

- 1) Escolha a opção adequada ao tentar compilar e rodar o código a seguir:

```
1 class Teste {  
2     public static void main(String[] args) {  
3         for (int i = 0; i < 20; i++) {  
4             System.out.println(i);  
5         }  
6         int i = 15;  
7         System.out.println(i);  
8     }  
9 }
```

- a) Erro de compilação na linha 6. A variável `i` não pode ser redeclarada.
- b) Erro de compilação na linha 7. A variável `i` é ambígua.
- c) Compila e roda, imprimindo de 0 até 19 e depois 15.
- d) Compila e roda, imprimindo de 0 até 19, depois ocorre um erro de execução na linha 6.
- e) Compila e roda, imprimindo de 0 até 19 e depois 19 novamente.

2) Escolha a opção adequada ao tentar compilar e rodar o código a seguir:

```
1 class Teste {  
2     static int x = 15;  
3  
4     public static void main(String[] x) {  
5         x = 200;  
6         System.out.println(x);  
7     }  
8 }
```

- a) O código compila e roda, imprimindo 200.
- b) O código compila e roda, imprimindo 15.
- c) O código não compila.
- d) O código compila mas dá erro em execução.

3) Escolha a opção adequada ao tentar compilar e rodar o código a seguir:

```
1 class Teste {  
2     static int i = 3;  
3  
4     public static void main(String[] a) {  
5         for (new Teste().i = 10; new Teste().i < 100;  
6             new Teste().i++) {  
7                 System.out.println(i);  
8             }  
9         }  
10    }
```

- a) Não compila a linha 4.
- b) Não compila a linha 5.
- c) Compila e imprime 100 vezes o número 3.
- d) Compila e imprime os números de 10 até 99.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir sem nenhum parâmetro na linha de comando, como `java D`:

```
1 package a.b.c;
2
3 import java.util.*;
4
5 class D {
6     public static void main(String[] args) {
7         ArrayList<String> lista = new ArrayList<String>();
8
9         for (String arg : args) {
10             if (new E().existe(arg))
11                 lista.add(arg);
12         }
13     }
14 }
15
16 import java.io.*;
17
18 class E {
19     public boolean existe(String nome) {
20         File f = new File(nome);
21         return f.exists();
22     }
23 }
```

- a) O arquivo não compila.
- b) O arquivo compila mas dá erro de execução pois o array é nulo.
- c) O arquivo compila mas dá erro de execução pois o array tem tamanho zero.
- d) Roda e imprime `false`.
- e) Roda e imprime `true`.
- f) Roda e não imprime nada.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class Teste {  
2     int Teste = 305;  
3  
4     void Teste() {  
5         System.out.println(Teste);  
6     }  
7  
8     public static void main(String[] args) {  
9         new Teste();  
10    }  
11 }
```

- a) O código não compila: erros nas linhas 24, 25 e 26.
- b) O código não compila: erro na linha 25.
- c) O código não compila: erros nas linhas 24 e 26.
- d) O código compila e, ao rodar, imprime 305.
- e) O código compila e não imprime nada.
- f) O código compila e, ao rodar, imprime uma linha em branco.

3) Escolha a opção adequada ao tentar compilar o arquivo a seguir:

```
1 package br.com.teste;  
2  
3 import java.util.ArrayList;
```

- a) Erro na linha 1: definimos o pacote mas nenhum tipo.
- b) Erro na linha 3: importamos algo desnecessário ao arquivo.
- c) Compila sem erros.

4) Escolha a opção adequada ao tentar compilar o arquivo A.java:

```
1 class A implements B {  
2 }  
3 public interface B {  
4 }  
5 class C extends A {  
6 }  
7 class D extends A, implements B {  
8 }
```

- a) Não compila: erro na linha 7.
- b) Não compila: erro na linha 1.
- c) Não compila: erro na linha 1, 5 e 7.
- d) Não compila: erro na linha 3.
- e) Compila.

1) Qual é uma assinatura válida do método `main` para executar um programa java?

- a) `public static void main(String... args)`
- b) `public static int main(String[] args)`
- c) `public static Void main(String []args)`
- d) `protected static void main(String[] args)`
- e) `public static void main(int argc, String[] args)`

- 2) Escolha a opção adequada para compilar e rodar o arquivo A.java, existente no diretório b:

```
1 package b;
2 class A {
3     public static void main(String[] args) {
4         System.out.println("rodando");
5     }
6 }
```

- a) javac A e java A
- b) javac A.java e java A
- c) javac b/A.java e java A
- d) javac b/A.java e java b.A
- e) javac b.A.java e java b.A
- f) javac b.A e java b.A

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         System.out.println(args);  
4         System.out.println(args.length);  
5         System.out.println(args[0]);  
6     }  
7 }
```

- a) Não compila: array não possui membro `length`.
- b) Não compila: o método `println` não consegue imprimir um array.
- c) Ao rodar sem argumentos, ocorre uma `NullPointerException` na linha 5.
- d) Ao rodar sem argumentos, ocorre uma `NullPointerException` na linha 4.
- e) Ao rodar sem argumentos, são impressos os valores “1” e “A”.
- f) Ao rodar com o argumento “certificacao”, são impressos os valores “2” e “A”.

- 4) Escolha a opção adequada para rodar a classe A.java presente no diretório b, que foi compactado em um arquivo chamado programa.jar, sendo que não existe nenhum arquivo de manifesto:

```
1 package b;  
2 class A {  
3     public static void main(String[] args) {  
4         System.out.println(args[0]);  
5     }  
6 }
```

- a) java jar programa.jar
- b) java jar programa.jar b.A
- c) java -jar programa.jar
- d) java -jar programa.jar b.A
- e) java -cp programa.jar
- f) java -cp programa.jar b.A

- 5) Escolha a opção adequada para compilar a classe `A.java`, definida como no pacote `b` presente no diretório `b`, e adicionar também o arquivo `programa.jar` na busca de classes durante a compilação. Lembre-se que `.` significa o diretório atual.
- a) `javac -cp b.A.java -cp programa.jar`
 - b) `javac -jar programa.jar b.A.java`
 - c) `javac -cp programa.jar:b A.java`
 - d) `javac -cp programa.jar:.. b.A.java`
 - e) `javac -cp . -cp programa.jar`
 - f) `javac -jar programa.jar:.. b/A.java`
 - g) `javac -cp programa.jar:b b/A.java`
 - h) `javac -cp programa.jar:.. b/A.java`

- 1) Escolha a opção adequada ao tentar compilar e rodar o `Teste`. Arquivo no diretório atual:

```
1 import modelo.Cliente;
2 class Teste {
3     public static void main(String[] args) {
4         new Cliente("guilherme").imprime();
5     }
6 }
```

Arquivo no diretório `modelo`:

```
1 package modelo;
2
3 class Cliente {
4     private String nome;
5     Cliente(String nome) {
6         this.nome = nome;
7     }
8     public void imprime() {
9         System.out.println(nome);
10    }
11 }
```

- a) Não compila: erro na classe `Teste`.
- b) Não compila: erro na classe `Cliente`.
- c) Erro de execução: método `main`.
- d) Roda e imprime “Guilherme”.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import modelo.basico.Cliente;
2 import modelo.avancado.Cliente;
3
4 class Teste {
5     public static void main(String[] args) {
6         System.out.println("Bem vindo!");
7     }
8 }
```

- a) O código não compila, erro ao tentar importar duas classes com o mesmo nome.
- b) O código compila, mas ao rodar dá erro por ter importado duas classes com o mesmo nome.
- c) O código compila e roda imprimindo `Bem vindo!`, uma vez que nenhuma das classes importadas é usada no código, não existe ambiguidade.

- 3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sabendo que existem duas classes Cliente, uma no pacote basico e outra no pacote avancado:

```
1 import modelo.basico.Cliente;
2 import modelo.avancado.*;
3
4 class Teste {
5     public static void main(String[] args) {
6         System.out.println("Bem vindo!");
7     }
8 }
```

- a) O código não compila, erro ao tentar importar duas classes com o mesmo nome.
- b) O código compila mas ao rodar dá erro por ter importado duas classes com o mesmo nome.
- c) O código compila e roda imprimindo Bem vindo!.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import modelo.basico.Cliente;
2 import modelo.basico.Cliente;
3
4 class Teste {
5     public static void main(String[] args) {
6         System.out.println("Bem vindo!");
7     }
8 }
```

- a) O código não compila, erro ao tentar importar duas classes com o mesmo nome.
- b) O código compila, mas ao rodar dá erro por ter importado duas classes com o mesmo nome.
- c) O código compila e roda imprimindo `Bem vindo!`, uma vez que não há ambiguidade.

5) Escolha a opção adequada ao tentar compilar os arquivos a seguir:

a/A.java:

```
1 package a;  
2 class A {  
3     b.B variavel;  
4 }
```

a/C.java:

```
1 package a;  
2 class C {  
3     b.B variavel;  
4 }
```

a/b/B.java:

```
1 package a.b;  
2 class B {  
3 }
```

- a) Erro de compilação somente no arquivo A.
- b) Erro de compilação somente no arquivo B.
- c) Erro de compilação somente no arquivo C.
- d) Erro de compilação nos arquivos A e B.
- e) Erro de compilação nos arquivos A e C.
- f) Erro de compilação nos arquivos B e C.
- g) Compila com sucesso.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 package A;  
2 class B{  
3     public static void main(String[] a) {  
4         System.out.println("rodei");  
5     }  
6 }
```

- a) Não compila: a variável do método `main` deve se chamar `args`.
- b) Não compila: pacote com letra maiúscula.
- c) Compila mas não roda: a classe B não é pública.
- d) Compila e roda.

7) Escolha a opção adequada ao tentar compilar os arquivos a seguir:

a/A.java:

```
1 package a;
2 public class A {
3     public static final int VALOR = 15;
4     public void executa(int x) {
5         System.out.println(x);
6     }
7 }
```

b/B.java:

```
1 package b;
2 import static a.A.*;
3 class B{
4     void m() {
5         A a = new A();
6         a.executa(VALOR);
7     }
8 }
```

- a) B não compila: erro na linha 2.
- b) B não compila: erro na linha 5.
- c) B não compila: erro na linha 6.
- d) Tudo compila.

8) Escolha a opção adequada ao tentar compilar os arquivos a seguir:

a/A.java:

```
1 package a;
2 public class A {
3     public static final int VALOR = 15;
4     public void executa(int x) {
5         System.out.println(x);
6     }
7 }
```

b/B.java:

```
1 package b;
2 import a.A;
3 static import a.A.*;
4 class B{
5     void m() {
6         A a = new A();
7         a.executa(VALOR);
8     }
9 }
```

- a) B não compila: erro na linha 3.
- b) B não compila: erro na linha 5.
- c) B não compila: erro na linha 6.
- d) Tudo compila.

9) Escolha a opção adequada ao tentar compilar os arquivos a seguir:

A.java:

```
1 public class A {  
2     public static final int VALOR = 15;  
3     public void executa(int x) {  
4         System.out.println(x);  
5     }  
6 }
```

b/B.java:

```
1 package b;  
2 import static A.*;  
3 class B{  
4     void m() {  
5         A a = new A();  
6         a.executa(VALOR);  
7     }  
8 }
```

- a) Não compila
- b) Tudo compila.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int  
4             idade  
5             = 100;  
6         System.out.println(idade);  
7     }  
8 }
```

- a) O código não compila: erros a partir da linha que define uma variável do tipo int.
- b) O código não compila: a variável idade não foi inicializada, mas foi usada em System.out.println.
- c) O código compila e imprime o.
- d) O código compila e imprime 100.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int idade;  
4         if(args.length > 0) {  
5             idade = Integer.parseInt(args[0]);  
6         } else {  
7             System.err.println("Por favor passe sua idade como  
8                             primeiro parâmetro");  
9         }  
10        System.out.println("Sua idade é " + idade);  
11    }  
12 }
```

- a) Não compila: erro na linha que tenta acessar a variável `idade`.
- b) Compila e imprime o ou a idade que for passada na linha de comando.
- c) Compila e imprime a idade que for passada na linha de comando.
- d) Compila e imprime a mensagem de erro ou “Sua idade é ”e a idade.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         boolean array = new boolean[300];  
4         System.out.println(array[3]);  
5     }  
6 }
```

- a) Imprime true.
- b) Imprime false.
- c) Imprime 0.
- d) Imprime -1.
- e) Imprime null.
- f) Não compila.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         boolean[] array = new boolean[300];  
4         System.out.println(array[3]);  
5     }  
6 }
```

- a) Imprime true.
- b) Imprime false.
- c) Imprime 0.
- d) Imprime -1.
- e) Imprime null.
- f) Não compila.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         boolean argumentos;  
4         if(args.length > 0)  
5             argumentos = 1;  
6         else  
7             argumentos = 0;  
8         System.out.println(argumentos);  
9     }  
10 }
```

- a) Não compila: o método de impressão não recebe boolean.
- b) Não compila: atribuição inválida.
- c) Não compila: o método length de array não é uma propriedade.
- d) Não compila: o método length de String[] não é uma propriedade.
- e) Compila e imprime 0 ou 1.
- f) Compila e imprime false ou true.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int n = 09;  
4         int m = 03;  
5         int x = 1_000;  
6         System.out.println(x - n + m);  
7     }  
8 }
```

- a) Não compila: erro na linha que declara `n`.
- b) Não compila: erro na linha que declara `x`.
- c) Não compila: erro na linha que declara `m`.
- d) Compila e imprime um número menor que 1000.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(char c='a';c <= 'z';c++) {  
4             System.out.println(c);  
5         }  
6     }  
7 }
```

- a) Não compila: não podemos somar um em um caractere.
- b) Não compila: não podemos comparar caracteres com <.
- c) Compila e imprime o alfabeto entre a e z, inclusive.

8) Qual das palavras a seguir não é reservada em Java?

- a) strictfp
- b) native
- c) volatile
- d) transient
- e) instanceof

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         boolean BOOLEAN = false;  
4         if(BOOLEAN) {  
5             System.out.println("Sim");  
6         }  
7     }  
8 }
```

- a) Não compila: não podemos declarar uma variável com o nome de uma palavra reservada.
- b) Não compila: não podemos declarar uma variável iniciando com letras maiúsculas.
- c) Compila e roda, imprimindo Sim.
- d) Compila e roda, não imprimindo nada.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x = 15;  
4         int y = x;  
5         y++;  
6         x++;  
7         int z = y;  
8         z--;  
9         System.out.println(x + y + z);  
10    }  
11 }
```

- a) Imprime 43.
- b) Imprime 44.
- c) Imprime 45.
- d) Imprime 46.
- e) Imprime 47.
- f) Imprime 48.
- g) Imprime 49.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     int v = 15;  
3 }  
4 class A {  
5     public static void main(String[] args) {  
6         B x = new B();  
7         B y = x;  
8         y.v++;  
9         x.v++;  
10        B z = y;  
11        z.v--;  
12        System.out.println(x.v + y.v + z.v);  
13    }  
14 }
```

- a) Imprime 43.
- b) Imprime 44.
- c) Imprime 45.
- d) Imprime 46.
- e) Imprime 47.
- f) Imprime 48.
- g) Imprime 49.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     int c;  
3     void c(int c) {  
4         c = c;  
5     }  
6 }  
7 class A {  
8     public static void main(String[] args) {  
9         B b = new B();  
10        b.c = 10;  
11        System.out.println(b.c);  
12        b.c(30);  
13        System.out.println(b.c);  
14    }  
15 }
```

- a) Não compila: conflito de nome de variável membro e método em B.
- b) Não compila: conflito de nome de variável membro e variável local em B.
- c) Compila e roda, imprimindo 10 e 30.
- d) Compila e roda, imprimindo outro resultado.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2  
3 }  
4 class A {  
5     public static void main(String[] args) {  
6         B b;  
7         for(int i = 0;i < 10;i++)  
8             b = new B();  
9         System.out.println("Finalizando!");  
10    }  
11 }
```

- a) Não compila.
- b) Compila e garbage coleta 10 objetos do tipo B na linha do System.out.
- c) Compila e não podemos falar quantos objetos do tipo B foram garbage coletados na linha do System.out.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2  
3 }  
4 class A {  
5     public static void main(String[] args) {  
6         B b = new B();  
7         for(int i = 0;i < 10;i++)  
8             b = new B();  
9         System.out.println("Finalizando!");  
10    }  
11 }
```

- a) Não compila.
- b) Compila e 10 objetos do tipo B podem ser garbage coletados ao chegar na linha do System.out.
- c) Compila e 11 objetos do tipo B podem ser garbage coletados ao chegar na linha do System.out.
- d) Compila e garbage coleta 11 objetos do tipo B na linha do System.out.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2  
3 }  
4 class A {  
5     public static void main(String[] args) {  
6         B[] bs = new B[100];  
7         System.out.println("Finalizando!");  
8     }  
9 }
```

- a) Compila e 100 objetos do tipo B são criados, mas não podemos falar nada sobre o garbage collector ter jogado os objetos fora na linha do System.out.
- b) Compila e nenhum objeto do tipo B é criado.
- c) Compila, cria 100 e joga fora todos os objetos do tipo B ao chegar no System.out.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {
2     void x() {
3         System.out.println("vazio");
4     }
5     void x(String... args) {
6         System.out.println(args.length);
7     }
8 }
9 class C {
10    void x(String... args) {
11        System.out.println(args.length);
12    }
13    void x() {
14        System.out.println("vazio");
15    }
16 }
17 class A {
18     public static void main(String[] args) {
19         new B().x();
20         new C().x();
21     }
22 }
```

- a) Não compila: conflito entre método com varargs e sem argumentos.
- b) Compila e imprime vazio/vazio.
- c) Compila e imprime vazio/o.
- d) Compila e imprime o/vazio.
- e) Compila e imprime o/o.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     void x(int... x) {  
3         System.out.println(x.length);  
4     }  
5 }  
6 class A {  
7     public static void main(String[] args) {  
8         new B().x(23789,673482);  
9     }  
10 }
```

- a) Não compila: varargs tem método e não atributo length.
- b) Compila e ao rodar imprime os dois números.
- c) Compila e ao rodar imprime 2.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     void x(int... x) {  
3         System.out.println(x.length);  
4     }  
5 }  
6 class A {  
7     public static void main(String[] args) {  
8         new B().x(new int[]{23789,673482});  
9     }  
10 }
```

- a) Não compila: varargs tem método e não atributo length.
- b) Não compila: não podemos passar um array para um varargs.
- c) Compila e ao rodar imprime os dois números.
- d) Compila e ao rodar imprime 1.
- e) Compila e ao rodar imprime 2.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{
2     void x(Object... x) {
3         System.out.println(x.length);
4     }
5 }
6 class A {
7     public static void main(String[] args) {
8         new B().x(new Object[]{23789,673482});
9     }
10 }
```

- a) Não compila: varargs tem método e não atributo length.
- b) Não compila: não podemos passar um array para um varargs.
- c) Compila e ao rodar imprime os dois números.
- d) Compila e ao rodar imprime 1.
- e) Compila e ao rodar imprime 2.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         StringBuilder sb = new StringBuilder();  
4         sb.append("guilherme").delete(2,3);  
5         System.out.println(sb);  
6     }  
7 }
```

- a) O código não compila: erro na linha que tenta imprimir o `StringBuilder`.
- b) O código compila e imprime `glherme`.
- c) O código compila e imprime `guherme`.
- d) O código compila e imprime `gilherme`.
- e) O código compila e imprime `gulherme`.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         Stringbuilder sb = new Stringbuilder("guilherme");  
4         System.out.println(sb.indexOf("e") + sb.lastIndexOf("e"));  
5         System.out.println(sb.indexOf("k") + sb.lastIndexOf("k"));  
6     }  
7 }
```

- a) O código imprime 13 e -2.
- b) O código imprime 13 e 0.
- c) O código imprime 13 e -1.
- d) O código imprime 13 e 8.
- e) O código imprime 13 e 10.
- f) O código imprime 15 e -2.
- g) O código imprime 15 e 0.
- h) O código imprime 15 e -1.
- i) O código imprime 15 e 8.
- j) O código imprime 15 e 10.

1) Considere o seguinte código dentro de um `main`:

```
class A{  
    public static void main(String [] args){  
        String s = "aba";  
        for(int i = 0; i < 9; i++) {  
            s = s +"aba";  
        }  
        System.out.println(s.length);  
    }  
}
```

- a) Não compila.
- b) Compila e imprime 3.
- c) Compila e imprime 30.
- d) Compila e imprime 33.
- e) Compila e imprime 36.

2) Dada a seguinte classe:

```
class B {  
    String msg;  
  
    void imprime() {  
        if (!msg.isEmpty())  
            System.out.println(msg);  
        else  
            System.out.println("vazio");  
    }  
}
```

O que acontece se chamarmos `new B().imprime()` ?

- a) Não compila.
- b) Compila, mas dá exceção na hora de rodar.
- c) Compila, roda e não imprime nada.
- d) Compila, roda e imprime “vazio”.

3) Dada a seguinte classe:

```
class B {  
  
    void imprime() {  
        String msg;  
        if (!msg.isEmpty())  
            System.out.println(msg);  
        else  
            System.out.println("vazio");  
    }  
}
```

O que acontece se chamarmos `new B().imprime()` ?

- a) Não compila.
- b) Compila, mas dá exceção na hora de rodar.
- c) Compila, roda e não imprime nada.
- d) Compila, roda e imprime “vazio”.

4) Qual é a saída nos dois casos?

```
String s = "Caelum";
s.concat(" - Ensino e Inovação");
System.out.println(s);

StringBuffer s = new StringBuffer("Caelum");
s.append(" - Ensino e Inovação");
System.out.println(s);
```

- a) 'Caelum' e 'Caelum - Ensino e Inovação'.
- b) 'Caelum - Ensino e Inovação' e 'Caelum - Ensino e Inovação'.
- c) 'Caelum' e 'Caelum'.
- d) 'Caelum - Ensino e Inovação' e 'Caelum'.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String vazio = null;  
4         String full = "Bem-vindo " + vazio;  
5         System.out.println(full);  
6     }  
7 }
```

- a) Não compila pois vazio é nulo.
- b) Não compila por outro motivo.
- c) Compila e imprime “Bem-vindo “.
- d) Compila e imprime “Bem-vindo vazio”.
- e) Compila e imprime outro resultado que não foi mencionado nessas alternativas.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String vazio;  
4         String full = "Bem-vindo " + vazio;  
5         System.out.println(full);  
6     }  
7 }
```

- a) Não compila pois vazio é nulo.
- b) Não compila por outro motivo.
- c) Compila e imprime “Bem-vindo “.
- d) Compila e imprime “Bem-vindo vazio”.
- e) Compila e imprime outro resultado que não foi mencionado nessas alternativas.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     String vazio;  
3     public static void main(String[] args) {  
4         String full = "Bem-vindo " + vazio;  
5         System.out.println(full);  
6     }  
7 }
```

- a) Não compila pois vazio é nulo.
- b) Não compila por outro motivo.
- c) Compila e imprime “Bem-vindo “.
- d) Compila e imprime “Bem-vindo vazio”.
- e) Compila e imprime outro resultado que não foi mencionado nessas alternativas.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     static String vazio;  
3     public static void main(String[] args) {  
4         String full = "Bem-vindo " + vazio;  
5         System.out.println(full);  
6     }  
7 }
```

- a) Não compila pois vazio é nulo.
- b) Não compila por outro motivo.
- c) Compila e imprime “Bem-vindo “.
- d) Compila e imprime “Bem-vindo vazio”.
- e) Compila e imprime outro resultado que não foi mencionado nessas alternativas.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s = null;  
4         String s2 = new String(s);  
5         System.out.println(s2);  
6     }  
7 }
```

- a) Não compila ao tentar invocar o construtor.
- b) Compila e não imprime nada.
- c) Compila e imprime `null`.
- d) Compila e dá erro de execução ao tentar criar a segunda `String`.

10) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s = "estudando para a certificação";  
4         System.out.println(s.substring(3, 6));  
5     }  
6 }
```

- a) Não compila: caractere com acento e cedilha dentro de uma String.
- b) Não compila: substring é subString.
- c) Compila e imprime “uda”.
- d) Compila e imprime “tuda”.
- e) Compila e imprime “tud”.

11) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s2 = new String(null);  
4         System.out.println(s2);  
5     }  
6 }
```

- a) Não compila ao tentar invocar o construtor.
- b) Compila e não imprime nada.
- c) Compila e imprime null.
- d) Compila e dá erro de execução ao tentar criar a String.

12) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int valor = 10;  
4         int dividePor = 4;  
5         double resultado = valor / dividePor;  
6         System.out.println(valor + dividePor +  
7                             " são os valores utilizados.");  
8         System.out.println(resultado + " é o resultado");  
9     }  
10 }
```

- a) Imprime os números 10, 4 e 2.5.
- b) Imprime os números 14 e 2.5.
- c) Nenhuma das outras alternativas.

13) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s = "estudando para a certificação";  
4         s.replace("e", 'a');  
5         System.out.println(s);  
6     }  
7 }
```

- a) Não compila.
- b) Compila e imprime “estudando para a certificação”.
- c) Compila e imprime “astudando para a cartificação”.
- d) Compila e imprime “studando para a crtificação”.

14) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s = "guilherme";  
4         s.substring(0,2) = "gua";  
5         System.out.println(s);  
6     }  
7 }
```

- a) Erro de compilação.
- b) Compila e imprime “guilherme”.
- c) Compila e imprime “gualherme”.

1) Qual código a seguir compila?

- a) `short s = 10;`
`char c = s;`
- b) `char c = 10;`
`long l = c;`
- c) `char c = 10;`
`short s = c;`

2) Faça contas com diferentes operandos:

```
int i1 = 3/2;
double i2 = 3/2;
double i3 = 3/2.0;

long x = 0; double d = 0;
double zero = x + d;
System.out.println(i1 + i2 + i3 + x + d + zero);
```

Qual o resultado?

- a) 3
- b) 3.5
- c) 4
- d) 4.5

- 3) O código a seguir pode lançar um `NullPointerException`. Como evitar isso mantendo a mesma lógica?

```
void metodo(Carro c) {  
    if(c != null & c.getPreco() > 100000) {  
        System.out.println("possivel sequestro");  
    }  
}
```

- a) Trocando `!=` por `==`
- b) Trocando `>` por `<`
- c) Trocando `&` por `|`
- d) Trocando `&` por `&&`

4) Alguns testes interessantes com tipos primitivos:

```
int i = (byte) 5;
long l = 3.0;
float f = 0.0;
char c = 3;
char c2 = -2;
```

Quais compilam?

- a) i, f e c
- b) i, f, c e c2
- c) i, f e c2
- d) i e c
- e) f e c
- f) f e c2
- g) i e l
- h) l, f e c
- i) i, c e c2

5) A expressão a seguir pode ser reduzida, como podemos fazer?

```
if ((trem && !carro) || (!trem && carro)) {  
    // ....  
}
```

- Trocando para usar um operador & e um |
- Trocando para usar dois operadores & e um |
- Trocando para usar um operador ! e um ^
- Trocando para usar um operador ^
- Removendo os parênteses
- Removendo o || do meio
- Removendo os !

- 6) Imprima a divisão por 0 de números inteiros e de números com ponto flutuante:

```
System.out.println(3 / 0);  
System.out.println(3 / 0.0);  
System.out.println(3.0 / 0);  
System.out.println(-3.0 / 0);
```

Quais os resultados?

```
7} class Xyz {  
2   public static void main(String[] args) {  
3     int y;  
4     for(int x = 0; x<10; ++x) {  
5       y = x % 5 + 2;  
6     }  
7     System.out.println(y);  
8   }  
9 }
```

Qual o resultado desse código?

- a) Erro de compilação na linha 3
- b) Erro de compilação na linha 7
- c) 1
- d) 2
- e) 3
- f) 4
- g) 5
- h) 6

```
8) class Teste {  
2     public static void main(String[] args){  
3         byte b = 1;  
4         int i = 1;  
5         long l = 1;  
6         float f = 1.0;  
7     }  
8 }
```

O código:

- a) Não compila a linha 3 pois “1” é `int` e não pode ser colocado em um `byte`
- b) Não compila a linha 4 pois “1” é `long` e não pode ser colocado em um `int`
- c) Não compila a linha 5 pois “1” é `int` e não pode ser colocado em um `long`
- d) Não compila a linha 6 pois “1.0” é `double` e não pode ser colocado em um `float`
- e) Todas as linhas compilam

```
9) class $_o0o_$ {
    public static void main(String[] args) {
        int $$ = 5;
        int __ = $$++;
        if (__ < ++$$ || __-- > $$)
            System.out.print("A");
        System.out.print($$);
        System.out.print(__);
    }
}
```

O estranho código:

- a) Não compila por causa do nome da classe
- b) Não compila por causa dos nomes das variáveis
- c) Compila mas dá erro na execução
- d) Compila, roda e imprime A76
- e) Compila, roda e imprime A75
- f) Compila, roda e imprime A74
- g) Compila, roda e imprime 76

1) O que acontece com o seguinte código? Compila? Roda?

```
public class Teste{  
    public static void main(String[] args) {  
        byte b1 = 5;  
        byte b2 = 3;  
        byte b3 = b1 + b2;  
    }  
}
```

1) O que acontece com seguinte código?

```
1 public class Teste{  
2     public static void main(String[] args) {  
3         byte b1 = 127;  
4         byte b2 = -128;  
5         byte b3 = b1 + b2;  
6         System.out.println(b3);  
7     }  
8 }
```

- a) Não compila por um erro na linha 3
- b) Não compila por um erro na linha 4
- c) Não compila por um erro na linha 5
- d) Compila e imprime -1

```
1) public class Teste {  
2      public static void main(String[] args) {  
3          int i;  
4          for (i = 0; i < 5; i++) {  
5              if (++i % 3 == 0) {  
6                  break;  
7              }  
8          }  
9          System.out.println(i);  
10     }  
11 }
```

Qual é o resultado do código:

- a) imprime 1
- b) imprime 2
- c) imprime 3
- d) imprime 4

- 2) Considerando o mesmo código da questão anterior, e se trocarmos para pós-incremento dentro do `if`?

```
1 public class Teste {  
2     public static void main(String[] args) {  
3         int i;  
4         for (i = 0; i < 5; i++) {  
5             if (i++ % 3 == 0) {  
6                 break;  
7             }  
8         }  
9         System.out.println(i);  
10    }  
11 }
```

Qual é o resultado?:

- a) imprime 1
- b) imprime 2
- c) imprime 3
- d) imprime 4

3) Qual é o resultado do seguinte código:

```
public class Teste {  
    public static void main(String[] args) {  
        int i;  
        for (i = 0; i < 5; i++) {  
            if (++i % 3 == 0) {  
                break;  
            }  
        }  
        System.out.println(i);  
    }  
}
```

4) E se trocarmos o pré-incremento para pós-incremento (`i++`)?

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         byte b1 = 100;  
4         byte b2 = 131;  
5         System.out.println(b1);  
6     }  
7 }
```

- a) Compila e imprime um número positivo.
- b) Compila e imprime um número negativo.
- c) Compila e dá uma exception de estouro de número.
- d) Compila e imprime um número que não sabemos dizer ao certo.
- e) Compila e imprime “Not A Number”.
- f) Não compila.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         char c = 65;  
4         char c2 = -3;  
5         System.out.println(c + c2);  
6     }  
7 }
```

- a) Não compila nas duas declarações de `char`.
- b) Não compila nas três linhas dentro do método `main`.
- c) Não compila somente na declaração de `c2`.
- d) Não compila somente na soma de caracteres.
- e) Compila e roda, imprimindo 62.
- f) Compila e roda, imprimindo um outro valor.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         char c = 65;  
4         char c2 = 68 - 65;  
5         System.out.println(c + c2);  
6     }  
7 }
```

- a) Não compila nas duas declarações de `char`.
- b) Não compila nas três linhas dentro do método `main`.
- c) Não compila somente na declaração de `c2`.
- d) Não compila somente na soma de caracteres.
- e) Compila e roda, imprimindo 62.
- f) Compila e roda, imprimindo um outro valor.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         double resultado = 15 / 0;  
4         System.out.println(resultado);  
5     }  
6 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e imprime positivo infinito.
- d) Compila e imprime o.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String resultado = "divisao dá: " + 15 / 0.0;  
4         System.out.println(resultado);  
5     }  
6 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e imprime positivo infinito.
- d) Compila e imprime o.

10) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         System.out.println(1==true);  
4     }  
5 }
```

- a) Não compila.
- b) Compila e imprime verdadeiro.
- c) Compila e imprime falso.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String resultado = ("divisao dá: " + 15) / 0.0;  
4         System.out.println(resultado);  
5     }  
6 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e imprime positivo infinito.
- d) Compila e imprime o.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         System.out.println(((!(true==false))==true ? 1 : 0)==0);  
4     }  
5 }
```

- a) Imprime true.
- b) Imprime false.
- c) Não compila.
- d) Imprime 1.
- e) Imprime 0.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s1 = "s1";  
4         String s2 = "s" + "1";  
5         System.out.println(s1==s2);  
6         System.out.println(s1==("" + s2));  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime true, false.
- c) Compila e imprime true, true.
- d) Compila e imprime false, false.
- e) Compila e imprime false, true.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s1 = "s1";  
4         String s2 = s1.substring(0, 1) + s1.substring(1,1);  
5         System.out.println(s1==s2);  
6         System.out.println(s1.equals(s2));  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime true, false.
- c) Compila e imprime true, true.
- d) Compila e imprime false, false.
- e) Compila e imprime false, true.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String s1 = "s1";  
4         String s2 = s1.substring(0, 2);  
5         System.out.println(s1==s2);  
6         System.out.println(s1.equals(s2));  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime true, false.
- c) Compila e imprime true, true.
- d) Compila e imprime false, false.
- e) Compila e imprime false, true.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B extends C{}
2 class C {
3     int x;
4     public boolean equals(C c) {
5         return c.x==x;
6     }
7 }
8 class A {
9     public static void main(String[] args) {
10         C a = new C();
11         C b = new B();
12         a.x = 1;
13         b.x = 1;
14         System.out.println(a==b);
15         System.out.println(a.equals(b));
16     }
17 }
```

- a) Não compila.
- b) Compila e imprime true, false.
- c) Compila e imprime true, true.
- d) Compila e imprime false, false.
- e) Compila e imprime false, true.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B extends C{}
2 class D {
3     int x;
4 }
5 class C {
6     int x;
7     public boolean equals(Object c) {
8         return c.x==x;
9     }
10 }
11 class A {
12     public static void main(String[] args) {
13         C a = new C();
14         C b = new D();
15         a.x = 1;
16         b.x = 1;
17         System.out.println(a==b);
18         System.out.println(a.equals(b));
19     }
20 }
```

- a) Não compila.
- b) Compila e imprime true, false.
- c) Compila e imprime true, true.
- d) Compila e imprime false, false.
- e) Compila e imprime false, true.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         if(args.length > 0)  
4             System.out.println("Um ou mais argumentos");  
5         else  
6             System.out.println("0");  
7     }  
8 }
```

- a) Não compila: `length` é método.
- b) Não compila: faltou chaves no `if` e `else`.
- c) Se invocarmos sem argumentos, imprime 0.
- d) Nunca imprimirá 0.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     final boolean valor = false;  
3 }  
4 class A {  
5     public static void main(String[] args) {  
6         B b = new B();  
7         if(b.valor = true) {  
8             System.out.println("verdadeiro");  
9         }  
10    }  
11 }
```

- a) Não compila.
- b) Compila e imprime verdadeiro.
- c) Compila e não imprime nada..

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int quantidade = 15;  
4         if(quantidade=15) {  
5             System.out.println("sim");  
6         } else {  
7             System.out.println("nao");  
8         }  
9     }  
10 }
```

- a) Não compila.
- b) Imprime sim.
- c) Imprime não.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         if(args.length==1)  
4             System.out.println("Um");  
5         elseif(args.length==2)  
6             System.out.println("Dois");  
7         elseif(args.length==3)  
8             System.out.println("Três");  
9         else  
10            System.out.println("Quatro");  
11    }  
12 }
```

- a) Não compila.
- b) Roda e imprime “Um” quando passamos um argumento.
- c) Roda e imprime “Três” quando passamos 4 argumentos.
- d) Roda e não imprime nada quando passamos nenhum argumento.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String nome = args[0];  
4         if(nome.equals("guilherme"))  
5             System.out.println(nome);  
6             System.out.println("bom");  
7         else  
8             System.out.println("melhor ainda");  
9             System.out.println(nome);  
10    }  
11 }
```

- a) Erro de compilação no `if`.
- b) Erro de compilação no `else`.
- c) Compila e imprime o nome e “bom” caso o primeiro argumento seja `guilherme`.
- d) Compila e dá erro de execução caso não passe nenhum argumento na linha de comando.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int tamanho = args.length;  
4         switch(tamanho) {  
5             case 1:  
6                 System.out.println("1");  
7             case 2:  
8                 System.out.println("2");  
9             default:  
10                 System.out.println("mais argumentos");  
11         }  
12     }  
13 }
```

- a) Não compila.
- b) Ao rodar sem argumentos joga uma exception..
- c) Ao rodar com dois argumentos, imprime somente “2”..
- d) Ao rodar com 5 argumentos, imprime “mais argumentos”.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int tamanhoEsperado = 1;  
4         int tamanho = args.length;  
5         switch(tamanho) {  
6             case tamanhoEsperado:  
7                 System.out.println("1");  
8                 break;  
9             default:  
10                 System.out.println("cade o argumento?");  
11         }  
12     }  
13 }
```

- a) Não compila.
- b) Ao rodar sem argumentos joga uma exception.
- c) Ao rodar com um argumento, imprime somente “1”.
- d) Ao rodar com 5 argumentos, imprime “cade o argumento?”

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         switch("Guilherme") {  
4             case "Guilherme":  
5                 System.out.println("Guilherme");  
6                 break;  
7             case "42":  
8                 System.out.println("42");  
9             default:  
10                 System.out.println("Outro nome");  
11         }  
12     }  
13 }
```

- a) Não compila, pois um número não pode ser comparado com `String`.
- b) Compila e imprime `Guilherme`.
- c) Não compila, pois o código do `case 42` e `default` nunca serão executados.

```
1 class A {  
2     public static void main(String[] args) {  
3         int count = args.length;  
4         switch(count) {  
5             case 0 {  
6                 System.out.println("nenhum");  
7                 break;  
8             } case 1 {  
9             } case 2 {  
10                System.out.println("ok");  
11            } default {  
12                System.out.println("default");  
13            }  
14        }  
15    }  
16 }
```

- a) Erro de compilação.
- b) Se rodar com 1 argumento, imprime ok e mais uma mensagem.
- c) Se rodar com 1 argumento, não imprime nada.
- d) Se rodar com 5 argumentos, imprime default.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         switch(10) {  
4             case < 10:  
5                 System.out.println("menor");  
6             default:  
7                 System.out.println("igual");  
8             case > 10:  
9                 System.out.println("maior");  
10        }  
11    }  
12 }
```

- a) Erro de compilação.
- b) Compila e imprime “igual”.
- c) Compila e imprime “igual” e “maior”.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         switch(10) {  
4             case 10:  
5                 System.out.println("a");  
6                 break;  
7                 System.out.println("b");  
8             default:  
9                 System.out.println("c");  
10            case 11:  
11                System.out.println("d");  
12        }  
13    }  
14 }
```

- a) Não compila.
- b) Imprime a e b e c e d.
- c) Imprime a.

1) Escolha a opção que não compila:

- a) int [] x;
- b) int x[];
- c) int []x;
- d) int [] x;
- e) int[] x;
- f) []int x;

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x[] = new int[30];  
4         int y[] = new int[3] {0,3,5};  
5     }  
6 }
```

- a) A primeira linha não compila.
- b) A segunda linha não compila.
- c) O código compila e roda.

- 3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, em relação as linhas dentro do método main:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x[] = new int[0];  
4         int x[] = new int[] {0,3,5};  
5         int x[] = {0,3,5};  
6     }  
7 }
```

- a) A primeira e segunda linhas não compilam.
- b) A segunda e terceira linhas não compilam.
- c) Somente a terceira linha não compila.
- d) O programa compila e roda, dando uma exception.
- e) O programa compila e roda, imprimindo nada.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x[] = new int[3];  
4         for(int i=x.length;i>=0;i--) x[i]=i*2;  
5         System.out.println("Fim!");  
6     }  
7 }
```

- a) O programa não compila
- b) O programa imprime Fim.
- c) O programa compila e dá erro em execução.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x[] = new int[3];  
4         for(x[1]=x.length-1;x[0]==0;x[1]--) {  
5             x[x[1]]=-5;  
6             System.out.println(x[1]);  
7         }  
8     }  
9 }
```

- a) Não compila.
- b) Compila, imprime alguns números e dá uma Exception.
- c) Compila e não imprime nada.
- d) Compila e imprime 2.
- e) Compila e imprime -5.
- f) Compila e imprime 2, -5.
- g) Compila e imprime 2, -5, -5.
- h) Compila e imprime 2, 1, -5.
- i) Compila e imprime -5, -5.
- j) Dá exception.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x[] = new int[3];  
4         for(x[1]=x.length-1;x[1]>=0;x[1]--) {  
5             x[x[1]]=-5;  
6             System.out.println(x[1]);  
7         }  
8     }  
9 }
```

- a) Não compila.
- b) Compila, imprime alguns números e dá uma `Exception`.
- c) Compila e não imprime nada.
- d) Compila e imprime 2.
- e) Compila e imprime -5.
- f) Compila e imprime 2, -5.
- g) Compila e imprime 2, -5, -5.
- h) Compila e imprime 2, 1, -5.
- i) Compila e imprime -5, -5.
- j) Dá exception.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String[] valores = new String[2];  
4         valores[0] = "Certificação";  
5         valores[1] = "Java";  
6         Object[] vals = (Object[]) valores;  
7         vals[1] = "Daniela";  
8         System.out.println(vals[1].equals(valores[1]));  
9     }  
10 }
```

- a) O código não compila.
- b) O código compila e dá erro em execução.
- c) O código compila e imprime `false`.
- d) O código compila e imprime `true`.

8) Quais das maneiras adiante são declarações e inicializações válidas para um array?

- a) int[] array = new int[10];
- b) int array[] = new int[10];
- c) int[] array = new int[];
- d) int array[] = new int[];
- e) int[] array = new int[2]{1, 2};
- f) int[] array = new int[]{1, 2};
- g) int[] array = int[10];
- h) int[] array = new int[1, 2, 3];
- i) int array[] = new int[1, 2, 3];
- j) int array[] = {1, 2, 3};

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int zyx[][]=new int[3];  
4         int []x=new int[20];  
5         int []y=new int[10];  
6         int []z=new int[30];  
7         zyx[0]=x;  
8         zyx[1]=y;  
9         zyx[2]=z;  
10        System.out.println(zyx[2].length);  
11    }  
12 }
```

- a) Não compila, erro ao declarar `zyx`.
- b) Compila e dá erro ao tentar atribuir o segundo array a `zyx`.
- c) Compila e dá erro ao tentar imprimir o tamanho do array.
- d) Compila e imprime 10.
- e) Compila e imprime 20.
- f) Compila e imprime 30.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int zyx[][]=new int[3][];  
4         int []x=new int[20];  
5         int []y=new int[10];  
6         int []z=new int[30];  
7         zyx[0]=x;  
8         zyx[1]=y;  
9         zyx[2]=z;  
10        System.out.println(zyx[2].length);  
11    }  
12 }
```

- a) Não compila, erro ao declarar `zyx`.
- b) Compila e dá erro ao tentar atribuir o segundo array a `zyx`.
- c) Compila e dá erro ao tentar imprimir o tamanho do array.
- d) Compila e imprime 10.
- e) Compila e imprime 20.
- f) Compila e imprime 30.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int zyx[][]=new int[3][10];  
4         int []x=new int[20];  
5         int []y=new int[10];  
6         int []z=new int[30];  
7         zyx[0]=x;  
8         zyx[1]=y;  
9         zyx[2]=z;  
10        System.out.println(zyx[2].length);  
11    }  
12 }
```

- a) Não compila, erro ao declarar `zyx`.
- b) Não compila, erro ao atribuir arrays de tamanho diferente de 10 em `zyx`.
- c) Compila e dá erro ao tentar atribuir o segundo array a `zyx`.
- d) Compila e dá erro ao tentar imprimir o tamanho do array.
- e) Compila e imprime 10.
- f) Compila e imprime 20.
- g) Compila e imprime 30.

```
4) class Teste {  
    public static void main(String[] args){  
        int[] idades = new int[10];  
        idades[0] = 1.0;  
  
        int[][][] cubo = new int[][][];  
    }  
}
```

- a) O código não compila.
- b) O código compila e dá erro em execução.
- c) O código compila e roda.

Compila? Roda?

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         ArrayList<String> c = new ArrayList<String>();  
4         c.add("a");  
5         c.add("c");  
6         System.out.println(c.remove("a"));  
7     }  
8 }
```

- a) Não compila: erro ao declarar a `ArrayList`.
- b) Não compila: erro ao invocar `remove`.
- c) Compila e ao rodar imprime `a`.
- d) Compila e ao rodar imprime `true`.
- e) Compila e ao rodar imprime `false`.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> c = new ArrayList<String>();
5         c.add("a");
6         c.add("c");
7         System.out.println(c.remove("a"));
8     }
9 }
```

- a) Não compila: erro ao declarar a `ArrayList`.
- b) Não compila: erro ao invocar `remove`.
- c) Compila e ao rodar imprime `a`.
- d) Compila e ao rodar imprime `true`.
- e) Compila e ao rodar imprime `false`.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> c = new ArrayList<String>();
5         c.add("a");
6         c.add("a");
7         System.out.println(c.remove("a"));
8         System.out.println(c.size());
9     }
10 }
```

- a) Não compila: erro ao declarar a `ArrayList`.
 - b) Não compila: erro ao invocar `remove`.
 - c) Compila e ao rodar imprime `a e o`.
 - d) Compila e ao rodar imprime `true e o`.
 - e) Compila e ao rodar imprime `a e 1`.
 - f) Compila e ao rodar imprime `true e 1`.
 - g) Compila e ao rodar imprime `a e 2`.
 - h) Compila e ao rodar imprime `true e 2`.
-

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> list = new ArrayList<>();
5         list.add("a");list.add("b");
6         list.add("a");list.add("c");
7         list.add("a");list.add("b");
8         list.add("a");
9         System.out.println(list.lastIndexOf("b"));
10    }
11 }
```

- a) Não compila.
- b) Compila e imprime -1.
- c) Compila e imprime 0.
- d) Compila e imprime 1.
- e) Compila e imprime 2.
- f) Compila e imprime 3.
- g) Compila e imprime 4.
- h) Compila e imprime 5.
- i) Compila e imprime 6.
- j) Compila e imprime 7.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> l = new ArrayList<String>();
5         l.add("a");
6         l.add("b");
7         l.add(1,"amor");
8         l.add(3,"baixinho");
9         System.out.println(l);
10        String[] array = l.toArray();
11        System.out.println(array[2]);
12    }
13 }
```

- a) Não compila.
- b) Compila e imprime “amor”.
- c) Compila e imprime “b”.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> a = new ArrayList<String>();
5         ArrayList<String> b = new ArrayList<String>();
6         ArrayList<String> c = new ArrayList<String>();
7         b.add("a");c.add("c");
8         b.add("b");c.add("d");
9         a.addAll(b);
10        a.addAll(c);
11        System.out.println(a.get(0));
12        System.out.println(a.get(3));
13    }
14 }
```

- a) Não compila
- b) Compila e imprime a e d.
- c) Compila e imprime c e b.
- d) Compila e não sabemos a ordem em que os elementos serão impressos.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> a = new ArrayList<String>();
5         a.add("a", 0);
6         a.add("b", 0);
7         a.add("c", 0);
8         a.add("d", 0);
9         System.out.println(a.get(0));
10        System.out.println(a.get(1));
11        System.out.println(a.get(2));
12        System.out.println(a.get(3));
13    }
14 }
```

- a) Não compila.
- b) Compila e imprime abcd.
- c) Compila e imprime dcba.
- d) Compila e imprime adcb.
- e) Compila e imprime bcda.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.*;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> a = new ArrayList<String>();
5         a.add(0,"b");
6         a.add(0,"a");
7         for(Iterator<String> i=a.iterator();i.hasNext();i.next()) {
8             String element = i.next();
9             System.out.println(element);
10        }
11    }
12 }
```

- a) Não compila.
- b) Compila e imprime a.
- c) Compila e imprime a e b.
- d) Compila e imprime b e a.
- e) Compila e imprime b.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.util.ArrayList;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> ss = new ArrayList<String>();
5         ss.add("a");
6         ss.add("b");
7         ss.add("c");
8         ss.add("d");
9
10        for(String s:ss){
11            if(s.equals("c")) s = "b";
12            else if(s.equals("b")) s= "c";
13        }
14        for(String s:ss) System.out.println(s);
15    }
16 }
```

- a) Não compila, s é final por padrão.
- b) Compila e imprime a, c, b, d.
- c) Compila e imprime a, b, c, d.
- d) Compila e imprime a, c, c, d.
- e) Compila e imprime a, c, b, d.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int a = 10;  
4         while(a>100) a++;  
5         System.out.println(a);  
6     }  
7 }
```

- a) Não compila pois nunca entra no loop.
- b) Compila e imprime 99.
- c) Compila e imprime 100.
- d) Compila e imprime 101.
- e) Compila e imprime outro valor.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         boolean rodar = true;  
4         while(rodar) {  
5             System.out.println(rodar);  
6         }  
7         System.out.println("Terminou");  
8     }  
9 }
```

- a) Transformar a variável em `final` faz o código compilar.
- b) Colocar uma linha dentro do laço que faz `rodar = false` faz o código compilar.
- c) O código compila e roda em loop infinito.
- d) O código compila e roda, após algumas passagens pelo laço ele imprime uma exception e para.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(;;) {  
4             System.out.println("a");  
5         }  
6         System.out.println("b");  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime a infinitamente.
- c) Compila e imprime b.
- d) Compila e imprime a, depois b, depois para.
- e) Compila, imprime a diversas vezes e depois dá um StackOverflowError.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(;false;) {  
4             System.out.println("a");  
5             break;  
6         }  
7         System.out.println("b");  
8     }  
9 }
```

- a) Não compila.
- b) Compila e imprime b.
- c) Compila, imprime a e b.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(int i=0, int j=1; i<10; i++, j++) System.out.println(i);  
4     }  
5 }
```

- a) Não compila.
- b) Compila e imprime o até 9.
- c) Compila e imprime o até 10.
- d) Compila e imprime 1 até 10.
- e) Compila e imprime 1 até 11.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(int i=0, j=1; i<10 ;i++, j++) System.out.println(i);  
4     }  
5 }
```

- a) Não compila.
- b) Compila e imprime o até 9.
- c) Compila e imprime o até 10.
- d) Compila e imprime 1 até 10.
- e) Compila e imprime 1 até 11.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(int i=0; i<10, false; i++) {  
4             System.out.println('a');  
5         }  
6         System.out.println('b');  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime 'a' e 'b'.
- c) Compila e imprime 'b'.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         for(int i=0; i<2; i++, System.out.println(i)) {  
4             System.out.println(i);  
5         }  
6     }  
7 }
```

- a) Não compila.
- b) Compila e imprime 0 1 2.
- c) Compila e imprime 0 0 1 1 2 2.
- d) Compila e imprime 0 1 1 2 2.
- e) Compila e imprime 0 1 1 2.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         boolean i = false;  
4         do {  
5             System.out.println(i);  
6         } while(i);  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime `false`.
- c) Compila e não imprime nada.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         if(args.length < 10) {  
4             do {  
5                 if(args.length>2) return;  
6             } while(true);  
7         }  
8         System.out.println("Finalizou");  
9     }  
10 }
```

- a) Não compila.
- b) Compila e entra em loop infinito caso seja passado zero, um ou dois argumentos. Não imprime nada caso 3 a 9 argumentos. Imprime 'Finalizou' caso 10 ou mais argumentos.
- c) Compila e entra em loop infinito caso seja passado zero, um ou dois argumentos. Imprime 'Finalizou' caso contrário.
- d) Compila e sempre entra em loop infinito.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int i = 0;  
4         do System.out.println(i); while(i++>10);  
5     }  
6 }
```

- a) Não compila.
- b) Compila e não imprime nada.
- c) Compila e imprime 0.
- d) Compila e imprime de 0 até 9.
- e) Compila e imprime de 0 até 10.
- f) Compila e não imprime nada.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int i = 0;  
4         do System.out.println(i) while(i++<10);  
5     }  
6 }
```

- a) Não compila.
- b) Compila e imprime de 0 até 9.
- c) Compila e imprime de 0 até 10.
- d) Compila e não imprime nada.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int i = 0;  
4         do; while(i++<10);  
5     }  
6 }
```

- a) Não compila.
- b) Compila e entra em loop infinito.
- c) Compila e sai.

- 1) Qual o laço mais simples de ser usado quando desejamos iterar por duas coleções ao mesmo tempo?
- a) for
 - b) while
 - c) enhanced for
 - d) do... while

- 2) Qual o melhor laço a ser usado para, dependendo do valor de um elemento, removê-lo de nossa lista?
- a) for
 - b) enhanced for

- 3) Para todos os números entre `i` e `100` devo imprimir algo, sendo que, mesmo que `i` seja maior que `100`, devo imprimir algo pelo menos uma vez. Qual laço devo usar?
- a) enhanced for
 - b) do... while
 - c) while

- 4) Qual o laço a ser usado caso queira executar um código eternamente?
- a) enhanced for
 - b) for
 - c) for ou while
 - d) for ou while ou do...while
 - e) while ou do...while

5) Qual laço deve ser usado para inicializar os valores de um array?

- a) enhanced for
- b) for

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         fora: for(int a=0;a<30;a++)  
4             for(int b=0;b<1;b++)  
5                 if(a+b==25) continue fora;  
6                 else if(a+b==20) break fora;  
7                 if(a==0) break fora;  
8                 else System.out.println(a);  
9     }  
10 }
```

- a) Não compila.
- b) Compila e imprime 1 até 29
- c) Compila e não imprime nada
- d) Compila e imprime 1 até 19, 21 até 24, 26 até 29
- e) Compila e imprime 1 até 24, 26 até 29
- f) Compila e imprime 1 até 19

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         fora: for(int a=0;a<30;a++)  
4             for(int b=0;b<1;b++)  
5                 if(a+b==25) continue fora;  
6                 else if(a+b==20) break;  
7                 else System.out.println(a);  
8     }  
9 }
```

- a) Não compila.
- b) Compila e imprime o até 29.
- c) Compila e não imprime nada.
- d) Compila e imprime o até 19, 21 até 24, 26 até 29.
- e) Compila e imprime o até 24, 26 até 29.
- f) Compila e imprime o até 19.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int a = args.length;  
4         int i = 0;  
5         switch(a) {  
6             case 0:  
7             case 1:  
8                 for(i=0;i<15;i++, System.out.println(i))  
9                     if(i==5) continue;  
10                if(i==15) break;  
11                case 2:  
12                    System.out.println("2");  
13            }  
14            System.out.println("fim");  
15        }  
16    }
```

- a) Não compila.
- b) Compila e ao rodar com o argumentos imprime 0 até 14, 2, fim.
- c) Compila e ao rodar com o argumentos imprime 1 até 15, 2, fim.
- d) Compila e ao rodar com o argumentos imprime 0 até 4, 6 até 14, 2, fim.
- e) Compila e ao rodar com o argumentos imprime 1 até 4, 6 até 15, fim.
- f) Compila e ao rodar com o argumentos imprime 0 até 4, 6 até 9, 2, fim.
- g) Compila e ao rodar com o argumentos imprime 1 até 4, 6 até 9, fim.
- h) Compila e ao rodar com o argumentos imprime 1 até 4, 6 até 15, 2, fim.
- i) Compila e ao rodar com o argumentos imprime 1 até 15, 2, fim.
- j) Compila e ao rodar com o argumentos imprime 1 até 15, fim.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         x(args.length);  
4     }  
5     static int x(final int l) {  
6         for(int i=0;i<100;i++) {  
7             switch(i) {  
8                 case 1:  
9                     System.out.println(1);  
10                if(l==i) return;  
11                case 0:  
12                    System.out.println(0);  
13                }  
14            }  
15            System.out.println("Fim");  
16            return -1;  
17        }  
18    }
```

- a) Não compila.
- b) Compila e ao rodar com cinco parâmetros, imprime 0, 5 e Fim.
- c) Compila e ao rodar com cinco parâmetros, imprime 0, 5, -1 e Fim.
- d) Compila e ao rodar com cinco parâmetros, imprime 0 e 5.
- e) Compila e ao rodar com cinco parâmetros, imprime 0, 5 e -1.
- f) Compila e ao rodar com cinco parâmetros, imprime 0, 5, 0 e Fim.
- g) Compila e ao rodar com cinco parâmetros, imprime 0, 5, 0, -1 e Fim.
- h) Compila e ao rodar com cinco parâmetros, imprime 0 e 5.
- i) Compila e ao rodar com cinco parâmetros, imprime 0, 5, 0 e -1.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         x(args.length);  
4     }  
5     static int x(final int l) {  
6         for(int i=0;i<100;i++) {  
7             switch(i) {  
8                 case 1:  
9                     System.out.println(1);  
10                if(l==i) return 3;  
11                case 0:  
12                    System.out.println(0);  
13                }  
14            }  
15            System.out.println("Fim");  
16            return -1;  
17        }  
18    }
```

- a) Não compila.
- b) Compila e ao rodar com cinco parâmetros, imprime 0, 5 e Fim.
- c) Compila e ao rodar com cinco parâmetros, imprime 0, 5, -1 e Fim.
- d) Compila e ao rodar com cinco parâmetros, imprime 0 e 5.
- e) Compila e ao rodar com cinco parâmetros, imprime 0, 5 e -1.
- f) Compila e ao rodar com cinco parâmetros, imprime 0, 5, 0 e Fim.
- g) Compila e ao rodar com cinco parâmetros, imprime 0, 5, 0, -1 e Fim.
- h) Compila e ao rodar com cinco parâmetros, imprime 0 e 5.
- i) Compila e ao rodar com cinco parâmetros, imprime 0, 5, 0 e -1.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         System.out.println(a(args.length));  
4     }  
5     static int a(int l) {  
6         if(l<10) return b(l);  
7         else return c();  
8     }  
9     static int b(int l) {  
10        if(l<10) return b(l);  
11        else return c();  
12    }  
13    static long c() {  
14        return 3;  
15    }  
16 }
```

- a) Não compila: erro ao invocar o método `b`.
- b) Não compila: erro ao invocar o método `c`.
- c) Não compila por um motivo não listado.
- d) Compila e, ao chamar com 15 argumentos, imprime 3.
- e) Compila e, ao chamar com 15 argumentos, entra em loop infinito.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         System.out.println(a(args.length)[0]);  
4     }  
5     static int,int a(int l) {  
6         if(l==0) return {0, 1};  
7         else return {1, 0};  
8     }  
9 }
```

- a) Não compila.
- b) Ao invocar com nenhum parâmetro, imprime o.
- c) Ao invocar com 5 parâmetros, imprime o.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         x();  
4     }  
5     static x() {  
6         System.out.println("x");  
7         y();  
8     }  
9     static y() {  
10        System.out.println("y");  
11    }  
12 }
```

- a) Não compila.
- b) Imprime x, y.
- c) Imprime y, x.
- d) Imprime x.
- e) Imprime y.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         x();  
4     }  
5     static void x() {  
6         System.out.println("x");  
7         y();  
8     }  
9     static void y() {  
10        System.out.println("y");  
11    }  
12 }
```

- a) Não compila.
- b) Imprime x, y.
- c) Imprime y, x.
- d) Imprime x.
- e) Imprime y.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     void y() {  
3         this.z();  
4     }  
5     static void z() {  
6         System.out.println("z");  
7     }  
8 }  
9 class A {  
10    public static void main(String[] args) {  
11        new A().x();  
12    }  
13    static void x() {  
14        new B().y();  
15    }  
16 }
```

- a) Não compila ao tentar invocar `y`.
- b) Não compila ao tentar invocar `z`.
- c) Não compila ao tentar invocar `x`.
- d) Compila e imprime `z`.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     static void x() {  
3         System.out.println("x");  
4     }  
5     static void y() {  
6         System.out.println("y");  
7     }  
8 }  
9 class A extends B {  
10    public static void main(String[] args) {  
11        this.x();  
12        A.y();  
13    }  
14 }
```

- a) Não compila, erro ao invocar x.
- b) Imprime x, y.
- c) Não compila, erro ao invocar y.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     static void x() {  
3         System.out.println("x");  
4     }  
5     static void y() {  
6         System.out.println("y");  
7     }  
8 }  
9 class A extends B {  
10    public static void main(String[] args) {  
11        x();  
12        A.y();  
13    }  
14 }
```

- a) Não compila, erro ao invocar x.
- b) Imprime x, y.
- c) Não compila, erro ao invocar y.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x = b(15);  
4         System.out.println(x);  
5         System.out.println(15);  
6         System.out.println(15.0);  
7     }  
8     static int b(int i) { return i; }  
9     static double b(int i) { return i; }  
10 }
```

- a) Não compila.
- b) Compila e imprime 15, 15, 15.
- c) Compila e imprime 15, 15, 15.0.
- d) Compila e imprime 15, 15.0, 15.0.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int x = b(15);  
4         System.out.println(x);  
5         System.out.println(15);  
6         System.out.println(15.0);  
7     }  
8     static int b(int i) { return i; }  
9     static double b(double i) { return i; }  
10 }
```

- a) Não compila.
- b) Compila e imprime 15, 15, 15.
- c) Compila e imprime 15, 15, 15.0.
- d) Compila e imprime 15, 15.0, 15.0.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         System.out.println("[]");  
4     }  
5     public static void main(String... args) {  
6         System.out.println("...");  
7     }  
8 }
```

- a) Não compila.
- b) Compila e imprime “[]”.
- c) Compila e imprime “...”.
- d) Compila e dá exception.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{}
2 class C{}
3 class D extends B{}
4 class A {
5     int a(D d) { return 1; }
6     int a(C c) { return 2; }
7     int a(B b) { return 3; }
8     int a(A a) { return 4; }
9     public static void main(String[] args) {
10         System.out.println(a(new D()));
11     }
12 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.
- e) Compila e imprime 4.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{}
2 class C{}
3 class D extends B{}
4 class A {
5     int a(D d) { return 1; }
6     static int a(C c) { return 2; }
7     static int a(B b) { return 3; }
8     static int a(A a) { return 4; }
9     public static void main(String[] args) {
10         System.out.println(a(new D()));
11     }
12 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.
- e) Compila e imprime 4.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{}  
2 class C{}  
3 class D extends B{}  
4 class A {  
5     static int a(D d) { return 1; }  
6     static int a(C c) { return 2; }  
7     static int a(B b) { return 3; }  
8     static int a(A a) { return 4; }  
9     public static void main(String[] args) {  
10         System.out.println(a(new D()));  
11     }  
12 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.
- e) Compila e imprime 4.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{}
2 class C{}
3 class D extends B{}
4 class A {
5     static int a(D d, B b) { return 1; }
6     static int a(C c, C c) { return 2; }
7     static int a(B b, B b) { return 3; }
8     static int a(A a, A a) { return 4; }
9     public static void main(String[] args) {
10         System.out.println(a(new D(), new D()));
11     }
12 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.
- e) Compila e imprime 4.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{}
2 class C{}
3 class D extends B{}
4 class A {
5     static int a(D d, B b2) { return 1; }
6     static int a(C c, C c2) { return 2; }
7     static int a(B b, B b2) { return 3; }
8     static int a(A a, A a2) { return 4; }
9     public static void main(String[] args) {
10         System.out.println(a(new D(), new D()));
11     }
12 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.
- e) Compila e imprime 4.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     final String n;  
3     A() {  
4         a();  
5         n = "aprendendo";  
6     }  
7     void a() {  
8         System.out.println("testa");  
9     }  
10 }  
11 class B extends A {  
12     void a() {  
13         System.out.println(n.length());  
14     }  
15     public static void main(String[] args) {  
16         new B();  
17     }  
18 }
```

- a) Não compila.
- b) Compila e imprime “testa”.
- c) Compila e imprime `length`.
- d) Compila e dá exception.
- e) Compila e não imprime nada.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { B() { this(1); } B(int i) { this(); } }
2 class A {
3     public static void main(String[] args) {
4         new B();
5     }
6 }
```

- a) Não compila.
- b) Compila e joga exception.
- c) Compila e não imprime nada.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B() { B(A a) {} B() {} }
2 class C() { C(B b) {} C() {} }
3 class A {
4     public static void main(String[] args) {
5         new A(); new B(); new C();
6     }
7 }
```

- a) Não compila ao invocar o construtor de A.
- b) Não compila ao invocar o construtor de B.
- c) Não compila ao invocar o construtor de C.
- d) Não compila na definição das classes B e C.
- e) Compila e joga exception.
- f) Compila e não imprime nada.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { B(A a) {} B() {} }
2 class C { C(B b) {} C() {} }
3 class A {
4     public static void main(String[] args) {
5         new A(); new B(); new C();
6     }
7 }
```

- a) Não compila ao invocar o construtor de A.
- b) Não compila ao invocar o construtor de B.
- c) Não compila ao invocar o construtor de C.
- d) Não compila na definição das classes B e C.
- e) Compila e joga exception.
- f) Compila e não imprime nada.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { B(A a) {} B() {} }
2 class C { C(B b) {} C() {} }
3 class A {
4     public static void main(String[] args) {
5         new C(new B(new A()));
6     }
7 }
```

- a) Não compila ao invocar o construtor de A.
- b) Não compila ao invocar o construtor de B.
- c) Não compila ao invocar o construtor de C.
- d) Não compila na definição das classes B e C.
- e) Compila e joga exception.
- f) Compila e não imprime nada.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { B(A a) {new C(); } B() { new C(this);} }
2 class C { C(B b) {new B(new A());} C() {new B();} }
3 class A {
4     public static void main(String[] args) {
5         new C(new B(new A()));
6     }
7 }
```

- a) Não compila ao invocar o construtor de A.
- b) Não compila ao invocar o construtor de B.
- c) Não compila ao invocar o construtor de C.
- d) Não compila na definição das classes B e C.
- e) Compila e joga exception.
- f) Compila e não imprime nada.

- 1) Escolha a opção adequada ao tentar compilar e rodar o Teste. Arquivo no diretório atual:

```
1 import modelo.Cliente;
2 class Teste {
3     public static void main(String[] args) {
4         new Cliente("guilherme").imprime();
5     }
6 }
```

Arquivo no diretório modelo:

```
1 package modelo;
2
3 public class Cliente {
4     private String nome;
5     Cliente(String nome) {
6         this.nome = nome;
7     }
8     public void imprime() {
9         System.out.println(nome);
10    }
11 }
```

- a) Não compila: erro na classe Teste.
- b) Não compila: erro na classe Cliente.
- c) Erro de execução: método main.
- d) Roda e imprime “Guilherme”.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     private static int a(int b) {  
3         return b(b)-1;  
4     }  
5     private static int b(int b) {  
6         return b-1;  
7     }  
8     public static void main(String[] args) {  
9         System.out.println(new A().a(5));  
10    }  
11 }
```

- a) Não compila nas invocações de métodos.
- b) Não compila na declaração de variáveis e métodos.
- c) Compila e imprime 3.
- d) Compila e dá erro.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     private public int a(int b) {  
3         return b(b)-1;  
4     }  
5     private static int b(int b) {  
6         return b-1;  
7     }  
8     public static void main(String[] args) {  
9         System.out.println(new A().a(5));  
10    }  
11 }
```

- a) Não compila nas invocações de métodos.
- b) Não compila na declaração de variáveis e métodos.
- c) Compila e imprime 3.
- d) Compila e dá erro.

- 4) Escolha a opção adequada ao tentar compilar e rodar os arquivos a seguir, cada um em seu diretório adequado:

```
1 package a;
2 import b.*;
3 public class A extends B { protected int a(String s)
4                               {return 2;} }

1 package b;
2 import a.*;
3 public class B { public int a(Object s) {return 1;} }

1 import a.*;
2 import b.*;
3 class A {
4     public static void main(String[] args) {
5         System.out.println(new A().a("a"));
6     }
7 }
```

- a) Não compila.
- b) Imprime 1.
- c) Imprime 2.
- d) Erro em execução.

- 5) Escolha a opção adequada ao tentar compilar e rodar os arquivos a seguir, cada um em seu diretório adequado:

```
1 package a;
2 import b.*;
3 public class A extends B { protected int a(String s)
4                               {return 2;} }

1 package b;
2 import a.*;
3 public class B { public int a(Object s) {return 1;} }

1 import a.*;
2 import b.*;
3 class C {
4     public static void main(String[] args) {
5         System.out.println(new A().a("a"));
6     }
7 }
```

- a) Não compila.
- b) Imprime 1.
- c) Imprime 2.
- d) Erro em execução.

- 6) Escolha a opção adequada ao tentar compilar e rodar os arquivos a seguir, cada um em seu diretório adequado:

```
1 package a;
2 import b.*;
3 public class A extends B { protected int a(String s)
4                               {return 2;} }

1 package b;
2 import a.*;
3 public class B { default int a(Object s) {return 1;} }

1 import a.*;
2 import b.*;
3 class C {
4     public static void main(String[] args) {
5         System.out.println(new A().a("a"));
6     }
7 }
```

- a) Não compila.
- b) Imprime 1.
- c) Imprime 2.
- d) Erro em execução.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     static int bs=0;  
3     final int b = ++bs;  
4     private B() {}  
5     static B b() { return new B(); }  
6 }  
7 class A {  
8     public static void main(String[] args) {  
9         System.out.println(B.b().b);  
10    }  
11 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 0.
- d) Compila e dá erro de execução.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     private int b;  
3     public int getB() { return b; }  
4     public void setB(int b) { this.b= b; }  
5 }  
6 class A {  
7     public static void main(String[] args) {  
8         new B().setB(5);  
9         System.out.println(new B().getB());  
10    }  
11 }
```

- a) Não compila.
- b) Compila e imprime o.
- c) Compila e imprime 5.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     private int b;  
3     public int getB() { return b; }  
4     public void setB(int b) { this.b= b; }  
5 }  
6 class A {  
7     public static void main(String[] args) {  
8         B b = new B();  
9         b.setB(5);  
10        System.out.println(b.getB());  
11    }  
12 }
```

- a) Não compila.
- b) Compila e imprime o.
- c) Compila e imprime 5.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     private int b;  
3     public int getB() { return b; }  
4     public void setB(int b) { b= b; }  
5 }  
6 class A {  
7     public static void main(String[] args) {  
8         B b = new B();  
9         b.setB(5);  
10        System.out.println(b.getB());  
11    }  
12 }
```

- a) Não compila.
- b) Compila e imprime o.
- c) Compila e imprime 5.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     int b;  
3     public void setB(int b) { b= b; }  
4 }  
5 class A {  
6     public static void main(String[] args) {  
7         B b = new B();  
8         b.setB(5);  
9         System.out.println(b.b);  
10    }  
11 }
```

- a) Não compila, pois não é possível ter `setter` sem `getter`.
- b) Compila e imprime o.
- c) Compila e imprime 5.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B{  
2     private final int b;  
3     B(int b) { this.b = b; }  
4     public int getB() { return b; }  
5     public void setB(int b) { b= b; }  
6 }  
7 class A {  
8     public static void main(String[] args) {  
9         B b = new B(10);  
10        b.setB(5);  
11        System.out.println(b.getB());  
12    }  
13 }
```

- a) Não compila.
- b) Compila e imprime 0.
- c) Compila e imprime 5.
- d) Compila e imprime 10.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int i = 150;  
4         i = ++s(i);  
5         System.out.println(i);  
6     }  
7     static int s(int i) {  
8         return ++i;  
9     }  
10 }
```

- a) Não compila.
- b) Compila e imprime 150.
- c) Compila e imprime 151.
- d) Compila e imprime 152.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         int[] i = {150, 151};  
4         i = s(i);  
5         System.out.println(i[1]);  
6     }  
7     static int[] s(int[] i) {  
8         int[] j = {i[0], i[1]};  
9         i[1]++;  
10        return j;  
11    }  
12 }
```

- a) Não compila.
- b) Compila e imprime 150.
- c) Compila e imprime 151.
- d) Compila e imprime 152.
- e) Compila e imprime 153.

```
1) class A {
    public void metodo(long l) {
    }
}
class B extends A{
    protected void metodo(int i) {
    }
}
```

Compila?

```
2) import java.io.*;
class Veiculo {
    protected void liga () throws IOException {}
}
class Carro extends Veiculo {
    public void liga() throws FileNotFoundException {}
}
```

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B extends C { int m(int a) { return 1; } }
2 class C extends A { int m(double b) { return 3; } }
3 class A extends B {
4     int m(String c) { return 3; }
5     public static void main(String[] args) {
6         System.out.println(new C().m(3));
7     }
8 }
```

- a) O código não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.
- e) Compila e imprime 1, 2, 3.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { int m(int a) { return 1; } }
2 class C { int m(double b) { return 2; } }
3 class A extends B, C{
4     public static void main(String[] args) {
5         System.out.println(new C().m(3));
6     }
7 }
```

- a) O código não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 1, 2.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { private B() {} static B B(String s)
2             { return new B(); } }
3 class A {
4     public static void main(String[] args) {
5         B b = B.B("t");
6     }
7 }
```

- a) Não compila.
- b) Compila e imprime “t”.
- c) Compila e não imprime nada.
- d) Compila e joga uma exception.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B { private B() {} static B B(String s)
2             { return new B(); } }
3 class A extends B {
4     public static void main(String[] args) {
5         B b = B.B("t");
6     }
7 }
```

- a) Não compila.
- b) Compila e imprime “t”.
- c) Compila e não imprime nada.
- d) Compila e joga uma exception.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     private String s;  
3     protected B() {}  
4     static A B(String s) {  
5         return new A();  
6     }  
7 }  
8 class A extends B {  
9     A(String s) {  
10         this.s = s;  
11     }  
12     public static void main(String[] args) {  
13         B b = A.B("t");  
14         System.out.println(b.s);  
15     }  
16 }
```

- a) Não compila.
- b) Compila e imprime “t”.
- c) Compila e não imprime nada.
- d) Compila e joga uma exception.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     protected String s;  
3     protected B() {}  
4     static A B(String s) {  
5         return new A();  
6     }  
7 }  
8 class A extends B {  
9     A(String s) {  
10         this.s = s;  
11     }  
12     public static void main(String[] args) {  
13         A b = A.B("t");  
14         System.out.println(b.s);  
15     }  
16 }
```

- a) Não compila.
- b) Compila e imprime “t”.
- c) Compila e não imprime nada.
- d) Compila e joga uma exception.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     void x() throws IOException {  
3         System.out.println("c");  
4     }  
5 }  
6 class C extends B {  
7     void x() throws FileNotFoundException {  
8         System.out.println("b");  
9     }  
10 }  
11 class A {  
12     public static void main(String[] args) {  
13         new C().x();  
14     }  
15 }
```

- a) Não compila.
- b) Compila e imprime ‘b’.
- c) Compila e imprime ‘c’.
- d) Compila e não imprime nada.
- e) Compila e dá exception.
- f) Compila e entra em loop.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x() throws IOException {
4         System.out.println("c");
5     }
6 }
7 class C extends B {
8     void x() throws FileNotFoundException {
9         System.out.println("b");
10    }
11 }
12 class A {
13     public static void main(String[] args) throws IOException {
14         new C().x();
15     }
16 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e não imprime nada.
- e) Compila e dá exception.
- f) Compila e entra em loop.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x(double i) throws IOException {
4         System.out.println("c");
5     }
6 }
7 class C extends B {
8     void x(int i) throws FileNotFoundException {
9         System.out.println("b");
10    }
11 }
12 class A {
13     public static void main(String[] args) throws IOException {
14         new C().x(3.2);
15     }
16 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e não imprime nada.
- e) Compila e dá exception.
- f) Compila e entra em loop.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x(double i) throws IOException {
4         System.out.println("c");
5     }
6 }
7 class C {
8     void x(int i) throws FileNotFoundException {
9         System.out.println("b");
10    }
11 }
12 class A {
13     public static void main(String[] args) throws IOException {
14         new C().x(3.2);
15     }
16 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e não imprime nada.
- e) Compila e dá exception.
- f) Compila e entra em loop.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 interface B {
3     public void x(double i) throws IOException {
4         System.out.println("c");
5     }
6 }
7 class C implements B {
8     public void x(int i) throws FileNotFoundException {
9         System.out.println("b");
10    }
11 }
12 class A {
13     public static void main(String[] args) throws IOException {
14         new C().x(3);
15     }
16 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e não imprime nada.
- e) Compila e dá exception.
- f) Compila e entra em loop.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x(int i) throws IOException {
4         System.out.println("c");
5     }
6 }
7 abstract class C extends B throws IOException {
8     abstract void x(int i);
9 }
10 abstract class D extends C {
11     void x(int i) throws IOException {
12         System.out.println("d");
13     }
14 }
15 class E extends D {
16 }
17 class A {
18     public static void main(String[] args) throws IOException {
19         new E().x(32);
20     }
21 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e imprime 'd'.
- e) Compila e não imprime nada.
- f) Compila e dá exception.
- g) Compila e entra em loop.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x(int i) throws IOException {
4         if(i<0) return;
5         x(-1);
6         System.out.println("c");
7     }
8 }
9 abstract class C extends B {
10    void x(int i) throws IOException {
11        System.out.println("b");
12        super.x(i);
13    }
14 }
15 abstract class D extends C {
16    void x(int i) throws IOException {
17        super.x(i);
18    }
19 }
20 class E extends D {
21 }
22 class A {
23     public static void main(String[] args) throws IOException {
24         new E().x(32);
25     }
26 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e imprime 'd'.
- e) Compila e não imprime nada.
- f) Compila e dá exception.
- g) Compila e entra em loop.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x(int i) throws IOException {
4         if(i<0) return;
5         this.x(-1);
6         System.out.println("c");
7     }
8 }
9 abstract class C extends B {
10    void x(int i) throws IOException {
11        System.out.println("b");
12        super.x(i);
13    }
14 }
15 abstract class D extends C {
16    void x(int i) throws IOException {
17        super.x(i);
18    }
19 }
20 class E extends D {
21 }
22 class A {
23     public static void main(String[] args) throws IOException {
24         new E().x(32);
25     }
26 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e imprime 'b','c'.
- e) Compila e não imprime nada.
- f) Compila e dá exception.
- g) Compila e entra em loop.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 import java.io.*;
2 class B {
3     void x(int i) throws IOException {
4         if(i<0) return;
5         super.x(-1);
6         System.out.println("c");
7     }
8 }
9 abstract class C extends B {
10    void x(int i) throws IOException {
11        System.out.println("b");
12        super.x(i);
13    }
14 }
15 abstract class D extends C {
16    void x(int i) throws IOException {
17        super.x(i);
18    }
19 }
20 class E extends D {
21 }
22 class A {
23     public static void main(String[] args) throws IOException {
24         new E().x(32);
25     }
26 }
```

- a) Não compila.
- b) Compila e imprime 'b'.
- c) Compila e imprime 'c'.
- d) Compila e imprime 'b','c'.
- e) Compila e não imprime nada.
- f) Compila e dá exception.
- g) Compila e entra em loop.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class D extends C {  
2     void x() { System.out.println(1); }  
3 }  
4 class C extends B {  
5     void x() { System.out.println(2); }  
6 }  
7 class B {  
8     void x() { System.out.println(3); }  
9     void y(B b) {  
10         b.x();  
11     }  
12     void y(C c) {  
13         c.x();  
14     }  
15     void y(D d) {  
16         d.x();  
17     }  
18 }  
19 class A {  
20     public static void main(String[] args) {  
21         new B().y(new C());  
22     }  
23 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class D extends C {  
2     void x() { System.out.println(1); }  
3 }  
4 class C extends B {  
5     void x() { System.out.println(2); }  
6 }  
7 class B {  
8     void x() { System.out.println(3); }  
9     void y(B b) {  
10         b.x();  
11     }  
12     void y(C c) {  
13         c.x();  
14     }  
15     void y(D d) {  
16         d.x();  
17     }  
18 }  
19 class A {  
20     public static void main(String[] args) {  
21         new B().y(new C());  
22     }  
23 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class D extends C {  
2     void x() { System.out.println(1); }  
3 }  
4 class C extends B {  
5     void x() { System.out.println(2); }  
6 }  
7 class B {  
8     void x() { System.out.println(3); }  
9     void y(B b) {  
10         b.x();  
11     }  
12 }  
13 class A {  
14     public static void main(String[] args) {  
15         new B().y(new C());  
16     }  
17 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class D extends C {
2     void x() { System.out.println(1); }
3     void y(C b) {
4         x();
5     }
6 }
7 class C extends B {
8     void x() { System.out.println(2); }
9 }
10 class B {
11     void x() { System.out.println(3); }
12     void y(B b) {
13         b.x();
14     }
15 }
16 class A {
17     public static void main(String[] args) {
18         new B().y(new C());
19     }
20 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class D extends C {  
2     void x() { System.out.println(1); }  
3     void y(C b) {  
4         x();  
5     }  
6 }  
7 class C extends B {  
8     void x() { System.out.println(2); }  
9 }  
10 class B {  
11     void x() { System.out.println(3); }  
12     void y(B b) {  
13         b.x();  
14     }  
15 }  
16 class A {  
17     public static void main(String[] args) {  
18         new D().y(new C());  
19     }  
20 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e imprime 3.

- 6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 package financeiro;
2 public class ContaFinanceira extends modelo.Conta {
3     void fecha() {
4         System.out.println("fechando financeiro");
5     }
6 }
7
8 package modelo;
9 public class Conta {
10     void fecha() {
11         System.out.println("fechando conta normal");
12     }
13 }
14
15 package codigo;
16 import modelo.*;
17 import financeiro.*;
18 class A {
19     public static void main(String[] args) {
20         new Conta().fecha();
21     }
22 }
```

- a) Não compila.
- b) Compila e roda jogando exception.
- c) Compila e roda, imprimindo 'fechando financeiro'.
- d) Compila e roda, imprimindo 'fechando conta normal'.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
package financeiro;
public class ContaFinanceira extends modelo.Conta {
    void fecha() {
        System.out.println("fechando financeiro");
    }
}

package modelo;
public class Conta {
    public void fecha() {
        System.out.println("fechando conta normal");
    }
}

1 package modelo;
2 import modelo.*;
3 import financeiro.*;
4 class A {
5     public static void main(String[] args) {
6         new Conta().fecha();
7     }
8 }
```

- a) Não compila.
- b) Compila e roda jogando exception.
- c) Compila e roda, imprimindo 'fechando financeiro'.
- d) Compila e roda, imprimindo 'fechando conta normal'.

- 8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 package financeiro;
2 public class ContaFinanceira extends modelo.Conta {
3     void fecha() {
4         System.out.println("fechando financeiro");
5     }
6 }
7
8 package modelo;
9 public class Conta {
10     protected void fecha() {
11         System.out.println("fechando conta normal");
12     }
13 }
14
15 package codigo;
16 import modelo.*;
17 import financeiro.*;
18 class A {
19     public static void main(String[] args) {
20         new Conta().fecha();
21     }
22 }
```

- a) Não compila.
- b) Compila e roda jogando exception.
- c) Compila e roda, imprimindo 'fechando financeiro'.
- d) Compila e roda, imprimindo 'fechando conta normal'.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 package financeiro;
2 public class ContaFinanceira extends modelo.Conta {
3     public void fecha() {
4         System.out.println("fechando financeiro");
5     }
6 }

7 package modelo;
8 public class Conta {
9     public void fecha() {
10        System.out.println("fechando conta normal");
11    }
12 }

13 package codigo;
14 class A {
15     public static void main(String[] args) {
16         new Conta().fecha();
17     }
18 }
```

- a) Não compila.
- b) Compila e roda jogando exception.
- c) Compila e roda, imprimindo ‘fechando financeiro’.
- d) Compila e roda, imprimindo ‘fechando conta normal’.

10) O que acontece com o código a seguir?

```
interface Veiculo {  
    int getMarcha();  
    void liga();  
}  
  
abstract class Carro implements Veiculo {  
    public void liga() {  
        System.out.println("ligado!");  
    }  
}  
  
class CarroConcreto extends Carro implements Veiculo {  
    public int getMarcha() {  
        return 1;  
    }  
}
```

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {};
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         B b = new C();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda sem exception.
- d) Compila e roda dando exception.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         C c = (C) new B();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda sem exception.
- d) Compila e roda dando exception.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         Y y = new D();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda sem exception.
- d) Compila e roda dando exception.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         Y y = (Y) new D();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda sem exception.
- d) Compila e roda dando exception.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         Z z = (Z) (B) new D();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda sem exception.
- d) Compila e roda dando exception.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         Y y = (Y) new A();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda sem exception.
- d) Compila e roda dando exception.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         D d = (D) (Y) (B) new D();
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e roda.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         System.out.println(((B) (Z) (W) (Y) new D()) instanceof D);
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e imprime `true`.
- d) Compila e imprime `false`.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 interface Z {}
2 interface W {}
3 interface Y extends Z, W {}
4 class B {}
5 class C extends B implements Y {}
6 class D extends B implements Z, W {}
7 class E extends C {}
8 class A {
9     public static void main(String[] args) {
10         System.out.println(((B) (Z) (W) (Y) new D()) instanceof D);
11     }
12 }
```

- a) Não compila na definição das classes e interfaces.
- b) Não compila dentro do método `main`.
- c) Compila e imprime `true`.
- d) Compila e imprime `false`.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     int x = 1;  
3 }  
4 class A extends B {  
5     static int x = 2;  
6     public static void main(String[] args) {  
7         System.out.println(x);  
8     }  
9 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e dá exception.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     int x = 1;  
3 }  
4 class A extends B {  
5     static int x = 2;  
6     public static void main(String[] args) {  
7         System.out.println(this.x);  
8     }  
9 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e dá exception.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     int x = 1;  
3 }  
4 class A extends B {  
5     static int x = 2;  
6     public static void main(String[] args) {  
7         System.out.println(super.x);  
8     }  
9 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e dá exception.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     int x = 1;  
3 }  
4 class A extends B {  
5     static int x = 2;  
6     public static void main(String[] args) {  
7         System.out.println(new A().super.x);  
8     }  
9 }
```

- a) Não compila.
- b) Compila e imprime 1.
- c) Compila e imprime 2.
- d) Compila e dá exception.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     void B() {  
3     }  
4     void B(String s) {  
5         this();  
6         this(s);  
7     }  
8 }  
9 class A {  
10    public static void main(String[] args) {  
11        B b = new B();  
12    }  
13 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e não imprime nada.
- d) Compila e entra em loop infinito.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {
2     B() {
3     }
4     B(String s) {
5         this();
6         this(s);
7     }
8 }
9 class A {
10     public static void main(String[] args) {
11         B b = new B();
12     }
13 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e não imprime nada.
- d) Compila e entra em loop infinito.

7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     B() {  
3     }  
4     B(String s) {  
5         this();  
6     }  
7 }  
8 class A {  
9     public static void main(String[] args) {  
10         B b = new B();  
11     }  
12 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e não imprime nada.
- d) Compila e entra em loop infinito.

8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     B() {  
3     }  
4     B(String s) {  
5         this();  
6     }  
7 }  
8 class A {  
9     public static void main(String[] args) {  
10         String s = null;  
11         B b = new B(s);  
12     }  
13 }
```

- a) Não compila.
- b) Compila e dá exception.
- c) Compila e não imprime nada.
- d) Compila e entra em loop infinito.

9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {  
2     int x() { return y();}  
3     int y() { return 3; }  
4 }  
5 class C extends B {  
6     C() {  
7         this(x());  
8     }  
9     C(int i) {  
10        System.out.println(i);  
11    }  
12     int y() { return 2; }  
13 }  
14 class A {  
15     public static void main(String[] args) {  
16         new C();  
17     }  
18 }
```

- a) Não compila.
- b) Compila e imprime '2'.
- c) Compila e imprime '3'.

10) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class B {
2     int x() { return y(); }
3     int y() { return 3; }
4 }
5 class C extends B {
6     C() {
7         super();
8         z(x());
9     }
10    void z(int i) {
11        System.out.println(i);
12    }
13    int y() { return 2; }
14 }
15 class A {
16     public static void main(String[] args) {
17         new C();
18     }
19 }
```

- a) Não compila.
- b) Compila e imprime '2'.
- c) Compila e imprime '3'.

- 1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 abstract class B {  
2     void x() {  
3         System.out.println(y());  
4     }  
5     abstract int y();  
6 }  
7 abstract class C extends B {  
8     int y() { return 1; }  
9 }  
10 class D extends C {  
11     int y() { return 2; }  
12 }  
13 class A {  
14     public static void main(String[] args) {  
15         D d = (D) (C) new D();  
16         d.x();  
17     }  
18 }
```

- a) Não compila.
- b) Compila e imprime '1'.
- c) Compila e imprime '2'.
- d) Compila e roda com exception.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 abstract class B {  
2     abstract void x() {  
3         System.out.println(y());  
4     }  
5     abstract int y();  
6 }  
7 abstract class C extends B {  
8     int y() { return 1; }  
9 }  
10 class D extends C {  
11     int y() { return 2; }  
12 }  
13 class A {  
14     public static void main(String[] args) {  
15         D d = (D) (C) new D();  
16         d.x();  
17     }  
18 }
```

- a) Não compila.
- b) Compila e imprime '1'.
- c) Compila e imprime '2'.
- d) Compila e roda com exception.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 abstract class B {  
2     void x() {  
3         System.out.println(y());  
4     }  
5     abstract int y();  
6 }  
7 abstract class C extends B {  
8     abstract int y();  
9 }  
10 class D extends C {  
11     int y() { return 1; }  
12 }  
13 class A {  
14     public static void main(String[] args) {  
15         D d = (D) (C) new D();  
16         d.x();  
17     }  
18 }
```

- a) Não compila.
- b) Compila e imprime '1'.
- c) Compila e imprime '2'.
- d) Compila e roda com exception.

4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 abstract class B {  
2     void x() {  
3         System.out.println(y());  
4     }  
5     int y() {  
6         return 2;  
7     }  
8 }  
9 abstract class C extends B {  
10    abstract int y();  
11 }  
12 class D extends C {  
13     int y() { return 1; }  
14 }  
15 class A {  
16     public static void main(String[] args) {  
17         D d = (D) (C) new D();  
18         d.x();  
19     }  
20 }
```

- a) Não compila.
- b) Compila e imprime '1'.
- c) Compila e imprime '2'.
- d) Compila e roda com exception.

5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 abstract class B {
2     void x() {
3         System.out.println(y());
4     }
5     final int y() {
6         return 2;
7     }
8 }
9 abstract class C extends B {
10    int y() {
11        return 3;
12    }
13 }
14 class D extends C {
15    int y() { return 1; }
16 }
17 class A {
18     public static void main(String[] args) {
19         D d = (D) (C) new D();
20         d.x();
21     }
22 }
```

- a) Não compila.
- b) Compila e imprime '1'.
- c) Compila e imprime '2'.
- d) Compila e roda com exception.

6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 abstract class B {  
2     void x() {  
3         System.out.println(y());  
4     }  
5     Object y() { return "a"; }  
6 }  
7 abstract class C extends B {  
8     abstract String y();  
9 }  
10 class D extends C {  
11     String y() { return "b"; }  
12 }  
13 class A {  
14     public static void main(String[] args) {  
15         D d = (D) (C) new D();  
16         d.x();  
17     }  
18 }
```

- a) Não compila.
- b) Compila e imprime 'a'.
- c) Compila e imprime 'b'.
- d) Compila e roda com exception.

1) Dentre as classes a seguir qual delas não é checked?

- `java.io.IOException`
- `java.sql.SQLException`
- `java.lang.Exception`
- `java.lang.IndexOutOfBoundsException`
- `java.io.FileNotFoundException`

1) Escolha 2 opções.

Exception é um mecanismo para...

- tratar entrada de dados do usuário.
- que você pode usar para determinar o que fazer quando algo inesperado acontece.
- que a VM usa para fechar o programa caso algo inesperado aconteça.
- controlar o fluxo da aplicação.
- separar o tratamento de erros da lógica principal.

2) De que maneira a API de exceptions pode ajudar a melhorar o código de seu programa?

Escolha 2 opções:

- Permitindo separar o tratamento de erro da lógica do programa.
- Permitindo tratar o erro no mesmo ponto onde ele ocorre.
- Permitindo estender as classes que já existem e criar novas exceptions.
- Disponibilizando várias classes com todas as exceptions possíveis prontas.
- Aumentando a segurança da aplicação disponibilizando os erros nos logs.

1) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String nome;  
4         try {  
5             nome.toLowerCase();  
6             System.out.println("a");  
7         } catch(NullPointerException ex) {  
8             System.out.println("b");  
9         }  
10        System.out.println("c");  
11    }  
12 }
```

- a) Não compila.
- b) Compila e, ao rodar, imprime “abc”.
- c) Compila e, ao rodar, imprime “bc”.
- d) Compila e, ao rodar, imprime “a”.
- e) Compila e, ao rodar, imprime “b”.

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String nome = null;  
4         try {  
5             nome.toLowerCase();  
6             System.out.println("a");  
7         } catch(NullPointerException ex) {  
8             System.out.println("b");  
9         }  
10        System.out.println("c");  
11    }  
12 }
```

- a) Não compila..
- b) Compila e, ao rodar, imprime “abc”.
- c) Compila e, ao rodar, imprime “bc”.
- d) Compila e, ao rodar, imprime “a”.
- e) Compila e, ao rodar, imprime “b”.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         String nome;  
4         try {  
5             nome.toLowerCase();  
6             System.out.println("a");  
7         } catch(NullPointerException ex) {  
8             System.out.println("b");  
9         } finally {  
10            System.out.println("c");  
11        }  
12        System.out.println("d");  
13    }  
14 }
```

- a) Não compila.
- b) Compila e, ao rodar, imprime “abcd”.
- c) Compila e, ao rodar, imprime “bcd”.
- d) Compila e, ao rodar, imprime “ac”.
- e) Compila e, ao rodar, imprime “bc”.
- f) Compila e, ao rodar, imprime “ad”.
- g) Compila e, ao rodar, imprime “bd”.

- 1) Qual classe podemos colocar no código a seguir para que ele compile?

```
1 import java.io.*;
2 class X {
3     InputStream y() throws NOME_AQUI {
4         return new FileInputStream("a.txt");
5     }
6     void z() throws NOME_AQUI{
7         InputStream is = y();
8         is.close();
9     }
10 }
```

* java.io.IOException * java.sql.SQLException *
java.lang.Exception* java.lang.IndexOutOfBoundsException
* java.io.FileNotFoundException

2) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     void m2() {  
3         System.out.println("e");  
4         int[][]x = new int[15][20];  
5         System.out.println("f");  
6     }  
7     void m() {  
8         System.out.println("c");  
9         m2();  
10        System.out.println("d");  
11    }  
12    public static void main(String[] args) {  
13        System.out.println("a");  
14        new A().m();  
15        System.out.println("b");  
16    }  
17 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

3) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:

```
1 class A {  
2     void m2() {  
3         System.out.println("e");  
4         int[] x = new int[15];  
5         x[20] = 13;  
6         System.out.println("f");  
7     }  
8     void m() {  
9         System.out.println("c");  
10        m2();  
11        System.out.println("d");  
12    }  
13    public static void main(String[] args) {  
14        System.out.println("a");  
15        new A().m();  
16        System.out.println("b");  
17    }  
18 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 4) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class A {  
2     void m2() {  
3         System.out.println("e");  
4         new java.io.FileInputStream("a.txt");  
5         System.out.println("f");  
6     }  
7     void m() {  
8         System.out.println("c");  
9         m2();  
10        System.out.println("d");  
11    }  
12    public static void main(String[] args) {  
13        System.out.println("a");  
14        new A().m();  
15        System.out.println("b");  
16    }  
17 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 5) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class A {  
2     void m2() throws java.io.FileNotFoundException {  
3         System.out.println("e");  
4         new java.io.FileInputStream("a.txt");  
5         System.out.println("f");  
6     }  
7     void m() throws java.io.IOException {  
8         System.out.println("c");  
9         m2();  
10        System.out.println("d");  
11    }  
12    public static void main(String[] args)  
13        throws java.io.FileNotFoundException {  
14        System.out.println("a");  
15        new A().m();  
16        System.out.println("b");  
17    }  
18 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 6) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class A {  
2     void m2() throws java.io.FileNotFoundException {  
3         System.out.println("e");  
4         new java.io.FileInputStream("a.txt");  
5         System.out.println("f");  
6     }  
7     void m() throws java.io.FileNotFoundException {  
8         System.out.println("c");  
9         m2();  
10        System.out.println("d");  
11    }  
12    public static void main(String[] args) throws  
13        java.io.IOException {  
14        System.out.println("a");  
15        new A().m();  
16        System.out.println("b");  
17    }  
18 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 7) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class A {  
2     void m2() throws java.io.FileNotFoundException {  
3         System.out.println("e");  
4         new java.io.FileInputStream("a.txt");  
5         System.out.println("f");  
6     }  
7     void m() throws java.io.FileNotFoundException {  
8         System.out.println("c");  
9         try {  
10             m2();  
11         } catch(java.io.FileNotFoundException ex) {  
12         }  
13         System.out.println("d");  
14     }  
15     public static void main(String[] args) throws  
16         java.io.IOException {  
17         System.out.println("a");  
18         new A().m();  
19         System.out.println("b");  
20     }  
21 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime acedb.
- f) Compila e imprime ace, joga uma Exception e imprime db.

- 8) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class MyException extends RuntimeException {  
2  
3 }  
4 class A {  
5     void m2() throws java.io.FileNotFoundException {  
6         System.out.println("e");  
7         new MyException();  
8         System.out.println("f");  
9     }  
10    void m() throws java.io.FileNotFoundException {  
11        System.out.println("c");  
12        try {  
13            m2();  
14        } catch(java.io.FileNotFoundException ex) {  
15            }  
16        System.out.println("d");  
17    }  
18    public static void main(String[] args) throws  
19        java.io.IOException {  
20        System.out.println("a");  
21        new A().m();  
22        System.out.println("b");  
23    }  
24 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 9) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class MyException extends RuntimeException {  
2  
3 }  
4 class A {  
5     void m2() throws java.io.FileNotFoundException {  
6         System.out.println("e");  
7         throws new MyException();  
8         System.out.println("f");  
9     }  
10    void m() throws java.io.FileNotFoundException {  
11        System.out.println("c");  
12        try {  
13            m2();  
14        } catch(java.io.FileNotFoundException ex) {  
15        }  
16        System.out.println("d");  
17    }  
18    public static void main(String[] args) throws  
19        java.io.IOException {  
20        System.out.println("a");  
21        new A().m();  
22        System.out.println("b");  
23    }  
24 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 10) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class MyException extends RuntimeException {  
2  
3 }  
4 class A {  
5     void m2() throws java.io.FileNotFoundException {  
6         System.out.println("e");  
7         boolean sim = true;  
8         if(sim) throws new MyException();  
9         System.out.println("f");  
10    }  
11    void m() throws java.io.FileNotFoundException {  
12        System.out.println("c");  
13        try {  
14            m2();  
15        } catch(java.io.FileNotFoundException ex) {  
16            }  
17            System.out.println("d");  
18        }  
19        public static void main(String[] args) throws  
20        java.io.IOException {  
21            System.out.println("a");  
22            new A().m();  
23            System.out.println("b");  
24        }  
25 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 11) Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir, sem que o arquivo exista?

```
1 class MyException extends RuntimeException {  
2  
3 }  
4 class A {  
5     void m2() throws java.io.FileNotFoundException {  
6         System.out.println("e");  
7         boolean sim = true;  
8         if(sim) throw new MyException();  
9         System.out.println("f");  
10    }  
11    void m() throws java.io.FileNotFoundException {  
12        System.out.println("c");  
13        try {  
14            m2();  
15        } catch(java.io.FileNotFoundException ex) {  
16            System.out.println("d");  
17        }  
18    }  
19    public static void main(String[] args) throws  
20        java.io.IOException {  
21        System.out.println("a");  
22        new A().m();  
23        System.out.println("b");  
24    }  
25 }
```

- a) Não compila.
- b) Compila e imprime acefdb.
- c) Compila e imprime ace e joga uma Exception.
- d) Compila e imprime acedb e joga uma Exception.
- e) Compila e imprime ace, joga uma Exception e imprime db.

- 1) Escolha a opção adequada que indica o `Throwable` que ocorrerá no código a seguir:

```
1 class A {  
2     public static void main(String[] args) {  
3         main(args);  
4     }  
5 }
```

- a) `IndexOutOfBoundsException`
- b) `ArrayIndexOutOfBoundsException`
- c) `NullPointerException`
- d) `OutOfMemoryError`
- e) `StackOverflowError`
- f) `ExceptionInInitializationError`

2) Escolha a opção adequada que indica o `Throwable` que ocorrerá no código a seguir:

```
1 import java.util.*;
2 class A {
3     public static void main(String[] args) {
4         ArrayList<String> strings = new ArrayList<String>();
5         for(int i=0;i<10;i++) {
6             for(int j=0;j<10;i++)
7                 strings.add("string " + i + " " + j);
8         System.out.println(strings.get(99999));
9     }
10 }
```

- a) `IndexOutOfBoundsException`
- b) `ArrayIndexOutOfBoundsException`
- c) `NullPointerException`
- d) `OutOfMemoryError`
- e) `StackOverflowError`
- f) `ExceptionInInitializationError`

```
1 // O que acontece ao tentar compilar e executar o código a seguir
2 class Test{
3     public static void main(String[] args){
4         System.out.print("a");
5         System.out.println('b'); // A
6         System.out.print();      // B
7         System.out.println("c");
8     }
9 }
10
11
12 /*
13     a) Não compila por erro na Linha B
14     b) compila e imprime
15     c) Não compila por erro na Linha A
16 */
```

```
1 // Ao tentar compilar e executar o código a seguir, o que acontece:  
2 public class Test {  
3     public static void main(String[] args) {  
4         System.out.print("a");  
5         System.out.println("b");  
6         System.out.printf("c");  
7         System.out.print("d");  
8         System.out.println("\n");  
9         System.out.print("e");  
10    }  
11 }  
12  
13  
14 /*  
15     a) ab  
16     cd  
17     e  
18     b) não compila  
19     c) ab  
20     c  
21     d  
22  
23     e  
24     d) ab  
25     cd  
26  
27     e  
28 */
```

```
1 // O que acontece ao compilar e executar o código a seguir?
2 public class Test {
3     public static void main(String[] args) {
4         System.out.printf("%s",12); //A
5         System.out.printf("%d",new Integer(321)); //B
6         System.out.printf("%d", (short)(byte)(double) 127); //C
7     }
8 }
9
10 /*
11     a) Compila e executa normalmente
12     b) Erro de compilação na Linha B
13     c) Erro de compilação na Linha C
14     d) Erro de compilação na Linha A
15     e) Erro de compilação nas linhas B e C
16 */
```

```
/* Qual dos códigos a seguir NÃO imprime >00012.45<? Escolha 1 alternativa:  
a) System.out.printf(">%-(8.2f<",12.45);  
b) System.out.printf(">%0,8.2f<",12.45);  
c) System.out.format(">%1$08.2f<",12.45);  
d) System.out.printf(">%0,(8.2f<",12.45);  
e) System.out.format(">%0(8.2f<",12.45);  
*/
```

```
2 // Ao tentar compilar e executar o seguinte código, o que é impresso?
3 public class Testes {
4     public static void main(String[] args) {
5         System.out.println(new char[]{'a','b','c'}); // A
6         System.out.println(new byte[]{'a','b','c'}); // B
7         System.out.println("abc"); // C
8         System.out.println(new String[]{"abc"}); // D
9     }
10 }
11
12 /*
13     a) Todas as Linhas imprimem abc
14     b) Linhas A e C imprimem abc
15     c) Linhas A, C e D imprimem abc
16     d) Linha C imprime abc
17     e) Não compila na Linha B
18 */
```

```
25 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
26 class A {
27     public static void main(String[] args) {
28         int i = 10;
29         m1(i);
30     }
31
32     private static void m1(Integer j) {
33         System.out.println("go!");
34     }
35 }
36
37 /*
38     a) Imprime 10
39     b) Imprime go!
40     c) Compila mas Lança exception
41     d) Erro de compilação
42 */
```

```
48 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
49 class A {
50
51     static int i;
52
53     public static void main(String[] args) {
54         i = Integer.parseInt("10");
55         m1(i + 1);
56     }
57
58     private static void m1(Integer j) {
59         System.out.println(i);
60     }
61 }
62
63 /*
64 a) Compila mas Lança exception
65 b) Imprime 11
66 c) Imprime 10
67 d) Erro de compilação
68 */
```

```
79 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
80 class A {
81
82     static int i;
83
84     public static void main(String[] args) {
85         i = Integer.valueOf("10");
86         m1(i + 1);
87     }
88
89     private static void m1(Integer j) {
90         System.out.println(j);
91     }
92 }
93
94 /*
95 a) Compila mas lança exception
96 b) Imprime 11
97 c) Erro de compilação
98 d) Imprime 10
99 */
```

```
111 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
112 public class Test {
113     static long i;
114
115     public static void main(String[] args) {
116         i = Integer.valueOf("10",8); // A
117         m1(i); // B
118     }
119
120     private static void m1(Integer j) { // C
121         System.out.println(j);
122     }
123 }
124
125 /*
126 a) Erro de compilação na Linha B
127 b) Erro de compilação na Linha C
128 c) Erro de compilação na LinhaA
129 d) Compila mas Lança exception
130 e) Imprime 10
131 */
```

```
139 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
140 public class Test {
141     public static void main(String[] args) {
142         int a = Short.parseShort("126"); // A
143         short s = Integer.parseInt("23").shortValue(); //B
144         double h = Double.valueOf("27").floatValue(); //C
145         System.out.println(a + s);
146     }
147 }
148
149 /*
150 a) Erro de compilação na Linha C
151 b) Imprime 149
152 c) Erro de compilação na Linha A
153 d) Imprime 176
154 e) Erro de compilação na Linha B
155 */
```

```
164 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
165 public class Test {
166     public static void main(String[] args) {
167         char c = new Character('x').charValue();
168         System.out.println(c);
169     }
170 }
171
172 /*
173     a) Erro de compilação
174     b) Imprime 120
175     c) Imprime x
176 */
```

```
185 // Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
186 public class Test {
187     public static void main(String[] args) {
188         int a = Integer.parseInt("10",2);
189         int b = a == 10 ? null : 3;
190         System.out.println(a + b);
191     }
192 }
193
194 /*
195     a) Imprime 5
196     b) Ao executar Lança NullPointerException
197     c) Imprime 13
198     d) Erro de compilação.
199     e) Imprime 10
200 */
```

```
/*
    Todas as classes da nova API de datas são:
    a) mutáveis
    b) imutáveis
*/

/*
    A classe que representa um horário sem data é:
    a) Time
    b) LocalTime
    c) TimeZone
    d) LocalDateTime
*/

/*
    Qual código cria um objeto com a data e hora atual?
    a) LocalDateTime.current()
    b) LocalDateTime.now()
    c) new LocalDateTime()
    d) LocalDate.now()
*/
```

```
/*
Qual o código que calcula a diferença em dias entre dois LocalDate d1 e d2?
a) ChronoUnit.DAYS.between(d1,d2);
b) ChronoUnit.DAYS.difference(d1,d2);
c) Period.between(d1,d2);
d) ChronoUnit.Days.between(d1,d2);
*/
```

```
// Qual trecho que, se inserido no Local, indicado irá imprimir corretamente a idade?
public class Test {
    public static void main(String[] args) {
        LocalDate birthday = LocalDate.of(1975, 9, 23);
        LocalDate today = LocalDate.now();

        //code here

        System.out.println("You are " + d.getYears() + " years, " +
                           d.getMonths() + " months, and " +
                           d.getDays() + " days old.");
    }
}

/*
a) Period d = Period.difference(birthday, today)
b) Duration d = Duration.between(birthday, today)
c) Period d = Period.between(today,birthday)
d) Period d = Period.between(birthday, today)
*/
```

```
/*
Qual o trecho de código que converte um objeto Date em um objeto LocalDate?
a) LocalDateTime.ofInstant(d.toInstant()).toLocalDate()
b) LocalDateTime.ofInstant(d.toInstant(), ZoneId.systemDefault()).toLocalDate()
c) LocalDateTime.from(d.toInstant()).toLocalDate()
d) LocalDate.from(d.toInstant())
*/
```

```
// Sobre o código, ao tentarmos compilar e executar, o que acontece?
public class Test {
    public static void main(String[] args) {
        System.out.println(YearMonth.now().isSupported(
            ChronoField.DAY_OF_MONTH));
        System.out.println(MonthDay.now().isSupported(
            ChronoUnit.DAYS));
        System.out.println(LocalDate.now().isSupported(
            ChronoUnit.DAYS));
        System.out.println(LocalDateTime.now().isSupported(
            ChronoField.DAY_OF_MONTH));
    }
}

/*
a) Compila e imprime true,true,true,true
b) Compila e imprime true,true,false,true
c) Compila e imprime false,true,true,true
d) Compila, mas lança exception ao executar
e) Não compila
*/
```

```
// Sobre o código, ao tentarmos compilar e executar, o que acontece?
class Test {
    public static void main(String[] args) {
        System.out.println(YearMonth.now().isSupported(
            ChronoUnit.MONTHS));
        System.out.println(YearMonth.now().isSupported(
            ChronoField.DAY_OF_MONTH));
        System.out.println(MonthDay.now().isSupported(
            ChronoField.DAY_OF_MONTH));
        System.out.println(LocalDate.now().isSupported(
            ChronoUnit.DAYS));
        System.out.println(LocalDate.now().isSupported(
            ChronoUnit.YEAR));
        System.out.println(LocalDateTime.now().isSupported(
            ChronoField.HOUR_OF_AMPM));
        System.out.println(LocalDateTime.now().isSupported(
            ChronoField.YEAR));
    }
}

/*
a) Compila e imprime true,false,true,true, true,true,true,true
b) Compila e imprime true,true,true,true, true,true,false,true
c) Compila, mas Lança exception ao executar
d) Não compila
*/
```

```
// Qual código, se incluído na linha indicada, imprime a mensagem "Hello World"?
interface Printer{
    void printMessage();
}

class A {
    public static void main(String[] args) {
        Printer p = null;
        //code here
        p.printMessage();
    }
}

/*
a) p = () -> System.out.println("Hello World");
b) p = -> System.out.println("Hello World");
c) p = () => System.out.println("Hello World");
d) p -> System.out.println("Hello World");
*/
```

```
/*
Qual das expressões a seguir é inválida?
a) Predicate<String> startsWithA = s -> s.startsWith("A")
b) Predicate<Integer> big = list -> list.size() > 100
c) Predicate<List> bigger = list -> list.size() > 1000
*/
```

```
// Considere a seguinte interface:  
interface Calculator<T>{  
    T function(T a, T b);  
}  
  
/*  
Qual dos códigos a seguir não é valido?  
a) Calculator<Integer> multiply = (Integer a, Integer b) -> (int) a * b  
b) Calculator<Integer> subtract = (a,b) -> {return a - b;}  
c) Calculator<Integer> sum = (a,b) -> a + b  
d) Calculator<Integer> divide = (int a, int b) -> {return (Integer) a / b;}  
*/
```

```
// Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
class A {
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++){
            new Thread(() -> System.out.println(i)).start();
        }
    }
}

/*
a) Imprime os números de 0 a 9
b) Imprime 0 dez vezes
c) Compila e executa, mas não imprime nada
d) Erro de compilação
*/
```

```
// Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
import java.util.function.Predicate;

class A{
    int a = 0;

    public static void main(String[] args) {
        new A().method(1, a -> a > 0); // A
    }

    private void method(int a, Predicate<Integer> d) {
        if(d.test(a)){ // B
            System.out.println(a);
        }
    }
}

/*
a) Compila e não imprime nada
b) Compila e imprime 0
c) Não compila por erro na Linha B
d) Compila e imprime 1
e) Não compila por erro na Linha A
*/
```

```
// Escolha a opção adequada ao tentar compilar e rodar o arquivo a seguir:
class A{
    int a = 0;

    public static void main(String[] args) {
        new A().method(1);
    }

    private void method(int a) {
        Predicate<Integer> d = a -> a > 0; // A

        if(d.test(a)){ // B
            System.out.println(a);
        }
    }
}

/*
a) Compila e não imprime nada
b) Não compila por erro na linha A
c) Não compila por erro na linha B
d) Compila e imprime 0
e) Compila e imprime 1
*/
```

/*

Para aumentar o encapsulamento de uma classe, devemos:

Deixar os métodos e atributos privados

Deixar os métodos e atributos públicos

Deixar os métodos públicos e atributos públicos

Deixar os métodos públicos e atributos privados

Deixar os métodos privados e atributos públicos

a) Deixar os métodos públicos e atributos públicos

b) Deixar os métodos privados e atributos públicos

c) Deixar os métodos e atributos privados

d) Deixar os métodos e atributos públicos

e) Deixar os métodos públicos e atributos privados

*/

```
// Given:  
class Product {  
    double price;  
}  
  
class Test {  
    public void updatePrice(Product product, double price) {  
        price = price * 2;  
        product.price = product.price + price;  
    }  
    public static void main(String[] args) {  
        Product prt = new Product();  
        prt.price = 200;  
        double newPrice = 100;  
  
        Test t = new Test();  
        t.updatePrice(prt, newPrice);  
        System.out.println(prt.price + " : " + newPrice);  
    }  
}  
/*  
A) 200.0 : 100.0  
B) 400.0 : 200.0  
C) 400.0 : 100.0  
D) Compilation fails  
*/
```

```
// Given the code fragment
class Test {
    public static void main(String[] args) {
        int aVar = 9;

        if(aVar++ < 10) {
            System.out.println(aVar + " Hello World!");
        } else {
            System.out.println(aVar + " Hello Universe!");
        }
    }

/* WHAT'S THE RESULT IF THE INTEGER aVar IS 9
A) Hello World!
B) Hello Universe!
C) Hello World!
D) Compilation fails
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        Short s1 = 200;  
        Integer s2 = 400;  
        Long s3 = (long) s1 + s2;           // line n1  
        String s4 = (String) (s3 + s2);     // line n2  
        System.out.println("Sum is " + s4);  
    }  
}  
  
/* WHAT'S THE RESULT:  
 A) Sum is 600  
 B) Compilation fails at line n1  
 C) Compilation fails at line n2  
 D) A ClassCastException is thrown at line n1  
 E) A ClassCastException is thrown at line n2  
 */
```

```
/* What is the name of the Java concept that uses access  
modifiers to protect variables and hide them within a class ?  
  
A) Encapsulation  
B) Inheritance  
C) Abstraction  
D) Instantiation  
E) Polymorphism  
  
Reference: http://www.tutorialspoint.com/java/java\_access\_modifiers.htm  
*/
```

```
// Given the code fragment:  
abstract class Planet {  
    protected void revolve() {          // line n1  
    }  
  
    abstract void rotate();           // line n2  
}  
  
class Earth extends Planet {  
    void revolve() {                  // line n3  
    }  
  
    protected void rotate() {         // line n4  
    }  
}  
  
/* Which two modifications, made independently, enable the code  
to compile ?  
  
A) Make the method at line n1 public  
B) Make the method at line n2 public  
C) Make the method at line n3 public  
D) Make the method at line n3 protected  
E) Make the method at line n4 public  
*/
```

```
class Vehicle {  
    String type = "4w";  
    int maxSpeed = 100;  
  
    Vehicle(String type, int maxSpeed) {  
        this.type = type;  
        this.maxSpeed = maxSpeed;  
    }  
}  
  
class Car extends Vehicle {  
    String trans;  
  
    Car(String trans) {  
        this.trans = trans;  
    }  
  
    Car(String type, int maxSpeed, String trans) {  
        super(type, maxSpeed);  
        this(trans);  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Car c1 = new Car("Auto");  
        Car c2 = new Car("4w", 150, "Manual");  
        System.out.println(c1.type + " " + c1.maxSpeed + " " + c1.trans);  
        System.out.println(c2.type + " " + c2.maxSpeed + " " + c2.trans);  
    }  
}  
/* WHAT IS THE RESULT ?  
A) aw 100 Auto  
    4w 150 Manual  
B) Null 0 Auto  
    4w 150 Manual  
C) Compilation fails only at line n1  
D) Compilation fails only at line n2  
E) Compilation fails at both line n1 and line n2  
*/
```

```
2 // Given the code fragment:  
3 class X {  
4     public void printFileContent() {  
5         /* code goes here */  
6         throw new IOException();  
7     }  
8 }  
9 class Test {  
10    public static void main(String[] args) {  
11        X xobj = new X();  
12        xobj.printFileContent();  
13    }  
14 }  
15  
16 /* Which two modifications should you make so that the code  
17 compiles successfully ?  
18  
19 A) replace line 10 with:  
20     public static void main(String[] args) throws Exception {  
21 B) replace line 12 with:  
22     try { xobj.printFileContent(); }  
23     catch(Exception e) { }  
24     catch(IOException e) { }  
25 C) replace line 4 with:  
26     public void printFileContent() throws IOException {  
27 D) replace line 6 with:  
28     throw IOException("Exception raised");  
29 E) At line 13, insert:  
30     throw new IOException();  
31 */
```

```

// Given the following two classes:
public class Customer {
    ElectricAccount acct = new ElectricAccount();

    public void useElectricity(double kwh) {
        acct.addKWh(kwh);
    }
}

public class ElectricAccount {
    private double kwh;
    private double rate = 0.07;
    private double bill;

    // line n1
}

```

/* How should you write methods in the ElectricAccount class at line n1 so that the member variable bill is always equal to the value of the member variable kwh multiplied by the member variable rate?

Any amount of electricity used by a customer (represented by an instance of the customer class) must contribute to the customer's bill (represented by the member variable bill) through the method useElectricity method. An instance of the customer class should never be able to tamper with or decrease the value of the member variable bill

*/

```

// A
public void addKwh(double kwh) {
    this.kwh += kwh;
    this.bill = this.kwh * this.rate;
}

// B
public void addKwh(double kwh) {
    if(kwh > 0) {
        this.kwh += kwh;
        this.bill = this.kwh * this.rate;
    }
}

```

```

// C
private void addKwh(double kwh) {
    if(kwh > 0) {
        this.kwh += kwh;
        this.bill = this.kwh * this.rate;
    }
}

// D
public void addKwh(double kwh) {
    if(kwh > 0) {
        this.kwh += kwh;
        setBill(this.kwh);
    }
}

public void setBill(double kwh) {
    bill = kwh * rate;
}

```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        StringBuilder sb = new StringBuilder(5);  
        String s = "";  
  
        if(sb.equals(s)) {  
            System.out.println("Match 1");  
        } else if(sb.toString().equals(s.toString()))  
            System.out.println("Match 2");  
        } else {  
            System.out.println("No Match");  
        }  
    }  
}  
  
/*  
A) Match 1  
B) Match 2  
C) No Match  
D) NullPointerException is thrown at runtime  
*/
```

```
// Given:  
interface Readable {  
    public void readBook();  
    public void setBookMark();  
}  
  
abstract class Book implements Readable {          // line n1  
    public void readBook() { }                      // line n2  
}  
  
class EBook extends Book {                         // line n3  
    public void readBook() { }                      // line n4  
}  
/* Which option enables the code to compile ?  
A) replace the code at line n1 with:  
    class Book implements Readable {  
B) At line n2 insert:  
    public abstract void setBookMark();  
C) replace the code at line n3 with:  
    abstract class EBook extends Book {  
D) At line n4 insert:  
    public void setBookMark() { }  
E) ***|  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String ta = "A ";  
        ta = ta.concat("B ");  
        String tb = "C ";  
        ta = ta.concat(tb);  
        ta.replace('C', 'D');  
        ta = ta.concat(tb);  
        System.out.println(ta);  
    }  
}  
  
/* WHAT IS THE RESULT:  
 A) A B C D  
 B) A C D  
 C) A B C C  
 D) A B D  
 E) A B D C  
 */
```

```
// Given:  
class CD {  
    int r;  
    CD(int r) {  
        this.r = r;  
    }  
}  
  
class DVD extends CD {  
    int c; |  
    DVD(int r, int c) {  
        // line n1  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        DVD dvd = new DVD(10, 20);  
    }  
}
```

/* Which code fragment should you use at line n1
to instantiate the dvd object successfully ?

- A) super.r = r;
 this.c = c;
- B) super(r);
 this(c);
- C) super(r);
 this.c = c;
- D) this.c = r;
 super(c);

*/

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int a[] = {1, 2, 3, 4, 5};  
        for(XXX) {  
            System.out.print(a[e]);  
        }  
    }  
}  
  
/* Which option can replace XXX to enable the  
code to print 135 ?  
  
A) int e = 0; e <= 4; e++  
B) int e = 0; e < 5; e += 2  
C) int e = 1; e <= 5; e += 1  
D) int e = 1; e < 5; e+ =2  
*/
```

```
/* Which statement best describes encapsulation?  
A) Encapsulation ensures that classes can be designed so  
    that only certain fields and methods of an object are  
    accessible from other objects.  
B) Encapsulation ensures that classes can be designed  
    so that their methods are inheritable.  
C) Encapsulation ensures that classes can be designed  
    with some fields and methods declared as abstract.  
D) Encapsulation ensures that classes can be designed  
    so that if a method has an argument MyType x, any  
    subclass of MyType can be passed to that method.
```

*/

```
1 // SalesMan.java
2 package sales;
3 public class SalesMan { }
4
5 // Product.java
6 package sales.products;
7 public class Product { }
8
9 // Market.java
10 package market;
11 insert code here
12 public class USMarket {
13     SalesMan sm;
14     Product p;
15 }
16
17 /* Which code fragment, when inserted at line 14, enables
18    the code to compile ?
19    A) import sales.*;
20    B) import java.sales.products.*;
21    C) import sales;
22        import sales.products.*;
23    D) import sales.*;
24        import products.*;
25    E) import sales.*;
26        import sales.products.*;
27 */
```

```
// Given the following class:  
class CheckingAccount {  
    public int amount;  
  
    public CheckingAccount(int amount) {  
        this.amount = amount;  
    }  
  
    public int getAmount() {  
        return amount;  
    }  
  
    public void changeAmount(int x) {  
        amount += x;  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        CheckingAccount acct = new CheckingAccount((int) (Math.random()*1000));  
        // line n1  
        System.out.println(acct.getAmount());  
    }  
}  
/* Which three lines, when inserted independently at line n1, cause the program  
to print a 0 balance ?  
A) this.amount = 0;  
B) amount = 0;  
C) acct(0);  
D) acct.amount = 0;  
E) acct.getAmount() = 0;  
F) acct.changeAmount(0);  
G) acct.changeAmount(-acct.amount);  
H) acct.changeAmount(-acct.getAmount());  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String shirts[][] = new String[2][2];  
        shirts[0][0] = "red";  
        shirts[0][1] = "blue";  
        shirts[1][0] = "small";  
        shirts[1][1] = "medium";  
    }  
}
```

// Which code fragment prints red: blue: small: medium ?

```
// A  
for(int index = 1; index < 2; index++) {  
    for(int idx = 1; idx < 2; idx++) {  
        System.out.print(shirts[index][idx] + ":";  
    }  
}  
  
// B  
for(int index = 0; index < 2; index++) {  
    for(int idx = 0; idx < index; idx++) {  
        System.out.print(shirts[index][idx] + ":";  
    }  
}  
  
// C  
for(String c : colors) {  
    for(String s : sizes) {  
        System.out.print(s + ":";  
    }  
}  
  
// B  
for(int index = 0; index < 2;) {  
    for(int idx = 0; idx < 2;) {  
        System.out.print(shirts[index][idx] + ":";  
        idx++;  
    }  
    index++;  
}
```

```
// Given the code fragment:  
public class Test {  
    void readCard(int cardNo) throws Exception {  
        System.out.println("Reading card");  
    }  
  
    void checkCard(int cardNo) throws RuntimeException { // line n1  
        System.out.println("Checking Card");  
    }  
  
    public static void main(String[] args) {  
        Test ex = new Test();  
        int cardNo = 12344;  
        ex.checkCard(cardNo); // line n2  
        ex.readCard(cardNo); // line n3  
    }  
}  
  
/* What is the result:  
  
A) reading card  
    checking card  
B) Compilation fails only at line n1  
C) Compilation fails only at line n2  
D) Compilation fails only at line n3  
E) Compilation fails at both line n2 and line n3  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int x = 5;  
        while(isAvailable(x)) {  
            System.out.print(x);  
        }  
    }  
  
    public static boolean isAvailable(int x) {  
        return x-- > 0 ? true : false;  
    }  
}  
  
/* Which modifications enables the code to print 54321 ?  
A) replace line 7 with System.out.print(--x);  
B) at line 1, insert x--  
C) replace line 7 with --x, and, at line 8, insert System.out.print(x);  
D) Replace line 12 With return (x > 0) ? false: true;  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        boolean opt = true;  
        switch(opt) {  
            case true:  
                System.out.print("True");  
                break;  
            default:  
                System.out.print("***");  
        }  
  
        System.out.println("Done");  
    }  
}  
  
/* Which modifications enables the code fragment to print TrueDone ?  
A) Replace line 5 With String result = "true";  
   Replace line 7 with case "true":  
B) Replace line 5 with boolean opt = 1;  
   Replace line 7 with case 1=   
C) At line 9, remove the break statement.  
D) Remove the default section.  
*/
```

```
// Given the following main method:  
class Test {  
    public static void main(String[] args) {  
        int num = 5;  
        do {  
            System.out.print(num-- + " ");  
        } while(num == 0);  
    }  
}  
  
/* Which is the result:  
A) 5 4 3 2 1 0  
B) 5 4 3 2 1  
C) 4 2 1  
D) 5  
E) Nothing is printed  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int x = 100;  
        int a = x++;  
        int b = ++x;  
        int c = x++;  
        int d = (a < b) ? (a < c) ? a: (b < c )? b: c;  
        System.out.println(d);  
    }  
}  
  
/* What is the result ?  
 A) 100  
 B) 101  
 C) 102  
 D) 103  
 E) compilation fails  
 */
```

```
// Given:  
class Test {  
    public static void main(String[] args) {  
        String[][] chs = new String[2][];  
        chs[0] = new String[2];  
        chs[1] = new String[5];  
        int i = 97;  
  
        for(int a = 0; a < chs.length; a++) {  
            for(int b = 0; b < chs.length; b++) {  
                chs[a][b] = "" + i;  
                i++;  
            }  
        }  
  
        for(String[] ca : chs) {  
            for(String c : ca) {  
                System.out.print(c + " ");  
            }  
            System.out.println();  
        }  
    }  
  
/* What is the result ?  
A) 91 98  
   99 100 null null null  
B) 91 98  
   99 100 101 102 103  
C) Compilations fails  
D) A NullPointerException is thrown at runtime.  
E) An ArrayIndexOutOfBoundsException is thrown at runtime.  
*/
```

```

// Given the code fragment:
public class Employee {
    String name;
    boolean contract;
    double salary;

    Employee() {
        // line n1
    }

    public String toString() {
        return name + ":" + contract + ":" + salary;
    }

    public static void main(String[] args) {
        Employee e = new Employee();
        // line n2
        System.out.print(e);
    }
}

/* Which two modifications, when made independently. enable
the code to print joe:true:100.0 ?
A) Replace line n2 with:
    e.name = "Joe";
    e.contract = true;
    e.salary = 100;
B) Replace line n2 with:
    this.name = "Joe";
    this.contract = true;
    this.salary = 100;
C) Replace line n1 with:
    this.name = new String("Joe");
    this.contract = new Boolean(true);
    this.salary = new Double(100);
D) Replace line n1 with:
    name = "Joe";
    contract = TRUE;
    salary = 100.0f;
E) Replace line n1 with:
    this("Joe", true, 100);
*/

```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        List<String> names = new ArrayList<>();  
        names.add("Robb");  
        names.add("Bran");  
        names.add("Rick");  
        names.add("Bran");  
  
        if(names.remove("Bran")) {  
            names.remove("Jon");  
        }  
  
        System.out.println(names);  
    }  
}
```

/* WHAT IS THE RESULT ?

- A) [Robb, Rick, Bran]
- B) [Robb, Rick]
- C) [Robb, Bran, Rick, Bran]
- D) An exception is thrown at runtime

*/

```
// Given:  
class A {  
    public A() {  
        System.out.print("A ");  
    }  
}  
  
class B extends A {  
    public B() { // line n1  
        System.out.print("B ");  
    }  
}  
  
class C extends B {  
    public C() { // line n2  
        System.out.print("C ");  
    }  
}  
  
public static void main(String[] args) {  
    C c = new C();  
}  
}  
  
/* WHAT IS THE RESULT ?  
A) C B A  
B) C  
C) A B C  
D) Compilation fails at line n1 and line n2  
*/
```

```
// Given:  
class X {  
    static int i;  
    int j;  
  
    public static void main(String[] args) {  
        X x1 = new X();  
        X x2 = new X();  
        x1.i = 3;  
        x1.j = 4;  
        x2.i = 5;  
        x2.j = 6;  
        System.out.println(  
            x1.i + " " +  
            x1.j + " " +  
            x2.i + " " +  
            x2.j);  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) 3 4 5 6  
B) 3 4 3 6  
C) 5 4 5 6  
D) 3 6 4 6  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        /* INSERT CODE HERE */  
        array[0] = 10;  
        array[1] = 20;  
        System.out.print(array[0] + ":" + array[1]);  
    }  
}  
  
/* Which code fragment, when inserted at line 5, enables  
the code to print 10:20 ?  
  
A) int[] array n= new int[2];  
B) int[] array;  
   array = int[2];  
C) int array = new int[2];  
D) int array [2];  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String[] arr = {"A", "B", "C", "D"};  
        for(int i = 0; i < arr.length; i++) {  
            System.out.print(arr[i] + " ");  
            if(arr[i].equals("C")) {  
                continue;  
            }  
            System.out.println("Word done");  
            break;  
        }  
    }  
}  
  
/*  
 A) A B C Word done  
 B) A B C D Word done  
 C) A Word done  
 D) Compilations fails  
 */
```

/*

Which three are advantages of the Java exception mechanism ?

- A) Improves the program structure because the error handling code is separated from the normal program function
- B) Provides a set of standard exceptions that covers all the possible errors
- C) Improves the program structure because the programmer can choose where to handle exceptions
- D) Improves the program structure because exceptions must be handled in the method in which they occurred
- E) Allows the creation of new exceptions that are tailored to the particular program being created

Reference: <http://javajee.com/introduction-to-exceptions-in-java>

*/

```
// Given the code from the Greeting.java file:  
public class Greeting {  
    public static void main(String[] args) {  
        System.out.println("Hello " + args[0]);  
    }  
}  
  
/* Which set of commands prints Hello Duke in the console ?  
  
A) javac Greeting  
|   java Greeting Duke  
  
B) javac Greeting.java Duke  
|   java Greeting  
  
C) javac Greeting.java  
|   java Greeting Duke  
  
D) javac Greeting,java  
|   java Greeting.class Duke  
*/
```

```
// Given:  
class Alpha {  
    int ns;  
    static int s;  
  
    Alpha(int ns) {  
        if(s < ns) {  
            s = ns;  
            this.ns = ns;  
        }  
    }  
  
    void doPrint() {  
        System.out.println("ns = " + ns + " s = " + s);  
    }  
}
```

```
public class TestA {  
    public static void main(String[] args) {  
        Alpha ref1 = new Alpha(50);  
        Alpha ref2 = new Alpha(125);  
        Alpha ref3 = new Alpha(100);  
        ref1.doPrint();  
        ref2.doPrint();  
        ref3.doPrint();  
    }  
}
```

/* WHAT IS THE RESULT ?

- A) ns = 50 s = 125
ns = 125 s = 125
ns = 100 s = 125

- B) ns = 50 s = 125
ns = 125 s = 125
ns = 0 s = 125

- C) ns = 50 s = 50
ns = 125 s = 125
ns = 100 s = 100

- D) ns = 50 s = 50
ns = 125 s = 125
ns = 0 s = 125

*/

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int ii = 0;  
        int jj = 7;  
        for(ii = 0; ii < jj - 1; ii = ii + 2) {  
            System.out.print(ii + " ");  
        }  
    }  
}  
  
/* WHAT IS THE RESULT ?  
  
A) 2 4  
B) 0 2 4 6  
C) 0 2 4  
D) compilation fails  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        LocalDate date1 = LocalDate.now();  
        LocalDate date2 = LocalDate.of(2014, 6, 20);  
        LocalDate date3 = LocalDate.parse("2014-06-20", DateTimeFormatter.ISO_DATE);  
        System.out.println("date1 = " + date1);  
        System.out.println("date2 = " + date2);  
        System.out.println("date3 = " + date3);  
    }  
}  
  
/* Assume that the system date is June 20, 2014. What is the result ?  
  
A) date1 = 2014-06-20  
    date2 = 2014-06-20  
    date3 = 2014-06-20  
  
B) date1 = 06/20/2014  
    date2 = 2014-06-20  
    date3 = Jun 20, 2014  
  
C) compilation fails  
  
D) A DateParseException is thrown at runtime  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        StringBuilder sb1 = new StringBuilder("Duke");  
        String str1 = sb1.toString();  
        // INSERT CODE HERE  
        System.out.print(str1 == str2);  
    }  
}  
  
/* Which code fragment, when inserted at line 7, enables  
the code to print true ?  
  
A) String str2 = str1;  
B) String str2 = new String(str1);  
C) String str2 = sb1.toString();  
D) String str2 = "Duke";  
*/
```

```
// Given the code fragment:  
class Test {  
    static int count = 0;  
    int i = 0;  
  
    public void changeCount() {  
        while( i < 5) {  
            i++;  
            count++;  
        }  
    }  
  
    public static void main(String[] args) {  
        Test check1 = new Test();  
        Test check2 = new Test();  
        check1.changeCount();  
        check2.changeCount();  
        System.out.print(check1.count + " : " + check2.count);  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) 10 : 10  
B) 5 : 5  
C) 5 : 10  
D) compilation fails  
*/
```

```

// Given the code fragment:
class Test {
    public static void main(String[] args) {
        double discount = 0;
        int qty = Integer.parseInt(args[0]);
        // line n1
    }
}

/* And given the requirements:

if the value of the qty variable is greater than or equal to 90,
discount = 0.5

if the value of the qty variable is between 80 and 90,
discount = 0.2

Which two code fragments can be independently placed at line n1
to meet the requirements ? */

```

```

// A
if(qty >= 90) { discount = 0.5; }
if(qty > 80 && qty < 90) { discount = 0.2; }

// B
discount = (qty >= 90) ? 0.5 : 0;
discount = (qty > 80) ? 0.2 : 0;

// C
discount = (qty >= 90) ? 0.5 : (qty > 80) ? 0.2 : 0;

// D
if(qty > 80 && qty < 90) discount = 0.2;
else discount = 0;

if(qty >= 90) discount = 0.5;
else discount = 0;

// E
discount = (qty > 80) ? 0.2 : (qty >= 90) ? 0.5 : 0;

```

```
// Given:  
class Test {  
    public static void main(String[] args) {  
        if(args[0].equals("Hello") ? false : true) {  
            System.out.println("Success");  
        } else {  
            System.out.println("Failure");  
        }  
    }  
}  
  
/* And Given the commands:  
  
javac Test.java  
java Test Hello  
  
WHAT IS THE RESULT  
  
A) Success  
B) Failure  
C) Compilation fails  
D) An exception is thrown at runtime  
*/
```

Which three statements describe the object-oriented features of the Java language?

A.

Objects cannot be reused.

B.

A subclass can inherit from a superclass.

C.

Objects can share behaviors with other objects.

D.

A package must contain more than one class.

E.

Object is the root class of all other objects.

F.

A main method must be declared in every class.

```
// Given the following code:  
class Test {  
    public static void main(String[] args) {  
        String[] planets = {"Mercury", "Venus", "Earth", "Mars"};  
  
        System.out.println(planets.length);  
        System.out.println(planets[1].length());  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) 4 and 4  
B) 3 and 5  
C) 4 and 7  
D) 5 and 4  
E) 4 and 5  
F) 4 and 21  
*/
```

```

// You are developing a banking module. You have developed a
// class named ccMask that has a maskcc method.
// GIVEN THE CODE FRAGMENT:

class CCmask {
    public static String maskCC(String creditCard) {
        String x = "xxxx-xxxx-xxxx-";
        // line n1
    }

    public static void main(String[] args) {
        System.out.println(maskCC("1234-5678-9101-1121"));
    }
}

// You must ensure that the maskcc method returns a string that hides all
// digits of the credit card number except the four last digits (and the
// hyphens that separate each group of four digits).

// Which two code fragments should you use at line n1, independently,
// to achieve this requirement ?

```

```

// A
StringBuilder sb = new StringBuilder(creditCard);
sb.substring(15, 19);
return x + sb;

// B
return x + creditCard.substring(15, 19);

// C
StringBuilder sb = new StringBuilder(x);
sb.append(creditCard, 15, 19);
return sb.toString();

// D
StringBuilder sb = new StringBuilder(creditCard);
StringBuilder s = sb.insert(0, x);
return s.toString();

```

```
// Given:  
// Acc.java  
package p1;  
public class Acc {  
    int p;  
    private int q;  
    protected int r;  
    public int s;  
}  
  
// Test.java  
package p2;  
import p1.Acc;  
public class Test extends Acc {  
    public static void main(String[] args) {  
        Acc obj = new Test();  
    }  
}  
  
/* WHICH STATEMENT IS TRUE ?  
A) Both p and s are accessible by obj.  
B) Only s is accessible by obj.  
C) Both r and s are accessible by obj.  
D) p, r, and s are accessible by obj.  
*/
```

```
// Base.java
class Base {
    public void test() {
        System.out.println("Base ");
    }
}

// DerivedA.java
class DerivedA extends Base {
    public void test() {
        System.out.println("DerivedA ");
    }
}
```

```
// DerivedB.java
class DerivedB extends DerivedA {
    public void test() {
        System.out.println("DerivedB ");
    }

    public static void main(String[] args) {
        Base b1 = new DerivedB();
        Base b2 = new DerivedA();
        Base b3 = new DerivedB();
        b1 = (Base) b3;
        Base b4 = (DerivedA) b3;
        b1.test();
        b4.test();
    }
}

/*
WHAT IS THE RESULT ?
A) Base and DerivedA
B) Base and DerivedB
C) DerivedB and DerivedB
D) DerivedB and DerivedA
E) A classCastException is thrown at runtime
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        ArrayList myList = new ArrayList();  
        String[] myArray;  
  
        try {  
            while(true) {  
                myList.add("My String");  
            }  
        }  
        catch(RuntimeException re) {  
            System.out.println("Caught a RuntimeException");  
        }  
        catch(Exception e) {  
            System.out.println("Caught an Exception");  
        }  
  
        System.out.println("Ready to use");  
    }  
}
```

- /*
- A) Execution terminates in the first catch statement, and caught a RuntimeException is printed to the console.
 - B) Execution terminates In the second catch statement, and caught an Exception is printed to the console.
 - C) A runtime error is thrown in the thread "main"
 - D) Execution completes normally, and Ready to use is printed to the console.
 - E) The code fails to compile because a throws keyword is required
- */

```
// Given:  
class Test {  
    public static void main(String[] args) {  
        System.out.println("5 + 2 = " + 3 + 4);  
        System.out.println("5 + 2 = " + (3 + 4));  
    }  
}  
  
/* WHAT IS THE RESULT ?  
  
A) 5 + 2 = 34  
    5 + 2 = 34  
  
B) 5 + 2 + 3 + 4  
    5 + 2 = 7  
  
C) 7 = 7  
    7 + 7  
  
D) 5 + 2 = 34  
    5 + 2 = 7  
*/
```

```

// Person.java
public class Person {
    String name;
    int age;

    public Person(String n, int a) {
        name = n;
        age = a;
    }

    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}

// Test.class
import java.util.*;
import java.util.function.*;

class Test {
    public static void checkAge(List<Person> list, Predicate<Person> pred) {
        for(Person p : list) {
            if(pred.test(p)) {
                System.out.println(p.name + " ");
            }
        }
    }

    public static void main(String[] args) {
        List<Person> iList = Arrays.asList(new Person("Hank", 45),
                                            new Person("Charlie", 40),
                                            new Person("Smith", 38));

        // Line n1
    }
}

/* Which code fragment, when inserted at line n1, enables the code
to print Hank ?
A) checkAge(iList, () -> p.getAge() > 40);
B) checkAge(iList, Person p -> p.getAge() > 40);
C) checkAge(iList, p -> p.getAge() > 40);
D) checkAge(iList, (Person p) -> {p.getAge() > 40;});
*/

```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String[][] arr = {{"A", "B", "C"}, {"D", "E"}};  
        for(int i = 0; i < arr.length; i++) {  
            for(int j = 0; j < arr[i].length; j++) {  
                System.out.print(arr[i][j] + " ");  
                if(arr[i][j].equals("B")) {  
                    break;  
                }  
            }  
            continue;  
        }  
    }  
  
/* WHAT IS THE RESULT ?  
 A) A B C  
 B) A B C D E  
 C) A B D E  
 D) compilations fails  
 */
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String str = " ";  
        str.trim();  
        System.out.println(str.equals("") + " " + str.isEmpty());  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) true true  
B) true false  
C) false false  
D) false true  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String str1 = "Java";  
        String str2 = new String("java");  
        // Line n1  
        {  
            System.out.println("Equal");  
        } else {  
            System.out.println("Not Equal");  
        }  
    }  
}  
  
/* which code fragment, when inserted at line n1, enables  
the App class to print Equal ?  
  
A) String str3 = str2;  
   if(str1 == str3)  
  
B) if(str1.equalsIgnoreCase(str2));  
  
C) String str3 = str2;  
   if(str1.equals(str3))  
  
D) if(str1.toLowerCase() == str2.toLowerCase())  
*/
```

```
// Given:  
class Test {  
    public static void doSum(Integer x, Integer y) {  
        System.out.println("Integer sum is " + (x + y));  
    }  
  
    public static void doSum(double x, double y) {  
        System.out.println("double sum is " + (x + y));  
    }  
  
    public static void doSum(float x, float y) {  
        System.out.println("float sum is " + (x + y));  
    }  
  
    public static void doSum(int x, int y) {  
        System.out.println("int sum is " + (x + y));  
    }  
  
    public static void main(String[] args) {  
        doSum(10, 20);  
        doSum(10.0, 20.0);  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) int sum is 30 and float sum is 30.0  
B) int sum is 30 and double sum is 30.0  
C) Integer sum is 30 and double sum is 30.0  
D) Integer sum is 30 and float sum is 30.0  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        String[] strs = new String[2];  
        int idx = 0;  
  
        for(String s : strs) {  
            strs[idx].concat(" element " + idx);  
            idx++;  
        }  
  
        for(idx = 0; idx < strs.length; idx++) {  
            System.out.println(strs[idx]);  
        }  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) element 0  
   element 1  
  
B) Null element 0  
   Null element 1  
  
C) Null  
   Null  
  
D) A NullPointerException is thrown at runtime.  
*/
```

```
// Given:  
class Vehicle {  
    int x;  
  
    Vehicle() {  
        this(10); // Line n1  
    }  
  
    Vehicle(int x) {  
        this.x = x;  
    }  
}  
  
class Car extends Vehicle {  
    int y;  
  
    Car() {  
        super();  
        this(20); // Line n2  
    }  
  
    Car(int y) {  
        this.y = y;  
    }  
  
    public String toString() {  
        return super.x + ":" + this.y;  
    }  
}
```

```
class Test {  
    public static void main(String[] args) {  
        Vehicle y = new Car();  
        System.out.println(y);  
    }  
}
```

```
/* WHAT IS THE RESULT ?  
A) 10:20  
B) 0:20  
C) Compilation fails at line n1  
D) Compilation fails at line n2  
*/
```

```
// Given the definition of MyString class and the Test class:  
package p1;  
class MyString {  
    String msg;  
    MyString(String msg) {  
        this.msg = msg;  
    }  
}  
  
package p1;  
public class Test {  
    public static void main(String[] args) {  
        System.out.println("Hello " + new StringBuilder("Java SE 8"));  
        System.out.println("Hello " + new MyString("Java SE 8"));  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) Hello Java SE 8  
   Hello Java SE 8  
  
B) Hello java.lang.StringBuilder@<<hashcode1>>  
   Hello p1.MyString@<<hashcode2>>  
  
C) Hello Java SE 8  
   Hello p1.MyString@<<hashcode2>>  
  
D) Compilation fails at the Test class  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int iVar = 100;  
        float fVar = 100.100f;  
        double dVar = 123;  
  
        iVar = fVar;  
        fVar = iVar;  
        dVar = fVar;  
        fVar = dVar;  
        dVar = iVar;  
        iVar = dVar;  
    }  
}
```

/* WHAT IS THE RESULT ?

- A) LINE 9
- B) LINE 10
- C) LINE 11
- D) LINE 12
- E) LINE 13
- F) LINE 14

*/

```
// Given:  
public class MainTest {  
    public static void main(int[] args) {  
        System.out.println("int main " + args[0]);  
    }  
  
    public static void main(Object[] args) {  
        System.out.println("Object main " + args[0]);  
    }  
  
    public static void main(String[] args) {  
        System.out.println("String main " + args[0]);  
    }  
}  
  
// and commands:  
// javac MainTest.java  
// java MainTest 1 2 3  
  
/* WHAT IS THE RESULT ?  
A) int main 1  
B) Object main 1  
C) String main 1  
D) Compilation fails  
E) An Exception is thrown at runtime  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int num[][] = new int[1][3];  
  
        for(int i = 0; i < num.length; i++) {  
            for(int j = 0; j < num[i].length; j++) {  
                num[i][j] = 10;  
            }  
        }  
    }  
}
```

/* Which option represents the state of the num array after successful completion of the outer loop ?

- A) num[0][0] = 10
 num[0][1] = 10
 num[0][2] = 10
- B) num[0][0] = 10
 num[1][0] = 10
 num[2][0] = 10
- C) num[0][0] = 10
 num[0][1] = 0
 num[0][2] = 0
- A) num[0][0] = 10
 num[0][1] = 10
 num[0][2] = 10
 num[0][3] = 10
 num[1][0] = 0
 num[1][1] = 0
 num[1][2] = 0
 num[1][3] = 0

*/

```
// Given the code fragment:  
class Person {  
    String name;  
    int age = 25;  
  
    public Person(String name) {  
        this(); // Line n1  
        this.name = name;  
    }  
  
    public Person(String name, int age) {  
        Person(name); // Line n2  
        setAge(age);  
    }  
  
    // Getter and Setter methods does here  
  
    public String show() {  
        return name + " " + age + " " + number;  
    }  
  
    public static void main(String[] args) {  
        Person p1 = new Person("Jesse");  
        Person p2 = new Person("Walter", 52);  
        System.out.println(p1.show());  
        System.out.println(p2.show());  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) Jesse 25 and Walter 52  
B) compilation fails only at line n1  
C) compilation fails only at line n2  
D) compilation fails at both line n1 and line n2  
*/
```

You are asked to develop a program for a shopping application, and you are given the following information:

The application must contain the classes Toy, EduToy, and consToy. The Toy class is the superclass of the other two classes.

The int calculatePrice (Toy t) method calculates the price of a toy.

The void printToy (Toy t) method prints the details of a toy.

Which definition of the Toy class adds a valid layer of abstraction to the class hierarchy?

- A)

```
public abstract class Toy{  
    public abstract int calculatePrice(Toy t);  
    public void printToy(Toy t) { /* code goes here */ }  
}
```
- B)

```
public abstract class Toy {  
    public int calculatePrice(Toy t) ;  
    public void printToy(Toy t) {  
}
```
- C)

```
public abstract class Toy {  
    public int calculatePrice(Toy t);  
    public final void printToy(Toy t){ /* code goes here */ }  
}
```
- D)

```
public abstract class Toy {  
    public abstract int calculatePrice(Toy t) { /* code goes here */ }  
    public abstract void printToy(Toy t) { /* code goes here */ }  
}
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int[] intArr = {15, 30, 45, 60, 75};  
        intArr[2] = intArr[4];  
        intArr[4] = 90;  
  
        for(int i : intArr) {  
            System.out.print(intArr);  
        }  
    }  
  
    /* WHAT IS THE RESULT ?  
     * A) 15, 60, 45, 90, 75  
     * B) 15, 90, 45, 90, 75  
     * C) 15, 30, 75, 60, 90  
     * D) 15, 30, 90, 60, 90  
     * E) 15, 4, 45, 60, 90  
    */
```

```
// Given the following array:  
class Test {  
    public static void main(String[] args) {  
        int[] intArr = {8, 16, 32, 64, 128};  
    }  
}  
  
/* Which two code fragments, independently, print each element in this array?  
  
A)  
    for(int i : intArr) {  
        System.out.print(intArr[i] + " ");  
    }  
  
B)  
    for(int i : intArr) {  
        System.out.print(i + " ");  
    }  
  
C)  
    for(int i = 0 : intArr) {  
        System.out.print(intArr[i] + " ");  
        i++;  
    }  
  
D)  
    for(int i = 0; i < intArr.length; i++) {  
        System.out.print(i + " ");  
    }  
  
E)  
    for(int i = 0; i < intArr.length; i++) {  
        System.out.print(intArr[i] + " ");  
    }  
  
F)  
    for(int i; i < intArr.length; i++) {  
        System.out.print(intArr[i] + " ");  
        i++;  
    }  
*/
```

```
// Given the content of three files:  
  
// A.java  
public class A {  
    public void a() {}  
    int a;  
}  
  
// B.java  
public class B {  
    private int doStuff() {  
        private int x = 100;  
        return x++;  
    }  
}  
  
// C.java  
import java.io.*;  
package p1;  
public class C {  
    public void main(String fileName) throws IOException { }  
}  
  
/* WHICH STATEMENT IS TRUE ?  
A) Only the A.java file compiles successfully.  
B) Only the B.java file compiles successfully.  
C) Only the C.java file compiles successfully.  
D) The A.java and B.java files compile successfully.  
E) The B.java and C.java files compile successfully.  
F) The A.java and C.java files compile successfully.  
*/
```

Given the code fragment:

```
int[] array = {1, 2, 3, 4, 5};
```

And given the requirements:

Process all the elements of the array in the order of entry.

Process all the elements of the array in the reverse order of entry.

Process alternating elements of the array in the order of entry.

Which two statements are true?

A.

Requirements 1, 2, and 3 can be implemented by using the enhanced for loop.

B.

Requirements 1, 2, and 3 can be implemented by using the standard for loop.

C.

Requirements 2 and 3 CANNOT be implemented by using the standard for loop.

D.

Requirement 1 can be implemented by using the enhanced for loop.

E.

Requirement 3 CANNOT be implemented by using either the enhanced for loop or the standard for loop.

```
// Given:  
class TestScope {  
    public static void main(String[] args) {  
        int var1 = 200;  
        System.out.print( doCalc(var1) );  
        System.out.print(" " + var1);  
    }  
  
    static int doCalc(int var1) {  
        var1 = var1 * 2;  
        return var1;  
    }  
}  
  
/* WHAT IS THE RESULT ?  
 A) 400 200  
 B) 200 200  
 C) 400 400  
 D) compilation fails  
*/
```

Given the following class declarations:

```
public abstract class Animal
```

```
public interface Hunter
```

```
public class Cat extends Animal implements Hunter
```

```
public class Tiger extends Cat
```

Which answer fails to compile?

- A) `ArrayList<Animal> myList = new ArrayList<>();
myList.add(new Tiger());`
- B) `ArrayList<Hunter> myList = new ArrayList<>();
myList.add(new Cat());`
- C) `ArrayList<Hunter> myList = new ArrayList<>();
myList.add(new Tiger());`
- D) `ArrayList<Tiger> myList = new ArrayList<>();
myList.add(new Cat());`
- E) `ArrayList<Animal> myList = new ArrayList<>();
myList.add(new Cat());`

Which statement is true about Java byte code?

- A.**
It can run on any platform.
- B.**
It can run on any platform only if it was compiled for that platform.
- C.**
It can run on any platform that has the Java Runtime Environment.
- D.**
It can run on any platform that has a Java compiler.
- E.**
It can run on any platform only if that platform has both the Java Runtime Environment and a Java compiler.

Reference: <http://www.math.uni-hamburg.de/doc/java/tutorial/getStarted/intro/definition.html>

```
// Given:  
class MarkList {  
    public static void main(String[] args) {  
        int num;  
  
        public static void graceMarks(MarkList obj4) {  
            obj4.num += 10;  
        }  
  
        public static void main(String[] args) {  
            MarkList obj1 = new MarkList();  
            MarkList obj2 = obj1;  
            MarkList obj3 = null;  
  
            obj2.num = 60;  
            graceMarks(obj2);  
        }  
    }  
}  
  
/* How many MarkList instances are created in memory at runtime ?  
A) 1  
B) 2  
C) 3  
D) 4  
*/
```

```
// Given:  
class Triangle {  
    static double area;  
    int b = 2, h = 3;  
  
    public static void main(String[] args) {  
        double p, b, h; // Line n1  
  
        if(area == 0) {  
            b = 3;  
            h = 4;  
            p = 0.5;  
        }  
  
        area = p * b * h; // Line n2  
        System.out.println("Area is " + area);  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) Area is 6.0  
B) Area is 3.0  
C) Compilation fails at line n1  
D) Compilation fails at line n2.  
*/
```

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        // Line n1  
  
        switch(x) {  
            case 1:  
                System.out.println("One");  
                break;  
            case 2:  
                System.out.println("Two");  
                break;  
        }  
    }  
  
/* Which three code fragments can be independently inserted  
at line n1 to enable the code to print one ?  
  
A) Byte x = 1;  
B) short x = 1;  
C) String x = "1";  
D) Long x = 1;  
E) Double x = 1;  
F) Integer x = new Integer("1");  
*/
```

```
// Given:  
class App {  
    public static void main(String[] args) {  
        Boolean[] bool = new Boolean[2];  
  
        bool[0] = new Boolean(Boolean.parseBoolean("true"));  
        bool[1] = new Boolean(null);  
  
        System.out.println(bool[0] + " " + bool[1]);  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) True false  
B) True null  
C) compilation fails  
D) A NullPointerException is thrown at runtime.  
*/
```

```
// Given the following code for the classes MyException and Test:  
class MyException extends RuntimeException { }  
  
class Test {  
    public static void main(String[] args) {  
        try {  
            method1();  
        }  
        catch (MyException ne) {  
            System.out.print("A");  
        }  
    }  
  
    public static void method1() { // Line n1  
        try {  
            throw Math.random() > 0.5 ? new MyException() : new RuntimeException();  
        }  
        catch (RuntimeException re) {  
            System.out.print("B");  
        }  
    }  
}  
  
/* WHAT IS THE RESULT ?  
 A) A  
 B) B  
 C) Either A or B  
 D) A B  
 E) A compile time error occurs at line n1  
 */
```

```
// Given:  
class App {  
    String myStr = "7007";  
  
    public void doStuff(String str) {  
        int myNum = 0;  
        try {  
            String myStr = str;  
            myNum = Integer.parseInt(myStr);  
        } catch (NumberFormatException ne) {  
            System.err.println("Error");  
        }  
  
        System.out.println("myStr: " + myStr + ", myNum: " + myNum);  
    }  
  
    public static void main(String[] args) {  
        App obj = new App();  
        obj.doStuff("9009");  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) myStr: 9009, myNum: 9009  
B) myStr: 7007, myNum: 7007  
C) myStr: 7007, myNum: 9009  
D) Compilation fails  
*/
```

Which two are benefits of polymorphism?

A.

Faster code at runtime

B.

More efficient code at runtime

C.

More dynamic code at runtime

D.

More flexible and reusable code

E.

Code that is protected from extension by other classes

```
// Given the code fragment:  
class Test {  
    public static void main(String[] args) {  
        int nums1[] = new int[3];  
        int nums2[] = {1, 2, 3, 4, 5};  
        nums1 = nums2;  
  
        for(int x : nums1) {  
            System.out.print(x + ":" );  
        }  
    }  
}  
  
/* WHAT IS THE RESULT ?  
A) 1:2:3:4:5:  
B) 1:2:3:  
C) Compilation fails  
D) An ArrayoutofBoundsException is thrown at runtime  
*/
```

```
// Given
class Product {
    int id;
    String name;

    public Product(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

// and given the code fragment:
class Test {
    public static void main(String[] args) {
        Product p1 = new Product(101, "Pen");
        Product p2 = new Product(101, "Pen");
        Product p3 = p1;

        boolean ans1 = p1 == p2;
        boolean ans2 = p1.name.equals(p2.name);
        System.out.print(ans1 + " : " + ans2);
    }
}

/* WHAT IS THE RESULT ?
   A) true : true
   B) true : false
   C) false : true
   D) false : false
*/
```

```
// Given the following classes:  
class Employee {  
    public int salary;  
}  
  
class Manager extends Employee {  
    public int budget;  
}  
  
class Director extends Manager {  
    public int stockOptions;  
}  
  
class Test {  
    public static void main(String[] args) {  
        Employee employee = new Employee();  
        Manager manager = new Manager();  
        Director director = new Director();  
  
        // Line n1  
    }  
}  
  
/* Which two options fail to compile when placed at line n1 of the main method ?  
A) employee.salary = 50_000;  
B) director.salary = 80_000;  
C) employee.budget = 200_000;  
D) manager.budget = 1_000_000;  
E) manager.stockOption = 500;  
F) director.stockOptions = 1_000;  
*/
```