40 preguntas java

```
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?
```

3false3
3false1
2false1
3true1

2true3

La longitud del arreglo se cuenta normal por ql numero de elementos y no empezado desde 0 como su indexacion y accesibilidad .
El metodo “getClass” obtiene da acceso a la clase instanciada.

Which two statments are true?
*

1/1

An interface CANNOT be extended by another interface.
An abstract class can be extended by a concrete class.
An abstract class CANNOT be extended by an abstract class.
An interface can be extended by an abstract class.
An abstract class can implement an interface.

Las clases abstractas solo puede extender clases abstractas o implementar interfaces

1/1

```
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?
```

Gamma gamma
Beta beta
Alpha beta
Compilation fails

No se puede convertir directamente u-n objeto de una superclase a una subclase.

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

1/1

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

Private int blipvert(long x) { return 0; }

Protected int blipvert(long x) { return 0; }

Protected long blipvert(int x, int y) { return 0; }

Public int blipvert(int x) { return 0; }

Private int blipvert(int x) { return 0; }
Protected long blipvert(int x) { return 0; }
Protected long blipvert(long x) { return 0; }

*

Un método sobrescrito debe tener el mismo nombre, tipo de retorno y parámetros que el método de la clase padre, y un modificador de acceso igual o menos restrictivo.

Un método sobrecargado debe tener el mismo nombre que otro método en la misma clase, pero una firma diferente (diferente número o tipo de parámetros).

0/1

```
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)
```

Change line 2 to: public int a;

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

Change line 2 to: protected int a;

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

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

Respuesta correcta

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

`public Sub() { super(5); }`: Esta solución llama al constructor de la clase padre (`Super`) pasando el valor 5. Esto inicializa correctamente el atributo a en la clase padre, ya que el constructor de `Super` tiene acceso a a.
`public Sub() { this(5); }`: Esta solución llama al otro constructor de la clase Sub (el que toma un argumento entero). Este constructor a su vez llama al constructor de la clase padre, inicializando a correctamente.

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

It compiles without error.

It does not compile because an abstract class cannot have private methods
It does not compile because an abstract class cannot have instance variables.
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.

No infringe ninguna regla.
Las clases abstractas puedes tener me todos abstractos y concretos,
pueden tener metodos privado que solo son usados por la misma clase,
puede tener constructores

What is the result?

*

1/1

```
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(); }
}
```

Compilation fails.
Atom granite.
Granite granite.
Atom granite granite.
An exception is thrown at runtime.

Atom granite atom granite.

Se creo el constructor de Rock con “granite”.
Al mandar a llamar a new Muntain() se ejecuta los constructores desde la clase padre(Atom) que imprime “Atom” luego se ejecuta el constructor de Rock que imprime “granite”.
Y basicamente al llamar new Rock() ocurre lo mismo

What is printed out when the program is excuted?

*

1/1

```
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");
        }
}
```

one
two
three

four
There is no output.

La palabra reservada final no afecta al funcionamiento del metodo main

```
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();
        }
}
```

Cougar c f.
Feline cougar c c.
Feline cougar c f.

Compilation fails.

Respuesta correcta
Feline cougar c c.

Al llamar a “new Cougar()” primerero se ejecuta el constructor de la clase padre “Feline” que imprime Feline, despues el constructor de la clase Cougar() que imprime “cougar”.

Con this.type=”c” se le cambia el valor a la variable de instancia de la clase Feline porlo que imprime “CC”

```
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());
        }
}
```

Alpha beta
Beta beta.
Gamma gamma.
Compilation fails.

Es posible asignar un objeto de una subclase a una variable de su superclase (por ejemplo, `Alpha a = new Beta();`), no es posible hacer lo contrario directamente.

What is the result?
*
1/1

```
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);
        }
}
```

11
111

1111
An exception is thrown at runtime.

Puede recibir valores no enteros

What is the result?
*
1/1

```
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 ");
        }
}
```

The program prints 1 then 2 after 5 seconds.
The program prints: 1 thrown to main.

The program prints: 1 2 thrown to main.
The program prints:1 then t1 waits for its notification.

El método `go()` está marcado como `synchronized`. Esto significa que, para ejecutarlo, un hilo debe adquirir el bloqueo intrínseco del objeto `Bees`. Sin embargo, el hilo principal (que ejecuta `main`) no tiene este bloqueo cuando llama a `t1.wait(5000)`. Esto provoca una `IllegalMonitorStateException` en tiempo de ejecución.

```
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);
        }
}
```

420 is the output.

An exception is thrown at runtime.
All constructors must be declared public.
Constructors CANNOT use the private modifier.
Constructors CANNOT use the protected modifier.

La variable de instancia se inicializa a 420 en el constructor de ClassA.

The SINGLETON pattern allows:
*

0/1

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

Respuesta correcta

Having a single instance of a class, while allowing all classes have access to that instance.

El patrón Singleton se utiliza para restringir la instanciación de una clase a una sola instancia y proporciona un punto de acceso global a esa instancia. Esto es útil en

situaciones donde una única instancia de una clase debe controlar la coordinación de acciones en todo el sistema.

What is the result?
*
0/1

```
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)); }
        }
}
```

111.234 222.567
111.234 222.568

111.234 222.5678
An exception is thrown at runtime.

Respuesta correcta
111.234 222.5678

nf.parse(s) convierte la cadena en un número. Aunque setMaximumFractionDigits configura la cantidad máxima de dígitos fraccionarios para la salida formateada, no afecta al parseo directo.

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");
        }
}
```

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

```
Shape: constructor
Shape: foo
Square: foo
```

```
Square square = new Square ("bar"); square.foo ("bar"); square.foo();
Square square = new Square ("bar"); square.foo ("bar"); square.foo ("bar");
Square square = new Square (); square.foo (); square.foo(bar);
Square square = new Square (); square.foo (); square.foo("bar");
Square square = new Square (); square.foo (); square.foo ();
```

Respuesta correcta
Square square = new Square (); square.foo (); square.foo("bar");

square.foo("bar"); llama al método foo(String label) de la clase Square. Este método imprime "Square: foo" debido al argumento proporcionado.

```
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());
```

2
3
4

Compilation fails

El codigo es correcto y esta haciendo un map de 4

```
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("---");
        }
}
```

int x = 4, y = 3, z = 2;
int x = 3, y = 2, z = 3;
int x = 2, y = 3, z = 3;
int x = 2, y = 3, z = 4;
int x = 4, y = 2, z= 3;

Respuesta correcta

int x = 2, y = 3, z = 4;