Lecture 7
Chapter 7: Designing classes

How to write classes in a way that they are easily understandable, maintainable and reusable

Main concepts to be covered

- Responsibility-driven design
- Coupling
- Cohesion
- Refactoring

Software changes

- Software is not like a novel that is written once and then remains unchanged.
- Software is extended, corrected, maintained, ported, adapted…
- The work is done by different people over time (often decades).

Change or die

- There are only two options for software:
  - Either it is continuously maintained
  - or it dies.
- Software that cannot be maintained will be thrown away.

Code quality

Two important concepts for quality of code:

- Coupling
- Cohesion

Coupling

- Coupling refers to links between separate units of a program.
- If two classes depend closely on many details of each other, we say they are *tightly coupled*.
- We aim for *loose coupling*.

Loose coupling

Loose coupling makes it possible to:

- understand one class without reading others;
- change one class without affecting others.
- Thus: improves maintainability.

The Spanish holiday example

- Remember how we used abstraction and modularisation in planning a trip to Spain
- We noted planning was easier if the modules were independent
  - In other words, if they were loosely coupled
- In fact, there is coupling in this example
  - Which airport to use depends on the destination
  - How to get to the airport depends on which airport you use
- We aim to reduce coupling but we can't avoid it completely
  - If there was no coupling, classes wouldn't interact at all!

Cohesion

- Cohesion refers to the the number and diversity of tasks that a single unit is responsible for.
- If each unit is responsible for one single logical task, we say it has *high cohesion*.
- Cohesion applies to classes and methods.
- We aim for high cohesion.

High cohesion

High cohesion makes it easier to:

- understand what a class or method does;
- use descriptive names;
- reuse classes or methods.

Cohesion

Cohesion of methods :

- A method should be responsible for one and only one well defined task

Cohesion of classes:

- Classes should represent one single, well defined entity

Code duplication

Code duplication

- is an indicator of bad design,
- makes maintenance harder,
- can lead to introduction of errors during maintenance:
  - One version is updated, not the other(s)

Responsibility-driven design

- Question: where should we add a new method (which class)?
- Each class should be responsible for manipulating its own data.
- The class that owns the data should be responsible for processing it.
- RDD leads to low coupling.

Localizing change

- One aim of reducing coupling and responsibility-driven design is to localize change.
- When a change is needed, as few classes as possible should be affected.

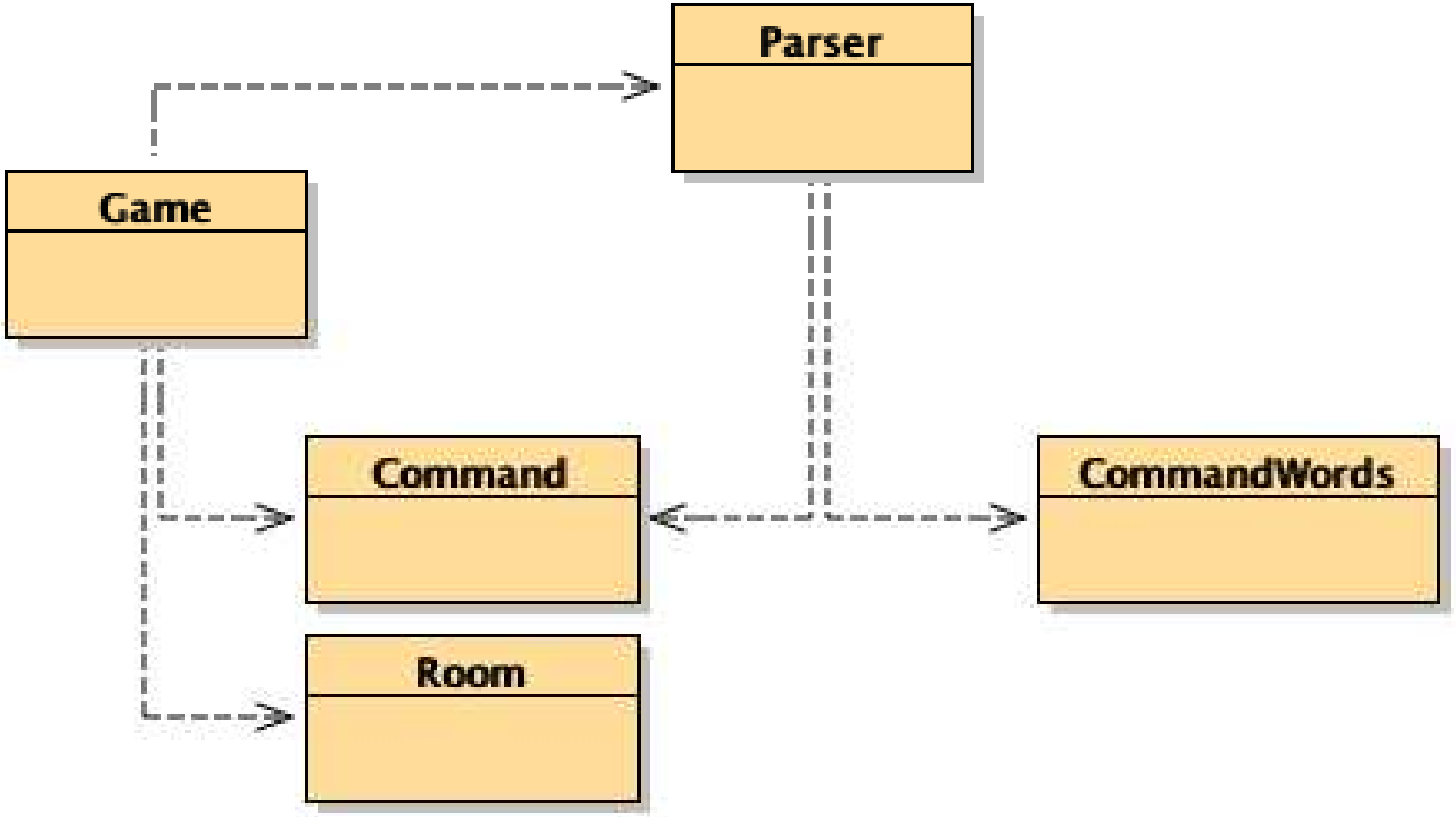
Thinking ahead

- When designing a class, we try to think what changes are likely to be made in the future.
- We aim to make those changes easy.

Refactoring

- Refactoring: restructuring classes and methods to meet new requirements.
- When classes are maintained, often code is added.
- Classes and methods tend to become longer.
- Every now and then, classes and methods should be *refactored* to maintain cohesion and low coupling.

World of Zuul



Class Functionality

- **CommandWords** defines valid player commands using an array of Strings.
- **Command** objects represents player's commands.
- **Parser** reads text and tries to make Command objects.
- **Room** has exits leading to other rooms.
- **Game** has main loop reading and executing commands.

Why use command objects?

- When the parser recognises a command, it could just call a method to handle it
  - Why represent the command with an object?
- It lets us use the power of OOP:
  - We can store, pass around, manipulate, reason about, and base decisions on commands
  - E.g. We can implement an undo feature
- The principle is:
  - Instead of hardwiring logic into the code, turn it into a class
- This is a very powerful principle
- We will see much more on this in later lectures

Refactoring and testing

- When refactoring code, separate the refactoring from making other changes.
- First do the refactoring only, without changing the functionality.
- Test before and after refactoring to ensure that nothing was broken.

Design questions

Common questions:

- How long should a class be?
- How long should a method be?

Cohesion and coupling *suggest*:

- A class should only represent 1 logical entity
- A method should only perform 1 logical task

However, those are just guidelines

Review

- Programs are continuously changed.
- It is important to make this change possible.
- Quality of code requires much more than just performing correctly at one time.
- Code must be understandable and maintainable.

Review

- Good quality code avoids duplication, displays high cohesion, low coupling.
- Coding style (commenting, naming, layout, etc.) is also important.
- There is a big difference in the amount of work required to change poorly structured and well structured code.