code
URJC

Ampliación de Ingeniería del Software

Colección de pruebas de evaluación

Universidad Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Francisco Gortázar
Correo: francisco.gortazar@urjc.es
Twitter: @fgortazar

Michel Maes
michel.maes@urjc.es
Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Índice de Pruebas de Evaluación

Prácticas

- Práctica 1 - Convocatoria Extraordinaria 2021
- Práctica 1 - Convocatoria Extraordinaria 2022
- Práctica 1 - Convocatoria Ordinaria 2021
- Práctica 1 - Convocatoria Ordinaria 2022
- Práctica 2 - Convocatoria Extraordinaria 2021
- Práctica 2 - Convocatoria Extraordinaria 2022
- Práctica 2 - Convocatoria Ordinaria 2021
- Práctica 2 - Convocatoria Ordinaria 2022
- Práctica 3 - Convocatoria Extraordinaria 2021
- Práctica 3 - Convocatoria Extraordinaria 2022
- Práctica 3 - Convocatoria Ordinaria 2021
- Práctica 3 - Convocatoria Ordinaria 2022

Índice de Pruebas de Evaluación

Exámenes

- Exámen Final (Móstoles) - Convocatoria Extraordinaria 2021
- Exámen Final (Móstoles) - Convocatoria Extraordinaria 2022
- Exámen Final (Móstoles) - Convocatoria Ordinaria 2021
- Exámen Final (Móstoles) - Convocatoria Ordinaria 2022
- Exámen Final (Vicálvaro) - Convocatoria Extraordinaria 2021
- Exámen Final (Vicálvaro) - Convocatoria Extraordinaria 2022
- Exámen Final (Vicálvaro) - Convocatoria Ordinaria 2021
- Exámen Final (Vicálvaro) - Convocatoria Ordinaria 2022

Práctica 1. Pruebas y calidad del software

Convocatoria Extraordinaria 2021

Enunciado

Se desean implementar ciertos controles de calidad de una aplicación que gestiona una librería online. Esta librería ofrece un interfaz web y una API REST para la gestión de libros. Se proporciona el código de dicha aplicación.

Importante (nuevo): Al crear un libro se lanzará un error si su descripción o su título están vacíos

El control de calidad se realizará mediante:

- Implementación de pruebas automáticas
- Análisis estático de código

Pruebas automáticas

Las pruebas automáticas que deben implementarse son de diferentes tipos:

- **Tests unitarios de la lógica de la aplicación (BookService)**
  - Comprobar que cuando se edita un libro (con 'title' y 'description' correctos) utilizando BookService, se guarda en el repositorio y se lanza una notificación
  - Comprobar que cuando se edita un libro (con 'title' o 'description' en blanco) utilizando BookService, NO se guarda en el repositorio y NO se lanza una notificación

- **Tests E2E de la API REST (RESTAssured)**
  - Comprobar que al añadir un nuevo libro y editarlo para que su título termine en v2, al pedir todos los libros, el libro (editado) se encuentra entre ellos.

- **Tests E2E de la interfaz web (Selenium)**
  - Comprobar que al añadir un nuevo libro, al editarlo para que su título termine en v2, el libro editado aparece en la página principal

Consideraciones:

- Los tests deben ser independientes entre sí (no depender de información que otros tests hayan creado o eliminado). Además de ser un anti-patrón, JUnit no garantiza el orden de ejecución. Por tanto se deberán crear los recursos necesarios antes de ejecutar el/los tests.

- Se valorará la modularización de los tests en paquetes o clases diferentes, dado que son de diferente naturaleza.
- En los test unitarios, es obligatorio el uso de Mocks en todas las dependencias. Por ejemplo, la persistencia se realiza utilizando una base de datos H2 (y queremos evitarlo en este tipo de test), además de hacer las comprobaciones sobre sus llamadas.
- Puede modificarse el HTML proporcionado para añadir únicamente parámetros (como id 's) a las etiquetas HTML que faciliten los test de Selenium.
- Se valorará positivamente que no se repita código, haciendo uso de métodos auxiliares y/o métodos anotados con @BeforeEach @AfterEach

Análisis estático de código

Se debe analizar el código usando Sonarqube y analizar los issues reportados por la herramienta:

- Los issues de tipo “Vulnerability” deben ser corregidos. Para ello, se usará la información reportada por la herramienta sobre el issue como punto de partida para buscar una solución a la misma.
- Los issues de tipo “Code Smell” deben ser analizados. Si se considera que son un “falso positivo” porque el código es correcto no se corregirán y se marcarán en Sonarqube como “Resolve as false positive”. Si se considera que son un issue real, será corregido.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2020-z.gonzalez2020.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá describir el funcionamiento de los tests implementados, de forma que un desarrollador que no conozca los tests pueda entender qué hacen los tests.
- Se debe describir la estrategia seguida para resolver los issues de tipo “Vulnerability” reportados por Sonarqube indicando las páginas en las que se ha obtenido información.

- Se debe describir qué valoración se ha hecho de los issues de tipo “Code Smell”. Si se decide no resolverlos, hay que explicar el motivo. Si se decide resolverlos, hay que explicar cómo se ha hecho.
- La memoria deberá tener una longitud de 4 o 5 páginas (incluyendo la página de la portada).

Práctica 1. Testing en Java y Spring

Convocatoria Extraordinaria 2022

Enunciado

Se desean implementar ciertos controles de calidad de la aplicación ***Nitflex***, que gestiona películas. Esta aplicación ofrece un interfaz web y una API REST para la gestión de películas. Se proporciona el código de dicha aplicación en el aula virtual.

Las pruebas automáticas que deben implementarse son de diferentes tipos:

- **Tests unitarios de la lógica de la aplicación (FilmService) (2 puntos)**
  Queremos comprobar que**:**
  - Cuando se guarda una película (con una URL incorrecta) utilizando en FilmService, NO se guarda en el repositorio y NO se lanza una notificación
  - Cuando se guarda una película (con una URL correcta) y un título vacío utilizando FilmService, NO se guarda en el repositorio y NO se lanza una notificación

- **Tests de integración (UrlUtils) (2 puntos)**
  Queremos comprobar que**:**
  - Cuando una URL tiene el formato correcto y NO es una imagen, debemos dar la URL por inválida (Ej: ***https://www.themoviedb.org/***)
  - Cuando una URL tiene el formato correcto, es una imagen y NO existe debemos dar la URL por inválida (Ej: ***https://www.themoviedb.org/image.png***)

- **Tests E2E de la API REST (RESTAssured) (3 puntos)**
  Queremos comprobar que**:**
  - Al añadir una película con una URL no válida nos devuelve un error
  - Al editar una película añadiendole " - *Parte 2*" en el título, la modificación se realiza correctamente.

- **Tests E2E de la interfaz web (Selenium) (3 puntos)**
  Queremos comprobar que**:**
  - Añadir una nueva película con una URL inválida y comprobar que da un mensaje de error
  - Al crear una película y editarla para añadir '- parte 2' en su titulo, el cambio se ha aplicado

Consideraciones:

- Los tests deben ser independientes entre sí (no depender de información que otros tests hayan creado o eliminado). Además de ser un anti-patrón, JUnit no garantiza el orden de ejecución. Por tanto se deberán crear los recursos necesarios antes de ejecutar el/los tests.
- No se deben utilizar las peliculas creadas por defecto por la aplicación
- Para los test que requieran tener la aplicación disponible, deberán ser los test los que lancen la aplicación.
- Se valorará la modularización de los tests en paquetes o clases diferentes, dado que son de diferente naturaleza.
- Si una funcionalidad debe lanzar un error, se deben hacer las comprobaciones pertinentes sobre que el error se lanza con el mensaje esperado
- De forma similar, si hay que comprobar que una funcionalidad no debe lanzar un error, se puede utilizar la aserción `assertDoesNotThrow(...)`.
- En los test unitarios, es obligatorio el uso de Mocks en todas las dependencias. Por ejemplo, la persistencia se realiza utilizando una base de datos H2 (y queremos evitarlo en este tipo de test). Se deben comprobar las llamadas a métodos de las clases mockeadas.
- Puede modificarse el HTML proporcionado para añadir únicamente parámetros (como id 's) a las etiquetas HTML que faciliten los test de Selenium.
- Se valorará especialmente que los test estén escritos en inglés y se penalizará utilizar variables o nombres de clases con Ñ o acentos.
- Se valorará positivamente el uso de @DisplayName para dar más semántica a los test.
- **Importante:** El código debe compilar y todos los test deben pasar al ejecutar el comando "**mvn test**". Los test que no pasen, no puntuarán.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- **No se garantiza que se respondan dudas de la práctica 48 horas antes de su entrega.**
- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- **No será necesario realizar una memoria**
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2021-z.gonzalez2021.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Práctica 1. Pruebas y calidad del software

Convocatoria Ordinaria 2021

Enunciado

Se desean implementar ciertos controles de calidad de una aplicación que gestiona una librería online. Esta librería ofrece un interfaz web y una API REST para la gestión de libros. Se proporciona el código de dicha aplicación.

El control de calidad se realizará mediante:

- Implementación de pruebas automáticas
- Análisis estático de código

Pruebas automáticas

Las pruebas automáticas que deben implementarse son de diferentes tipos:

- **Tests unitarios de la lógica de la aplicación (BookService)**
  - Comprobar que cuando se guarda un libro utilizando BookService, se guarda en el repositorio y se lanza una notificación
  - Comprobar que cuando se borra un libro utilizando BookService, se elimina del repositorio y se lanza una notificación

- **Tests E2E de la API REST (RESTAssured)**
  - Comprobar que al añadir un libro podemos recuperarlo
  - Comprobar que al borrar un libro no podemos recuperarlo.

- **Tests E2E de la interfaz web (Selenium)**
  - Añadir un nuevo libro y comprobar que se ha creado
  - Borrar un libro y comprobar que no existe

Consideraciones:

- Los tests deben ser independientes entre sí (no depender de información que otros tests hayan creado o eliminado). Además de ser un anti-patrón, JUnit no garantiza el orden de ejecución. Por tanto se deberán crear los recursos necesarios antes de ejecutar el/los tests.
- Se valorará la modularización de los tests en paquetes o clases diferentes, dado que son de diferente naturaleza.
- En los test unitarios, es obligatorio el uso de Mocks en todas las dependencias. Por ejemplo, la persistencia se realiza utilizando una base de datos H2 (y queremos evitarlo en este tipo de test).
- Puede modificarse el HTML proporcionado para añadir únicamente parámetros (como id 's) a las etiquetas HTML que faciliten los test de Selenium.

Análisis estático de código

Se debe analizar el código usando Sonarqube y analizar los issues reportados por la herramienta:

- Los issues de tipo “Vulnerability” deben ser corregidos. Para ello, se usará la información reportada por la herramienta sobre el issue como punto de partida para buscar una solución a la misma.
- Los issues de tipo “Code Smell” deben ser analizados. Si se considera que son un “falso positivo” porque el código es correcto no se corregirán y se marcarán en Sonarqube como “Resolve as false positive”. Si se considera que son un issue real, será corregido.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2020-z.gonzalez2020.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá describir el funcionamiento de los tests implementados, de forma que un desarrollador que no conozca los tests pueda entender qué hacen los tests.
- Se debe describir la estrategia seguida para resolver los issues de tipo “Vulnerability” reportados por Sonarqube indicando las páginas en las que se ha obtenido información.
- Se debe describir qué valoración se ha hecho de los issues de tipo “Code Smell”. Si se decide no resolverlos, hay que explicar el motivo. Si se decide resolverlos, hay que explicar cómo se ha hecho.
- La memoria deberá tener una longitud de 4 o 5 páginas (incluyendo la página de la portada).

Práctica 1. Testing en Java y Spring

Convocatoria Ordinaria 2022

Enunciado

Se desean implementar ciertos controles de calidad de la aplicación ***Nitflex***, que gestiona películas. Esta aplicación ofrece un interfaz web y una API REST para la gestión de películas. Se proporciona el código de dicha aplicación en el aula virtual.

Las pruebas automáticas que deben implementarse son de diferentes tipos:

- **Tests unitarios de la lógica de la aplicación (FilmService) (2 puntos)**
  Queremos comprobar que**:**
  - Cuando se guarda una película (con una URL correcta) utilizando FilmService, se guarda en el repositorio y se lanza una notificación
  - Cuando se borra una película utilizando FilmService, se elimina del repositorio y se lanza una notificación

- **Tests de integración (UrlUtils) (2 puntos)**
  Queremos comprobar que**:**
  - Cuando una URL NO tiene el formato correcto, `UrlUtils` debe dar la URL por inválida (Ej: ***esto-no-es-una-url***)
  - Cuando una URL tiene el formato correcto, es una imagen y existe, `UrlUtils` debe dar la URL por válida (Ej: ***https://www.urjc.es/images/Logos/logo-urjc-25.png***)

- **Tests E2E de la API REST (RESTAssured) (2 puntos)**
  Queremos comprobar que**:**
  - Al añadir una película podemos recuperarla
  - Al borrar una película no podemos recuperarla

- **Tests E2E de la interfaz web (Selenium) (3 puntos)**
  Queremos comprobar que**:**
  - Al añadir una nueva película podemos comprobar que se ha creado
  - Al borrar una película, vemos el mensaje de borrado y podemos comprobar que la película ya no está en la página principal

Extra: Nos informan de que un cliente ha notificado un bug en la aplicación. Deberemos solucionarlo siguiendo los siguientes pasos **(1 punto)**:

- Localizar el bug
- Escribir un test que detecte el bug
- Ejecutar el test (debe fallar)

- Arreglar el fallo
- Ejecutar el test de nuevo (debe pasar)

Consideraciones:

- Los tests deben ser independientes entre sí (no depender de información que otros tests hayan creado o eliminado). Además de ser un anti-patrón, JUnit no garantiza el orden de ejecución. Por tanto se deberán crear los recursos necesarios antes de ejecutar el/los tests.
- Se valorará la modularización de los tests en paquetes o clases diferentes, dado que son de diferente naturaleza.
- Si una funcionalidad debe lanzar un error, se deben hacer las comprobaciones pertinentes sobre que el error se lanza con el mensaje esperado
- De forma similar, si hay que comprobar que una funcionalidad no debe lanzar un error, se puede utilizar la aserción `assertDoesNotThrow(...)`.
- En los test unitarios, es obligatorio el uso de Mocks en todas las dependencias. Por ejemplo, la persistencia se realiza utilizando una base de datos H2 (y queremos evitarlo en este tipo de test).
- Puede modificarse el HTML proporcionado para añadir únicamente parámetros (como id 's) a las etiquetas HTML que faciliten los test de Selenium.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- **No se garantiza que se respondan dudas de la práctica 48 horas antes de su entrega.**
- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2021-z.gonzalez2021.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá describir el funcionamiento de los tests implementados, de forma que un desarrollador que no conozca los tests pueda entender qué hacen los tests.
- En caso de realizar el punto extra, se deberá detallar en la memoria cada uno de los pasos (con capturas de pantalla cuando sea necesario)

- La memoria deberá tener una longitud de 4 o 5 páginas (incluyendo la página de la portada).

Práctica 2. Test Driven Development

Convocatoria Extraordinaria 2021

Enunciado

Se desea ampliar la funcionalidad de la aplicación que gestiona la librería online utilizada en la Práctica 1.

Los dueños de la librería nos han pedido que las descripciones de los libros se guarden en la base de datos separada por líneas con un número de caracteres máximo en cada línea. Es decir, como ocurre con un procesador de textos cuando se llega al final de la línea, que se empieza una nueva. Las palabras no se parten, si no cabe la última palabra de la línea, pasa a la línea siguiente. No obstante, si la línea es tan pequeña que una palabra no cabe, esa palabra se partirá. Para que se sepa que la palabra se ha partido, se deberá poner un guión como último carácter de cada línea donde se parte una palabra larga.

Además, nos han pedido que que hagamos un procesamiento de los espacios de la siguiente forma:

- Sustituir varios espacios consecutivos entre dos palabras por un único espacio.
- No permitir espacios al final de una línea.
- No permitir espacios al principio de una línea.

El cliente no tiene claro todavía qué longitud de línea establecer, así que nos han pedido que la aplicación esté preparada para cualquier tamaño de línea, aunque de momento será de 10. Nunca será menor que 2.

La práctica consiste en implementar un servicio Spring llamado `LineBreaker` que divida en líneas de una longitud indicada como parámetro un texto pasado también como parámetro:

```
@Service
public class LineBreaker {
  public String breakLine(String text, int lineLength){
    ...
  }
}
```

Este servicio debe usarse desde el `BookService` para separar en líneas la descripción del libro antes de ser guardado en la base de datos.

```
public Book save(Book book) {

  newBook.setDescription(lineBreaker.breakLine(newBook.getDescription(),10));
```

```
  Book newBook = repository.save(book);
  notificationService.notify("Book Event: book with title="+
    newBook.getTitle()+" was created");
  return newBook;
}
```

Para la implementación del servicio LineBreaker debe usarse la técnica del Desarrollo Guiado por Pruebas o *Test Driven Development*.

Tal y como se indica en el material de la asignatura, lo primero que hay que tener para poder aplicar TDD es una lista ordenada con diferentes ejemplos concretos que muestren la funcionalidad que se quiere implementar. Estos ejemplos irán creciendo en complejidad. Una vez tengamos los ejemplos, podremos seguir los pasos de TDD:

- 1) Seleccionar el primer ejemplo.
- 2) Implementar el ejemplo como un test.
- 3) Verificar que el test falla (porque está sin implementar)
- 4) Implementar el código más sencillo posible para que el test pase (se ponga verde).
- 5) Refactorizar el código para que tenga calidad (sin código duplicado, nombres adecuados, comprensible, etc).
- 6) Si hay más ejemplos, volver al paso 2 con el siguiente ejemplo.

A continuación se especifican algunos ejemplos para que se entienda bien cómo debe implementarse la funcionalidad de dividir líneas.

Texto original	**Tamaño línea**	**Resultado al dividir**
`""`	2	`""`
`"test"`	4	`"test"`
`"test"`	5	`"test"`
`"test test"`	4	`"test\ntest"`
`"test test"`	5	`"test\ntest"`
`"test test"`	6	`"test\ntest"`
`"test test test test"`	9	`"test test\ntest test"`
`"test  test"`	4	`"test\ntest"`
`"test   test"`	6	`"test\ntest"`
`"testtest"`	5	`"test-\ntest"`
`"testtesttest"`	5	`"test-\ntest-\ntest"`
`"test test"`	3	`"te-\nst\nte-\nst"`
`"test 1234567 test"`	6	`"test\n12345-\n67\ntest"`
`"123456789"`	3	`"12-\n34-\n56-\n789"`

Para aplicar TDD debe utilizarse esta lista de ejemplos y en el orden en que se especifican. Si el alumno considera que quedan casos por cubrir y que su código todavía no contempla, o bien quiere estar seguro de que su código gestiona correctamente otras situaciones que no tienen tests, puede añadir más ejemplos al final de la lista.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2020-z.gonzalez2020.zip).
- En el fichero `pom.xml` se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos, igual que el nombre del fichero zip):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Se puede partir del código que los alumnos hayan entregado en la práctica 1. Aquellos alumnos que no hayan entregado la práctica 1 pueden usar como base el código que acompañaba al enunciado de la práctica 1.

Se debe entregar el código fuente con la funcionalidad implementada y los tests. Es posible que los tests de la práctica 1 tengan que ser modificados para que sigan pasando al incluir la nueva funcionalidad.

Además, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá incluir la siguiente información **por cada ciclo TDD** que se haya realizado:
    - Código del test.
    - Mensaje de error obtenido al fallar el test.
    - Implementación que hace pasar el test.
    - Implementación después de refactorizar.
- No hay límite de páginas.

Práctica 2. Test Driven Development

Convocatoria Extraordinaria 2022

Enunciado

Se desea implementar en Java la clase **StringCalculator** para *parsear* expresiones de sumas. Esta clase debe tener un método add() que recibe una expresión matemática en forma de String y debe devolver su resultado como un número, **siguiendo los pasos que se especifican a continuación**

1. Crearemos una calculadora que sume **hasta dos números** separados por comas.
    a. Cadena vacía devolverá 0
    b. Un único número devolera su propio valor
    c. “1,2” devolverá 3
2. Permitiremos que el método add maneje una cantidad ilimitada de números.
    // "1,2,3,4,5,6,7" -> 28 El método debe devolver la suma de N números

3. Permitiremos que el método add maneje nuevas líneas entre números.
    // "1\n2,3" -> 6 El método debe devolver la suma de los dos números separados por comas o saltos de linea

4. Soportaremos diferentes delimitadores.
    // "//;\n1;2" -> 3 El método debe soportar cualquier separador de números con el siguiente formato "//[delimiter]\n[numbers…]"

5. Al llamar al método add con un número negativo lanzará una excepción. Si hay varios números negativos muestrelos todos en la excepción.
    // "-1,2" -> "negatives not allowed: -1" El método debe devolver un mensaje de error (excepción) si algún número negativo es encontrado, indicándolo

6. Los números mayores de 1000 deben ser ignorados.
    // "1000,2" -> 2 El método debe ignorar números mayores que 1000

7. Los delimitadores pueden ser de cualquier longitud con el siguiente formato "//[delimitador]\n".
    // "//[***]\n1***2***3" -> 6 El método debe soportar cualquier separador de números (de más de un char) con el siguiente formato "//[delimiter]\n[numbers…]"

8. Permitir múltiples delimitadores
    // "//[*][%]\n1*2%3" -> 6 El método debe soportar varios separadores de números con el siguiente formato "//[delim1][delim2]\n[numbers…]"

9. Permitir múltiples delimitadores con una longitud superior a un caracter

// "//[**][%%]\n1**2%%3" -> 6 El método debe soportar varios separadores de números (de más de un char) con el siguiente formato "//[delim1][delim2]\n[numbers…]"

Para la implementación esta clase debe usarse la técnica del Desarrollo Guiado por Pruebas o *Test Driven Development*.

Tal y como se indica en el material de la asignatura, lo primero que hay que tener para poder aplicar TDD es una lista ordenada con diferentes ejemplos concretos que muestren la funcionalidad que se quiere implementar. Estos ejemplos irán creciendo en complejidad. Una vez tengamos los ejemplos, podremos seguir los pasos de TDD:

- 1) Seleccionar el primer ejemplo.
- 2) Implementar el ejemplo como un test.
- 3) Verificar que el test falla (porque está sin implementar)
- 4) Implementar el código más sencillo posible para que el test pase (se ponga verde).
- 5) Refactorizar el código para que tenga calidad (sin código duplicado, nombres adecuados, comprensible, etc).
- 6) Si hay más ejemplos, volver al paso 2 con el siguiente ejemplo.

Para aplicar TDD debe utilizarse la lista de ejemplos y en el orden en que se especifican. Si el alumno considera que quedan casos por cubrir y que su código todavía no contempla, o bien quiere estar seguro de que su código gestiona correctamente otras situaciones que no tienen tests, puede añadir más ejemplos al final de la lista.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- **No se garantiza que se respondan dudas de la práctica 48 horas antes de su entrega.**
- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2021-z.gonzalez2021.zip).
- En el fichero `pom.xml` se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos, igual que el nombre del fichero zip):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Se debe entregar el código fuente con la funcionalidad implementada y los tests.

Además, se deberá elaborar una memoria explicativa del mismo **en formato PDF**.

- Deberá guardarse en la carpeta raíz del proyecto.

- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá incluir la siguiente información **por cada ciclo TDD** que se haya realizado:
    - Código del test.
    - Mensaje de error obtenido al fallar el test.
    - Implementación que hace pasar el test.
    - Implementación después de refactorizar.
- No hay límite de páginas.

Práctica 2. Test Driven Development

Convocatoria Ordinaria 2021

Enunciado

Se desea ampliar la funcionalidad de la aplicación que gestiona la librería online utilizada en la Práctica 1.

Los dueños de la librería nos han pedido que las descripciones de los libros se guarden en la base de datos separada por líneas con un número de caracteres máximo en cada línea. Es decir, como ocurre con un procesador de textos cuando se llega al final de la línea, que se empieza una nueva. Las palabras no se parten, si no cabe la última palabra de la línea, pasa a la línea siguiente. No obstante, si la línea es tan pequeña que una palabra no cabe, esa palabra se partirá. Para que se sepa que la palabra se ha partido, se deberá poner un guión como último carácter de cada línea donde se parte una palabra larga.

Además, nos han pedido que que hagamos un procesamiento de los espacios de la siguiente forma:

- Sustituir varios espacios consecutivos entre dos palabras por un único espacio.
- No permitir espacios al final de una línea.
- No permitir espacios al principio de una línea.

El cliente no tiene claro todavía qué longitud de línea establecer, así que nos han pedido que la aplicación esté preparada para cualquier tamaño de línea, aunque de momento será de 10. Nunca será menor que 2.

La práctica consiste en implementar un servicio Spring llamado `LineBreaker` que divida en líneas de una longitud indicada como parámetro un texto pasado también como parámetro:

```
@Service
public class LineBreaker {
  public String breakLine(String text, int lineLength){
    ...
  }
}
```

Este servicio debe usarse desde el `BookService` para separar en líneas la descripción del libro antes de ser guardado en la base de datos.

```
public Book save(Book book) {

  newBook.setDescription(lineBreaker.breakLine(newBook.getDescription(),10));
```

```
  Book newBook = repository.save(book);
  notificationService.notify("Book Event: book with title="+
    newBook.getTitle()+" was created");
  return newBook;
}
```

Para la implementación del servicio LineBreaker debe usarse la técnica del Desarrollo Guiado por Pruebas o *Test Driven Development*.

Tal y como se indica en el material de la asignatura, lo primero que hay que tener para poder aplicar TDD es una lista ordenada con diferentes ejemplos concretos que muestren la funcionalidad que se quiere implementar. Estos ejemplos irán creciendo en complejidad. Una vez tengamos los ejemplos, podremos seguir los pasos de TDD:

- 1) Seleccionar el primer ejemplo.
- 2) Implementar el ejemplo como un test.
- 3) Verificar que el test falla (porque está sin implementar)
- 4) Implementar el código más sencillo posible para que el test pase (se ponga verde).
- 5) Refactorizar el código para que tenga calidad (sin código duplicado, nombres adecuados, comprensible, etc).
- 6) Si hay más ejemplos, volver al paso 2 con el siguiente ejemplo.

A continuación se especifican algunos ejemplos para que se entienda bien cómo debe implementarse la funcionalidad de dividir líneas.

Texto original	**Tamaño línea**	**Resultado al dividir**
`""`	2	`""`
`"test"`	4	`"test"`
`"test"`	5	`"test"`
`"test test"`	4	`"test\ntest"`
`"test test"`	5	`"test\ntest"`
`"test test"`	6	`"test\ntest"`
`"test test test test"`	9	`"test test\ntest test"`
`"test  test"`	4	`"test\ntest"`
`"test   test"`	6	`"test\ntest"`
`"testtest"`	5	`"test-\ntest"`
`"testtesttest"`	5	`"test-\ntest-\ntest"`
`"test test"`	3	`"te-\nst\nte-\nst"`
`"test 1234567 test"`	6	`"test\n12345-\n67\ntest"`
`"123456789"`	3	`"12-\n34-\n56-\n789"`

Para aplicar TDD debe utilizarse esta lista de ejemplos y en el orden en que se especifican. Si el alumno considera que quedan casos por cubrir y que su código todavía no contempla, o bien quiere estar seguro de que su código gestiona correctamente otras situaciones que no tienen tests, puede añadir más ejemplos al final de la lista.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2020-z.gonzalez2020.zip).
- En el fichero `pom.xml` se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos, igual que el nombre del fichero zip):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Se puede partir del código que los alumnos hayan entregado en la práctica 1. Aquellos alumnos que no hayan entregado la práctica 1 pueden usar como base el código que acompañaba al enunciado de la práctica 1.

Se debe entregar el código fuente con la funcionalidad implementada y los tests. Es posible que los tests de la práctica 1 tengan que ser modificados para que sigan pasando al incluir la nueva funcionalidad.

Además, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá incluir la siguiente información **por cada ciclo TDD** que se haya realizado:
    - Código del test.
    - Mensaje de error obtenido al fallar el test.
    - Implementación que hace pasar el test.
    - Implementación después de refactorizar.
- No hay límite de páginas.

Práctica 2. Test Driven Development

Convocatoria Ordinaria 2022

Enunciado

Se desea implementar en Java la clase **CalculatorParser** para *parsear* expresiones matemáticas. Esta clase debe tener un método parse() que recibe una expresión matemática en forma de String y debe devolver su resultado como un número.

Ejemplo:

```
CalculatorParser calculator = new CalculatorParser();
calculator.parse(“1 + 2 + 3”)
=> 6
```

Para la implementación esta clase debe usarse la técnica del Desarrollo Guiado por Pruebas o *Test Driven Development*.

Tal y como se indica en el material de la asignatura, lo primero que hay que tener para poder aplicar TDD es una lista ordenada con diferentes ejemplos concretos que muestren la funcionalidad que se quiere implementar. Estos ejemplos irán creciendo en complejidad. Una vez tengamos los ejemplos, podremos seguir los pasos de TDD:

- 1) Seleccionar el primer ejemplo.
- 2) Implementar el ejemplo como un test.
- 3) Verificar que el test falla (porque está sin implementar)
- 4) Implementar el código más sencillo posible para que el test pase (se ponga verde).
- 5) Refactorizar el código para que tenga calidad (sin código duplicado, nombres adecuados, comprensible, etc).
- 6) Si hay más ejemplos, volver al paso 2 con el siguiente ejemplo.

A continuación se especifican algunos ejemplos para que se entienda bien cómo debe implementarse la función parse:

N	Expresión	Resultado	Comentario
1	"1 + 1"	2	Se incluye la suma
2	"2 + 3 + 4"	9	
3	"54 + 23"	77	
4	"5 - 3"	2	Se incluye la resta
5	"5 - 9"	-4	
6	"-5 + 9"	4	Se incluyen números negativos
7	"7 + 1 - 5"	3	
8	"9 - 5 + 4"	8	
9	"9 + 1 - 6 - 2"	2	
10	"3 x 2"	6	Se incluye la multiplicación
11	"3 x 4 x 2"	24	
12	"3 x 2 + 7"	13	Se incluye multiplicación y suma (Mult > Sum/Res)
13	"3 x 2 - 7"	-1	
14	"4 + 3 x 5"	19	
15	"-5 + 6 x 2"	7	
16	"9 / 3"	3	Se incluye la división
17	"5 / 2"	2.5	
18	"14 / 2 + 5"	12.0	
19	"5 x 6 / 8 x 2"	7.5	
20	"3 x 4 + 2 - 7 / 3"	11.66	Se incluyen números decimales periódicos

Para aplicar TDD debe utilizarse esta lista de ejemplos y en el orden en que se especifican. Si el alumno considera que quedan casos por cubrir y que su código todavía no contempla, o bien quiere estar seguro de que su código gestiona correctamente otras situaciones que no tienen tests, puede añadir más ejemplos al final de la lista.

Evaluación: Para aprobar será necesario implementar, como mínimo, los 9 primeros ejemplos. La nota máxima que se puede obtener con estos ejemplos es de un 9 sobre 10. Para alcanzar la máxima calificación, se deberán completar los 20 ejemplos siguiendo TDD.

Formato de entrega

La práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- **No se garantiza que se respondan dudas de la práctica 48 horas antes de su entrega.**
- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del proyecto Maven. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf2021-z.gonzalez2021.zip).
- En el fichero `pom.xml` se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos, igual que el nombre del fichero zip):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Se debe entregar el código fuente con la funcionalidad implementada y los tests.

Además, se deberá elaborar una memoria explicativa del mismo **en formato PDF**.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá incluir la siguiente información **por cada ciclo TDD** que se haya realizado:
    - Código del test.
    - Mensaje de error obtenido al fallar el test.
    - Implementación que hace pasar el test.
    - Implementación después de refactorizar.
- No hay límite de páginas.

Práctica 3. Integración, entrega y despliegue continuo

Convocatoria Extraordinaria 2021

Enunciado

Se desean implementar ciertos controles de calidad de una aplicación que gestiona una librería online. Esta librería ofrece un interfaz web y una API REST para la gestión de libros, con sus correspondientes pruebas unitarias, de API REST y Selenium. Se proporciona el código de dicha aplicación, que podrá obtenerse como se indica más abajo.

El control de calidad se realizará mediante los siguientes pasos ejecutados en este orden:

1. Implantar el modelo de desarrollo **Trunk Based Development**
2. Definición de workflows que automaticen la ejecución de pruebas, la publicación de releases y el despliegue en producción de la aplicación, tal como se describe en la sección correspondiente. Estos workflows deben funcionar correctamente antes de pasar al paso 3 (**NOTA:** No hay que implementar los workflows siguiendo TBD)
3. Desarrollo de una funcionalidad nueva utilizando **Trunk Based Development**, que se describe más abajo en su propia sección.

Preparación del repositorio inicial

Para realizar la práctica, se proporciona el código inicial de una aplicación que contiene pruebas unitarias y pruebas extremo a extremo de dos tipos: Selenium y pruebas de API REST.

Cada alumno (o grupo de dos alumnos) deberá crear un repositorio de código en GitHub que:

- Use como plantilla el siguiente repositorio: https://github.com/AIS-Mostoles-2021/ais-practica-3-base (click en “Use this template” y seleccionar que se cree en tu cuenta de usuario)
- Debe ser privado
- Debe nombrarse cómo **ais-usuario-urjc-2021**
    - Ejemplo: ais-p.perez-2021
- Debe incluir al profesor Michel Maes (Usuario en Github: **Maes95**) invitándole al repositorio

Consideraciones:

- Los repositorios privados están limitados en cuanto a los recursos que pueden consumir, concretamente:
  - Permite un máximo de 2.000 minutos de ejecución al mes
  - Permite un almacenamiento de artefactos y logs de 500 Mb

  Al realizar múltiples jobs que suban artefactos, nos podemos encontrar con que agotemos este almacenamiento, por ello se proponen algunas soluciones:
  - Utilizar la propiedad **retention-days**: esta propiedad nos permite definir durante cuánto tiempo existirá el artefacto (mínimo 1 día, máximo 90)

```
- name: 'Upload Artifact'
   uses: actions/upload-artifact@v2
   with:
     name: target
     path: target/*.jar
     retention-days: 1
```

  - **Borrar los artefactos**: Una vez completado el workflow, se pueden borrar los artefactos generados (https://docs.github.com/es/actions/managing-workflow-runs/removing-workflow-artifacts)

1. Implantación del modelo de desarrollo Trunk Based Development

Se creará la rama main, que deberá ser la rama por defecto (la que en el modelo de desarrollo trunk based development llaman la rama trunk).
Algunas consideraciones:

- Cuando haya que mezclar cambios de una rama a main se hará mediante pull-request, es decir, cada cambio se implementará en una rama y se integrará en main mediante pull request.

2. Definición de workflows

Se deben ejecutar diferentes workflows dependiendo de las acciones realizadas en el repositorio git. Para ello en esta fase se incluirán los workflows necesarios como se indica:

- Cada vez que se quiera integrar un cambio en main:
  - Se ejecutarán las pruebas unitarias y de API REST.
- Al integrar con la rama de producción (main):
  - Se ejecutarán todas las pruebas
  - Se desplegará la aplicación en Heroku. El nombre de la aplicación será **ais-nombrealumnosinpunto-2021**
  - Se ejecutarán las pruebas de API REST y Selenium contra la URL de la aplicación desplegada (smoke test)

- Al crear una rama release (**release-***):
    - Se publicará una versión de la aplicación como una imagen Docker en Docker Hub, con el número de versión indicado en el pom.xml

- Cada noche, en la rama main:
    - Se ejecutarán todas las pruebas (unitarias, de API REST y Selenium).
    - Se publicará una imagen Docker en Docker Hub con una versión de desarrollo etiquetada como "dev-*fecha*", donde fecha es la fecha del día (ver instrucciones más abajo sobre cómo coger la fecha de un sistema Linux).

Consideraciones adicionales:

- **IMPORTANTE**: Se pueden crear ramas auxiliares de cara a comprobar el correcto funcionamiento de los workflows. Estas ramas deberán borrarse al terminar las pruebas, antes de pasar al paso 3.
- Deben utilizarse runners de Ubuntu (*ubuntu-20.04*)
- De cara a facilitar la identificación de los errores cuando fallan las pruebas, cuando haya que ejecutar varios tipos de prueba, cada tipo de prueba (unitaria, API REST, Selenium) debe ejecutarse en su propio step. Para ello, se pueden filtrar los test por el nombre del paquete usando el parámetro de Maven "-Dtest":

```
mvn -B '-Dtest=es.urjc.code.daw.library.e2e.selenium.*Test' test
```

- Para extraer la versión de la aplicación por línea de comandos se puede utilizar el siguiente comando que guarda la versión en una variable de entorno:

    - `VERSION=$(mvn -q help:evaluate -Dexpression=project.version -DforceStdout)`

- En el caso de los smoke tests, será necesario pasarle al test una variable que le indique dónde está lanzada la aplicación:

    - `mvn -Dhost=<HOST> test`

    La recogemos de las propiedades del sistema (localhost es el valor por defecto si no se le pasa esta propiedad). Deberá ser utilizada para realizar las peticiones REST o conectar el driver (utilizando el valor de "host" cuándo se realice un smoke test y "localhost" para el resto de casos)

    - `String host = System.getProperty("host", "localhost");`

3. Desarrollo de una funcionalidad nueva

Se desea introducir en la aplicación el servicio de linebreaker y sus tests de la práctica 2 (convocatoria ordinaria) que se proporciona. Para introducir esta funcionalidad, se proporciona la solución a la práctica dos en el aula virtual. El objetivo es incluir esta funcionalidad en la aplicación **siguiendo estrictamente el modelo de desarrollo de Trunk Based Development**, y acabar desplegando una versión de la aplicación en Heroku con esta funcionalidad de forma automática a través de las correspondientes GitHub Actions. La inclusión de la nueva funcionalidad se traducirá, además, en una nueva release (1.1.0).

Formato de entrega

Toda la práctica se debe desarrollar en el repositorio GitHub correspondiente que se haya creado siguiendo el procedimiento explicado al principio. Concretamente, este repositorio debe tener las ramas exigidas por el modelo de desarrollo **Trunk Based Development**, los cambios realizados deben hacerse siguiendo este modelo, y los profesores evaluarán que el modelo se haya seguido concienzudamente. Por ello, no se deberá borrar ninguna rama.

Adicionalmente la práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del repositorio GitHub. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf-z.gonzalez.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá especificar la URL del repositorio GitHub utilizado.
- Se deberá especificar la URL de la aplicación desplegada en Heroku
- Se deberá describir el funcionamiento de los workflows implementados: cuándo se ejecutan (en base a qué eventos en qué ramas) y qué hacen
- Se deben describir los pasos seguidos para incluir la nueva funcionalidad, indicando las acciones realizadas en el repositorio (creación de ramas, pull requests, etc) hasta el despliegue a producción. Para ello, la memoria incluirá qué comandos git se han utilizado para esta parte, y qué hace cada uno de ellos.
- La memoria deberá tener una longitud de 7 u 8 páginas (incluyendo la página de la portada).

Práctica 3. Integración, entrega y despliegue continuo

Convocatoria Extraordinaria 2022

Enunciado

Se desean implementar ciertos controles de calidad de la aplicación de la Práctica 1. Esta aplicación ofrece un interfaz web y una API REST para la gestión de películas, con sus correspondientes pruebas unitarias, de integración, de API REST y Selenium. Se proporciona el código de dicha aplicación, que podrá obtenerse como se indica más abajo.

El control de calidad se realizará mediante los siguientes pasos ejecutados en este orden:

1. Definición de workflows que automaticen la ejecución de pruebas, la publicación de releases y el despliegue en producción de la aplicación, tal como se describe en la sección correspondiente. Estos workflows deben funcionar correctamente antes de pasar al paso 2.
2. Desarrollo de una funcionalidad nueva utilizando TBD (Trunk Based Development), que se describe más abajo en su propia sección.

Preparación del repositorio inicial

Para realizar la práctica, se proporciona el código inicial de una aplicación que contiene pruebas unitarias, pruebas de integración y pruebas extremo a extremo de dos tipos: Selenium y pruebas de API REST.

Cada alumno (o grupo de dos alumnos) deberá crear un repositorio de código en GitHub que:

- Use como plantilla el siguiente repositorio: https://github.com/URJC-AIS/AIS-Practica-3-2022-template (click en "*Use this template*" y seleccionar que se cree en tu cuenta de usuario)
    - Si el alumno ha desarrollado la Práctica 1, puede utilizar sus test para complementar los existentes (muy limitados)
- Debe ser **privado**
- Debe nombrarse cómo **ais-usuario-urjc-2022-tbd**
    - Ejemplo: ais-p.perez-2022-tbd
- Debe incluir al profesor Michel Maes (Usuario en Github: **Maes95**) invitándole al repositorio

Paso 1. Definición de workflows

Se deben ejecutar diferentes workflows dependiendo de las acciones realizadas en el repositorio git. Para ello en esta fase se incluirán los workflows necesarios como se indica:

- **Workflow 1**: Cada vez que se termine una feature y antes de integrarse en la rama main (trunk):
  - Se ejecutarán las pruebas unitarias y de integración.
- **Workflow 2:** Al integrar con la rama de producción (trunk):
  - Se ejecutarán todas las pruebas
- **Workflow 3:** Al crear una rama release (**release-***) y hacer un commit en ella:
  - Se ejecutarán todas las pruebas
  - Se publicará una versión de la aplicación como una imagen Docker en Docker Hub, cuya versión sea el hash del commit
  - Se desplegará esta versión en Heroku. El nombre de la aplicación será **ais-nombrealumnosinpunto-2022**
  - Se ejecutará un smoke test (ver más abajo) sobre la aplicación desplegada
- **Workflow 4**: Cada noche, en la rama de main (trunk):
  - Se ejecutarán todas las pruebas
  - Se publicará una imagen Docker en Docker Hub con una versión de desarrollo etiquetada como “dev-*fecha*”, donde *fecha* es la fecha y hora del día.

Consideraciones adicionales:

- Los test de distinta naturaleza deben ir en steps separados (unitarios, integración, REST, e2e …)
- El nombre de las imagenes debe ser *nitflex*
- En el caso de los smoke tests, se deberá ejecutar el método *smokeTest* de la clase *SeleniumTest*. Será necesario pasarle al test una variable que le indique dónde está lanzada la aplicación:

  - `mvn -Dhost=<HOST> test`

  La recogemos en el test de las propiedades del sistema (localhost es el valor por defecto si no se le pasa esta propiedad). Hay que modificar el código Java del test para utilizar localhost o la URL de la aplicación:

  - `String host = System.getProperty("host", "localhost");`

- Los runners deben ser ubuntu-latest

- Cuando la aplicación se despliega debe usarse la imagen construida y publicada en DockerHub. No se construirá una nueva imagen para el despliegue.

Paso 2. Desarrollo de una funcionalidad nueva

Se desea incluir el campo 'director' en la entidad Film, de manera que se puedan crear nuevas películas introduciendo el nombre del director de la película. El objetivo es incluir esta funcionalidad en la aplicación **siguiendo estrictamente el modelo de desarrollo de Trunk Based Development**, y acabar desplegando una versión de la aplicación en Heroku con esta funcionalidad de forma automática a través de las correspondientes GitHub Actions. La inclusión de la nueva funcionalidad se traducirá, además, en una nueva release (cuya versión habrá que marcar con el commit).

Consideraciones adicionales:

- Se utilizará el enfoque de TBD para equipos grandes (creación de ramas feature de corta duración que se mezclarán con la rama destino usando Pull Requests)
- Para marcar la versión se cambiará la versión del pom.xml (eliminando el SNAPSHOT)
- **IMPORTANTE:** La práctica **se considerará suspensa** ante cualquiera de los siguientes supuestos:
    - No se sigue el desarrollo utilizando TBD
    - Los workflows no se ejecutan correctamente (dan fail)
    - La aplicación no se despliega correctamente
    - No se entrega satisfactoriamente al menos 1 artefacto

Formato de entrega

Toda la práctica se debe desarrollar en el repositorio GitHub correspondiente que se haya creado siguiendo el procedimiento explicado al principio. Concretamente, este repositorio debe tener las ramas exigidas por el modelo de desarrollo TBD, los cambios realizados deben hacerse siguiendo este modelo, y los profesores evaluarán que el modelo se haya seguido concienzudamente. Por ello, no se deberá borrar ninguna rama. Pueden borrarse las ejecuciones de workflows generados al hacer pruebas (pero no los que se ejecuten durante el desarrollo de la funcionalidad).

Adicionalmente la práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del repositorio GitHub. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf-z.gonzalez.zip)

Además del código fuente, se deberá elaborar una memoria en Markdown:

- Deberá guardarse en la carpeta raíz del proyecto como un README.md

- Deberá contener el nombre del alumno o alumnos.
- Se deberá especificar la URL del repositorio GitHub utilizado.
- Se deberá especificar la URL de la aplicación desplegada en Heroku
- Se deben describir los pasos seguidos para incluir la nueva funcionalidad, indicando las acciones realizadas en el repositorio (creación de ramas, pull requests, etc) hasta el despliegue a producción. Para ello, la memoria incluirá qué comandos git se han utilizado para esta parte, y qué hace cada uno de ellos.
    - Si una de las acciones desencadena un workflow, se pondrá un link al resultado de su ejecución.
    - En el caso de publicar una imagen, se deberá incluir la URL de DockerHub de la misma
- En el caso del workflow de Nightly, se describirá cuándo se lanza, qué hace, un link a la última ejecución del workflow y un link a la última imagen generada

Práctica 3. Integración, entrega y despliegue continuo

Convocatoria Ordinaria 2021

Enunciado

Se desean implementar ciertos controles de calidad de una aplicación que gestiona una librería online. Esta librería ofrece un interfaz web y una API REST para la gestión de libros, con sus correspondientes pruebas unitarias, de API REST y Selenium. Se proporciona el código de dicha aplicación, que podrá obtenerse como se indica más abajo.

El control de calidad se realizará mediante los siguientes pasos ejecutados en este orden:

1. Implantación del modelo de desarrollo git flow para el proyecto. Básicamente crear las ramas necesarias.
2. Definición de workflows que automaticen la ejecución de pruebas, la publicación de releases y el despliegue en producción de la aplicación, tal como se describe en la sección correspondiente. Estos workflows deben funcionar correctamente antes de pasar al paso 3.
3. Desarrollo de una funcionalidad nueva, que se describe más abajo en su propia sección.

Preparación del repositorio inicial

Para realizar la práctica, se proporciona el código inicial de una aplicación que contiene pruebas unitarias y pruebas extremo a extremo de dos tipos: Selenium y pruebas de API REST.

Cada alumno (o grupo de dos alumnos) deberá crear un repositorio de código en GitHub que:

- Use como plantilla el siguiente repositorio: https://github.com/AIS-Mostoles-2021/ais-practica-3-base (click en “Use this template” y seleccionar que se cree en tu cuenta de usuario)
- Debe ser privado
- Debe nombrarse cómo **ais-usuario-urjc-2021**
    - Ejemplo: ais-p.perez-2021
- Debe incluir a los profesores Emilio López (Usuario en Github: **emilopezcano**) y Michel Maes (Usuario en Github: **Maes95**) invitándoles al repositorio

Consideraciones:

- Los repositorios privados están limitados en cuanto a los recursos que pueden consumir, concretamente:
  - Permite un máximo de 2.000 minutos de ejecución al mes
  - Permite un almacenamiento de artefactos y logs de 500 Mb

  Al realizar múltiples jobs que suban artefactos, nos podemos encontrar con que agotemos este almacenamiento, por ello se proponen algunas soluciones:
  - Utilizar la propiedad **retention-days**: esta propiedad nos permite definir durante cuánto tiempo existirá el artefacto (mínimo 1 día, máximo 90)

    ```
    - name: 'Upload Artifact'
      uses: actions/upload-artifact@v2
      with:
        name: target
        path: target/*.jar
        retention-days: 1
    ```

  - **Borrar los artefactos**: Una vez completado el workflow, se pueden borrar los artefactos generados (https://docs.github.com/es/actions/managing-workflow-runs/removing-workflow-artifacts)

Implantación del modelo de desarrollo Git Flow

Se crearán todas las ramas necesarias, tal como indica el modelo de desarrollo Git Flow. Algunas consideraciones:

- Cuando haya que mezclar cambios con otra rama se hará mediante pull requests.
- Cada vez que se abra una rama de release para preparar una nueva versión se debe hacer lo siguiente:
  - En la rama de release, eliminar el sufijo “-SNAPSHOT” de la versión de la aplicación que se indica en el pom.xml.
  - En la rama de integración (rama develop en git flow), aumentar el *minor version* de la versión de la aplicación en el pom.xml.

Definición de workflows

Se deben ejecutar diferentes workflows dependiendo de las acciones realizadas en el repositorio git. Para ello en esta fase se incluirán los workflows necesarios como se indica:

- Cada vez que se termine una feature (ramas feature/*), y antes de integrarse en la rama correspondiente (rama develop):
  - Se ejecutarán las pruebas unitarias y de API REST.
- Por cada commit en la rama de integración (rama develop):
  - Se ejecutarán todas las pruebas (unitarias, API REST, Selenium)

- Cada noche, en la rama de integración:
  - Se ejecutarán todas las pruebas (unitarias, de API REST y Selenium) contra la rama de integración (rama develop).
  - Se publicará una imagen Docker en Docker Hub con una versión de desarrollo etiquetada como "dev-*fecha*", donde fecha es la fecha del día (ver instrucciones más abajo sobre cómo coger la fecha de un sistema Linux).
- Cuando se esté preparando una release, en la rama release/**:
  - Se ejecutarán las pruebas unitarias y de API REST por cada commit en la rama.
- Al integrar con la rama de producción:
  - Se ejecutarán todas las pruebas
  - Se publicará una versión de la aplicación como una imagen Docker en Docker Hub, con el número de versión indicado en el pom.xml
  - Se desplegará la aplicación en Heroku. El nombre de la aplicación será **ais-nombrealumnosinpunto-2021**
  - Se ejecutarán las pruebas de API REST y Selenium contra la URL de la aplicación desplegada

Consideraciones adicionales:

- De cara a facilitar la identificación de los errores cuando fallan las pruebas, cuando haya que ejecutar varios tipos de prueba, cada tipo de prueba (unitaria, API REST, Selenium) debe ejecutarse en su propio step. Para ello, se pueden filtrar los test por el nombre del paquete usando el parámetro de Maven "-Dtest":

```
mvn -B '-Dtest=es.urjc.code.daw.library.e2e.selenium.*Test' test
```

- Para extraer la versión de la aplicación por línea de comandos se puede utilizar el siguiente comando que guarda la versión en una variable de entorno:

  - `VERSION=$(mvn -q help:evaluate -Dexpression=project.version -DforceStdout)`

- En el caso de los smoke tests, será necesario pasarle al test una variable que le indique dónde está lanzada la aplicación:

  - `mvn -Dhost=<HOST> test`

  La recogemos de las propiedades del sistema (localhost es el valor por defecto si no se le pasa esta propiedad)

  - `String host = System.getProperty("host", "localhost");`

Desarrollo de una funcionalidad nueva

Se desea introducir en la aplicación el servicio de linebreaker y sus tests de la práctica 2. Para introducir esta funcionalidad, se proporciona la solución a la práctica dos en el aula virtual. El objetivo es incluir esta funcionalidad en la aplicación **siguiendo estrictamente el modelo de desarrollo de Git Flow**, y acabar desplegando una versión de la aplicación en Heroku con esta funcionalidad de forma automática a través de las correspondientes GItHub Actions.

Formato de entrega

Toda la práctica se debe desarrollar en el repositorio GitHub correspondiente que se haya creado siguiendo el procedimiento explicado al principio. Concretamente, este repositorio debe tener las ramas exigidas por el modelo de desarrollo git flow, los cambios realizados deben hacerse siguiendo este modelo, y los profesores evaluarán que el modelo se haya seguido concienzudamente. Por ello, no se deberá borrar ninguna rama.

Adicionalmente la práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del repositorio GitHub. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf-z.gonzalez.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF.

- Deberá guardarse en la carpeta raíz del proyecto.
- La portada deberá contener el nombre de la práctica, la asignatura, el curso académico y el nombre del alumno o alumnos.
- Se deberá especificar la URL del repositorio GitHub utilizado.
- Se deberá especificar la URL de la aplicación desplegada en Heroku
- Se deberá describir el funcionamiento de los workflows implementados: cuándo se ejecutan (en base a qué eventos en qué ramas) y qué hacen
- Se deben describir los pasos seguidos para incluir la nueva funcionalidad, indicando las acciones realizadas en el repositorio (creación de ramas, pull requests, etc) hasta el despliegue a producción. Para ello, la memoria incluirá qué comandos git se han utilizado para esta parte, y qué hace cada uno de ellos.

- La memoria deberá tener una longitud de 7 u 8 páginas (incluyendo la página de la portada).

Práctica 3. Integración, entrega y despliegue continuo

Convocatoria Ordinaria 2022

Enunciado

Se desean implementar ciertos controles de calidad de la aplicación de la Práctica 1. Esta aplicación ofrece un interfaz web y una API REST para la gestión de películas, con sus correspondientes pruebas unitarias, de integración, de API REST y Selenium. Se proporciona el código de dicha aplicación, que podrá obtenerse como se indica más abajo.

El control de calidad se realizará mediante los siguientes pasos ejecutados en este orden:

1. Definición de workflows que automaticen la ejecución de pruebas, la publicación de releases y el despliegue en producción de la aplicación, tal como se describe en la sección correspondiente. Estos workflows deben funcionar correctamente antes de pasar al paso 2.
2. Desarrollo de una funcionalidad nueva utilizando GitFlow, que se describe más abajo en su propia sección.

Preparación del repositorio inicial

Para realizar la práctica, se proporciona el código inicial de una aplicación que contiene pruebas unitarias, pruebas de integración y pruebas extremo a extremo de dos tipos: Selenium y pruebas de API REST.

Cada alumno (o grupo de dos alumnos) deberá crear un repositorio de código en GitHub que:

- Use como plantilla el siguiente repositorio: https://github.com/URJC-AIS/AIS-Practica-3-2022-template (click en “Use this template” y seleccionar que se cree en tu cuenta de usuario)
    - Si el alumno ha desarrollado la Práctica 1, puede utilizar sus test para complementar los existentes (muy limitados)
- Debe ser **privado**
- Debe nombrarse cómo **ais-usuario-urjc-2022**
    - Ejemplo: ais-p.perez-2022
- Debe incluir al profesor Michel Maes (Usuario en Github: **Maes95**) invitándole al repositorio

Paso 1. Definición de workflows

Se deben ejecutar diferentes workflows dependiendo de las acciones realizadas en el repositorio git. Para ello en esta fase se incluirán los workflows necesarios como se indica:

- **Workflow 1**: Cada vez que se termine una feature (ramas feature/*), y antes de integrarse en la rama correspondiente (rama develop):
  - Se ejecutarán las pruebas unitarias y de integración.
- **Workflow 2**: Por cada commit en la rama de integración (rama develop):
  - Se ejecutarán todas las pruebas
- **Workflow 3**: Cada noche, en la rama de integración:
  - Se ejecutarán todas las pruebas contra la rama de integración (rama develop).
  - Se publicará una imagen Docker en Docker Hub con una versión de desarrollo etiquetada como "dev-*fecha*", donde fecha es la fecha del día (ver instrucciones más abajo sobre cómo coger la fecha de un sistema Linux).
- **Workflow 4:** Cuando se esté preparando una release, en la rama release/**:
  - Se ejecutarán las pruebas unitarias y de integración por cada commit en la rama.
- **Workflow 5:** Al integrar con la rama de producción:
  - Se ejecutarán todas las pruebas
  - Se publicará una versión de la aplicación como una imagen Docker en Docker Hub, con el número de versión indicado en el pom.xml
  - Se desplegará la aplicación en Heroku. El nombre de la aplicación será **ais-nombrealumnosinpunto-2022**
  - Se ejecutará un smoke test (ver más abajo) sobre la aplicación desplegada

Consideraciones adicionales:

- Los test de distinta naturaleza deben ir en steps separados (unitarios, integración, REST, e2e …)
- En el caso de los smoke tests, se deberá ejecutar el método *smokeTest* de la clase *SeleniumTest*. Será necesario pasarle al test una variable que le indique dónde está lanzada la aplicación:

  - `mvn -Dhost=<HOST> test`

  La recogemos en el test de las propiedades del sistema (localhost es el valor por defecto si no se le pasa esta propiedad)

  - `String host = System.getProperty("host", "localhost");`

- Los runners deben ser ubuntu-latest

Paso 2. Desarrollo de una funcionalidad nueva

Se desea incluir el campo 'director' en la entidad Film, de manera que se puedan crear nuevas películas introduciendo el nombre del director de la película. El objetivo es incluir esta funcionalidad en la aplicación **siguiendo estrictamente el modelo de desarrollo de Git Flow**, y acabar desplegando una nueva versión de la aplicación en Heroku con esta funcionalidad de forma automática a través de las correspondientes GItHub Actions.

Algunas consideraciones:

- Cuando haya que mezclar cambios con otra rama se hará mediante pull requests.
- Cada vez que se abra una rama de release para preparar una nueva versión se debe hacer lo siguiente:
    - En la rama de release, eliminar el sufijo "-SNAPSHOT" de la versión de la aplicación que se indica en el pom.xml.
    - En la rama de integración (rama develop en git flow), aumentar el *minor version* de la versión de la aplicación en el pom.xml.

Formato de entrega

Toda la práctica se debe desarrollar en el repositorio GitHub correspondiente que se haya creado siguiendo el procedimiento explicado al principio. Concretamente, este repositorio debe tener las ramas exigidas por el modelo de desarrollo git flow, los cambios realizados deben hacerse siguiendo este modelo, y los profesores evaluarán que el modelo se haya seguido concienzudamente. Por ello, no se deberá borrar ninguna rama. Pueden borrarse los workflows generados al hacer pruebas (pero no los que se ejecuten durante el desarrollo de la funcionalidad).

Adicionalmente la práctica se entregará por el aula virtual teniendo en cuenta los siguientes aspectos:

- Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas sólo será entregada por uno de los alumnos.
- La práctica se entregará como un fichero .zip del repositorio GitHub. El nombre del fichero .zip será el usuario URJC del alumno. En caso de dos alumnos, el nombre del zip será el usuario URJC separado por guión (p.perezf-z.gonzalez.zip)
- En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o los alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Además del código fuente, se deberá elaborar una memoria en Markdown:

- Deberá guardarse en la carpeta raíz del proyecto como un README.md
- Deberá contener el nombre del alumno o alumnos.
- Se deberá especificar la URL del repositorio GitHub utilizado.
- Se deberá especificar la URL de la aplicación desplegada en Heroku
- Se deben describir los pasos seguidos para incluir la nueva funcionalidad, indicando las acciones realizadas en el repositorio (creación de ramas, pull requests, etc) hasta el despliegue a producción. Para ello, la memoria incluirá qué comandos git se han utilizado para esta parte, y qué hace cada uno de ellos.
    - Si una de las acciones desencadena un workflow, se pondrá un link al resultado de su ejecución.
    - En el caso de publicar una imagen, se deberá incluir la URL de DockerHub de la misma
- En el caso del workflow de Nightly, se describirá cuándo se lanza, que hace, un link a la última ejecución del workflow y un link a la última imagen generada

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 1
Curso 2020-2021

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una clase Java GestorMatriculas que permite determinar aquellos alumnos que pueden optar a matrícula de honor. Específicamente, la clase tiene un método, **posibleMatricula**, que determina, en función de las notas del alumno en 3 exámenes, si éste puede optar a matrícula de honor o no. La clase GestorMatrículas y el método posibleMatricula se indican a continuación.

```
public class GestorMatricula {

  private BaseDatosAlumnos alumnos;

  public GestorMatricula(BaseDatosAlumnos alumnos){
     this.alumnos = alumnos;
  }

  public boolean posibleMatricula(long idAlumno){
     double[] notas = this.alumnos.getNotas(idAlumno);
     double suma = 0;
     for(double nota : notas){
        suma += nota;
     }
     return suma == 30;
  }
}
```

Se pide verificar que si las notas de un alumno suman 30, el gestor devuelve que es una posible matrícula y que si las notas de un alumno no suman 30, devuelve que no es una posible matrícula. Para ello se realizarán dos test unitarios solitarios del método **posibleMatricula**. Se valorará no duplicar código.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 1
Curso 2020-2021

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 2
Curso 2020-2021

Apellidos:	Nombre:
D.N.I.:	

Pregunta 2. Preguntas cortas (4p)

a) ¿Qué es la "deuda técnica"? Pon un ejemplo

b) Describa brevemente los roles y responsabilidades del equipo de Scrum

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 2
Curso 2020-2021

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1) Indica la afirmación correcta en relación a la pirámide del testing

    a) Se recomienda tener el mismo número de pruebas unitarias que de integración y de sistema, porque hay que probar todos los aspectos de forma similar.
    b) Las pruebas de sistema (o E2E) son las más importantes, por eso están a la cabeza de la pirámide. Por eso hay que tener muchas de este tipo. Las unitarias son las menos importantes (parte baja) y muchas veces se ignoran.
    c) Se recomienda tener un mayor número de tests unitarios, una cantidad razonable de pruebas de integración y un en menor medida tests de sistema o E2E.

2) Indique la afirmación falsa respecto a las metodologías de desarrollo relacionadas con los tests.

    a) El Test Driven Development (TDD) es una metodología en la que los tests se programan partiendo de los requisitos, junto con el análisis, diseño e implementación y que guían el desarrollo.
    b) El Behavior Driven Development (BDD) es una metodología en la que los tests se implementan una vez que la implementación ya está completa para verificar el funcionamiento del software.
    c) Proceso Unificado de Desarrollo es una metodología que promueve que las pruebas se definan una vez que se tiene el diseño del software.

3) Indica la opción falsa en cuanto a la calidad de los propios tests

    a) Se debe tener una cobertura de, al menos, el 100% para que los tests se consideren adecuados.
    b) Una misma línea de código puede ejecutarse correctamente con unos valores de variables e incorrectamente con otros.
    c) Los tests no garantizan que el software esté libre de defectos.

4) Indica en qué consiste el mantenimiento adaptativo

    a) Modificar el sistema tras la detección de defectos o errores
    b) Modificar el sistema para acomodarlo a cambios del entorno
    c) Mejorar el sistema para cumplir con las nuevas necesidades/peticiones de los usuarios/negocio

5) Indica qué elemento de los siguientes no forma parte del ciclo de desarrollo de TDD

    a) Refactorizaciones y mejora del código
    b) Optimización del código
    c) Implementación del test de una nueva funcionalidad

6) Señala la respuesta FALSA a la pregunta ¿qué es la refactorización?

    a) Añadir funcionalidades que mejoren el software
    b) Mejorar la calidad interna de un sistema software
    c) Redudir la deuda técnica de un software

7) Indica la afirmación FALSA

    a) Un artefacto puede ser un zip con binarios
    b) Un artefacto puede ser una imagen Docker
    c) Un artefacto puede ser el diseño de la aplicación

8) Si queremos implementar una nueva funcionalidad de una aplicación web utilizando GitFlow:

    a) Creamos una nueva rama a partir de master para tener la última versión. Desarrollamos la funcionalidad y creamos una rama de release a partir de nuestra nueva rama. Desde esta rama de release se desplegará la web. Finalmente, se mergeará con master para poder crear nuevas ramas con nuevas funcionalidades.
    b) Creamos una nueva rama a partir de master que tendrá una duración máxima de 2 días. Nos ocupamos de que siempre se pueda construir el proyecto y ejecuten los test antes de mergear con la rama master/main.
    c) Creamos una nueva rama a partir de develop, implementamos la funcionalidad, mergeamos con la rama de desarrollo, creamos una rama de release y mergeamos esta última con las ramas de desarrollo y master/main.

9) El diagrama de Gantt es una herramienta para la planificación y seguimiento del proyecto que consiste en:

    a) Un diagrama de barras en forma de tabla donde se hace una referencia cruzada entre las tareas (filas) y los tiempos de duración de las mismas (columnas)
    b) Una representación gráfica del proyecto que relaciona las actividades
    c) Una matriz de amenazas y consecuencias que nos permite gestionar el riesgo del proyecto

10) En la técnica Kanban, al medir los flujos de trabajo...

    a) El "lead time" se cuenta desde que se hace una petición hasta que se hace la entrega.
    b) El "lead time" mide desde que el trabajo sobre una tarea comienza hasta que se termina.
    c) El "cycle time" se cuenta desde que se hace una petición hasta que se hace la entrega.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 1
Curso 2021-2022

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una clase Java que facilita el cálculo de precios de la empresa AllYouWant

```
public class AllYouWant {

    private ProductDatabase productDatabase;
    private TaxService taxService;

    public AllYouWant(ProductDatabase productDatabase, TaxService taxService) {
        this.productDatabase = productDatabase;
        this.taxService = taxService;
    }

    public double getFinalPrice(String code) {
        double rawPrice = productDatabase.getPrice(code);
        double tax = taxService.getTax(code);
        return rawPrice + rawPrice * tax;
    }
}
```

Se desea comprobar la funcionalidad del método **getFinalPrice** a través de test unitarios solitarios con los siguientes ejemplos.

Producto	**Código**	**IVA (tax)**	**Precio bruto**
El camino de los reyes	BOOK_123	4%	20€
Aceite de Girasol	FOOD_123	10%	2.5€
Elden Ring	GAME_123	21%	50€

- Se asume que están disponibles todas las librerias necesarias y no hay que realizar ningún import.

- Para obtener la máxima calificación, habrá que evitar el código repetido.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 2
Curso 2021-2022

Apellidos:	Nombre:
D.N.I.:	

Pregunta 2. Preguntas cortas (4p)

a) En el modelo de desarrollo Trunk-based Development, ¿que ocurre con la rama "trunk"? ¿Qué característica o características tienen el resto de ramas? (2 puntos)

b) Describa brevemente las tres reglas de kanban (2 puntos)

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 2
Curso 2021-2022

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1 Indica la afirmación incorrecta respecto de la calidad del software

1.a La calidad externa es perceptible por los usuarios y la interna no es perceptible directamente por ellos
1.b La calidad interna se mide únicamente por la ausencia de errores
1.c Uno de los aspectos que determinan la calidad interna del código es si se usan los mismos términos que usan los usuario (vocabulario de negocio)

2 Con Selenium no podemos...

2.a Grabar interacciones manuales con una web para luego poder reproducir esas interacciones.
2.b Testear APIs REST
2.c Testear Páginas web que usen JavaScript

3 Indica cuál de los siguientes aspectos NO favorece que el código sea mantenible

3.a Código seguro (sin vulnerabilidades)
3.b Código con pruebas automáticas
3.c Código con buena organización (buen diseño y buena arquitectura)

4 Indica en qué consiste el mantenimiento preventivo

4.a Modificar el sistema tras la detección de defectos o errores
4.b Mejorar el sistema para cumplir con las nuevas necesidades/peticiones de los usuarios/negocio
4.c Modificar el sistema con los cambios necesarios para mantener la calidad, eficiencia y fiabilidad del software

5 ¿Cuál de las siguientes afirmaciones NO indica cómo debe refactorizarse?

5.a Con pruebas automáticas
5.b Usando un compilador que te avise de los errores
5.c Con ayuda del IDE y de herramientas de análisis estático de código

6 ¿Cúal de estos pasos no corresponde al desarrollo guiado por pruebas (TDD)?

6.a Refactorizaciones y mejora del código
6.b Comprobación de que el test falla cuando no está implementada la funcionalidad
6.c Implementación de todos los test que hemos definido (utilizando una lista ordenada de ejemplos)

7 ¿Cuándo se genera un artefacto software?

7.a Cuando hemos comprobado que todos los test pasan en la rama master
7.b Por cada commit
7.c Las anteriores son correctas

8 Señala cúal de las siguientes afirmaciones es FALSA

8.a En GitHub Actions, un Job es la unidad mínima de ejecución
8.b En GitHub Actions, cada Workflow debe definirse en un único fichero
8.c En GitHub Actions, podemos desencadenar un Workflow sin realizar ningun commit

9 ¿Cuál de las siguientes afirmaciones sobre los equipos Ágiles es FALSA?

9.a Son equipos con una gran autonomía
9.b El jefe de proyecto es el encargado de llevar la planificación
9.c Es común la realización de breves reuniones diarias para coordinar el trabajo

10 ¿Cuál de los siguientes NO es un principio del manifiesto ágil?

10.a Tener reuniones diarías de sincronización
10.b Entregar frecuentemente software que funciona
10.c La simplicidad es esencial

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Mayo – Parte 1
Curso 2020-2021

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una web cuya página principal tiene un formulario para crear tweets con el siguiente HTML:

```
<html>
<body>
   <form method="post">
       Tweet: <input id="tweet-input" type="text" name="tweet">
       <input id="submit" type="submit" value="Send Tweet">
   </form>
</body>
</html>
```

Si el tweet tiene una longitud igual o menor a 140 caracteres, aparecerá está página:

```
<html>
<body>
   <p id="message">El tweet se creo correctamente</p>
</body>
</html>
```

En caso contrario, se muestra esta página de error:

```
<html>
<body>
   <p id="message">La longitud del tweet excede los 140 caracteres</p>
</body>
</html>
```

Se pide: Implementar dos tests para verificar el funcionamiento de la web:

- Cuando el número de caracteres del tweet enviado es igual o menor a 140 caracteres, se muestra una página que nos informa de que el tweet se creó correctamente
- Cuando el número de caracteres del tweet enviado es mayor que 140 caracteres, entonces aparece el mensaje de error.

La implementación debe realizarse con las siguientes consideraciones:

- Todos los tests pertenecerán a la misma clase de test.
- No hay que escribir ningún import.
- La web se asume que está ya disponible en la URL https://localhost:8080/ (no hay que iniciar ni parar la aplicación web)
- Se puede asumir que existe un atributo en la clase de test llamado driver y de tipo WebDriver que está correctamente inicializado y se cierra cuando finaliza el test.
- Para la evaluación se prestará especial atención a que **no haya código duplicado**.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Mayo – Parte 2
Curso 2020-2021

Apellidos: Nombre:

D.N.I.:

Pregunta 2. Preguntas cortas (4p)

a) Nos encontramos desarrollando un proyecto utilizando Git como SCM. Estamos realizando Git Flow para el desarrollo de una nueva funcionalidad, ¿que pasos deberíamos seguir? (No utilices comandos, explícalo con tus propias palabras) (2 puntos)

b) ¿Qué es la gestión de proyectos? (2 puntos)

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Mayo – Parte 2
Curso 2020-2021

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1) ¿En qué se diferencian los mocks y los stubs?

    a) En los stubs se verifica si los métodos del doble han sido invocados, mientras que en los mocks no.
    b) En los mocks se puede definir el comportamiento de los métodos y en los stubs no.
    c) En los mocks se verifica si los métodos del doble han sido invocados, mientras que en los stubs no.

2) Con Selenium no podemos...

    a) Grabar interacciones manuales con una web para luego poder reproducir esas interacciones.
    b) Testear APIs REST
    c) Testear Páginas web que usen JavaScript

3) ¿Cómo deben ser los módulos del diseño modular?

    a) Pequeños, cohesivos y poco acoplados
    b) Grandes, cohesivos y poco acoplados
    c) Pequeños, poco cohesivos y acoplados

4) Indica en qué consiste el mantenimiento perfectivo

    a) Modificar el sistema tras la detección de defectos o errores
    b) Mejorar el sistema para cumplir con las nuevas necesidades/peticiones de los usuarios/negocio
    c) Modificar el sistema con los cambios necesarios para mantener la calidad, eficiencia y fiabilidad del software

5) ¿Cuál de las siguientes afirmaciones NO es una de las excusas típicas para no refactorizar?

    a) Los managers no permiten que se refactorice
    b) No se puede refactorizar sin aplicar patrones de diseño
    c) Las bases de datos no se pueden refactorizar

6) ¿Cuál de las siguientes NO es una ventaja del TDD?

   a) Mejora la productividad
   b) Reduce bugs
   c) Se aplican patrones de diseño

7) ¿Cuándo se genera un artefacto software?

   a) Cuando hemos comprobado que todos los test pasan en la rama master
   b) Por cada commit
   c) Las anteriores son correctas

8) ¿En que consiste la integración continua?

   a) Ejecutar los test en tu máquina con cada cambio que realices. Para ello debes escribir el test, hacer que falle, implementar la funcionalidad mínima y después refactorizar
   b) Ejecutar los test en una máquina remota y desplegar la aplicación en un servidor ofrecido por un proveedor de cloud (cómo Heroku)
   c) Ninguna de las anteriores

9) ¿Cuál de los siguientes organigramas de equipo no tiene un líder de grupo permanente?

   a) Descentralizado Democrático (DD)
   b) Descentralizado Controlado (DC)
   c) Centralizado Controlado (CC)

10) ¿Cuál de los siguientes NO es un principio del manifiesto ágil?

   a) Limitar el trabajo en progreso (work in progress)
   b) Dar bienvenida a los cambios
   c) La simplicidad es esencial

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Mayo – Parte 1
Curso 2021-2022

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una clase Java que facilita el envio de tweets a Twitter a través de una API

```
public class Twitter {

    private TwitterAPI api;

    public Twitter(TwitterAPI api) {
        this.api = api;
    }

    public void sendTweet(String tweet) {
        if (!this.api.isValidTweet(tweet)) {
            throw new IllegalArgumentException("Tweet too long");
        }
        this.api.publishTweet(tweet);
    }

}
```

Se desea comprobar la funcionalidad del método **sendTweet** a través de test unitarios solitarios para los siguientes casos:

- El tweet es válido
- El tweet no es válido

Se asume que están disponibles todas las librerias necesarias y no hay que realizar ningún import.

Para obtener la máxima calificación, habrá que evitar el código repetido y realizar las verificaciones que se consideren necesarias.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Mayo – Parte 2
Curso 2021-2022

Apellidos: Nombre:

D.N.I.:

Pregunta 2. Preguntas cortas (4p)

a) ¿Que es una feature toggle? Pon al menos un ejemplo (2 puntos)

b) Describa los valores del manifiesto ágil. (2 puntos)

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Mayo – Parte 2
Curso 2021-2022

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1) Cuál de las siguientes NO es una limitación que podrían tener los DOCs (Depended-on Component) para probar el SUT:

    a) Que su preparación sea costosa en memoria o CPU.
    b) Que no sea determinista.
    c) Que sea muy grande.

2) Indica cuál de los siguientes términos NO se usa para referirse al código de buena calidad

    a) Código limpio
    b) Código con buen diseño
    c) Código óptimo

3) Indica cuál de los siguientes aspectos no está relacionado con la degradación de la calidad del software

    a) Mejor conocimiento del dominio
    b) Avances en la tecnología
    c) Reimplementar el software en un nuevo lenguaje

4) Indica en qué consiste el mantenimiento correctivo

    a) Modificar el sistema tras la detección de defectos o errores
    b) Mejorar el sistema para cumplir con las nuevas necesidades/peticiones de los usuarios/negocio
    c) Modificar el sistema con los cambios necesarios para mantener la calidad, eficiencia y fiabilidad del software

5) Señana que afirmación es FALSA

    a) En Git, el comando "git checkout ." descarta todos los cambios en el directorio
    b) En Git, si ejecutamos el comando "git fetch" se ejecuta el comando "git pull" seguido de "git merge"

c) En Git, el comando "git push" intenta hacer una mezcla de las ramas con "fast-foward"

6) Señala cúal de las siguientes afirmaciones es FALSA

a) En entrega continua es necesario ejecutar los test antes de generar el artefacto
b) En entrega continua es necesario ejecutar los smoke test sobre la aplicación ya desplegada (en Heroku, por ejemplo)
c) El entrega continua, el artefacto generado debe guardarse en algun repositorio como Maven, NPM o DockerHub

7) En general, un proyecto:

a) Debe hacerse con el menor número de recursos posibles
b) No es relevante el tiempo en que se realice si se consigue el objetivo
c) Sus actividades son esencialmente repetitivas

8) ¿Cuál es la idea detrás de la Ley de Brooks?

a) El software es intangible
b) Siempre que se pueda debemos usar diagramas de Gantt para el seguimiento de proyectos
c) En los proyectos de desarrollo de software no existe el hombre-mes

9) Señale cuál de las siguientes opciones contiene algunas de las actividades de XP:

a) Comunicación, Simplicidad, feedback
b) Codificar, Probar, Escuchar
c) Pair Programming, Refactoring, Small releases

10) ¿Qué es la velocidad de un equipo Scrum?

a) Cantidad de puntos de historia o historias de usuario que terminan por iteración
b) El número de funcionalidades que el product owner es capaz de definir en un sprint
c) La media de horas que un equipo tada en resolver una historia de usuario

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Junio – Parte 1
Curso 2020-2021

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una clase de Chat implementada como se indica. Se desea comprobar que cuando un usuario envía un mensaje, éste llega a los demás usuarios, pero no al usuario que mandó el mensaje. La clase de chat utiliza un servicio (UserService) para saber qué usuarios hay conectados al Chat. Se pide **implementar un test unitario solitario** que verifique que dados tres usuarios conectados al chat, cuando uno de ellos escribe un mensaje, los otros dos usuarios lo reciben a través del método **sendMessage** de la clase User, pero el usuario que lo envía no lo recibe. Para ello será necesario mockear la clase UserService.

```
public class Chat(){

   private UserService userService;

   public Chat(UserService userService){
       this.userService = userService;
   }

   public void sendMessage(String message){
       for(User user: this.userService.getAllUsers()){
           if(user != userService.getMe()) {
             user.sendMessage(message);
           }
       }
   }
}
```

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Junio – Parte 2
Curso 2020-2021

Apellidos: Nombre:

D.N.I.:

Pregunta 2. Preguntas cortas (4p)

a) Enumera 4 ventajas de utilizar TDD

b) En Planificación de proyectos, ¿en qué consiste el diagrama de descomposición de actividades (WBS por sus siglas en inglés)?

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Junio – Parte 2
Curso 2020-2021

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1) ¿Qué características pueden probar los tests?

    a) Características funcionales o no funcionales
    b) Características de caja blanca o de caja negra
    c) Características unitarias o de sistema

2) Qué es falso respecto a la automatización de las pruebas?

    a) Las pruebas automáticas se pueden crear grabando las interacciones de un usuario real con el sistema para reproducirlas posteriormente
    b) Las pruebas automáticas se tienen que implementar con el lenguaje de programación con el que está implementado el sofware que se está probando.
    c) Existen frameworks de alto nivel que facilitan la implementación de pruebas.

3) Un sistema de integración continua...

    a) ...sirve para compartir código fuente entre los miembros de un equipo de desarrollo. Dispone de mecanismos para "integrar" ese código de forma continua.
    b) ...se ejecuta en cada máquina de cada desarrollor para verificar la calidad del software antes de integrarlo con el de sus compañeros.
    c) ...permite la construcción del software y ejecución de pruebas cuando el código se ha integrado en un repositorio.

4) Indican cuál de los siguientes no es un tipo de mantenimiento:

    a) Correctivo
    b) Evolutivo
    c) Perfectivo

5) Indica la afirmación FALSA en relación con la Deuda Técnica.

    a) Es una metáfora inspirada en la deuda económica que genera intereses.
    b) Se recomienda tener mucha deuda técnica porque eso te permite programar más rápido durante todo el proyecto.
    c) Cuando se acuñó el término no se consideraba que los desarrolladores pudieran adquirir deuda técnica de forma deliberada.

6) ¿Cuál de las siguientes afirmaciones NO indica cómo debe refactorizarse?

    a) Con pruebas automáticas
    b) Usando un compilador que te avise de los errores
    c) Con ayuda del IDE y de herramientas de análisis estático de código

7) Indica la afirmación FALSA

    a) Un artefacto puede ser un zip con binarios
    b) Un artefacto puede ser una imagen Docker
    c) Un artefacto puede ser el diseño de la aplicación

8) Si queremos implementar una nueva funcionalidad de una aplicación web utilizando TBD (Trunk-base development) en un equipo grande:

    a) Creamos una nueva rama a partir de master para tener la última versión. Desarrollamos la funcionalidad y creamos una rama de release a partir de nuestra nueva rama. Desde esta rama de release se desplegará la web. Finalmente, se mergeará con master para poder crear nuevas ramas con nuevas funcionalidades.
    b) Creamos una nueva rama a partir de master que tendrá una duración máxima de 2 días. Nos ocupamos de que siempre se pueda construir el proyecto y ejecuten los test antes de mergear con la rama master/main.
    c) Creamos una nueva rama a partir de develop, implementamos la funcionalidad, mergeamos con la rama de desarrollo, creamos una rama de release y mergeamos esta última con las ramas de desarrollo y master/main.

9) En un plan de proyecto, ¿qué es un hito?

   a) Un evento del proyecto alcanzable en una fecha, que puede coincidir con la finalización de una tarea
   b) Cada una de las tareas con sus fechas de inicio y fin y su duración
   c) Cada uno de los productos entregables

10) Señale cuál de las siguientes opciones contiene algunos de los valores de XP:

   a) Pair Programming, Refactoring, Small releases
   b) Codificar, Probar, Escuchar
   c) Comunicación, Simplicidad, feedback

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Junio – Parte 1
Curso 2021-2022

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una web cuya página principal tiene un formulario para comprobar que un DNI es correcto o no

```
<html>
<body>
   <form method="post">
       DNI: <input id="dni-input" type="text" name="dni">
       <input id="send" type="submit" value="Send DNI">
   </form>
</body>
</html>
```

Si el formato de DNI es correcto, aparecerá está página:

```
<html>
<body>
   <p id="message">El formato de DNI es correcto</p>
</body>
</html>
```

En caso contrario, se muestra esta página de error:

```
<html>
<body>
   <p id="message">El formato de DNI no es correcto</p>
</body>
</html>
```

Se pide: Implementar dos tests para verificar el funcionamiento de la web:

- Cuando el DNI sigue el formato NNNNNNNN-L (dónde N es un número y L una letra mayúscula) se considera correcto
- Cuando el DNI NO sigue el formato especificado, se considerará incorrecto.

La implementación debe realizarse con las siguientes consideraciones:

- Todos los tests pertenecerán a la misma clase de test.
- No hay que escribir ningún import.

- La web se asume que está ya disponible en la URL https://localhost:8080/ (no hay que iniciar ni parar la aplicación web)
- Se puede asumir que existe un atributo en la clase de test llamado driver y de tipo WebDriver que está correctamente inicializado y se cierra cuando finaliza el test.
- Para la evaluación se prestará especial atención a que **no haya código duplicado**.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 2
Curso 2021-2022

Apellidos:	Nombre:
D.N.I.:	

Pregunta 2. Preguntas cortas (4p)

a) ¿De qué 3 pasos se compone un test? Pon un ejemplo (2 puntos)

b) ¿Cómo debe ser un buen proyecto? (2 puntos)

Ampliación de Ingeniería del Software (Móstoles)
Convocatoria de Junio – Parte 2
Curso 2021-2022

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1 En qué se diferencian las pruebas sociables de las pruebas solitarias

1.a Las pruebas unitarias sociables permiten la colaboración con dependencias reales, mientras que en las solitarias todas las dependencias son dobles
1.b Las pruebas sociales son las pruebas de integración con sistemas externos. Las pruebas solitarias son las pruebas sin acceso a sistemas externos.
1.c Las pruebas sociales son las de sistema porque prueba todos los elementos "socialmente". Las pruebas solitarias son las que prueban un subsistema completo de forma aislada.

2 ¿Qué NO son las pruebas de aceptación?

2.a Pruebas de sistema manuales que permiten validar si el sistema cumple con los objetivos para los que fue diseñado.
2.b Las pruebas que verifican si se aceptan correctamente los valores correctos en las entradas al sistema.
2.c Pruebas creadas en la metodología BDD (Behavior Driven Development)

3 Marca la opción verdadera en relación con el código legible

3.a Es mejor comentar todo el código para que sea más comprensible
3.b Cuando un código no se ejecute es mejor dejarlo comentado (pero no borrarlo) por si se vuelve a necesitar en el futuro
3.c Es mejor ser consistente en el código (por ejemplo en los nombres de los métodos que realizan operaciones similares)

4 Indique la afirmación verdadera en relación con los Patrones de diseño:

4.a Un patrón de diseño es una librería o un framework de programación que permite reutilizar una solución ya creada previamente.
4.b Existen patrones de creación, comportamiento, estructurales y de sistema
4.c El patrón singleton se utiliza para crear objetos de una clase mediante un método estático en vez de crearlos con el constructor.

5 ¿Cuál de las siguientes afirmaciones no es un motivo para realizar mantenimiento del software?

5.a Proveer continuidad del servicio
5.b Dar soporte a peticiones de mejora
5.c Instalar el software en un nuevo servidor

6 Señala la respuesta FALSA a la pregunta ¿qué es la refactorización?

6.a Añadir funcionalidades que mejoren el software
6.b Mejorar la calidad interna de un sistema software
6.c Redudir la deuda técnica de un software

7 Indica la afirmación incorrecta en relación con el TDD

7.a Para aplicar TDD es necesario programar tests
7.b El TDD favorce la evolución del software
7.c La programación con TDD reduce la productividad porque hay que implementar un test por cada funcionalidad

8 Si queremos implementar una nueva funcionalidad de una aplicación web utilizando GitFlow:

8.a Creamos una nueva rama a partir de master para tener la última versión. Desarrollamos la funcionalidad y creamos una rama de release a partir de nuestra nueva rama. Desde esta rama de release se desplegará la web. Finalmente, se mergeará con master para poder crear nuevas ramas con nuevas funcionalidades.
8.b Creamos una nueva rama a partir de master que tendrá una duración máxima de 2 días. Nos ocupamos de que siempre se pueda construir el proyecto y ejecuten los test antes de mergear con la rama master/main.
8.c Creamos una nueva rama a partir de develop, implementamos la funcionalidad, mergeamos con la rama de desarrollo, creamos una rama de release y mergeamos esta última con las ramas de desarrollo y master/main.

9 Señale cuál de las siguientes opciones contiene algunas de las prácticas de XP

9.a Comunicación, Simplicidad, feedback
9.b Codificar, Probar, Escuchar
9.c Pair Programming, Refactoring, Small releases

10 ¿Cuál de los siguientes NO es un principio del manifiesto ágil?

10.a Los procesos ágiles promueven el desarrollo sostenible
10.b La simplicidad no es esencial
10.c El software que funciona es la principal medida de cambio

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 1
Curso 2020-2021

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Se dispone de una clase Java que permite la gestión de nóminas de empleados

```
public class EmployeeManager {

  private EmployeeRepository employeeRepository;

  private IRFPService irpfService;

  private MailService mailService;

  public EmployeeManager(EmployeeRepository employeeRepository,
                                        IRFPService irpfService,
                                       MailService mailService){
     this.employeeRepository = employeeRepository;
     this.irpfService = irpfService;
     this.mailService = mailService;
  }

  public double calculatePayroll(long employeeId){
     Employee employee = this.employeeRepository.getById(employeeId);

     double grossSalary = employee.getGrossSalary()
     double retention = this.irpfService.calculateIRPFRetention(grossSalary)

     double netSalary = grossSalary - retention;

     this.mailService.sendPayrollEmail(employee.getEmail(), netSalary);

     return netSalary;
}
```

Se desea implementar un test unitario solitario del método **calculatePayroll** de la clase EmployeeManager. Para ello habrá que mockear las dependencias y verificar que se llaman a los métodos de las mismas, además de comprobar el resultado. En el test se puede usar un empleado de la siguiente forma:

```
Employee employee = new Employee(1, 31000, "dummy.employee@company.com");
```

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 1
Curso 2020-2021

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 2
Curso 2020-2021

Apellidos:	Nombre:
D.N.I.:	

Pregunta 2. Preguntas cortas (4p)

a) Indica los pasos del ciclo de desarrollo del TDD (2 puntos)

b) Describe los valores del manifiesto ágil (2 puntos)

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 2
Curso 2020-2021

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1) Indique la afirmación falsa en relación con los principios SOLID:

   a) El Single Responsibility Principle indica que una clase debe tener una única razón para cambiar.
   b) El Open/Close principle indica que una clase tiene que estar abierta para la modificación y cerrada para la extensión
   c) El Principio de sustitución de Liskov se obtiene al aplicar correctamente el polimorfismo

2) ¿Qué NO son las pruebas de aceptación?

   a) Pruebas de sistema manuales que permiten validar si el sistema cumple con los objetivos para los que fue diseñado.
   b) Las pruebas que verifican si se aceptan correctamente los valores correctos en las entradas al sistema.
   c) Pruebas creadas en la metodología BDD (Behavior Driven Development)

3) En qué se diferencian las pruebas sociables de las pruebas solitarias

   a) Las pruebas unitarias sociables permiten la colaboración con dependencias reales, mientras que en las solitarias todas las dependencias son dobles
   b) Las pruebas sociales son las pruebas de integración con sistemas externos. Las pruebas solitarias son las pruebas sin acceso a sistemas externos.
   c) Las pruebas sociales son las de sistema porque prueba todos los elementos "socialmente". Las pruebas solitarias son las que prueban un subsistema completo de forma aislada.

4) Indica cuál de las siguientes es una de las leyes de Lehman y Belady

   a) Ley de la evolución
   b) Ley de complejidad creciente
   c) Ley de la calidad continua

5) Señala la respuesta FALSA a la pregunta ¿para qué se refactoriza?

    a) Para que el software sea más comprensible
    b) Para que el software sea más óptimo
    c) Para que sea más fácil de evolucionar

6) Indica cuál de los siguientes no es un sinónimo del enfoque Top-down para TDD

    a) London TDD
    b) Mockist TDD
    c) Detroit TDD

7) ¿En que consiste la trazabilidad de los artefactos software?

    a) Un artefacto debe ir empaquetado en una imagen Docker, que es un estándar
    b) Un artefacto deben generarse a partir de una revisión concreta del código
    c) Un artefacto debe contar con la documentación necesario para poder ejecutarse como una aplicación

8) Señala cúal de las siguientes afirmaciones es FALSA

    a) En GitHub Actions, un Job es la unidad mínima de ejecución
    b) En GitHub Actions, cada Workflow debe definirse en un único fichero
    c) En GitHub Actions, podemos desencadenar un Workflow sin realizar ningún commit

9) En general, un proyecto:

    a) Implica un principio y un final
    b) No tiene un objetivo
    c) Sus actividades son esencialmente repetitivas

10) ¿Cuál de los siguientes artefactos constituye una lista priorizada y estimada de historias de usuario?

    a) El Product Backlog
    b) El Sprint Review
    c) La Sprint retrospective

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 1
Curso 2021-2022

Apellidos:	Nombre:	
D.N.I.:		

Pregunta 1. Ejercicio (3p)

Wordle es un juego sencillo en el que cada día tenemos que adivinar una palabra de 5 letras. Se dispone de una clase Java que permite jugar al Wordle. Esta clase utiliza una API de la que obtiene del servidor de Wordle la palabra diaria.

```
public class Wordle {

    private WordleAPI api;

    public Wordle(WordleAPI api) {
        this.api = api;
    }

    public char[] checkWord(String word) {
        if(word.length() != 5) {
            throw new RuntimeException("La palabra debe tener 5 letras");
        }
        String wordOfTheDay = api.getWord();
        char[] result = new char[5];
        for(int i = 0; i < wordOfTheDay.length(); i++) {
            if(word.charAt(i) == wordOfTheDay.charAt(i)) {
                result[i] = 'O';
            }else if(wordOfTheDay.contains(String.valueOf(word.charAt(i)))) {
                result[i] = '-';
            }else{
                result[i] = 'X';
            }
        }
        return result;
    }
}
```

Funcionamiento: Si la palabra a adivinar es SESGO y el usuario empieza con AUREO, el resultado de Wordle será **X X X - O**

- Con **O** se indica que la letra está en la posición correcta
- Con **X** se indica que la letra NO está en la palabra a adivinar
- Con **-** se indica que la letra está en la palabra a adivinar, pero no en su posición

Se pide: comprobar el funcionamiento de **checkWord** a través de dos test unitarios solitarios en Java (se asumen los imports y librerias necesarias):

- El primer test comprobara el funcionamiento del programa cuando el usuario introduce las palabras AUREO, CESTO y SESGO, siendo la palabra a adivinar SESGO.
- El segundo test comprobará el funcionamiento esperado cuando se use la palabra UNO

Para obtener la máxima calificación, habrá que evitar el código repetido.

Universidad Rey Juan Carlos

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 2
Curso 2020-2021

Apellidos:	Nombre:
D.N.I.:	

Pregunta 2. Preguntas cortas (4p)

a) Describe las diferencias entre Integración Continua y Despliegue Continuo (2 puntos)

b) Describa brevemente las reuniones que existen en Scrum (2 puntos)

Ampliación de Ingeniería del Software (Vicálvaro)
Convocatoria de Mayo – Parte 2
Curso 2021-2022

Pregunta 3. Test (3p)
10 preguntas de tipo test. Respuesta correcta +0.3. Respuesta incorrecta -0.15

	1	2	3	4	5	6	7	8	9	10
Resp:										

1) Marque la opción incorrecta sobre los matchers.

    a) Un matcher determina si un valor cumple una determinada propiedad.
    b) Los matchers se usan para definir el comportamiento de los dobles en función del valor de los parámetros.
    c) Los matchers mejoran los mensajes de error cuando un test falla

2) Indica cuál de los siguientes términos NO se usa para referirse al código de mala calidad

    a) Cruft
    b) Código espagueti
    c) Código sucio

3) Indican cuál de los siguientes no es un tipo de mantenimiento:

    a) Correctivo
    b) Evolutivo
    c) Perfectivo

4) Indica la afirmación FALSA en relación con la Deuda Técnica.

    a) Es una metáfora inspirada en la deuda económica que genera intereses.
    b) Se recomienda tener mucha deuda técnica porque eso te permite programar más rápido durante todo el proyecto.
    c) Cuando se acuñó el término no se consideraba que los desarrolladores pudieran adquirir deuda técnica de forma deliberada.

5) Si queremos implementar una nueva funcionalidad de una aplicación web utilizando TBD (Trunk-base development) en un equipo grande:

    a) Creamos una nueva rama a partir de master para tener la última versión. Desarrollamos la funcionalidad y creamos una rama de release a partir de nuestra nueva rama. Desde esta rama de release se desplegará la web. Finalmente, se mergeará con master para poder crear nuevas ramas con nuevas funcionalidades.
    b) Creamos una nueva rama a partir de master que tendrá una duración máxima de 2 días. Nos ocupamos de que siempre se pueda construir el proyecto y ejecuten los test antes de mergear con la rama master/main.
    c) Creamos una nueva rama a partir de develop, implementamos la funcionalidad, mergeamos con la rama de desarrollo, creamos una rama de release y mergeamos esta última con las ramas de desarrollo y master/main.

6) ¿En que consiste la integración continua?

   a) Ejecutar los test en tu máquina con cada cambio que realices. Para ello debes escribir el test, hacer que falle, implementar la funcionalidad mínima y después refactorizar
   b) Ejecutar los test en una máquina remota y desplegar la aplicación en un servidor ofrecido por un proveedor de cloud (cómo Heroku)
   c) Ninguna de las anteriores

7) ¿Cuál de los siguientes organigramas de equipo tiene una comunicación vertical?

   a) Descentralizado Democrático (DD)
   b) Descentralizado Controlado (DC)
   c) Centralizado Controlado (CC)

8) En un plan de proyecto, ¿qué es un hito?

   a) Un evento del proyecto alcanzable en una fecha, que puede coincidir con la finalización de una tarea
   b) Cada una de las tareas con sus fechas de inicio y fin y su duración
   c) Cada uno de los productos entregables

9) Señale cuál de las siguientes opciones contiene algunos de los valores de XP:

   a) Comunicación, Simplicidad, feedback
   b) Codificar, Probar, Escuchar
   c) Pair Programming, Refactoring, Small releases

10) ¿Qué rol Scrum ejerce el rol clásico de jefe de proyecto?

   a) No existe el rol clásico de jefe de proyecto
   b) Product Owner
   c) Scrum Master