Design and Implementation [Iteration 1, Phase 3 & 4]

Slide Series 5

Must have Something to Run Week 3

The task for now is to create a very basic runnable version of the model

To be able to fulfil this we need to simplify

- Only a small part of application implemented (i.e part of the model)
- Normal flow, no exception handling, simplest possible IO, no MVC, no subsystems, hard coding values, everything in same package, ...
- Probably clumsy design

Need a running version to deepening our understanding, as a start point for further explorations

Remainder: Domain Driven Design

We don't bother about all the services the model needs to become a fully functional program. Not part of the core solution...!!

- Will blur the model, leave out for now...

We focus on the model

- The model is the solution to the problem!

Design: Starting Out

From RE and analysis in RAD we have

- A few high priority use cases
- The analysis model (class diagram)
- A GUI (mockup or possible some implementation)

We pick 1-2 high priority use cases and classes involved!

Developing the Design Model

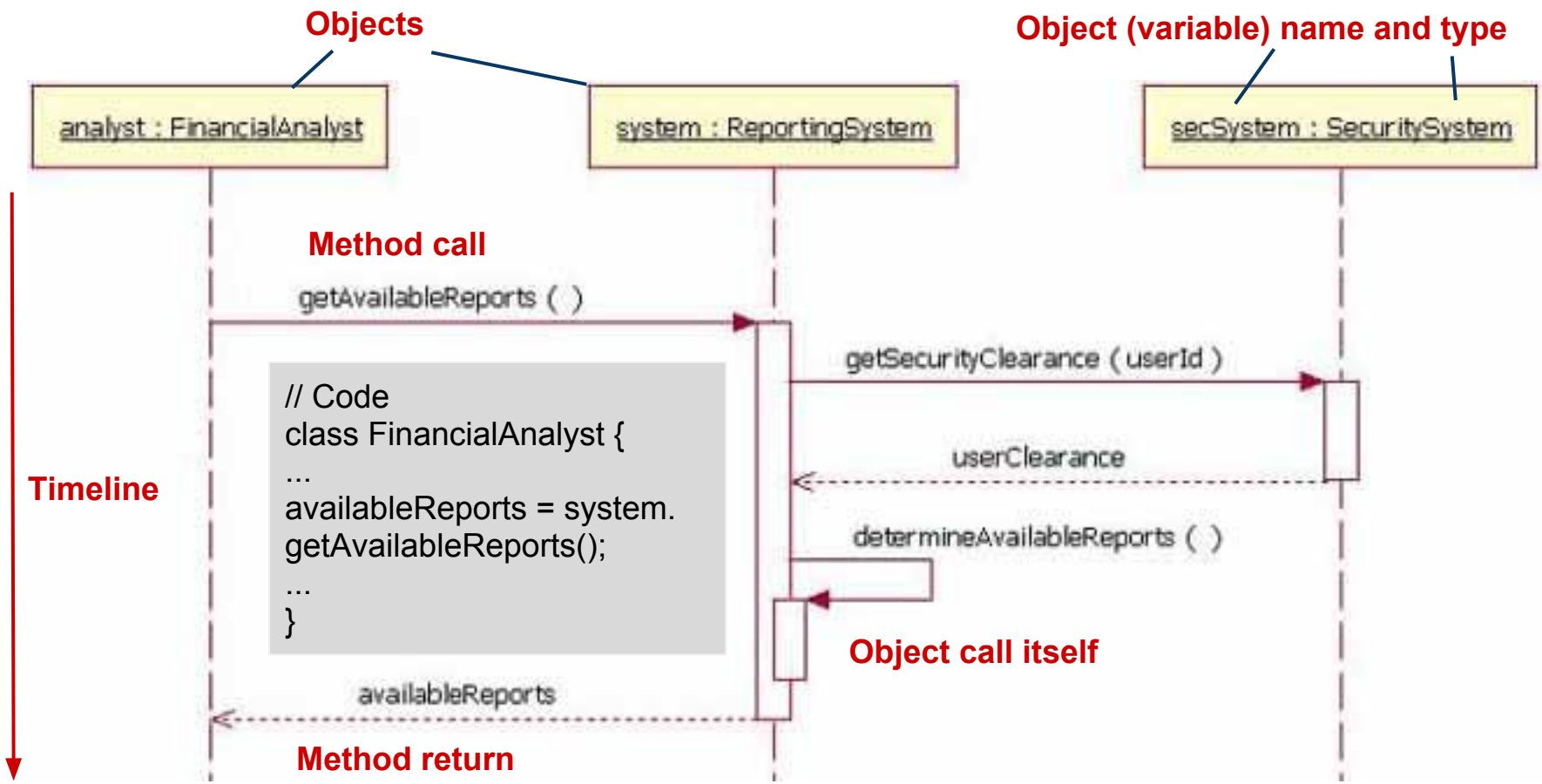
Common techniques

- Using UML sequence diagrams, upcoming ...
- Prototyping (quick'n'dirty coding)

...the above interact

- diagram gives overview
- prototyping clarify details, use in parallel

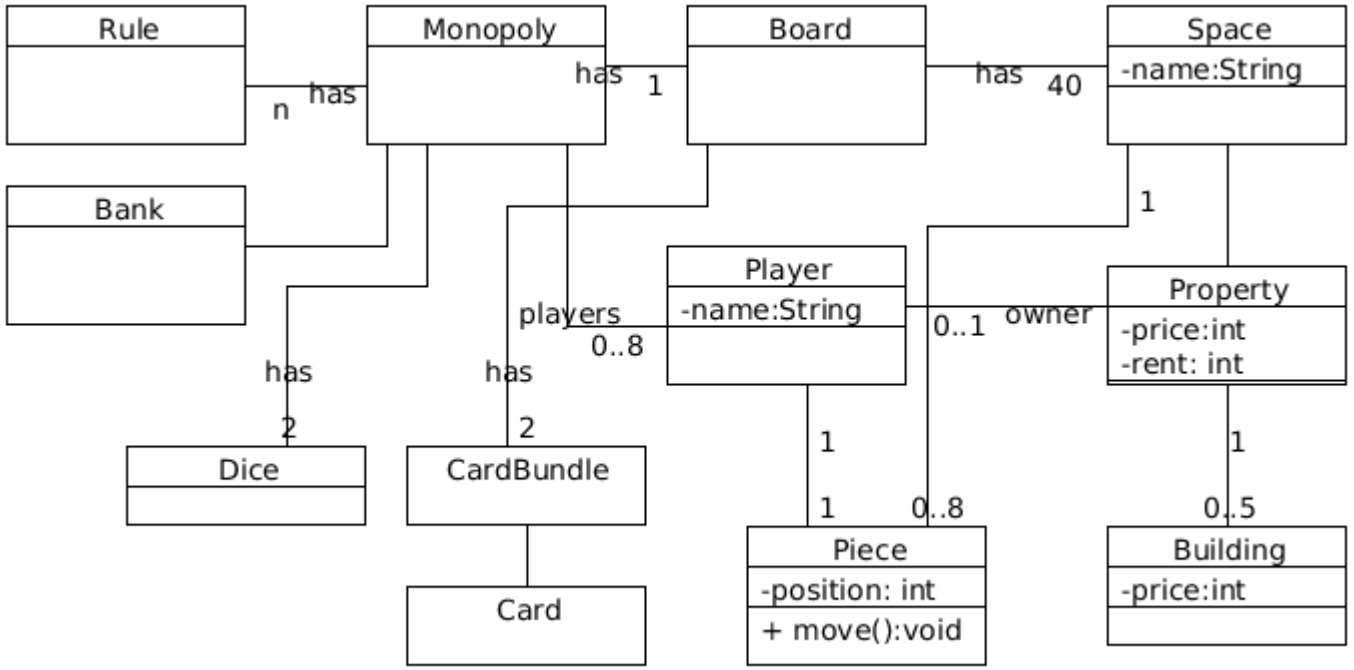
UML Sequence Diagram



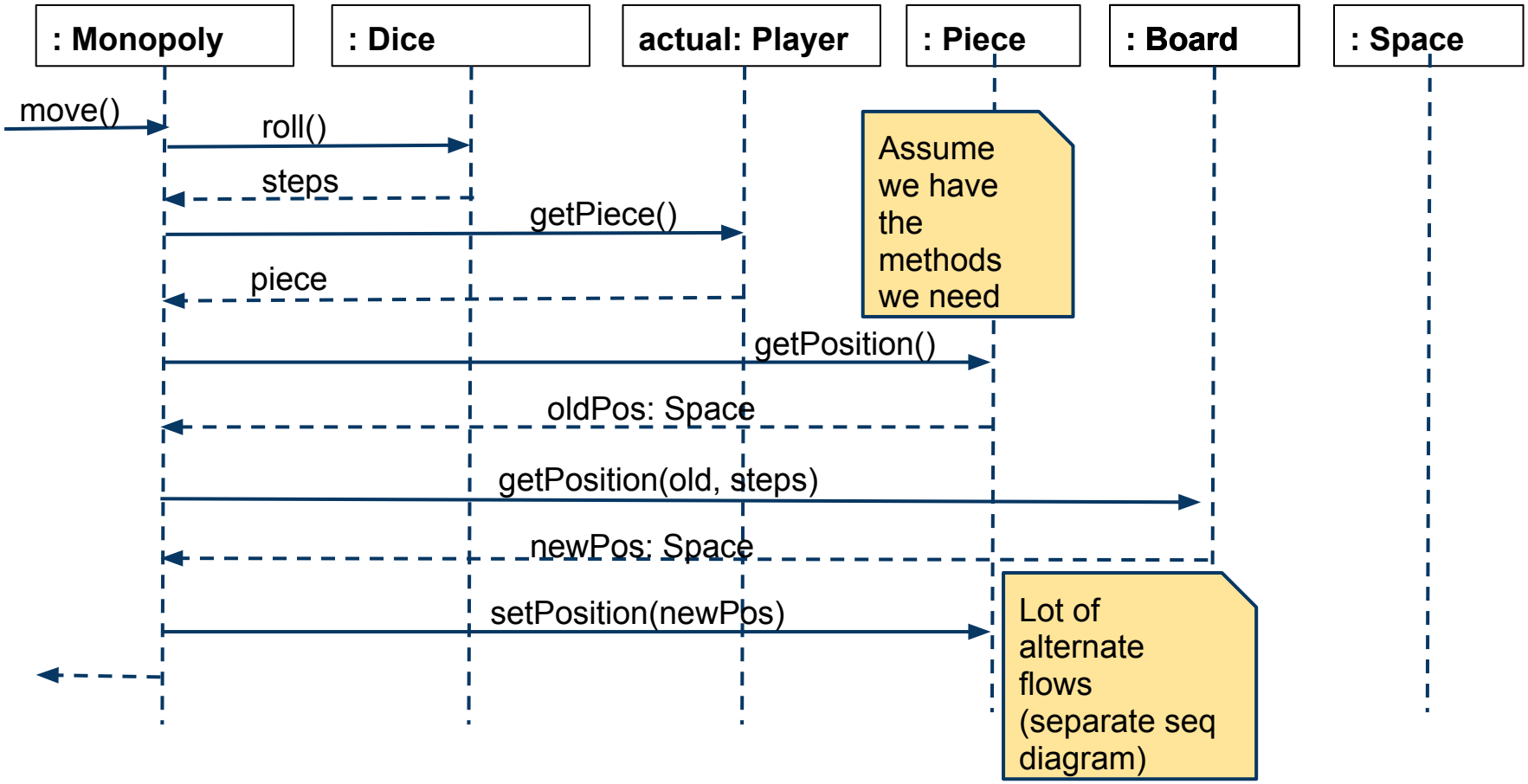
First Running UC for MP

Selected UCs: Move (first to be implemented), EndTurn
Classes: .. at least Piece and Board, we'll see...

Analysis model from RAD



Sequence Diagram for UC Move from MP



This is one way to do it, there are others...

Final Step to Run MP UC Move

We'll run the UC's as a JUnit test

- Not a real test just a simple way to try out some UC's (alternative; use a Main class, with public static void main)
- Add constructors to connect objects
- Hard code smallest possible model (in test setup)
- Input: method calls hardcoded in test methods
- Output: override toString() using System.out.println() (should not be used in "real" test just for now..)

DEMO time...version 0.1

General Style of Interaction

This is what we try to achieve (more later)

```
// Interaction loop
while( !finished ){
    get input
    call model //supply input, change state
    get model state
    display state in GUI
}
```

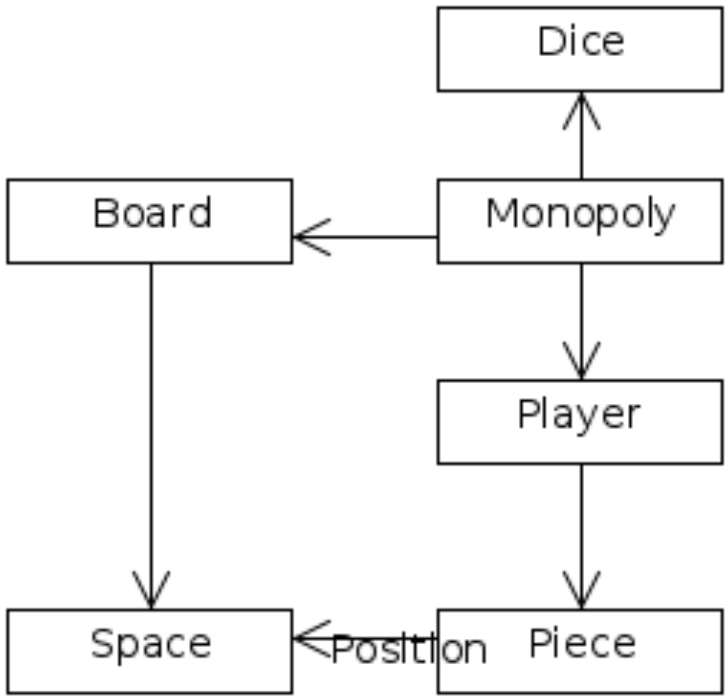
New insight!

In general we have to record all state changes

Example: If player hits space owned by another, should record a debt (later in GUI inform player, need user interaction, player possible must sell), **new model class Debt**

Design Model for MP 0.1

From class diagram, sequence diagram and code we get this (good no mutual or circular associations!)



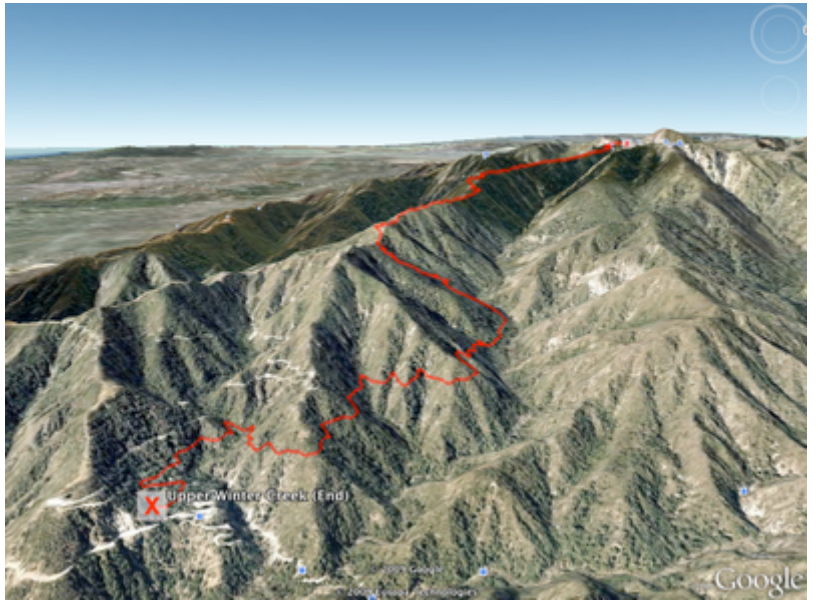
Other insight!
Need to remember dice result i.e. a **stateful Dice**

Design Reviews

When first version running we review the design

- Possible to continue implementing UC's, or should we redesign?
- Often hard to say... if no idea have to just continue...
- Goal: Have a mostly **stable design** after 2 iterations
- You will feel when design is stable; adding, modifying and extending the application will be "downhill"

Hopefully we haven't done a bad analysis, ... but if so have to go back... (analysis model must be stable before design model)

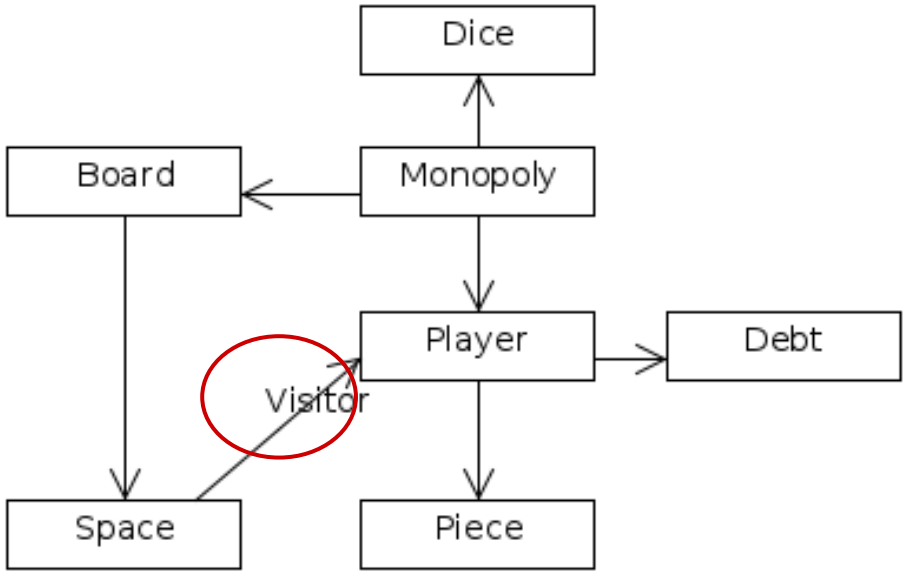


Design Review MP

Now we review the MP design?

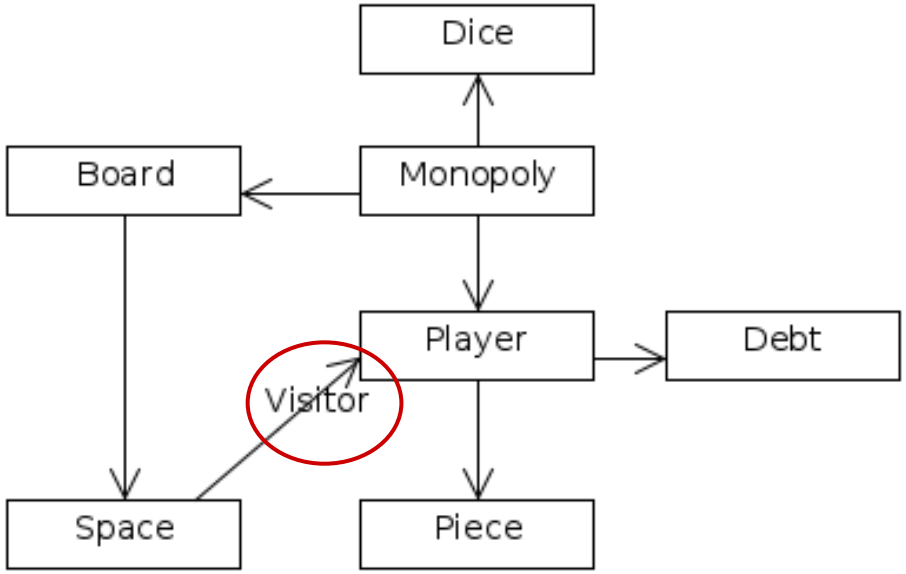
- Spaces have a visual ordering, no ordering for now (needed?)
- How to get model state to GUI? Possible the below design model is better (board holding near all data for GUI)
- **Demo:** Monopoly 0.1-alt

Would like to be able to easily get a read-only "snapshot" of model state to display in GUI
Is this a solution??



UC Move with Redesigned Model

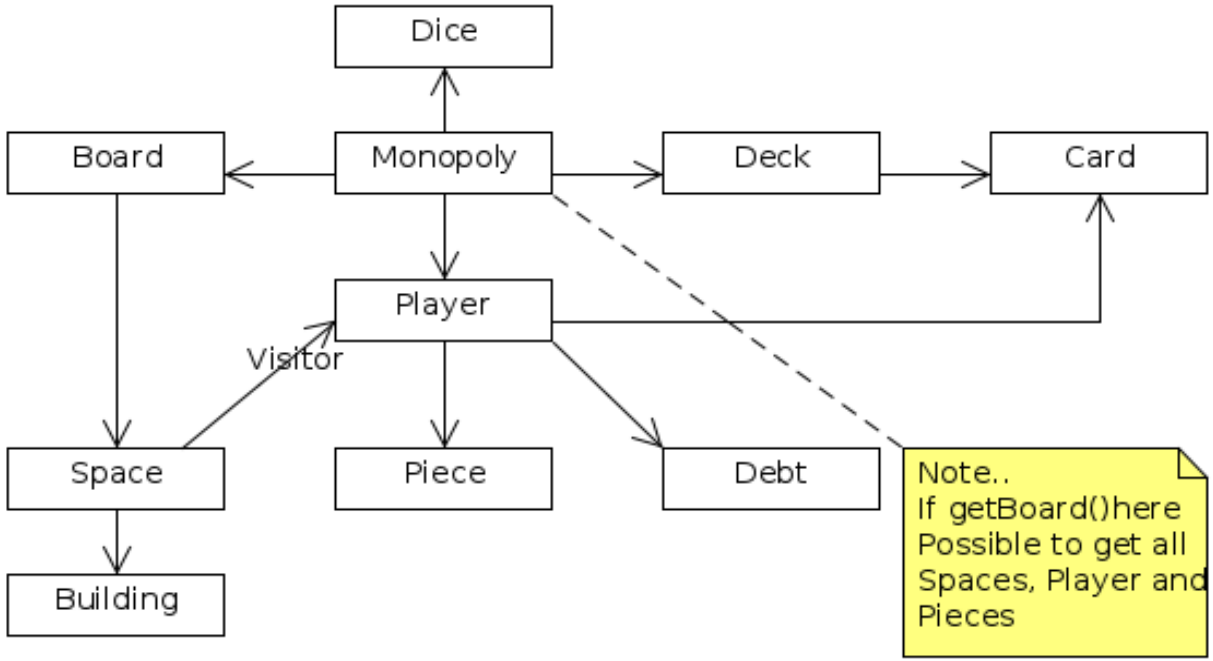
You try... a sequence diagram



Expanding the MP Model

When first UC is up and design reviewed we try to expand the model

- Add another UC: EndTurn
- **Demo:** Monopoly 0.1alt (using previous slide model slightly expanded)



Refactoring

During implementation we continuously refactor

Monopoly case:

- Could move absolute (if a card says so) or random (dices)
- Need two different move-methods with lot common ...
- .. do some refactoring, simple example in Monopoly0.2

Reminder: Test Driven Development

So far have used tests to get model up and running

- Remainder: Could have done otherwise

Assume design model getting more stable

- During more detailed development of classes and expansion of model each unit should have tests (non-trivial methods, real tests i.e. no System.out, just a boolean outcome)
- Remainder: Test code in separate source folder using same package structure

First GUI Version

Have a GUI from RAD

- Possible a simple basic implementation of GUI from analysis
- Identify the basic input/output elements needed to get the use cases to work
- Implement what's needed (if not done)
- As simple as possible!

First GUI Version MP

Partition application into distinct parts

- Model (in one package)
- Ctrl-package, for controls (one package)
- View-package, for GUI related, classes (one package)

No real MVC model, just trying to use GUI and model together

- GUI interaction with model, how to?
- Possible discover missing features in model

Also more serious use of packages (full hierarchy)

First GUI Version: Support Classes

Introducing Factories

- Separating out complex construction from use, MonopolyFactory-class returns complete model!

Dedicated Main-class for main-method (i.e. handle application start)

Restricting interface to Board, IBoard

- Just a getSpaces()-method
- This is a way to promote the "snapshot" of the model

First GUI Version: Demo

.. and inspection of Monopoly version 0.3

Benefits of OOA/D/P: Tracing

If using DDD it should now be possible to trace the development

- Running objects (classe) should be able to trace backwards, possible all the way to the UC's
- Because we use the domain language, names should be the same all the way, classes, attributes should be evident, they have a meaning in the real problem (domain)

Ensuring Quality

Have the JUnit tests, also recommended to use "code coverage"

No circular (mutual) class/packages dependencies!

- Use a tool to check, recommended STAN (plugin to Eclipse)

Other tools

- Findbugs, ... (also as Eclipse plugin)

If using these regularly the project quality will increase!

Ultimate Question

Is the model stable? Or...

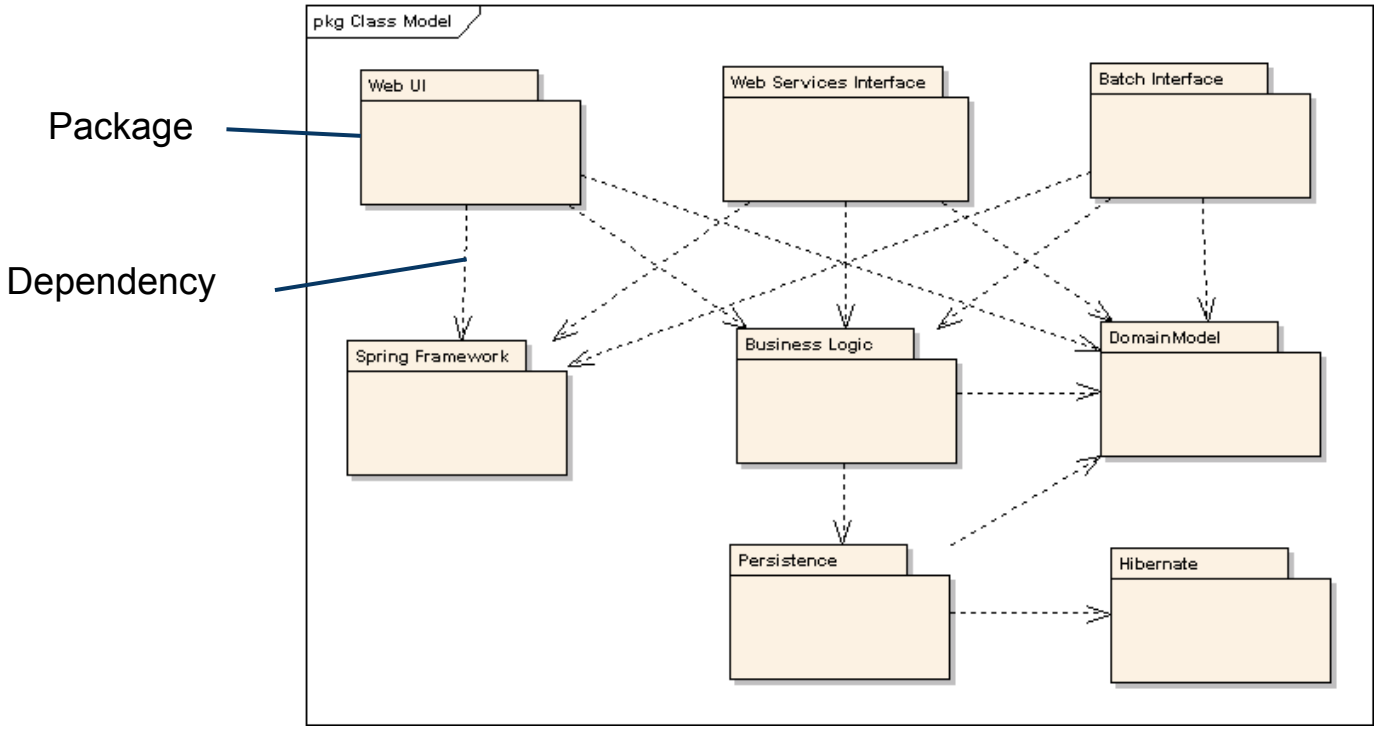
- Wrong
- Not complete
- Inconsistent, ...

```
while (unstable) {
   // Stay here and work out any problems!
}
```

Package Diagrams

Number of classes has grown

- Can't have all in same class diagram
- Need a "higher level" view of application
- Use a **package diagram**



Top level package missing

Documenting the Design

Design documented in SDD

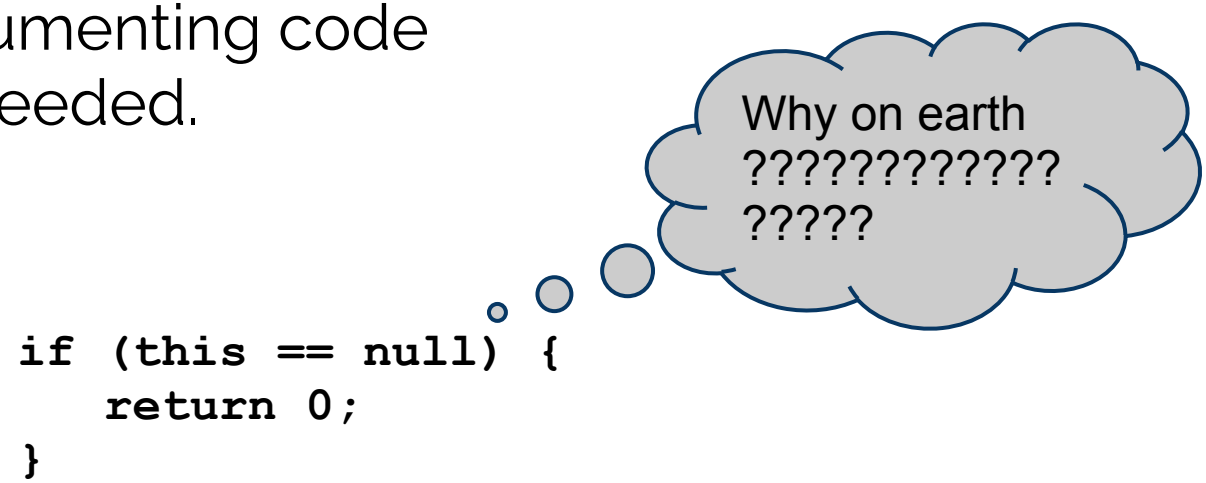
- Package diagram(s)
- One class diagram/"interesting package" (i.e. not GUI, class diagrams with arrows)
- At least 2 sequence diagrams (in appendix)
  - Sequence diagrams very time consuming, 2 will do
  - Note: Sequence diagrams also expanded when GUI added (also later when full MVC model) . Possible have to partition into more diagrams.
- Any kind of other high level information easing the understanding of the application; layering, MVC-style, service, use of design patterns, data formats, interfaces ...

SDD, updated after each iteration

Documenting the Implementation

No low level documentation in SDD

- Code it the ultimate low level documentation
- Tests also counts as documentation
- During implementation put comments in code ... even better write self documenting code
- NO Javadoc needed.



Summary

Using RAD as input

- We selected a 1-2 high priority use case
- Created some UML sequence diagram from the use cases (a dynamic model) using the objects (classes) from analysis model
- We got something very basic up and running
- From the above we got a basic design model (class diagrams, package diagram)
- Started to expand the running model, putting a simple GUI over model, started to do some serious testing

Next: Iteration 2