# Prueba técnica.

Hola primero de todo, felicidades por llegar a la última fase del proceso de contratación. En este punto tocará programar un poco, el objetivo del ejercicio es conocer como trabajas con una tarea real.

Te agradecemos el tiempo dedicado.

Preguntas:

# Buenas prácticas:

1. ¿Qué piensas de Clean Code y los principios S.O.L.I.D ?

Pienso que todos los programadores deberían conocer como mínimo la intención de cada uno de estas buenas prácticas ya que permiten una fácil lectura de código y sobretodo un buen mantenimiento y evolución del mismo.

1. Define acoplamiento y cohesión en clases.

**Acoplamiento**: hablamos de acoplamiento entre clases cuando existe dependencia de una o más clases para realizar una operación. Esto quiere decir que cuantas más dependencias existan más fuerte será el acoplamiento.

**Cohesión**: hablamos de cohesión entre clases cuando nos referimos al nivel de relación que existe entre los elementos de una clase. Esto quiere decir que cuantas más dependencias existan más débil será la cohesión.

Se podría decir que acoplamiento y cohesión van de la mano y son proporcionalmente inversas ya que a más dependencias para realizar una operación, menos nivel de cohesión existirá dentro de nuestra clase, la cohesión es al fin y al cabo eso, una manera de medir el nivel de relación que existe entre las dependencias (si las hay) que componen nuestra clase.

1. ¿Qué nos indica una clase con baja cohesión?

Una clase con baja cohesión nos está pidiendo que sea refactorizada puesto que existen muchas dependencias con otras clases. Aunque para operaciones complejas es complicado conseguir aumentar la cohesión tenemos que almenos contemplar la posibilidad de hacerlo.

1. Define el patrón Decorator:

Antes de comenzar a dar una respuesta he de decir que había leído acerca de este patrón, sin embargo no recordaba en absoluto su objetivo por lo que he tenido que revisarlo, una vez entendido y puesto en práctica puedo decir que el patrón Decorator:

Busca dotar a los objetos de la capacidad para aumentar/modificar su funcionalidad sin tener que recurrir a la herencia.

He hecho un pequeño ejemplo y lo he subido en mi github: <https://github.com/bgtiban/patronDecorador>

# Java:

1. ¿Qué es un record?

Una clase de tipo record tiene como objetivo almacenar información sencilla, lo que sería una clase POJO. La particularidad de este tipo de clases es que no es necesario definir los constructores, atributos, getters, setters de la manera en la que estamos acostumbrados, bastaría con poner public record Coche(String matricula, String marca) {} y los getters, setters, constructores serán creados (no visibles) de manera automática.

1. ¿Qué es un streams?
2. ¿Cuando usarías un HashMap o TreeMap?
3. synchronized(this) vs synchronized(A.class)
4. Checked vs Unchecked Exception
5. ¿Cómo funciona el paso de parámetros? (Valor vs Referencia)

# SQL:

1. Diferencia entre inner join y left join
2. ¿Para que sirve la instrucción HAVING?
3. ¿Que es una regla de integridad referencial?
4. ¿Para qué se usan los índices? Explica sus ventajas e inconvenientes en lectura y escritura.

# Shell en Unix:

1. Cómo se edita el crontab:
2. Diferencia entre comillas dobles y simples:
3. Secuencia de comando para matar un proceso llamado PRUEBA:

# Kata:

Descripción.

La tarea consiste en implementar, sin usar dependencias externas (se pueden usar sólo para testing), un asistente de contratación en memoria que ofrezca al cliente la combinación de servicios más económica, que es aquella cuya media de los importes es mínima. Cada servicio sólo se puede combinar con su sucesor.

Por ejemplo si tenemos el siguiente listado de servicios: S1(10€), S2(5€), S3(5€).

Tendríamos las siguientes combinaciones:

S1,S2 => (10+5)/2 = 7.5

S2,S3 => (5+5)/2 = 5

S1,S2,S3 => (10+5+5) /3 = 6,67

En este caso la mejor opción para el ejercicio sería: S2,S3.

El asistente debe implementar los métodos de la interfaz es.gossan.assistant.HiringAssistant:

- List<Services> searchMinimalAmount(): Devuelve la combinación más económica del listado

de servicios.

- void add(Service service): Añade un servicio al listado.

Definimos un servicio como:

record Service (int id, String name, double amount){}

Ejemplos:

Caso 1:

[

{"id": 0,"name":"S1","amount": 4.0},

{"id": 1,"name":"S2","amount": 2.0},

{"id": 2,"name":"S3","amount": 2.0},

{"id": 3,"name":"S4","amount": 5.0},

{"id": 4,"name":"S5","amount": 1.0},

{"id": 5,"name":"S6","amount": 5.0},

{"id": 6,"name":"S7","amount": 8.0}

]

Mejor combinación: S2,S3

Caso 2:

[

{"id": 0,"name":"S1","amount": 5.0},

{"id": 1,"name":"S2","amount": 6.0},

{"id": 2,"name":"S3","amount": 3.0},

{"id": 3,"name":"S4","amount": 4.0},

{"id": 4,"name":"S5","amount": 9.0}

]

Mejor combinación: S3,S4

Caso 3:

[

{"id": 0,"name":"S1","amount": 5.0},

{"id": 1,"name":"S2","amount": 6.0},

{"id": 2,"name":"S3","amount": 2.0},

{"id": 3,"name":"S4","amount": 3.0},

{"id": 4,"name":"S5","amount": 2.0}

]

Mejor combinación: S3,S4,S5

Importante: Respecto a la plantilla entregada, siéntete libre modificar lo que consideres oportuno:

nombres paquetes, interfaz, cambiar el record Service a clase o la versión de Java (mín 8). Lo

importante es que tu algoritmo sea eficiente y sigas buenas prácticas de programación.