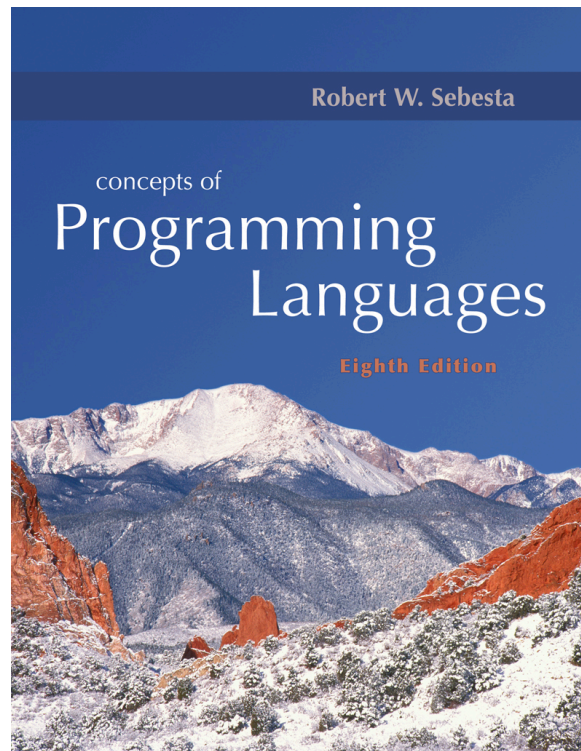


Chapter 11

Abstract Data Types and Encapsulation Concepts



ISBN 0-321-49362-1

Chapter 11 Topics

- The Concept of Abstraction
- Introduction to Data Abstraction
- Design Issues for Abstract Data Types
- Language Examples
- Parameterized Abstract Data Types
- Encapsulation Constructs
- Naming Encapsulations

The Concept of Abstraction

- An abstraction is a view or representation of an entity that includes only the most significant attributes
- The concept of abstraction is fundamental in programming (and computer science)
- Nearly all programming languages support process abstraction with subprograms
- Nearly all programming languages designed since 1980 support data abstraction

1-

Introduction to Data Abstraction

- An abstract data type is a user-defined data type that satisfies the following two conditions:
 - The representation of, and operations on, objects of the type are defined in a single syntactic unit
 - The representation of objects of the type is hidden from the program units that use these objects, so the only operations possible are those provided in the type's definition

1-

Advantages of Data Abstraction

- Advantage of the first condition
 - Program organization, modifiability (everything associated with a data structure is together), and separate compilation
- Advantage the second condition
 - Reliability--by hiding the data representations, user code cannot directly access objects of the type or depend on the representation, allowing the representation to be changed without affecting user code

1-

Design Issues

- A syntactic unit to define an ADT
- Built-in operations
 - Assignment
 - Comparison
- Common operations
 - Iterators
 - Accessors
 - Constructors
 - Destructors
- Parameterized ADTs

1-

Language Examples: Ada

- The encapsulation construct is called packages
 - Specification package (the interface)
 - Body package (implementation of the entities named in the specification)
- Information Hiding
 - The representation of type appears in a part of the specification called the private part
 - More restricted form with limited private types
 - Define the ADT as a pointer and provide the pointed-to structure's definition in the body package

1-

An Example in Ada

```
package Stack_Pack is
  type stack_type is limited private;
  max_size: constant := 100;
  function empty(stk: in stack_type) return Boolean;
  procedure push(stk: in out stack_type; elem:in Integer);
  procedure pop(stk: in out stack_type);
  function top(stk: in stack_type) return Integer;

  private -- hidden from clients
  type list_type is array (1..max_size) of Integer;
  type stack_type is record
    list: list_type;
    topsub: Integer range 0..max_size) := 0;
  end record;
end Stack_Pack
```

1-

Language Examples: C++

- Based on C struct type and Simula 67 classes
- The class is the encapsulation device
- All of the class instances of a class share a single copy of the member functions
- Each instance of a class has its own copy of the class data members
- Instances can be static, stack dynamic, or heap dynamic

1-

Language Examples: C++

- Information Hiding
 - Private clause for hidden entities
 - Public clause for interface entities
 - Protected clause for inheritance

1-

Language Examples: C++

- Constructors:
 - Functions to initialize the data members of instances (they do not create the objects)
 - May also allocate storage if part of the object is heap-dynamic
 - Can include parameters to provide parameterization of the objects
 - Implicitly called when an instance is created
 - Can be explicitly called
 - Name is the same as the class name

1-

Language Examples: C++

- Destructors
 - Functions to cleanup after an instance is destroyed; usually just to reclaim heap storage
 - Implicitly called when the object's lifetime ends
 - Can be explicitly called
 - Name is the class name, preceded by a tilde (~)

1-

An Example in C++

```
class stack {
private:
    int *stackPtr, maxLen, topPtr;
public:
    stack() { // a constructor
        stackPtr = new int [100];
        maxLen = 99;
        topPtr = -1;
    };
    ~stack () {delete [] stackPtr;};
    void push (int num) {...};
    void pop () {...};
    int top () {...};
    int empty () {...};
}
```

1-

Evaluation of ADTs in C++ and Ada

- C++ support for ADTs is similar to expressive power of Ada
- Both provide effective mechanisms for encapsulation and information hiding
- Ada packages are more general encapsulations

1-

Language Examples: C++ (continued)

- Friend functions or classes – to provide access to private members to some unrelated units or functions
 - Necessary in C++

1-

Language Examples: Java

- Similar to C++, except:
 - All user-defined types are classes
 - All objects are allocated from the heap and accessed through reference variables
 - Individual entities in classes have access control modifiers (private or public), rather than classes
 - Java has a second scoping mechanism, package scope, which can be used in place of friends
 - All entities in all classes in a package that do not have access control modifiers are visible throughout the package

1-

An Example in Java

```
class StackClass {
    private:
        private int [] *stackRef;
        private int [] maxLen, topIndex;
        public StackClass() { // a constructor
            stackRef = new int [100];
            maxLen = 99;
            topPtr = -1;
        };
        public void push (int num) {...};
        public void pop () {...};
        public int top () {...};
        public boolean empty () {...};
}
```

1-

Language Examples: C#

- Based on C++ and Java
- Adds two access modifiers, internal and protected internal
- All class instances are heap dynamic
- Default constructors are available for all classes
- Garbage collection is used for most heap objects, so destructors are rarely used
- structs are lightweight classes that do not support inheritance

1-

Language Examples: C# (continued)

- Common solution to need for access to data members: accessor methods (getter and setter)
- C# provides properties as a way of implementing getters and setters without requiring explicit method calls

1-

C# Property Example

```
public class Weather {
    public int DegreeDays { /** DegreeDays is a property
        get {return degreeDays;}
        set {degreeDays = value;}
    }
    private int degreeDays;
    ...
}
...
Weather w = new Weather();
int degreeDaysToday, oldDegreeDays;
...
w.DegreeDays = degreeDaysToday;
...
oldDegreeDays = w.DegreeDays;
```

1-

Parameterized Abstract Data Types

- Parameterized ADTs allow designing an ADT that can store any type elements
- Also known as generic classes
- C++ and Ada provide support for parameterized ADTs
- Java 5.0 provides a restricted form of parameterized ADTs

1-

Parameterized ADTs in Ada

- Ada Generic Packages
 - Make the stack type more flexible by making the element type and the size of the stack generic

```
generic
Max_size: Positive;
type Elem_Type is Private;
package Generic_Stack is
...
function Top(Stk: in out StackType) return Elem_type;
...
end Generic_Stack;
```

```
Package Integer_Stack is new Generics_Stack(100,Integer);
Package Float_Stack is new Generics_Stack(100,Float);
```

1-

Parameterized ADTs in C++

- Classes can be somewhat generic by writing parameterized constructor functions

```
template <class type>
class stack {
    ...
    stack (int size) {
        stk_ptr = new int [size];
        max_len = size - 1;
        top = -1;
    };
    ...
}
stack stk(100);
```

1-

Encapsulation Constructs

- Large programs have two special needs:
 - Some means of organization, other than simply division into subprograms
 - Some means of partial compilation (compilation units that are smaller than the whole program)
- Obvious solution: a grouping of subprograms that are logically related into a unit that can be separately compiled (compilation units)
- Such collections are called encapsulation

1-

Nested Subprograms

- Organizing programs by nesting subprogram definitions inside the logically larger subprograms that use them
- Nested subprograms are supported in Ada and Fortran 95

1-

Encapsulation in C

- Files containing one or more subprograms can be independently compiled
- The interface is placed in a header file
- Problem: the linker does not check types between a header and associated implementation
- `#include` preprocessor specification

1-

Encapsulation in C++

- Similar to C
- Addition of friend functions that have access to private members of the friend class

Ada Packages

- Ada specification packages can include any number of data and subprogram declarations
- Ada packages can be compiled separately
- A package's specification and body parts can be compiled separately

Naming Encapsulations

- Large programs define many global names; need a way to divide into logical groupings
- A naming encapsulation is used to create a new scope for names
- C++ Namespaces
 - Can place each library in its own namespace and qualify names used outside with the namespace
 - C# also includes namespaces

1-

Naming Encapsulations (continued)

- Java Packages
 - Packages can contain more than one class definition; classes in a package are partial friends
 - Clients of a package can use fully qualified name or use the import declaration
- Ada Packages
 - Packages are defined in hierarchies which correspond to file hierarchies
 - Visibility from a program unit is gained with the with clause

1-

Summary

- The concept of ADTs and their use in program design was a milestone in the development of languages
- Two primary features of ADTs are the packaging of data with their associated operations and information hiding
- Ada provides packages that simulate ADTs
- C++ data abstraction is provided by classes
- Java's data abstraction is similar to C++
- Ada and C++ allow parameterized ADTs
- C++, C#, Java, and Ada provide naming encapsulation