Codificação Back-End – SA2

**Questionário 2**

**1)** As linguagens de programação adotam diferentes abordagens para resolver um problema. Uma mesma linguagem pode ter diferentes formas de codificação de programas.

Desta forma, indique as principais vantagens da linguagem C# considerando que ela oferece suporte à forma de programação orientada a objetos.

1. Reutilizar código e abstrair responsabilidades.
2. Depuração complexa e baixa portabilidade.
3. Programação de baixo nível e interpretada.
4. Fracamente tipada e alto custo de processamento.
5. Curva de aprendizado baixa e manutenção complexa.

**Dica**: lembre-se que as vantagens de uma linguagem de programação estão ligadas às suas características, e não à característica em si.

**Feedback**: a alternativa correta é A. Tendo em vista que a linguagem C# oferece suporte à Programação Orientada a Objetos, podemos concluir que as vantagens desse paradigma também podem ser aplicadas à linguagem C#. Entre as vantagens da POO, podemos citar a reutilização de código e a abstração de responsabilidades.

**2)** O paradigma de programação orientada a objetos (POO) surgiu como uma alternativa para aproximar a representação dos itens do mundo real para o mundo da programação. Indique se a afirmativa é verdadeira ou falsa.

Verdadeira Falsa

**Dica**: a abstração é um conceito que aproxima o mundo real da programação.

**Feedback**: a alternativa é verdadeira. O principal objetivo da programação é solucionar problemas do mundo real. Representar objetos do mundo real na programação pode facilitar a solução de problemas reais. Esse conceito é chamado de abstração e é um dos pilares da POO.

**3)** Na Programação Orientada a Objetos, existem elementos básicos que podem ser aplicados durante o desenvolvimento de um sistema. Indique quais são esses elementos.

1. Objeto, classe, interface, polimorfismo, abstração e encapsulamento.
2. Software, hardware e desenvolvedores.
3. Compiladores e interpretadores.
4. Visão, controlador e modelo.
5. Linearidade, funcionalidade e operacionalidade.

**Dica**: a Programação Orientada a Objetospossui blocos de construção formados pelos seus elementos característicos.

**Feedback**: a alternativa correta é A. Os elementos objeto, classe, interface, polimorfismo, abstração e encapsulamento são considerados como blocos de construção principais em POO.

**4)** Os modificadores de acesso são palavras-chave utilizadas para especificar a acessibilidade de uma propriedade ou de um método cujo acesso pode ser mais ou menos restritivo.

Considerando os modificadores de acesso existentes na linguagem C#, indique aquele que permite acesso apenas dentro da própria classe.

1. Public
2. Protected
3. Private
4. Security
5. Default

**Dica**: os modificadores de acesso são de três tipos: o que torna o acesso público, o que protege o acesso e o que restringe o acesso, tornando o dado privado.

**Feedback**: a alternativa correta é C. O modificador de acesso **private** permite que o tipo ou membro possa ser acessado somente na mesma *class* ou *struct*.

**5)** A Linguagem de Modelagem Unificada (UML) é uma linguagem-padrão que auxilia na modelagem de sistemas. Esta linguagem é expressa através de diagramas, e um dos tipos mais populares é o diagrama de classes.

Assim, e considerando um exemplo de um automóvel que possui cor, ano, placa e pode acelerar e frear, indique a forma correta do seu diagrama de classes.



|  |
| --- |
| Automovel |
| +cor  +ano  +placa |
| +Acelerar()  +Frear() |

|  |
| --- |
| +cor  +ano  +placa |
| Automovel |
| +Acelerar()  +Frear() |

|  |
| --- |
| +cor  +ano  +placa |
| +Acelerar()  +Frear() |
| Automovel |

|  |
| --- |
| +Acelerar()  +Frear() |
| +cor  +ano  +placa |
| Automovel |

|  |
| --- |
| Automovel  +cor  +ano  +placa  +Acelerar()  +Frear() |

**Dica**: lembre-se que o diagrama de classes deve mostrar de forma gráfica, simples e clara todos os elementos que compõem uma classe, seguindo o padrão UML.

**Feedback**: a alternativa correta é A.

|  |
| --- |
| Automovel |
| +cor  +ano  +placa |
| +Acelerar()  +Frear() |

O nome da classe é o primeiro dado a ser inserido; os atributos são posicionados na parte intermediária e podem ser classificados como – (privados) ou + (públicos). Por fim, na parte inferior, são inseridos os métodos da classe.

**6)** **(essa questão precisa de ajustes, foi deletada do quiz no AVA)** A Linguagem de Modelagem Unificada (UML) é uma linguagem-padrão que auxilia na modelagem de sistemas e é expressa através de diagramas. Cada diagrama é composto por elementos que possuem relação entre si. Um dos tipos mais populares é o diagrama de classes.

Analisando o diagrama de classes a seguir, indique qual dos conceitos de Orientação a Objetos está sendo aplicado.

Diagrama

Descrição gerada automaticamente

1. Herança
2. Polimorfismo
3. Encapsulamento
4. Interface
5. Sobrecarga

**Dica**: na UML, o sentido das setas indica a classe pai e as classes filhas.

**Feedback**: a alternativa correta é A. Na UML, as setas indicam herança. No exemplo, as classes **ContaCorrente** e **Poupanca** são filhas da classe **Banco** e, portanto, herdam os atributos e métodos da classe pai. Nas classes filhas, aparecem apenas os atributos e métodos próprios, além dos métodos reescritos ou com sobrecarga.

**AJUSTE:** a classe pai deve ser CONTA e não BANCO e a seta deve ser como a do print a seguir

Diagrama

Descrição gerada automaticamente

**7)** Com base nas classes Forma e Triangulo, indique o resultado do método Main() a seguir.

class Forma

{

public double base {get; set;}

public double altura {get; set;}

public double area {get; set;}

public virtual void CalcularArea()

{

area = base \* altura;

}

}

class Triangulo : Forma

{

public override void CalcularArea()

{

area = base \* altura / 2;

}

}

class Program

{

static void Main(string[] args)

{

Triangulo t = new Triangulo();

t.base = 6;

t.altura = 4;

t.CalcularArea();

Console.WriteLine(t.area);

}

}

1. 0
2. 6
3. 10
4. 12
5. 24

**Dica**: o polimorfismo permite que um método da classe pai seja reescrito de outra forma na classe filha.

**Feedback**: a alternativa correta é a D. O polimorfismo ocorre quando um método da classe pai é reescrito ou sobrescrito (*override*) ou recebe argumentos diferentes (sobrecarga) na classe filha. No exemplo, o método **CalcularArea** (**area = base \* altura;**), da classe pai **Forma**, é reescrito na classe filha **Triangulo** (**area = base \* altura / 2;**). Note a presença das palavras-chave **virtual** na classe pai e **override** na classe filha, necessárias para o **override**.

**8)** Com base na classe PedidoCompra, indique o resultado do método Main() a seguir.

class PedidoCompra

{

public int numero {get; set;}

public string produto {get; set;}

public int qtde {get; set;}

public double preco {get; set;}

public double desconto {get; set;}

public double total {get; set;}

public void CalcularPedido(int q, double p, double d)

{

this.qtde = q;

this.preco = p;

this.desconto = d;

this.total = q \* p - d;

}

}

class Program

{

static void Main(string[] args)

{

PedidoCompra p = new PedidoCompra();

p.CalcularPedido(3, 20.00, 5.00);

Console.WriteLine(p.total);

}

}

1. 0
2. 20
3. 35
4. 55
5. 60

**Dica**: lembre-se que classes concretas podem gerar inúmeros objetos, que herdam todos os atributos e métodos da classe. Lembre-se também que existe uma ordem de prioridades entre as operações aritméticas.

**Feedback**: a alternativa correta é a alternativa D. No código do exemplo, temos a implementação da classe **PedidoCompra** com atributos e método públicos. Dentro do método **Main**, temos o objeto **p**, instância da classe **PedidoCompra**, e a declaração dos valores das variáveis usadas no método **CalcularPedido**, sendo **q** = 3, **p** = 20.00 e **d** = 5.00. O resultado de **CalcularPedido** é armazenado na variável **total** e é mostrado no console (**Console.Writeline**). O valor de total é **q\*p-d**, ou seja, 3\*20.00-5.00 = 55.

**9)** Considerando a ideia de controle de um elevador, indique em qual andar o elevador ficou parado ao final da execução do programa a seguir.

class Elevador

{

public int andar {get; set;}

public void Subir()

{

andar = andar + 1;

}

public void Descer()

{

andar = andar - 1;

}

}

class Program

{

static void Main(string[] args)

{

Elevador e = new Elevador();

e.andar = 5;

e.Subir();

e.Subir();

e.Descer();

e.Descer();

e.Descer();

e.Subir();

e.Descer();

e.Descer();

Console.WriteLine(e.andar);

}

}

1. 0
2. 3º
3. 4º
4. 5º
5. 6º

**Dica**: lembre-se que, por padrão, o código é executado linha a linha.

**Feedback**: a alternativa correta é a B. A classe **Elevador** tem a variável **andar** e os métodos **Subir** e **Descer**. Dentro de **Main**, temos o objeto **e**, que é uma instância da classe **Elevador**, portanto, herda seus atributos e métodos. Ainda dentro de **Main**, é declarado que o andar de **e** é 5. Executando os comandos linha a linha, temos: 5, 6, 7, 6, 5, 4, 5, 4, 3; assim, o andar final é mostrado no console com o método **Console.writeLine**.

**10)** Com base nas classe Carro e CarroCorrida, indique o resultado do método Main() ou de seu comportamento.

class Carro

{

private int velocidade {get; set;}

public virtual void Acelerar()

{

velocidade = velocidade + 1;

}

public void Desacelerar()

{

velocidade = velocidade - 1;

}

}

class CarroCorrida : Carro

{

public override void Acelerar()

{

velocidade = velocidade + 10;

}

}

class Program

{

static void Main(string[] args)

{

Carro c = new Carro();

CarroCorrida cc = new CarroCorrida();

c.Acelerar();

c.Acelerar();

c.Desacelerar();

cc.Acelerar();

Console.WriteLine(c.velocidade);

}

}

1. 0
2. 1
3. 10
4. 11
5. Erro: propriedade inacessível devido ao seu nível de proteção (*private*)

**Dica**: lembre-se que os modificadores de acesso podem restringir ou impossibilitar o acesso de dados.

**Feedback**: o modificador de acesso ***private*** na classe pai (**private int velocidade {get; set;}**) não permite que classes filhas acessem o atributo velocidade da classe pai. O compilador, portanto, acusará erro e encerrará a compilação, não executando o programa.