Un exemple pràctic

Jordi Brínquez Jiménez

PID_00161658

Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

Aquest mòdul pretén dotar els estudiants d'un exemple complet del procés de desenvolupament d'una aplicació partint de zero perquè puguin assimilar millor els continguts dels mòduls anteriors, i també perquè es familiaritzin amb els procediments per a resoldre problemes.

Aquest mòdul està organitzat en tres grans blocs:

- Una primera part en què plantegem un enunciat que ens servirà per a la resta del mòdul.

- A continuació crearem el diagrama UML resultant del problema plantejat.

- Finalment resoldrem pas a pas els problemes de codificació que puguin sorgir.

Objectius

Els objectius d'aquest mòdul són:

1. Facilitar la comprensió dels conceptes explicats durant els mòduls anteriors, perquè els estudiants sigueu capaços de resoldre correctament la pràctica de l'assignatura.

2. Dotar l'estudiant d'un model de resolució de problemes mitjançant la programació orientada a objectes.

1. Enunciat

La UOC vol afegir una aplicació nova al conjunt d'utilitats que hi ha al campus per a facilitar l'estudi als estudiants.

L'aplicació en concret és un sistema gestor de preguntes i proves que permetrà realitzar i corregir en línia tests de qualsevol temàtica. D'aquesta manera, els usuaris (els estudiants) poden avaluar els seus coneixements sobre qualsevol tema que sigui del seu interès, per a poder preparar-se per a exàmens reals.

Per a optar a aquest servei, cal estar enregistrat com a estudiant dins el sistema de la UOC.

L'equip docent podrà crear i afegir preguntes en el sistema, a partir de les quals es podran generar els tests. D'aquesta manera, es poden aprofitar preguntes per a més d'un test.

Cal que els tests siguin donats d'alta en el sistema perquè els estudiants els puguin resoldre. El procés que cal seguir per a elaborar un test consisteix a destriar quines preguntes cal afegir-hi d'entre totes les preguntes donades d'alta en el sistema. Una característica que es vol que tingui el sistema és que permeti crear els tests en diferents instants de temps, és a dir, que permeti començar a crear un test, deixar-lo a mitges i continuar-lo més endavant. Fins que un test no estigui acabat, no pot estar disponible per als estudiants.

De moment, han definit un seguit de possibles tipus de preguntes, encara que no descarten que es puguin ampliar en un futur. Els diferents tipus de preguntes són:

- Preguntes textuals. Aquest tipus de preguntes esperen una resposta textual, en forma de cadena de text. Cal tenir en compte que les respostes han de ser curtes perquè es puguin corregir fàcilment de manera automatitzada i, per tant, mai no s'hauria de demanar que es desenvolupés un tema.

  Preguntes que entrarien en aquest format

  - Quina és la capital de Dinamarca?
  - Per quin nom es coneix també *La Gioconda*?

- Preguntes d'opció múltiple. Aquestes preguntes són les típiques dels tests, en què es donen diferents opcions i hi ha una única resposta correcta. Una pregunta d'opció múltiple ha de tenir, com a mínim, dues respostes possibles.

Sobre el funcionament respecte de l'estudiant, volen que pugui seleccionar de la llista dels tests disponibles dins el sistema el test que vol resoldre. Quan l'estudiant selecciona un test per resoldre'l, passa a estar disponible per a l'estudiant. Un estudiant pot tenir com a màxim un test disponible per resoldre a cada moment. L'estudiant no ha de tenir cap limitació de temps i ha de poder contestar les preguntes en l'ordre que cregui necessari. Quan l'estudiant indiqui al sistema que ja ha acabat el test, es considerarà tancat i no se'n podrà modificar cap resposta. En aquest instant, es procedirà a la correcció automàtica del test. Totes les preguntes no contestades es consideraran errònies i, en les altres, caldrà comprovar les respostes donades amb les respostes emmagatzemades.

Es considera de gran utilitat que es pugui accedir a la història de totes les proves realitzades per un estudiant. Es vol que hi figurin quines són les proves que ha realitzat, la nota de la prova i, en cas que tingui algun test a mitges, cal que també s'hi reflecteixi.

Es vol mantenir un registre de la resposta que l'estudiant ha donat a cada pregunta de cada test, per a poder donar una realimentació o *feedback* a l'estudiant que indiqui els temes que ha de repassar. Aquesta realimentació és un text associat amb cada pregunta independent de les respostes de l'estudiant. Un estudiant només pot realitzar un test una única vegada.

Per a intentar aclarir una mica quines dades volem que s'emmagatzemin, us oferim una llista d'atributs d'algunes entitats. A banda, s'han d'emmagatzemar aquelles dades que sorgeixin de la lectura de l'enunciat:

- L'entitat *Estudiant* té els atributs següents:
  - Un identificador únic dins el sistema
  - Nom
  - Cognoms
  - Adreça
  - Telèfon
  - Correu electrònic

- L'entitat *Test* té els atributs següents:
  - Un identificador únic dins el sistema
  - Títol (nom que es dóna a la prova)
  - Descripció (descripció curta, genèrica, que fa referència a tota la prova)

- L'entitat *Pregunta* té els atributs següents:
  - Un identificador únic dins el sistema
  - Enunciat de la pregunta
  - Realimentació (en cas de resposta errònia)
  - Resposta correcta
  - Puntuació de la pregunta

A partir de la descripció del problema que volem resoldre, us mostrem les operacions que volem que el nostre sistema permeti realitzar:

- La *Gestió de preguntes* ha de permetre:
  - Afegir una pregunta
  - Consultar una pregunta
  - Fer una llista de totes les preguntes del sistema

- La *Gestió de tests* ha de permetre:
  - Afegir un test
  - Consultar un test
  - Afegir una pregunta en un test
  - Eliminar una pregunta en un test
  - Fer una llista de les preguntes d'un test
  - Fer una llista de tots els tests del sistema
  - Finalitzar un test (fer-lo disponible perquè els alumnes el resolguin)

- La *Gestió d'estudiants* ha de permetre:
  - Afegir un estudiant
  - Modificar les dades d'un estudiant
  - Consultar les dades d'un estudiant
  - Fer una llista de tots els estudiants del sistema

- La *Resolució de tests* ha de permetre:
  - Iniciar la resolució d'un test per un estudiant
  - Contestar una pregunta del test
  - Donar un test per finalitzat

- L'*Avaluació* ha de permetre:
  - Consultar la nota d'un estudiant en una prova
  - Consultar les notes d'un estudiant

2. Resolució

Per a resoldre aquest problema, primer hem de generar un diagrama UML amb les classes que ens serviran per a modelitzar l'aplicació i, posteriorment, hem de codificar la solució en el llenguatge de programació que fem servir. Aquesta separació és necessària perquè el llenguatge de programació no ha de condicionar en cap moment el nostre disseny. Ateses les característiques d'un llenguatge determinat, en alguns casos podríem incórrer en errors de disseny.

2.1. Identificació de les classes

Inicialment, com hem comentat, caldrà identificar les classes que formaran part del nostre model, i també els seus atributs.

Per a fer la llista de classes necessàries subratllarem tots els substantius i les frases nominals del text.

Vegeu el mètode per a identificar classes explicat en el mòdul "Abstracció i classificació".

La UOC vol afegir una nova aplicació al conjunt d'utilitats que hi ha al campus per a facilitar l'estudi a l'estudiant.

L'aplicació en concret és un sistema gestor de preguntes i proves que permetrà realitzar i corregir en línia tests de qualsevol temàtica. D'aquesta manera, els usuaris (els estudiants) poden avaluar els seus coneixements sobre qualsevol tema que sigui del seu interès, per a poder preparar-se per a exàmens reals.

Per a optar a aquest servei, cal estar enregistrat com a estudiant dins el sistema de la UOC.

L'equip docent podrà crear i afegir preguntes en el sistema, a partir de les quals es podran generar els tests. D'aquesta manera, es poden aprofitar preguntes per a més d'un test.

Cal que els tests siguin donats d'alta en el sistema perquè els estudiants els puguin resoldre. El procés que cal seguir per a elaborar un test consisteix a destriar quines preguntes cal afegir-hi d'entre totes les preguntes donades d'alta en el sistema. Una característica que es vol que tingui el sistema és que permeti crear els tests en diferents instants de temps, és a dir, que permeti començar a crear un test, deixar-lo a mitges i continuar-lo més endavant. Fins que un test no estigui acabat, no pot estar disponible per als estudiants.

De moment, han definit un seguit de possibles tipus de preguntes, encara que no descarten que, en un futur, es puguin ampliar. Els diferents tipus de preguntes són:

Preguntes textuals. Aquest tipus de preguntes esperen una resposta textual, en forma de cadena de text. Cal tenir en compte que les respostes han de ser curtes perquè es puguin corregir fàcilment de manera automatitzada i, per tant, mai no s'hauria de demanar que es desenvolupés un tema.

Preguntes d'opció múltiple. Aquestes preguntes són les típiques preguntes de test, en què es donen diferents opcions de resposta i hi ha una única resposta correcta. Una pregunta d'opció múltiple ha de tenir, com a mínim, dues respostes possibles.

Sobre el funcionament, respecte de l'estudiant, volen que pugui seleccionar de la llista dels tests disponibles dins el sistema el test que vol resoldre. Quan l'estudiant selecciona un test per a resoldre'l, passa a estar disponible per a l'estudiant. Un estudiant pot tenir com a màxim un test disponible per a resoldre a cada moment. L'estudiant no ha de tenir cap limitació de temps i ha de poder contestar les preguntes en l'ordre que cregui necessari. Quan l'estudiant indiqui al sistema que ja ha acabat el test, es considerarà tancat i no se'n podrà modificar cap

resposta. En aquest instant, es procedirà a la correcció automàtica del test. Totes les preguntes no contestades es consideraran errònies i, en les altres, caldrà comprovar les respostes donades amb les respostes emmagatzemades.

Es considera de gran utilitat que es pugui accedir a la història de totes les proves realitzades per un estudiant. Es vol que hi figurin quines són les proves que ha realitzat, la nota de la prova i, en cas que tingui algun test a mitges, cal que també s'hi reflecteixi.

Es vol mantenir un registre de la resposta que l'estudiant ha donat a cada pregunta de cada test, per a poder donar una realimentació a l'estudiant que indiqui els temes que ha de repassar. Aquesta realimentació és un text associat amb cada pregunta independent de les respostes de l'estudiant. Un estudiant només pot realitzar un test una única vegada.

Després de llegir un altre cop i subratllar els substantius i les frases nominals, obtenim la llista següent:

- La UOC
- Aplicació
- Campus
- Estudiant
- Sistema gestor de preguntes i proves
- Tests
- Temàtica
- Usuaris
- Exàmens reals
- L'equip docent
- Preguntes
- Preguntes textuals
- Resposta textual
- Respostes
- Pregunta d'opció múltiple
- Resposta correcta
- Preguntes no contestades
- Respostes emmagatzemades
- Història de totes les proves realitzades per un estudiant
- Nota de la prova
- Test a mitges
- Registre de la resposta que l'estudiant ha donat a cada pregunta de cada test
- Realimentació

Un cop tenim aquesta llista, cal eliminar les classes incorrectes. Si seguim la metodologia explicada, arribem a les conclusions següents:

Vegeu el mòdul "Abstracció i classificació" d'aquesta asignatura.

a) Les classes següents són redundants:

– Sistema gestor de preguntes i proves: és un sinònim d'aplicació.
– Usuaris: representa el mateix concepte que estudiants.

b) Les classes següents resulten ser irrellevants:

– Campus: és un concepte que no farem servir per a resoldre el problema, però que podria ser necessari tenir-lo en compte en altres problemes.

– Exàmens reals: només denota que els tests emmagatzemats en el sistema simulen exàmens reals, però això no és de cap utilitat per al nostre sistema.
– Resposta textual: és un concepte que vol afegir informació al concepte de `resposta`, però que no aporta res de nou.

c) Les classes següents són vagues:

– La UOC: és una classe sobre la qual no se'ns demana que emmagatzemem cap dada directament.
– Temàtica: és un concepte que s'esmenta, però no es fa servir enlloc ni té una finalitat ben definida.

d) Els conceptes següents resulten ser atributs:

– Nota de la prova: aquest concepte no és una classe en si mateixa, sinó una propietat d'una relació entre un estudiant i un test.
– Realimentació: és un concepte inherent a la pregunta; per tant, ha de ser un atribut seu.
– Resposta correcta: haurem de tenir certa cura a l'hora de tractar aquest concepte perquè hi ha diversos tipus de preguntes.

e) Trobem els rols següents:

– Estudiant: és un rol de `Persona`, com ho és el concepte d'equip docent, però en aquest cas, l'equip docent serà l'encarregat d'inserir les dades en el sistema.

f) Sobre les estructures o conceptes de la implementació:

– Respostes emmagatzemades: no serà una classe directament, però sí un concepte que hem d'emmagatzemar per a cada test resolt de cada estudiant.

– Preguntes no contestades: si en comptes d'emmagatzemar el resultat de totes les preguntes, emmagatzemem les preguntes contestades amb la resposta donada (tal com demana l'enunciat), podem obtenir les preguntes no contestades sense haver de tenir una classe que les representi.

– Història de totes les proves realitzades per un estudiant: aquest concepte és una llista de les proves realitzades, no una classe, però caldrà tenir-la en compte perquè durant el disseny hem de garantir poder obtenir-lo.

– Registre de la resposta que l'estudiant ha donat a cada pregunta de cada test: aquest concepte no és directament una classe, però cal tenir-lo en compte a l'hora de realitzar el disseny.

– Test a mitges: és un concepte que no és directament una classe, però cal tenir-lo present durant el disseny.

Vegem quines classes són les que ens han quedat del procés de filtratge anterior:

- Aplicació
- Estudiant
- Tests
- Preguntes
- Preguntes textuals
- Pregunta d'opció múltiple
- Respostes

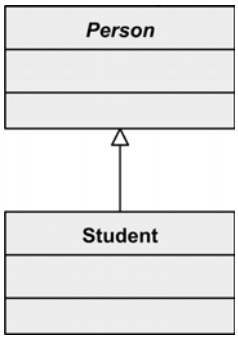
2.2. Creació de diagrames parcials de classes

Amb les classes de la llista anterior i l'enunciat construirem el diagrama de classes de la nostra aplicació.

Més que construir un diagrama de classes sencer, el que hem de fer és crear diagrames petits que més tard ajuntarem per crear el diagrama definitiu.

Primerament ens fixem en els usuaris del sistema. Tal com hem comentat anteriorment, la classe `Usuari` és una especialització de `Person` (`Persona`). Com que en principi no tindrem cap instància de la classe `Person` (o com a mínim l'enunciat no ens diu res sobre això), la podem definir com una classe abstracta i posteriorment, en cas de futures ampliacions del sistema, podem heretar d'aquesta per crear els rols nous.

D'aquesta manera, obtenim aquest tros del diagrama final:

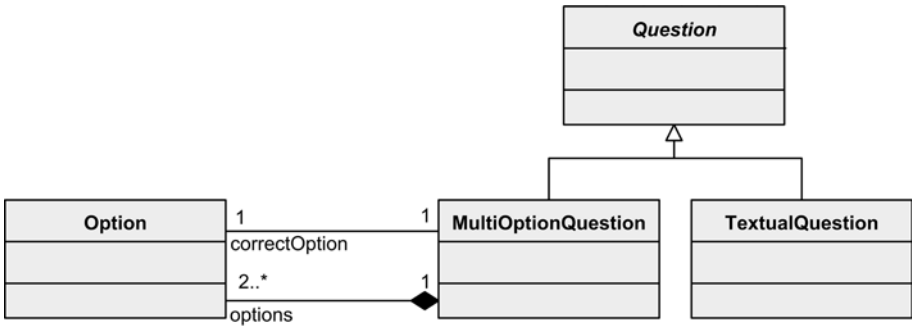


Tot seguit, ens fixarem en l'estructura de les preguntes del test. Repassem l'enunciat per tenir clares les relacions que hi ha:

> L'equip docent podrà crear i afegir preguntes en el sistema, a partir de les quals es podran generar els tests. D'aquesta manera, es poden aprofitar preguntes per a més d'un test.
>
> De moment, han definit un seguit de possibles tipus de preguntes, encara que no descarten que, en un futur, es puguin ampliar. Els diferents tipus de preguntes són:
>
> - Preguntes textuals. Aquest tipus de preguntes esperen una resposta textual, en forma de cadena de text. Cal tenir en compte que les respostes han de ser curtes perquè es puguin corregir fàcilment de manera automatitzada i, per tant, mai no s'hauria de demanar que es desenvolupés un tema.
> - Preguntes d'opció múltiple. Aquestes preguntes són les típiques preguntes de test, en què es donen diferents opcions de resposta i hi ha una única resposta correcta. Una pregunta d'opció múltiple ha de tenir, com a mínim, dues respostes possibles.

Com que ara ens concentrem en l'apartat de les preguntes, cal que destriem de l'enunciat el text que s'hi refereix.

Com que hi ha una entitat abstracta anomenada *preguntes* de la qual es diu que hi ha de diversos tipus (textuals i d'opció múltiple), podem deduir que hi ha una relació d'especialització/generalització (herència) entre aquestes classes, que podem modelitzar de la manera següent:



Com podeu veure, tenim una **classe abstracta,** `Question` (`Pregunta`), que ens ajuda a organitzar la jerarquia de classes de les preguntes. A més, hem representat gràficament que una `MultiOptionQuestion` (`PreguntaMultiOpcio`) té associades unes respostes, una de les quals és correcta.

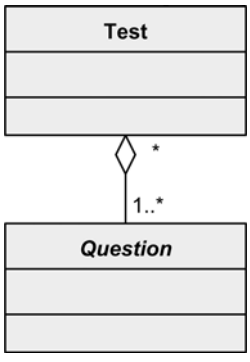
Caldria expressar que la resposta correcta ha de ser una de les respostes que formen part de la llista d'opcions possibles. En UML, es pot representar gràficament, però ja que no ho hem explicat en aquesta assignatura, posarem la restricció de paraula, que també és una opció vàlida.

Expresar condicions sobre els valors, relacions o altres conceptes d'un diagrama UML s'anomena crear una "restricció textual".

Després, en analitzar la relació entre els tests i les preguntes, trobem el següent:

> El procés que cal seguir per a elaborar un test consisteix a destriar quines preguntes cal afegir-hi d'entre totes les preguntes donades d'alta en el sistema.

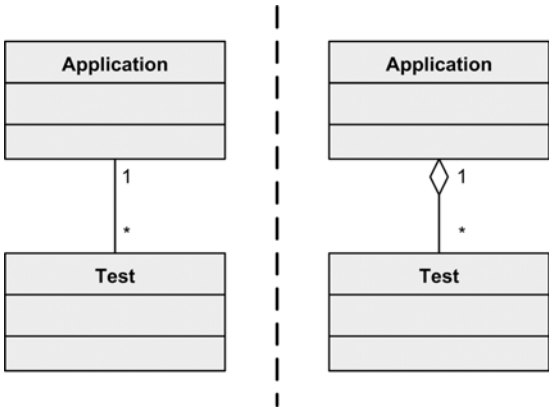
Per tant, `Test` és una agregació de preguntes, i el representarem així:



Aquesta relació és una agregació, ja que les preguntes tenen sentit fora del test (és a dir, hi pot haver preguntes sense cap test associat). Si les preguntes no tinguessin sentit sense un test, en comptes d'una agregació seria una composició.

En aquest punt, només cal veure com s'organitzen els tests dintre de l'estructura de l'aplicació; per tant, com que la UOC vol emmagatzemar tots

els tests i les preguntes, podem crear dos diagrames diferents segons el significat que volem que tingui dins la nostra aplicació:

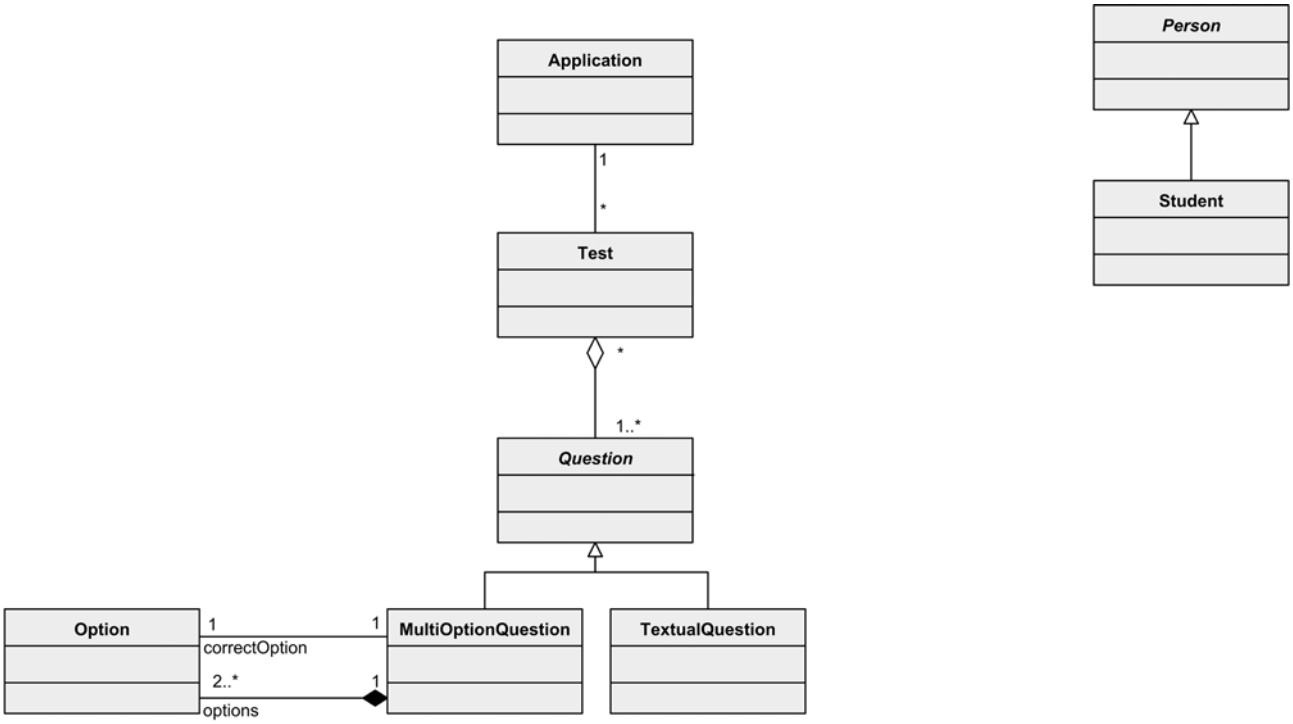


En el diagrama de la dreta, hi ha una relació d'agregació entre `Application` (`Aplicacio`) i `Test`; per tant, indiquem que l'aplicació té tests i que aquests formen part de l'aplicació. En l'altre diagrama, en canvi, tant el test com l'aplicació tenen entitat per si mateixos, i l'única cosa que els uneix és una relació mútua.

Totes dues solucions són vàlides i representen el nostre problema; l'elecció d'una o de l'altra només depèn del significat que volem que tingui aquesta relació. Nosaltres farem servir el diagrama de l'esquerra, el que representa la relació entre classes.

2.3. Creació del diagrama de classes

Ara ja podem ajuntar tots els diagrames petits i per crear un diagrama preliminar amb totes les classes:

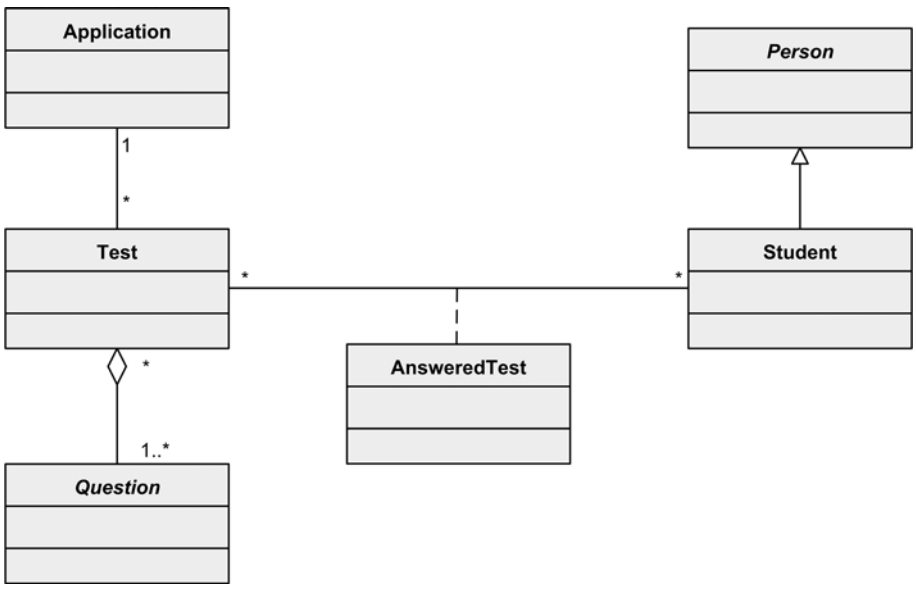


Com podeu observar, falta enllaçar els estudiants amb la resta de classes, principalment amb la classe `Application`, i afegir tot el que es relaciona amb la resolució dels tests per part dels alumnes, que començarem a resoldre a continuació.

Revisem una vegada més l'enunciat:

> Sobre el funcionament respecte de l'estudiant, volen que pugui seleccionar de la llista dels tests disponibles dins el sistema el test que vol resoldre. Quan l'estudiant selecciona un test per resoldre'l, passa a estar disponible per a l'estudiant. Un estudiant pot tenir com a màxim un test disponible per a resoldre a cada moment. L'estudiant no ha de tenir cap limitació de temps i ha de poder contestar les preguntes en l'ordre que cregui necessari. Quan l'estudiant indiqui al sistema que ja ha acabat el test, el test es considerarà tancat i no se'n podrà modificar cap resposta. En aquest instant, es procedirà a la correcció automàtica del test. Totes les preguntes no contestades es consideraran errònies i, en les altres, caldrà comprovar les respostes donades amb les respostes emmagatzemades.
>
> Es considera de gran utilitat que es pugui accedir a la història de totes les proves realitzades per un estudiant. Es vol que hi figurin quines són les proves que ha realitzat, la nota de la prova i, en cas que tingui algun test a mitges, cal que també s'hi reflecteixi.
>
> Es vol mantenir un registre de la resposta que l'estudiant ha donat a cada pregunta de cada test, per a poder donar una realimentació a l'estudiant que indiqui els temes que ha de repassar. Aquesta realimentació és un text associat amb cada pregunta independent de les respostes de l'estudiant. Un estudiant només pot realitzar un test una única vegada.

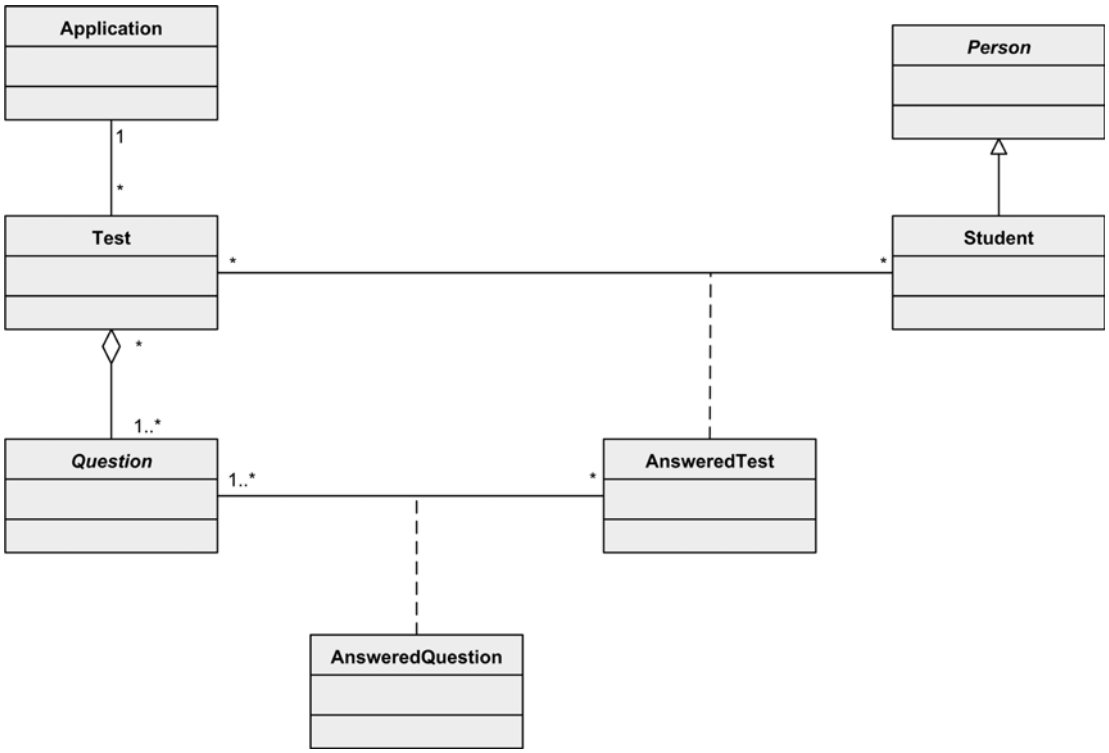
Per tant, tenim que un `Student` (`Estudiant`) només pot tenir "assignat" un `Test` a cada moment per resoldre'l i, un cop resolt –és a dir, quan se'l consideri tancat–, aquest estudiant no el pot tornar a resoldre mai més. Del concepte de resolució d'un `Test` per part d'un `Student` cal emmagatzemar algunes dades (nota, preguntes contestades, etc.); per tant, el que tenim és una classe associativa entre `Student` i `Test`, que anomenarem `AnsweredTest` (`TestContestat`). Afegint aquestes modificacions al disseny, resulta el diagrama següent:



Hem eliminat del diagrama tota l'estructura referent a les `Preguntes`, ja que no aportaven cap informació per a aquesta part.

Cal representar el concepte que un test és actiu perquè el pugui resoldre un alumne. Per a resoldre aquest concepte, farem servir un atribut de tipus booleà que posarem en la classe `AnsweredTest` i només podrem tenir una instància activada per a cada estudiant.

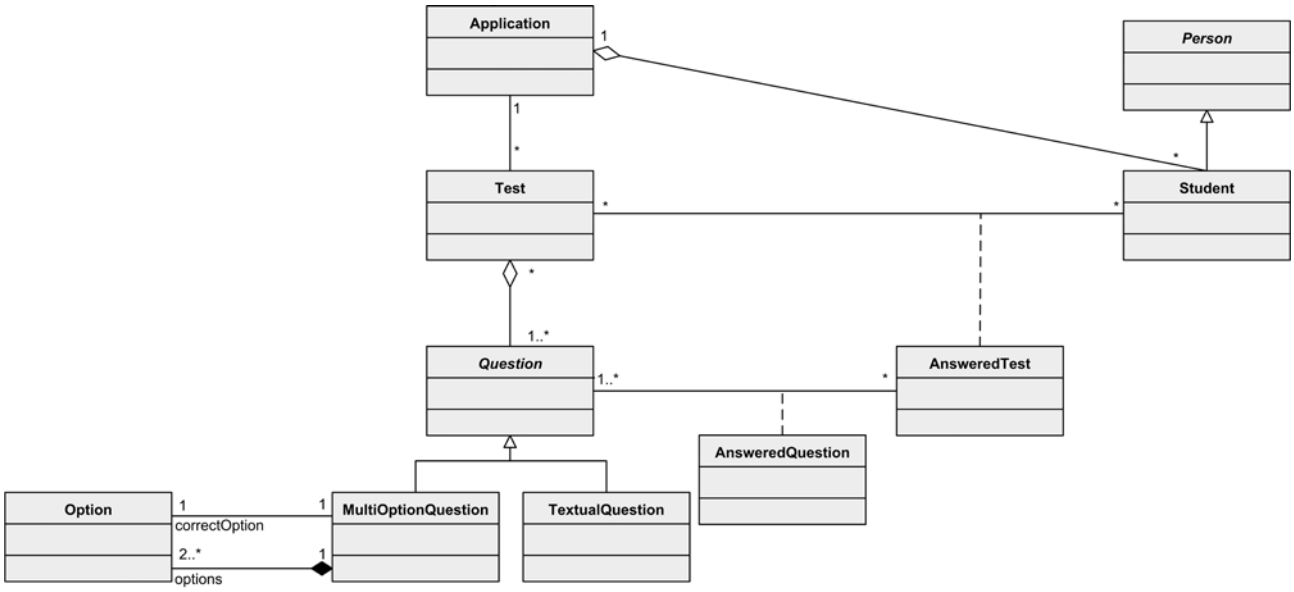
Ara que ja tenim representat el concepte de resolució d'un test, ens falta poder mantenir les respostes donades per a cada pregunta del test, que podem representar mitjançant una altra classe associativa entre `AnsweredTest` i `Question`, que anomenarem `AnsweredQuestion` (`PreguntaResolta`).



I llavors resta relacionar els estudiants (`Student`) amb l'aplicació, a partir d'aquest punt de l'enunciat:

> Per a optar a aquest servei, cal estar enregistrat com a estudiant dins el sistema de la UOC.

Vegem que hem de crear una relació entre `Application` i `Student` per tenir l'enunciat representat completament en el nostre diagrama, amb la qual cosa quedaria de la manera següent:



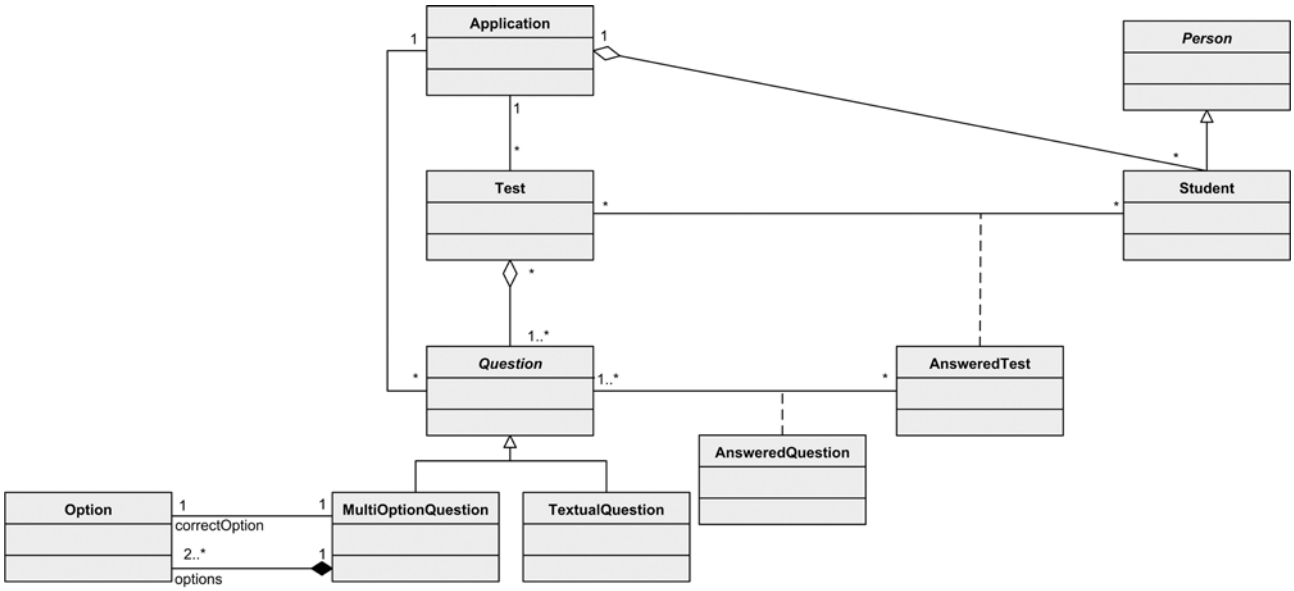
Malgrat que sembli acabat, cal repassar el diagrama diverses vegades per a assegurar-nos que compleix tots els requisits del problema donat. Hem de tenir en compte que l'enunciat del problema és un text que pot induir a diverses interpretacions segons la persona que el llegeixi, però un diagrama de classes té una única interpretació.

Si repassem l'enunciat una vegada més, podem arribar a veure que ens falta una de les relacions que l'enunciat esmenta, concretament, la relació entre preguntes i aplicació.

> L'equip docent podrà crear i afegir preguntes en el sistema, a partir de les quals es podran generar els tests. D'aquesta manera, es poden aprofitar preguntes per a més d'un test.

Per tant, cal afegir una nova relació entre `Application` i `Question`. En aquest cas, hi ha el mateix problema que hi havia abans entre `Application` i `Test`. Hi ha dues interpretacions possibles: nosaltres, com hem fet abans, representarem una relació binària entre les classes, i no farem servir l'agregació.

Finalment el diagrama de classes resultant es el següent:



2.4. Assignació d'atributs i mètodes

Un cop tenim enllestit el diagrama de classes, cal omplir-lo afegint els atributs i mètodes que l'enunciat ens demana, i també aquells resultants de les decisions de disseny que hàgim pogut prendre durant el procés anterior.

És a dir, per a cada classe del diagrama anterior hem d'indicar els atributs i els mètodes.

2.4.1. Classe `Person`

Com indica l'enunciat, la classe `Person` ha de tenir els atributs següents:

- Un identificador únic dins el sistema
- Nom
- Cognoms
- Adreça
- Telèfon
- Correu electrònic

Cal que definim un atribut de classe que funcioni d'identificador únic per a assignar un identificador diferent a cada persona. A banda, tindrem un mètode constructor al qual haurem de donar tots aquests atributs (a excepció de l'identificador, que atès que serà únic en el sistema, l'haurem d'assignar en el moment de crear la instància).

Entre altres mètodes que podem necessitar, hi ha els que ens permeten consultar les dades de la persona, també anomenats *getters*, que es defineixen així:

```
getFieldName():fieldType
```

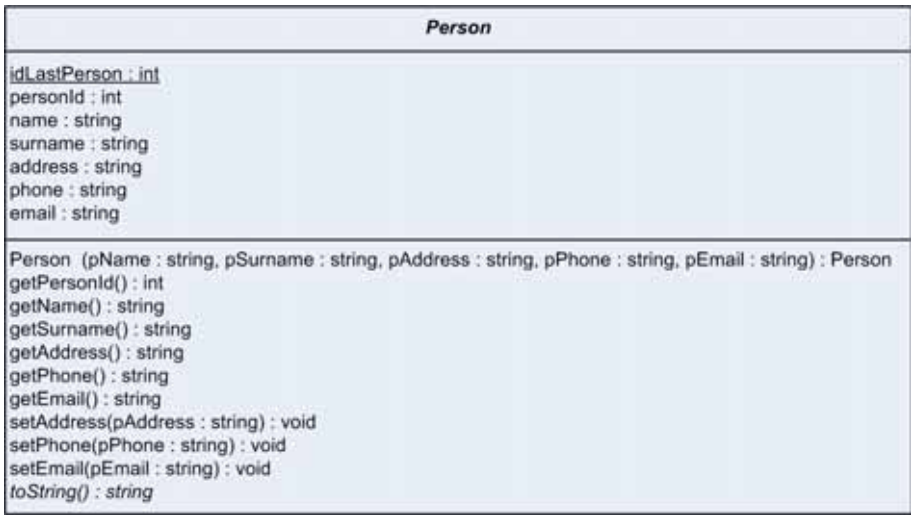
A banda, també hem de definir els mètodes que ens permetran modificar les dades, també anomenats *setters*, que es defineixen de manera similar als *getters*.

El nom de *getters* i *setters* deriva del significat de les paraules *get* i *set* en anglès ('recuperar' i 'posar').

```
setFieldName(fieldType value):void
```

En aquest cas, no necessitem tots els mètodes *setter*, ja que en una persona no canviarem l'identificador, ni el nom i cognoms, però sí que podria ser que ens calgués canviar l'adreça, el telèfon o el correu electrònic. Podríem afegir un *getter* i un *setter* per a l'atribut `personId` (`identificadorDePersona`), però com que és un atribut de classe que només llegirem i modificarem en el mètode constructor de la classe, ens podem estalviar afegir-los al diagrama amb la finalitat de guanyar claredat.

La classe `Person` resultant és la següent:



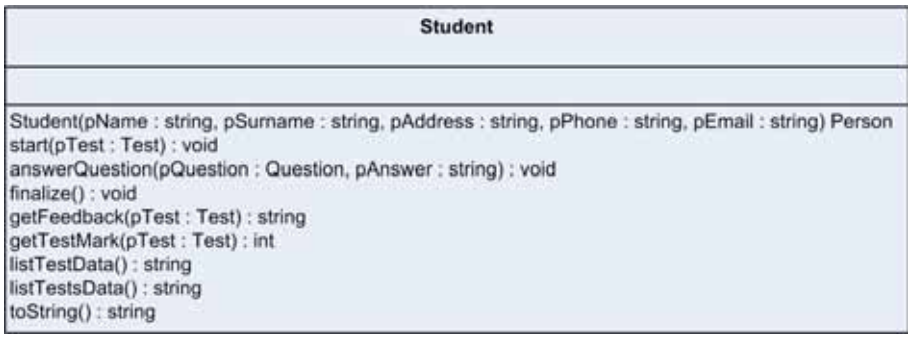
2.4.2. Classe `Student`

En aquest cas, com que la classe `Student` hereta de la classe `Person`, hem de tenir en compte que tots els atributs i mètodes definits en la classe `Person` és com si estiguessin definits en la classe `Student`; per tant, hem d'analitzar què diferencia la persona de l'estudiant per veure els atributs i mètodes que hem de definir en aquesta classe.

Quant als atributs, l'enunciat no ens indica que hàgim d'afegir-ne cap. Pel que fa als mètodes, a banda de necessitar un mètode constructor amb els mateixos paràmetres que la classe `Person`, haurem de permetre crear una llista de les dades dels estudiants (una espècie de resum de totes les dades) i també els tests realitzats per l'estudiant; per tant, caldrà que afegim un parell de mètodes que ofereixin aquesta informació en forma de cadena de text.

Pel que fa als atributs, caldrà afegir un mètode per a començar a resoldre un test, un altre per a contestar-ne les preguntes i un altre per a finalitzar-ne la resolució. I també necessitarem un parell de mètodes per a recuperar la nota i els comentaris dels tests ja realitzats.

Per tant, la classe `Student`, afegint els mètodes proposats, queda de la manera següent:



2.4.3. Classe `Question`

Com indica l'enunciat, una pregunta ha de tenir els camps següents:

- Un identificador únic dins el sistema
- Enunciat de la pregunta
- Realimentació (en cas de resposta errònia)
- Resposta correcta
- Puntuació de la pregunta

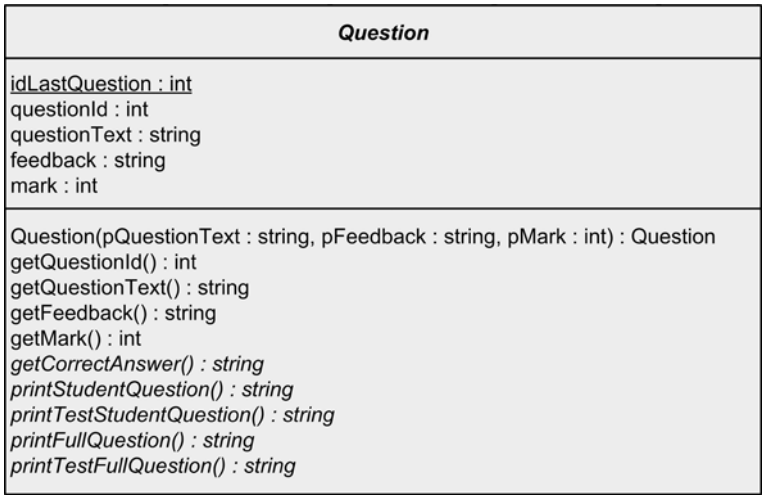
Com ens passa amb la classe `Person`, necessitem un atribut de classe per a emmagatzemar l'últim identificador assignat i un altre d'instància per a emmagatzemar l'identificador de cada instància.

L'enunciat de la pregunta i la seva realimentació i puntuació també seran atributs i, com que estem definint una classe abstracta, deixarem per a les

classes que l'especialitzen la definició de la resposta correcta, ja que en cada una d'elles caldrà definir-la d'una manera o d'una altra.

Sobre els mètodes d'aquesta classe, tindrem, com en casos anteriors, els *getters* dels atributs i l'operació constructora que, tot i tenir-la definida, redefinirem en les classes que l'especialitzen. Dels *getters*, n'hi haurà un que definirem com a abstracte, perquè cada subclasse el definirà d'una manera o d'una altra. També definirem un mètode que ens permeti mostrar totes les dades de la pregunta i un altre que ens doni les dades de la pregunta preparades per a ser mostrades a l'estudiant.

A partir de les premisses anteriors, la classe `Question` queda definida així:

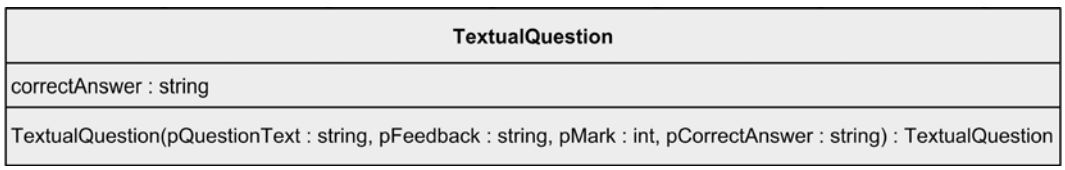


2.4.4. Classe `TextualQuestion`

Com en el cas de l'`Student`, la classe `TextualQuestion` (`PreguntaTextual`) hereta d'una altra classe, en aquest cas de `Pregunta`; per tant, com hem fet abans, no ens hem de preocupar de definir un altre cop els atributs i mètodes descrits anteriorment.

El que sí que haurem de fer és definir la representació de la pregunta correcta que, per a la classe `TextualQuestion`, serà un atribut de tipus `string`, i també n'haurem de definir l'operació constructora.

D'aquesta manera, la classe `TextualQuestion` queda definida com en la figura següent:



2.4.5. Classe `MultiOptionQuestion`

En el cas de la `MultiOptionQuestion` (`PreguntaMultiOpcio`), ens succeeix una cosa semblant a la del cas anterior, però atès que hem d'emmagatzemar les diferents opcions, la resposta correcta i el número de l'opció, haurem de modificar l'operació constructora per tal que reflecteixi aquestes dades i haurem d'afegir l'atribut que ens permeti emmagatzemar la resposta correcta.

MultiOptionQuestion
correctAnswer : Option
MultiOptionQuestion(pQuestionText : string, pFeedback : string, pMark : int, pCorrectAnswer : int, pOptions : Option[]) : MultiOptionQuestion

2.4.6. Classe `Option`

La classe `Option` només ha d'emmagatzemar el text de cada pregunta; per tant, tindrà un atribut amb el text de l'opció, una operació constructora i una altra de consultora del text emmagatzemat.

Option
optionText : string
Option(pOptionText : string) : Option getOptionText() : string

2.4.7. Classe `Test`

Aquesta classe ha d'emmagatzemar les dades referents al test que hem indicat anteriorment:

- Un identificador únic dins el sistema
- Títol (nom que se li dóna a la prova)
- Descripció (descripció curta, genèrica, que fa referència a tota la prova)

Donats aquests atributs i els mètodes típics (constructor, *getters* i *setters*), vegem què ens diu l'enunciat que hem de poder fer sobre un test:

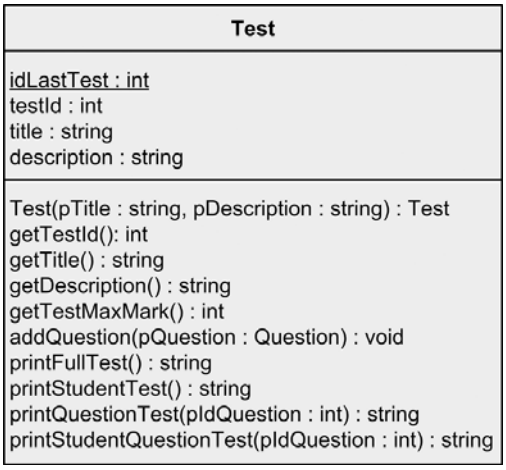
> Cal que els tests siguin donats d'alta en el sistema perquè els estudiants els puguin resoldre. El procés que cal seguir per a elaborar un test consisteix a destriar quines preguntes cal afegir-hi d'entre totes les preguntes donades d'alta en el sistema. Una característica que es vol que tingui el sistema és que permeti crear els tests en diferents instants de temps, és a dir, que permeti començar a crear un test, deixar-lo a mitges i

continuar-lo més endavant. Fins que un test no estigui acabat, no pot estar disponible per als estudiants.

...

Sobre el funcionament respecte de l'estudiant, volen que pugui seleccionar de la llista dels tests disponibles dins el sistema el test que vol resoldre. Quan l'estudiant selecciona un test per resoldre'l, passa a estar disponible per a l'estudiant. Un estudiant pot tenir com a màxim un test disponible per a resoldre a cada moment. L'estudiant no ha de tenir cap limitació de temps i ha de poder contestar les preguntes en l'ordre que cregui necessari. Quan l'estudiant indiqui al sistema que ja ha acabat el test, el test es considerarà tancat i no se'n podrà modificar cap resposta. En aquest instant, es procedirà a la correcció automàtica del test. Totes les preguntes no contestades es consideraran errònies i, en les altres, caldrà comprovar les respostes donades amb les respostes emmagatzemades.

Per tant, hem d'oferir també un mètode per afegir una pregunta al test, un mètode que ens permeti crear una llista amb les dades d'una pregunta i un altre que ens permeti crear una llista del test complet (tant per a l'alumne com per a l'equip docent); finalment, hem de donar la nota màxima del test perquè l'estudiant pugui avaluar-ne posteriorment el resultat.



2.4.8. Classe `AnsweredTest`

En aquest cas, un test contestat, com que és una classe associativa, el que hem de fer és emmagatzemar la relació entre un objecte de la classe `Test` i un altre de la classe `Student`.

En aquesta classe, a banda de la relació entre les classes, hem d'emmagatzemar la nota del test i si l'alumne té el test a mitges. Definirem aques concepte de la manera següent: un test a mitges és aquell test que l'alumne té asignat per a la seva resolució, però que encara no té nota assignada. Si ens fixem en els mètodes que aquesta classe ha de tenir, necessitem un mètode que ens permeti donar el test per acabat –i per tant, poder corregir-lo i assignar-li la nota–, i també un altre mètode per a emmagatzemar la resposta de cada una de les preguntes.

AnsweredTest
mark : int
AnsweredTest(pTest : Test, pStudent : Student) : AnsweredTest getMark() : int getFeedback() : string answerQuestion(pQuestion : Question, Answer : string) : void finalize() : bool printStudentTest() : string printReport() : string

2.4.9. Classe `AnsweredQuestion`

Aquesta classe ens servirà per a emmagatzemar les respostes donades per un estudiant a una pregunta d'un test. No hem d'emmagatzemar cap altra dada, però sí que hem de definir un mètode que ens permeti comprovar si la resposta donada és correcta o no.

AnsweredQuestion
answer : string
AnsweredQuestion(pTest : Test, pStudent : Student, pQuestion : Question, pAnswer : string) : AnsweredQuestion isCorrectAnswer() : bool getMark() : int

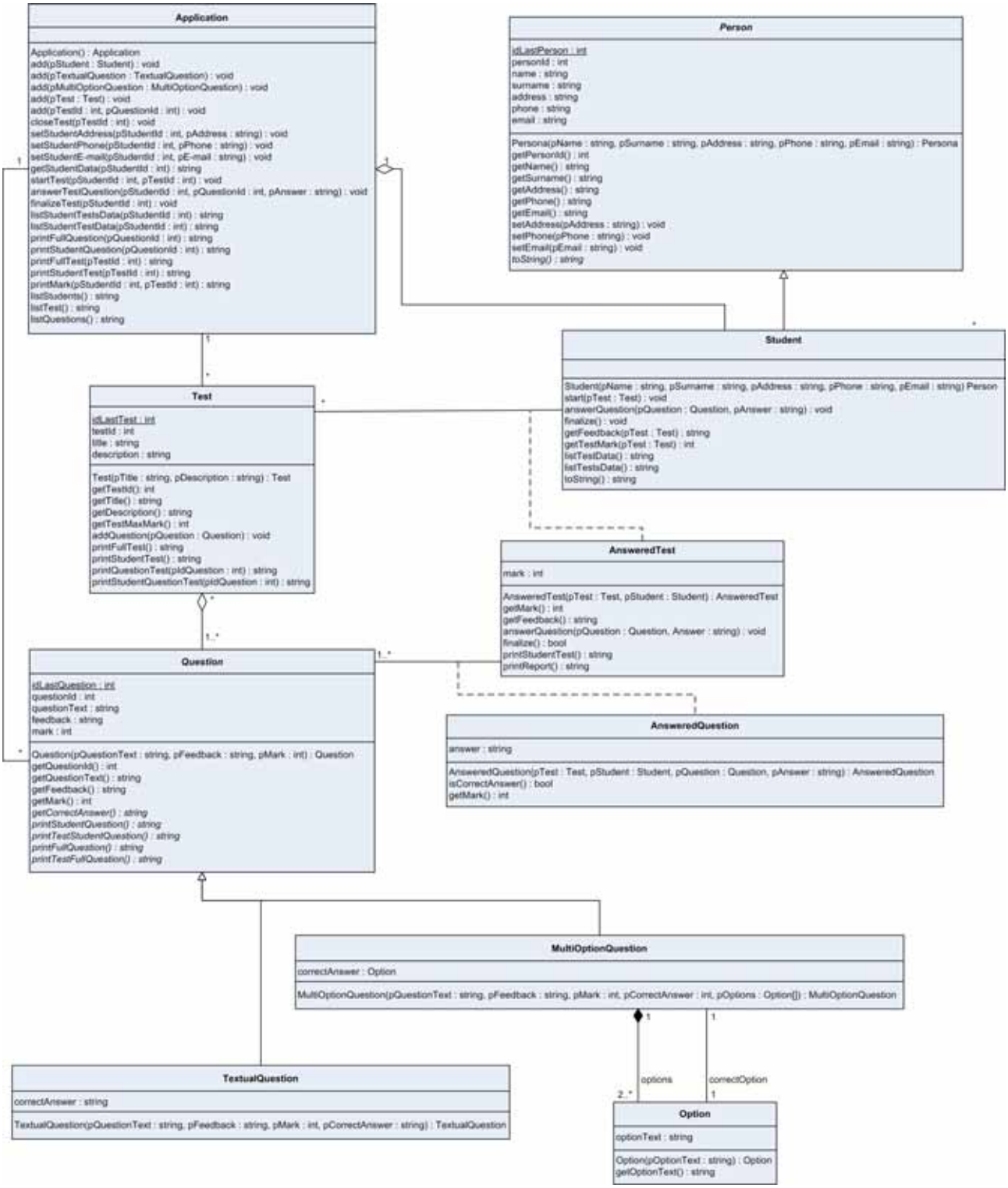
2.4.10. Classe `Application`

En aquesta classe, en canvi, segons el que hem plantejat en el problema que cal resoldre, hem de definir una gran quantitat de mètodes per a poder realitzar totes les tasques que necessitem que faci.

Aplication
Aplication() : Aplication add(pStudent : Student) : void add(pTextualQuestion : TextualQuestion) : void add(pMultiOptionQuestion : MultiOptionQuestion) : void add(pTest : Test) : void add(pTestId : int, pQuestionId : int) : void closeTest(pTestId : int) : void setStudentAddress(pStudentId : int, pAddress : string) : void setStudentPhone(pStudentId : int, pPhone : string) : void setStudentE-mail(pStudentId : int, pE-mail : string) : void getStudentData(pStudentId : int) : string startTest(pStudentId : int, pTestId : int) : void answerTestQuestion(pStudentId : int, pQuestionId : int, pAnswer : string) : void finalizeTest(pStudentId : int) : void listStudentTestsData(pStudentId : int) : string listStudentTestData(pStudentId : int) : string printFullQuestion(pQuestionId : int) : string printStudentQuestion(pQuestionId : int) : string printFullTest(pTestId : int) : string printStudentTest(pTestId : int) : string printMark(pStudentId : int, pTestId : int) : string listStudents() : string listTest() : string listQuestions() : string

2.5. Construcció del diagrama complet

Un cop hem completat totes les classes, és recomanable dibuixar el diagrama amb les classes completes per a veure si necessitem alguna dada més o si ens hem deixat algun mètode.



Noteu que la distribució ha canviat per a facilitar-ne el dibuix.

2.6. Inclusió de la navegabilitat en el diagrama de classes

Pot semblar que el diagrama anterior ja està acabat i que ja ens podem posar a picar el codi directament, però encara ens falta un últim pas per a poder dir que hem completat el diagrama: afegir la navegabilitat.

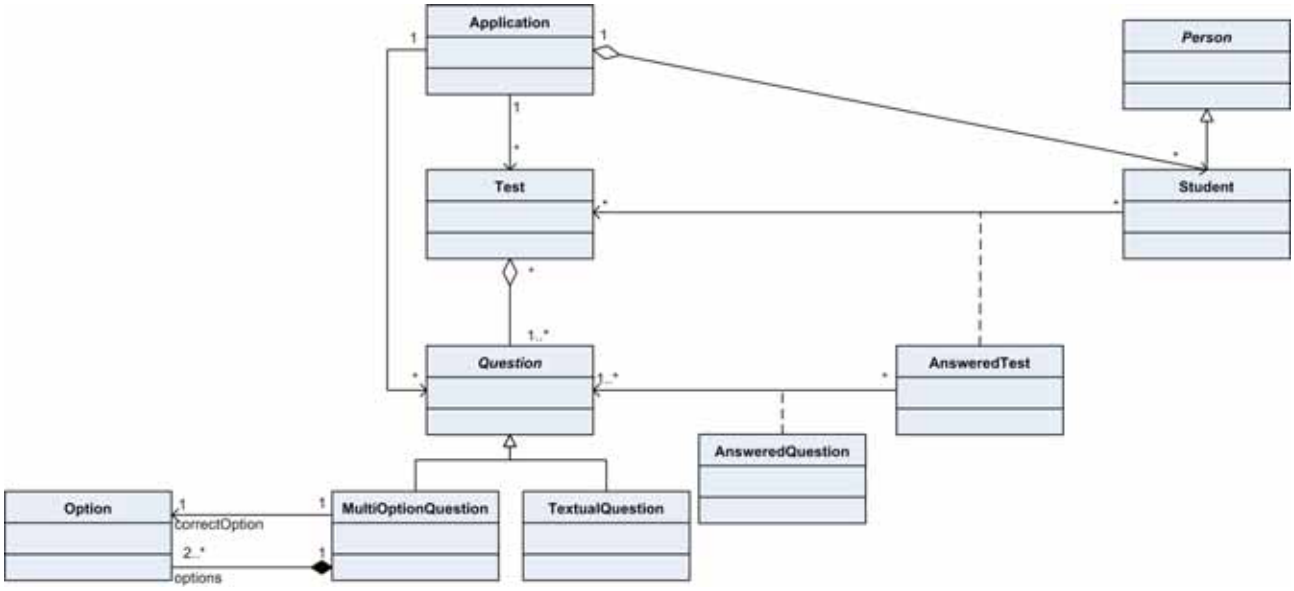
Com ja hem vist, la navegabilitat ens mostra quines classes tenen constància de l'existència de les altres; per exemple, en el nostre cas, l'aplicació ha de saber quins tests té, però un test no cal que sàpiga en quina aplicació és, ja que no necessita invocar-ne cap operació.

Per a assignar la navegabilitat correctament, cal tornar a llegir el text del problema i assignar la navegabilitat on correspongui.

Després de llegir de nou l'enunciat, podem definir les navegabilitats següents:

- `Application` → `Test`
- `Application` → `Student`
- `Application` → `Question`
- `Student` → `Test`
- `MultiOptionQuestion` → `Option`
- `AnsweredTest` → `Question`

El diagrama de classes final queda de la manera següent (hem eliminat els atributs i mètodes novament per facilitar la comprensió).



2.7. Codificació

Sobre la codificació, només comentarem alguns canvis que s'han produït a partir del disseny original (el diagrama de classes anterior) a partir de les facilitats/limitacions que té el llenguatge de programació Java que farem servir.

Per començar, la classe `Option` ha estat eliminada i transformada en una llista de `Strings` dintre de la classe `MultiOptionQuestion`.

Aquesta decisió ha estat presa perquè la classe no té cap altra utilitat que emmagatzemar unes respostes per a cada pregunta i, d'aquesta manera, se simplifica lleugerament el disseny.

Per a la implementació, cal comentar que no es pot fer tot de cop –és a dir, s'ha d'estructurar la codificació inicialment en parts independents– i, de mica en mica, les hem d'ajuntar per crear finalment l'aplicació demanada.

Una recomanació és començar implementant les classes que no depenen de cap altra classe, que en el nostre problema són `Person` i `Question`. Un cop les hem implementat, no les podem provar directament, ja que són classes abstractes.

Per tant, hem de continuar implementant les classes que hereten de les dues classes anteriors, `Student`, `TextualQuestion` i `MultiOptionQuestion`. Podem implementar i provar aquestes classes creant petits programes que accedeixin a tots els seus mètodes per a saber si funcionen correctament.

Un cop ja hem implementat la part relativa a les preguntes i a les persones, el més adient serà implementar la classe `Test` (ja tenim la classe `Question` i les classes que hereten); per tant, podem implementar sense cap dificultat la classe `Test` i l'agregació de test i preguntes.

Arribats a aquest punt, és interessant crear ja la classe `Application`, perquè d'aquesta manera establim el lligam entre tests, preguntes i persones i, posteriorment, implementarem les classes que ens permeten utilitzar les classes associatives `AnsweredTest` i `AnsweredQuestion`.

No podem implementar totalment la classe `Application` sense haver implementat les classes `AnsweredTest` i `AnsweredQuestion`, però un cop que hàgim implementat aquestes classes, ho podem deixar gairebé tot enllestit per acabar definitivament tot el codi.

Com podeu veure, resoldre un problema es redueix a resoldre problemes més petits que, tots junts, formen el problema inicial.

El codi i la documentació necessaris per a resoldre aquest enunciat, els podeu trobar a la vostra aula.

Resum

En aquest mòdul, hem vist com, des d'un text que ens descriu un problema, hem anat descomponent el problema en unitats petites que hem resolt de manera gairebé mecànica.

Inicialment cal identificar les entitats del problema, cosa que hem fet seguint un mètode senzill que ens servirà en la majoria dels casos, encara que l'experiència és sempre la millor amiga en aquestes tasques.

Posteriorment hem creat relacions entre aquestes entitats sense tenir en compte el contingut exacte d'aquestes, però pensant en el concepte que representen i la relació que hi ha entre les entitats del món real que volem representar.

Més endavant, hem emplenat de contingut aquestes entitats afegint-los atributs i mètodes, de manera que no només són entitats abstractes, sinó que les hem dotat de significat.

Una de les tasques en què hem de ser més crítics és en la comprovació que el model construït representa el nostre problema, atès que ens podria passar que alguna cosa de les que necessitem representar no es pot representar per alguna limitació del nostre model.

Posteriorment hem procedit a codificar el model en el llenguatge determinat.

Exercicis d'autoavaluació

1. Es decideix crear un nou tipus de pregunta tipus test de resposta múltiple. Què s'hauria de modificar per a poder introduir aquest nou tipus?

2. Què s'hauria de canviar del model original si no ens interessés mantenir un historial dels tests resolts?

3. A quines classes afectaria la creació d'un nou atribut en la classe `Persona`?

4. Si volem afegir un comptador per a cada test que ens digui quantes vegades s'ha resolt, en quina classe hem de posar aquest atribut? Quines classes afectaria?

Solucionari

1. Per a incloure aquest nou tipus de pregunta, el que hauríem de fer és crear una nova subclasse de `Pregunta` amb característiques semblants a les de la classe `PreguntaMultiOpcio`, però que permetés emmagatzemar respostes múltiples.

Una altra manera de resoldre aquest problema seria modificar la classe `PreguntaMultiOpcio` i permetre que accepti directament més d'una resposta vàlida. En aquest cas, si només hi ha una resposta vàlida, funcionarà de la mateixa manera que la classe del model original.

2. Únicament hauríem de canviar la multiplicitat de la relació entre la classe `Estudiant` i la classe `Test`. Hauríem de modificar el símbol * del costat de la classe `Test` per un 1, i d'aquesta manera només tindrem un test emmagatzemat per a cada estudiant.

Conseqüentment, haurem d'eliminar de la classe `UOC` i la classe `Estudiant` els mètodes que permeten accedir a la informació d'aquest registre històric.

3. La inclusió d'un nou atribut en la classe `Persona` fa que hàgim de modificar la classe `Estudiant` (el mètode constructor ha de permetre un paràmetre nou) i la classe `UOC` perquè accepti aquest paràmetre nou, tant en la creació d'estudiants com en els mètodes d'accés (accessors) i modificadors.

No hem de modificar cap altra classe més, ja que aquest canvi és transparent per a les altres classes.

4. Aquest atribut, l'hem de posar únicament en la classe `Test`; ha de ser un atribut privat amb un únic mètode d'accés per a consultar-ne el valor i un altre per a incrementar-lo en una unitat.

Les classes afectades únicament serien dues: la classe `Test`, per la inclusió de l'atribut i del mètode, i la classe `Estudiant`, que hauria d'invocar el mètode cada vegada que l'estudiant inicia la resolució d'un test.