PREVIOUSLY ON ASD…

- Domain Modelling:
    - Used to represent concepts from an area / domain
    - Show relations
    - Omits software elements, e.g. Database storage entity

ADVANCED SOFTWARE DESIGN
LECTURE 3
STRUCTURAL MODELLING

Kiko Fernandez-Reyes

OBJECTIVES

Use UML diagrams for expressing **static OO* design**

Identify key responsibilities in system

Assign responsibilities to classes

Generalise classes

Apply ideas to Open Source Taiga software

OO* = Object-Oriented

INDEX

- Structural Modelling (class diagrams)
    - Arrows
    - Classes, methods and attributes
    - Mapping to code
- Guidelines for producing class diagrams

STRUCTURAL MODELLING

- Make **domain model more concrete — class model**
- Assign **responsibilities** — fill in operations — **methods**
- Introduce **generalisation/inheritance** — **reuse**
- Model software artefacts

CLASS GUIDELINES

Write the name of a *class*, the *responsibilities* of that class, and the *collaborators*, other classes that will help carry out each responsibility.

Responsibilities – high level purpose of class, may be realised by many methods

Guideline: each class should not have more than three or four responsibilities – ***high cohesion*** *(good)*

Guideline: generally class should have only a few collaborators – ***low coupling*** *(good)*

MORE "CLASSES"

Interfaces

Abstract classes

Parameterised classes

} As in Java

LEARN BY DOING

- Simple description of Taiga Scrum board (slides)

LEARN BY DOING

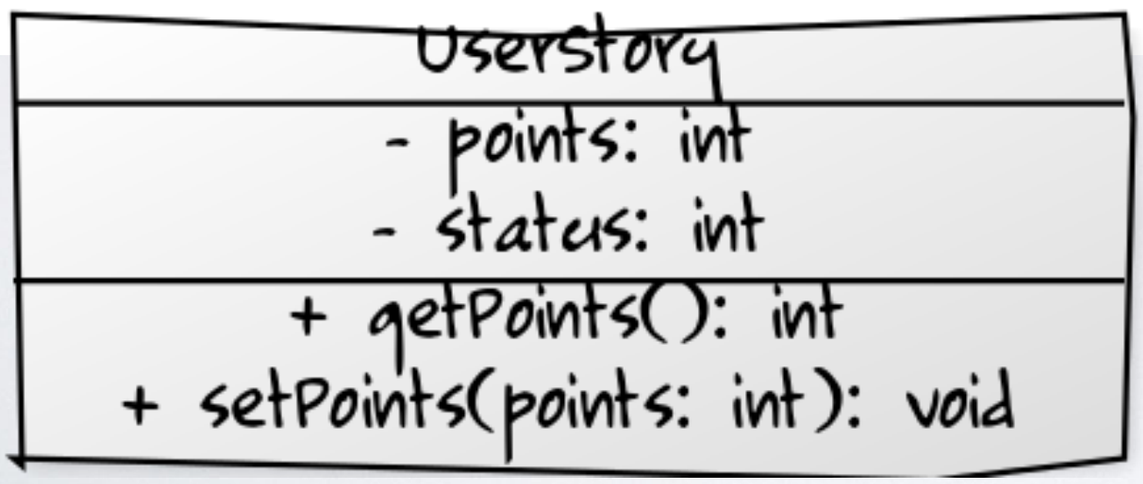
- Simple description of Taiga Scrum board (slides)

LEARN BY DOING

- Simple description of Taiga Scrum board (slides)
    - Scrum board
    - User stories
    - Tasks for each user story
    - Burndown chart

USER STORIES

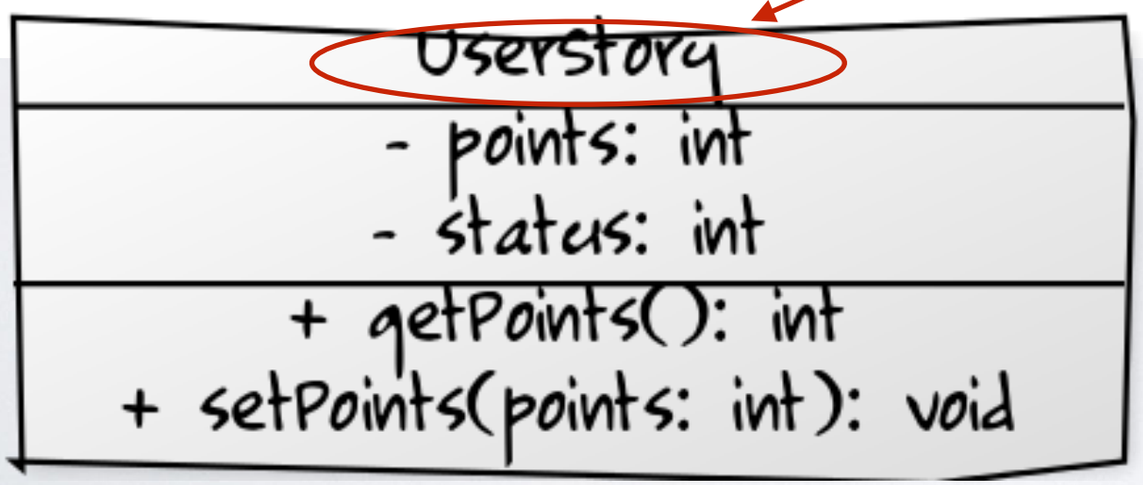
```
public class UserStory {
  private int points;
  private int status;

  public void setPoints(points: int){
      this.points = points;
  }

  public int getPoints(){
     return this.points;
  }
}
```



USER STORIES

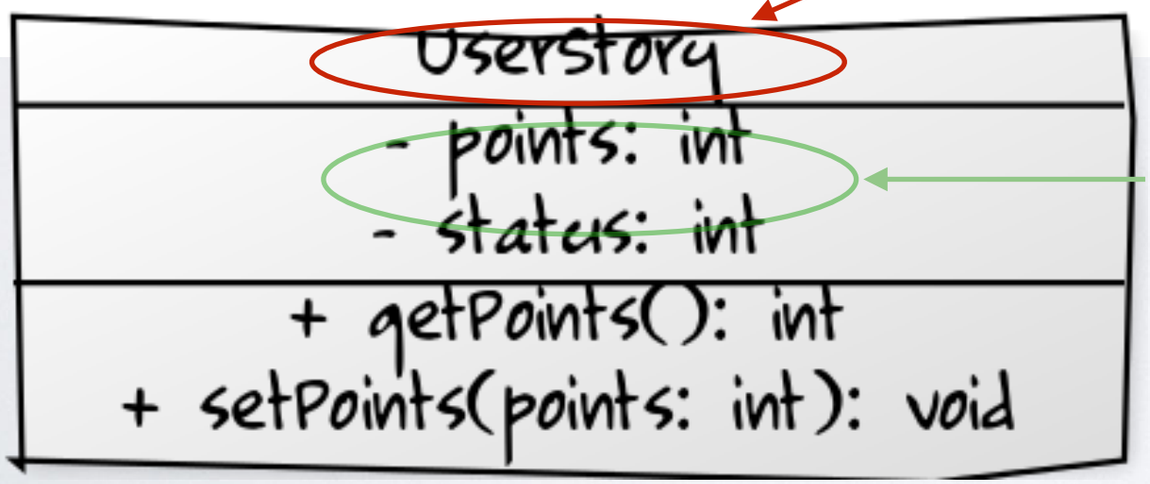
```
public class UserStory {
  private int points;
  private int status;

  public void setPoints(points: int){
      this.points = points;
  }

  public int getPoints(){
     return this.points;
  }
}
```

class name



USER STORIES

```
public class UserStory {
  private int points;
  private int status;

  public void setPoints(points: int){
      this.points = points;
  }

  public int getPoints(){
     return this.points;
  }
}
```

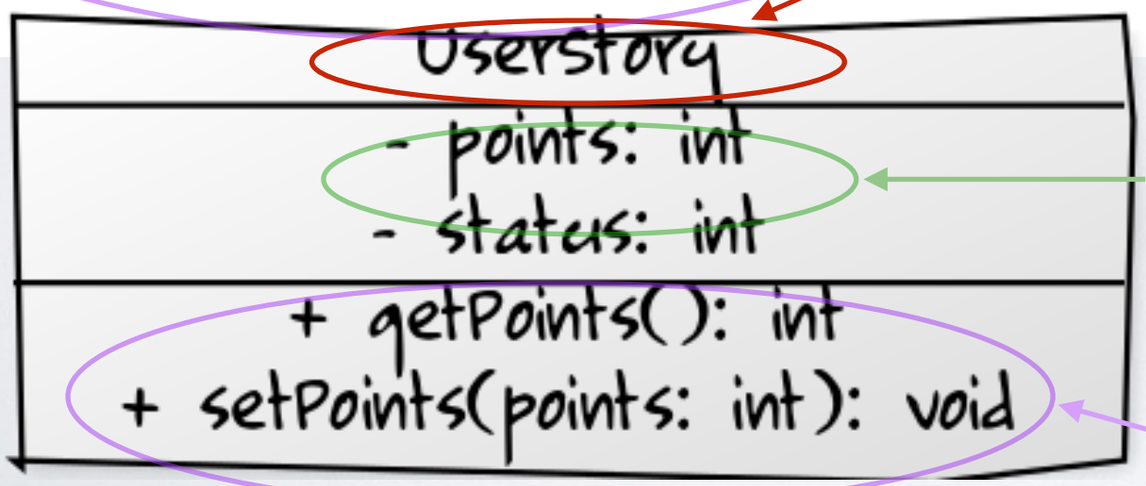


class name

attributes of the class:
<name : Type>

USER STORIES

```
public class UserStory {
  private int points;
  private int status;

  public void setPoints(points: int){
      this.points = points;
  }

  public int getPoints(){
     return this.points;
  }
}
```



class name

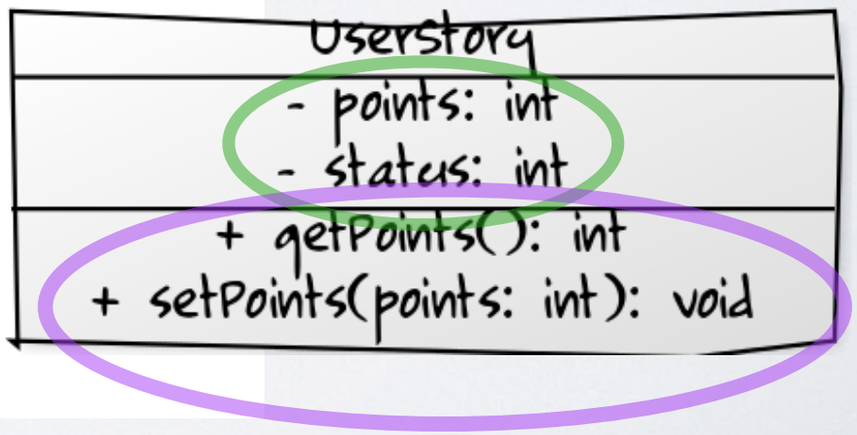
attributes of the class:
<name : Type>

methods and arguments

VISIBILITY

Private methods/attr. are denoted by **-**
Public methods/attr. are denoted by **+**

```
public class UserStory {
  private int points;
  private int status;

  public void setPoints(points: int){
      this.points = points;
  }

  public int getPoints(){
     return this.points;
  }
}
```

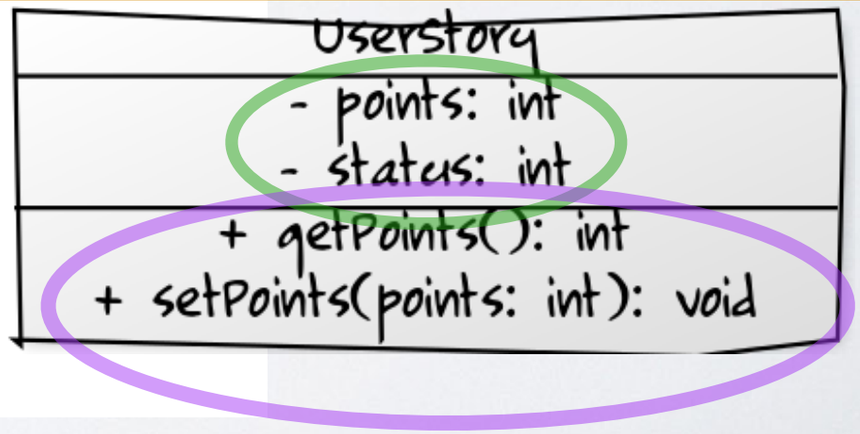


VISIBILITY

Private methods/attr. are denoted by **-**
Public methods/attr. are denoted by **+**

```
public class UserSt
  private int point
  private int statu

  public void setPo
      this.points =
  }

  public int getPoints(){
     return this.points;
  }
}
```

When would you choose public visibility and when would you use private?

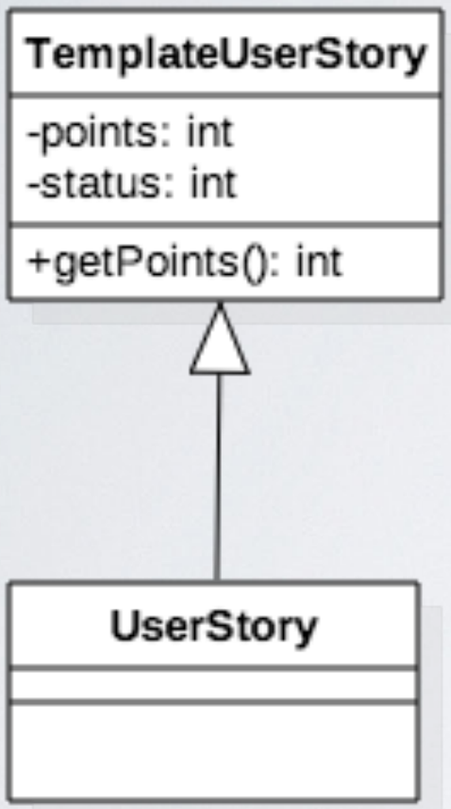


RELATIONSHIPS

IT'S COMPLICATED!

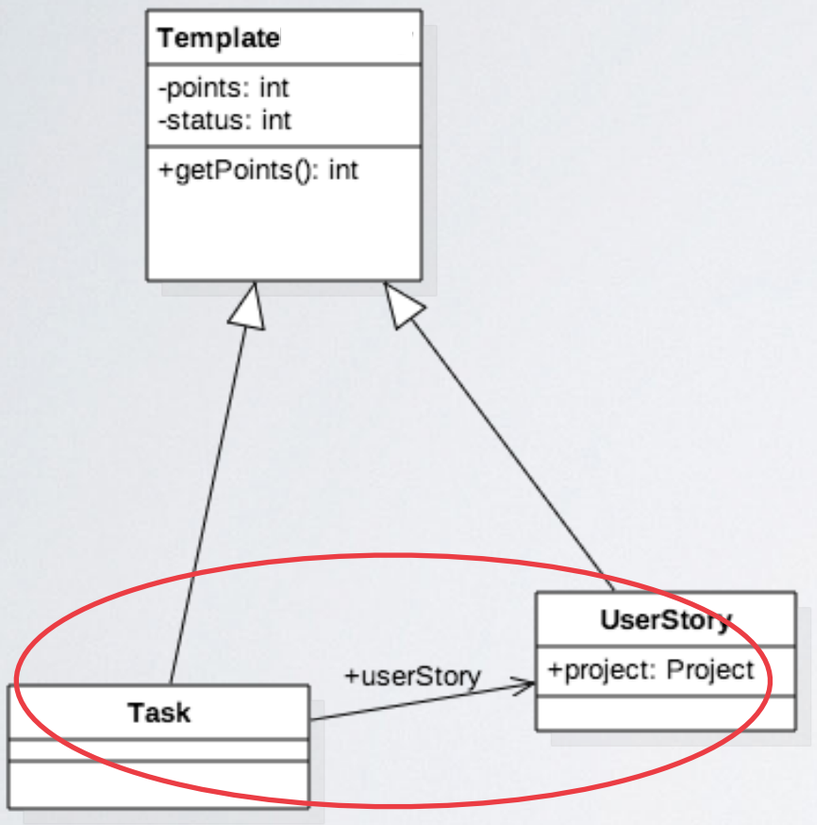
- Generalisation (inheritance)
- Association
- Composition and Aggregation
- Dependency

GENERALISATION



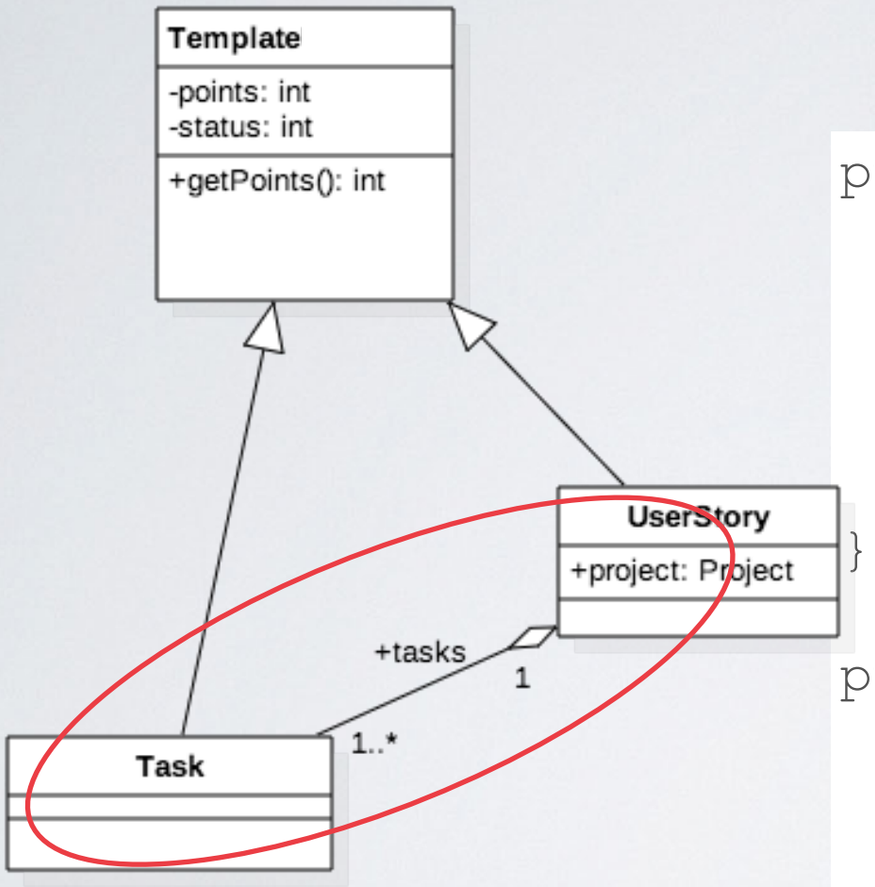

```
public class TemplateUserStory {
  private int points;
  private int status;

  public void setPoints(points: int){
      this.points = points;
  }

  public int getPoints(){
     return this.points;
  }
}

public class UserStory
                 extends TemplateUserStory {
   …
}
```

ASSOCIATION




```
public class Task extends TemplateUserStory{
  public UserStory userStory;

  public Task(userStory: UserStory){
     …
  }
}

public class TemplateUserStory {
  private int points;
  private int status;

  …
}
```

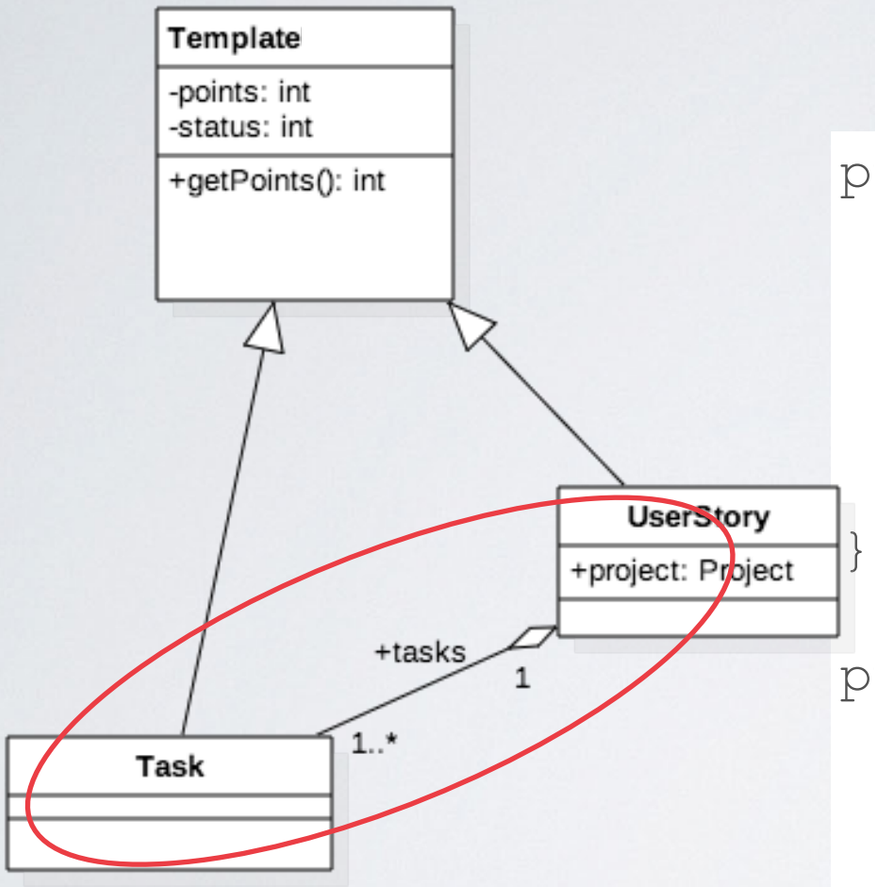
AGGREGATION




```
public class UserStory extends TemplateUserStory {
  public List<Task> tasks;

  public UserStory(tasks: List<Task>){
    this.tasks = tasks;
  }
}
```

```
public class TemplateUserStory {
  private int points;
  private int status;

  …
}
```

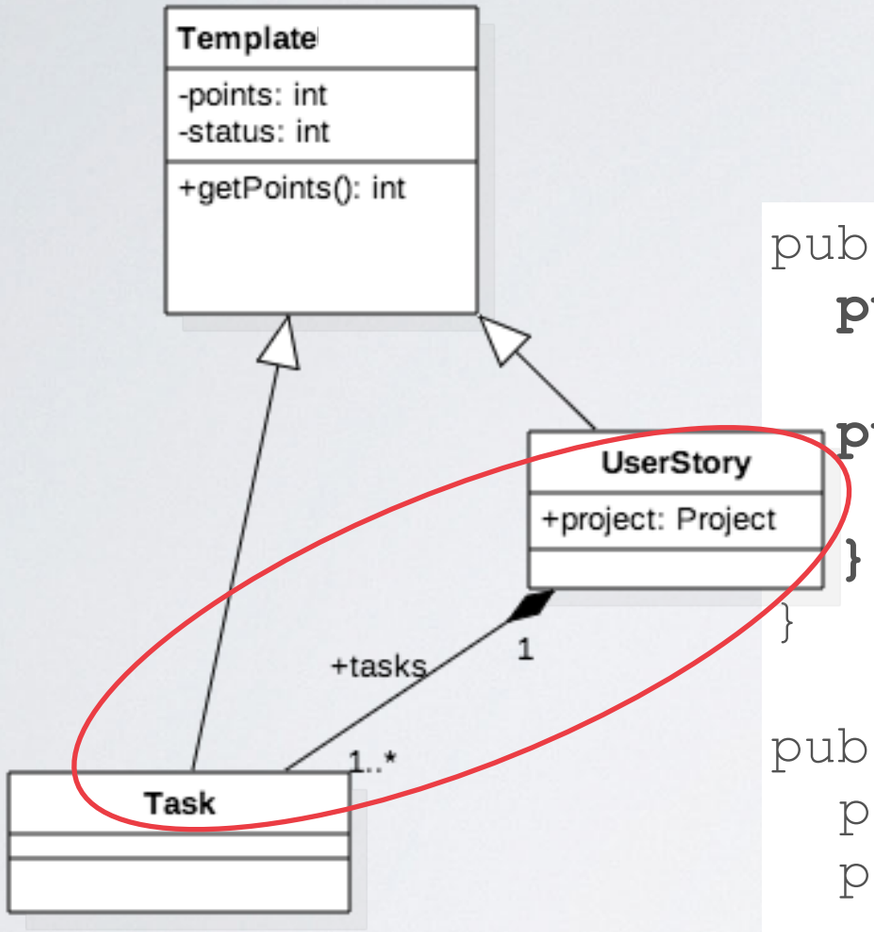
AGGREGATION




```
public class UserStory extends TemplateUserStory {
  public List<Task> tasks;

  public UserStory(tasks: List<Task>){
    this.tasks = tasks;
  }
}

public class TemplateUserStory {
  private int points;
  private int status;

  …
}
```

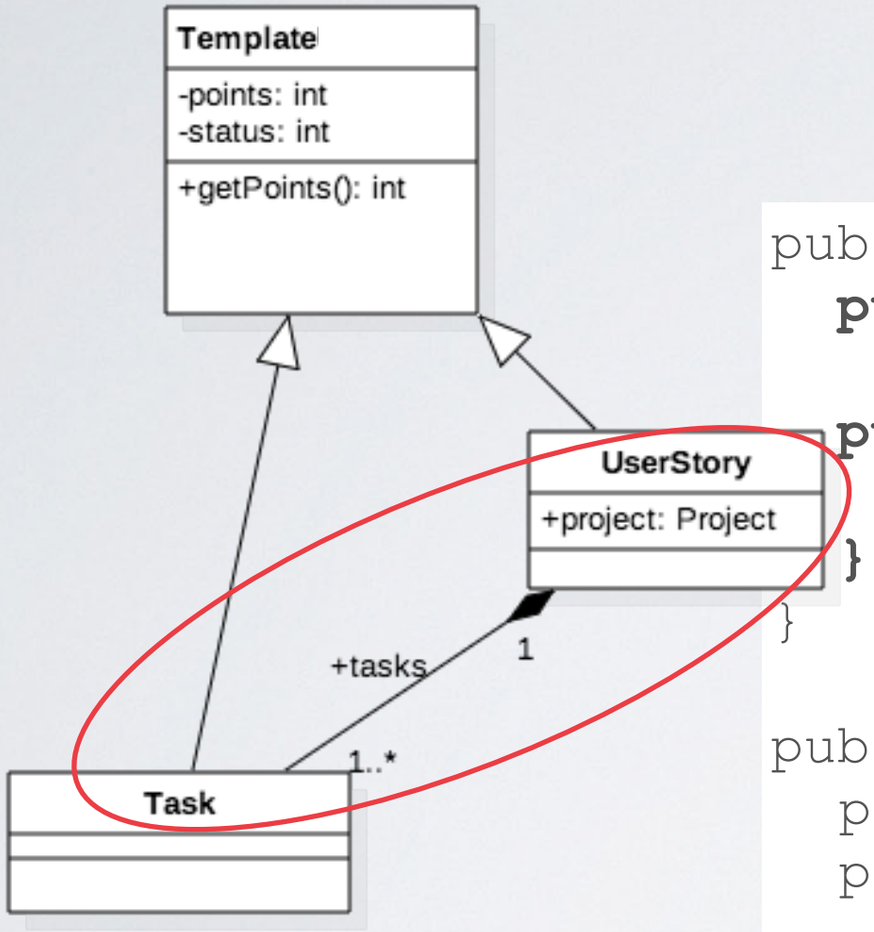
Notion of whole-part

COMPOSITION




```
public class UserStory extends TemplateUserStory {
  public final List<Task> tasks;

  public UserStory(){
    this.tasks = new List<Task>();
  }
}

public class TemplateUserStory {
  private int points;
  private int status;

  …
}
```

COMPOSITION




```
public class UserStory extends TemplateUserStory {
  public final List<Task> tasks;

  public UserStory(){
    this.tasks = new List<Task>();
  }
}

public class TemplateUserStory {
  private int points;
  private int status;

  …
}
```

Notion of whole-part AND lifetime of objects

AGGREGATION

Conceptual notion of whole-part: one object is a part of another.

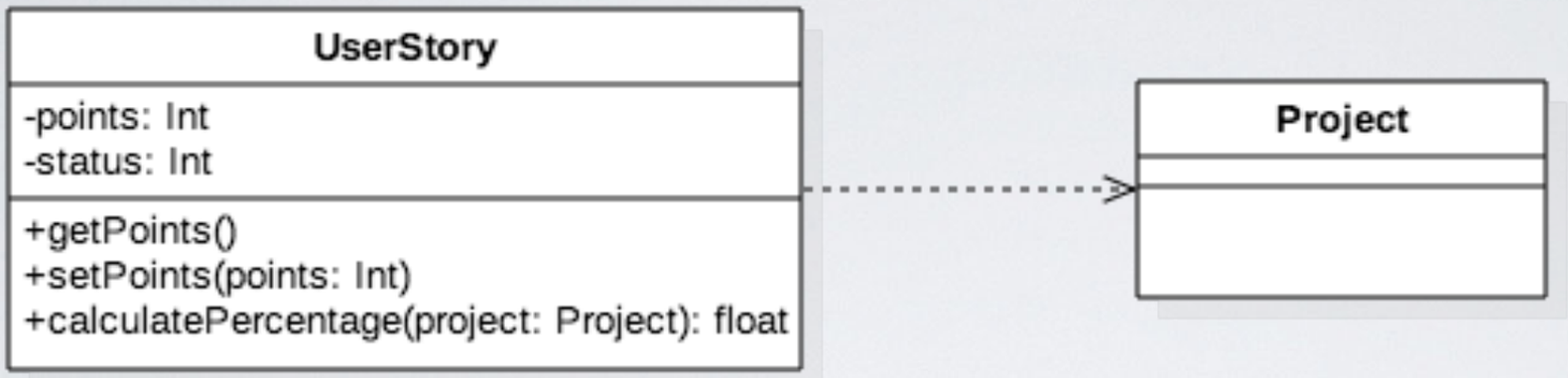
COMPOSITION

Refinement of aggregation: the whole *strongly owns* the part.

If whole is deleted, then so are all the parts.

Don't overuse.

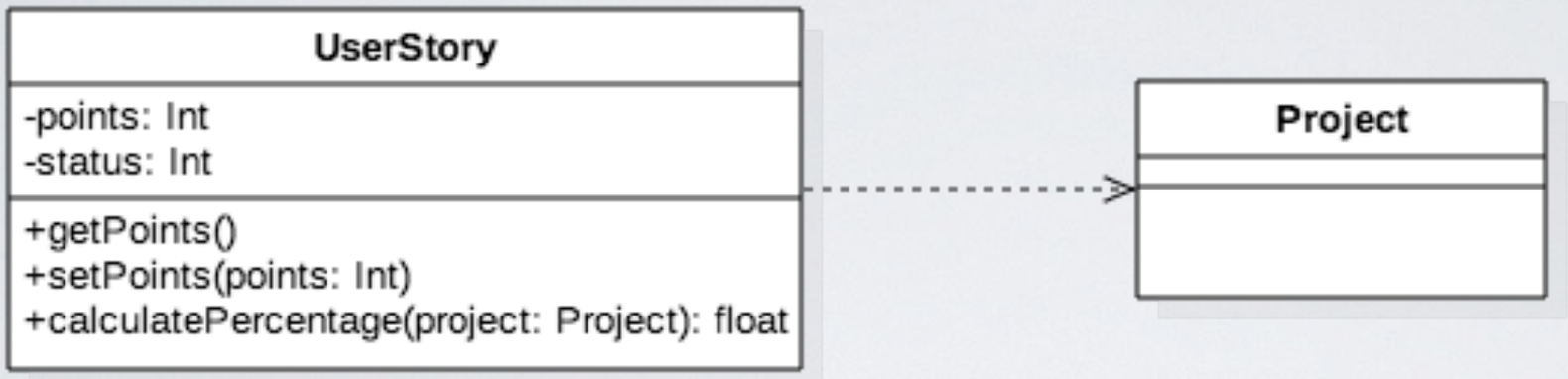
DEPENDENCY




```
public class Project { … }
```

```
public class UserStory{

  private int points;
  private int status;

  public float calculatePercentage(Project p):{
     return this.points /p.getCurrentSprint()
  }
}
```

DEPENDENCY



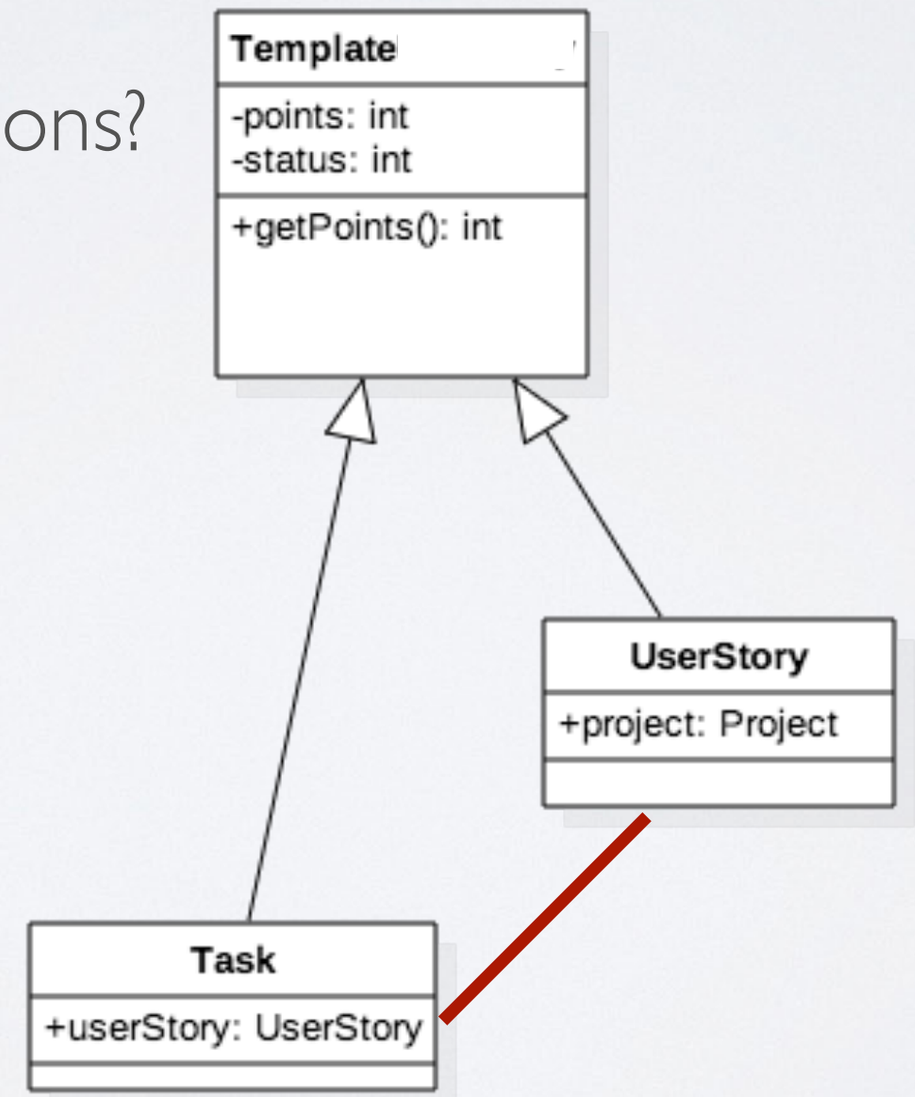

```
public class Project { … }

public class UserStory{

  private int points;
  private int status;

  public float calculatePercentage(Project p):{
     return this.points /p.getCurrentSprint()
  }
}
```

UserStory does NOT have an attribute of class Project!

RELATIONSHIPS:

What kind of relationship is valid from the **`userStory`** attribute in `Task` to `UserStory` class?

What are the implications?



Association

or

Dependency

or

Composition

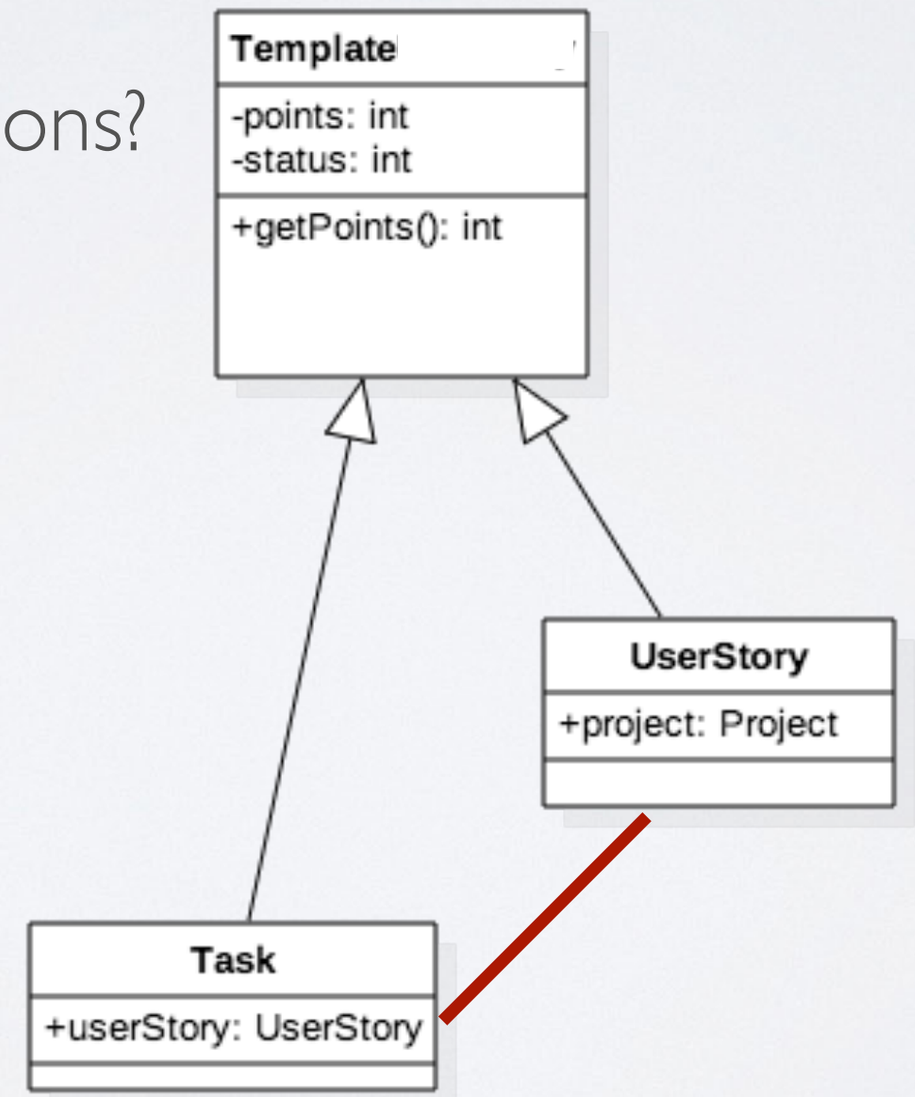
or

Aggregation

RELATIONSHIPS:

What kind of relationship is valid from the **`userStory`** attribute in `Task` to `UserStory` class?

What are the implications?



Association

or

Dependency

or

Composition

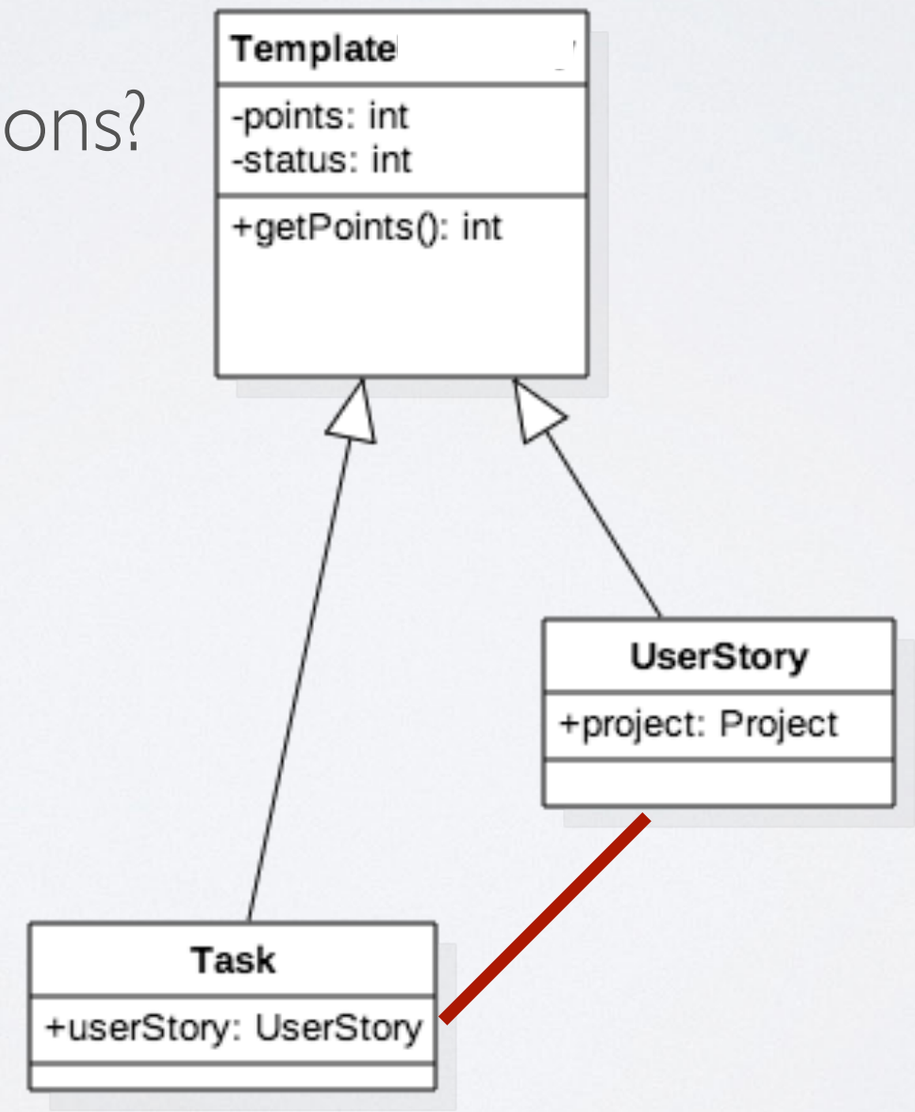
or

Aggregation

RELATIONSHIPS:

What kind of relationship is valid from the **`userStory`** attribute in `Task` to `UserStory` class?

What are the implications?



Association

or

~~Dependency~~

or

Composition

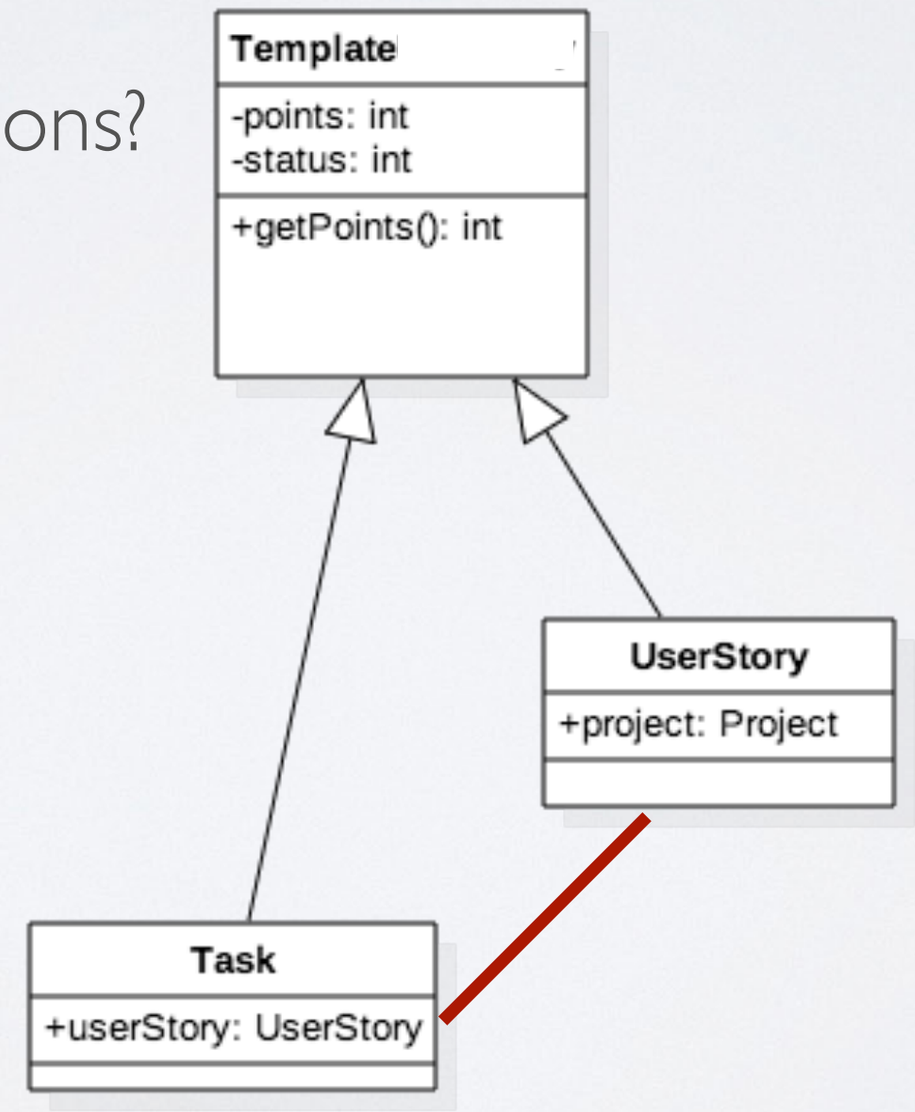
or

Aggregation

RELATIONSHIPS:

What kind of relationship is valid from the **`userStory`** attribute in `Task` to `UserStory` class?

What are the implications?



Association

or

~~Dependency~~

or

~~Composition~~

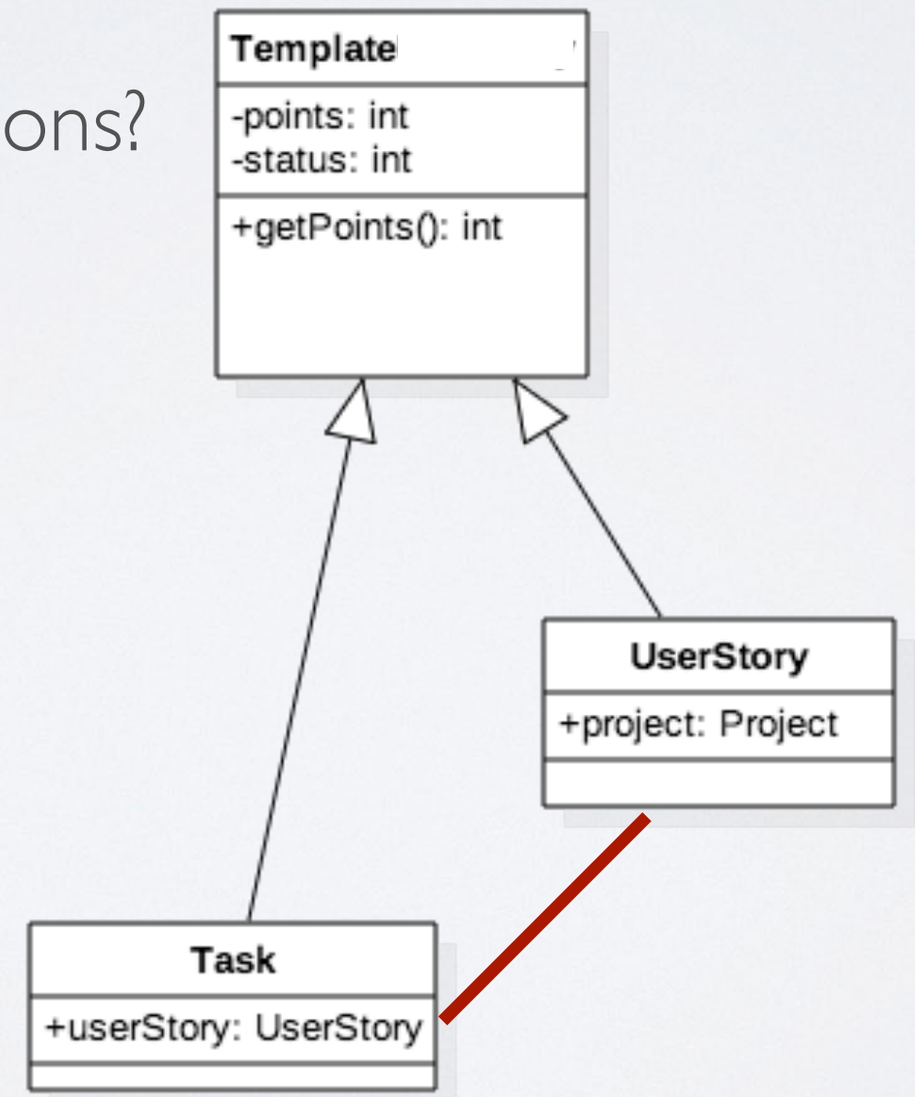
or

Aggregation

RELATIONSHIPS:

What kind of relationship is valid from the **`userStory`** attribute in `Task` to `UserStory` class?

What are the implications?



Association

or

~~Dependency~~

or

~~Composition~~

or

~~Aggregation~~

METHODS

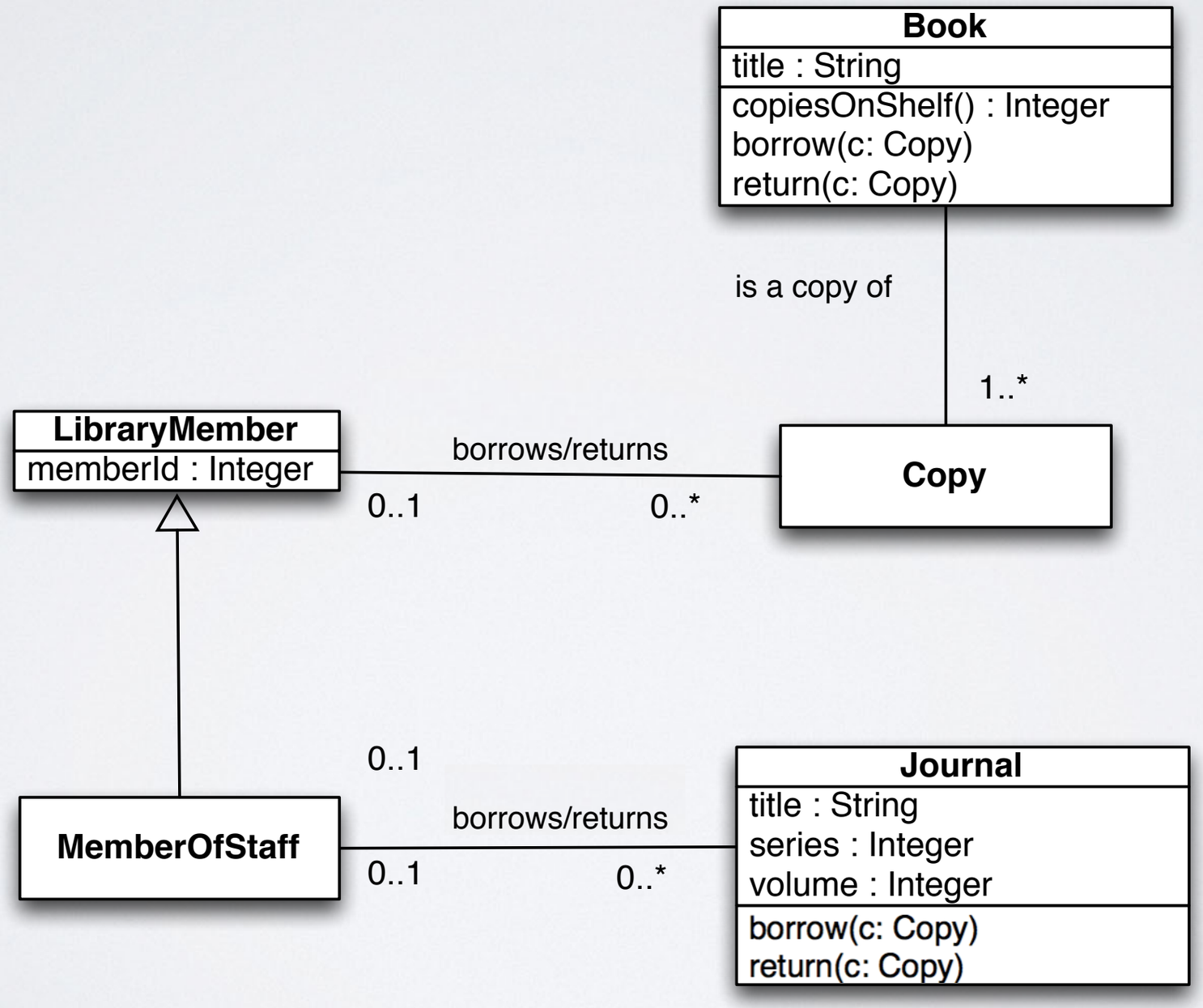
From the use cases, **identify responsibilities**

Find **which class** handles **each responsibility**

Identify which methods are required to realise each use case

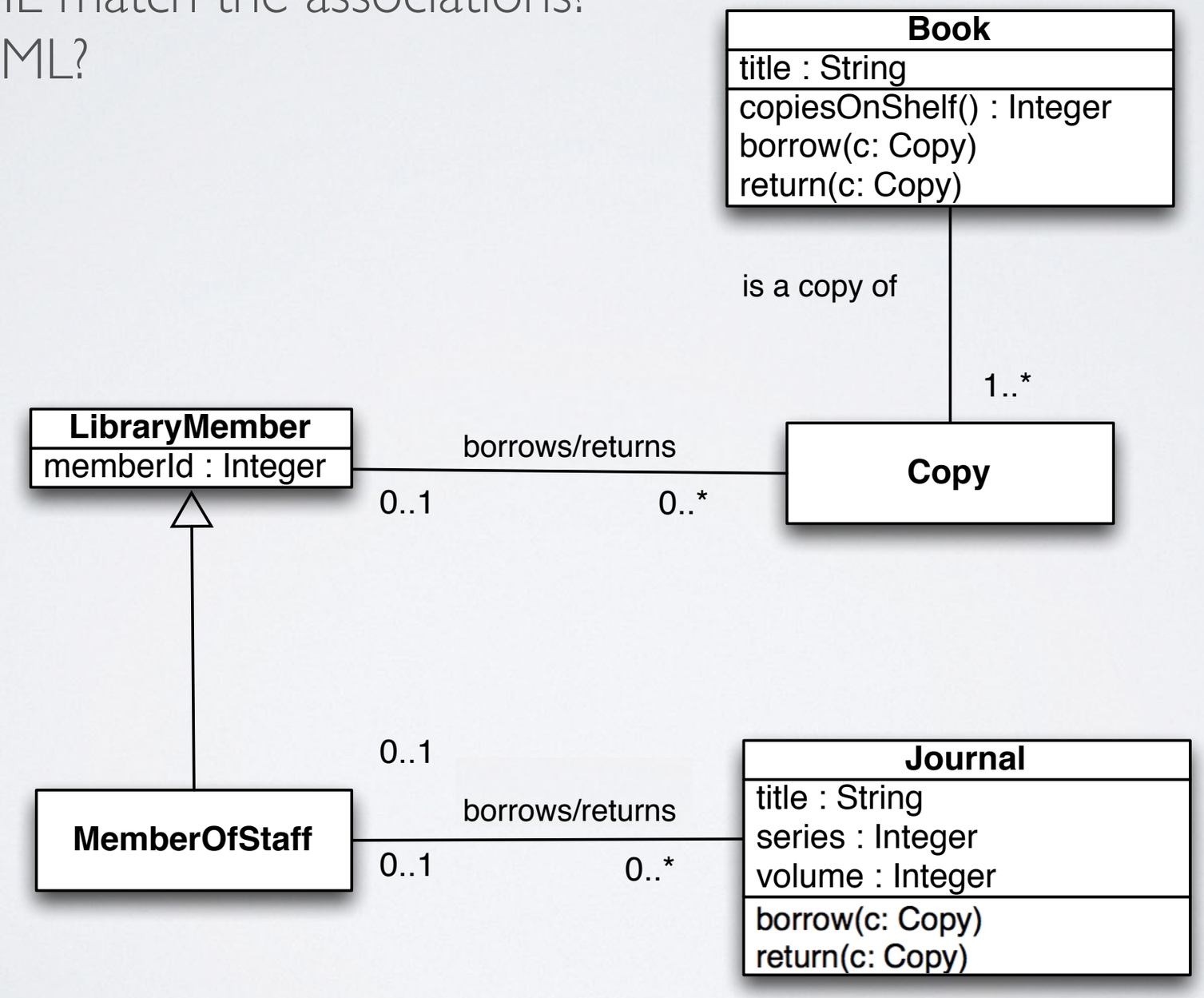
Refine when doing dynamic/behavioural modelling

EXAMPLE:


Book
title : String
copiesOnShelf() : Integer
borrow(c: Copy)
return(c: Copy)
is a copy of
1..*
LibraryMember
memberId : Integer
borrows/returns
0..1
0..*
Copy
0..1
MemberOfStaff
borrows/returns
0..1
0..*
Journal
title : String
series : Integer
volume : Integer
borrow(c: Copy)
return(c: Copy)

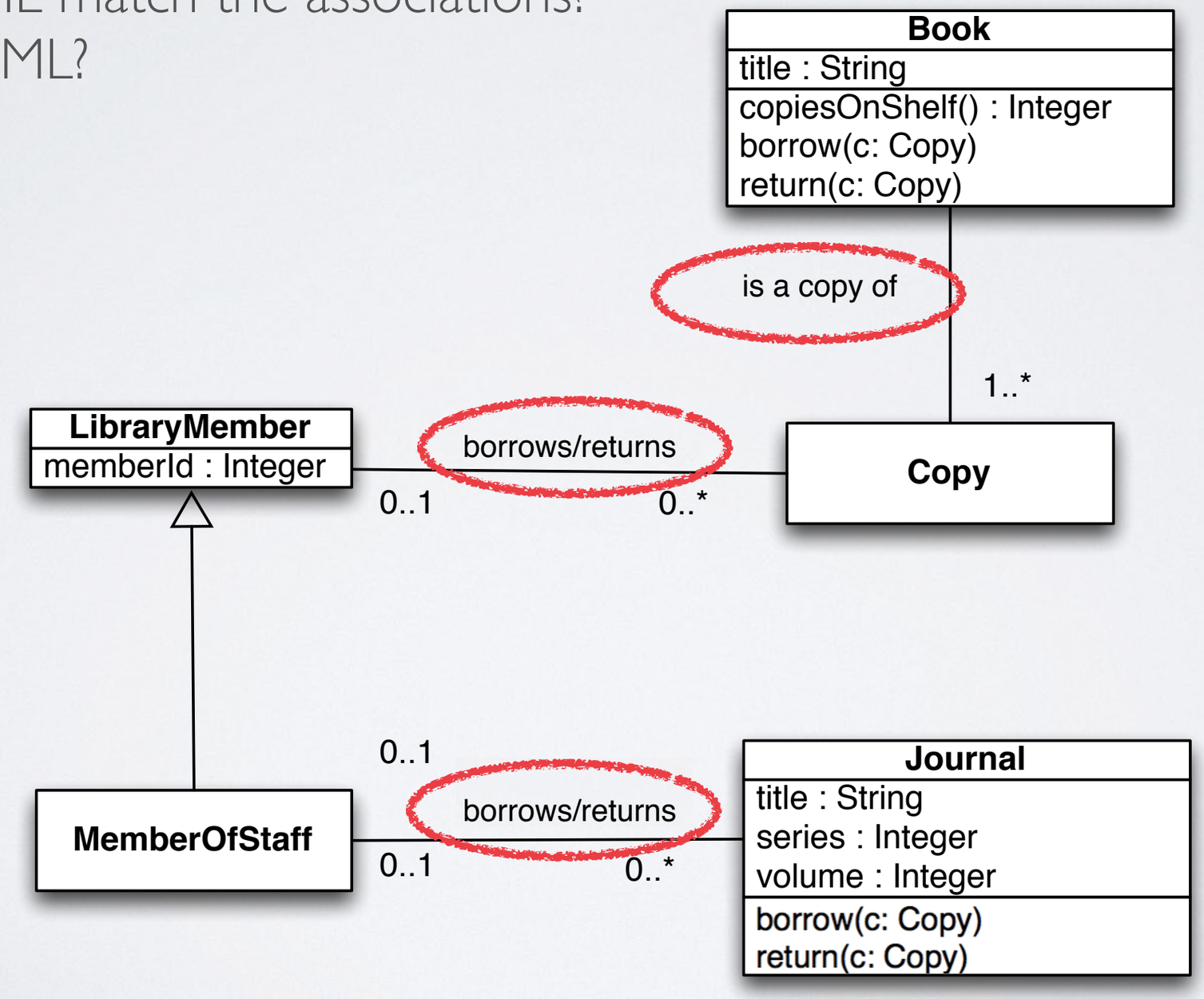
EXAMPLE:

Does the UML match the associations?
Is this valid UML?



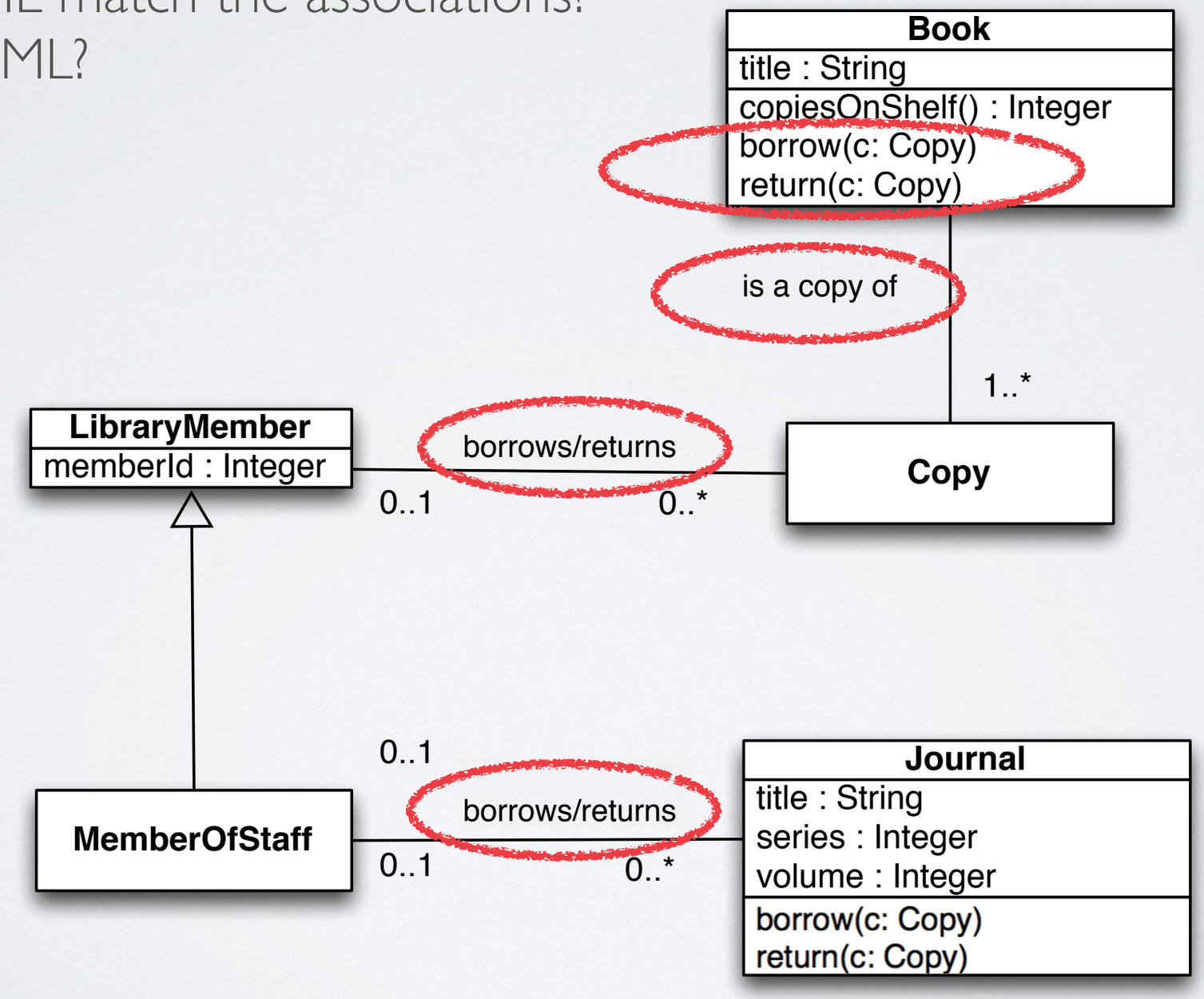
EXAMPLE:

Does the UML match the associations?
Is this valid UML?



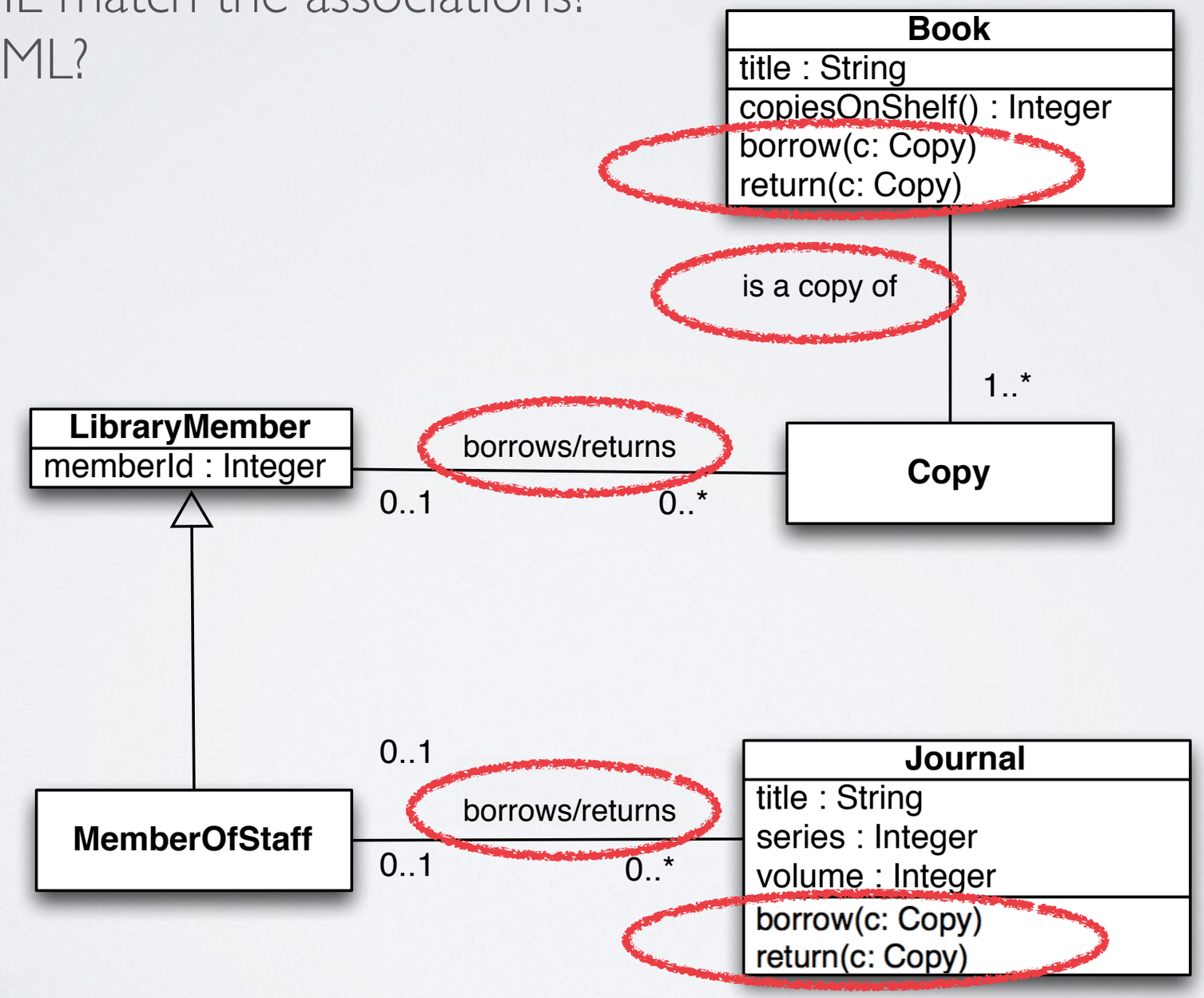
EXAMPLE:

Does the UML match the associations?
Is this valid UML?

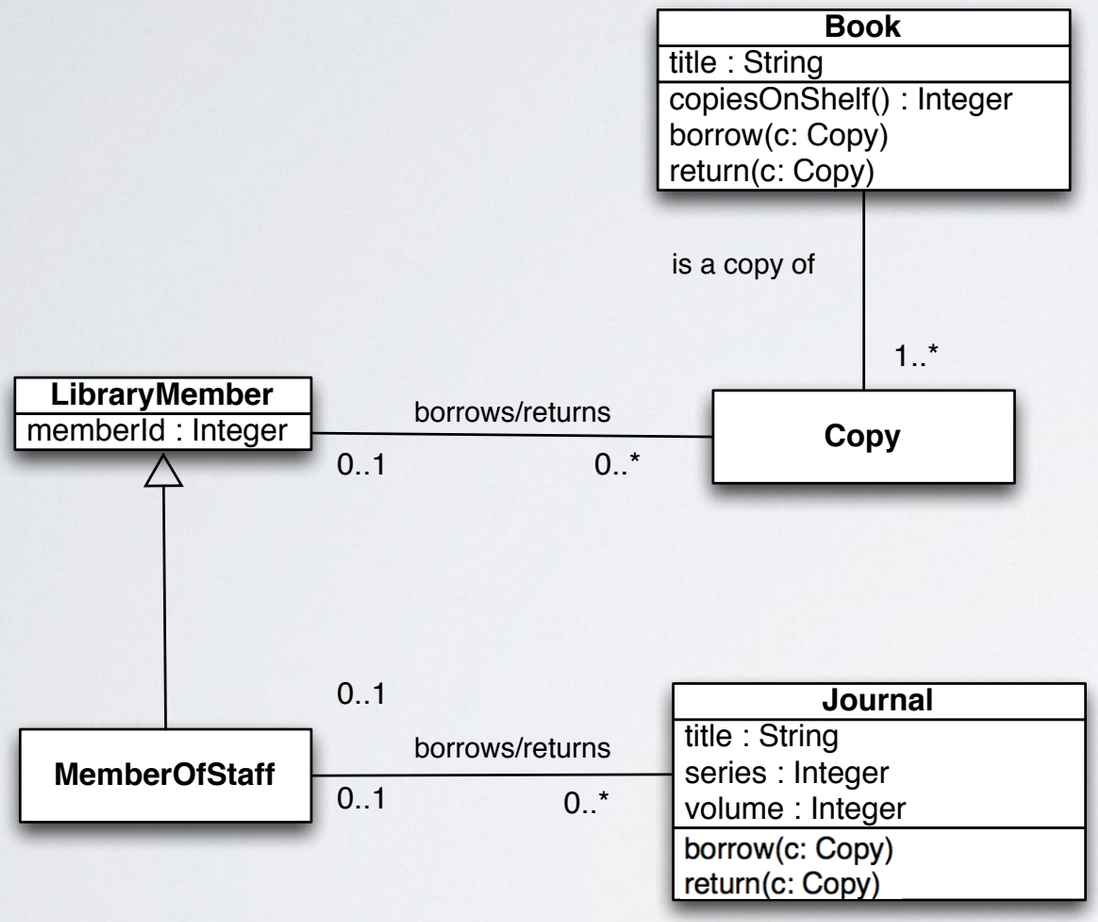


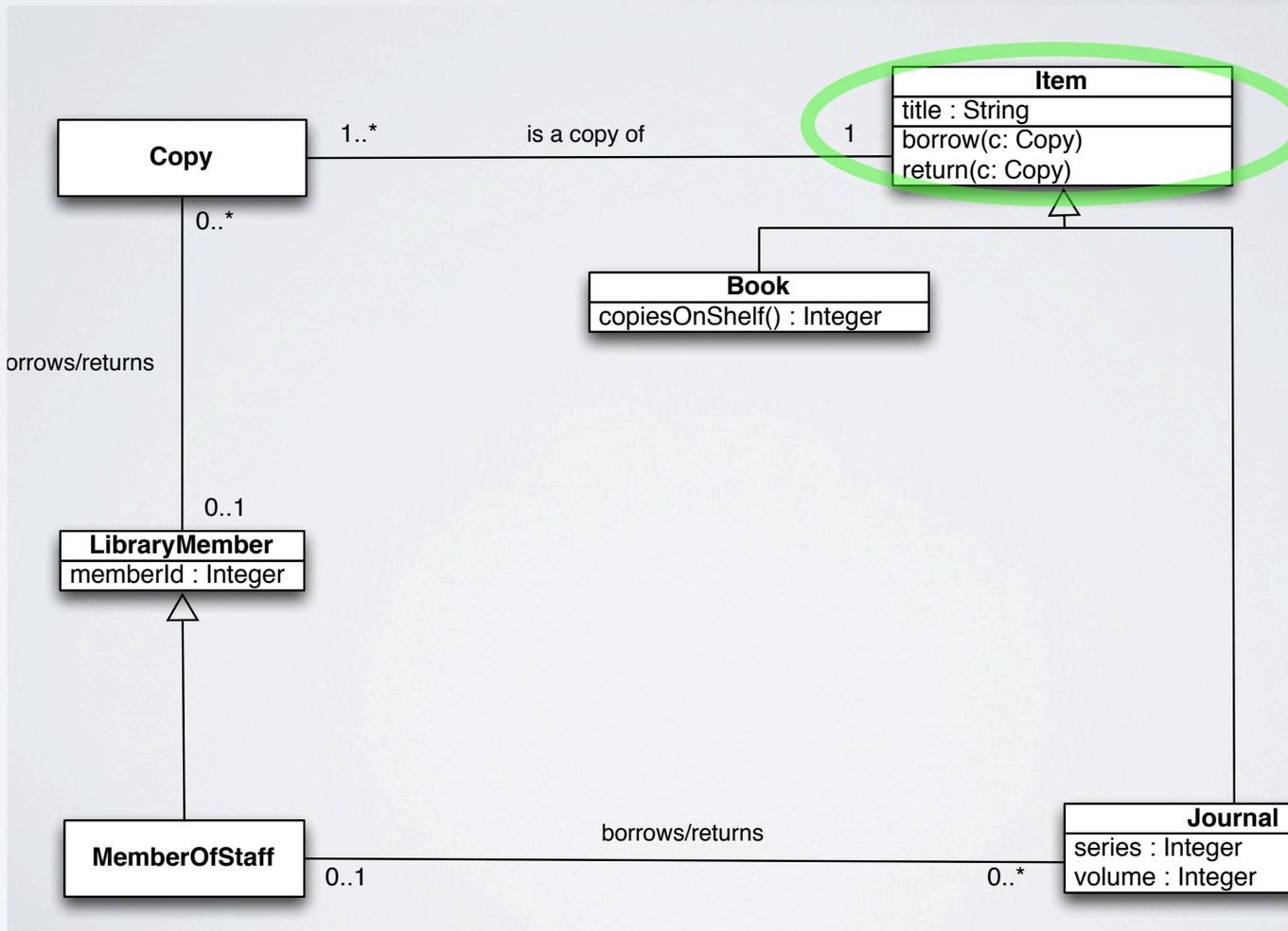
EXAMPLE:

Does the UML match the associations?
Is this valid UML?



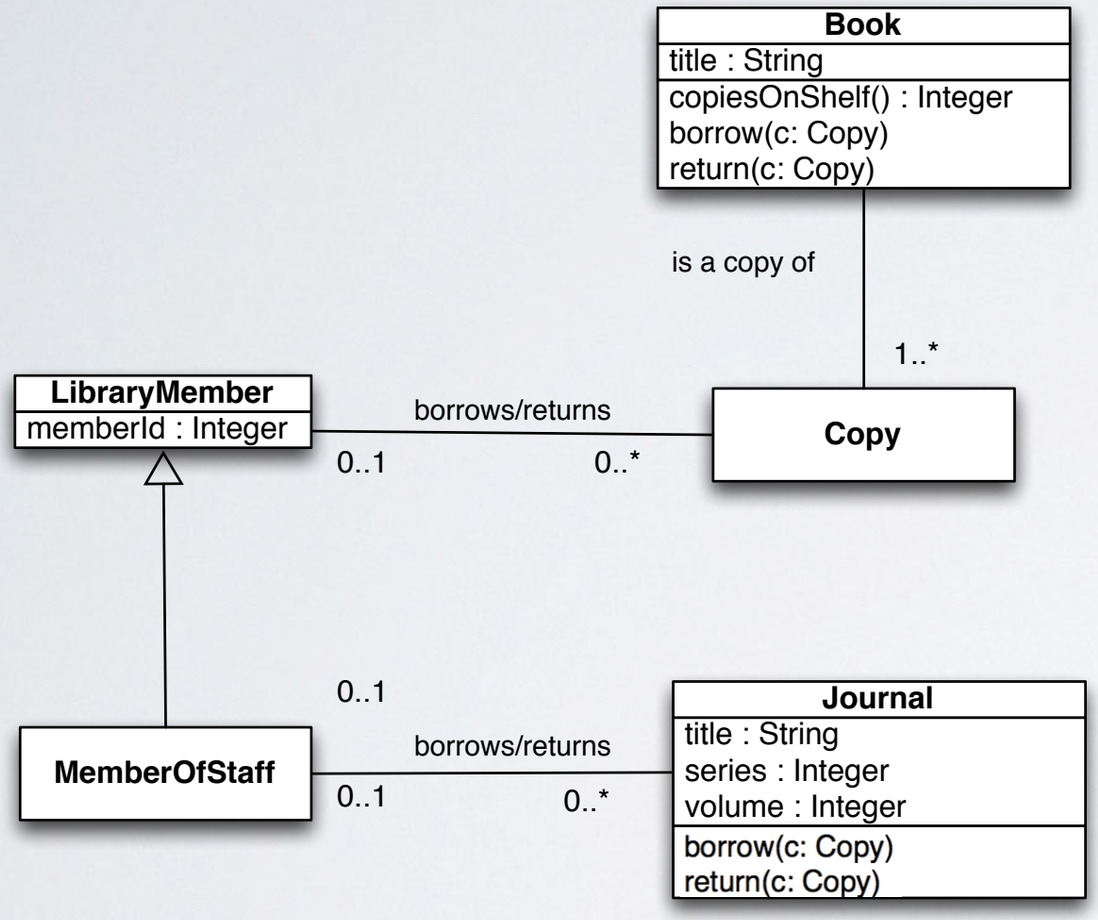
GENERALISATION

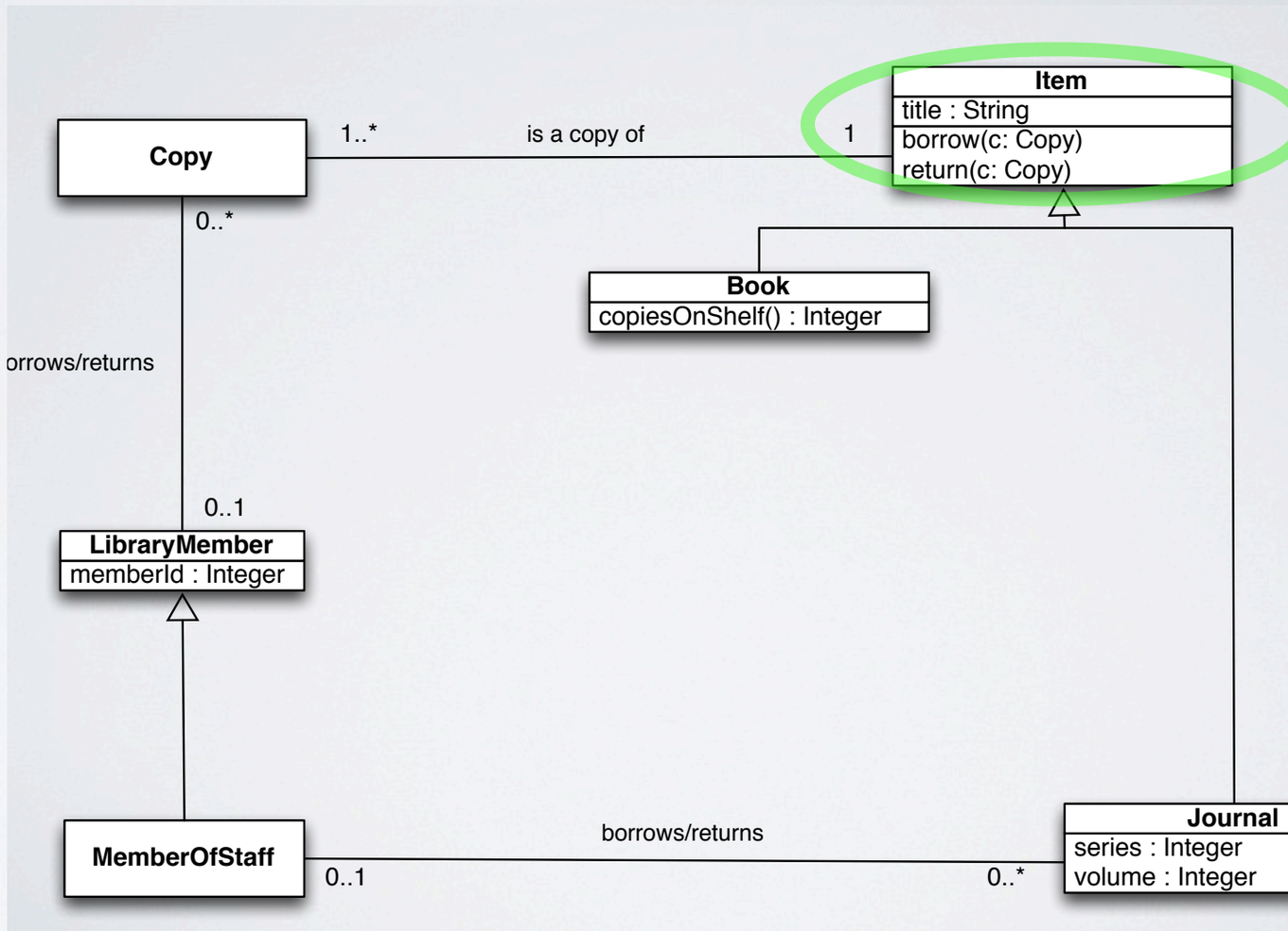


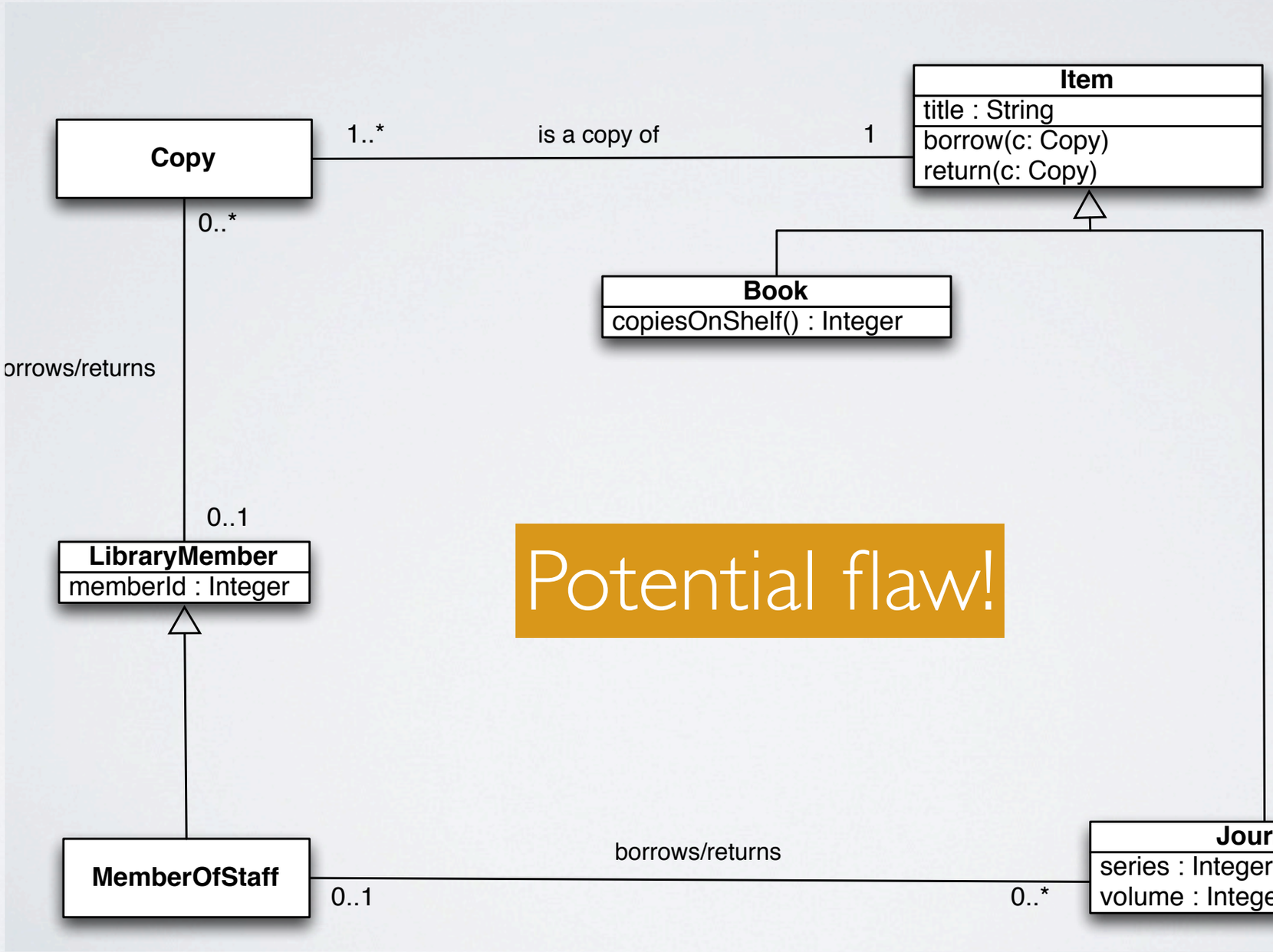


GENERALISATION

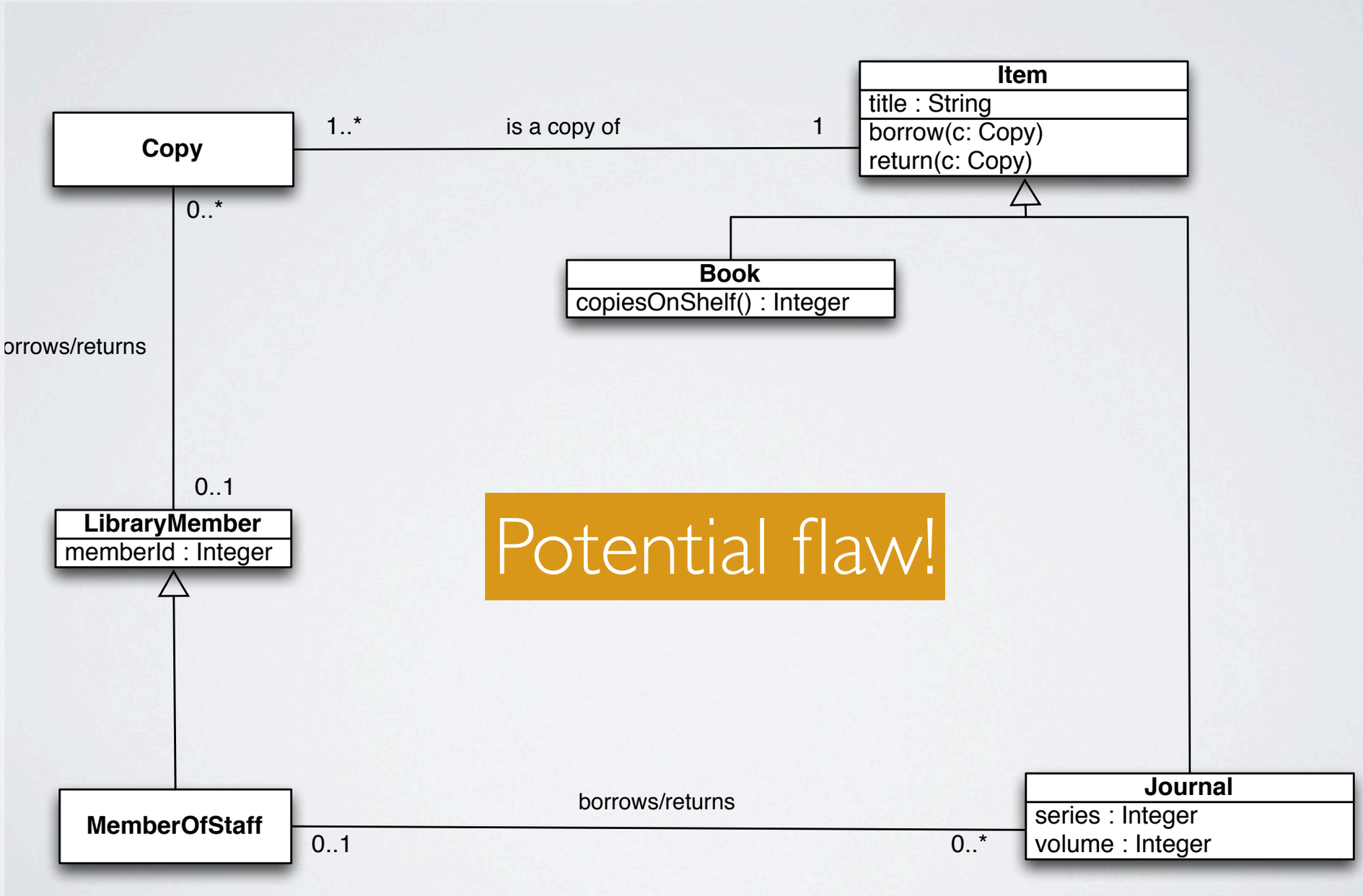
GOOD?



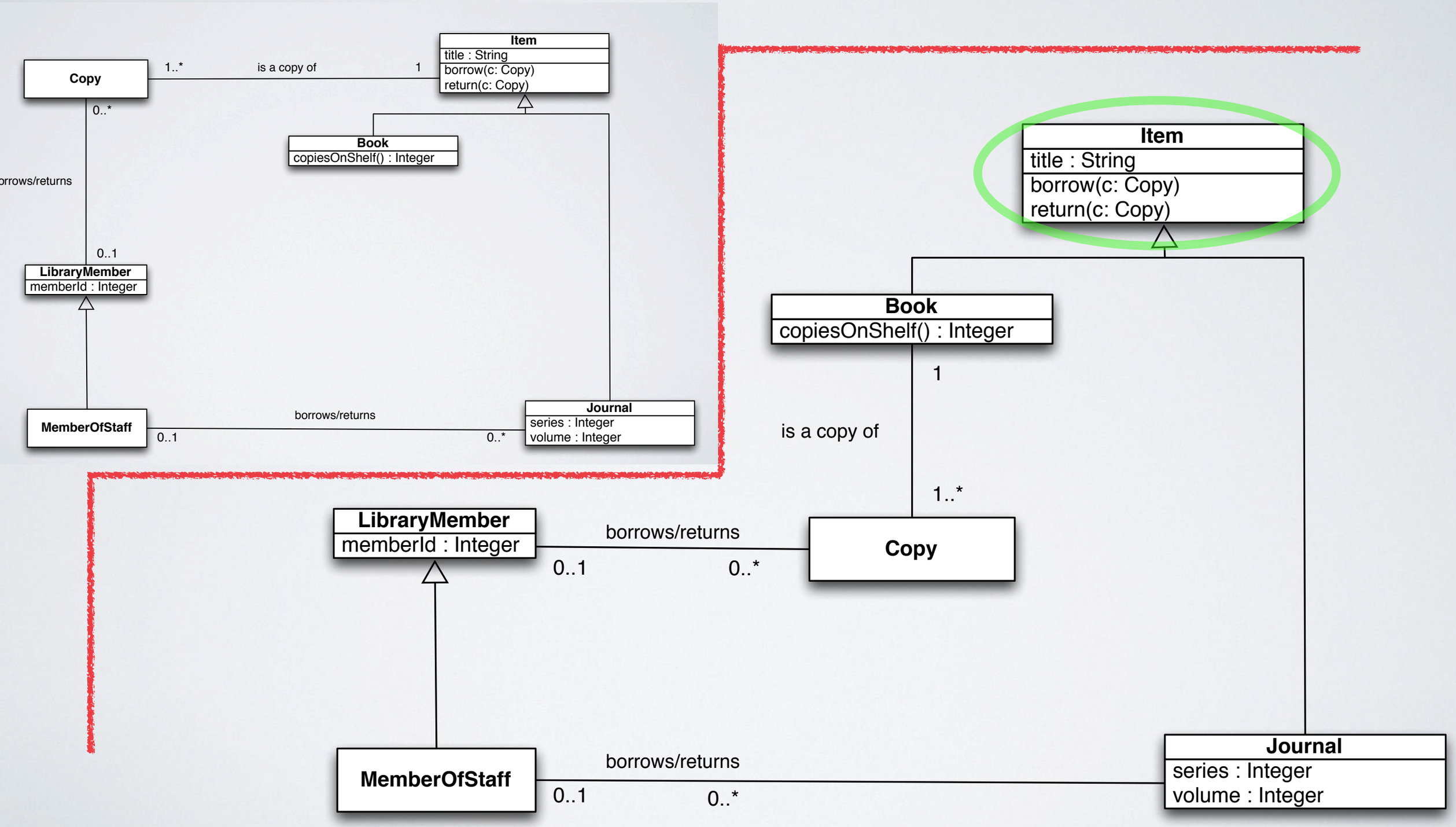



Item
title : String
borrow(c: Copy)
return(c: Copy)
Copy
1..*
is a copy of
1
0..*
Book
copiesOnShelf() : Integer
orrows/returns
0..1
LibraryMember
memberId : Integer
Potential flaw!
MemberOfStaff
borrows/returns
0..1
0..*
Journal
series : Integer
volume : Integer

```
public class Copy {
    public Item item;
    public Copy(Item item){ this.item = item; }
    public Copy(Book book){ this.item = book; }
}
```

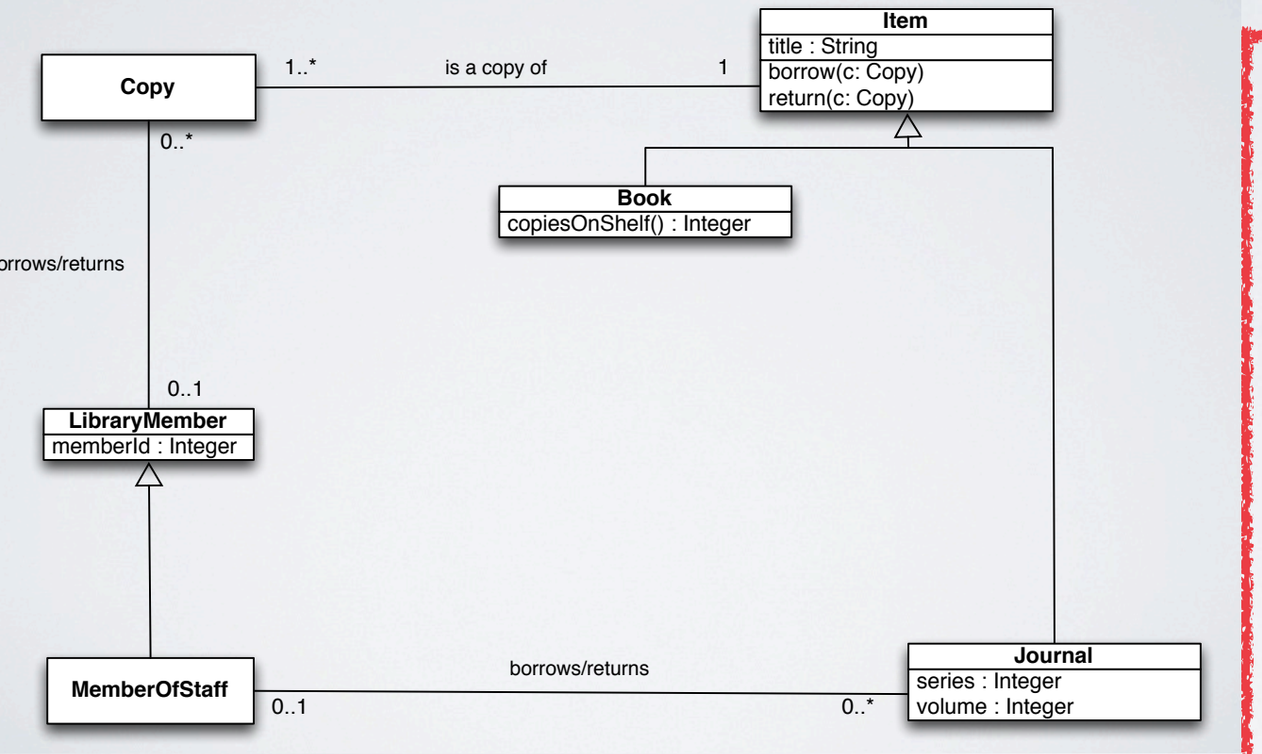


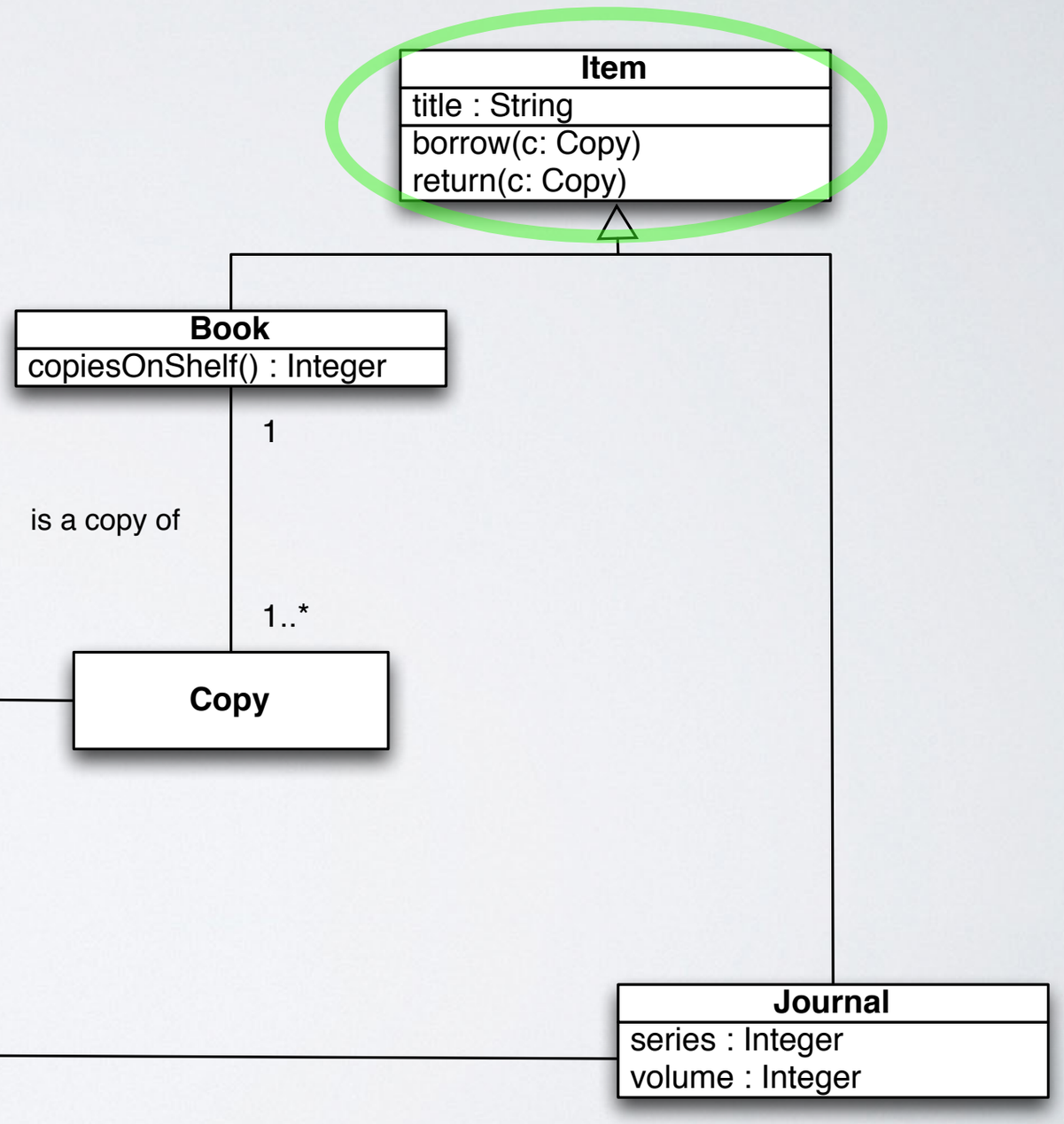
GENERALISATION



GENERALISATION

Is this a good design?

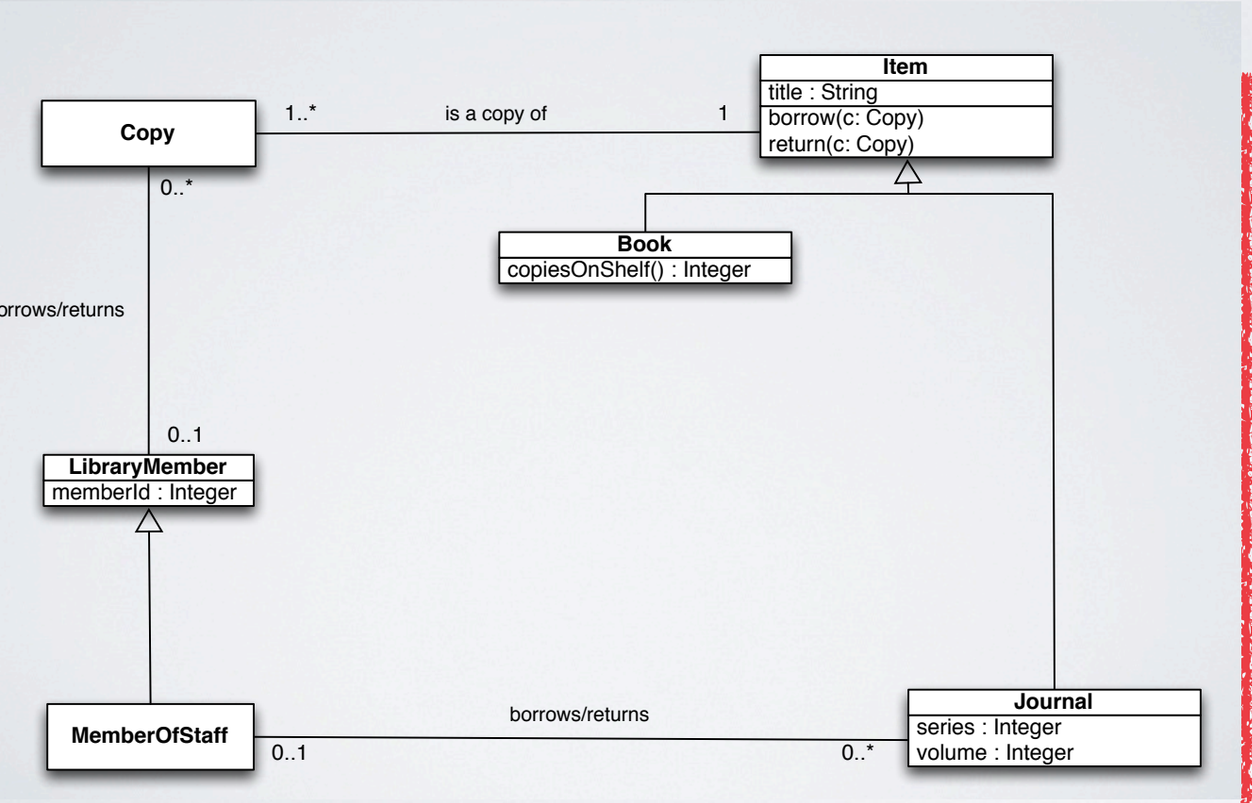


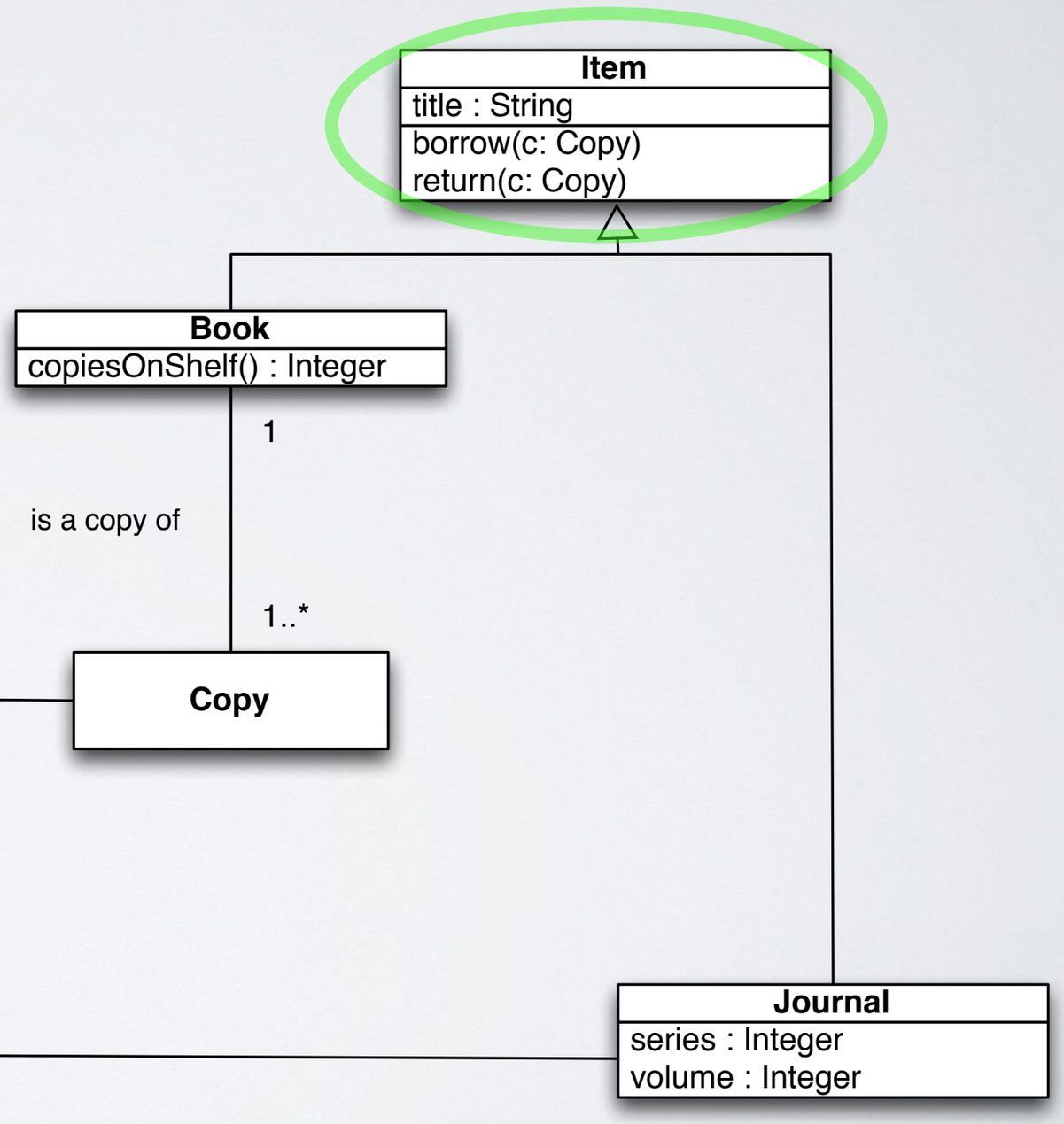


GENERALISATION

Is this a good design?

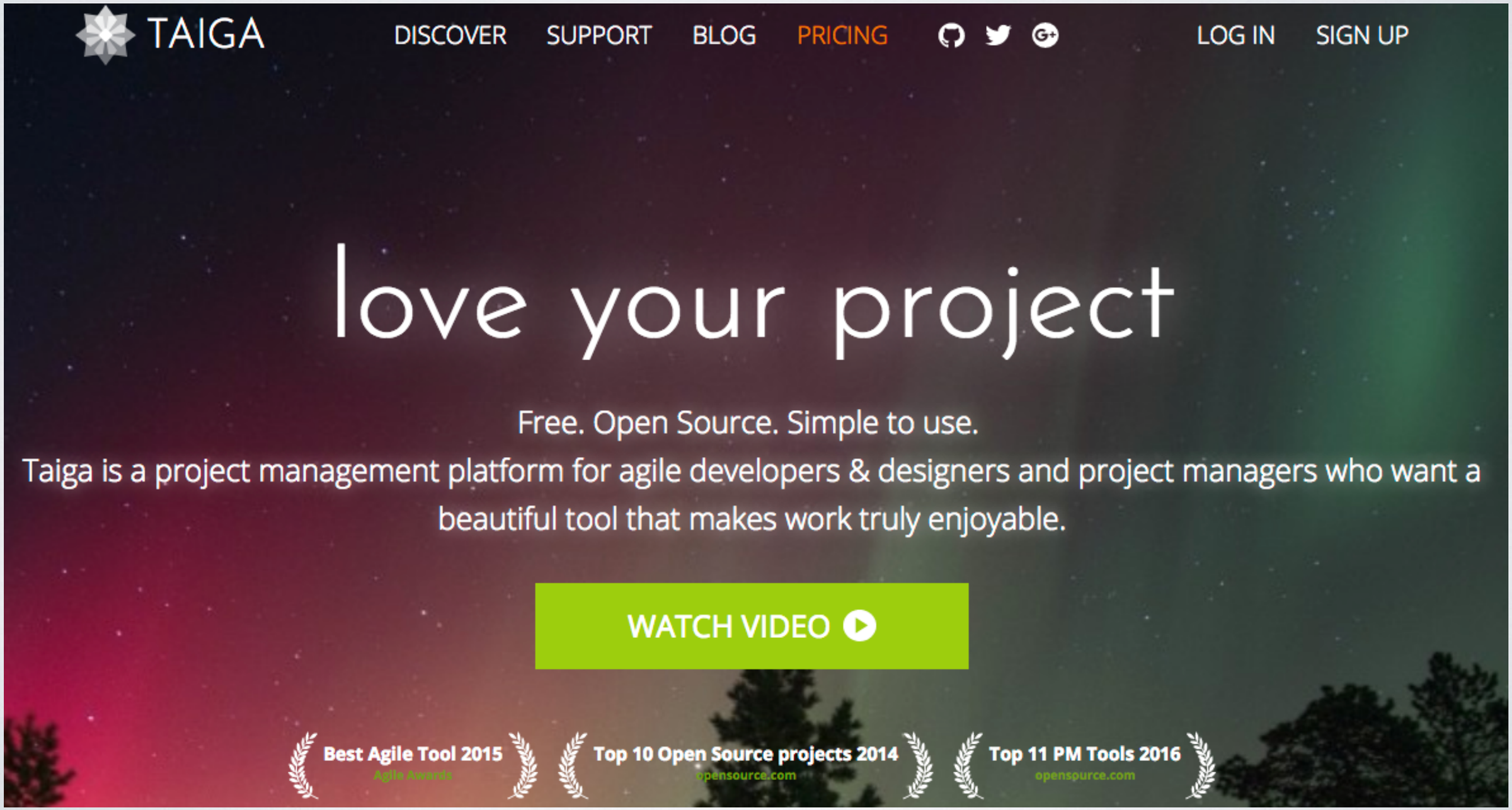
- Abstract class,
- Polymorphism,
- Classes with attributes





IN-CLASS EXERCISE

Group work



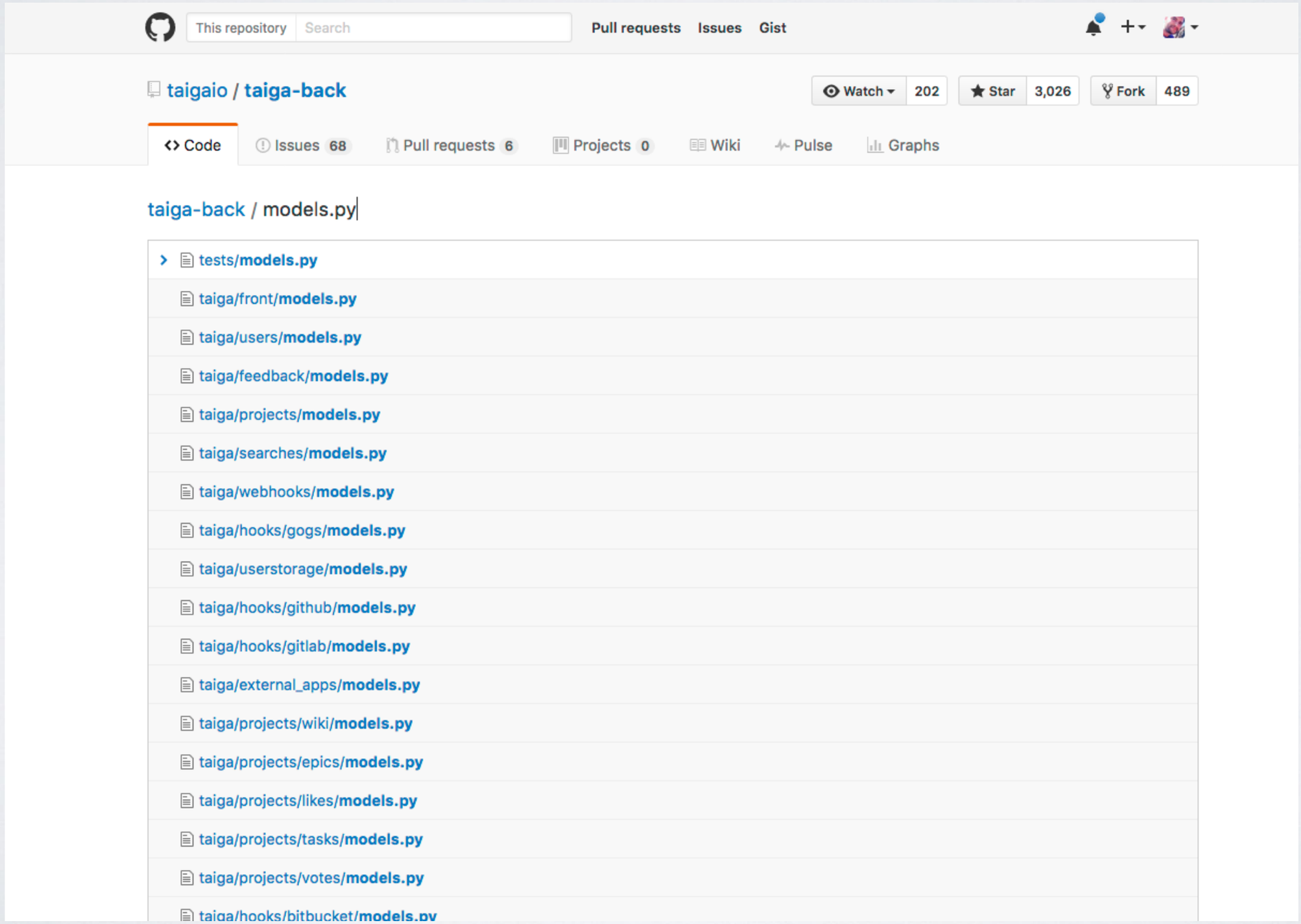
Group work

UML CLASS DIAGRAM

(Add UserStories, Projects, Points, etc)

TAIGA

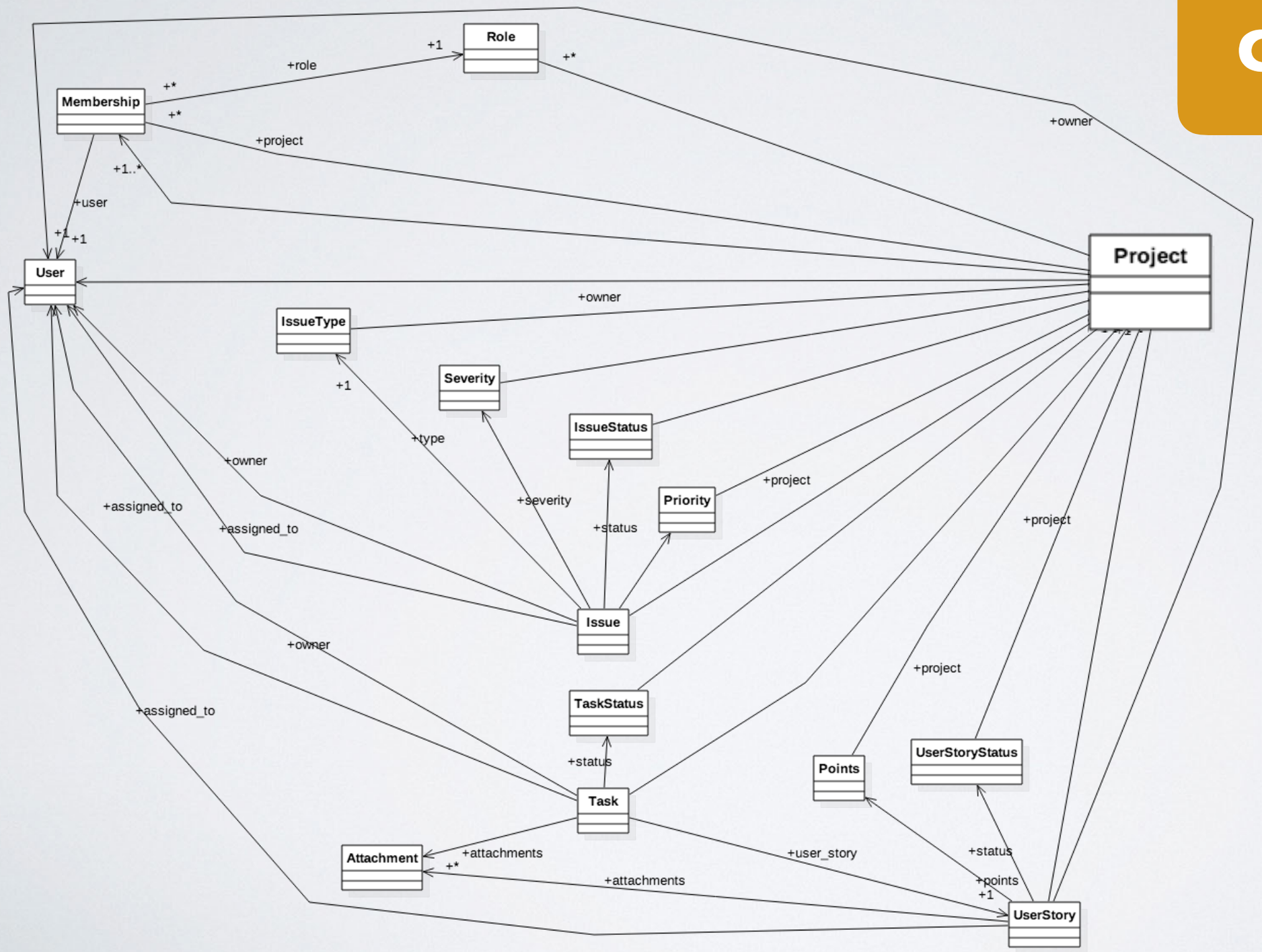
- Let's look at the code



ALTERNATIVE SOLUTION

CLASS DIAGRAM

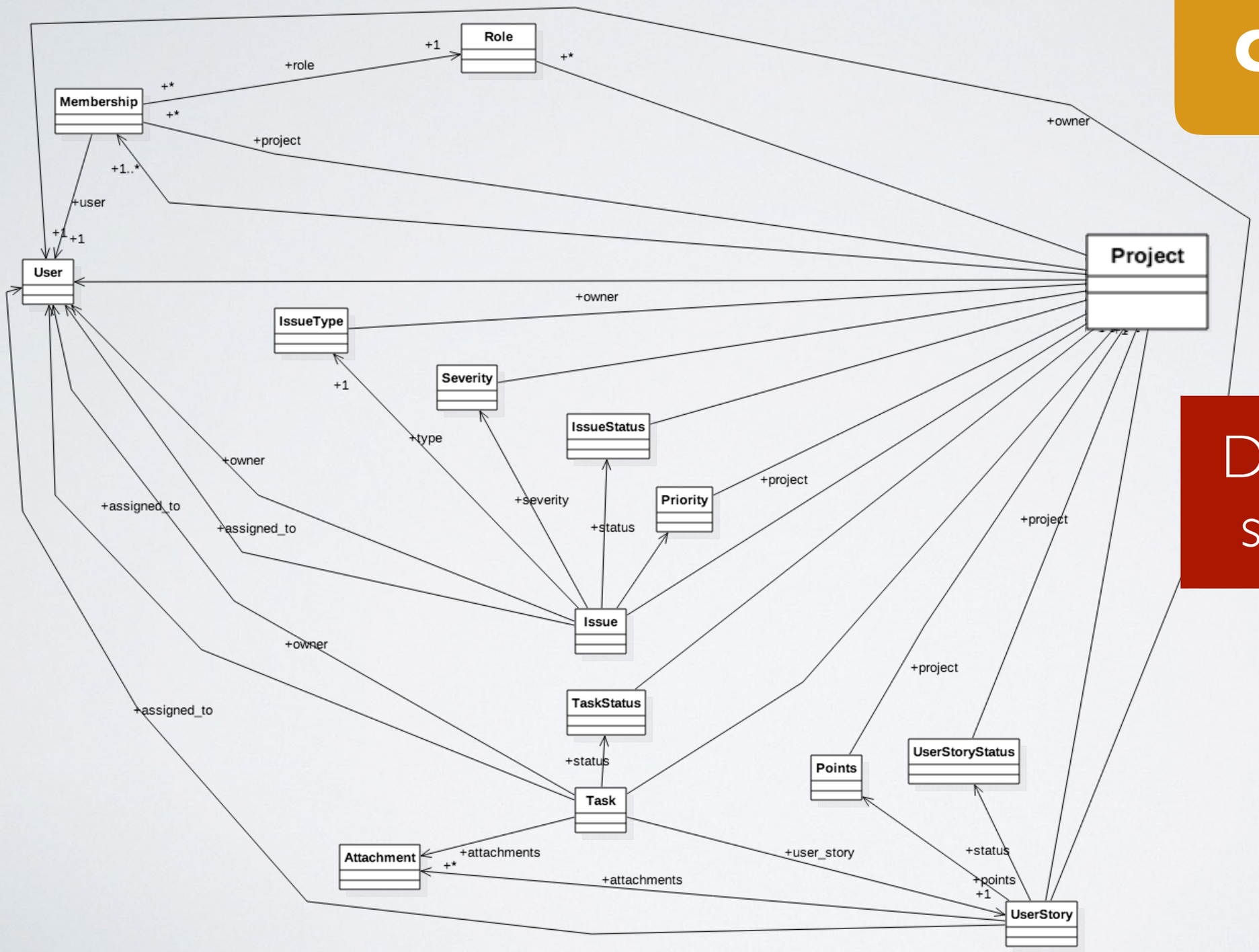
Group work



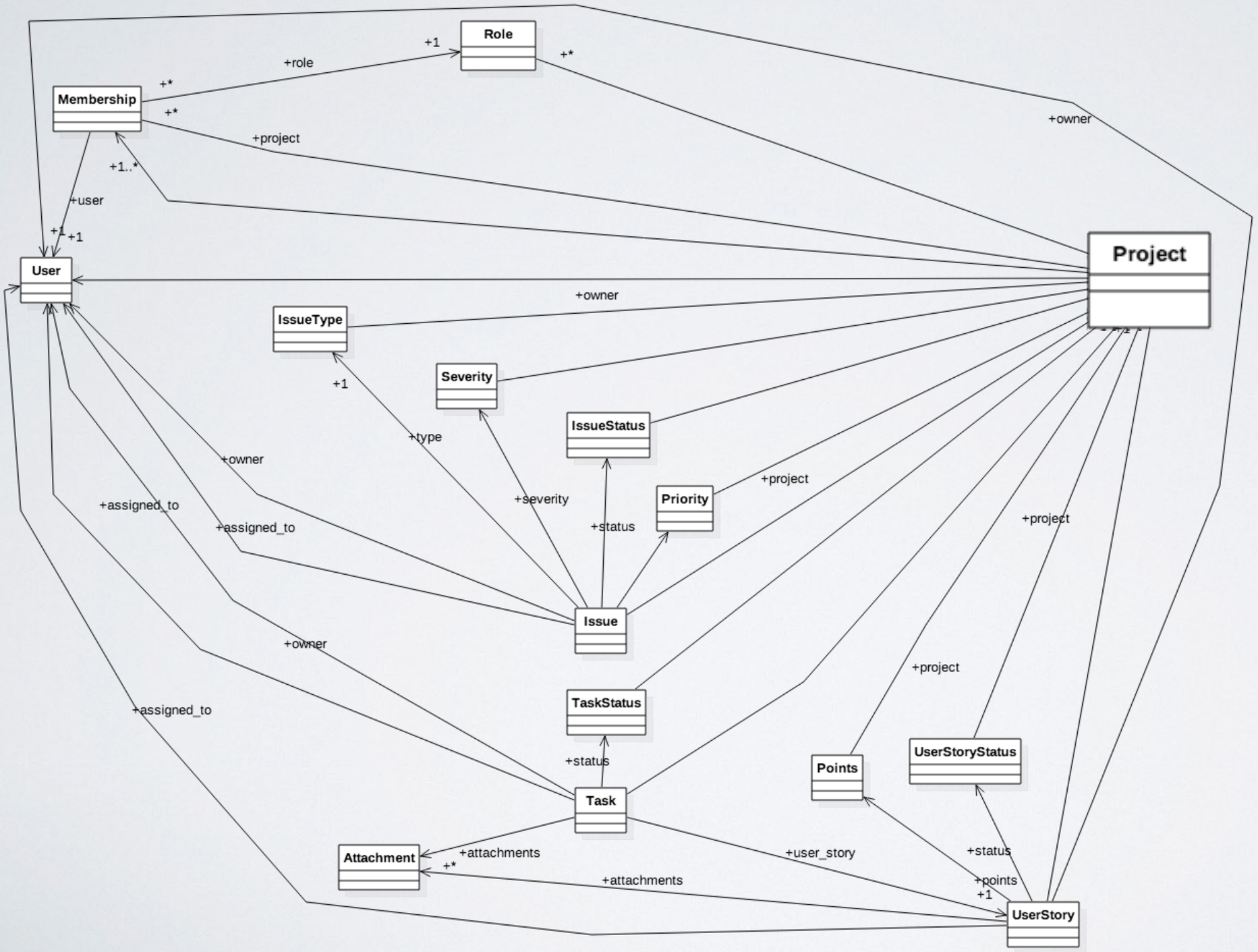
CLASS DIAGRAM

Group work

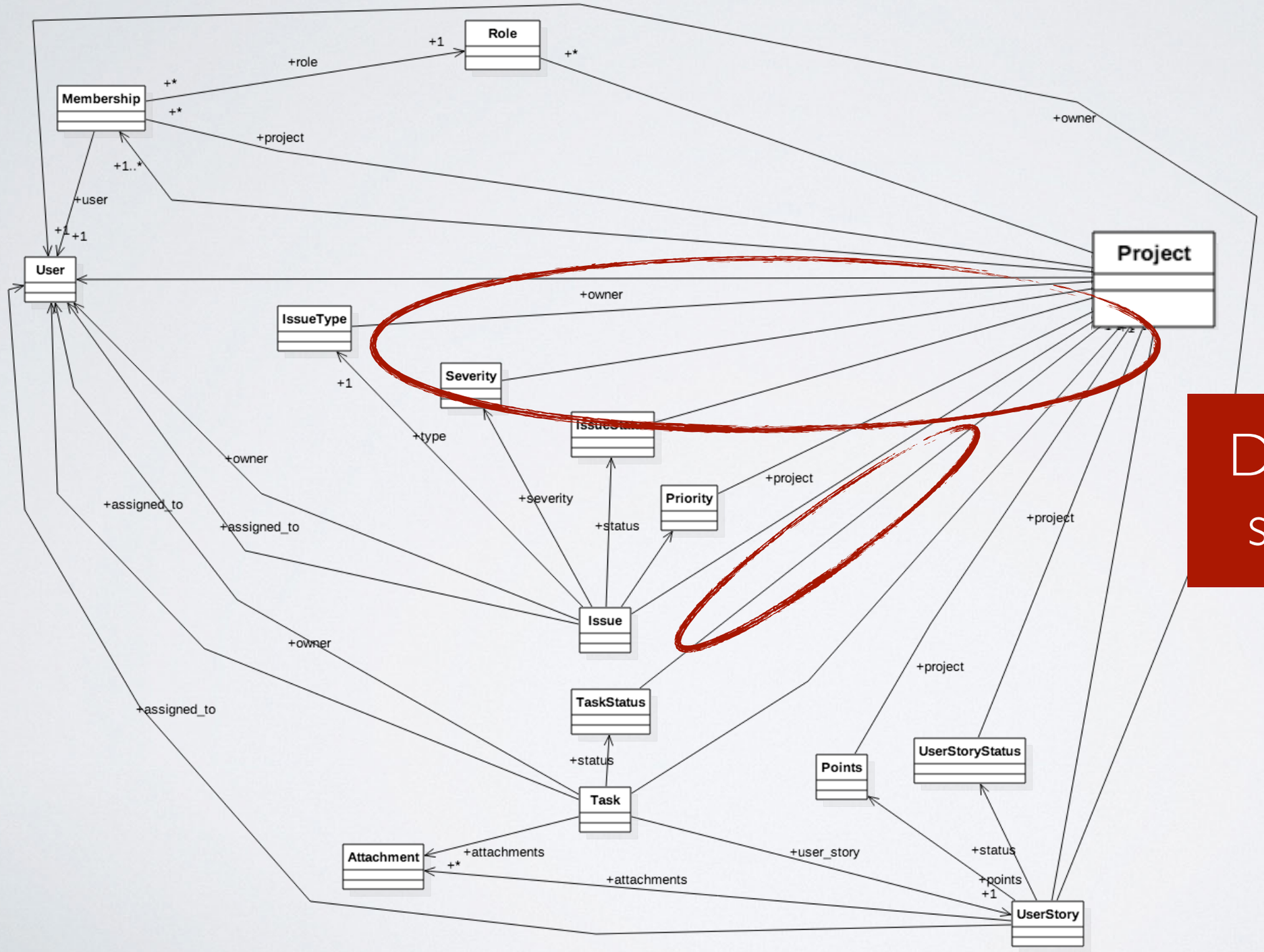
Does this design seems correct?



CLASS DIAGRAM

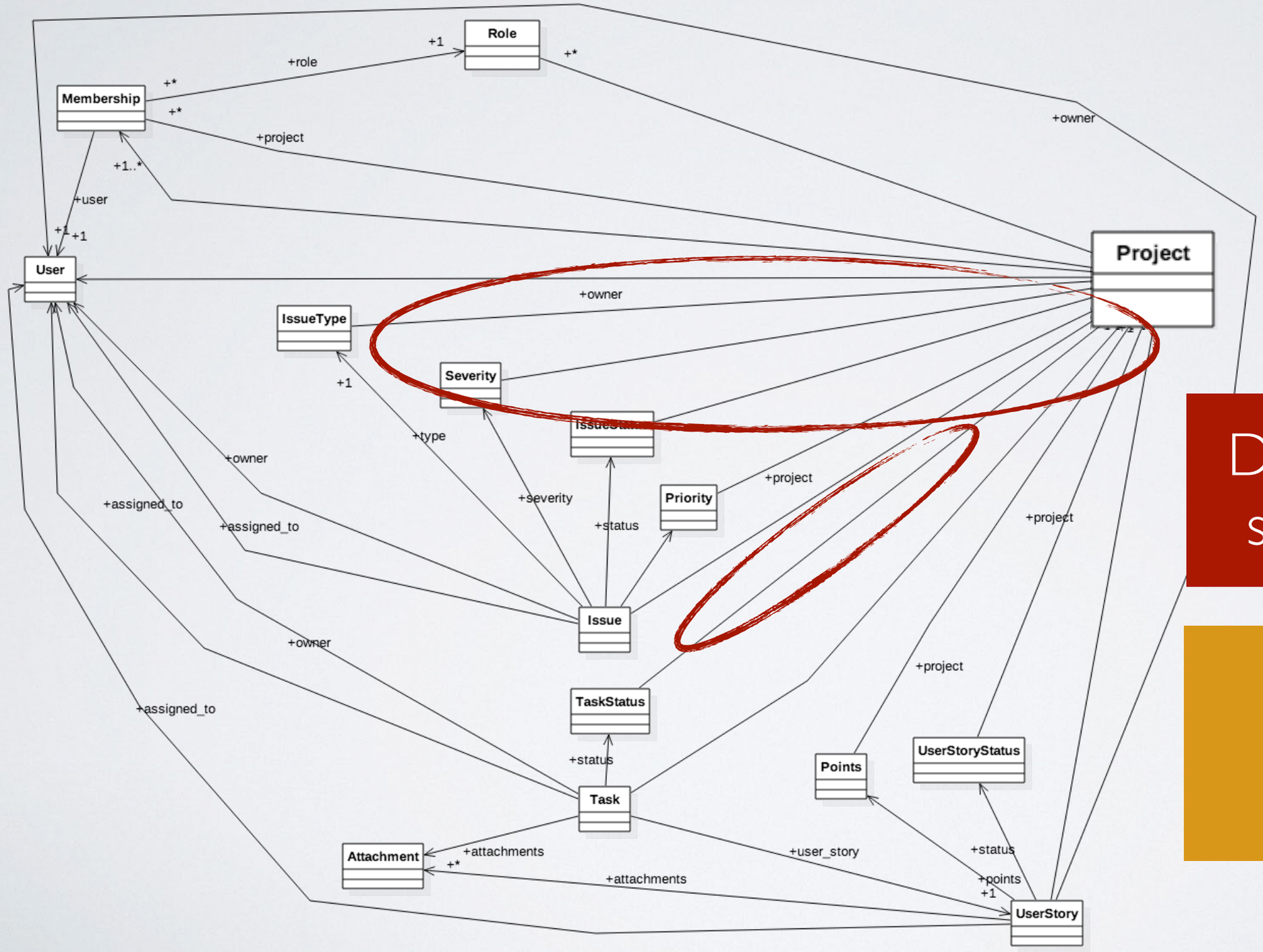


CLASS DIAGRAM



Does this design seems correct?

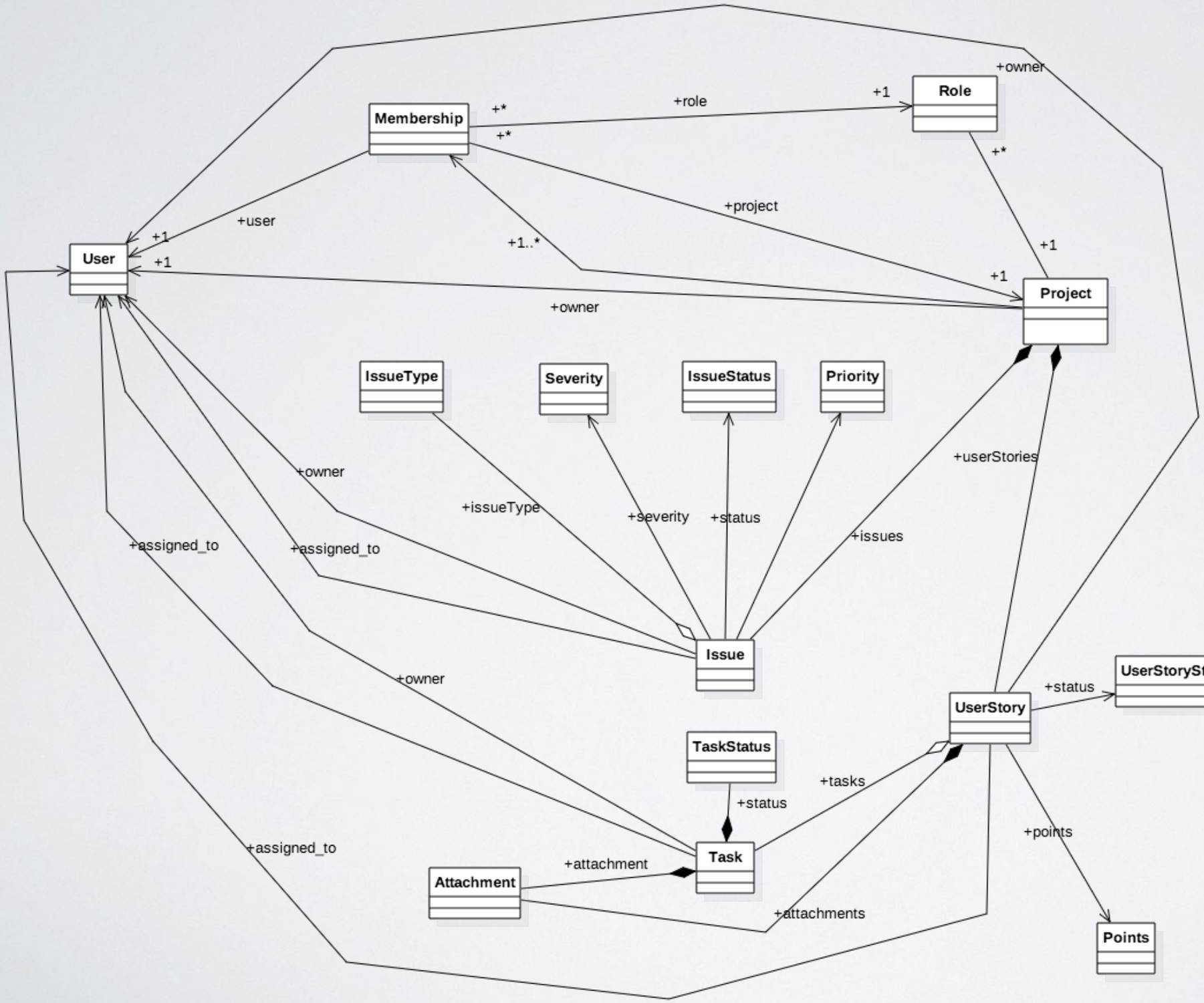
CLASS DIAGRAM



Does this design seems correct?

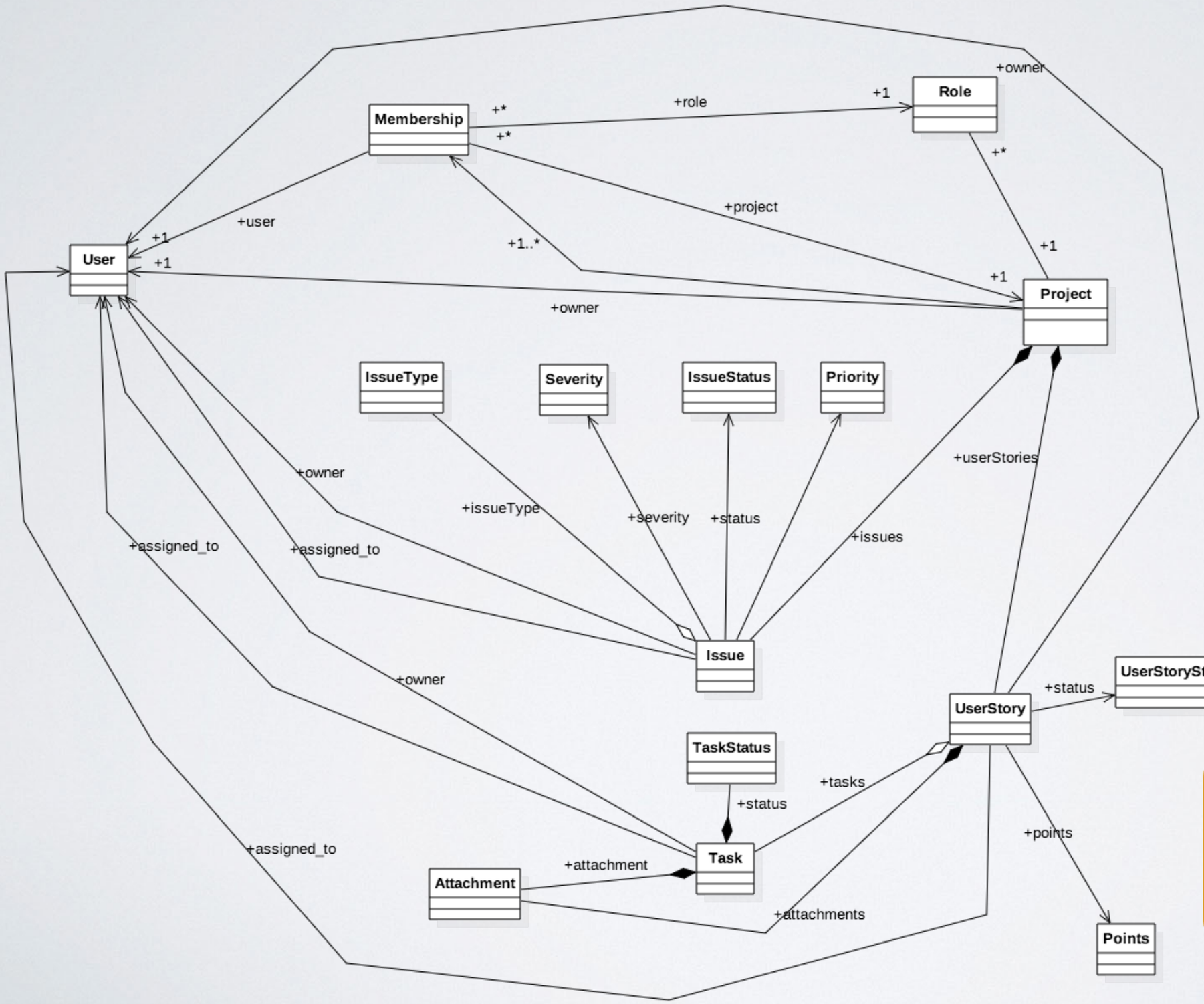
Let's fix it!

CLASS DIAGRAM



More precise

CLASS DIAGRAM

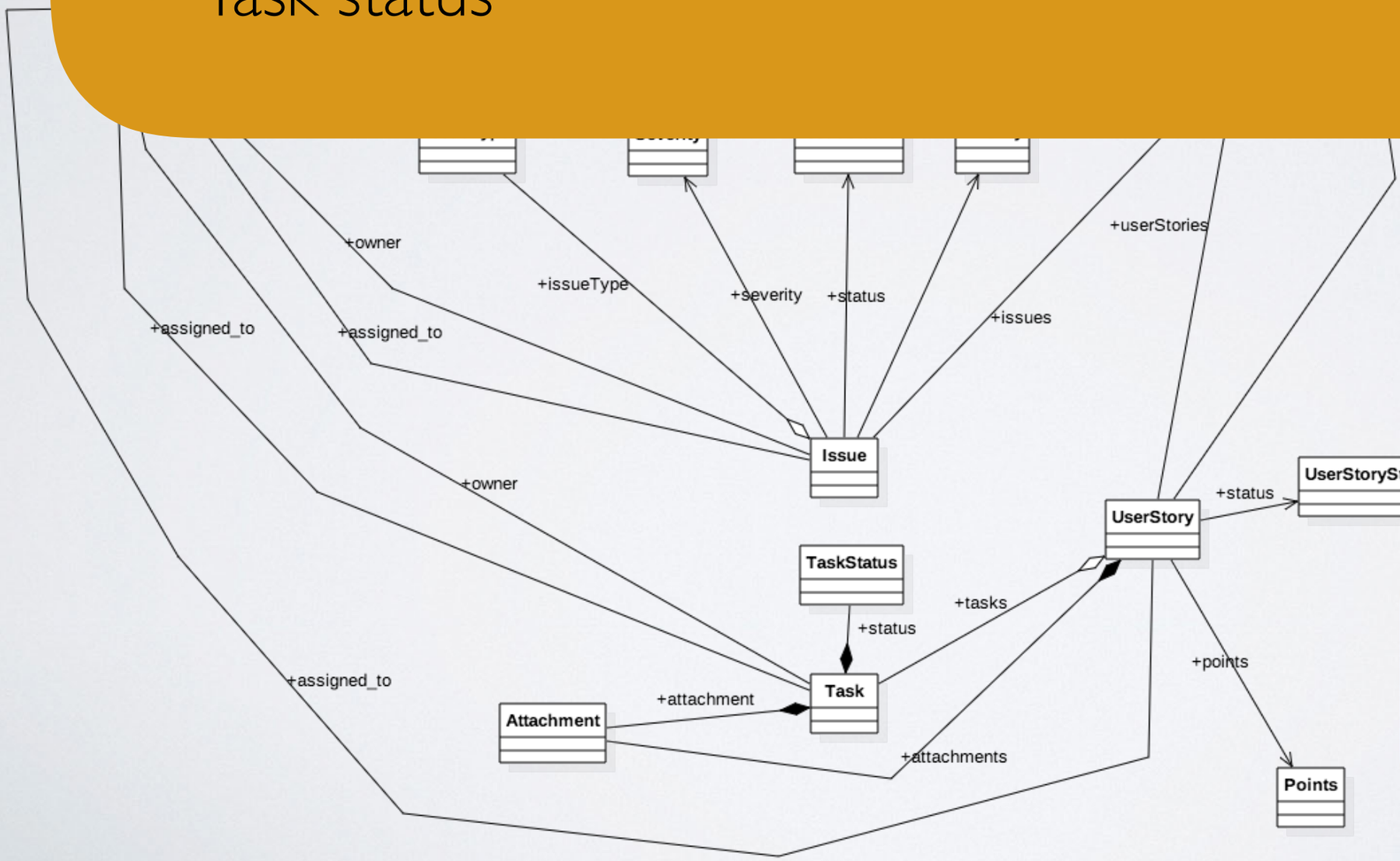


More precise

Did we lose anything?

From a project, we cannot filter issues based on:

- Priority
- Issue status
- Severity
- Issue type
- Task status

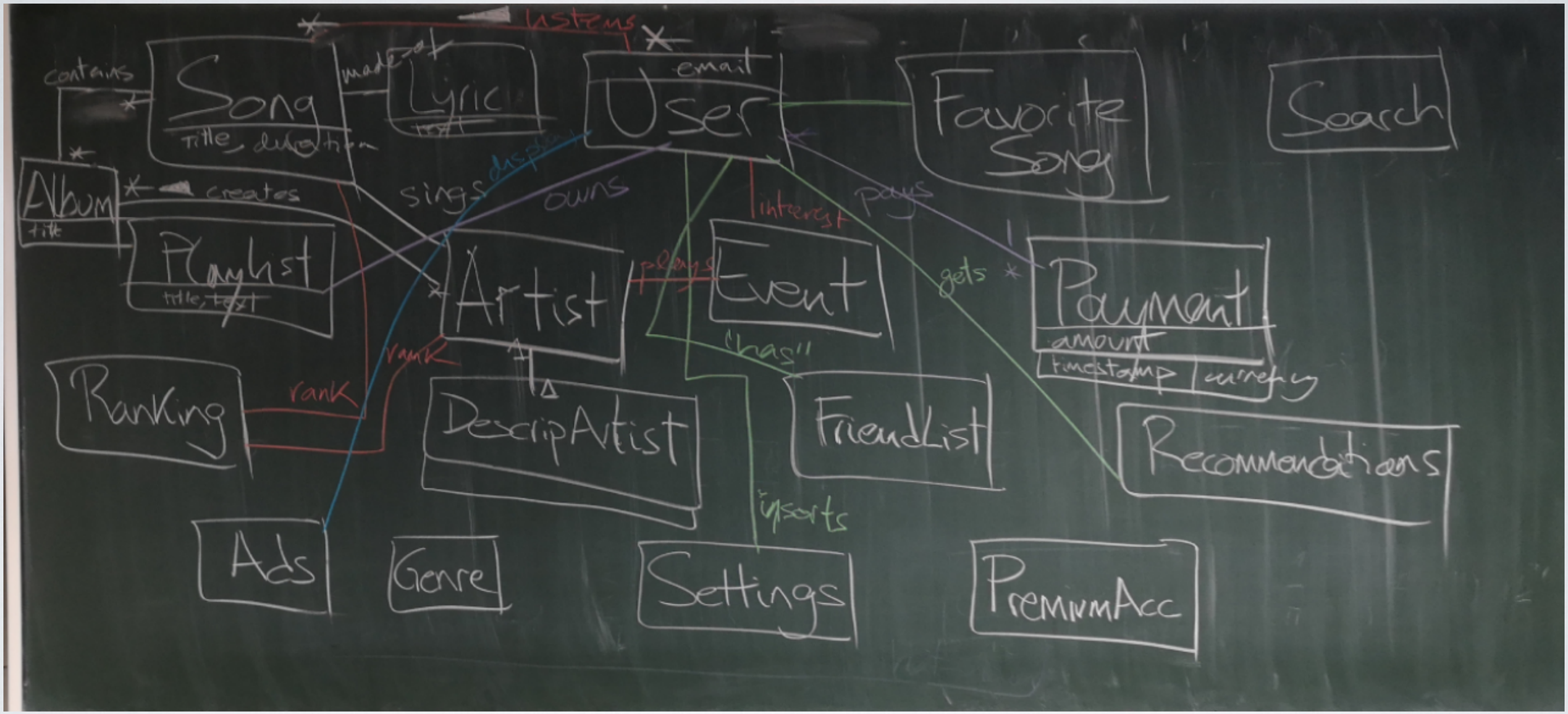


More precise

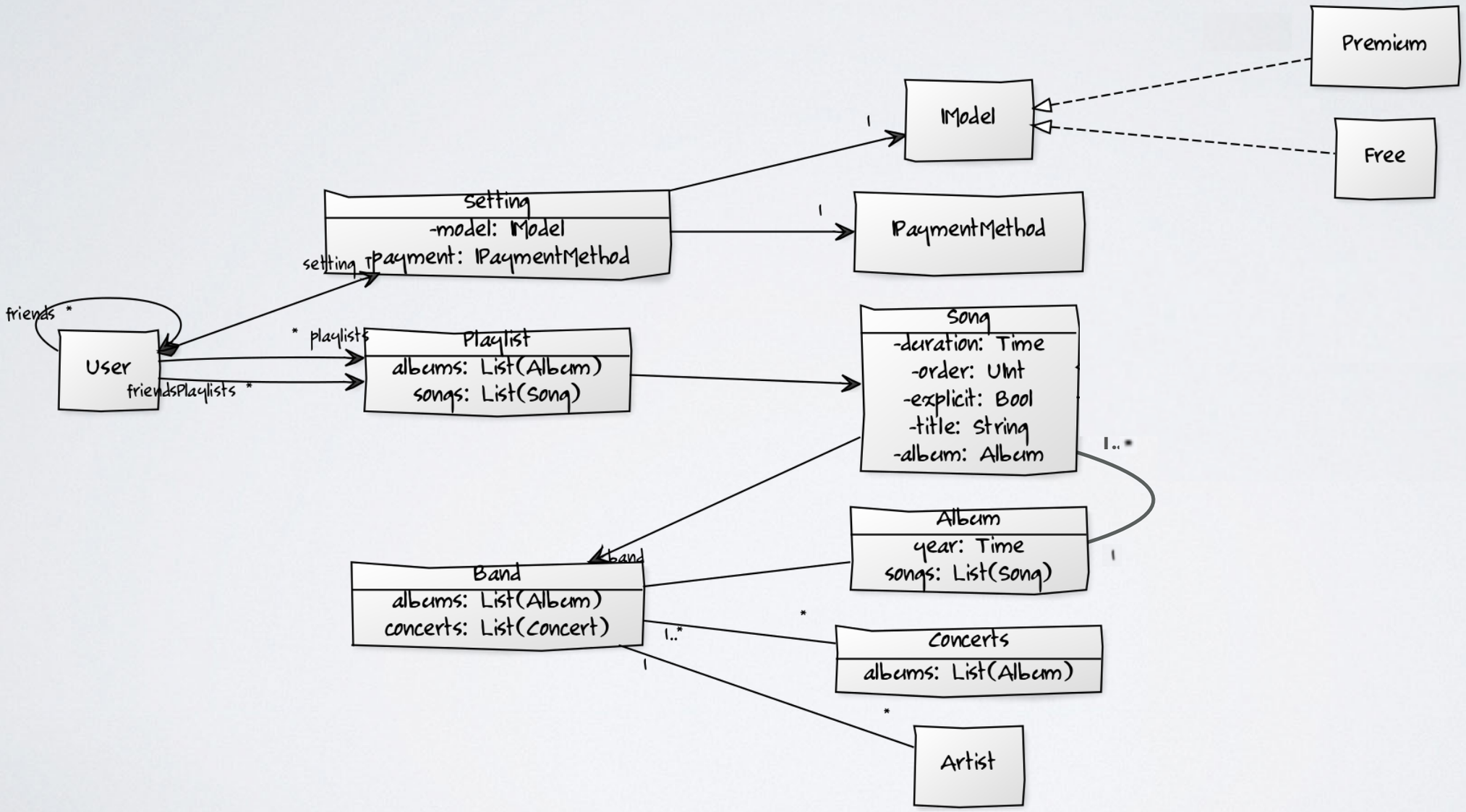
Did we lose anything?

Group work

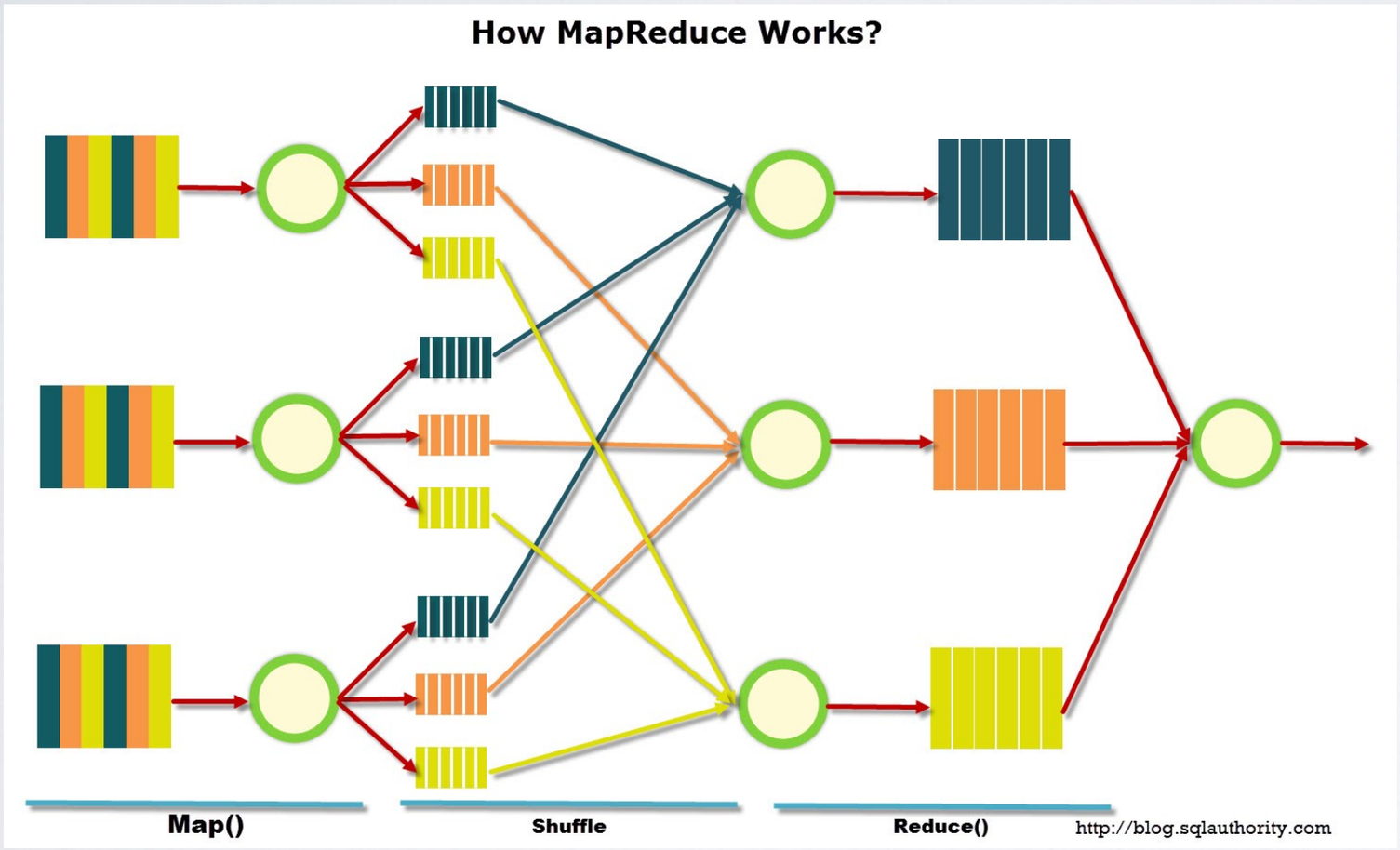
CREATE CLASS DIAGRAM


listens
contains
Song
Title, duration
made
Lyric
text
email
User
Favorite Song
Search
Album
title
creates
sings
displays
owns
Playlist
title, text
Artist
interest
pays
plays
Event
"has"
gets
Payment
amount
timestamp
currency
Ranking
rank
rank
DescripArtist
FriendList
Recommendations
inserts
Ads
Genre
Settings
PremiumAcc

SPOTIFY


Premium
IModel
Free
Setting
-model: IModel
setting
-payment: IPaymentMethod
IPaymentMethod
friends *
User
* playlists
Playlist
albums: List(Album)
songs: List(Song)
friendsPlaylists *
Song
-duration: Time
-order: UInt
-explicit: Bool
-title: String
-album: Album
1..*
Album
year: Time
songs: List(Song)
band
Band
albums: List(Album)
concerts: List(concert)
1..*
Concerts
albums: List(Album)
Artist

MAP_REDUCE FRAMEWORK




```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator
values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);

  Emit(AsString(result));
```

MAP_REDUCE FRAMEWORK

Most Visited | NewZealand | LastPaper | splash19 | DisCoTec | UU | ACM-W | Visiting Research | Actor | Artefact guidelines

All Classes

Packages

org.apache.hadoop
org.apache.hadoop.classification
org.apache.hadoop.conf
org.apache.hadoop.contrib.utils.join
org.apache.hadoop.crypto
org.apache.hadoop.crypto.key
org.apache.hadoop.crypto.key.kms
org.apache.hadoop.crypto.key.kms.server
org.apache.hadoop.crypto.random
org.apache.hadoop.examples
org.apache.hadoop.examples.dancing
org.apache.hadoop.examples.pi
org.apache.hadoop.examples.pi.math
org.apache.hadoop.examples.terasort
org.apache.hadoop.filecache
org.apache.hadoop.fs
org.apache.hadoop.fs.adl
org.apache.hadoop.fs.adl.oauth2

All Classes

AbfsHttpConstants
AbfsRestOperationException
AbstractCounters
AbstractDelegationTokenIdentifier
AbstractDelegationTokenSecretManager
AbstractDNSToSwitchMapping
AbstractEvent
AbstractFileSystem
AbstractLaunchableService
AbstractLivelinessMonitor
AbstractMapWritable
AbstractMetric
AbstractService
AccessControlException
AccessControlList
AccessRequest
AccessTokenProvider
AccessTokenTimer
AclEntry
AclEntryScope
AclEntryType
AclStatus
AddingCompositeService
AddressTypes
Adl
AdlConfKeys
AdlFileSystem
AdminSecurityInfo
AggregatedLogFormat
AggregatedLogFormat.LogKey
AggregatedLogFormat.LogReader
AHSClient
AHSProxy
AllocateRequest
AllocateRequest.AllocateRequestBuilder
AllocateResponse

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

Hierarchy For Package org.apache.hadoop.mapreduce

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.**Object**
  - org.apache.hadoop.mapreduce.counters.**AbstractCounters**<C,G> (implements java.lang.Iterable<T>, org.apache.hadoop.io.Writable)
    - org.apache.hadoop.mapreduce.**Counters**
  - org.apache.hadoop.mapreduce.**Cluster**
  - org.apache.hadoop.mapreduce.**ClusterMetrics** (implements org.apache.hadoop.io.Writable)
  - org.apache.hadoop.mapreduce.**ID** (implements org.apache.hadoop.io.WritableComparable<T>)
    - org.apache.hadoop.mapred.**ID**
      - org.apache.hadoop.mapreduce.**JobID** (implements java.lang.Comparable<T>)
      - org.apache.hadoop.mapreduce.**TaskAttemptID**
      - org.apache.hadoop.mapreduce.**TaskID**
  - org.apache.hadoop.mapreduce.**InputFormat**<K,V>
  - org.apache.hadoop.mapreduce.**InputSplit**
  - org.apache.hadoop.mapreduce.task.JobContextImpl (implements org.apache.hadoop.mapreduce.JobContext)
    - org.apache.hadoop.mapreduce.**Job** (implements java.lang.AutoCloseable, org.apache.hadoop.mapreduce.JobContext)
  - org.apache.hadoop.mapreduce.**JobStatus** (implements java.lang.Cloneable, org.apache.hadoop.io.Writable)
  - org.apache.hadoop.mapreduce.**Mapper**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  - org.apache.hadoop.mapreduce.**MarkableIterator**<VALUE>
  - org.apache.hadoop.mapreduce.**OutputCommitter**
  - org.apache.hadoop.mapreduce.**OutputFormat**<K,V>
  - org.apache.hadoop.mapreduce.**Partitioner**<KEY,VALUE>
  - org.apache.hadoop.mapreduce.**QueueAclsInfo** (implements org.apache.hadoop.io.Writable)
  - org.apache.hadoop.mapreduce.**QueueInfo** (implements org.apache.hadoop.io.Writable)
  - org.apache.hadoop.mapreduce.**RecordReader**<KEYIN,VALUEIN> (implements java.io.Closeable)
  - org.apache.hadoop.mapreduce.**RecordWriter**<K,V>
  - org.apache.hadoop.mapreduce.**Reducer**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  - org.apache.hadoop.mapreduce.**TaskCompletionEvent** (implements org.apache.hadoop.io.Writable)
  - org.apache.hadoop.mapreduce.**TaskTrackerInfo** (implements org.apache.hadoop.io.Writable)

Interface Hierarchy

- java.lang.**Iterable**<T>
  - org.apache.hadoop.mapreduce.counters.**CounterGroupBase**<T> (also extends org.apache.hadoop.io.Writable)
    - org.apache.hadoop.mapreduce.**CounterGroup**
- org.apache.hadoop.mapreduce.MRJobConfig
  - org.apache.hadoop.mapreduce.**JobContext**
    - org.apache.hadoop.mapreduce.**TaskAttemptContext** (also extends org.apache.hadoop.util.Progressable)
      - org.apache.hadoop.mapreduce.**TaskInputOutputContext**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
        - org.apache.hadoop.mapreduce.**MapContext**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
        - org.apache.hadoop.mapreduce.**ReduceContext**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
- org.apache.hadoop.util.**Progressable**
  - org.apache.hadoop.mapreduce.**TaskAttemptContext** (also extends org.apache.hadoop.mapreduce.JobContext)
    - org.apache.hadoop.mapreduce.**TaskInputOutputContext**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
      - org.apache.hadoop.mapreduce.**MapContext**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
      - org.apache.hadoop.mapreduce.**ReduceContext**<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
- org.apache.hadoop.io.**Writable**
  - org.apache.hadoop.mapreduce.**Counter**
  - org.apache.hadoop.mapreduce.counters.**CounterGroupBase**<T> (also extends java.lang.Iterable<T>)
    - org.apache.hadoop.mapreduce.**CounterGroup**

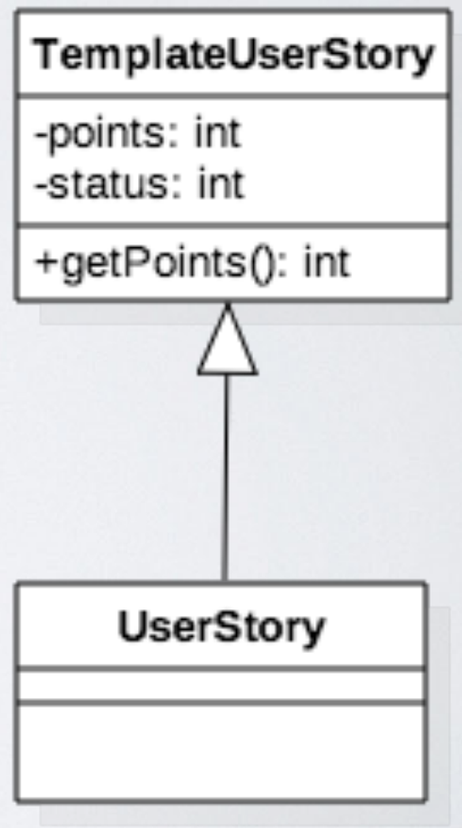
CONCLUSION

- **Understand responsibilities of your classes**
- Understanding of:
  - Association, Aggregation, Composition and Dependency
  - Mapping from UML to Code

DISCLAIMER

- Slides have shown basic understanding
- Deliverables (class diagrams) should have:
  - **attribute** with access level modifiers (public vs private)
  - **methods** with access level modifiers (public vs private)
  - show signature of methods (including the **types**)
  - Interfaces, abstract classes, static classes and attributes

missing methods
and attributes



NEXT LECTURE

- Behavioural modelling → Interaction between objects

(if you are not familiar yet with Taiga,
please make sure you are for the next lecture)