Lecture Set #15: Algorithms and Design

1. Algorithmic Design
2. Algorithmic Analysis
3. System Design

What is an algorithm?

- The method used to solve a particular problem is called an **algorithm**.
- **Example**: Make a peanut butter and jelly sandwich:
  - Get a loaf of bread
  - Remove two slices
  - Get a jar of peanut butter
  - Get a knife
  - Open the jar
  - Using the knife, get some peanut butter and spread it on one slice
  - …blah, blah, blah
- There is essentially **one sequential process** being described.

Low-Level Design: Pseudo-Code and Algorithms

- We have already talked about pseudo-code as a design technique
  - NOT English
  - NOT a program
  - Something in-between
    - Captures the logic, flow of desired code
    - Note that pseudo-code could be translated into any programming language (not just Java)
- Pseudo-code is used to represent algorithms = step-by-step solutions to problems
- Algorithms are often coded as single methods

Concerns at the Algorithmic Level of Design

- Correctness

  Does my algorithm correctly solve the problem?
- Efficiency

  Is my algorithm fast enough for the job?
- Clarity

  Is my algorithm understandable? and is it implementable?

Putting all your eggs in one basket

- Problem: I have 16 baskets full of 12 eggs each; I want to “put all of my eggs in one basket”. ☺
- Algorithm #1 ??
  - Combine #1 and #2
  - Combine result with #3
  - Combine result with #4; etc.
- Algorithm #2 ??
  - Combine #1, #2; combine #3, #4; combine #5, #6…
  - Combine <1,2> with <3,4>; Combine <5,6> with <7,8>…
  - Combine <1,2,3,4> with <5,6,7,8>
  - Combine last two …

Algorithmic Efficiency Analysis

- Measuring which is better for time.
- What if the time required for the merging machine is constant?
  - both have 15 calls to the merging machine when there are 16 baskets to merge
  - what if the number of baskets is another value?
- What if the time required for the merging machine is dependent on the number of eggs being merged?
  - for example 1 second per egg (i.e. merging two baskets of 12 each takes 24 seconds)
  - this takes more math when you want to generalized on the number of baskets

Big-O Notation

- Categories of formulas
- What takes over as n approaches infinity
- O(log n)
- O(n)
- O(n * log n)
- $O(n^2)$
- $O(n^2 * \log n)$

Coding vs. Software Design

- Coding: writing of (Java) code to implement classes, methods, etc.
  - Projects so far have been primarily coding
  - We have told you what to code
- Design: determination of what to code
  - What classes are needed?
  - How should classes interact?
  - What methods belong in each class?
  - How should method functionality be implemented?

High-level → Low-level

Interfaces and Design

- Next level up the design hierarchy: what methods should go in classes?
- This information can be captured using interfaces
- These interfaces can also be used to identify opportunities for polymorphism (reusable code)
- Rules of Thumb
  - Keep interfaces small
  - Think carefully about operations needed “between classes”
  - Use interfaces to support polymorphism (and keep code size down)

Upper Levels of Software Design

- Where do ideas for classes, interactions between classes come from?
  - Software development part of larger system design process
  - System design requires identifying what system users expect system to do
  - These user requirements often suggest system components and how they fit together
- First part of software design: understand system design

System Design: What Is It?

- System design is concerned with:
  - coordinating a collection of entities…
  - … to achieve a complex process
- Each entity has its own responsibilities to the others to achieve an overall objective
- E.g. Running a restaurant involves a coordinated interaction of many entities within one system:
  - Entities Chef, owners, waiters, etc.
  - System Restaurant

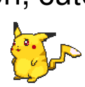
Other Examples of Systems

Classroom environment: Lecturers, TAs, students, …

Library: Circulation (checkout and return), indexing services (online catalogue), library users, book buyers, shelvers, …

Pharmacy: Patients (and medical records), pharmacists, doctors, drug retailers, the pharmacy (products in stock), …

Video game: Race cars, motorcycles, warriors, space ships, death squads, monsters, aliens, mutants, guns, swords, weapons of mass destruction, cute Japanese cartoon animals with huge eyes, …



Pikachu visits Doom3

Essential Questions

- **Challenges:** System design is very hard. Once the number of entities and interactions becomes large, it is very hard to foresee all the possible consequences of these interactions.
- **Essential Questions**:
  - What is the **desired behavior** of the program (as a whole)?
    - system design - overview
  - What are the **entities** that produce this behavior?
    - classes or objects
  - How do these entities **interact**?
    - API for each class
  - How does each one **work**?
    - algorithm for each task

Behavior

- **Specifying Desired Behavior**: A **use case** is a description of the interaction of a user and the system. It includes:
  - **Prerequisites** (**pre-conditions**): What must hold for this use case to arise?
  - **Possible actions and interactions**: What happens?
  - **Effects** (**post-conditions**): What conditions hold, what changes have taken place, as a result of these actions.
- **Example**: Customer in a restaurant.
  - **Pre-conditions**:
    Customer: hungry and has money
    Restaurant: has food
  - **Actions**: get menu, order food, be served, eat, pay, leave
  - **Post-conditions**:
    Customer: less hungry and less money
    Restaurant: more money and less food.

Principal Design Elements

- **Components**:
  - What are the **entities** that make up our system?
  - What are the **roles** they play?
  - How do we separate the system into **distinct units**?
- **State**: What is the current status/state of the units that define our system?
- **Contract**: What are the **responsibilities** and services associated with each component? What **guarantees** does it make?
- **Communication**: How do components request interactions with each other?
- **Example**: **Pharmacy Store System**

  Components: Pharmacist, customers, doctors, prescription, store stock.

  State: For a patient: Current prescriptions, number of times refilled, date of last refill, health insurance information.

  Fill-prescription Contract: A valid prescription is presented by the customer. Check patient records and inform of possible side-effects. Dispense the prescription. Update patient records. Deliver medication to patient.

Relationship to Java

- **System**: A Java **program**
- **Components** (or **community members**): Java **class objects**
- **State**: Each object stores information about its current status. These are stored in class **instance variables**.
- **Contract** (or **specification**): This is called an **API** (Application Programmer Interface), or simply an **interface**. This is the **external** (class user) **view** of an object. It provides an abstraction of what the object does, without indicating how it is implemented. The interface provides the **signatures**, that is, details on how invoke, each action.

  The contract is implemented by the object's class **methods**.