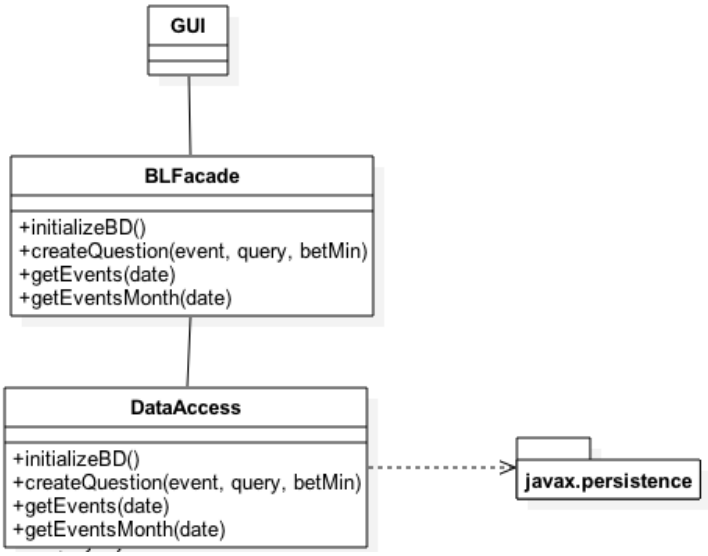
ARQUITECTURA PARA DESARROLLAR LOS CASOS DE PRUEBA

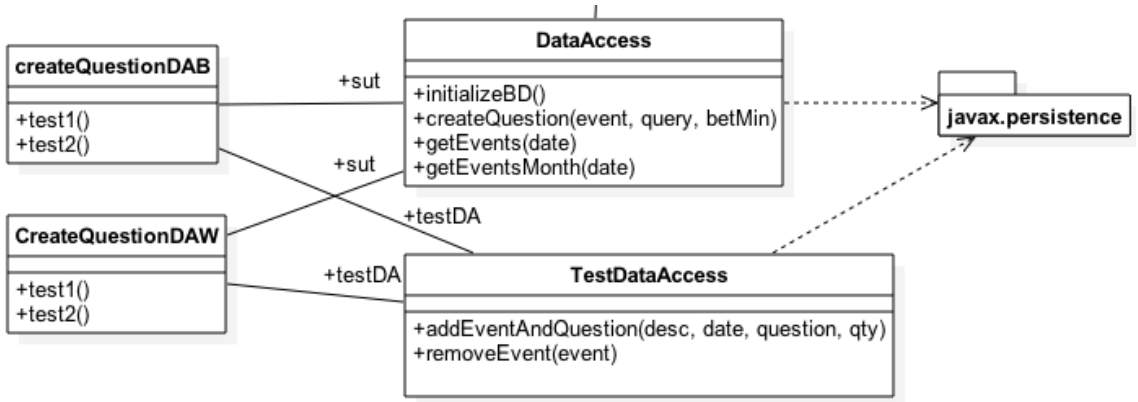
Primera arquitectura de la aplicación

Como se puede ver en la primera imagen, la aplicación está diseñada en 3 niveles:
1.GUI-s o interfaces gráficas .
2. BLFacade o Lógica de Negocio, donde están ubicadas las funcionalidades de la aplicación.
3. DataAccess o acceso a datos, es decir, la clases que gestiona cómo se almacenan los datos. Esta clase utiliza el paquete javax.persistance para almacenar los datos en la BD (en nuestro caso, *objectdb*).



Arquitectura para probar DataAccess

En este caso, queremos probar los métodos de la clase DataAccess, para eso crearemos una clase para cada método que queremos probar, el nombre de la clase será nombreMetodo+DAW(DataAccessWhite) si vamos a implementar las pruebas de caja blanca, o nombreMetodoDAB (DataAccessBlack) si lo que queremos en implementar las pruebas de caja negra. Por ejemplo, CreateQuestionDAW, CreateQuestionDAB getEventsDAW, etc..... En la siguiente figura puedes ver la arquitectura:

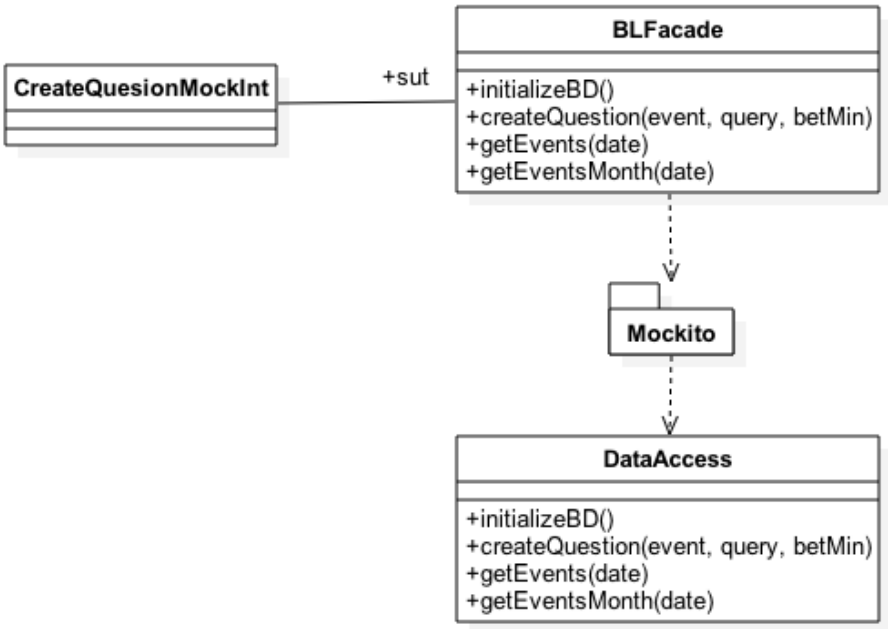


Cada clase implementa las pruebas de un método de la clase DataAccess. En el atributo **sut** (System Under Test) se guarda un objeto de la clase DataAccess, y en

la clase TestDataAccess se implementarán los métodos adicionales que son necesarios para ejecutar los test. Por ejemplo, para probar el método getEvents(), primera hay que crear unos eventos en la BD, pero este método no está implementado aun en la clase DataAccess, y eso método NO lo podemos implementar en esa clase, ya que no podemos modificar la clase DataAccess. Por eso, para crear eventos (addEventAndQuestion) o para luego eliminarlos (removeEvent) habrá que implementar esos métodos en la clase TestDataAccess.

Arquitectura para probar FacadeInplementation (utilizando el Mock de DataAccess)

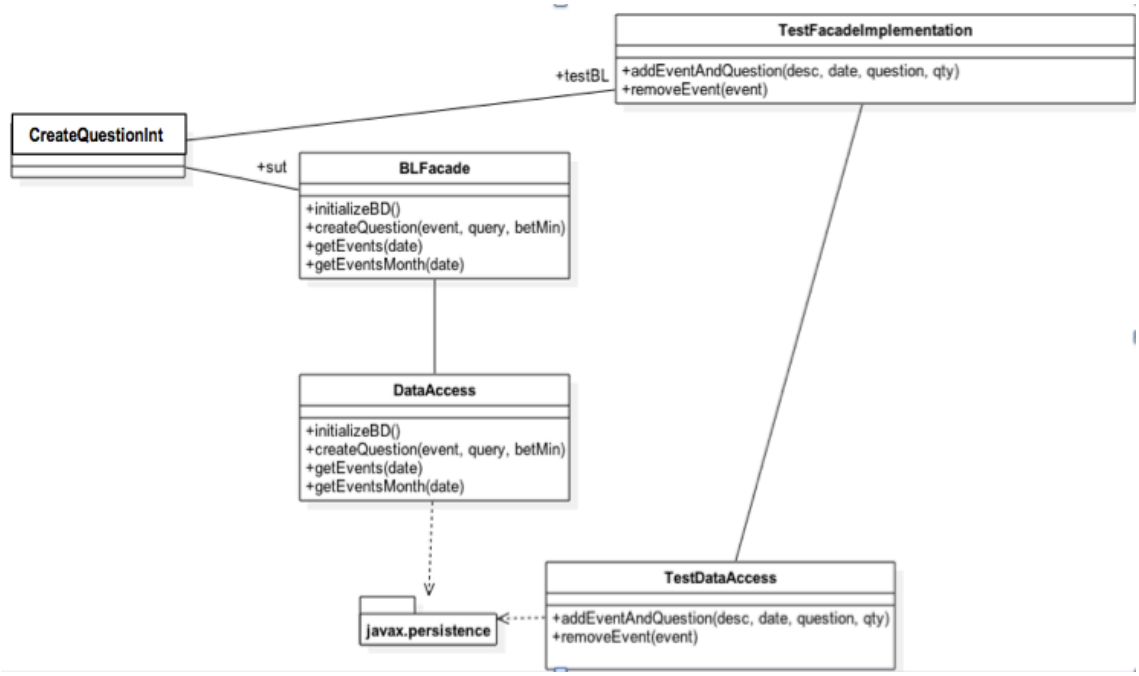
En este caso, probaremos los métodos de la Lógica de Negocio. Para ello, crearemos una clase con el nombre nombreMetodo+MockInt (Mockito + Integración) para cada método de la clase BLFacade que queramos probar (porEjemplo, CreateQuestionMockInt)



En esta arquitectura, utilizaremos una clase que simula a DataAcess, como un actor "doble" de una película. Para ello, utilizaremos *Mockito*.

Arquitectura para probar FacadeInplementation (utilizando DataAccess)

En este apartado, la clase BLFacadeImplementation para probar sus métodos, utilizará la clase DataAccess. Para cada método que se quiere probar, se creará un clase con el nombre nombreMetodo+Int, por ejemplo CreateQuestionInt.java.



En este caso también, si necesitamos algún método adicional para realizar el test, lo implementaremos en la clase TestFacadeImplementation.

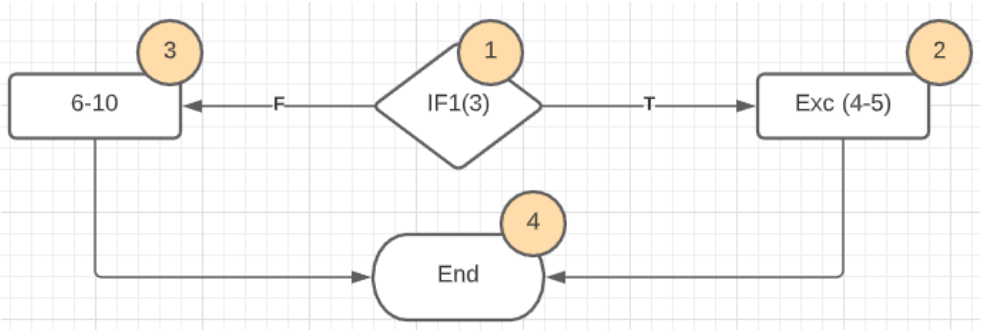
Implementación del proyecto de pruebas (en las diferentes arquitecturas)

En el proyecto https://github.com/jononekin/Bets2021 en la carpeta src/test/java puedes encontrar un ejemplo para el método **createQuestion** en las 4 arquitectura diferentes que se han expuesto en este documento. Además en la carpeta src/test/java/test puedes encontrar las clases TestFacadeImplementation.java y TestDataAccess.java, necesarias para ejecutar los casos de prueba.

createQuestion ANALISIS DE CAJA BLANCA (DataAccess.java)

```
/**
         * This method creates a question for an event, with a question text and the minimum bet
         *
         * @param event to which question is added
         * @param question text of the question
         * @param betMinimum minimum quantity of the bet
         * @return the created question, or null, or an exception
         * @throws QuestionAlreadyExist if the same question already exists for the event
         */
1        public Question createQuestion(Event event, String question, float betMinimum) throws  QuestionAlreadyExist {
2           Event ev = db.find(Event.class, event.getEventNumber());
3           if (ev.DoesQuestionExists(question))
4                   throw new QuestionAlreadyExist(ResourceBundle.getBundle("Etiquetas").
5                                                                                   getString("ErrorQueryAlreadyExist"));
6                   db.getTransaction().begin();
7                   Question q = ev.addQuestion(question, betMinimum);
8                   db.persist(ev);
9                   db.getTransaction().commit();
10                  return q;
11       }
```

Grafo de flujo de control:



V(G)=**2**

Tabla de prueba:

#	Condición	Contexto de prueba		Resultado esperado	
		Estado BD	Entrada	Estado BD	Salida
1	IF4(T)	e∈BD, **q'∈e,** q'∈BD	e, q, 2.0	No cambia	Exception QuestionAlreadyExist
2	IF4(F)	e∈BD, **q'∉e**	e, q, 2.0	q'∈BD **q'∈e**	q'

valores: e=(#, “event1”, 05/10/2022), q="query1", q'=(e,q, 2.0)

Inplementación: **createQuestionDAW.java**
Defectos encontrados: Todos los casos de prueba han dado los resultados esperados.

createQuestion ANALISIS DE CAJA NEGRA (DataAccess.java)

public Question createQuestion(Event event, String question, **float** betMinimum) **throws** QuestionAlreadyExist

Condición entrada	Clases de equivalencia válidad	Clases de equivalencia inválidas
valor de Event	Event!=null (1)	Event==null (7)
valor de Question	Question!=null (2)	Question==null (8)
betMinimum	betMinimum>=2 (3)	betMinimum<2 (9)
Event finalizado	Event.date> now (4)	Event.date < now (10)
Event en BD	Event ∈ BD (5)	Event **∉BD** (11)
Event no tiene Question	question ∉ event.questions (6)	question ∈ event.questions (12)

Casos de prueba:

#	Clases cubiertas	Contexto de prueba		Resultado esperado	
		Estado BD	Entrada	Estado BD	Salida (Question)
1	1,2,3,4,5,6	e∈BD, q'**∉e**	e, q, 2.0	q'∈BD, **q'∈e**	q'
2	7	*	null,q,2.0	*	null
3	8	e∈BD	e,null,2.0	**q'∈e** q'**∉**BD	null
4	9	*	e,q,1.0	*	null
5	10	*	e1,q,4	*	null
6	11	e**∉**BD	e,q,2.0	e**∉**BD	null
7	12	e∈BD, q'∈**e**	e,q,5.0	Ez da aldatzen	QuestionAlreadyExist

valores: e=(#, “event1”, 05/10/2022), e1=(#, “event1”, 05/10/2020), q= “query1”, q'=(e,q, 2.0)

Implementación: createQuestionDAB.java (implementados los casos: 1,2,3,7)

Defectos encontrados:

#	Contexto de prueba		Resultado esperado		Resultado devuelto	
	Estado BD	Entrada	Estado BD	Salida (Question)	Estado BD	Salida (Question)
2	*	null, q, 2.0	*	null	*	NullPointerException
3	*	e,null,2.0	e∈BD	null	?	NullPointerException

createQuestion ANALISIS DE CAJA NEGRA (BLFacadeImplementation.java)

```
public Question createQuestion(Event event, String question, float betMinimum)
throws EventFinished, QuestionAlreadyExist
```

Condición entrada	Clases de equivalencia válidad	Clases de equivalencia inválidas
valor de Event	Event!=null (1)	Event==null (7)
valor de Question	Question!=null (2)	Question==null (8)
betMinimum	betMinimum>=2 (3)	betMinimum<2 (9)
Event finalizado	Event.date> now (4)	Event.date < now (10)
Event en BD	Event ∈ BD (5)	Event ∉ BD (11)
Event no tiene Question	question ∉ event.questions (6)	question ∈ event.questions (12)

Casos de prueba

#	Clases cubiertas	Contexto de prueba		Resultado esperado	
		Estado BD	Entrada	Estado BD	Salida (Question)
1	1,2,3,4,5,6	e∈BD, q'∉e	e, q, 2.0	q'∈BD, q'∈e	q'
2	7	*	null,q,2.0	*	null
3	8	e∈BD	e,null,2.0	q'∈e q'∉BD	null
4	9	*	e,q,1.0	*	null
5	10	*	e1,q,4	*	EventFinished
6	11	e∉BD	e,q,2.0	e∉BD	null
7	12	e∈BD, q'∈e	e,q,5.0	No cambia	QuestionAlreadyExist

valores: e=(#, “event1”, 05/10/2022), e1=(#, “event1”, 05/10/2020), q= “query1”, q'=(e,q, 2.0)

Implementación: CreateQuestionMockInt.java (implementados los casos: 1,2,7)

Defectos encontrados:

#	Proba kontextua		Itxaron emaitzak		Itzulitako emaitzak	
	DB Egoera	Sarrera	DB Egoera	Irteera (Question)	DB Egoera	Irteera (Question)
2	*	null, q, 2.0	*	null	*	NullPointerException