5 Abstract Data Types

- A data type consists of a collection of values together with a set of basic operations defined on the values
- A data type is an **Abstract Data Type (ADT)** if programmers using the type **do not** have access to the details of how the values and operations are implemented

5.1 Classes To Produce ADTs

- To define a class so it is an **ADT**
  - Separate the specification of how the type is used by a programmer from the details of how the type is implemented
  - Make all member variables private members
  - Basic operations a programmer needs should be public member functions
  - Fully specify how to use each public function
  - Helper functions should be private members

5.1.1 ADT Interface

- The ADT interface tells how to use the ADT in a program
  - The interface consists of
    - The public member functions
    - The comments that explain how to use the functions
  - The interface should be all that is needed to know how to use the ADT in a program

5.1.2 ADT Implementation

- The ADT implementation tells how the interface is realized in C++
  - The implementation consists of
    - The private members of the class
    - The definitions of public and private member functions

5.2 ADT Benefits

- Changing an ADT implementation does not require changing a program that uses the ADT
- ADT's make it easier to divide work among different programmers
  - One or more can write the ADT
  - One or more can write code that uses the ADT
- **Writing and using ADTs breaks the larger programming task into smaller tasks**

5.3 Interface Preservation

- To preserve the interface of an ADT so that programs using it do not need to be changed
  - Public member **declarations** cannot be changed
  - Public member **definitions** can be changed
  - Private member functions can be added, deleted,or changed

5.4 Information Hiding

- Information hiding was referred to earlier as writing functions so they can be used like black boxes
- ADT's implement information hiding because
  - The interface is all that is needed to use the ADT
  - Implementation details of the ADT are not needed to know how to use the ADT
  - Implementation details of the data values are not needed to know how to use the ADT

5.5 Program Example : The BankAccount ADT

- **BankAccount** class in **sample26.cpp** is ADT
- Here we change this ADT implementation as:
  - Data is stored as three member variables
    - The dollars part of the account balance
    - The cents part of the account balance
    - The interest rate
  - The interest rate is stored as a fraction
  - The public portion of the class definition remains unchanged from **sample26.cpp**

5.5.1 sample27.cpp

```
// class definition
class BankAccount
{
public:
    BankAccount(int dollars, int cents, double rate);
    // Postcondition: The account balance has been set $dollars.cents;
    // The interest rate has been set to rate percent.

    BankAccount(int dollars, double rate);
    // Postcondition: The account balance has benn set to $dollars.00.
    // The interest rate has been set to rate percent.

    BankAccount(); // Default constructor
    // Initializes the account balance to $0.00 and the interest rate to 0.0%

    void update();
    // Postcodition: One year of simple interest has been
    // added to the account balance.

    double get_balance();
    // Returns the current account balance.

    double get_rate();
    // Returns the current account interest rate as a percentage.

    void output(std::ostream& outs);
    // Precondition: If outs is a file output stream, outs has already
    // been connected to a file.
    // Postcondition: Account balance and interest rate have been
    // written to the stream outs.

private:
    int dollar_part;     // dollar
    int cents_part;      // cent
    double interest_rate; // fraction

    double fraction(double percent);
    // Converts a percentage to a fraction.

    double percent(double fraction_value);
    // Converts a fraction to a percentage
};
```