Ciclo formativo	CSIFC02. Ciclo superior desenvolvemento de aplicacións multiplataforma. (Gr. B)		
Módulo profesional	MP0487. Contornos de desenvolvemento		
Data e horario	20/03/2023. 16:20 - 19:20		
Apellidos e nome			
DNI:		**Sinatura**:	

INSTRUCCIONES

- La **duración máxima** del examen será de **3 horas**.
- No está permitido el uso de ningún dispositivo electrónico (móvil, tablet, etc.)
- El examen se entrega grapado y debe entregarse grapado. No se corregirá ninguna hoja suelta
- El examen debe realizarse con bolígrafo azul o negro. No se permite bolígrafo rojo
- No se permite bebida o comida durante la realización de la prueba.
- El examen consta de 2 bloques, uno compuesto por preguntas tipo test y otro por preguntas de desarrollo y un ejercicio.
- Las respuestas al bloque de preguntas test se anotarán en la página 2 (RESPUESTAS DE TEST). **Sólo se corregirá las respuestas indicadas en esta página**.
- El primer bloque de **preguntas tipo test** consta de **18** preguntas tipo test. Todas las preguntas valen lo mismo y **2** respuestas incorrectas **restan 1** respuesta correcta. La **puntuación máxima** de este bloque es **5 puntos**
- El segundo bloque consta de **6 cuestiones y 1 ejercicio**. La **puntuación máxima** de este bloque es **5 puntos.** (0,5 puntos cada cuestión y 2 puntos el ejercicio)

1. Marca todas las opciones verdaderas. Las pruebas de caja blanca:
   a. Analizan el código, por lo que se necesita tener conocimiento de programación
   b. No verifican la funcionalidad de los bluques
   c. Verifican mala o inexistente calidad en las estructuras de código
   d. Se centran exclusivamente en la funcionalidad de la aplicación sin tener en cuenta el código
   e. Todas son verdaderas

2. Dentro de las pruebas funcionales, la tecnica que explora la posibilidad de obtener el mismo resultado con difierentes escenarios es:
   a. Test AD-HOC
   b. Test basados en decisión
   c. Análisis de valores límite
   d. Test de equivalencia
   e. Test de alternativas

3. Dentro de las pruebas funcionales, la tecnica que tiene como objetivo romper el sistema es:
   a. Test AD-HOC
   b. Test basados en decisión
   c. Análisis de valores límite
   d. Test de equivalencia
   e. Test de alternativas

4. Cuales de los siguientes tipos de prueba se encuadran dentro de las pruebas no funcionales:
   a. Fiabilidad
   b. Tests basados en decisión
   c. Test de conjetura de errores
   d. Eficiencia
   e. Ninguna de las anteriores

5. Señala la respuesta correcta (Sólo hay una opción válida en esta pregunta). Una prueba de regresión:
   a. un tipo de prueba de validación, que se realiza durante todas las fases de desarrollo de software al final de cada una de las misma
   b. un tipo depruebas que se realiza cada vez que se realiza un cambio en un sistema
   c. Un paso en la depuración de un programa
   d. Pruebas asciadas a la fase de diseño
   e. Ninguna de las anteriores

6. El objetivo de las pruebas unitarias es:
   a. Probar cada uno de los requisitos funcionales de nuestro sistema
   b. Probar la interconexión de varios módulos de nuestro sistema operando al mismo tiempo
   c. Probar el correcto funcionamiento de cada módulo por separado
   d. Verificar el rendimiento de todo el sistema
   e. Ninguna de las anteriores

7. Cuales de los siguientes documentos que se generan durante el ciclo de vida del software están relacionados con las pruebas a las que someterá nuestro software:
   a. Plan de pruebas
   b. Registro de pruebas
   c. Especificación de caso de prueba
   d. Registro de pruebas
   e. Todas son correctas

8. ¿Cuál es el método disponible en Junit al que le pasamos el resultado que nosotros esperamos y la función que estamos testeando??
   a. assertTrue
   b. assertFalse
   c. assertNotNull
   d. assertEquals
   e. Ninguna de las anteriores

9. Con los test de equivalencia diseñamos casos de prueba:
   a. Al menos uno por cada clase de equivalencia tomando valores representativos de cada clase
   b. Solo nos fijamos en los valores límite de cada clase, el resto no son importantes
   c. Únicamente nos fijaremos en aquellos valores que creamos que puedan dar fallo, no es necesario que todas las clases tengan definido un caso de prueba
   d. Solo probaremos los valores que quedan fuera de los rangos admitidos

10. ¿Cómo se llama el almacén de versiones de GIT?
   a. Directorio
   b. Repositorio
   c. Módulo
   d. Etiqueta
   e. Ninguna de las anteriores

11. ¿Que documento genera Javadoc?
   a. Genera un archivo ejecutable
   b. Genera un nuevo código fuente, con comentarios Javadoc
   c. Genera un archivo HTML con la información de clases y método
   d. No produce ningún tipo de documento adicional
   e. Ninguna de las anteriores

12. Indica la respuesta correcta. La refactorización:
   a. Es una técnica de prueba complementario
   b. Es una técnica para documentar el código
   c. Es una técnica de programación no presente en los IDE
   d. Utiliza una serie de patrones de aplicación sobre el código fuente

13. Se produce legado rechazado cuando:

a. Cuando un método utiliza más elementos de otra clase que de la propia
b. Cuando hay subclases que usan pocas características de las superclases
c. Cuando tenemos código duplicado en una clase
d. Cuando la lista de parámetros de nuestra clase es extensa

14. Cual de las siguientes etiquetas de JavaDoc se usa para describir un método obsoleto:

a. @deprecated
b. @obsolet
c. @since
d. @old_version
e. @exception

15. Cuando NO debemos refactorizar:

a. Cuando el código no funciona
b. Cuando se necesitan nuevas funcionalidades
c. Cuando esté próxima la fecha de entrega
d. Cuando el código funciona.
e. Ninguna de las anteriores

16. La técnica de almacenamiento de versiones que almacena completa la última versión y los cambios necesarios para reconstruir cada versión anterior es:

a. Deltas directos
b. Marcado selectivo
c. Deltas inversos
d. Ninguna de las anteriores

17. ¿Cuales de las siguientes son herramientas de control de versiones?:

a. Mercurial
b. NetBeans
c. Visual Studio Code
d. GIT
e. Ninguna de las anteriores

18. ¿Cuales de las siguientes herramientas son analizadores de código?:

a. Mercurial
b. Sonarcube
c. PMD
d. GIT
e. Visual Studio Code

19. Indica cuales son los objetivos de las pruebas de rendimiento
(Videos tutoria colectiva, presentación Tipos de pruebas.pdf)

OBJETIVOS DE LAS PRUEBAS DE RENDIMIENTO

- **Localizar los cuellos de botella**
- **Identifican el sistema más lento.**
- **Cumplir requisitos del cliente.**
- **Permite asegurar:**
  - Velocidad
  - Escalabilidad
  - Estabilidad
- **Ahorran dinero**

20. Indica las ventajas básicas que proporciona la ejecución de pruebas individuales:
(Teoría del ministerio, archivo ED03 Version imprimible para uso offline.pdf, página 21, sólo habia que citarlas, así que con el texto en negrita era suficiente)

correctas. Las pruebas individuales nos proporcionan cinco ventajas básicas:

1. **Fomentan el cambio**: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
2. **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.
3. **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
4. **Separación de la interfaz y la implementación**: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
5. **Los errores están más acotados y son más fáciles de localizar**: dado que tenemos pruebas unitarias que pueden desenmascararlos.

21. Indica las ventajas de la refactorización:
(Videos tutoria colectiva, presentación 4,1 Refactorización.pdf)

VENTAJAS DE LA REFACTORIZACIÓN

- **Facilita la comprensión del código fuente.**
- **Reduce los errores.**
- **Permite programar más rápido**
- **Facilita los cambios.**

22. Indica y describe 5 patrones de refactorización
(Teoría ministerio, archivo ED04 Version imprimible para uso offline.pdf, página 6, ***sólo 5)***

- **Renombrar.** Cambiar el nombre de un paquete, clase, método o campo, por un nombre más significativo.
- **Encapsular campos.** Crear métodos de asignación y de consulta (**getters y setters**) para los campos de la clase, que permitan un control sobre el acceso de estos campos, debiendo hacerse siempre mediante el uso de estos métodos.
- **Sustituir bloques de código por un método.** En ocasiones se observa que un bloque de código puede constituir el cuerpo de un método, dado que implementa una función por si mismo o aparece repetido en múltiples sitios. De esta forma, cada vez que queramos acceder a ese bloque de código, bastaría con invocar al método.
- **Modificar la extensión del código.** Hacer un código más extenso si se gana en claridad o menos extenso sólo si con eso se gana eficiencia.
- **Reorganizar código condicional complejo.** Patrón aplicable cuando existen varios if o condiciones anidadas o complejas.
- **Crear código común** (en una clase o método) para evitar el código repetido.
- **Mover la clase.** Mover una clase de un paquete a otro, o de un proyecto a otro. Esto implica la actualización en todo el código fuente de las referencias a la clase en su nueva localización.
- **Borrado seguro.** Garantizar que cuando un elemento del código ya no es necesario, se borran todas la referencias a él que había en cualquier parte del proyecto.
- **Cambiar los parámetros del método.** Permite añadir/modificar/eliminar los parámetros en un método y cambiar los modificadores de acceso.
- **Extraer la interfaz.** Crea un nueva interfaz de los métodos public non-static seleccionados en una clase o interfaz.

23. Clasifica las herramientas de control de versiones de acuerdo al modo de organizar la información, explicando cada uno de las tipos
(Teoría ministerio, archivo ED04 Version imprimible para uso offline.pdf, página 24)

- **Sistemas locales:** se trata del control de versiones donde la información se guarda en diferentes directorios en función de sus versiones. Toda la gestión recae sobre el responsable del proyecto y no se dispone de herramientas que automaticen el proceso. Es viable para pequeños proyectos donde el trabajo es desarrollado por un único programador.
- **Sistemas centralizados:** responden a una arquitectura cliente-servidor. Un único equipo tiene todos los archivos en sus diferentes versiones, y los clientes replican esta información en sus entornos de trabajo locales. El principal inconveniente es que el servidor es un dispositivo crítico para el sistema ante posibles fallos.
- **Sistemas distribuidos:** en este modelo cada sistema hace con una copia completa de los ficheros de trabajo y de todas sus versiones. El rol de todos los equipos es de igual a igual y los cambios se pueden sincronizar entre cada par de copias disponibles. Aunque técnicamente todos los repositorios tienen la posibilidad de actuar como punto de referencia; habitualmente funcionan siendo uno el repositorio principal y el resto asumiendo un papel de clientes sincronizando sus cambios con éste.

24. Define y explica que es un breakpoint
(Teoría del ministerio, archivo ED03 Version imprimible para uso offline.pdf, página 41)

Los puntos de ruptura son marcadores que pueden establecerse en cualquier línea de código ejecutable (no sería válido un comentario, o una línea en blanco). Una vez insertado el punto de ruptura, e iniciada la depuración, el programa a evaluar se ejecutaría hasta la línea marcada con el punto de ruptura. En ese momento, se pueden realizar diferentes labores, por un lado, se pueden examinar las variables, y comprobar que los valores que tienen asignados son correctos, o se pueden iniciar una depuración paso a paso, e ir comprobando el camino que toma el programa a partir del punto de ruptura. Una vez realiza la comprobación, podemos abortar el programa, o continuar la ejecución normal del mismo.

25. El pseudocódigo que se muestra representa un modulo que forma parte de una aplicación de control de stocks que sirve para generar un aviso cuando la cantidad de existencias baja de un determinado número Se pide:

a. Obtener el grafo.
b. Calcular la complejidad ciclomática
c. Indicar los caminos de prueba
d. Diseñar casos de prueba para cada camino del apartado c