Aalto University
School of Science

Program design and UML

CS-C2120, Programming studio 2
CS-C2105, Programming studio A

20.1.2021

News

- Chapter 15 opens today at 14.
- A+ course page includes a link to Code vault
- Zoom exercise sessions begin on Friday 22nd
- Teaching assistants follow and respond to questions in Zulip
- UML task submission DL Wed 27th at noon.
- UML task grading will take 1-2 weeks after submission DL
- Demo session on Friday 22nd includes more design examples and aspects

OO Design

- Object-oriented (OO) analysis and design can be described as
  - Identifying the objects of a system.
  - Identifying their relationships.
  - Making a design, which can be converted to executables using OO languages.

OO analysis: Identifying objects

- During OO analysis, the most important purpose is to identify objects and describe them in a proper way.
- The objects should be identified with *responsibilities*, that is, the functions performed by the object.
  - Every object has some type of responsibilities to be performed.

OO Design – identifying relations

- Here emphasis is placed on the requirements and their fulfilment.
- Objects should *collaborate* according to their intended association.
  - Objects collaborate with other objects to carry out their responsibilities. We need to identify these associations.
- After the association is complete, the design is also complete.

CRC cards / Responsibility-Driven Design

- Responsibility-Driven Design focuses on identifying class responsibilities
  - Which functions a class should implement self and which ones need collaboration with other classes?
- CRC (Class-Responsibility-Collaborators)
  - CRC cards provide a method to support and document OO analysis and design.
  - Worth trying out.

CRC card

Class title

Responsibilities	Collaborators
What the class should do?	Which other classes are involved?

Back to Dungeon game

- Let us continue to design our Dungeon game.

Scenario: maze

Places	Scenario 2				
			Classes		
Dungeon consists of Levels					
			Dungeon	Level	Location
Levels consist of an 2D array of Locations					Floor
					Trapdoor
Caves and Corridors are not classes but parts of a level map					Wall
					Door / Hidden door
Locations can be Floors, Walls, Trapdoors, Stairs, Entrance, Doors					Stairs / Entrance
Areas between caves can be Walls					
Doors can be Hidden doors					
Entrance can be Stairs					

CRC card

DungeonGame

Responsibilities	Collaborators
Create the world	Level
Create the player	Player
Advance the game	
Game end	

What are responsibilities of Level?

- Zoom-poll

CRC card

Level

Responsibilities	Collaborators
Create caves, corridors and stairs for level	Location Grid
Knows the maze structure	
Creates initial Monsters in maze	Monster
Maintains monster status in the level	
Creates initial Items in maze	Item

CRC card

Location

Responsibilities	Collaborators
Knows the type of location	Monster
Knows what the location containts	Item
Knows its coordinates in Grid	Coordinates
Knows properties (lighting, mapping status)	

Creature classes

Creatures				
		Classes		
Creatures can be the Player or Monsters				
		CreatureType		
Monsters can be Floating eyes, dragons, and many others		***Player***	***Monster***	Properties
			Floating eye	life point
Creatures have Properties			Umberhulk	experience level
			Dragon	skill
			...	

CRC card

Player

Responsibilities	Collaborators
Knows current location	
Knows carried Items Can manage Items Knows Items in use Knows own properties (life points, symbol…) Can move and attack Can develop one's properties Can die	

With which classes Player should collaborate?

- Zoom-poll

CRC card

Player

Responsibilities	Collaborators
Knows current location	Level Location CompassDir
Knows carried Items	Item
Can manage Items	
Knows Items in use	
Knows own properties (life points, symbol…)	
Can move and attack	Monster
Can develop one’s properties	
Can die	

CRC card

Monster

Responsibilities	Collaborators
Knows current location	Level Location CompassDir MonsterType
Knows own properties (life points, symbol, …)	
Knows own MonsterType	
Can define whereTo move	
Can move and attack	Me
Can develop	
Can die	

CRC card

Weapon

Responsibilities	Collaborators
Knows WeaponType	WeaponType
Knows own properties (spell, curse, symbol, …)	

CRC card

Ring

Responsibilities	Collaborators
Knows RingType	RingType
Knows own properties (spell, curse, symbol, …)	

Testing design

- CRC cards could be tested with the help of *User stories,* which are very brief informal descriptions of relevant actions in the application.

User stories, examples

- I want to proceed through this level
- I want to proceed stairs down to the next level
- I want to pick up this item
- I want to attack this monster
- I want to use this thing
- Monster wants to find you
- Monster wants to attack you

User stories, test

- I want to proceed through this level

CRC card and Zoom poll

Level

Responsibilities	Collaborators
Create caves, corridors and stairs for level	Location Grid
Knows the maze structure	
Creates initial Monsters in maze	Monster
Maintains monster status in the level	
Creates initial Items in maze	Item

User stories, examples

- I want to proceed through this level (Done)
- I want to proceed stairs down to the next level
- I want to pick up this item
- I want to attack this monster
- I want to use this thing
- Monster wants to find you
- Monster wants to attack you

Implementation

- Design is implemented using OO languages such as Java, Scala, C++, etc.
- But this is not straightforward
  - Many details need to be added
  - Choice of data structures and algorithms
  - Top-down vs. Bottom up vs. Both
  - Iteration and refinement of design is often needed => *Code restructuring*
    - Model project resource includes several examples of this.

Break, 15 minutes

- We continue soon, 15.15.

UML, Unified modeling language

- Graphical description method for software design
- Allows to abstract details away and focus on key concepts, components, their relations and processes.
- Supports structural, behavioral and architectural modeling.

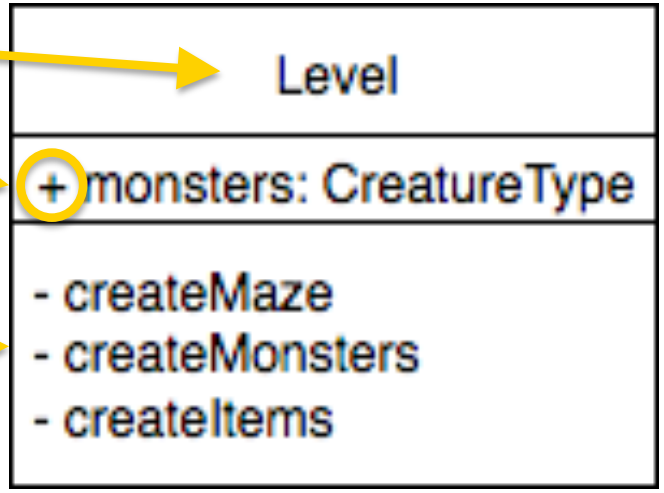
UML, Unified modeling language

- Graphical description method for software design
- Allows to abstract details away and focus on key concepts, components, their relations and processes.
- Supports structural, behavioral and architectural modeling

We focus on this only

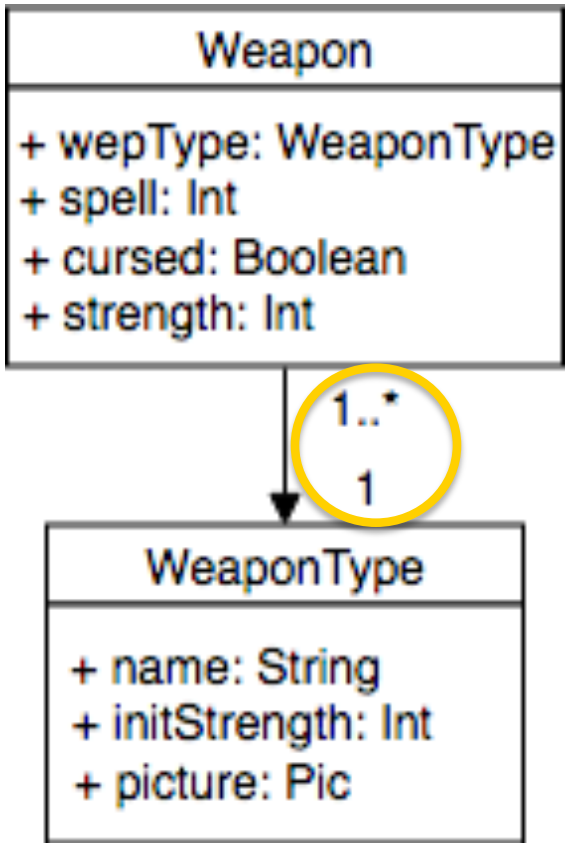
UML Class diagram

- Presents a class
  - Class name
  - Instance variables
    - Visibility
  - Methods
  - Possible attribute of class type (trait, abstract class)



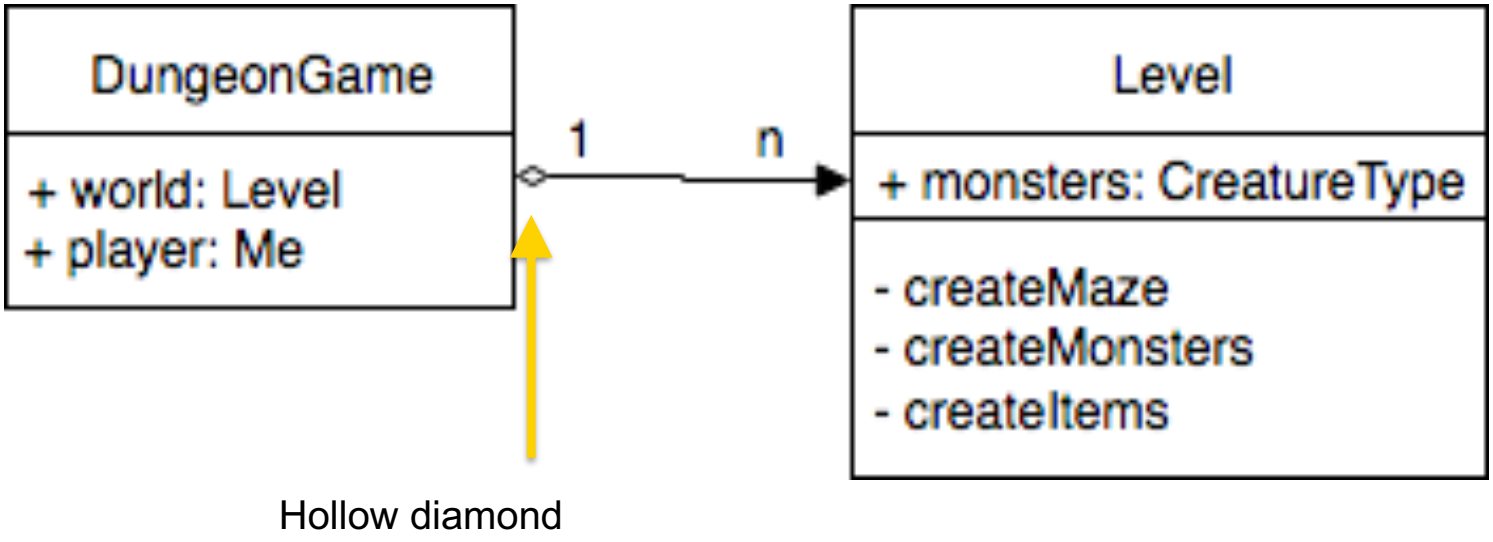
Relations: Association

- Association
  - Each Weapon is associated with one WeaponType
  - WeaponType can be associated with many Weapons



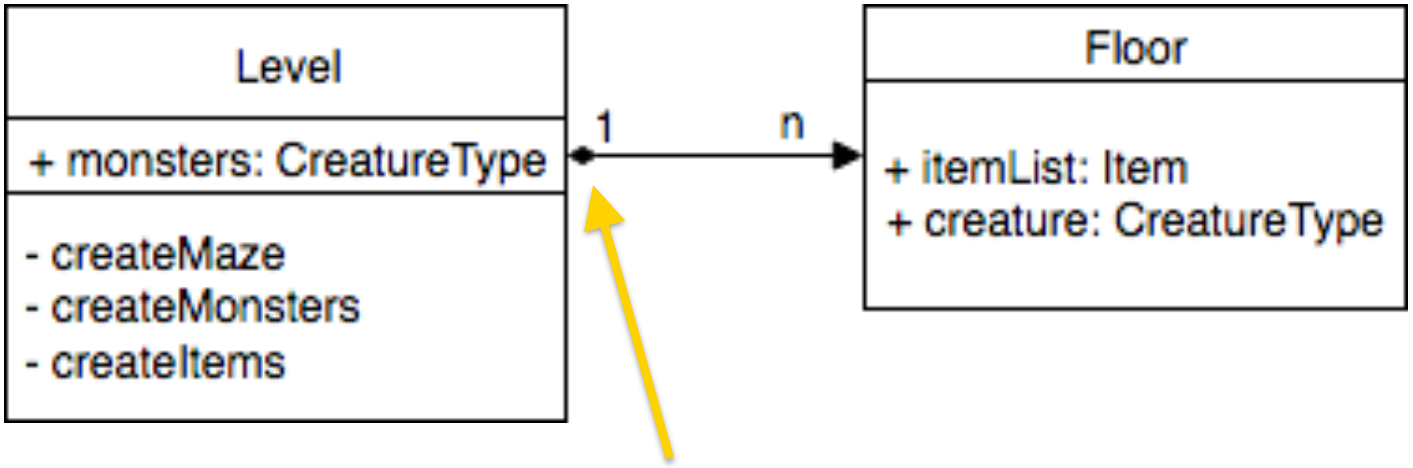
Relations: Aggregation

- DungeonGame has many Levels, which can exist independently



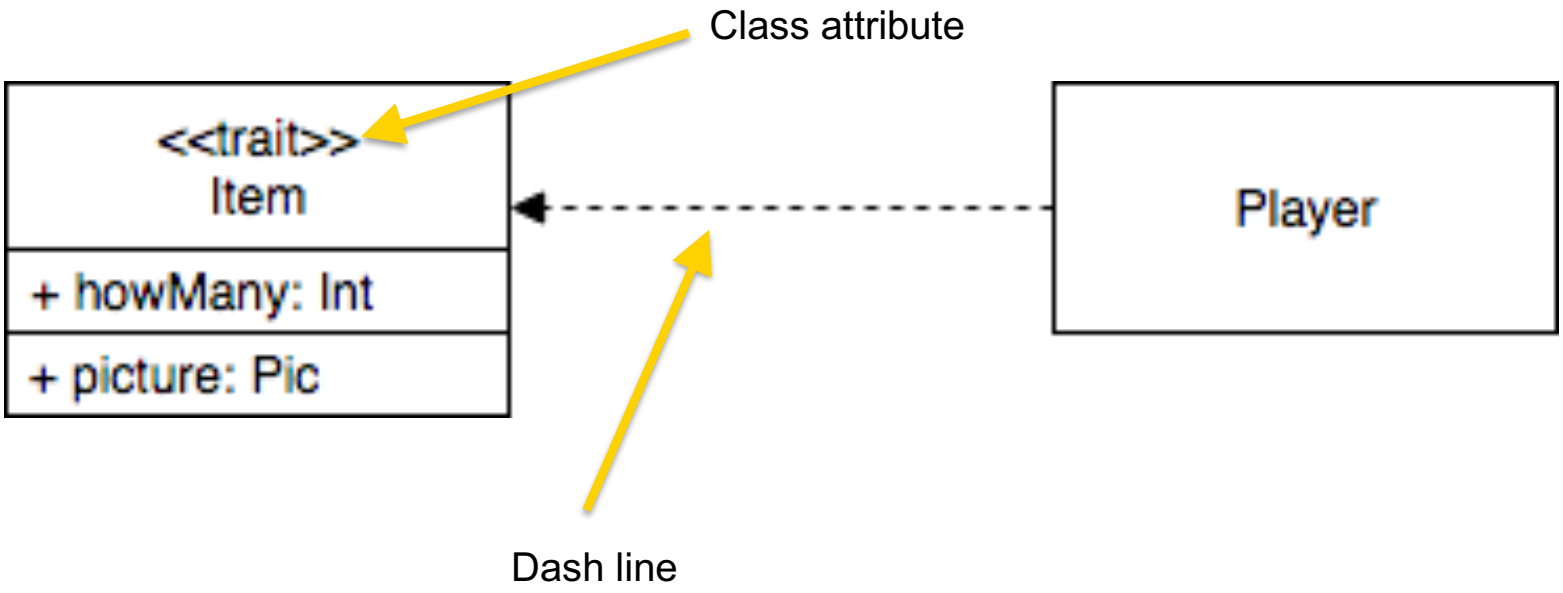
Relations: Composition

- Levels consist of Floor locations which cease to exist if Level is destroyed



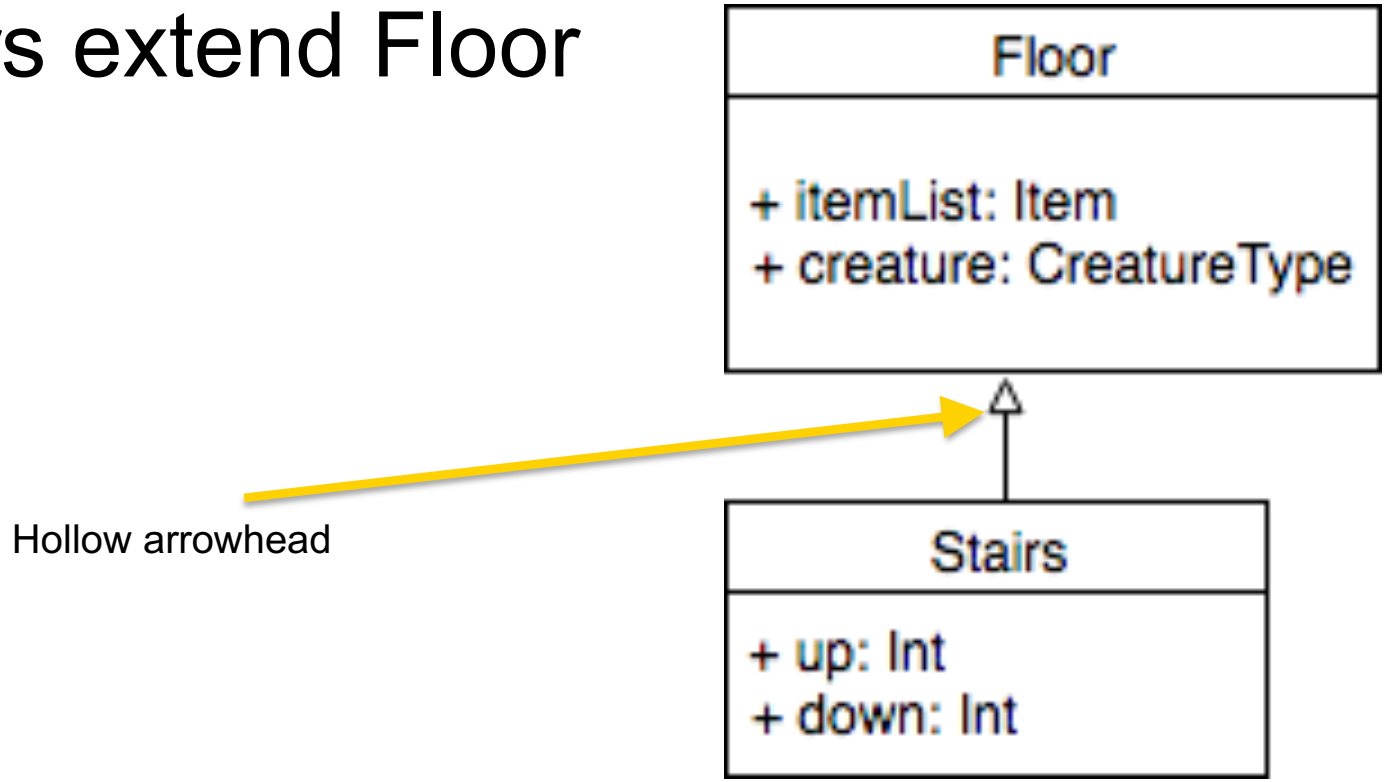
Relations: Dependency

- Player's functions depend on what kind of Items there are in the game.



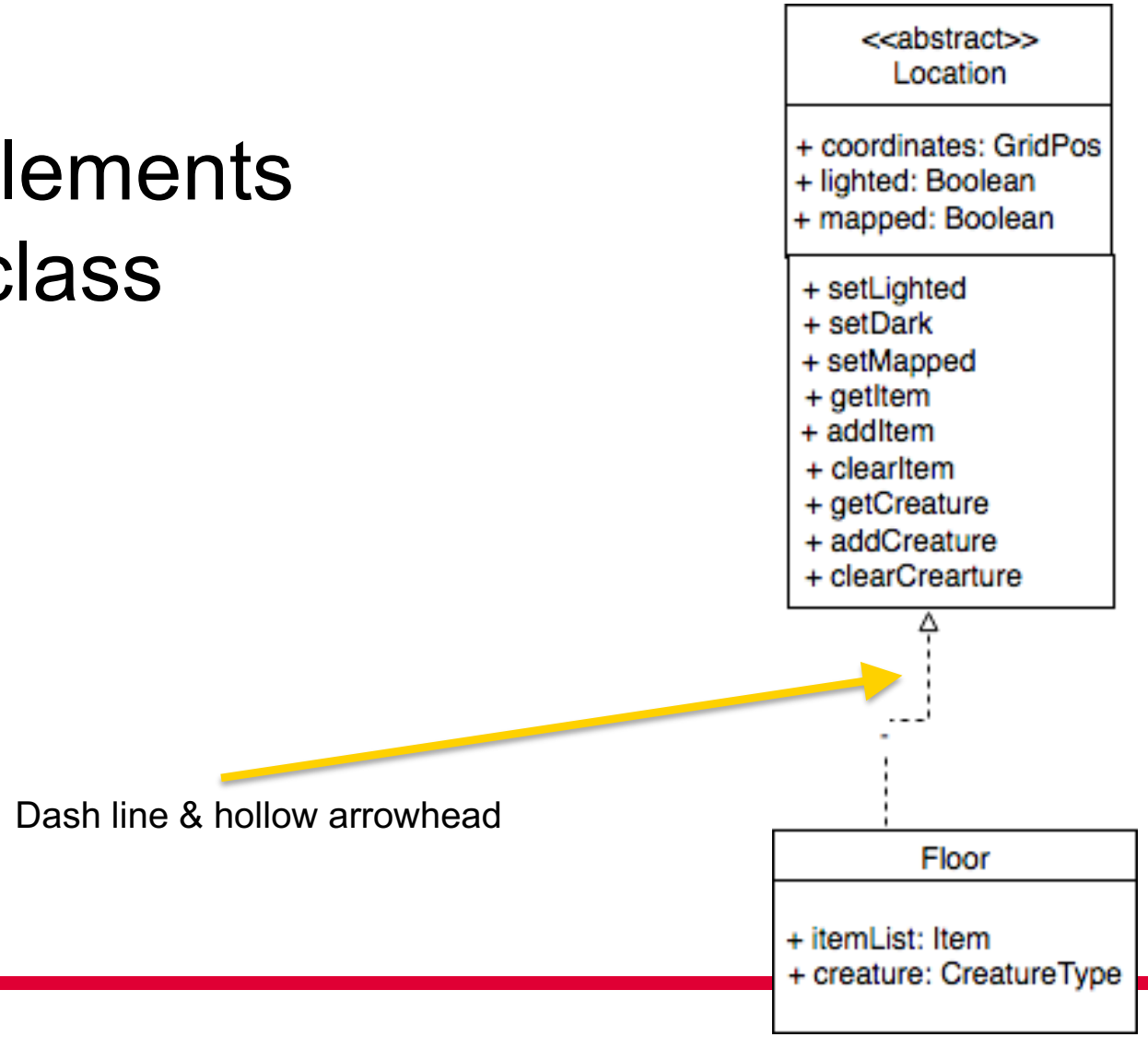
Relations: Inheritance

- Stairs extend Floor

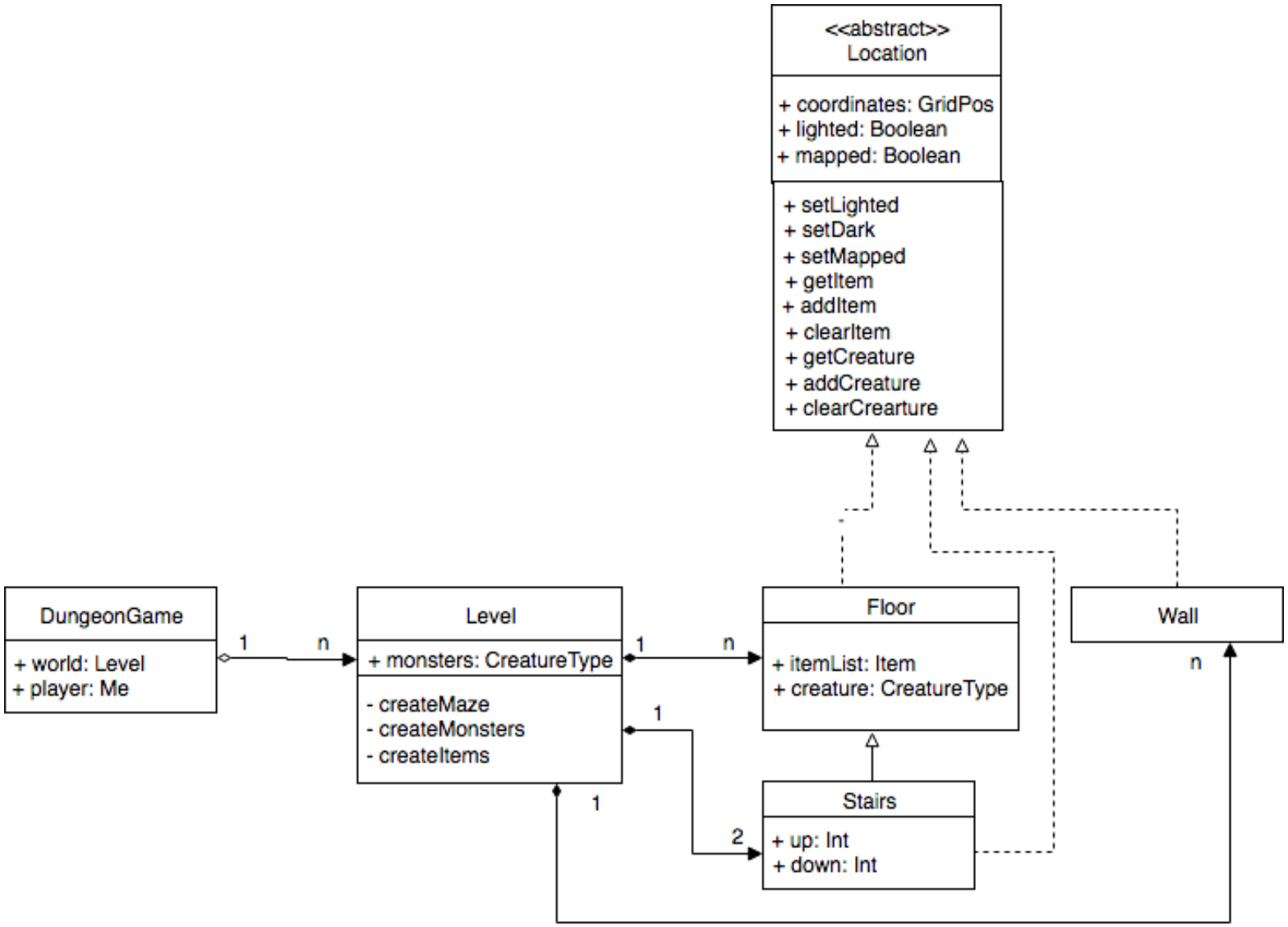


Relations: Implements

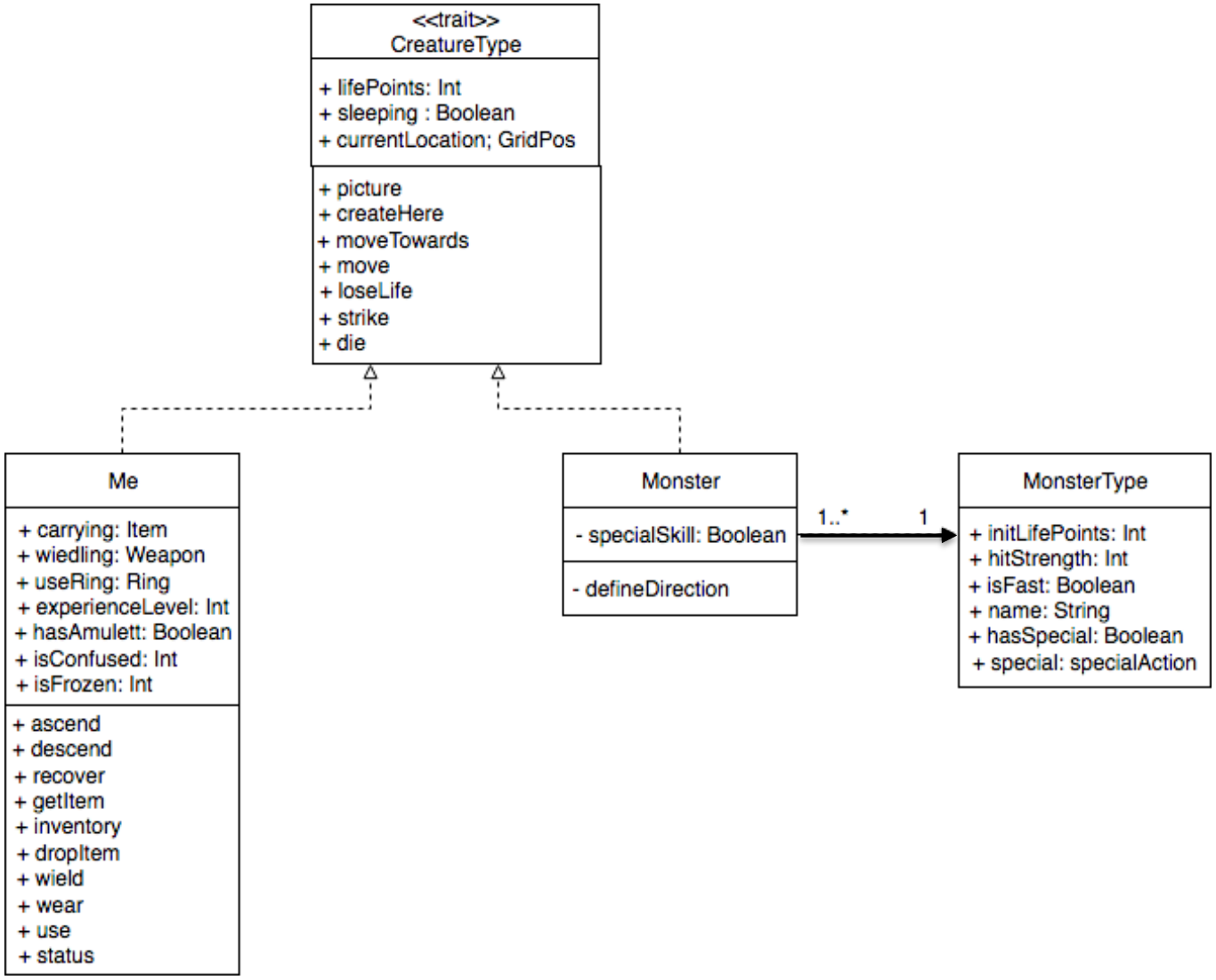
- Floor implements abstract class Location



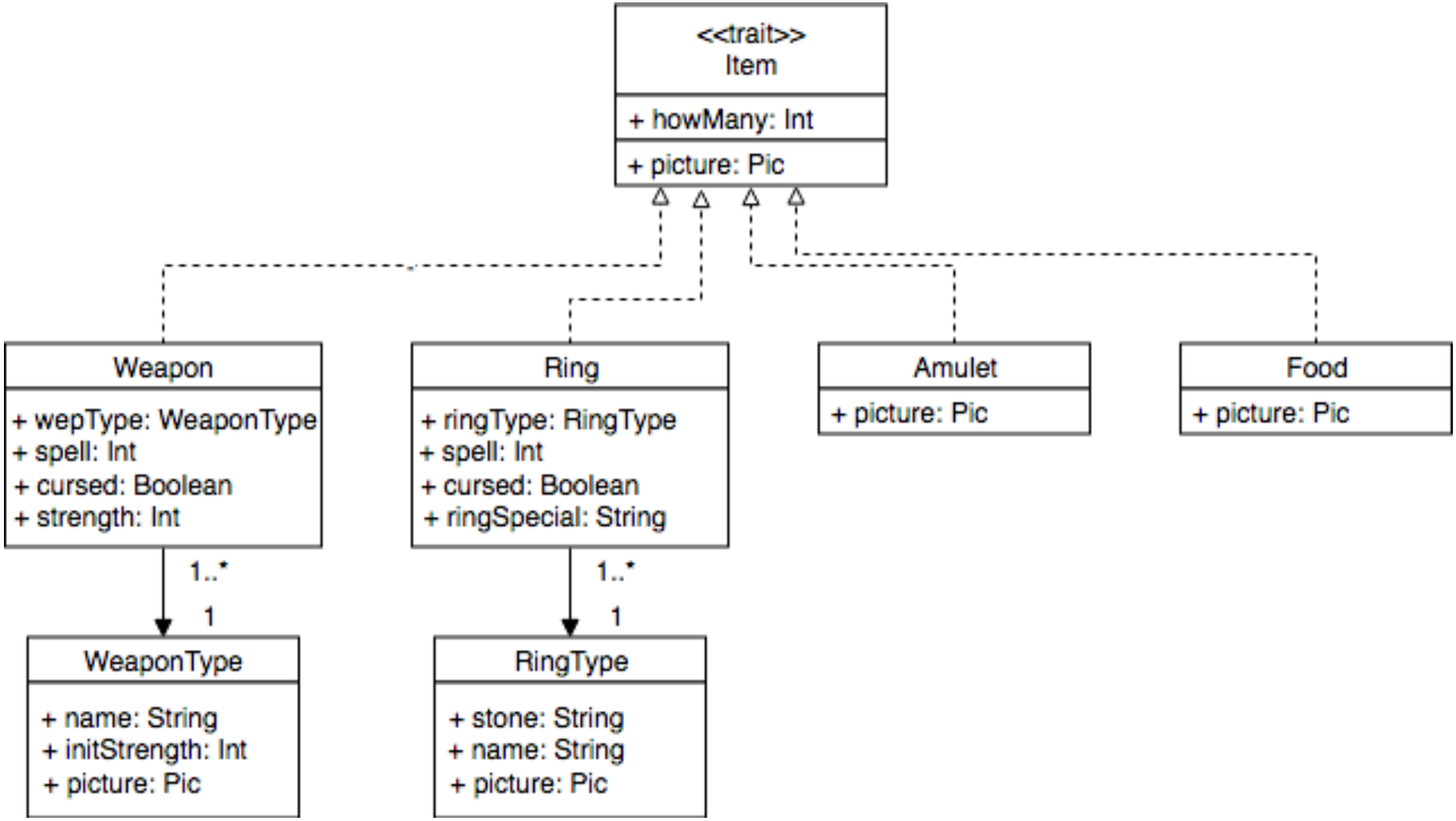
Example: Dungeon



Example: Creatures



Example: Items



Critical questions

- Are all relations of classes visible?
- Are variables and methods in appropriate classes, especially in the case of superclass/subclass hierarchies?
- Has visibility of variables and methods been considered?
- Can user stories be implemented in this structure?

Quality aspects

- Cohesion
  - Does a class implement many different things or does it focus on presenting and manipulating one concept/thing?
  - Might there be something, which could be better implemented in another class or a new dedicated class?

Quality aspects cont.

- Coupling
  - How complex is the interface between two classes which use methods / variables?
  - Does a class need information of the internals of another class?
  - Does its own implementation depend on such information?
    - For example, is it relevant to know the data structures used in another class?
    - => If yes, there is a risk of cumulative needs for changes

Friday demo & next week

- More discussion on the example design
- Other examples of design