreta. Violar o princípio da alta coesão tende a dar origem a classes com múltiplas responsabilidades, o que torna o código menos flexível, ou seja, difícil de se alterar e manter.

10 1 / 1 ponto

Maria é uma desenvolvedora Java em uma startup de tecnologia. Ela está trabalhando em um sistema de agendamento de reuniões e precisa criar um método para agendar reuniões. As reuniões podem ser agendadas de várias maneiras diferentes.

Maria decide usar a sobrecarga de métodos para resolver o problema e cria a seguinte classe.

```
1   class Agendamento {
2       void agendarReuniao(String data) {
3           // código para agendar a reunião
4       }
5
6       void agendarReuniao(String data, String horario) {
7           // código para agendar a reunião
8       }
9
10      void agendarReuniao(String data, String horario, String sala) {
11          // código para agendar a reunião
12      }
13  }
```

Escolha a alternativa correta.

- ✓ A sobrecarga de métodos em Java é determinada pelo nome do método e pelo número, ordem e tipo de argumentos. Portanto, o código de Maria compila.
- Os métodos sobrecarregados em Java devem necessariamente ter diferentes modificadores de acesso. Como todos os métodos de Maria têm o mesmo modificador de acesso, sua implementação está incorreta.
- É impossível dizer se a sobrecarga feita por Maria é válida, já que seus métodos não possuem modificadores de acesso explícitos.
- A implementação de Maria causa erro em tempo de execução já que todos os parâmetros envolvidos na sobrecarga têm tipo igual.
- A implementação de Maria está incorreta pois todos os seus métodos têm tipo de retorno igual a “void”.

Feedback

Feedback geral

A sobrecarga de métodos em Java é determinada pelo nome do método e pelo número, ordem e tipo de argumentos. Portanto, o código de Maria compila.
Correta. Essa é a definição da sobrecarga: métodos de nomes iguais, com listas de parâmetros diferentes no número, tipo e/ou ordem dos parâmetros.

É impossível dizer se a sobrecarga feita por Maria é válida, já que seus métodos não possuem modificadores de acesso explícitos.
Incorreta. A existência explícita de modificadores de acesso (public ou private, por exemplo) é irrelevante no que diz respeito à sobrecarga de métodos.

A implementação de Maria causa erro em tempo de execução já que todos os parâmetros envolvidos na sobrecarga têm tipo igual.
Incorreta. As listas de parâmetros têm número de parâmetros diferentes e a sobrecarga é válida. Além disso, ainda que houvesse algum problema com a sobrecarga, ela seria resolvida em tempo de compilação (pelo compilador) e não em tempo de execução (pela JVM).

Os métodos sobrecarregados em Java devem necessariamente ter diferentes modificadores de acesso. Como todos os métodos de Maria têm o mesmo modificador de acesso, sua implementação está incorreta.
Incorreta. A sobrecarga de métodos é caracterizada por métodos de nomes iguais e listas de parâmetros diferentes, no número, tipo e/ou ordem de parâmetros. Os modificadores de acesso não têm nada a ver com isso.

A implementação de Maria está incorreta pois todos os seus métodos têm tipo de retorno igual a “void”.
Incorreta. A sobrecarga de métodos é caracterizada por métodos de nomes iguais e listas de parâmetros diferentes, no número, tipo e/ou ordem de parâmetros. O tipo de retorno nada tem a ver com isso.