Object Oriented Design

Development Process

- The process leading to solving a problem with a program includes much more than “write Java”.
- Major steps:
  - establish requirements
  - design
  - implement design
  - test
- Most people jump to “implement” too fast.

Requirements

- The "requirements" indicate what the finished product should do.
- eg. user input, corresponding output, files created, how fast, etc.
- Typically don't specify *how* to do it.
- Often provided: by your employer, part of an assignment, etc.
- Often incomplete & must be clarified.

Design

- Deciding how to program will meet the requirements.
- What programming language, classes to create, methods of the classes, algorithms, etc.
- Careful design now will save time later.
  - A well-designed system will be more flexible and easier to work with & modify.
- Don't ignore this step!

Implementation

- Important decision should have been made during the design process.
- Implementation is a much more technical process.
- Take the design and actually implement it in some programming language.

Testing

- Mistakes during previous steps are inevitable.
- Must test the program and make sure it meets the requirements.
- Should also test parts as implementing
  - It's much easier to debug small parts of code.
  - Functions and objects help logically separate code to help debugging.
- Some testing can be automated. (later)

Design Notes

- The process from requirements through testing is never directly linear; there is interaction:
  - part of the design requires clarifying the requirements
  - details of implementation require refining the design
  - testing uncovers bugs that need to be fixed
- Inevitable, but experience and attention to detail will minimize & increase efficiency.

Object Design

- We will focus on object-oriented design.
  - We will concentrate on object-oriented design and programming in this course.
  - Not the only way to design programs, but it's widely used.
- Design consists of the specification of the classes that will be used to build the program.
  - If the classes are well designed, it should be easy to put them together to complete the program.

Identifying Classes & Objects

- General idea: read the specification and identify the **nouns**.
  - **Some** of these will before classes (or objects == instances of those classes) in the program.
- This isn't a rule, but often a good starting point.

Example

- A partial problem statement:

  The user selects television programs to record. A recording will have a start and end time specified. The quality of recording can vary, depending on the user's needs. Each recording will be stored on disk and can be accessed from a menu of programs.

- Nouns highlighted.

Example

- In this program, we might create classes for:
  - user
  - TV program
  - recording
  - time
  - quality
  - user's needs
  - disk (file?)
  - menu

But…

- Unclear from the specification we have:
  - user: Do we need to support multiple users (which might require a `User` class), or is that just indicating some user interface?
  - recording: Does that need to be a separate object, or is it just a `Program` with a flag that says it has been recorded?
  - user's needs == preferences? What kind? Different for each user or each recording?
  - …

Problem Details

- We need a lot of clarification of the example problem statement.
  - Not surprising: one paragraph is very short.
- The level of detail in a problem statement will vary.
  - Often requires intelligent interpretation for design.
  - … or requires discussion with the "client" to work out the details.

Designing Classes

- There is always a balance in designing classes.
- Don't create a new class for everything you can imagine.
  - Some things can be represented with built-in types.
  - Some things just don't make sense as a class.
  - Some nouns can be combined in a single class.

Example

- "user" may just refer to possible input they may make: no class needed.
- "quality" might just be a number (1–100)
  - … but maybe we need to create methods for a "quality" that `int`s don't have, so it does need to be a class.
- There are classes in the standard library for "times". Maybe they'll do?

Assigning Responsibilities

- Once you have decided what the classes will be, what will each one do?
  - What methods will they have? What variables do they need to store?
  - Don't need to specify every single detail in the design.
- What parts of the functionality will each class be "responsible" for?

Methods and Responsibilities

- The methods will have to implement the individual behaviours of the program.
  - Methods should generally be verbs: the action that the class/object will take.
  - Need to decide which class gets what functions.
- eg. to initiate a recording, what class do we access?
  - Could be any of: `program.record();` `channel.record();` `user.setRecording()`