CMSC 131
Object-Oriented Programming I

Algorithms, System Design

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane
Overview

- Algorithms
- 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.
- 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
Categories of formulas
What takes over as n approaches infinity

\( O(1) \)
- Constant time
- **Example**: array indexing

\( O(\log n) \)
- **Example**: binary search

\( O(n) \)
- **Example**: linear search

\( O(n \log(n)) \)
- **Example**: comparison-based sorting algorithm

\( O(n^2) \)
- **Example**: 2D Matrix addition
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?

Diagram:
- 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

- **Review:** We have already discussed software design techniques:
  - *Software lifecycle* (analysis, design, implementation, ...)
  - *Incremental* (stepwise) **design** (from high-level to low-level)
  - *Pseudocode*

- **Wider view** of program development

- **Two principal aspects of system design:**
  - **A single computational entity:** (for small tasks) What is the step-by-step process to produce a desired outcome.
  - **Coordinating a community of entities:** (for large tasks) How to design a collection of entities that coordinate to solve a complex task.

Today we focus on this aspect of software design.
A Single Computational Entity

- Consider a **single computational entity** to solve a specific problem. The method used to solve the problem is called an **algorithm**.
- There is essentially **one sequential process** being described.
System Design: What is it?

- **System Design**: Is concerned with coordinating a community of computational entities to achieve a complex process. Each entity has its own responsibilities to the others, to achieve an overall objective:
  - Single entity: Make a sandwich
  - System design: Run a restaurant

- Running a restaurant involves the coordinated interaction of many entities:
  - Owner
  - Chefs
  - Waiters
  - Diners
System Design: What is it?

- **System Design**: Identifying entities, assigning responsibilities, defining how these entities act and interact with each other.

- **Other Examples of Systems**:
  - **Classroom environment**: Lecturers, TAs, students, ...
  - **Library**: Circulation (checkout and return), indexing services (online catalogue), library users, book buyers, ...
  - **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)?
  - What are the *entities* that produce this behavior?
  - How does each one *work*?
  - How do these entities *interact*?
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?

- **Contract:** What are the responsibilities and services associated with each component? What guarantees does it make?

- **State:** What is the current status/state of the units that define our system?

- **Communication:** How do components request interactions with each other?

- **Example: Pharmacy Store System**
  - **Components:** Pharmacist, customers, doctors, prescription, store stock.
  - **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.
  - **State:** For a patient: Current prescriptions, number of times refilled, date of last refill, health insurance information.
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**.
**Printer System**: A printer system used by students to print projects, homeworks, etc.
- Students enter a room where the printer resides. **Students** submit print requests from a **submission console**.
- An **employee** operates a **printer controller**, and hands out a print job submitted by students.
- A **technician** maintains the printer controller if anything breaks.
Use Case Example 1: A student prints a document

Pre-conditions (prerequisites): A document exist and is ready to be printed.

Actions:

if ( printer operational and there is space available in the printer queue )

• specify name of document to print through the console
• printer controller accesses document and sends it to the printer

else

• printing of the document does not occur
• an appropriate error message is generated on the console

Post-conditions (effects): A document has been printed, or an error message has been generated.
Printer Controller Example

- **Use Case Example 2**: A technician fixes a printer problem

  **Pre-conditions** (prerequisites): A problem has been identified in the printer controller

  **Actions:**
  - A technician enters his **password** in the printer controller (to gain access to the system)
  - The technician asks for a **status report** from the controller
  - The nature of the **problem is identified**
  - The technician proceeds to **repair** the problem area in the printer.

  **Post-conditions** (effects): The printer is **now operational** and the controller reflects the **new status**.
Use Case Example 3: An employee picks up a student print-out

Pre-conditions (prerequisites): A document has been sent to the printer by a student through the console.

Actions:
- a student inquires about his/her particular document
- the employee checks the stack of printed documents in the printer tray.
- if document is found
  - the document is handed to the student
- else
  - student is informed that the document is not printed

Post-conditions (effects): A student has a printed document or has been informed that the document is not been printed.
Printer Controller Example: Components

- Examples of some components:
  - **Name:** Printer  
    **Description:** Provide paper printouts of documents.  
    **State:** Paper quantity and other printer status information.
  - **Name:** Printer Controller  
    **Description:** Used by the Employee and Technician to interact with printer.  
    **State:** Keeps track of documents to be processed, documents already finished, and the operational status of the printer, technician’s access password.
  - **Name:** Printer Submission Console  
    **Description:** Used by students to submit a document to print.  
    **State:** The set of user’s allowed in the system, the set of students that are currently in the system.
Printer Controller Example: Interactions

Examples of some interactions:

**Interaction 1: Controller and Printer**
The controller *sends documents* to process to the printer. It also gathers *status information* from the printer which is made available to the employee or technician.

**Interaction 2: Console and Controller**
The console *sends a query* to the controller which determines whether the document can be printed or not. It *schedules* the job to printed. Status information about the job is sent back to the console.