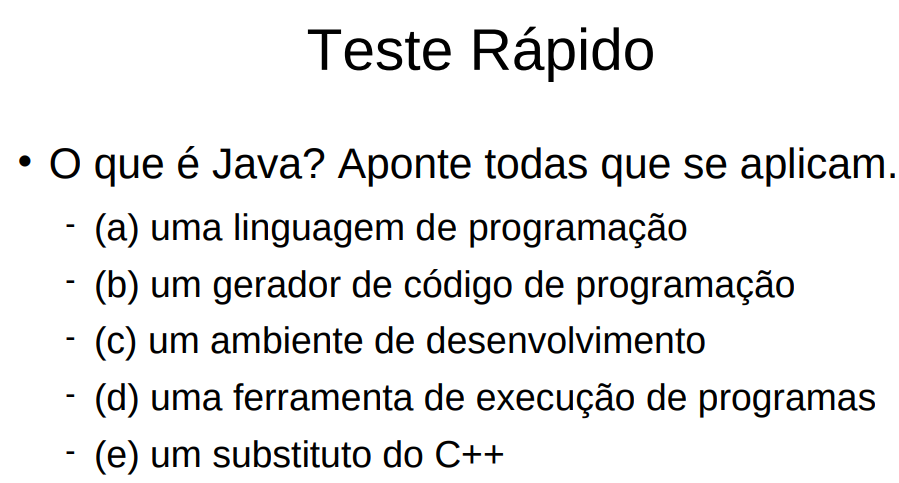
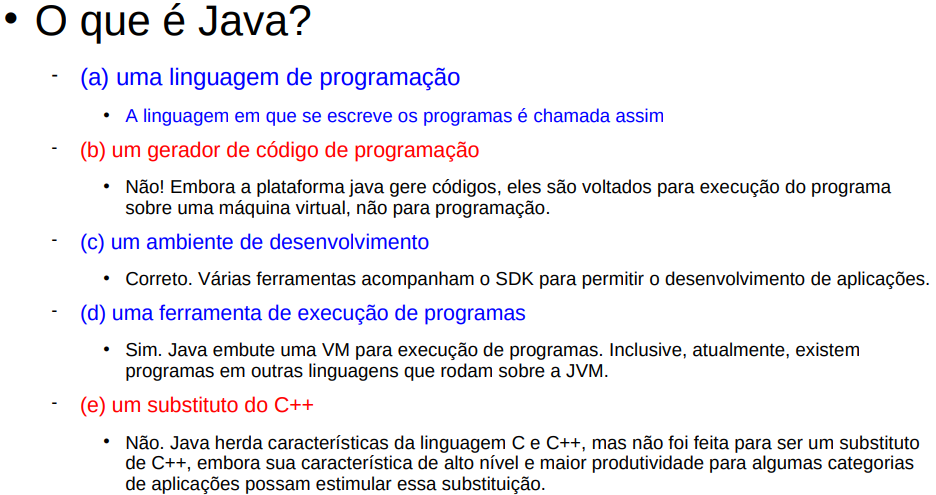
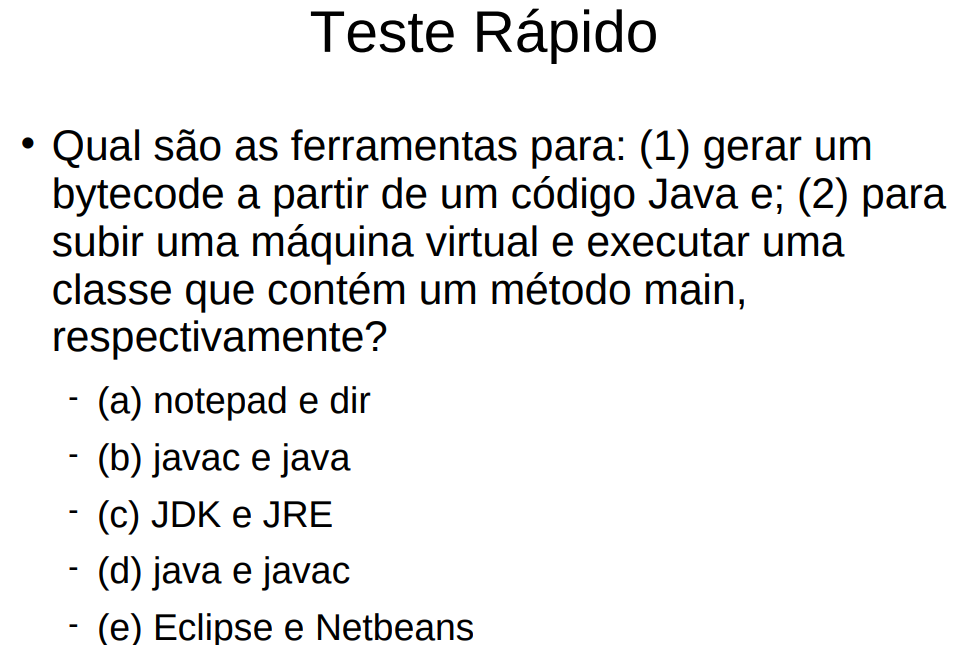
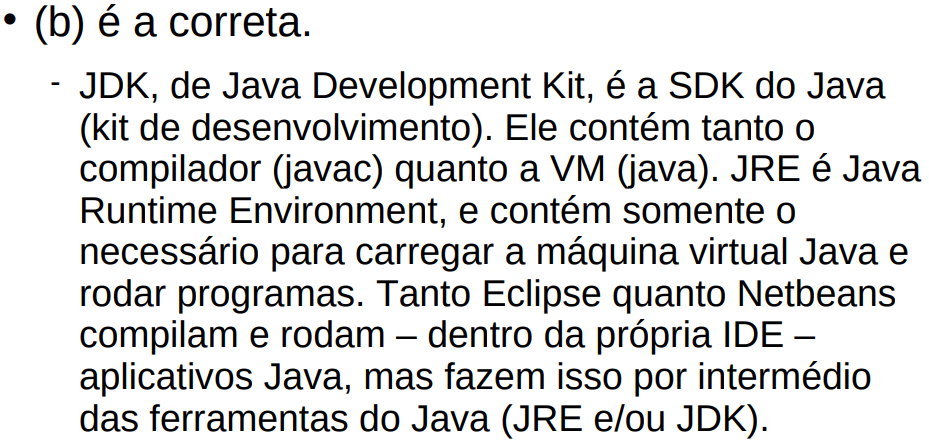
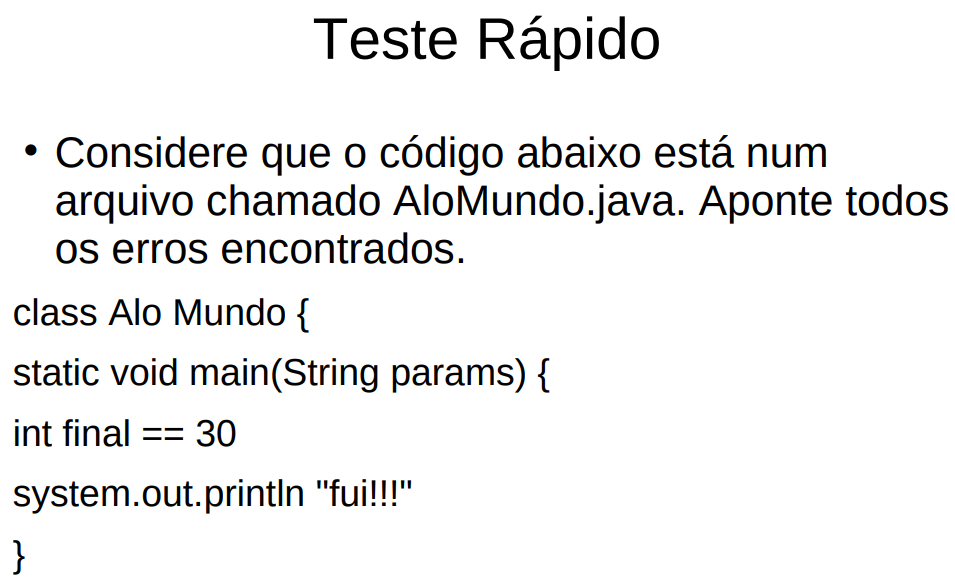
# Exercícios 3POB

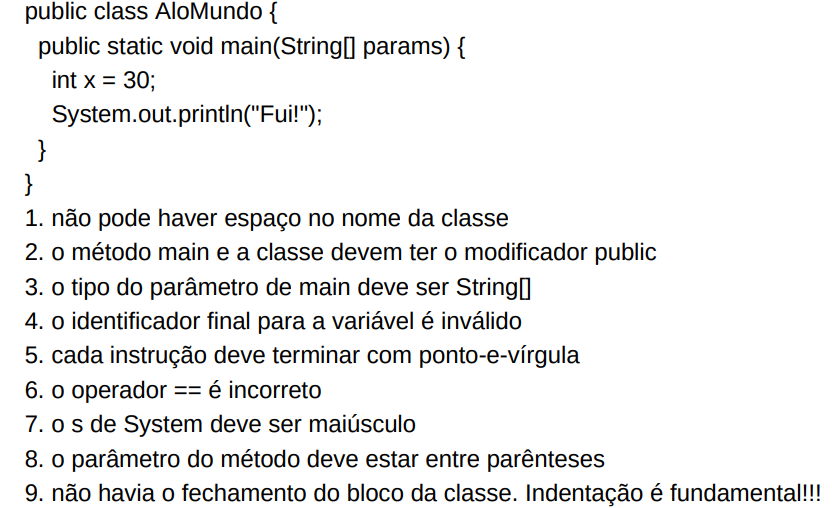


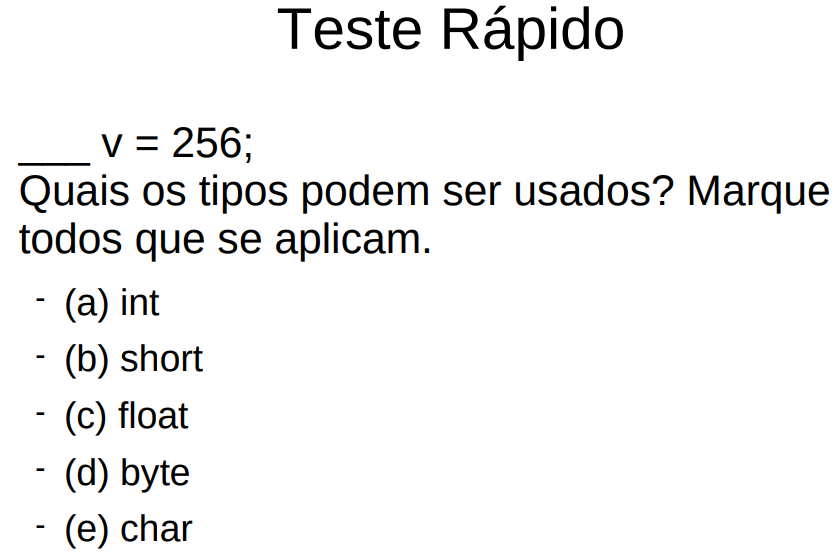


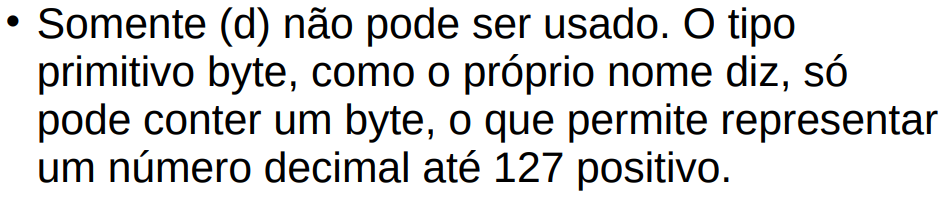












**Questão 1 – Primitivos**

byte b=4;

short s = ++b;

System.out.print(b + " " + s);

Escolha uma opção:

1. Não compilará!
2. Dará erro de execução!
3. 5 5
4. 4 4
5. 4 5

Resultará em

5 5

Repare que b é inicializado com o valor 4. Na segunda linha o valor de **b** é pré-incrementado. Isso significa que o valor de variável **b** é incrementado e depois o valor resultante é usado como operando da expressão restante, ou seja, o valor de variável **b** muda para 5 e esse mesmo valor é atribuído para **s**. A diferença de tipos aqui não é um problema porque se está atribuindo um byte para um short, um tipo de menor capacidade para um tipo de maior capacidade. Em geral, isso não dá problema.

A resposta correta é: 5 5

**Questão 2 – Primitivos**

int x = 10.0;

x /= 2;

System.out.print(x);

Escolha uma opção:

1. Erro de compilação.
2. 5
3. 5.0
4. Erro de execução.
5. 2.5

Dará erro de compilação: não pode converter um double para um inteiro.

O erro é sutil, mas existe. Ao colocar **10.0**, o Java entende que isso é um valor **double**. Ao se tentar atribuir a uma variável do tipo **int** - de menor capacidade - o compilador Java reclama.

O resultado matemático seria 5, não importa se inteiro ou com valor decimal zero. Matematicamente são iguais. Mas lembre-se: o compilador coloca valores em variáveis, de determinado tipo. Precisamos obedecer às limitações que esses tipos impõem. Alguns desses cuidados são importantes, como no caso de arredondamentos.

A resposta correta é: Erro de compilação.

**Questão 3 – Primitivos**

long y = 11;

y = y++ \* (y/2);

System.out.print(y);

Escolha uma opção:

1. 55
2. 72
3. Erro de execução!
4. 66
5. Erro de compilação!

Essa questão exige atenção para a diferença no uso do operador de pré-incremento e pós-incremento. No caso em questão foi usado o pós-incremento: o valor do y (11) é "guardado" como o primeiro operando para a multiplicação e depois é incrementado, passando a ser doze. Quando o Java vai avaliar a expressão do segundo operando da multiplicação, o valor de y já será 12. Ou seja, a multiplicação a ser realizada é:

11 \* (12/2) = 11 \* 6 = 66

A resposta correta é: 66

**Questão 4 - Primitivos**

double y = 10.2f;

boolean b= (y < 2 && (++y > 10.5));

System.out.print(y+" "+b);

Escolha uma opção:

1. 11.2 false
2. 10.2 true
3. 11.2 true
4. 10.2 false
5. Erro de compilação!

A primeira linha pode sugerir duas incorreções:

1. que a letra f, ao lado do número 10.2, pode ser um erro. Na verdade isto é correto. A letra f, que pode ser digitada em minúsculas também, indica que o valor 10.2 é um literal do tipo float.

2. o fato de eu estar atribuindo um float a um double seria um problema? Não. Um double em maior capacidade, então não haverá perda de precisão. Nesse caso o compilador Java converte implicitamente o valor em float para double.

Sobre a segunda linha: tenho uma expressão do tipo E lógico (&&), com curto-circuito. Em contraponto existe o E lógico sem curto-circuito (apenas um &). Usando com curto circuito, como o primeiro operando terá o valor falso - y não é menor do que 2, pois 10.2 não é menor do que 2 - haverá o curto circuito. Se o primeiro operando é falso, não importa se o segundo é verdadeiro ou falso, o resultado da expressão será falso. Dessa forma, o Java nem avalia o segundo operando, o da expressão **++y > 10.5**. Ou seja, o Y não será incrementado como se imagina.

Na sequência, o valor resultante da expressão lógico - no caso falso - é atribuído à variável b, que na linha seguinte tem de ser valor impresso.

Por que então é impresso 10.199999809265137 false?

Porque como **y** é uma variável do tipo double, o valor literal **10.2** é representado como uma aproximação binária desse valor em ponto flutuante. A representação em ponto flutuante binário, impõe, eventualmente, uma perda de precisão, que foi o que ocorreu nesse caso. Mas para efeitos práticos podemos considerar o abaixo:

**10.199999809265137 = 10.2**

A resposta correta é:

10.2 false

**Questão 1 – Desvios**

Qual o resultado da execução desse código?

int x=5;

boolean b=false;

if (x > 1 && (b=false))

System.out.println(x);

System.out.println("fim");

Escolha uma opção:

1. 5

fim

1. 5
2. Erro de execução!
3. Erro de compilação!
4. Fim

Imprime apenas "fim". A expressão **b=false**, na expressão que usa o E lógico da terceira linha, atribui **false** à variável **b** (que já era **false**) e resulta em **false**, o que faz com que a sentença aninhada ao if não seja executada. Na sentença que imprime fim, embora sua endentação sugira que esteja aninhada ao primeiro **if**, na verdade ela está no fluxo incondicional. Lembre-se: para aninhar sentenças a um bloco, deve-se colocá-las entre chaves {}

A resposta correta é: fim

**Questão 2 – Desvios**

Considere o código a seguir. Qual será o resultado?

char c='x';

int vo=0;

int co=0;

switch(c) {

default: co++;

case 'a': case 'e': case 'i': case 'o': case 'u':

++vo; break;

}

System.out.print(vo + co);

Escolha uma opção:

1. Não executa!
2. 0
3. Não compila!
4. 2
5. 1

2! Como o caractere 'x' não bate com nenhum case, a execução entra no bloco default e executa até encontrar o break. Na saída, está sendo feita a soma para depois imprimir. A expressão resulta em concatenação apenas quando um dos dois operados é string.

A resposta correta é: 2

**Questão 1 – Laços**

Considere o código abaixo:

int y=0;

for (int x=10; x > 0; --x) {

y += x;

}

System.out.println(x);

System.out.println(y);

Escolha uma opção:

1. 10

55

1. 0

55

1. Dá erro de execução!
2. Código não compila!

Código não compila! A variável x não pode ser acessada fora do bloco for, onde foi definida. Mas se somente y fosse impresso, daria 55 como resultado.

A resposta correta é: Código não compila!

**Questão – Laços**

Responda o que faz o código a seguir.

int x=2;

while (x < 100) {

if (++x % 2 == 1) continue;

System.out.println(x);

}

Escolha uma opção:

1. Imprime os valores pares de 4 até 100.
2. Imprime os valores pares de 3 até 99.
3. Imprime os valores pares de 3 até 100.
4. Imprime os valores pares de 4 até 98.
5. Imprime os valores pares de 2 até 100.

Sua resposta está correta.

Imprime os valores pares de 4 até 100.

O valor de **x** inicia com 2, mas como está sendo usado o operador de pré-incremento, o valor a ser computado na expressão **++x** será **3**, também o novo valor de **x**. Como 3 não é par (a expressão 3 % 2 resulta em 1), a instrução continue é executada, desviando a execução novamente para a instrução while, que avaliará a condição (x < 100). Na iteração seguinte, o valor da expressão ++x será 4, o resultado de 4 % 2 == 1 será false e a instrução continue não será executada, desviando para fora do **if**, onde ele encontra a instrução **System.out.println(x)**. Ou seja, imprime o valor pois ele é par.

Isso se repete até que x será incrementado para o valor 99 e o **continue** executado (o será pela última vez). Após a nova avaliação da condição do while, a expressão ++x resulta em 100, que sabemos resultado em false e levará à execução da instrução **System.out.println(x)**.

A resposta correta é: Imprime os valores pares de 4 até 100.

**Questão 1 - Vetores**

Considere o programa a seguir e indique o que será exibido no console.

int v [] = {5, 7, 8, 4};

for (int i=1; i < 4; i++) {

v[i]++;

System.out.print(v[i]+",");

}

Escolha uma opção:

1. 7,8,4,
2. 8,9,5,
3. 5,7,8,
4. 5,7,8,4
5. 1,2,3

Lembre-se que os vetores em java são baseados em 0, portanto os índices válidos para o vetor v vai de 0 a 3.

A resposta correta é: 8,9,5,

**Questão 2 - Vetores**

Considere o código a seguir e indique o que será impresso no console:

double [] v = {20.5, 13.5, 5.6, 2};

for (int c=0; c<4; c++) {

v[c]-=v[c+1];

System.out.println(v[c]);

}

Escolha uma ou mais:

1. 7.0

7.9

3.5999999999999996

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4 at teste.Main.main(Main.java:9)

1. Não compila.
2. -7.0

-7.9

-3.6

1. 34

19.1

7.6

1. Compila mas dá exceção ao executar.

O código compila pois, embora 2 seja um literal inteiro, ele pode ser atribuído a um double.

O valor de uma posição recebe o seu próprio valor subtraído do valor do elemento sucessor.

Na última iteração, para c igual a 3, tenta-se acessar o elemento de índice 4 (v[c+1]), que não existe.

As respostas corretas são:

Compila mas dá exceção ao executar.;

7.0  
7.9  
3.5999999999999996  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4  
    at teste.Main.main(Main.java:9)

**Questão 1 – Fundamentos**

Diferencie a classe de um objeto do tipo de um objeto

A classe de um objeto define uma implementação para o objeto. O tipo do objeto define um conjunto de mensagens que podem ser recebidas por um objeto. Todas as mensagens que podem ser recebidas por um objeto são chamadas a interface do objeto.

**Questão 2 – Fundamentos**

Diferencie herança de classe de herança de tipo

A herança de classe permite a definição da implementação de um objeto em termos da implementação de outro objeto. Já herança de tipo refere-se à capacidade de usar um objeto no lugar de outro, sendo ambos desse mesmo tipo.

**Questão 3- Fundamentos**

O que é herança de classe?

A herança de classe é um princípio que permite o reuso através da criação de uma classe que reaproveita o código definido numa superclasse.

**Questão 4 – Fundamentos**

Quais os dois mecanismos de reuso de código em POO mais comuns? Diferencie o uso de um frente ao outro.

1. Herança de classe - Na herança de classe (também conhecida como reuso por caixa-branca) cria um reuso estático (em tempo de compilação), sendo portanto um relacionamento forte.
2. Composição de Objetos usando delegação - Na Composição podemos criar o reuso (também conhecido como reuso por caixa-preta) através do uso da delegação de responsabilidades. Além disso, possibilita uma dependência mais fraca.

**Questão 5 – Fundamentos**

O que é delegação?

É a transferência da responsabilidade de um objeto de realizar uma operação (ou parte dela), para outro objeto relacionado ao primeiro (tipicamente por composição).

**Questão 6 – Fundamentos**

Qual a diferença entre Aplicativos, Bibliotecas de Classes e Frameworks.

Aplicativos: software de usuário final

Bibliotecas de Classes: classes que realizam funcionalidades para reuso em outras aplicações. O controle do fluxo de execução fica por conta do software aplicativo, que chama a funcionalidade de um objeto ou de uma classe contida na biblioteca de classes.

Framework: classes reutilizáveis que impõem um desenho específico (arquitetura) para o projeto de uma aplicação, voltadas para um domínio de aplicações. O fluxo de controle fica por conta das classes do Framework (Inversão de Controle), e o desenvolvedor que o utiliza deve implementar determinadas classes ou tipos específicos que serão os pontos de extensão para o framework.

**Considere a afirmação:**

− “O paradigma OO é superior ao paradigma procedural pois garante o reuso.”

V ou F, justificando

Falso. Embora a OO ofereça mais recursos para o reuso, como a herança e a composição de objetos, não há garantia que se fará reuso a menos que um bom projeto de software seja realizado..

**Teste Rápido**

Um objeto armazena os itens comprados, calcula os impostos, gera a fatura em PDF, e realiza o envio de informações contábeis para o sistema de contabilidade. Que princípio de OO está sendo violado?

Coesão. Esse objeto está com muitas responsabilidades, apontando a necessidade de distribuição dessas responsabilidades para outros objetos.

**Teste Rápido**

A herança quebra o encapsulamento. V ou F, e justifique.

Verdadeiro. O reuso por herança expõe, para a subclasse (ou filha), detalhes da implementação herdados da super classe (ou pai), portanto quebrando o encapsulamento. Entretanto, isso não invalida o uso da herança, contanto que seja aplicada com muito cuidado.

**Questão 1 – Fixação 01**

Escreva uma classe chamada Ponto, que tenha duas variáveis privadas de instância, x e y, do tipo inteiro. Crie 4 métodos: getX e getY – que retornam o valor da respectiva variável de instância – e setX e setY – que recebe um inteiro como parâmetro e modifica a respectiva variável.

public class Ponto {  
 private int x;  
 private int y;  
 public int getX() {return x;}  
 public int getY() {return y;}  
 public void setX(int v) {x=v;}  
 public void setY(int v) {y=v;}  
}

**Questão 2 – Fixação 01**

Considere o código a seguir:

public class Jogador {

private String nome;

private int pontos;

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

public void setPontos(int p) {this.pontos = p;}

public String getNome() {return nome;}

public int getPontos() {return pontos;}

}

1. Complete a linha de código abaixo de forma a definir uma variável de referência e instanciar um objeto da classe anterior:

\_\_\_\_\_\_\_\_ variavel = new \_\_\_\_\_\_\_\_ ();

Jogador variavel = new Jogador();

1. Complete a linha de código abaixo de forma a configurar o nome do jogador como Pelé Jogador j1;

j1 = new Jogador();

j1.\_\_\_\_\_\_\_\_

j1.setNome(“Pelé”);

1. Complete a linha de código abaixo de forma a configurar o de nome Joaquim com 50 pontos:

Jogador j1 = new Jogador();

j1.setNome("Joaquim");

j1.\_\_\_\_\_\_\_\_

j1.setPontos(50);

**Questão 1 – Fixação 02**

Crie uma classe Funcionario, que tenha os seguintes atributos: matrícula, nome e salario. Crie um construtor que receba argumentos para os três parâmetros. A classe funcionário define um valor de classe privado, chamado salarioMinimo, que recebe inicialmente o valor do salário mínimo vigente. Encapsule os campos de tal forma que as seguintes regras sejam respeitadas:

* A matrícula deve ser um valor inteiro entre 1 e 1000. Após a criação, a matrícula é um atributo de leitura apenas.
* O nome deve ter mais do que 3 caracteres e até 40 caracteres. Embora o nome possa ser recebido em caixa baixa, a exibição deve sempre ser feita em caixa alta. Atributo nome de leitura e gravação;
* O salário não pode ser maior que 18000,00 nem menor do que o salário mínimo. Atributo salário de leitura e gravação;
* Caso alguma regra seja violada, use os valores-padrão:
  + matricula=999
  + nome="FULANO DA SILVA"
  + salario = salarioMinimo

// Arquivo Funcionario.java

public class Funcionario {  
    private static double salarioMinimo=460.0;  
    private int matricula;  
    private String nome;  
    private double salario;  
    public Funcionario(int m,String n,double s) {  
        setMatricula(m); setNome(n); setSalario(s);  
    }  
    private void setMatricula(int m) {  
        if(m<1 || m>1000) matricula=999; else matricula=m;  
    }  
    public void setNome(String n) {  
        int tam=n.length();  
        if(tam>3 && tam<=40) nome=n; else nome="FULANO DA SILVA";  
    }  
    public void setSalario(double s) {  
        if(s<salarioMinimo || s>18000.00) salario=salarioMinimo;  
        else salario=s;  
    }  
    public int getMatricula() { return matricula; }  
    public String getNome() { return nome.toUpperCase(); }  
    public double getSalario() { return salario; }  
}

**Questão 2 – Fixação 03**

// Dada a classe:

class Pessoa {

private String nome;

public Pessoa(String n) { nome=n; }

public void setNome(String nome) { nome=nome; }

public String getNome() {

return nome.toLowerCase()+nome.length();

}

}

// e outro código que acessa Pessoa:

Pessoa p = new Pessoa();

p.setNome("Ana");

System.out.print(p.getNome());

Escolha uma opção:

1. ANA3
2. O código dá erro de execução!
3. O código não compila!
4. ana4
5. ana3

O código não compila, pois, a classe Pessoa não tem construtor padrão, e o código que instância Pessoa tenta executá-lo. Quando se define um construtor, o java não mais fornece o construtor padrão.

Além disso, há um erro de lógica: o nome do parâmetro setNome é ambíguo com o nome da variável de instância. Deve-se tirar a ambiguidade com a instrução

this: this.nome=nome;

Sem esses erros, o resultado do código seria: "ana3"

A resposta correta é: O código não compila!

**Questão 3 – Fixação 02**

// Considere as seguintes classes:

class Pai {

public String nome;

private int idade;

private void setNome(String n) { nome=n; }

public int getIdade() { return idade; }

}

class Filho extends Pai {

private boolean castigo=false;

public boolean getCastigo() { return castigo; }

public void setNome(String n) { nome=n; }

}

// E um código que acessa a classe Filho

Filho f=new Filho();

// Que membros de f posso acessar?

Escolha uma ou mais:

1. método getCastigo (da classe Filho)
2. idade (da classe Pai)
3. variável nome (herdado da classe Pai)
4. método getIdade (herdado da classe Pai)
5. método setNome (da classe Filho)

* variável nome (herdado da classe Pai)
* método getIdade (herdado da classe Pai)
* método getCastigo (da classe Filho)
* método setNome (da classe Filho)

As respostas corretas são: variável nome (herdado da classe Pai), método getIdade (herdado da classe Pai), método getCastigo (da classe Filho), método setNome (da classe Filho)

**Questão 4 – Fixação 02**

Considere o código a seguir:

public class Main {

public static void main(String args[]) {

Loja loja1 = new Loja(8);

Loja loja2= new Loja(4);

int i = loja1.getValorMais(2);

double d = loja2.getValorMais(3.6);

System.out.println(i + " " + d);

}

}

class Loja {

int valor;

static int num\_vend=5;

public Loja(int val) { valor = val; }

public int getValorMais(int mais) {

num\_vend=2;

return (valor + mais);

}

public double getValorMais(double mais) {

return (mais+valor+num\_vend);

}

}

Escolha uma opção:

1. Dá erro em tempo de execução!
2. 8 12.6
3. 10 12.6
4. Não compila!
5. 10 9.6

Duas questões estão envolvidas neste exercício:

1. a escolha do método sobrecarregado que será invocado depende do tipo do argumento passado. Na primeira chamada à setMais, passa-se um inteiro, portanto a versão que recebe um int será a chamada. Na segunda, a que recebe um double é que será executado.

2. a invocação do método setMais(int) no objeto loja1 altera o valor da variável de classe num\_vend. Assim, ao invocar setMais(double) no objeto loja2, como o valor da variável de classe é o mesmo para todas as instâncias, o valor de num\_vend é 2.

A resposta correta é: 10 9.6

**Questão 5 – Fixação 02**

Considere o código abaixo:

public class Main {

public static void main(String[] args) {

/\* O método de classe obterFuncionarios da classe Fabrica retorna um vetor de 100 objetos da classe Funcionario \*/

Funcionario[] lista = Fabrica.obterFuncionarios();

// complete aqui! TRECHO 1

}

}

class Funcionario {

private static double INICIAL=460.00;

private int matricula;

private String nome;

private double salario;

public Funcionario(int m,String n) {

matricula=m; nome=n; salario=INICIAL;

}

public void aumentarSalario(double fatorAumento) {

// completar aqui! TRECHO 2

}

}

Implemente os trechos 1 e 2 de tal forma que, após a execução do trecho 1, todos os 100 funcionários tenham seus salários aumentados em 5%. Nenhum outro código pode ser inserido que não seja nos trechos indicados.

/\* TRECHO 1 com FOR curto \*/  
public class Main {  
  public static void main(String[] args) {  
    Funcionario[] lista = Fabrica.obterFuncionarios();  
    for( Funcionario func : lista ) {  
      func.aumentarSalario( 1.05 );  
    }  
  }  
}  
class Funcionario {  
    private static double INICIAL=460.00;  
    private int matricula;  
    private String nome;  
    private double salario;  
    public Funcionario(int m,String n) {  
        matricula=m; nome=n; salario=INICIAL;  
    }  
    public void aumentarSalario(double fatorAumento) {  
      salario = salario \* fatorAumento;  
    }

}

/\* TRECHO 2 com FOR com contador \*/  
  
public class Main {  
  public static void main(String[] args) {  
    /\* O método de classe obterFuncionarios da classe  
       Fabrica retorna um vetor de 100 objetos da  
       classe Funcionario \*/  
    Funcionario[] lista = Fabrica.obterFuncionarios();  
    for( int c = 0; c < lista.length ; c++ ) {  
      Funcionario func = lista[c];  
      func.aumentarSalario( 1.05 );  
    }  
  }  
}  
class Funcionario {  
    private static double INICIAL=460.00;  
    private int matricula;  
    private String nome;  
    private double salario;  
    public Funcionario(int m,String n) {  
        matricula=m; nome=n; salario=INICIAL;  
    }  
    public void aumentarSalario(double fatorAumento) {  
      salario = salario \* fatorAumento;  
    }  
}

