Reporte Sprint #0

Instrucciones

Objetivos

- Tomar decisiones sobre el proyecto de desarrollo de software SOS.
- Aprender pruebas unitarias y programación de GUI en el lenguaje de tu elección.

Entregables y políticas de calificación

Lean el documento "descripción del Proyecto 3S2" cuidadosamente y toma las decisiones para el desarrollo del software.

Usen el siguiente template para completar tu reporte.

1. Decisiones claves para el proyecto SOS of the SOS (2 puntos)

Lenguaje de programación orientado a objetos	Java
Librería GUI (recomendable)	• java.awt • javax.swing
IDE (Integrated Development Environment)	IntelliJ IDEA Edition 2023.1
Framework xUnit (JUnit for Java por ejemplo)	Junit
Guía de estilo de programación (debe ser leído con cuidado)	https://google.github.io/styleguide/javaguide.html
Sitio de alojamiento del proyecto	https://github.com/miguelvega/Proyecto_CC3S2
Otras decisiones si procede	

Ejemplos de guía de estilo de programación:

- Guía de estilo de Java Google: https://google.github.io/styleguide/javaguide.html
- Guía de estilo de C++ Google: https://google.github.io/styleguide/cppguide.html
- Guía de estilo Python Google: https://google.github.io/styleguide/pyguide.html

2. Pruebas unitarias (8 puntos)

Encuentren un tutorial sobre el framework de pruebas unitarias que has elegido y escriban al menos dos pruebas xUnit de un programa que hayas escrito o encontrado en otro lugar. Adjunta aquí (1) la captura de pantalla de la ejecución de tu programa.

build.gradle (Sprint0_Junit) | TestKnapsackProblem.java | Element.java | Knapsack.java

```java
package com.example.demo;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class TestKnapsackProblem {
    @Test
    void testRepeatedElement() {
        var element1 = new Element( weight: 5, benefit: 10);
        var element2 = new Element( weight: 5, benefit: 10);
        var knapsack = new Knapsack( maximumWeight: 30, numElements: 2);
        knapsack.addElement(element1);
        assertTrue(knapsack.existElement(element2));
    }
    @Test
    void testDeleteElement(){
        var element1 = new Element( weight: 5, benefit: 10);
        var element2 = new Element( weight: 4, benefit: 8);
        var knapsack = new Knapsack( maximumWeight: 30, numElements: 2);
        knapsack.addElement(element1);
        knapsack.addElement(element2);
        knapsack.deleteElement(element2);
        assertFalse(knapsack.existElement(element1));
    }
}
```

Run: TestKnapsackProblem

Tests failed: 1, passed: 1 of 2 tests – 40 ms

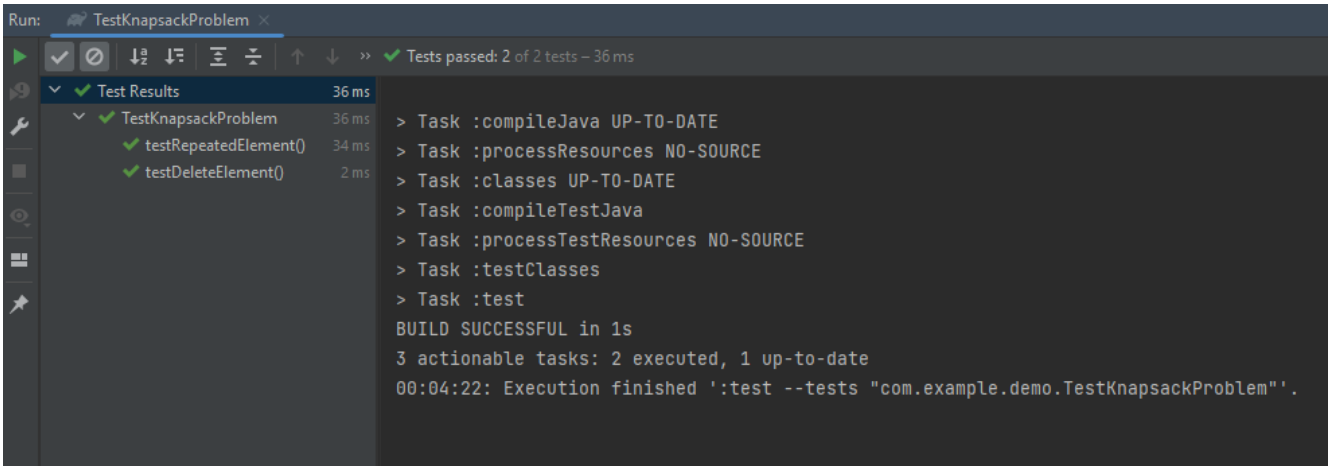
- Test Results 40 ms
  - TestKnapsackProblem 40 ms
    - testRepeatedElement() 29 ms
    - testDeleteElement() 11 ms

```
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test FAILED

expected: <false> but was: <true>
Expected :false
Actual   :true
<Click to see difference>
```

build.gradle (Sprint0_Junit) | TestKnapsackProblem.java | Element.java | Knapsack.java

```java
package com.example.demo;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class TestKnapsackProblem {
    @Test
    void testRepeatedElement() {
        var element1 = new Element( weight: 5, benefit: 10);
        var element2 = new Element( weight: 5, benefit: 10);
        var knapsack = new Knapsack( maximumWeight: 30, numElements: 2);
        knapsack.addElement(element1);
        assertTrue(knapsack.existElement(element2));
    }
    @Test
    void testDeleteElement(){
        var element1 = new Element( weight: 5, benefit: 10);
        var element2 = new Element( weight: 4, benefit: 8);
        var knapsack = new Knapsack( maximumWeight: 30, numElements: 2);
        knapsack.addElement(element1);
        knapsack.addElement(element2);
        knapsack.deleteElement(element2);
        assertFalse(knapsack.existElement(element2));
    }
}
```

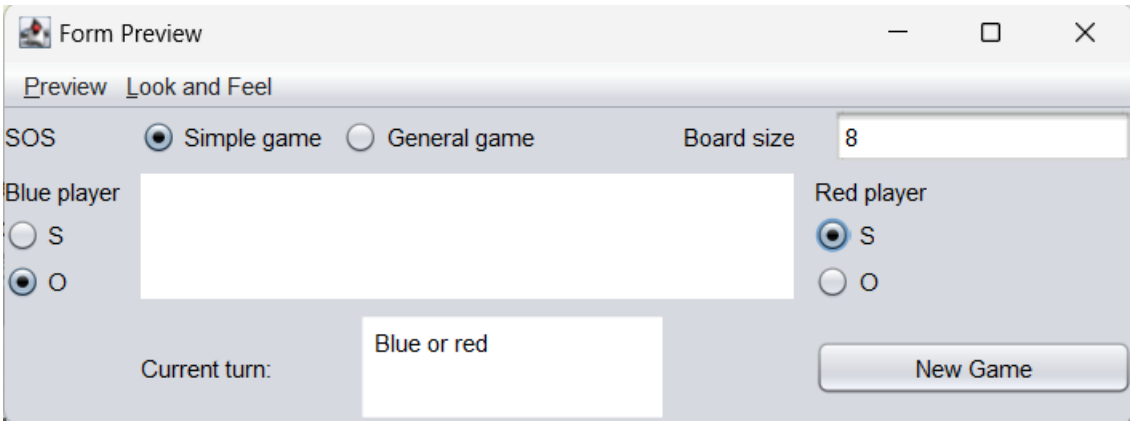


Código fuente del programa (adjunto a la carpeta: Sprint0_Junit)

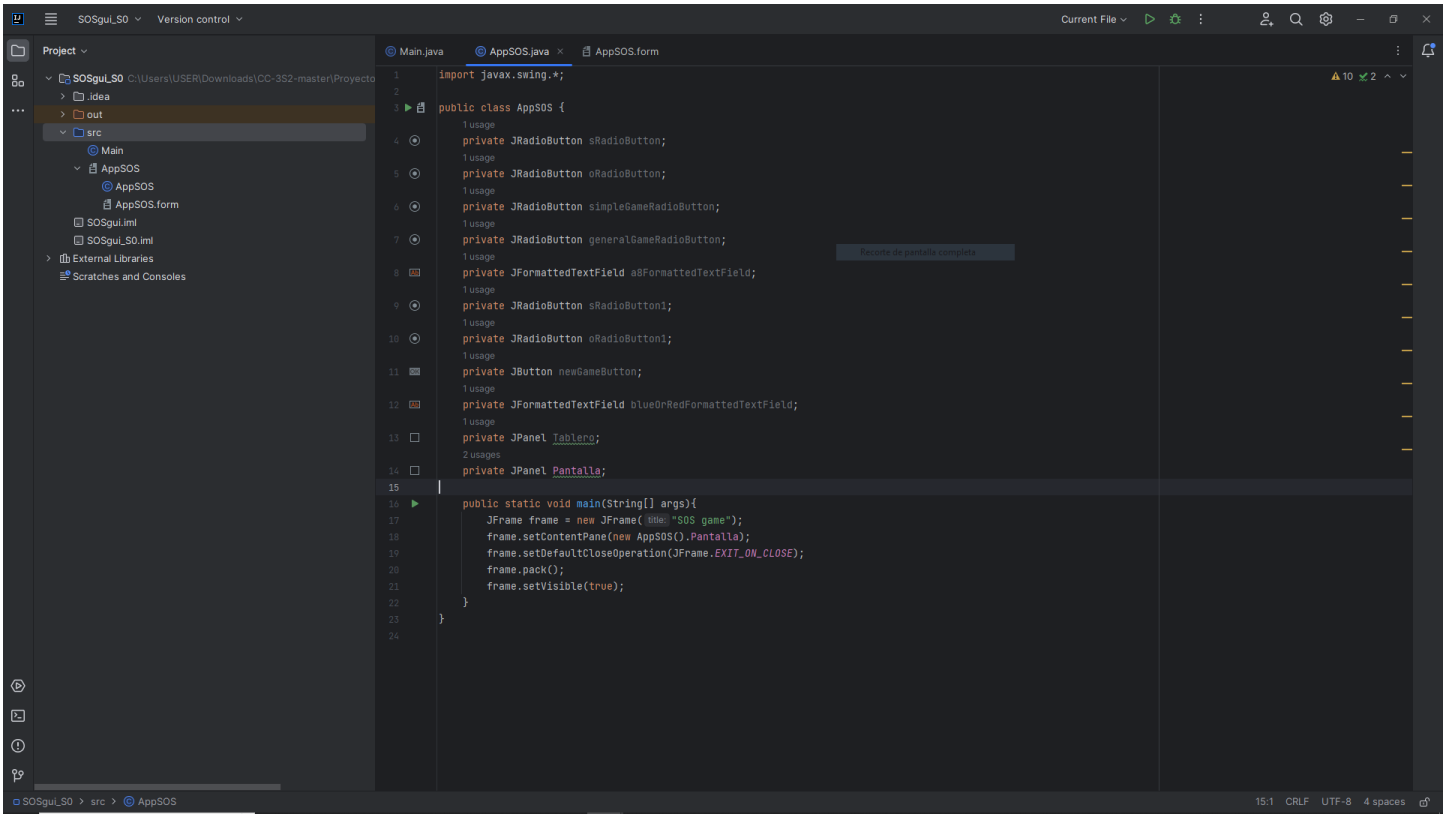
3. **Programación GUI (10 puntos)**

Escriban un programa GUI en el lenguaje que hayas elegido para tu proyecto SOS. La GUI de tu programa debe incluir texto, líneas, una casilla de verificación y botones de opción. Si bien se recomienda considerar la GUI para el tablero de juego SOS, no es obligatorio. En esta tarea, cualquier programa GUI de tu propio trabajo es aceptable.
Adjunten aquí (1) la captura de pantalla de la ejecución de tu programa y (2) el código fuente de tu programa.

Para esta primera parte del proyecto se muestra solamente la interfaz grafica del proyecto en GUI utilizando IntelliJ IDEA.



Código fuente del programa (adjunto a la carpeta: SOSgui_S0)


SOSgui_S0
Version control
Current File
Project
SOSgui_S0 C:\Users\USER\Downloads\CC-3S2-master\Proyecto
.idea
out
src
Main
AppSOS
AppSOS
AppSOS.form
SOSgui.iml
SOSgui_S0.iml
External Libraries
Scratches and Consoles
Main.java
AppSOS.java
AppSOS.form
Recorte de pantalla completa
import javax.swing.*;
public class AppSOS {
1 usage
private JRadioButton sRadioButton;
1 usage
private JRadioButton oRadioButton;
1 usage
private JRadioButton simpleGameRadioButton;
1 usage
private JRadioButton generalGameRadioButton;
1 usage
private JFormattedTextField a8FormattedTextField;
1 usage
private JRadioButton sRadioButton1;
1 usage
private JRadioButton oRadioButton1;
1 usage
private JButton newGameButton;
1 usage
private JFormattedTextField blueOrRedFormattedTextField;
1 usage
private JPanel Tablero;
2 usages
private JPanel Pantalla;
public static void main(String[] args){
JFrame frame = new JFrame(title: "SOS game");
frame.setContentPane(new AppSOS().Pantalla);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
}
}
SOSgui_S0 > src > AppSOS
15:1 CRLF UTF-8 4 spaces