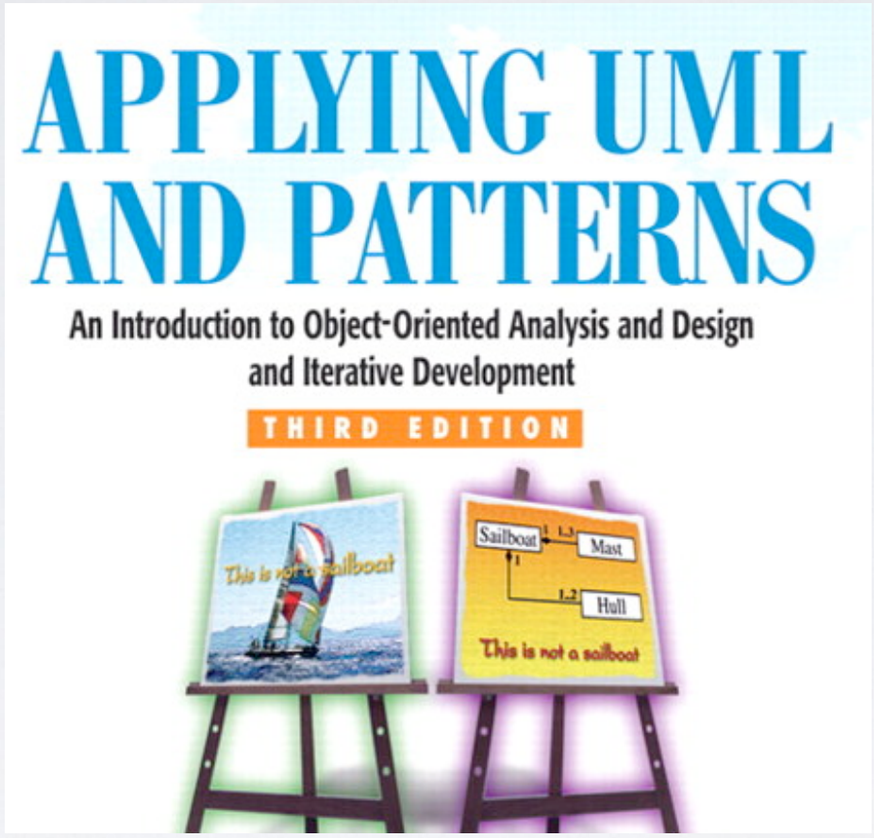
PREVIOUSLY IN ASD…

- Structured modelling
    - UML: Class diagrams
    - Identify responsibilities
        - Generalisation / code re-use
- Behaviour modelling
    - Capture interaction between objects
    - Sequence diagrams vs Communication diagrams

ADVANCED SOFTWARE DESIGN
LECTURE 5
GRASP

Kiko Fernandez

TODAY'S LECTURE



We will discuss and apply the GRASP design patterns.

These provide principles for evaluating and improving designs.

WARM UP

WHAT IS GOOD DESIGN?

GOOD DESIGN:
DESIGN THAT COPES WITH CHANGE.

EMBRACE CHANGE

ASSUME THAT YOUR DESIGN IS IMPERFECT

ROLE OF DESIGN EVALUATION

Improve design – if incorporated into design process.

Provide advice for potential implementors – if done after design.

APPROACHES

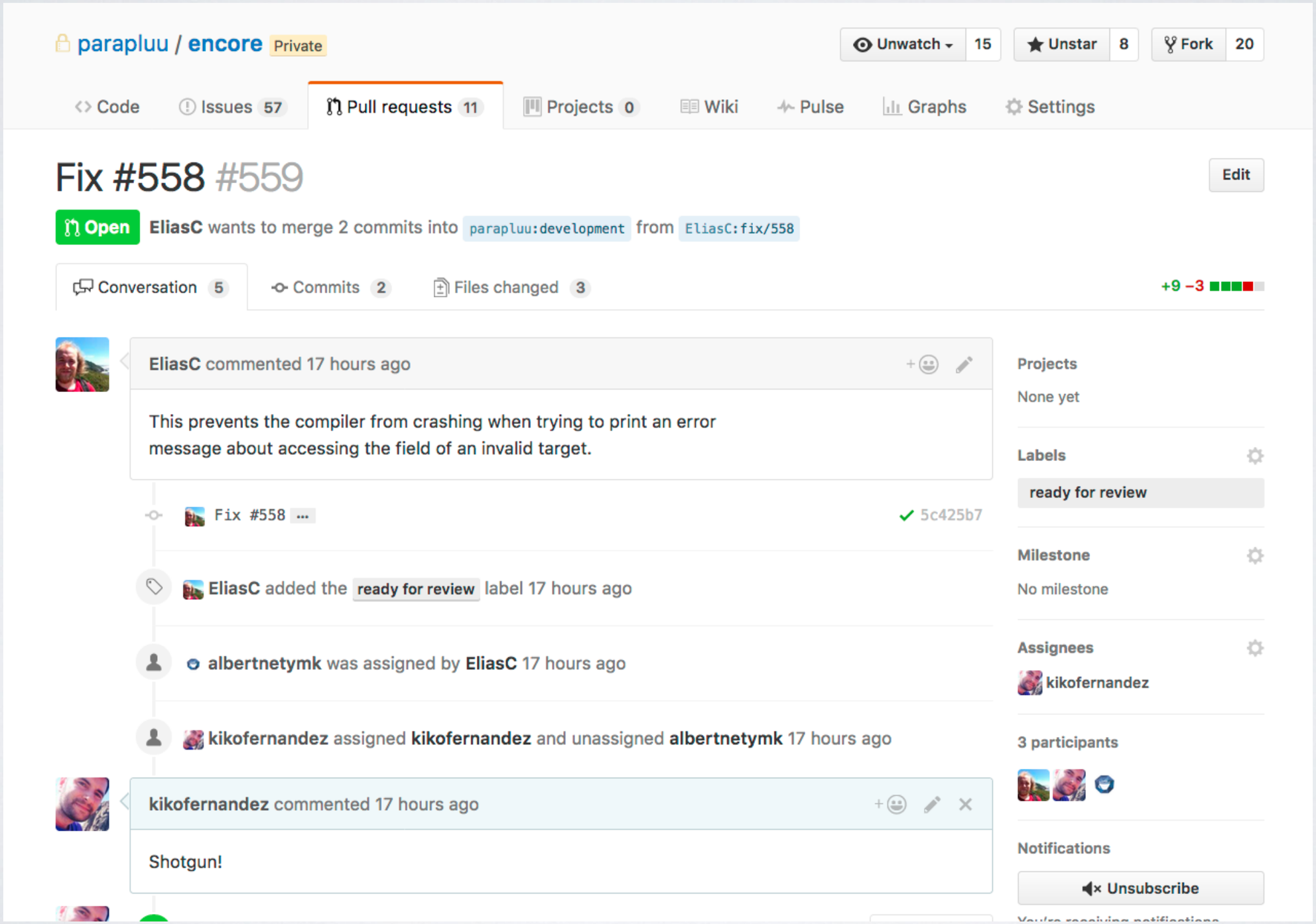
Metrics – class width, hierarchy depth

Heuristics

Checks of pattern usage

Design critiques, similar to art criticism or reviews

CODE REVIEW (GITHUB)



CODE REVIEW (GITHUB)

kikofernandez approved these changes an hour ago — View changes

This fix is simple and will prevent many Encore newcomers from trying to access the attribute of an active object.

There is a suggestion that the committer can leave or take, but the PR looks good.

As a suggestion for future PRs, I think it's better to provide a meaningful and short title instead of the issue, as there is no link in the title and I do not know what I am going to review until I read the description.

`src/types/Typechecker/Typechecker.hs`

```
...  ...  @@ -833,7 +833,7 @@ instance Checkable Expr where
833  833        eTarget <- typecheck target
834  834        let targetType = AST.getType eTarget
835  835        unless (isThisAccess target || isPassiveClassType targetType) $
836       -        tcError $ CannotReadFieldError target
     836  +        tcError $ CannotReadFieldError eTarget
837  837        fdecl <- findField targetType name
```

kikofernandez an hour ago

this looks good. May I suggest to skip the `let` statement and go for:

```
ty <- ftype <$> findField targetType name
```

one less line to maintain and, IMO, it's understandable that you find the field and get its type

EliasC an hour ago

Fixed! Please squash when merging.

Reply...

New changes since you last viewed — View changes

Refactoring — c99f89a

Projects

None yet

Labels

ready for review

Milestone

No milestone

Assignees

kikofernandez

3 participants

Notifications

Unsubscribe

You're receiving notifications because you were assigned.

Lock conversation

WHAT IF I DON'T…

- Do code reviews and/or Create a flexible design

WHAT IF I DON'T…

- Do code reviews and/or Create a flexible design

Facebook case study

REAL-WORLD CASE

- iOs app
  - 400+ devs contributing
  - 16000+ classes
- Design problem?

REAL-WORLD CASE

- iOs app
  - 400+ devs contributing
  - 16000+ classes
- Design problem?

```
_FBPushTokenMuteNotificationsMutationCall.h
_FBQrcodeCreateMutationCall.h
_FBReactionAcornSportsContentSettingsSetDoNotFollowLikedPagesMutationCall.h
_FBReactionAcornSportsContentSettingsSetFollowLikedPagesMutationCall.h
_FBReactionAcornSportsContentSettingsSetProfilesMutationCall.h
_FBReactionAcornSportsContentSettingsSetShouldNotPushNotificationsMutationCall.h
_FBReactionAcornSportsContentSettingsSetShouldPushNotificationsMutationCall.h
_FBReactionAcornTvContentSettingsSetDoNotFollowLikedPagesMutationCall.h
_FBReactionAcornTvContentSettingsSetFollowLikedPagesMutationCall.h
_FBReactionAcornTvContentSettingsSetProfilesMutationCall.h
_FBReactionAcornTvContentSettingsSetShouldNotPushNotificationsMutationCall.h
_FBReactionAcornTvContentSettingsSetShouldPushNotificationsMutationCall.h
_FBReactionAdapterHelperDelegate-Protocol.h
_FBReactionAdapterProfileHelperDelegate-Protocol.h
_FBReactionCardAdapterActionHelper.h
_FBReactionCardAdapterNotificationHelper.h
_FBReactionCardAdapterNotificationHelperDelegate-Protocol.h
_FBReactionCardAdapterProfileHelper.h
_FBReactionCardAdapterURLHelper.h
_FBReactionCardAdapterVideoHelper.h
_FBReactionCardAdapterVideoHelperDelegate-Protocol.h
_FBReactionUnitAdsAfterPartyTipDisableMutationCall.h
_FBReactionUnitUserSettingsDisableUnitTypeMutationCall.h
_FBReactionUnitUserSettingsEnableUnitTypeMutationCall.h
_FBRedspaceStoryViewMutationCall.h
```

http://www.slideshare.net/quellish/simon-whitaker
http://www.darkcoding.net/software/facebooks-code-quality-problem/

REAL-WORLD CASE

- iOs app
  - 400+ devs contributing
  - 16000+ classes
- Design problem?

Git & Xcode

```
_FBPushTokenMuteNotificationsMutationCall.h
_FBQrcodeCreateMutationCall.h
_FBReactionAcornSportsContentSettingsSetDoNotFollowLikedPagesMutationCall.h
_FBReactionAcornSportsContentSettingsSetFollowLikedPagesMutationCall.h
_FBReactionAcornSportsContentSettingsSetProfilesMutationCall.h
_FBReactionAcornSportsContentSettingsSetShouldNotPushNotificationsMutationCall.h
_FBReactionAcornSportsContentSettingsSetShouldPushNotificationsMutationCall.h
_FBReactionAcornTvContentSettingsSetDoNotFollowLikedPagesMutationCall.h
_FBReactionAcornTvContentSettingsSetFollowLikedPagesMutationCall.h
_FBReactionAcornTvContentSettingsSetProfilesMutationCall.h
_FBReactionAcornTvContentSettingsSetShouldNotPushNotificationsMutationCall.h
_FBReactionAcornTvContentSettingsSetShouldPushNotificationsMutationCall.h
_FBReactionAdapterHelperDelegate-Protocol.h
_FBReactionAdapterProfileHelperDelegate-Protocol.h
_FBReactionCardAdapterActionHelper.h
_FBReactionCardAdapterNotificationHelper.h
_FBReactionCardAdapterNotificationHelperDelegate-Protocol.h
_FBReactionCardAdapterProfileHelper.h
_FBReactionCardAdapterURLHelper.h
_FBReactionCardAdapterVideoHelper.h
_FBReactionCardAdapterVideoHelperDelegate-Protocol.h
_FBReactionUnitAdsAfterPartyTipDisableMutationCall.h
_FBReactionUnitUserSettingsDisableUnitTypeMutationCall.h
_FBReactionUnitUserSettingsEnableUnitTypeMutationCall.h
_FBRedspaceStoryViewMutationCall.h
```

http://www.slideshare.net/quellish/simon-whitaker
http://www.darkcoding.net/software/facebooks-code-quality-problem/

REAL-WORLD CASE

- iOs app
  - 400+ devs contributing
  - 16000+ classes
- Design problem?

Git & Xcode

```
_FBPushTokenMuteNotificationsMutationCall.h
_FBQrcodeCreateMutationCall.h
_FBReactionAcornSportsContentSettingsSetDoNotFollowLikedPagesMutationCall.h
_FBReactionAcornSportsContentSettingsSetFollowLikedPagesMutationCall.h
_FBReactionAcornSportsContentSettingsSetProfilesMutationCall.h
_FBReactionAcornSportsContentSettingsSetShouldNotPushNotificationsMutationCall.h
_FBReactionAcornSportsContentSettingsSetShouldPushNotificationsMutationCall.h
_FBReactionAcornTvContentSettingsSetDoNotFollowLikedPagesMutationCall.h
_FBReactionAcornTvContentSettingsSetFollowLikedPagesMutationCall.h
_FBReactionAcornTvContentSettingsSetProfilesMutationCall.h
_FBReactionAcornTvContentSettingsSetShouldNotPushNotificationsMutationCall.h
_FBReactionAcornTvContentSettingsSetShouldPushNotificationsMutationCall.h
_FBReactionAdapterHelperDelegate-Protocol.h
_FBReactionAdapterProfileHelperDelegate-Protocol.h
_FBReactionCardAdapterActionHelper.h
_FBReactionCardAdapterNotificationHelper.h
_FBReactionUnitAdsAfterPartyTipDisableMutationCall.h
_FBReactionUnitUserSettingsDisableUnitTypeMutationCall.h
_FBReactionUnitUserSettingsEnableUnitTypeMutationCall.h
_FBRedspaceStoryViewMutationCall.h
```

Anti-pattern:
Functional Decomposition

http://www.slideshare.net/quellish/simon-whitaker
http://www.darkcoding.net/software/facebooks-code-quality-problem/

GRASP

General **R**esponsibility **A**ssignment **S**oftware **P**atterns or Principles (GRASP)

A collection of patterns/principles for achieving good design – patterns of assigning responsibility.

Refer to *software objects* not *domain objects*.

LIST OF GRASP PATTERNS

Low coupling

High cohesion

Creator

Information expert

Controller

Polymorphism

Indirection

Pure fabrication

Protected variations

LOW COUPLING

COUPLING

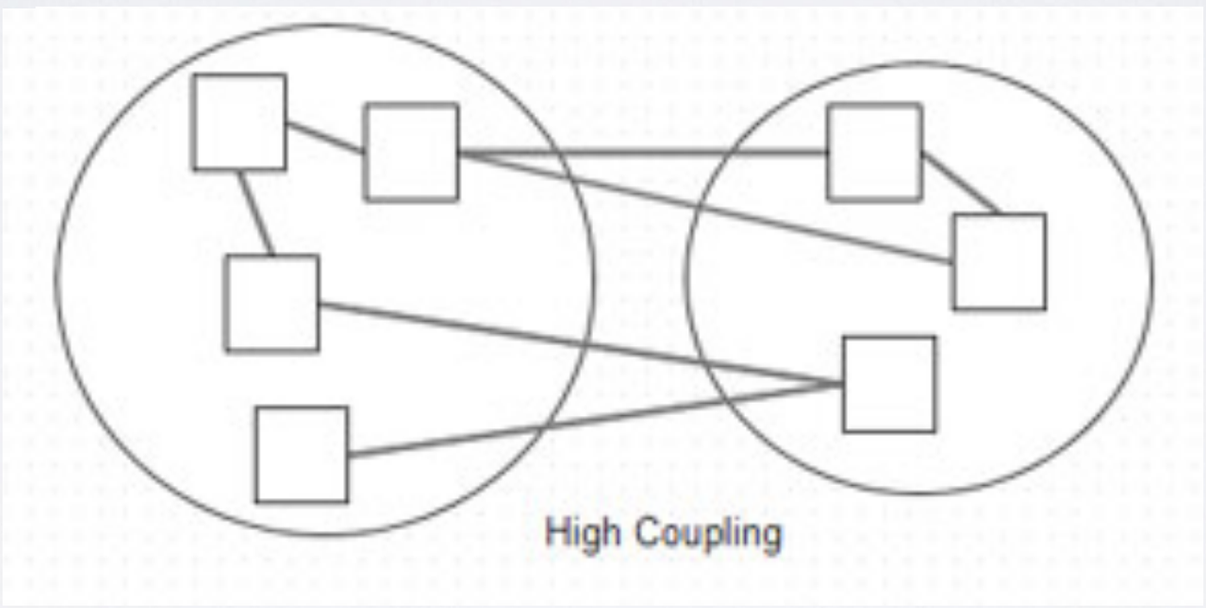
- Measure **how strongly one element is connected to**, has knowledge of or relies on **other elements**.

- An element with weak low (or weak) coupling is not dependent on too many other elements.

LOW COUPLING

Problem: How to reduce the impact of change?

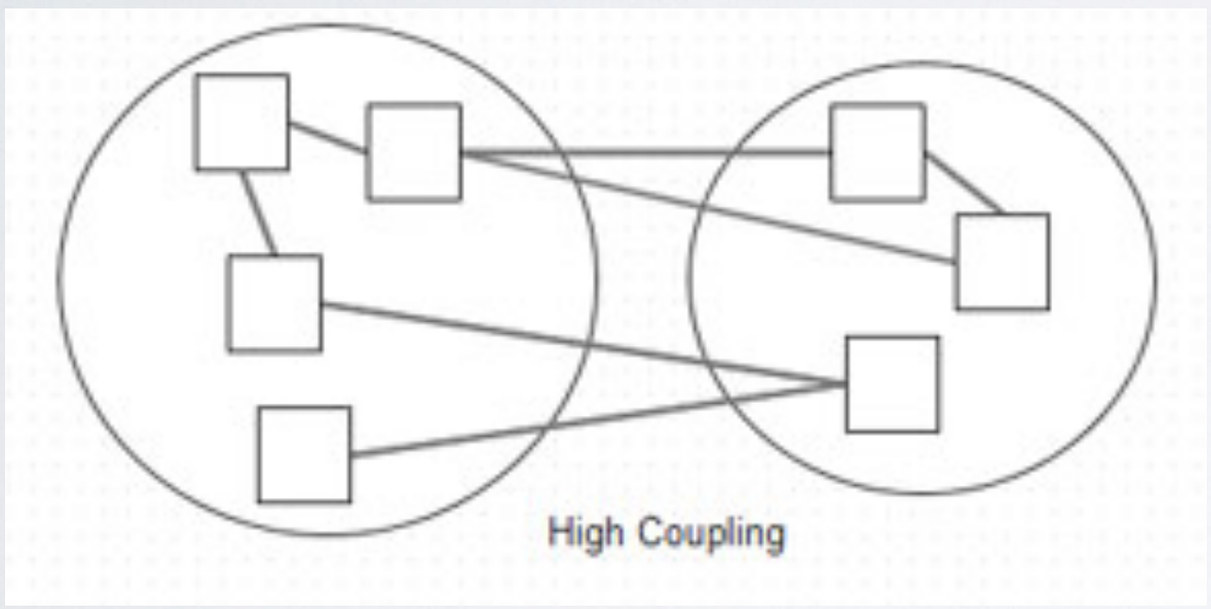
Advice: Assign responsibilities so that (unnecessary) coupling remains low.

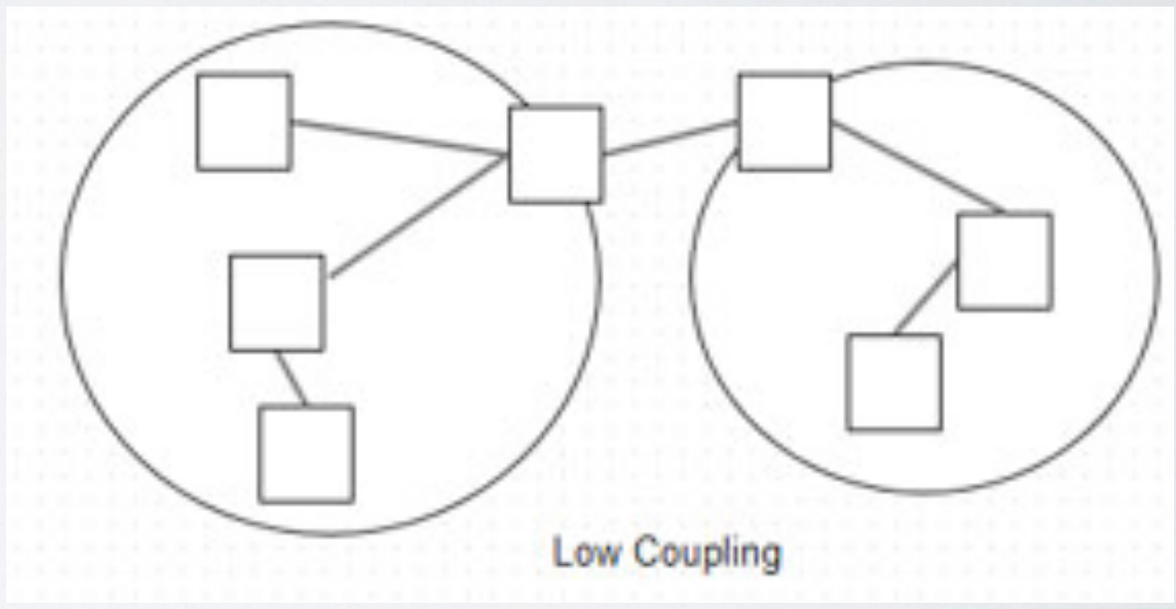


LOW COUPLING

Problem: How to reduce the impact of change?

Advice: Assign responsibilities so that (unnecessary) coupling remains low.

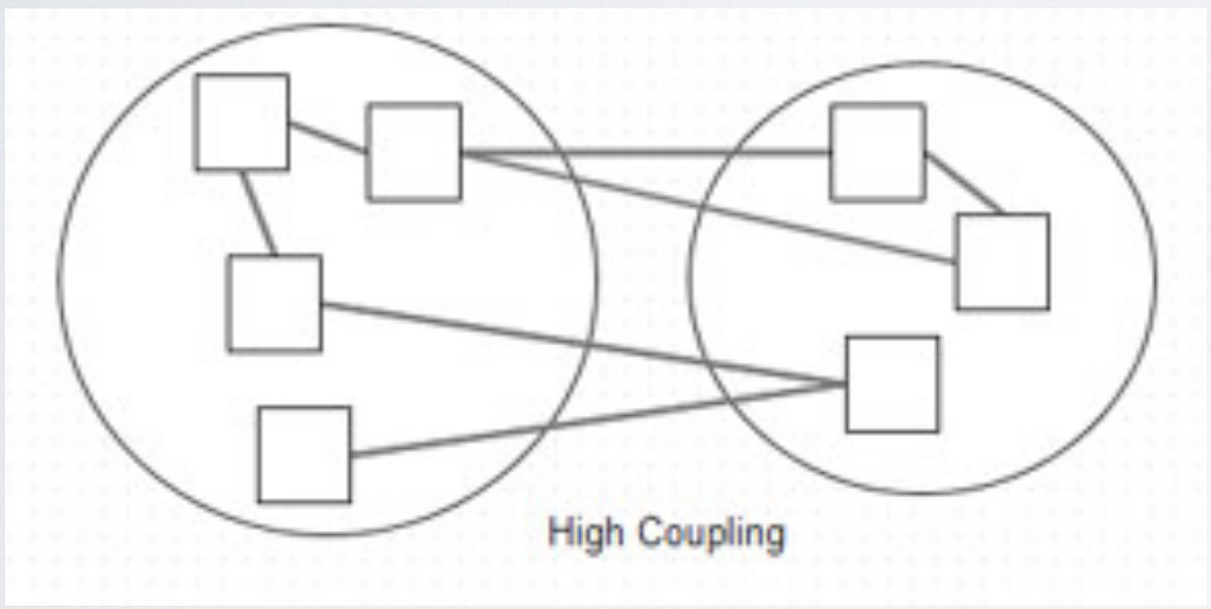


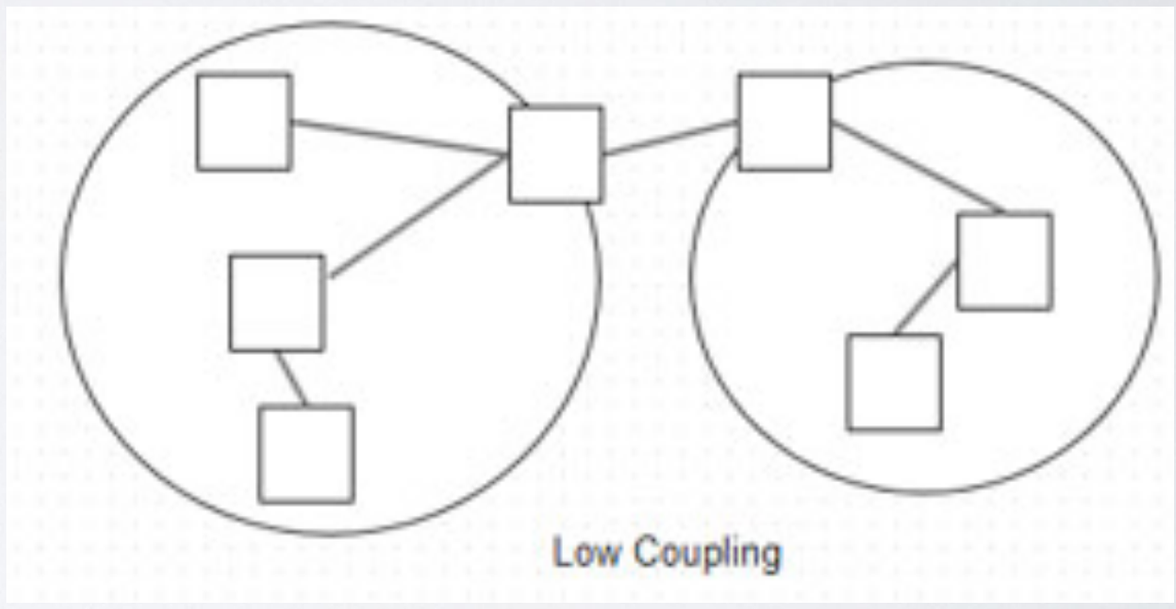


LOW COUPLING

Problem: How to reduce the impact of change?

Advice: Assign responsibilities so that (unnecessary) coupling remains low.





may add additional layers of indirection.

WHEN ARE TWO CLASSES COUPLED?

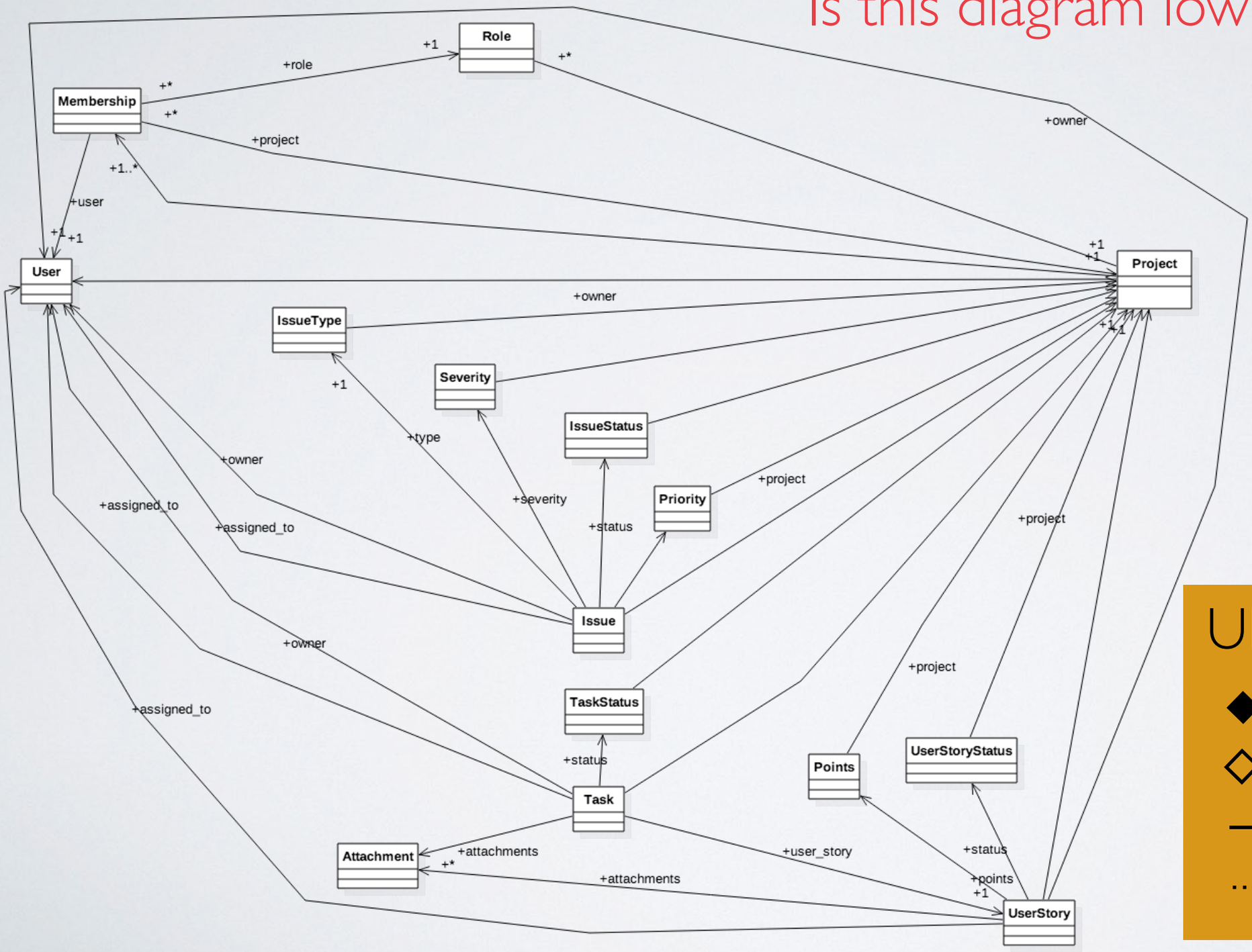
- Common forms of coupling from *TypeX* to *TypeY*:
  - *TypeX* **has an attribute** that refers to a *TypeY* instance.
  - A *TypeX* **object calls on services** of a *TypeY* object.
  - *TypeX* **has a method that references an instance** of *TypeY* (parameter, local variable, return type).
  - *TypeX* is **a direct or indirect subclass** of *TypeY*.
  - *TypeY* **is an interface** and *TypeX* **implements that interface.**

SOLUTION

- Assign responsibility so that coupling remains low.
- Use this principle to evaluate alternatives.

EXAMPLE

Is this diagram low or high coupled?



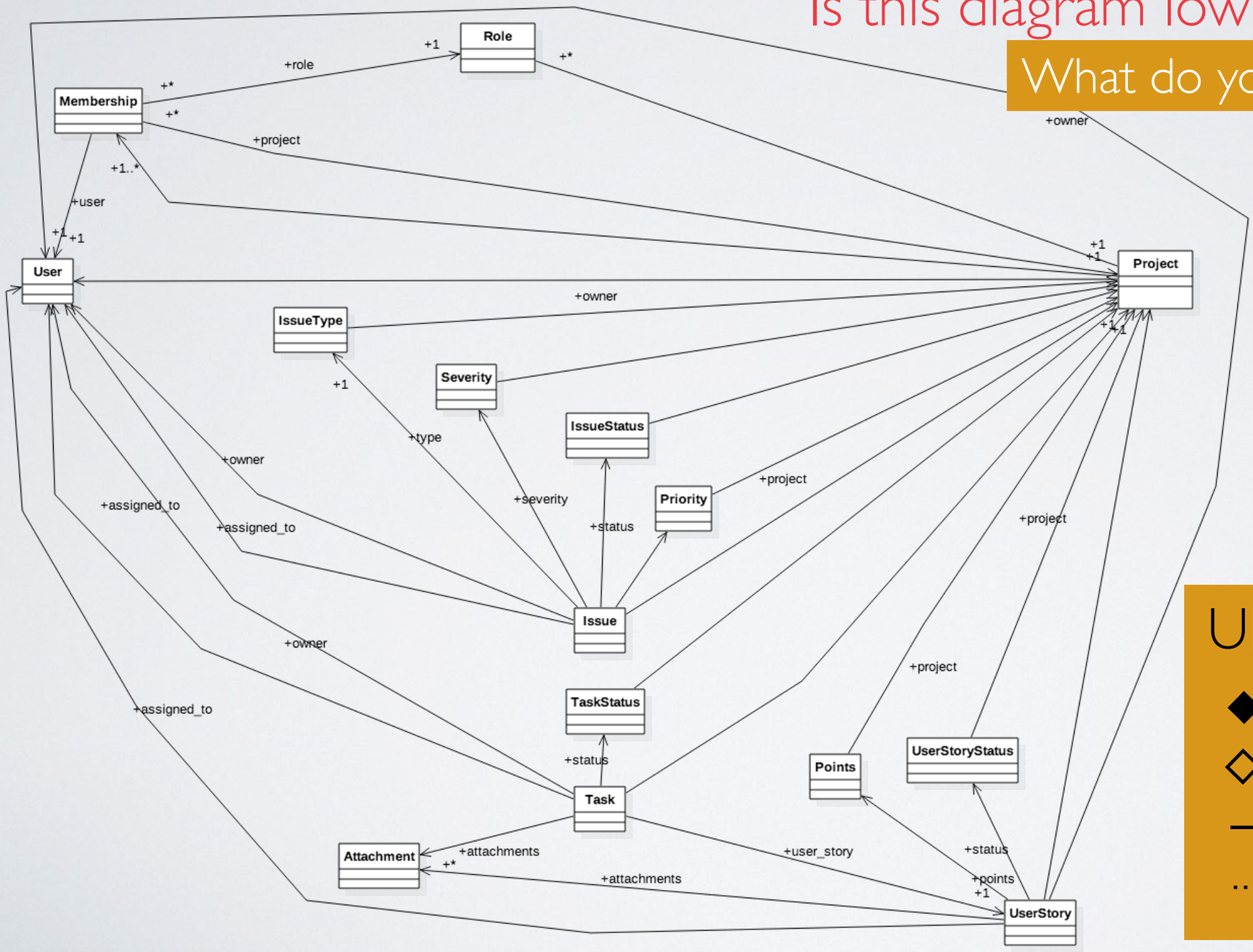
UML Reminder

- Composition
- Aggregation
- Association
- Dependency

EXAMPLE

Is this diagram low or high coupled?

What do you observe?



UML Reminder

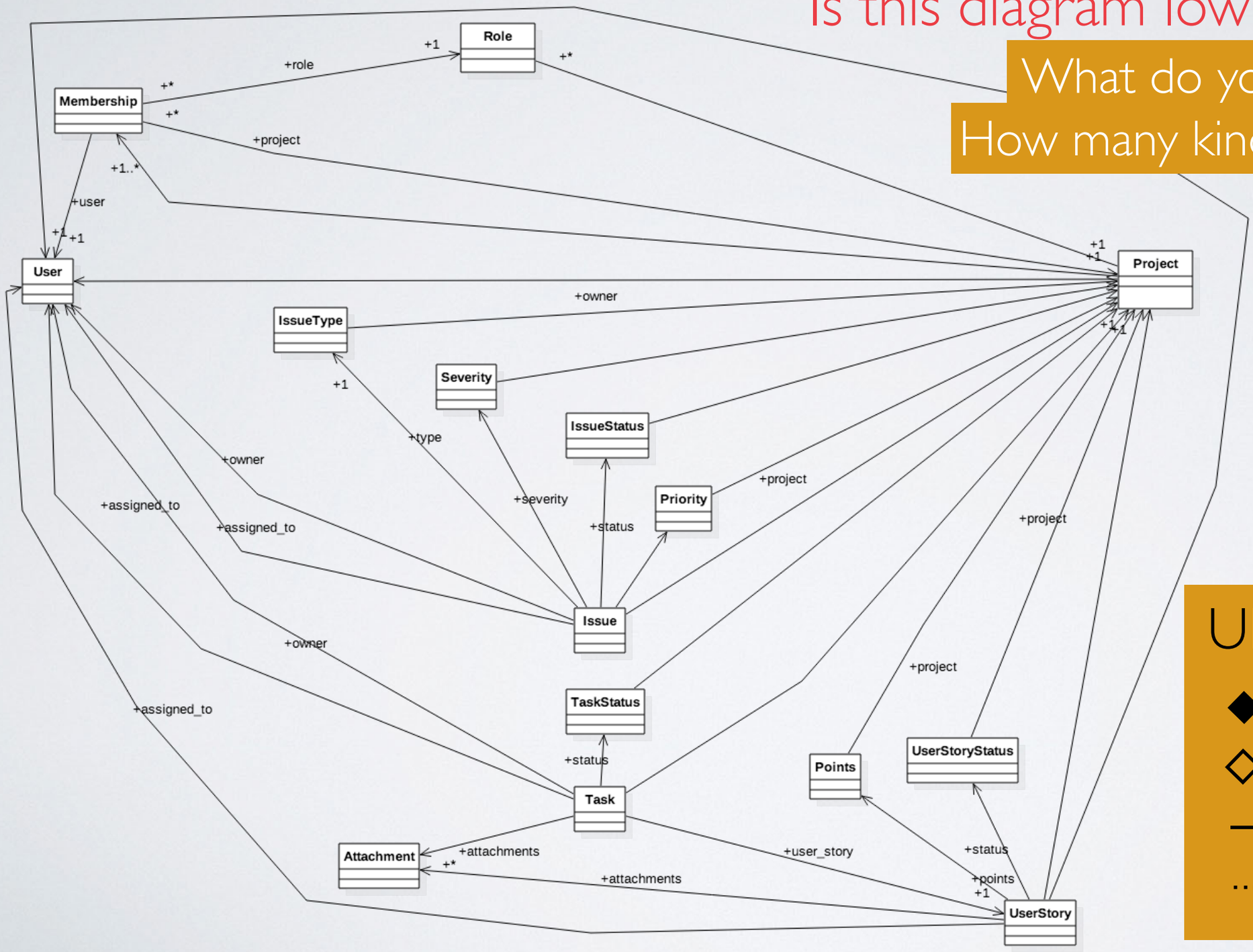
- Composition
- Aggregation
- Association
- Dependency

EXAMPLE

Is this diagram low or high coupled?

What do you observe?

How many kinds of relations?



UML Reminder

- Composition
- Aggregation
- Association
- Dependency

Calculate total of an order

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}
```

Calculate total of an order

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}
```

Calculate total of an order

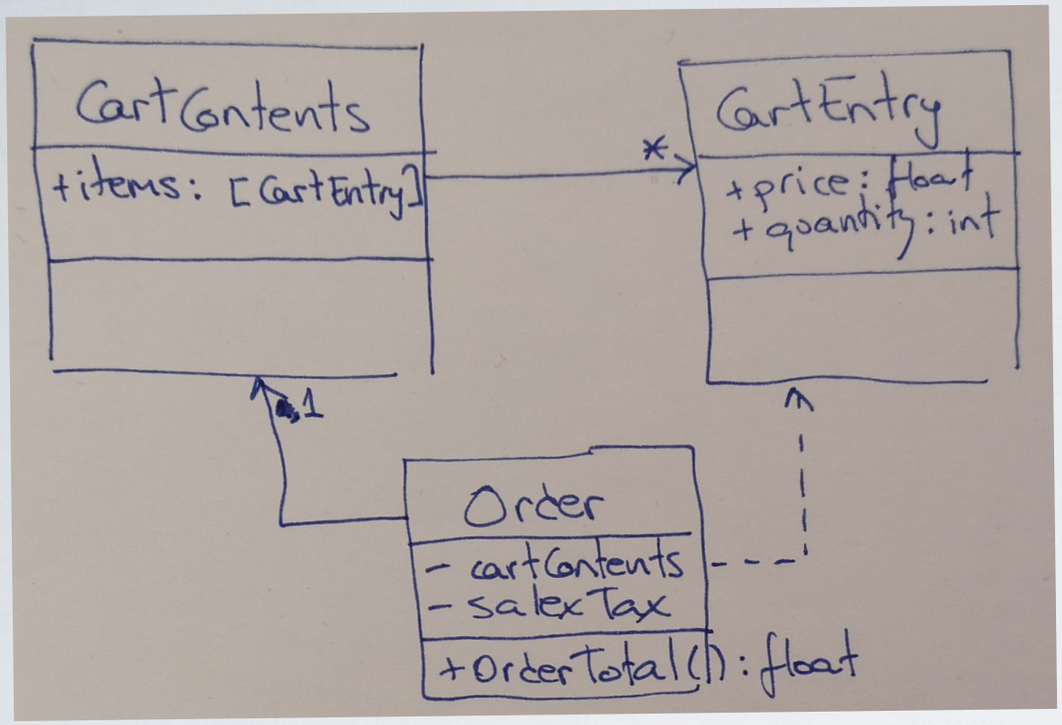
```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}
```

```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.Length; i++)
        {
            cartTotal += cart.items[i].Price * cart.items[i].Quantity;
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```

Calculate total of an order

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}
```

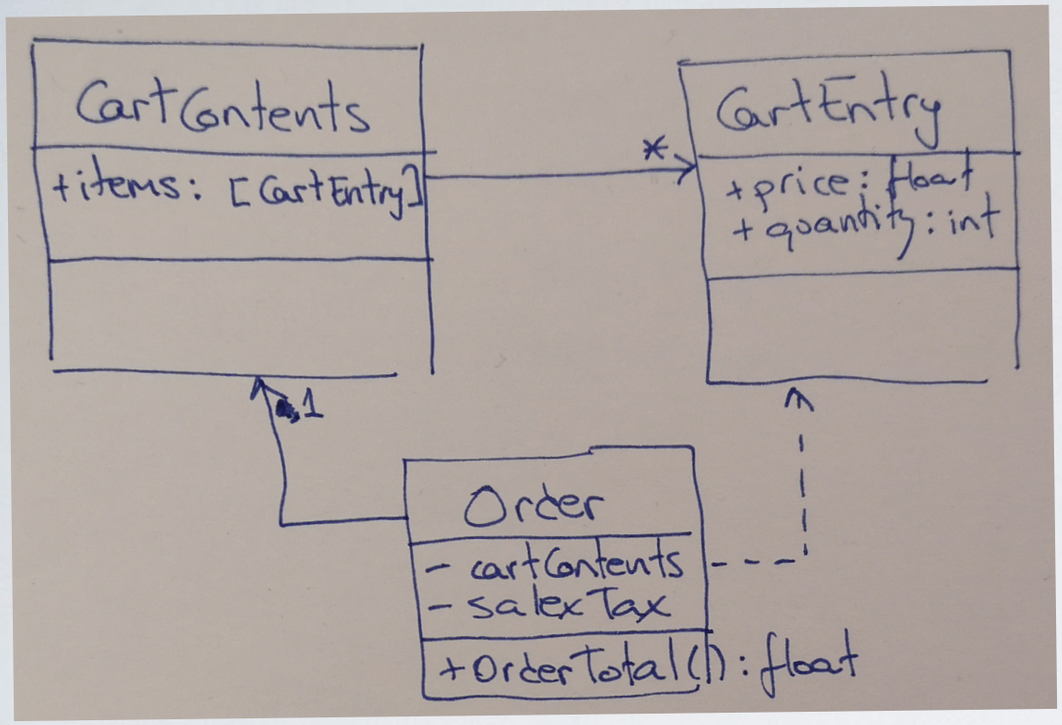
```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.Length; i++)
        {
            cartTotal += cart.items[i].Price * cart.items[i].Quantity;
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```



Calculate total of an order

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}
```

```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.Length; i++)
        {
            cartTotal += cart.items[i].Price * cart.items[i].Quantity;
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```



Calculate total of an order

OrderTotal method (and thus the Order class) depends on the implementation details of the CartContents and the CartEntry classes

(source)

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}
```

```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.Length; i++)
        {
            cartTotal += cart.items[i].Price * cart.items[i].Quantity;
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```

Calculate total of an order

we could change the implementation of any of these classes without having to change the other classes

```
public class CartEntry
{
    private float Price;
    private int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}
```

```
public class CartEntry
{
    private float Price;
    private int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}
```

Calculate total of an order

we could change the implementation of any of these classes without having to change the other classes

Calculate total of an order

we could change the implementation of any of these classes without having to change the other classes

```
public class CartEntry
{
    private float Price;
    private int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}
```

```
public class CartContents
{
    private CartEntry[] items;

    public float GetCartItemsTotal()
    {
        float cartTotal = 0;
        foreach (CartEntry item in items)
        {
            cartTotal += item.GetLineItemTotal();
        }
        return cartTotal;
    }
}
```

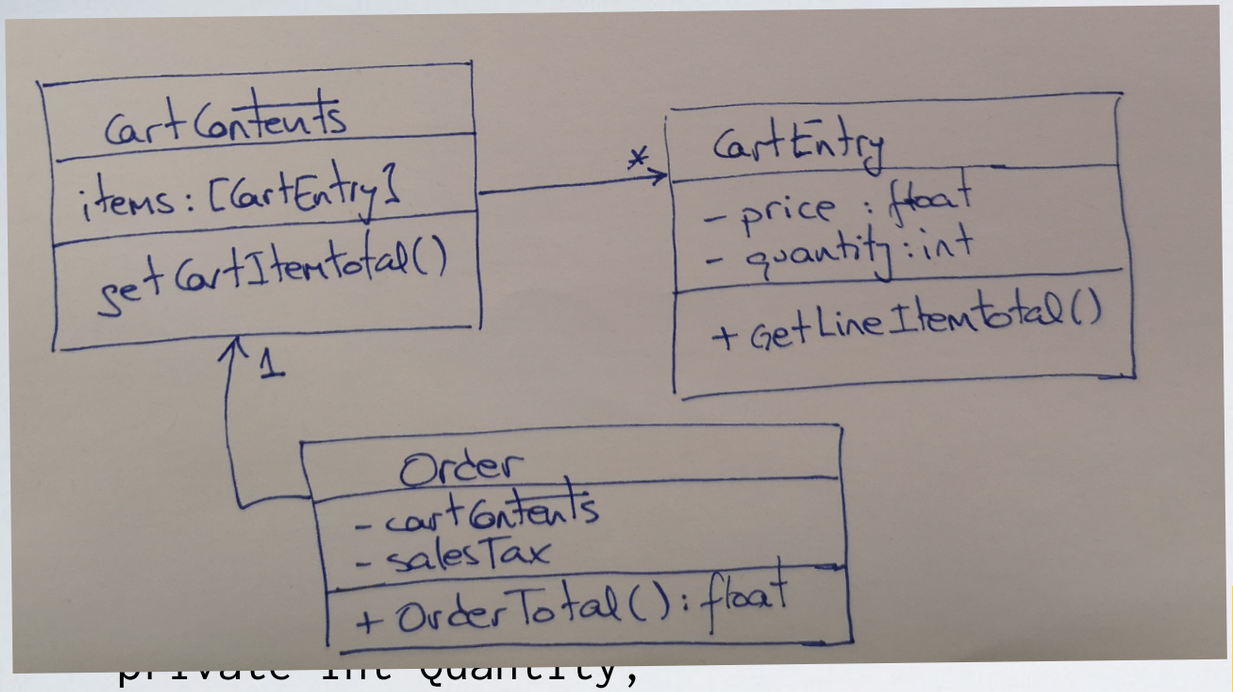
Calculate total of an order

we could change the implementation of any of these classes without having to change the other classes

```
public class CartEntry
{
    private float Price;
    private int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}
```

```
public class CartContents
{
    private CartEntry[] items;

    public float GetCartItemsTotal()
    {
        float cartTotal = 0;
        foreach (CartEntry item in items)
        {
            cartTotal += item.GetLineItemTotal();
        }
        return cartTotal;
    }
}
```

```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        return cart.GetCartItemsTotal() *
               (1.0f + salesTax);
    }
}
```




```
    private int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}
```

```
public class CartContents
{
    private CartEntry[] items;

    public float GetCartItemsTotal()
    {
        float cartTotal = 0;
        foreach (CartEntry item in items)
        {
            cartTotal += item.GetLineItemTotal();
        }
        return cartTotal;
    }
}
```

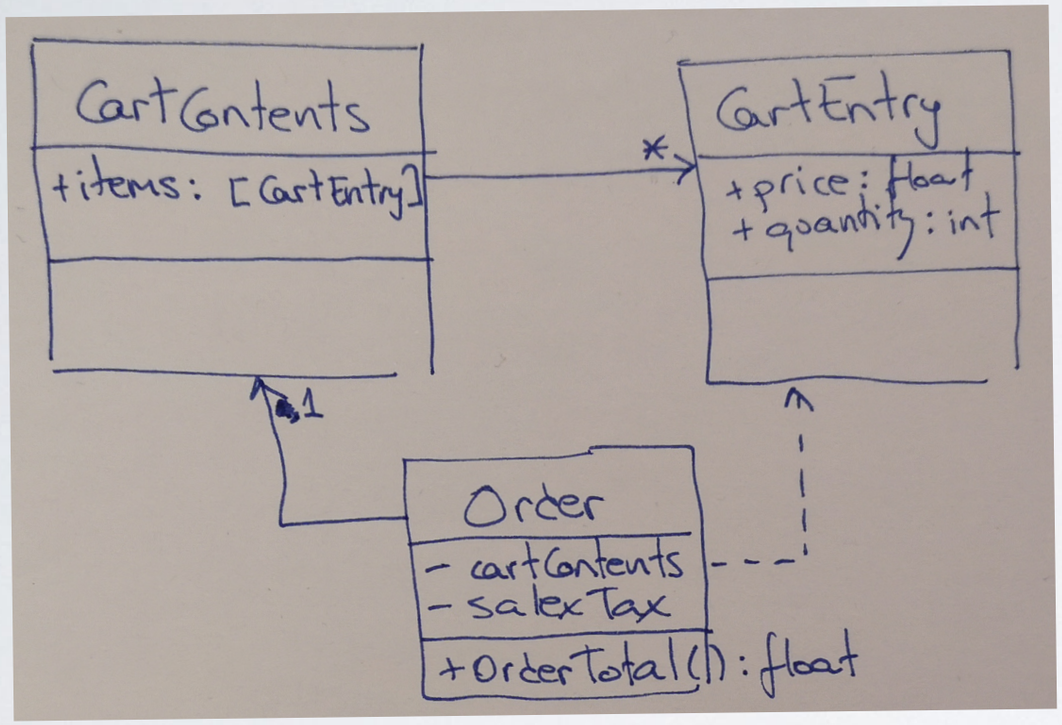
Calculate total of an order

we could change the implementation of any of these classes without having to change the other classes

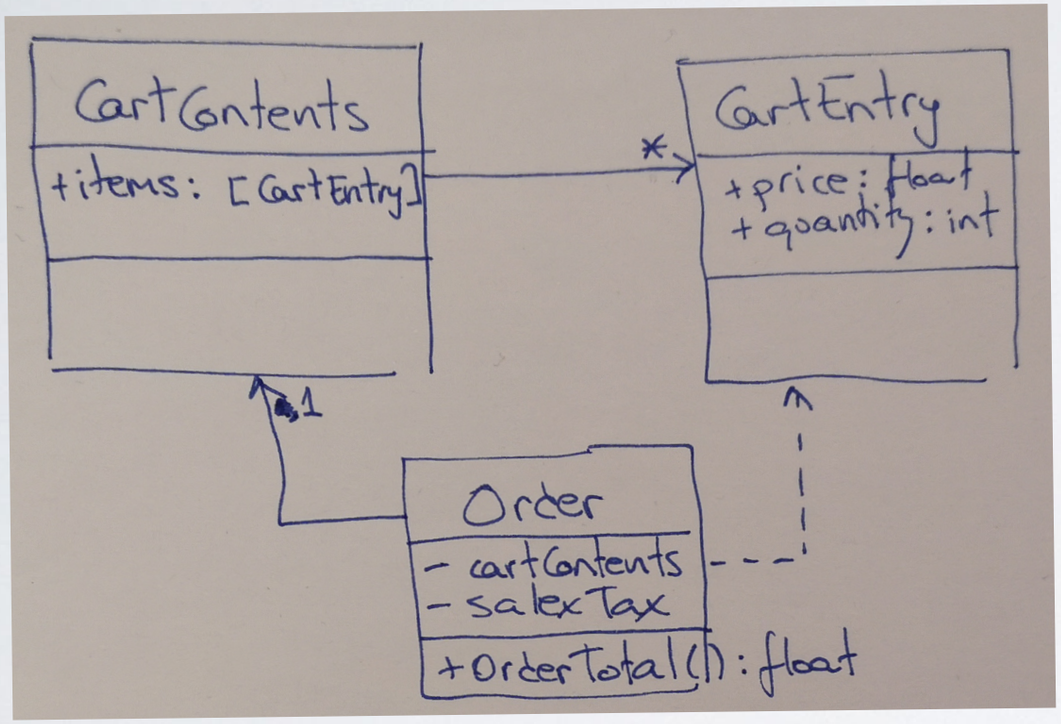
```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        return cart.GetCartItemsTotal() *
               (1.0f + salesTax);
    }
}
```

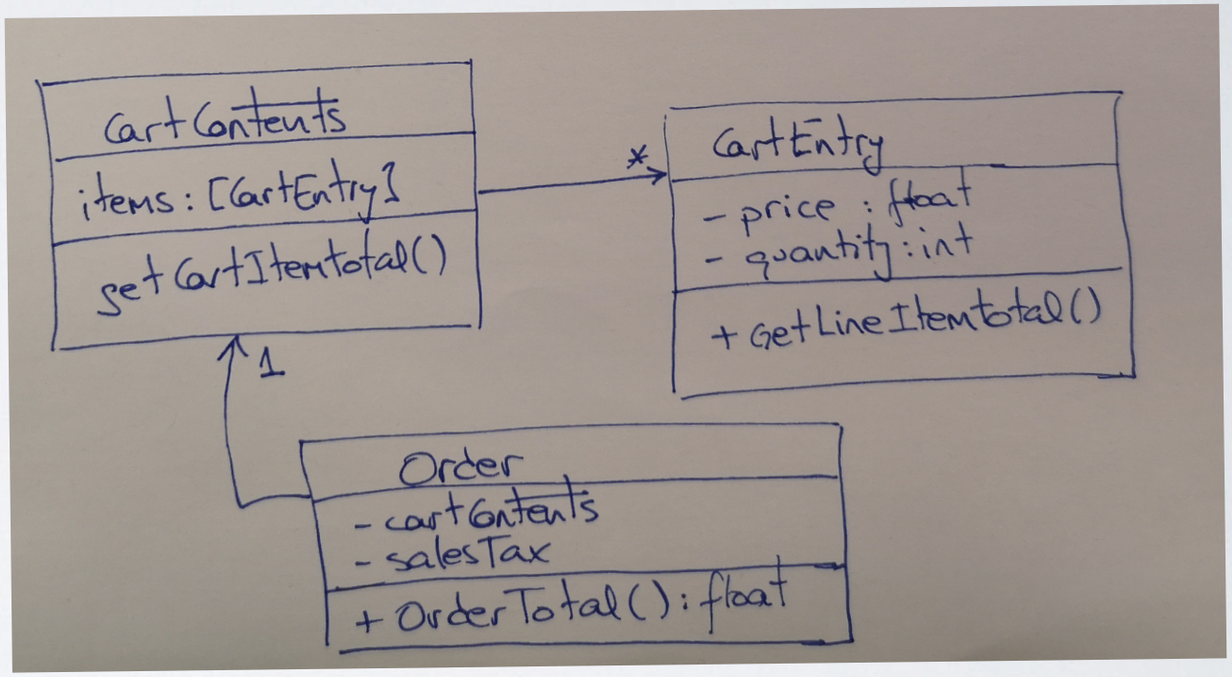
Comparison

Comparison


CartContents
+items: [CartEntry]
*
CartEntry
+price: float
+quantity: int
1
Order
- cartContents
- salexTax
+OrderTotal(): float

Comparison


CartContents
+items: [CartEntry]
CartEntry
+price: float
+quantity: int
*
1
Order
- cartContents
- salexTax
+OrderTotal(): float


CartContents
items: [CartEntry]
setCartItemtotal()
CartEntry
- price : float
- quantity: int
+ getLineItemtotal()
*
1
Order
- cartContents
- salesTax
+ OrderTotal(): float

DISCUSSION.

- **Low coupling encourages designs to be more independent**, which reduces the impact of change.

- **Needs to be considered with other patterns** such as Information Expert (later) and High Cohesion.

- **Subclassing increases coupling**

- High coupling to stable "global" objects is not problematic – to Java libraries such as *java.util.*

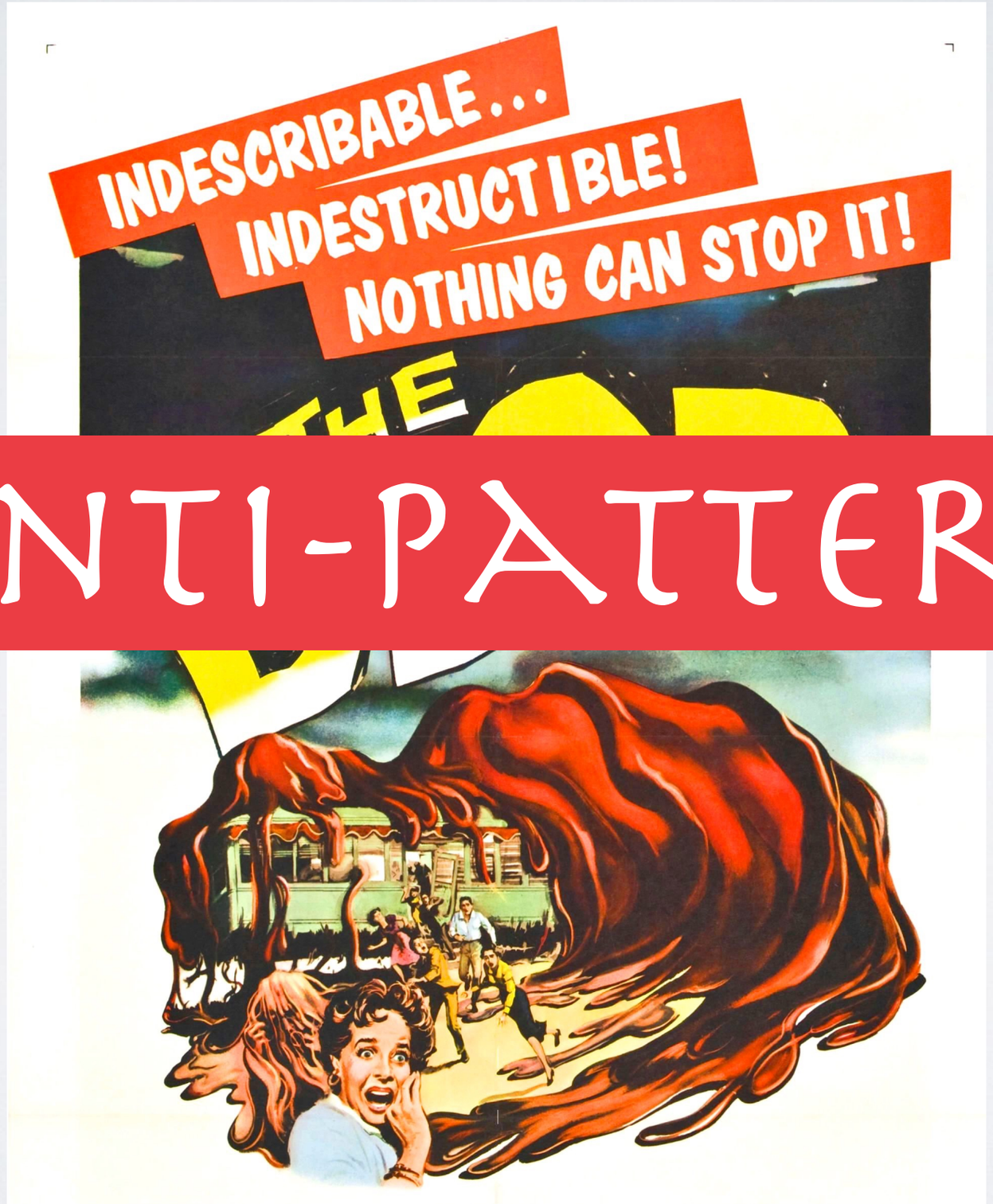
- **Do not consider patterns in isolation**

PICK YOUR BATTLES

- The **problem** is not high coupling per se; it is **high coupling to *unstable* elements**.
- Designers can *future proof* various aspects of the system using lower coupling and encapsulation, but there needs to be good motivation.
- Focus on points of realistic high instability and evolution.

HIGH COUPLING

HIGH COUPLING



ANTI-PATTERN

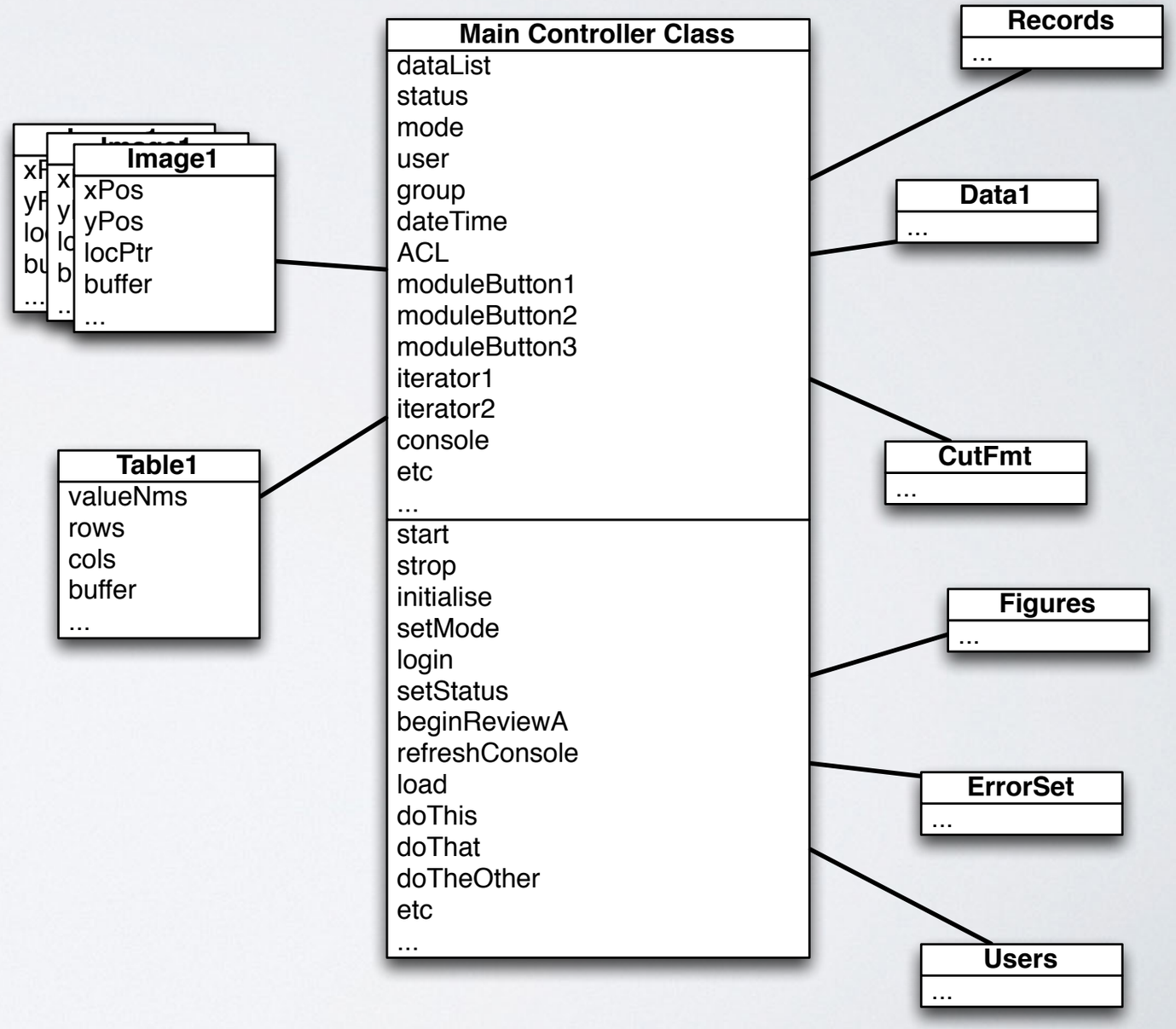
ANTI-PATTERN: THE BLOB

A single class monopolises most of the functionality.

Large class, unrelated attributes and functionality.

Single controller + simple data-object classes.

Migrated (unrefactored) legacy application



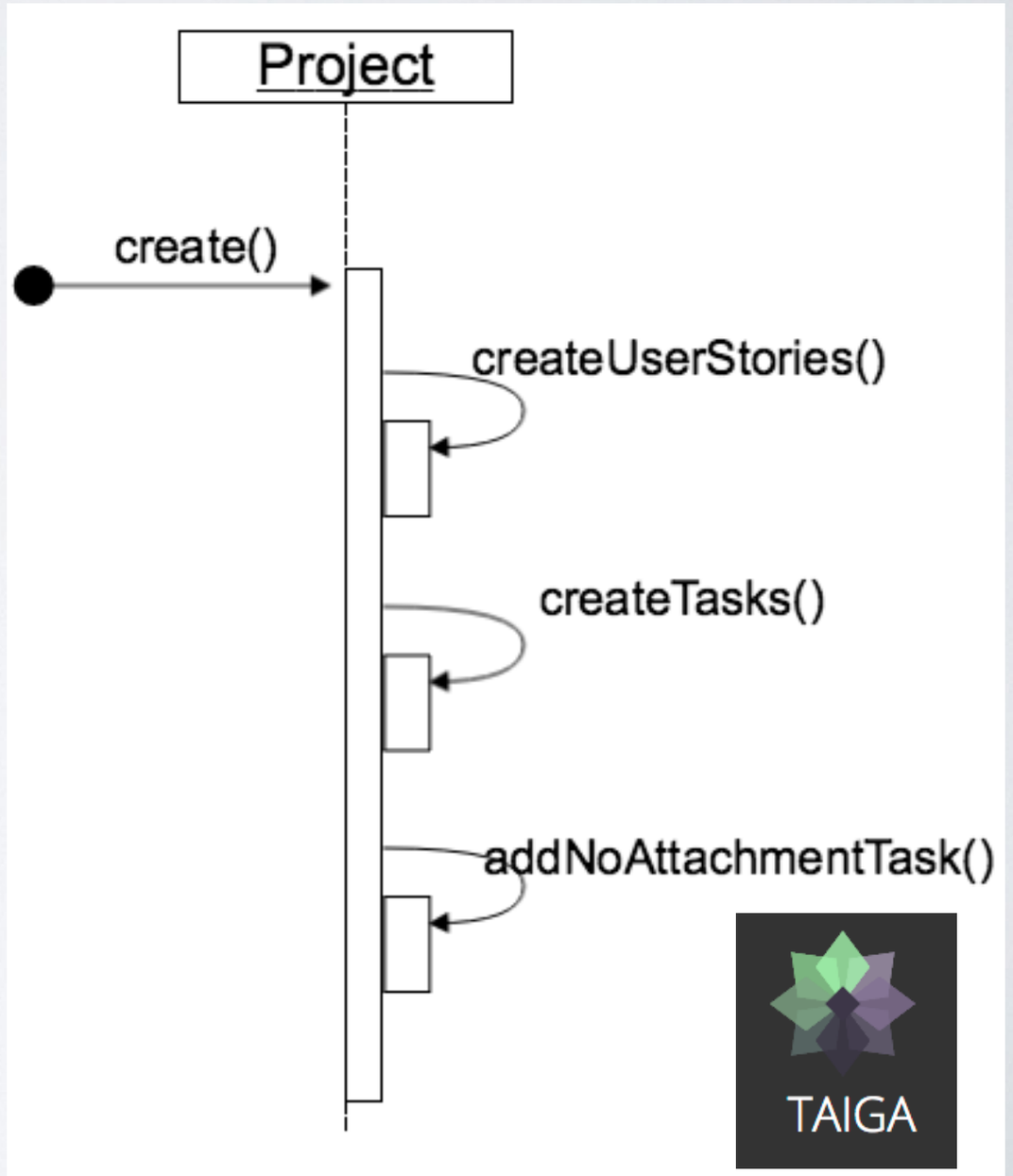
HIGH COHESION

COHESION

Problem: How to **keep objects focussed**, understandable, and manageable, and as a side effect, support Low Coupling?

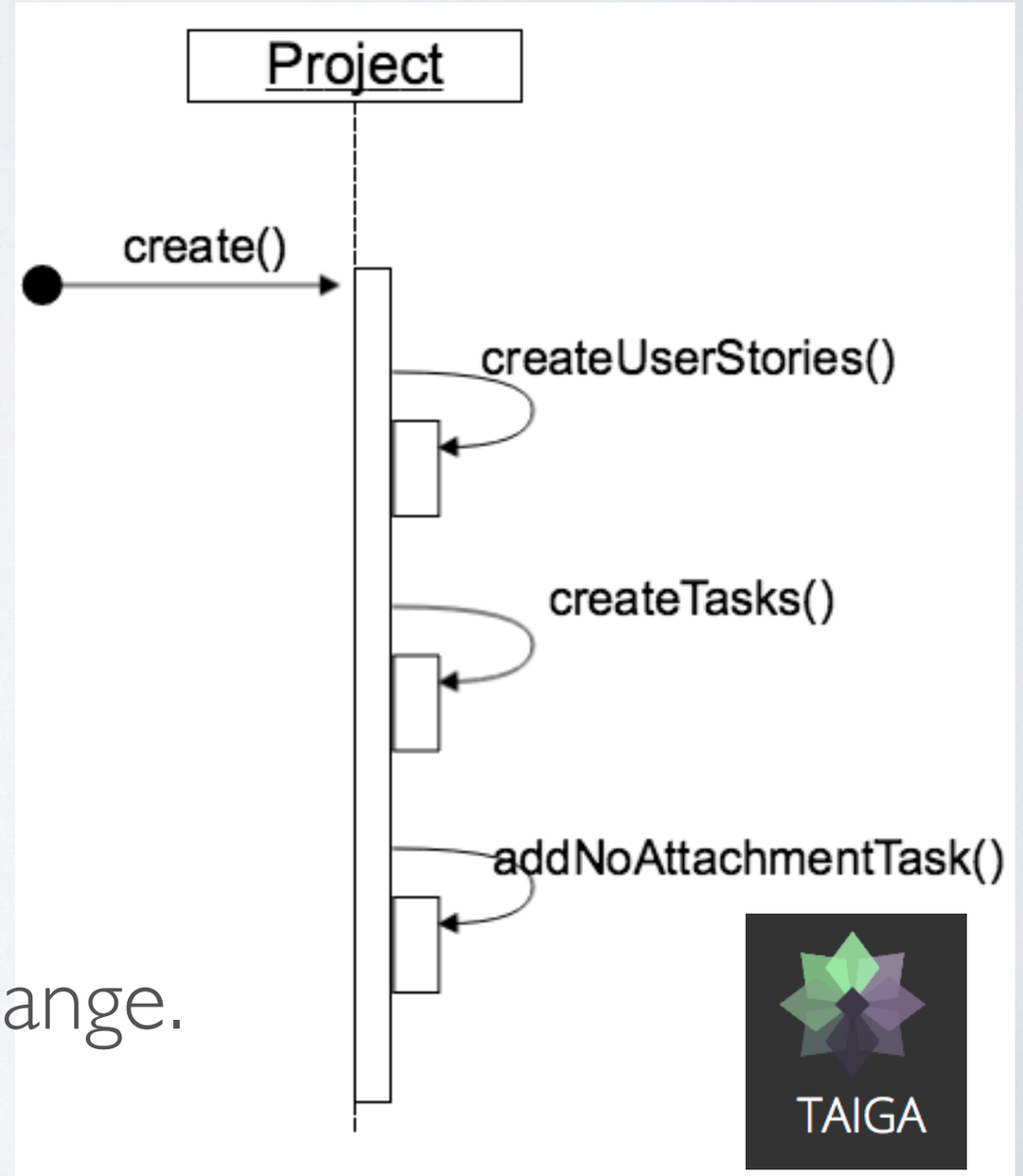
COHESION

Problem: How to **keep objects focussed**, understandable, and manageable, and as a side effect, support Low Coupling?



COHESION

Problem: How to **keep objects focussed**, understandable, and manageable, and as a side effect, support Low Coupling?



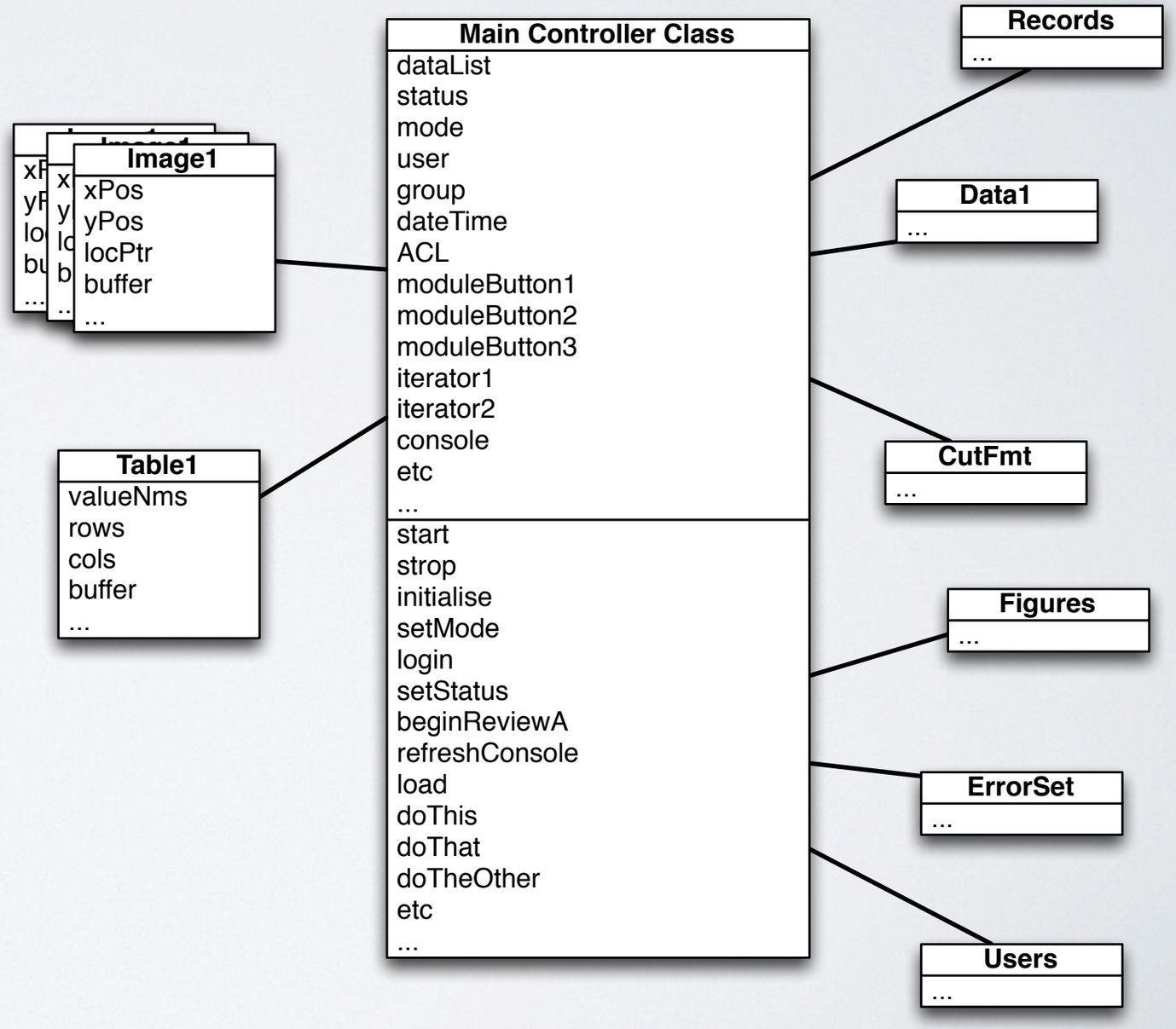
Low Cohesion:

- hard to comprehend
- hard to reuse
- hard to maintain
- delicate; constantly affected by change.

ANTI-PATTERN: THE BLOB

Coupling refers to relations to other classes

Cohesion refers to too many responsibilities



RULES OF THUMB

- For *high cohesion*, a class must
  - **have few methods**
  - have a *small number of lines of code*
  - not do too much work
  - have **high relatedness of code**

DISCUSSION

- *Very low cohesion*: a class that does two completely different tasks, e.g., database connectivity and RPC.

- *Low cohesion*: a class that has sole responsibility for a complex task in one functional area, e.g., one single interface responsible for all database access.

- *Moderate cohesion*: a lightweight class, solely responsible for a few logically related areas, e.g., *Company* that knows the employees and the financial details.

- *High cohesion*: a class with moderate responsibilities in one functional area that collaborates with other classes.

COUPLING AND COHESION

COUPLING AND COHESION

- Coupling: Amount of relations between sub-systems
  - Low coupling general design goal: Independence, supportability
  - May lead to extra layers of abstraction
- Cohesion: Amount of relations within a sub-system
  - High cohesion general design goal: Reduce complexity
  - Often a trade-off

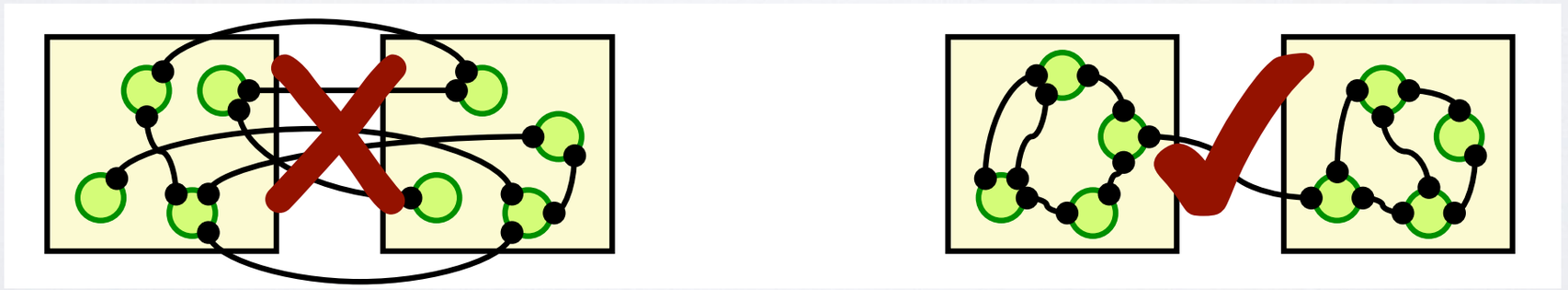
PROPERTIES OF A GOOD ARCHITECTURE

minimises coupling between modules

- Goal: modules don't need to know much about one another to interact
- Low coupling makes future change easier

maximises cohesion within modules

- Goal: the contents of each module are strongly inter-related
- High cohesion makes a module easier to understand



Applies also to classes

CREATOR

CREATOR

Problem: Who creates an A?

Advice: Assign class B the responsibility to create an instance of class A if one of these is true (the more the better):

- B "contains" or compositely aggregates A.
- B records A.
- B closely uses A.
- B has the initialising data for A.

CREATOR

Basic idea is to **find a creator** that needs to **be connected to the created object** in any event.

Need **initialisation data nearby** – sometimes requires that it is passed into client.

Creator applies to every object.

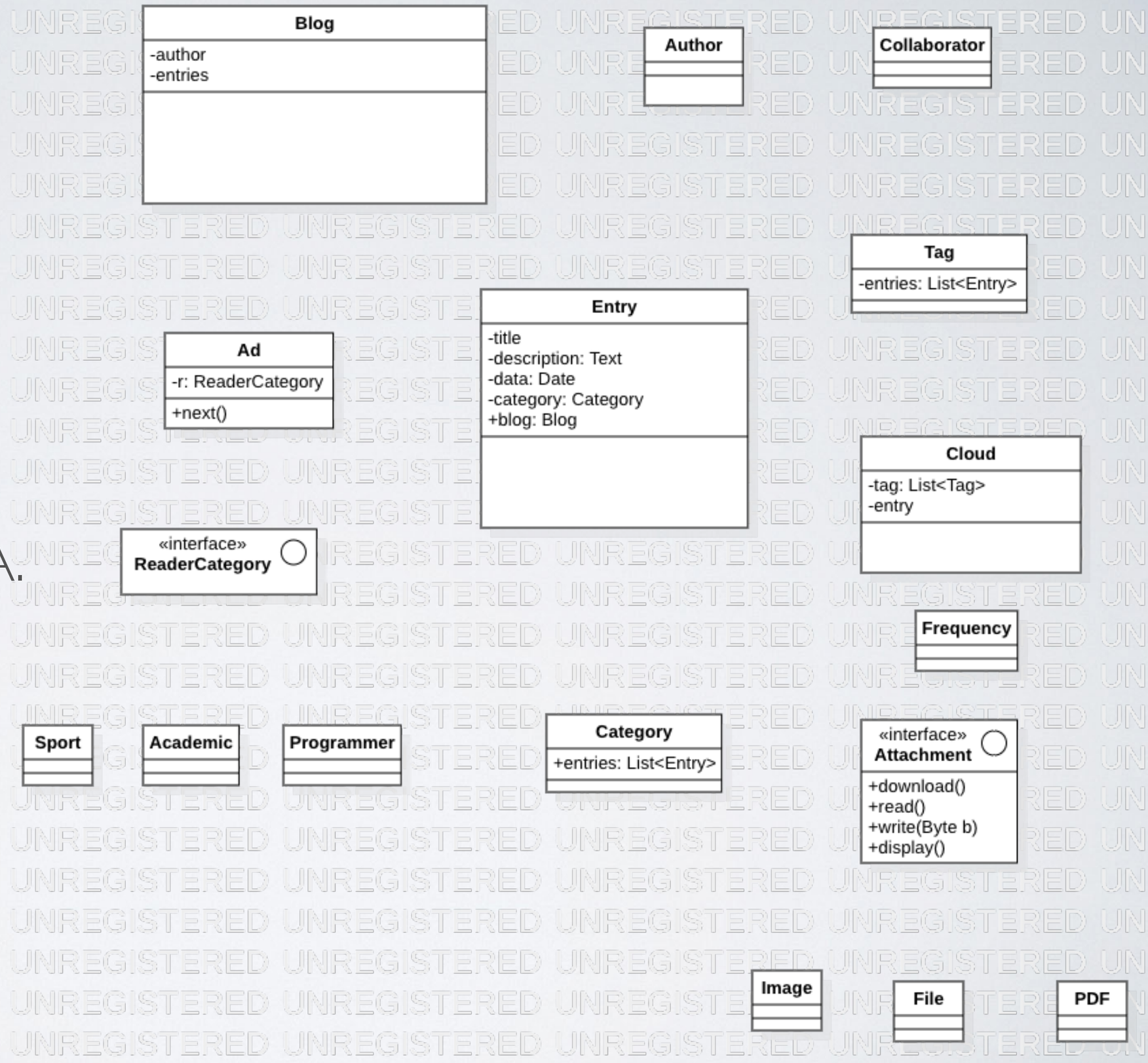
EXAMPLE

Consider a ***LinkedList***.

A **client creates the objects stored in the list**, but the *LinkedList* object creates the **links** of the list.

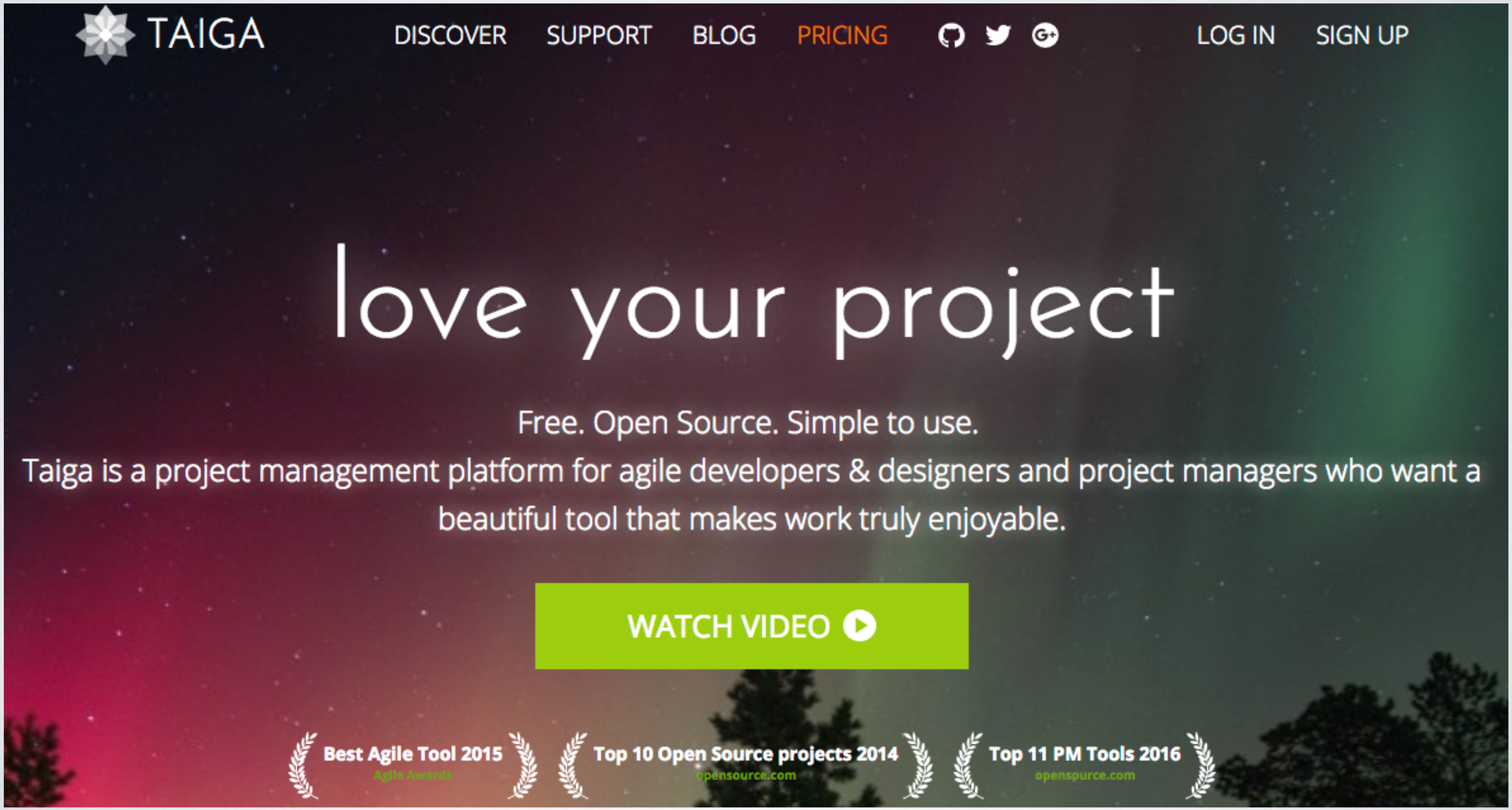
Find creators

- B "contains" A.
- B records A.
- B closely uses A.
- B has the data for A.



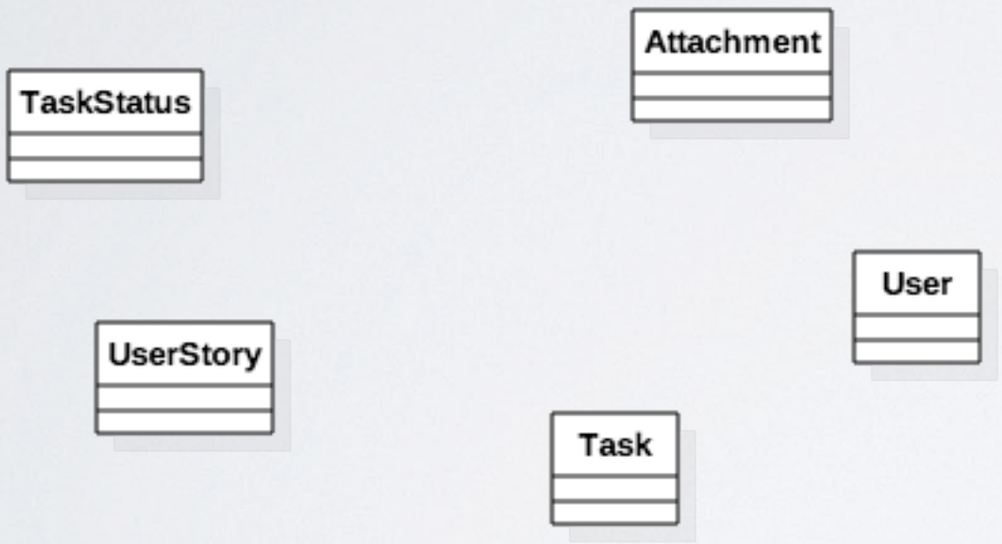
IN-CLASS EXERCISE

Group work



CREATOR

Use UML relations (aggregation, composition and association) to show who is in charge of the lifetime of an object.



Who is in charge of…

- creating attachments
- creating task status
- creating a task

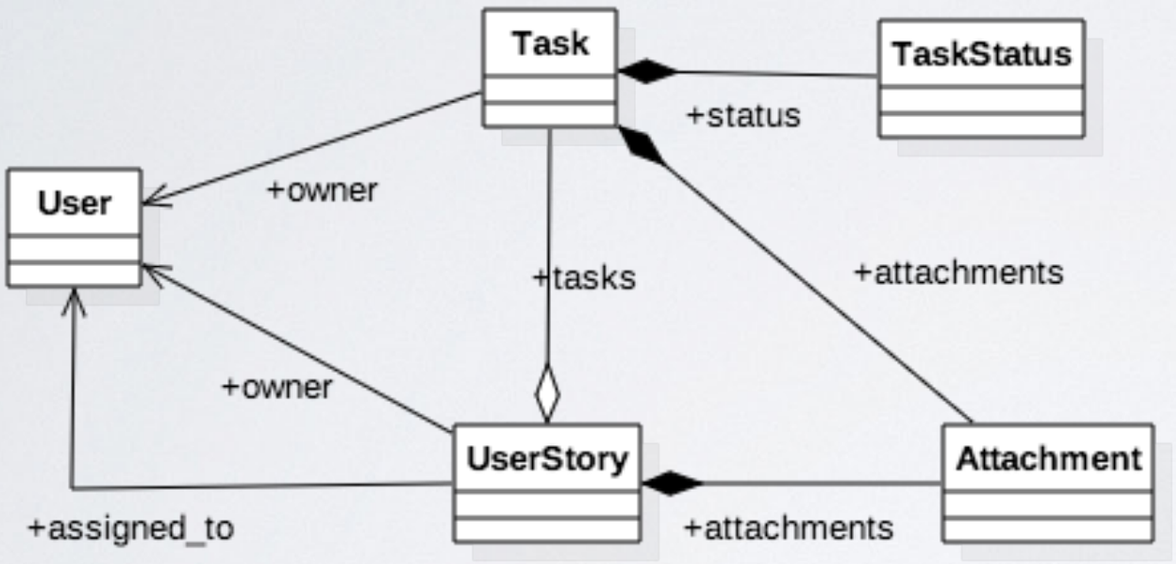
UML Reminder

- Composition
- Aggregation
- Association
- Dependency

TAIGA

CREATOR

Use **UML relations** (aggregation, composition and association) to show who is in charge of the lifetime of an object.



Who is in charge of…

- creating attachments
- creating task status
- creating a task

UML Reminder

- Composition
- Aggregation
- Association
- Dependency

TAIGA

INFORMATION EXPERT

INFORMATION EXPERT

Problem: What basic principle do you use to assign responsibilities/functionality to objects?

Choose well and design will be easier to understand, maintain, extend, and reuse.

SOLUTION

Assign responsibility to the information expert class – the **class that has the information to fulfil the responsibility**.

SOLUTION

Assign responsibility to the information expert class – the **class that has the information to fulfil the responsibility**.

Start by clearly stating the responsibility.

SOLUTION

Assign responsibility to the information expert class – the **class that has the information to fulfil the responsibility**.

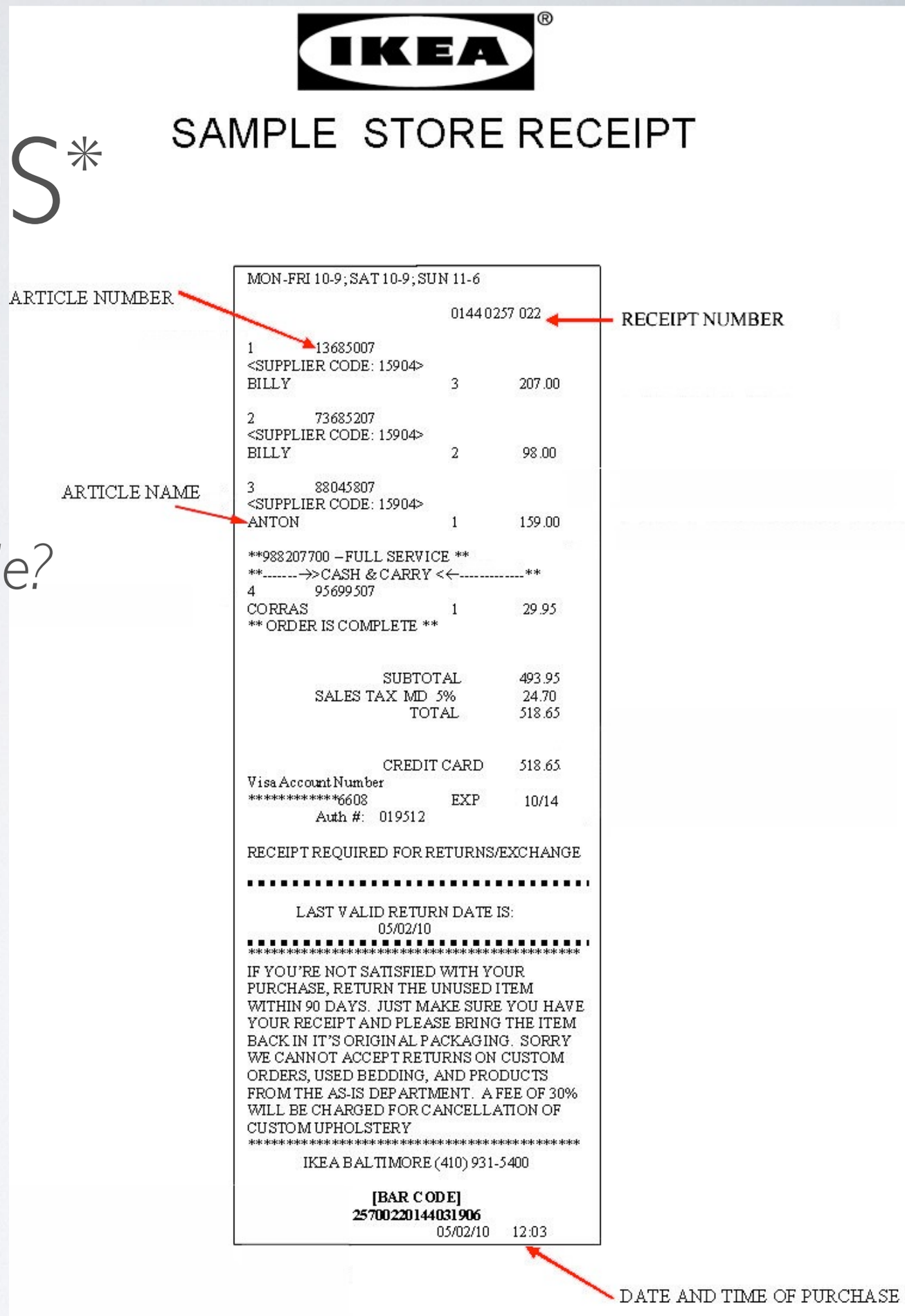
Start by clearly stating the responsibility.

If that is not helpful, **look to Domain Model** and attempt to use (or expand) its classes to inspire the create of design classes.

EXAMPLE: POS*

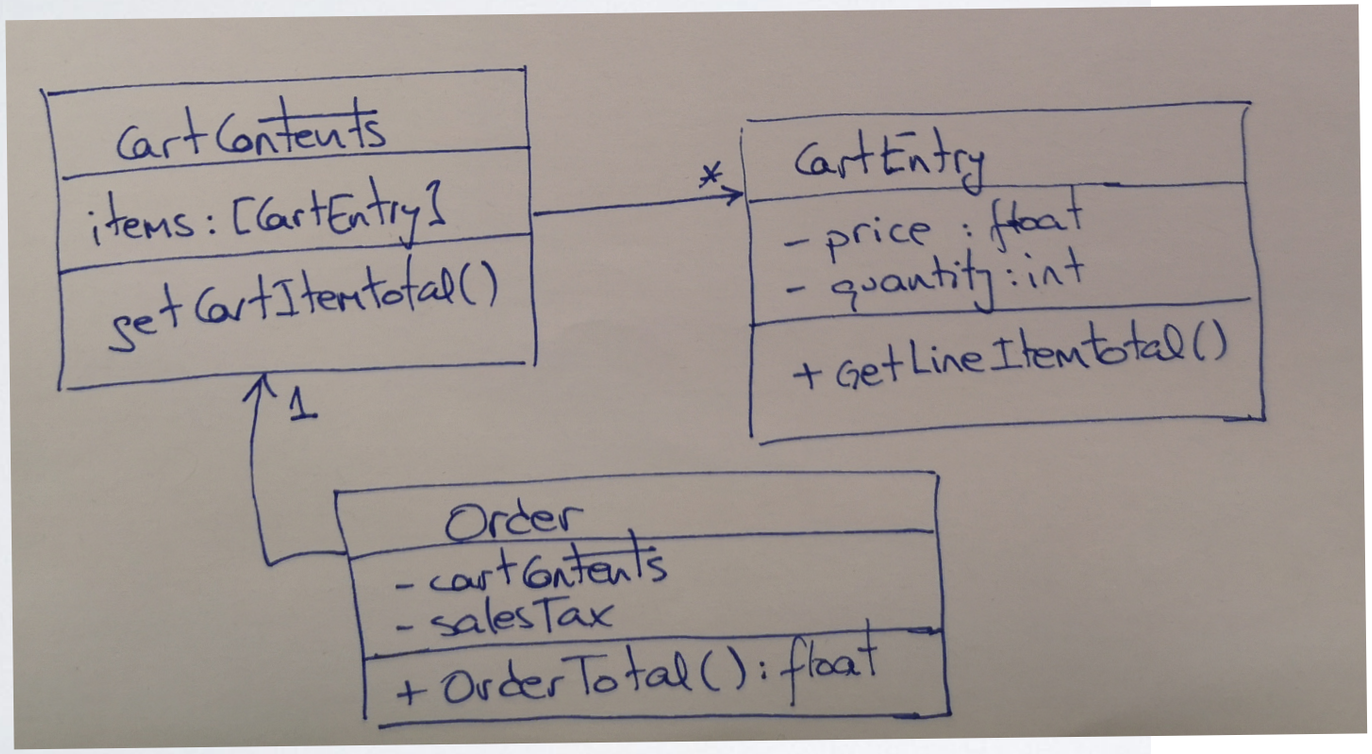
Who should be responsible for knowing the grand total of a sale?

POS* = Point Of Sale



EXAMPLE: POS*

Who should be responsible for knowing the grand total of a sale?



POS* = Point Of Sale

SAMPLE STORE RECEIPT

ARTICLE NUMBER

ARTICLE NAME

RECEIPT NUMBER

MON-FRI 10-9; SAT 10-9; SUN 11-6

0144 0257 022

1 13685007
<SUPPLIER CODE: 15904>
BILLY 3 207.00

2 73685207
<SUPPLIER CODE: 15904>
BILLY 2 98.00

3 88045807
<SUPPLIER CODE: 15904>
ANTON 1 159.00

**988207700 –FULL SERVICE **
------->>CASH & CARRY <<----------
4 95699507
CORRAS 1 29.95
** ORDER IS COMPLETE **

SUBTOTAL 493.95
SALES TAX MD 5% 24.70
TOTAL 518.65

CREDIT CARD 518.65
Visa Account Number
**********6608 EXP 10/14
Auth #: 019512

RECEIPT REQUIRED FOR RETURNS/EXCHANGE

LAST VALID RETURN DATE IS:
05/02/10

****
IF YOU'RE NOT SATISFIED WITH YOUR PURCHASE, RETURN THE UNUSED ITEM WITHIN 90 DAYS. JUST MAKE SURE YOU HAVE YOUR RECEIPT AND PLEASE BRING THE ITEM BACK IN IT'S ORIGINAL PACKAGING. SORRY WE CANNOT ACCEPT RETURNS ON CUSTOM ORDERS, USED BEDDING, AND PRODUCTS FROM THE AS-IS DEPARTMENT. A FEE OF 30% WILL BE CHARGED FOR CANCELLATION OF CUSTOM UPHOLSTERY
****
IKEA BALTIMORE (410) 931-5400

[BAR CODE]
25700220144031906
05/02/10 12:03

DATE AND TIME OF PURCHASE

Calculate total of an order

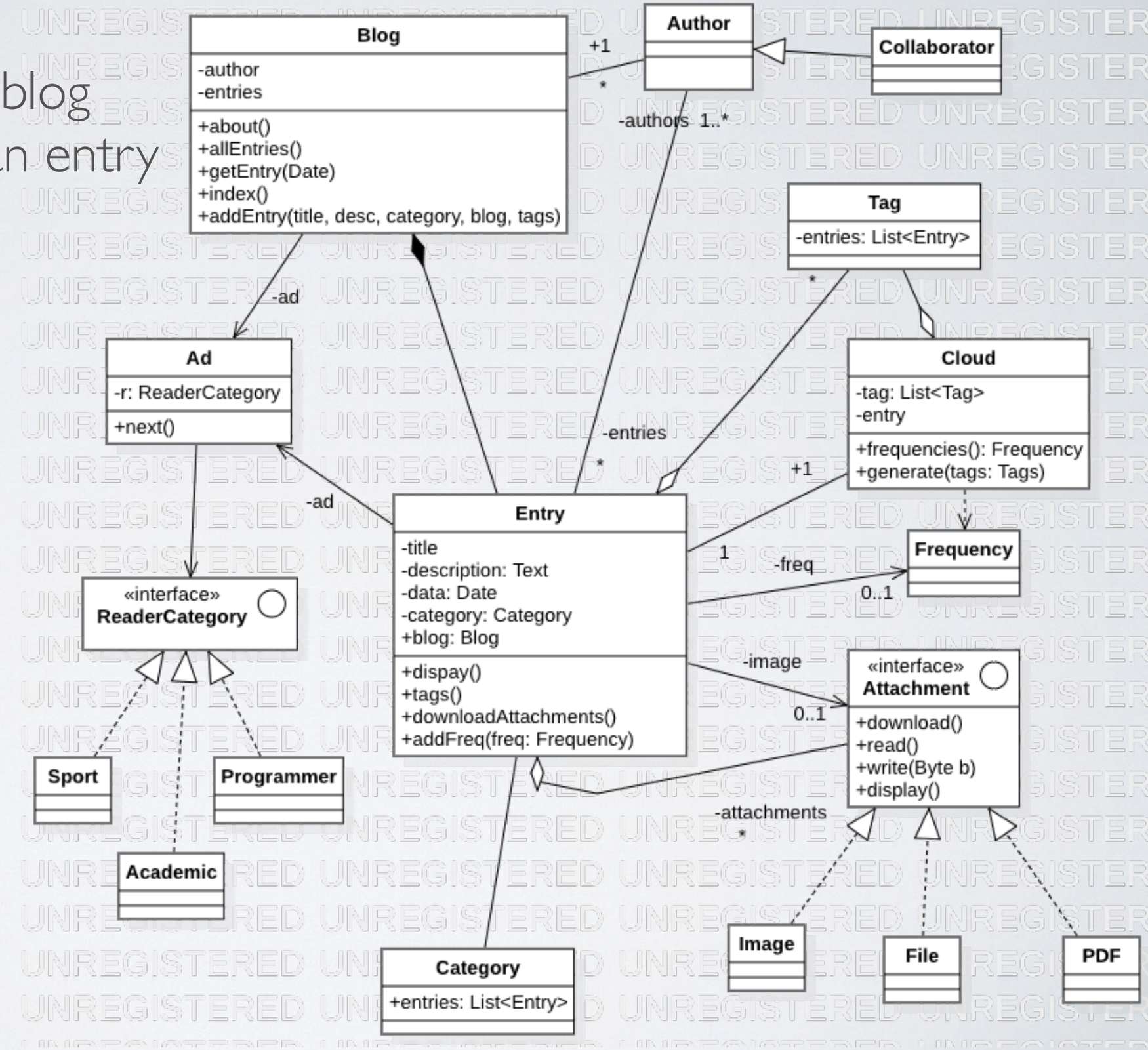
```
public class CartEntry
{
    private float Price;
    private int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}
```

```
public class CartContents
{
    private CartEntry[] items;

    public float GetCartItemsTotal()
    {
        float cartTotal = 0;
        foreach (CartEntry item in items)
        {
            cartTotal += item.GetLineItemTotal();
        }
        return cartTotal;
    }
}
```

```
public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        return cart.GetCartItemsTotal() *
               (1.0f + salesTax);
    }
}
```

Find Inf.Experts for:

- listing entries of a blog
- listing authors of an entry



DISCUSSION

Notice that it required **information spread across different classes** of objects.

Partial information experts collaborate in the task.

Sometimes other considerations overrule.

> For instance, should a *Sale* object save itself in database?
>
> If so, then all database code would be scattered across codebase.

Information Expert is about responsibilities, not location of data.

CONTROLLER

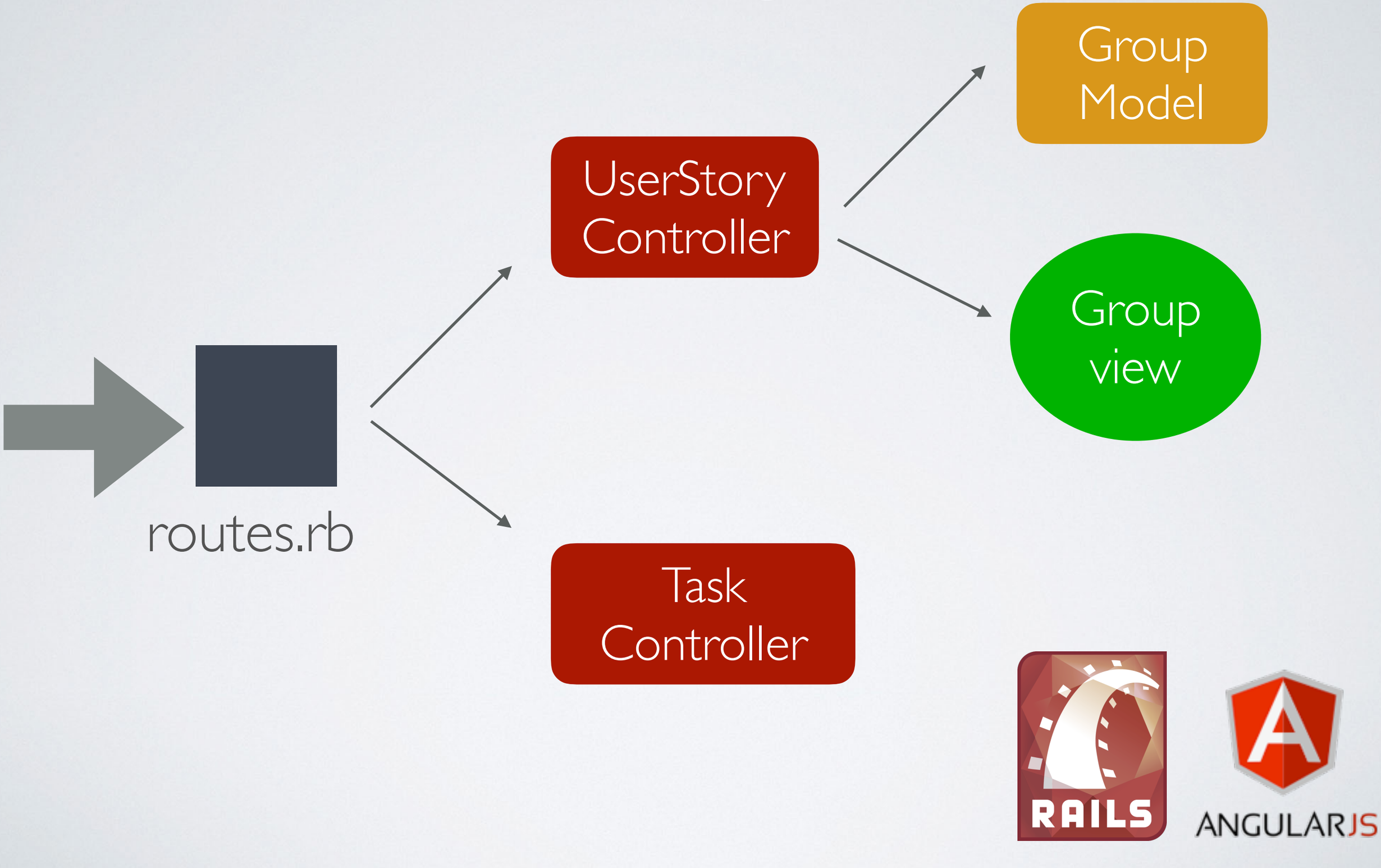
CONTROLLER

Problem: What first object beyond the UI layer receives and coordinates ("controls") a system operation?

Advice: Assign the responsibility to an object representing one of these choices:

- overall "system", a "root object"
- use case within the system

MVC


Group
Model
UserStory
Controller
Group
view
routes.rb
Task
Controller
RAILS
ANGULARJS

CONTROLLER

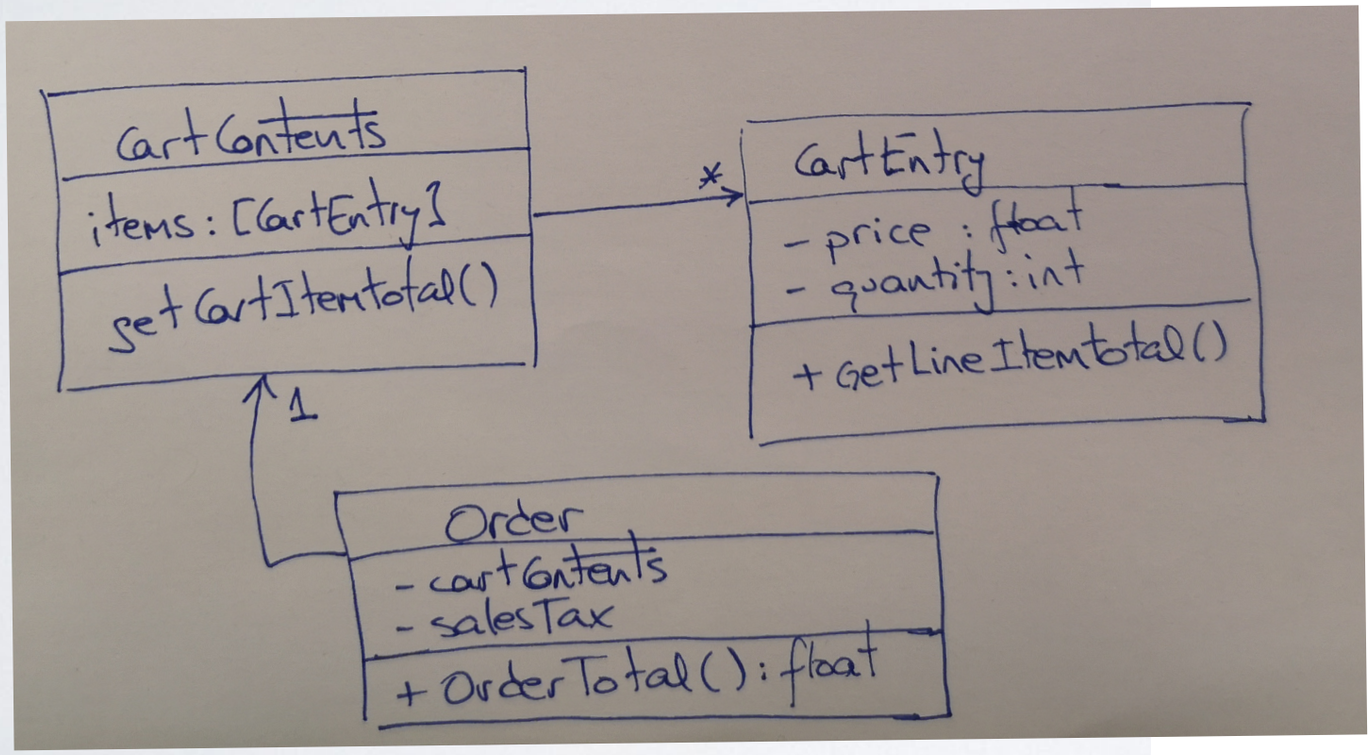
Delegation pattern – the UI should not contain application logic.

Increases potential for **reuse** and pluggable interfaces.

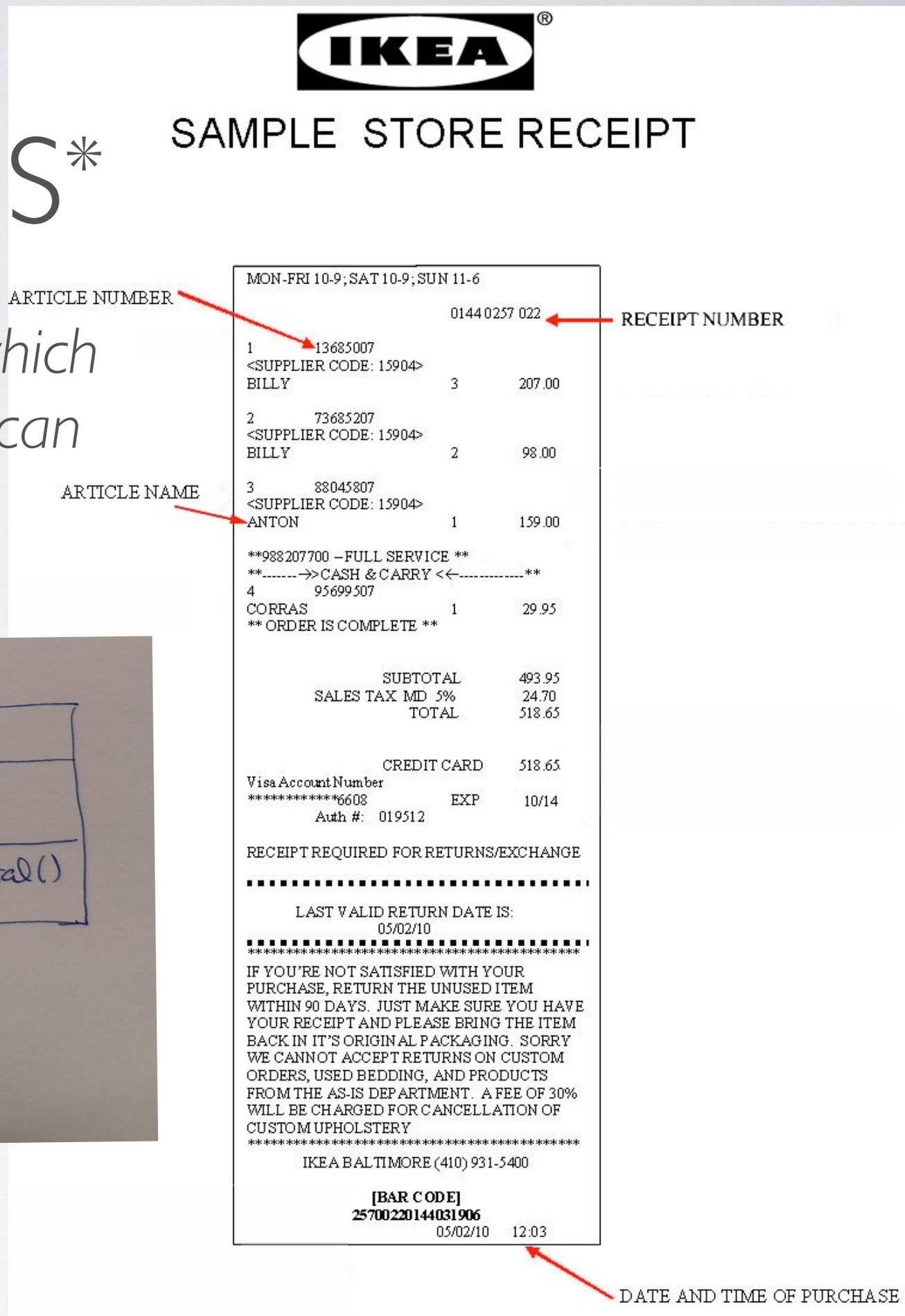
Creates opportunity to **reason about state of use case**, for example, to ensure that operations occur in a legal sequence.

EXAMPLE: POS*

Who should be the controller, which manages interaction between scan items and machine?

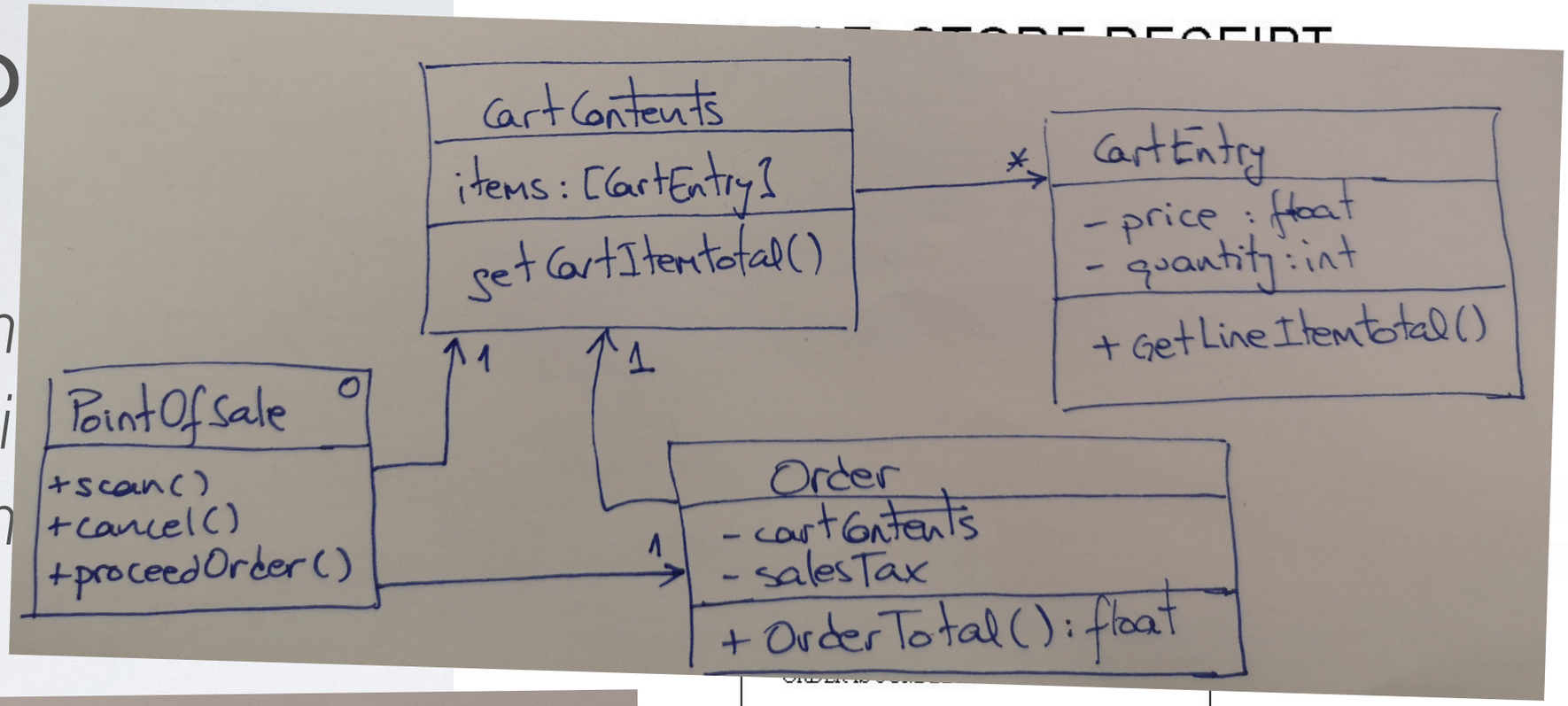


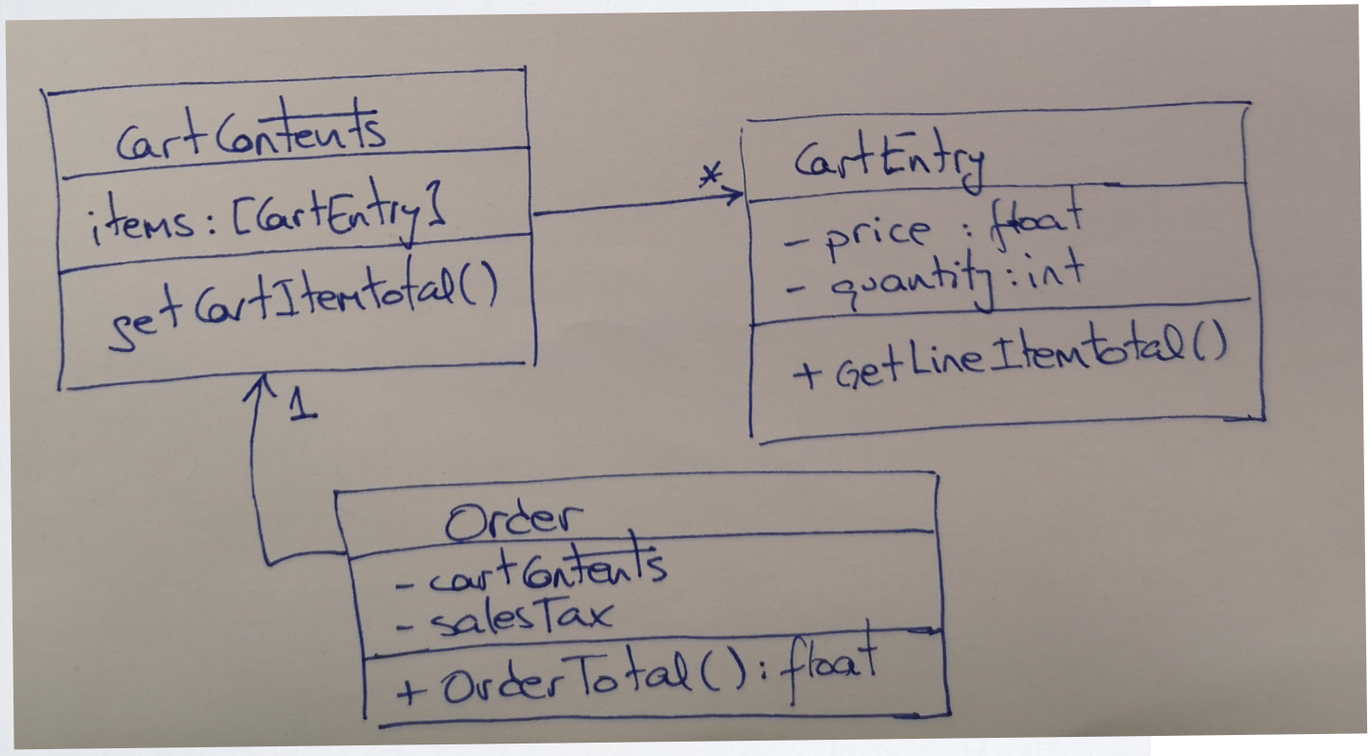
POS* = Point Of Sale



EXAMP

Who should be th
manages interacti
items and machin





IKEA®

SUBTOTAL 493.95
SALES TAX MD 5% 24.70
TOTAL 518.65

CREDIT CARD 518.65
Visa Account Number
**********6608 EXP 10/14
Auth #: 019512

RECEIPT REQUIRED FOR RETURNS/EXCHANGE

LAST VALID RETURN DATE IS:
05/02/10

****

IF YOU'RE NOT SATISFIED WITH YOUR PURCHASE, RETURN THE UNUSED ITEM WITHIN 90 DAYS. JUST MAKE SURE YOU HAVE YOUR RECEIPT AND PLEASE BRING THE ITEM BACK IN IT'S ORIGINAL PACKAGING. SORRY WE CANNOT ACCEPT RETURNS ON CUSTOM ORDERS, USED BEDDING, AND PRODUCTS FROM THE AS-IS DEPARTMENT. A FEE OF 30% WILL BE CHARGED FOR CANCELLATION OF CUSTOM UPHOLSTERY

****

IKEA BALTIMORE (410) 931-5400

[BAR CODE]
25700220144031906
05/02/10 12:03

DATE AND TIME OF PURCHASE

POS* = Point Of Sale

PROBLEM

A bloated controller is an example of low cohesion.

Problem: **Bloated Controllers**

- a single controller that receives all system events, does too much of the work handling events, has too many attributes (duplicating information found elsewhere), etc.

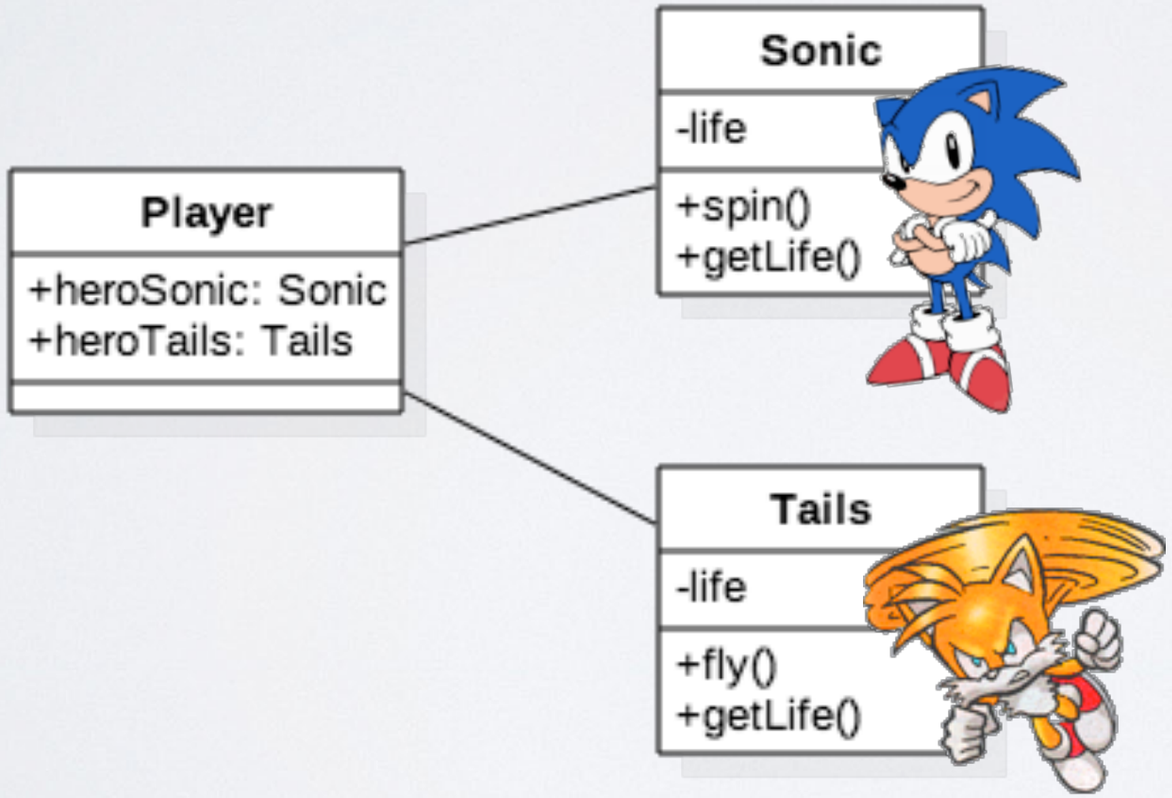
Remedies:

- **Add more controllers**
- Design controller so that it delegates the fulfilment of each system operation to other objects.

POLYMORPHISM

POLYMORPHISM

Problem: How to handle alternatives based on type? How to create pluggable software components?

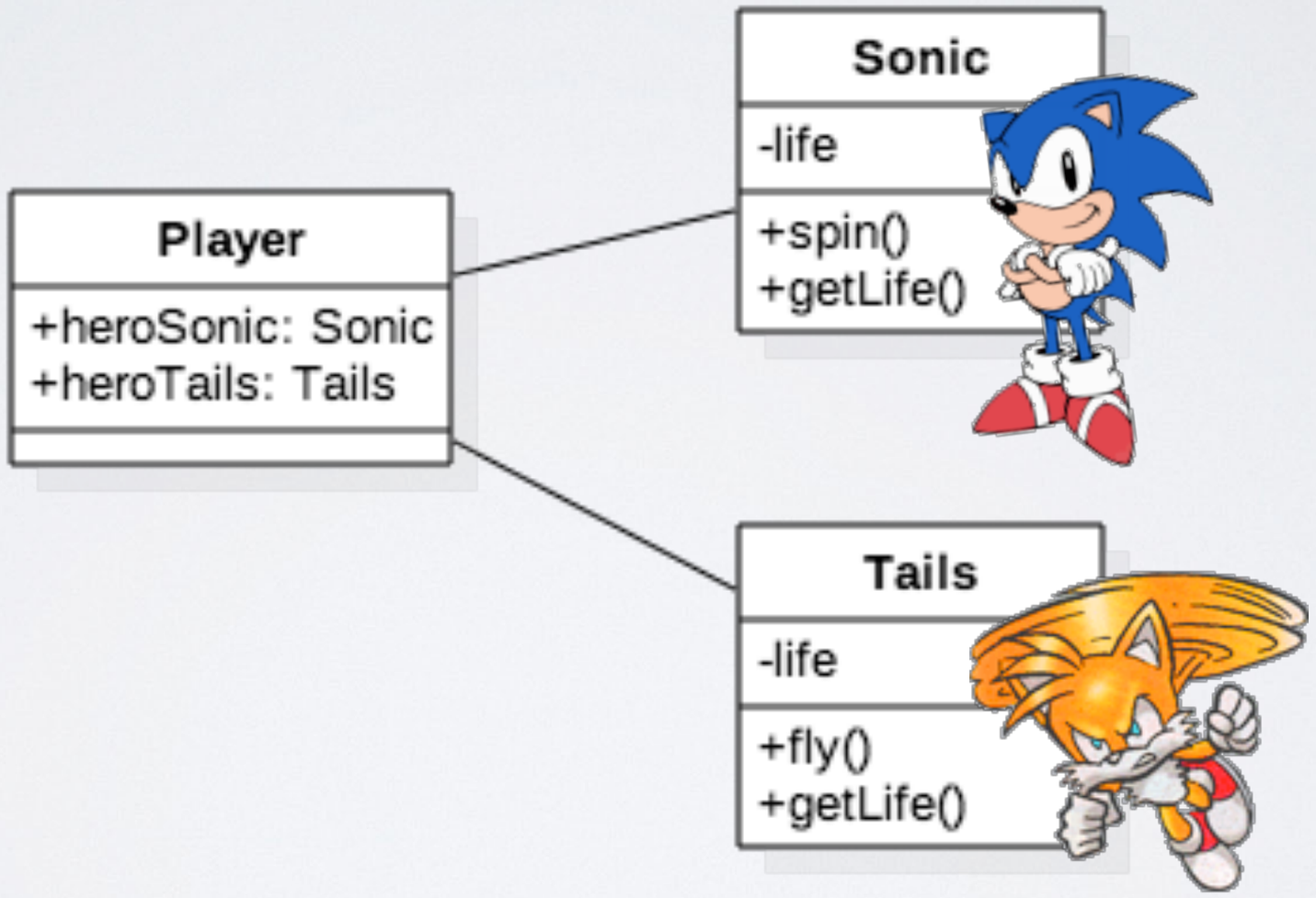


SOLUTION

When related alternatives or behaviours vary by type, assign responsibility for the behaviour – using polymorphic operations – to types for which the behaviour varies.

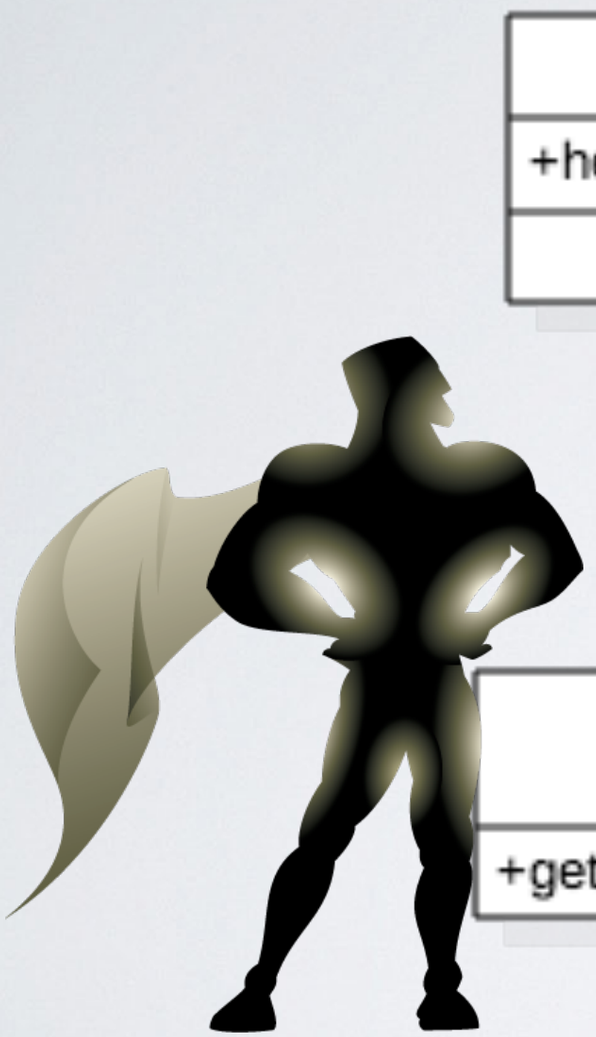
Do not test for the type of an object and use conditional logic to perform varying alternatives based on type.

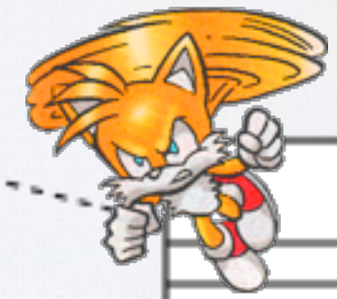
NOT POLYMORPHIC



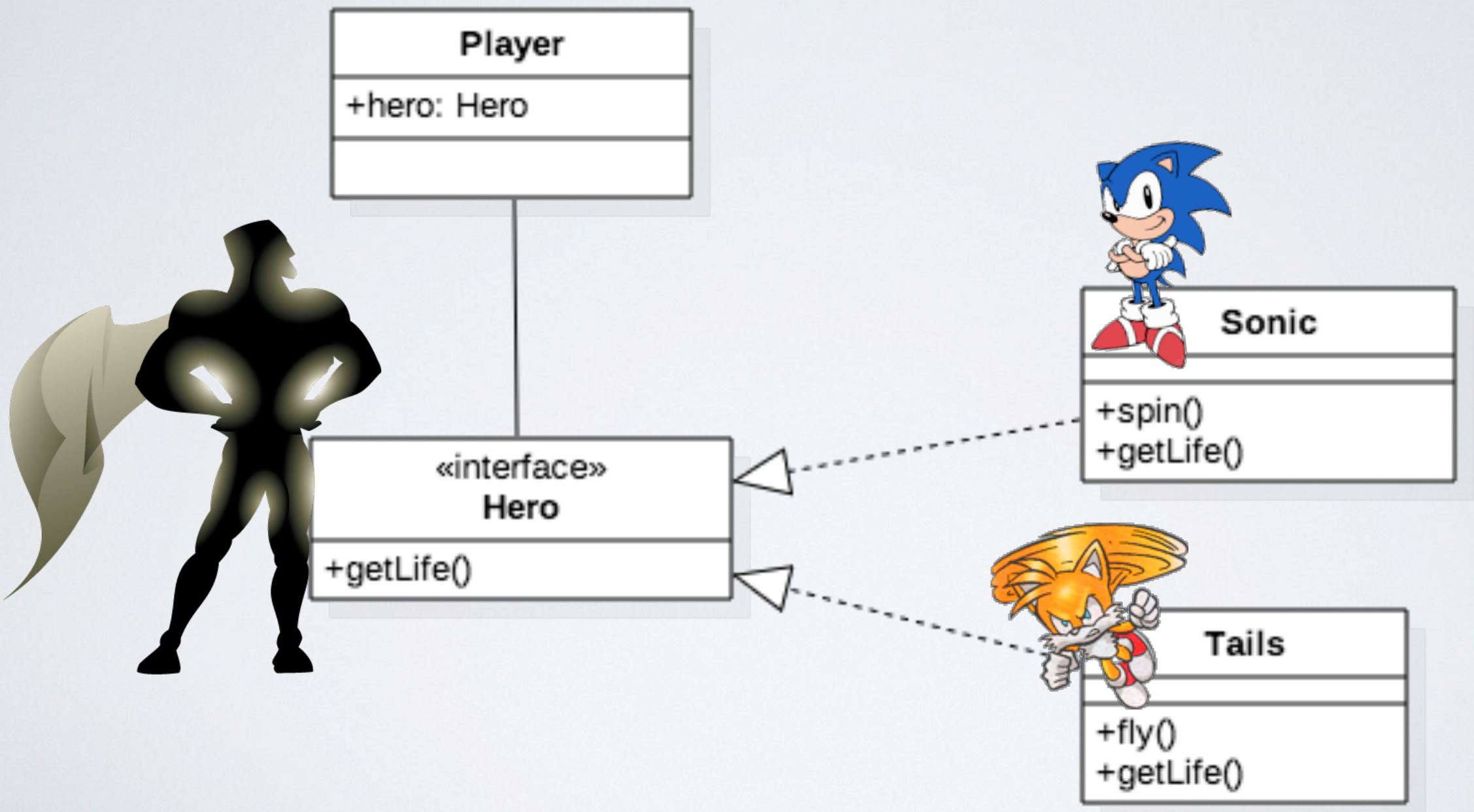
POLYMORPHIC…

POLYMORPHIC…



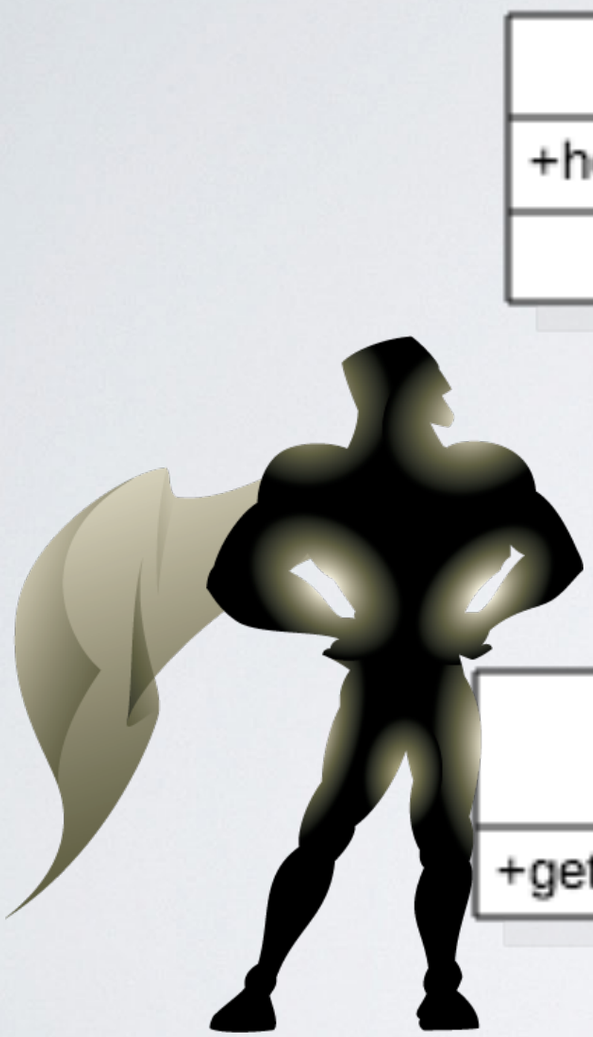


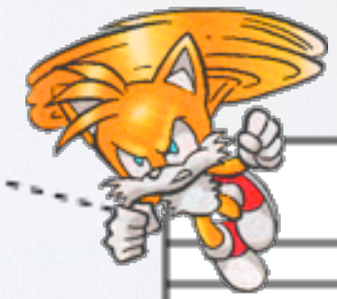
POLYMORPHIC…ERROR



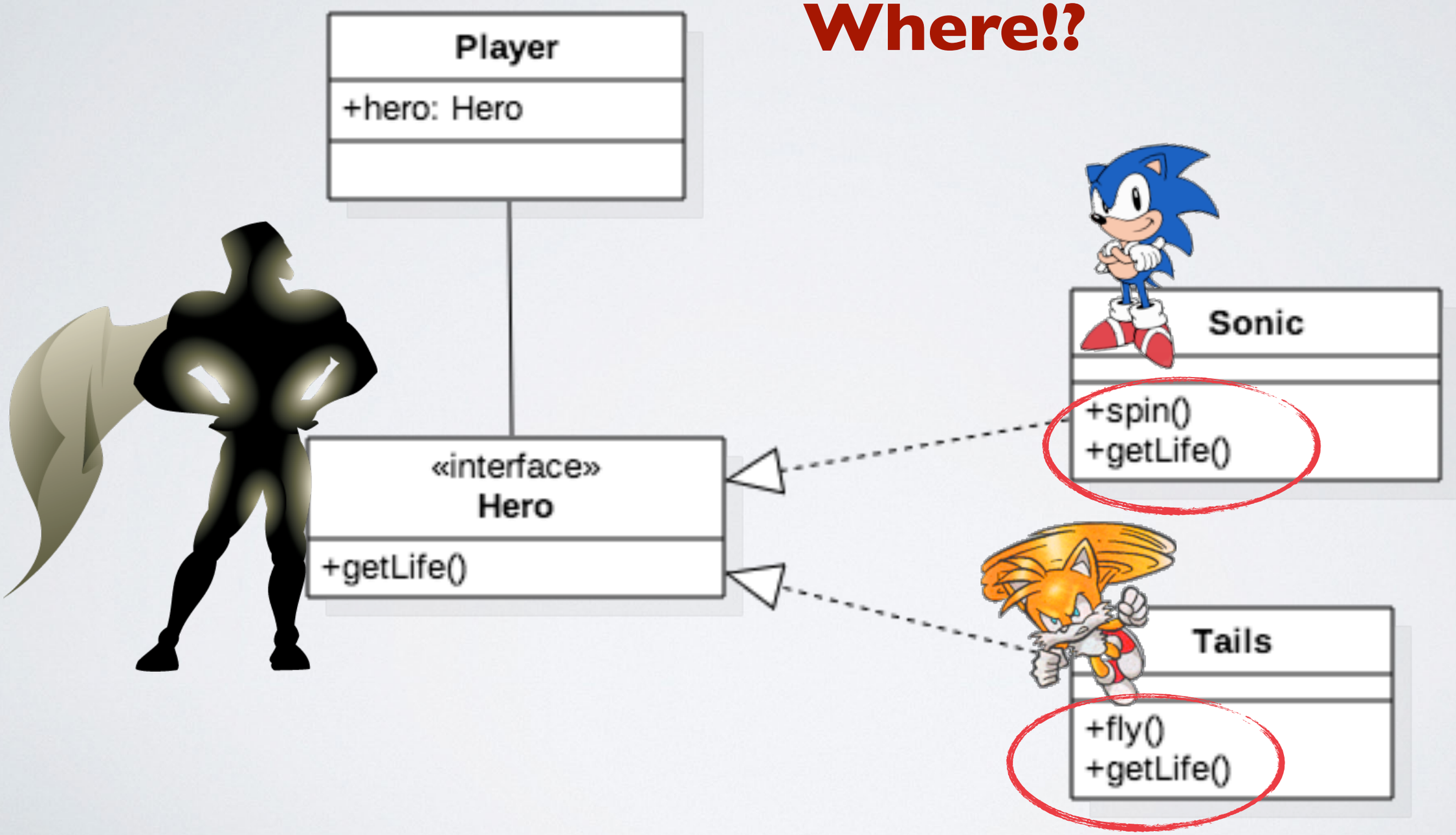
POLYMORPHIC…ERROR

Where!?





POLYMORPHIC…ERROR

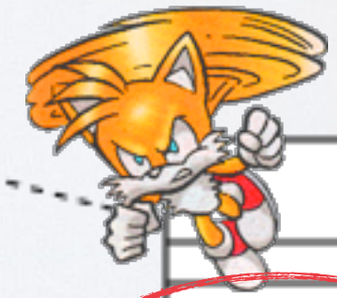


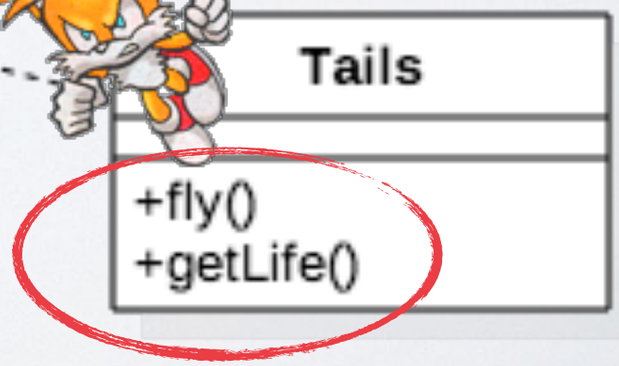
Where!?

POLYMORPHIC…ERROR

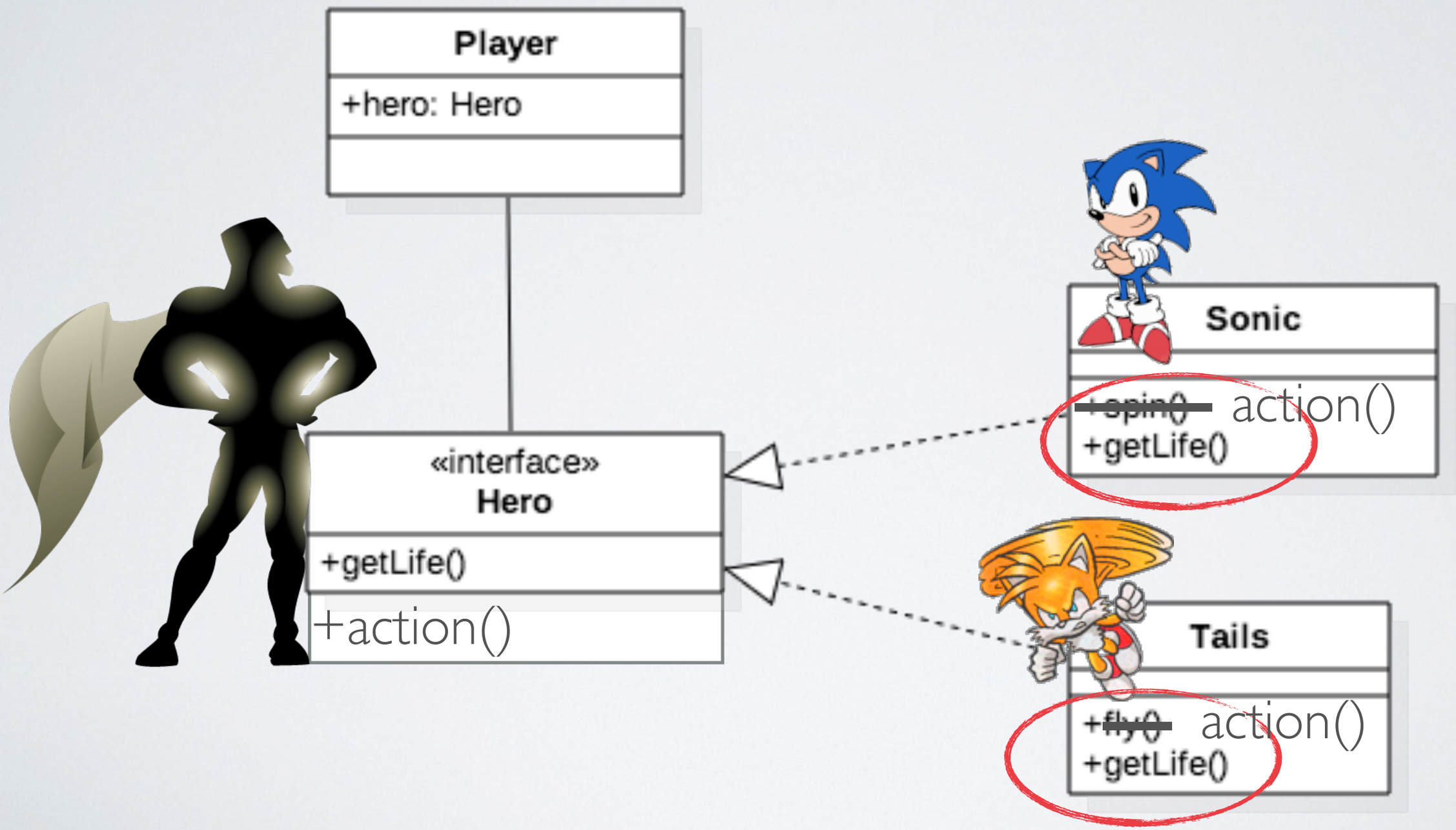
Where!?

Specific methods cannot be accessed!





POLYMORPHIC



DISCUSSION

Interfaces or superclasses:

- Guideline: use interfaces when you want polymorphism without committing to a particular class hierarchy.
- Liskov substitution principle – a value can be replaced by a subtype without changing important properties of program

Future-proofing:

- if variability at a particular point is very probably, then expend the effort to accommodate flexibility.
- Avoid adding flexibility just because it is possible.

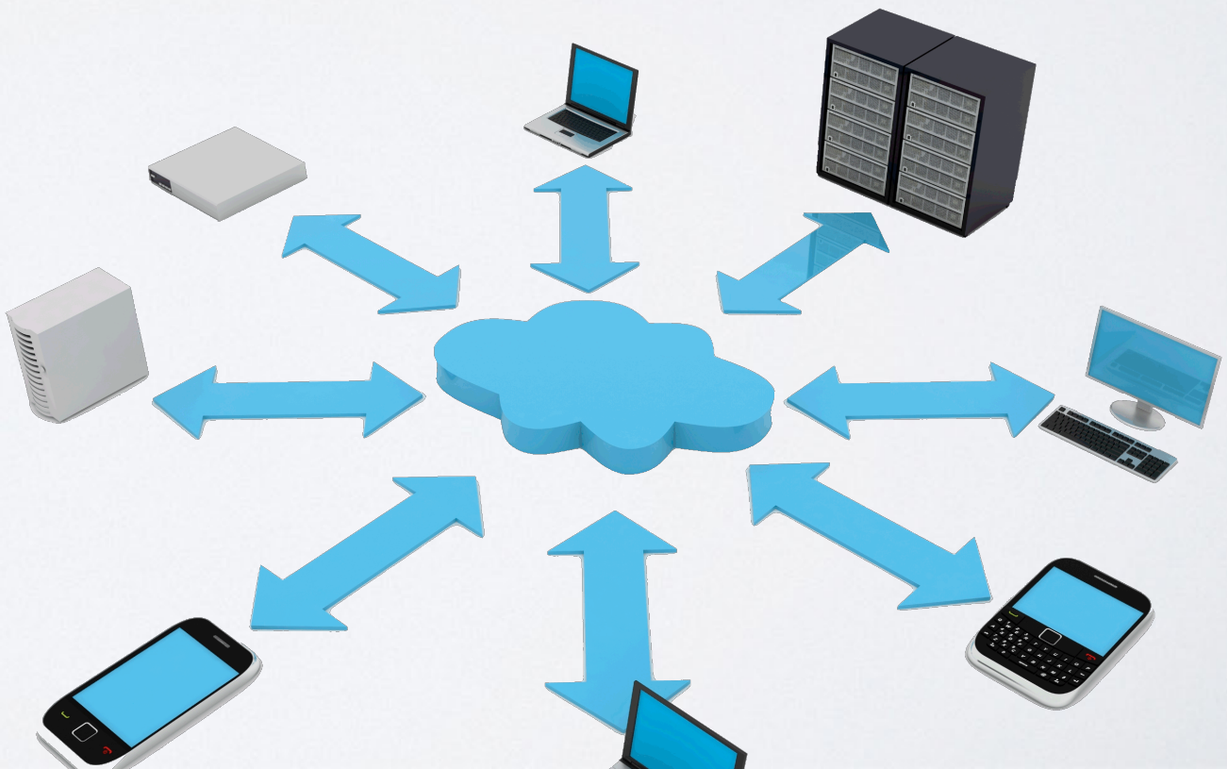
INDIRECTION

INDIRECTION

Problem:

How to assign responsibility to avoid direct coupling between two (or more) things?

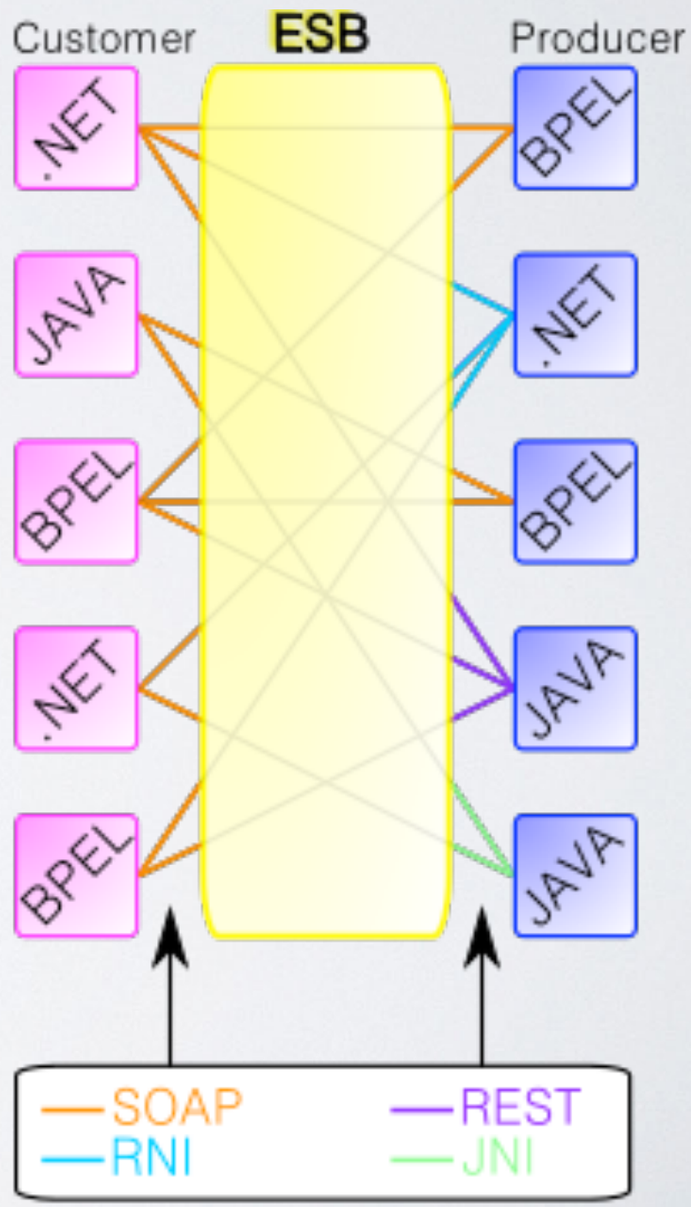
How to decouple objects so that low coupling is supported and reuse potential remains higher?



SOLUTION

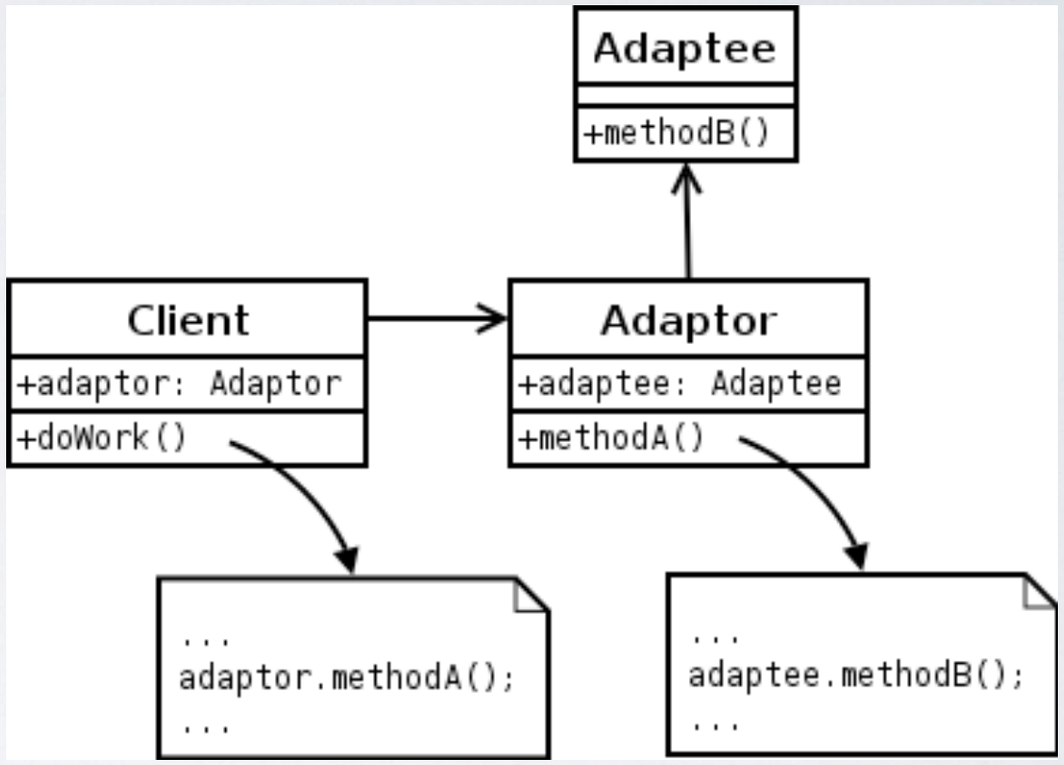
Assign the responsibility to an intermediate object to mediate between other components or services so that they are not directly coupled.

The intermediary creates *indirection* between the other components.



EXAMPLE

Adapter pattern



The adaptor acts as a level of indirection to external systems.

PURE FABRICATION

PURE FABRICATION

What objects should have a responsibility, when you do not want to violate High Cohesion and Low Coupling, and solutions offered by Information Expert are not appropriate?

PURE FABRICATION

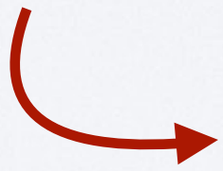
What objects should have a responsibility, when you do not want to violate High Cohesion and Low Coupling, and solutions offered by Information Expert are not appropriate?

Classes from the domain layer?

PURE FABRICATION

What objects should have a responsibility, when you do not want to violate High Cohesion and Low Coupling, and solutions offered by Information Expert are not appropriate?

Classes from the domain layer?



violate High Cohesion and Low Coupling

PURE FABRICATION

What objects should have a responsibility, when you do not want to violate High Cohesion and Low Coupling, and solutions offered by Information Expert are not appropriate?

Classes from the domain layer?

What to do when we cannot choose a class from the domain?

and Low Coupling

SOLUTION

Assign a highly cohesive set of responsibilities to a convenience class, not representing a problem domain concept.

Fabrication – made up.

Pure – keep it clean: high cohesion, low coupling.

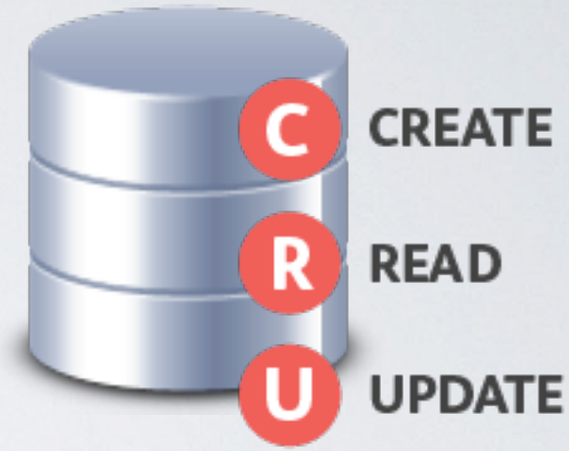
"Pure fabrication" – English idiom that implies making something up.

Most classes not appearing in the domain model will be pure fabrications.

EXAMPLE

Need to save *Sale* instances in a relation database.

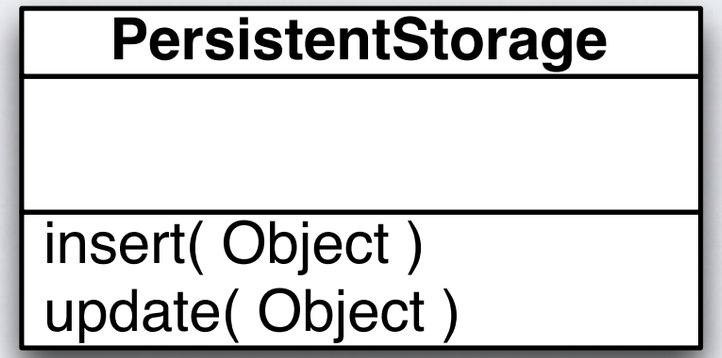
Information Expert says assign functionality to *Sale*.



Implications:

- Task needs large number of supporting database-oriented operations, none related to the concept of a *Sale*. Incohesion!
- *Sale* becomes coupled to database interface, coupling goes up.
- Saving objects in a database is a general task – many classes will need it.

SOLUTION



Understandable concept.
Pure software concept.
Not in domain model.

Sale unaffected.
Cohesive concept.
Generic and reusable.

Alternatives: (Page 612 Larman, 3rd Edition)
Chapter 37: Designing a Persistence Framework with Patterns

SOLUTION

```
String sql = "SELECT id, first_name, last_name, phone, birth_date, sex
              FROM persons
              WHERE id = 10”;
Result res = db.execSql(sql);
String name = res[0]["FIRST_NAME"];
```

Low-level

```
Person p = repository.GetPerson(10);
String name = p.getFirstName();
```

Repository

```
Person p = Person.Get(10);
Person p = Person.Get(Person.Properties.Id == 10);
```

Embedded

PROTECTED VARIATIONS

PROTECTED VARIANTS

Problem: How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

SOLUTION

Identify points of predicted variation or **instability**

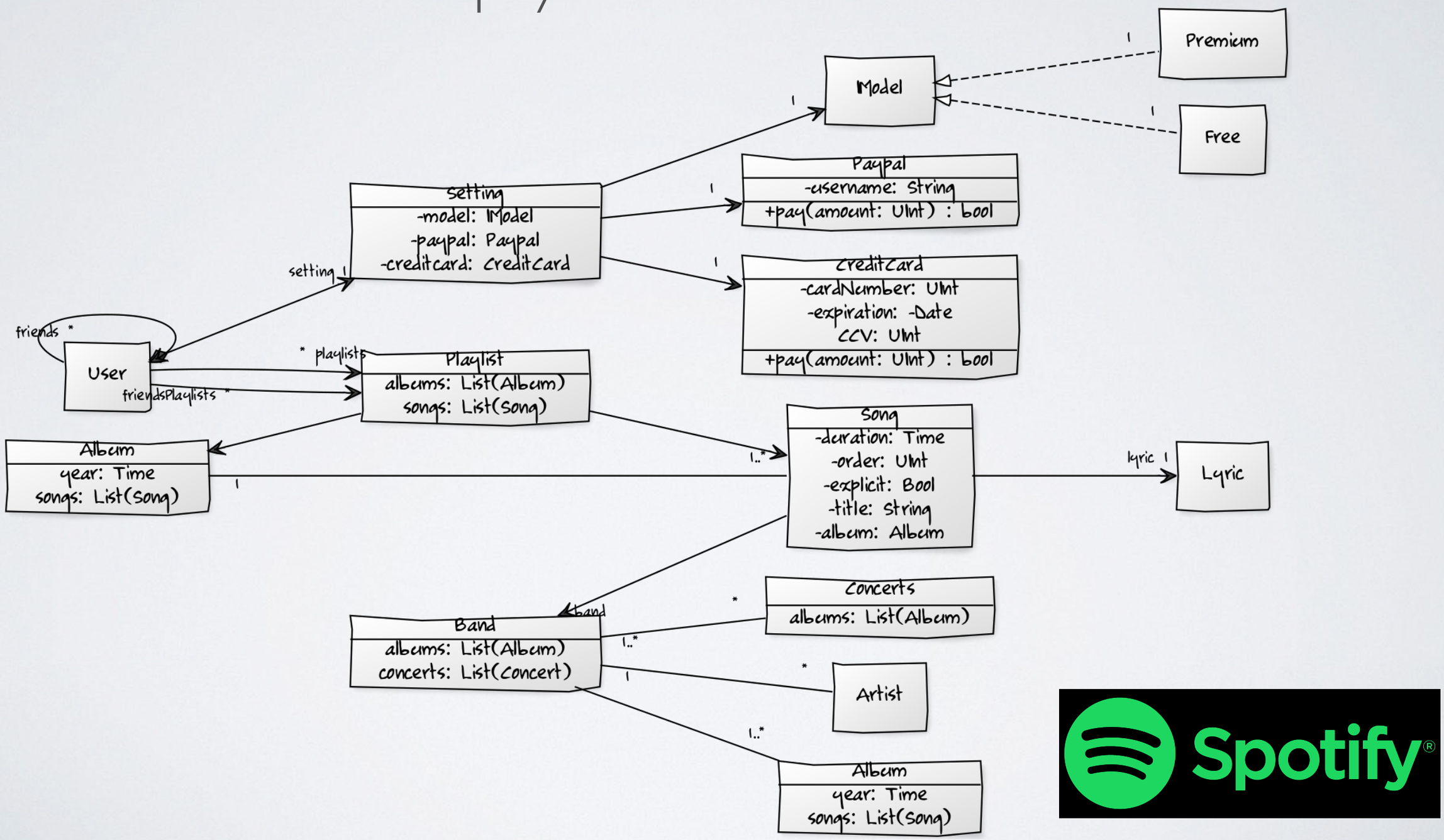
Assign responsibilities to create a stable interface around them.

"Interface" in broadest sense – not just Java interface.

Information hiding!!!

EXAMPLE

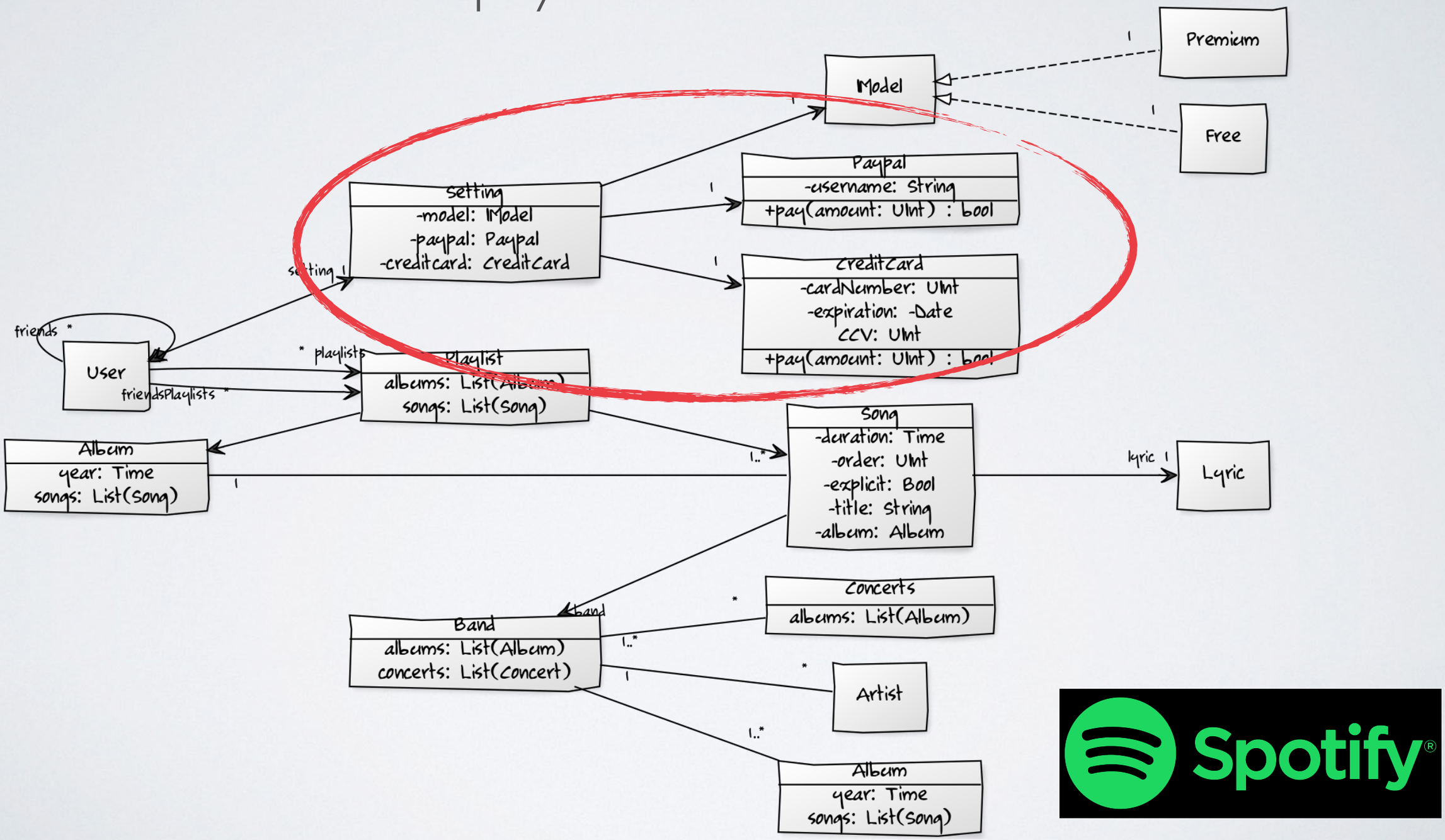
How to add a new payment method?



Spotify®

EXAMPLE

How to add a new payment method?



Free

Paypal
-username: String
+pay(amount: UInt) : bool

Setting
-model: IModel
-paypal: Paypal
-creditcard: CreditCard

CreditCard
-cardNumber: UInt
-expiration: -Date
CCV: UInt
+pay(amount: UInt) : bool

Previous version

Playlist
albums: List(Album)
songs: List(Song)

Song
-duration: Time
-order: UInt
-explicit: Bool
-title: String
-album: Album

Lyric

Concerts
albums: List(Album)

Band
albums: List(Album)
concerts: List(Concert)

Artist

Album
year: Time
songs: List(Song)

Revised version

Premium

Free

Model

Paypal
-username: String
+pay(amount: UInt) : bool

Setting
-model: IModel
-payment: PaymentMethod

PaymentMethod
pay(amount: UInt) : bool

CreditCard
-cardNumber: UInt
-expiration: -Date
CCV: UInt
+pay(amount: UInt) : bool

User

Playlist

Song
-order: UInt
-explicit: Bool
-title: String
-album: Album

Lyric

Album
year: Time
songs: List(Song)

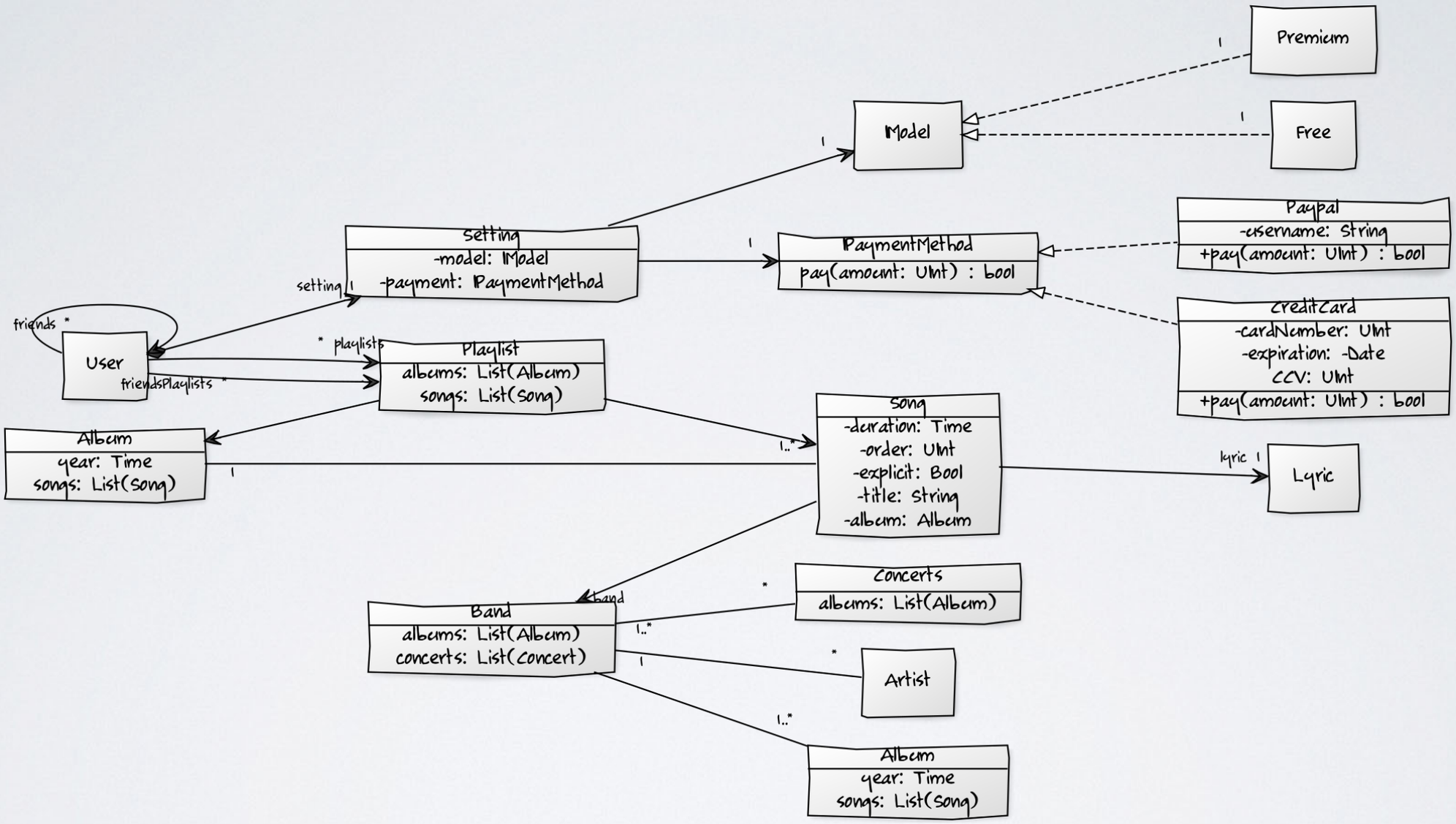
Concerts
albums: List(Album)

Band
albums: List(Album)
concerts: List(Concert)

Artist

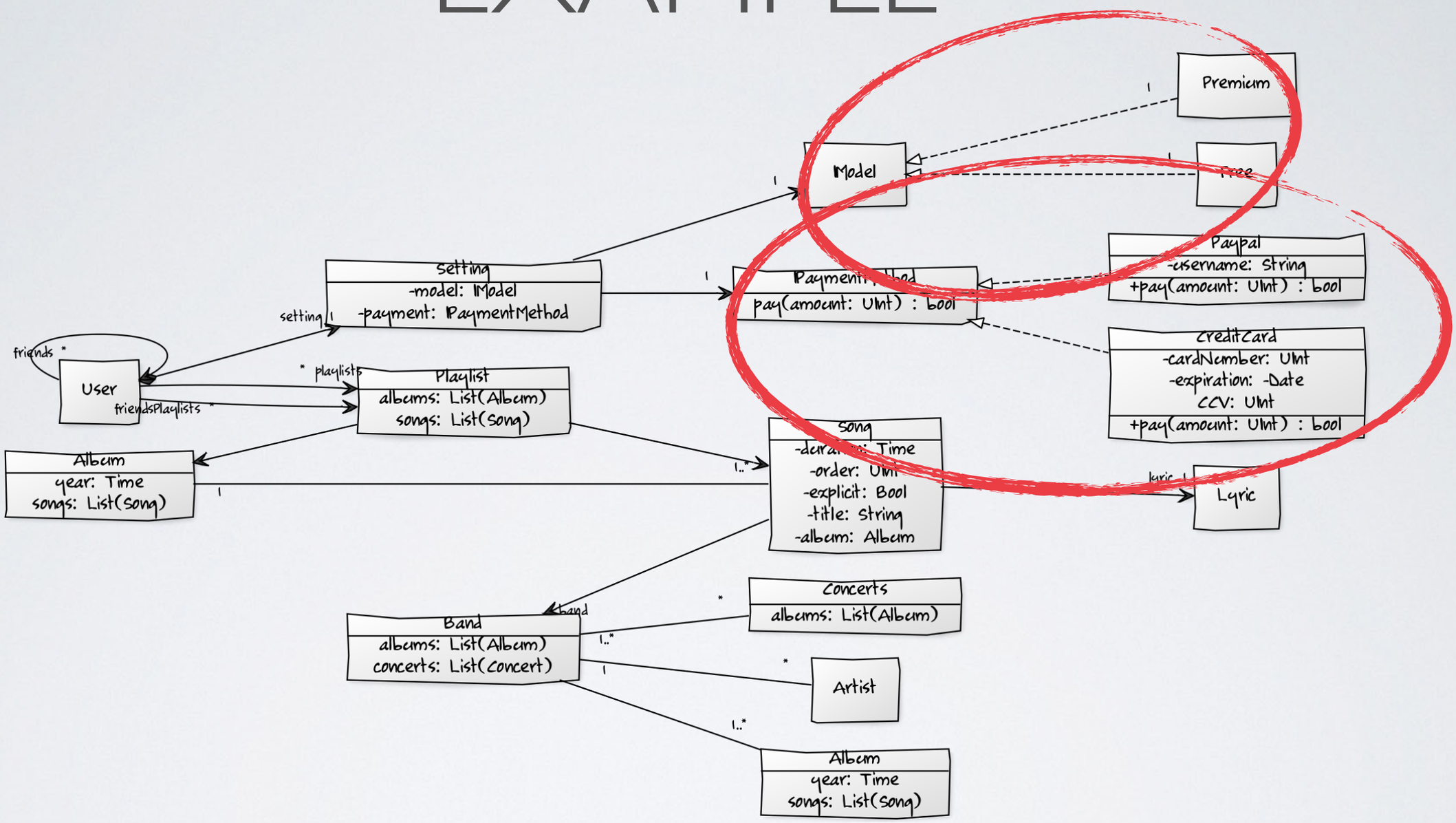
Spotify®

EXAMPLE



Spotify®

EXAMPLE



The *IPaymentMethod* interface (from Polymorphism) allows for future payment methods that may not yet have been thought of.

Spotify®

OTHER APPROACHES TO PROTECTED VARIATIONS

Protected variation mechanisms: data encapsulation, interfaces, polymorphism, standards, virtual machines, operating systems.

Service lookup: REST for Web services.

Law of Demeter: objects never talk to objects they are not directly connected to.

CONCLUSION

- GRASP:
    - basic design principles
        - use throughout any software project
- If in doubt, try to apply them
- No silver bullet
- You will learn more complex patterns

NEXT LECTURE

- Architectural patterns