Parte 1 - Verificación

Proyecto BETS - Ingeniería del Software II

16 de Octubre de 2023

—

UPV/EHU

Índice

Equipo

Miembro	Horas de Dedicación	Método Analizado
Nicolás Aguado	~10 horas	gertaerakSortu(...)
Xiomara G. Cáceres	13 horas	EmaitzakIpini(...)
Eider Fernández	+18 horas	gertaeraEzabatu(..)

Enlaces

Enlace del grupo de trabajo en github:

https://github.com/por0tos

Enlace al proyecto en github:

<!> OJO! Los laboratorios se solapan <!>

<!> Preferir el código que hay en el anexo al código en github <!>

https://github.com/por0tos/Bet22porotos

Al haber mandado dos trabajos distintos con distinta fecha de entrega, cada miembro del grupo se ha organizado de una manera distinta. Entonces, el enlace al proyecto de github incluye los cambios realizados DESPUÉS de TODOS los laboratorios.

Para facilitar esta tarea, se han añadido (algunas) “etiquetas” que indican el estado del proyecto antes y después del trabajo de los miembros del grupo.

https://github.com/por0tos/Bet22porotos/tags

Enlace de sonarcloud:

https://sonarcloud.io/project/overview?id=por0tos_Bet22porotos

Cambios realizados

Al proyecto proporcionado se le han añadido más tarde unos ficheros de “ejemplo” para consulta y referencia posterior. Estos ficheros incluían una serie de fallos y han sido solucionados ANTES de empezar a trabajar. Los cambios estáticos realizados que se propusieron en el primer laboratorio de sonar son éstos.

Métodos Analizados

gertaerakSortu()

Autor: Nicolás Aguado

Código

Descripción: Este método servirá para crear un nuevo evento en una fecha, de un deporte y con unos equipos específicos. Este evento será insertado en la base de datos siempre y cuando no exista ya, siempre que el deporte esté registrado en la base de datos y siempre que se proporcione una descripción del estilo de “Equipo 1 - Equipo 2”.

Código a analizar:

```
public boolean gertaerakSortu(String description,Date eventDate, String sport) {
    boolean b = true;
    db.getTransaction().begin();
    Sport spo =db.find(Sport.class, sport);
    if(spo!=null) {
        TypedQuery<Event> Equery = db.createQuery("SELECT e FROM Event e WHERE e.getEventDate() =?1 ",Event.class);
        Equery.setParameter(1, eventDate);
        for(Event ev: Equery.getResultList()) {
            if(ev.getDescription().equals(description)) {
                b = false;
            }
        }
        if(b) {
            String[] taldeak = description.split("-");
            Team lokala = new Team(taldeak[0]);
            Team kanpokoa = new Team(taldeak[1]);
            Event e = new Event(description, eventDate, lokala, kanpokoa);
            e.setSport(spo);
            spo.addEvent(e);
            db.persist(e);
        }
    }else {
        return false;
    }
    db.getTransaction().commit();
    return b;
}
```

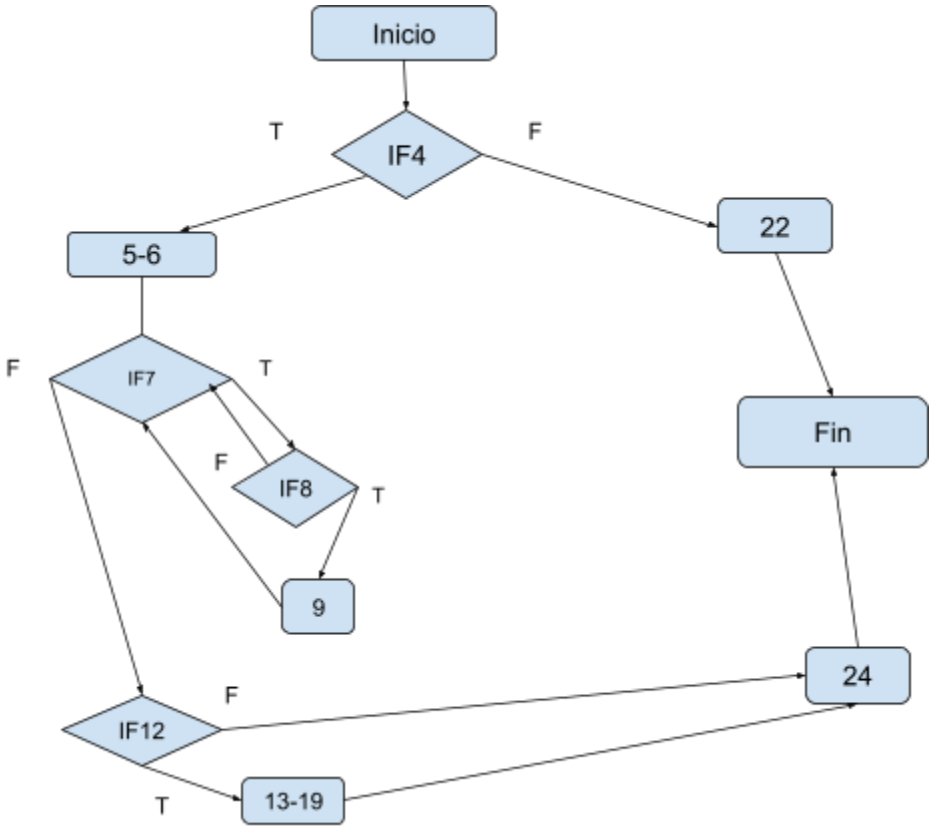
Una copia de este método en formato texto se puede encontrar en el Anexo, en su apartado correspondiente.

Pruebas Unitarias

Probaremos la clase DataAccess.class

Diseño - Caja Blanca

Primero analizaremos el grafo de diagrama de flujo



$$V(G) = 4 + 1 = 5$$

Del cual obtendremos los siguientes caminos

- Camino #1 = IF4 (F) 22 Fin
- Camino #2 = IF4 (T) 5-6 IF7 (T) IF8(T) 9 IF12(F) 24 Fin
- Camino #3 = IF4 (T) 5-6 IF7 (T) IF8(F) IF12(T) 13-19 24 Fin
- Camino #4 = IF4 (T) 5-6 IF7 (F) IF12(T) 13-19 24 Fin
- Camino #5 = IF4 (T) 5-6 IF7 (F) IF12(F) 24 Fin

Entonces, definiremos los siguientes casos de prueba

#	Camino	Condición	Entrada		Resultado Esperado	
			Parámetros	Estado BD	Salida	Estado BD
1	IF4 (F) 22 Fin	sport == null	sport = “Golf”	sport $\notin$ bd	FALSE	Sin cambios
2	IF4 (T) 5-6 IF7 (T) IF8(T) 9 IF12(F) 24 Fin	sport ! = null and eventDate $\in$ bd and description $\in$ resultList	description = “Madrid-Barça” eventDate = “1/10/2022” sport = “Futbol”	Evento con misma fecha y descripcion en db	FALSE	Sin cambios
3	IF4 (T) 5-6 IF7 (T) IF8(F) IF12(T) 13-19 24 Fin	sport ! = null and eventDate $\in$ bd and description $\notin$ resultList	description = “Madrid-Barça” eventDate = “22/10/2022” sport = “Futbol”	Eventos en fecha consultada pero no el de la descripcion	TRUE	Introducido evento en db con relaciones correspondient-es hechas
4	IF4 (T) 5-6 IF7 (F) IF12(T) 13-19 24 Fin	sport ! = null and eventDate $\in$ bd and description $\notin$ resultList	description = “Madrid-Barça” eventDate = “25/10/2022” sport = “Futbol”	Ningun evento en el día consultado	TRUE	Introducido evento en db con relaciones correspondient-es hechas
5	IF4 (T) 5-6 IF7 (F) IF12(F) 24 Fin	Situación imposible?				

Diseño - Caja Negra

Para hacer las pruebas de caja negra tomaremos la signatura del método

```
public boolean gertaerakSortu(String description,Date
eventDate, String sport)
```

El método *gertaerakSortu* insertará (devolverá true) un nuevo evento en la base de datos, en una fecha concreta, en un deporte concreto. Si ya hay un evento con mismo nombre y fecha o si el deporte no se encuentra clasificado en la db no se insertará el evento (devuelve false).

Una vez conocemos la información, obtendremos las clases de equivalencia

Condición de Entrada	Clases de Equivalencia Válidas	Clases de Equivalencia NO Válidas
Deporte clasificado	$sport \in db$ (1)	$sport \notin db$ (2)
Evento en db	$\nexists e \in db / e.desc = description \wedge e.sport = sport \wedge e.date = date$ (3)	$\exists e \in db / e.desc = description \wedge e.sport = sport \wedge e.date = date$ (4)

Y generaremos una batería de casos de prueba. No obtendremos valores límite porque no trabajamos con números, sino que son todo valores binarios. (Está / No está, Cumple / No cumple, etc...)

Clase(s) Cubiertas	Contexto de Entrada		Contexto de Salida	
	Estado DB	Parámetros	Salida	DB Resultante
2	$sport \notin db$	sport = “Pilla-Pilla”	FALSE	Sin cambios
1,4	$sport \in db$ and $e \in db$	sport=”Futbol” description=”Madrid-Barça” date=20/10/2022	FALSE	Sin cambios
1,3	$sport \in db$ and $e \notin db$	sport=”Tennis” description=”Alcaraz-Federer” date=20/10/2023	TRUE	Evento (con los parámetros) insertado en la DB

Implementación - Caja Blanca

Para alcanzar un 100% del código cubierto, se han hecho 4 casos de prueba. En este caso todos los test pasan correctamente sin nada que destacar.

El fichero *GertaerakSortuDAWTest.java* en el que se han implementado los tests queda subido al proyecto de github y también se encuentra ubicado en su correspondiente sección en el anexo.

Implementación - Caja Negra

Se han hecho 3 casos de prueba, pero no se ha alcanzado un 100% de cobertura. Esto ha sido así por la manera en la que se ha implementado el método, haciendo dos comprobaciones (primero por fecha y luego por nombre de evento) en lugar de hacerlo todo de una vez. El ingeniero que prueba el método, como nos encontramos en una fase de caja negra, no podría saber eso así que dá su trabajo en este apartado por finalizado.

En este caso todos los test pasan correctamente sin nada que destacar.

El fichero *GertaerakSortuDABTest.java* en el que se han implementado los tests queda subido al proyecto de github y también se encuentra ubicado en su correspondiente sección en el anexo.

Defectos Encontrados

Como tal, en la implementación del método NO se han encontrado defectos. Eso sí, se podría haber simplificado la programación para haber reducido su complejidad ciclomática y su posterior lectura por desarrolladores ajenos.

Pruebas de Integración

Ahora estudiaremos la clase *BLFacadeImplementation.class* ; que es desde donde se llamará a la clase de acceso a datos.

Diseño - Caja Negra

Para hacer las pruebas de caja negra tomaremos la signatura del método

```
        public boolean gertaerakSortu(String description,Date
    eventDate, String sport) throws EventFinished
```

El método *gertaerakSortu* insertará (devolverá true) un nuevo evento en la base de datos, en una fecha concreta, en un deporte concreto. Si ya hay un evento con mismo nombre y fecha o si el deporte no se encuentra clasificado en la db no se insertará el evento (devuelve false). Además, si la fecha en la que queremos crear el evento es posterior al momento actual, se lanzará la excepción “EventFinished”.

Estudiaremos las clases de equivalencia

Condición de Entrada	Clases de Equivalencia Válidas	Clases de Equivalencia NO Válidas
Deporte clasificado	$sport \in db$ (1)	$sport \notin db$ (2)

Fecha posterior	$eventDate > now()$ (3)	$eventDate <= now()$ (4)
Evento en db	$\nexists e \in db / e.desc = description \wedge e.sport = sport \wedge e.date = date$ (5)	$\exists e \in db / e.desc = description \wedge e.sport = sport \wedge e.date = date$ (6)

Y generaremos una batería de casos de prueba. No obtendremos valores límite porque no trabajamos con números, sino que son todo valores binarios. (Está / No está, Cumple / No cumple, etc...)

Clase(s) Cubiertas	Contexto de Entrada		Contexto de Salida	
	Estado DB	Parámetros	Salida	DB Resultante
2	$sport \notin db$	sport = "Pilla-Pilla"	FALSE	Sin cambios
4	Da igual	$date < now(...)$	EXCEPCIÓN	Sin cambios
1,3,6	$sport \in db$ and $e \in db$	sport="Futbol" description="Madrid-Barça" $date > now(...)$	FALSE	Sin cambios
1,3,5	$sport \in db$ and $e \notin db$	sport="Tennis" description="Alcaraz-Federer" $date > now(...)$	TRUE	Evento (con los parámetros) insertado en la DB

Implementación - Pruebas + Mockito

Como no queremos probar la clase de acceso a datos sino la lógica de negocio, usaremos Mockito para simular cómo de bien está implementada. Para esto, tendremos que emular los métodos a los que se llama desde la clase que estamos probando, *BLFacadeImplementation.class*. En concreto, la clase sólo llama a un método, y éste es *gertaerakSortu* de la clase *DataAccess.class*. Entonces este será el que emularemos.

El fichero GertaerakSortuBLBMTest.java en el que se han implementado los tests queda subido al proyecto de github y también se encuentra ubicado en su correspondiente sección en el anexo.

EmaitzakIpini()

Autor: Xiomara G. Cáceres

Código

Descripción: Este método inserta la respuesta a cada pregunta en base al resultado obtenido en el evento relacionado con la cuota ofrecida por el usuario. Se puede haber obtenido una apuesta *premiada* o *perdida.*

Código a analizar:

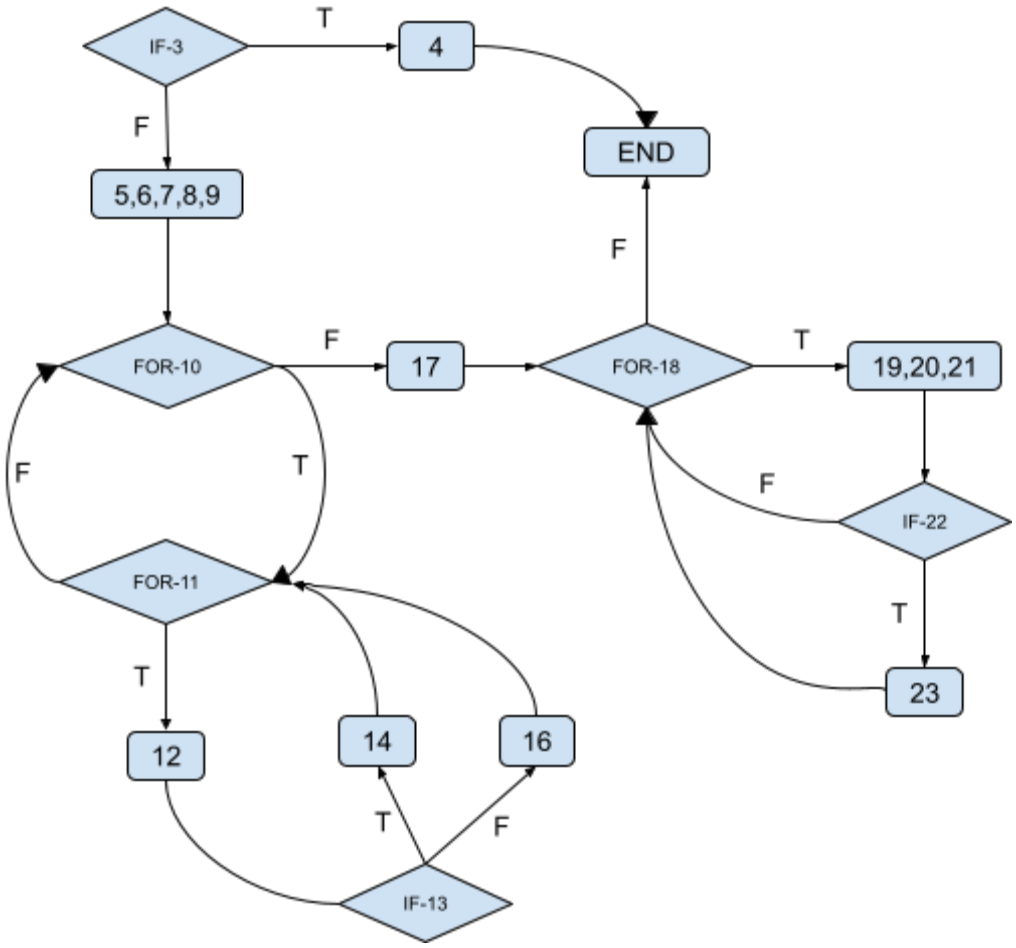
```
public void EmaitzakIpini(Quote quote) throws EventNotFinished{

    Quote q = db.find(Quote.class, quote);
    String result = q.getForecast();

    if(new Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
        throw new EventNotFinished();

    Vector<Apustua> listApustuak = q.getApustuak();
    db.getTransaction().begin();
    Question que = q.getQuestion();
    Question question = db.find(Question.class, que);
    question.setResult(result);
    for(Quote quo: question.getQuotes()) {
        for(Apustua apu: quo.getApustuak()) {

            Boolean b=apu.galdutaMarkatu(quo);
            if(b) {
                apu.getApustuAnitza().setEgoera("galduta");
            }else {
                apu.setEgoera("irabazita");
            }
        }
    }
    db.getTransaction().commit();
    for(Apustua a : listApustuak) {
        db.getTransaction().begin();
        Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
        db.getTransaction().commit();
        if(bool) {
            this.ApustuaIrabazi(a.getApustuAnitza());
        }
    }
}
```

Una copia de este método en formato texto se puede encontrar en el Anexo, en su apartado correspondiente.

Pruebas Unitarias

Diseño - Caja Blanca

1. Grafo de flujo de control



$$V(G) = 6 + 1 = 7$$

2. Caminos

- Camino #1 = IF3 (T) 4 END
- Camino #2 = IF3 (F) 5-9 FOR10 (F) 17 FOR18 (F) END
- Camino #3 = IF3 (F) 5-9 FOR(10) (T) FOR11 (F) FOR10 (F) 17 FOR18 (F) END
- Camino #4 = IF3 (F) 5-9 FOR(10) (T) FOR11 (T) 12 IF13 (F) 16 FOR11 (F) FOR10 (F) 17 FOR18 (F) END
- Camino #5 = IF3 (F) 5-9 FOR(10) (T) FOR11 (T) 12 IF13 (T) 14 FOR11 (F) FOR10 (F) 17 FOR18 (T) 19-21 IF22(T) 23FOR18(F) END
- Camino #6 = IF3 (F) 5-9 FOR(10) (T) FOR11 (T) 12 IF13 (F) 16 FOR11 (F) FOR10 (F) 17 FOR18 (T) 19-21 IF22(F) FOR18(F) END

3. Casos de prueba

#	Camino	Condición	Entrada		Resultado Esperado	
			Parámetros	Estado BD	Salida	Estado BD
1	IF3 (T) 4 END	!(new Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)	quote	quote ∈ BD	EventNotFinished	no hay cambios
2	IF3 (F) 5-9 FOR10 (F) 17 FOR18 (F) END	¿Situación imposible?				
3	IF3 (F) 5-9 FOR(10) (T) FOR11 (F) FOR10 (F) 17 FOR18 (F) END	new Date().compareTo(q.getQuestion().getEvent(). getEventDate())<0 && question.getQuotes().lenght==1 && quo.getApustuak().lenght==0 && listApustuak.lenght==0	quote	quote ∈ BD	no devuelve nada	apuesta *en juego*
4	IF3 (F) 5-9 FOR(10) (T) FOR11 (T) 12 IF13 (F) 16 FOR11 (F) FOR10 (F) 17 FOR18 (F) END	¿Situación imposible?				
5	IF3 (F) 5-9 FOR(10) (T) FOR11 (T) 12 IF13 (T) 14 FOR11 (F) FOR10 (F) 17 FOR18 (T) 19-21 IF22(T) 23 FOR18(F) END	new Date().compareTo(q.getQuestion().getEvent(). getEventDate())<0 && question.getQuotes().lenght==1 && quo.getApustuak().lenght==1 && b && listApustuak.lenght==1 && bool	quote	quote ∈ BD	no devuelve nada	apuesta *perdida*
6	IF3 (F) 5-9 FOR(10) (T) FOR11 (T) 12 IF13 (T) 14 FOR11 (F) FOR10 (F) 17 FOR18 (T) 19-21 IF22(F) FOR18(F) END	new Date().compareTo(q.getQuestion().getEvent(). getEventDate())<0 && question.getQuotes().lenght==1 && !b && listApustuak.lenght==1 && !bool	quote	quote ∈ BD	no devuelve nada	apuesta *premiada*

Diseño - Caja Negra

1. Signatura

   ```
   public void EmaitzakIpini(Quote quote) throws EventNotFinished
   ```

2. Clases de equivalencia

Condición Entrada	Clases de equivalencia válidas	Clases de equivalencia no válidas
Cuota existente	quote $\in$ BD (1)	quote $\notin$ BD (2)
Cuota de evento no terminado	ocurre antes de la fecha actual (3)	ocurre después de la fecha actual (4)

3. Casos de prueba

Clases Cubiertas	Contexto de Entrada		Contexto de Salida	
	Parámetros	Estado BD	Resultado	Estado BD
2	quote	quote $\notin$ BD	NullPointerException	no hay cambios
4	quote	quote $\in$ BD	EventNotFinished	no hay cambios
1,3	quote	quote $\in$ BD && que$\in$ BD	no devuelve nada	se actualiza el estado de la apuesta a la predicción, “perdida” o “premiada”

4. Valores límite

   En este caso tampoco obtendremos valores límite porque estamos evaluando condiciones de tipo binario donde las respuestas son siempre ‘verdadero’ o ‘falso’.

Implementación - Caja Blanca

Se han establecido 7 casos de prueba para alcanzar una cobertura completa del código. Además hemos desarrollado la clase *EmaitzakIpiniDAWTest.java* donde hemos tratado todos los casos de prueba mencionados en ese apartado. Se encuentra tanto en el proyecto de github como en su correspondiente sección en el anexo.

Implementación - Caja Negra

Hemos desarrollado 3 casos de prueba, pero no hemos alcanzado un 100% de la cobertura del código. En esta fase el ingeniero que prueba el método no puede obtener mucha más información así que daría su trabajo en este apartado por finalizado.

Hemos creado *EmaitzakIpiniDABTest.java* para probar esos casos de prueba y funcionan correctamente todos los tests. Se encuentra tanto en el proyecto de github como en su correspondiente sección en el anexo.

Defectos Encontrados

NO hay fallos en la implementación del método pero cabe destacar la complejidad del código ya que se podría haber realizado de una manera más óptima reduciendo así su complejidad ciclomática. Además sería más sencillo para la compresión de futuros desarrolladores.

Pruebas de Integración

Ahora estudiaremos la clase *BLFacadeImplementation.class* que es desde donde se llamará a la clase de acceso a datos. Sin embargo, podemos ver que no se trata nada previamente por lo que las pruebas de caja negra serían las mismas a las expuestas anteriormente.

Implementación - Pruebas + Mockito

Se ha implementado la clase EmaitzakIpiniBLBMTest.java para probar la clase *BLFacadeImplementation.class* en la que emularemos el comportamiento de la BD en el método *EmaitzakIpini(...).* Se encuentra tanto en el proyecto de github como en su correspondiente sección en el anexo.

gertaeraEzabatu()

Autor: Eider Fernández

Código

Descripción: El método gertaeraEzabatu() elimina de la base de datos el evento pasado por parámetro. Antes de eso, comprueba que todas las Question tienen una Result, en caso negativo, se indicará que el evento no puede ser eliminado, si no, el método continuará para seguir analizando el evento introducido. Si la fecha del evento es posterior a la actual, se obtiene la lista de Quotes del evento y se analiza la lista de apuestas para cada Quote. Cada una de esas apuestas es eliminada de la base de datos y se actualiza la cantidad de apuestas en el Sport relacionado con el evento. Finalmente, el evento es eliminado de la base de datos.

Código a analizar:

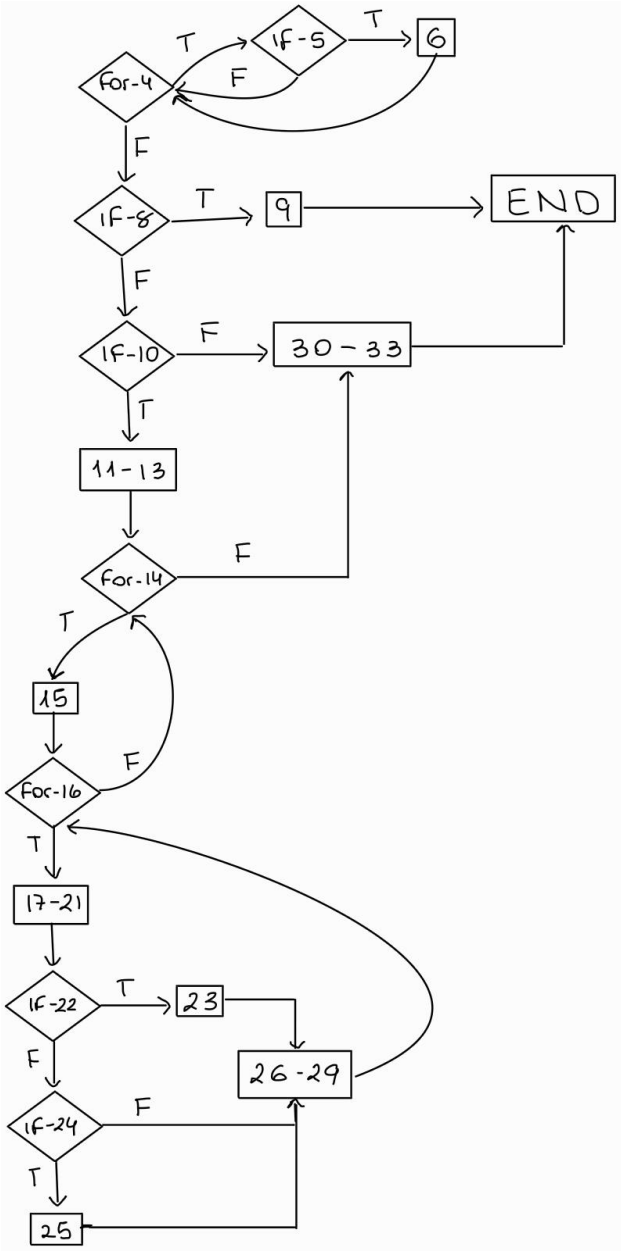
```
public boolean gertaeraEzabatu(Event ev) {
    Event event  = db.find(Event.class, ev);
    boolean resultB = true;
    List<Question> listQ = event.getQuestions();

    for(Question q : listQ) {
        if(q.getResult() == null) {
            resultB = false;
        }
    }
    if(resultB == false) {
        return false;
    }else if(new Date().compareTo(event.getEventDate())<0) {
        TypedQuery<Quote> Qquery = db.createQuery("SELECT q FROM Quote q WHERE q.getQuestion()."
                + "getEvent().getEventNumber() =?1", Quote.class);
        Qquery.setParameter(1, event.getEventNumber());
        List<Quote> listQUO = Qquery.getResultList();
        for(int j=0; j<listQUO.size(); j++) {
            Quote quo = db.find(Quote.class, listQUO.get(j));
            for(int i=0; i<quo.getApustuak().size(); i++) {
                ApustuAnitza apustuAnitza = quo.getApustuak().get(i).getApustuAnitza();
                ApustuAnitza apl = db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
                db.getTransaction().begin();
                apl.removeApustua(quo.getApustuak().get(i));
                db.getTransaction().commit();
                if(apl.getApustuak().isEmpty() && !apl.getEgoera().equals("galduta")) {
                    this.apustuaEzabatu(apl.getUser(), apl);
                }else if(!apl.getApustuak().isEmpty() && apl.irabazitaMarkatu()){
                    this.ApustuaIrabazi(apl);
                }
                db.getTransaction().begin();
                Sport spo =quo.getQuestion().getEvent().getSport();
                spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
                db.getTransaction().commit();
            }
        }

    }
    db.getTransaction().begin();
    db.remove(event);
    db.getTransaction().commit();
    return true;
}
```

Una copia de este método en formato texto se puede encontrar en el Anexo, en su apartado correspondiente.

Pruebas Unitarias

Diseño - Caja Blanca

Primero analizaremos el grafo de diagrama de flujo:



Del cual obtendremos 9 caminos, pues la complejidad ciclomática del método es 9:

- Camino #1 = FOR4 (0) IF8 (F) IF10 (F) 30-33 Fin
- Camino #2 = FOR4 (1) IF5 (F) IF8 (T) IF10 (F) 30-33 Fin
- Camino #3 = FOR4 (1) IF5 (F) IF8 (F) IF10 (F) 30-33 Fin
- Camino #4 = FOR4 (1) IF5 (T) IF8 (T) 9 Fin
- Camino #5 = FOR4 (n) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (0) 30-33 Fin
- Camino #6 = FOR4 (n) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (0) 30-33 Fin
- Camino #7 = FOR4 (n) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (1) 17-21 IF22 (F) IF24 (F) 26-33 Fin
- Camino #8 = FOR4 (n) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (1) 17-21 IF22 (T) 23 IF24 (F) 26-33 Fin
- Camino #9 = FOR4 (n) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (1) 17-21 IF22 (F) IF24 (T) 25-33 Fin

Entonces, definiremos los siguientes casos de prueba

#	Camino	Condición	Entrada		Resultado Esperado	
			Parámetros	Estado BD	Salida	Estado BD
1	FOR4 (0) IF8 (F) IF10 (F) 30-33 Fin	q.length == 0 resultB != false new Date().compareTo(Event.getEventDate()) >= 0	ev	ev ∈ BD, ev no tiene Questions, La fecha del evento es anterior a la actual	true	ev ∉ BD
2	FOR4 (1) IF5 (F) IF8 (T) IF10 (F) 30-33 Fin	Situación imposible				
3	FOR4 (1) IF5 (F) IF8 (F) IF10 (F) 30-33 Fin	q.length == 1 q.getResult != null result != false Date().compareTo(Event.getEventDate()) >= 0	ev	ev ∈ BD, ev tiene 1 Question con resultado, La fecha del evento es anterior a la actual	true	ev ∉ BD

4	FOR4 (1) IF5 (T) IF8 (T) 9 Fin	q.length == 1 q.getResult == null result == false	ev	ev ∈ BD, ev tiene 1 Question sin resultado	false	ev ∈ BD
5	FOR4 (1) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (0) 30-33 Fin	q.length == 1 q.getResult == null result == false Date().compareTo(Event.getEventDate()) >= 0 listQUO.size() > 0	ev	ev ∈ BD, ev no tiene Question sin resultados la fecha del evento es posterior a la actual listQUO.size() == 0	true	ev ∉ BD
6	FOR4 (1) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (0) 30-33 Fin	q.length == 1 q.getResult == null result == false Date().compareTo(Event.getEventDate()) >= 0 listQUO.size() > 0 quo.size() > 0	ev	ev ∈ BD, ev no tiene Question sin resultados la fecha del evento es posterior a la actual listQUO.size() == 1 quo.getApustual.size() == 0	true	ev ∉ BD
7	FOR4 (1) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (1) 17-21 IF22 (F) IF24 (F) 26-33 Fin	q.length == 1 q.getResult == null result == false Date().compareTo(Event.getEventDate()) >= 0 listQUO.size() > 0 quo.size() > 0 ap1.getApustuak().isEmpty && !ap1.getEgoera().equals(“galduta”) !ap1.getApustuak().isEmpty && ap1.getEgoera().equals(“galduta”)	ev	ev ∈ BD, ev no tiene Question sin resultados la fecha del evento es posterior a la actual listQUO.size() == 1 quo.size() == 0 ap1.getApustuak().isEmpty && ap1.getEgoera().equals(“galduta”)	true	spo.Apustuak- - ev ∉ BD
8	FOR4 (1) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16	q.length == 1 q.getResult == null result == false Date().compareTo(Event.getEventDate()) >= 0	ev	ev ∈ BD, ev no tiene Question sin resultados la fecha del	true	ap1 ∉ BD ev ∉ BD

	(1) 17-21 IF22 (T) 23 IF24 (F) 26-33 Fin	listQUO.size() > 0 quo.size() > 0 ap1.getApustuak().isEmpty && !ap1.getEgoera().equals("galduta") !ap1.getApustuak().isEmpty && ap1.irabazitaMarkatu()		evento es posterior a la actual listQUO.size() == 1 quo.size() == 1 ap1.getApustuak().isEmpty && !ap1.getEgoera().equals("galduta")		
9	FOR4 (1) IF5 (F) IF8 (F) IF10 (T) 11-13 FOR14 (1) 15 FOR16 (1) 17-21 IF22 (F) IF24 (T) 25-33 Fin	q.length == 1 q.getResult == null result == false Date().compareTo(Event.getEventDate()) >= 0 listQUO.size() > 0 quo.size() > 0 ap1.getApustuak().isEmpty && !ap1.getEgoera().equals("galduta") !ap1.getApustuak().isEmpty && ap1.irabazitaMarkatu()	ev	ev ∈ BD, ev no tiene Question sin resultados la fecha del evento es posterior a la actual listQUO.size() == 1 quo.size() == 1 !ap1.getApustuak().isEmpty && ap1.irabazitaMarkatu()	true	ap1 aparece como ganada en BD ev ∉ BD

Diseño - Caja Negra

Para hacer las pruebas de caja negra tomaremos la signatura del método

```
public boolean gertaeraEzabatu(Event ev)
```

El método gertaeraEzabatu() elimina de la base de datos el evento pasado por parámetro. Antes de eso, comprueba que todas las Question tienen una Result, en caso negativo, se indicará que el evento no puede ser eliminado, si no, el método continuará para seguir analizando el evento introducido. Si la fecha del evento es anterior a la actual, el evento será eliminado directamente, si no, de ApustuAnitzak se decrementará la cantidad de apuestas en uno y se borrará la apuesta. Además, en caso de que la apuesta haya sido perdida, se marcará como perdida y se decrementa en uno la cantidad de apuestas del deporte. En cambio, si la apuesta ha sido ganada, la apuesta se marca como ganada.

Ahora que hemos entendido lo que hace nuestra función, obtendremos las clases de equivalencia:

Condición de Entrada	Clases de Equivalencia Válidas	Clases de Equivalencia NO Válidas
Preguntas con resultado	Todas las preguntas tienen respuesta (1)	q ∈ listQ, q.getResult == null (2)
Fecha anterior	la fecha del evento es anterior a la actual (3)	
Fecha posterior	la fecha del evento es posterior a la actual (4)	
Apuesta perdida	la apuesta ha sido perdida (5)	
Apuesta ganada	la apuesta ha sido ganada (6)	

Por último, generamos una batería de casos de prueba. Al trabajar con valores binarios y no con números, no habrá que tener en cuenta valores límite:

Nº Caso	Clases Cubiertas	Contexto de Entrada		Contexto de Salida	
		Parámetros	Estado BD	Resultado	Estado BD
1	2	ev	ev ∈ BD	“False”	ev ∈ BD
2	1, 3	ev	ev ∈ BD ev.Date < fecha actual	“True”	ev ∉ BD
3	1, 4	ev	ev ∈ BD ev.Date > fecha actual	“True”	ev ∉ BD en ApustuAnitza se decrementa en uno la cantidad de apuestas del deporte Se elimina la apuesta
4	1, 4, 5	ev	ev ∈ BD ev.Date > fecha actual apuesta perdida	“True”	ev ∉ BD en ApustuAnitza se decrementa en uno la cantidad de apuestas del deporte

					se elimina la apuesta asociada al usuario Se elimina la apuesta
5	1, 4, 6	ev	ev ∈ BD ev.Date > fecha actual apuesta ganada	"True"	ev ∉ BD en ApustuAnitza se decrementa en uno la cantidad de apuestas del deporte la apuesta se guarda como ganada Se elimina la apuesta

Implementación - Caja Blanca

Deberían haberse implementado los 8 casos de prueba definidos en el apartado Diseño - Caja Blanca anterior para alcanzar una cobertura del 100%. Sin embargo, debido a que el método gertaeraEzabatu() evalúa algunos aspectos del evento a eliminar que no podemos recrear ni con la ayuda de la clase TestDataAccess, solo se han implementado 5 de esos 8 tests.

Implementación - Caja Negra

Se han diseñado 5 casos de prueba de caja negra teniendo en cuenta únicamente la información que tendría el ingeniero si solo se le hubiera proporcionado una descripción del método, sin mostrarle el código del mismo. Por ello, aún habiendo conseguido implementar esos 5 casos de prueba con JUnit (cosa que se ha hecho imposible por la misma razón que en el caso anterior), no se habría llegado a alcanzar una cobertura del 100%.

Defectos Encontrados

El método gertaeraEzabatu es muy largo y contiene múltiples niveles de anidación. Esta complejidad dificulta la comprensión y el mantenimiento del código. Sería mejor dividirlo en métodos más pequeños y enfocados en tareas específicas. Además, el código no maneja adecuadamente las excepciones que pueden ocurrir al acceder a la base de datos o al trabajar con fechas. La falta de manejo de excepciones puede resultar en errores inesperados y comportamiento incontrolado. Otro defecto es que el código depende de implementaciones concretas, como db. Esto hace que sea difícil de probar y mantener. Por último, se repite la lógica de inicio y confirmación de transacciones (db.getTransaction().begin() y db.getTransaction().commit()) varias veces. Esto es propenso a errores y redundante.

Pruebas de Integración

Ahora analizaremos la clase BLFacadeImplementation.class, calse que se encargará de llamar a la clase DataAccess. Como no se trata nada previamente, las pruebas de caja negra serían las mismas a las planteadas antes, por lo que no hará falta volver a hacerlas.

Implementación - Pruebas + Mockito

Se ha implementado la clase GertaeraEzabatuBLBMTest.java para probar la clase BLFacadeImplementation.class en la que emularemos el comportamiento de la BD en el método gertaeraEzabatu(). Se encuentra tanto en el proyecto de github como en su correspondiente sección en el anexo.

Anexos

Para una navegación más cómoda en esta sección, se recomienda consultar el índice.

gertaerakSortu(...)

```
public boolean gertaerakSortu(String description,Date eventDate, String
sport) {
	boolean b = true;
	db.getTransaction().begin();
	Sport spo =db.find(Sport.class, sport);
	if(spo!=null) {
		TypedQuery<Event> Equery = db.createQuery("SELECT e FROM Event
e WHERE e.getEventDate() =?1 ",Event.class);
		Equery.setParameter(1, eventDate);
		for(Event ev: Equery.getResultList()) {
			if(ev.getDescription().equals(description)) {
				b = false;
			}
		}
		if(b) {
			String[] taldeak = description.split("-");
			Team lokala = new Team(taldeak[0]);
			Team kanpokoa = new Team(taldeak[1]);
			Event e = new Event(description, eventDate, lokala,
kanpokoa);
			e.setSport(spo);
			spo.addEvent(e);
			db.persist(e);
		}
	}else {
		return false;
	}
	db.getTransaction().commit();
	return b;
```

```
    }
```

GertaerakSortuDAWTest.java

```
@Before
    public void antesDeCada() {
       sut = new DataAccess();
       testDA = new TestDataAccess();
    }

    @Test
    public void testCamino1() {
       // El deporte no existira en la db

       String desc = "Barsa-Madrid";
       Calendar cal = Calendar.getInstance();
       cal.set(2050, 10, 12); // 12 de octubre del 2050
       Date date = cal.getTime();
       String depo = "B4aloncest0";

       // Esperamos que devuelva falso (no se ha insertado porq el
       // deporte no existe
       boolean expected = false;

       boolean actual = sut.gertaerakSortu(desc, date, depo);
       assertEquals(actual, expected);
    }

    @Test
    public void testCamino2() {
       // El deporte SI existira en la bd, entonces NO se insertara

       String desc = "Barsa-Madrid";
       Calendar cal = Calendar.getInstance();
       cal.set(2050, 10, 12); // 12 de octubre del 2050
       Date date = cal.getTime();
       String depo = "Futbol";

       // Insertamos el evento en la db
       Event insertado = testDA.addEventWithQuestion(desc, date,
"PreguntaGenerica", 1); // 1 euro minimum bet, nos da igual

       // Esperamos que devuelva falso (no se ha insertado porq el
       // evento ya existe con esa descripcion en ese dia)
```

```
        boolean expected = false;

        boolean actual = sut.gertaerakSortu(desc, date, depo);
        assertEquals(actual, expected);

        // Quitamos el evento de la db para no contaminar
        testDA.removeEvent(insertado);

    }

    @Test
    public void testCamino3() {
        // Habrá eventos en la fecha consultada, pero no el nuestro.
Entonces SI
        // que se insertará

        String desc = "Barsa-Madrid";
        String desc2 = "Atletic-Real";
        Calendar cal = Calendar.getInstance();
        cal.set(2050, 10, 12); // 12 de octubre del 2050
        Date date = cal.getTime();
        String depo = "Futbol";

        // Insertamos el evento en la db
        Event insertado = testDA.addEventWithQuestion(desc2, date,
"PreguntaGenerica", 1); // 1 euro minimum bet, nos da igual

        // Esperamos que devuelva true (SI se ha insertado porq el
        // evento NO existe con esa descripcion en ese dia)
        boolean expected = true;

        boolean actual = sut.gertaerakSortu(desc, date, depo);
        assertEquals(actual, expected);

        // Quitamos los eventos de la db para no contaminar
        testDA.removeEvent(insertado);
        testDA.removeEventsWithDescDate(desc, date);

    }

    @Test
    public void testCamino4() {
```

```
        // No habrá nada en la fecha consultada. Entonces SI
        // que se insertará

        String desc = "Barsa-Madrid";
        Calendar cal = Calendar.getInstance();
        cal.set(2050, 10, 12); // 12 de octubre del 2050
        Date date = cal.getTime();
        String depo = "Futbol";

        // Esperamos que devuelva true (SI se ha insertado porq no hay
        // eventos ese dia)
        boolean expected = true;

        boolean actual = sut.gertaerakSortu(desc, date, depo);
        assertEquals(actual, expected);

        // Quitamos los eventos de la db para no contaminar
        testDA.removeEventsWithDescDate(desc, date);

    }
```

GertaerakSortuDABTest.java

```
import static org.junit.Assert.assertEquals;
import java.util.Calendar;
import java.util.Date;
import org.junit.*;
import dataAccess.DataAccess;
import domain.Event;
import test.dataAccess.TestDataAccess;

public class GertaerakSortuDABTest {

    DataAccess sut;
    TestDataAccess testDA;

    @Before
    public void antesDeCada() {
       sut = new DataAccess();
       testDA = new TestDataAccess();
    }

    @Test
    public void test1() {
       // Intentamos insertar evento cuyo deporte no existe en la bd

       String sport = "Pilla-pilla";
       String description = "Juan-Nerea";
       Calendar cal = Calendar.getInstance();
       cal.set(2050, 10, 12); // 12 de octubre del 2050
       Date date = cal.getTime();

       boolean expected = false;
       boolean actual = sut.gertaerakSortu(description, date, sport);

       assertEquals(expected,actual);
    }

    @Test
    public void test2() {
       // Intentamos insertar evento que ya está en db
```

```
        String sport = "Futbol";
        String description = "Madrid-Barcelona";
        Calendar cal = Calendar.getInstance();
        cal.set(2050, 10, 12); // 12 de octubre del 2050
        Date date = cal.getTime();

        // Insertamos el evento en bd
        Event insertado = testDA.addEventWithQuestion(description, date,
"Pregunta", 1); // 1€ apuesta minima, nos da igual

        // probamos el metodo que nos interesa
        boolean expected = false;
        boolean actual = sut.gertaerakSortu(description, date, sport);

        assertEquals(expected,actual);
        // Limpiamos db para no contaminar
        testDA.removeEvent(insertado);
    }

    @Test
    public void test3() {
        // Todo bien, insertamos evento que no esta en db y con deporte
existente

        String sport = "Futbol";
        String description = "Madrid-Barcelona";
        Calendar cal = Calendar.getInstance();
        cal.set(2050, 10, 12); // 12 de octubre del 2050
        Date date = cal.getTime();

        // probamos el metodo que nos interesa
        boolean expected = true;
        boolean actual = sut.gertaerakSortu(description, date, sport);

        assertEquals(expected,actual);

        // Limpiamos db para no contaminar
        testDA.removeEventsWithDescDate(description, date);
    }

}
```

GertaerakSortuBLBMTest.java

```
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.mockito.*;
import businessLogic.*;
import exceptions.*;
import dataAccess.*;

public class GertaerakSortuBLBMTest {

    @InjectMocks
    BLFacadeImplementation sut;

    @Mock
    DataAccess da;

    @Before
    public void antesDeCada() {
       da = Mockito.mock(DataAccess.class);
       sut = new BLFacadeImplementation(da);
    }

    @Test
    public void test1() {
       // Intentamos insertar evento cuyo deporte no existe en la bd

       String sport = "Pilla-pilla";
       String description = "Juan-Nerea";
       Calendar cal = Calendar.getInstance();
       cal.set(2050, 10, 12); // 12 de octubre del 2050
       Date date = cal.getTime();

       // Mockeamos
       Mockito.doReturn(false).when(da).gertaerakSortu(description, date,
sport);

       // Lo que esperamos
       boolean expected = false;
       boolean actual;
       try {
```

```
            actual = sut.gertaerakSortu(description, date, sport);
            assertEquals(expected,actual);

        } catch (EventFinished e) {
            // Si salta la excepcion algo ha ido mal
            fail("Algo ha ido mal, no deberia de haber saltado
excepcion");
        }

    }

    @Test
    public void test2() {
        // Intentamos insertar evento (valido) cuya fecha es anterior a
ahora

        String sport = "Futbol";
        String description = "Madrid-Barcelona";
        Calendar cal = Calendar.getInstance();
        cal.set(1995, 10, 12); // 12 de octubre del 1995
        Date date = cal.getTime();

        // No hace falta mockear porq no tocamos la clase DataAccess (se
supone!)

        try {
            sut.gertaerakSortu(description, date, sport);
            // Si no salta excepcion, algo ha ido mal
            fail("No ha saltado una excepcion al intentar insertar fecha
anterior");
        } catch (EventFinished e) {
            // Bien, no ha dejado insertar el evento
            assertTrue(true);
        }

    }

    @Test
    public void test3() {
        // Intentamos insertar evento que ya está en db

        String sport = "Futbol";
```

```
        String description = "Madrid-Barcelona";
        Calendar cal = Calendar.getInstance();
        cal.set(2050, 10, 12); // 12 de octubre del 2050
        Date date = cal.getTime();

        // Mockeamos
        Mockito.doReturn(false).when(da).gertaerakSortu(description, date,
sport);

        // Lo que esperamos
        boolean expected = false;
        boolean actual;
        try {
              actual = sut.gertaerakSortu(description, date, sport);
              assertEquals(expected,actual);

        } catch (EventFinished e) {
              // Si salta la excepcion algo ha ido mal
              fail("Algo ha ido mal, no deberia de haber saltado
excepcion");
        }
    }

    @Test
    public void test4() {
        // Todo bien, insertamos evento que no esta en db y con deporte
existente

        String sport = "Futbol";
        String description = "Madrid-Barcelona";
        Calendar cal = Calendar.getInstance();
        cal.set(2050, 10, 12); // 12 de octubre del 2050
        Date date = cal.getTime();

        // Mockeamos
        Mockito.doReturn(true).when(da).gertaerakSortu(description, date,
sport);

        // Lo que esperamos
        boolean expected = true;
        boolean actual;
        try {
```

```
            actual = sut.gertaerakSortu(description, date, sport);
            assertEquals(expected,actual);

        } catch (EventFinished e) {
            // Si salta la excepcion algo ha ido mal
            fail("Algo ha ido mal, no deberia de haber saltado
excepcion");
        }
    }

}
```

EmaitzakIpini(...)

```
public void EmaitzakIpini(Quote quote) throws EventNotFinished{

		Quote q = db.find(Quote.class, quote);
		String result = q.getForecast();

		if(new
Date().compareTo(q.getQuestion().getEvent().getEventDate())<0)
			throw new EventNotFinished();

		Vector<Apustua> listApustuak = q.getApustuak();
		db.getTransaction().begin();
		Question que = q.getQuestion();
		Question question = db.find(Question.class, que);
		question.setResult(result);
		for(Quote quo: question.getQuotes()) {
			for(Apustua apu: quo.getApustuak()) {

				Boolean b=apu.galdutaMarkatu(quo);
				if(b) {
					apu.getApustuAnitza().setEgoera("galduta");
				}else {
					apu.setEgoera("irabazita");
				}
			}
		}
		db.getTransaction().commit();
		for(Apustua a : listApustuak) {
			db.getTransaction().begin();
			Boolean bool=a.getApustuAnitza().irabazitaMarkatu();
			db.getTransaction().commit();
			if(bool) {
				this.ApustuaIrabazi(a.getApustuAnitza());
			}
		}
	}
```

EmaitzakIpiniDAWTest.javaw

```
import static org.junit.Assert.*;

import java.util.*;
import org.junit.*;

import dataAccess.*;
import domain.*;
import exceptions.*;

public class EmaitzakIpiniDAWTest {

	DataAccess sut;
	Calendar calendar;
	Quote q, q2;

	@Before
	public void antesDeCada() {
		sut = new DataAccess();
		calendar = Calendar.getInstance();
	}

	@Test
	public void testCamino1() {
		//La fecha de ese evento es posterior a la fecha actual
		calendar.set(2023, Calendar.NOVEMBER, 01, 0, 0, 0);
		Collection<Quote> quote = sut.findQuote(new Question(15, "Who
will win the match?", 1.0, new Event(21, "Atletico-Athletic",
calendar.getTime(), new Team("Atletico"), new Team("Athletic"))));
		for (Quote qt: quote)
		{
			if (qt.getForecast().equals("2")) q=qt;
		}
		assertThrows(EventNotFinished.class, () ->
{sut.EmaitzakIpini(q);});
	}


	@Test
	public void testCamino3()  {
		//Una cuota sin apuestas
```

```
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            Collection<Quote> quote = sut.findQuote(new Question(19,
"Emaitza?", 1.0, new Event(21, "Real Madrid-Barcelona", calendar.getTime(),
new Team("Real Madrid"), new Team("Barcelona"))));
            for (Quote qt: quote)
            {
                  if (qt.getForecast().equals("1")) q=qt;
            }
            try {
                  sut.EmaitzakIpini(q);
                  Vector<Quote>quotes = q.getQuestion().getQuotes();
                  assertEquals("jokoan",
quotes.get(0).getApustuak().get(0).getApustuAnitza().getEgoera());
            } catch (EventNotFinished e) {
                  fail("Todo debería ir bien");
            }
      }

      @Test
      public void testCamino5() {
            //Cuota con apuestas y sera una puesta "perdida"
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            Collection<Quote> quote = sut.findQuote(new Question(19,
"Emaitza?", 1.0, new Event(21, "Real Madrid-Barcelona", calendar.getTime(),
new Team("Real Madrid"), new Team("Barcelona"))));
            for (Quote qt: quote)
            {
                  if (qt.getForecast().equals("2")) q=qt;
            }
            try {
                  sut.EmaitzakIpini(q);
                  Vector<Quote>quotes = q.getQuestion().getQuotes();
                  assertEquals("galduta",
quotes.get(0).getApustuak().get(2).getApustuAnitza().getEgoera());

            } catch (EventNotFinished e) {
                  fail("Todo debería ir bien");
            }

      }

      @Test
```

```
	public void testCamino6() {
		//Cuota con apuestas y sera una puesta "perdida"
		calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
		Collection<Quote> quote = sut.findQuote(new Question(19,
"Emaitza?", 1.0, new Event(21, "Real Madrid-Barcelona", calendar.getTime(),
new Team("Real Madrid"), new Team("Barcelona"))));
		for (Quote qt: quote)
		{
			if (qt.getForecast().equals("2")) q=qt;
		}
		try {
			sut.EmaitzakIpini(q);
			Vector<Quote>quotes = q.getQuestion().getQuotes();
			assertEquals("irabazita",
quotes.get(0).getApustuak().get(1).getApustuAnitza().getEgoera());

		} catch (EventNotFinished e) {
			fail("Todo debería ir bien");
		}
	}

}
```

EmaitzakIpiniDABTest.java

```
import static org.junit.Assert.*;

import java.util.*;
import org.junit.*;

import dataAccess.*;
import domain.*;
import exceptions.*;

public class EmaitzakIpiniDABTest {

	DataAccess sut;
	Calendar calendar;
	Quote q, q2;

	@Before
	public void antesDeCada() {
		sut = new DataAccess();
		calendar = Calendar.getInstance();
	}

	@Test
	public void test1() {
		//La cuota no existe en la BD
		q = new Quote(2.0, "232");
		assertThrows(NullPointerException.class,() ->
{sut.EmaitzakIpini(q);});
	}

	@Test
	public void test2() {
		//La fecha de ese evento es posterior a la fecha actual
		calendar.set(2023, Calendar.NOVEMBER, 01, 0, 0, 0);
		Collection<Quote> quote = sut.findQuote(new Question(15, "Who
will win the match?", 1.0, new Event(21, "Atletico-Athletic",
calendar.getTime(), new Team("Atletico"), new Team("Athletic"))));
		for (Quote qt: quote)
		{
			if (qt.getForecast().equals("2")) q=qt;
		}
```

```
            assertThrows(EventNotFinished.class, () ->
{sut.EmaitzakIpini(q);});
      }

      @Test
      public void test3_1() {
            //Una cuota sin apuestas
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            Collection<Quote> quote = sut.findQuote(new Question(19,
"Emaitza?", 1.0, new Event(21, "Real Madrid-Barcelona", calendar.getTime(),
new Team("Real Madrid"), new Team("Barcelona"))));
            for (Quote qt: quote)
            {
                  if (qt.getForecast().equals("1")) q=qt;
            }
            try {
                  sut.EmaitzakIpini(q);
                  Vector<Quote>quotes = q.getQuestion().getQuotes();
                  assertEquals("jokoan",
quotes.get(0).getApustuak().get(0).getApustuAnitza().getEgoera());
            } catch (EventNotFinished e) {
                  fail("Todo debería ir bien");
            }
      }

      @Test
      public void test3_2() {
            //Cuota con apuestas y sera una puesta "perdida"
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            Collection<Quote> quote = sut.findQuote(new Question(19,
"Emaitza?", 1.0, new Event(21, "Real Madrid-Barcelona", calendar.getTime(),
new Team("Real Madrid"), new Team("Barcelona"))));
            for (Quote qt: quote)
            {
                  if (qt.getForecast().equals("2")) q=qt;
            }
            try {
                  sut.EmaitzakIpini(q);
                  Vector<Quote>quotes = q.getQuestion().getQuotes();
                  assertEquals("galduta",
quotes.get(0).getApustuak().get(2).getApustuAnitza().getEgoera());
```

```
			} catch (EventNotFinished e) {
				fail("Todo debería ir bien");
			}
		}

		@Test
		public void test3_3() {
			//Cuota con apuestas y sera una puesta "perdida"
			calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
			Collection<Quote> quote = sut.findQuote(new Question(19,
"Emaitza?", 1.0, new Event(21, "Real Madrid-Barcelona", calendar.getTime(),
new Team("Real Madrid"), new Team("Barcelona"))));
			for (Quote qt: quote)
			{
				if (qt.getForecast().equals("2")) q=qt;
			}
			try {
				sut.EmaitzakIpini(q);
				Vector<Quote>quotes = q.getQuestion().getQuotes();
				assertEquals("irabazita",
quotes.get(0).getApustuak().get(1).getApustuAnitza().getEgoera());

			} catch (EventNotFinished e) {
				fail("Todo debería ir bien");
			}
		}

}
```

EmaitzakIpiniBLBMTest.java

```
import static org.junit.Assert.*;
import java.util.*;
import org.junit.*;
import org.mockito.*;

import businessLogic.*;
import exceptions.*;
import dataAccess.*;

import domain.Event;
import domain.Question;
import domain.Quote;
import domain.Team;

public class EmaitzakIpiniBLBMTest {
      Calendar calendar;
      Quote q;

      @InjectMocks
      BLFacadeImplementation sut;

      @Mock
    DataAccess da;

    @Before
    public void antesDeCada() {
      calendar = Calendar.getInstance();
      da = Mockito.mock(DataAccess.class);
      sut = new BLFacadeImplementation(da);
    }

    @Test
      public void test1() {
            //La cuota no existe en la BD
      q = new Quote(245.0, "2sedrf");
      try {
                  Mockito.doThrow(new
NullPointerException()).when(da).EmaitzakIpini(q);
                  da.EmaitzakIpini(q);
                  fail("No deberia de ir bien");
```

```
            } catch (EventNotFinished | NullPointerException e) {
                  assertTrue(true);
            }
      }

      @Test
      public void test2() {
            //La fecha de ese evento es posterior a la fecha actual
            calendar.set(2023, Calendar.NOVEMBER, 01, 0, 0, 0);
            q = new Quote(100.0, "2", new Question(15, "Who will win the
match?", 1.0, new Event(21, "Atletico-Athletic", calendar.getTime(), new
Team("Atletico"), new Team("Athletic"))));
            try {
                  Mockito.doThrow(new
EventNotFinished()).when(da).EmaitzakIpini(q);
                  da.EmaitzakIpini(q);
                  fail("No deberia de ir bien");
            } catch (EventNotFinished e) {
                  assertTrue(true);
            }

      }

      @Test
      public void test3_1() {
            //Una cuota sin apuestas
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            q = new Quote(5.0, "1", new Question(19, "Emaitza?", 1.0, new
Event(21, "Real Madrid-Barcelona", calendar.getTime(), new Team("Real
Madrid"), new Team("Barcelona"))));

            try {
                  da.EmaitzakIpini(q);
                  Mockito.verify(da,
Mockito.times(1)).EmaitzakIpini(Mockito.any(Quote.class));
            } catch (EventNotFinished e) {
                  fail("Todo debería ir bien");
            }
      }

      @Test
      public void test3_2() {
```

```
            //Cuota con apuestas y sera una puesta "perdida"
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            q = new Quote(2.5, "2", new Question(19, "Emaitza?", 1.0, new
Event(21, "Real Madrid-Barcelona", calendar.getTime(), new Team("Real
Madrid"), new Team("Barcelona"))));
            try {
                  da.EmaitzakIpini(q);
                  Mockito.verify(da,
Mockito.times(1)).EmaitzakIpini(Mockito.any(Quote.class));
            } catch (EventNotFinished e) {
                  fail("Todo debería ir bien");
            }
      }

      @Test
      public void test3_3() {
            //Cuota con apuestas y sera una puesta "premiada"
            calendar.set(2023, Calendar.SEPTEMBER, 21, 0, 0, 0);
            q = new Quote(2.5, "2", new Question(19, "Emaitza?", 1.0, new
Event(21, "Real Madrid-Barcelona", calendar.getTime(), new Team("Real
Madrid"), new Team("Barcelona"))));
            try {
                  da.EmaitzakIpini(q);
                  Mockito.verify(da,
Mockito.times(1)).EmaitzakIpini(Mockito.any(Quote.class));
            } catch (EventNotFinished e) {
                  fail("Todo debería ir bien");
            }
      }
}
```

GertaeraEzabatu(...)

```
public boolean gertaeraEzabatu(Event ev) {
            Event event  = db.find(Event.class, ev);
            boolean resultB = true;
            List<Question> listQ = event.getQuestions();

            for(Question q : listQ) {
                  if(q.getResult() == null) {
                        resultB = false;
                  }
            }
            if(resultB == false) {
                  return false;
            }else if(new Date().compareTo(event.getEventDate())<0) {
                  TypedQuery<Quote> Qquery = db.createQuery("SELECT q FROM
Quote q WHERE q.getQuestion().getEvent().getEventNumber() =?1",
Quote.class);
                  Qquery.setParameter(1, event.getEventNumber());
                  List<Quote> listQUO = Qquery.getResultList();
                  for(int j=0; j<listQUO.size(); j++) {
                        Quote quo = db.find(Quote.class, listQUO.get(j));
                        for(int i=0; i<quo.getApustuak().size(); i++) {
                              ApustuAnitza apustuAnitza =
quo.getApustuak().get(i).getApustuAnitza();
                              ApustuAnitza ap1 =
db.find(ApustuAnitza.class, apustuAnitza.getApustuAnitzaNumber());
                              db.getTransaction().begin();
                              ap1.removeApustua(quo.getApustuak().get(i));
                              db.getTransaction().commit();
                              if(ap1.getApustuak().isEmpty() &&
!ap1.getEgoera().equals("galduta")) {
                                    this.apustuaEzabatu(ap1.getUser(),
ap1);
                              }else if(!ap1.getApustuak().isEmpty() &&
ap1.irabazitaMarkatu()){
                                    this.ApustuaIrabazi(ap1);
                              }
                              db.getTransaction().begin();
                              Sport spo
```

```
=quo.getQuestion().getEvent().getSport();

spo.setApustuKantitatea(spo.getApustuKantitatea()-1);
                                db.getTransaction().commit();
                            }
                        }

                }
                db.getTransaction().begin();
                db.remove(event);
                db.getTransaction().commit();
                return true;
        }
```

GertaeraEzabatuDAWTest

```
import static org.junit.Assert.*;

import java.util.Calendar;
import java.util.Date;

import javax.persistence.EntityManager;

import org.junit.Before;
import org.junit.Test;

import configuration.UtilDate;
import dataAccess.DataAccess;
import domain.Event;
import domain.Question;
import domain.Quote;
import domain.Team;
import exceptions.QuestionAlreadyExist;
import exceptions.QuoteAlreadyExist;
import test.dataAccess.TestDataAccess;

public class GertaeraEzabatuDAWTest {

	DataAccess sut;
	TestDataAccess tda;
	EntityManager db;
	Calendar today;
	int month;
	int year;
	boolean result;
	boolean expected;

	@Before
	public void setUp(){
		sut = new DataAccess();
		tda = new TestDataAccess();
		today = Calendar.getInstance();
		month=today.get(Calendar.MONTH);
		month+=1;
		year=today.get(Calendar.YEAR);
		if (month==12) { month=0; year+=1;}
```

```
	}

	@Test
	public void testCamino1() {
		Event ev = tda.addEventWithNoQuestion("Raimon-Royal Academy",
UtilDate.newDate(year, month, 10));
		result = sut.gertaeraEzabatu(ev);
		expected = true;
		assertEquals(result, expected);
	}

	@Test
	public void testCamino3() {
		Event ev = tda.addEventWithQuestionAndResult("Raimon-Royal
Academy", UtilDate.newDate(year-1, month, 10), "¿Quién ganará?", 1,
"Raimon");
		result = sut.gertaeraEzabatu(ev);
		expected = true;
		assertEquals(result, expected);
	}

	@Test
	public void testCamino4() {
		Event ev = tda.addEventWithQuestion("Raimon-Royal Academy",
UtilDate.newDate(year, month, 10), "¿Quién ganará?", 1);
		result = sut.gertaeraEzabatu(ev);
		expected = false;
		assertEquals(result, expected);
	}

	@Test
	public void testCamino5() {
		Event ev = tda.addEventWithQuestionAndResult("Raimon-Royal
Academy", new Date(), "¿Quién ganará?", 1, "Raimon");
		result = sut.gertaeraEzabatu(ev);
		expected = true;
		assertEquals(result, expected);
	}

}
```

GertaeraEzabatuDABTest

```
import static org.junit.Assert.*;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import javax.persistence.EntityManager;

import org.junit.Before;
import org.junit.Test;

import configuration.UtilDate;
import dataAccess.DataAccess;
import test.dataAccess.TestDataAccess;
import domain.*;
public class GertaeraEzabatuDABTest {
	DataAccess sut;
	TestDataAccess tda;
	EntityManager db;
	Calendar today;
	int month;
	int year;
	boolean result;
	boolean expected;

	@Before
	public void setUp(){
		sut = new DataAccess();
		tda = new TestDataAccess();
		today = Calendar.getInstance();
		month=today.get(Calendar.MONTH);
		month+=1;
		year=today.get(Calendar.YEAR);
		if (month==12) { month=0; year+=1;}

	}

	@Test
	public void test1() {
```

```
                Event ev = tda.addEventWithQuestion("Raimon-Royal
Academy", UtilDate.newDate(year, month, 30), "¿Quién ganará el partido?",
1);
                result = sut.gertaeraEzabatu(ev);
                expected = false;
                assertEquals(result, expected);
                tda.removeEvent(ev);
        }


        @Test
        public void test2() {
                Event ev = tda.addEventWithQuestionAndResult("Raimon-Royal
Academy", UtilDate.newDate(year, month, 10), "¿Quién ganará?", 1,
"Raimon");
                result = sut.gertaeraEzabatu(ev);
                expected = true;
                assertEquals(result, expected);
        }

}
```

GertaeraEzabatuBLBMTest

```
import static org.junit.Assert.*;
import org.junit.Test;

import java.util.*;
import org.junit.*;
import org.mockito.*;

import businessLogic.*;
import configuration.UtilDate;
import exceptions.*;
import dataAccess.*;
import domain.Event;
import domain.Question;
import domain.Team;

public class GertaeraEzabatuBLBMTest {

	@InjectMocks
	BLFacadeImplementation sut;

	@Mock
    DataAccess da;

	int month;
	int year;
	Calendar today;
	boolean result;
	boolean expected;

    @Before
    public void setUp() {
	  da = Mockito.mock(DataAccess.class);
	  sut = new BLFacadeImplementation(da);
	  today = Calendar.getInstance();
		month=today.get(Calendar.MONTH);
		month+=1;
		year=today.get(Calendar.YEAR);
		if (month==12) { month=0; year+=1;}
    }
	@Test
```

```
	public void test1() {
		Event ev = new Event("Raimon-Royal Academy",
UtilDate.newDate(year, month, 01), new Team("Raimon"), new Team("Royal
academy"));
		Mockito.doReturn(false).when(da).gertaeraEzabatu(ev);
		boolean result;
		boolean expected = false;

		try {
			result = sut.gertaeraEzabatu(ev);
			assertEquals(result, expected);

		} catch(Exception e) {
			 fail("Algo ha ido mal.");
		}
	}



	@Test
	public void test2() {
		Event ev = new Event("Raimon-Royal Academy",
UtilDate.newDate(year-1, month, 01), new Team("Raimon"), new Team("Royal
academy"));
		Question q = ev.addQuestion("¿Quién gana?", 1.5);
		q.setResult("Raimon");
		Mockito.doReturn(true).when(da).gertaeraEzabatu(ev);
		boolean result;
		boolean expected = true;

		try {
			result = sut.gertaeraEzabatu(ev);
			assertEquals(result, expected);

		} catch(Exception e) {
			 fail("Algo ha ido mal.");
		}
	}

	@Test
	public void test3() {
		Event ev = new Event("Raimon-Royal Academy",
```

```
UtilDate.newDate(year, month, 01), new Team("Raimon"), new Team("Royal
academy"));
            Mockito.doReturn(true).when(da).gertaeraEzabatu(ev);
            Question q = ev.addQuestion("¿Quién gana?", 1.5);
            q.setResult("Raimon");
            boolean result;
            boolean expected = true;

            try {
                 result = sut.gertaeraEzabatu(ev);
                 assertEquals(result, expected);

            } catch(Exception e) {
                  fail("Algo ha ido mal.");
            }
      }

      @Test
      public void test4() {
            Event ev = new Event("Raimon-Royal Academy",
UtilDate.newDate(year, month, 30), new Team("Raimon"), new Team("Royal
academy"));
            Mockito.doReturn(true).when(da).gertaeraEzabatu(ev);
            Question q = ev.addQuestion("¿Quién gana?", 1.5);
            q.setResult("Raimon");
            boolean result;
            boolean expected = true;

            try {
                 result = sut.gertaeraEzabatu(ev);
                 assertEquals(result, expected);

            } catch(Exception e) {
                  fail("Algo ha ido mal.");
            }
      }

      @Test
      public void test5() {
            Event ev = new Event("Raimon-Royal Academy",
UtilDate.newDate(year, month, 30), new Team("Raimon"), new Team("Royal
academy"));
```

```
            Mockito.doReturn(true).when(da).gertaeraEzabatu(ev);
            Question q = ev.addQuestion("¿Quién gana?", 1.5);
            q.setResult("Raimon");
            boolean result;
            boolean expected = true;

            try {
                 result = sut.gertaeraEzabatu(ev);
                 assertEquals(result, expected);

            } catch(Exception e) {
                  fail("Algo ha ido mal.");
            }
      }

}
```