eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

PROBA PROIEKTUA

Software Ingenieritza II

Itsaso Yan Garcia eta Oihane Zaldua

Estimatutako denbora:

Itsaso Yan García Pérez: 24 ordu
Oihane Zaldua Benito: 20 ordu

GitHub esteka: https://github.com/Ittssasso

SonarCloud esteka: https://sonarcloud.io/dashboard?id=Bets21_ItsasoOihane

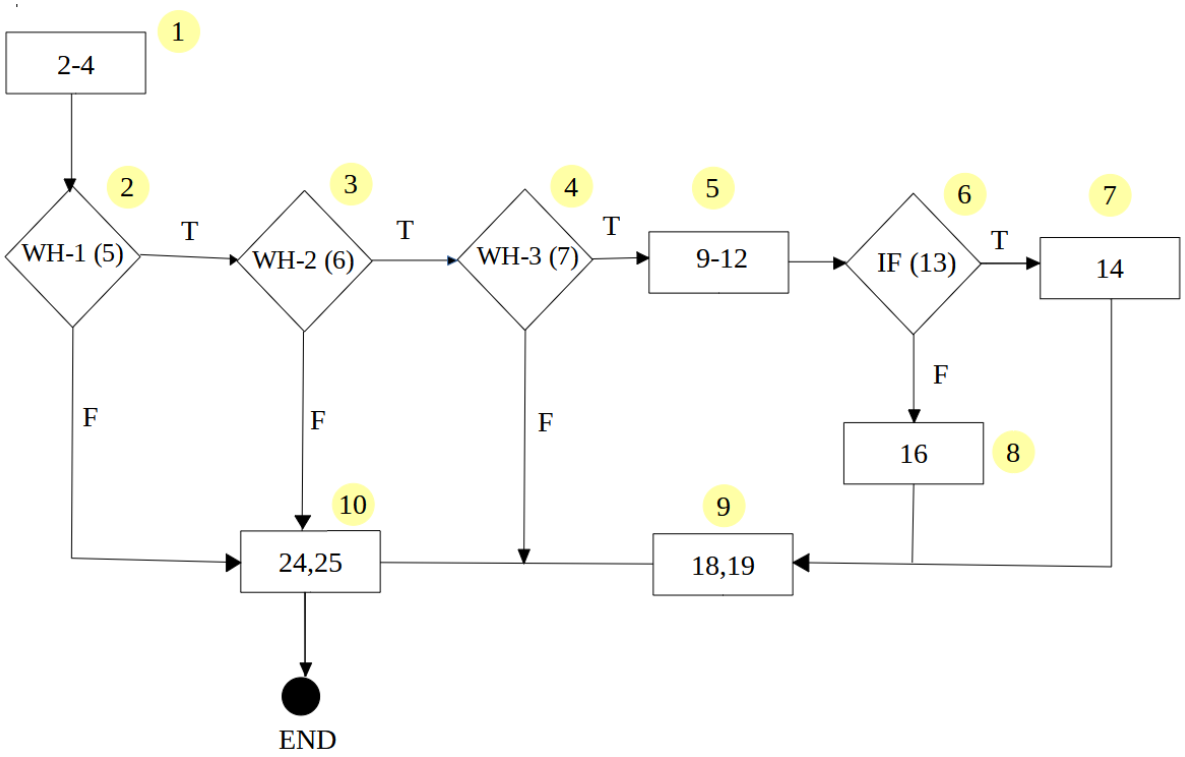
***deleteEvent metodoa** (Oihane Zaldua):

```
/**
 * This method delete a event
 * @param event to be deleted
 */
public void deleteEvent(Event event) {
    db.getTransaction().begin();
    Event ev = db.find(Event.class, event.getEventNumber());
    Vector<Question> questions = event.getQuestions();
    for (Question quest : questions) {
        for (Prediction predict : quest.getPredictions()) {
            for (Bet bet : predict.getBets()) {
                // return the money if any person has bet in any deleted question
                Bet b1 = db.find(Bet.class, bet.getBetNumber());
                RegisteredClient rc1 = db.find(RegisteredClient.class, b1.getClient().getEmail());

                float extra = bet.getMoney();
                if(bet.getPredictions().size()==1) {
                    rc1.addMovements2(ResourceBundle.getBundle("Etiquetas").getString("DeleteEvent")+": "+
                        bet.getPredictions().get(0).getQuestion().getEvent().getDescription(), +bet.getMoney(), new Date(), "-");
                }else {
                    rc1.addMovements2(ResourceBundle.getBundle("Etiquetas").getString("DeleteEvent")+": "+
                        bet.getPredictions().get(0).getQuestion().getEvent().getDescription(), +bet.getMoney(), new Date(), "x");
                }
                rc1.setBalance(rc1.getBalance() + extra);
                db.persist(rc1);
            }
        }
    }
    // delete the event
    db.remove(ev);
    db.getTransaction().commit();
}
```

PROBA BATERAGARRIAK:

Kutxa txuria: *DeleteEventDAW*

1.1 Fluxu-grafoa:



1.2 Konplexutasun ziklomatikoa:

V(G) = erabaki-nodo-kop + 1 = 4 + 1 = 5

2. Oinarrizko bide guztien definizioa:

#1) 1 2(T) 3(T) 4(T) 5 6(T) 7 9 10 (END)
#2) 1 2(T) 3(T) 4(T) 5 6(F) 8 9 10 (END)
#3) 1 2(T) 3(T) 4(F) 10 (END)
#4) 1 2(T) 3(F) 10 (END)
#5) 1 2(F) 10 (END)

3. Proba kasuen taula:

(Metodoa *void* denez ez du irteerarik, eta ez du salbuespenerik deitzen).

#kasua	Sarrerako baldintza	Datu-base egoera	Sarrerako datuak	Irteera: datu-base egoera

1	ev ∈ db
	ev.getQuestions().size()>0
	q[]=ev.getQuestions();
	qX.getPredictions().size()>0
	p[]=qX.getPredictions(); pX.getBets().size()>0
	b[]=pX.getBets();
	bX.getPredictions().size()==1
		ev=Event("Real Sociedad – Athletic", 2021/11/10)
		ev2=Event("Osasuna – Alavés", 2021/11/12)
		q=ev.addQuestion("Zenbat gol?", 5)
		p=q.addPrediction("3", 1.3)
		p1=q.addPrediction("1", 1.1)
		b=Bet(10, p1)
		b1=Bet(6, p)
		p1.addBet(b)
		p.addBet(b1)
			ev	·ev2∈db
				·Movements("Delete event: Real Sociedad - Athletic", 10, *eguneko_data*, -)∈db
				·Movements("Delete event: Real Sociedad - Athletic", 6, *eguneko_data*, -)∈db

2	ev ∈ db ev.getQuestions().size()>0 q[]=ev.getQuestions(); qX.getPredictions().size()>0 p[]=qX.getPredictions(); pX.getBets().size()>0 b[]=pX.getBets(); bX.getPredictions().size()>1	ev=Event("Real Sociedad – Athletic", 2021/11/10) ev2=Event("Osasuna – Alavés", 2021/11/12) q=ev.addQuestion("Zenbat gol?", 5) q2=ev2.addQuestion("Zenbat gol?", 3) p=q.addPrediction("3", 1.3) p1=q.addPrediction("1", 1.1) p2=q2.addPrediction("0", 1.2) v = new Vector<Prediction>(); p1, p2 ∈ v b=Bet(10, v) b1=Bet(6, p) p1.addBet(b) p2.addBet(b) p.addBet(b1)	ev	·ev2, q2, p2∈db ·Movements("Delete event: Real Sociedad - Athletic", 10, *eguneko_data*, x)∈db ·Movements("Delete event: Real Sociedad - Athletic", 6, *eguneko_data*, -)∈db
3	ev ∈ db ev.getQuestions().size()>0 q[]=ev.getQuestions(); qX.getPredictions().size()>0 p[]=qX.getPredictions(); pX.getBets().size()==0	ev=Event("Real Sociedad – Athletic", 2021/11/10) ev2=Event("Osasuna – Alavés", 2021/11/12) q=ev.addQuestion("Zenbat gol?", 5) p=q.addPrediction("3", 1.3) p1=q.addPrediction("1", 1.1)	ev	ev2∈db
4	ev ∈ db ev.getQuestions().size()>0 q[]=ev.getQuestions(); qX.getPredictions().size()==0	ev=Event("Real Sociedad – Athletic", 2021/11/10) ev2=Event("Osasuna – Alavés", 2021/11/12) q=ev.addQuestion("Zenbat gol?", 5)	ev	ev2∈db
5	ev ∈ db ev.getQuestions().size()==0	ev=Event("Real Sociedad – Athletic", 2021/11/10) ev2=Event("Osasuna – Alavés", 2021/11/12)	ev	ev2∈db

`deleteEvent` metodoaren funtzioa Event-ak ezabatzea da, Event horren galdera, predikzio eta apustuak barne. Metodo honen zuzentasuna frogatzeko datu-basea taulan adierazi bezala prestatu dugu, Event-ak soilik, baita bere galdera eta eratorri guztiak ere zuzen ezabatzen direla. `deleteEvent` exekutatu ondoren, datu-basean ezabatu nahi zen Event-a eta bere eratorriak ez daudela frogatu da, bai, ordea, ezabatu nahi ez ziren gertaerak eta, kasuren batean, sortu diren mugimendu berriak. Gainera, apustua egin duen bezeroari apustuen dirua itzuli zaiola egiaztatu da.

Testen amaieran datu-basea hasierako egoera berean utzi behar da, baina klasean hitz egin bezala, nahiz eta sortutako RegisteredClient-a ezabatzea lortu, Test2-n ev2 Event-a ezabatzerakoan *Attempt to remove a detached entity object* errorea ematen zuen eta ezin

izan da datu-basea hasierako egoeran utzi. Hala ere, *initialize* bezala zehaztuta dagoenez, hurrengo proban datu-basea berriro hutsik ezarriko da.

EclEmma edo Coverage as -> JUnit Test erabilita deleteEvent-en %100 proben bidez estalita geratu dela ziurtatu da.

Proba kasu guztiek esperotako emaitza eman dute.

Kutxa beltza: *DeleteEventDAB*

Sarrerako baldintza	BK egokia	BK ez egokia
ev null ez izatea	ev != null (1)	ev == null (2)
ev datu-basean	ev ∈ db (3)	ev ∉ db (4)

Estalitako baliokidetasun-klaseak	Datu-basearen egoera	Sarrera	Emaitza
1, 3	ev = Event("Osasuna – Alavés", 12/11/2021) ev ∈ db	ev	(ev datu-basetik ezabatu)
2	ø	null	Errorea/Salbuespena
4	ev = Event("Osasuna – Alavés", 12/11/2021) ev ∉ db	ev	Errorea/Salbuespena

(Kasu honetan ez dago muga-baliorik (ez dago balio zehatzik, tarterik, …), parametro bakarra Event motakoa baita).

Exekuzioaren ondoren ikusi dugu metodo honek ez duela akatsik. Pasa behar zaion Event motako parametro bakarra null izanda edo datu-basean ez egonda, errorea ematen du, hau da, salbuespen bat altxatzen da; Event motakoa bada eta datu-basean badago, ordea, ikus daiteke *deleteEvent* exekutatu ondoren jada Event-a ez dagoela datu-basean.

Proba kasu guztiek esperotako emaitza eman dute.

INTEGRAZIO PROBAK:

Kutxa beltza: *DeleteEventMockInt* eta *DeleteEventInt*

Sarrerako baldintza	BK egokia	BK ez egokia
ev null ez izatea	ev != null (1)	ev == null (2)
ev datu-basean	ev ∈ db (3)	ev ∉ db (4)

Estalitako baliokidetasun-klaseak	Datu-basearen egoera	Sarrera	Emaitza
1, 3	ev = Event("Osasuna – Alavés", 12/11/2021) ev ∈ db	ev	(ev datu-basetik ezabatu)
2	ø	null	Errorea/Salbuespena
4	ev = Event("Osasuna – Alavés", 12/11/2021) ev ∉ db	ev	Errorea/Salbuespena

DeleteEventINT klasean, DataAccess erabiliz, BLFacadeImplementation-eko *deleteEvent* metodoa frogatu da, bi klaseen arteko komunikazio egokia eta emaitza zuzena aztertuz. Honetarako, datu-basean dagoen Event-a, datu-basean ez dagoena eta null parametroa pasatuz froga desberdinak egin dira, esperotako salbuespen edota emaitzak lortuz (kontuan izanda BLFacadeImplementation klasean ez dela horietako aukerarik aztertzen; jasotako parametroa DataAccessera igarotzen da eta hemendik jasotzen du Negozio Logikak jaso beharreko salbuespena).

DeleteEventMockInt klasean, DataAccess klasearen “doble” bat erabiltzen da, eta DataAccess-en jokaera zuzenki simulatuz, bien arteko mezu bidalketa edo pasatako eta jasotako parametroak, exekuzio kopurua eta deitutako metodoak aztertuz.

Ez dugu akatsik aurkitu.

(Ez DataAccess-en ez BLFacadeImplementation-en, zentzu batean, ez da aztertzen Kutxa Beltzetako probetan deleteEvent metodoak eman ditzakeen salbuespenik, kontuan izanda programa berak interfazeen bidez balio-klase ez egoki horiek ekiditen dituela)

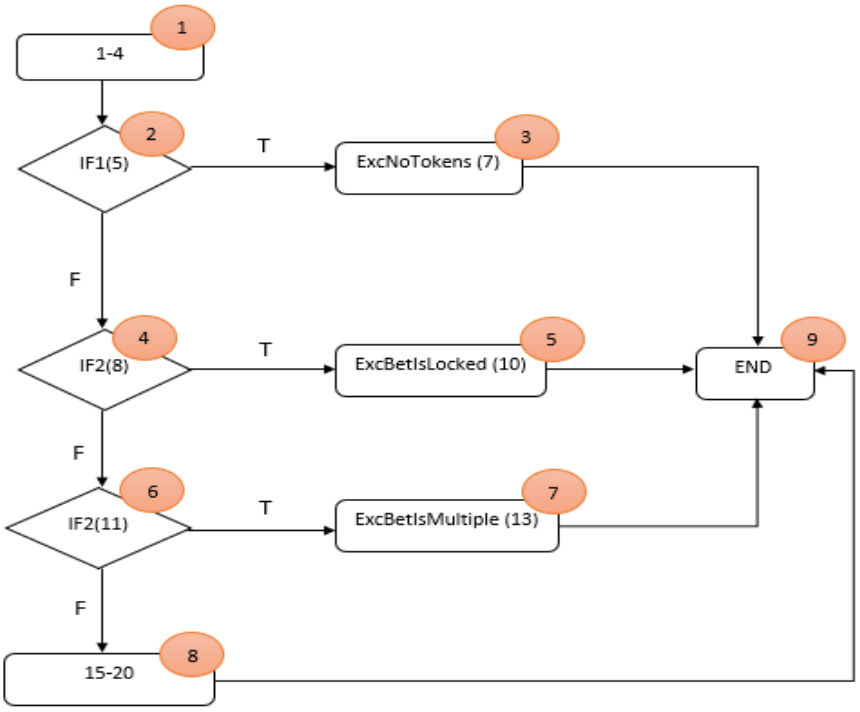
***useToken metodoa** (Itsaso García):

```
/**
 * This method looks for the tokens of the client to be used in a bet
 * @param b Bet where the token is wanted to be used
 * @param rc the client who wants to use a token
 * @throws BetIsMultiple if the bet where the token is wanted to be used is multiple
 * @throws NoTokens if the client has no tokens
 */
public int useToken(Bet b, RegisteredClient rc) throws BetIsMultiple, NoTokens, BetIsLocked {

    db.getTransaction().begin();
    Bet bet = db.find(Bet.class, b.getBetNumber());
    RegisteredClient client = db.find(RegisteredClient.class, rc.getEmail());
    int tokenAmount = client.getTokens();
    if (tokenAmount <= 0) {
        db.getTransaction().commit();
        throw new NoTokens();
    } else if (bet.getLocked()) {
        db.getTransaction().commit();
        throw new BetIsLocked();
    } else if (bet.getMultiple()) {
        db.getTransaction().commit();
        throw new BetIsMultiple();
    } else {
        bet.setFeeMultiplier();
        client.setTokens(tokenAmount - 1);
        db.persist(client);
        db.persist(bet);
        db.getTransaction().commit();
        return client.getTokens();
    }

}
```

PROBA BATERAGARRIAK:

Kutxa txuria: *UseTokenDAW*

1.1 Fluxu-grafoa:



1.2 Konplexutasun ziklomatikoa:

V(G) = erabaki nodo kop + 1 = 3 + 1 = 4

2. Oinarrizko bide guztien definizioa:

#1) 1 2(T) 3 9(END)
#2) 1 2(F) 4(T) 5 9(END)
#3) 1 2(F) 4(F) 6(T) 7 9(END)
#4) 1 2(F) 4(F) 6(F) 8 9 (END)

3. Proba kasuen taula:

(Metoda int motako balio bat itzultzen du eta hiru salbuespen ditu)

#	BALDINTZA	PROBA KONTEXTUA		ITXARON EMAITZAK	
		DB EGOERA	SARRERA	DB EGOERA	IRTEERA
1	IF1(T): tokenAmount<=0	b ∈ DB rC ∈ DB	b, rC.tokens=0,	Ez da aldatzen	Exception NoTokens
2	IF1(F) & IF2(T): tokenAmount>0 & bet.getLocked()	b ∈ DB rC ∈ DB	rC.tokens=2, b.getLocked=true.	Ez da aldatzen	Exception BetIsLocked
3	IF1(F) & IF2(F) & IF3(T): tokenAmount>0 & bet.getLocked()==false & bet.getMultiple()	b ∈ DB rC ∈ DB	rC.tokens=2, b.getLocked=false, b.getMultiple=true.	Ez da aldatzen	Exception BetIsMultiple
4	IF1(F) & IF2(F) & IF3(F): tokenAmount>0 & bet.getLocked()==false & bet.getMultiple()==false	b ∈ DB rC ∈ DB	rC.tokens=2, b.getLocked=false, b.getMultiple=false.	rC.tokens -1 ∈ DB	rC.getTokens()= rC.getTokens-1

rC= (“Alex”, “Garcia”, 18-09-21, “12345678K”, “abc@gmail.com”, “123SI2”, “A24”, true)

b= (27, p)

`UseToken` metodoaren funtzioa RegisteredClient-ari fitxa bat kentzea da erabili duelako. Hortaz, metodo honek fitxa erabili ondoren, bezero erregistratuak duen fitxa kopurua itzultzen du (fitxa bat gutxiago). `UseTokenDAW` klasean metodo honen zuzentasuna frogatzeko datu-basea taulan adierazi bezala prestatu dugu. `UseToken` exekutatu ondoren, RegisteredClient-ak fitxa bat gutxiago zuela frogatu da eta testaren amaieran datu-basea hasierako egoera itzuli dela ere bermatu da.

EclEmma edo Coverage as -> JUnit Test erabilita `UseToken` metodoa % 100 proben bidez estalita geratu dela ziurtatu da.

Proba kasu guztiek esperotako emaitza eman dute.

Kutxa beltza: *UseTokenDAB*

- *Baliokidetasun klaseen test analisia:*

SARRERAKO BALDINTZA	BK EGOKIA	BK EZ EGOKIA
Bet b eta RegisteredClient rc DBan egotea.	b ∈ DB && rc ∈ DB (1)	b ∉ DB (2) rc ∉ DB (3)
Bet b null ez izatea.	b != null (4)	b == null (5)
RegisteredClient rc null ez izatea.	rc != null (6)	rc == null (7)
Registered clientak fitxa kopurua bat baino gehiago edukitzea.	rc.tokens > 1 (8)	rc.tokens <= 0 (9)
Bet blokeatuta ez egotea.	b.isLoceked == false (10)	b.isLoced == true (11)
Bet anizkoitza ez izatea.	b.isMultiple == false (12)	b.isMultiple == true (13)

ESTALITAKO BK	DB EGOERA	SARRERA	EMAITZA
1,4,6,8,10,12	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 5) ∈ DB	b, rc *rc.tokens = 5	rc.tokens = 4
2	b= Bet(100, p) ∉ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 2) ∈ DB	b, rc	Errorea/Salbuespena
3	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 2) ∉ DB	b, rc	Errorea/Salbuespena
5	rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 2) ∈ DB	b == null rc	Errorea/Salbuespena
7	b= Bet(100, p) ∈ DB	b rc == null	Errorea/Salbuespena
9	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 0) ∈ DB	b, rc *rc.tokens = 0	NoTokensException
11	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 5) ∈ DB	b, rc *b.isLocked == true	BetIsLockedException
13	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 5) ∈ DB	b, rc *b.isMultiple == true	BetIsMultipleException

Vector<Prediction> p = new Vector<Prediction>()

`UseTokenDAB` klasean, `DataAccess` motako *sut* izeneko objektu bat eta `TestDataAcess` motako *testDA* izeneko objektu bat erabili dira `UseToken` metodoaren proba kasuak egiteko. Metodo honen exkuzioaren ondoren ikusi dugu metodo honek ez dituela akatsik. Pasa behar zaio parametroren bat null bada edo datu-basean ez badago, errorea ematen du, hau da, salbuespen bat altxatzen da. Aldiz, Bet eta RegisteredClient motako parametroak pasatzen bazaizkio eta datu-basean badaude, `UseToken` ongi exekutatuko da (barneko baldintzak ongi betez), pasa zaion bezero erregistratuari fitxa bat kenduz. Gainera, datu-basea hasierako egoerara itzuliko da.

Proba kasu guztiek esperotako emaitza eman dute.

- *Muga balioen ikerketa "Tokens" atributuarentzat.*

#	ESTALITAKO BK	DB EGOERA	SARRERA	EMAITZA
1	9	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, -1) ∈ DB	b, rc *rc.tokens = -1	NoTokensException
2	9	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 0) ∈ DB	b, rc *rc.tokens = 0	NoTokensException
3	1,4,6,8,10,12	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 1) ∈ DB	b, rc *rc.tokens = 1	rc.tokens = 0

Vector<Prediction> p = new Vector<Prediction>()

`UseToken` metodoaren muga balioa RegisteredClient-en fitxa kupurua da. Izan ere, fitxen kopuruaren arabera, metodoa ongi funtzionatzen du edota errorea ematen du. Hortaz, proba kasuen eraginkortasuna hobetzeko, baliokidetasun klaseen muga balioen analisia egin behar izan da. Kasu honetan bezero erregistratuak -1 fitxa (zorretan), 0 fitxa (ez du fitxarik) edo 1 fitxa (fitxa bat dauka) balioekin egin dira probak.

INTEGRAZIO PROBAK:

Kutxa beltza: *UseTokenMockInt* eta *UseTokenInt*

SARRERAKO BALDINTZA	BK EGOKIA	BK EZ EGOKIA
Bet b eta RegisteredClient rc DBan egotea.	b ∈ DB && rc ∈ DB (1)	b ∉ DB (2) rc ∉ DB (3)
Bet b null ez izatea.	b != null (4)	b == null (5)
RegisteredClient rc null ez izatea.	rc != null (6)	rc == null (7)
Registered clientak fitxa kopurua bat baino gehiago edukitzea.	rc.tokens > 1 (8)	rc.tokens <= 0 (9)
Bet blokeatuta ez egotea.	b.isLoceked == false (10)	b.isLoced == true (11)
Bet anizkoitza ez izatea.	b.isMultiple == false (12)	b.isMultiple == true (13)

ESTALITAKO BK	DB EGOERA	SARRERA	EMAITZA
1,4,6,8,10,12	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 5) ∈ DB	b, rc *rc.tokens = 5	rc.tokens = 4
2	b= Bet(100, p) ∉ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 2) ∈ DB	b, rc	Errorea/Salbuespena
3	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 2) ∉ DB	b, rc	Errorea/Salbuespena
5	rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 2) ∈ DB	b == null rc	Errorea/Salbuespena
7	b= Bet(100, p) ∈ DB	b rc == null	Errorea/Salbuespena
9	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 0) ∈ DB	b, rc *rc.tokens = 0	NoTokensException
11	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 5) ∈ DB	b, rc *b.isLocked == true	BetIsLockedException
13	b= Bet(100, p) ∈ DB rc = RegisteredClient("Ane", "Garcia", 2001-11-27, "12345678V", "client@email.com", "12345678", "1234987612346789", true, 5) ∈ DB	b, rc *b.isMultiple == true	BetIsMultipleException

Vector<Prediction> p = new Vector<Prediction>()

`UseTokenMockInt` klasean, `DataAccess` klasearen *dataAccess* izeneko “doble” bat eta `BLFacadeImplementation` motako *sut* izeneko objektu bat erabiltzen dira. `UseToken` metodoan bi klase hauen arteko komunikazioa egokia eta emaitza zuzena dela frogatu da. Hortaz, aurreko taulan agertzen diren egoerak aztertu eta proba kasu desberdinak egin dira, espero ziren salbuespenak eta emaitzak lortuz. Proba kasu guztietan prozedura bera erabili da: parametroak prestatu, *MOCK* objektuaren emaitzak konfiguratu, *SUT*-eri deitu eta *MOCK* deiak egin direla ziurtatu (salbuespen bat altxatu behar ez denean).

`UseTokenInt` klasean, `UseTokenDAB` (eta `UseTokenMockInt`) klasean egin diren proba kasu berdinak egin dira, espero ziren salbuespenak eta emaitzak lortuz. Kasu honetan, `TestDataAcces` motako objektu bat erabili beharrean, *testBL* izeneko `TestFacadeImplementation` motako objektu bat erabili da. Baita *sut* izeneko `BLFacadeImplementation` motako objektu bat ere.

Bi klase hauetan *sut* objektua mota berekoa da eta bi klaseetan sut-ek DataAcces-eri deitzen dio, kasu batean `DataAccess` datu baseari eta bestean “double”-a den datu-baseari.

Ez dugu akatsik aurkitu.