|  |  |  |
| --- | --- | --- |
| |  | | --- | | Lab01 Object-Oriented Programming | |  | |
| Exercise 1: UML Class Diagram (15 min) |
| A definition of class Cat is shown below:  class Cat {  string name;  int age;  public:  string nickname;  private:  void reset ();  public:  void setName (string dname);  void setAge (int dage);  string getName ();  int getAge ();  };   1. Identify the following items in the Cat class:    1. Attributes:    2. Behaviours/Methods:    3. Private members:    4. Public members: 2. Draw an UML Class Diagram for the Cat class. |
| Exercise 2: Composition (15 min) |
| Study the following incomplete program.  class Point {  int x;  int y;  public:  void setX (int x);  void setY (int y);  int getX();  int getY();  double distance (Point p);  };  class Triangle {  Point p[3];  double perimeter;  void calcPerimeter();  public:  void set3Points();  void display();  };  int main() {  Triangle t;  t.set3Points();  t.display();  }  Expected output:  Enter 3 points in the following format:  x1 y1 x2 y2 x3 y3  1 1 1 4 5 1  Points:  (1,1)  (1,4)  (5,1)  Perimeter = 12  Identify the following items in Triangle class:   1. Class attributes: 2. Class behaviors: 3. Private members: 4. Public members: 5. Object/Instance of Point class: 6. Is there a relationship/association between Triangle class and Point class? Name it if your answer is yes. 7. Draw a UML Class Diagram based on the 2 classes in the program. |
| Exercise 3: Separating Interface from Implementation in One File (30 min) |
| Implement the Point class and Triangle class outside of its declaration. Ensure that the main function work and produce the expected output. |
| Exercise 4: Separating Interface from Implementation into Multiple Files (40 min) |
| To fully comply with encapsulation, we shall split our program in Exercise 3 into 5 files:   * 1. Point.hpp header file – contain Point class interface.   2. Point.cpp source file – contain Point class implementation.   3. Triangle.hpp header file – contain Triangle class interface.   4. Triangle.cpp source file – contain Triangle class implementation.   5. Main.cpp source file – contain the main function that uses Point class and Triangle class.  1. First create Point.hpp and Point.cpp, and then use the following g++ command line to compile Point.cpp to see there is any compile error (the switch -c is to inform the compiler that the Point.cpp does not have a main function). Correct if there is error. File Point.o will be produced if the compilation is successful with -c.   g++ -c Point.cpp   1. Second create Triangle.hpp and Triangle.cpp, and then use the following g++ command line to compile Triangle.cpp. Correct if there is error.   g++ -c Point.cpp Triangle.cpp   1. Lastly, create main.cpp, and then use the following g++ command line to compile main.cpp. Correct if there is error.   g++ Point.cpp Triangle.cpp main.cpp   1. To run the program, type a.exe.   Next is to see how separation of interface and implementation enables class user use a class without knowing the implementation.   1. Move the Point.cpp and Triangle.cpp away from the current folder. Now we have only the following 5 files in the current folder:    1. Point.hpp    2. Point.o    3. Triangle.hpp    4. Triangle.o    5. main.cpp 2. Use NotePad to open Point.o or Triangle.o. An .o file is called object file. They contain the binary version of the implementation and are not human-readable. 3. Re-compile the program with the following g++ command:   g++ Point.o Triangle.o main.cpp   1. Run a.exe to confirm that the program still work. As you can see, class user does not need to know the implementation (Point.cpp and Triangle.cpp) to use the classes. |
| Exercise 5: Using Code::Blocks to Compile Multiple Files (10 min) |
| To compile header files and sources using Code::Blocks, we need to create a project and include the files into the project.   1. Launch Code::Blocks. 2. Create a new project, select Empty project. Click Go/OK. 3. Type **TriangleProject** for the Project Title field, select Desktop as the project folder. Click OK. Desktop\TriangleProject folder is created. 4. Launch Windows Explorer, copy all 5 files (Point.hpp, Point.cpp, Triangle.hpp, Triangle.cpp, main.cpp) into the Desktop\TriangleProject folder. 5. In Code::Blocks workspace, right-click TriangleProject, select Add files. Select the 3 files. 6. Click OK for both Debug and Release build. 7. Select Build menu > Build and run. |

|  |
| --- |
| **Take Home Exercises** |

* 1. Implement a class Address. An address has a house number, a street, an optional apartment/house number, a city, a state, and a postal code. Supply two constructors: one with an apartment number and one without. Supply a print function that prints the address with the street on one line and the city, state and postal code on the next line. Supply a member function comes\_before that tests whether one address comes before another when the addresses are compared by postal code.
  2. Computer games often contain different characters or creatures. For example, you might design a game in which alien beings possess specific characteristics such as color, number of eyes, or number of lives. Design a character for a game, creating a class to hold at least three attributes for each character. Include methods to get and set each of the character’s attributes. Then write a driver program in which you create at least two characters each of which has a random age up to 100, a random number of eyes up to 10, and a random number of legs, up to 12. In turn, pass each character to a display method that displays the character’s attributes.
  3. Define a class named GroceryItem. Include private fields that hold an item’s stock number, price, quantity in stock, and total value. Write a public function named dataEntry that calls four private functions. Three of the private functions prompt the user for keyboard input for a value for one of the data fields stock number, price, and quantity in stock. The function that sets the stock number requires the user to enter a value between 1000 and 9999 inclusive; continue to prompt the user until a valid stock number is entered. The functions that set the price and quantity in stock require non-negative values; continue to prompt the user until valid values are entered. Include a fourth private function that calculates the GroceryItem’s total value field (price times quantity in stock). Write a public function that displays a GroceryItem’s values. Write a driver program that declares an array of 10 GroceryItem objects. Assign values to all 10 items and display them.