PROBA PROIEKTUA - SONAR

Itsaso Yan García Pérez eta Oihane Zaldua Benito

Hasierako proiektuaren helbidea: https://gitlab.com/powerinstashoot/3iterazioa

SonarCloud-en proiektuaren helbidea:
https://sonarcloud.io/dashboard?id=SonarItsasoOihane

BUGS: (Bug guztiak eta Vulnerability bakarra aztertu ditugu eta zuzendu ditugu, klasean hitz egin bezala, zailegiak zirenak izan ezik)

1. *DataAccess.java - A "NullPointerException" could be thrown; "replicatedClient" is nullable here.* (652. eta 666. lerroetan)

`DataAccess`-en aurkitzen den `putResult` metodoan `RegisteredClient` motako *replicatedClient* sortzen da, erreplikatuak izan diren erabiltzaileen apustuetatik lor daitekena. Noski, 631 lerroko for-ean, une horretan aztertzen ari den apustua erreplikatua ez bada, *replicatedClient*-en balioa *null* izango da. Ondorioz, *replicatedClient* erabili aurretik, baldintza batekin ezberdin null izan behar dela ez bada zehazten, `NullPointerException` gertatzeko arriskua dago. Hala ere, Bug-a ematen zuten lerroetara iristeko jada zeuden *if*-ak (651. eta 665. lerroetan), kode horiek soilik apustua erreplikatua bazen exekutatzeko. If-a adierazteko modua SonarLint-ak ez zuen baldintza apropos bezala ikusten, eta horregatik baldintza aldatu dut *replicatedClient!=null* ezarriz, hau da, soilik apustua erreplikatutakoa bada eta, ondorioz, erreplikatua izan den bezero bat badauka, exekutatuko dira 652. eta 666. lerroak.

Aldaketa hauek egin eta gero, *percentage* aldagaia ez zen erabiltzen, soilik bi baldintza horietarako (651. eta 665. lerroetan) sortua zegoelako eta, ondorioz, aldagaia ezabatu dut.

2. *CreatePredictionGUI.java - Use the "equals" method if value comparison was intended.* (348. eta 352. lerroak)

Ohartarazpen hau ematen da bi Integer konparatzerakoan == ikurrak erabiltzeagatik, `equals` erabili ordez, == ikurrek memorian duten kokapena konparatzen baitu, eta `equals`-ek balioa. Bug hau zuzentzeko konparaketan `equals` jartzearekin nahikoa litzateke, nahiz eta kasu honetan memorian kokapen berdina izan behar duten konparaketako bi aldagaiek.

3. *PutResultGUI.java - Use the "equals" method if value comparison was intended.* (347. lerroa)

`CreatePredictionGUI`-n azaldutako arazo bera topatzen dugu hemen, konponbide berdinarekin.

4. *RegisterGUI.java - Refactor this repetition that can lead to a stack overflow for large inputs.* (641. lerrotik aurrera)

Espresio erregularrak patroi originala baino askoz handiagoa den patroi bat deskriba dezake, eta honek stack-ak edo pilak gainezka egitea eta aplikazioa blokeatzea eragin dezake. Hala gertatzen da gure programako espresio erregularrarekin. Hala ere, klasean hitz egin bezala, ez dakigu modurik hau sinplifikatzeko eta horrela geratuko zela erabaki genuen.

5. *RegisterGUI.java - The return value of "toUpperCase" must be used.* (500. lerroa)

Ohartarazpen honek ikustarazten gaitu `toUpperCase` metodora egindako deiak ez duela efekturik, hau da, metodo horren aplikazioak ez du eraginik gure erabileran. Kasu honetan DNI kodea osatzen duen String-a maiuskulaz jartzen ari gara, jarraian soilik zenbakiak diren karaktereak erabiltzeko. Ondorioz, `toUpperCase` metodora deia ken daiteke.

6. *BLFacadeImplementation.java - Use an “instanceof” comparison instead.* (271. eta 301. lerroak).

Ohartarazpen honek esaten du hobeagoa dela “instanceof” erabiltzea konparaketa gisan. *BLFacadeImplementation* klasean *ToBet()* metodoan ezinbestekoa da baieztatzea erabiltzailea erabiltzaile erregistratu bat dela apustua egiteko eta *deleteBet()* metodoan apustua ezabatzeko. Hortaz, getClass().getSimpleName() eta equals() funtzioak erabili genituen zihurtatzeko erabiltzailea erabiltzaile erregistratu dela. Baina hori arriskutsua izan daiteke; hortaz, u.getClass().getSimpleName().equals(“RegisteredClient”) jarri ordez hobeagoa da (u instanceof domain.Register erabiltzea.

7. *LoginGUI.java - Use an “instanceof” comparison instead.* (78., 82. eta 87. lerroak).

Ohartarazpen hau aurrekoaren berdina da, baina kasu honetan logeatzen den pertsona zer motakoa den jakin nahi da. Hau da, admin, worker edo registeredClient.Hortaz, honako hau jarri beharko dugu:

- Logged_person instanceof domain.Worker
- Logged_person instanceof domain.RegisteredClient
- Logged_person instanceof domain.Admin.

8. DataAccess.java - *Cast one of the operands of this multiplication operation to a "float".* (887. lerroa)

Ohartarazpen honek esaten digu biderketaren eragingai bat float motan bihurtu behar dela kalkulu errorerik ez sortzeko Hortaz, -quantityTokens*5 jarri ordez, -quantityTokens*(float)5 jarri dugu arazoa konpontzeko.

9. *ToBetGUI.java - Remove the unboxing from “float”.* (362. lerroa)

Ohartarazpen honek esaten digu float unboxing prozesua ezabatzeko. Unboxing prozesua objektuatik balio primitiboa lortzean datza. Hortaz, interesatzen zaigunez Float objektu bat sortzea, ez du zentzurik *float.value()* metodoa erabiltzea *Float.valueOf()* metodoaren barruan. Izan ere, *float.value()* metodoak float balio primitiboa itzultzen du eta hori guri ez zaigu interetzen. Beraz, hori ezabatu behar dugu eta horrela arazoa konponduko da.

CODE SMELL: (Blocker bakarra zegoen, eta Vulnerability bat zen, beraz aurreko atalean zuzendu behar zen, baina klasean adostu bezala, zailegia zen eta ez dugu egin. Info bakarra zegoen, “TODO” bat ezabatu ez izanagatik soilik zegoena. Beraz, ezin izan dugu zorroztasun bakoitzatik bat egin; zuzenketa guztiak Critical, Major eta Minor dira)

1. Minor: *DataAccess.java - Immediately return this expression instead of assigning it to the temporary variable "rc".* (896. lerroa)

Zerrenda bat hutsik dagoela probatzeko size()==0 erabiltzen zen, baina isEmpty() erabiltzeak kodea erraztu dezake. Gainera, size() metodoaren konplexutasuna O(n) izan daiteke, baina isEmpty()-rena O(1) da beti, beraz aukera hobea litzateke isEmpty() erabiltzea zerrenda hutsik dagoela frogatzeko.

2. Minor: *DataAccess.java - Use isEmpty() to check whether the collection is empty or not.* (497. lerroa)

Aldagai bat definitzea eta jarraian itzultzea jardute okerra dela dio ohartarazpenak. Beraz, *db.find(RegisteredClient.class, email)* zuzenean itzuliz konpondu dugu, aldagairik definitu gabe.

3. Minor: *DataAccess.java - Replace this if-then-else statement by a single return statement.* (531.lerroa).

Ohartarazpen honek esaten du If-then-else egitura return bakar batean sinplifikatu behar dela itzultzen duena boolean motakoa delako; hau da, false edo

true itzuliko du. Hortaz, soilik bi erantzun izan ditzakeenez, zuzenean kasu honetan *return rc==0* jar daiteke.

4. Minor: *ViewBetsGUI.java - Declare "selectedYear2" on a separate line.* (52.lerroa).

Ohartarazpen honek gomendatzen du lerro bakar batean hainbat atributu ez adierazteko, irakurtzeko zaila izan daitekelako. Hortaz, Integer motako 4 atributu lerro batean adierazi ordez, lau lerro desberdinetan idatzi ditugu.

5. Major: *DataAccess.java - This block of commented-out lines of code should be removed.* (21 lerro desberdinetan)

Arazoa kodea duten lerro asko komentaturik daudela da. Lerro horiek programatzerakoan lagungarri baziren ere, orain ez dute programan eraginik eta soilik programa puztu eta irakurketa zailtzen dute. Ondorioz, ezabatu ditugu.

6. Major: *Person.java - Change the visibility of this constructor to "protected".* (28 eta 32.lerroa).

Klase abstraktuetan eraikitzailea(k) ez dira publikoak izan behar. Izan ere, abstraktuak diren klaseen eraikitzaileak soilik haien umeen eraikitzaileetan deitu daitezke. Hortaz, ez du zentzurik publikoak izatea, protected baizik.

7. Major: *ToBetGUI.java - Remove this unused “rG" private field.* (99.lerroa)

ToBetGUI klasean “rG” izeneko atributu pribatu bat sortu genuen, baina ondoren ez genuen klase horretan erabili. Hortaz, ohartarazpenak esaten digu atributu hori ezabatzeko atributu “hila” baita. Ezabatzerakoan mantentzeko gaitasuna hobetuko da garatzaileek ez dutelako pentsatu beharko zertarako den atributu hori.

8. Major: *ObjectdbManagerServer.java - Rename "c" which hides the field declared at line 29.*

29. lerroan ConfigXML motako “c” izeneko atributu bat definitzen dugu, baina ondoren, 103. lerroan izen bera duen ConfigXML objektua sortzen da. Hortaz, ohartarazpenak esaten digu izena aldatzeko. Beraz, arazoa konpotzeko 103. lerroan sortutako objektuari “c” deitu ordez “cc” deitu diogu.

9. Critical: *DataAccess.java - Use indentation to denote the code conditionally executed by this "for".* (781.lerroa)

Begizta edo baldintza batek lerro bakarra duenenan giltzen erabilera aukerazkoa da. 780. lerroko *for*-erako, ez zen giltzik erabiltzen, baina begizta barruko lerroa ez zuen koska (indentation) edo ez zegoen tabulatuta. Ondorioz, programatzailearen intentzioa ez da garbia eta mantenuarentzako nahasgarria izan liteke. Konponbidea 781. lerroa barrurago sartzea edo tabulatzea izan da.

10. Critical: *EditBetGUI.java - Define a constant instead of duplicating this literal "Etiquetas" 22 times. [+22 locations]* (4 string desberdinekin)

Errepikatutako Stringek errefaktorizazioa (refactoring, edo kode garbiketa) zailtzen dute, string berdinaren agerpen guztiak eguneratu behar direlako. Baina String hori konstante batean gordetzen badugu, konstantea string-a zegoen leku guztietan erabili ahal izango dugu, eta eguneratu beharreko kasuetan, leku bakar batean egin ahal izango dugu.

Horregatik, “Etiquetas” String-a, 22 alditan erabiltzen dena, “ETIQUETAS” konstantearekin ordezkatu dugu (*private static final String ETIQUETAS= "etiquetas"*), eta berdin jokatu dugu beste string errepikatuekin.

11. Critical: *RegisterGUI.java - Add a default case to this switch* (193, 319 eta 361.lerroan)

Ohartarazpen honek switch sententzian *default* kasua gehitzeko esaten du switchean dagoen kasurik betetzen ez bada hori exeketuta dadin. Berez, ez da derrigorrezkoa jartzea, switchean dagoen kasurik betetzen ez bada, switch sententziatik kanpo dagoen kodea exekutatzen jarraituko lukeelako inongo arazorik gabe, baina hala ere, jarri dugu inongo arazorik sorta ez dadin.