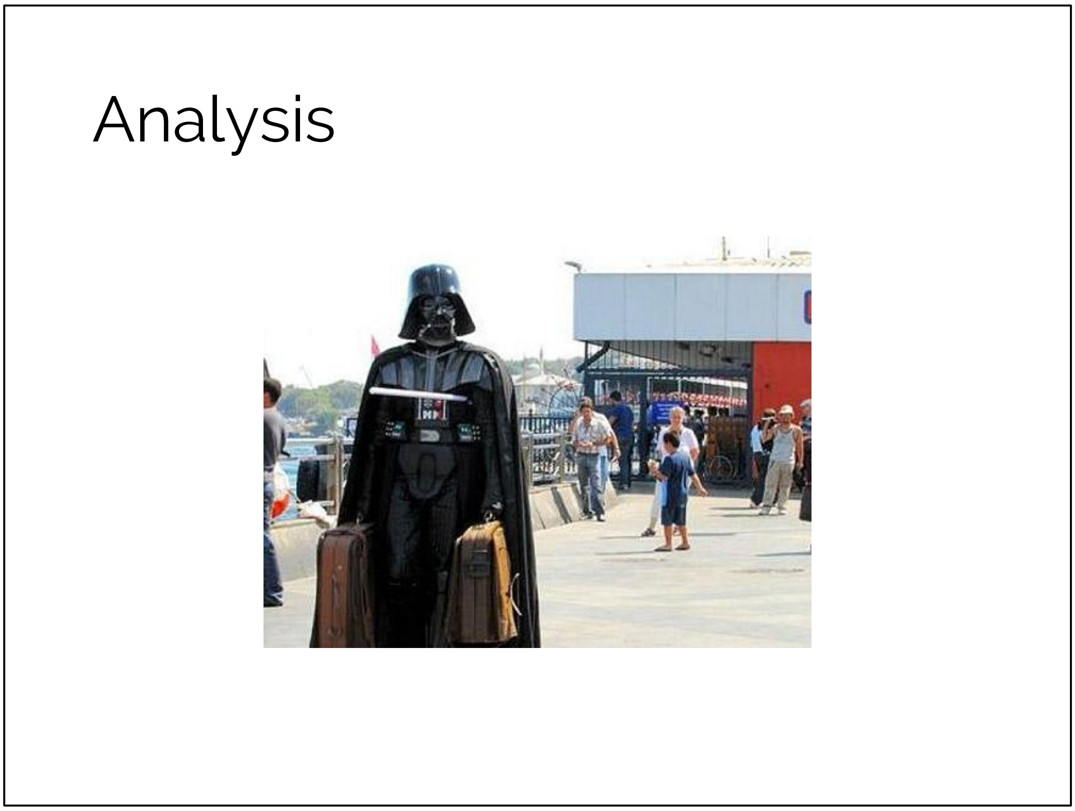
Analysis

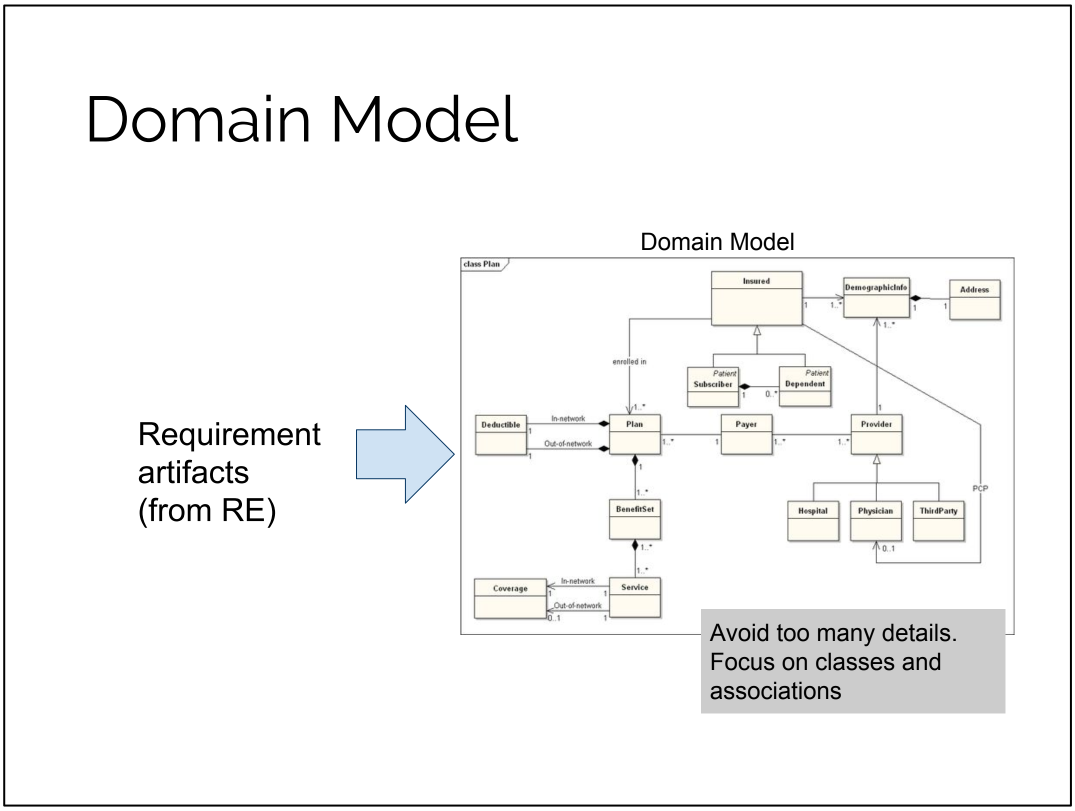
Slide Series #3



Analysis

Analysis is the second phase in the process

- During analysis we try to create a <u>model of the problem domain</u> as a collection of interacting objects
- Picture: Not much to say … bit absurd



Domain Model

Requirement artifacts (from RE)

Domain Model

Avoid too many details. Focus on classes and associations

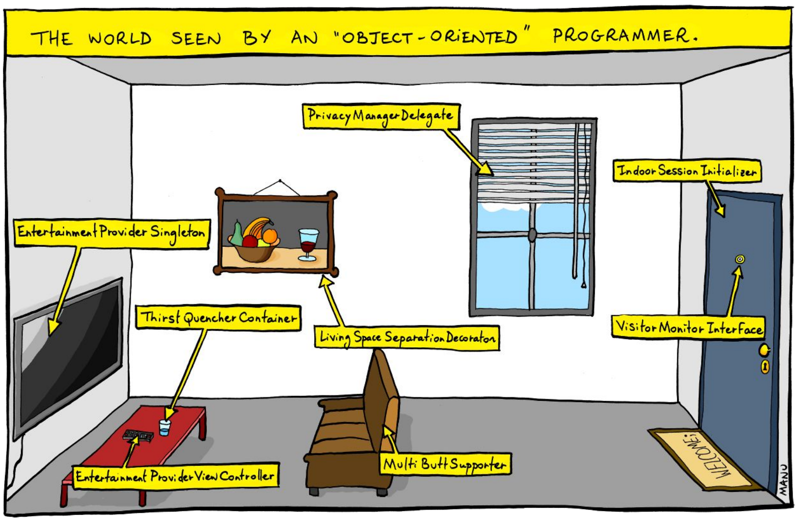
The Domain model

- Is the core of our application (domain modelling)
- The model is an abstraction of some problem
    - What do we mean to abstract? Are you good at abstracting (this is optional)?

Using input form RE, have to find...

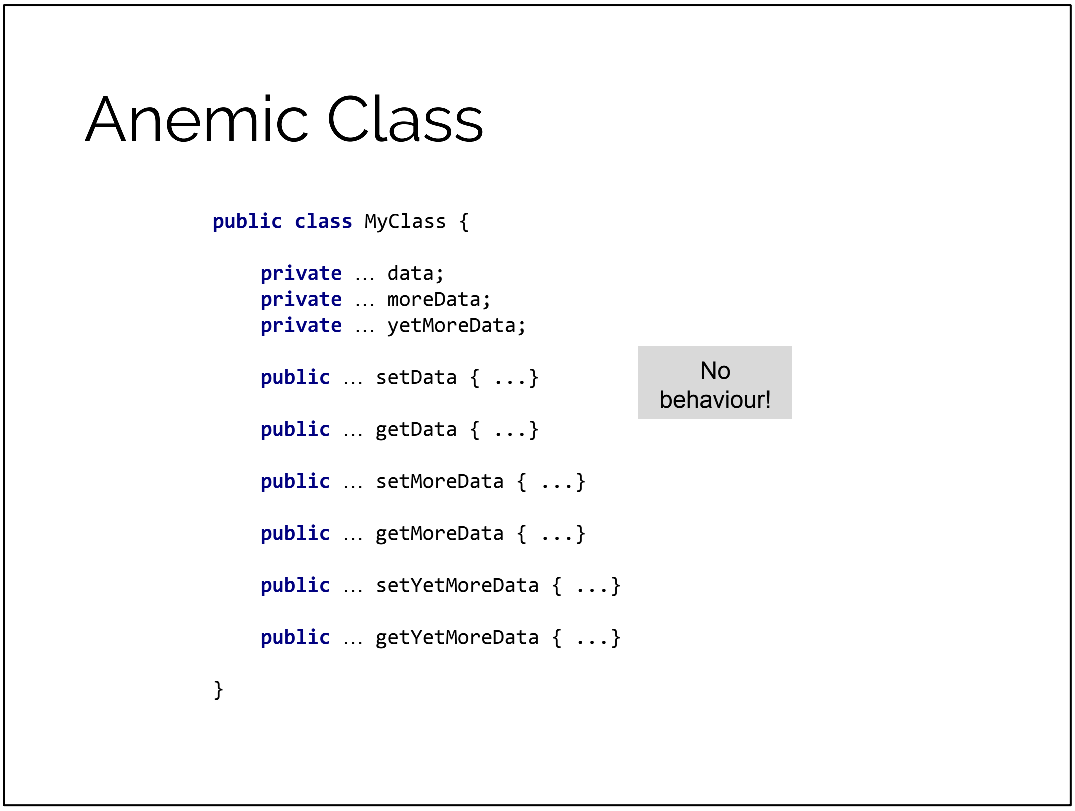
- Objects and how they are related (associations)
- Classes for the objects
- To a lesser degree; attributes, behavior (methods)
- Avoid too many details (inheritance, ...)

Technobabble


THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.
Privacy Manager Delegate
Indoor Session Initializer
Entertainment Provider Singleton
Thirst Quencher Container
Living Space Separation Decorator
Visitor Monitor Interface
Entertainment Provider View Controller
Multi Butt Supporter
WELCOME!
MANU

Remember: Use domain language

- … not technobabble (like in picture)!!!



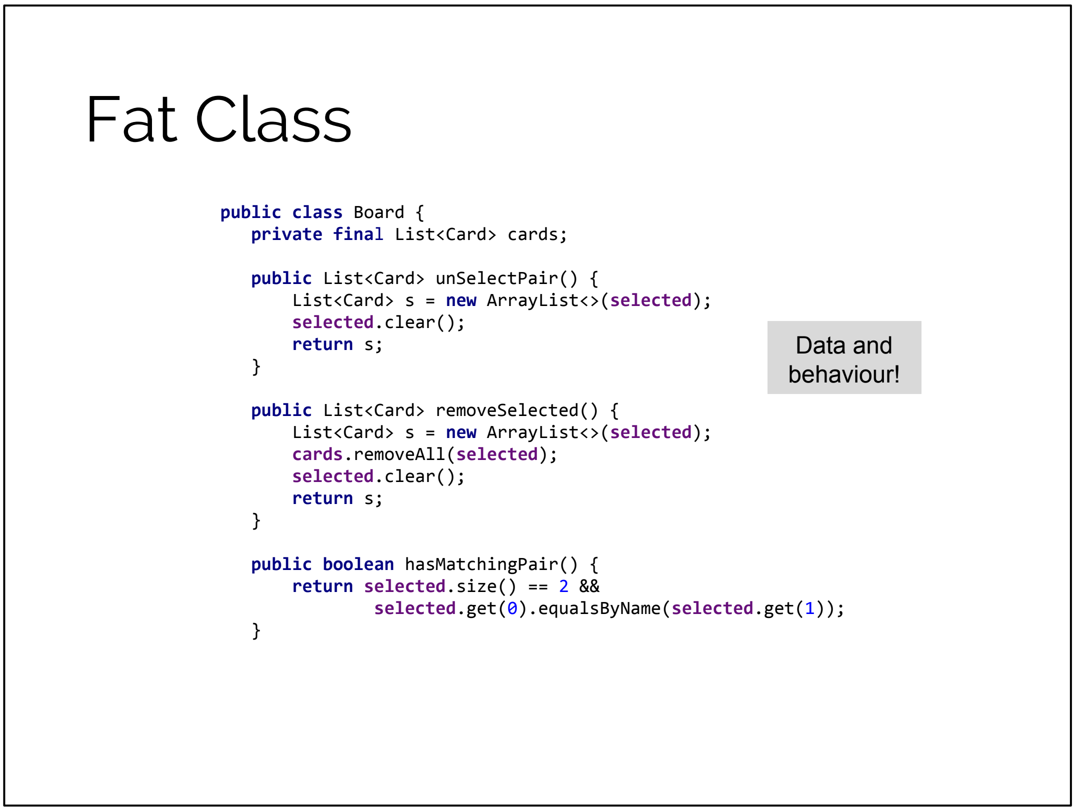
Anemic Class

```
public class MyClass {

    private … data;
    private … moreData;
    private … yetMoreData;

    public … setData { ...}

    public … getData { ...}

    public … setMoreData { ...}

    public … getMoreData { ...}

    public … setYetMoreData { ...}

    public … getYetMoreData { ...}

}
```

No behaviour!

Anemic class

- No behaviour
- Anemic is ok for some data heavy classes (entity classes), …
- … but if all classes are like this, no domain driven design..



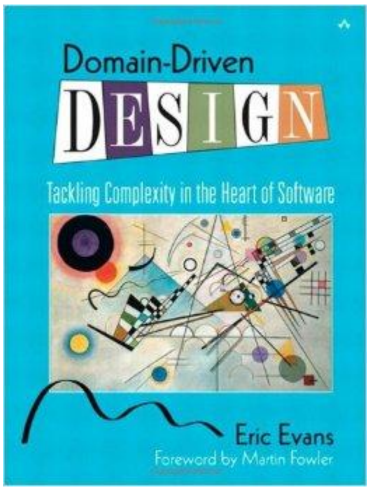
Fat Class

```java
public class Board {
   private final List<Card> cards;

   public List<Card> unSelectPair() {
       List<Card> s = new ArrayList<>(selected);
       selected.clear();
       return s;
   }

   public List<Card> removeSelected() {
       List<Card> s = new ArrayList<>(selected);
       cards.removeAll(selected);
       selected.clear();
       return s;
   }

   public boolean hasMatchingPair() {
       return selected.size() == 2 &&
               selected.get(0).equalsByName(selected.get(1));
   }
```

Data and behaviour!

Class holds data and have behaviour.

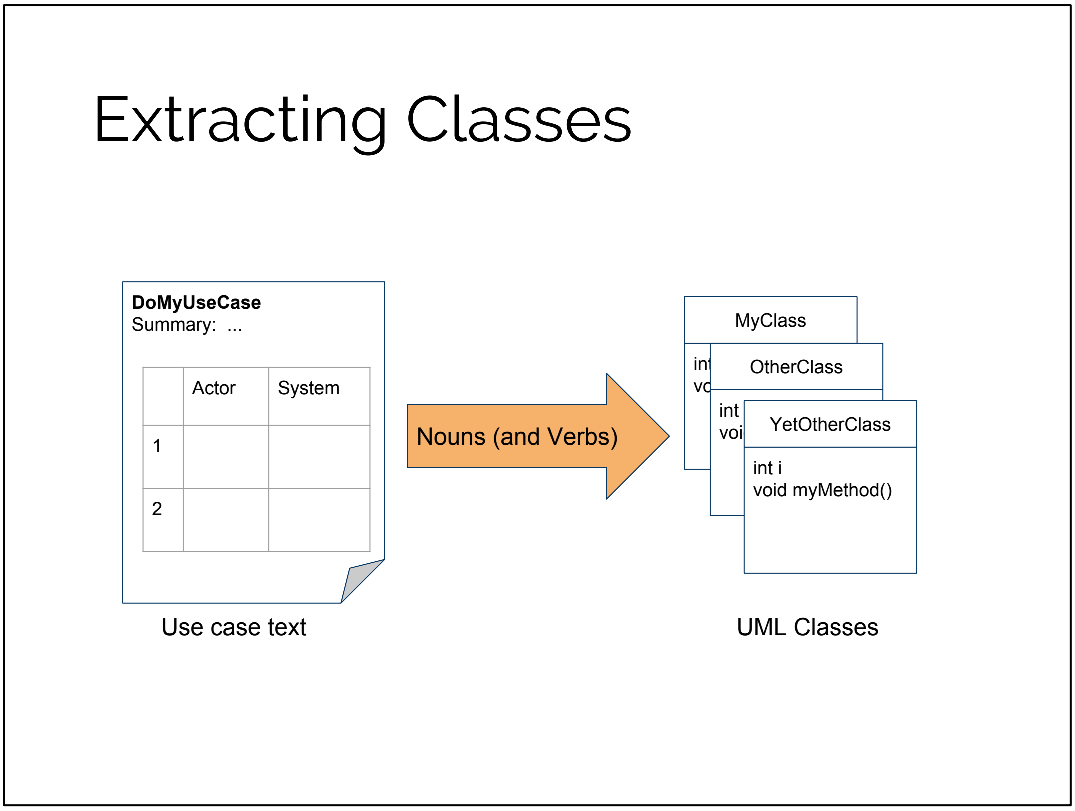
Domain Driven Design


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

During this phase we adhere to [domain driven design]

- Using the language of the domain
- Placing primary focus on the core domain and domain logic,
- Solutions to the problem is in a domain model (implies fat classes)
- Design application on model of the domain.

For short: "Focus on the model" (more later)



Extracting Classes

DoMyUseCase
Summary: ...

	Actor	System
1		
2		

Use case text

Nouns (and Verbs)

MyClass

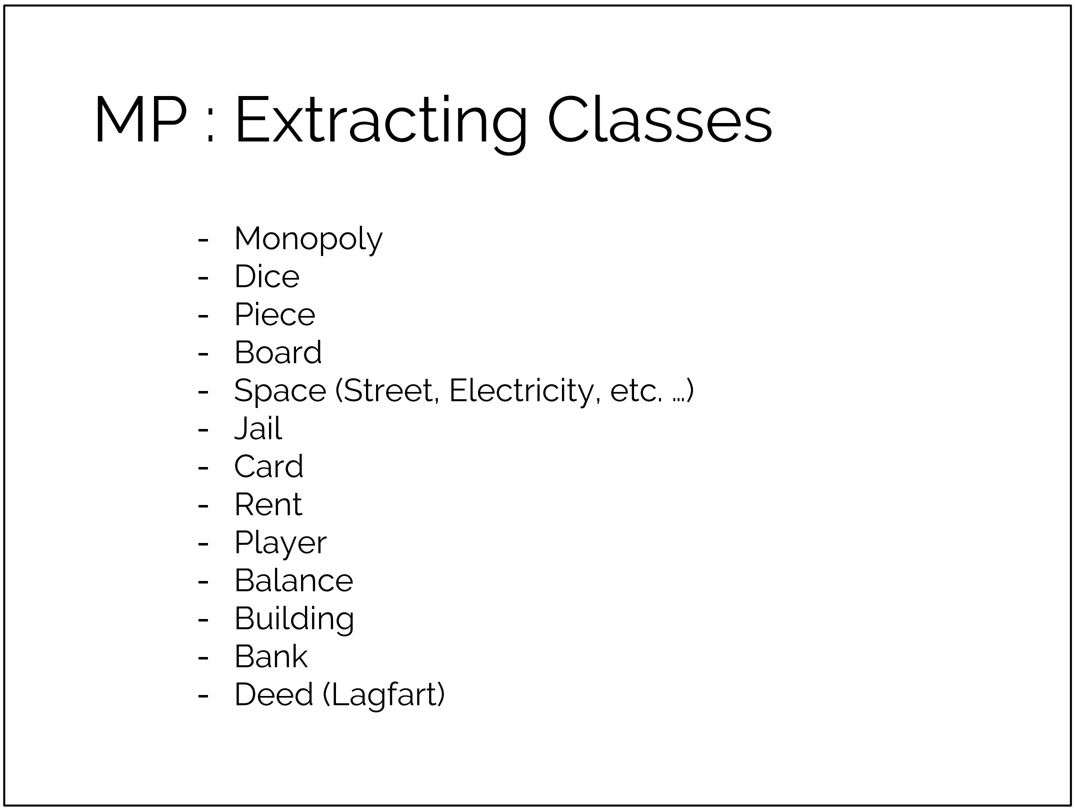
OtherClass

YetOtherClass

int i
void myMethod()

UML Classes

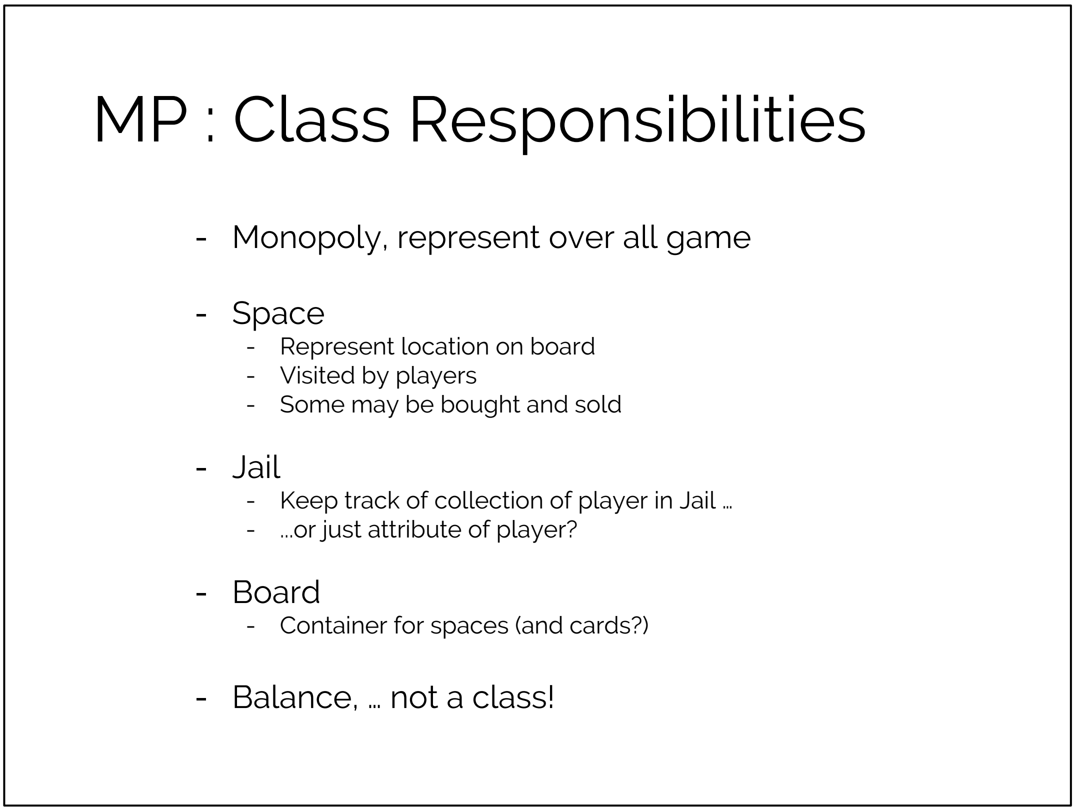
Have the use cases from RAD, simple method:
- Underline nouns in use cases, will become classes
- Underline verbs in use cases will become methods
  - Sometimes hard to know which method belongs to which class …
  - … for now put them in any that seems sensible, more later …
  - … or leave out for now, will show up later!
- Include as much as possible.
  - Easy to skip later, …



MP : Extracting Classes

- Monopoly
- Dice
- Piece
- Board
- Space (Street, Electricity, etc. …)
- Jail
- Card
- Rent
- Player
- Balance
- Building
- Bank
- Deed (Lagfart)

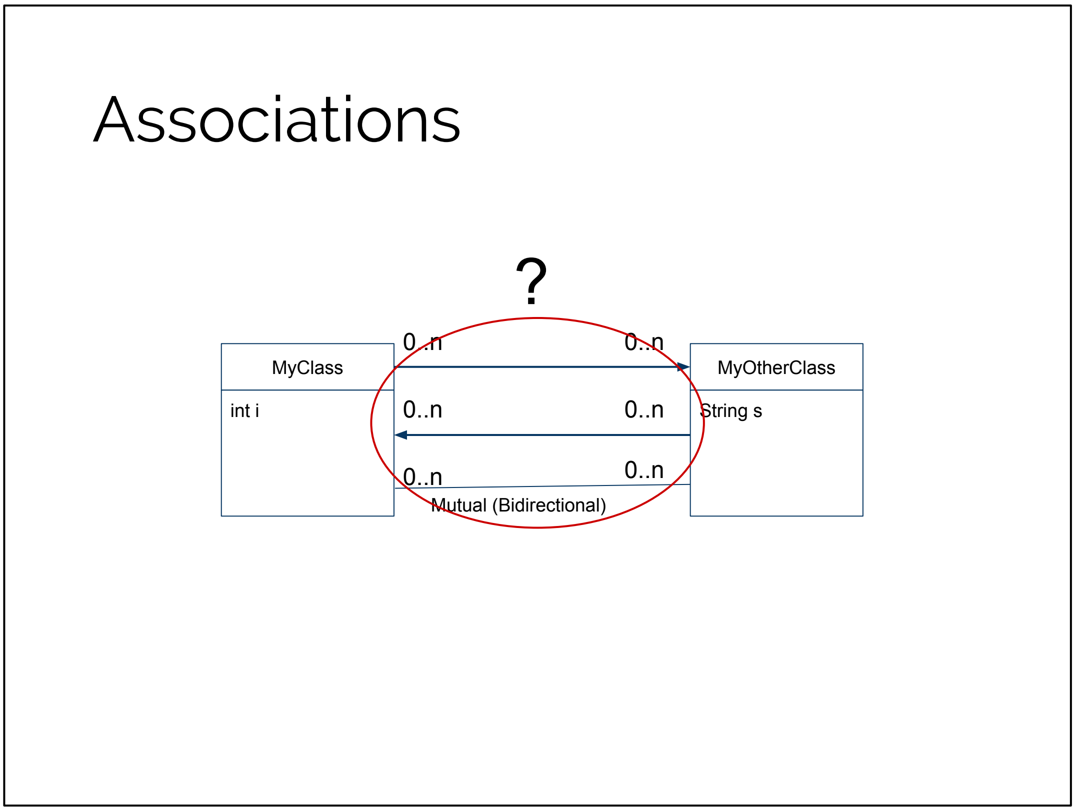
Not all of this will end up as classes
- Some will be (atomic) values



MP : Class Responsibilities

- Monopoly, represent over all game
- Space
  - Represent location on board
  - Visited by players
  - Some may be bought and sold
- Jail
  - Keep track of collection of player in Jail …
  - …or just attribute of player?
- Board
  - Container for spaces (and cards?)
- Balance, … not a class!

Try to formulate!
- If hard, class possible not a class, or not needed … or..
- Atomic values not classes (except for technical reasons like Integer)



Associations

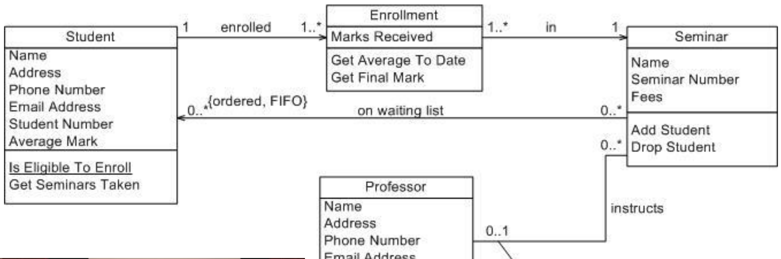
Have found many classes from nouns. Possibly some methods from verbs.

- Must find associations to create a class diagram
- We don't distinguish between association, composition and aggregation.

How to find the associations?

- Examine use cases: has, owns, knows, is a, has a, ...
- Visualize the real (physical) situation
- Possibly: Skip (or add a few) directions for now, just note the association

UML Class Diagram

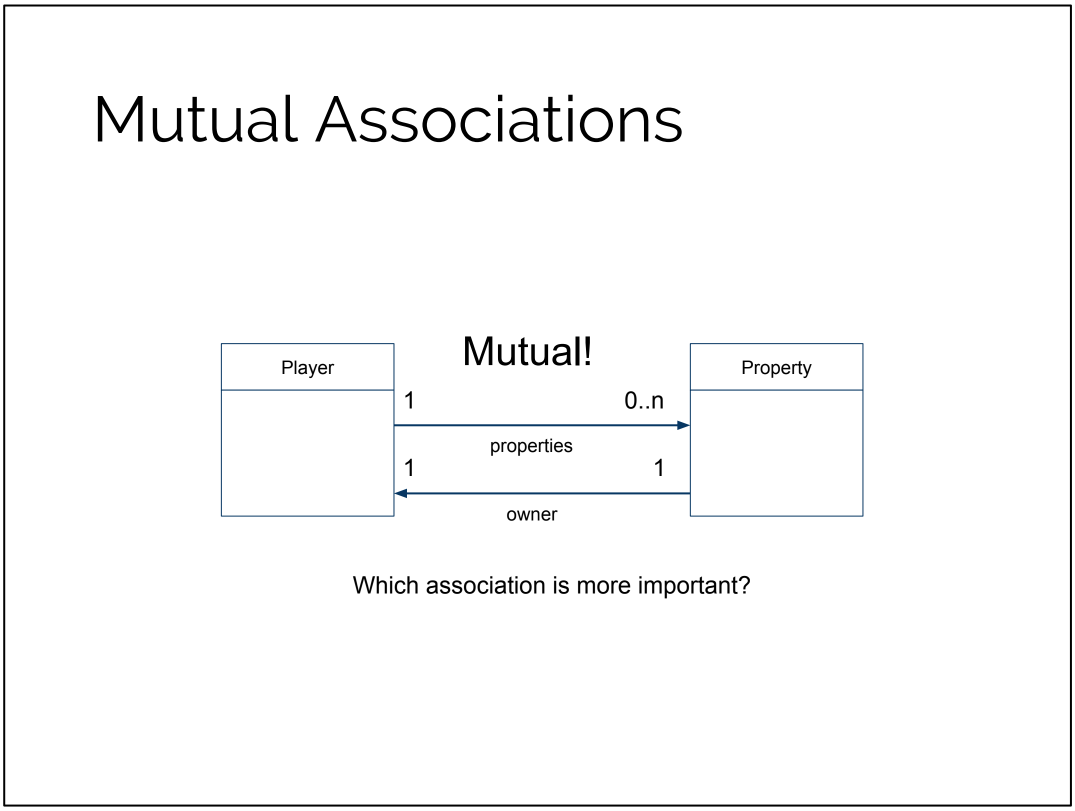


Model represented as an UML class diagram

- Possibly have to break down
    - Also see Package diagrams later
- A static view
- NOTE: Associations and multiplicity is between objects

The diagram has a meaning.

- Symbols, notations etc should end up as runnable code!
- Exercise: Transform diagram to Java!



Mutual Associations

Mutual!

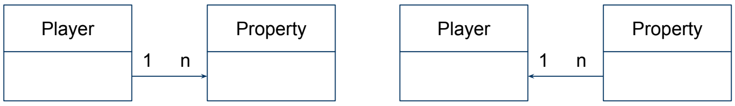
Which association is more important?

Mutual (bidirectional) associations are bad (or at least we avoid)

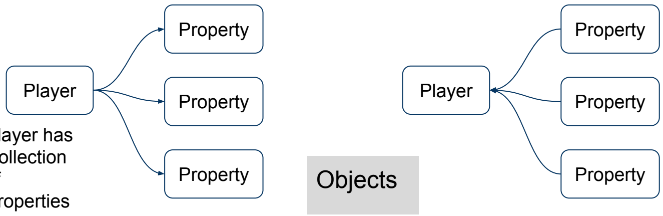
- Must keep two object in synch (reference each other) i.e. if new owner have to change 2 references
- Domino effects (change one, affect other)
- Classes not understood in separation

Select association that seems to be used most, remove other.

Multiplicity and Direction



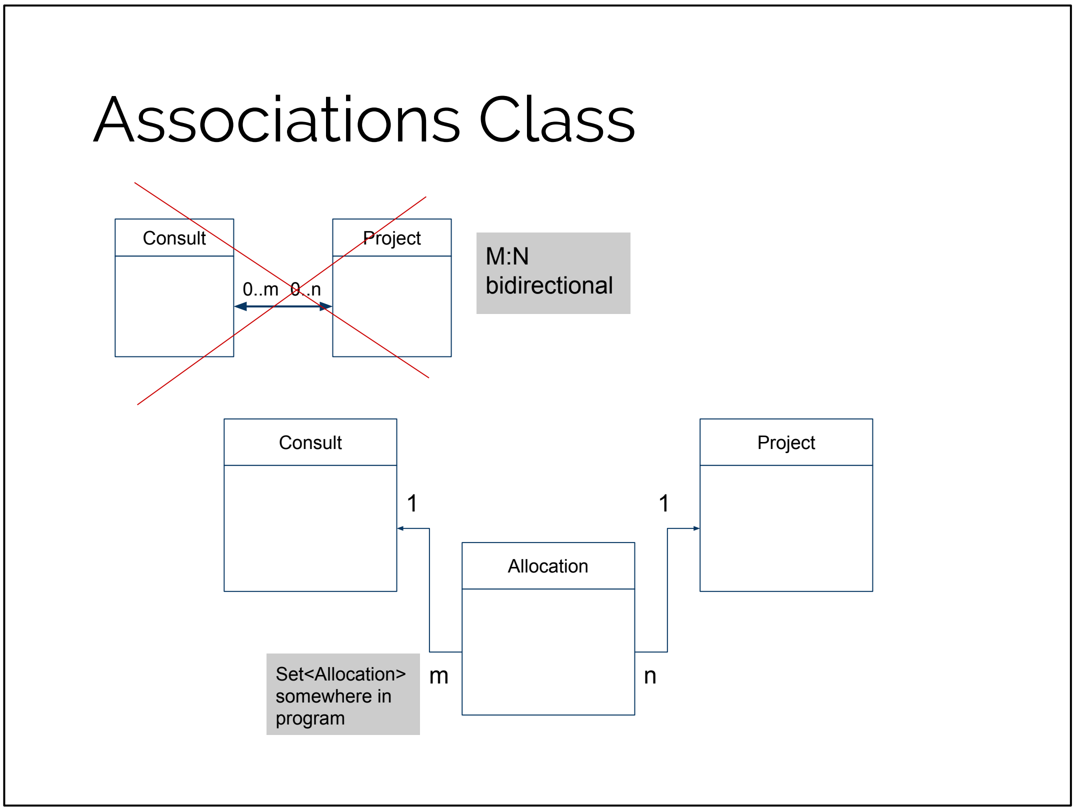
Same multiplicity



Player has Collection of Properties

Objects

Each Properties has reference to same player



Associations Class

M:N bidirectional

Set<Allocation> somewhere in program

If mutual and many to many association.

- Create an association class (Allocation)

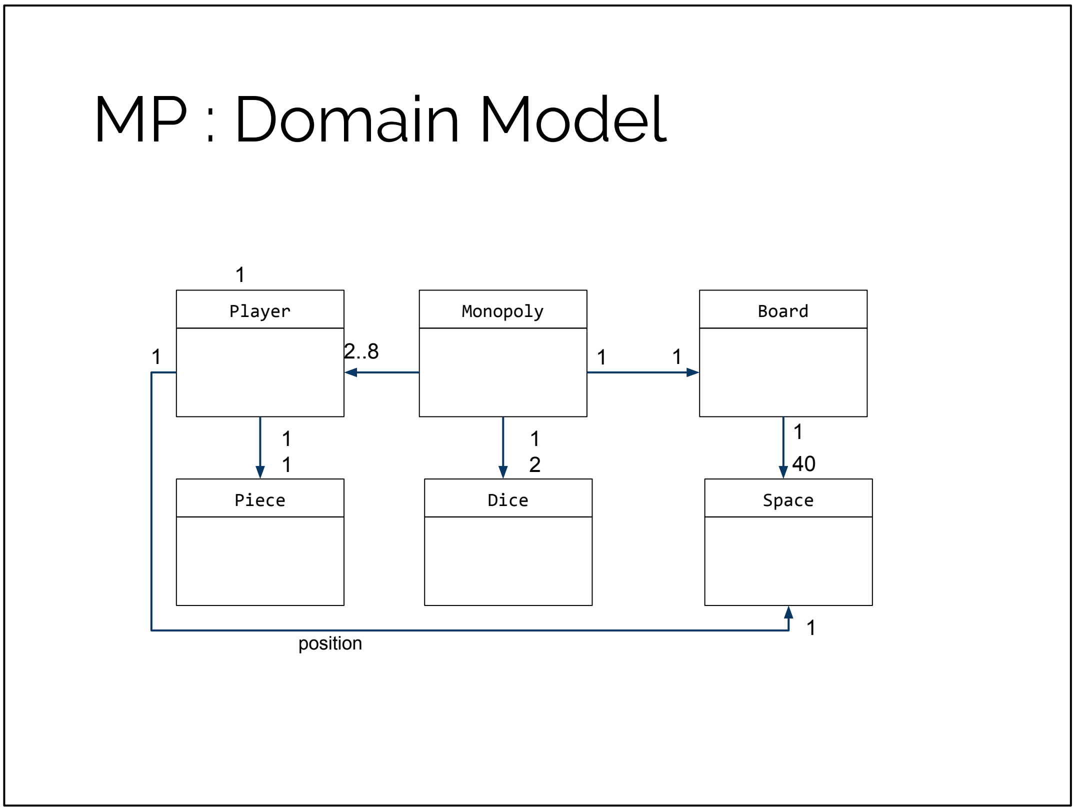
MP : Associations

Some considerations

- Player has (shared) dices or …
- … is it Monopoly that has dices?
- Board has spaces or does a space reference the board?
- Player has position (a Space) or a space has visitors (Players)?
- Player owns spaces or space has an owner?

Which directions seems most natural/useful?

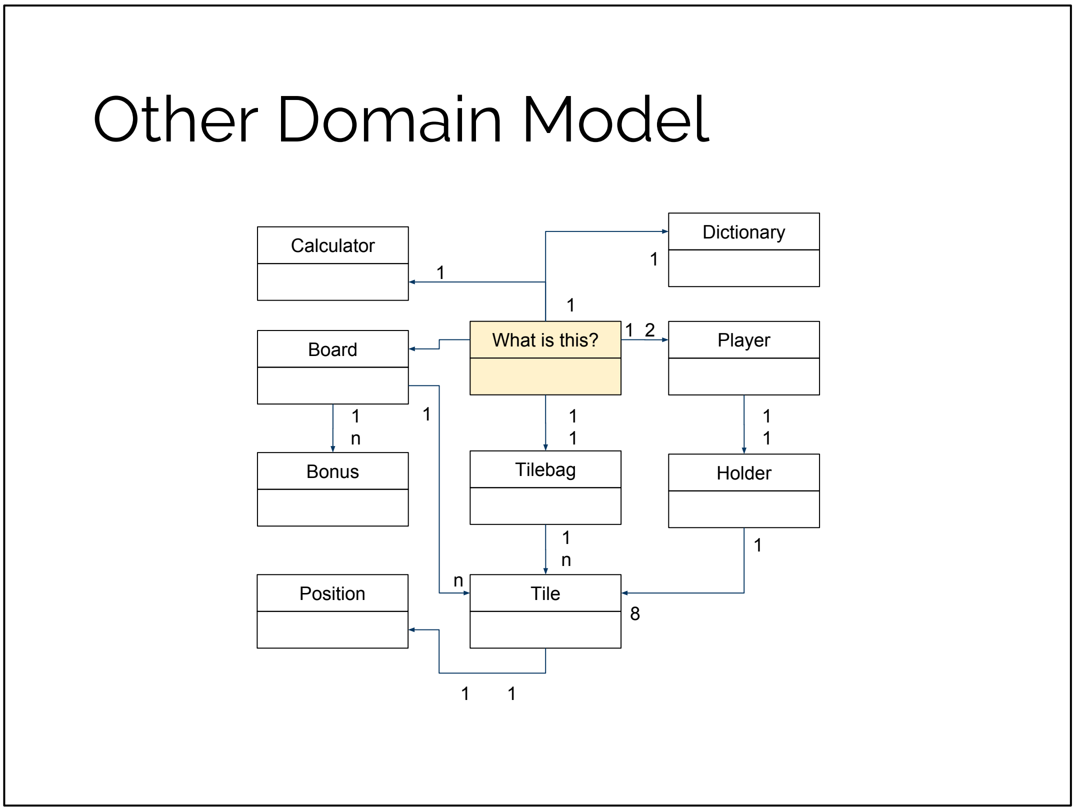
- What questions do we need to answer?



MP : Domain Model

The first domain model (iteration 1)

- Targeting the highest priority use cases: Move and End Turn
- I.e. here are the (minimal number of) classes we need to be able to run the use cases (hopefully?)
- Remainder: This is a model of the domain NOT the full software

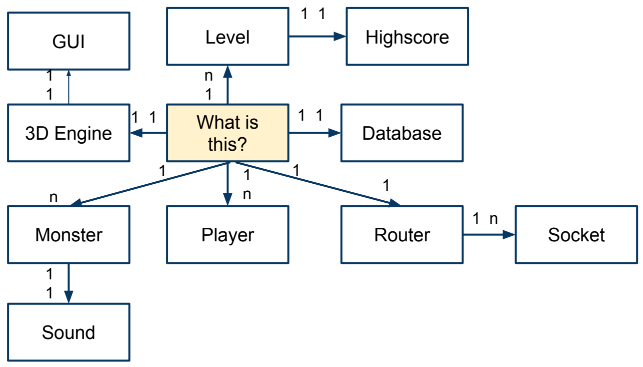


Other Domain Model

What is this?

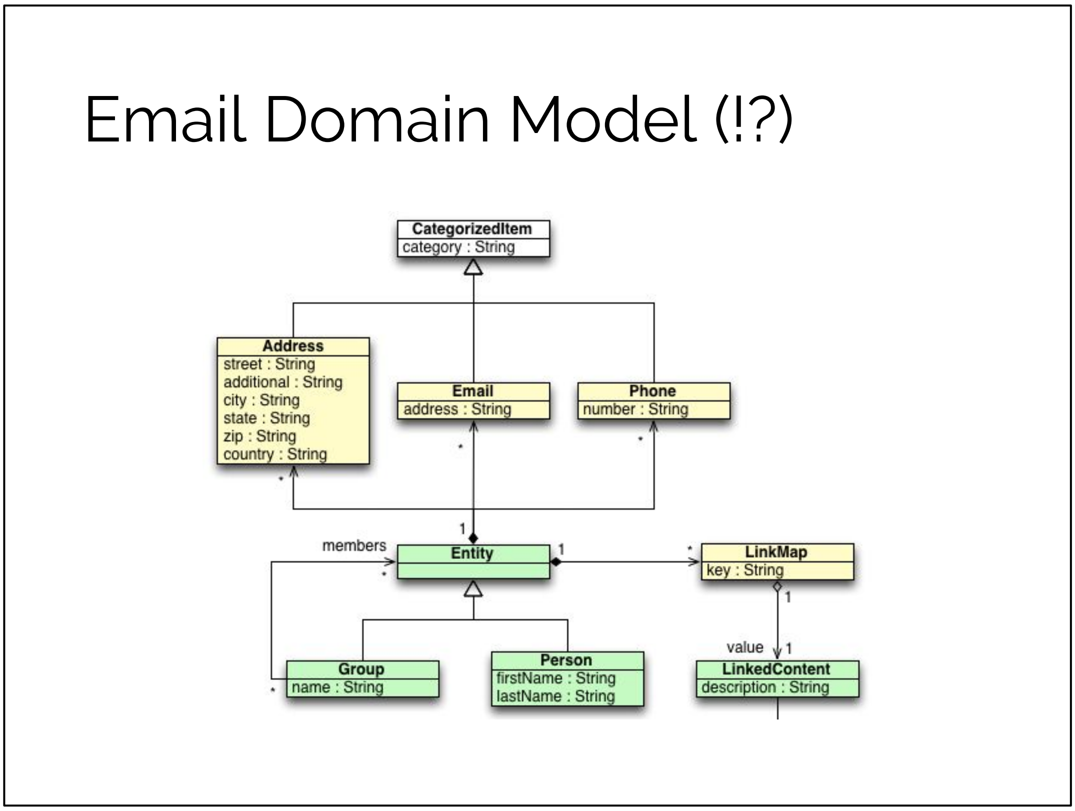
- The model should be able to communicate something!

Yet Another Domain Model



What is this?

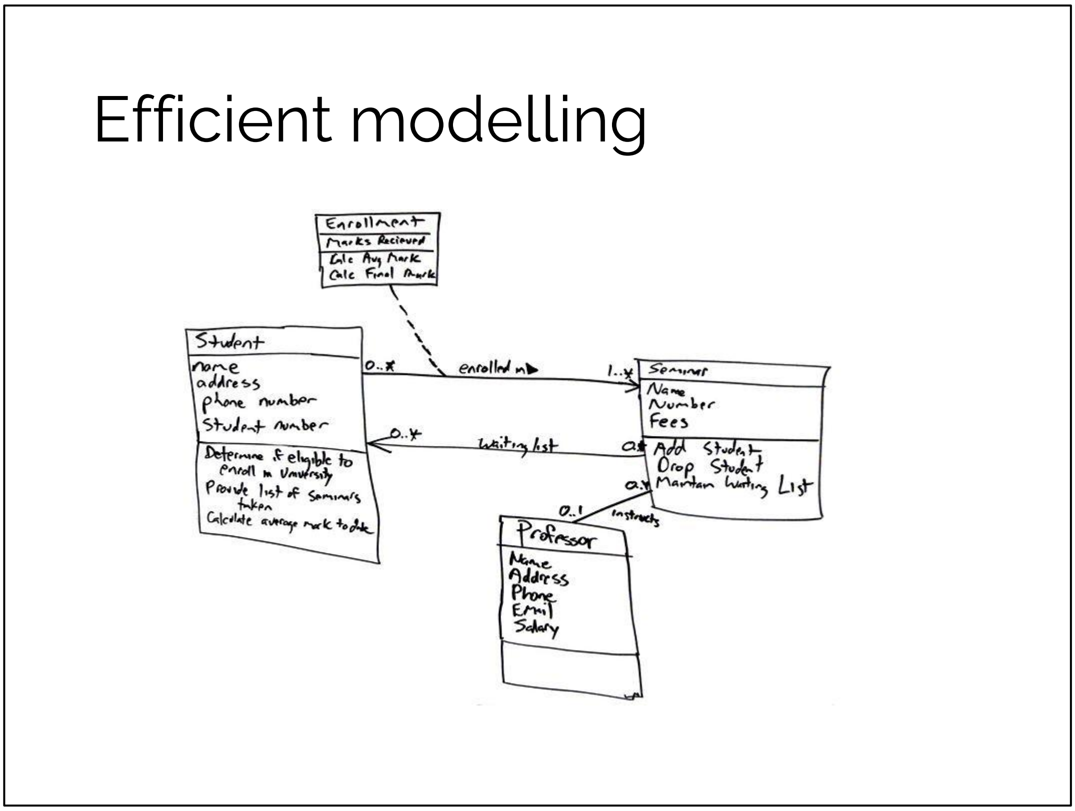
- This is NOT a model of some problem domain …
- … it's a mess of technical details and domain concepts



Email Domain Model (!?)

Found on the Web

- “The UML diagram for the model created in this tutorial is:...”
- What’s you opinion ...?
- My opinion: Strange (bad)?!?
    - Where is sender, receiver, inbox, outbox, what is LinkMap ... in domain language?



Efficient modelling

Optimal is to first draw on whiteboard!
- Very fast drawing
- Very fast communication, everyone can participate
- Use phone/camera to document

Later, Tools to draw UML
- When model getting more stable
- UMLet plugin to Eclipse, fastest possible
- Linux : Dia
- Mac/Win? ...

Important Object Characteristics

- Unique identity?
- Equality?
- Immutable?
- Persistence?
  - Will any objects survive the execution of the program?
- Lifecycle.
  - When is object created?
  - How long does it exist?
  - When destroyed?
- ... other ...

Some valuable characteristics to note for the model objects (if applicable)
- Use stereotypes to annotate (example <<Persistent>> for surviving objects)

MP Characteristics
- Player names must be unique!
- Space names must be unique!
- No objects will survive ... (for now)
- All objects (model) created at start of game, exists until end.
  - How does this affect the players?
- More .. ?!?

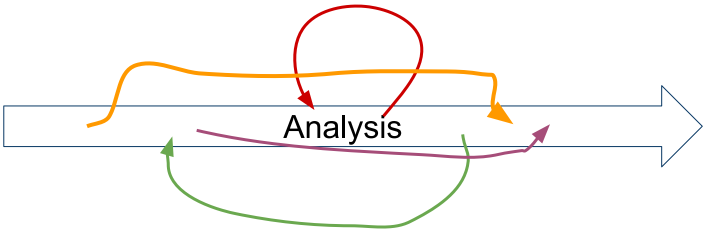
Finishing RAD

4. Domain Model ✓
4.1 Class responsibilities ✓

From previous phase we have recorded requirement elicitation in the RAD

- Analysis (i.e domain model) is also documented in RAD!

Analysis: Real world version



Same as for RE!

- It's not linear!

Summary

Analysis focus on building a domain model

- We used the requirements from RAD (use cases) to extract the domain model
- We expressed the model as an UML-class diagram
- We documented model in RAD

Next: From domain model to first implementation