Aalto University
School of Science

Program design and UML

CS-C2120, Programming studio 2

CS-C2105, Programming studio A

20.1.2021

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.

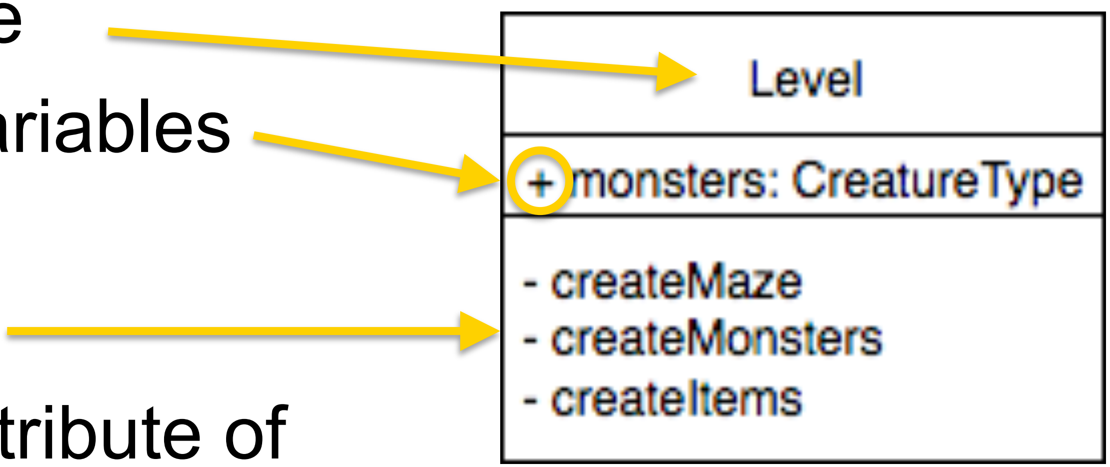
UML, Unified modeling language

- 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

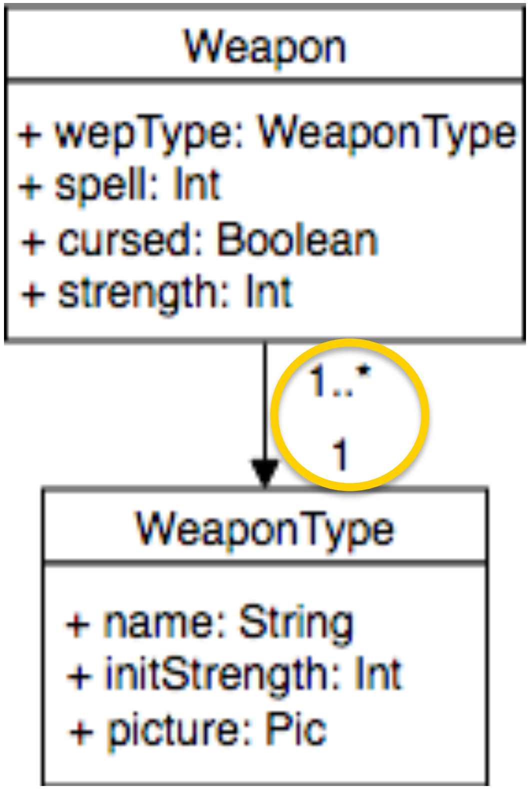
UML Class diagram

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



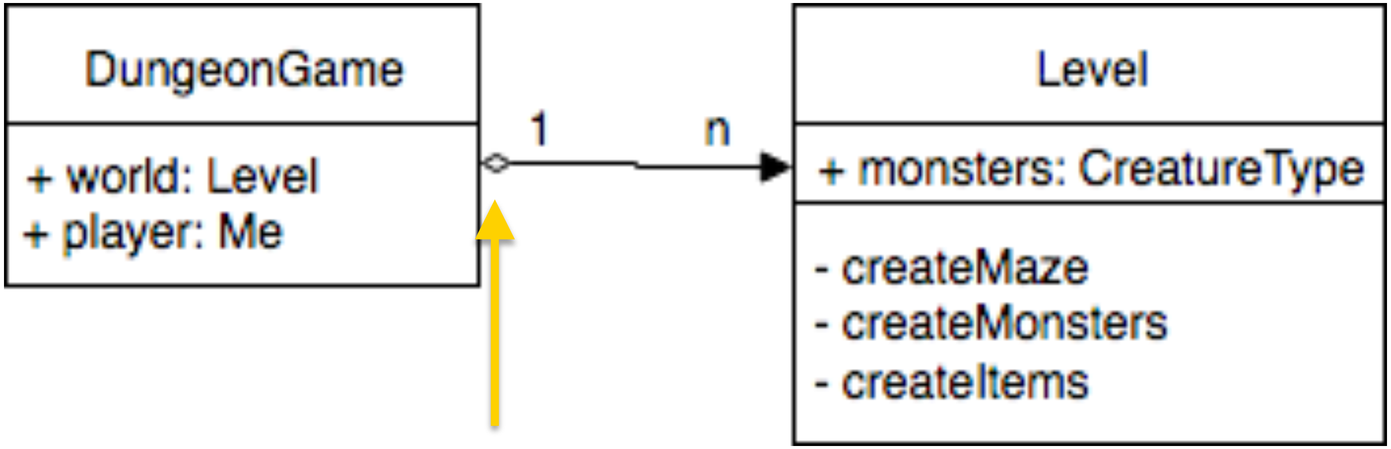
Relations: Association

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



Relations: Aggregation

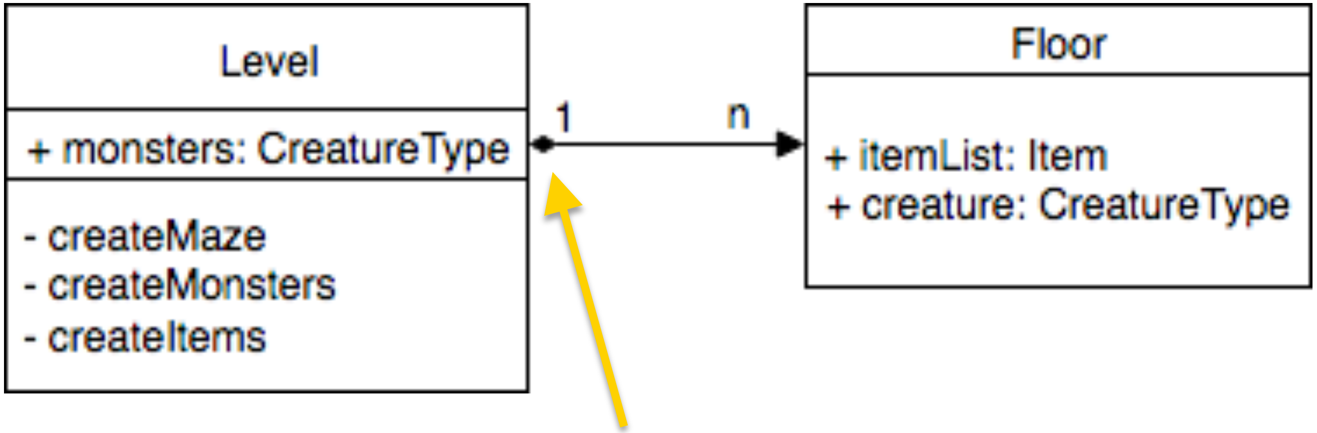
- DungeonGame has many Levels, which can exist independently



Hollow diamond

Relations: Composition

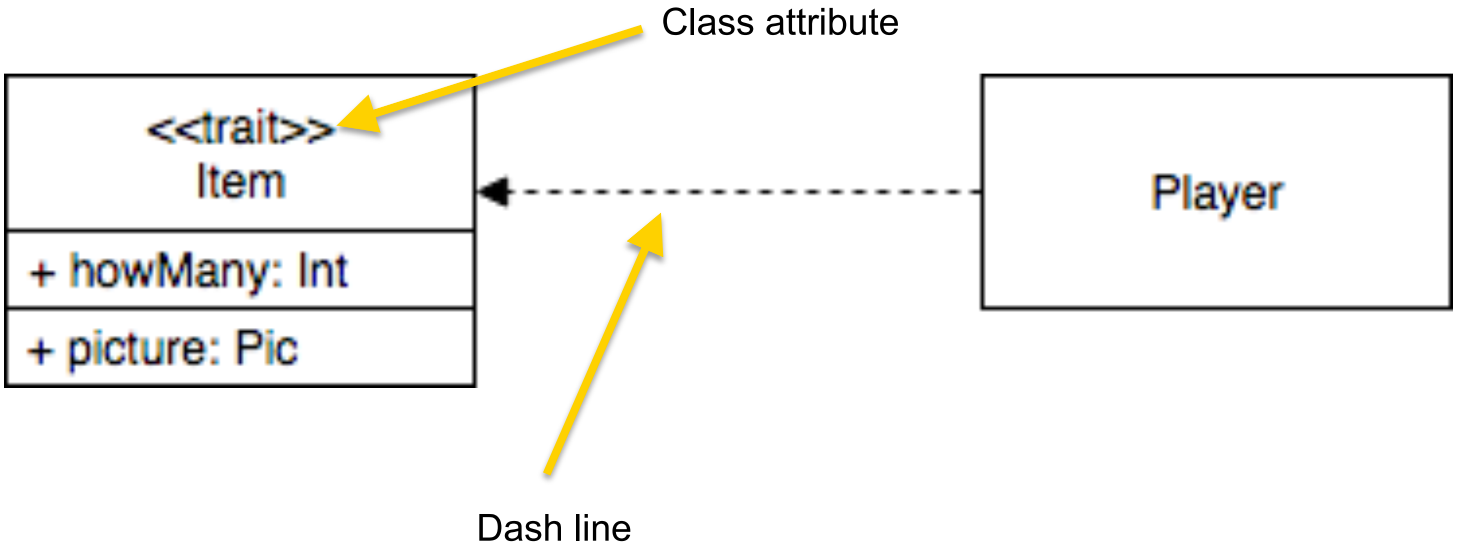
- Levels consist of Floor locations which cease to exist if Level is destroyed



Filled diamond

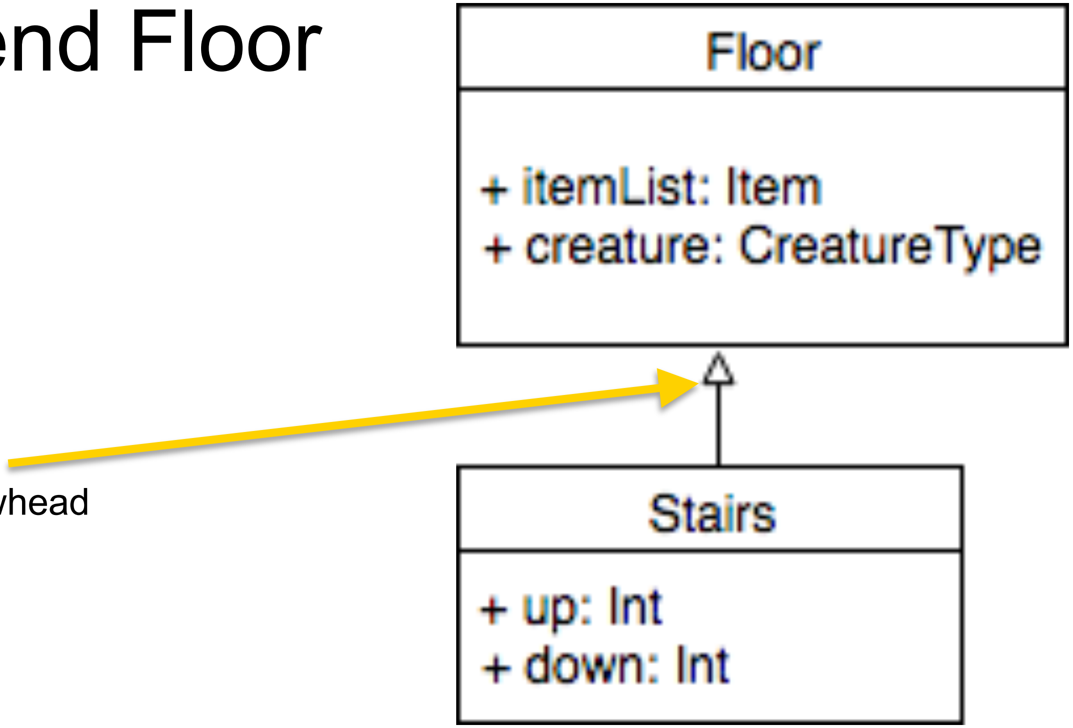
Relations: Dependency

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



Relations: Inheritance

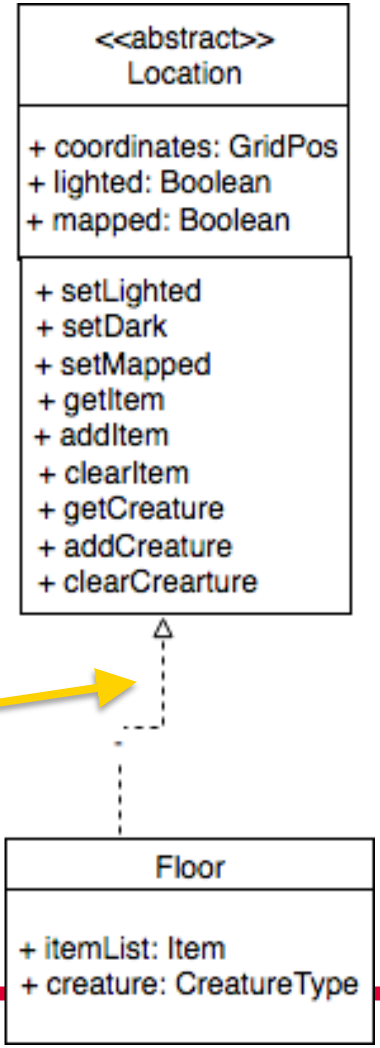
- Stairs extend Floor



Hollow arrowhead

Relations: Implements

- Floor implements abstract class Location



<<abstract>>
Location

+ coordinates: GridPos
+ lighted: Boolean
+ mapped: Boolean

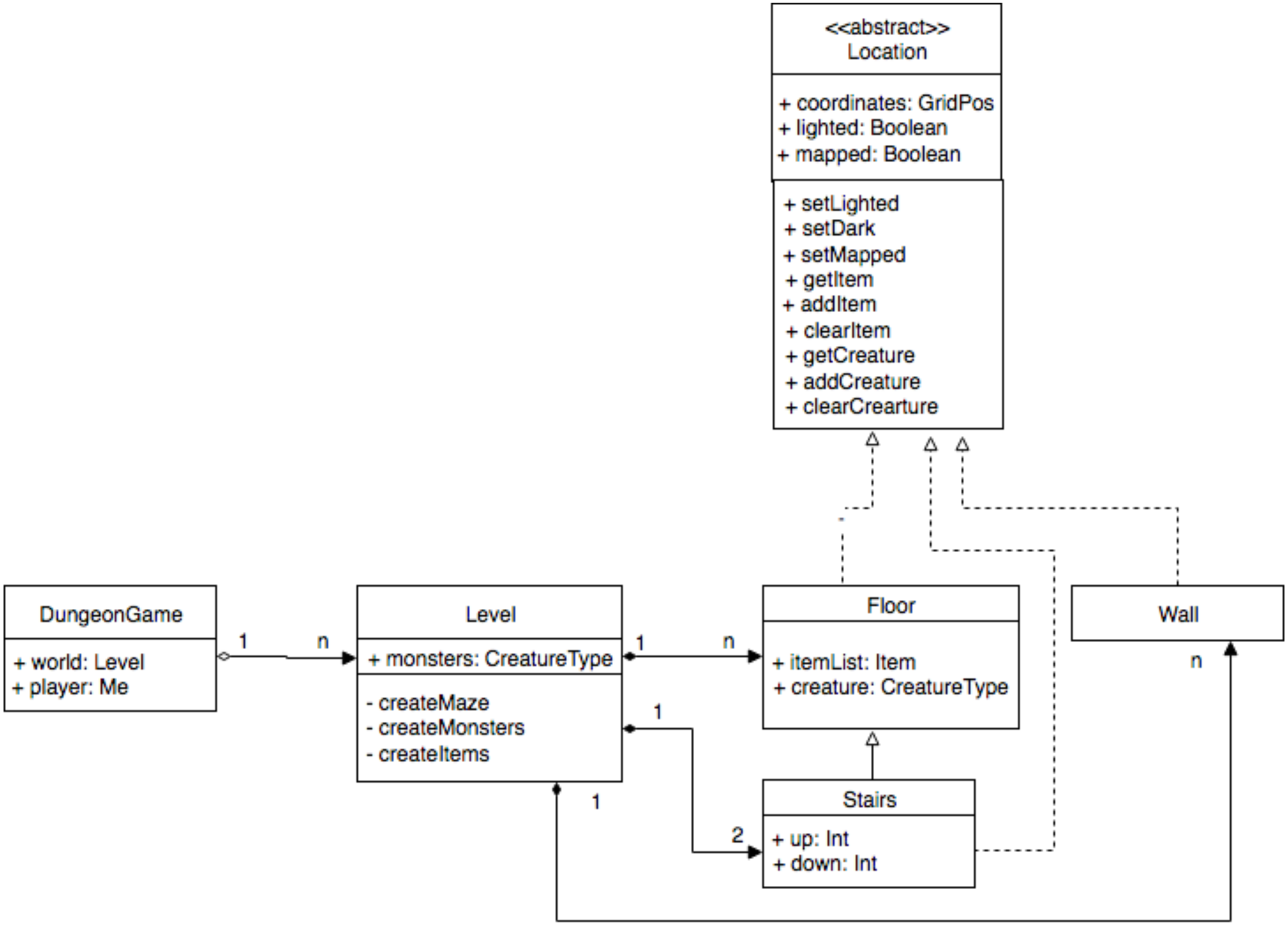
+ setLighted
+ setDark
+ setMapped
+ getItem
+ addItem
+ clearItem
+ getCreature
+ addCreature
+ clearCrearture

Floor

+ itemList: Item
+ creature: CreatureType

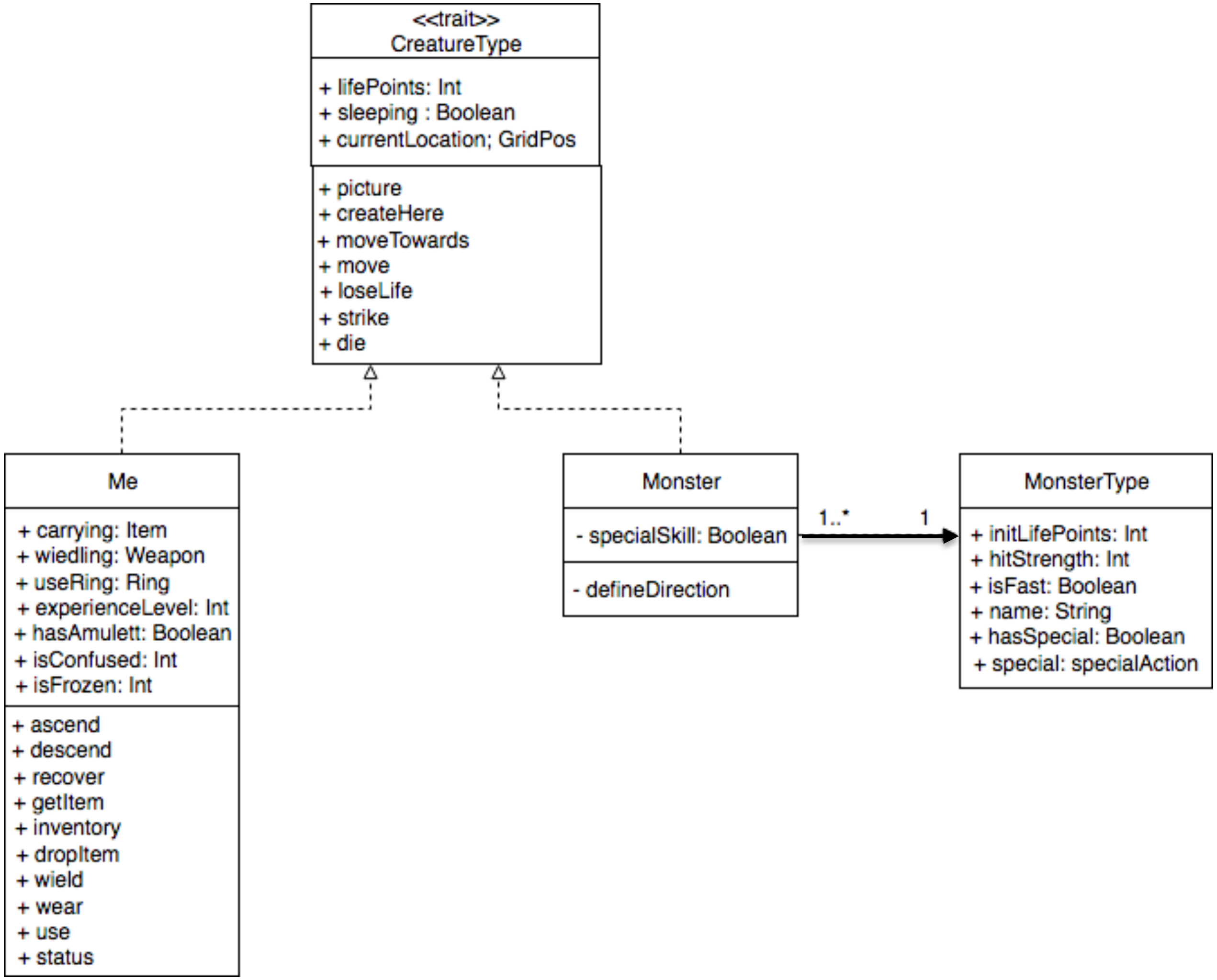
Dash line & hollow arrowhead

Example: Dungeon



School of Science

Example: Creatures



<<trait>>
CreatureType

+ lifePoints: Int
+ sleeping : Boolean
+ currentLocation; GridPos

+ picture
+ createHere
+ moveTowards
+ move
+ loseLife
+ strike
+ die

Me

+ carrying: Item
+ wiedling: Weapon
+ useRing: Ring
+ experienceLevel: Int
+ hasAmulett: Boolean
+ isConfused: Int
+ isFrozen: Int

+ ascend
+ descend
+ recover
+ getItem
+ inventory
+ dropItem
+ wield
+ wear
+ use
+ status

Monster

- specialSkill: Boolean

- defineDirection

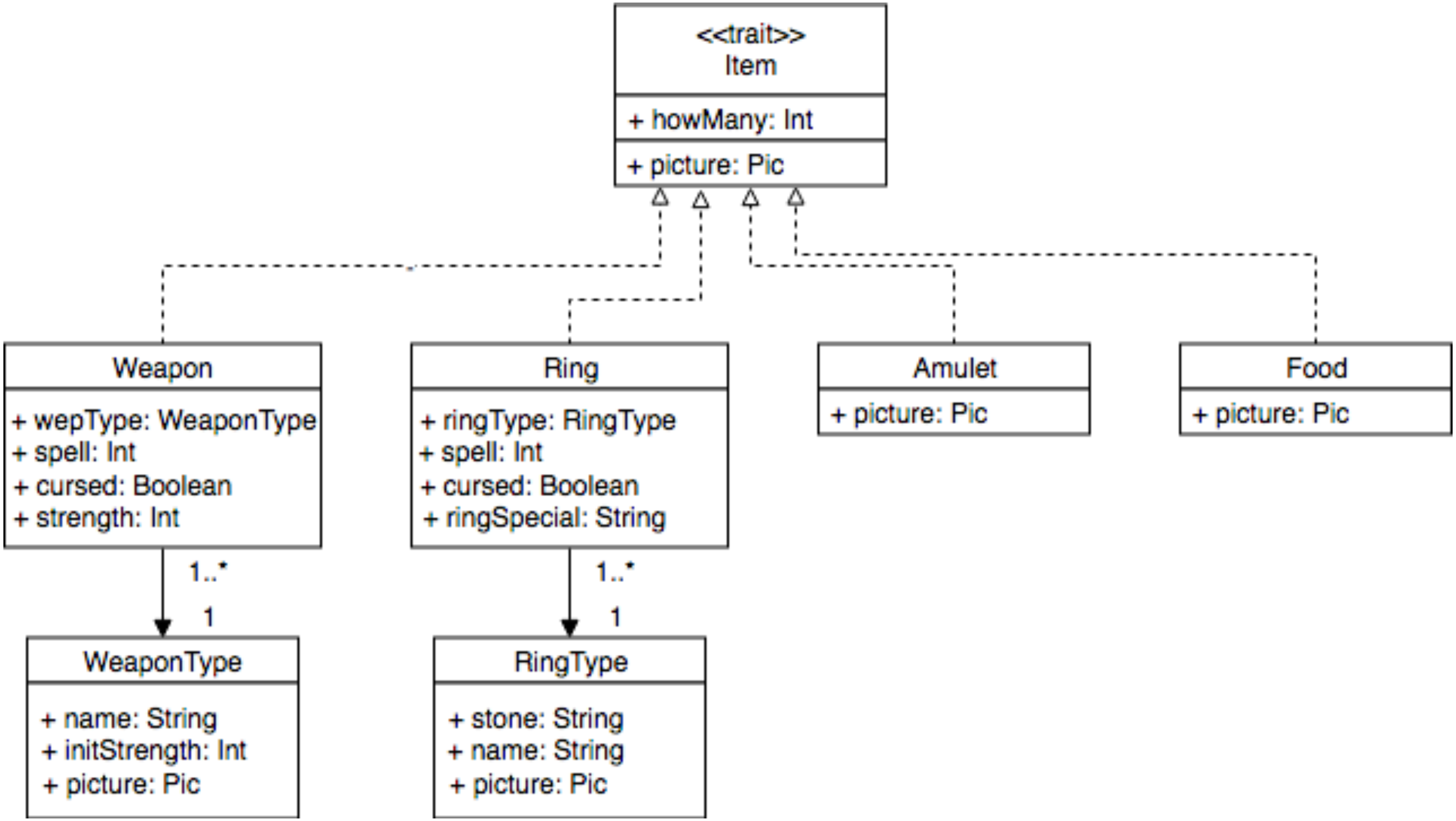
1..*

1

MonsterType

+ initLifePoints: Int
+ hitStrength: Int
+ isFast: Boolean
+ name: String
+ hasSpecial: Boolean
+ special: specialAction

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