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.
Introduction to Data Abstraction

An abstract data type is a user-defined data type that satisfies the following two conditions:

1. The representation of, and operations on, objects of the data type are defined in a single syntactic unit.

2. The representation of objects of the data type is hidden from the program units that use these objects, so the only operations possible are those provided in the type's definition.

Advantages of Data Abstraction

Advantages of the first condition:

1. Better program organization
2. Improved modifiability (everything associated with a data structure is together)
3. Separate compilation is possible

Advantage of the second condition:

Reliability—by hiding the data representations, user code cannot directly access objects of the data type, which depends on the representation, allowing the representation to be changed without affecting user code.
**Design Issues**

1. A syntactic unit to define an ADT

2. Built-in operations
   - Assignment
   - Comparison

3. Common operations
   - Iterators
   - Accessors
   - Constructors
   - Destructors

4. Parameterized ADTs

---

**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 the data type appears in a part of the specification called the **private** part
  - There is a 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
An Example in Ada

package Stack_Pack is

  type Stack_Type is limited private;  -- visible entities
  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

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
Language Examples C++

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

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
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 (~)

An Example in C++

```cpp
class stack {

private:
    int *stackPtr;
    int maxLen;
    int topPtr;

public:
    stack() { // a constructor
        stackPtr = new int [100];
        maxLen = 99;
        topPtr = -1;
    }
    ~stack() {delete [] stackPtr;} // a destructor
    void push(int num) {...};
    void pop() {...};
    int top() {...};
    int empty() {...};
};
```
Evaluation of ADTs in C++ and Ada

Support for ADTs in C++ is similar to expressive power of Ada

Both provide effective mechanisms for encapsulation and information hiding

Ada packages are more general encapsulations

Language Examples C++

Friend functions or classes

Allows unrelated units or functions to have access to private members

Necessary in C++
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 clauses
  – 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

An Example in Java

class StackClass {
  private int [] *stackRef;
  private int maxLen;
  private int 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 () {...};
}
Language Examples C#

- Based on C++ and Java
  - Adds two access modifiers
    - `internal`
    - `protected internal`

- All class instances are heap dynamic
  - Default `constructors` made for all classes
  - Garbage collection used for most heap objects
    - `destructors` are rarely used

- Classes made with `struct` are lightweight classes that do not support inheritance

Language Examples C#

- Common solution to need for access to data members
  - accessor methods
    - `getter`
    - `setter`

- C# provides `properties` as a way of implementing getters and setters without requiring explicit method calls
C# Property Example

```csharp
public class Weather {
    public int DegreeDays {
        get {return degreeDays;}
        set {degreeDays = value;}
    }
    private int degreeDays;
    ...
    }

Weather w = new Weather();
int degreeDaysToday, oldDegreeDays; ...
w.DegreeDays = degreeDaysToday; //** DegreeDays.set ...
oldDegreeDays = w.DegreeDays; //** DegreeDays.get
```

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
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

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
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
C# Assemblies

- A collection of files that appear to be a single dynamic link library or executable
- Each file contains a module that can be separately compiled
- A DLL is a collection of classes and methods that are individually linked to an executing program
- C# has an access modifier called **internal**; an internal member of a class is visible to all classes in the assembly in which it appears

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
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

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