Orientación a Objetos 2
Trabajo Práctico 7
Documentación de Refactoring

Grupo 39
Nahuel Bigurrarena 15782/3
Ariel Dapia Graziani 16056/2

Detectando Bad Smells y planificando su refactoring en el modelo.

Answer y Question

Empezando con la clases Question y Answer nos encontramos que ambas tienen varias cosas en común, como variables y métodos. Esto es un bad smell de Duplicate Code y utilizamos Extract to Superclass creando una clase Publication que será padre de Question y Answer. Extrajimos las variables de instancia: timestamp, user, votes y descriptions. Y extrajimos los métodos que se mencionan al final de esta página.

Antes:

```
Object subclass: #Question
	instanceVariableNames: 'title answers topics timestamp user votes
description'

Object subclass: #Answer
	instanceVariableNames: 'question timestamp user votes description'
```

Después:

```
Publication subclass: #Question
	instanceVariableNames: 'title answers topics'

Publication subclass: #Answer
	instanceVariableNames: 'question'

Object subclass: #Publication
	instanceVariableNames: 'timestamp user votes description'
```

Los métodos a continuación fueron movidos de las clases Answer y Question a la clase Publication con Pull Up Method:
- addVote:
- description
- description:
- timestamp

- timestamp:
- user
- user:
- votes
- negativeVotes
- positiveVotes

La clase Answer resultante no contiene métodos ni constructores, tan solo conserva una variable de instancia question. Queda una clase anémica que no se borrara debido a que consideramos que el refactoring no debería modificar a grandes rasgos el modelo que se nos ha proporcionado. Eliminar una clase impacta en cómo se maneja el sistema, y cambiar cómo se maneja el mismo no es el objetivo de realizar refactoring.

También encontramos que los initialize tenían un bad smell de duplicate code y utilizamos Pull Up Method para refactorizarlo.
Antes:

```
<Answer>
initialize
     votes := OrderedCollection new.
     timestamp := DateAndTime now.

<Question>
initialize
     answers := OrderedCollection new.
     topics := OrderedCollection new.
     votes := OrderedCollection new.
     timestamp := DateAndTime now.
```

Después:

```
<Publication>
initialize
     votes := OrderedCollection new.
     timestamp := DateAndTime now.

<Answer>
remove method initialize

<Question>
initialize
```

```
    super initialize.
    topics := OrderedCollection new.
    answers := OrderedCollection new.
```

También hemos notado que en los métodos #positiveVotes y #negativeVotes se intenta reinventar la rueda, ya que se usa un #do en vez de un #select o #reject. Por ende estamos presenciando un bad smell de duplicate code. Será necesario aplicar el refactoring de Substitute algorithm.

Antes:

```
positiveVotes
    | r |
    r := OrderedCollection new.
    votes do:[:vote | vote isLike ifTrue:[r add: vote]].
    ^r

negativeVotes
    | r |
    r := OrderedCollection new.
    votes do:[:vote | vote isLike ifFalse:[r add: vote]].
    ^r
```

Después:

```
positiveVotes
    | r |
    r := votes select:[:vote | vote isLike].
    ^r

negativeVotes
    | r |
    r := votes reject:[:vote | vote isLike].
    ^r
```

No solo eso, sino que además se crea una variable temporal innecesaria.
Lo solucionaremos usando el refactoring replace temp with query.

Después:

```
positiveVotes
    ^votes select:[:vote | vote isLike].
```

```
negativeVotes
     ^votes reject:[:vote | vote isLike].
```

Finalizamos de refactorizar las clases Answer y Question corrigiendo los problemas en sus constructores que ocasionamos por agregar una jerarquía.

Antes:

```
<Question>
<CLASE>
newWithTitle: title description: aDescription user: aUser
     ^ self new
          title: title;
          description: aDescription;
          user: aUser;
          yourself.
<CLASE>
newWithTitle: title description: aDescription user: aUser
topic: aTopic
     ^ self new
          title: title;
          description: aDescription;
          addTopic: aTopic;
          user: aUser;
          yourself.

<Answer>
<CLASE>
new: anAnswer user: aUser question: aQuestion
     ^ self new
          description: anAnswer;
          user: aUser;
          yourself
```

Después:

```
<Publication>
<CLASE>
description: aDescription user: aUser
^ self new
          description: aDescription;
```

```
          user: aUser;
          yourself .

<Question>
<CLASE>
newWithTitle: title description: aDescription user: aUser
     ^ (self description: aDescription user: aUser)
          title: title;
          yourself.
<CLASE>
newWithTitle: title description: aDescription user: aUser
topic: aTopic
     ^ (self description: aDescription user: aUser )
          title: title;
          addTopic: aTopic;
          yourself.




<Answer>
<CLASE>
new: anAnswer user: aUser
     ^ self new
          description: anAnswer;
          user: aUser;
          yourself .
```

El constructor de Answer y Publication eran identicos, asi que estamos presenciando nuevamente duplicate code, por lo tanto decidimos aplicar el refactoring substitute algorithm, y remover el constructor de Answer.

```
<Answer>
<CLASE>
new: anAnswer user: aUser
     ^ self description: anAnswer user: aUser;
          yourself
```

Debido a este último refactoring tuvimos que modificar el test de Answer, en el set up .

Antes:

```
setUp
     answerWithoutVotes := Answer new:'An answer'  user:(User
new).
```

Después:

```
setUp
     answerWithoutVotes := Answer description: 'An answer'
user:(User new).
```

Notar que en el constructor de Publication se están enviando dos mensajes para setear las variables de instancia de la clase. Si bien no se presencia un bad smell, creemos conveniente crear un nuevo método que se encargue de realizar el seteo de todas las variables de Publication en un solo mensaje debido a las heurísticas aprendidas.

Después:

```
<Publication>
<CLASE>
description: aDescription user: aUser
^ new
          description: aDescription user: aUser;
          yourself .
<INSTANCIA>
description: aDescription user: aUser
     description:= aDescription.
     user:= aUser.
```

En los siguientes métodos de la clase Publication reciben un parámetro con un nombre que no hacer referencia a lo que se está guardando , debido a las heurísticas aprendidas en la materia se le cambió el nombre por uno que represente al parámetro solicitado.

Antes:

```
description: anObject
     description := anObject
```

Después:

```
description: aDescription
     description := aDescription
```

Antes:

```
timestamp: anObject
     timestamp := anObject
```
Después:
```
timestamp: aTimestamp
     timestamp := aTimestamp
```
Antes:
```
user: anObject
     user := anObject
```
Después:
```
user: aUser
     user := aUser
```

Vote

En los siguientes métodos de la clase Vote se reciben parámetros con un nombre que no hacen referencia a lo que se está guardando , debido a las heurísticas aprendidas en la materia se le cambió el nombre por uno que represente al parámetro solicitado.

Antes:
```
publication: anObject
     publication := anObject
```
Después:
```
publication: aPublication
     publication := aPublication
```
Antes:
```
user: anObject
     user := anObject
```
Después:
```
user: aUser
     user := aUser
```

Se detecta que se utilizan 2 variables que cumplen la misma función, timstamp y timestamp, pudo haberse generado por un mal tipeo. Se cambia la variable mal escrita timstamp por la variable timestamp.

Resultado:

```
intialize: aUser value: aBoolean
     user:=aUser.
     isLike:= aBoolean.
     timestamp := DateAndTime now.
```

```
Object subclass: #Vote
     instanceVariableNames: 'isLike publication ~~timstamp~~ user
timestamp'
```

Esto desencadena duplicate code en el mensaje #intialize:value, ya que la clase Vote en #initialize setea el timestamp en DateAndTime now. Por esa razón se realiza el refactoring de substitute algorithm. Dejando solo el seteo de DateAndTime now en el #initialize.

Antes:

```
intialize: aUser value: aBoolean
     user:=aUser.
     isLike:= aBoolean.
     ~~timestamp := DateAndTime now.~~
```

Después:

```
intialize: aUser value: aBoolean
     user:=aUser.
     isLike:= aBoolean.
```

Existe código duplicado en los constructores user:dislikesPublication: y user:likesPublication:, ya que en #positive y #user:likesPublication hacen lo mismo solamente que en publication le asigna la publicacion; asi mismo, en #negative y #user:dislikesPublication: se comporta de manera similar. Se utilizará refactoring de substitute algorithm en #user:likesPublication: y #user:dislikesPublication:.

Antes:

```
user: aUser dislikesPublication: aPublication
     ^ self new
          user: aUser;
          publication: aPublication;
          dislike;
          yourself
```

Después:

```
user: aUser dislikesPublication: aPublication
     ^ (self negative: aUser) publication: aPublication;
          yourself
```

Antes:

```
user: aUser likesPublication: aPublication
     ^ self new
          user: aUser;
          publication: aPublication;
          yourself
```

Después:

```
user: aUser likesPublication: aPublication
     ^ (self positive: aUser) publication: aPublication;
          yourself
```

Topic

En los siguientes métodos de la clase Topic se reciben parámetros con un nombre que no hacen referencia a lo que se está guardando , debido a las heurísticas aprendidas en la materia se le cambió el nombre por uno que represente al parámetro solicitado.

Antes:

```
description: anObject
     description := anObject
```

Después:

```
description: aDescription
     description := aDescription
```

Antes:

```
name: anObject
     name := anObject
```

Después:

```
name: aName
     name := aName
```

Antes: (nombre mal escrito)

```
addQuestion: aQuetion
     questions add: aQuetion
```

Después:

```
addQuestion: aQuestion
     questions add: aQuestion
```

Si bien no se presencia un bad smell en los setters separados de #description: y #name:, creemos conveniente crear un nuevo método que se encargue de realizar el seteo de todas las variables de Topic en un solo mensaje debido a las heurísticas aprendidas.

Antes:

```
<CLASE>
name: aName description: aDescription
     ^ self new
          name: aName;
          description: aDescription;
          yourself
```

Después:

```
<INSTANCIA>
name: aName description: aDescription
     name := aName.
     description := aDescription.
```

```
<CLASE>
name: aName description: aDescription
     ^ self new
          name: aName description: aDescription;
          yourself
```

CuOOra

Hay duplicate code en los mensajes #addQuestion: y #addQuestion:forUser: , en ambos se agrega una pregunta a la collection questions.
Se resuelve con el refactoring Substitute Algorithm.

Antes:

```
<CuOOra>
addQuestion: aQuestion
     questions add: aQuestion

addQuestion: aQuestion forUser: aUser
     aUser addQuestion: aQuestion.
     questions add: aQuestion.
```

Después:

```
addQuestion: aQuestion forUser: aUser
     aUser addQuestion: aQuestion.
     self addQuestion: aQuestion.
```

User

En los siguientes métodos de la clase User se recibe un parámetro con un nombre que no hacen referencia a lo que se está guardando, debido a las heurísticas aprendidas en la materia se le cambió el nombre por uno que represente al parámetro solicitado.

Antes:

```
username: anObject
     username := anObject

password: anObject
     password := anObject
```

Después:

```
username: aUsername
     username := aUsername

password: aPassword
     password := aPassword
```

Si bien no se presencia un bad smell en los setters separados de #username: y #password:, creemos conveniente crear un nuevo método que se encargue de realizar el seteo de todas las variables de User en un solo mensaje debido a las heurísticas aprendidas.

Antes:

```
username: aUsername password: aPassword
     ^ self new
          username: aUsername;
          password: aPassword;
          defineInterestInSocial;
          yourself
```

Después:

```
username: aUsername password: aPassword
     username := aUsername.
     password := aPassword
```

```
username: aUsername password: aPassword
     ^ self new
          username: aUsername password: aPassword;
          defineInterestInSocial;
          yourself
```

Options se utiliza en #questionsOfInterest en un switch generando long method y switch statement; aplicaremos el refactoring de Replace Type Code with Strategy. Generamos una jerarquía QuestionsOfInterest con 3 subclases: QuestionsOfInterestSocial, QuestionsOfInterestTopics y QuestionsOfInterestNegativeTopics.
La variable de instancia options pasará a guardar una instancia de alguna subclase de QuestionsOfInterest. Se usará el refactoring de extract y move method en los distintos bloques del switch hacia las distintas subclases de QuestionsOfInterest según corresponda.

Antes:

```
questionsOfInterest
     | temp1 temp2 temp3 temp4 |
     temp1 := OrderedCollection new.
     option = #social ifTrue:[
               temp3 := OrderedCollection new.
               self following do:[ :follow | temp3 addAll:
follow questions; addAll: follow questionsOfInterest ].
               temp3:= temp3 reject:[:q | q user = self].
               temp2 := temp3 asSortedCollection:[ :a :b | a
positiveVotes size > b positiveVotes size ].
               temp1 := temp2 first: (10 min: temp2 size).
          ].
     option = #topics ifTrue:[
               temp4 := OrderedCollection new.
               self topics do:[ :topic | temp4 addAll: topic
questions ].
               temp4 := temp4 select: [  :q | (DateAndTime now
- q timestamp) asDays < 90 ].
               temp4 := temp4 reject:[:q | q user = self].
               temp2 := temp4 asSortedCollection:[ :a :b | a
positiveVotes size > b positiveVotes size ].
               temp1 := temp2 last: (10 min: temp2 size).
          ].
     option = #negativetopics ifTrue:[
               temp4 := OrderedCollection new.
               self topics do:[ :topic | temp4 addAll: topic
questions ].
               temp4 := temp4 reject:[:q | q user = self].
               temp2 := temp4 asSortedCollection:[ :a :b | a
negativeVotes  size > b negativeVotes size ].
               temp1 := temp2 last: (10 min: temp2 size).
          ].

     ^temp1
```

Después:

```
<User>
questionsOfInterest
     ^option questionsOfInterestOf: self.
```

```
<QuestionsOfInterestSocial>
questionsOfInterestOf: aUser
     | temp3 temp2 temp1 |
     temp3 := OrderedCollection new.
     aUser following
          do: [ :follow |
               temp3
                    addAll: follow questions;
                    addAll: follow questionsOfInterest ].
     temp3 := temp3 reject: [ :q | q user = aUser].
     temp2 := temp3
          asSortedCollection: [ :a :b | a positiveVotes size >
b positiveVotes size ].
     temp1 := temp2 first: (10 min: temp2 size).
     ^temp1.

<QuestionsOfInterestTopics>
questionsOfInterestOf: aUser
     questionsOfInterestOf: aUser
     | temp4 temp2 temp1 |
     temp4 := OrderedCollection new.
     aUser topics do: [ :topic | temp4 addAll: topic questions
].
     temp4 := temp4
          select: [ :q | (DateAndTime now - q timestamp)
asDays < 90 ].
     temp4 := temp4 reject: [ :q | q user = aUser].
     temp2 := temp4
          asSortedCollection: [ :a :b | a positiveVotes size >
b positiveVotes size ].
     temp1 := temp2 last: (10 min: temp2 size).
     ^ temp1

<QuestionsOfInterestNegativeTopics>
questionsOfInterestOf: aUser
     | temp4 temp2 temp1 |
     temp4 := OrderedCollection new.
     aUser topics do: [ :topic | temp4 addAll: topic questions
].
     temp4 := temp4 reject: [ :q | q user = aUser ].
     temp2 := temp4
```

```
          asSortedCollection: [ :a :b | a negativeVotes size >
b negativeVotes size ].
     temp1 := temp2 last: (10 min: temp2 size).
     ^ temp1
```

Se modificó los mensajes #defineInterestInNegativeTopics ,#defineInterestInSocial y #defineInterestInTopics para que asignen a la variable option la subclase de QuestionsOfInterest correspondiente.

Resultado:

```
<User>
defineInterestInNegativeTopics
     option := QuestionsOfInterestNegativeTopics new.

defineInterestInSocial
     option := QuestionsOfInterestSocial new.

defineInterestInTopics
     option := QuestionsOfInterestTopics new.
```

Existe envidia de atributos en QuestionsOfInterestSocial porque se manipula la colección de followers del usuario. Esto se resuelve aplicando extract y move method hacia la clase User. Además existe código duplicado ya que las tres subclases realizan el mismo reject cuando se trata del mismo usuario, por consiguiente, se extraerá y moverá dicho método hacia User.

```
<User>
removeMyQuestions: aCollection
^aCollection reject: [ :q | q user = self ].

questionsOfFollowing
     | temp |
     temp := OrderedCollection new.
     self following
          do: [ :follow |
               temp
                    addAll: follow questions;
                    addAll: follow questionsOfInterest ].
     ^ temp
```

Existe duplicate code en #questionsOfFollowers, ya que con un #flatCollect: y #union: nos ahorrariamos el do y los addAll, y con ellos la variable temporal (reinventar la rueda). Incluso se podría considerar como un long method. Se aplica substitute algorithm.

Resultado:

```
<User>
questionsOfFollowers
    ^self following flatCollect: [ :follow | follow questions
union: follow questionsOfInterest; yourself]  .
```

```
<QuestionsOfInterestSocial>
questionsOfInterestOf: aUser
     | temp3 temp2 temp1 |
     temp3 := aUser questionsOfFollowers.
     temp3 := aUser removeMyQuestions: temp3.
     temp2 := temp3
          asSortedCollection: [ :a :b | a positiveVotes size >
b positiveVotes size ].
     temp1 := temp2 first: (10 min: temp2 size).

^temp1
```

QuestionsOfInterestTopics y QuestionOfInterestNegativeTopics se realiza envidia de atributos de la misma forma que QuestionsOfInterestSocial pero con la colección de tópicos que posee User. Por consiguiente se realiza move y extract method hacia User. Y de paso se reemplaza el código repetido que queda en el reject de la colección por el método nuevo que creamos en User removeMyQuestions.

Resultado:

```
<QuestionsOfInterestNegativeTopics>
questionsOfInterestOf: aUser
     | temp4 temp2 temp1 |
     temp4 := aUser questionsOfTopics.
     temp4 := aUser removeMyQuestions: temp4.
     temp2 := temp4
          asSortedCollection: [ :a :b | a negativeVotes size >
b negativeVotes size ].
     temp1 := temp2 last: (10 min: temp2 size).
     ^ temp1
```

```
<QuestionsOfInterestTopics>
questionsOfInterestOf: aUser
     | temp4 temp2 temp1 |
     temp4 := aUser questionsOfTopics.
     temp4 := temp4
          select: [ :q | (DateAndTime now - q timestamp)
asDays < 90 ].
     temp4 := aUser removeMyQuestions: temp4.
     temp2 := temp4
          asSortedCollection: [ :a :b | a positiveVotes size >
b positiveVotes size ].
     temp1 := temp2 last: (10 min: temp2 size).
     ^ temp1
```

De la misma forma en el método de User #questionsOfTopic se produce duplicate code, y lo solucionamos con substitute algorithm. Reemplazamos el mensaje de las colecciones #do: y #addAll:, por flatCollect.

Antes:

```
<User>
questionsOfTopics
     | temp |
     temp := OrderedCollection new.
     self topics do: [ :topic | temp addAll: topic questions].
     ^ temp
```

Después:

```
<User>
questionsOfTopics
   ^self topics flatCollect: [ :topic | topic questions].
```

Se presencia código repetido en los mensajes #questionsOfInterestOf de las subclases de QuestionsOfInterest, la mejor forma de solucionar esto es con un Form Template Method. Vemos como los algoritmos para obtener las preguntas de interés siguen cierta lógica. Primero agarran una colección, ya se filtrando por tópico o por followers, luego en NegativeTopics se filtran las preguntas que tienen menos de 90 días, en los demás este paso se omite, pero en todos los QuestionsOfInterest, se filtran las preguntas que son del mismo usuario. Por último se prosigue a ordenar la colección dependiendo de alguna propiedad y finaliza tomando las primeras o últimas diez preguntas dependiendo de qué estrategia se esté utilizando. Se realiza

extract y pull up method en los casos base, y en los específicos se sobreescribirá en la estrategia donde se utilice.
El template method termina quedando en los siguientes pasos:

+ Obtener las preguntas
+ Eliminar las preguntas del usuario
+ Ordenar Preguntas
+ Tomar 10 preguntas contiguas

Después:

```
<QuestionsOfInterest>
questionsOfInterestOf: aUser
     ^ self getTenQuestions: (self orderQuestions: (aUser
removeMyQuestions: (self getQuestions: aUser)))

orderQuestions: aQuestions
^aQuestions asSortedCollection: [ :a :b | a positiveVotes size
> b positiveVotes size ].

getTenQuestions: aQuestions
^ aQuestions last: (10 min: aQuestions size).

getQuestions: aUser
^self subclassResponsibility .
```

```
<QuestionsOfInterestSocial>
getQuestions: aUser
     ^aUser questionsOfFollowing.

getTenQuestions: aQuestions
^ aQuestions first: (10 min: aQuestions size).
questionsOfInterestOf: aUser
<remove method>
```

```
<QuestionsOfInterestNegativeTopics>
getQuestions: aUser
     ^aUser questionsOfTopics.

orderQuestions: aQuestions
^aQuestions asSortedCollection: [ :a :b | a negativeVotes size
> b negativeVotes size ].

questionsOfInterestOf: aUser
<remove method>
```

```
<QuestionsOfInterestTopics>
getQuestions: aUser
     ^aUser questionsOfTopics select: [ :q | (DateAndTime now
- q timestamp) asDays < 90 ].

questionsOfInterestOf: aUser
<remove method>
```

En #getQuestions: de QuestionsOfInterestTopics se produce envidia de atributos, ya que, se manipula una colección que pertenece a User. Se decidió realizar un extract y move method hacia User, recibiendo como parámetro la cantidad de días máximos que pasaron desde que se emitió la pregunta para permitir la reutilización del código.

Resultado:

```
<QuestionsOfInterestTopics>
getQuestions: aUser
     ^aUser questionsOfTopicsFromLastDays: 90
```

```
<User>
questionsOfTopicsFromLastDays: aDays
     ^self questionsOfTopics select: [ :q | (DateAndTime now -
q timestamp) asDays < aDays ]
```

Por último, existe envidia de atributos cuando se ordena por votos positivos o negativos, ya que se cuenta la colección de votos que pertenece a las publicaciones. Por lo tanto se decidió realizar extract y move method hacia Publication.

Después:

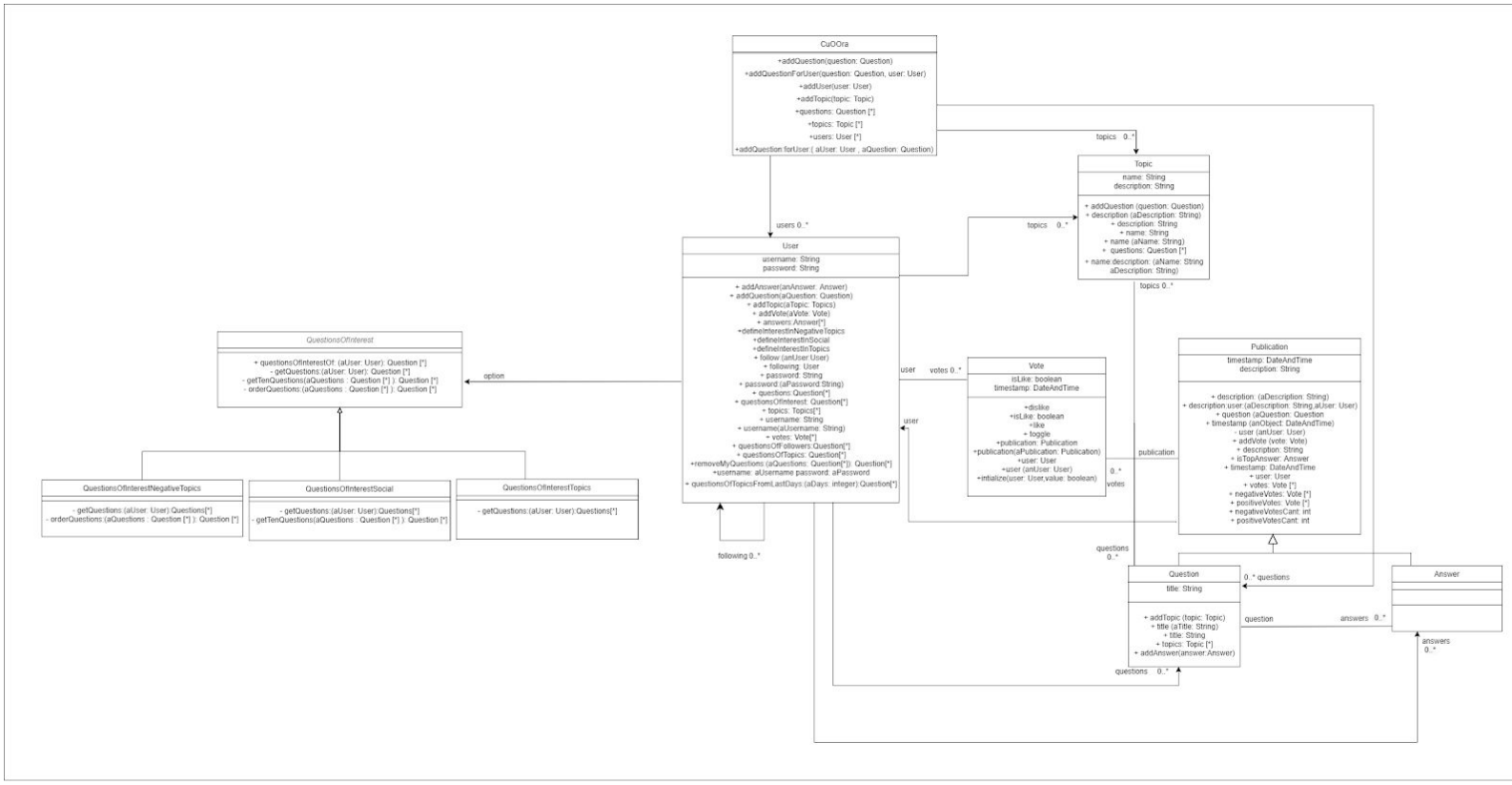
```
<Publication>
negativeVotesCant
^self negativeVotes size.

positiveVotesCant
^self positiveVotes size.
```

```
<QuestionsOfInterestNegativeTopics>
orderQuestions: aQuestions
^aQuestions asSortedCollection: [ :a :b | a negativeVotesCant
> b negativeVotesCant ].
```

```
<QuestionsOfInterest>
orderQuestions: aQuestions
^aQuestions asSortedCollection: [ :a :b | a positiveVotesCant
> b positiveVotesCant ].
```

UML



Drive: https://drive.google.com/file/d/11D6oJXU2XwhHzR3WQwYoNcnb8MVwI4rf/view?usp=sharing
Imagen: https://drive.google.com/file/d/1nVNR03N9ZJnrmqp_6hSQyh5pSUW91urX/view?usp=sharing