Getting Your Project Started

Architecture Design
& Starting to Code

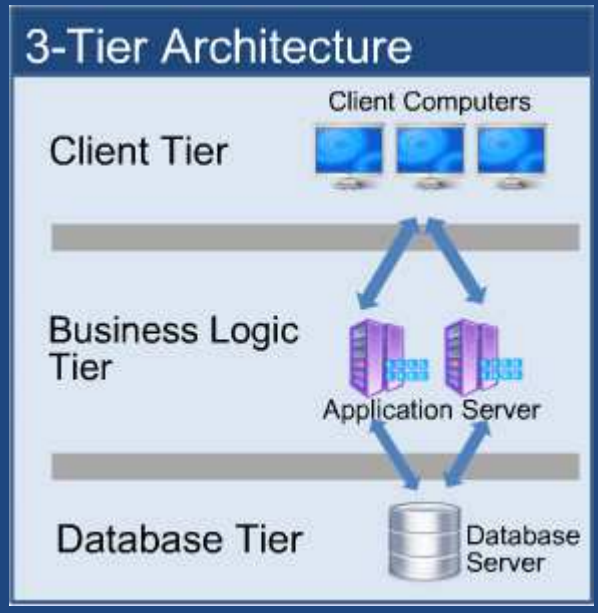
Topics

1) How to design a system's architecture
2) How to do OOD
3) How to get started coding
4) Managing complexity

Layered Architectures

3-Tier Application

- Good “default” 3-tier application architecture:
  - ..
    UI displays data, interacts with user
  - ..
    application logic
  - ..
    simple data objects and
    persistent data storage
    (database/file system)

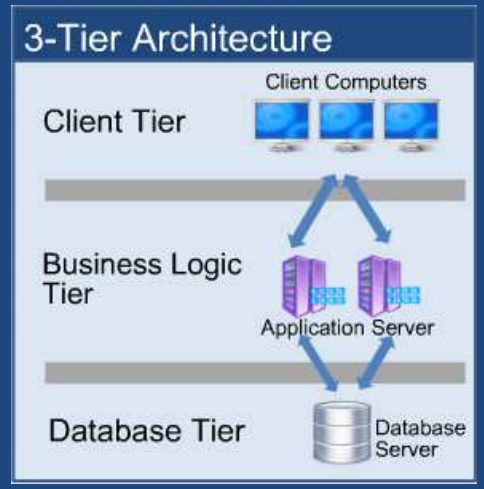


3-Tier Application (cont)

- Advantage:

  ..

  - easier to understand and code:
    smaller, simpler self-contained parts
  - easier to maintain:
    changing UI does not change how data is stored
  - easier to test:
    can test business logic and
    data tiers without UI.



Example functionality

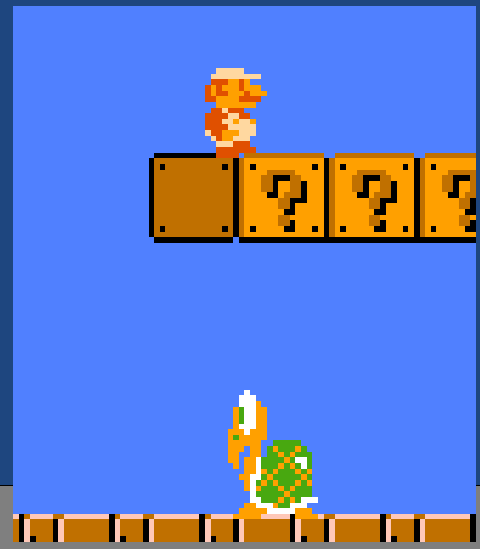
- Super Mario style game functionality
  - Store which direction Mario is facing ____
  - Draw Mario facing correct direction ____
  - Accept user input to move Mario left ____
  - Adjust Mario's position to left ____
  - Calculate if Mario collided with an enemy ____

Options

Pres

Logic

Data



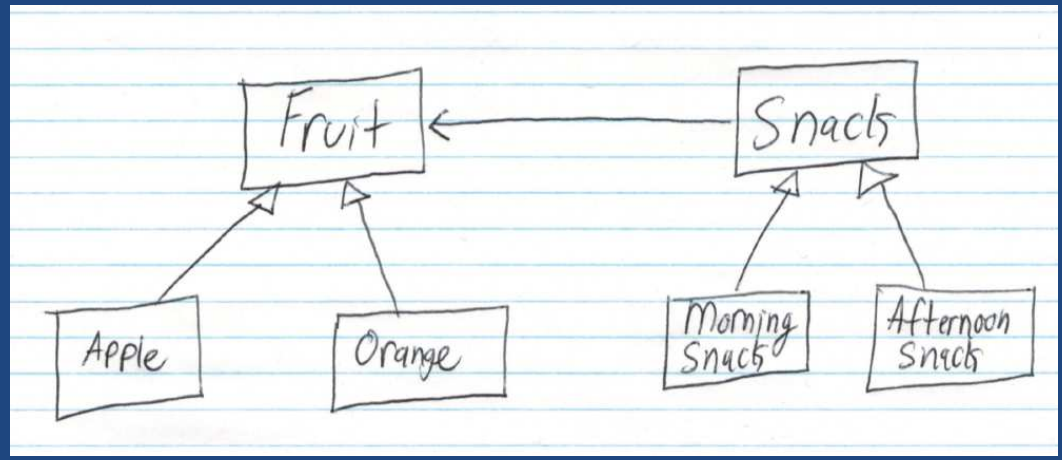
Recommended Steps

- Identify architecture
  - single app, client-server, cloud based, etc.
- Identify layers (3-Tier)
- CRC cards to analyze high-priority user stories ("Class-Responsibility-Collaborator cards")
  - data classes: what information system process
  - business logic classes: what classes process the data
- high-level UML diagram for class relationships
- make paper UI mockup before UI design/coding

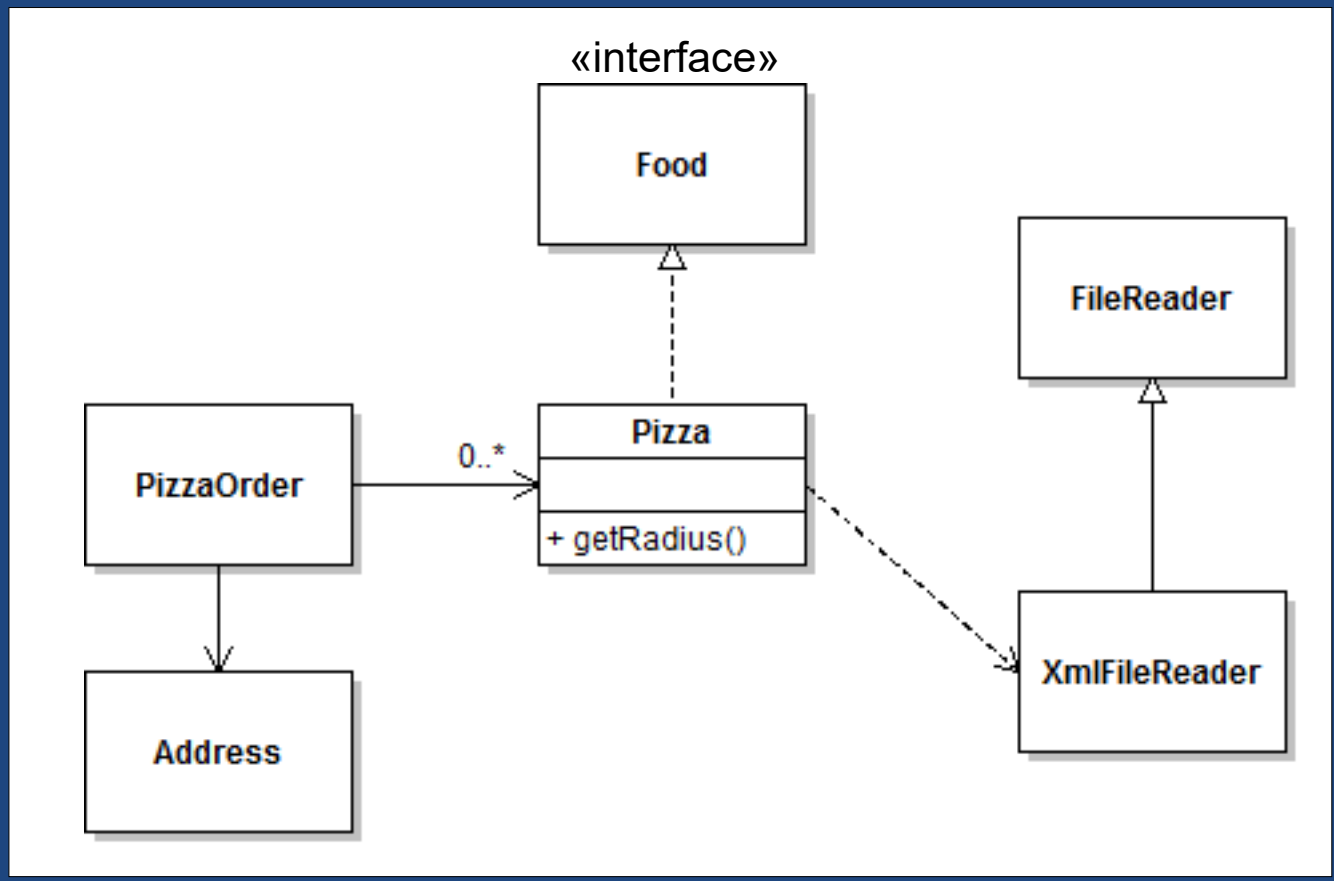
Class Design

UML

- Draw UML Class Diagrams
  - **Informal**: draw on whiteboard to build understanding
  - Don’t over-design it:
    - identify classes and big features;
    - not method names, data types, parameters, ...
  - Industry mostly uses informal UML



Class Connections



Exercise

- Create a UML class diagram including:
  - Book interface
  - ElectronicBook, PrintBook, HardCoverBook classes
  - Reader uses many books
  - Reader loads books from file via a BookFileReader

Understanding Design

- Design is
  - ..
  - ..
    You make many mistakes
  - ..
    No single approach always works
  - ..
    Many passes to make a good design;
    continually evolving during development

Tips

- Avoid inheritance
  - delay inheritance use till later in development process and it becomes obviously needed.
  - prefer dependency over inheritance.
  - use interfaces for polymorphism when possible
- Encapsulation
  - Allow access through well designed interfaces.
- Avoid strings: use enums or custom classes
- Use Observer for model/view
  - don't initially design the observer classes, just mention something will be observable.

Getting Started Coding:
Tracer Bullets

Tracer Bullets?

- Gun Analogy
  - Guns can fire tracer bullets to show where it is shooting
  - User can then correct aim
- Software Development
  - Hard to know how many new subsystems will connect in a large new system.
- Software Tracer Bullets..

Using Tracer Bullets

- Get started coding with Tracer Bullets
  - First implement the entire path through the system, connecting all subsystems.
  - Don't implement all the features/conditions along the way, but ensure it's a working path.
- Not a prototype
  - (Throwaway) Prototype:
    investigate one question, and throw away.
  - Tracer bullets is production quality code; that is the foundation of your implementation.

Why Use Tracer Bullets

- Tracer Bullets Benefits
  - ..
    Hard to start coding with a blank page; this gives a place to start
  - ..
    Always integrate new modules into full architecture
  - Small code body has low inertia:
    quicker to change than a large flushed out system
  - System is always end-to-end demoable.

Managing Complexity

"Primary Technical Imperative:
Managing Complexity"

Software Complexity

- Software is complex!
  - Software systems arguably among the most complex things humans have built.
- Limit complexity:

  ..
  - Can actively think of 7 +/- 2 items (short term memory)
  - Unnecessary details take up “spots”
    - Limits ability to work efficiently; requires “mental juggling”
  - ..

Coding Standards

- Reduces complexity of:
  - reading code that is formatted in different styles
  - writing code:..
- Teams need coding standard specifying:
  - Naming conventions (classes, methods, variables, constants)
  - Brackets, indentation, spacing.
  - Comments
- Repo manager could lead the effort to identify one.
  - CMPT 373 website lists one for Java

Encapsulate

- Reasons to Encapsulate
  It's the gift you give yourself!
  - You can think about higher-level objects
  - You can change details that are hidden
  - You can forget about details that are hidden
- Tips
  - Private fields; few setters
  - ..
    - encapsulate creation details
      Ex: constructor calls init-functions vs client code

Your Project Must...

Backlog Requirements

- Maintain Prioritized Backlog (GitLab issues OK)
  - Allow re-order/prioritizing user stories
  - Allow estimating “points” per story
  - Team / customer / instructor / TA can always view
- Creates GitLab issue when starting task
  - Merge Request closes one or more issues.
  - May create GitLab issues for this iteration’s user stories
  - Suggested GitLab issue boards:
    Backlog, This Sprint, In Progress, Done
  - Suggested GitLab priority labels:
    Low, Medium, High, Critical

Marking

- For marking:
  - “master” branch is marked
  - Massive commits ignored
  - Merge requests go to master
    - If needed, merge master to ‘deploy’ or ‘stable’
  - If adding a framework, place stock framework in one commit, place your changes in another.
    - I can ignore the framework, and then count your contribution!

Additional Requirements

- Web based projects runs on the VM at end of each iteration; customer tries it out from your VM
  - If not web based, have an easy installation process for customer to run
- Git email setting configured with SFU email

```
git config --global user.name "Aria Smith"
git config --global user.email "asmith@sfu.ca
```

Summary

- 3-tier: presentation, business logic, database
  - Encourages good modularity.
- Informal UML for class design.
- Tracer bullets to get started writing and integrating subsystems
  - Avoids big-bang integration
- Manage complexity
  - Unnecessary complexity reduces ability to see big picture.
  - Primary technical imperative