Chapter 4

Data Abstraction: The Walls
Abstract Data Types

• Modularity
  – Keeps the complexity of a large program manageable by systematically controlling the interaction of its components
  – Isolates errors
  – Eliminates redundancies
  – A modular program is
    • Easier to write
    • Easier to read
    • Easier to modify
Abstract Data Types

• Procedural abstraction
  – Separates the purpose and use of a module from its implementation
  – A module’s specifications should
    • Detail how the module behaves
    • Identify details that can be hidden within the module

• Information hiding
  – Hides certain implementation details within a module
  – Makes these details inaccessible from outside the module
Abstract Data Types

Figure 4-1

Isolated tasks: the implementation of task $T$ does not affect task $Q$
Abstract Data Types

- The isolation of modules is not total
  - Methods’ specifications, or contracts, govern how they interact with each other

Figure 4-2
A slit in the wall
Abstract Data Types

• Typical operations on data
  – Add data to a data collection
  – Remove data from a data collection
  – Ask questions about the data in a data collection

• Data abstraction
  – Asks you to think what you can do to a collection of data independently of how you do it
  – Allows you to develop each data structure in relative isolation from the rest of the solution
  – A natural extension of procedural abstraction
Abstract Data Types

- Abstract data type (ADT)
  - An ADT is composed of
    - A collection of data
    - A set of operations on that data
  - Specifications of an ADT indicate
    - What the ADT operations do, not how to implement them
  - Implementation of an ADT
    - Includes choosing a particular data structure
Abstract Data Types

- **Data structure**
  - A construct that is defined within a programming language to store a collection of data
  - Example: arrays
- **ADTs and data structures are not the same**
- **Data abstraction**
  - Results in a wall of ADT operations between data structures and the program that accesses the data within these data structures
Abstract Data Types

Figure 4-4
A wall of ADT operations isolates a data structure from the program that uses it.
Specifying ADTs

- In a list
  - Except for the first and last items, each item has
    - A unique predecessor
    - A unique successor
  - Head or front
    - Does not have a predecessor
  - Tail or end
    - Does not have a successor

Figure 4-5
list A grocery
The ADT List

• ADT List operations
  – Create an empty list
  – Determine whether a list is empty
  – Determine the number of items in a list
  – Add an item at a given position in the list
  – Remove the item at a given position in the list
  – Remove all the items from the list
  – Retrieve (get) the item at a given position in the list

• Items are referenced by their position within the list
The ADT List

- Specifications of the ADT operations
  - Define the contract for the ADT list
  - Do not specify how to store the list or how to perform the operations

- ADT operations can be used in an application without the knowledge of how the operations will be implemented
The ADT List

Figure 4-7
The wall between displayList and the implementation of the ADT list
The ADT Sorted List

• The ADT sorted list
  – Maintains items in sorted order
  – Inserts and deletes items by their values, not their positions
Designing an ADT

• The design of an ADT should evolve naturally during the problem-solving process

• Questions to ask when designing an ADT
  – What data does a problem require?
  – What operations does a problem require?
Axioms (Optional)

- For complex abstract data types, the behavior of the operations must be specified using axioms
  - Axiom: A mathematical rule
Axioms (Optional)

- Axioms for the ADT List
  - (aList.createList()).size() = 0
  - (aList.add(i, x)).size() = aList.size() + 1
  - (aList.remove(i)).size() = aList.size() – 1
  - (aList.createList()).isEmpty() = true
  - (aList.add(i, item)).isEmpty() = false
  - (aList.createList()).remove(i) = error
  - (aList.add(i, x)).remove(i) = aList
  - (aList.createList()).get(i) = error
  - (aList.add(i, x)).get(i) = x
  - aList.get(i) = (aList.add(i, x).get(i+1)
  - aList.get(i+1) = (aList.remove(i)).get(i)
Implementing ADTs

• Choosing the data structure to represent the ADT’s data is a part of implementation
  – Choice of a data structure depends on
    • Details of the ADT’s operations
    • Context in which the operations will be used

• Implementation details should be hidden behind a wall of ADT operations
  – A program would only be able to access the data structure using the ADT operations
Implementing ADTs

Figure 4-8
ADT operations provide access to a data structure
Implementing ADTs

Figure 4-9
Violating the wall of ADT operations

© 2011 Pearson Addison-Wesley. All rights reserved
Java Classes Revisited

- Object-oriented programming (OOP) views a program as a collection of objects

- Encapsulation
  - A principle of OOP
  - Can be used to enforce the walls of an ADT
  - Combines an ADT’s data with its method to form an object
  - Hides the implementation details of the ADT from the programmer who uses it
Java Classes Revisited

Figure 4-10
An object’s data and methods are encapsulated
Java Classes Revisited

• A Java class
  – A new data type whose instances are objects
  – Class members
    • Data fields
      – Should almost always be private
    • Methods
      – All members in a class are private, unless the programmer designates them as public
Java Classes Revisited

• A Java class (Continued)
  – Constructor
    • A method that creates and initializes new instances of a class
    • Has the same name as the class
    • Has no return type
  – Java’s garbage collection mechanism
    • Destroys objects that a program no longer references
Java Classes Revisited

• Constructors
  – Allocate memory for an object and can initialize the object’s data
  – A class can have more than one constructor
  – Default constructor
    • Has no parameters
    • Typically, initializes data fields to values the class implementation chooses
Java Classes Revisited

• **Constructors (Continued)**
  – Compiler-generated default constructor
  • Generated by the compiler if no constructor is included in a class

• **Client of a class**
  – A program or module that uses the class
Java Classes Revisited

- **Inheritance**
  - Base class or superclass
  - Derived class or subclass
    - Inherits the contents of the superclass
    - Includes an `extends` clause that indicates the superclass
    - `super` keyword
      - Used in a constructor of a subclass to call the constructor of the superclass
Java Classes Revisited

- **Object Equality**
  - `equals` method of the `Object` class
  - **Default implementation**
    - Compares two objects and returns true if they are actually the same object
  - **Customized implementation for a class**
    - Can be used to check the values contained in two objects for equality
Java Interfaces

• An interface
  – Specifies methods and constants, but supplies no implementation details
  – Can be used to specify some desired common behavior that may be useful over many different types of objects
  – The Java API has many predefined interfaces
    • Example: java.util.Collection
Java Interfaces

• A class that implements an interface must
  – Include an `implements` clause
  – Provide implementations of the methods of the interface

• To define an interface
  – Use the keyword `interface` instead of `class` in the header
  – Provide only method specifications and constants in the interface definition
Java Exceptions

• Exception
  – A mechanism for handling an error during execution
  – A method indicates that an error has occurred by throwing an exception
Java Exceptions

• Catching exceptions
  – try block
    • A statement that might throw an exception is placed within a try block
  • Syntax
    
    ```java
    try {
        statement(s);
    } // end try
    ```
Java Exceptions

• Catching exceptions (Continued)
  – catch block
    • Used to catch an exception and deal with the error condition
    • Syntax
      
catch (exceptionClass identifier) {
      statement(s);
    } // end catch
Java Exceptions

- Types of exceptions
  - Checked exceptions
    - Instances of classes that are subclasses of the `java.lang.Exception` class
    - Must be handled locally or explicitly thrown from the method
    - Used in situations where the method has encountered a serious problem
Java Exceptions

• Types of exceptions (Continued)
  – Runtime exceptions
    • Used in situations where the error is not considered as serious
    • Can often be prevented by fail-safe programming
    • Instances of classes that are subclasses of the RuntimeException class
    • Are not required to be caught locally or explicitly thrown again by the method
Java Exceptions

• Throwing exceptions
  – A throw statement is used to throw an exception
    ```java
    throw new exceptionClass
    (stringArgument);
    ```

• Defining a new exception class
  – A programmer can define a new exception class
An Array-Based Implementation of the ADT List

• An array-based implementation
  – A list’s items are stored in an array \( \text{items} \)
  – A natural choice
    • Both an array and a list identify their items by number
  – A list’s \( k \)th item will be stored in \( \text{items}[k-1] \)
An Array-Based Implementation of the ADT List

Figure 4-11
An array-based implementation of the ADT list
Summary

• Data abstraction: a technique for controlling the interaction between a program and its data structures

• An ADT: the specifications of a set of data management operations and the data values upon which they operate

• The formal mathematical study of ADTs uses systems of axioms to specify the behavior of ADT operations

• Only after you have fully defined an ADT should you think about how to implement it
Summary

• A client should only be able to access the data structure by using the ADT operations.

• An object encapsulates both data and operations on that data.
  – In Java, objects are instances of a class, which is a programmer-defined data type.

• A Java class contains at least one constructor, which is an initialization method.

• Typically, you should make the data fields of a class private and provide public methods to access some or all of the data fields.