P2: Exercise 2 Discussion

Pooja Rani

March 9, 2018

Exercise 2

SkipPlayer: Skip the next player’s turn?

- Main problem: Find the next player

Approaches

- Use boolean flag
- Use an ArrayList to get random index
- (…more advanced / dynamic solutions possible …)

Wormholes

Wormhole: How to find all the exits?

- Main problem: Entrances need to be aware of all exits

Approaches

- Let the game keep track of exits
- List in WormholeExit class. Add “this” to the list when constructing an exit.
- (…more advanced / dynamic solutions possible …)

Wormholes

```
@Override
public ISquare landHereOrGoHome() {
    return this.destination().landHereOrGoHome();
}

private ISquare destination() {
    return game.wormholeExits().
        get(new Random().nextInt(game.wormholeExits().size()));
}
```

Wormholes

```
public List<ISquare> wormholeExits() {
    List<ISquare> exits = new LinkedList<>();
    for (ISquare square : squares) {
        if (square.isWormholeExit()) { exits.add(square); };
    }
    return exits;
}
```

Wormholes

```
public List<ISquare> wormholeExits() {
    List<ISquare> exits = new LinkedList<>();
    for (ISquare square : squares) {
        if (square.isWormholeExit()) { exits.add(square); };
    }
    return exits;
}
```

Game knows about Wormholes

- Is it really the Game's responsibility?

→ Not necessarily. See design pattern lecture!

Wormholes

```
public List<ISquare> wormholeExits() {
    List<ISquare> exits = new LinkedList<>();
    for (ISquare square : squares) {
        if (square.isWormholeExit()) { exits.add(square); };
    }
    return exits;
}
```

Game knows about Wormholes

- Is it really the Game's responsibility?

→ Not necessarily. See design pattern lecture!

Methods like `ISquare.isWormhole()` clutter the interface

JavaDoc: Examples

```
/**
 * Square that sends a player to a random exit square.
 */
public class WormholeEntrance extends Square {
  // ...
}
```

JavaDoc: Examples

Missing details

```
/**
 * Square that sends a player to a random exit square.
 */
public class WormholeEntrance extends Square {
  // ...
}
```

JavaDoc: Examples

```
/**
 * The class WormholeEntrance contains methods for transferring a player
 * from the square 'WormholeEntrance' to a random `WormholeExit` square.
 * It returns a randomly chosen WormholeExit when enter is called.
 */
public class WormholeEntrance extends Square {
  // ...
}
```

JavaDoc: Examples

Filler words: “The class WormholeEntrance …”

```
/**
 * The class WormholeEntrance contains methods for transferring a player
 * from the square 'WormholeEntrance' to a random `WormholeExit` square.
 * It returns a randomly chosen WormholeExit when enter is called.
 */
public class WormholeEntrance extends Square {
  // ...
}
```

JavaDoc: Examples

Better

```
/**
 * Transports entering player to a randomly selected Wormhole Exit Square.
 *
 * Requires at least one WormholeExit Square, otherwise throws IllegalStateException.
 *
 * Is is created and called inside the {@link Game} class.
 * Extends {@link TransportingSquare}.
 *
 * The first time a player lands on an entrance Square scans the board's squares
 * and adds those that are Wormhole Exits to the wormExits ArrayList.
 * Throws IllegalException if no exits are found.
 */

public class WormholeEntrance extends Square {
  // ...
}
```

Git

These are **not** good commit messages:

- No more errors!
- I hate git
- FIRST TRY
- v3
- slooowly getting there

I could go on…

Git

These are **better**:

```
Implemented SwapSquare

Implemented skipPlayer to skip turn of next player in list.
Overrides toString method
```

```
Implement wormholes

- Game.java: Implemented the WormhoeEntrance and the Wormhole Exit in the main
  method.
- WormholeEntrance.java: Get a random exit from the list of wormhole exits given
  by the game.
- WormholeExit.java: The Exit now knows that it is an exit.
```

```
Add exercise 2
```

Design by Contract, Assertions, and Exceptions

Exception or Assertion?

```
/**
 * Sets the refresh rate for the current display.
 * @param rate
 */
public void setRefreshRate(int rate) {
    // what if rate < 0?
}
```

Exception or Assertion?

```
/**
 * Sets the refresh rate for the current display.
 * @param rate new refresh rate, must be >= 0
 */
public void setRefreshRate(int rate) {
    assert rate >= 0;
}
```

Exception or Assertion?

```
/**
 * Sets the refresh rate for the current display.
 * @param rate new refresh rate
 * @throws IllegalArgumentException if rate is not valid
 */
public void setRefreshRate(int rate)
        throws IllegalArgumenException {
    if (rate < 0) {
        throw new IllegalArgumentException();
    }
}
```

Assertions

- Use when you expect a property to hold
- Use for contracts
  - Pre-/postconditions, invariants
- Use inside complex code
  - For example in an algorithm to make sure an intermediate result holds

Assertions

```
/**
  * Draw a vertical line, starting from position,
  * with a length of steps + 1.
  *
  * @param position start location of the line, must not be null
  * @param steps length of the line
  */
public void drawVertical(Point position, int steps) {
    assert position != null;
    // Implementation omitted
    assert(invariant());
}
```

Assertions

- Favor assertions/preconditions for checking method parameteres in private/internal API
  - Senders come from within your project ⇒ go fix the bug!
  - Simplifies design
- Use assertions for postconditions and invariants

Exceptions

- Error handling
- Expected behaviour
  - Deal with it in try-catch blocks, or
  - throw it up to the caller

```
public void matches(String filename)
        throws NotImplementedException {
    throw new NotImplementedException();
}
```

Exceptions

Do not abuse exceptions

```
try {
    int index = 0;
    while (true) {
        players[index++] = new Player();
    }
} catch (ArrayIndexOutOfBoundsException e) {}
```

Exceptions

Do not abuse exceptions

```
for (int index = 0; index < players.length; index++) {
    players[index] = new Player();
}
```

Exceptions

- Favor exceptions for checking method parameters in public/external API
  - Can’t trust user to read JavaDoc
- **Always use exceptions to check user input!**

Checked and Uncheked Exceptions

- Checked exceptions must either be declared

```
public void foobar() throws TodoException { /* ... */ }
```

- or wrapped inside a try-catch block

```
public void foobar() {
    try {
        // something that throws a TodoException
    } catch (TodoException e) {
        // handle exception
    }
}
```

- Use checked exceptions **unless you have a very good reason not to**!

NullPointerException

- Very common unchecked exception
- Often hard to tell where it came from
  - Value may be passed around for a while before it is used

→ Include null checks where appropriate

NullPointerException

```
private void newGame() {
    setPlayer(null);
    execute();
}
private void setPlayer(Player player) {
    this.player = player;
}
private void execute() {
    this.player.move();
}
```

NullPointerException

```
private void newGame() {
    setPlayer(null);
    execute();
}
private void setPlayer(Player player) {
    this.player = player;
}
private void execute() {
    this.player.move();
}
```

```
Exception in thread "main" java.lang.NullPointerException
at exercise_03.SomeClass.execute(SomeClass.java:79)
at exercise_03.SomeClass.newGame(SomeClass.java:65)
at exercise_03.SomeClass.main(SomeClass.java:7)
…
Process finished with exit code 1
```

NullPointerException

```
private void newGame() {
    setPlayer(null);
    execute();
}
private void setPlayer(Player player) {
    this.player = player;
}
private void execute() {
    this.player.move();
}
```

```
Exception in thread "main" java.lang.NullPointerException
at exercise_03.SomeClass.execute(SomeClass.java:79)
at exercise_03.SomeClass.newGame(SomeClass.java:65)
at exercise_03.SomeClass.main(SomeClass.java:7)
…
Process finished with exit code 1
```

Why is player == null here?

NullPointerException

```
private void newGame() {
    setPlayer(null);
    execute();
}
/** @param player must not be null */
private void setPlayer(Player player) {
    assert player != null;
    this.player = player;
}
private void execute() {
    this.player.move();
}
```

NullPointerException

```
private void newGame() {
    setPlayer(null);
    execute();
}
/** @param player must not be null */
private void setPlayer(Player player) {
    assert player != null;
    this.player = player;
}
private void execute() {
    this.player.move();
}
```

```
Exception in thread "main" java.lang.AssertionError
at exercise_03.SomeClass.setPlayer(SomeClass.java:74)
at exercise_03.SomeClass.newGame(SomeClass.java:64)
at exercise_03.SomeClass.main(SomeClass.java:7)
Process finished with exit code 1
```

Another example

```
/**
  * Look up the object at the top of
  * this stack and return it.
  *
  * @return the object at the top
  */
public E top() {
    return top.item;
}
```

Another example

```
/**
  * Look up the object at the top of
  * this stack and return it.
  *
  * @return the object at the top
  */
public E top() {
    return top.item;
}
```

What if the stack is empty?

Another example

```
/**
  * Look up the object at the top of
  * this stack and return it.
  * Returns null if called on an empty stack.
  *
  * @return the object at the top
  */
public E top() {
    if (this.isEmpty())
        return null;
    return top.item;
}
```

Another example

```
/**
  * Look up the object at the top of
  * this stack and return it.
  * Returns null if called on an empty stack.
  *
  * @return the object at the top
  */
public E top() {
    if (this.isEmpty())
        return null;
    return top.item;
}
```

What if the stack contains null values?

Another example

```
/**
  * Look up the object at the top of
  * this stack and return it.
  * Throws an EmptyStackException this
  * stack is empty.
  *
  * @return the object at the top
  */
public E top() throws EmptyStackException {
    if (this.isEmpty())
        throw new EmptyStackException();
    return top.item;
}
```

Another example

```
/**
  * Look up the object at the top of
  * this stack and return it.
  * Throws an EmptyStackException this
  * stack is empty.
  *
  * @return the object at the top
  */
public E top() throws EmptyStackException {
    if (this.isEmpty())
        throw new EmptyStackException();
    return top.item;
}
```

:-)

Introduction to UML

Mathias Stocker

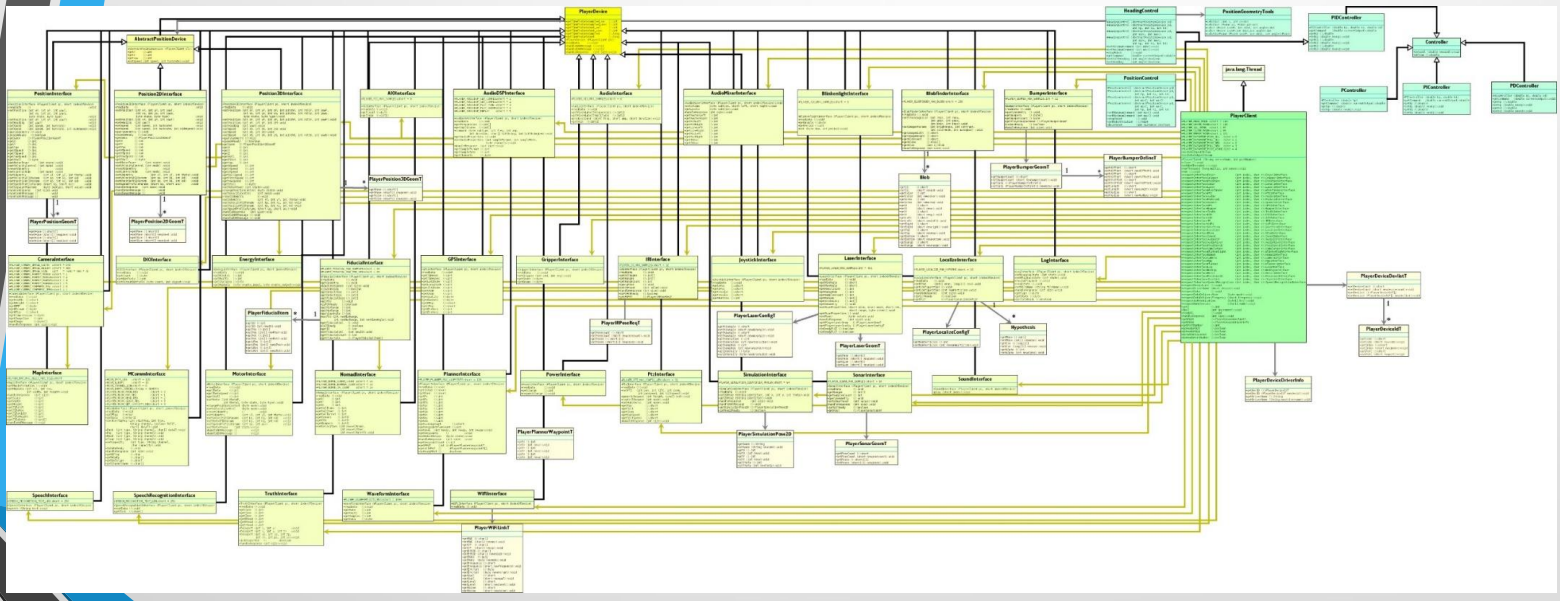
Main areas of application

- Documentation
- Drafts

Documentation

- Can be done automatically
- Can be an «overkill»

Documentation



source: java-player.sourceforge.net

A draft helps you to…

… simplify reality

… unterstanding an existing solution

… deciding how to build something from scratch

… capture requirements and discuss your idea with others

… reduce your effort to test different approaches

Modeling your system...

structure

- class diagram
- component diagram
- composite structure diagram
- object diagram
- package diagram
- profile diagram

behaviour

- activity diagram
- comunication diagram
- interaction overview diagram
- sequence diagram
- state machine diagram
- timing diagram

Modeling your system...

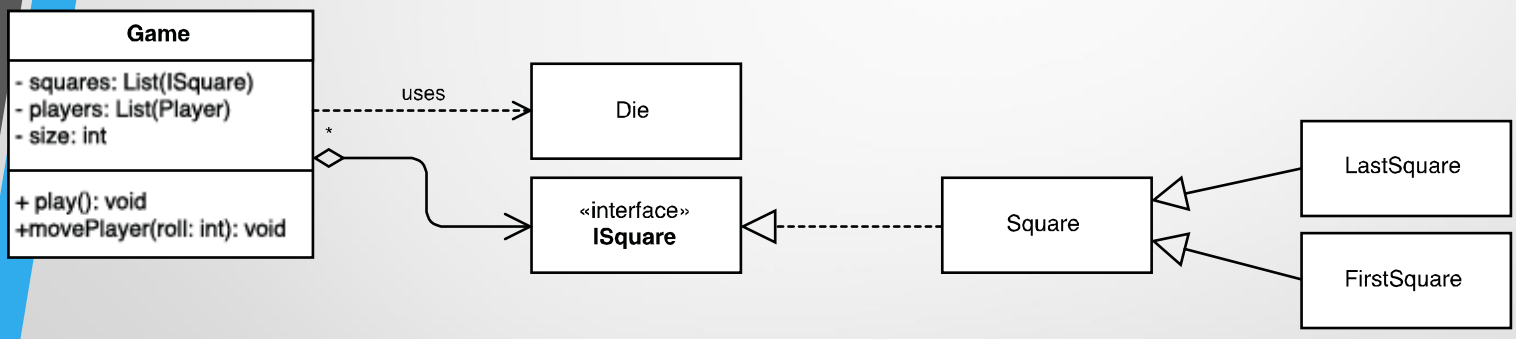
structure

- class diagram
- component diagram
- composite structure diagram
- object diagram
- package diagram
- profile diagram

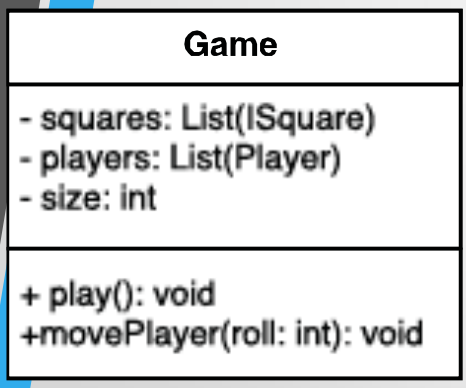
behaviour

- activity diagram
- comunication diagram
- interaction overview diagram
- sequence diagram
- state machine diagram
- timing diagram

Class diagram



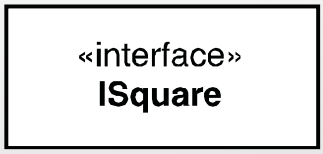
Classes and Interfaces



Name

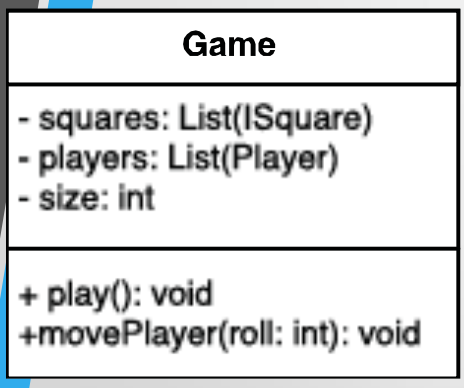
Attributes

Methods



Interface annotation

Classes and Interfaces

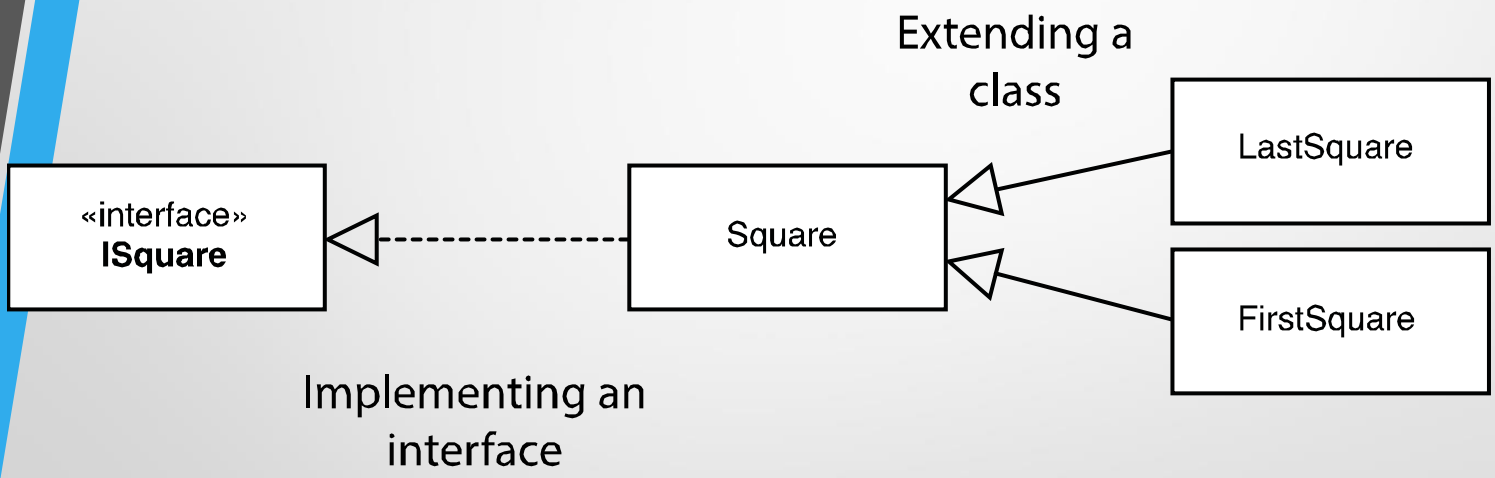


Access modifiers
+ public, - private, # protected, static

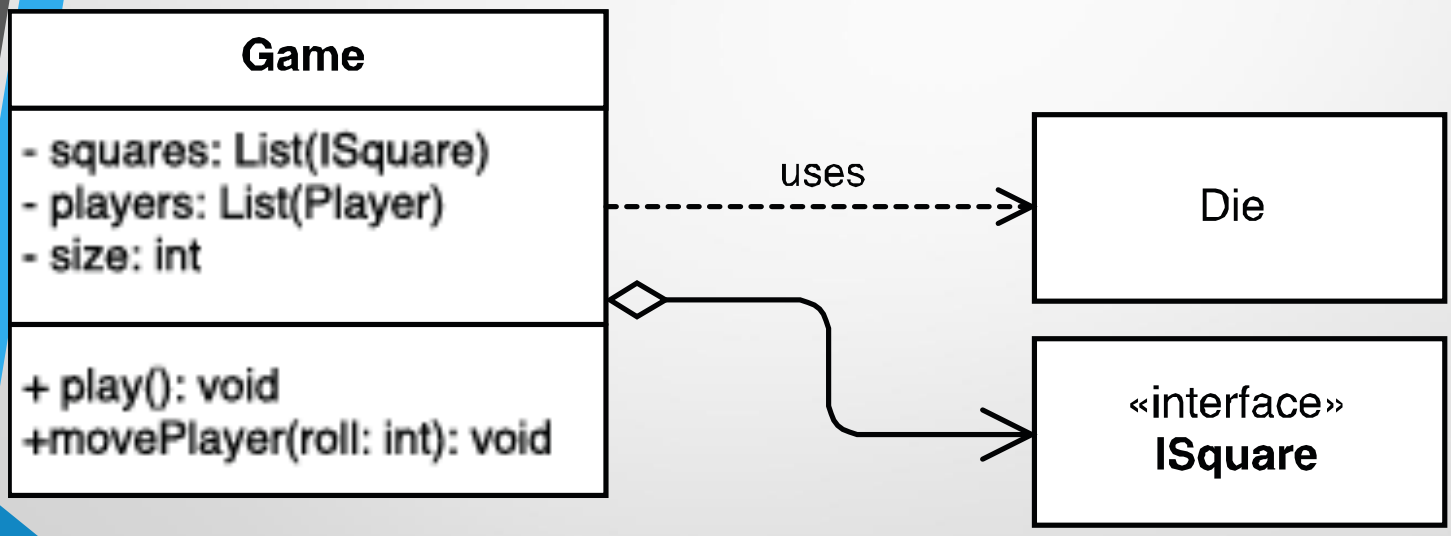
Attributes
acessIdentifier: type
Example: - size: int

Methods
accessIdentifier(parameter: type): returnType

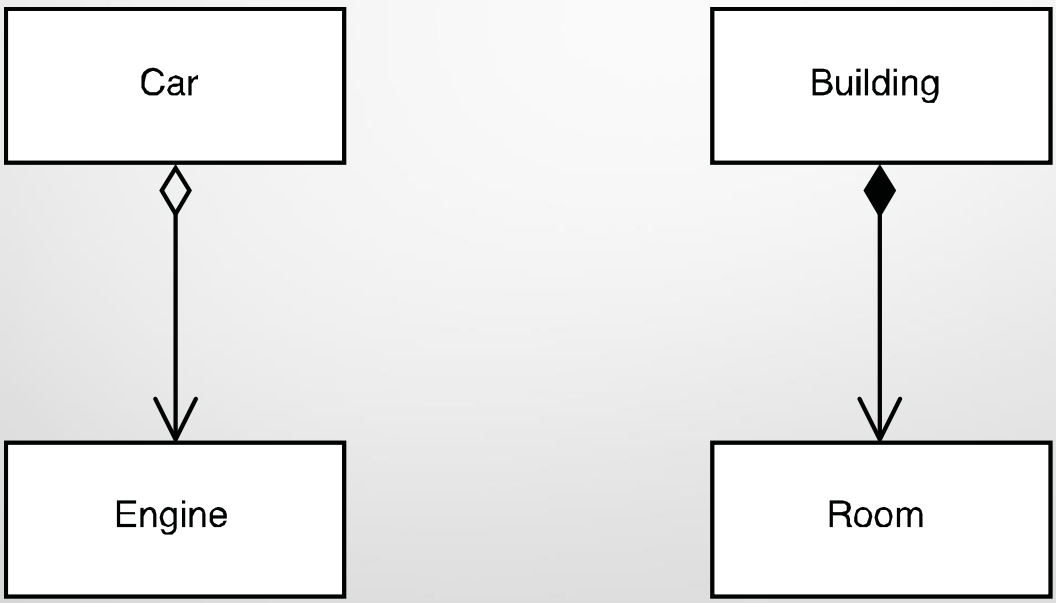
Implementation and extension



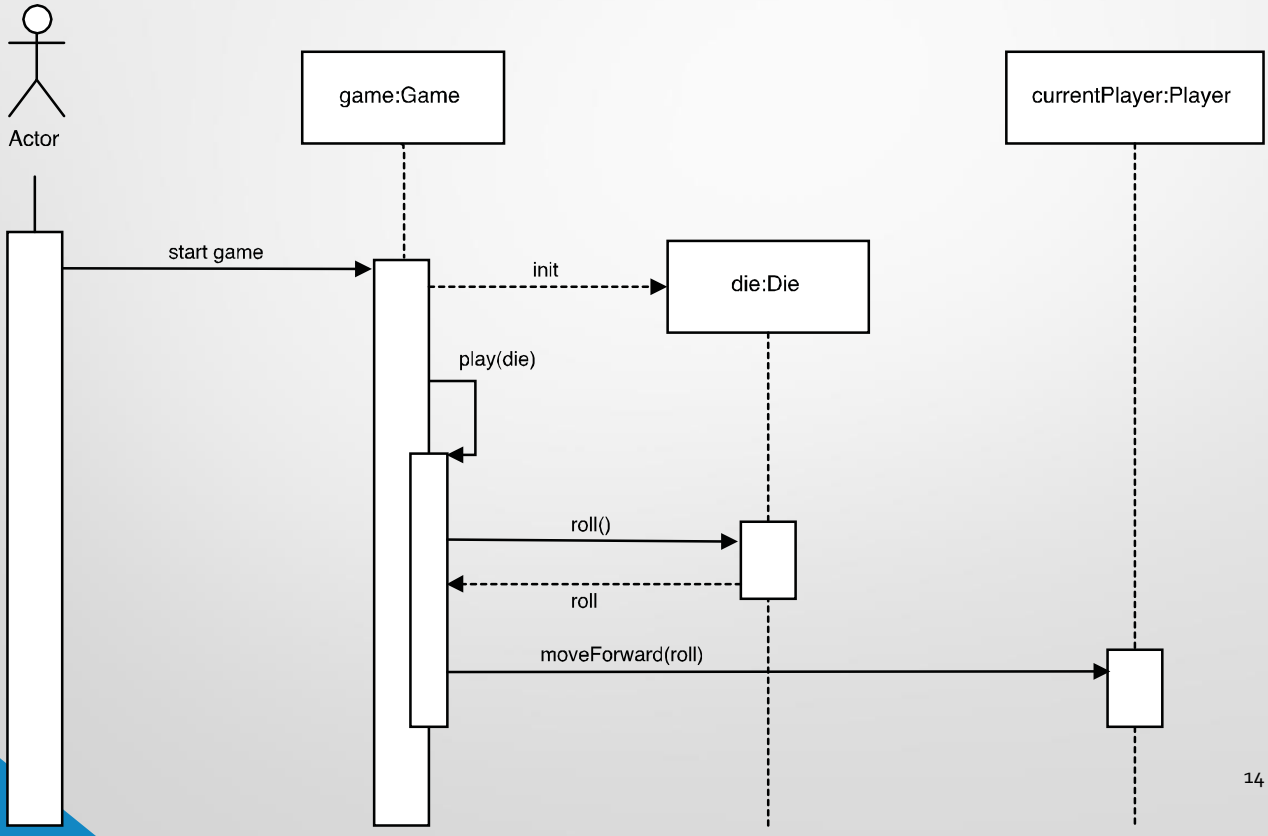
Dependency


Game
- squares: List(ISquare)
- players: List(Player)
- size: int
+ play(): void
+movePlayer(roll: int): void
uses
Die
«interface»
ISquare

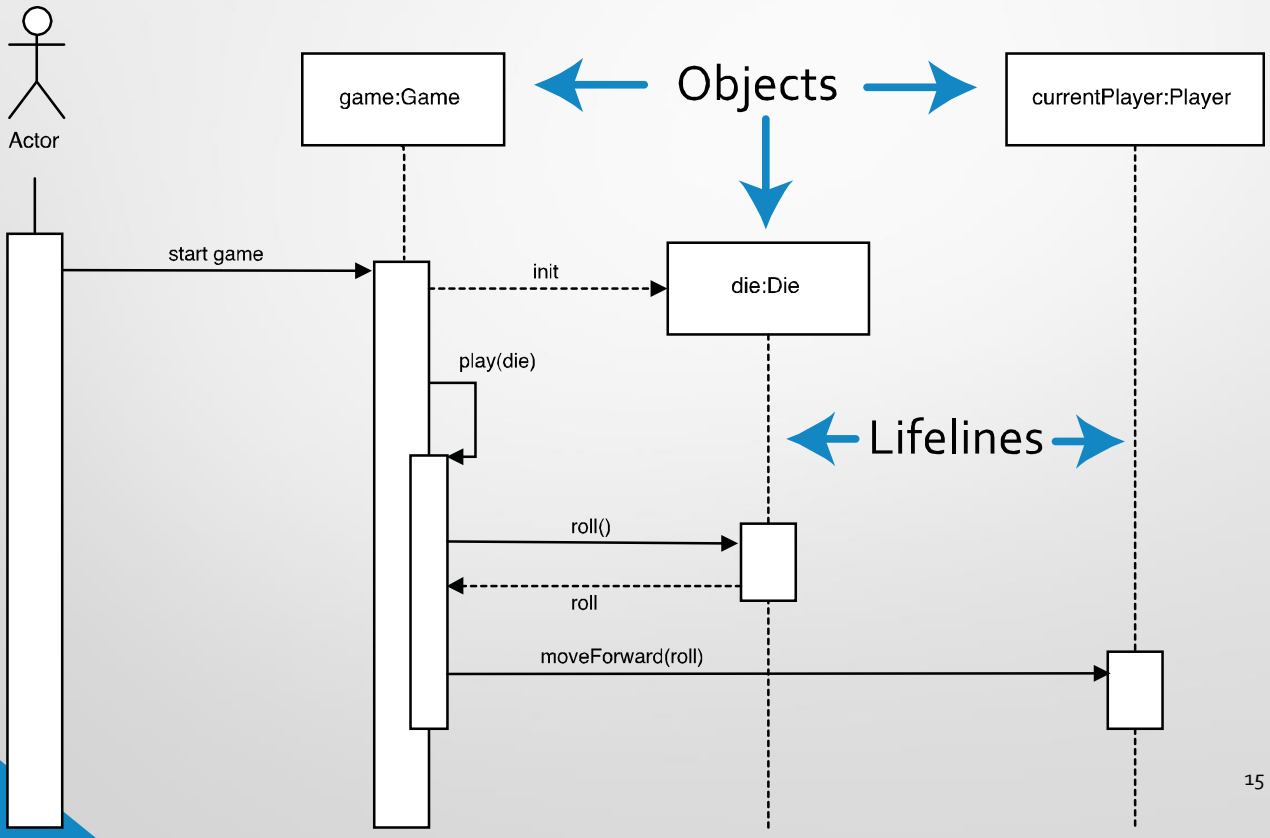
Aggregation vs. Composition



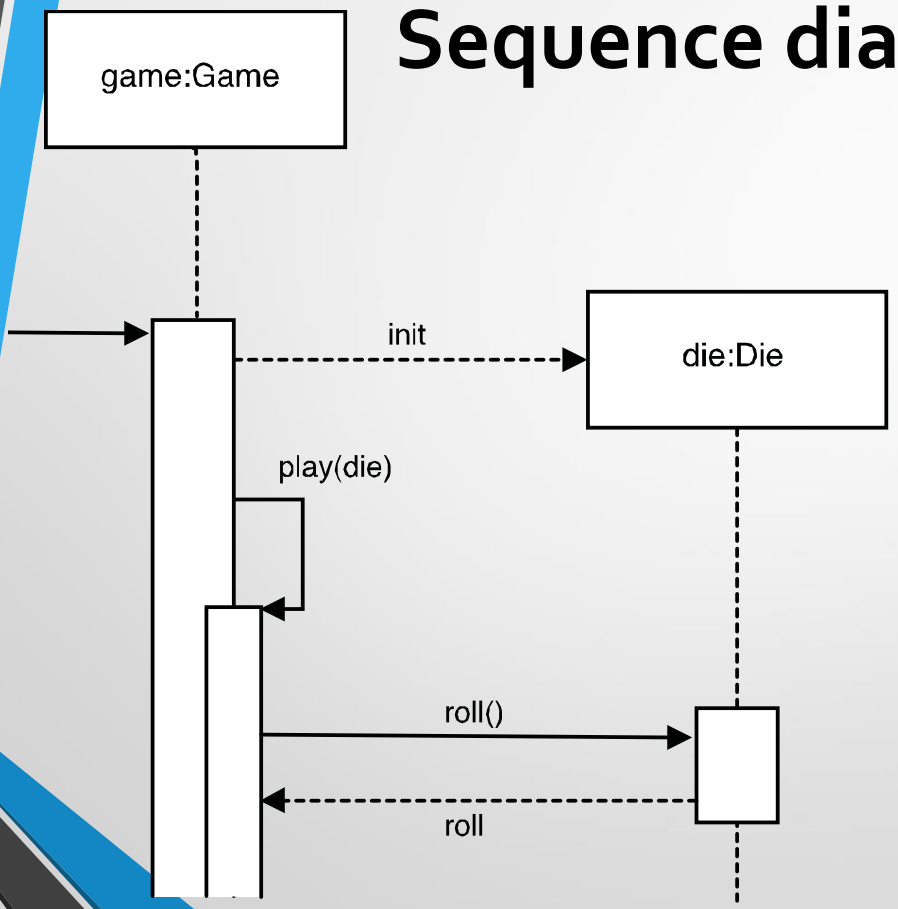
Sequence diagram



Sequence diagram



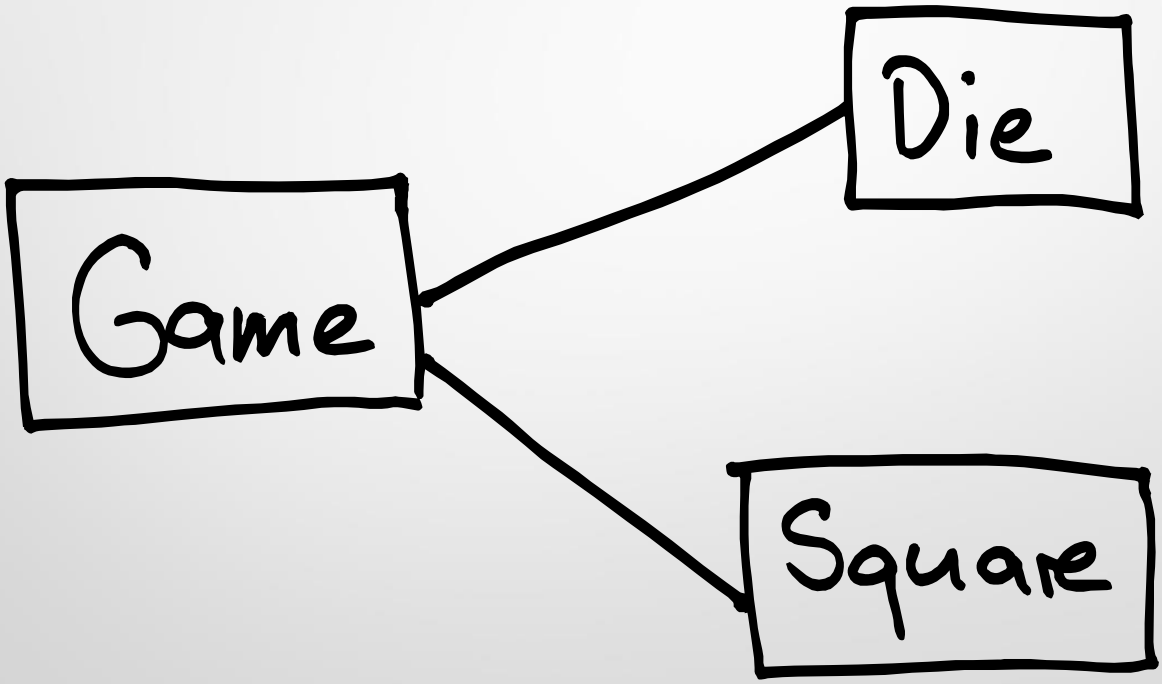
Sequence diagram



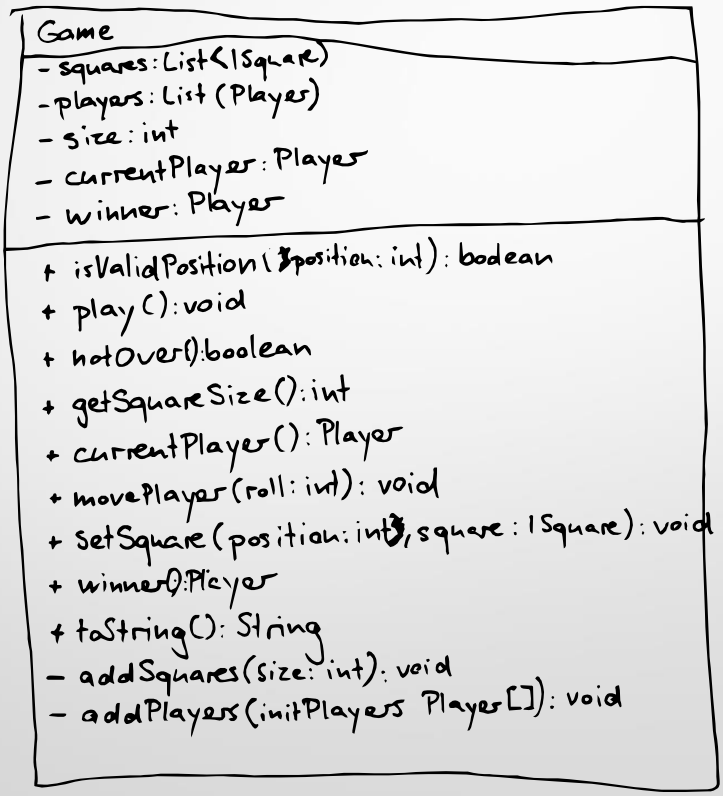
Keep in mind

- Different aspects, different diagram type
- Keep it simple
- Focus on what you want to communicate, forget the rest

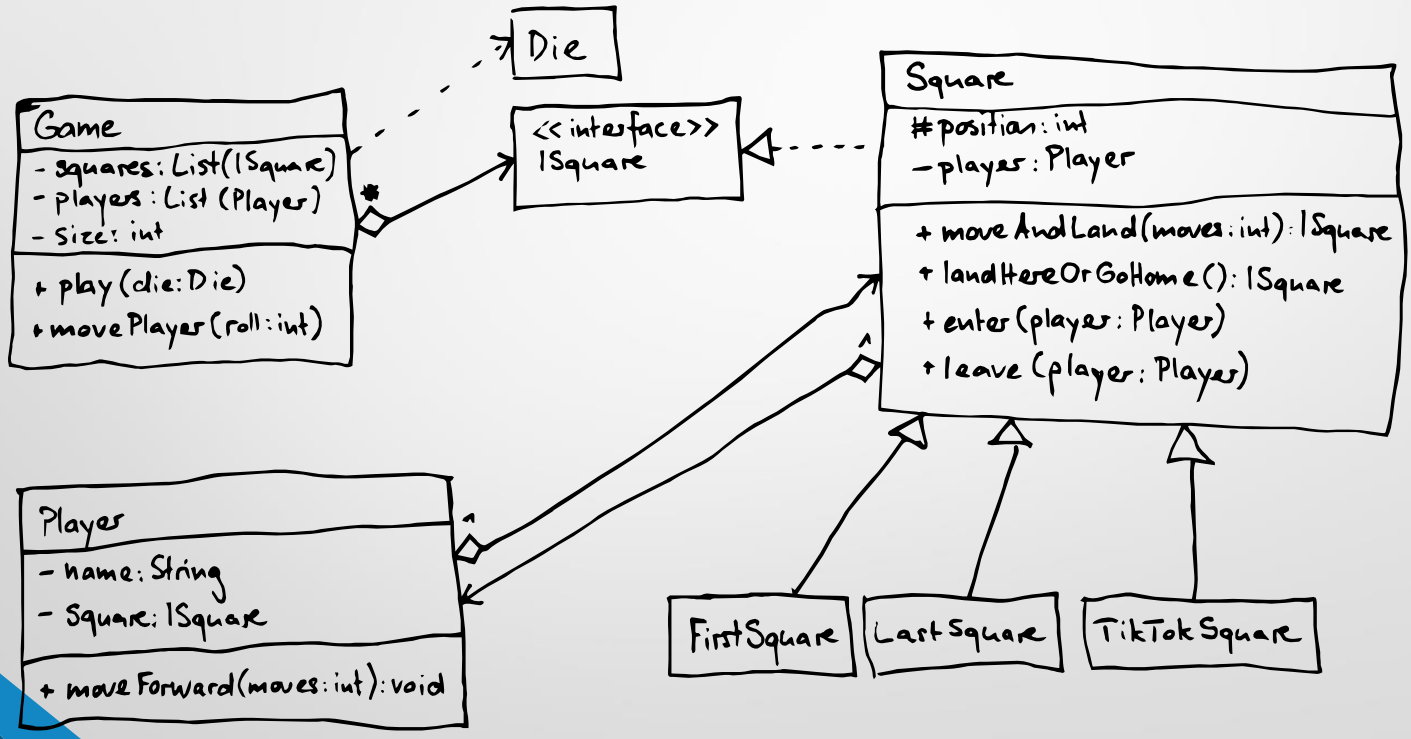
On paper: Not enough information



On paper: Too much information



On paper


Die
Game
- squares: List(ISquare)
- players: List (Player)
- size: int
+ play (die: Die)
+ movePlayer (roll: int)
<<interface>>
ISquare
Square
#position: int
-player: Player
+ moveAndLand(moves: int): ISquare
+ landHereOrGoHome(): ISquare
+ enter (player: Player)
+ leave (player: Player)
Player
- name: String
- Square: ISquare
+ moveForward(moves: int): void
FirstSquare
LastSquare
TikTokSquare

Exercise 3

Use the information from the lecture and from this presentation to solve the UML related tasks in Exercise 3

Add both diagrams in a common format (e.g. JPG, PDF) to the exercise root in your group folder.

If you do not have a scanner, you can just take a photo of the UML diagrams with a smartphone.

To learn more

- http://scg.unibe.ch/teaching/p2/ (P2 reading material, UML Reference)
- Book: UML Distilled, Martin Fowler

Exercise 3

Turtle Game

Demo

Turtle Game

- A turtle that moves around a 100x100 board
  - Move left, right, up, down
  - Leave a red trail

- Input: String representing a turtle program

```
right 5
down 4
left 3
jump 20 20
down 10
```

Turtle Game

- You start with
  - TurtleRenderer: GUI
  - BoardMaker: Class that gets text from GUI and returns a Boolean array of size 100x100
- You implement
  - Parse input program (split lines into commands)
  - Execute turtle actions
  - Keep track of trail

Turtle Game

- You start with
  - TurtleRenderer: GUI
  - Boar urns a Boolean array

As always: git pull p2-exercises master

Read exercise_03.md

- You implement
  - Parse input program (split lines into commands)
  - Execute turtle actions
  - Keep track of trail