# Basic Software Design

- References:
  - Text book : Appendix C
  - Sun Java Tutorial : http://download.oracle.com/javase/1.5.0/docs/guide/collections/index.html

## 1. Intro to Software Design

- **Problem solving:**
  - The process of taking the statement of a problem and developing computer program that solve problem.
  - May need to pass through many phases to obtain better solutions
- **Object-oriented solution:**
  - A Java computer program consists of modules
  - A module could be a class or a set of classes that work closely together.
  - Each module usually store, move and alter data.
  - Modules use algorithms to communicate with one another
  - To obtain a solution, we need to analyze the requirements of the problem and design a good set of modules (modularity).
  - An important concept to create/design a good solution : abstraction

- **Abstraction and information hiding**
  - Each module can be viewed as a box
  - It contains specifications (what it does, not how to do it) – abstraction
  - Implementation details should be hidden from users
  - This course focus on various general ways to organize and access data (data abstraction)
  - Abstract data type (ADT) is an abstract model that specifies the type of data stored and the operations that support data.
  - To implement an ADT:
    - Define data fields, such as arrays, to store data and operations.
    - Writing programs for operations to manipulate the data.

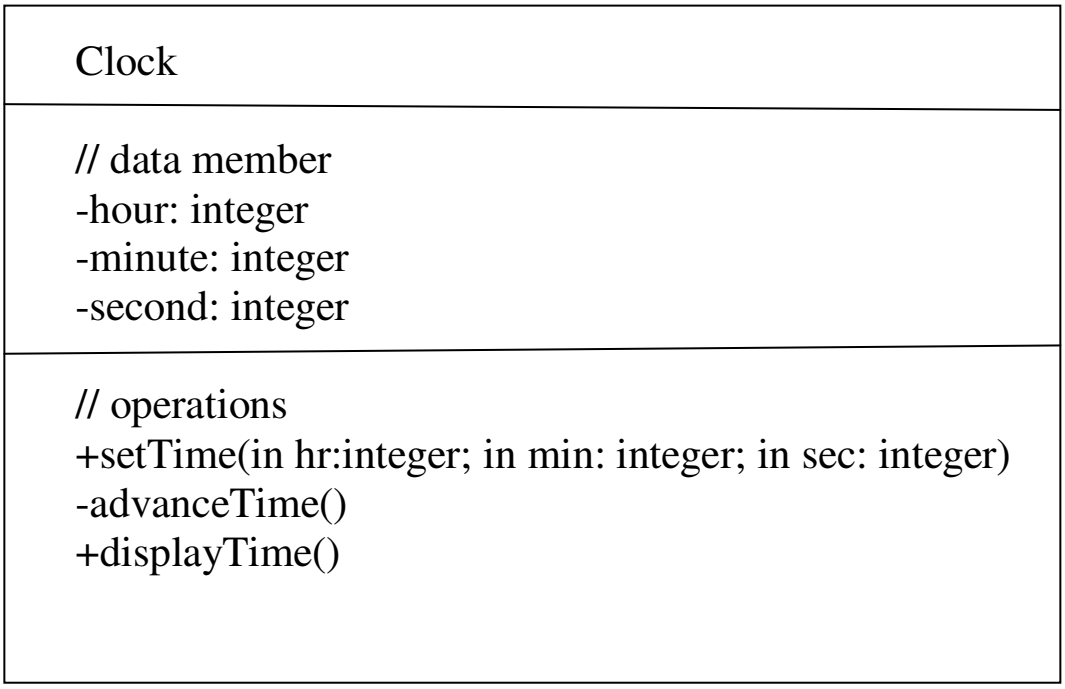
- **Object oriented programming**
  - Hiding the representation of objects and implementation of operations from its users (encapsulation)
  - The designer may alter the representation objects and implementation of operations as long as the user interface remains the same.
  - Use OOP to support ADT implementations.

- **Basic Unified Modeling Language (UML)**
  - Use to express OOD.
  - See http://www.omg.org
  - Visibility of variables/methods and direction of parameters

    *visibility* : + (public), - (private) or # (protected)

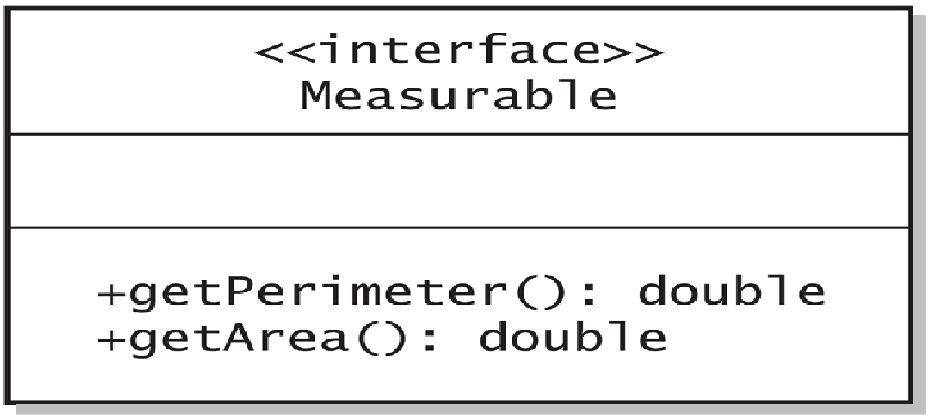
    *direction* is *in* (input), *out* (output) or *inout* (both)

  - Example : A class : Clock

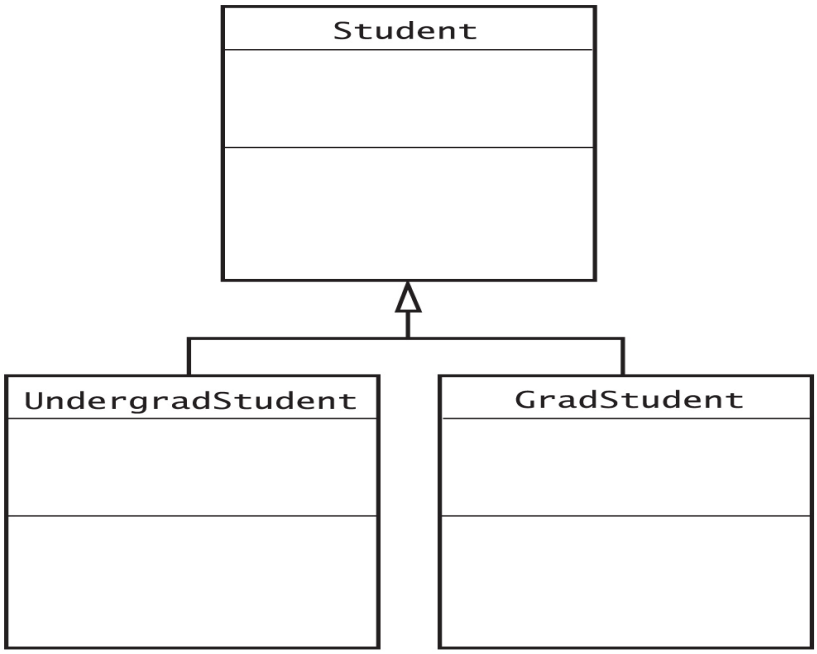



**Note: For an abstract class, use a name start with "Abstract". Example: AbstractList, AbstractSet**

Example : An interface : Measurable (note: <<interface>> )




- Example : A class diagram showing the base class Student and two derived classes.




Note: A solid line and arrow indicate “extend” a base class (or interface)
A dashed line and arrow indicate “implement” an interface

## 2. More on Abstract Data Types (ADT)

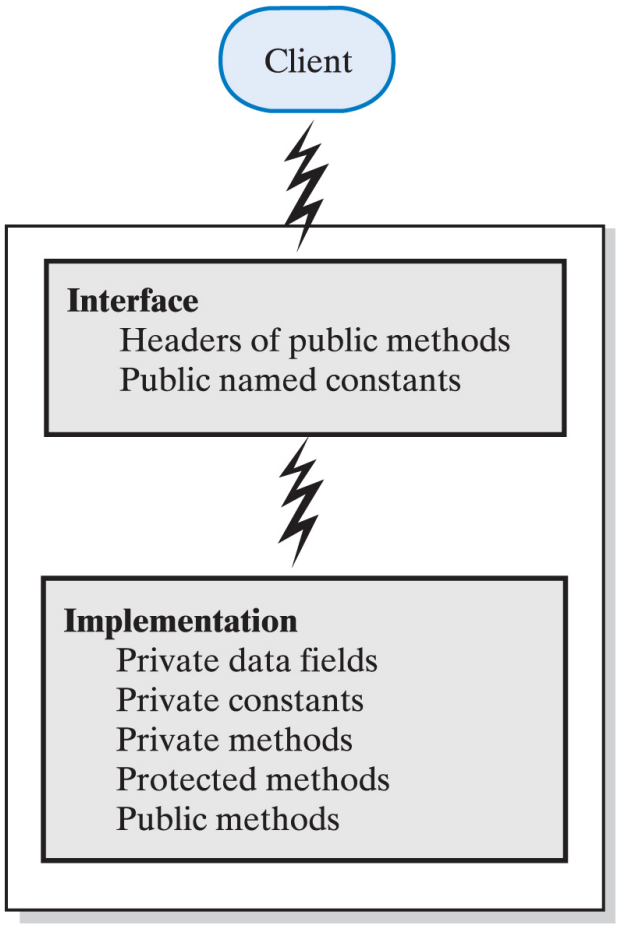
- Recall: a module development

  data abstraction : write specification on module, including a collection of data, set of operations on data.
  Note : focus on "what" a module does and not on "how" it does it.

  information hiding : Should *hide* the implementation details and make them *inaccessible* from outside the module.

  i.e. provide only *client interface* : The user of a module should only know a set of operations that can be performed on the data.

  The client interface is the what
  The implementation is the how



- More on designing classes using Java
  - Java interface is a way to specify an ADT (set of methods or API). It is a contract between the interface designer and the programmer who code a class to implement the interface.
  - Should specify what each method does. These may include

    Purpose of a method

    Pre-condition: Defines responsibility of client code

    Post-condition: Specifies what will happen if the preconditions are met

    Exception-condition: Specifies what conditions may throw exception

    Description of each parameter and return value

    Example:

```
/** An interface for the ADT list.
* Entries in the list have positions that begin with 1.
*/
public interface ListInterface < T >
{
   /** Task: Adds a new entry to the end of the list.
    * Entries currently in the list are unaffected.
    * The lists size is increased by 1.
    * @param newEntry the object to be added as a new entry
    * @return true if the addition is successful, or false if the list
    * is full */

    public boolean add (T newEntry);

    // other methods…..
 }
```

- Data structure: implementation of an ADT within a programming language
- Collection: an ADT that contains a group of objects

  Container: a class that implements the collection

  These last two terms are sometimes used interchangeably
- Major common data structures
  - Bag : Unordered collection, may contain duplicates
  - List : A collection that numbers its items
  - Stack: Orders items chronologically. Last in first out (LIFO)
  - Queue: Orders items chronologically. First in first out (FIFO)
  - Dictionary: Pairs of items – one is a key. Can be sorted or not
  - Tree: Arranged in a hierarchy
  - Graph: Generalization of a tree

- Java Collections Framework
  http://download.oracle.com/javase/1.5.0/docs/guide/collections/index.html
  http://java.sun.com/developer/onlineTraining/collections/
  http://www.javabeginner.com/java-collections/java-collections-framework
  http://www.docjar.com/docs/api/java/util/package-index.html (source)

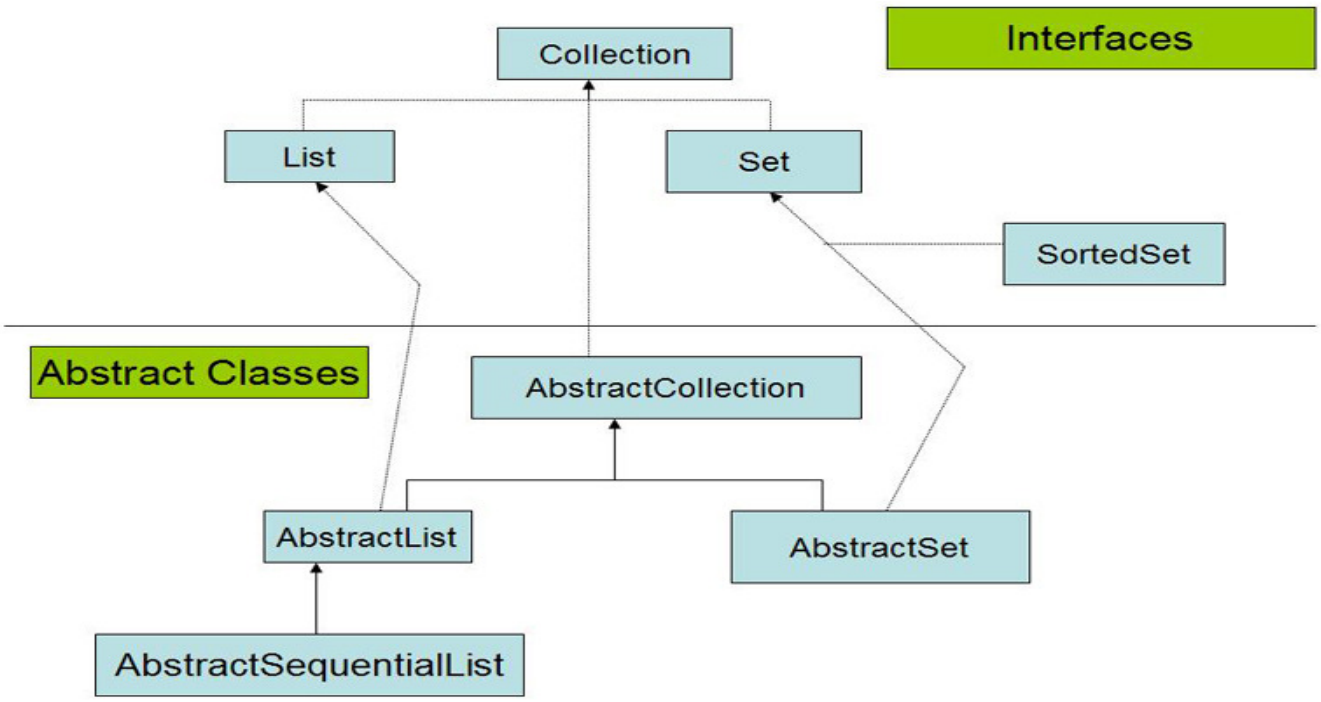
  - It is provided in the java.util package

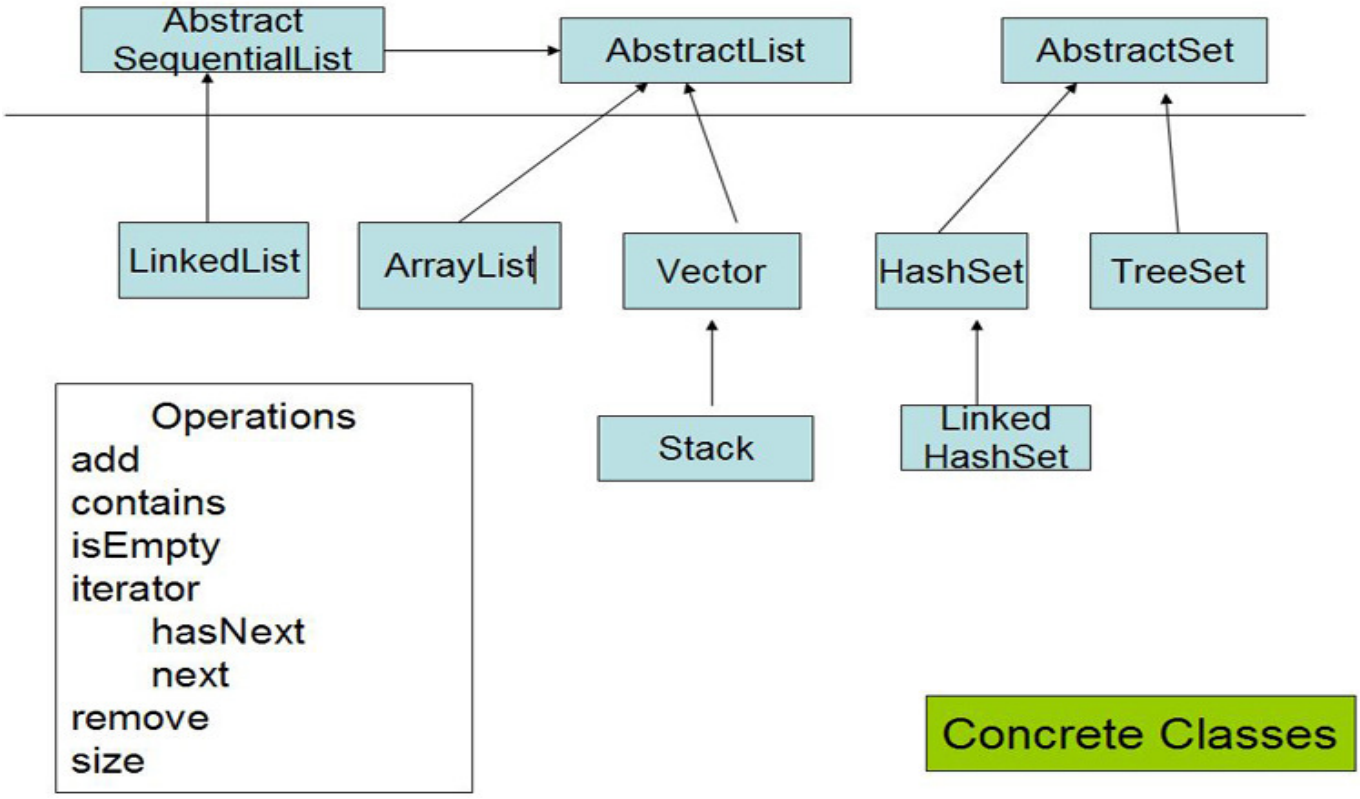
  - It provides a convenient API to many of the abstract data types familiar from computer science data structure curriculum: maps, sets, lists, trees, arrays, hashtables and other collections.

  - A Collections Framework is defined by a set of **interfaces**, **concrete** class implementations for most of the interfaces and a set of **standard utility methods and algorithms**.

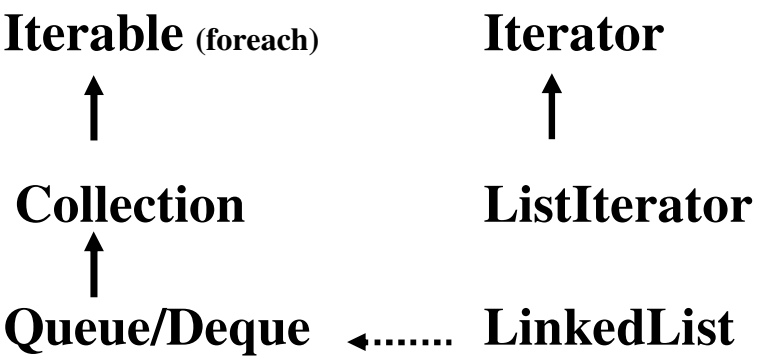
  - In addition, the framework also provides several abstract implementations, which are designed to make it easier for you to create new and different implementations for handling collections of data.

    Part of Collection Hierarchy:






Also, some other interfaces




Another diagram:
http://blog.ifanchu.com/wp-content/uploads/2009/12/collections-2.png