Parte 2

1 – Encuentra un error en la siguiente función:

public String calcularSueldo(){

double sueldoContractor = tarifaContractor\*getHorasTrabajadas();

return sueldoContractor;

}

El método pide que se devuelva un String, pero en cambio se devuelve un double.

2 – Encuentra 2 errores en el siguiente código:

private class Guitarra extends Instrumento{

@Override

public void tocar {

System.out.println("Tocar guitarra");

}

}

La clase guitarra debería ser pública y al método tocar le faltan los ().

3 – Crear un getter para la siguiente función:

private int sumar(int a, int b){

return a + b;

}

Parte 3

1 – Crear un objeto de tipo perro que tenga unos métodos propios (ladrar, jugar) y unos atributos heredados de la clase Animales (cantidad de patas, edad, pelaje)

2 – Crear un array de 5 números enteros y calcular el número más grande

3 – Convertir las letras de una palabra de minúsculas a mayúsculas

**Parte 1**

**1 - Explica qué es el testing manual y por qué es importante**

El testing manual es un tipo de testing donde no se utilizan herramientas de automatización para las pruebas de test, esto quiere decir que los casos de prueba se escriben a mano, así como también los reportes de bugs (errores). Son importantes ya que se ejecutan desde el punto de vista del cliente, lo que permite que se verifique con más certeza si el software funciona como debería según la documentación. Tiene también otros beneficios como la fácil aplicación por no requerir conocimientos de programación, su uso para testear diseños dinámicos y cambiantes y ser rentable.

**2 - Describe los pasos que seguirías para diseñar un conjunto de pruebas manuales para una nueva función.**

En primera instancia, analizaría los **requisitos**. Después identificaría los diferentes **escenarios de prueba**, los cuales están siempre enfocados a una función o característica específica con los pasos a seguir. Dentro de este paso también se definen los datos de prueba donde se documentan los valores de entrada y los resultados esperados. Después viene la **ejecución de los casos de prueba**, donde se verifica si el software se comporta según lo que se espera. Durante la ejecución se van registrando los resultados obtenidos y/o errores que se vayan encontrando. Por último, la **documentación de los resultados** obtenidos durante la ejecución de los casos de prueba, ya que así se obtiene un seguimiento y control adecuado de los problemas que se identifiquen y una referencia para pruebas futuras.

**3 - ¿Cuál es la diferencia entre el testing funcional y el testing no funcional?**

El testing funcional verifica que el sistema y sus funciones trabajen como se espera (para lo cual fueron desarrollados), mientras que el testing no funcional se enfoca en factores que relevan a la experiencia de uso del usuario, como la accesibilidad y la usabilidad.

**4 - ¿Qué es la caja negra y la caja blanca en el contexto del testing?**

Los tipos de caja hace referencia al enfoque que el tester hará sobre las pruebas que va a realizar en el sistema. En la **caja negra,** el tester no tiene un conocimiento detallado sobre la implementación del sistema, e intenta evaluar el sistema desde un nivel externo sin conocer cómo está implementado el código. Estas pruebas son más orientadas al comportamiento funcional.

Por otro lado, en la **caja blanca,** el tester sí tiene conocimiento detallado de la implementación interna del sistema ya que tiene acceso al código fuente, e intenta evaluar desde un nivel interno teniendo en cuenta la estructura del código y su lógica.

**5 - ¿Cómo abordarías la identificación y reporte de un error?**

La forma en la que abordaría la identificación y reporte del error sería mediante la documentación en un bug report, ya que ahí se describe todo lo necesario para que el equipo de desarrollo pueda corregir posteriormente el error identificado. En la pregunta 21 describo en detalle la información requerida en un bug report.

**6 - ¿Qué importancia tienen las pruebas de regresión y cómo las llevarías a cabo? ¿Cuándo hay que implementarlas?**

Las pruebas de regresión son bastante importantes, ya que su finalidad principal es descubrir bugs con respecto al comportamiento esperado del software, que pueden ser causados por la implementación de nuevos cambios en el programa. Es importante llevarlas a cabo regularmente para verificar que no haya errores por motivos de factores externos, no necesariamente relacionados a nuevas implementaciones de código. Siempre se deben ejecutar pruebas de regresión. La frecuencia dependerá de la estabilidad y la cantidad de cambios introducidos.

**8 - ¿Qué es la caja gris en el contexto del testing?**

Es una combinación de las técnicas de caja negra y caja blanca. O sea, ejecutar pruebas teniendo un conocimiento parcial de la implementación del código fuente sin dejar de lado los requisitos y diseños establecidos en las pruebas funcionales.

**9 - ¿Cómo manejarías la presión de tiempo al realizar pruebas manuales en un proyecto con plazos ajustados? ¿Cómo priorizarías las pruebas?**

Es importante tener en cuenta los requisitos mencionados por el cliente a la hora de intentar priorizar pruebas manuales. Sin embargo, también es necesario evaluar la prioridad de los casos positivos/negativos en términos funcionales y visuales haciendo foco en el impacto que pueden llegar a tener las fallas a nivel de negocio. Una vez priorizados los escenarios, en base al tiempo disponible, ejecutaría dichos casos para minimizar, en lo posible, la cantidad de errores, y así no desperdiciar tiempo ejecutando pruebas que no tienen tanto impacto.

**11 - Explica la diferencia entre pruebas de aceptación del usuario (UAT) y pruebas de sistema.**

En las pruebas de aceptación del usuario, el usuario se somete a un producto de software para que evalúe si se cumple con las expectativas como usuario. Mientras que las pruebas del sistema son para asegurarse de que todo funcione como fue planeado en la teoría, o sea, para validar las especificaciones técnicas.

**12 - ¿Qué es la metodología ágil y cuáles son sus principios fundamentales?**

La metodología ágil es un enfoque para el desarrollo de software, en las cuales se pretende ofrecer el producto adecuado mediante la entrega incremental y frecuente de pequeños trozos de funcionalidad a través de pequeños equipos con múltiples funciones, dando paso así a un feedback frecuente con el cliente y su corrección sobre la marcha conforme sea necesario.

**13 - ¿Cuál es la diferencia entre desarrollo en cascada (waterfall) y desarrollo ágil? ¿Cuáles son las ventajas y desventajas de cada uno?**

El desarrollo en cascada consiste en las entregas de grandes productos en largos periodos de tiempo, mientras que las entregas del desarrollo ágil son pequeñas entregas en periodos cortos llamados sprints.

La metodología del modelo waterfall tiene como ventajas que los requisitos se dan de forma clara y precisa ya que permanecen invariables durante todo el proyecto. También se fijan los puntos de inicio y fin de cada fase, lo cual facilita la medición del progreso. Por lo mismo, la fecha del producto acabado se puede definir incluso antes de haber comenzado el proyecto. Como desventajas de este modelo, los requisitos deben conocerse antes de empezar el desarrollo, por lo que puede retrasar el inicio del proyecto. Es poco flexible con la introducción de cambios, el cliente no tiene la oportunidad de familiarizarse con el sistema ya que ve el producto una vez finalizado.

Las metodologías ágiles tienen como ventajas que responden al mercado y al cliente ya que se adapta de forma más rápida a sus necesidades con la posibilidad de cambiar de dirección conforme lo exija la situación, lo cual facilita el feedback con el cliente.

**14 - ¿Cómo manejarías el cambio de requisitos en mitad de un proyecto?**

Si ya se estaban haciendo pruebas sobre los requisitos iniciales, detenerlos de inmediato y comenzar a hacer de vuelta los escenarios de prueba junto con la ejecución y documentación para no perder tiempo testeando sobre requisitos que ya no están vigentes. En caso de que el cambio de requisitos no sea absoluto, intentaría salvar la mayor cantidad de escenarios con su ejecución y documentación para no desperdiciar el tiempo invertido.

**15 - ¿Qué proceso seguirías para diseñar casos de prueba para una nueva función?**

Como mencioné anteriormente, para diseñar casos de prueba para una nueva función, en primera instancia analizarìa los requisitos y después identificaría los diferentes escenarios de prueba. Posteriormente haría la ejecución de los casos de prueba, para después realizar una documentación de los resultados. A todo esto, revisaría qué casos de prueba se podrían automatizar para agilizar el proceso de testing. Así, cuando las pruebas requieran ejecutarse con frecuencia, no se perdería tiempo en ejecuciones manuales.

**16 - ¿Cómo determinarías qué características del software deben probarse y cuáles no?**

Primero revisaría los requisitos establecidos en el criterio de aceptación de las historias de usuario ya que ahí podemos verificar lo que es necesario probar para que la historia se considere validada correctamente.

**17 - ¿Cómo equilibrarías entre casos de prueba positivos y negativos en el conjunto de pruebas?**

Lo ideal siempre es hacer todos los casos de prueba que sean posibles dentro de las características del software que deban probarse, por lo que estaría bueno tener un amplio porcentaje tanto de pruebas positivas como de pruebas negativas. Habiendo dicho esto, es importante priorizar los casos de prueba antes de su ejecución sin importar si son positivos o negativos ya que ambos son necesarios para que el cliente se sienta satisfecho. En la mayoría de los casos, la cantidad de pruebas positivas o negativas va a depender de los requerimientos.

**18 - ¿Cuál sería el enfoque para la priorización de casos de prueba, cuáles casos de prueba ejecutarías primero?**

En primera instancia priorizaría los casos de prueba que más impacto tengan para el negocio, sin importar si son visuales o funcionales. Para priorizar, hay que tener en cuenta la frecuencia con la que estos escenarios van a ser utilizados por el cliente y los riesgos que estos escenarios implican.

**19 - Explica la diferencia entre casos de prueba funcionales y no funcionales. ¿Cómo diseñarías pruebas no funcionales?**

Como mencione anteriormente las pruebas funcionales verifican que el sistema y sus funciones trabajen como se espera (para lo cual fueron desarrollados), mientras que las pruebas no funcionales se enfocan en factores que relevan a la experiencia de uso del usuario, como la accesibilidad y la usabilidad.

Hay varias pruebas no funcionales que pueden diseñarse, como las de rendimiento, de carga, de estrés, de volumen, de seguridad, de compatibilidad, entre otras. Para el diseño de las mismas siempre es importante conocer qué requisitos no funcionales tenemos que tener en cuenta para saber qué tipo de prueba tenemos que llevar a cabo. Asimismo, podremos definir los objetivos de prueba y poder diseñar sus escenarios, y según sea necesario, utilizar las herramientas que nos permitan ejecutarlas. En el caso de las de rendimiento podemos utilizar Jmeter, por ejemplo.

**20 - ¿Cómo decide qué casos de prueba automatizar y cuáles no?**

Cuando las pruebas que se van a realizar son más simples y rápidas para ejecutar manualmente, no tiene sentido automatizar en este caso. Tampoco es recomendable automatizar cuando los datos de prueba no sean estables o predecibles. Si estos cambian con frecuencia, es mejor ejecutarlos manualmente.

Es mejor dejar la automatización para las pruebas que se necesiten ejecutar con frecuencia ya que ahorra tiempo y esfuerzo, y porque de esta forma se reducen errores en el código cuando se ejecuta ese conjunto de pruebas. Un ejemplo podría ser un escenario de log in en el que se deberán validar los campos de entrada y el registro correcto de un usuario. Al ser un área tan impactante para el usuario y al no tener complejidad de ejecución, se automatiza para ahorrar tiempo.

**21 - ¿Qué información consideras esencial incluir en un informe de error?**

En el bug report, es esencial incluir el título del error que se está presentando, así como también una descripción clara y concisa de lo que ocurre. La frecuencia con la que sucede y el ambiente (browsers, por ejemplo) en el que se desarrolla. Es importante aclarar la severidad y la prioridad que consideramos que tiene el tipo de error (P1,P2...). También se incluyen en el reporte el resultado esperado (qué es lo que tendría que pasar si no existiera este bug), y el resultado actual (lo que está sucediendo con el error). Como este reporte va hacia el equipo de desarrollo para que puedan reproducir y posteriormente corregir el error que se presenta, en el informe de error también se detallan los pasos para reproducir el error y en caso de ser necesario, las evidencias, que pueden ser capturas o grabaciones de pantallas. Finalmente, se detalla la fecha y hora en el informe, así como el nombre del tester que encontró el bug.

**22 - ¿Cómo priorizaría los errores que ha encontrado durante las pruebas?**

Teniendo en cuenta la prioridad y la severidad del bug encontrado. La prioridad es qué tanto este error afecta al uso del programa (qué tanto bloquea al usuario). Mientras que la severidad es qué tanto afecta al negocio, por lo que es importante tener en cuenta estos dos datos a la hora de priorizar los errores y asignar correctamente su valor en el bug report (Blocker, P1,P2,P3,P4).

**23 - ¿Cuál sería el enfoque para reproducir un error antes de informarlo?**

Antes de informar un error, es importante entender de dónde viene y por qué, bajo qué circunstancias y con qué frecuencia se presenta. Quizás sucede el 100% de las veces o quizás el 50%. Esta información es crucial para que el programador tenga las herramientas a disposición para poder arreglarlo.

**24 - ¿Qué información incluirías en un informe de error si el error solo ocurre en ciertos sistemas operativos o navegadores?**

La información sigue siendo la misma que se incluye en un informe normal; se completa de la misma manera, solo que en la parte de ambientes (browsers) se especifica que solamente ocurre en esos sistemas operativos o navegadores. Idealmente se aclara en el título para que, desde antes de leer su descripción, ya se tenga un contexto..

**25 - ¿Cómo manejarías la situación si encontraras un error que ya está documentado, pero aún no se ha corregido?**

En este caso dependería de la antigüedad que tenga ese bug y la severidad/prioridad que tenga asignada. Si considero como tester que ese bug afecta al cliente o a la aplicación de manera directa, se lo comentaría al equipo de desarrollo o de diseño para que evalúen si tiene que ser corregido con urgencia.

**26 - Explica la diferencia entre el frontend y el backend en el desarrollo web.**

El **front end** es la parte visual y gráfica (es la parte del cliente, donde interactúan los usuarios directamente). Esta presenta la interfaz de usuario, la cual permite al mismo interactuar con la aplicación. Está compuesta por las páginas web, formularios, botones, menús y todos los elementos de la interfaz propiamente dicha.

El **back end** es la parte lógica (donde se procesan, almacenan y gestionan los datos). Acá los usuarios no interactúan directamente. Está compuesta por los servidores, la base de datos, las APIs y la lógica de aplicación.

**27 - Explica la diferencia entre una base de datos SQL y una base de datos NoSQL. ¿Cuándo elegirías una sobre la otra?**

En la base de datos relacionales SQL se crean entidades separadas que dan estructuras y se relacionan entre ellas. Esta base de datos define cómo se deben componer todos los datos insertados en la base de datos. Los elementos se organizan como un conjunto de tablas con columnas y filas.

Por otra parte, las bases de datos no relacionales NoSQL no tienen una estructura definida, por lo que no hace falta tener el diseño de la base de datos, y permite almacenar y manipular datos no estructurados y semiestructurados.

Se puede elegir el tipo de base de datos SQL cuando una aplicación tenga un esquema fijo y bien definido, con las tablas y relaciones claras y estables. Asimismo cuando el tipo de datos que se esté manejando tenga una estructura relacional.

Se puede elegir una base de datos NoSQL cuando una aplicación tenga un esquema dinámico, con estructuras cambiantes y no predefinidas. También, cuando la aplicación maneje grandes volúmenes de datos y se necesite un rendimiento rápido en la lectura y escritura. De la misma forma, cuando el tipo de datos no requiere de una estructura relacional.

**28 - ¿Cuáles son las diferencias entre HTML, CSS y JavaScript? ¿Cuál es su papel en el desarrollo frontend?**

HTML es un lenguaje de marcado de hipertexto que define la estructura básica de una página web mediante el uso de etiquetas. Mientras que el CSS y el JavaScript son lenguajes de programación. El CSS permite crear y realizar cambios en un documento de HTML, para determinar el aspecto de los mismos, como la fuente, los colores, fondos, bordes y demás elementos. Por lo que, HTML es la estructura de la web y CSS es el diseño que define la parte visual. Por otra parte, JavaScript es interpretado y orientado a objetos. Permite que el contenido de una página web reaccione a las acciones del usuario.

El papel de las tres es bastante importante en el desarrollo del frontend, ya que a partir de estas tecnologías se va a construir el frontend: HTML define la estructura del contenido, CSS se encarga del diseño y la presentación, y JavaScript maneja la interactividad y la lógica del lado del cliente.

**29 - ¿Qué es una API?**

La API (Interfaz de programación de aplicaciones) es una interfaz que permite que las aplicaciones se comuniquen y compartan datos entre ellas. Es un puente entre el cliente y el servidor, se encarga de llevar la solicitud y la respuesta. Nos permite aprovechar el desarrollo que otras empresas han hecho para nuestra aplicación.

**30 – Explica el concepto de Programación Orientada a Objetos (POO)**

Es un paradigma de programación que se basa en el concepto de "objetos", donde los objetos representan entidades del mundo real. Para comprender cómo se comportan los objetos, es importante conocer que tienen características (atributos) y comportamientos o acciones (métodos).

**31 - ¿Cuál es la diferencia entre una clase y un objeto?**

Las clases son plantillas o moldes que definen la estructura y el comportamiento de los objetos. Una clase es como un plano que describe cómo se construirán los objetos de ese tipo. La clase representa al conjunto de objetos que comparten una estructura y comportamiento en común. Esta va a definir los atributos que componen el objeto y los métodos que pueden emplearse para trabajar con dichos objetos. Por otro lado, el objeto es una instancia de una clase que tiene atributos y métodos específicos de esa clase. Por lo que, toda clase es un objeto en Java, pero no todo objeto es una clase.

**32 – Explica el concepto de herencia, polimorfismo y encapsulamiento**

El concepto de **herencia** en Java es idéntico que en el mundo real, haciendo referencia a heredar características o rasgos en común de nuestros padres o abuelos. Con la creación de nuevas clases heredadas de clases ya existentes (clase padre) de la cual se pueden heredar campos, métodos, atributos o funcionalidades.

El **polimorfismo** también es un principio orientado a objetos, y hace referencia a la capacidad de que un objeto pueda comportarse de diferentes maneras. En el poliformismo, la clase anidada recibe los mismos atributos y métodos que la clase madre, pero cuenta con su propia implementación para cada método.

Por último, el **encapsulamiento** es la única forma de proteger el código de cambios no deseados y esconder variables para que no se pueda acceder a ellas de forma directa.