Nome: Cezar Faria **RA: G87CEE9**

1. **Qual o conceito de máquina virtual em Java?**

   JVM (Máquina Virtual Java) é uma abstração que executa o bytecode Java, garantindo portabilidade e independência de plataforma.

2. **Por que a linguagem de programação Java é chamada de multiplataforma?**

   Porque o código compilado (bytecode) pode ser executado em qualquer sistema que tenha uma JVM

3. **Em linguagens baseadas na sintaxe do C, como o Java, recomenda-se o camel case para se definir nomes no código. O que é o camel case e por que ele é usado?**

   Camel case é uma convenção de nomenclatura onde cada palavra, exceto a primeira, começa com letra maiúscula (ex: `minhaVariavel`), facilitando a leitura.

4. **A principal utilização do tratamento de exceções em um programa é para:**

   Evitar a interrupção abrupta do programa, tratando erros que ocorrem em tempo de execução

5. **Finally{}, sempre será executado depois do bloco try/catch, exceto:**

   O bloco `finally` será executado exceto se o programa terminar abruptamente com `System.exit()`.

6. **O que não podemos considerar como um motivo para ocorrer uma exceção:**

   Erros de sintaxe não geram exceções em tempo de execução, pois são tratados durante a compilação.

7. **Explique o código a seguir:**

```
throw new Exception("Ocorreu uma excecao ");
try {
   //bloco de código 1
} catch (umaExcecao e1) {
         //bloco de código 2
} catch (outraExcecao e2) {
         //bloco de código 3
} catch (maisUmaExcecao e3) {
         //bloco de código 4
}
finally {
         //bloco de código 5
}
```

Esse código lança uma exceção e captura diferentes tipos de exceções, permitindo tratar cada uma individualmente. O bloco `finally` será executado independentemente de qualquer exceção.

8. **Considerando um programa com interface gráfica, explique o motivo de não ser indicado colocar todo o código que será executado no método que receberá o evento clique no botão.**

Isso pode tornar o código difícil de manter. Idealmente, deve-se delegar responsabilidades a outros métodos ou classes.

9. **Escreva um programa em Java, que escreva "Olá mundo" na tela.**

```
public class OlaMundo {
    public static void main(String[] args) {
        System.out.println("Olá Mundo");
```

10. **Ao instalar o JDK no computador, o que estamos instalando? E quem deve instalar o JDK no seu computador?**

O JDK (Java Development Kit) contém ferramentas para desenvolvimento, como o compilador e a JVM. Deve ser instalado por desenvolvedores.

11. **Como podemos compilar um programa java, sem o uso de IDE?**

Utilizando o terminal: `javac NomeDoArquivo.java`.

12. **Ao compilar um programa java no windows, ele não gera um .exe, explique o motivo.**

    Java gera arquivos .class com bytecode, que é interpretado pela JVM em qualquer sistema.

13. **Explique o slogan que a Sun criou para o Java - A Sun criou um slogan para a plataforma Java: "Write Once, RUn Anywhere" ou seja, uma vez criada a aplicação, ela pode ser executada em qualquer máquina.**

    O Java permite que uma vez que o código seja compilado em bytecode, ele possa ser executado em qualquer sistema com JVM.

14. **O desenvolvimento da plataforma Java permitiu o surgimento de outras linguagens de programação que rodam sob a máquina virtual. É o caso de Groovy por exemplo. Explique como isso é possível.**

    O Java permite que uma vez que o código seja compilado em bytecode, ele possa ser executado em qualquer sistema com JVM.

15. **Explique os diferentes ambientes em que o Java pode ser empregados (JSE, JEE, JME e JavaFX)**

    - JSE: Aplicações desktop.
    - JEE: Aplicações corporativas.
    - JME: Aplicações móveis.
    - JavaFX: Interfaces gráficas avançadas.

16. **O que permite o uso do Jshell?**

    JShell permite executar expressões Java interativamente. Permite que você execute comandos Java em tempo real

17. **O que é um jar?**

    Um JAR é um arquivo compactado que contém classes Java e metadados para distribuição.

18. **No java, como podemos escrever um método main válido?**

```
public static void main(String[] args) {
}
```

19. Ao executar o código a seguir, o que será exibido na tela?

```
import javax.swing.JOptionPane;

public class OlaMundoSwing {
   public static void main(String[] args){
         JOptionPane.showMessageDialog(null, "Olá mundo Swing");
   }
}
```

Mostra uma caixa de diálogo com a mensagem "Olá mundo Swing".

20. Considerando que aluno possui nome, notaB1, notaB2 e notaPim e que pode ter sua média calculada, implementa a regra aluno (usando as regras de média da UNIP para o curso de ADS).

```
public class Aluno {
      private double np1;
      private double np2;
      private double pim;
      private double media;
      public void setNp1(double np1) {
      this.np1 = np1;
      }
      public void setNp2(double np2) {
      this.np2 = np2;
      }
      public void setPim(double pim) {
      this.pim = pim;
      }
      public void setMedia(double media) {
      this.media = media;
      }
      public double getNp1() {
      return np1;
      }
      public double getNp2() {
      return np2;
      }
      public double getPim() {
      return pim;
      }
      public double getMedia() {
      return media;
      }
      public double CalcularMediaSemExame(){
      setMedia(np1*0.4 + np2*0.4 + pim*0.2);
      }
}
```

21. Considerando a classe Aluno, como podemos criar um objeto?

```
Aluno aluno = new Aluno();
```

22. Qual a função dos pacotes?

Organizar e evitar conflitos de nomes em projetos grandes.

23. Qual o padrão para nomes de pacotes?

São escritos em minúsculas, normalmente usando o domínio reverso da empresa (ex: `com.empresa.projeto`).

24. Considerando que importou duas bibliotecas com a classe Endereco, ao instanciar o objeto, quais os métodos disponíveis neste objeto?

```
package revisaob1.ads;

public class Endereco {
        private String rua;
        private String numero;
        private String cidade;

        public String getRua() {
        return rua;
        }

        public void setRua(String rua) {
        this.rua = rua;
        }

        public String getNumero() {
        return numero;
        }

        public void setNumero(String numero) {
        this.numero = numero;
        }

        public String getCidade() {
        return cidade;
        }

        public void setCidade(String cidade) {
        this.cidade = cidade;
        }
```

```
package revisaob1.cc;


public class Endereco {
        private String rua;
        private String numero;

        public String getRua() {
        return rua;
        }

        public void setRua(String rua) {
        this.rua = rua;
        }

        public String getNumero() {
        return numero;
        }

        public void setNumero(String numero) {
        this.numero = numero;
        }

        @Override
        public String toString() {
        return "Enderco{" + "rua=" + rua + ", numero=" + numero + '}';
        }
```

@Override
public String toString() {
return "Endereco{" + "rua=" + rua + ", numero=" + numero + ", cidade=" + cidade + '}';
}
 	
}
	}

Depende de qual das duas classes for especificada, o que é obrigatório quando elas têm o mesmo nome. Por exemplo:
revisaob1.ads.Endereco enderecoADS = new revisaob1.ads.Endereco();
Contendo:
getRua(): Retorna o valor da rua.
setRua(String rua): Define o valor da rua.
getNumero(): Retorna o valor do número.
setNumero(String numero): Define o valor do número.
getCidade(): Retorna o valor da cidade.
setCidade(String cidade): Define o valor da cidade.
toString(): Retorna uma representação em string da classe com os atributos rua, numero e cidade.
ou
revisaob1.cc.Endereco enderecoCC = new revisaob1.cc.Endereco();
Contendo:
getRua(): Retorna o valor da rua.
setRua(String rua): Define o valor da rua.
getNumero(): Retorna o valor do número.
setNumero(String numero): Define o valor do número.
toString(): Retorna uma representação em string da classe com os atributos rua e numero.

25. Considerando que já possui o .class, e o manifest.mf, como podemos construir um jar por linha de comando?

Use o comando `jar -cvf NomeDoArquivo.jar -C CaminhoDoDiretório .`

26. Como executar um jar por linha de comando

Use `java -jar NomeDoArquivo.jar`.

27. **Considere a classe Valores a seguir, escreva o código necessário para instanciar essa classe e receber os valores v1,v2 e v3 do usuário, após isso, exibir os valores na tela.**

```
package revisaob1;


public class Valores {
        private int v1;
        private String v2;
        private double v3;

        public Valores() {
        }

        public int getV1() {
        return v1;
        }

        public void setV1(int v1) {
        this.v1 = v1;
        }

        public String getV2() {
        return v2;
        }

        public void setV2(String v2) {
        this.v2 = v2;
        }

        public double getV3() {
        return v3;
        }

        public void setV3(double v3) {
        this.v3 = v3;
        }

        @Override
        public String toString() {
        return "Valores {" + "v1=" + v1 + ", v2=" + v2 + ", v3=" + v3 + '}';
        }



}
```

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        // Instanciando um objeto da classe Valores
        Valores valores = new Valores();

        // Recebendo os valores do usuário
        System.out.print("Digite o valor inteiro (v1): ");
        int v1 = scanner.nextInt();
        valores.setV1(v1);

        scanner.nextLine(); // Consumindo a quebra de linha

        System.out.print("Digite o valor de texto (v2): ");
        String v2 = scanner.nextLine();
        valores.setV2(v2);

        System.out.print("Digite o valor decimal (v3): ");
        double v3 = scanner.nextDouble();
        valores.setV3(v3);

        // Exibindo os valores
        System.out.println("Valores informados:");
        System.out.println(valores);

        // Fechando o scanner
        scanner.close();
    }
}
```

28. Defina Wrapper

Classes que encapsulam tipos primitivos em um objeto (ex: `Integer` para `int`).

29. Dado os tipos primitivos a seguir, informe quais seus wrappers:

a. boolean
b. char
c. byte
d. short
e. int
f. long
g. float
h. double

a. boolean -> `Boolean`
b. char -> `Character`
c. byte -> `Byte`
d. short -> `Short`
e. int -> `Integer`
f. long -> `Long`
g. float -> `Float`
h. double -> `Double`

30. **O programa a seguir permite receber dois valores em sua execução, informe como poderemos executar este programa passando o nome e sobrenome.**

```
1   package parametro;
2
3   public class Parametro {
4
5       public static void main(String[] args) {
6           String nome = args[0];
7           String sobreNome = args[1];
8           System.out.println("Informado: " + nome + " " + sobreNome);
9       }
10
11  }
```

Ao executar o programa, deve-se passar os valores como argumento. Por exemplo, em linha de comando:
java Parametro valor1 valor2

31. **Dado o vetor a seguir, informe como podemos exibir todos os valores deste array:**

```
package revisaob1;


public class RevisaoB1 {

        public static void main(String[] args) {
        int vetor[] = { 5,4,3,2,1,0};

        }

}
```

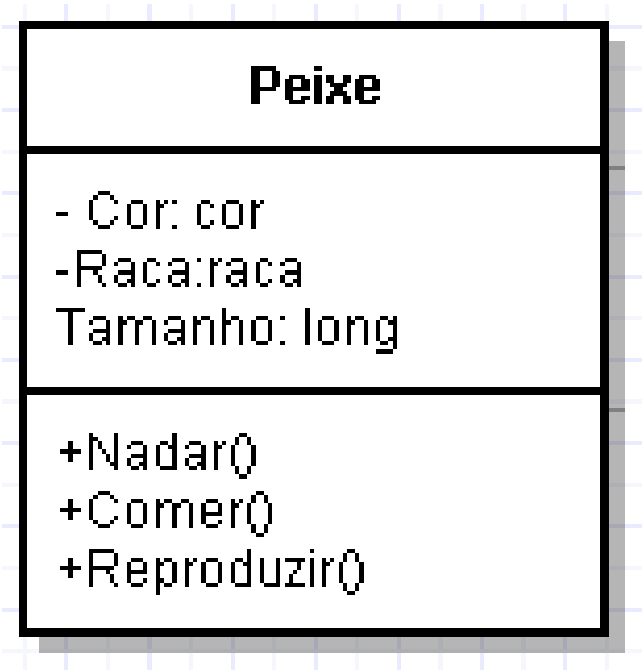
```
for (int valor : vetor) {
    System.out.println(valor);
}
```

32. **Dado a matriz a seguir, escreva o programa para exibir todos os valores contidos nela**

```
package revisaob1;


public class RevisaoB1 {

        public static void main(String[] args) {
        int mat[][] = {
        {1,2,3},
        {1,2,3},
        {1,2,3}
        };
}
```

```
        for (int[] linha : mat) {
           for (int valor : linha) {
               System.out.println(valor);
           }
       }
```

33. **Implemente a classe peixe a seguir, fazendo com que cada método imprima seu nome na tela.**




```
public class Peixe {

    // Atributos da classe Peixe
    private String cor;
    private String raca;
    private long tamanho;
```

```
    // Construtor
    public Peixe(String cor, String raca, long tamanho) {
        this.cor = cor;
        this.raca = raca;
        this.tamanho = tamanho;
    }

    // Método para o peixe nadar
    public void nadar() {
        System.out.println("O peixe da raça " + this.raca + " e de cor " +
this.cor + " está nadando.");
    }

    // Método para o peixe comer
    public void comer() {
        System.out.println("O  peixe  da  raça  "  +  this.raca  +  "  está
comendo.");
    }

    // Método para o peixe se reproduzir
    public void reproduzir() {
        System.out.println("O  peixe  da  raça  "  +  this.raca  +  "  está  se
reproduzindo.");
    }

    // Método para exibir as informações do peixe
    public void imprimirInformacoes() {
        System.out.println("Informações do Peixe:");
        System.out.println("Cor: " + this.cor);
        System.out.println("Raça: " + this.raca);
        System.out.println("Tamanho: " + this.tamanho + " cm");
    }

    // Método main para testar a classe
    public static void main(String[] args) {
        // Criando um objeto da classe Peixe
        Peixe peixe = new Peixe("Azul", "Tilápia", 30);

        // Chamando os métodos para imprimir os atributos e ações
        peixe.imprimirInformacoes();
        peixe.nadar();
        peixe.comer();
        peixe.reproduzir();
    }
}
```

Ao executar o programa, a saída será:

```
Informações do Peixe:
Cor: Azul
Raça: Tilápia
Tamanho: 30 cm
O peixe da raça Tilápia e de cor Azul está nadando.
O peixe da raça Tilápia está comendo.
O peixe da raça Tilápia está se reproduzindo.
```

34. **Ao definir um atributo com a visibilidade private, ele não fica acessível para objetos do tipo desta classe, uma forma de permitir este acesso é criando os métodos get e set, faça o exemplo de uma implementação e justifique o motivo de usarmos get e set e não tornarmos o atributo público**.

```
private String nome;

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}
```

Usar encapsulamento e modificadores de acesso como private ou protected garantem maior controle de quem e como pode ter acesso a um método ou atributo.

35. Explique construtores e sobrecarga de construtores

Um construtor é um método especial que é chamado quando um objeto de uma classe é criado. Ele é utilizado para inicializar o objeto, ou seja, atribuir valores iniciais aos atributos de uma classe. Um construtor:

- Tem o mesmo nome da classe.
- Não tem tipo de retorno (nem mesmo void).
- Pode ser padrão (sem parâmetros) ou receber parâmetros para inicialização personalizada.

Se nenhum construtor for definido, o compilador Java fornece um construtor padrão sem parâmetros automaticamente. No entanto, se você definir um construtor, o construtor padrão não será mais gerado automaticamente.

Exemplo de Construtor Simples:

```
class Pessoa {
    private String nome;
    private int idade;

    // Construtor
    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    public void exibirInfo() {
        System.out.println("Nome: " + nome + ", Idade: " + idade);
    }
}

public class Main {
    public static void main(String[] args) {
```

```
        Pessoa pessoa = new Pessoa("João", 25);
        pessoa.exibirInfo();  // Saída: Nome: João, Idade: 25
    }
}
```

Sobrecarga de Construtores

A sobrecarga de construtores ocorre quando uma classe tem mais de um construtor, com diferentes listas de parâmetros. Isso permite criar objetos de diferentes maneiras, dependendo dos argumentos fornecidos. O compilador Java diferencia os construtores pela assinatura (número e tipo dos parâmetros).

Exemplo com Sobrecarga de Construtores:

```
class Pessoa {
    private String nome;
    private int idade;

    // Construtor 1: inicializa nome e idade
    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    // Construtor 2: inicializa apenas o nome, idade padrão
    public Pessoa(String nome) {
        this.nome = nome;
        this.idade = 0; // Idade padrão
    }

    // Construtor 3: construtor padrão (nenhum argumento)
    public Pessoa() {
        this.nome = "Desconhecido";
        this.idade = 0;
    }

    public void exibirInfo() {
        System.out.println("Nome: " + nome + ", Idade: " + idade);
    }
}

public class Main {
    public static void main(String[] args) {
        // Usando diferentes construtores
        Pessoa pessoa1 = new Pessoa("João", 25);
        Pessoa pessoa2 = new Pessoa("Maria");
        Pessoa pessoa3 = new Pessoa();

        pessoa1.exibirInfo();  // Saída: Nome: João, Idade: 25
        pessoa2.exibirInfo();  // Saída: Nome: Maria, Idade: 0
        pessoa3.exibirInfo();  // Saída: Nome: Desconhecido, Idade: 0
    }
}
```

Explicação:

- Construtor 1: Inicializa nome e idade com valores passados.
- Construtor 2: Inicializa apenas o nome e define um valor padrão para idade.
- Construtor 3: Define valores padrão para ambos os atributos (construtor padrão).

Essa técnica é útil para fornecer diferentes formas de inicializar objetos, oferecendo mais flexibilidade ao programador. Por exemplo, pode-se usar valores padrão ou personalizados dependendo da situação.

Quando Usar Sobrecarga de Construtores:

- Quando você quer dar opções ao usuário de criar objetos com diferentes conjuntos de parâmetros.
- Quando você deseja fornecer valores padrão em alguns casos, mas ainda permitir a personalização em outros.

36. Quando devemos utilizar Herança

Utilizamos herança quando desejamos criar uma nova classe que herda propriedades e comportamentos de uma classe existente. A herança estabelece uma relação "é-um" (is-a), onde a classe filha (subclasse) herda atributos e métodos da classe pai (superclasse), permitindo reutilização de código e extensibilidade.

Quando Usar Herança:

- Quando várias classes compartilham características e comportamentos comuns, mas também têm características específicas que precisam ser implementadas.
- Quando queremos evitar duplicação de código e promover a reutilização.
- Quando existe uma relação lógica "é-um" entre as classes. Por exemplo, "Cachorro" é um tipo de "Animal".

Exemplo:

Vamos criar uma superclasse Animal e subclasses Cachorro e Gato, que herdam comportamentos de Animal, mas têm comportamentos específicos.

```
// Superclasse
class Animal {
    protected String nome;

    public Animal(String nome) {
        this.nome = nome;
    }
```

```
    public void fazerSom() {
        System.out.println("O animal faz um som.");
    }
}

// Subclasse Cachorro (herda de Animal)
class Cachorro extends Animal {

    public Cachorro(String nome) {
        super(nome); // Chama o construtor da superclasse
    }

    // Sobrescreve o método fazerSom
    @Override
    public void fazerSom() {
        System.out.println(nome + " faz: Au Au!");
    }
}

// Subclasse Gato (herda de Animal)
class Gato extends Animal {

    public Gato(String nome) {
        super(nome);
    }

    // Sobrescreve o método fazerSom
    @Override
    public void fazerSom() {
        System.out.println(nome + " faz: Miau!");
    }
}

public class Main {
    public static void main(String[] args) {
        // Criando um Cachorro e um Gato
        Cachorro cachorro = new Cachorro("Rex");
        Gato gato = new Gato("Mimi");

        // Chamando o método sobrescrito em cada subclasse
        cachorro.fazerSom();  // Saída: Rex faz: Au Au!
        gato.fazerSom();      // Saída: Mimi faz: Miau!
    }
}

Saída:

Rex faz: Au Au!
Mimi faz: Miau!
```

Explicação:

- A classe Animal é a superclasse, que contém um atributo nome e o método fazerSom().
- As classes Cachorro e Gato herdam de Animal e sobrescrevem o método fazerSom() para fornecer comportamentos específicos para cada tipo de animal.
- As subclasses reutilizam o código da superclasse (como o construtor) e podem adicionar ou modificar funcionalidades conforme necessário.

Quando Usar:

- Quando existe uma relação clara de “é-um”.
- Para evitar duplicação de código quando várias classes têm comportamentos comuns.
- Quando se deseja estender o comportamento de uma classe base em classes derivadas.

Herança deve ser usada com cuidado, pois uma hierarquia mal planejada pode dificultar a manutenção do código.

37. Faça uma classe que use Composição.

É quando uma classe é composta por uma ou mais instâncias de outras classes, o que permite construir relações do tipo “tem um” (has-a). Na composição, uma classe contém uma referência a outra classe e, geralmente, o ciclo de vida do objeto composto está vinculado ao ciclo de vida da classe principal.

Exemplo:

Imagine que temos uma classe Motor e uma classe Carro.
O Carro “tem um” Motor, portanto, podemos modelar essa relação através da composição.

```
// Classe Motor (componente)
class Motor {
    private int potencia;

    public Motor(int potencia) {
        this.potencia = potencia;
    }

    public int getPotencia() {
        return potencia;
    }

    @Override
    public String toString() {
        return "Potência do motor: " + potencia + " cavalos";
    }
}

// Classe Carro (composta por Motor)
class Carro {
    private String modelo;
    private Motor motor;
```

```
    public Carro(String modelo, int potenciaMotor) {
        this.modelo = modelo;
        // Composição: o Carro possui um Motor
        this.motor = new Motor(potenciaMotor);
    }

    public String getModelo() {
        return modelo;
    }

    public Motor getMotor() {
        return motor;
    }

    @Override
    public String toString() {
        return "Modelo do carro: " + modelo + ", " + motor.toString();
    }
}

public class Main {
    public static void main(String[] args) {
        // Criando um carro que possui um motor
        Carro carro = new Carro("Ferrari", 500);
        System.out.println(carro.toString());
    }
}

Saída:

Modelo do carro: Ferrari, Potência do motor: 500 cavalos
```

Explicação:

- A classe Carro contém uma instância da classe Motor, representando a relação de composição.
- Quando criamos um objeto Carro, também criamos um Motor internamente no construtor, com a potência especificada.
- O método toString() em Carro utiliza o toString() de Motor para exibir as informações de ambos de forma integrada.

Nesse caso, se o objeto Carro for destruído, o objeto Motor também deixará de existir, demonstrando a relação de dependência forte entre os dois.

38. Defina classe abstrata e diga quando ela deve ser utilizada

Deve ser usada quando não faz sentido instanciar a classe diretamente.
Classe abstrata não pode ser instanciada diretamente. Ela pode ou não conter implementações de métodos.
As classes que herdam dela devem implementar os métodos abstratos.
Ela pode ser usada para definir um modelo base para outras classes. Isso é útil quando várias classes compartilham comportamentos comuns, mas cada uma pode ter sua própria implementação de alguns métodos.

39. Defina classe interface e diga quando ela deve ser utilizada.

Define métodos que devem ser implementados por outras classes.
É como um contrato que define tudo que uma outra classe deve obrigatoriamente implementar.

40. Faça um exemplo de uma classe SaldoInsuficenteException para ser um exception a ser lançada quando o saldo de uma conta não for suficiente.

```
public class SaldoInsuficienteException extends Exception {
    public SaldoInsuficienteException(String mensagem) {
        super(mensagem);
    }
}
```

41. Uma interface com o usuário construída com AWT é composta de três itens: Componentes, Containers e Gerenciadores de layout. Explique cada um deles.

a. **Componentes:**
b. **Containers:**
c. **Gerenciador de layout:**

a. **Componentes:** Botões, campos de texto.

   Componentes: São os elementos gráficos que formam a interface do usuário. Eles são as partes visíveis da interface, que interagem diretamente com o usuário.

   Exemplos:

   Button, Label, Checkbox

b. **Containers:** Janelas, painéis.

   Containers são componentes que podem conter outros componentes. Eles servem como um contêiner para organizar e agrupar outros componentes.

Exemplos:

Frame, Panel, Dialog

c. **Gerenciadores de layout**: Definem como os componentes são organizados.

Um gerenciador de layout é responsável pela organização e posicionamento dos componentes dentro de um container. Ele define como os componentes devem ser dispostos e dimensionados em relação uns aos outros.
Exemplos:
FlowLayout, BorderLayout, GridLayout

42. Ao executar o código a seguir, o que será exibido na tela?

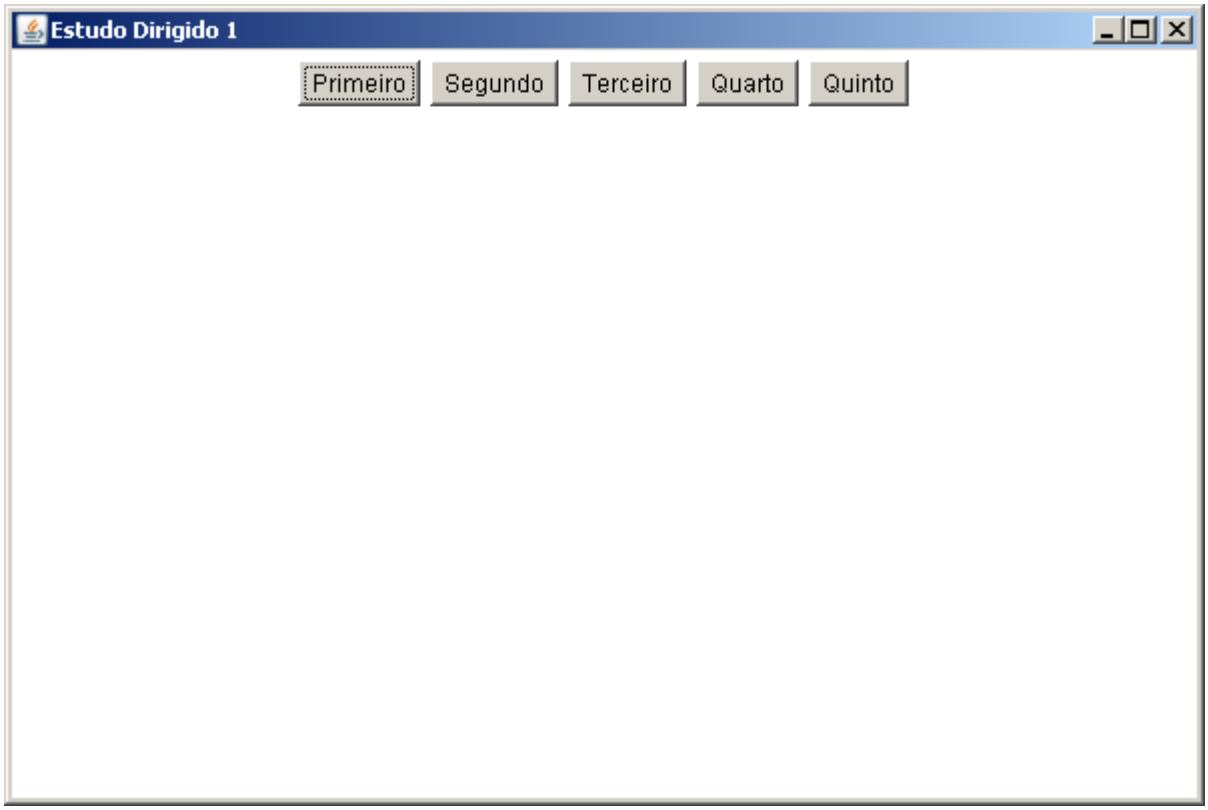
```
3  import java.awt.*;
4
5  public class OlaMundo extends  Frame {
6      public OlaMundo(){
7          this.setTitle("Olá mundo");
8          this.setSize(300,400);
9          this.setVisible(true);
10     }
11
12     public static void main(String[] args) {
13         OlaMundo olaMundo = new OlaMundo();
14     }
15 }
```

Abre uma janela com o título: “Olá mundo”, de tamanho 300x400.

43. Dado o código anterior, como podemos adicionar o texto “Olá mundo” na tela?

Adicionar na classe OlaMundo:
Label label = new Label("Olá mundo");
this.add(label);

44. Escreva o código para gerar a tela a seguir




```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import java.awt.BorderLayout;

public class TelaComAbas {
    public static void main(String[] args) {
        // Criando a janela principal
        JFrame frame = new JFrame("Estudo Dirigido 1");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        // Criando o painel com abas
        JTabbedPane tabbedPane = new JTabbedPane();

        // Criando os painéis para cada aba
        JPanel primeiroPanel = new JPanel();
        JPanel segundoPanel = new JPanel();
        JPanel terceiroPanel = new JPanel();
        JPanel quartoPanel = new JPanel();
        JPanel quintoPanel = new JPanel();

        // Adicionando os painéis às abas
        tabbedPane.addTab("Primeiro", primeiroPanel);
```

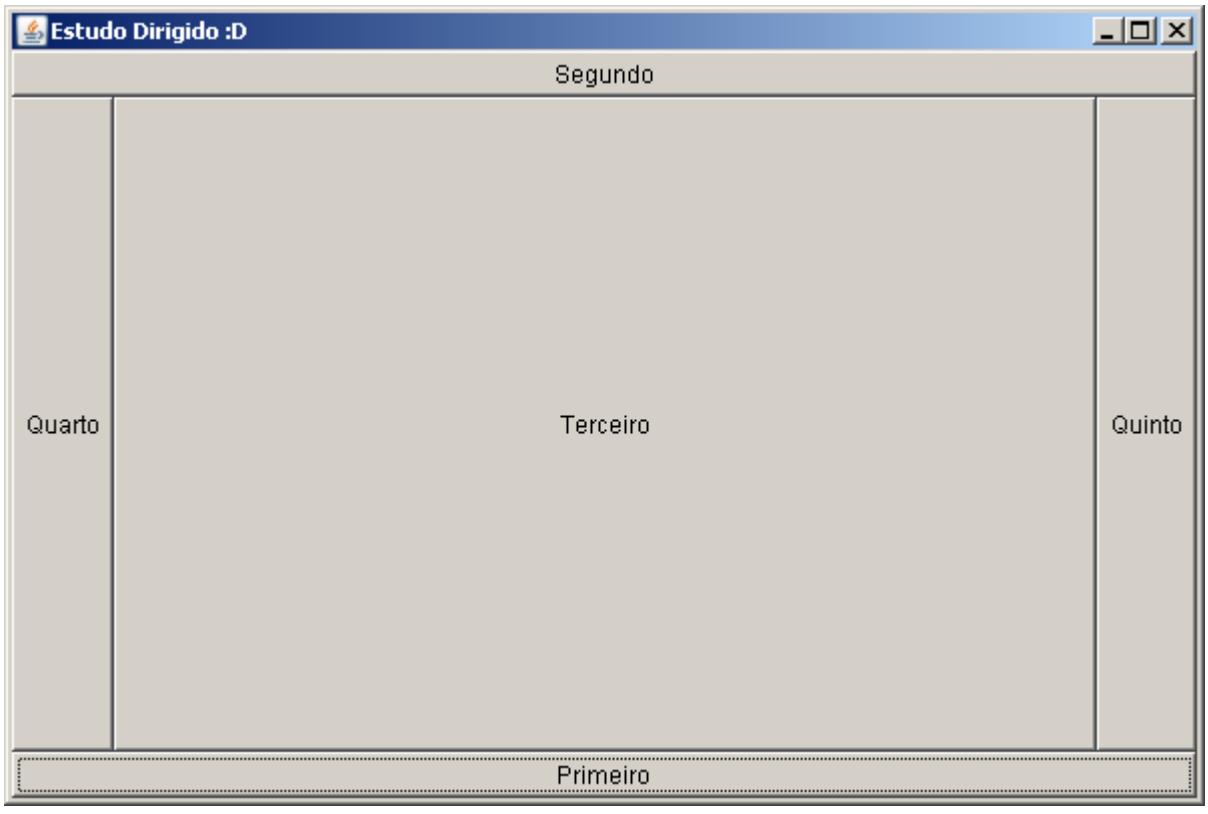
```
        tabbedPane.addTab("Segundo", segundoPanel);
        tabbedPane.addTab("Terceiro", terceiroPanel);
        tabbedPane.addTab("Quarto", quartoPanel);
        tabbedPane.addTab("Quinto", quintoPanel);

        // Adicionando o painel com abas à janela
        frame.add(tabbedPane, BorderLayout.CENTER);

        // Exibindo a janela
        frame.setVisible(true);
    }
}
```

45. Escreva o código para gerar a tela a seguir


```
        // Definindo o título da janela
        setTitle("Estudo Dirigido 1");

        // Configurando o layout para GridLayout (2 linhas, 3 colunas)
        setLayout(new GridLayout(2, 3));

        // Criando os botões com os nomes indicados na imagem
        Button primeiro = new Button("Primeiro");
        Button segundo = new Button("Segundo");
        Button terceiro = new Button("Terceiro");
        Button quarto = new Button("Quarto");
```

```
        Button quinto = new Button("Quinto");

        // Adicionando os botões ao layout (a sexta célula será vazia)
        add(primeiro);
        add(segundo);
        add(terceiro);
        add(quarto);
        add(quinto);

        // A sexta célula é deixada em branco, pois não adicionamos nada nela

        // Configurando o tamanho da janela
        setSize(400, 300);

        // Tornando a janela visível
        setVisible(true);
```

46. Escreva o código para gerar a tela a seguir




```
import java.awt.*;

public class meuAwt extends Frame {
    public meuAwt(){
        setTitle("Estudo Dirigido :D");

        // Configurando o layout para BorderLayout
```

```
        setLayout(new BorderLayout());

        // Criando os botões com os nomes indicados na imagem
        Button primeiro = new Button("Primeiro");
        Button segundo = new Button("Segundo");
        Button terceiro = new Button("Terceiro");
        Button quarto = new Button("Quarto");
        Button quinto = new Button("Quinto");

        // Adicionando os botões em suas respectivas regiões do BorderLayout
        add(primeiro, BorderLayout.SOUTH);
        add(segundo, BorderLayout.NORTH);
        add(terceiro, BorderLayout.CENTER);
        add(quarto, BorderLayout.WEST);
        add(quinto, BorderLayout.EAST);

        // Configurando o tamanho da janela
        setSize(400, 300);

        // Tornando a janela visível
        setVisible(true);
    }
}
```

47. Analise o código a seguir em seguida informe o que será exibido ao clicar no botão bomDiaBtn.

```
public class BomDiaBoaNoiteFrm extends JFrame {

        private JPanel panel;
        private JButton bomDiaBtn;
        private JButton boaNoiteBtn;

        public BomDiaBoaNoiteFrm() {
                initComponents();
        }

        private void initComponents() {
                this.setSize(200, 100);

        this.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
                panel = new JPanel();
                bomDiaBtn = new JButton("Bom dia");
                boaNoiteBtn = new JButton("Boa noite");
                panel.add(bomDiaBtn);
                panel.add(boaNoiteBtn);
                this.add(panel);
        }

        private void bomDiaActionEvent(ActionEvent evt) {
```

```
                JOptionPane.showMessageDialog(this, "Bom dia");
        }

        private void boaTardeActionEvent(ActionEvent evt) {
                JOptionPane.showMessageDialog(this, "Boa tarde");
        }

        public static void main(String[] args) {
                new BomDiaBoaNoiteFrm().setVisible(true);
        }

}
```

Mostra "Bom dia".

48. Para que serve o método tostring? faça um exemplo

O método toString() serve para fornecer uma representação em string de um objeto. É definido na classe base Object e pode ser sobrescrito em classes personalizadas para retornar uma representação mais útil.

Quando você imprime um objeto diretamente, o método toString() é chamado automaticamente. Se não for sobrescrito, ele retorna uma string com o nome da classe seguido por o hash code do objeto.

Exemplo:

```
class Pessoa {
    private String nome;
    private int idade;

    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    // Sobrescrevendo o método toString
    @Override
    public String toString() {
        return "Nome: " + nome + ", Idade: " + idade;
    }
}

public class Main {
```

```
    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa("Maria", 25);
        // Chama automaticamente o método toString
        System.out.println(pessoa);
    }
}
```

Saída:

Nome: Maria, Idade: 25

Nesse exemplo, a classe Pessoa sobrescreve o método toString() para fornecer uma descrição mais legível do objeto ao invés da implementação padrão.

49. Explique o que faz o programa a seguir

```
package revisaob1;

import java.awt.Button;
import java.awt.Frame;
import java.awt.Label;
import java.awt.Panel;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class Janela extends Frame {
	private Panel panel;
	private TextField textField;
	private Button button;
	private Label label;

	public Janela() {
	this.setTitle("Janela");
	this.setSize(300,400);
	this.setVisible(true);
	panel = new Panel();
	this.add(panel);
	textField = new TextField(20);
	button = new Button("OK");
	label = new Label();
	panel.add(textField);
	panel.add(button);
	panel.add(label);
	button.addActionListener(new ActionListener(){
	@Override
	public void actionPerformed(ActionEvent ae) {
		label.setText("Ola " + textField.getText());
	}
```

```
        });

        }

}
package revisaob1;


public class RevisaoB1 {

        public static void main(String[] args) {
        Janela jl = new Janela();
}
```

O programa exibe "Olá" seguido do texto inserido pelo usuário.

Abre uma janela com o título “Janela, de tamanho 300x400, contendo um panel, um TextField de tamanho 20, um botão OK, e uma label que ao clicar no botão OK exibe “Ola e o que o usuário digitou no TextField.

50. Um funcionário possui nome e telefone, onde telefone pode conter ddd e numero, faça as classes para representar o funcionário e um exemplo de atribuir e ler os valores.

```
public class Telefone{
private String ddd;
private String numero;
public String getDdd(){
return ddd;
}
public void setDdd(String ddd){
this.ddd = ddd;
}
public String getNumero(){
return numero;
}
public void setNumero(String numero){
this.numero = numero;
}
public class Funcionario{
private String nome;
private Telefone telefone;
public String getNome(){
return nome;
}
public void setNome(String nome){
```

```
this.nome = nome;
}
public Telefone getTelefone(){
return telefone;
}
public void setTelefone(Telefone telefone){
this.telefone = telefone;
}
import java.util.Scanner;
public class Main{
public static void main(String[] args){
Funcionario funcionario = new Funcionario();
Scanner scanner = new Scanner();
System.out.println("Entre com o nome");
funcionario.setNome(Scanner.nextLine());
funcionario.setTelefone(new telefone());
System.out.println("Entre com o DDD");
funcionario.getTelefone().setDDD(Scanner.nextLine());
System.out.println("Entre com o Numero");
funcionario.getTelefone().setNumero(Scanner.nextLine());
}
```

51. Um banco possui dois tipos de conta, sendo elas conta corrente e conta especial, onde a conta especial possui um limite que pode ficar negativa, ambas as contas, possuem os métodos depositar, sacar, ver saldo e transferir. Implemente as classes necessárias.

```
public interface IConta {
void sacar(double valor) throws SaldoInsuficienteException ;
void depositar(double valor);
}
public abstract class Conta implements IConta{ // implementa métodos
da interface IConta
private String nome;
protected double saldo;
public Conta(String nome){
this.nome = nome;
this.saldo = 0;
}
public String getNome(){
return this.nome;
}
public double getSaldo(){
return this.saldo;
}
public abstract void sacar(double valor) throws
SaldoInsuficienteException;
public void depositar(double valor){
saldo = saldo+valor;
}
public void transferir(IConta favorecido, double valor) throws
TransferenciaException{
```

```
try{
this.sacar(valor);
favorecido.depositar(valor);
}catch (SaldoInsuficienteException ex){
throw new TransferenciaException("Transfêrencia não realizada.");
}
}
public class ContaCorrente extends Conta{
public ContaCorrente(String nome){
super(nome);
}
// throws Exception avisa que é um método não seguro (tem risco de não
acontecer)
// no caso usando exceção personalizada SaldoInsuficienteException,
que tem que ser criada numa nova classe que herda de Exception
@Override // override do método da classe Conta
public void sacar(double valor) throws SaldoInsuficienteException {
if (valor <= saldo){
saldo =- valor;
}else{
throw new SaldoInsuficienteException("Saldo insuficiente");
}
}
}
public class ContaEspecial extends Conta{
private double limite;
public ContaEspecial(String nome, double limite) {
super(nome); // chama o construtor da classe mãe
this.limite = limite;
}
public double getLimite() {
return limite;
}
public void setLimite(double limite) {
this.limite = limite;
}
// implementação expecífica para ContaEspecial
@Override // override do método da classe Conta
public void sacar(double valor) throws SaldoInsuficienteException {
if((saldo + limite) >= valor){
saldo -= valor;
}else{
throw new SaldoInsuficienteException("Saldo insuficiente");
}
}
```