Aalto University
School of Science

From Design to Implementation

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

26.1.2024

Program design cont.

- Let us revisit the route planner application.
- From initial design to UML
  - Now proceeding directly to UML.
  - Using Draw.io as a tool.

Lecture learning goals

- Understand, how draw.io could be used in UML design
- Understand, how to proceed from UML to implementation.
- Understand, how simplified classes can be used to help implementation.

Some important terms

- Abstraction
- Interface
- Modularization
- Information hiding
- Encapsulation
- Packages

From UML to Coding

- How to proceed?
- Package design
- Traits vs. classes?
- Data structures
- Dummies, Stubs, Mocks
- Implementing and testing

How to proceed?

- UML design can be turned into class definitions in a straightforward way.
  - Class names
  - Inheritance
  - Variables
  - Methods
  - Visibility
- You will probably add more variables and methods later, as well as revise method parameter definitions.

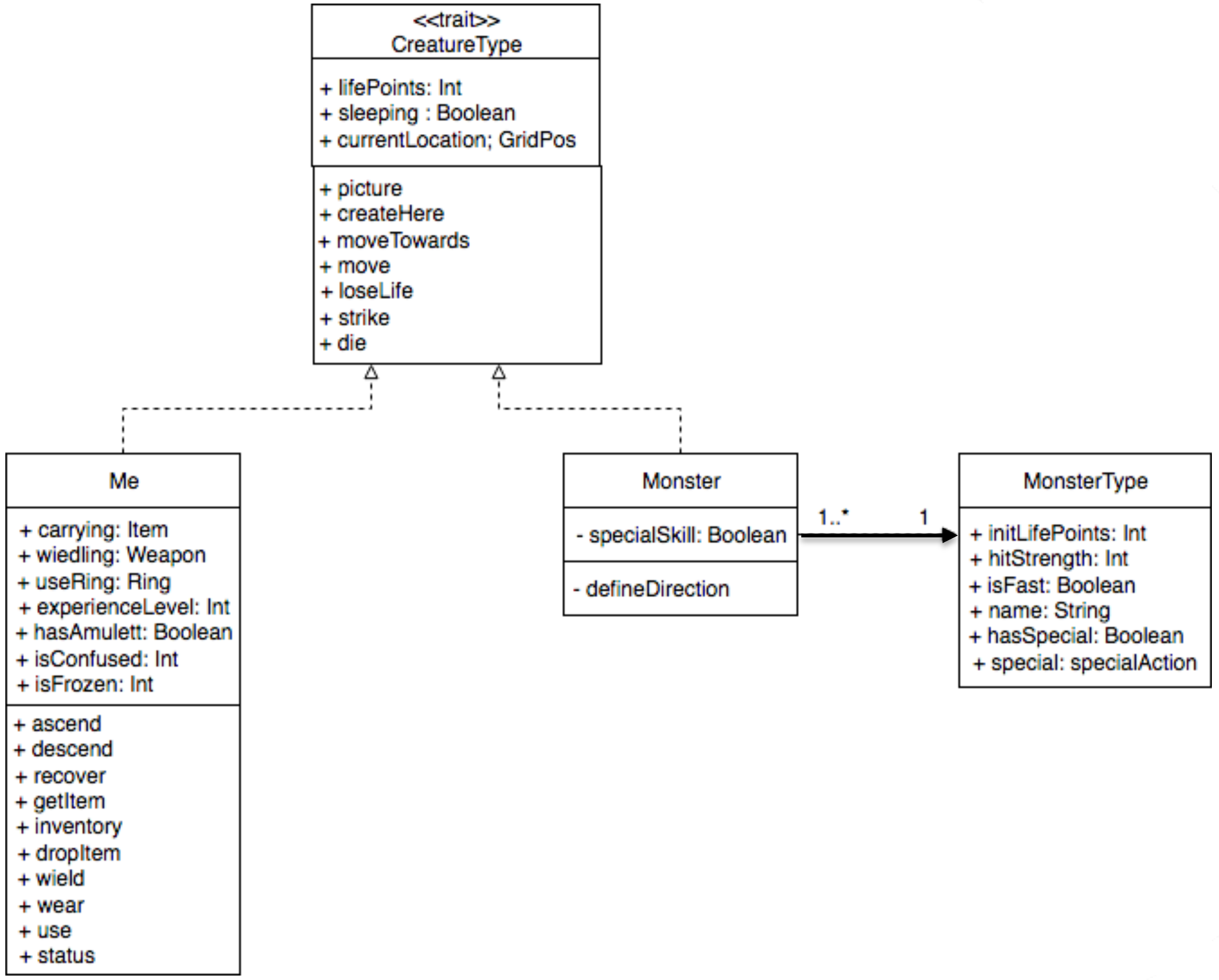
Package design

- When the number of classes grows, it is worthwhile to consider identifying major components in the program.
  - The program could be split into separate packages.
  - One way to implement this split is separating the following:
    - User interface operations (gui)
    - Program logic
    - File management
    - Code for testing your classes
  - Many exercise projects in O1 course have separated Gui and program logic. See examples there.

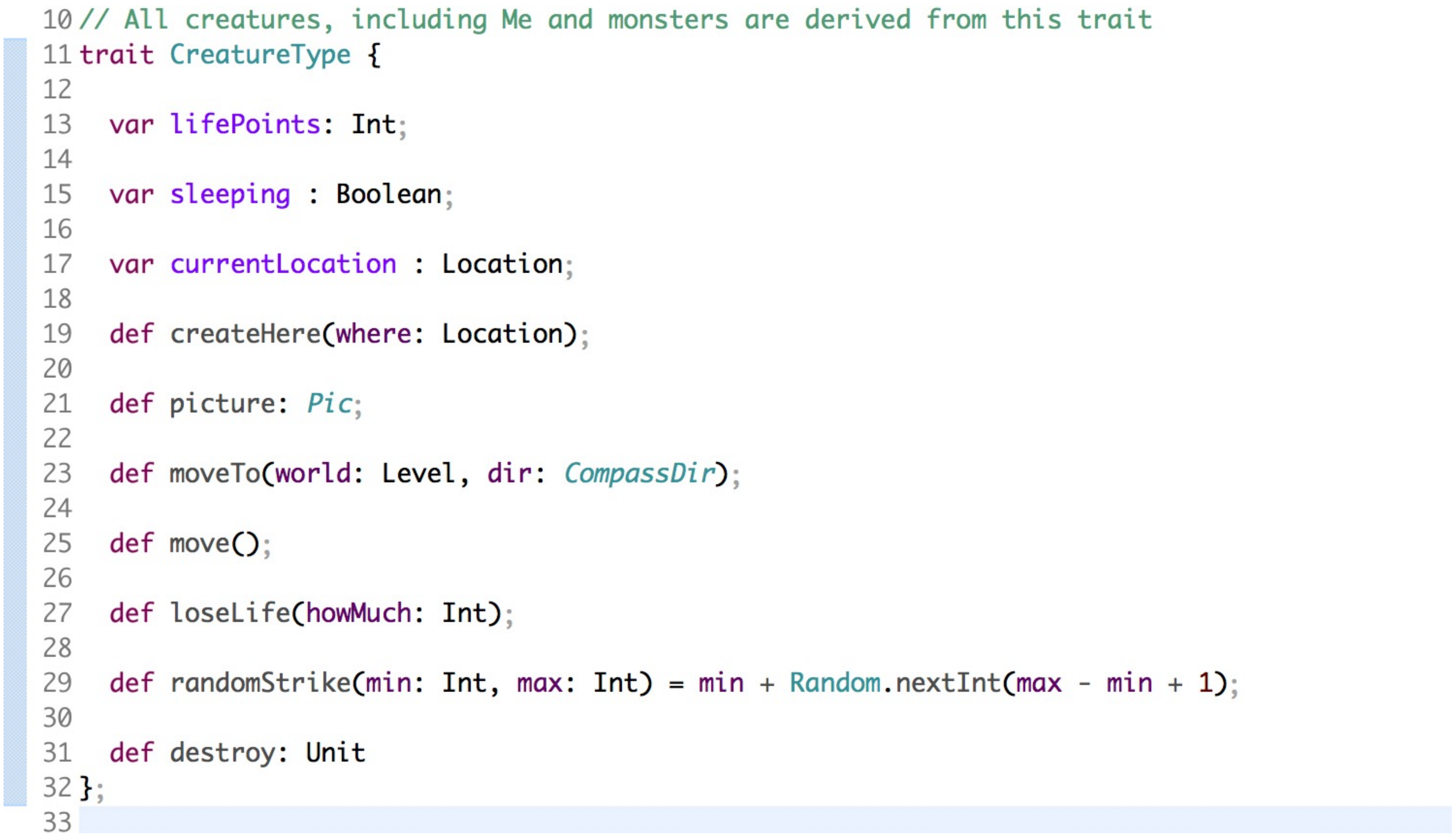
Traits vs. classes

- When would you use traits instead of ordinary classes?
- Recall
  - Traits cannot be instantiated.
  - Abstract classes not much needed in Scala 3, as traits can have parameters.
- Traits can be used to define abstract entities
  - A class which `extends` the trait has to implement the defined features.
- Classes can extend several traits at the same time.
  - This allows *adding new features (variables, methods) in classes* without using inheritance.

Example: Creatures

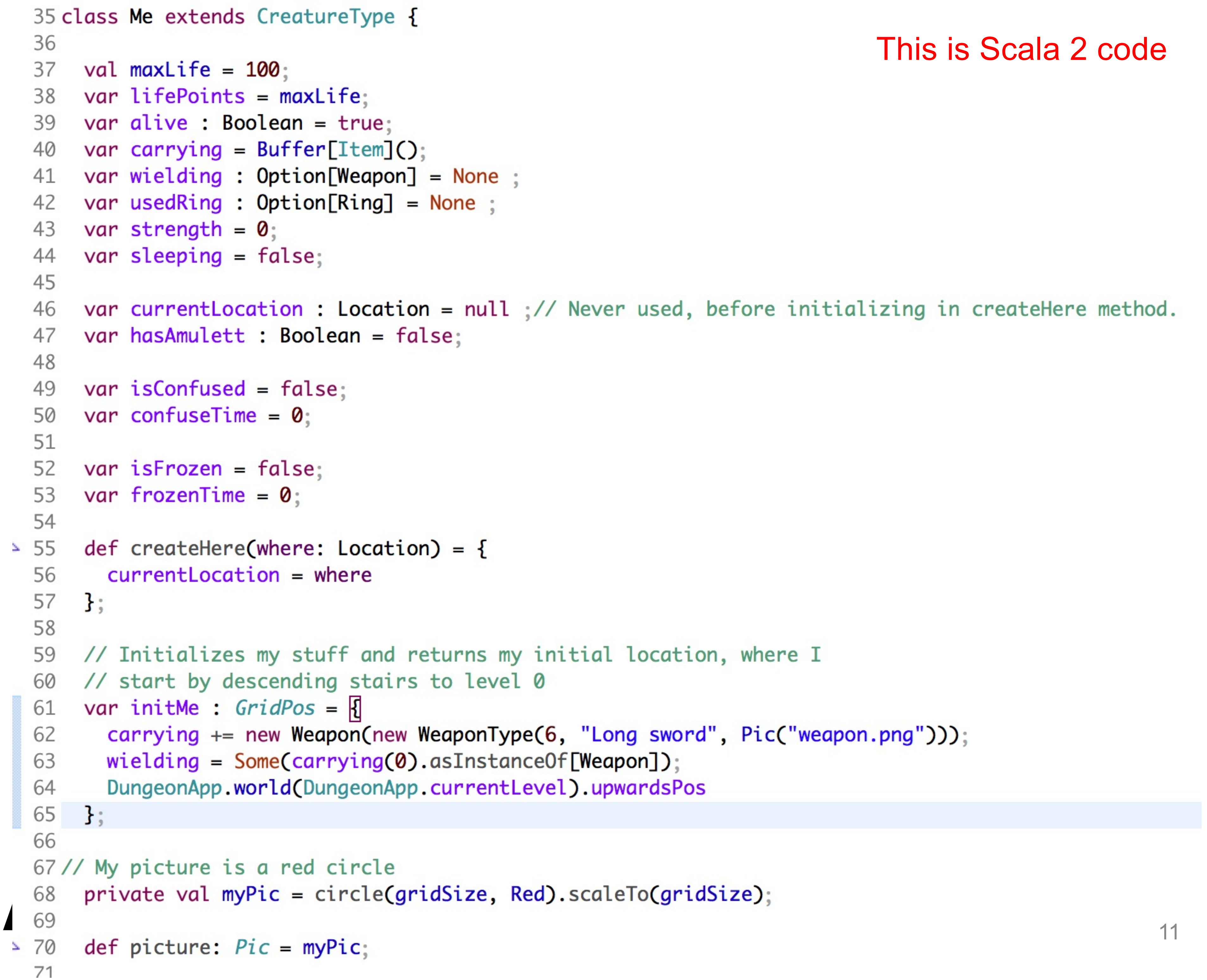


This is Scala 2 code



```scala
// All creatures, including Me and monsters are derived from this trait
trait CreatureType {

  var lifePoints: Int;

  var sleeping : Boolean;

  var currentLocation : Location;

  def createHere(where: Location);

  def picture: Pic;

  def moveTo(world: Level, dir: CompassDir);

  def move();

  def loseLife(howMuch: Int);

  def randomStrike(min: Int, max: Int) = min + Random.nextInt(max - min + 1);

  def destroy: Unit
};

```

This is Scala 2 code



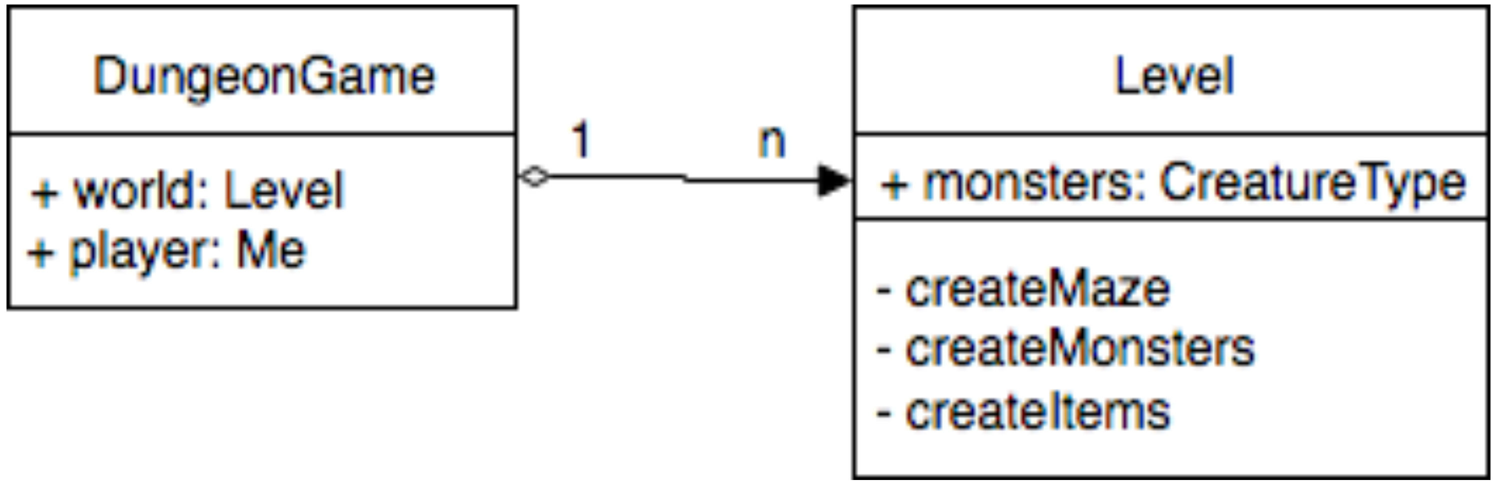
```
class Me extends CreatureType {

  val maxLife = 100;
  var lifePoints = maxLife;
  var alive : Boolean = true;
  var carrying = Buffer[Item]();
  var wielding : Option[Weapon] = None ;
  var usedRing : Option[Ring] = None ;
  var strength = 0;
  var sleeping = false;

  var currentLocation : Location = null ;// Never used, before initializing in createHere method.
  var hasAmulett : Boolean = false;

  var isConfused = false;
  var confuseTime = 0;

  var isFrozen = false;
  var frozenTime = 0;

  def createHere(where: Location) = {
    currentLocation = where
  };

  // Initializes my stuff and returns my initial location, where I
  // start by descending stairs to level 0
  var initMe : GridPos = {
    carrying += new Weapon(new WeaponType(6, "Long sword", Pic("weapon.png")));
    wielding = Some(carrying(0).asInstanceOf[Weapon]);
    DungeonApp.world(DungeonApp.currentLevel).upwardsPos
  };

// My picture is a red circle
  private val myPic = circle(gridSize, Red).scaleTo(gridSize);

  def picture: Pic = myPic;

```

About data structures

- Consider relevant questions
  - What kinds of data your program will manage?
  - What data is mutable, what is immutable?
  - How would you access data?
    - with indexes, sequentially, mapping, searching?
- Scala has quite extensive set of collections which help you in managing and storing data in your program.
  - They are highly useful.
  - See https://www.scala-lang.org/api/current/
  - You can learn more possible data structures on the course CS-A1140/1141.

Examples from DungeonGame

- DungeonGame has many Levels (fixed)



```
val world = Vector[Level]();
```

Examples from DungeonGame

- A level has a variable number of Items and Monsters.

```
val monsters = Buffer[CreatureType]();
```

- A location may have 0..* items

```
var itemList = Buffer[Item]();
```

- Player can carry 0..* items

```
var carrying = Buffer[Item]();
```

Dummies, Stubs and Mocks

- You do not need to complete all classes at once.
- Using skeleton classes helps compiling and testing still incomplete programs.
  - *Testing the program class by class will help you significantly in tracking errors.*

Dummys

- Use ??? as method “implementation”
  - – Calls a method of type `Nothing`
- Allows compilation without doing anything.
- Thus, you can write all method headings ready and delay implementation.

```
class Cafe (val coffeemaker: Coffeemaker) {

  def makeOrderTryCatch(amount: Int): Buffer[Coffee] = {
    ???
  }

  def makeOrderTry(amount: Int): Buffer[Coffee] = {
    ???
  }

  def addMilk(coffees: Buffer[Coffee]) = {
    ???
  }

  def addBeans(): Unit = {
    ???
  }

  def cleanMachine(): Unit = {
    ???
  }
}
```

Stubs

- Support step-by-step testing.
- Implements a method so that it returns a “prespecified” value.
- The method can be called when testing the calling method.

Stub example

- You are implementing a class which would manage data from a data base.
- You can write a stub class / method which returns a value without actually reading it from the data base yet, and use this value when testing operations.

Stub example…

```scala
// The interface class that the actual class would implement
trait CustomerDB {
  def getCustomerByName (name: String): Option [Customer]
}

// A stub that "replaces" the actual class
class CustomerDBStub extends CustomerDB {
  def getCustomerByName (name: String): Option [Customer] = {
    // does not really access the database, but creates this "on the fly"
    Some (new Customer (name))
  }
}
```

Mocks / Fakes

- An extension to a stub.
- Instead of returning always the same value, Mock can recognize given parameters and return prespecified values, which correspond to given parameter values.
- Thus, mocks support ”simulating” more complex cases when full implementation is still ahead.
- Terminology (dummy, mock, fake, stub, spy) is not consistent.
  - Course material is based on https://www.techtarget.com/searchsoftwarequality/tip/Inside-5-types-of-test-doubles

Stepwise development

- One good practice is to make a testing app with which you can test your classes and methods one by one
  - Creates/manages input test data which is given to methods as parameters or in collections
  - Checks the correctness of returned values or collection content.
  - Possibly prints out their values for observation
- Alternatively, create a simple user interface which allows giving values and observing returned results.
- Third option is building unit tests, which is discussed in Chapter 15.

Questions