
Proceso de selección de BQ

# Prueba Java

---

Marco Martínez Ávila

## 1. ¿Qué diferencia hay entre una clase abstracta y una interfaz?

Una clase abstracta incluye la implementación de métodos y el uso de variables. Una interfaz, por su parte, admite sólo la definición de los primeros y únicamente la utilización de constantes -atributos precedidos de las palabras clave *static final*, aunque no se indique explícitamente-.

## 2. ¿Una clase abstracta puede implementar métodos final?

Sí, una clase abstracta puede implementar métodos *final*.

## 3. Tenemos una clase con una propiedad name. Queremos que el nombre sea accesible para que se pueda usar por otras clases, pero que no puedan modificarlo. ¿Cómo podemos hacer esto?

Utilizando la palabra clave *final*, o *getters*.

### Palabra clave *final*

La palabra clave *final* impediría que la propiedad "name" fuera modificada, pero sería perfectamente accesible desde otras clases (siempre y cuando tenga la visibilidad adecuada).

```
public class Person {
    final protected String name = "Marco";
}
```

```
Person person = new Person();
String name = person.name;
```

### *getters*

Definiendo la propiedad "name" con visibilidad privada, se podría implementar un *getter* para acceder a su contenido. Omitiendo el *setter* correspondiente no sería posible modificar su valor, consiguiendo de esta forma el comportamiento deseado.

```
public class Person {
    final private String name = "Marco";

    public String getName() {
        return name;
    }
}
```

```
Person person = new Person();
String name = person.getName();
```

## 4. Estamos implementando un servicio de suscripción, en la que queremos hacer cobros recurrentes a nuestros clientes. Por el momento sólo soportamos el pago con PayPol, pero en el futuro soportaremos maś sistemas de pago. Define un interfaz que defina un método pay() al que se le pase un objeto Order y devuelva un booleano indicando si el pago ha sido correcto o no.

Se adjunta a este documento el código implementado. Se ha escrito teniendo en cuenta que se desconoce de dónde proviene la información almacenada en el objeto Order, y centrándose sólo en la funcionalidad requerida.

También se ha pretendido conseguir una implementación con una estructura adecuada de cara a la escalabilidad.

En un *enum* figuran los métodos de pago que se prevé aceptar.

```
1 public enum PaymentMethod {
2     PAYPAL, MASTERCARD, VISA;
3 }
```

Por su parte, el método "pay()" evalúa si el mecanismo escogido es PayPal, y devuelve un resultado acorde a lo estipulado en el enunciado.

```
1 public boolean pay(Order order) {
2     switch (order.getPaymentMethod()) {
3         case PAYPAL:
4             return true;
5         default:
6             return false;
7     }
8 }
```

## 5. Nombra dos patrones de diseño y pon un ejemplo de uso.

- **Patrón estado**: la máquina de Mealy.
- **Patrón MVC**: el framework Spring.

## 6. Te encuentras con un problema de JAVA que no sabes solucionar, ¿qué haces? ¿dónde acudes a buscar pistas de cómo resolverlo?

Buscar en Google, que seguramente me derive a StackOverflow. Si el criterio de búsqueda no es algo inmediato, como por ejemplo una excepción, es importante escoger la cadena adecuada para encontrar cuanto antes la forma de resolver el problema.

Entiendo que lo que define e identifica a un ingeniero no es sólo el hecho de poseer amplios conocimientos sobre un determinado tema; lo verdaderamente importante es la habilidad para solucionar problemas con unos determinados medios, para adaptarse a lo que requiera la situación. Una filosofía que en las nuevas generaciones se está perdiendo.

## 7. ¿Qué podemos hacer para evitar un ataque SQL Injection?

Usar PreparedStatement en lugar de concatenar cadenas de texto.

## 8. Tenemos dos tablas MySQL Client (id, name, invoicingAddress, sendAddress) y Address (id, address, city, province, zip, country). Un Client tiene dos relaciones 1aN con la tabla Address mediante los campos invoicingAddress y sendAddress. Queremos obtener un listado de los clientes con la dirección de facturación.

Los campos "id" de ambas tablas constituyen su clave primaria. En el caso de Client, invoicingAddress y sendAddress son claves externas que enlazan con el "id" de Address, formando así la relación mencionada.

Teniendo en cuenta que el objetivo es obtener una lista de clientes a partir de su dirección de facturación, la instrucción SQL a ejecutar sería:

```sql
SELECT * FROM Client WHERE invoicingAddress=X
/* Where X is the id of the Address table */
```

## 9. ¿Qué es un Web Service? ¿has usado alguno? ¿en qué formato suelen devolver las respuestas?

Un Web Service es una aplicación que corre en un servidor, cuyos resultados se proporcionan a través de la ejecución de diversos métodos. Esta información es accesible utilizando el protocolo HTTP.

Existen multitud de servicios web que devuelven una gran variedad de datos. Por ejemplo, en algunos proyectos personales se hizo uso de Web Services SOAP y REST para la conversión de monedas, de grados Celsius a Fahrenheit, u obtención de una lista de países, comunidades autónomas y provincias para combinar con la API de Google Maps, entre otros.

En los tipo SOAP, las respuestas obtenidas están en formato XML. Los REST, por su parte, amplian el repertorio con alternativas como JSON.

## 10. En un proyecto GIT, ¿cómo evitarías almacenar el directorio "vendor" en el repositorio?

Añadiendo dicho directorio al fichero ".gitignore".

## 11. En nuestra aplicación, tenemos un proceso de generación de PDFs que tarda mucho tiempo en ejecutarse. Cuando un usuario hace la petición tarda mucho tiempo en devolverle el fichero generado, y en ocasiones da un timeout. Además, cuando varios usuarios generan PDFs simultaneamente el servidor tiene problemas de rendimiento. ¿qué podemos hacer? ¿cómo solucionarías el problema?

Se podría invertir en un servidor con mejores prestaciones, que redujera los tiempos de procesamiento de PDFs.

Otra alternativa sería repartir la carga entre dos o más máquinas, utilizando la política de distribución que se considerase conveniente, como por ejemplo Round Robin. De esta forma, se podría redirigir la petición de los usuarios a un servidor u otro de forma transparente para ellos.

## 12. Escribe una clase con un método "generate" que genere quinielas aleatorias. Debe devolver quince valores. Los valores posibles son "1","X","2". Las condiciones que debe cumplir son: el número de valores "X" debe ser par, debe haber más valores "1" que "2" y debe haber al menos un valor "2".

Se ha creado la clase GeneratePool, cuyo método "generate" devuelve un array de String con los 15 valores de la quiniela.

En primer lugar, se calcula el número de ocurrencias de cada opción ("1", "X" o "2") atendiendo a las condiciones iniciales.

```
1     private int[] calculateResultNumber() {
2
3     /* In order to meet the three conditions, the number of '2' must be between 1
4     and 6 */
5     n2 = ThreadLocalRandom.current().nextInt(1, 7);
6
7     /* If the number of 'X', nX, must be even, n1+n2 must be odd. So if n2 is even, n1
8     must be odd, and vice versa */
9     if (n2%2 == 0) {
10        n1 = (ThreadLocalRandom.current().nextInt(n2/2, ((15-(n2+3))/2)+1)*2)+1;
11    }
12    else {
13        n1 = ThreadLocalRandom.current().nextInt((n2+1)/2, ((15-(n2+2))/2)+1)*2;
14    }
15    nX = 15 - (n1+n2);
16
17    return new int[]{n1, nX, n2};
18 }
```

A continuación, en el array "results" se almacenan los distintos resultados, ordenándose de forma aleatoria.

```
for (int i=0; i<TOTAL_RESULTS; i++) {
    int index = ThreadLocalRandom.current().nextInt(0, 3);

    /* If one option has completed its number of occurrences, another one must be
    written to the results array */
    while (resultNumber[index%3] == 0) {
        index++;
    }

    results[i] = RESULT[index%3];
    resultNumber[index%3]--;
}
```

El código completo y los test correspondientes pueden consultarse en los ficheros adjuntos.