Basic Object Oriented Design Concepts

Software Development

- Software involves four basic activities:

  1. Establishing the requirements
  2. Creating a design
  3. Implementing the code
  4. Testing the implementation

- These activities are not strictly linear
  - They overlap, interact and iterate

Requirements

- Software requirements specify the tasks that a program must accomplish – **what to do**, not how to do it

- Often an initial set of requirements is provided, but they should be critiqued and expanded. Create user stories.

- It is difficult to establish and document detailed, unambiguous, and complete requirements

- Careful attention to the requirements can save significant time and expense in the overall project

Design

- A software design specifies how a program will accomplish its requirements

- A software design determines:
    - how the solution can be broken down into manageable pieces
    - what each piece will do

- An object-oriented design determines which classes and objects are needed and specifies how they will interact

- Low level design details include how individual methods will accomplish their tasks

Object Oriented Design

- Design Methodology / Process
  - Analyze / decompose the requirements
  - Determine the classes required for a program
  - Define the relationships among classes

Identifying Classes and Objects

- The core activity of object-oriented design is determining the classes, and objects that represent the problem and its solution

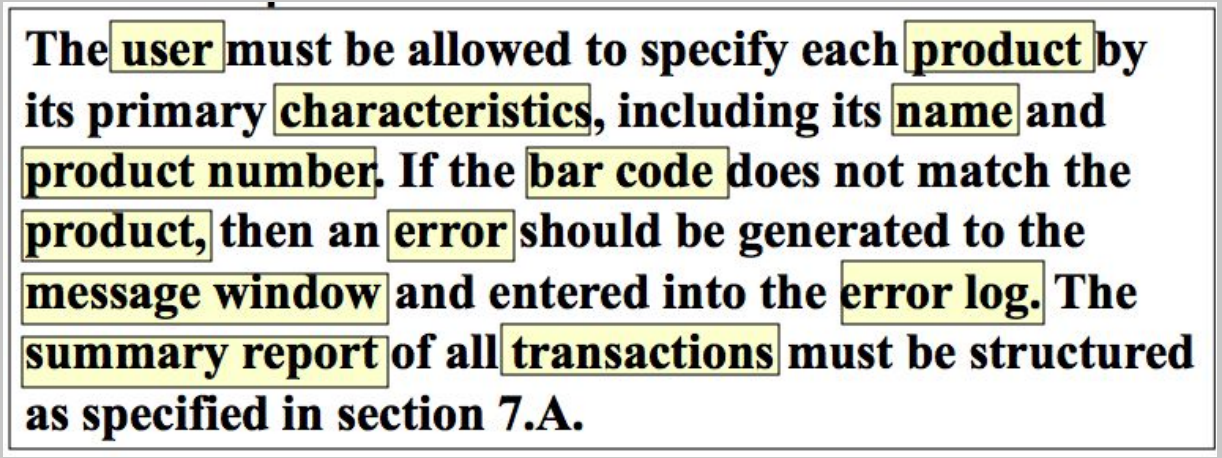
- The classes may be part of a class library, reused from a previous project, or newly written

- One way to identify potential classes is to identify the objects discussed in the requirements

- Objects are generally nouns, and the services that an object provides are generally verbs

Identifying Classes and Objects

Sample user story/requirement:



The user must be allowed to specify each product by its primary characteristics, including its name and product number. If the bar code does not match the product, then an error should be generated to the message window and entered into the error log. The summary report of all transactions must be structured as specified in section 7.A.

Not all nouns will correspond to a class or object in the final solution

Identifying Classes and Objects

- A class represents a group (a “classification”) of objects with the same attributes and behaviors
- Generally, classes that represent objects should be given names that are singular **nouns**. Examples: `Coin`, `Student`, `Message`
- A class represents the concept of one such object
- We are free to instantiate as many “instances” of each object as needed
- Good selection of object names for the instances can be helpful to understanding

Identifying Classes and Objects

- Sometimes it is challenging to decide whether something should be represented as a class

- For example, should an employee's address be represented as a set of instance variables or as an `Address` object

- The more you examine the problem and its details the more clear these issues become

- When a class becomes too complex, it often should be decomposed into multiple smaller classes to distribute the responsibilities

Identifying Classes and Objects

- We want to define classes with the proper amount of detail
- For example, it may be unnecessary to create separate classes for each type of appliance in a house
- It may be sufficient to define a more general `Appliance` class with appropriate instance data
- It all depends on the details of the problem being solved

Identifying Classes and Objects

- Part of identifying the classes we need is the process of assigning responsibilities to each class
- Every activity that a program must accomplish must be represented by one or more methods in one or more classes
- We generally use **verbs** for the names of methods
- In early stages it is not necessary to determine every method of every class – begin with primary responsibilities and evolve the design

Frank’s Approach to OO Design

- Write a few “Happy Path” user stories.
  - Select a user story:
    - I dentify the nouns and verbs in the user story
    - Identify which of the nouns exist and which are new
    - Identify relationship of nouns, if any: “*has-a*”, “*is-a*”, *“is-a-type of*”
  - Identify “high-level” nouns – these will become classes
  - Identify verbs/behaviors – these will be come methods
- Start coding lowest level classes:

  Instance variables, Static variables, Constructors, Getters/Setters, standard overloads, derived attributes, helper methods
- Code higher level classes:

  Instance variables, Static variables, Constructors, Getters/Setters, standard overloads, derived attributes, helper methods

Remember...

- **Nouns** are **classes/objects**
- **Verbs** are **methods**
- Interface names should be adjectives
- “***has-a***” relationship indicates ***Composition***/Containership
- ***“is-a***” relationship indicates ***Inheritance***
- “***is-a-type-of***” indicates ***Interface***
- Implement the functionality in vertical slices. Focus on functionality instead of coming up with all the necessary classes upfront.
- Create user stories to help you understand the requirements
- There is no one right/correct way to do things. Implement the problem as you understand it.

Questions