Program Development

- The creation of software involves four basic activities:
  - establishing the requirements
  - creating a design
  - implementing the code
  - testing the implementation
- These activities are not strictly linear – they overlap and interact

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
- It is difficult to establish 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
- That is, 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

Implementation

- *Implementation* is the process of translating a design into source code
- Novice programmers often think that writing code is the heart of software development, but actually it should be the least creative step
- Almost all important decisions are made during requirements and design stages
- Implementation should focus on coding details, including style guidelines and documentation

Testing

- *Testing* attempts to ensure that the program will solve the intended problem under all the constraints specified in the requirements

- A program should be thoroughly tested with the goal of finding errors

- *Debugging* is the process of determining the cause of a problem and fixing it

Identifying Classes and Objects

- The core activity of object-oriented design is determining the classes and objects that will make up the 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

- **A partial requirements document:**

> **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.**

Of course, not all nouns will correspond to a class or object in the final solution

Guidelines for Discovering Objects

- Limit responsibilities of each analysis class
- Use clear and consistent names for classes and methods
- Keep analysis classes simple

Limit Responsibilities

- Each class should have a clear and simple purpose for existence.

- Having classes with too many responsibilities make them difficult to understand and maintain.

- A good test for this is trying to explain the functionality of a class in a few sentences.

Limiting Responsibilities

- As the design progresses, and more feedback is gotten from potential end-users, the trend of an project is to become more complicated

- Therefore it is probably ok to have tiny objects.

- It is still possible to play out a skinny class in your project and later decide that it can be merged with other classes.

Use Clear and Consistent Names

- **Companies sometimes spend millions just to change their name into a catchier one.**

- **You should give a similar effort to let your classes and methods have suitable names.**

- **class names should be nouns.**

- **Not finding a good name could mean the boundaries of your class is too fuzzy**

- **Having too many simple classes is ok if you have good and descriptive names for them.**

Keep Classes Simple

- In this first step, your imagination should not be crippled with worrying about details like object relationships

Identifying Classes and Objects

- Remember that a class represents a group (classification) of objects with the same 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 of each object as needed

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

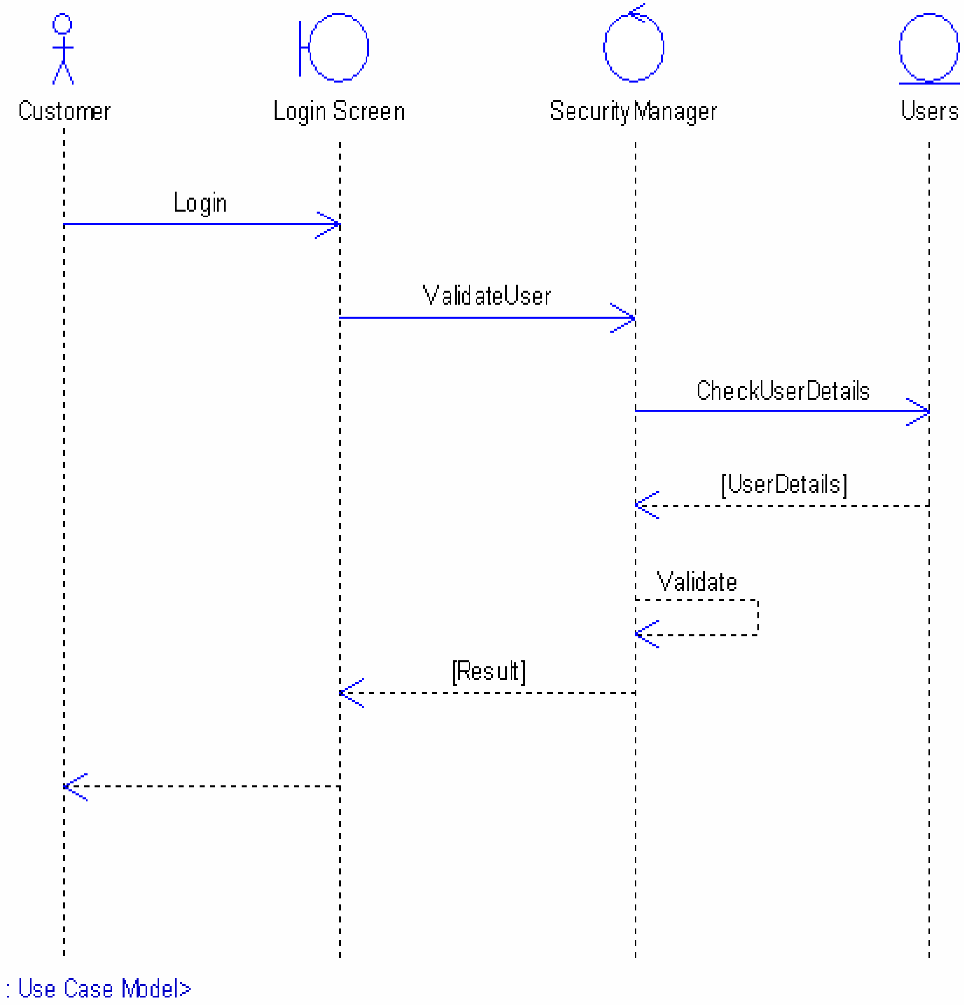
- In early stages it is not necessary to determine every method of every class – begin with primary responsibilities and evolve the design

Describe Behavior

- The set of methods also dictate how your objects interact with each other to produce a solution.

- Sequence diagrams can help tracing object methods and interactions

Sequence Diagram Example



Cohesion between Methods

- methods of an object should be in harmony. If a method seems out of place, then your object might be better off by giving that responsibility to somewhere else.

- For example, getPosition(), getVelocity(), getAcceleration(), *getColor()*

Use clear and Unambiguous Method Names

- Having good names may prevent others to have a need for documentation.

- If you cannot find a good name, it might mean that your object is not clearly defined, or you are trying to do too much inside your method.

Static Class Members

- A static method can be invoked through its class name

- For example, the methods of the `Math` class are static:

```
result = Math.sqrt(25)
```

- Variables can be static as well

- Determining if a method or variable should be static is an important design decision

The static Modifier

- We declare static methods and variables using the `static` modifier

- It associates the method or variable with the class rather than with an object of that class

- Static methods are sometimes called *class methods* and static variables are sometimes called *class variables*

- Let's carefully consider the implications of each

Static Variables

- Normally, each object has its own data space, but if a variable is declared as static, only one copy of the variable exists

```
private static float price;
```

- Memory space for a static variable is created when the class is first referenced
- All objects instantiated from the class share its static variables
- Changing the value of a static variable in one object changes it for all others

Static Methods

```
class Helper
{
   public static int cube (int num)
   {
      return num * num * num;
   }
}
```

Because it is declared as static, the method can be invoked as

```
value = Helper.cube(5);
```

Static Class Members

- **The order of the modifiers can be interchanged, but by convention visibility modifiers come first**

- **Recall that the `main` method is static – it is invoked by the Java interpreter without creating an object**

- **Static methods cannot reference instance variables because instance variables don't exist until an object exists**

- **However, a static method can reference static variables or local variables**

Static Class Members

- Static methods and static variables often work together

- The following example keeps track of how many objects have been created using a static variable, and makes that information available using a static method

```
class MyClass {
   private static int count = 0;

   public MyClass () {
      count++;
   }
   public static int getCount () {
      return count;
   }
}
-------------------------
      MyClass obj;

      for (int scan=1; scan <= 10; scan++)
         obj = new MyClass();

      System.out.println ("Objects created: " +
               MyClass.getCount());
```

Student Id prolem

- Let’s suppose we have a Student class

- How do we assign unique student id’s to each student object that we create?

- What if we also want to get the latest Student created? Like:

```
public static String getLatestStudent()
```