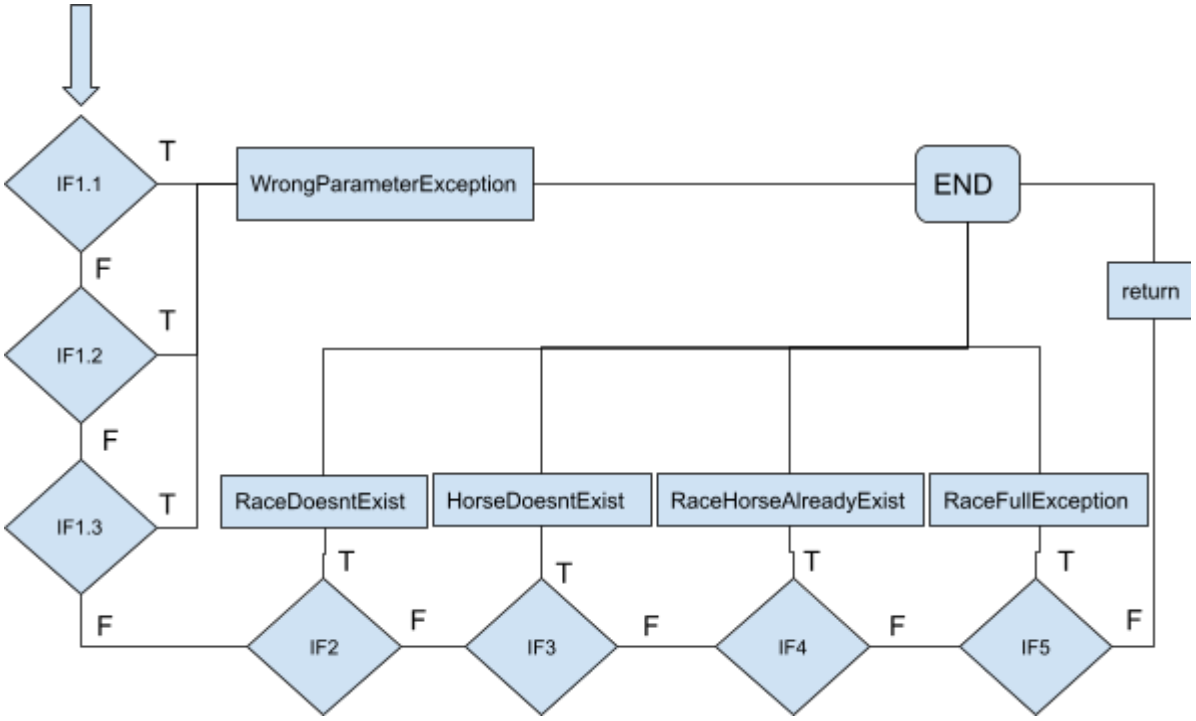
CREATE RACE HORSE (Julen Larrañaga ~18ordu)

```
public RaceHorse createRaceHorse(double winGain, Race race, Horse horse)
        throws RaceHorseAlreadyExist,WrongParameterException, RaceFullException, RaceDoesntExist, HorseDoesntExist{

    if(race==null || horse==null || winGain<1) throw new WrongParameterException();

    Race newRace = db.find(race.getClass(), race.getKey());
    if(newRace==null) throw new RaceDoesntExist();

    Horse newHorse = db.find(horse.getClass(), horse.getKey());
    if(newHorse==null) throw new HorseDoesntExist();

    RaceHorse raceHorse = new RaceHorse(winGain, newRace, newHorse);
    if(newRace.getRaceHorses().contains(raceHorse)) throw new RaceHorseAlreadyExist();

    db.getTransaction().begin();
    if(!newRace.addRaceHorse(raceHorse)) throw new RaceFullException();
    db.getTransaction().commit();
    System.out.println("DB>>> New RaceHorse created and added to data base");
    return raceHorse;
}
```

KUTXA TXURIKO PROBAK (DataAccess.java)

Lehenengo, fluxu kontrol grafoa irudikatu dugu, metodoaren konplexutasun ziklomatikoa kalkulatu ahal izateko:



Konplexutasuna kalkulatzeko, grafoko baldintza kopurua + 1 egingo dugu:
V(G) = 7 + 1 = **8**

Konplexutasuna 8 denez, programak segi ditzakeen 8 bide ezberdinak adieraziko ditugu:

1) 1.1 - WrongParameterException - END
2) 1.1 - 1.2 - WrongParameterException - END
3) 1.1 - 1.2 - 1.3 - WrongParameterException - END
4) 1.1 - 1.2 - 1.3 - 2 - RaceDoesntExist - END
5) 1.1 - 1.2 - 1.3 - 2 - 3 - HorseDoesntExist - END
6) 1.1 - 1.2 - 1.3 - 2 - 3 - 4 - RaceHorseAlreadyExist - END
7) 1.1 - 1.2 - 1.3 - 2 - 3 - 4 - 5 - RaceFullException - END
8) 1.1 - 1.2 - 1.3 - 2 - 3 - 4 - 5 - return - END

Ondoren, bide horietako bakoitzeko proba kasu bat adieraziko dugu:

#	DB egoera	Baldintza	Sarrera	Emaitza
1	-	race == null	1.5, null, h	WrongParameterException
2	-	horse==null	1.5, r, null	WrongParameterException
3	-	winGain<1	0.5, r, h	WrongParameterException
4	r∉DB	newRace==null	1.5,r,h	RaceDoesntExist
5	h∉DB r∈DB	newHorse==null	1.5,r,h	HorseDoesntExist
6	r, h∈DB	newRace.getRaceHorses().contains(rh)	1.5,r,h	RaceHorseAlreadyExist
7	r, h∈DB	!newRace.addRaceHorse(rh);	1.5,r,h	RaceFullException
8	r, h∈DB	newRace.addRaceHorse(rh);	1.5,r,h	return

Proba kasuen inplementazioa: **createRaceHorseDAW.java**

Aurkitutako akatsak: **Esperotako emaitzak lortu dira proba kasu guztietan.**

KUTXA BELTZEKO PROBAK (DataAccess.java)

```
public RaceHorse createRaceHorse(double winGain, Race race, Horse horse)
        throws RaceHorseAlreadyExist,WrongParameterException, RaceFullException, RaceDoesntExist
```

Lehenengo, baliokidetasun egoki eta ez egokiak definituko ditugu:

	Sarrera-baldintza	BK Egokia	BK Ez Egokia
Sarrera Parametroak	Race r	r != null **(1)**	r == null **(9)**
	Horse h	h != null **(2)**	h == null **(10)**
	double winGain	winGain>=1 **(3)**	winGain<1 **(11)**
	Race r	r∈DB **(4)**	r∉DB **(12)**
	Horse h	h∈DB **(5)**	h∉DB **(13)**
Programaren Semantika	Race r	r.getFinished()==false **(6)**	r.getFinished()==true **(14)**
	RaceHorse rh	rh∉DB **(7)**	rh∈DB **(15)**
	Race r	r.size()<r.numOfStreets() **(8)**	r.size()==r.numOfStreets() **(16)**

Eratorritako proba kasuak:

Estalitako BK	DB Egoera	Sarrera	Emaitza
1,2,3,4,5,6,7,8	r,h∈DB eta rh∉DB	(1.5, r, h)	return RaceHorse
9	-	(1.5, null, h)	WrongParameterException
10	-	(1.5, r, null)	WrongParameterException
11	-	(0.5, r, h)	WrongParameterException
12	r∉DB	(1.5, r, h)	RaceDoesntExist
13	h∉DB	(1.5, r, h)	HorseDoesntExist
14	r,h∈DB	(1.5, r, h)	RaceFinished
15	r,h,rh∈DB	(1.5, r, h)	RaceHorseAlreadyExist
16	r,h∈DB eta rh∉DB	(1.5, r, h)	RaceFullException

Proba kasuen inplementazioa: **createRaceHorseDAB.java**
Aurkitutako akatsak: **Lasterketa amaitu den ez dut egiaztatzen datu baseko funtzioan, beraz erroreak suerta daitezke.**

KUTXA BELTZEKO PROBAK (BLFacadeImplementation.java)

```
public RaceHorse createRaceHorse(double winGain, Race race, Horse horse)
        throws RaceHorseAlreadyExist,WrongParameterException, RaceFullException, RaceDoesntExist
```

Lehenengo, baliokidetasun egoki eta ez egokiak definituko ditugu:

	Sarrera-baldintza	BK Egokia	BK Ez Egokia
Sarrera Parametroak	Race r	r != null **(1)**	r == null **(9)**
	Horse h	h != null **(2)**	h == null **(10)**
	double winGain	winGain>=1 **(3)**	winGain<1 **(11)**
	Race r	r∈DB **(4)**	r∉DB **(12)**
	Horse h	h∈DB **(5)**	h∉DB **(13)**
Programaren Semantika	Race r	r.getFinished()==false **(6)**	r.getFinished()==true **(14)**
	RaceHorse rh	rh∉DB **(7)**	rh∈DB **(15)**
	Race r	r.size()<r.numOfStreets() **(8)**	r.size()==r.numOfStreets() **(16)**

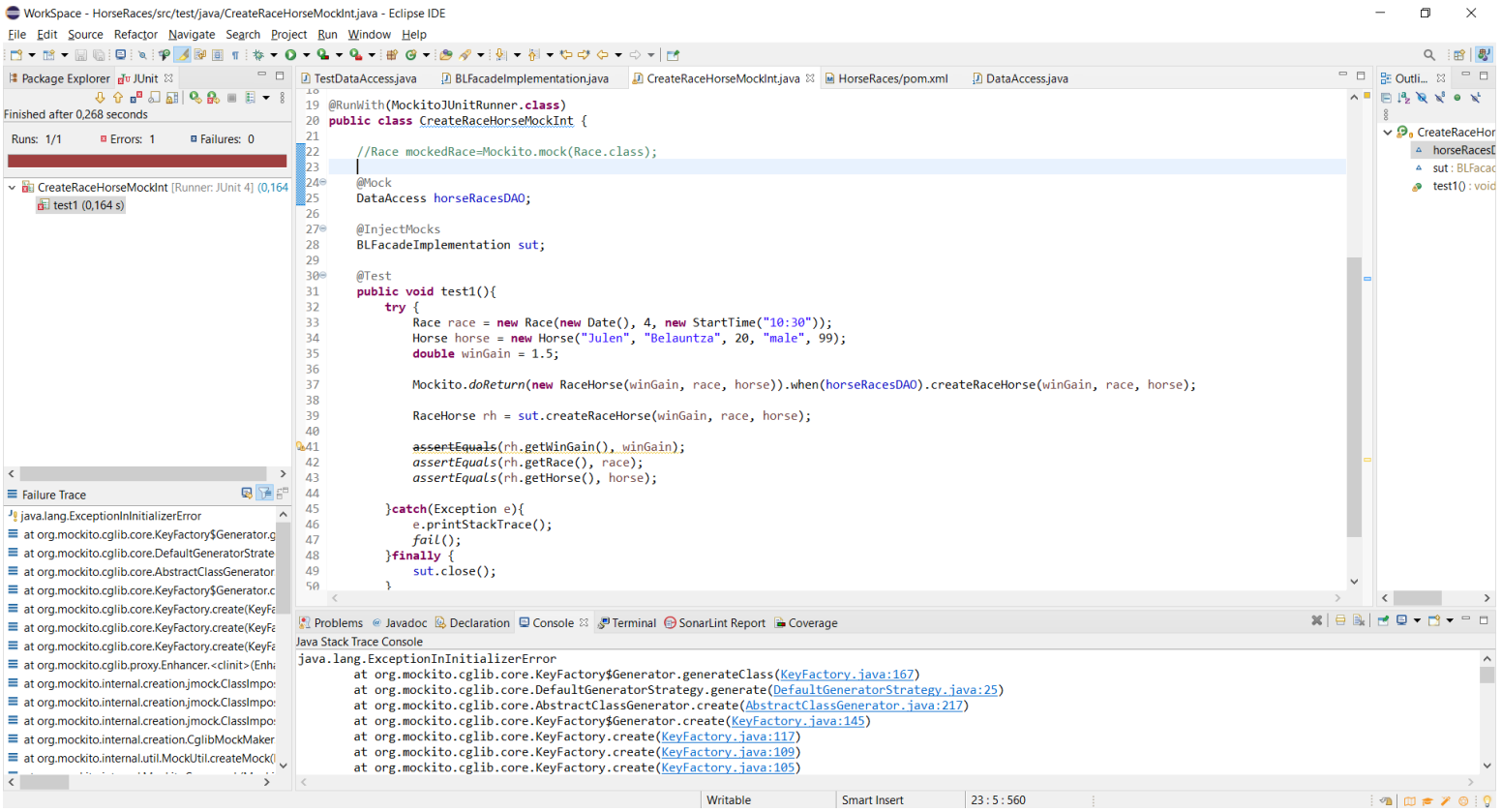
Eratorritako proba kasuak:

Estalitako BK	DB Egoera	Sarrera	Emaitza
1,2,3,4,5,6,7,8	r,h∈DB eta rh∉DB	(1.5, r, h)	return RaceHorse
9	-	(1.5, null, h)	WrongParameterException
10	-	(1.5, r, null)	WrongParameterException
11	-	(0.5, r, h)	WrongParameterException
12	r∉DB	(1.5, r, h)	RaceDoesntExist
13	h∉DB	(1.5, r, h)	HorseDoesntExist

14	r,h∈DB	(1.5, r, h)	RaceFinished
15	r,h,rh∈DB	(1.5, r, h)	RaceHorseAlreadyExist
16	r,h∈DB eta rh∉DB	(1.5, r, h)	RaceFullException

Proba kasuen inplementazioa: **createRaceHorseMockInt.java**
Aurkitutako akatsak:

Ezin izan ditut proba kasuak inplementatu, Mockito-ko @Mocks eta @InjectMocks etiketek honako errorea ematen zidatelako:



[GITHUB]

[SONARCLOUD]