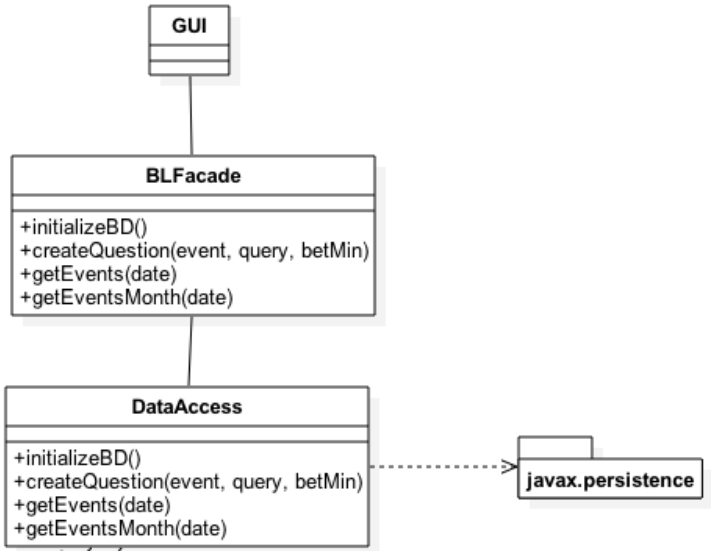
PROBA KASUAK GARATZEKO ARKITEKTURA

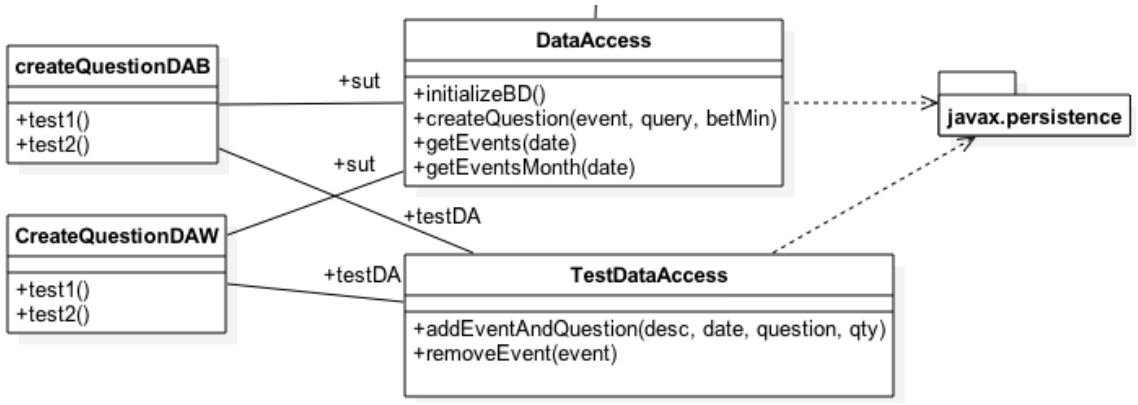
Aplikazioaren Lehendabiziko Arkitekturak

Hurrengo irudian agertzen den bezala, aplikazioa 3 mailatan banatuta dago:
1. GUI-ak edo interfaze grafikoak
2. BLFacade edo NegozioLogika, hau da, aplikazioren funtzionalitateak
3. DataAccess edo datu atzipena, hau da, datuak nola gordetzen diren kudeatzailea. Klase honek javax.persistance paketea erabiliko du datuak db batean gordetzeko (gure kasuan, objectdb)



DataAccess frogatzeko Arkitektura

Kasu honetan DataAccess klasean dauden metodoak frogatu nahi ditugu, horretarako frogatu nahi dugun medoto bakoitzarentzat klase bat sortuko dugu, metodaren izenarekin+DAW (DataAccessWhite) kutxa txuriko frogak inplementatzen baditugu edo , metodaren izena+DAB (DataAccessBlack) kutxa beltzekoak badira, hau da, CreateQuestionDAW, CreateQuestionDAB getEventsDAW, etb..... Hurrengo irudian ikusi dezakazu arkitektura

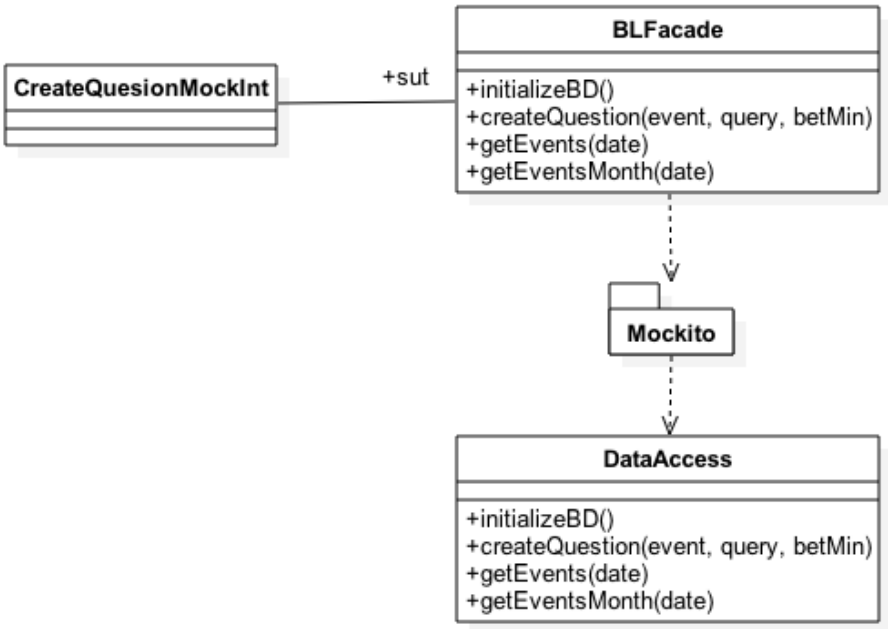


Klase bakoitzak DataAccess-eko metodo baten frogak inplementatzen ditu, **sut** (System Under Test) atributuan DataAccess klasearen objektu bat gordetzen du, eta TestDataAccess klasean test-ak exekutatzeko behar diren metodo osagarriak inplementatuko dira. Adibidez, getEvents() metodoa frogatzeko lehendabizi

gertaerak DB-an sortu behar dira, baina metodo hori DataAccess klasean ez dago inplementatu oraingoz, eta metodoa ezin dugu klase horretan inplementatu, DataAccess klasea ezin baitugu aldatu. Horregatik gertaerak sortzeko (addEventAndQuestion) edo ezabatzeko (removeEvent) klase honetan inplementatu dira.

FacadeInplementation frogatzeko Arkitektura (DataAccess-en Mock erabiliz)

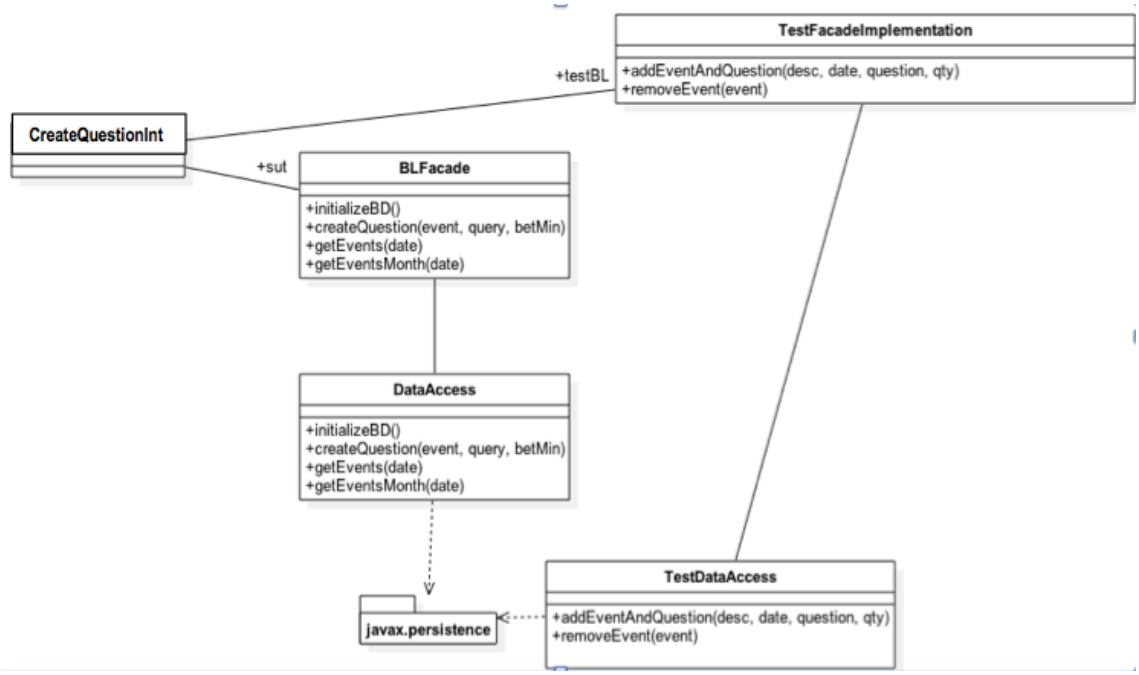
Kasu honetan, Negozio Logikako metodoak frogatuko ditugu. Horretarako metodoaren izena+MockInt (Mockito + Integrazioa) klaseak sortuko ditugu frogatu nahi ditugun metodo bakoitzeko (adibidez, CreateQuestionMockInt)



Arkitektura honetan, DataAcess-ko klase "simulatu" bat erabiliko dugu, aktore baten "doble" bat bezala. Horrerako Mockito erabiliko dugu.

FacadeInplementation frogatzeko Arkitektura (DataAccess erabiliz)

Hemen BLFacadeImplementation klaseak DataAccess klasea erabiliko du bere metodoak frogatzeko, kasu honetan, metodoak metoaren izena + Int deituko dira (adibidez, CreateQuestionInt.java)



Hemen ere, test-ak egiteko eragiketa osagarriren bat behar badugu, TestFacadeImplementation klasean inplementatu baharko dugu.

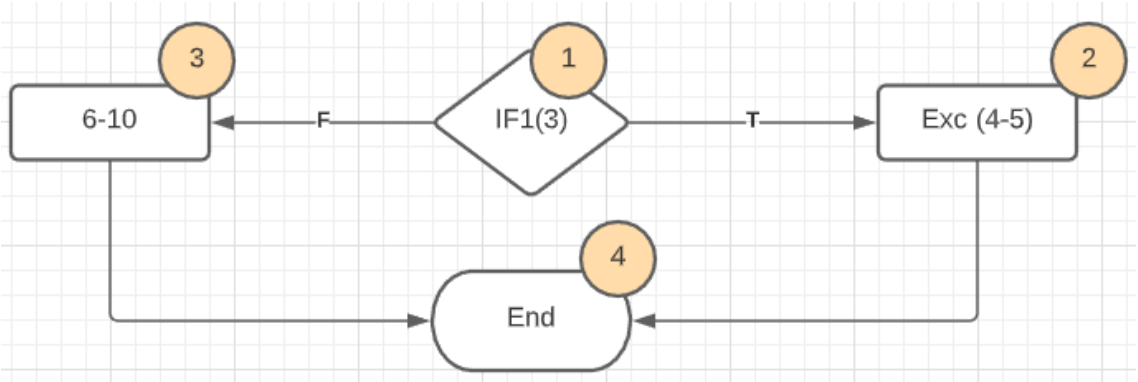
Proba proiektuen inplementazioa (arkitektura desberdinetan)

https://github.com/jononekin/Bets2021 proiektuan src/test/java karpetan 4 prototipoen adibideak aurkitu dezakezu dokumentu honen jarraian diseinatzen diren **createQuestion** metodoarentzat. Beste aldetik src/test/java/test karpetan TestFacadeImplementation.java eta TestDataAccess.java klaseak aurkituko dituzu.

createQuestion KUTXA TXURIKO PROBA ANALISIA (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       }
```

Fluxu kontrol grafoa:



V(G)=**2**

Proba taula:

#	Baldintza	Proba kontextua		Itxaron emaitzak	
		DB Egoera	Sarrera	DB Egoera	Inteera
1	IF4(T)	e∈BD, **q'∈e,** q'∈BD	e, q, 2.0	Ez da aldatzen	Exception QuestionAlreadyExist
2	IF4(F)	e∈BD, **q'∉e**	e, q, 2.0	q'∈BD **q'∈e**	q'

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

Inplementazioa: **createQuestionDAW.java**

Aurkitutako Akatsak: Proba kasu guztiak espero zuten emaitza eman dute.

createQuestion KUTXA BELTZEKO PROBA ANALISIA (DataAccess.java)

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

Sarrera-baldintza	Baliokidetasun-klase egokiak	Baliokidetasun-klase ez egokiak
Event balioa	Event!=null (1)	Event==null (7)
Question balioa	Question!=null (2)	Question==null (8)
betMinimum	betMinimum>=2 (3)	betMinimum<2 (9)
Event-a bukatu da	Event.date> now (4)	Event.date < now (10)
Event DB-an	Event ∈ BD (5)	Event ∉BD (11)
Event-ak ez dauka question-a	question ∉ event.questions (6)	question ∈ event.questions (12)

Eratorritako proba kasuak

#	Estalitako BK	Proba kontextua		Itxaron emaitzak	
		DB Egoera	Sarrera	DB Egoera	Irteera (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

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

Inplementazioa: **createQuestionDAB.java (1,2,3,7 kasuak inplementatuta)**

Aurkitutako Akatsak:

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

createQuestion KUTXA BELTZEKO PROBA ANALISIA (BLFacadeImplementation.java)

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

Proba kasuak (kutxa beltza)

Sarrera-baldintza	Baliokidetasun-klase egokiak	Baliokidetasun-klase ez egokiak
Event balioa	Event!=null (1)	Event==null (7)
Question balioa	Question!=null (2)	Question==null (8)
betMinimum	betMinimum>=2 (3)	betMinimum<2 (9)
Event-a bukatu da	Event.date> now (4)	Event.date < now (10)
Event DB-an	Event ∈ BD (5)	Event ∉BD (11)
Event-ak ez dauka question-a	question ∉ event.questions (6)	question ∈ event.questions (12)

Eratorritako proba kasuak

#	Estalitako BK	Proba kontextua		Itxaron emaitzak	
		DB Egoera	Sarrera	DB Egoera	Irteera (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	Ez da aldatzen	QuestionAlreadyExist

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

Inplementazioa: CreateQuestionMockInt.java (1,2,7 kasuak inplementatuta)

Aurkitutako Akatsak:

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