SWE 265P
Reverse Engineering and Modeling

Lecture 4: UML

Duplication of course material for any purpose without the explicit written permission of the professor is prohibited.

Reality

“…always find the particular ‘point of interest’ and then do chaining – chain backwards (who calls this – then who calls that – then who calls that) and if you iterate enough, you’ll get back to main() at some point. You can also forward-chain all the way down into the utility libraries and the ‘deepest’ parts of the call stack, at least for the feature you are investigating.” – Eric Dashofy [General Manager & Deputy CIO, The Aerospace Corporation]

Reality

If you can’t explain it, you don’t understand it well enough

[Albert Einstein]

Today

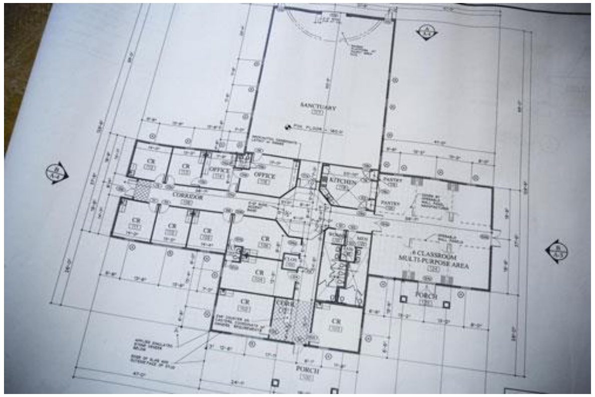
- Last week’s material
- UML
- Some useful tools
- Key expert practices

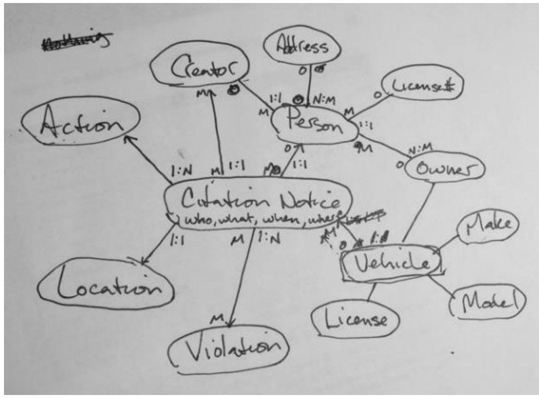
Last week's material

- Mental models
  - individual, uncertain, selective, flexible, dependent
  - external versus internal (software)
- Mental simulation
- Homework
- Any questions?

Models

- A model is a set of statements about some system under study
  - descriptive
  - prescriptive





Modeling in software development

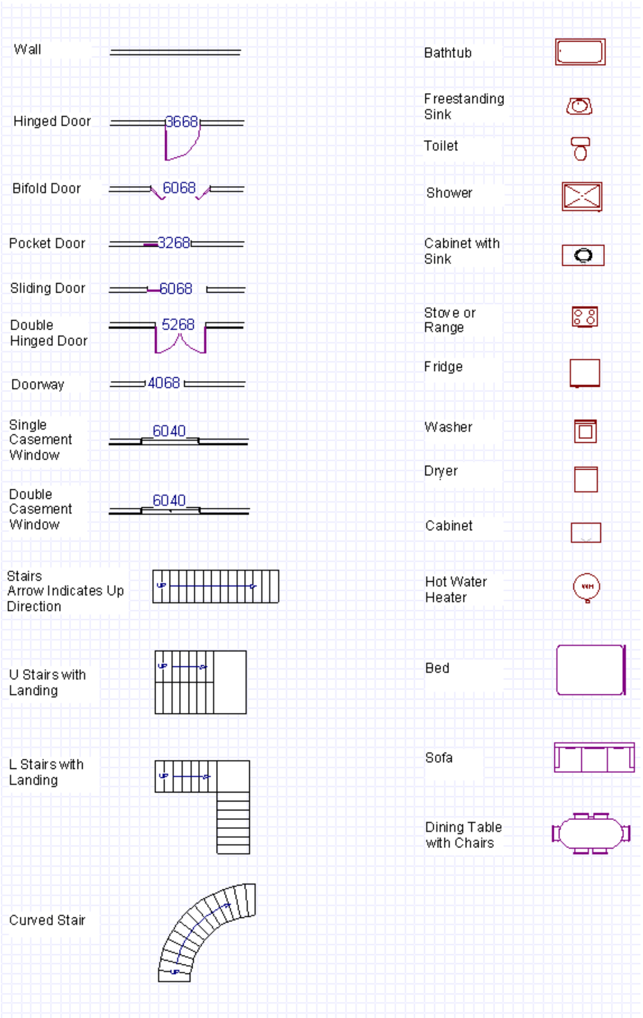
- Create a list of ‘types of things’ we may want to model as software developers

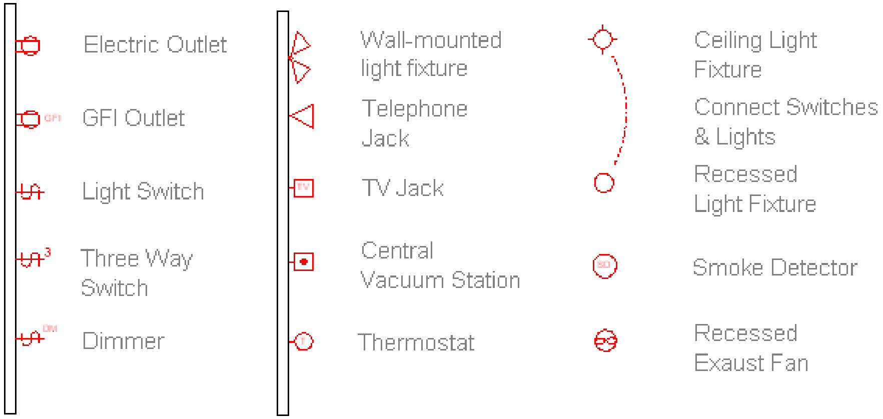
- https://jamboard.google.com/d/1etCI7DFRzRTxt9_NZK1rU3UK79aDyVH41J3juynhvvY/edit?usp=sharing

Modeling languages

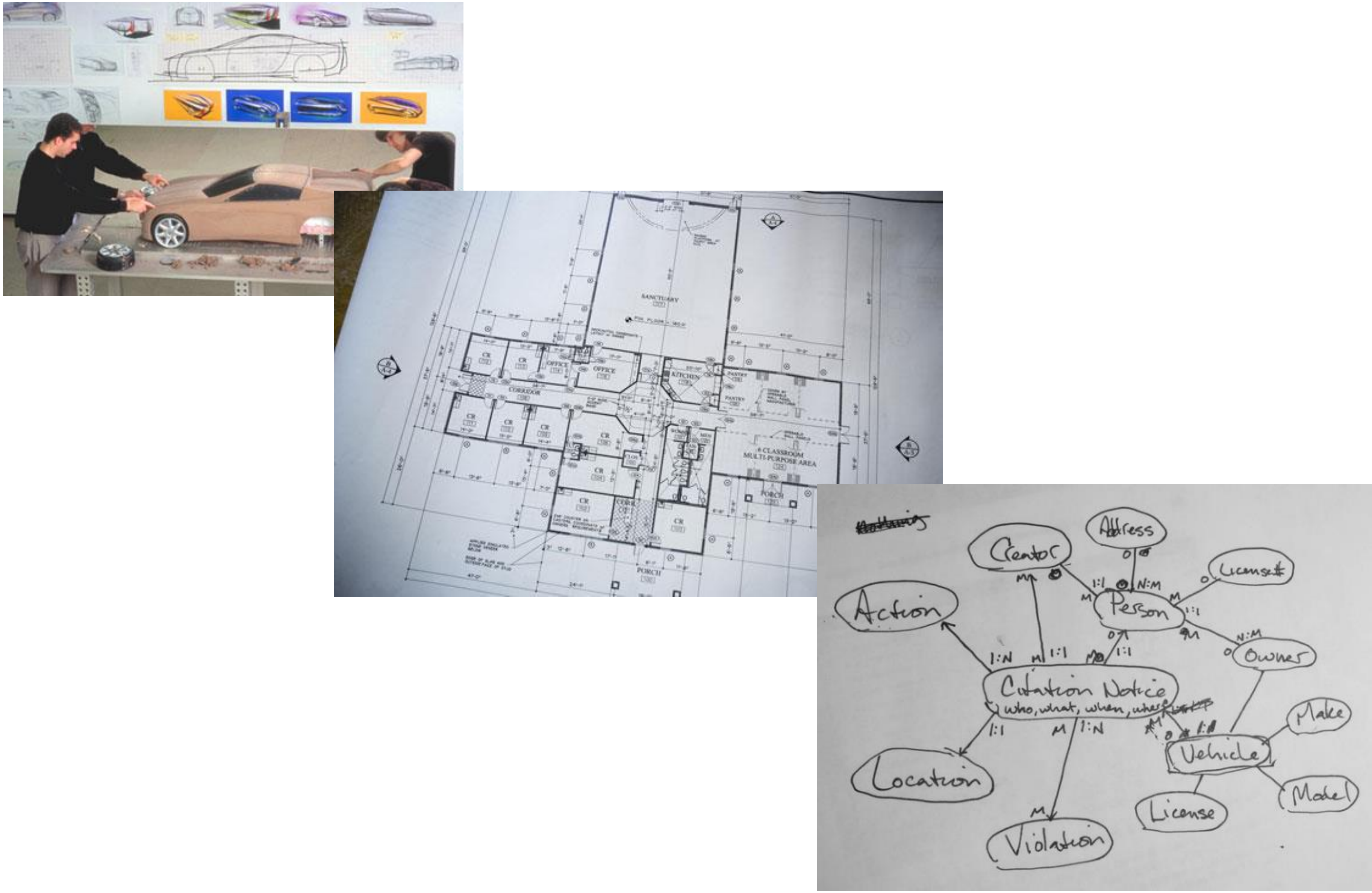
- A modeling language offers a vocabulary for specifying and interpreting models
  - textual and/or graphical
  - rules of composition
  - semantics
- Every modeling language invariably introduces abstraction
  - some information is readily available at the expense of obscuring or removing other information

Modeling languages

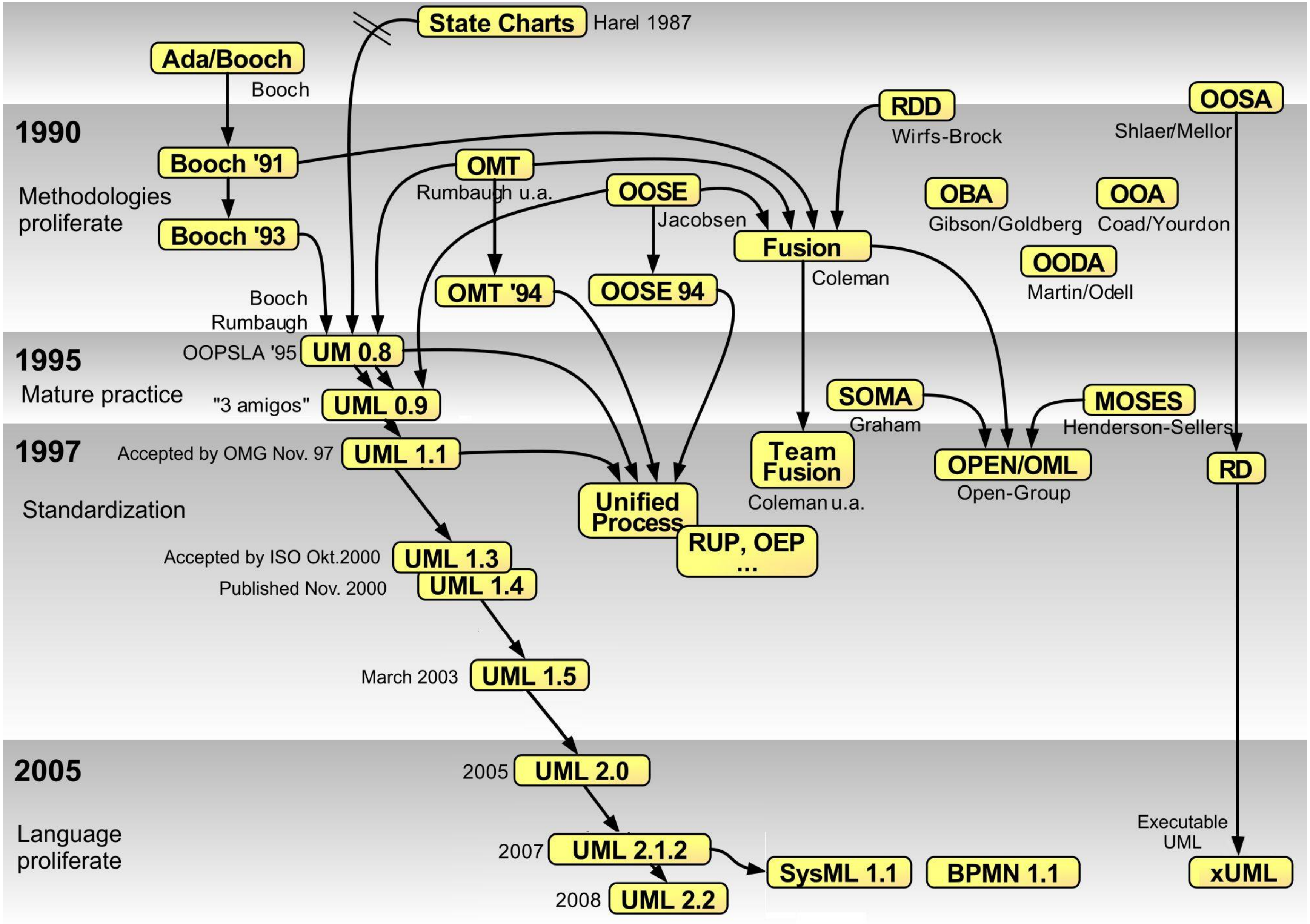




Modeling languages



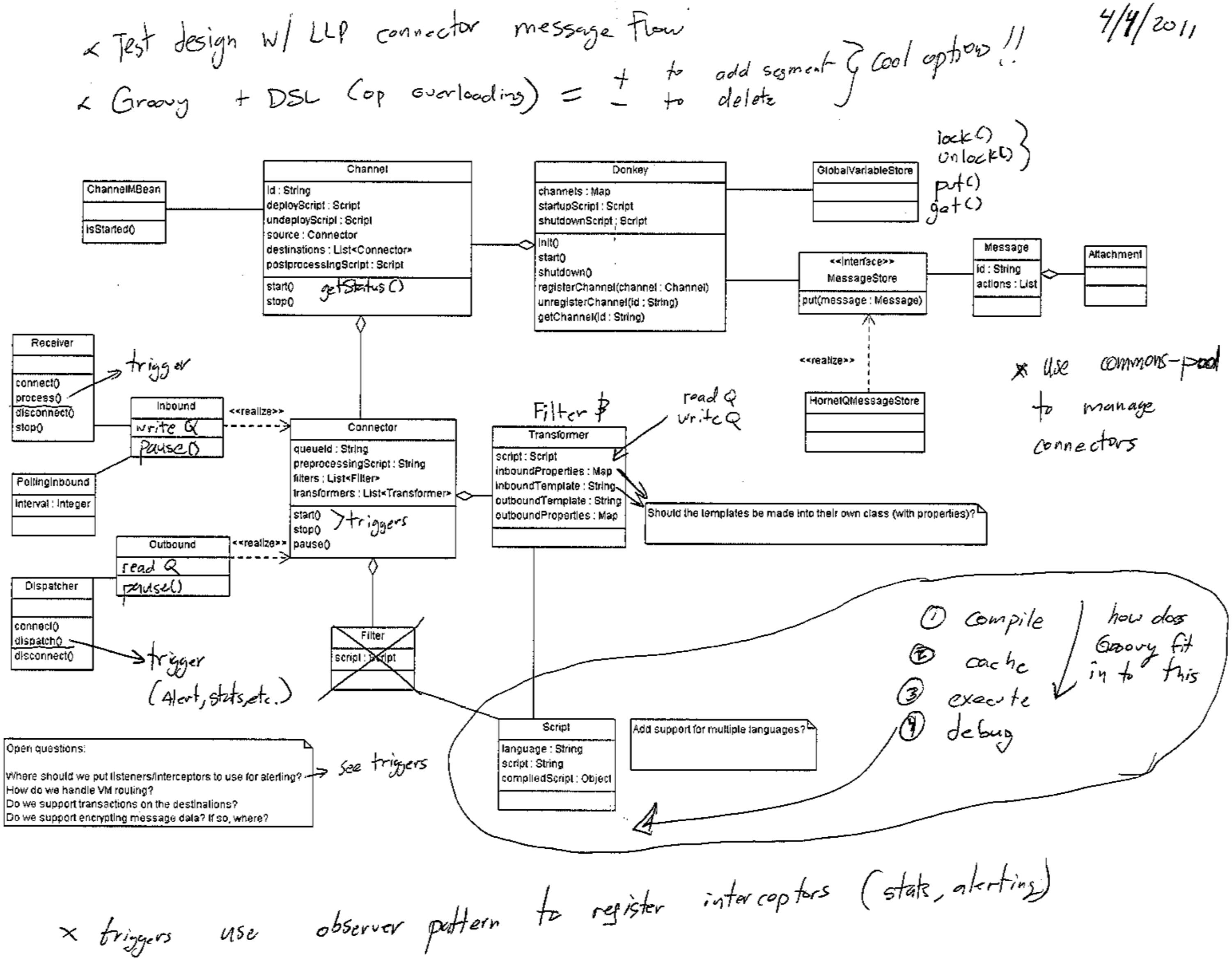
Unified modeling language



Structure versus behavior in UML

Structural models	Behavioral models
Class diagram	Use case diagram
Package diagram	Activity diagram
Component diagram	Statechart diagram
Deployment diagram	Sequence diagram
…	…

Example



4/4/2011

- Test design w/ LLP connector message flow
- Groovy + DSL (op overloading) = + to add segment, − to delete } cool option!!

- Use commons-pool to manage connectors

Should the templates be made into their own class (with properties)?

Add support for multiple languages?

Open questions:

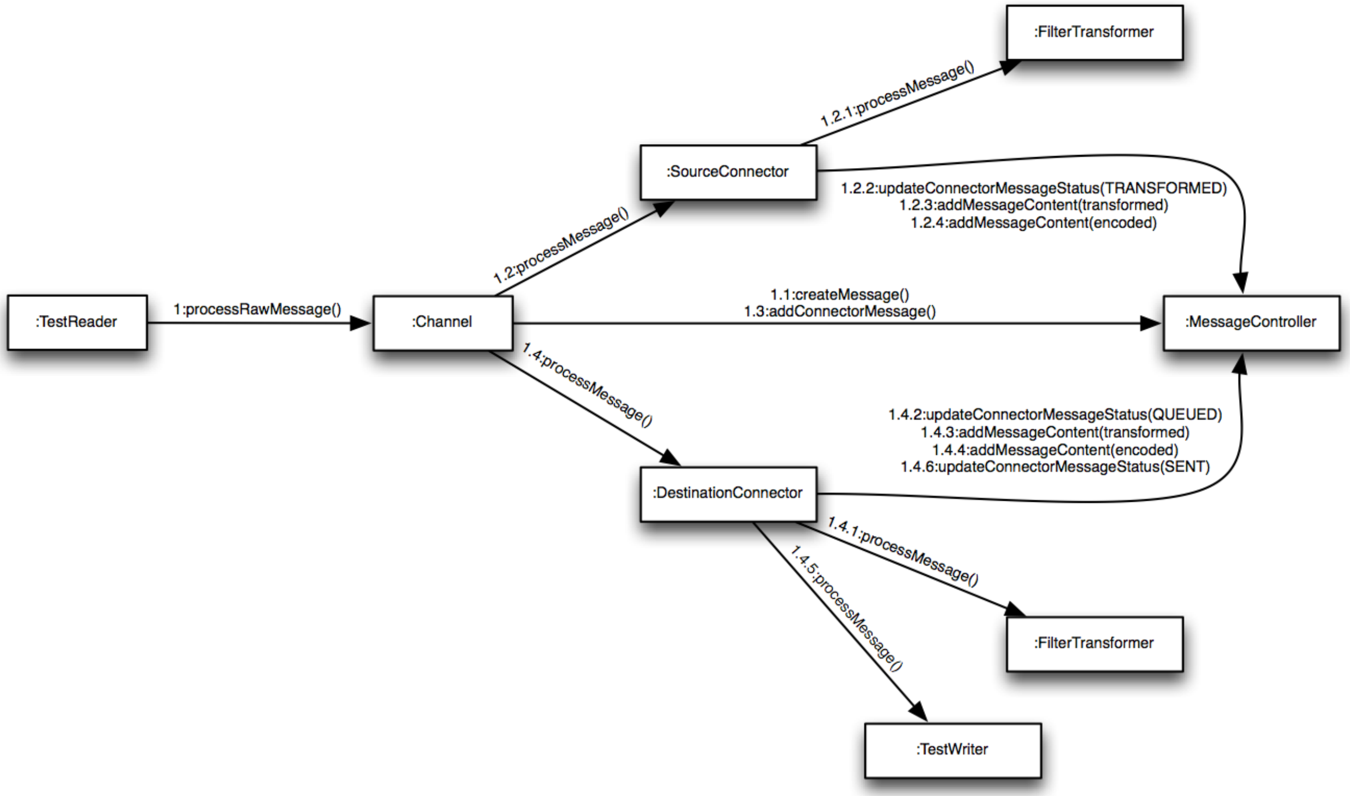
Where should we put listeners/interceptors to use for alerting?
How do we handle VM routing?
Do we support transactions on the destinations?
Do we support encrypting message data? If so, where?

1. compile
2. cache
3. execute
4. debug

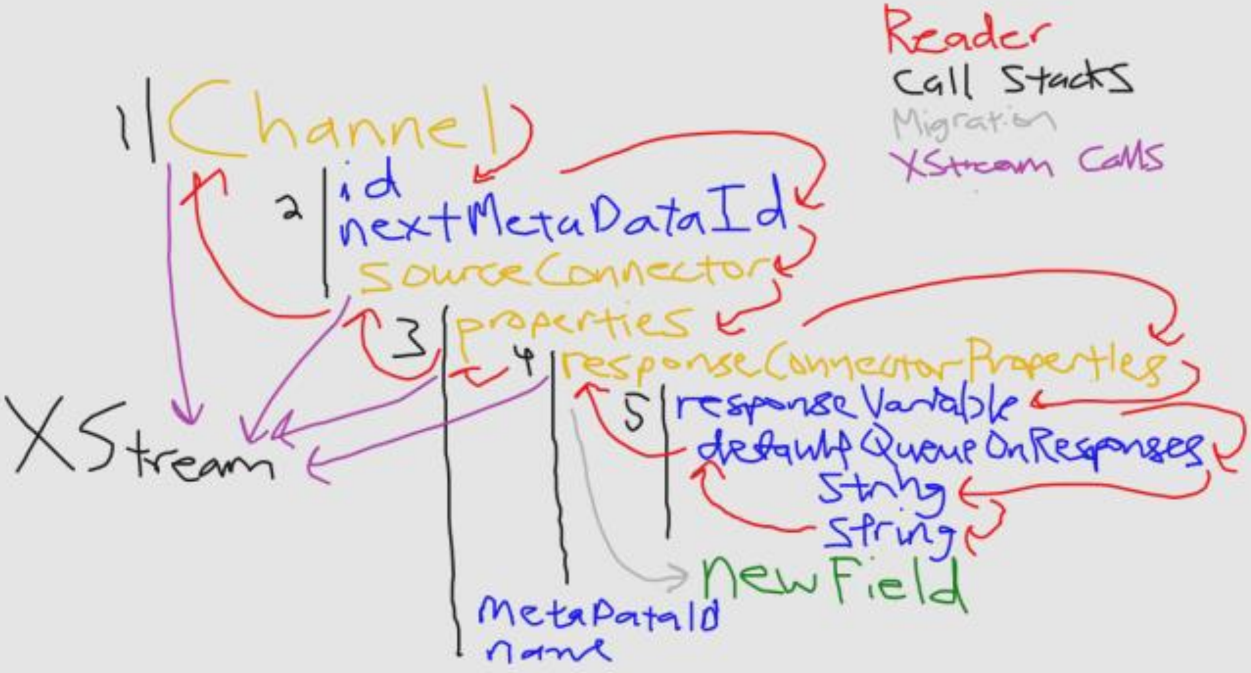
how does Groovy fit in to this

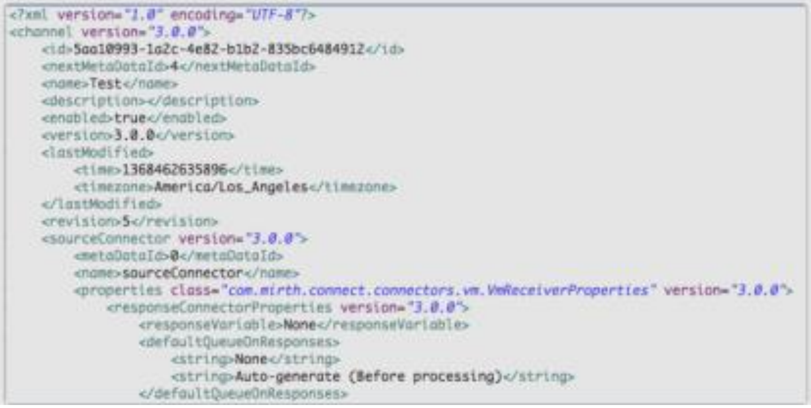
× triggers use observer pattern to register interceptors (stats, alerting)

Example


:FilterTransformer
1.2.1:processMessage()
:SourceConnector
1.2.2:updateConnectorMessageStatus(TRANSFORMED)
1.2.3:addMessageContent(transformed)
1.2.4:addMessageContent(encoded)
1.2:processMessage()
:TestReader
1:processRawMessage()
:Channel
1.1:createMessage()
1.3:addConnectorMessage()
:MessageController
1.4:processMessage()
1.4.2:updateConnectorMessageStatus(QUEUED)
1.4.3:addMessageContent(transformed)
1.4.4:addMessageContent(encoded)
1.4.6:updateConnectorMessageStatus(SENT)
:DestinationConnector
1.4.1:processMessage()
1.4.5:processMessage()
:FilterTransformer
:TestWriter

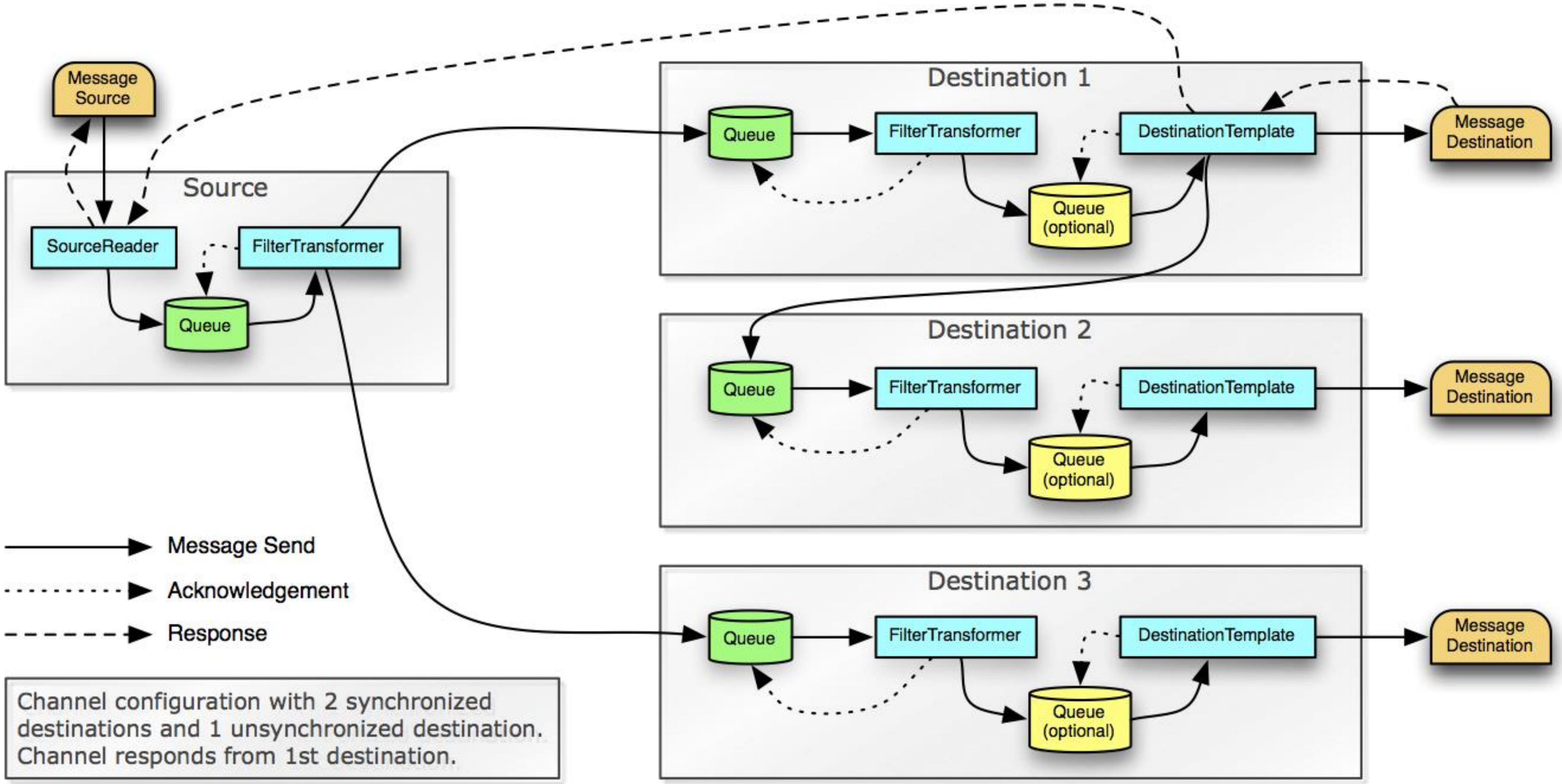
Example





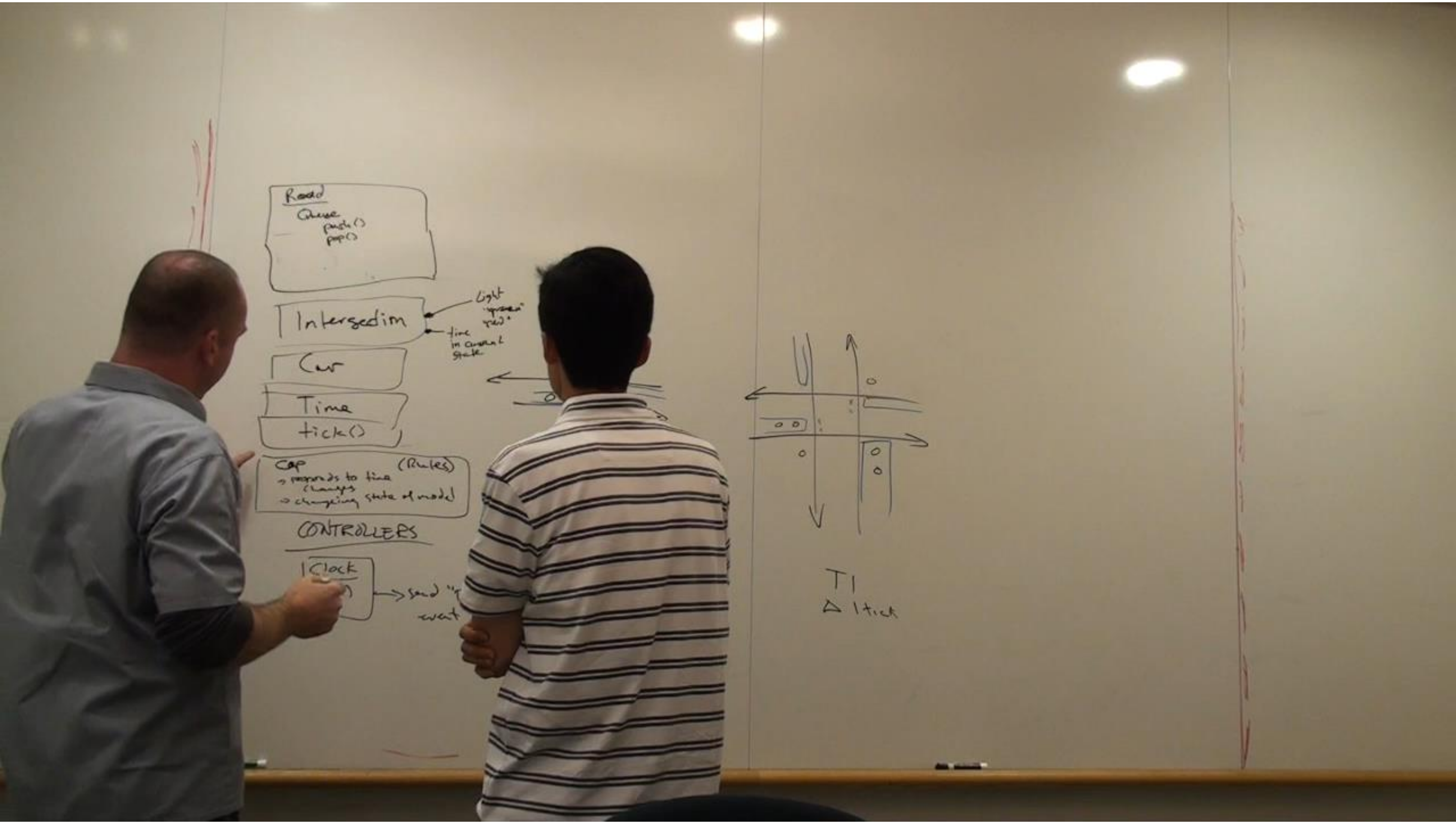
```
<?xml version="1.0" encoding="UTF-8"?>
<channel version="3.0.0">
    <id>5aa10993-1a2c-4e82-b1b2-835bc6484912</id>
    <nextMetaDataId>4</nextMetaDataId>
    <name>Test</name>
    <description></description>
    <enabled>true</enabled>
    <version>3.0.0</version>
    <lastModified>
        <time>1368462635896</time>
        <timezone>America/Los_Angeles</timezone>
    </lastModified>
    <revision>5</revision>
    <sourceConnector version="3.0.0">
        <metaDataId>0</metaDataId>
        <name>sourceConnector</name>
        <properties class="com.mirth.connect.connectors.vm.VmReceiverProperties" version="3.0.0">
            <responseConnectorProperties version="3.0.0">
                <responseVariable>None</responseVariable>
                <defaultQueueOnResponses>
                    <string>None</string>
                    <string>Auto-generate (Before processing)</string>
                </defaultQueueOnResponses>
```

Example

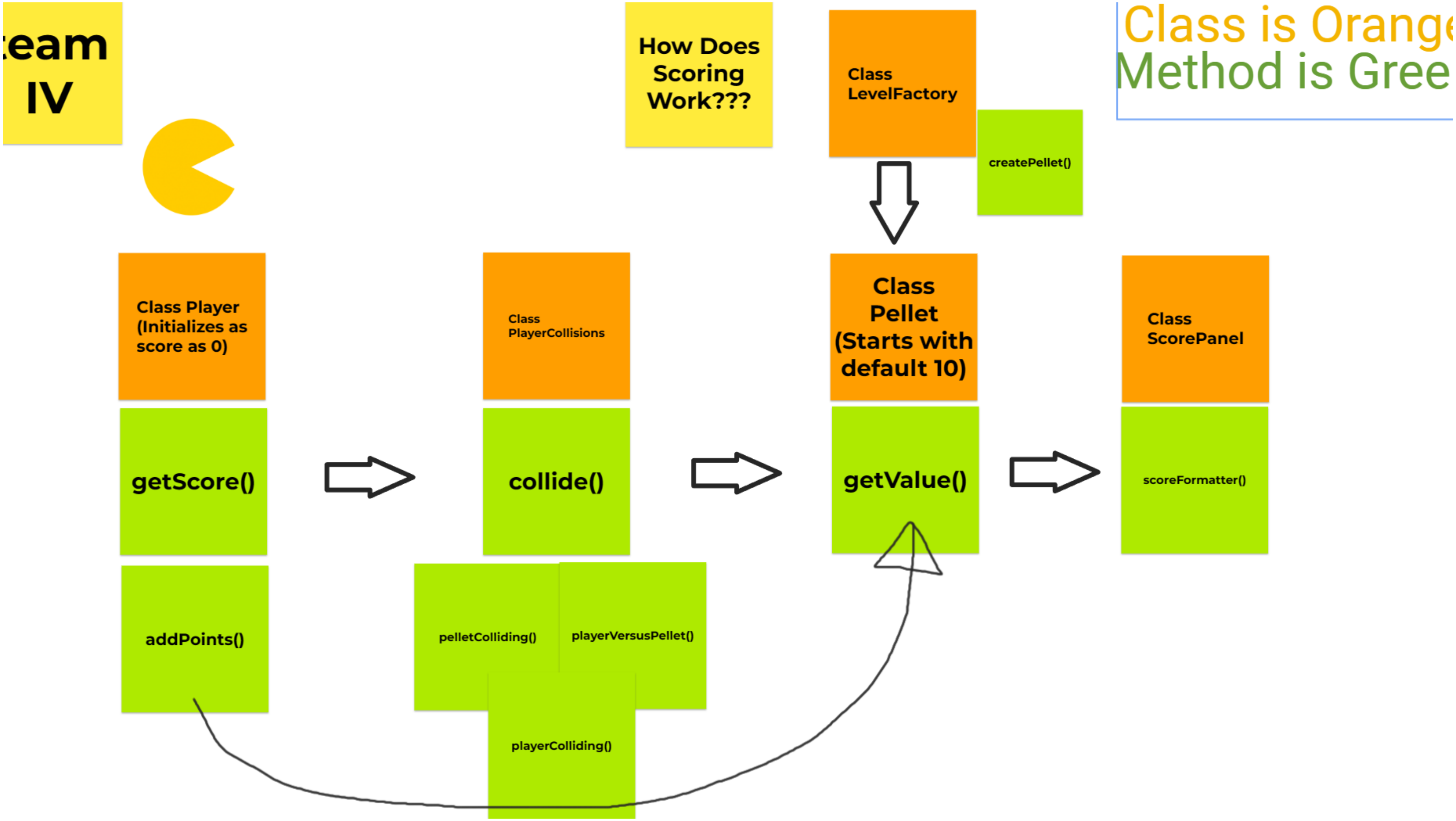


Channel configuration with 2 synchronized destinations and 1 unsynchronized destination. Channel responds from 1st destination.

Example


Road
Queue
push()
pop()
Intersedim
light
"green"
"red"
time
in current
state
Car
Time
tick()
Cap
(Rules)
→ responds to time
changes
→ changing state of model
CONTROLLERS
Clock
→ send "
event
T1
Δ 1tick

Example



eam IV

How Does Scoring Work???

Class LevelFactory

createPellet()

Class is Orang

Method is Gree

Class Player (Initializes as score as 0)

getScore()

addPoints()

Class PlayerCollisions

collide()

pelletColliding()

playerVersusPellet()

playerColliding()

Class Pellet (Starts with default 10)

getValue()

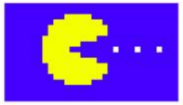
Class ScorePanel

scoreFormatter()

Example

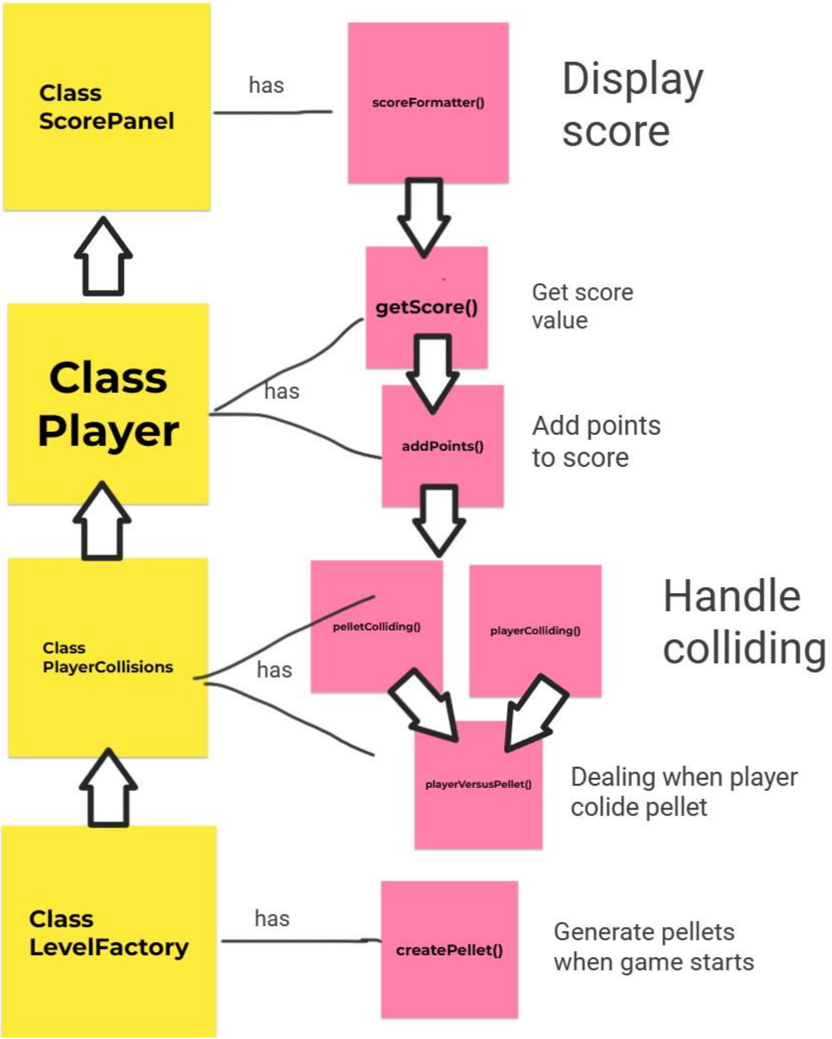
Display player score in the score panel UI

Define functions to add and get score



Player collide with pellets -> add points

Create Pellets



Class ScorePanel

has

scoreFormatter()

Display score

Class Player

has

getScore()

Get score value

addPoints()

Add points to score

Class PlayerCollisions

has

pelletColliding()

playerColliding()

Handle colliding

playerVersusPellet()

Dealing when player colide pellet

Class LevelFactory

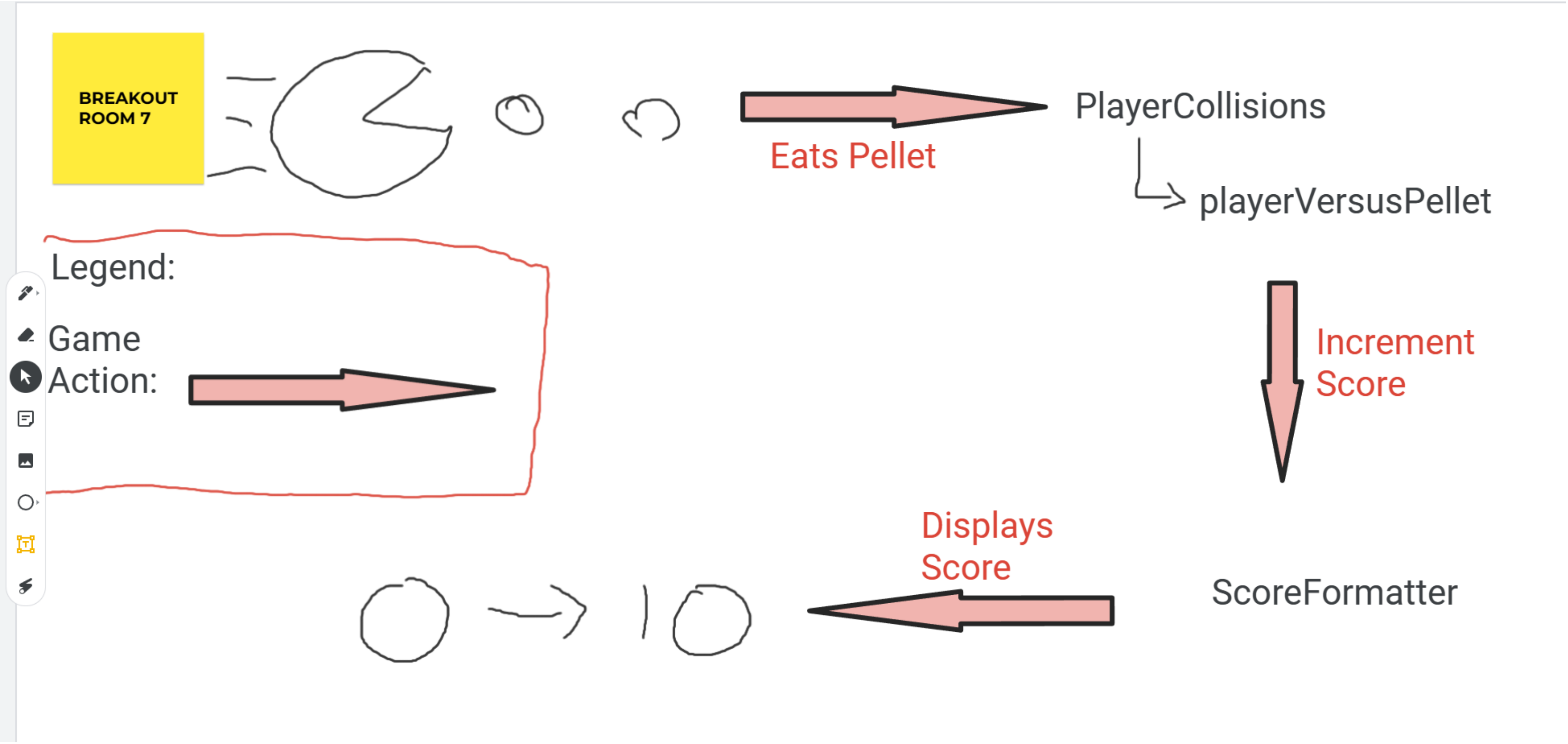
has

createPellet()

Generate pellets when game starts

BREAKOUT ROOM 5

Example



PlayerCollisions

playerVersusPellet

Eats Pellet

Increment Score

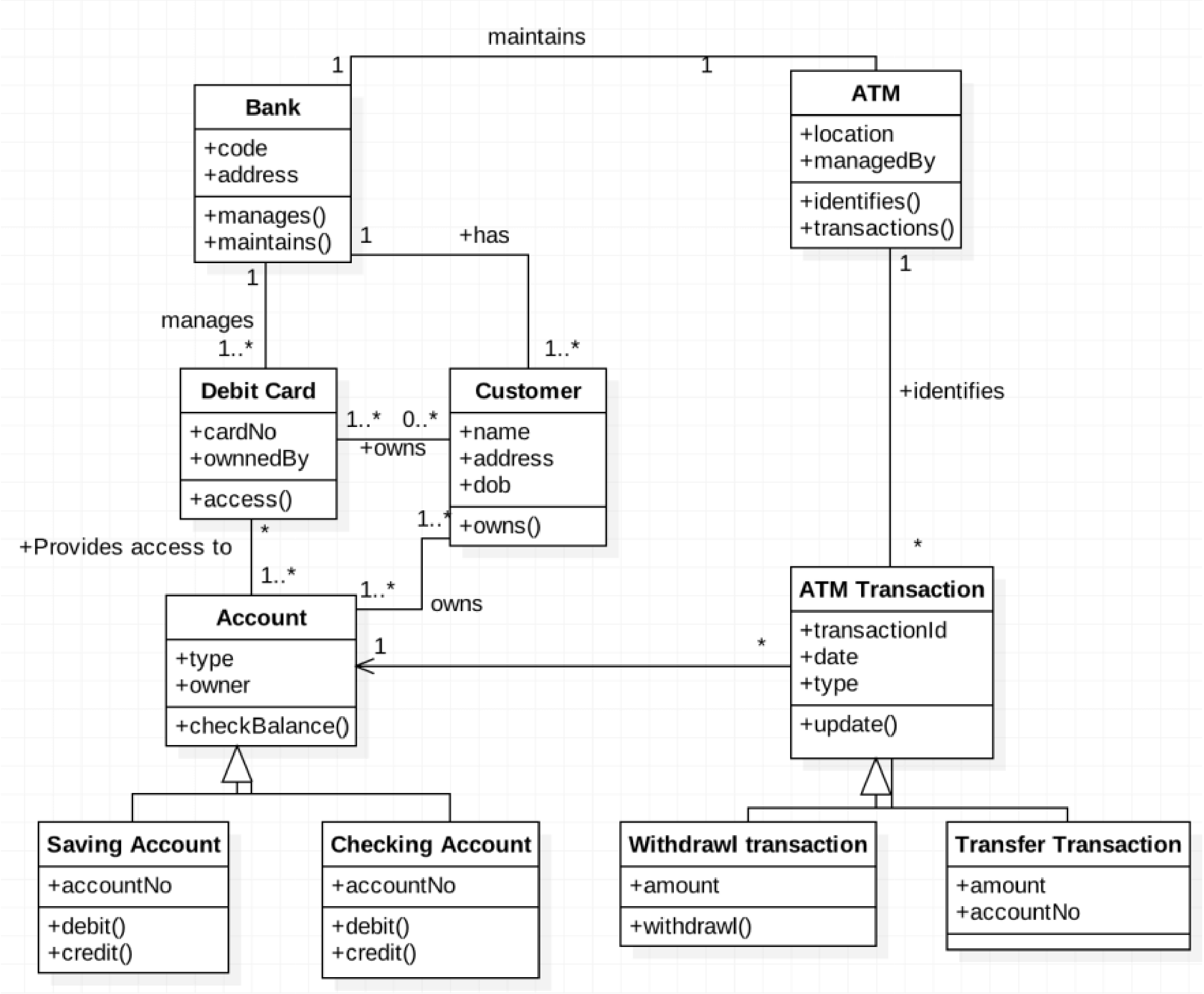
ScoreFormatter

Displays Score

Legend:

Game Action:

Reading UML: class diagram



maintains

1

1

Bank

+code
+address

+manages()
+maintains()

ATM

+location
+managedBy

+identifies()
+transactions()

1

+has

1

1

manages
1..*

1..*

+identifies

Debit Card

+cardNo
+ownnedBy

+access()

1..* 0..*
+owns

Customer

+name
+address
+dob

+owns()

1..*

*

+Provides access to

1..*

1..*

owns

*

Account

+type
+owner

+checkBalance()

1

*

ATM Transaction

+transactionId
+date
+type

+update()

Saving Account

+accountNo

+debit()
+credit()

Checking Account

+accountNo

+debit()
+credit()

Withdrawl transaction

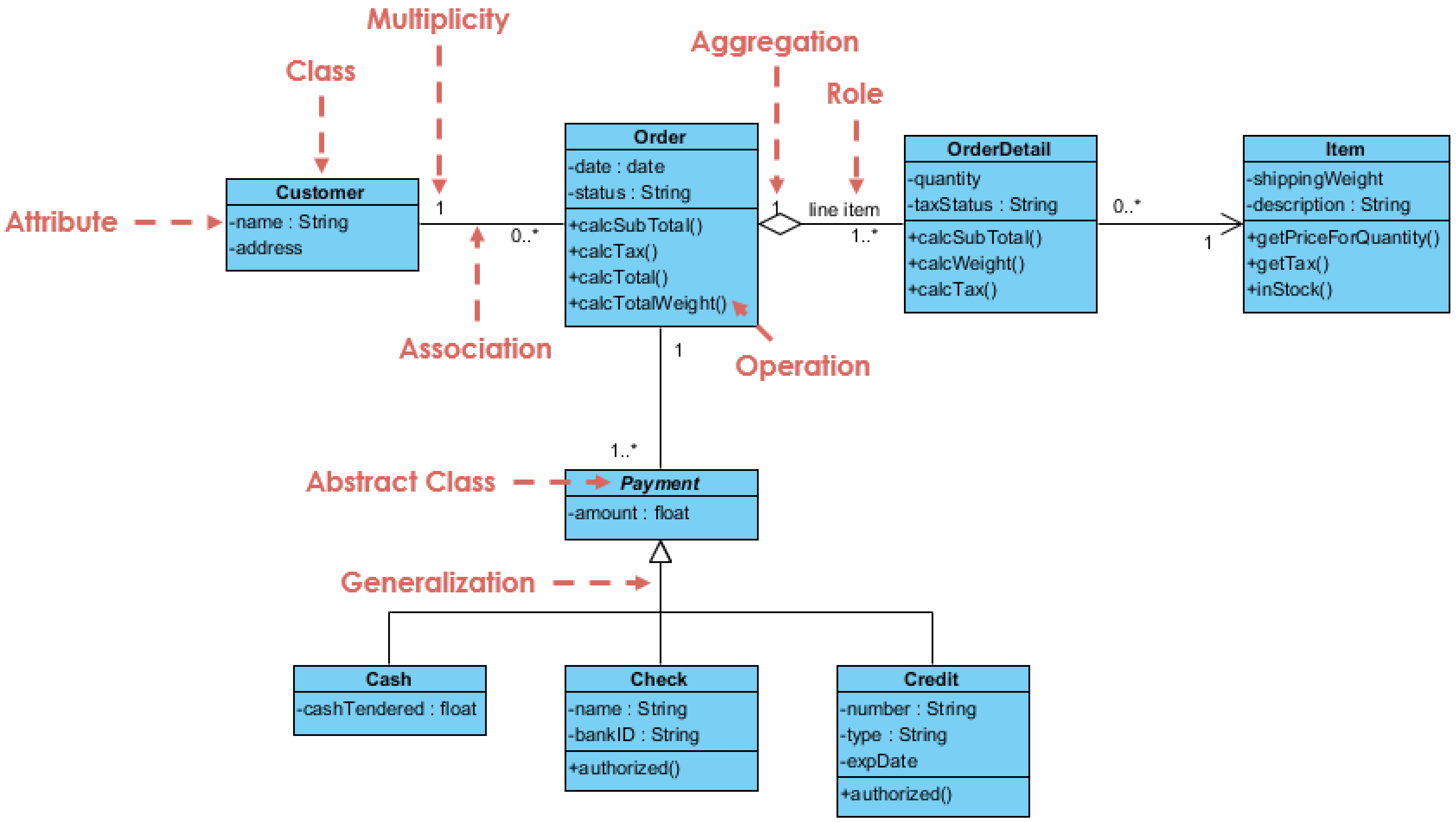
+amount

+withdrawl()

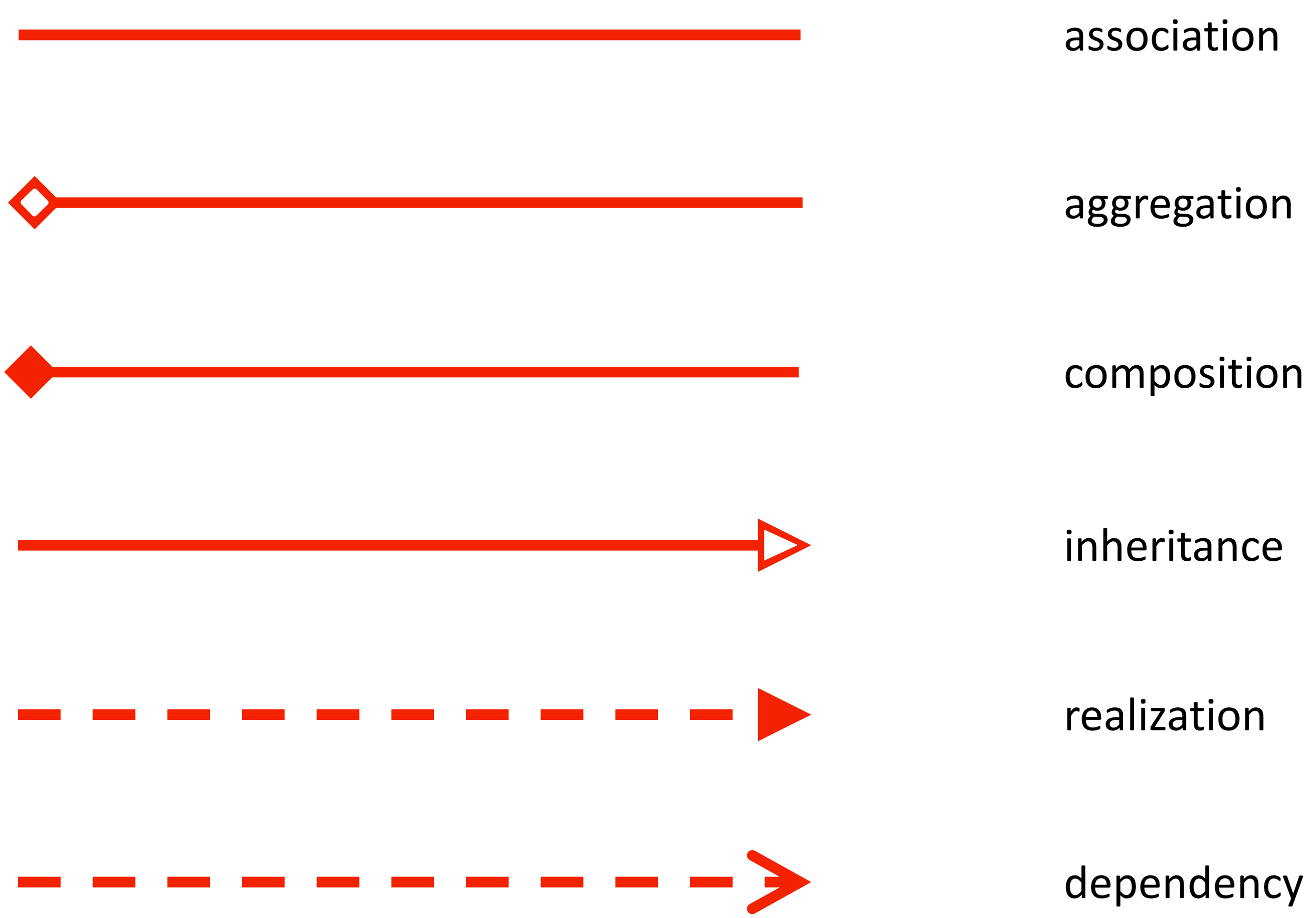
Transfer Transaction

+amount
+accountNo

Reading UML: class diagram



UML class diagram meaning of the arrows



association

aggregation

composition

inheritance

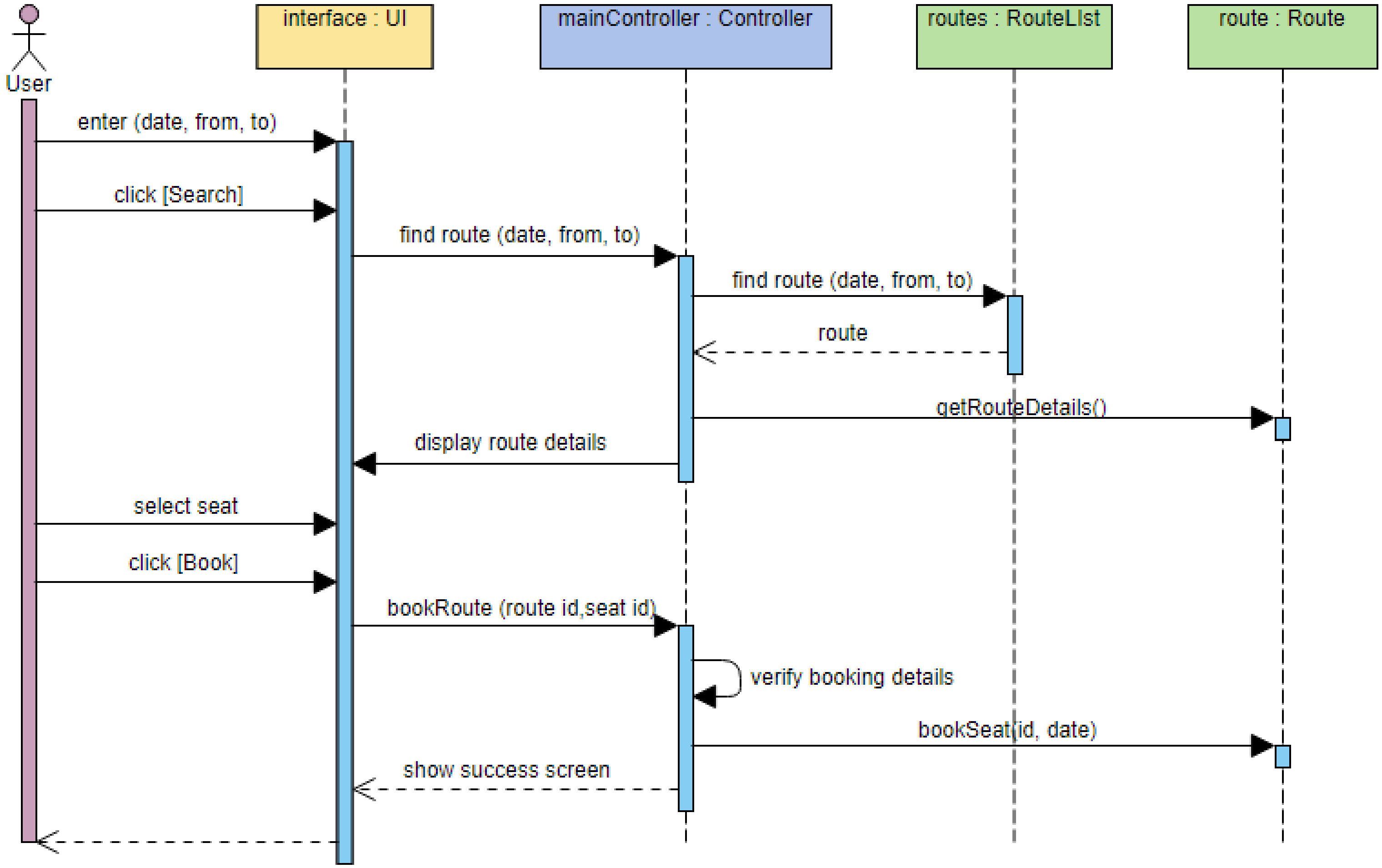
realization

dependency

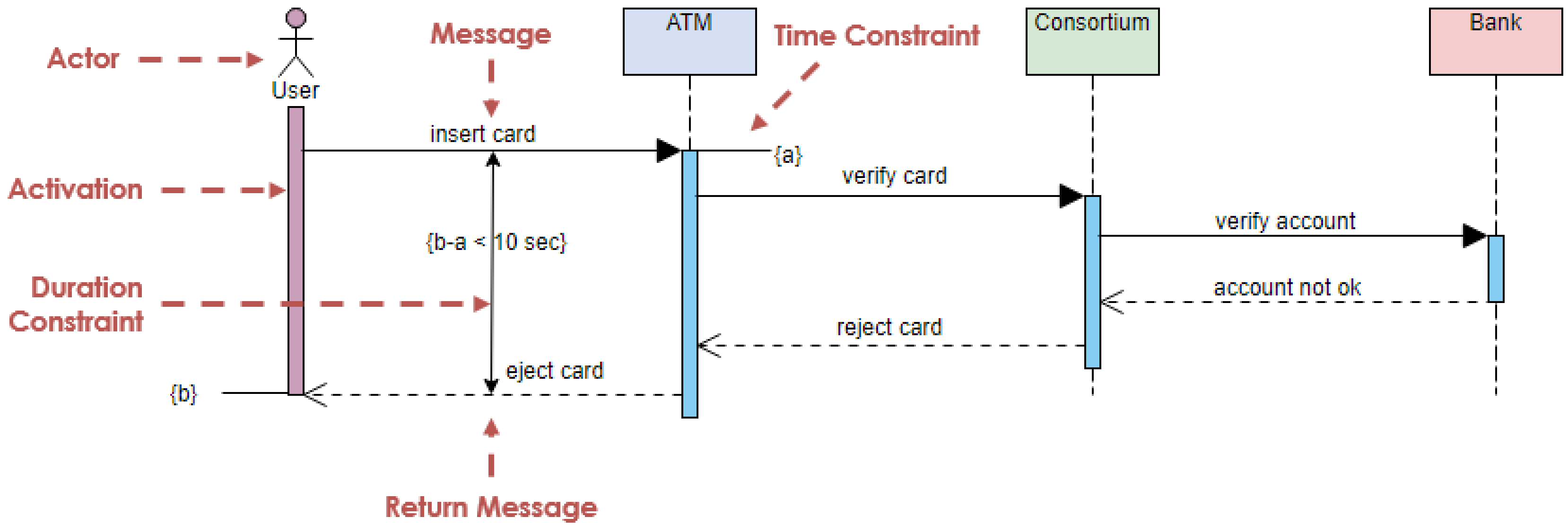
Association, aggregation, composition

- Association
  - most general kind of relationship
  - instructor <teaches a> class, kid <plays with a> friend
- Aggregation
  - more specific kind of relationship
  - has-a relationship, is-a-part-of relationship
  - child can exist independent of the parent
  - bird <is-part-of-a> flock, airplane type <has-a> engine model
- Composition
  - more specific yet
  - consists-of relationship, contains relationship
  - child cannot exist independent of the parent
  - house <consists-of a> room, university <contains a> department

Reading UML: sequence diagram



Reading UML: sequence diagram



Let’s practice: JPacMan3

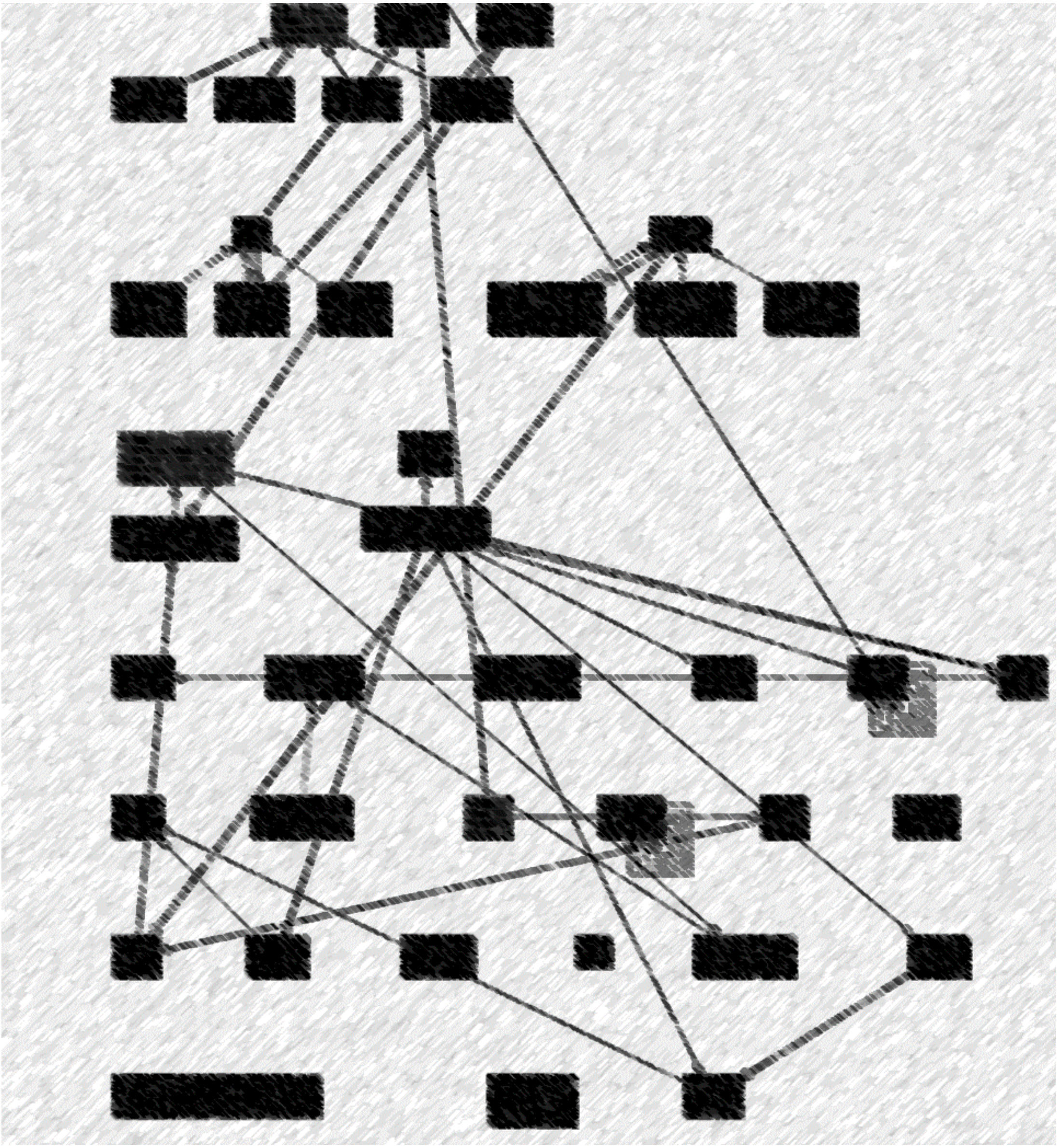
- You should still have a clone of JPacMan3, but if not
  - https://github.com/SWE-265P/jpacman3
- Open the project

JPacMan question 2

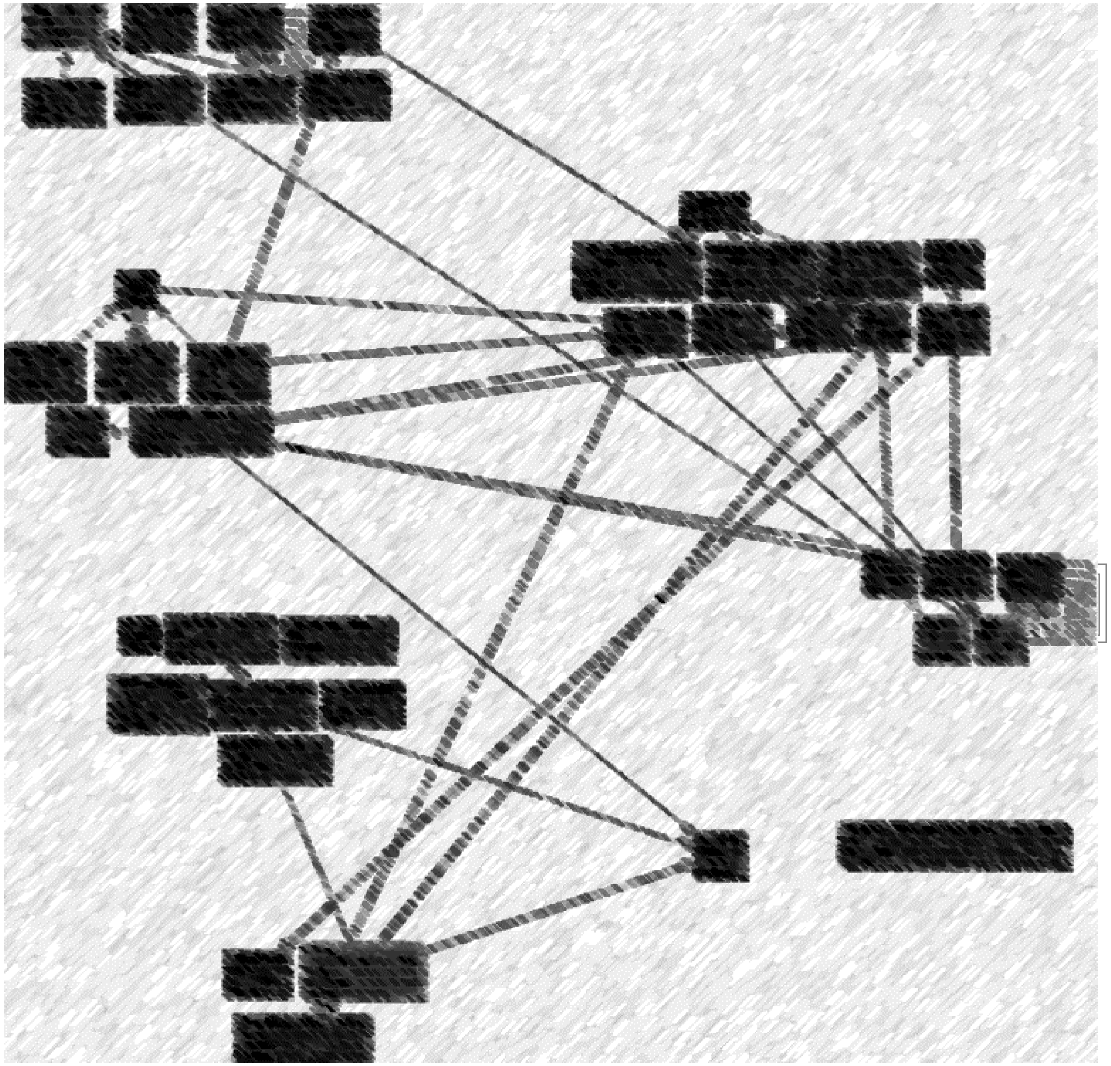
- Draw a class diagram model capturing the classes and relationships inside the level package

- https://jamboard.google.com/d/11Au__xyLS-JBlZ7BuAqYLjPIuMcdrJZjn1XHrhHOV5A/edit?usp=sharing

JPacMan



Reorganizing along folder structure

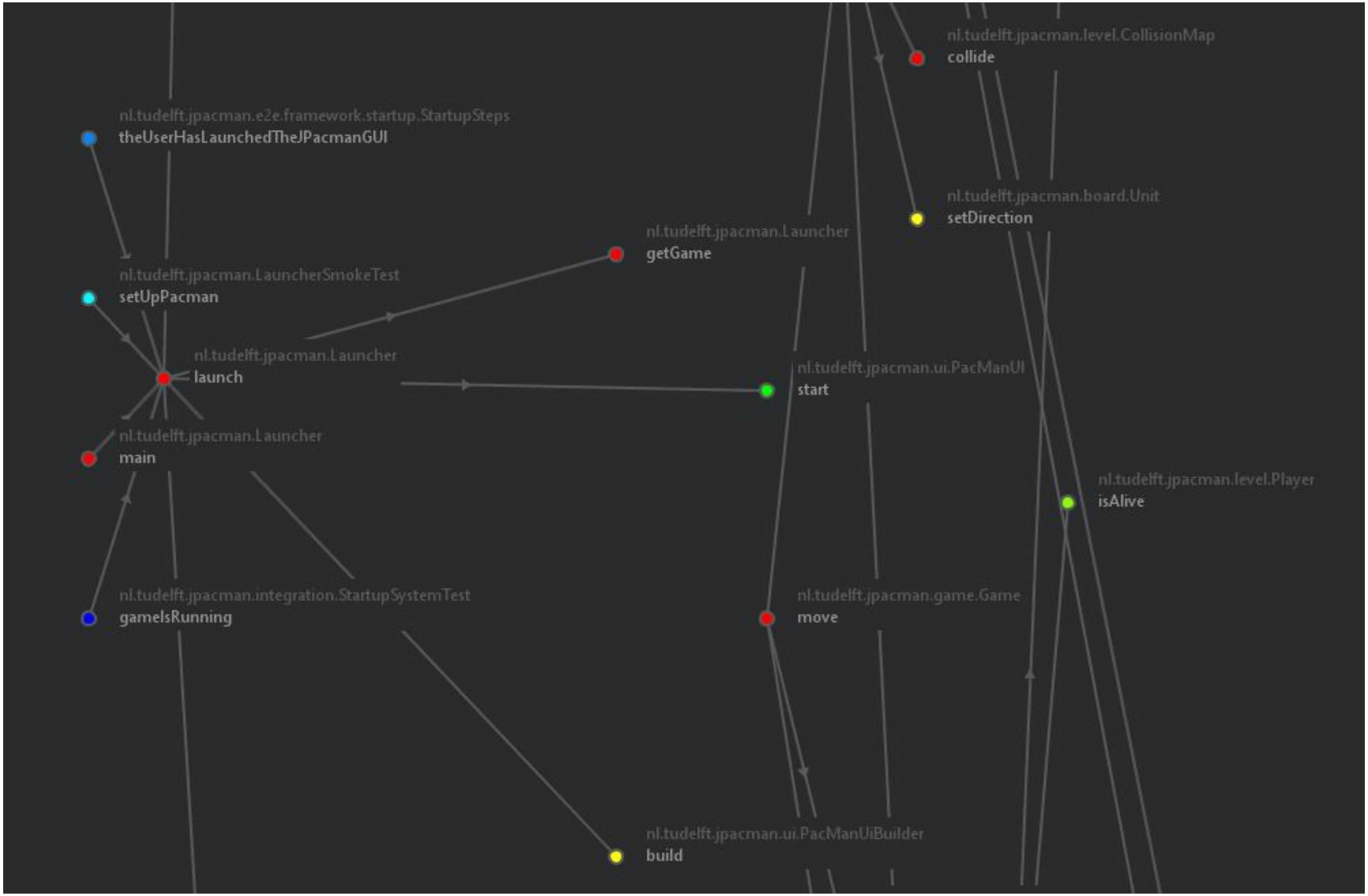


JPacMan question 3

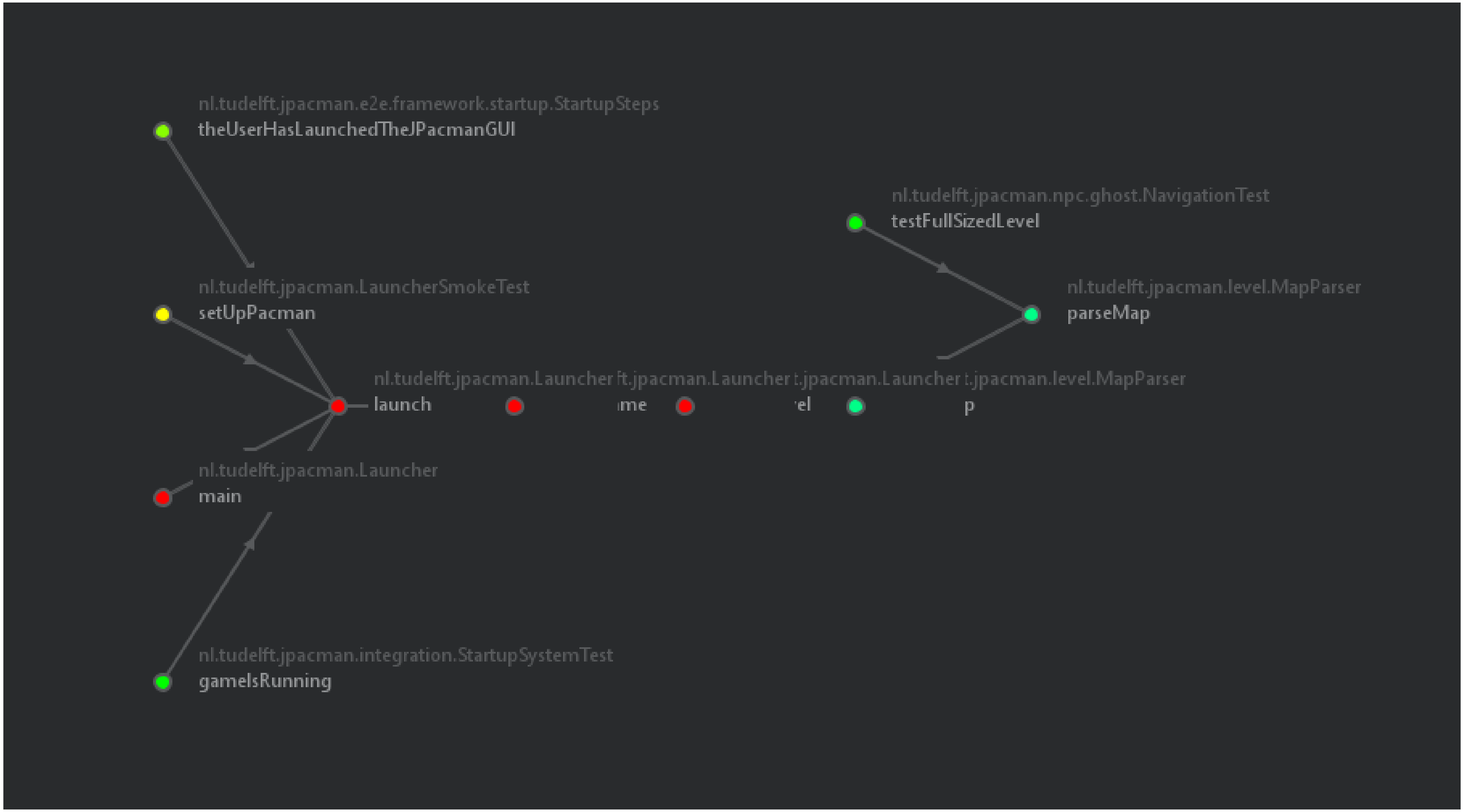
- Draw a sequence diagram model capturing how a level is created

- https://jamboard.google.com/d/1HCw3Rava1BXJwM6rEyrbW6iIlKuFe4KbgNHzIBRl86Q/edit?usp=sharing

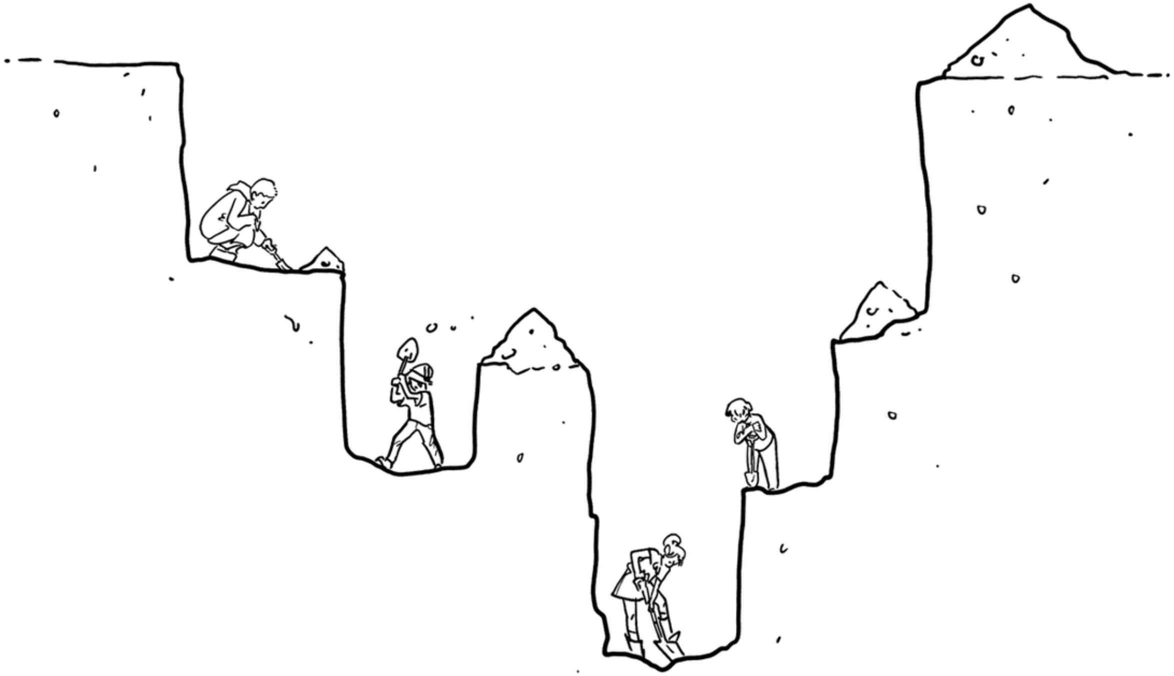
Call graphs


nl.tudelft.jpacman.level.CollisionMap
collide
nl.tudelft.jpacman.e2e.framework.startup.StartupSteps
theUserHasLaunchedTheJPacmanGUI
nl.tudelft.jpacman.board.Unit
setDirection
nl.tudelft.jpacman.Launcher
getGame
nl.tudelft.jpacman.LauncherSmokeTest
setUpPacman
nl.tudelft.jpacman.Launcher
launch
nl.tudelft.jpacman.ui.PacManUI
start
nl.tudelft.jpacman.Launcher
main
nl.tudelft.jpacman.level.Player
isAlive
nl.tudelft.jpacman.integration.StartupSystemTest
gameIsRunning
nl.tudelft.jpacman.game.Game
move
nl.tudelft.jpacman.ui.PacManUiBuilder
build

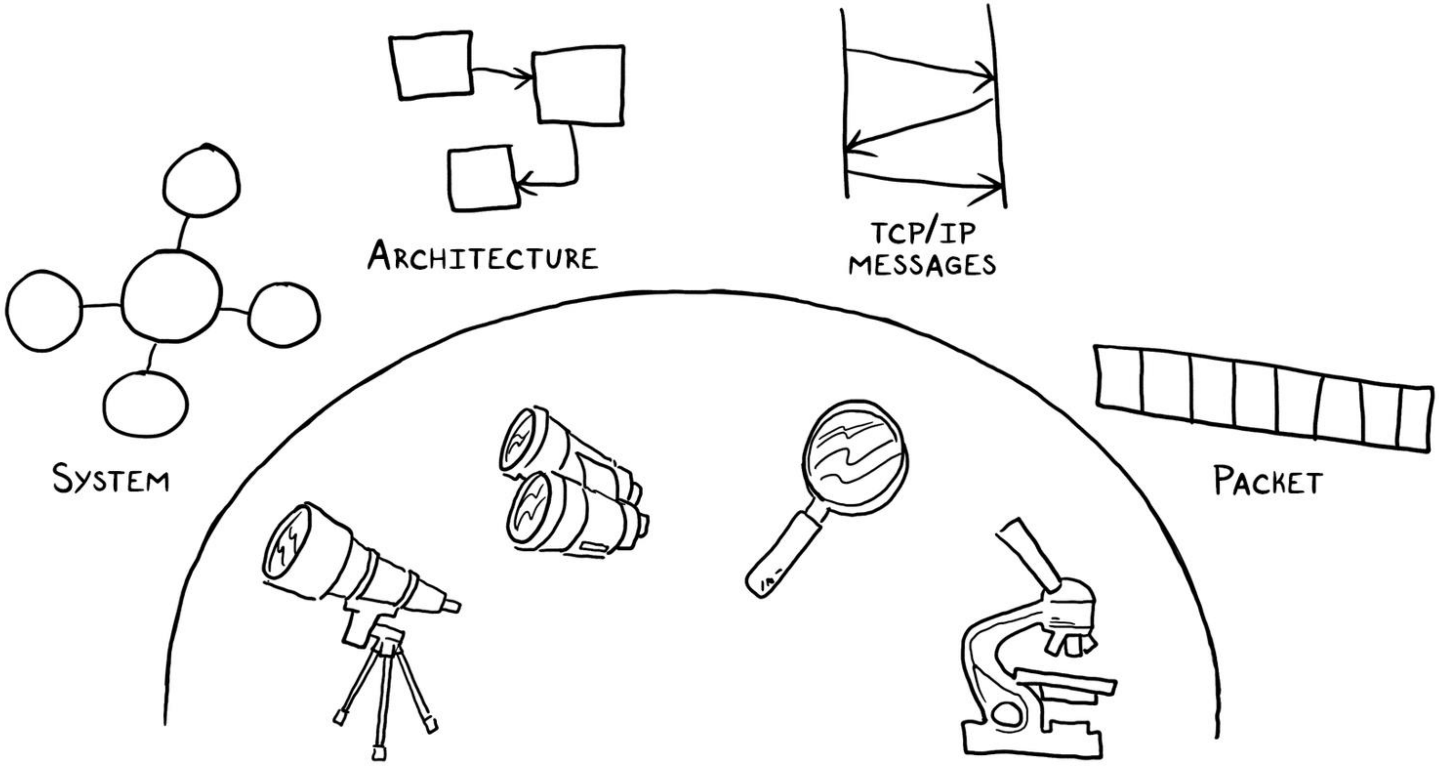
Going backwards (call graphs)


nl.tudelft.jpacman.e2e.framework.startup.StartupSteps
theUserHasLaunchedTheJPacmanGUI
nl.tudelft.jpacman.npc.ghost.NavigationTest
testFullSizedLevel
nl.tudelft.jpacman.LauncherSmokeTest
setUpPacman
nl.tudelft.jpacman.level.MapParser
parseMap
nl.tudelft.jpacman.Launcher
launch
ft.jpacman.Launcher
ıme
t.jpacman.Launcher
'el
t.jpacman.level.MapParser
p
nl.tudelft.jpacman.Launcher
main
nl.tudelft.jpacman.integration.StartupSystemTest
gameIsRunning

KEP #4: go as deep as needed

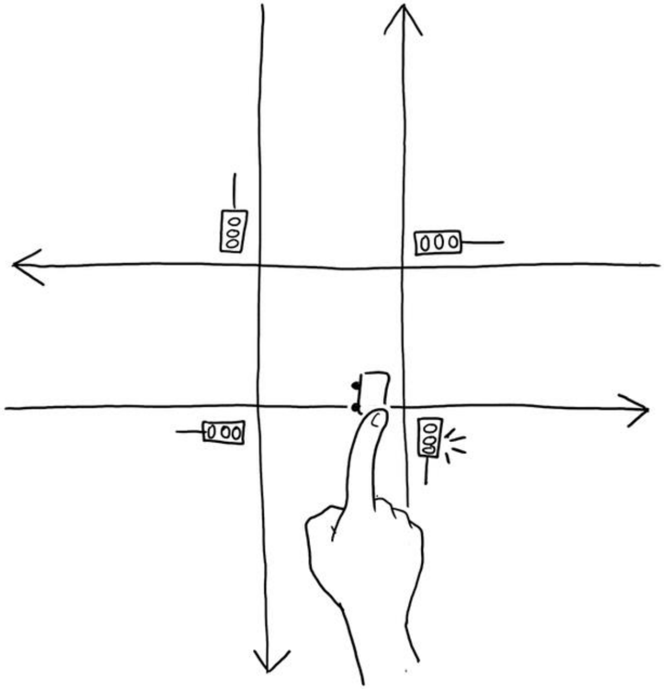


KEP #5: move along levels of abstraction



KEP #6: draw examples alongside their diagrams

```
IF (CAR APPROACHES INTERSECTION)
  IF (INTERSECTION.LIGHT IS RED)
            CAR.SPEED = 0
  ELSE (INTERSECTION.LIGHT IS YELLOW)
            SPEED += 10
```



Project work, part 1

- With your team, generate a UML class diagram for your entire system
- Submit as a single PDF, PNG, or JPEG
- Due: Tuesday @ 4pm

Project work, part 2

- With your team, decide upon two features that are essential in your system, and imagine that each of the two features will need to undergo some kind of change to be implemented by someone else
- For each of the features, highlight in the UML class diagram where its essence is implemented
- Submit as a single PDF, PNG, or JPEG
- Due: Tuesday @ 4pm

Project work, part 3

- With your team, imagine that each of the two features will need to undergo some kind of change to be implemented by someone else
- Prepare a packet, per feature, that would assist that other person in understanding the feature
- Submit both packets in a single PDF
- Due: Tuesday @ 4pm

Homework (individual)

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
- https://online.visual-paradigm.com/diagrams/tutorials/sequence-diagram-tutorial/
- https://www.youtube.com/watch?v=UI6lqHOVHic
- https://www.youtube.com/watch?v=pCK6prSq8aw
- https://creately.com/blog/diagrams/uml-diagram-types-examples/
- http://www.agilemodeling.com/artifacts/classDiagram.htm

Homework (individual)

- Make sure to regularly update your personal diary

Homework (individual)

- Perform your first team evaluation
  - https://forms.gle/nzGc6p56VR83chDy5
- Due: Thursday May 2 @ 4pm

Optional advanced material

- Download the codecrumbs tool and experiment with how you may be able to use it in externalizing your mental model
  - https://codecrumbs.io/

- Experiment with different UML tools
  - Star UML
  - UML designer
  - Visual Paradigm
  - …