Designing Classes

T. M. Murali

Mar 3, 2004
Designing Classes

- How do we write classes such that we can easily understand, maintain and reuse them.

- Main concepts
  - Coupling
  - Cohesion
  - Responsibility-driven design
  - Refactoring
Software Changes Constantly

- Software is not static.
- Software is extended, corrected, maintained, ported, adapted... over a long period of time by many people.
BlueJ Example: World of Zuul

- Two implementations: bad and better.
- Look at the bad example in detail and think about how to improve it.
- 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
- Results in improved maintainability.
Cohesion

- Cohesion refers to the number and variety 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

- A method should be responsible for exactly one well-defined task.

- A class should implement exactly one well-defined entity.
Code Duplication

- Code duplication
  - indicates that the design is bad
  - makes maintenance harder
  - leads to introduction of errors during maintenance
To which class should we add a new method or new functionality?
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.
Localising Change

- Reducing coupling and responsibility-driven design help to localise change.
- When a change is needed, as few classes as possible should be affected.
Thinking Ahead

- When designing a class, try to think what changes you may want to make in the future.
- Aim to make those changes easy.
Refactoring

- When you maintain a class, you often add code.
- Classes and methods tend to become longer.
- Every now and then, you should refactor classes and methods to maintain cohesion and low coupling.
Refactoring and Testing

- When refactoring code, separate the refactoring from making other changes.
- First do only the refactoring, without changing the functionality.
- Test before and after refactoring to ensure that your changes did not introduce new bugs.
Design Questions

- Common questions:
  - How long should a class be?
  - How long should a method be?
- You can answer these questions in terms of cohesion and coupling.
Design Guidelines

- A method is too long if it does more than one logical task.
- A class is too complex if it represents more than one logical entity.
- These guidelines still give a lot of choice to the designer.