Escuela Colombiana de Ingeniería
Procesos de Desarrollo de Software
Desarrollo Dirigido por Pruebas + DIP + DI + Contenedores Livianos

Maria Camila Fetecua – Daniel Alejandro Mejia

Parte I

- **Haga la especificación de los métodos calculateScore (de las tres variantes de GameScore), a partir de las especificaciones generales dadas anteriormente. Recuerde tener en cuenta: @pre, @pos, @param, @throws.**

```java
package hangman.model;

public interface GameScore {
    int calculateScore(int correctCount,int incorrectCount) throws HangmanException;
}
```

```java
package hangman.model;

public class OriginalScore implements GameScore{
    /**
     * @pre inicia con 100 puntos
     * @pos puntaje minimo 0
     * @param correctCount no se bonifican las letras correctas
     * @param incorrectCount se penaliza con 10 puntos cada letra incorrecta
     * @return el puntaje final
     * @throws HangmanException arroja una excepcion si los parametros son incorrectos
     */
    @Override
    public int calculateScore(int correctCount, int incorrectCount) throws HangmanException {
        return 0;
    }
}
```

```java
package hangman.model;

public class BonusScore implements GameScore {
    /**
     * @pre El juego inicia en 0 puntos
     * @param correctCount se bonifica con 10 puntos cada letra correcta
     * @param incorrectCount se penaliza con 5 puntos cada letra incorrecta
     * @return puntaje final
     * @pos el puntaje minimo es 0
     * @throws HangmanException no se sabe
     */
    @Override
    public int calculateScore(int correctCount, int incorrectCount) throws HangmanException {
        return 0;
    }
}
```

```java
package hangman.model;

public class PowerBonusScore implements GameScore{
    /**
     * @pre inicia con 0 puntos
     * @param correctCount la i-esima letra correcta se bonifica con 5^i
     * @param incorrectCount se penaliza con 8 puntos cada letra incorrecta
     * @return el puntaje final
     * @pos el puntaje minimo es 0, final es 500 puntos
     * @throws HangmanException no se sabe
     */
    @Override
    public int calculateScore(int correctCount, int incorrectCount) throws HangmanException {
        return 0;
    }
}
```

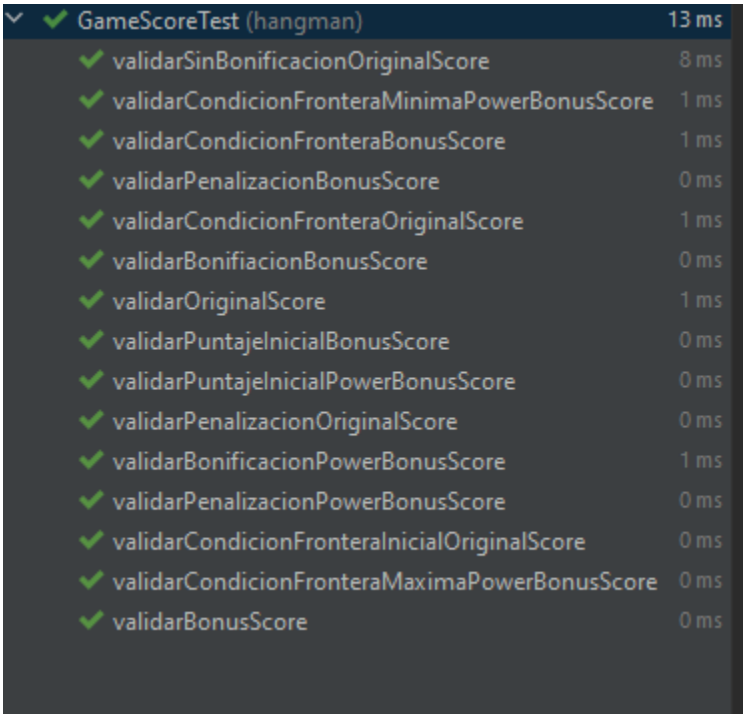
- **Actualice el archivo `pom.xml` e incluya las dependencias para la ultima versión de JUnit y la versión del compilador de Java a la versión 8.**

```xml
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.google.inject</groupId>
            <artifactId>guice</artifactId>
            <version>4.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

- **Teniendo en cuenta dichas especificaciones, en la clase donde se implementarán las pruebas (GameScoreTest), en los comentarios iniciales, especifique las clases de equivalencia para las tres variantes de GameScore, e identifique condiciones de frontera**

```java
import ...

/**
 * OriginalScore puntaje minimo 0, inicial 100, letras correctas sin bonificacion, penalizacion 10 puntos por letra incorrecta
 * BonusScore puntaje minimo 0, inicial 0, letras correctas 10 puntos de bonificacion, penalizacion 5 puntos por letra incorrecta
 * PowerBonusScore puntaje minimo 0, inicial 0, maximo 500, letras correctas con 5^i puntos de bonificacion, penalizacion 8 puntos por letra incorrecta
 */

public class GameScoreTest{
```

- **Realice la implementación de los 'cascarones' realizados anteriormente. Asegúrese que todas las pruebas unitarias creadas en los puntos anteriores se ejecutan satisfactoriamente.**



```java
package hangman.model;

public interface GameScore {
    int calculateScore(int correctCount,int incorrectCount);

    int getInitialScore();

}
```

```java
package hangman.model;

public class OriginalScore implements GameScore{
    private final int INITIAL_SCORE = 100;
    /**
     * @pre inicia con 100 puntos
     * @pos puntaje minimo 0
     * @param correctCount no se bonifican las letras correctas
     * @param incorrectCount se penaliza con 10 puntos cada letra incorrecta
     * @return el puntaje final
     */
    @Override
    public int calculateScore(int correctCount, int incorrectCount) {
        int score = 100;
        for(int i = 0; i < incorrectCount; i++){
            score -= 10;
            if(score < 0){
                score = 0;
            }
        }
        return score;
    }

    @Override
    public int getInitialScore() { return INITIAL_SCORE; }


}
```

```java
package hangman.model;

public class BonusScore implements GameScore {
    private final int INITIAL_SCORE = 0;
    /**
     * @pre El juego inicia en 0 puntos
     * @param correctCount se bonifica con 10 puntos cada letra correcta
     * @param incorrectCount se penaliza con 5 puntos cada letra incorrecta
     * @return puntaje final
     * @pos el puntaje minimo es 0
     */
    @Override
    public int calculateScore(int correctCount, int incorrectCount){
        int score = 0;
        for(int i = 0; i < correctCount; i++){
            score += 10;
        }
        for(int i = 0; i < incorrectCount; i++){
            score -= 5;
            if(score < 0){
                score = 0;
            }
        }
        return score;
    }

    @Override
    public int getInitialScore() { return INITIAL_SCORE; }
```

```java
package hangman.model;
import java.math.*;

public class PowerBonusScore implements GameScore{
    private final int INITIAL_SCORE = 0;
    /**
     * @pre inicia con 0 puntos
     * @param correctCount la i-esima letra correcta se bonifica con 5^i
     * @param incorrectCount se penaliza con 8 puntos cada letra incorrecta
     * @return el puntaje final
     * @pos el puntaje minimo es 0, final es 500 puntos
     */
    @Override
    public int calculateScore(int correctCount, int incorrectCount) {
        int score = 0;
        if (correctCount != 0){ score = (int)Math.pow(5,correctCount);}
        if(score > 500){ score = 500; }
        for(int i = 0; i < incorrectCount; i++){
            score -= 8;
            if(score < 0){ score = 0; }
        }
        return score;
    }

    @Override
    public int getInitialScore() {
        return INITIAL_SCORE;
    }
}
```

Parte II

1. **Utilizando el HangmanFactoryMethod (MétodoFabrica) incluya el OriginalScore a la configuración.**

```
abstract public class HangmanFactoryMethod {
    abstract public Language createLanguage();
    abstract public HangmanDictionary createDictionary();
    abstract public HangmanPanel createHangmanPanel();
    abstract public GameScore createGameScore();
}
```

```
public class HangmanDefaultFactoryMethod extends HangmanFactoryMethod {
    @Override
    public Language createLanguage() { return new English(); }

    @Override
    public HangmanDictionary createDictionary() { return new EnglishDictionaryDataSource(); }

    @Override
    public HangmanPanel createHangmanPanel() { return new HangmanStickmanPanel(); }

    @Override
    public GameScore createGameScore(){ return  new BonusScore() ;}

}
```

```
// Use Factory method
public GUI(HangmanFactoryMethod factoryMethod) {
    this.language = factoryMethod.createLanguage();
    this.dictionary = factoryMethod.createDictionary();
    this.hangmanPanel = factoryMethod.createHangmanPanel();
    this.gameScore = factoryMethod.createGameScore();
}
```

Incorpore el Contenedor Liviano Guice dentro del proyecto:

- **Revise las dependencias necesarias en el pom.xml.**
- **Modifique la inyección de dependencias utilizando guice en lugar del método fábrica..**
- **Configure la aplicación de manera que desde el programa SwingProject NO SE CONSTRUYA el Score directamente, sino a través de Guice, asi mismo como las otras dependencias que se están inyectando mediante la fabrica.**

```java
public static GUI createGUIUsingGuice() {
    Injector injector = Guice.createInjector(new HangmanFactoryServices());
    return injector.getInstance(GUI.class);
}

//method: main
//purpose: the entry-point to our application
public static void main(String[] args) {
    createGUIUsingGuice().play();
}
```

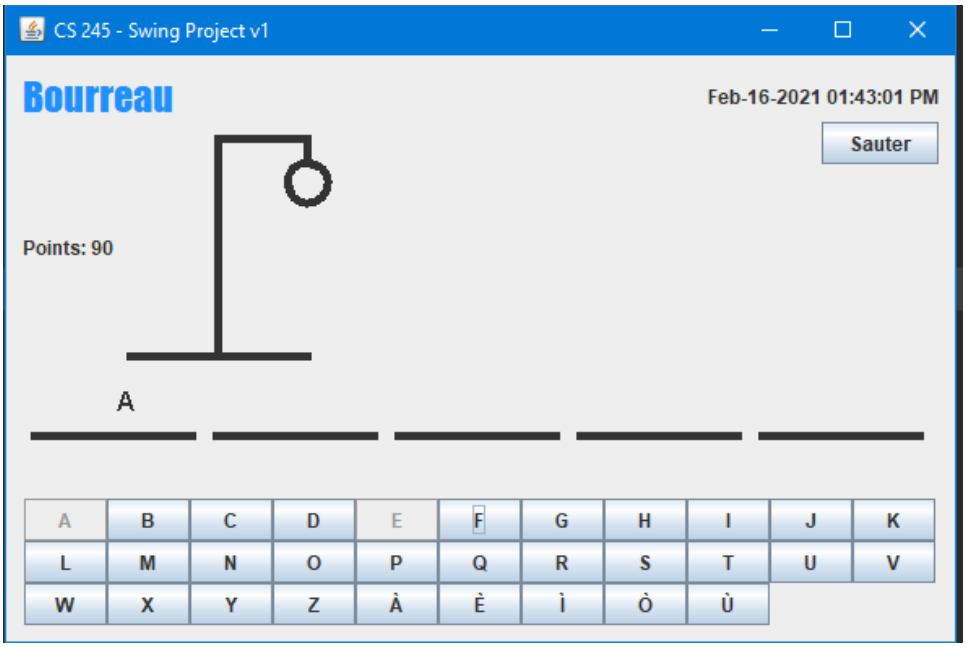
```java
public class HangmanFactoryServices extends com.google.inject.AbstractModule {

   @Override
   protected void configure() {
       /* Guice dependency injection */
       // bind(Interface.class).to(Concrete.class);
       bind(GameScore.class).to(PowerBonusScore.class);
       bind(Language.class).to(Spanish.class);
       bind(HangmanDictionary.class).to(SpanishDictionaryDataSource.class);
       bind(HangmanPanel.class).to(HangmanStickmanPanel.class);
   }

}
```

```java
@Inject
// Use Guice constructor
public GUI(Language language, HangmanDictionary dictionary, HangmanPanel hangmanPanel, GameScore gameScore){
    this.language = language;
    this.dictionary= dictionary;
    this.hangmanPanel = hangmanPanel;
    this.gameScore = gameScore;
}
```

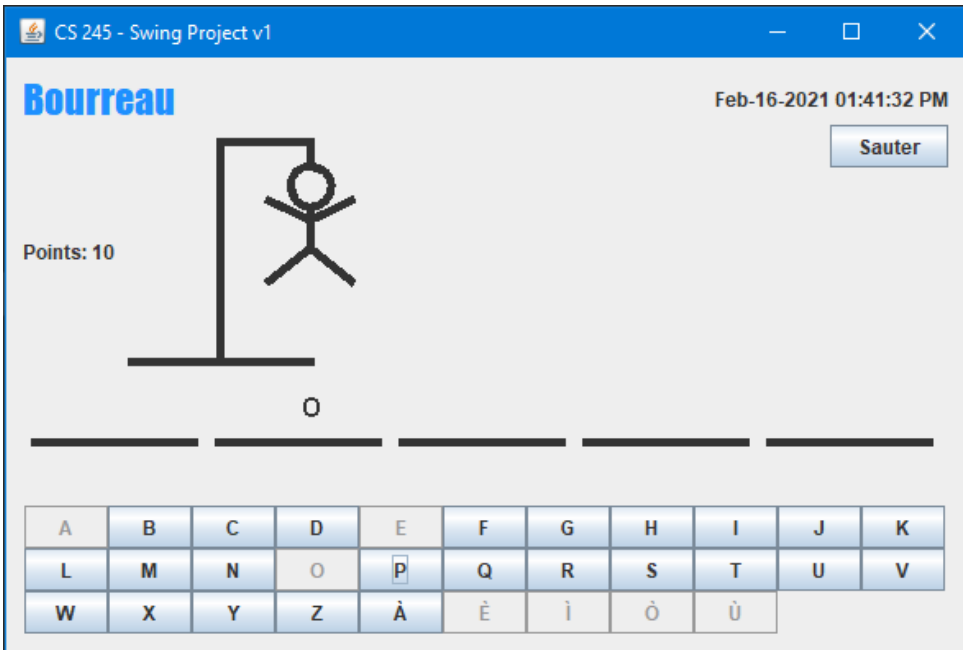
```java
GameModel gameModel = new GameModel(dictionary,gameScore);
```


```java
GameModel gameModel = new GameModel(dictionary,gameScore);
```


- **Mediante la configuración de la Inyección de Dependencias se pueda cambiar el comportamiento del mismo, por ejemplo:**
    - **Utilizar el esquema OriginalScore.**



- **Utilizar el esquema BonusScore.**



- **Utilizar el idioma francés.**
- **Utilizar el diccionario francés.**
- **etc...**