Respuestas prueba técnica
Medrar soluciones de software empresarial
Presentado por: **David Cano Baquero**

Preguntas Conceptuales

1. ¿Qué significa “alta cohesión bajo acoplamiento” en OOP?

La cohesión y el acoplamiento son unidades de medida para evaluar un diseño, cada una de estas es directamente proporcional a un buen o mal diseño.

Cohesión
Cuando se habla de cohesión, estamos hablando sobre el propósito, en este caso en el de un módulo, de esta manera cuando un módulo tiene un propósito o una responsabilidad específica, se dice que tiene alta cohesión, y cuando tiene varios propósitos se dice que su cohesión es baja, la mejor practica para tener un buen diseño es tener alta cohesión (módulos con propósitos únicos o específicos), para así de esta forma reutilizar los módulos(clases) ya que se puede utilizar por ejemplo esta clase en programas que se necesiten en el futuro, y así mismo al tener cada clase un único propósito es más sencillo ver que es lo que está pasando en el caso de un error no esperado.

Acoplamiento
Es una medida inversamente proporcional al diseño, es decir una aplicación que presente un alto acoplamiento tendrá un mal diseño, y viceversa, si presenta bajo acoplamiento tendrá un buen diseño, el acoplamiento se entiende como la interconexión entre las clases y el grado de dependencia entre ellas, una aplicación con un alto grado de acoplamiento y dependencia entre clases, ocasiona que los cambio que hagamos en una clase especifica repercutirán en muchas otras clases, por el contrario si tenemos un caso donde el acoplamiento sea bajo en muchas ocasiones esos cambios no repercutirán en las demás clases, o en un bajo grado.

Con esto se puede concluir que una alta cohesión y bajo acoplamiento llevan a que una aplicación cuente con una alta calidad en su diseño, obteniendo a su vez beneficios como: Reusabilidad, capacidad de prueba, mantenibilidad, legibilidad, entre otros.

2. ¿En qué caso favorecería el uso de la herencia sobre la composición en OOP?

1. Cuando ambas clases están dentro del mismo dominio lógico: en el caso de empleado y persona, el empleado siempre va a ser una persona, y a su vez la herencia no va a salir del dominio lógico(paquete), de esta manera queda la herencia acotada dentro del dominio.

2. Cuando la subclase es un subtipo de la superclase, usando el ejemplo anterior sabemos que un empleado es una persona y siempre va a serlo, no debería ser un animal, no debería.

3. Cuando la jerarquía de clases que se quiere montar está orientada a la extensión.

4. Ser un proyecto de baja duración, el cual no escalara mucho a nivel de extensión a lo largo del tiempo.

Estos literales nos indicarían que la herencia es un buen camino para elegir para modelar la solución de software.

3. ¿Por qué en OOP la encapsulación es importante?

La encapsulación es importante ya que por medio de esta podemos organizar datos y métodos de una estructura para así evitar el acceso a datos por cualquier otro medio distinto a los especificados, esto garantiza la integridad de los datos que están contenidos en un objeto.

4. ¿Qué implicaciones trae en el diseño de una aplicación que respete el principio “cerrado a la modificación abierto a la extensión”?

- Que su desarrollo pueda ser ligeramente más lento, ya que esto sentaría bases solidad para con base en el código ya estructurado se puede dar solución a requerimientos que puede aparecer en un futuro.

- Que una vez que la mayoría del código este completo este podrá adaptarse fácilmente a los cambios recurriendo a la extensión del comportamiento de sus entidades añadiendo nuevo código, pero nunca cambiando el código ya existente.

- Da soporte a la facilidad de mantenimiento, de reutilización y de verificación, siendo fácil de extender, y cerrado en el sentido de que está listo para ser utilizado sin tocar el código existente (modificarlo).

5. ¿Qué significa la palabra clave static, y dónde puede ser usada?

Los atributos de una clase pueden ser atributos de clase o atributos de instancia, en el caso de tener la palabra reservada static luego de un modificador de acceso como public/private lo que se logra es que la variable pase a ser un atributo de clase (es única para todas las instancias (objetos)), es decir si cambiamos dicha variable en cualquiera de las instancias, ya que es un atributo de clase repercutirá en todas las otras instancias creadas, un atributo estático como ya no le pertenece a un objeto sino directamente a una clase, se puede acceder al valor de la variable sin necesidad de instanciar un objeto de esa clase, lo único que se debe hacer es utilizar el nombre de la clase ejemplo *MiClase.variable*.

La palabra reservada static también puede ser usada en ámbito como métodos, tiene el mismo comportamiento que las variables estáticas en una clase puede ser accedido o invocado sin la necesidad de tener que instanciar un objeto de la clase.

6. ¿Qué es polimorfismo? Descríbalo con un ejemplo

Para explicar que es polimorfismo haremos uso del siguiente ejemplo:
En la Figura 1 se puede observar 3 clases diferentes en donde *VehiculoDeportivo* y *VehiculoTurimos* heredan de *Vehiculo*, teniendo a su vez cada uno sus propios atributos.

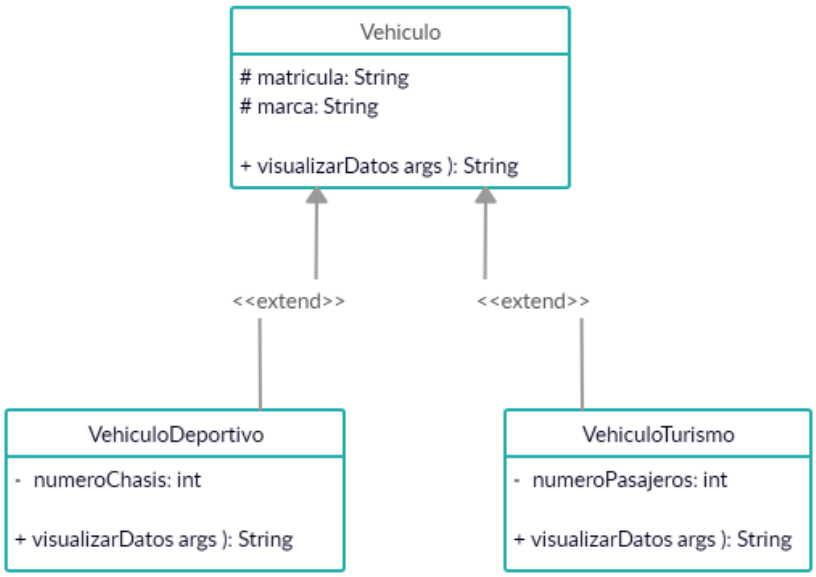

Figura 1. Digrama de clases

Vehiculo vehiculo= new Vehiculo(“SQD012”,”Spark”);

De esta manera crearíamos un objeto normal, corriente, pero según la definición de polimorfismo en la dice que el polimorfismo es *,”Es una relación de tipo herencia, en donde un objeto de la superclase puede almacenar un objeto de cualquiera de sus subclases”,* con esto podríamos hacer lo siguiente:

Vehiculo vehiculo= new VehiculoTurismo(“SQD012”,”Spark”,8541);

De esta manera funciona el polimorfismo en donde objetos de la clase padre pueden ser instanciados no solamente por la misma clase sino por sus clases hijas.

7. ¿Qué es el garbage collector?

En pocas palabras el Garbage Collector (GC) es el encargado de administrar de forma automática la memoria, liberando los objetos que ya no están en uso y no será usados en el futuro, en aplicaciones de grande envergadura es cuando el Garbage Collector se puede evidenciar de manera mas evidente, ya que en este tipo de aplicaciones fácilmente se puede perder el control de la cantidad de objetos que están siendo usados, y mas aun de los que ya fueron usados y no lo volverán a serlo, siendo el Garbage Collector encargado de estos últimos.

8. ¿Qué hace la palabra clave synchronized?

Esta palabra clave permite proteger un bloque de código(método) de un multiprocesamiento deseado, en otras palabras, en este caso el bloque de código es una zona critica en donde solo se requiere que un hilo se procese a la vez.

9. ¿Cuándo y por qué son los getters y setters importantes?

Los getters y setters son importantes cuando se quiere ocultar la representación interna de propiedad, permitir diferentes niveles de acceso, por ejemplo, la recibe puede ser público, pero el conjunto podría ser protegidos, entre otros muchos casos, esto con el fin de proteger la integridad de los atributos en conjunto con el encapsulamiento.

10. ¿Cuáles son las diferencias entre interfaces, clases abstractas, clases e instancias?

Entre las interfaces y las clases abstractas hay unas diferencias muy marcadas en cuanto a herencia múltiple, funcionalidad, comportamiento y visibilidad, de manera que por ejemplo:

Clase Abstracta	Interfaz
La palabra clave abstracta se usa para crear una clase abstracta y se puede usar con métodos.	La palabra clave de interfaz se usa para crear una interfaz, pero no se puede usar con métodos.
Una clase puede extender solo una clase abstracta.	Una clase puede implementar más de una interfaz.
Una clase abstracta puede tener métodos abstractos y no abstractos.	Una interfaz puede tener solo métodos abstractos.
Las variables no son definitivas por defecto. Puede contener variables no finales.	Las variables son finales por defecto en una interfaz.
Una clase abstracta puede proporcionar la implementación de una interfaz.	Una interfaz no puede proporcionar la implementación de una clase abstracta.
Puede tener métodos con implementaciones.	Proporciona una abstracción absoluta y no puede tener implementaciones de métodos.
Puede tener modificadores de acceso públicos, privados, estáticos y protegidos.	Los métodos son implícitamente públicos y abstractos en la interfaz de Java.
No admite herencias múltiples.	Es compatible con herencias múltiples.
Es ideal para la reutilización del código y la perspectiva de la evolución.	Es ideal para la declaración de tipo.

En cuanto a la diferencia entre clases e instancias tenemos que, una clase es un archivo que contiene funciones, procedimiento, variables, entre otros, y una instancia es una copia de una clase u objeto en memoria por esta razón se puede instanciar un objeto tantas veces como sea necesario, ya que los objetos se crean a partir de las clases.

11. ¿Qué es sobrecarga de métodos?

Es la creación de varios métodos con el mismo nombre, pero con diferente lista de tipos de parámetros. Java utiliza el número y tipo de parámetros para seleccionar cuál definición de método ejecutar y no por el tipo que devuelve.

12. ¿Cómo se maneja o controla una excepción?

Para capturar un excepción en Java se utilizan bloques try..catch , los cuales capturan las excepciones que se hayan podido producir en el bloque de código delimitado.

La cláusula catch recibe como argumento un objeto Throwable, en el siguiente bloque de codigo se puede observar cómo funciona el bloque delimitado por try..catch.

```
// Bloque 1
try {
        //Bloque 2
} catch ( Exception error ){
        //Bloque 3
}
//Bloque 4
```

Sin excepciones:	1->2->4
Con una excepción en el bloque 2:	1->2*->3->4
Con una excepción en el bloque 1:	1*

También en los bloques try..catch se le puede añadir una clausula finally que permite ejecutar un fragmento de código independientemente de si se produce o no una excepción.

```
// Bloque 1
try {
        //Bloque 2
} catch ( Exception error ){
        //Bloque 3
} finally {
        //Bloque 4
}
//Bloque 5
```

Sin excepciones:	1->2->4->5
Con una excepción en bloque 2:	1->2*->3->4->5

Lanzar una excepción

Tambien se puede lanzar una excepción, de manera que podemos utilizar estas para conducir el flujo del programa hacia donde queremos que vaya encaminado.

throw new Exception ("Mensaje de error")

Cuando se lanza una excepción:

- Se sale inmediatamente del bloque de c6digo actual
- Si el bloque tiene asociada una cláusula catch adecuada para el tipo de la excepci6n generada, se ejecuta el cuerpo de la cláusula *catch.*
- Si no, se sale inmediatamente del bloque (o método) dentro del cual está el bloque en el que se produjo la excepción y se busca una cláusula catch apropiada.
- El proceso continúa hasta llegar al método main de la aplicación. Si ahí tampoco existe una cláusula catch adecuada, la máquina virtual Java finaliza su ejecuci6n con un mensaje de error.

Propagación de excepciones (throws)

Si en el cuerpo de un método se lanza una excepción (de un tipo derivado de la clase Exception), en la cabecera del método hay que añadir una cláusula throws que incluye una lista de los tipos de excepciones que se pueden producir al invocar el método.

Ejemplo

Public String nombre(String atributo) throws IOException …

Creación de nuevos tipos de excepciones

Se puede crear un nuevo tipo de excepción fácilmente definiendo una subclase de un tipo de excepción ya existente, por ejemplo:

```
Public MiExcepcion extends ArithmeticException
{
        ….
}
```

Preguntas de algoritmos y estructuras de datos

1. ¿Qué diferencias existen entre las estructuras de datos: lista enlazada (linked list), un arreglo y un tabla de hash (hastable/hashmap)?

- Cualquier elemento en un array puede ser accedido instantáneamente (O(1)). En las listas enlazadas, aparte del acceso al primer y al último elemento (dependiendo de la implementación), para acceder a los otros elementos, se debe recorrer la lista hasta encontrar el elemento deseado, hecho que hace que su rendimiento promedio sea muchísimo menor que el de un array.

- La cantidad de memoria utilizada de un elemento en una lista enlazada y en una table de hash es mayor que del mismo elemento almacenado en un array, debido a la existencia del puntero al siguiente elemento y quizás al anterior (en las listas doblemente enlazadas), las tablas hash almacenan la información en posiciones pseudo-aleatorias, así que el acceso ordenado a su contenido es bastante lento, en un array los elementos están almacenados uno al lado del otro, por tanto, cada elemento no necesita de espacio extra para almacenar otro tipo de información.

- Eliminar un elemento que está al medio en una lista enlazada es muy simple y eficiente, consiste simplemente en reasignar los punteros del siguiente y del anterior elemento de los elementos que están entre el elemento a ser eliminado (O(1)). Eliminar un elemento que está al medio en un array es costoso pues consiste en remover el elemento y desplazar todos los elementos siguientes a él, un lugar hacia atrás (O(M-N)).

- Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

- Desaprovechamiento de la memoria en las tablas hash, si se reserva espacio para todos los posibles elementos, se consume más memoria de la necesaria.

2. ¿Bajo qué criterio una función recursiva es más costosa que una función iterativa?

Entre más iteraciones se realizar para dar una solución a un problema específico, la función recursiva consume muchísima memoria, ya que mantiene las variables del método mientras que se ejecuta, y también mucho tiempo, por otro lado, la función iterativa también debe guardar la información para dar paso a la siguiente iteración del bucle, pero a comparación con una funciona recursiva el consume de memoria es menor.

En términos de eficacia tanto la iteratividad como la recursividad son útiles, sin embargo en términos de eficiencia ambas discrepan entre si, siendo más difícil modelar un problema complejo con iteratividad que con recursividad (modelar un problema con recursividad es más fácil ya que se divide un problema grande en partes más pequeñas), pero una vez modelado por medio de iteraciones su tiempo de ejecución es menor que el mismo problema modelado con recursividad.

3. ¿Con qué estructura de datos modelaría el sistema de transporte Transmilenio para realizar cálculos sobre las rutas y tiempos totales de cada ruta y encontrar la ruta más rápida de estación a estación?

Para resolver este problema buscaría algoritmos que me ayudaran a encontrar distancias en un grafo de un punto A a un punto B, como por ejemplo algoritmos tales como el *Algoritmo de Johnson, Algoritmo de Búsqueda A**, o el mas conocido el *Algoritmo de Dijkstra,* a partir de hay buscaría el que se acomodara mas a lo que necesito.

Por ejemplo, empezando a analizar se sabe que no solo se tiene que saber la distancia de un punto A a un punto B, es todo un sistema completo, en este caso descartaría el *Algoritmo de Dijkstra,* ay que este resuelve el problema de los caminos más cortos desde un único vértice origen hasta todos los otros vértices del grafo, y así continuaría descartando algoritmos hasta encontrar el que mas se ajustara, en este punto llegaría a 2 algoritmos que con ampliamente conocidos para resolver este tipo de problemas, el Algoritmo de Floyd – Warshall y el Algoritmo de Johnson, en donde ambos cumplen los requerimientos de encontrar los caminos más cortos entre todos los vértices lo que los diferencia es que el primero es más rápido para grafos de alta densidad y el segundo para grafos de baja densidad, en este caso revisaría el sistema de transporte en detalle para identificar cual de estos 2 implementaría, una vez elegido el algoritmo y modelado el sistema que manera el aplicativo, se pasaría a crear un grafo y nodos(clases) para asi realizar cálculos sobre las rutas y tiempos totales de cada ruta y encontrar la ruta más rápida de estación a estación.

4. ¿Con qué tipo de algoritmo se enfrentaría al problema de distribución de paquetes por peso en camiones de una empresa de mensajería para que se realizara de manera óptima?

Lo plantearía de una manera similar que el problema anterior, conozco que existen algoritmos que apoyan la toma de decisiones basados en estadística, de esta manera se puede modelar muchos problemas y darles una solución optima, este tipo de problemas es abordado por métodos como el de la Esquina Noroeste y El método del costo mínimo que típicamente ofrece mejores valores iniciales, mas bajos que en la Esquina Noroeste, por tanto elegiría el método del costo mínimo para dar solución a este problema, pasando a modelarlo en un esquema, y a partir de hay empezar a realizar la solución del problema por este medio.

Ingeniería de software

1. Liste las etapas básicas de la construcción de software, independientes a la metodología de desarrollo y defina qué es calidad de software.

Etapas básicas de la construcción de software, independientes a la metodología de desarrollo:

-Planificación
- Análisis
- Diseño
- Implementación
- Pruebas
- Instalación o despliegue
- Uso y mantenimiento

Calidad de software
El término puede ser ambiguo e incluso subjetivo porque, la calidad depende de quien la observa, por tanto la definiré con base a la definición de calidad en el software propuesta por la organización internacional de estándares (ISO/IEC DEC 9126). Se puede decir que el software tiene calidad si cumple o excede las expectativas del usuario en cuanto a:

1. Funcionalidad (que sirva un propósito),
2. Ejecución (que sea práctico),
3. Confiabilidad (que haga lo que debe),
4. Disponibilidad (que funcione bajo cualquier circunstancia)
5. Apoyo, a un costo menor o igual al que el usuario está dispuesto a pagar.

Por tanto la suma de estos factores es lo que hace que un software de alta o baja calidad, asimismo el conjunto de actividades que se realizan durante el proceso de desarrollo influencia la calidad del producto de software final. La medición de la calidad del producto está en términos del conteo de defectos encontrados en el software durante su desarrollo y operación.

2. Describa los escenarios de prueba que le definiría al proceso de hervir un huevo

Caso de prueba	
Objetivo del caso de prueba	Validar el proceso de hervir un huevo
Identificador	Huevo_Test_v1
Nombre del Caso	Proceso de hervir un huevo
Precondiciones	• Tener una cazuela pequeña • Agua suficiente para cubrir el huevo • Una cucharada de sal • Gas suficiente para mantener el fuego durante el tiempo necesario. • Agua fría
Paso	**Resultado Esperado**
1) Pon el agua a hervir en una cazuela pequeña.	
2) Cuando el agua comience a hervir, introduce el huevo con cuidado.	Ver burbujas que indiquen que el agua esta hirviendo.
3)Añade una cucharada de sal para que luego se pele más fácilmente.	
4) Cuando introduzcas el huevo en la cazuela, el agua perderá algo del hervor. Espera a que empiece a hervir de nuevo.	Volver a ver las burbujas indicando que el agua esta hirviendo.
5) Esperar de 10 a 12 minutos a que huevo termine de cocerse.	
5) Pasado ese tiempo, saca los huevos de la cazuela y déjalos enfriar o refréscalos bajo el chorro de agua fría.	Que el huevo se encuentre en buenas condiciones(sin grietas)
6)Pela el huevo para dejarlo listo para comer.	Que el huevo este perfectamente cocido

3. ¿Cuáles la diferencia entre concurrencia, disponibilidad y consistencia?

Se conoce como concurrencia a la coincidencia de varios sucesos o cosas en un mismo tiempo, a diferencia de la disponibilidad que se define como a el acceso de personas u organismos a los datos con los que se trabaja, y de la misma manera la consistencia hace referencia a que los datos se mantengan íntegros a lo largo de todo el proceso.

4. Mencione algunos patrones de diseño: ejemplo Singleton.

- Fabrica
- Fabrica Abstracta
- Prototype
- Facade
- Decorador
- Proxy
- Comand
- Memento
- Observador
- Estrategia
- DAO
- Inyección de dependencias
- MVC (Modelo Vista Controlador)

5. Explique la inyección de dependencia y su relación con la inversión de control.

La inyección de dependencias es un subtipo de la inversión de control, hoy en día los principales frameworks implementan ambos conceptos con el mismo contenedor, como por ejemplo Spring tiene contenedores que asumen el control del ciclo de vida de los objetos (Inversión de Control) y además son capaces de inyectar dependencias (DI) que nos permiten a su vez aplicar el principio de inversión de las dependencias (DIP) en nuestra aplicación, existen dos formas genéricas, por constructor o por mutador(setter); la inyección de dependencias por constructor es cuando la creación de la instancia no depende de la clase. Sino que el constructor de la clase recibe como argumento un objeto de la clase abstracta ObjetoA lo que hace posible que un objeto sea capaz tener diferentes tipos de ObjetosA sin estar acoplado a alguno en particular.

Otra manera de inyectar dependencias es mediante el uso de setters, lo que permite mayor flexibilidad. Ya que incluso después de que un objeto de la clase haya sido creado, se podrá asignar un objeto diferente. Esto se logra trabajando con una Interfaz, y no con una clase normal, y gracias al polimorfismo el uso de una interfaz nos da la ventaja de cambiar de objetos a mitad de un proceso, entonces será posible cambiar la implementación de la dependencia utilizando un setter y de una forma muy sencilla, utilizar la inyección de dependencias permite crear componentes con bajo acoplamiento.

6. ¿Cuáles la diferencia entre un bloqueo optimista y un bloqueo pesimista?

El modelo de bloqueo optimista es un método de control de concurrencia utilizado en bases de datos relacionales que no utiliza bloqueo de registros. El bloqueo optimista permite que varios usuarios intenten actualizar el mismo registro sin informar a los usuarios de que otros también están intentando actualizar el registro, por el contrario, el modelo de bloqueo pesimista impide actualizaciones simultáneas de los registros. Tan pronto como un usuario empieza a actualizar un registro, se coloca un bloqueo sobre el mismo. Se informa a otros usuarios que intentan actualizar este registro de que otro usuario tiene una actualización en curso.