CMPT 225

Lecture 2 – Abstract Data Type (ADT)
Last Lecture

- What is CMPT 225 all about?
- Course resources
- Important points!
- Activity <- Thank you!
  - Feedback:
    - Well done! 😊
    - You should be able to answer questions: 1, 2, 4, 5, 6, 9
    - This semester, we will look into questions: 3, 7, 8, 10
      - So, don’t worry if you had difficulty answering these questions
    - Suggestion: Read questions carefully!
Learning Outcomes

- At the end of this lecture, a student will be able to:
  - Define abstraction, information hiding and "abstract data type" (ADT)
  - Write C++ code
  - Encapsulate methods and variables for an ADT into a C++ class
  - Differentiate between a class that has been designed as an ADT and a class that has not
  - Compare and contrast them
Today’s menu

- Abstract Data Type (ADT)
  - Definition + “Wall” metaphor
  - How to design an ADT
  - How to implement an ADT in C++
  - How to test an ADT
- Example: Temperature class
  - Implemented as an ADT
  - Implemented as a non-ADT
- Compare both implementations
Let’s start with a problem

- Problem Statement
  - Create a temperature conversion application
- Step 1 – Understanding the problem
  - What do we do in this step?
- Step 2 – Design Solution
- Step 3 – Implement Solution
- Step 4 – Testing solution
Why is Step 1 is important?
Step 2 – Design solution

UML Class Diagram

Exclude constructors, destructors, getters/setters. Why? Because we assume that we will develop these methods for our classes, if they are needed.

Temperature

- degrees
- scale
- inFahrenheit()
- inCelsius()
- raise( toWhat )
Step 3 – Implement solution

/* Header Comment Block */

/* Class Definition */
class Temperature {

private:

double myDegrees; // >= ABSOLUTE_ZERO for myScale
char myScale; // 'F' or 'C'

bool isValidTemperature( const double degrees,
const char scale );

public:
};
Step 3 – Implement solution – cont’d

... public:

// Constructors
Temperature();
Temperature(double degrees, char scale );

// Getters
double getDegrees( ) const;
char getScale( ) const;

// Setters
void setDegrees( const double degrees );

// Application-related methods
Temperature inFahrenheit( ) const;
Temperature inCelsius( ) const;
void raise( const double amount );

};
Step 3 – Implement solution

/* Header Comment Block */

Temperature::Temperature() : myDegrees(0.0), myScale('C') {} ...

Temperature Temperature::inFahrenheit() const {
    Temperature result;
    if ( myScale == 'F' )
        result = Temperature( myDegrees, 'F' );
    else if ( myScale == 'C' )
        result = Temperature( myDegrees * 1.8 + 32.0, 'F' );
    return result;
}

Implementation file: Temperature.cpp
/* Header Comment Block */

int main() {
    // Create a valid Celsius temperature
    // Create an invalid Celsius temperature
    // Create a valid Fahrenheit temperature
    // Create an invalid Fahrenheit temperature
    // Converting a valid Celsius temperature
    // to a Fahrenheit temperature
    // Raising a valid Celsius temperature to
    // an invalid amount of degrees
    ...
    return 0;
}
What makes this class an ADT?

- But what is an ADT (abstract data type)?
Abstraction – in the real world

- Abstraction
  - From the Latin *abs*, meaning *away from*
  - and *trahere*, meaning *to draw*
- “Process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics”  
  Source: https://whatis.techtarget.com/definition/abstraction
- Examples:
  - Car
  - Map
Abstraction and information hiding – in the software world

- We achieve abstraction by hiding information from “public view” -> from other classes (i.e., client code)
- We separate the purpose (what) of a class from its implementation (how) and hide the latter (i.e., class’ attributes and the implementation of its methods)
- This way, client code can use the class without knowing its implementation (only knowing the class’ interface)
- This reduces complexity and allows for easy modification
- And this is how we construct an ADT
When Temperature class implemented as an ADT

- Temperature’s “private” section hidden behind the wall
- Client code must use Temperature’s public methods (slits in the wall) to access Temperature’s private section
Why should we hide a class’ implementation?

/* Header Comment Block */

/* Class Definition */
class Temperature {

public:
    double myDegrees; // >= ABSOLUTE_ZERO for myScale
    char myScale; // 'F' or 'C'

...
Why should we hide a class’ implementation?
cont’d

...  

// Constructors
Temperature();
Temperature(double degrees, char scale);

// Application-related methods
Temperature inFahrenheit() const;
Temperature inCelsius() const;
void raise( const double amount );
bool isValidTemperature( const double degrees,
const char scale );
};
Step 4 – Test solution - Demo

- Create a valid Fahrenheit temperature -> testing Temperature( 32.0, 'F' )
  Actual Result: 'tempFahr' -> 32 degree F

- Changing the amount of degrees of 'tempFahr' to -976.02F by setting 'myDegrees' to -976.02 directly.
  Actual Result: 'tempFahr' -> -976.02 degree F
When Temperature class **not** implemented as an ADT

- Ensure no client code can tamper with the class’ hidden (private) sections
  - In our Temperature class example, client code can break Temperature class’ invariant

```
Temperature Converter.cpp (Client Code)
```

```
Temperature class
myDegrees
myScale
```
Advantages and disadvantages of ADT

- Advantages:
  - Preserve the invariant of a class

- Disadvantages:
  - Must have getters/setters methods
We can now ...

- describe what happens in the 4 steps of the software development process:
  - Step 1 - Problem statement
  - Step 2 – Design
  - Step 3 – Implementation
  - Step 4 - Testing
- construct an ADT class in C++
- define abstract data type (ADT), abstraction and information hiding
- differentiate between a class that has been designed/implemented as an ADT and a class that has not
- list some of the advantages and disadvantages of ADT
Next Lecture

- Introduce our first data structure: List
- Design a List class as an ADT
- Implement it using an array