

COPENHAGEN BUSINESS ACADEMY

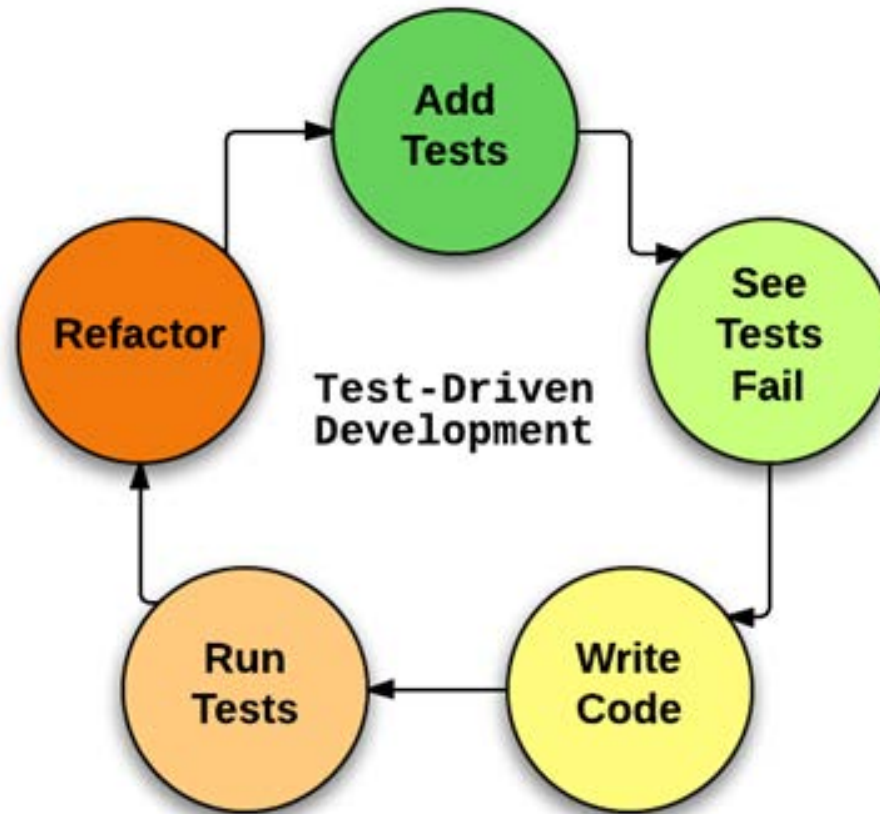


Test-Driven Development Principles & how to drive TDD

Workshop Goals

- **TDD principles** and maintaining the test-driven cycle
- Practice getting into the TDD mindset **by example**
- **Use mock objects** to support TDD cycle
 - Create any required mock objects
 - Create any real objects, including the target object
 - Specify how you *expect* the mock objects to be called by target object
 - *Assert* that any resulting values are valid and that all the expected calls have been made

The Test-Driven Cycle





Maintaining The Test-Driven Cycle

- Start each feature with an ***acceptance test***
 - Clarifies WHAT to do with no underlying tech focus
 - Shields acceptance test suite from changes in technical infrastructure
- Start with ***simplest success test case***
 - Gives you better idea of real structure of solution
 - Keep notepad next to you to jot down failure cases, refactoring and other technical tasks that need to be addressed
- Mockist testing develops from the [inputs to the outputs](#)
 - To reduce integration problems later on
 - In all circumstances, being agile you work on one story at a time
- Unit test ***behavior***, not methods
 - Makes object responsibility more understandable
 - A test called `testBidAccepted()` tells what it does, but not what it's *for*.

Monopoly - an Example (1)

- 2 use cases (~ stories)
 - Initialization
 - Playing game
- *Game loop* Algorithm
 - Terminology:
 - *turn* – a player rolling the dice and moving the piece
 - *round* - all the players taking one turn

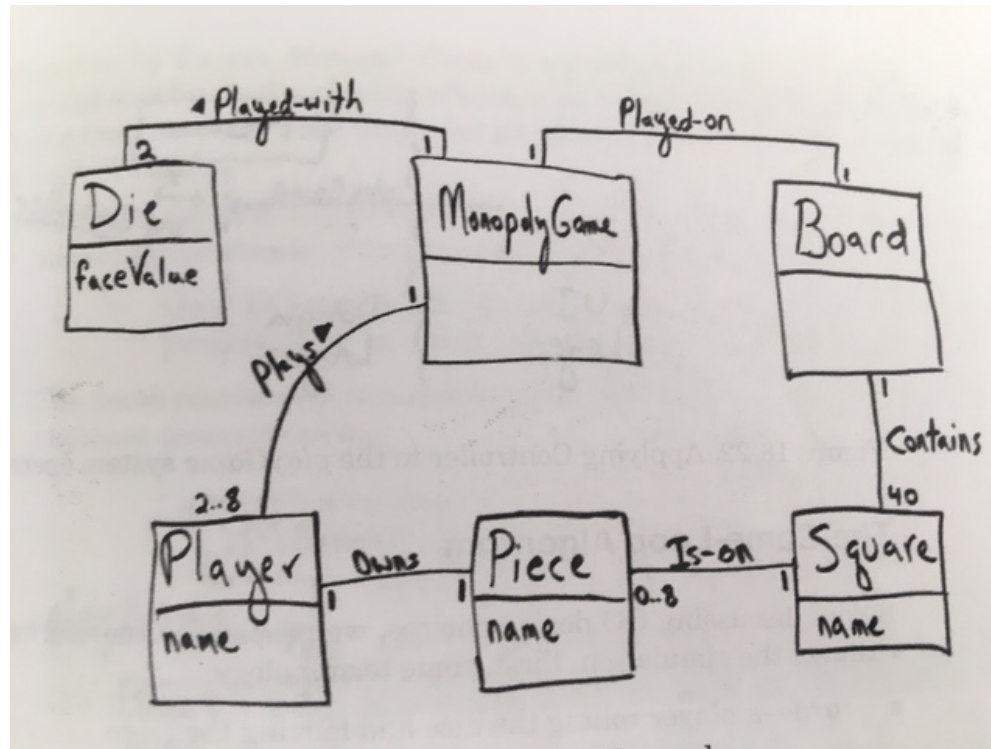
For N rounds

For each Player p

p takes a turn

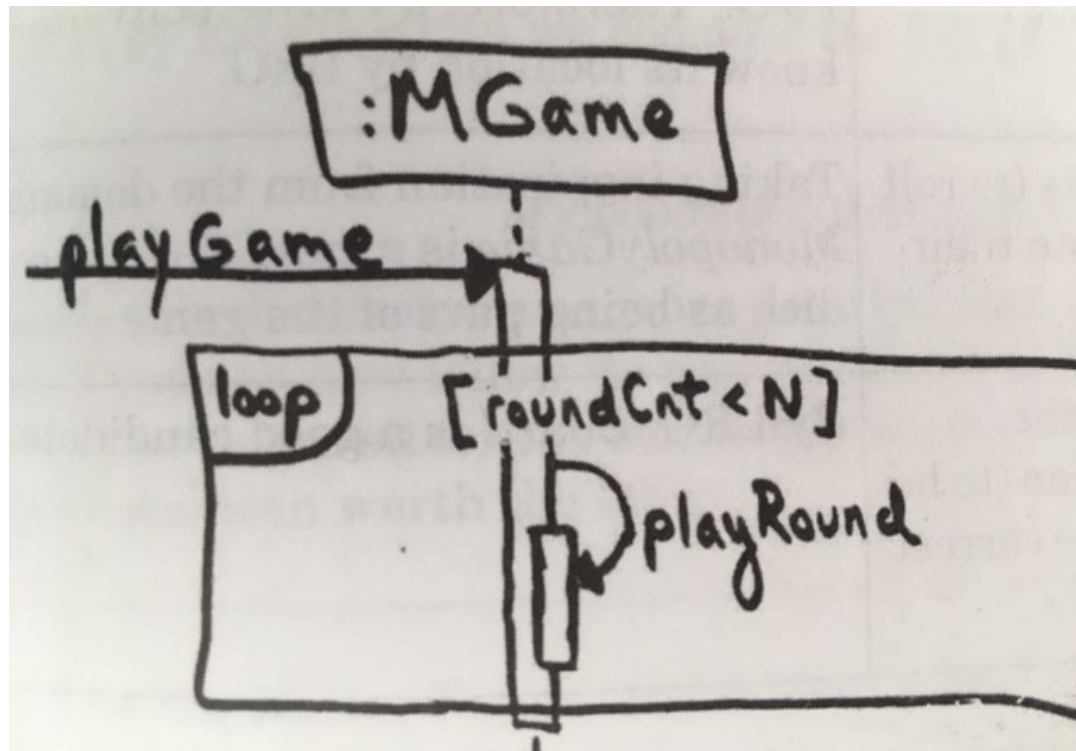
Monopoly - an Example (2)

- Who should be responsible for controlling *Game loop*?
- Let's use the Domain Model to make a decision:



Monopoly - an Example (3)

- *MonopolyGame* knows the players, so is a good choice (is information expert)



Monopoly - an Example (4)

- Who *takes a turn*?
 - Let's look at the Domain Model again
 - Candidates:
 - Player (not just because human player does the task IRL)
 - MonopolyGame
 - Board
- Guideline
 - Sometimes we need to look ahead to make a choice
 - In that case *Player* seems a fit candidate. Why?

Monopoly - an Example (5)

- *Taking a Turn* involves:
 1. Calculating random number total between 2 and 12 (range of two dice)
 2. Calculating the new square location
 3. Moving the player's piece from old location to a new square location

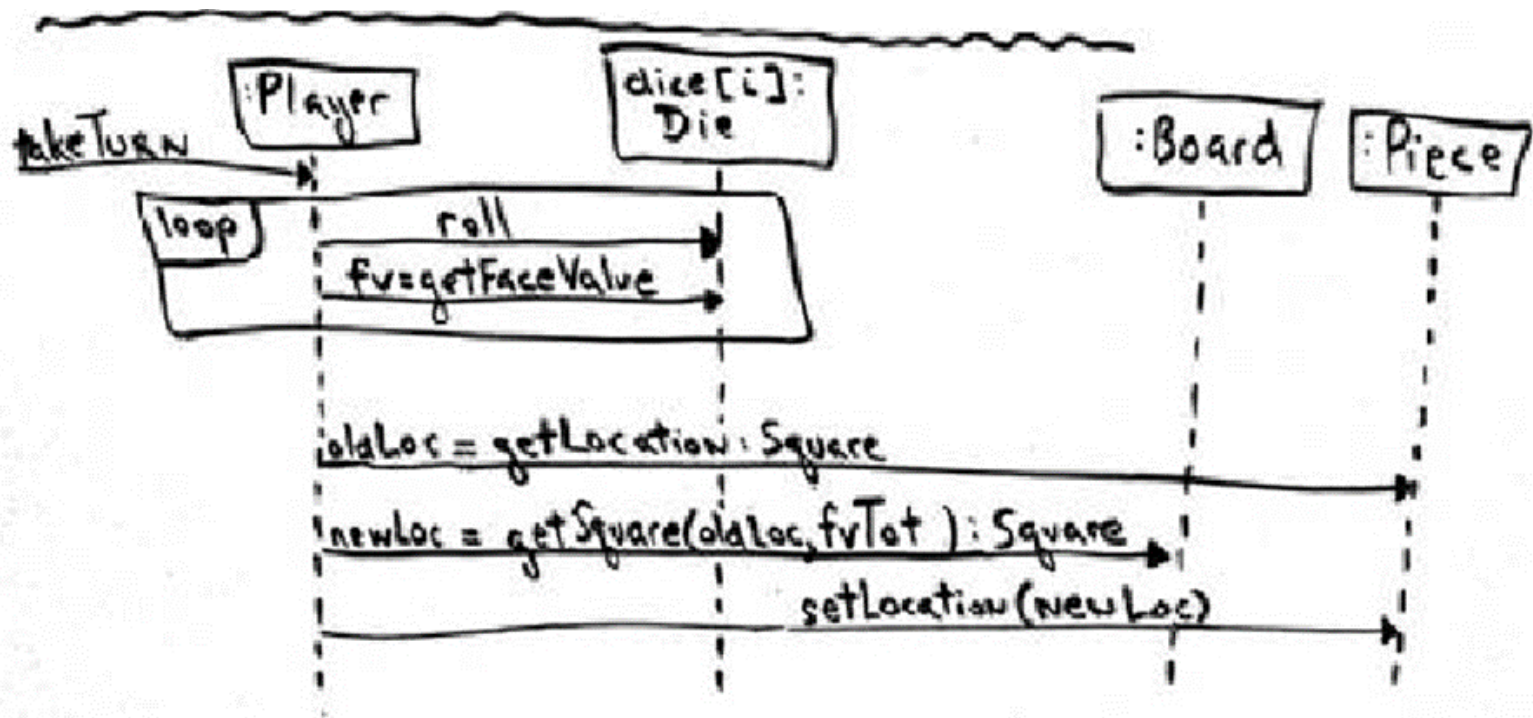
Random number problem: Die has face value and can roll

Calculate new square problem - Board knows its squares. Given an old square location + offset (dice total) → board can be responsible for knowing new location

Piece movement problem - Player knows its piece, and piece knows its location. So piece will set its new location, but could receive new location from player

Monopoly - an Example (6)

- How *Taking a Turn* design looks in a sequence diagram:



Monopoly - an Example (7)

- Demo time





Monopoly - an Example (8)

- We must remember The *Refactoring* step in TDD
- Can we get higher cohesion by using [Extract Method](#)?

```
public void takeTurn() {  
    int rollTotal = 0;  
  
    for (int i = 0; i < dice.length; i++) {  
        dice[i].roll();  
        rollTotal += dice[i].getFaceValue();  
    }  
  
    Square newLocation = board.getSquare(piece.getLocation(), rollTotal);  
    piece.setLocation(newLocation);  
}
```



Monopoly - an Example (9)

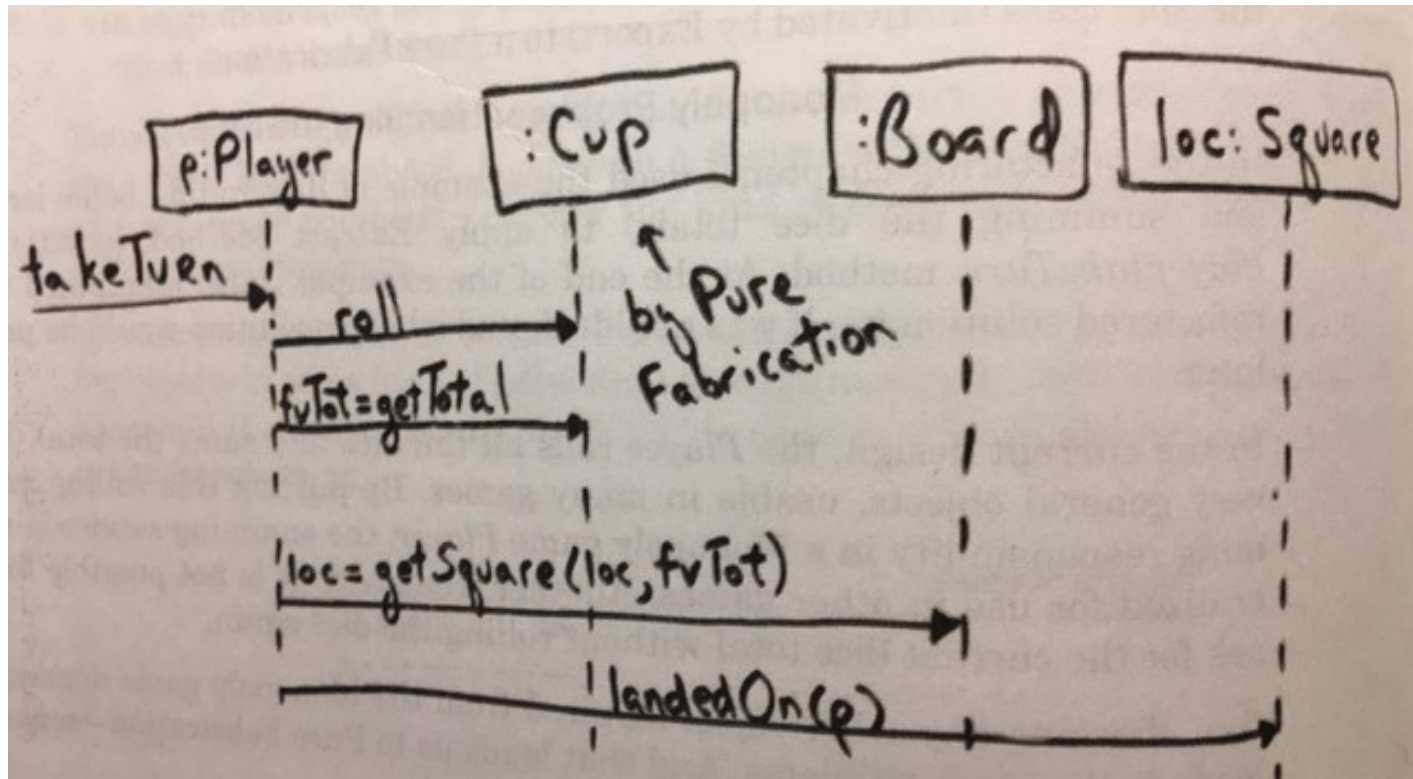
- We must remember The *Refactoring* step in TDD
- Can we get higher cohesion by using [Extract Method](#)?

```
public void takeTurn() {  
    int rollTotal = 0;  
  
    for (int i = 0; i < dice.length; i++) {  
        dice[i].roll();  
        rollTotal += dice[i].getFaceValue();  
    }  
  
    Square newLocation = board.getSquare(piece.getLocation(), rollTotal);  
    piece.setLocation(newLocation);  
}
```

Monopoly - an Example (10)

- Problems
 - How about reuse of the dice (in other apps)?
 - It is not possible to ask for current dice total without rolling again?

s
o
l
u
t
i
o
n



Workshop result for Monopoly – take turn

- <https://github.com/Tine-m/monopoly-with-mocking>