Curso de desarrollo de software

Examen Parcial

Reglas para la prueba calificada

- Queda terminantemente prohibido el uso de herramientas como ChatGPT, WhatsApp, o cualquier herramienta similar durante la realización de esta prueba. El uso de estas herramientas, por cualquier motivo, resultará en la anulación inmediata de la evaluación.
- Las respuestas deben presentarse con una explicación detallada, utilizando términos técnicos apropiados. La mera descripción sin el uso de terminología técnica, especialmente términos discutidos en clase, se considerará insuficiente y podrá resultar en que la respuesta sea marcada como incorrecta.
- Toda respuesta que incluya código debe ser acompañada de una explicación. La ausencia de esta explicación implicará que la pregunta se evaluará como cero.
- Se permite el uso de imágenes para explicar o complementar las respuestas, siempre y cuando estas sean de creación propia y relevantes para la respuesta, como es el caso de la verificación de los pasos de pruebas.
- Cada estudiante debe presentar su propio trabajo. Los códigos iguales o muy parecidos entre sí serán considerados como una violación a la integridad académica, implicando una copia, y serán sancionados de acuerdo con las políticas de la universidad.
- Todos los estudiantes deben subir sus repositorios de código a la plataforma del curso, según las instrucciones proporcionadas. La fecha y hora de la última actualización del repositorio serán consideradas como la hora de entrega.
- La claridad, orden, y presentación general de las evaluaciones serán tomadas en cuenta en la calificación final. Se espera un nivel de profesionalismo en la documentación y presentación del código y las respuestas escritas.
- No se otorgará tiempo adicional después de la hora de entrega establecida. Todas las entregas deben completarse dentro del plazo especificado.

Instrucciones de entrega para la prueba calificada

Para asegurar una entrega adecuada y organizada de su prueba calificada, sigan cuidadosamente las siguientes instrucciones. El incumplimiento de estas pautas resultará en que su prueba no sea revisada, sin importar el escenario.

- Cada estudiante debe tener un repositorio personal en la plataforma designada para el curso (por ejemplo, GitHub, GitLab, etc.). El nombre del repositorio debe incluir su nombre completo o una identificación clara que permita reconocer al autor del trabajo fácilmente.
- Dentro de su repositorio personal, deben crear una carpeta titulada **ExamenParcial-CC3S2.** Esta carpeta albergará todas las soluciones de la prueba.
- Dentro de la carpeta **ExamenParcial-CC3S2,** deben crear dos subcarpetas adicionales, nombradas **Ejercicio1 y Ejercicio2**, respectivamente y dentro de cada parte Ejercicio1 y Ejercicio deben estar Sprint1, Sprint2, Sprint3. Cada una de estas subcarpetas contendrá las soluciones y archivos correspondientes a cada ejercicio de la prueba.

Ejemplo de estructura:
/PracticaCalificada1-CC3S2
/Ejercicio1
- Sprint1
- Sprin2
- Sprint 3
/Ejercicio2
- Sprint1
- Sprin2
- Sprint 3

- Todas las respuestas y soluciones deben ser documentadas, completas y presentadas en archivos Markdown (.md) dentro de las subcarpetas correspondientes a cada ejercicio. Los archivos Markdown permiten una presentación clara y estructurada del texto y el código.
- Cada archivo Markdown de respuesta debe incluir el nombre del autor como parte del nombre del archivo. Esto es para asegurar la correcta atribución y reconocimiento del trabajo individual.
- No se aceptarán entregas en formatos como documentos de Word (.docx) o archivos PDF. La entrega debe cumplir estrictamente con el formato Markdown (.md) como se especifica.
- Es crucial seguir todas las instrucciones proporcionadas para la entrega de su prueba calificada. Cualquier desviación de estas pautas resultará en que su prueba no sea considerada para revisión.
- Antes de la entrega final, verifique la estructura de su repositorio, el nombramiento de las carpetas y archivos, y asegúrese de que toda su documentación esté correctamente formateada y completa.

Pregunta 1: Desarrollo de un juego de trivia en consola

Contexto del proyecto:

Este examen parcial está diseñado para evaluar tus habilidades prácticas en el desarrollo de software utilizando Java y Gradle, enfocándose particularmente en la implementación de un juego de trivia basado en consola. El proyecto se desarrollará en tres sprints, cada uno diseñado para avanzar progresivamente en la complejidad del juego y su funcionalidad, culminando en una aplicación de consola completamente funcional.

Descripción del juego de Trivia:

El juego de Trivia es un juego de preguntas y respuestas donde los jugadores deben responder preguntas de opción múltiple presentadas en la consola. Cada pregunta contiene exactamente una respuesta correcta entre varias opciones. El juego es simple pero debe ser implementado de manera que demuestre el manejo efectivo de la lógica básica de programación, estructuras de datos, y pruebas unitarias.

Reglas y funcionamiento del juego:

1. **Inicio del juego:** Al lanzar el juego, se debe mostrar un mensaje de bienvenida junto con las instrucciones sobre cómo jugar.

2. **Número de rondas:** El juego constará de un total de 10 rondas, cada una con una pregunta única.
3. **Preguntas:** Se presenta una pregunta con cuatro opciones de respuesta numeradas. Solo una opción es correcta.
4. **Selección de respuesta:** El jugador elige su respuesta ingresando el número correspondiente a la opción elegida.
5. **Puntuación:** Cada respuesta correcta otorga un punto. No se penaliza por respuestas incorrectas.
6. **Fin del juego:** Al finalizar las rondas, se muestra la puntuación total del jugador, junto con un desglose de respuestas correctas e incorrectas.

Formato de salida en consola:

- **Mensaje de Inicio:**
  Bienvenido al Juego de Trivia!

  Responde las siguientes preguntas seleccionando el número de la opción correcta.

- Durante el juego:

Pregunta 1: ¿Cuál es la capital de Francia?

1) Madrid

2) Londres

3) París

4) Berlín

Tu respuesta: 3

¡Correcto!

- Fin del juego:

Juego terminado. Aquí está tu puntuación:

Preguntas contestadas: 10

Respuestas correctas: 8

Respuestas incorrectas: 2

Detalle de los Sprints:

Sprint 1: Estructura básica y preguntas (2 puntos)

Objetivo: Configurar el entorno del proyecto y desarrollar la lógica básica para la manipulación y presentación de preguntas y respuestas.

Tareas:

1. Configurar el entorno del proyecto utilizando Gradle, incluyendo la dependencia para JUnit 5.
2. Desarrollar la clase **Question** con atributos para la descripción de la pregunta, las opciones disponibles y la respuesta correcta.
3. Implementar la clase **Quiz** que manejará el flujo del juego, incluyendo cargar preguntas, presentarlas al usuario y recibir respuestas.
4. Programar la lógica para mostrar preguntas de manera secuencial y permitir al usuario ingresar respuestas.
5. Escribir pruebas unitarias para asegurar que las preguntas se carguen correctamente y que las respuestas sean validadas adecuadamente.

Casos de prueba para JUnit:

- Verificar que la carga de preguntas desde un repositorio o archivo esté correcta.
- Asegurar que el sistema acepte y valide respuestas de manera adecuada, incluyendo la captura de entradas inválidas como letras o números fuera del rango.

Sprint 2: Lógica del juego y puntuación (2 puntos)

Objetivo: Implementar un sistema de puntuación y refinar la lógica del juego para manejar múltiples rondas y terminación del juego.

Tareas:

1. Ampliar la clase **Quiz** para incluir un sistema de puntuación que rastree las respuestas correctas e incorrectas.
2. Implementar lógica para múltiples rondas de juego, permitiendo al juego continuar hasta que se completen todas las preguntas.
3. Crear un mecanismo para terminar el juego una vez que se hayan respondido todas las preguntas, y mostrar el resultado final al usuario.
4. Escribir pruebas unitarias para verificar la precisión del sistema de puntuación y la correcta funcionalidad del flujo del juego.

Casos de prueba para JUnit:

- Asegurar que la puntuación se incremente correctamente con cada respuesta correcta.
- Comprobar que el juego concluya adecuadamente después de la última ronda.

Sprint 3: Mejoras en la interfaz y refinamiento (3 puntos)

Objetivo: Mejorar la interfaz de usuario en la consola para hacerla más amigable y agregar características adicionales como niveles de dificultad.

Tareas:

1. Refinar la presentación de preguntas y respuestas en la consola para mejorar la legibilidad y la interacción del usuario.

2. Introducir niveles de dificultad para las preguntas, permitiendo que el juego ajuste dinámicamente la dificultad basándose en el desempeño del jugador.
3. Implementar una funcionalidad para mostrar un resumen detallado al final del juego, incluyendo la puntuación total, respuestas correctas e incorrectas.
4. Desarrollar pruebas unitarias adicionales para validar las nuevas funcionalidades y las mejoras realizadas en la interfaz de usuario.

Casos de prueba para JUnit:

- Verificar que las preguntas se muestren con el formato y claridad adecuados.
- Comprobar que el resumen final refleje de manera precisa la puntuación y las respuestas del jugador.

Preguntas relacionadas al juego Trivia (4 puntos)

1. ¿Qué caracteriza a una metodología ágil y cómo se aplicaría al desarrollo del juego de trivia? Describe las características principales como la iteración rápida, la adaptabilidad y la colaboración, y explica cómo estas pueden mejorar el desarrollo y la entrega del juego.
2. Explica cómo el principio de entrega continua podría implementarse en el proyecto del juego de trivia usando Gradle. Discute el uso de Gradle para automatizar compilaciones, pruebas y despliegues, facilitando una integración y entrega continuas.
3. Describa el ciclo TDD y cómo se aplicaría a una nueva característica en el juego, como la implementación del sistema de puntuación. Explica el ciclo de "Red-Green-Refactor" y cómo utilizaría TDD para desarrollar y validar la lógica de puntuación del juego.
4. ¿Cuáles son los beneficios de utilizar TDD en el desarrollo de software y cómo ayuda a prevenir regresiones en el juego de trivia?
5. Proporciona un ejemplo de cómo podrías refactorizar un bloque de código del juego de trivia para mejorar su mantenibilidad.
6. Explica la importancia de la refactorización en los sprints de desarrollo ágil y cómo se integra en el proceso de TDD. Discute cómo la refactorización es una parte integral del ciclo de TDD y cómo contribuye a la mejora continua del código en un entorno ágil.
7. Describe cómo JUnit 5 puede utilizarse para implementar pruebas parametrizadas en el juego de trivia. Proporciona un ejemplo concreto. Explica la funcionalidad de las pruebas parametrizadas en JUnit 5 y cómo se podría usar para probar diferentes casos de entrada para validar la lógica del juego.
8. ¿Qué características nuevas introduce JUnit 5 que son particularmente útiles para proyectos complejos como el juego de trivia?

- **Cada pregunta puntúa 0.5 puntos**

Instrucciones de entrega:

Tu proyecto debe ser entregado a través del sistema dado en la evaluación, incluyendo todos los archivos de código fuente y un reporte que documente el diseño y las pruebas realizadas. Asegúrate de incluir capturas de pantalla de la aplicación en funcionamiento y los resultados de las pruebas unitarias ejecutadas.

Criterios de evaluación:

1. **Funcionalidad del código:** El juego debe funcionar según las especificaciones dadas sin errores.
2. **Calidad del código:** El código debe ser claro, bien organizado y seguir las mejores prácticas de desarrollo de software.
3. **Cobertura de pruebas:** Debes proporcionar un conjunto completo de pruebas unitarias que demuestren la robustez y fiabilidad de tu aplicación.
7. **Documentación:** El reporte debe explicar claramente cómo funciona tu aplicación y cómo se diseñaron e implementaron las pruebas unitarias.

Pregunta 2: Desarrollo de un juego de Wordz

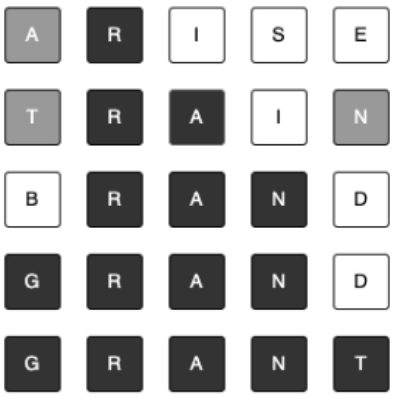
La aplicación Wordz se basa en un popular juego de adivinanzas. Los jugadores intentan adivinar una palabra de cinco letras. Los puntos se obtienen en función de la rapidez con que un jugador adivina la palabra. El jugador recibe comentarios sobre cada suposición para guiarlo hacia la respuesta correcta.

Para jugar Wordz, un jugador tendrá hasta seis intentos para adivinar una palabra de cinco letras. Después de cada intento, las letras de la palabra se resaltan de la siguiente manera:

- La letra correcta en la posición correcta tiene un fondo negro
- La letra correcta en la posición incorrecta tiene un fondo gris
- Las letras incorrectas que no están presentes en la palabra tienen un fondo blanco

El jugador puede usar esta retroalimentación para hacer una próxima suposición mejor. Una vez que un jugador adivina la palabra correctamente, obtiene algunos puntos. Obtienen seis puntos por una suposición correcta en el primer intento, cinco puntos por una suposición correcta en el segundo intento y un punto por una suposición correcta en el sexto y último intento. Los jugadores compiten entre sí en varias rondas para obtener la puntuación más alta. Wordz es un juego divertido, así como un entrenamiento cerebral suave.

Si bien la creación de una interfaz de usuario está fuera del alcance de esta actividad, es muy útil ver un posible ejemplo:



Crea un proyecto de Java y Gradle siguiendo metodologías ágiles para desarrollar un juego como Wordz organizando el proyecto en tres sprints, con énfasis en Test Driven Development (TDD),

patrón Arrange-Act-Assert (AAA), refactorización, código limpio, análisis de código estático y cobertura de código.

Sprint 1: Configuración inicial y funcionalidad básica (2 puntos)

En este sprint, se configura el entorno y se implementa la lógica básica del juego.

Clase Word

La clase **Word** será responsable de almacenar la palabra correcta y evaluar las suposiciones. Usaremos el código proporcionado para definir cómo se verifica cada suposición:

```
public class Word {

   private final String word;

   public Word(String correctWord) {

      this.word = correctWord;

   }

   public Score guess(String attempt) {

      var score = new Score(word);

      score.assess(attempt);

      return score;

   }

}
```

Clase Score

La clase **Score** evaluará la respuesta dada y almacenará los resultados de la evaluación:

```
import java.util.ArrayList;

import java.util.List;

public class Score {

   private final String correct;

   private final List<Letter> results = new ArrayList<>();

   public Score(String correct) {
```

```
        this.correct = correct;
    }
    public Letter letter(int position) {
        return results.get(position);
    }
    public void assess(String attempt) {
        for (int i = 0; i < attempt.length(); i++) {
            char current = attempt.charAt(i);
            if (isCorrectLetter(i, current)) {
                results.add(Letter.CORRECT);
            } else if (occursInWord(current)) {
                results.add(Letter.PART_CORRECT);
            } else {
                results.add(Letter.INCORRECT);
            }
        }
    }
private boolean isCorrectLetter(int position, char currentLetter) {
        return correct.charAt(position) == currentLetter;
    }
    private boolean occursInWord(char current) {
        return correct.contains(String.valueOf(current));
    }
}
```

Clase WordTest

Los tests verificarán la funcionalidad de las clases **Word** y **Score** usando JUnit:

```
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

import static com.wordz.domain.Letter.*;

public class WordTest {

   @Test

   public void oneIncorrectLetter() {

      var word = new Word("A");

      var score = word.guess("Z");

      assertScoreForGuess(score, INCORRECT);

   }

   @Test

   public void oneCorrectLetter() {

      var word = new Word("A");

      var score = word.guess("A");

      assertScoreForGuess(score, CORRECT);

   }


   @Test

   public void secondLetterWrongPosition() {

      var word = new Word("AR");

      var score = word.guess("ZA");

      assertScoreForGuess(score, INCORRECT, PART_CORRECT);
```

```
  }

  @Test

  public void allScoreCombinations() {

    var word = new Word("ARI");

    var score = word.guess("ZAI");

    assertScoreForGuess(score, INCORRECT, PART_CORRECT, CORRECT);

  }

  private void assertScoreForGuess(Score score, Letter... expectedScores) {

    for (int position = 0; position < expectedScores.length; position++) {

      Letter expected = expectedScores[position];

      assertThat(score.letter(position)).isEqualTo(expected);

    }

  }
```

}**Objetivos:**

- Configurar el entorno del proyecto con Gradle y JUnit 5.
- Implementar la lógica básica del juego Wordz para aceptar y validar suposiciones de palabras.
- Escribir casos de prueba iniciales siguiendo el enfoque TDD.

Tareas:

1. **Configuración del proyecto**: Crear un nuevo proyecto en Gradle y configurar dependencias para JUnit 5.
2. **Clase Word**: Implementar la clase básica con el método **guess** que acepte una suposición y devuelva un resultado básico. Utiliza los fragmentos dados donde defines la clase **Word** y sus métodos.
3. **Pruebas iniciales**: Escribir pruebas para validar suposiciones incorrectas y correctas utilizando JUnit 5. Usa el código de ejemplo proporcionado en las pruebas **oneIncorrectLetter** y **oneCorrectLetter**.
4. **Refactorización y código limpio**: Revisar el código para mejorar la claridad y la mantenibilidad.
5. **Análisis de código estático**: Configurar SonarQube para analizar el código y solucionar problemas identificados.

Sprint 2: Ampliación de la lógica de juego (2 puntos)

Objetivos:

- Ampliar la lógica del juego para incluir retroalimentación detallada sobre las suposiciones.
- Continuar el desarrollo utilizando TDD.

Tareas:

1. **Ampliar clase Word**: Implementar lógica para evaluar letras en posiciones correctas e incorrectas. Integra el código de **assess** y la evaluación de la posición de las letras.
2. **Pruebas de retroalimentación**: Ampliar el conjunto de pruebas para cubrir todos los casos posibles de retroalimentación, incluyendo **secondLetterWrongPosition** y **allScoreCombinations**.
3. **Refactorización**: Mejorar el diseño y la implementación basada en los resultados de las pruebas y el análisis de código estático.
4. **Cobertura de código**: Utilizar JaCoCo o otras herramienta para asegurar que la cobertura del código sea superior al 90%.

Sprint 3: Interfaz de usuario de consola y finalización (3 puntos)

Objetivos:

- Desarrollar una interfaz de usuario de consola simple para jugar el juego.
- Finalizar el proyecto con pruebas completas y documentación.

Tareas:

1. **Interfaz de usuario de consola**: Crear una interfaz de usuario básica utilizando la entrada y salida estándar de Java (System.in y System.out). Implementa un loop principal que acepte las suposiciones de los usuarios y muestre los resultados usando los métodos de **Word**.
2. **Pruebas de Integración**: Implementar pruebas que simulan la interacción del usuario con la consola. (opcional)
3. **Documentación**: Escribir documentación del código y del proyecto (opcional).
4. **Revisión final y ajustes**: Refactorizar el código según sea necesario y asegurar que todas las pruebas pasen.

Preguntas relacionadas al juego Wordz (2 puntos)

1. ¿Cuáles son los elementos clave que se deben considerar al diseñar un juego de adivinanza de palabras como "Wordz"? ¿Cómo influyen los sistemas de retroalimentación en la experiencia del jugador en juegos de trivia o adivinanzas?
2. Describe cómo el principio de "feedback continuo" en las metodologías ágiles podría aplicarse en el desarrollo del juego "Wordz".
3. ¿Cuáles son algunas características nuevas de JUnit 5 que lo hacen adecuado para pruebas unitarias en proyectos Java modernos? ¿Cómo se podría diseñar una suite de pruebas unitarias para validar la lógica de puntuación en "Wordz"?
4. ¿Por qué es importante la refactorización en el desarrollo continuo de un juego y cómo puede afectar la mantenibilidad del código? Proporciona un ejemplo de cómo podrías refactorizar un método complejo en el juego "Wordz" para mejorar la claridad y eficiencia.