PREGUNTAS JAVA.

1. What is the result?

```
Given

public static void main(String[] args){
      int[][] array2D = {{0,1,2}, {3,4,5,6}};
      System.out.print(array2D[0].length + "");
      System.out.print(array2D[1].getClass().isArray() + "");
      System.out.print(array2D[0][1]);
}
What is the result?
```

a) 3false3
b) 3false1
c) 2false1
d) 3true1
e) 2true3

Respuesta:

En el fragmento de código observamos que se ha inicializado un arreglo multidimensional de un tamaño de dos que almacena un arreglo de 3 y otro de 4.

En la siguiente línea se manda a imprimir el tamaño de primer arreglo almacenado es decir **“3”**.

Después se manda a llamar el método getClass que es un método de instancia de objetos el cual regresa la clase, ”*Class<?>”,* en tiempo de ejecución de este objeto a este valor de retorno se puede invocar el método isArray para determinar si el objeto Class representa la clase de Array y retorna un boolean por lo que se lee en el código devolverá “***true”*.**

Por ultimo se manda imprimir el segundo dato almacenado en el primer arreglo, es decir **“1”.**

Por lo que la respuesta es: **3true1**

2. Which two statments are true?

a) An interface CANNOT be extended by another interface.
b) An abstract class CANNOT be extended by an abstract class.
c) An interface can be extended by an abstract class.
d) An abstract class can implement an interface.
e) An abstract class can be extended by an interface.
f) An abstract class can be extended by a concrete class.

Respuesta:

La diferencia entre las clases abstractas y las interfaces

Una interfaz si puede ser extendida por otra interfaz, así que el primer enunciado es falso.

Una clase abstracta si puede se extendida por otra clase abstracta, por lo que el segundo enunciado es falso.

En el tercer enunciado es falso, porque una interfaz no puede ser extendida por una clase abstracta

La opción d) es verdadera, una clase abstracta puede implementar una interfaz.

La opción e) es falsa, una clase abstracta no pude ser extendida por una interfaz, las interfaces sólo extienden de otras interfaces.

La opción f) es verdadera una clase abstracta puede ser extendida por una clase concreta.

Por lo que las respuestas son: **d) y f).**

3. What is the result?

```
Given:

class Alpha{ String getType(){ return "alpha";}}
class Beta extends Alpha{String getType(){ return "beta";}}
public class Gamma  extends Beta { String getType(){ return "gamma";}
      public static void main(String[] args) {
        Gamma g1 = (Gamma) new Alpha();
        Gamma g2 = (Gamma) new Beta();
        System.out.print(g1.getType()+ " " +g2.getType());
    }
}
What is the result?
```

a) Gamma gamma
b) Beta beta
c) Alpha beta
d) Compilation fails

Respuesta:

La opción d) Compilation fails, debido a que los objetos creados nos son gamma.

4. **Which five methods, inserted independently at line 5, will compile? (Choose five)**

```
1 public class Blip{
2       protected int blipvert(int x){ return 0
3 }
4 class Vert extends Blip{
5       //insert code here
6 }
```

a) Private int blipvert(long x) { return 0; }
b) Protected int blipvert(long x) { return 0; }
c) Protected long blipvert(int x, int y) { return 0; }
d) Public int blipvert(int x) { return 0; }
e) Private int blipvert(int x) { return 0; }
f) Protected long blipvert(int x) { return 0; }
g) Protected long blipvert(long x) { return 0; }

Respuesta:

a), b), c), d), g)

La opción e) esta descartada debido a que no se puede reducir el acceso del método pasaría de protected a private.

La opción f) esta opción no podría recibir un int y devolver un long, necesitaría de otro int o un long.

5. Which two independently, will allow Sub to compile? (Choose two)

```
Given:

1. class Super{
2.    private int a;
3.    protected Super(int a){ this.a = a; }
4. }
...
11. class Sub extends Super{
12.   public Sub(int a){ super(a);}
13.   public Sub(){ this.a = 5;}
14. }
Which two independently, will allow Sub to compile? (Choose two)
```

a) Change line 2 to: public int a;
b) Change line 13 to: public Sub(){ super(5);}
c) Change line 2 to: protected int a;
d) Change line 13 to: public Sub(){ this(5);}
e) Change line 13 to: public Sub(){ super(a);}

Respuesta:

b) Change line 13 to: public Sub(){ super(5);}

d) Change line 13 to: public Sub(){ this(5);}

Estas opciones permiten invocar el constructor de la superclase sin afectar a la clase padre.

6. **What is true about the class Wow?**

```
public abstract class Wow {
        private int wow;
        public Wow(int wow) { this.wow = wow; }
        public void wow() { }
        private void wowza() { }
}
```

a) It compiles without error.
b) It does not compile because an abstract class cannot have private methods
c) It does not compile because an abstract class cannot have instance variables.
d) It does not compile because an abstract class must have at least one abstract method. It does not compile because an abstract class must have a constructor with no arguments.

Respuesta:

La opción a) es correcta debido a que una clase abstracta puede tener métodos concretos privados, más no métodos abstractos privados.

7. **What is the result?**

```
class Atom {
        Atom() { System.out.print("atom "); }
}
class Rock extends Atom {
        Rock(String type) { System.out.print(type); }
}
public class Mountain extends Rock {
        Mountain() {
                super("granite ");
                new Rock("granite ");
        }
        public static void main(String[] a)  { new Mountain(); }
}
```

a) Compilation fails.
b) Atom granite.
c) Granite granite.
d) Atom granite granite.
e) An exception is thrown at runtime.
f) Atom granite atom granite

Respuesta:

La opción f) es correcta, porque se hace una llamada implícita al constructor padre.

8. **What is printed out when the program is excuted?**

```
public class MainMethod {
        void main() {
                System.out.println("one");
        }
        static void main(String args) {
                System.out.println("two");
        }
        public static final void main(String[] args) {
                System.out.println("three");
        }
        void mina(Object[] args) {
                System.out.println("four");
        }
}
```

a) one
b) two
c) three
d) four
e) There is no output.

Respuesta:

La opción c) es correcta la palabra final no afecta su funcionamiento como punto de entrada.

9. What is the result?

```
class Feline {
        public String type = "f ";
        public Feline() {
                System.out.print("feline ");
        }
}
public class Cougar extends Feline {
        public Cougar() {
                System.out.print("cougar ");
        }
        void go() {
                type = "c ";
                System.out.print(this.type + super.type);
        }
        public static void main(String[] args) {
                new Cougar().go();
        }
}
```

a) Cougar c f.
b) Feline cougar c c.
c) Feline cougar c f.
d) Compilation fails.

Respuesta correcta

La opción b) Feline cougar c c, porque primero se ejecuta el constructor de la superclase, en seguida el contructor de la clase y en el método go sucede que al atributo type se le asigna un valor en este caso como es un atributo heredado y no hay sobreescritura de atributos, ambos se refieren al mismo atributo, por esa razón ambos contienen “c”.

10. What is the result?

```
class Alpha { String getType() { return "alpha"; } }
class Beta extends Alpha { String getType() { return "beta"; } }
public class Gamma extends Beta { String getType() { return "gamma"; }
        public static void main(String[] args) {
                Gamma g1 = new Alpha();
                Gamma g2 = new Beta();
                System.out.println(g1.getType() + " " + g2.getType());
        }
}
```

a) Alpha beta
b) Beta beta.
c) Gamma gamma.
d) Compilation fails.

Respuesta:

La opción d) es correcta ya que debería tener un casting, pero aún con el cast no es posible que un objeto inicializado como Alpha se castee a gamma, el caso posible es cuando se crea una variable Alpha que contenga una referencia a un objeto Gamma y después se hace un cast para guardar e una variable de tipo Gamma:

Alpha a1 = new Gamma();

Gamma g2 = (Gamma) a1;

11. What is the result?

```
import java.util.*;
public class MyScan {
        public static void main(String[] args) {
                String in = "1 a 10 . 100 1000";
                Scanner s = new Scanner(in);
                int accum = 0;
                for (int x = 0; x < 4; x++) {
                        accum += s.nextInt();
                }
                System.out.println(accum);
        }
}
```

a) 11
b) 111
c) 1111
d) An exception is thrown at runtime.

Respuesta:

La opción correcta es la d), debido a que recibiremos una *"InputMismatchException",* lo que quiere decir que el argumento in no coincide con el patrón del tipo esperado o que el token está fuera del rango del tipo esperado. Para que funcione debe recibir un InputStream, que en este caso estaría dado por ***"Sytem.in"***, así funcionaría.

12. What is the result?

```
public class Bees {
        public static void main(String[] args) {
                try {
                        new Bees().go();
                } catch (Exception e) {
                        System.out.println("thrown to main");
                }
        }
        synchronized void go() throws InterruptedException {
                Thread t1 = new Thread();
                t1.start();
                System.out.print("1 ");
                t1.wait(5000);
                System.out.print("2 ");
        }
}
```

a) The program prints 1 then 2 after 5 seconds.
b) The program prints: 1 thrown to main.
c) The program prints: 1 2 thrown to main.
d) The program prints:1 then t1 waits for its notification.

Respuesta:

La opción b), es correcta.

13. Which statement is true?

```
class ClassA {
        public int numberOfInstances;
        protected ClassA(int numberOfInstances) {
                this.numberOfInstances = numberOfInstances;
        }
}
public class ExtendedA extends ClassA {
        private ExtendedA(int numberOfInstances) {
                super(numberOfInstances);
        }
        public static void main(String[] args) {
                ExtendedA ext = new ExtendedA(420);
                System.out.print(ext.numberOfInstances);
        }
}
```

a) 420 is the output.
b) An exception is thrown at runtime.
c) All constructors must be declared public.
d) Constructors CANNOT use the private modifier.
e) Constructors CANNOT use the protected modifier.

Respuesta:

La opción a) es correcta, el código compila y corre sin errores.

14. The SINGLETON pattern allows: *

a) Have a single instance of a class and this instance cannot be used by other classes
b) Having a single instance of a class, while allowing all classes have access to that instance.
c) Having a single instance of a class that can only be accessed by the first method that calls it.

15. What is the result?

```
import java.text.*;
public class Align {
        public static void main(String[] args) throws ParseException {
                String[] sa = {"111.234", "222.5678"};
                NumberFormat nf = NumberFormat.getInstance();
                nf.setMaximumFractionDigits(3);
                for (String s : sa) { System.out.println(nf.parse(s)); }
        }
}
```

a) 111.234 222.567
b) 111.234 222.568
c) 111.234 222.5678
d) An exception is thrown at runtime.

Respuesta:

El efecto de setMaximumFractionDigits() no tiene efecto con el método parse(), por lo que los dígitos no se ven afectados, si se usará el método format() si aplicaría.

16. What is the result?

Given

```
public class SuperTest {
        public static void main(String[] args) {
                //statement1
                //statement2
                //statement3
        }
}
class Shape {
        public Shape() {
                System.out.println("Shape: constructor");
        }
        public void foo() {
                System.out.println("Shape: foo");
        }
}
class Square extends Shape {
        public Square() {
                super();
        }
        public Square(String label) {
                System.out.println("Square: constructor");
         }
        public void foo() {
                super.foo();
        }
        public void foo(String label) {
                System.out.println("Square: foo");
        }
}
```

Imagen sin leyenda

What should statement1, statement2, and statement3, be respectively, in order to produce the result?

Shape: constructor
Shape: foo
Square: foo

a) Square square = new Square ("bar"); square.foo ("bar"); square.foo();
b) Square square = new Square ("bar"); square.foo ("bar"); square.foo ("bar");
c) Square square = new Square (); square.foo (); square.foo(bar);
d) Square square = new Square (); square.foo (); square.foo("bar");
e) Square square = new Square (); square.foo (); square.foo ();

Respuesta:

La opción d) es correcta porque al usar el constructor sin paramentros se obtiene Shape: constructor, al ejecutar el método foo() sin pametros se hace uso del método de la superclase, y al hacer uso del método foo(String) se imprime Square: foo.

17. Which three implementations are valid?

```
interface SampleCloseable {
        public void close() throws java.io.IOException;
}
```

a) class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }
b) class Test implements SampleCloseable { public void close() throws Exception { // do something } }
c) class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }
d) class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
e) class Test implements SampleCloseable { public void close() { // do something }}

Respuesta:

La respuesta correcta es a), porque el método debe estar escrito como en la interfaz.

18. What is the result?

```
class MyKeys {
        Integer key;
        MyKeys(Integer k) { key = k; }
        public boolean equals(Object o) {
                return ((MyKeys) o).key == this.key;
        }
}
```

And this code snippet:

```
Map m = new HashMap();
MyKeys m1 = new MyKeys(1);
MyKeys m2 = new MyKeys(2);
MyKeys m3 = new MyKeys(1);
MyKeys m4 = new MyKeys(new Integer(2));
m.put(m1, "car");
m.put(m2, "boat");
m.put(m3, "plane");
m.put(m4, "bus");
System.out.print(m.size());
```

a) 2
b) 3
c) 4
d) Compilation fails

Respuesta:

La opción c) 4, ya que devolverá el tamaño del Map al que se le han añadido 4 key-value.

19. What value of x, y, z will produce the following result? 1234,1234,1234 -----, 1234, ------ *

```
public static void main(String[] args) {
        // insert code here
        int j = 0, k = 0;
        for (int i = 0; i < x; i ++) {
                do {
                        k = 0;
                        while (k < z) {
                                k++;
                                System.out.print(k + " ");
                        }
                        System.out.println(" ");
                        j++;
                } while (j < y);
                System.out.println("---");
        }
}
```

a) int x = 4, y = 3, z = 2;
b) int x = 3, y = 2, z = 3;
c) int x = 2, y = 3, z = 3;
d) int x = 2, y = 3, z = 4;
e) int x = 4, y = 2, z= 3;

Respuesta:

La respuesta es d), porque se para imprimir los 1234 se requiere que z sea igual a 4.

20. Which three lines will compile and output "Right on!"?

```
13.  public class Speak {
14.      public static void main(String[] args) {
15.          Speak speakIT = new Tell();
16.          Tell tellIt = new Tell();
17.          speakIT.tellItLikeItIs();
18.          (Truth) speakIT.tellItLikeItIs();
19.          ((Truth) speakIT).tellItLikeItIs();
20.          tellIt.tellItLikeItIs();
21.          (Truth) tellIt.tellItLikeItIs();
22.          ((Truth) tellIt).tellItLikeItIs();
23.      }
24.  }

class Tell extends Speak implements Truth {
    @Override
    public void tellItLikeItIs() {
        System.out.println("Right on!");
    }
}

interface Truth {
    public void tellItLikeItIs();
}
```

a) Line 17
b) Line 18
c) Line 19
d) Line 20
e) Line 21
f) Line 22

Respuesta:

Las líneas que compilaran y correrán son la 19,20,22, la línea 19 porque al hacer el cast reconoce el método tellItLikeItIs(), en la línea 20 el método es parte del objeto. En la línea 22 aunque hay un cast en truth existe el método.

21. What is the result?

```
class Feline {
    public String type = "f";
    public Feline() {
        System.out.print(s: "feline ");
    }
}
public class Cougar extends Feline{
    public Cougar() {
        System.out.print(s: "cougar ");
    }
    void go(){
        String type = "c";
        System.out.print(this.type + super.type);
    }

    Run | Debug
    public static void main(String[] args) {
        new Cougar().go();
    }
}
```

a) Feline cougar c f
b) Feline cougar c c
c) Feline cougar f f
d) No compila

Respuesta:

Es b) feline se imprime porque se ejecuta el constructor que esta implícito en el constructor de la subclase, cougar se imprime por el constructor de la subclase, el método go() asigna el valor para la variable de instancia en este caso this y super.type apuntan al mismo atributo.

22. ¿Cuál es el resultado?

```
import java.util.*;
public class App {
        public static void main(String[] args) {
                List p = new ArrayList();
                p.add(7);
                p.add(1);
                p.add(5);
                p.add(1);
                p.remove(1);
                System.out.println(p);
        }
}
```

a) [7, 5]
b) [7, 1]
c) [7, 5, 1]
d) [7, 1, 5, 1]

Respuesta:

La opción c) es la correcta porque removerá el elemento en el índice 1.

23. Which five methods, inserted independiently at line 5, will compile? (Choose five)

```
1 public class Blip{
2       protected int blipvert(int x){ return 0
3 }
4 class Vert extends Blip{
5       //insert code here
6 }
```

a) Public int blipvert(int x) { return 0; }
b) Protected long blipvert(int x) { return 0; }
c) Protected int blipvert(long x) { return 0; }
d) Private int blipvert(long x) { return 0; }
e) Protected long blipvert(int x, int y) { return 0; }
f) Private int blipvert(int x) { return 0; }
g) Protected long blipvert(long x) { return 0; }

Respuesta:

Respuesta correcta

Las opciones: a) Public int blipvert(int x) { return 0; }, g) Protected long blipvert(long x) { return 0; }, d) Private int blipvert(long x) { return 0; },e) Protected long blipvert(int x, int y) { return 0; }, c) Protected int blipvert(long x) { return 0; } son correctas.

La opción f) no es posible porque no se pude reducir su acceso,

24. What is the result?

```
Given:

1. class Super{
2.     private int a;
3.     protected Super(int a){ this.a = a; }
4. }
...
11. class Sub extends Super{
12.    public Sub(int a){ super(a);}
13.    public Sub(){ this.a = 5;}
14. }
Which two independently, will allow Sub to compile? (Choose two)
```

a) Change line 2 to: public int a;
b) Change line 13 to: public Sub(){ super(5);}
c) Change line 2 to: protected int a;
d) Change line 13 to: public Sub(){ this(5);}
e) Change line 13 to: public Sub(){ super(a);}

Respuesta:

B y d

25. What is the result?

```
Given

public static void main(String[] args){
      int[][] array2D = {{0,1,2}, {3,4,5,6}};
      System.out.print(array2D[0].length + "");
      System.out.print(array2D[1].getClass().isArray() + "");
      System.out.print(array2D[0][1]);
}
What is the result?
```

a) 3false3
b) 3false1
c) 2false1
d) 3true1
e) 2true3

Respuesta:

En la primera impresión se obtiene el tamaño del primer arreglo que es 3, la segunda impresión true ya que es de una clase Array, la ultima impresión se imprime el elemento en el primer arreglo en el índice 1 que es 1, por lo que la respuesta correcta es: d)3true1,

26. Which two statments are true?

a) An interface CANNOT be extended by another interface.
b) An abstract class can be extended by a concrete class.
c) An abstract class CANNOT be extended by an abstract class.
d) An interface can be extended by an abstract class.
e) An abstract class can implement an interface.
f) An abstract class can be extended by an interface.

27. What is the result?

```
Given:

class Alpha{ String getType(){ return "alpha";}}
class Beta extends Alpha{String getType(){ return "beta";}}
public class Gamma  extends Beta { String getType(){ return "gamma";}
      public static void main(String[] args) {
        Gamma g1 = (Gamma) new Alpha();
        Gamma g2 = (Gamma) new Beta();
        System.out.print(g1.getType()+ " " +g2.getType());
    }
}
What is the result?
```

a) Gamma gamma
b) Beta beta
c) Alpha beta
d) Compilation fails

Respuesta

La opción d), no compilará debido a que no son objetos instanciados de Gamma,

28. What is the result?

```
public class Test {
        public static void main(String[] args) {
                int b = 4;
                b--;
                System.out.print(--b);
                System.out.println(b);
        }
}
```

a) 22
b) 12
c) 32
d) 33

Respuesta:

La opción a) es correcta porque en la primera impresión se aplica antes de imprimir el decremento.

29. In Java the difference between throws and throw Is:*

a) Throws throws an exception and throw indicates the type of exception that the method.
b) Throws is used in methods and throw in constructors.
c) Throws indicates the type of exception that the method does not handle and throw an exception.

30. What is the result?

```
class Feline {
        public String type = "f ";
        public Feline() {
                System.out.print("feline ");
        }
}
public class Cougar extends Feline {
        public Cougar() {
                System.out.print("cougar ");
        }
        void go() {
                type = "c ";
                System.out.print(this.type + super.type);
        }
        public static void main(String[] args) {
                new Cougar().go();
        }
}
```

a) Cougar c f.
b) Feline cougar c c.
c) Feline cougar c f.
d) Compilation fails.

Respuesta

La opción b) el constructor de la superclase se ejecuta primero y luego el de la subclase, los atributos this.type y super.type, hacen referencia a lo mismo por lo que se imprime cc.

31. Which statement, when inserted into line " // TODO code application logic here", is valid in compilation time change?*

```
public class SampleClass {
        public static void main(String[] args) {
                AnotherSampleClass asc = new AnotherSampleClass();
                SampleClass sc = new SampleClass();
                // TODO code application logic here
        }
}
class AnotherSampleClass extends SampleClass { }
```

a) asc = sc;
b) sc = asc;
c) asc = (Object) sc;
d) asc= sc.clone();

Respuesta:

La opción b es válida, porque sc puede guardar un objeto de una de sus subclases con un cast implícito.

32. What is the result?

```
public class Test {
        public static void main(String[] args) {
                int[][] array = { {0}, {0,1}, {0,2,4}, {0,3,6,9}, {0,4,8,12,16} };
                System.out.println(array[4][1]);
                System.out.println(array[1][4]);
        }
}
```

a) 4 Null.
b) Null 4.
c) An IllegalArgumentException is thrown at run time.
d) 4 An ArrayIndexOutOfBoundsException is thrown at run time.

Respuesta:

La respuesta correcta es: d) 4 An ArrayIndexOutOfBoundsException is thrown at run time, porque el arreglo en array[1] solo tiene 2 valores almacenados así que no encuentra nada en el índice 4.

33. What is the result?

```
import java.util.*;
public class App {
        public static void main(String[] args) {
                List p = new ArrayList();
                p.add(7);
                p.add(1);
                p.add(5);
                p.add(1);
                p.remove(1);
                System.out.println(p);
        }
}
```

a) [7, 1, 5, 1]
b) [7, 5, 1]
c) [7, 5]
d) [7, 1]

Respuesta:

La opción correcta e la b), porque se eliminó el elemento en el índice 1.

34. Which three lines will compile and output "Right on!"?*

```
13.  public class Speak {
14.      public static void main(String[] args) {
15.          Speak speakIT = new Tell();
16.          Tell tellIt = new Tell();
17.          speakIT.tellItLikeItIs();
18.          (Truth) speakIT.tellItLikeItIs();
19.          ((Truth) speakIT).tellItLikeItIs();
20.          tellIt.tellItLikeItIs();
21.          (Truth) tellIt.tellItLikeItIs();
22.          ((Truth) tellIt).tellItLikeItIs();
23.      }
24.  }

class Tell extends Speak implements Truth {
    @Override
    public void tellItLikeItIs() {
        System.out.println("Right on!");
    }
}

interface Truth {
    public void tellItLikeItIs();
}
```

a) Line 17
b) Line 18
c) Line 19
d) Line 20
e) Line 21
f) Line 22

Respuesta:

Las líneas 19,20 y 22 compilaran con la salida “Rigth on!”, ya que en la línea 19 el cast está correcto y puede invocar el método, la linea 20 el método puede ser invocado por ese objeto, la línea 22 aunque se aplique el cast el método puede ser invocado por Truth.

35. What is the result?

```
public class Bees {
        public static void main(String[] args) {
                try {
                        new Bees().go();
                } catch (Exception e) {
                        System.out.println("thrown to main");
                }
        }
        synchronized void go() throws InterruptedException {
                Thread t1 = new Thread();
                t1.start();
                System.out.print("1 ");
                t1.wait(5000);
                System.out.print("2 ");
        }
}
```

a) The program prints 1 then 2 after 5 seconds.
b) The program prints: 1 thrown to main.
c) The program prints: 1 2 thrown to main.
d) The program prints:1 then t1 waits for its notification.

36. Which three are valid? (Choose three)*

```
class ClassA {}
class ClassB extends ClassA {}
class ClassC extends ClassA {}
And:
ClassA p0 = new ClassA();
ClassB p1 = new ClassB();
ClassC p2 = new ClassC();
ClassA p3 = new ClassB();
ClassA p4 = new ClassC();
```

a) p0 = p1;
b) p1 = p2;
c) p2 = p4;
d) p2 = (ClassC)p1;
e) p1 = (ClassB)p3;
f) p2 = (ClassC)p4;

Respuesta:

Las respuestas válidas son a), e) y f), en estas opciones el cast es correcto, en las otras no puede ser posible porque las variables no son aptas para los objetos.

37. Which three options correctly describe the relationship between the classes?*

```
class Class1 { String v1; }
class Class2 {
        Class1 c1;
        String v2;
}
class Class3 { Class2 c1; String v3; }
```

a) Class2 has-a v3.
b) Class1 has-a v2.
c) Class2 has-a v2.
d) Class3 has-a v1.
e) Class2 has-a Class3.
f) Class2 has-a Class1.

Respuesta:

Las opciones c), d), y f), la Class1 tiene v1, Class2 tiene v2 y el v1 de Class1, Class3 tiene v3 y el v2 y v1 porque contiene un Class2.

38. Which three implementations are valid?*

```
interface SampleCloseable {
        public void close() throws java.io.IOException;
}
```

a) class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }
b) class Test implements SampleCloseable { public void close() throws Exception { // do something } }
c) class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }
d) class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
e) class Test implements SampleCloseable { public void close() { // do something }}

Respuesta:

Las opciones correctas son a), c) y e), las dos primeras dos opciones porque a) tiene la firma igual al método, c) tiene casi la misma solo que lanza una excepción que es subclase de “java.io.IOException”, y la opción e) podría implementar un try and catch para lanzar la excepción.

39. What is the result?

```
class MySort implements Comparator<Integer> {
        public int compare(Integer x, Integer y) {
                return y.compareTo(x);
        }
}
```

And the code fragment:

```
Integer[] primes = {2, 7, 5, 3};
MySort ms = new MySort();
Arrays.sort(primes, ms);
for (Integer p2 : primes) { System.out.print(p2 + " "); }
```

a) 2 3 5 7
b) 2 7 5 3
c) 7 5 3 2
d) Compilation fails.

Respuesta:

La opción c) es la correcta.

40. Which two possible outputs?*

```
public class Main {
        public static void main(String[] args) throws Exception {
                doSomething();
        }
        private static void doSomething() throws Exception {
                System.out.println("Before if clause");
                if (Math.random() > 0.5) { throw new Exception();}
                System.out.println("After if clause");
        }
}
```

a) Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
b) Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15) After if clause.
c) Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
d) Before if clause After if clause.

Respuesta:

Las opciones de posible salida son: b) y d), antes del if se imprime, en el if se puede lanzar la excepción y después del if se imprime, no se ve interrumpida la ejecución.