P2: Design By Contract

Lorenzo Wipfli
12 March 2021

Contents

- Feedback Exercise 2
  - SwapSquare
  - WormHoleEntrance
  - SkipSquare
  - JavaDoc
  - Git
- Design by Contract
  - Assertions
  - Exceptions
- UML
- Exercise 3

Exercise 2: SwapSquare

Idea:

- Ask yourself, does the player stay on this square or not? Where would you place the logic?
- Get the target (or next) Player.
- Get the current position of target player.
- Move the target player to the swapsquare.
- Move the current player to the target player’s square.
- Note: Watch out that there is no swapping loop!

Exercise 2: SwapSquare

```
@Override
public ISquare landHereOrGoHome() {
           if(this.isOccupied())
                      return game.firstSquare();

           //logic to prevent infinite swap loop
           ...


           //Get the next player to change with
           Player nextPlayer = game.currentPlayer();

           //Get square on which that player is
           ISquare changeSquare = nextPlayer.square();

           //Tell the next player to move...
           ...


           return changeSquare.landHereOrGoHome();
           }

}
```

Exercise 2: WormholeEntrance

```
Idea:
•  Ask yourself, does the player stay on this square or not? Where would you place
   the logic?
•  Get all available wormhole exits.
•  Choose one at random (for example with Random().nextInt(int scope) gives a
   number from 0 to scope-1.)
•  Place the player at the exit.
```

Exercise 2: SkipSquare

Idea:

- Ask yourself, does the player stay on this square or not? Where would you place the logic?
- Tell the game to skip the next player.
- Use a boolean attribute maybe?

JavaDoc: Examples

```
/**
 *
 */
pub
     // …
}
```

Missing details

JavaDoc: Examples

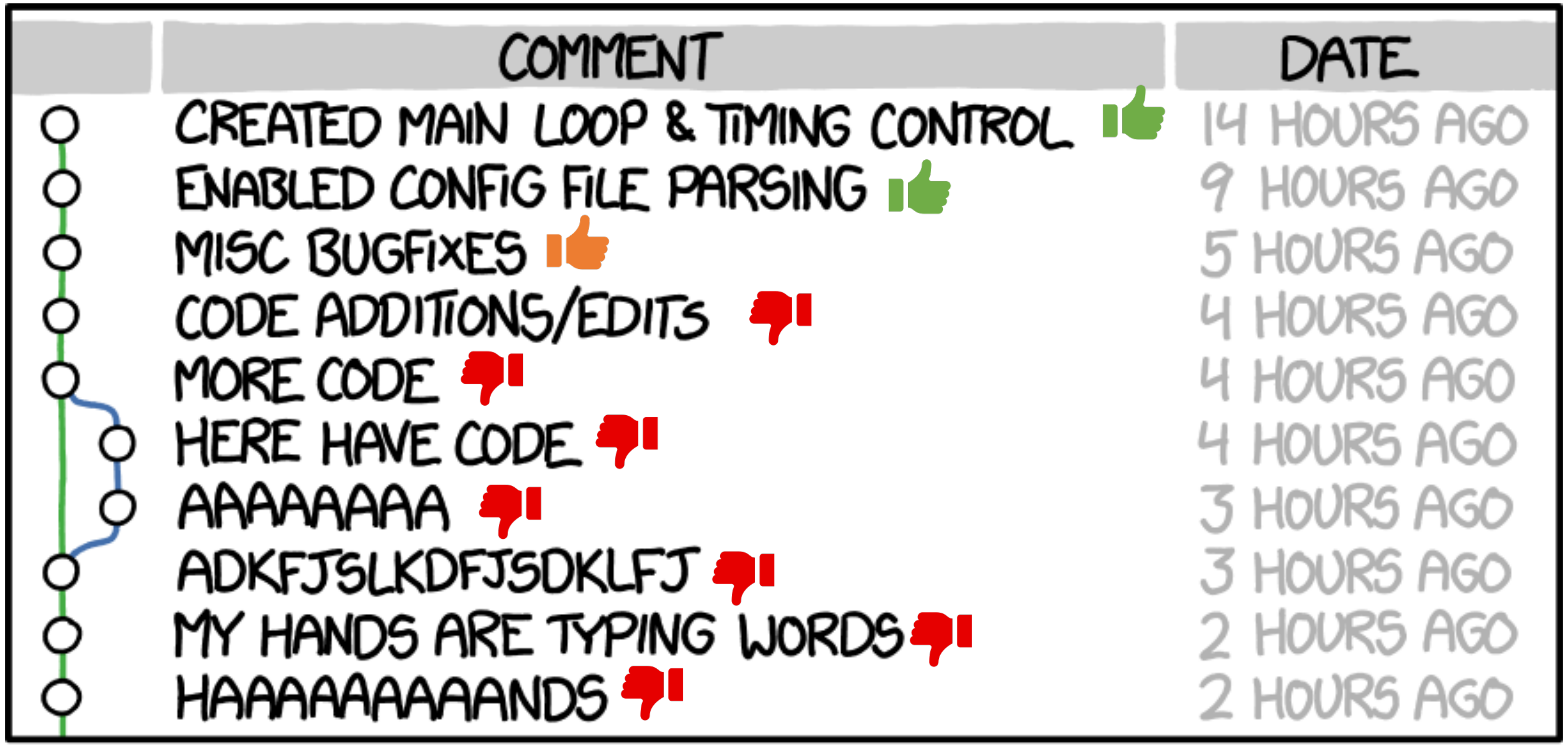
```
/**
 * The class SkipSquare contains functionality that
 *
 */
pub
     // …
}
```

Filler words: The class SkipSquare

JavaDoc: Examples

```
/**
 * Skips the next player after the current one.
 *
 * Is created and called inside the {@link Game} class.
 * Extends {@link Square}.
 *
 */
public class SkipSquare extends Square implements ISquare {
       // …
}
```

Git-messages



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL 👍	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING 👍	9 HOURS AGO
○	MISC BUGFIXES 👍	5 HOURS AGO
○	CODE ADDITIONS/EDITS 👎	4 HOURS AGO
○	MORE CODE 👎	4 HOURS AGO
○	HERE HAVE CODE 👎	4 HOURS AGO
○	AAAAAAAA 👎	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ 👎	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS 👎	2 HOURS AGO
○	HAAAAAAAANDS 👎	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

https://xkcd.com/1296/

Git-messages

- `No more errors!`
- `I hate git`
- `test`
- `first try`
- `solving exercise`
- `Here have some code`
- `Changes`
- `Fix`
- `.`
- `Remove if`
- `Do you see this?`
- `I have seen it yes.`
- `Its sunny outside.`

Git-messages

- `Implemented SwapSquare`
- `Implemented SkipSquare which skips the next player of the current Game.`
- `Added Player.toString() method.`

DBC - Example

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

DBC - Assertion Example

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

DBC – Exception Example

```
/**
 * 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 IllegalArgumentException {
       if (rate < 0) {
              throw new IllegalArgumentException();
       }
}
```

DBC – When to use Assertions

- Use when you expect a property to hold
- Calls inside the program
- Use for contracts
  - Pre-/postconditions, invariants
  - Simplifies design
- Use inside complex code
  - For example to make sure an intermediate result holds

Assertions – Pre-, and Postconditions

```
/**
 * 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;     // This is a precondition
       // Implementation here
       assert(invariant());         //This is a postcondition
}
```

DBC – When to use 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!

Exceptions

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

DBC – Checked Exceptions

- Declared Exception

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

- Wrapped inside a try-catch block

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

- Always use checked exceptions unless there is a **very good** reason not to!

NullPointerException

- Very common unchecked exception
- Often hard to tell where it originated
  - Value may be passed around for a while before it is used
- Include `null` checks where appropriate

NullPointerException

```
private void newGame() {
        se
        ex
}

private v
        th
}

private v
        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
```

we do not know why player == null

Exceptions

```
private void newGame() {
        setPlayer(null);
        ex
}
/** @para
private v
        as

}
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
```

Stacktrace shows where Nullpointer occured

DBC - Example

```java
/**
 * 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;
}
```

DBC - 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;
}
```

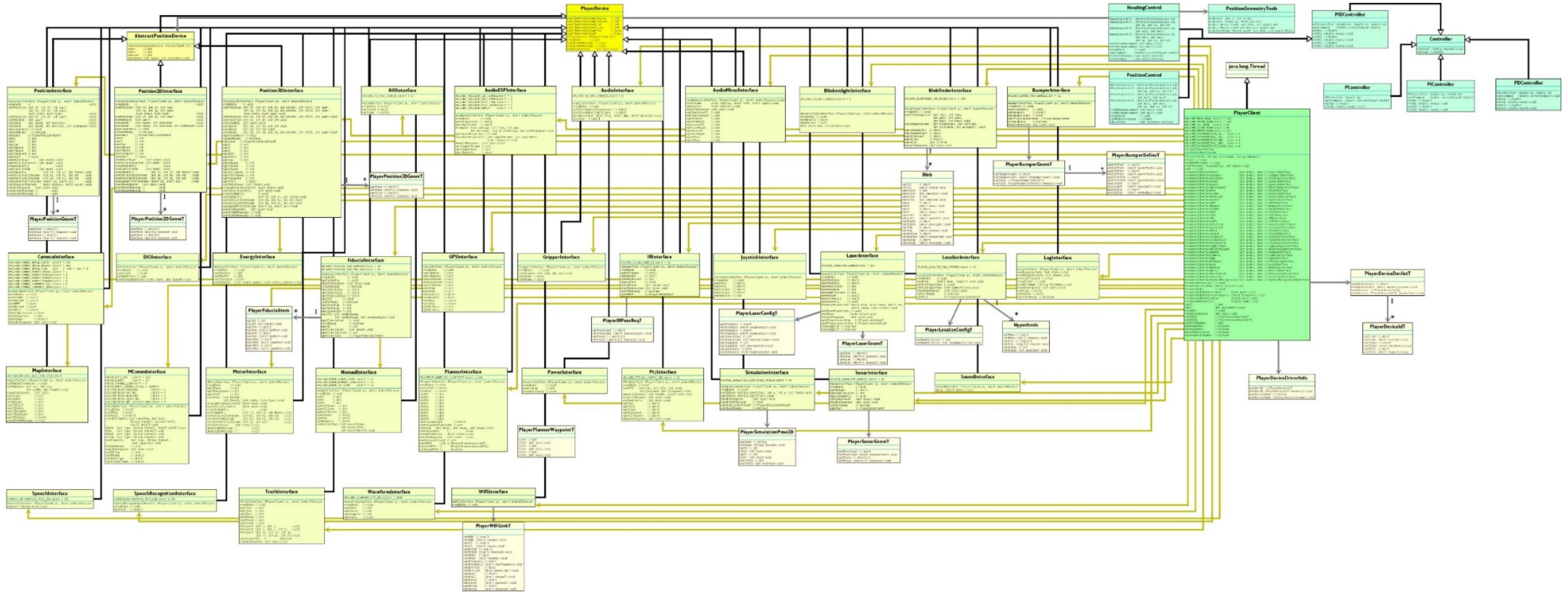
DBC - Example

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

UML

- Documentation
    - Can be done automatically
        - Can be an overkill (next slide)
- Drafts
    - Simplify reality
    - Understand 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

UML - Documentation



UML - Categories

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

UML - Categories

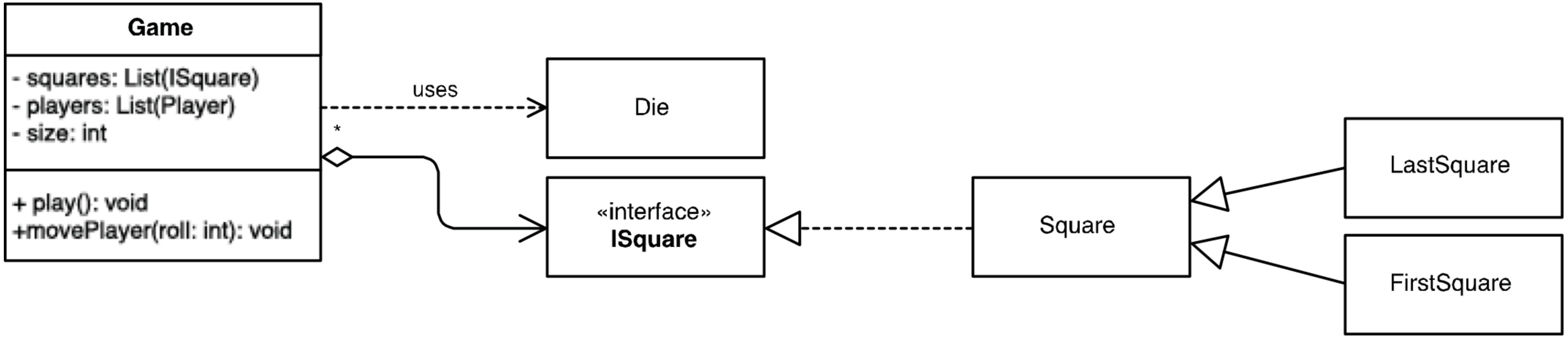
structure

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

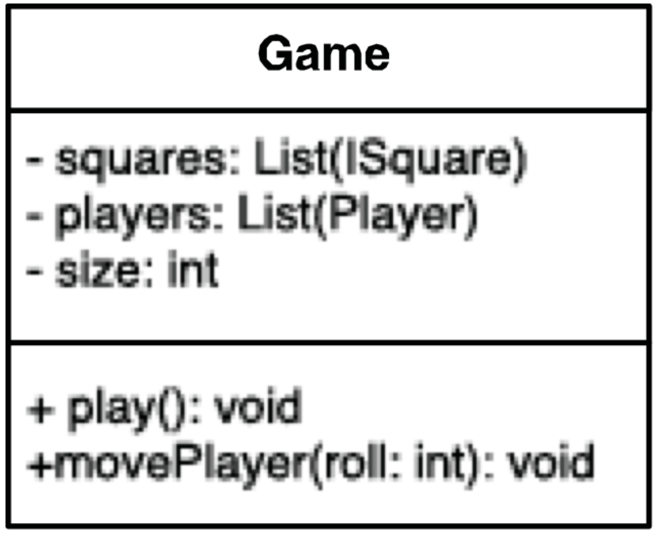
behaviour

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

UML - Example


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

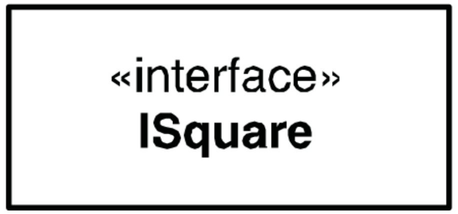
UML



Name

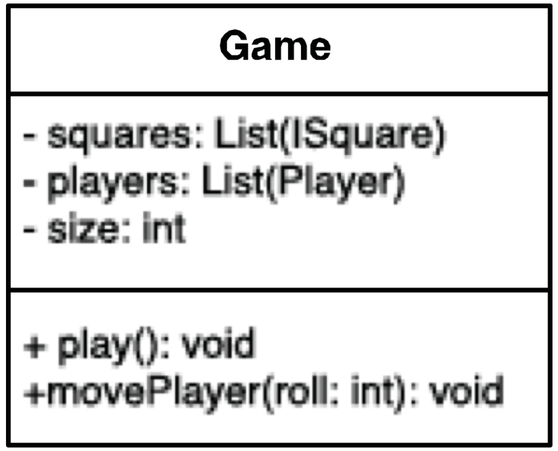
Attributes

Methods



Interface annotation

UML – Class annotation

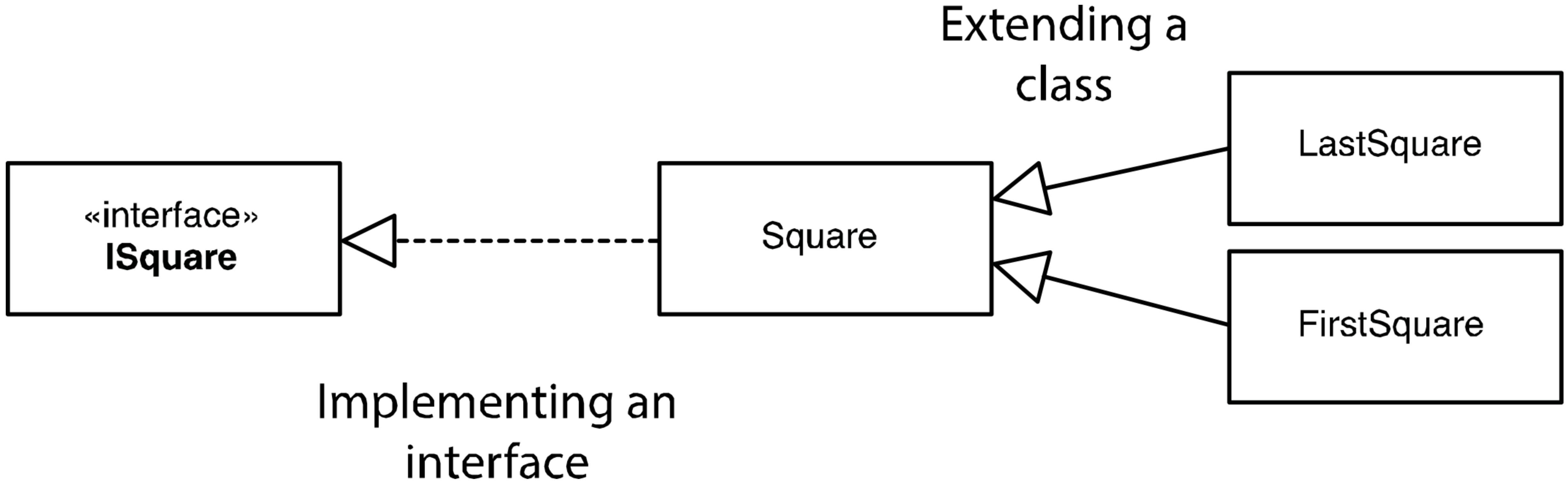


Access modifiers:
+ public, - private, # protected, <u>static</u>

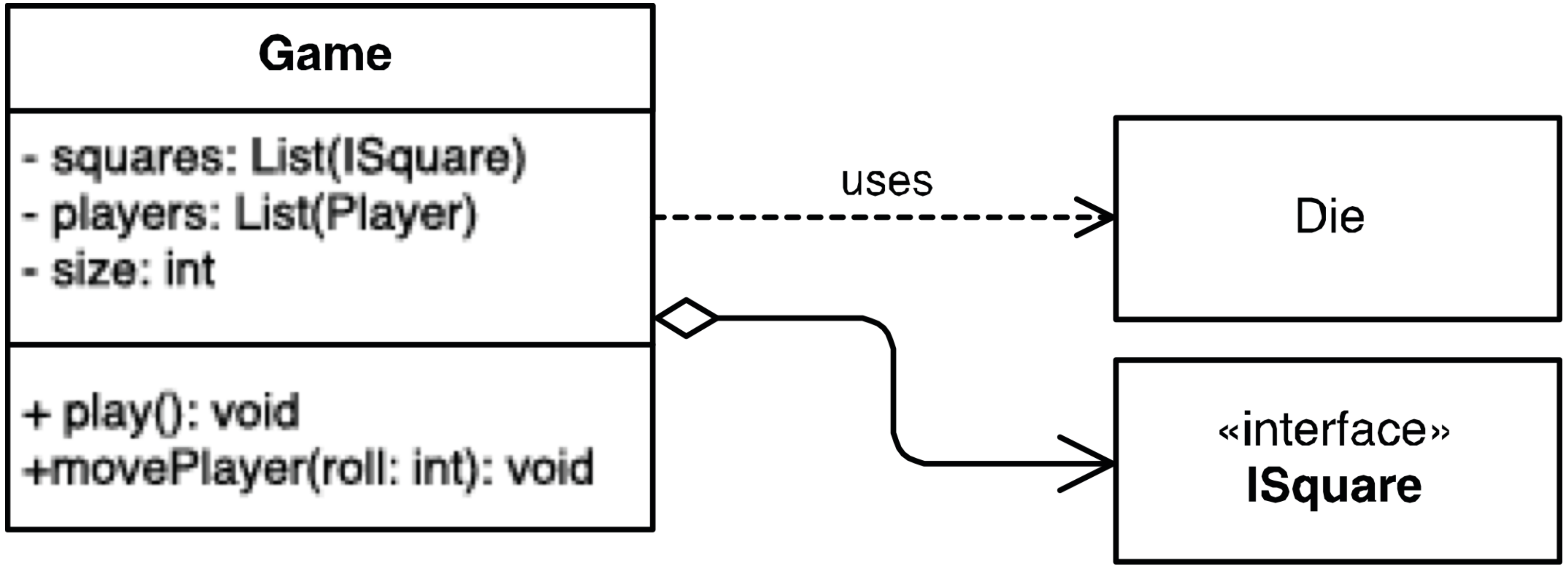
Attributes:
acessIdentifier: type
Example: - size: int

Methods:
accessIdentifier(parameter: type): returnType

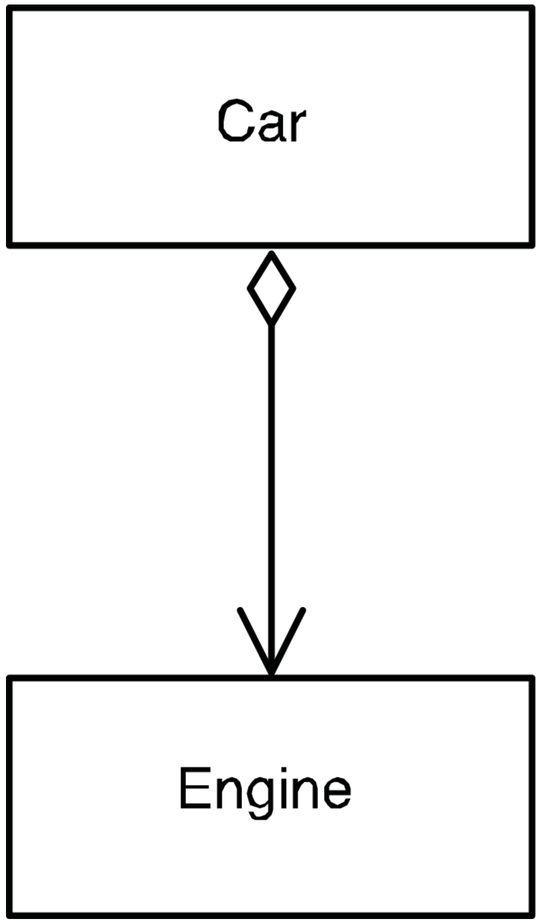
UML - Relationships

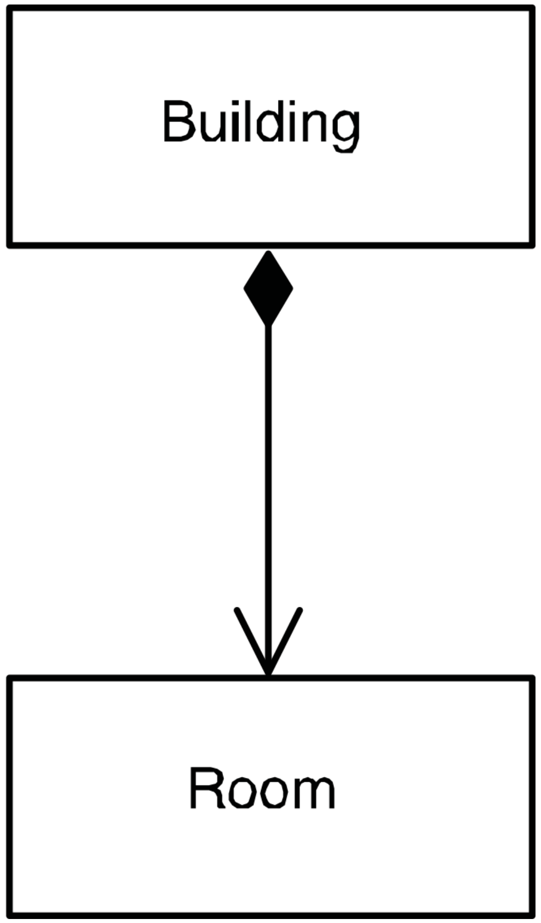


UML - Relationships

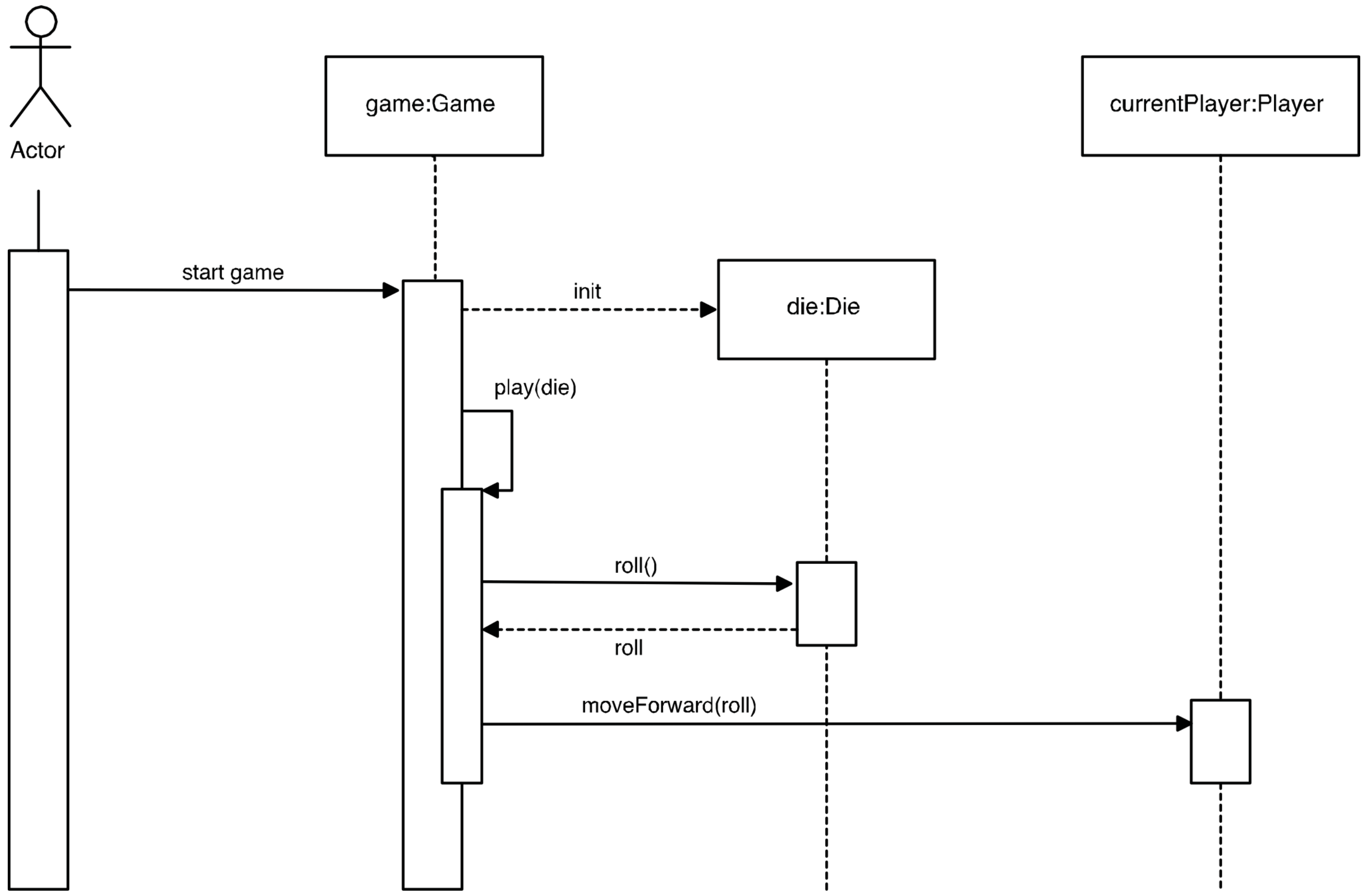


UML – Aggregation vs Composition

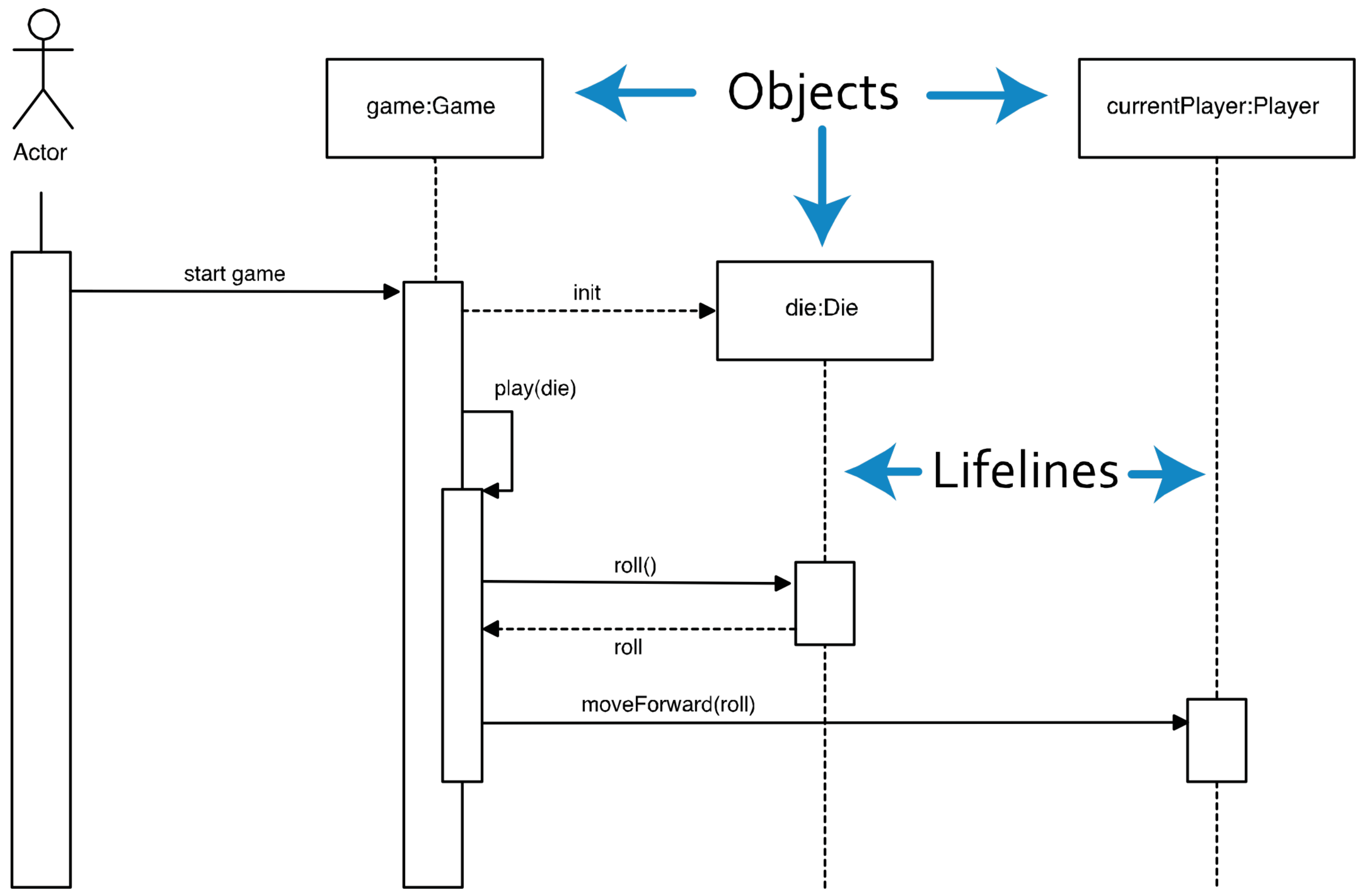




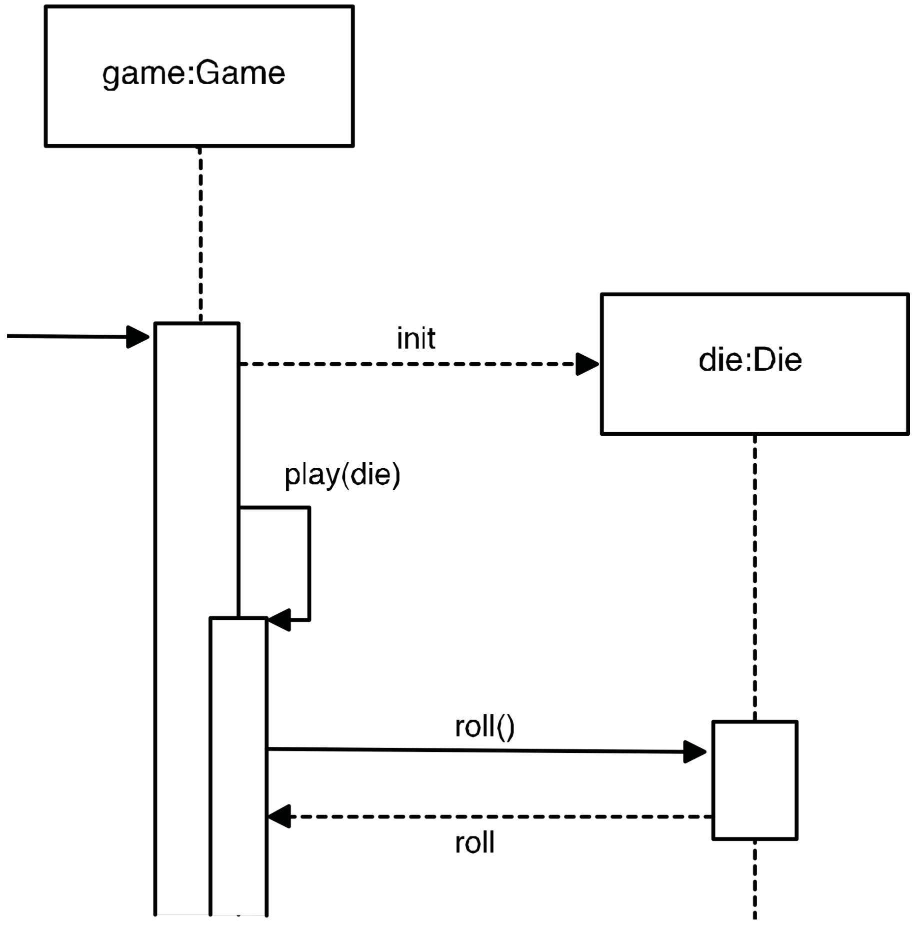
UML – Sequence Diagramm



UML – Sequence Diagramm



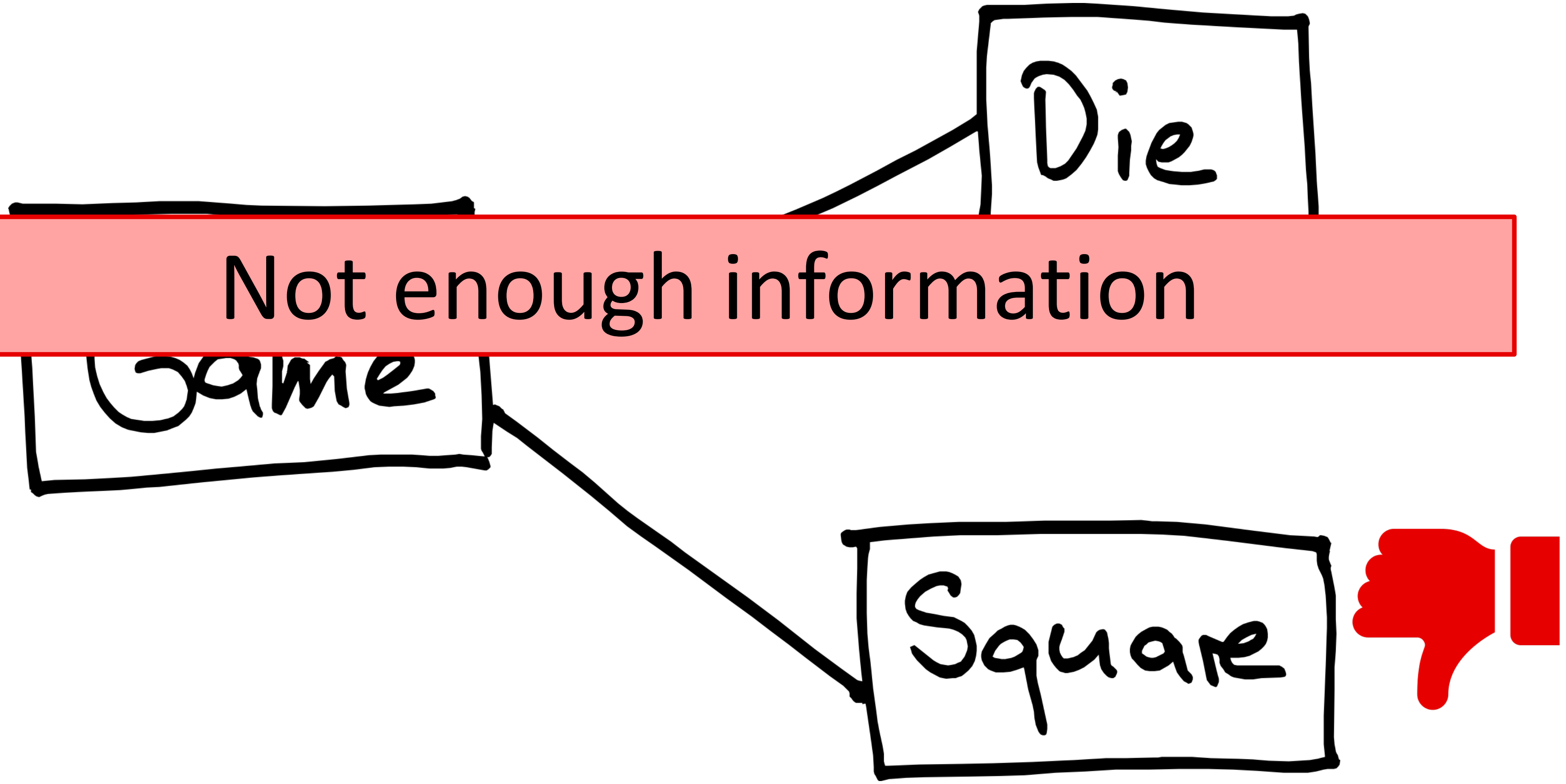
UML – Sequence Diagramm



UML - Tips

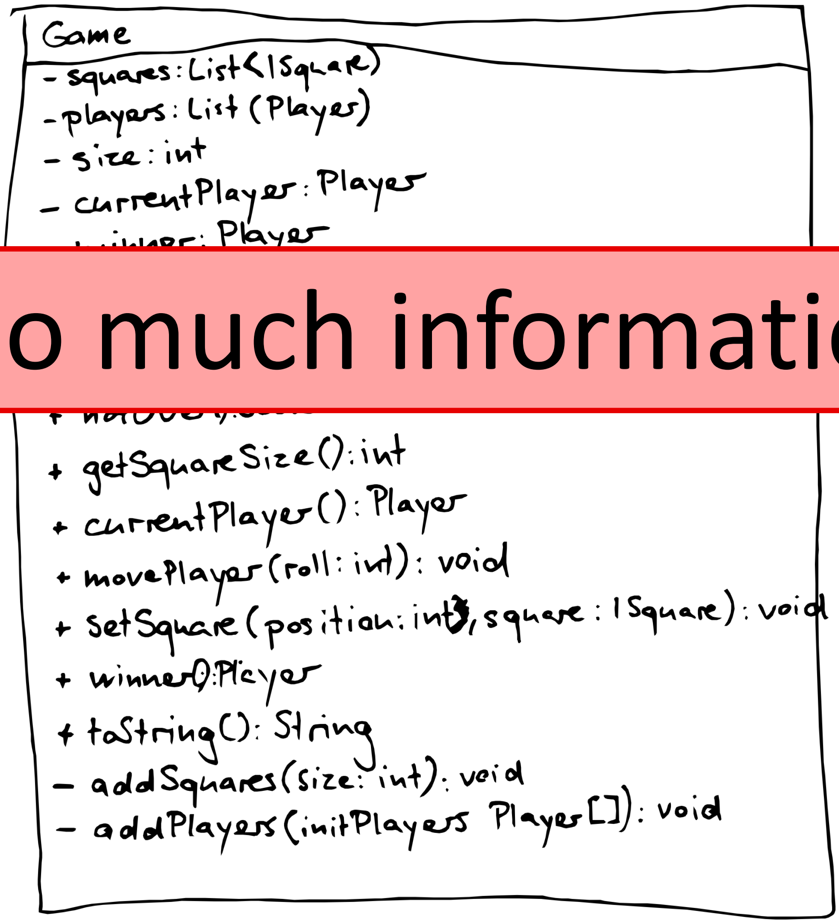
- Different aspects, different diagram type
- Keep it simple
- Focus on what you want to communicate, forget the rest

UML - Tips



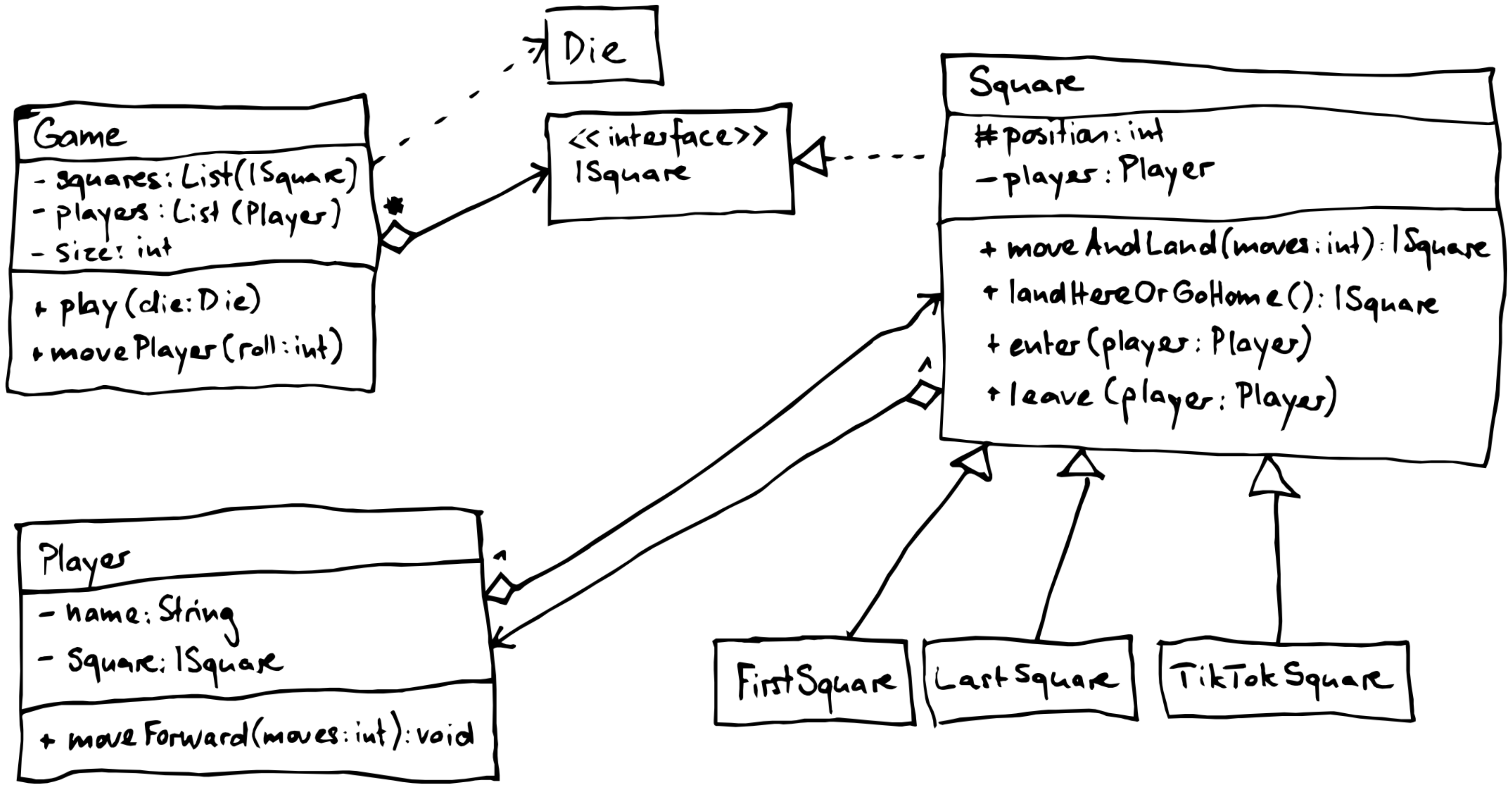
Not enough information

UML - Tips


Game
- squares: List<ISquare)
- players: List (Player)
- size: int
- currentPlayer: Player
winner: Player
+ getSquareSize(): int
+ currentPlayer(): Player
+ movePlayer(roll: int): void
+ setSquare(position: int, square: ISquare): void
+ winner(): Player
+ toString(): String
- addSquares(size: int): void
- addPlayers(initPlayers Player[]): void

Too much information

UML - Tips



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

Additional Material

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

Exercise 3 - Demo

- A hooman that moves around a 48x48 board
  - Commands : `right, left, up, down`
  - Leaves a trail
- Input: String representing a hooman program, which denotes where he should walk.
- Example:

```
right 5
down 4
left 3
up 10
```

Exercise 3 - Tips

- You start with
  - CovidRenderer: Handles GUI
  - Enviroment: Skeleton class that should handle the whole area
- Y

- **git pull p2-exercises master**
- Read exercise_03.md
- Happy Coding!

- Use the information from the lecture and form these slides to make the two UML diagrams
- Scan the UML or take a picture and add them both to your repository as a .png or .jpg