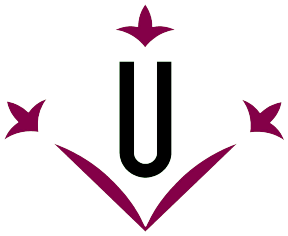

Universitat de Lleida
Escola Politècnica Superior

Enginyeria del programari

Pràctica 4: Disseny i proves unitàries

Pere Rollón Baiges
Jordi Rafael Lazo Florensa
Sergi Sabaté

17 de gener de 2021

Índex

1 Paquet *data*

Aquest paquet esta composat per les classes *DitalSignature*,*HealthCardID*,*ProductID* amb els seus tests corresponents. Tot seguit es procedirà a explicar com s'han decidit desenvolupar.

1.1 Classe *DitalSignature*

Aquesta classe s'encarrega de representa la signatura digital (en un array de bytes) de cada metge que anirà associaciada a una prescripció mèdica.

Al constructor se li passa per paràmetre un *String* i posteriorment amb el mètode *checkDigitalSignature* comprovem que se li ha passat correctament un *String* vàlid (que no és *null* o conté un array buit), en cas contrari és llançarà la seva excepció corresponent. Si és correcte, aquesta signatura del metge (*String*) és transformarà en una array de bytes i s'emmagatzemarà. Aquest mètode s'ha implementat a l'hora d'inicialitzar el constructor de la classe de manera que així s'asseguri la correcta creació d'una instància de *DigitalSignature*.

Finalment s'implementen els mètodes: *toString*, *equals* i *hashCode*.

1.1.1 Test *DigitalSignature*

Pel que fa al test d'aquesta classe, s'ha optat per seguir el mateix el disseny per a tots els tests.

Primer s'inicialitzen les instàncies que s'utilitzaran. A continuació es genera un *@BeforeEach* amb el mètode *setUp* el qual crea els objectes necessaris que s'utilitzaran en els posterior test. Un cop feta aquesta configuració inicial es procedeix a generar els tests necessaris en funció de classe implementada.

Per a la classe *DigitalSignature*, s'han implementat 3 test diferents:

1. Test que comprova una *DigitalSignature* és igual a una altra.
2. Test que comprova que una *DigitalSignature* sigui *null* o contingui un array buit i es llanci una excepció.
3. Test que comprova que els bytes d'una signatura del metge siguin correctes i coincideixen amb una signatura.

1.2 Classe *HealthCardID*

Aquesta classe s'encarrega de representar el codi d'identificació assignat a cada pacient.

Al constructor se li passa per paràmetre un *String* i posteriorment amb el mètode *checkPersonalID* comprovem que se li ha passat correctament un *String* vàlid, és a dir, que el seu format és correcte, en cas contrari es llançarà la seva excepció corresponent. Per comprovar que el format és vàlid s'ha creat un mètode *boolean* anomenat *isValidFormat* el qual retorna *true* si el *String* és correcte i *false* en cas contrari. Per comprovar-ho s'ha restringit el format de la *HealthCardID*, ja que consta de 8 lletres B, 2 lletres qualsevol, 6 números qualsevol i 12 números qualsevol. Si tot és correcte, es crearà una instància de *HealthCardID* correctament. Aquest mètode s'ha implementat a l'hora d'inicialitzar el constructor de la classe de manera que així s'asseguri la correcta creació d'una instància de *HealthCardID*.

Finalment s'implementen els mètodes:*toString*, *equals* i *hashCode*.

1.2.1 Test *HealthCardID*

Per a la classe *HealthCardID*, s'han implementat 3 test diferents:

1. Test que comprova s'ha obtingut correctament un *HealthCardID*.
2. Test que comprova que el format del *HealthCardID* és incorrecte i espera que es llanci una excepció (s'ha contemplat tots els errors possibles).
3. Test que comprova que un *HealthCardID* sigui *null* i es llanci una excepció.

1.3 Classe *ProductID*

Aquesta classe s'encarrega de representar el codi d'identificació d'un producte.

En aquesta classe, al constructor se li passa per paràmetre un *String* i posteriorment amb el mètode *checkUPCcode* comprovem que se li ha passat correctament un *String* vàlid (que no és *null*, no conté un array buit i tampoc un array amb un espai en blanc), en cas contrari es llançarà la seva excepció corresponent. Si tot és correcte, es crearà una instància de *ProductID* correctament. Aquest mètode s'ha implementat a l'hora d'inicialitzar el constructor de la classe de manera que així s'asseguri la correcta creació d'una instància de *ProductID*.

Finalment s'implementen els mètodes: *toString*, *equals* i *hashCode*.

1.3.1 Test *ProductID*

Per a la classe *ProductID*, s'han implementat 3 test diferents:

1. Test que comprova que el format del *ProductID* és incorrecte i espera que es llanci una excepció.
2. Test que comprova que dos *ProductID* siguin iguals.
3. Test que comprova que dos *ProductID* no siguin iguals.

2 Paquet *exceptions*

Aquest paquet està compost per un total de 14 excepcions les quals ja estaven proporcionades en l'enunciat de la pràctica. Així doncs, no s'ha decidit incloure'n més, ja que no s'ha trobat cap necessitat.

3 Paquet *medicalconsultation*

Aquest paquet esta compost per les classes *MedicalPrescription*, *MedicalPrescriptionLine*, *Posology*, *ProductSpecification*, *TakingGuidelines* amb els seus tests corresponents. Tot seguit es procedirà a explicar com s'han decidit desenvolupar.

3.1 Classe *MedicalPrescription*

Aquesta classe s'encarrega d'afegir, modificar o eliminar línies de la prescripció mèdica, per ferho s'utilitzen els mètodes de la classe *MedicalPrescriptionLine*. A més a més, s'ha implementat un mètode anomenat *previewMedicalPrescriptionLines* que permet retornar una llista formada per les línies de prescripció.

Primer s'ha generat els *setters* i *getters* dels atributs de la classe. A continuació, en el constructor s'ha creat una nova instància de la classe *MedicalPrescriptionLine* que és una estructura de dades *HashMap <K,V>*. Un cop creada aquesta instància, les funcions: *addLine*, *modifyLine* i *removeLine* són crides a les funcions amb el mateix nom de la classe *MedicalPrescriptionLine*. També s'ha implementat un nou mètode anomenat *previewMedicalPrescriptionLines* que permet retornar una llista formada per les línies de prescripció, per fer-ho es crida al mètode *toString* de la classe *MedicalPrescriptionLine*.

Finalment s'ha implementat el mètode: *toString*.

3.1.1 Test *MedicalPrescription*

Pel que fa als test, les funcions: *addLine*, *modifyLine* i *removeLine* són crides a les funcions amb el mateix nom de la classe *MedicalPrescriptionLine* mentre que el mètode *previewMedicalPrescriptionLines* és únic de la classe.

Així doncs, s'ha implementat 4 test:

1. Test que comprova que si les instruccions que s'afegeixen a la prescripció mèdica (array de *String*) són *null*, buides o contenen una imstància *null* de *ProductID* es llanci l'excepció corresponent.
2. Test que comprova que si les instruccions que es volen modificar són incorrectes llanci l'excepció corresponent.
3. Test que comprova que és possible eliminar una instància d'un *ProductID* i en conseqüència la seva instrucció, ja que es tracta d'una estructura *HashMap <K,V>*.
4. Test que comprova que és possible buscar una línia especifica de la prescripció mèdica.

3.2 Classe *MedicalPrescriptionLine*

Aquesta classe s'encarrega d'afegir, modificar o eliminar línies de la prescripció mèdica. A més a més, s'ha creat un mètode que permet retornar una llista formada per les línies de prescripció la qual s'utilitzarà en l'últim mètode implementat *toString*.

Per a la implementació d'aquesta classe s'han creat les 3 funcions principals que posteriorment s'utilitzaran en la classe *MedicalPrescription*. Tant en *addLine* com en *modifyLine* abans de poder realitzar aquesta acció primer es comprova que el producte i instruccions no siguin *null* i que les instruccios no continguin un array de *String* buit, en cas contrari llançarà l'excepció corresponent. Pel que fa a la funció *removeLine* només es comprova que el producte no sigui *null* i que aquest estigui contingut en l'estructura de dades.

Finalment s'implementen els mètodes: *toString* i *items* per poder retornar una llista formada per les línies de prescripció.

3.2.1 Test *MedicalPrescriptionLine*

Pel que fa al test, ja que aquestes funcions es criden en la classe *MedicalPrescription* s'ha decidit no implementar per evitar el *code smell* anomentat *Duplicated code*.

3.3 Classe *Posology*

Aquesta classe s'encarrega de representar les pautes amb les quals s'ha d'administrar una medicina.

Al constructor se li passa 2 per paràmetres de tipus *float* que representa la dosis del medicament i la freqüència i un tercer paràmetre que representa la freqüència d'administració de la medicina. Seguidament s'han generat els *getters* i *setters* dels 3 atributs de la classe.

Finalment s'implementen els mètodes: *toString*, *equals* i *hashCode*.

3.3.1 Test *Posology*

Pel que fa al test, en aquest cas no s'ha realitzat el test que contempli la possibilitat que a l'hora de generar el constructor no es passin els 2 paràmetres *float* establerts, ja que *Java* s'encarrega de llençar aquesta excepció.

Finalment s'ha comprovat que els *getters* i *setters* funcionin correctament.

3.4 Classe *ProductSpecification*

Aquesta classe s'encarrega d'afegir una descripció i un preu a un determinat producte.

Primer s'han generat els *getters* i *setters* dels 3 atributs de la classe. A continuació, s'ha creat un mètode anomenat *checkProductSpecification* que permet comprovar si es passa el tipus de dada desitjat, que la instància de *ProductID* que es passa per paràmetre no sigui *null* i que la descripció d'aquest producte i el preu tampoc sigui *null* ni buit. Si tot és correcte, es crearà una especificació d'un producte correctament, en cas contrari es llançarà l'excepció correctament.

Finalment s'ha implementat el mètode: *toString*.

3.4.1 Test *ProductSpecification*

Per a la classe *ProductSpecification*, s'han implementat 2 test diferents:

1. Test que comprova que qualsevol dels 3 paràmetres passats per una instància de *ProductSpecification* siguin *null* i en conseqüència llançarà l'excepció corresponent.
2. Test que comprova que la descripció d'un *ProductSpecification* sigui buida i llança l'excepció corresponent.

3.5 Classe *TakingGuidelines*

Aquesta classe s'encarrega de representar la informació relativa a les pautes d'administració d'una medicina, és a dir, com i quina quantitat.

Primer s'han generat els *getters* i *setters* dels atributs d'aquesta classe. En el constructor se li ha de passar 6 paràmetres dels quals 3 d'aquests són els mateixos que s'utilitzen per crear una instància de *Posology*. Així doncs, amb els paràmetres *dose*, *freq*, *freqUnit* es crearà una instància de *Posology* en el constructor i amb els altres 3 paràmetres *dayMoment*, *duration*, *instructions* s'inicialitzaran en el constructor.

Finalment s'implementen els mètodes: *toString*.

3.5.1 Test *TakingGuidelines*

Pel que fa al test, en aquest cas no s'ha realitzat el test que contempli la possibilitat que a l'hora de generar el constructor no es passin els paràmetres correctes, ja que *Java* s'encarrega de llençar aquesta excepció.

Finalment s'ha comprovat que els *getters* i *setters* funcionin correctament.

4 Paquet *services*

Aquest paquet està compost per les interfícies *HealthNationalService* i *ScheduledVisitAgenda*. S'han implementat, ja que per a la realització d'aquesta pràctica no es disposa d'aquest servei, per tant, les dues interfícies s'utilitzaran com a controladors en la classe i test de *ConsultationTerminal*.

4.1 Interfície *HealthNationalService*

Aquesta interfície està formada per quatre mètodes: *getePrescription*, *getProductsByKW*, *getProductSpecific*, *SendePrescription* els quals s'utilitzaran en el test de *ConsultationTerminal*, ja que simularà la interacció amb el *HealthNationalService*.

4.2 Interfície *ScheduledVisitAgenda*

Aquesta interfície està formada per un mètode: *getHealthCardID* que retorna la targeta sanitària d'un usuari.

5 Paquet *supervisedtreatment*

Durant la realització de la pràctica s'ha decidit crear aquest paquet el qual està compost per 1 classe anomenada *ConsultationTerminal* la qual és la més important, ja que executa totes les classes anomenades anteriorment.

5.1 Classe *ConsultationTerminal*

Aquesta classe s'encarrega de definir els esdeveniments d'entrada per al cas d'ús estudiat en les pràctiques anteriors de *Mòdul de Consulta Mèdica*.

Està composta per les següents funcions:

- *initRevision*: Mètode que s'encarrega d'inicialitzar la consulta mèdica i comprovar que el *SVA* retorna un *HealthCardID* vàlid. Si tot ha sortit correcte sortirà es generar la prescripció mèdica.
- *initPrescriptionEdition*: Comprova si s'ha generat algun error al intentar editar una prescripció mèdica.
- *searchForProducts*: Es connecta a l'*HNS* per descarregar de forma local els productes d'acord amb la seva paraula clau.
- *selectProduct*: Tria el producte desitjat i en descarrega el producte de l'*HNS* de forma sincronitzada.
- *enterMedicineGuidelines*: Introdueix les pautes d'administració a l'e-recepta.
- *enterTreatmentEndingDate*: Introdueix la data de finalització de l'e-recepta.
- *sendePrescription*: Prepara la e-recepta i envia la e-recepta a l'*HNS*.
- *printePresc*: Imprimeix la e-recepta, en cas de la pràctica no s'ha implementat, però si els possibles errors que es poguessin produir.

5.1.1 Test *ConsultationTerminal*

Pel que fa al test, s'han contemplat tots els possibles errors que podia generar les funcions explicades anteriorment (contemplat errors iguals i diferents al de les classes del paquets anteriors). La dificultat més gran ha estat a l'hora d'implementar el test doble per a les dues interfícies *HealthNationalService* i *ScheduledVisitAgenda*. Els casos que s'han comprovat són si s'introduïen de forma correcta els paràmetres de les funcions i que la funció d'enviament i retorn a l'*HNS* funcionés correctament.

6 Control de versions: *Github*

Per a la realització d'aquest projecte els membres del grup han utilitzat l'eina *Github*. Ja que es tractava del primer projecte en grup d'unes dimensions tan gran no s'han aprofitat tots els avantatges que proporciona aquesta eina. Per això, els membres del grup només han treballat sobre la branca *main*.

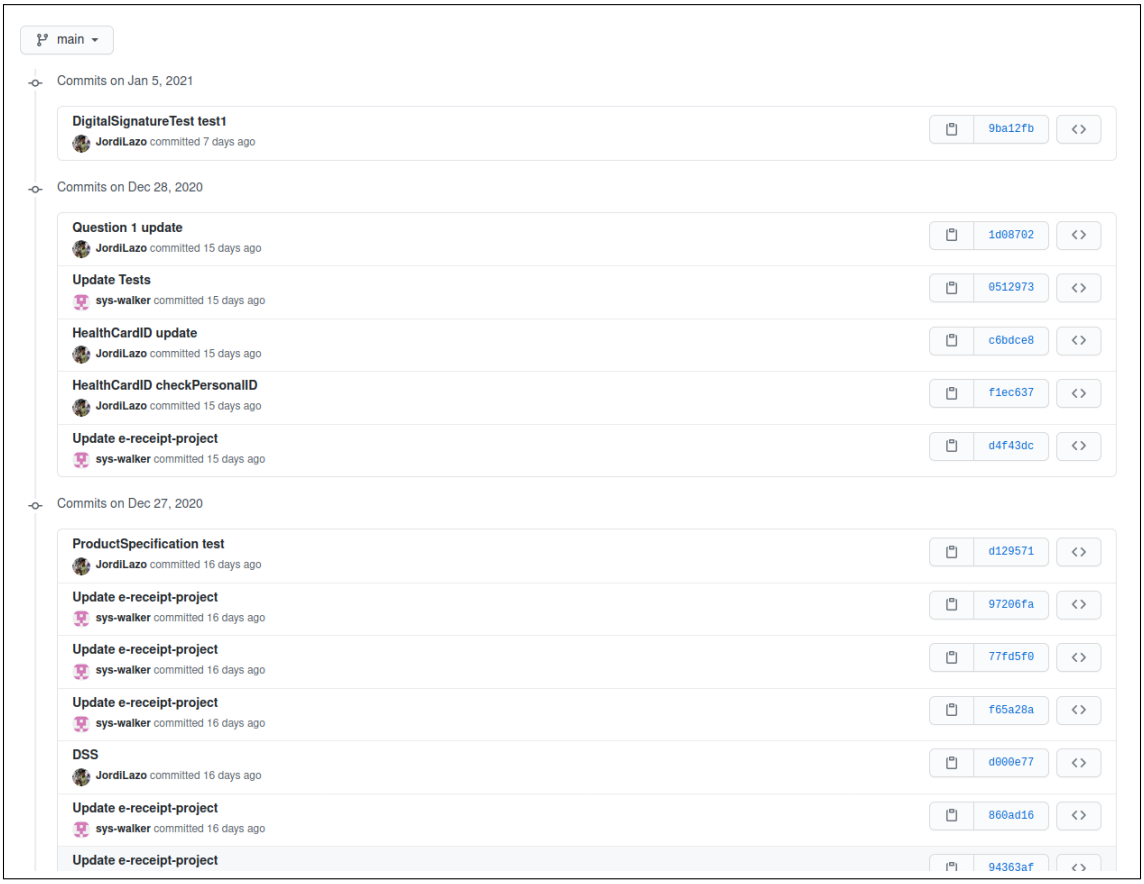

Figura 1: Canvis realitzats a la branca *main*.

Com es pot observar a la figura Figura 1, tots el *commit* i *push* s'han realitzat sobre la mateixa branca *main*. Per culpa de la manca de coneixements sobre les *branch* es va decidir repartir el treball a realitzar. Durant el transcurs de la pràctica, cada membre del grup ha creat la classe i test corresponent. Un cop finalitzat les tasques s'han fet el *push* i *commit* corresponents.

7 Conclusions

Després de la realització de les 4 pràctiques proposades durant tot el curs de l'assignatura *Enginyeria del Programari*, la conclusió final que s'ha arribat és que tot el desenvolupament d'aquestes pràctiques estan relacionades entre si.

Tots els passos realitzats anteriorment han sigut de vital importància per poder desenvolupar el programa en *Java* durant aquesta pràctica. El *DSS* i *Model de Domini* han sigut clau.

Pel que fa a les proves unitàries, per a poder realitzar la pràctica s'han hagut d'adquirir coneixements de l'API *Junit 5* que abans no es tenien. La utilització del *@BeforeEach* és indispensable a causa de la quantitat d'instàncies que requerien cada test i a la mateixa repetició de algun tests per a cada classe. Els exercicis proposats a classe han sigut útils per a la realització de tests. Sense cap dubte, la implementació d'un test doble ha sigut la part més difícil de desenvolupar.

Tot i això, gràcies a aquesta pràctica s'ha pogut relacionar tots els conceptes adquirits anteriorment com evitar la creació de *code smells*, saber implementar un diagrama *DSS* i la utilitat de les interfícies.

Finalment, per a la implementació de les classes ha sigut imprescindible tornar a repassar els coneixements teòrics i pràctics adquirits en l'assignatura de *Estructura de Dades*