Outline

° Problem: How can I store information in arrays without complicated array management?

° The Java language supports ArrayLists

° We can now solve big problems
  • Can we formalize our problem solving approaches

° Building a blueprint for problem solving
  • Problem definition, identification of classes/objects, code writing, testing
The ArrayList Class

° The ArrayList class is part of the java.util package

° Like an array, it can store a list of values and reference each one using a numeric index

° ➔ However, you cannot use the bracket syntax with an ArrayList object

° ArrayList object grows and shrinks as needed, adjusting its capacity as necessary.
  • Thus we do NOT need an increaseSize() method as we have done in the past!
  • This is called ‘dynamic storage allocation.”
Some methods of ArrayList

- **ArrayList()**
  - Constructs an empty list with an initial capacity of ten.

- **void add(int index, Object element)**
  - Inserts the specified element at the specified position in this list.

- **boolean add(Object o)**
  - Appends the specified element to the end of this list.

- **void clear()**
  - Removes all of the elements from this list.

- **boolean contains(Object elem)**
  - Returns true if this list contains the specified element.

- **Object get(int index)**
  - Returns the element at the specified position in this list.

- **int indexOf(Object elem)**
  - Searches for the first occurrence of the given argument.
ArrayList basics

- ArrayList stores Objects, not int, not boolean, yes String ...
  - Don’t use [] indexing
  - Must cast when getting object out of ArrayList
  - Can add to end, in middle with shifting

```java
ArrayList list = new ArrayList();
list.add(new String("hello"));
list.add(new String("world"));
String s = (String) list.get(0);  // what is it?
list.set(0, new String("big"));
list.add(0, new String("great"));
```
The ArrayList Class

- Elements can be inserted or removed with a single method invocation
- When an element is inserted, the other elements "move aside" to make room
- Likewise, when an element is removed, the list "collapses" to close the gap
- The indexes of the elements adjust accordingly
The ArrayList Class

° An ArrayList stores references to the Object class, which allows it to store any kind of object

° We can also define an ArrayList object to accept a particular type of object

° The following declaration creates an ArrayList object that only stores Family objects

    ArrayList<Family> reunion = new ArrayList<Family>

° This is an example of generics, which are discussed further in Chapter 12
ArrayList Efficiency

- The ArrayList class is implemented using an underlying array.
- The array is manipulated so that indexes remain continuous as elements are added or removed.
- If elements are added to and removed from the end of the list, this processing is fairly efficient.
- Elements are inserted and removed from the front or middle of the list sometimes,
  - The remaining elements are shifted and this tends to become somewhat inefficient.
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
- You cannot design and implement that which you do not understand!
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
  - 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
- ➔ Almost all important decisions are made during requirements and design stages
- Implementation should focus on coding details, including style guidelines and documentation
- Implementation (programming and testing) is really the ‘implementation of a design.’
- The DESIGN is the solution!
Testing

- **Testing** attempts to ensure that the program will solve the intended problem
  - Consider 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:
  - Math class, etc. Existing classes etc. in the API...
- 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.
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 – that is, create an object – as many of each class as needed.
Identifying Classes and Objects

- Sometimes it is challenging to decide whether something should be represented as a class.
- 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.
- ➔ If a class becomes too complex
  - It 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:
  - e.g. `printResults();` `rollDie();` `getGPA();` `calculatePay();`
- In early stages it is not necessary to determine every method of every class:
  - Begin with primary responsibilities and evolve the design.
Summary

° Our knowledge of arrays and loops is now complete
  • We can make arrays of arbitrary size

° Program design is fundamental
  • You wouldn’t create a portrait with thinking about what you are doing

° Class and object selection often follows the nouns in the problem description

° More of a focus on problem solving in coming weeks