COEN 244 (Fall 2024) - Assignment 3 Description

Problem One Statement – Polymorphism, Virtual Function, Override and Final

The context of the problem statement is adopted from the textbook exercises 11.9 and 12.12 and presented below.

11.9 ***(`Package` Inheritance Hierarchy)*** Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages. Use class `Package` as the base class of the hierarchy, then include classes `TwoDayPackage` and `OvernightPackage` that derive from `Package`.

Base-class `Package` should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. `Package`'s constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values. `Package` should provide a `public` member function `calculateCost` that returns a `double` indicating the cost associated with shipping the package. `Package`'s `calculateCost` function should determine the cost by multiplying the weight by the cost per ounce.

Derived-class `TwoDayPackage` should inherit the functionality of base-class `Package`, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. `TwoDayPackage`'s constructor should receive a value to initialize this data member. `TwoDayPackage` should redefine member function `calculateCost` so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base-class `Package`'s `calculateCost` function.

Class `OvernightPackage` should inherit directly from class `Package` and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. `OvernightPackage` should redefine member function `calculateCost` so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost.

Task 1.1 Design and draw the hierarchy of the inheritance of this package system in UML diagrams to represent the specification in the context. Annotate at the member function level and/or class level the "virtual, override and final" keywords as part of the signatures.

Task 1.2 Develop programs according to the design in Task 1.1 for each class to meet the specification in the context. (Kind reminder : virtual destructor should be applied properly.)

Task 1.3 Program the testDriver.cpp with the main() function to call testStaticBinding() and testDynamicBinding().

(a) Write the unit test function for static binding with objects created for each type of `Package` and tests member function `calculateCost`.

```
void testStaticBinding()
```

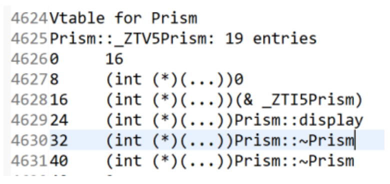
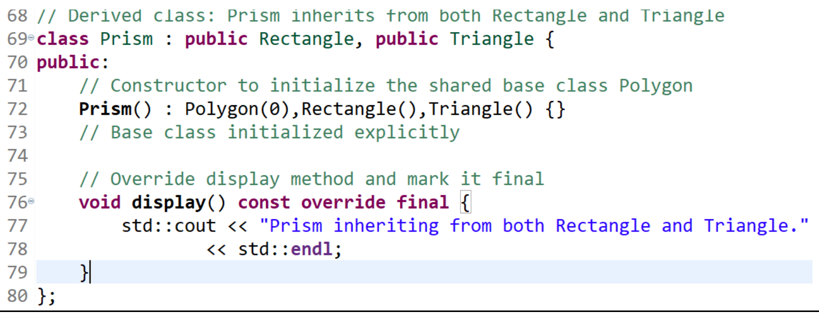
(b) Write the unit test function for polymorphism using dynamic binding. The case should contain an array or a vector of Package pointer to ten objects of classes TwoDayPackage (five objects) and OvernightPackage (five objects). Loop through the vector to process the Packages polymorphically. For each Package, invoke *get* functions to obtain the address information of the sender and the recipient, then print the two addresses as they would appear on mailing labels. Also, call each Package's calculateCost member function and print the result. Keep track of the total shipping cost for all Packages in the vector, and display this total when the loop terminates.

```
void testDynamicBinding()
```

(c) Write the testDriver.cpp file with the main() to invoke test functions of (a) and (b).

Task 1.4 Produce the vtable for each class and observe the virtual function entries of each class declared in the above context. Based on the vtable artifacts, fill in the template for your program.

(*The vtable concept is addressed in the text book 12.8. The technique of producing vtable was demonstrated during the lecture time and references were documented in the week 7's summary and references.*)

vtable entry for class X with virtual functions (text or screenshot)	program code with virtual functions of class X (text or screenshot)
For example 	For example 

```
Vtable for Prism
Prism::_ZTV5Prism: 19 entries
0     16
8     (int (*)(...))0
16    (int (*)(...))(& _ZTI5Prism)
24    (int (*)(...))Prism::display
32    (int (*)(...))Prism::~Prism
40    (int (*)(...))Prism::~Prism
```

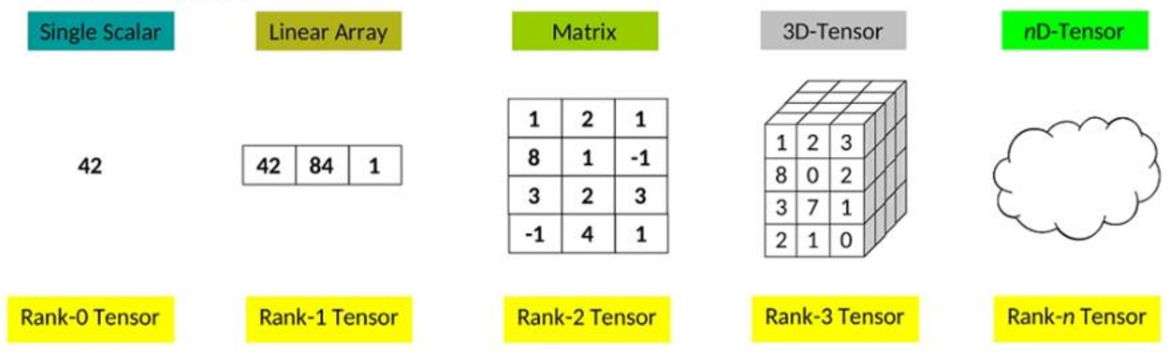
```
// Derived class: Prism inherits from both Rectangle and Triangle
class Prism : public Rectangle, public Triangle {
public:
    // Constructor to initialize the shared base class Polygon
    Prism() : Polygon(0),Rectangle(),Triangle() {}
    // Base class initialized explicitly

    // Override display method and mark it final
    void display() const override final {
        std::cout << "Prism inheriting from both Rectangle and Triangle."
                  << std::endl;
    }
};
```

Task 1.5 Write a report in PDF to present the Task 1.1, and Task 1.4.

Problem Two Statement – Polymorphism, Abstract Class and Operator Overloading

The context of problem two is about tensors. Tensors are data structures that generalize the concepts of vectors and matrices for an arbitrary number of dimensions. The number of different dimensions of a tensor is also called tensor rank.



```
#ifndef BASETENSOR_H_
#define BASETENSOR_H_

#include <iostream>
#include <vector>

// Base class for all tensors
class BaseTensor {
public:

      /**
       * Virtual method for load data to a tensor;
       * Given the dimensions are different for each type
       * of tensor, the loadData() function to initialize
       * the values of a tensor should overriden by
       * derived tensor classes;
       */
    virtual void loadData() = 0;

    /**
     * virtual for valueGen() is optional;
     * valueGen() can use random library to generate values
     * to initialize the tensor values;
     */
   double valueGen(){
       std::random_device rd;
```

```
        std::mt19937 mt(rd());

         // Create a uniform distribution between 0 and 1
         std::uniform_real_distribution<double> dist(0.0, 1.0);

        // Generate and print a random double value
        double randomValue = dist(mt);
        return randomValue;
   }
};

#endif /* BASETENSOR_H_ */
```

Task 2.1 Please program two derived class `RankOneTensor` and class `RankTwoTensor` from the base class `BaseTensor.` The virtual methods should be overridden by each derived class according to the tensor type.

Class `RankOneTensor` includes the private data member to hold the values for the tensor. The constructors/destructor definition are required.

```
Private:
    std::vector<double> data;

public:

    RankOneTensor();
    RankOneTensor(int size); // dimension size;
    RankOneTensor(const RankOneTensor& other);
    ~RankOneTensor();
```

class `RankTwoTensor` includes the private data member to hold the values for the tensor. The constructors/destructor definition are required.

```
Private:
    std::vector<std::vector<double>> data;

public:
     RankTwoTensor(int rows, int cols);
     RankTwoTensor(const RankTwoTensor& other);
     ~ RankTwoTensor();
```

Task 2.2 Define and program the following operators for both class `RankTwoTensor` and class `RankTwoTensor`.

Task 2.3 Program unit test functions listed in the table below in testDriver.cpp. Call these test functions in the main() function of testDrive.cpp.

Operator	Semantic Meaning	Example and Unit Test Function

++	Increment each element’s value by 1	RankOneTensor t (3); t++; ++t;
		
		`void testIncrementOperatorRankOne();`
		`void testIncrementOperatorRankTwo();`

+	Add two tensors of the same dimension element wise; *invalid_argument should be thrown if two tensor objects’ dimension does not match*	RankOneTensor t1(3), t2(3) ;
		t1 + t2 ;
		
		`void testAddOperatorRankOne();`
		`void testAddOperatorRankTwo();`

+	Add one tensor with a scalar value element wise	RankOneTensor t(3) + 0.5;
		
		`void testAddOperatorRankOne();`
		`void testAddOperatorRankTwo();`

=	Assign one tensor to another of the same dimension; *invalid_argument should be thrown if two tensor objects’ dimension does not match*	RankOneTensor t1(2), t2(2), t3(2) ;
		t1.loadData() ;
		t2.loadData() ;
		t3 = t2 + t3 ;
		
		`void testAssignmentOperatorRankOne();`
		`void testAssignmentOperatorRankTwo();`

>>	Output the tensor’s element values	RankOneTensor t(3) ;
		t.loadData() ;
		cout << t ;
		
		`void testCoutStreamOperatorRankOne();`
		`void testCoutStreamOperatorRankTwo();`

<<	Input the tensor’s element values by console input	RankOneTensor t(3);
		cin >> t;
		
		`void testCinStreamOperatorRankOne();`
		`void testCinStreamOperatorRankTwo();`

[ ]	Retrieve value by index; *invalid_argument should be thrown if the index is out of the range*	RankTwoTensor t(2,2);
		cin >> t;
		cout <<t[0][1];
		
		`void testIndexOperatorRankOne();`
		`void testIndexOperatorRankTwo();`

Submission

In the report, please include task 1.1 and task 1.4 and attach the backlog below with the team's contribution filled. The submission should include all the source code, the vtable file (.class) and the report.pdf.

Create **[SID_1]_[SID_2]_A3.zip** (.gz, .tar, .zip are acceptable. .rar file is NOT acceptable) file contains a folder named with your student IDs.

The submission must be through Moodle on the submission link for Programming Assignment 3. Email submissions will NOT be accepted. All cases of plagiarism will be reported to authorities as per Concordia's plagiarism policy.

Task (1.1 – 2.3, 10 marks each task)	Team members' development contribution
Task 1.1	
Task 1.2	
Task 1.3 (a)	
Task 1.3 (b)	
Task 1.3 (c)	
Task 1.4	
Task 1.5	
Task 2.1 `RankOneTensor`	
Task 2.1 `RankTwoTensor`	
Task 2.2 Operator ++	
Task 2.2 Operator +	
Task 2.2 Operator =	
Task 2.2 Operator >>	
Task 2.2 Operator <<	
Task 2.2 Operator [ ]	
Task 2.2 `void testIncrementOperatorRankOne();`	
Task 2.2 `void testIncrementOperatorRankTwo();`	

Task 2.2 `void testAddOperatorRankOne();`	
Task 2.2 `void testAddOperatorRankTwo();`	
Task 2.2 `void testAssignmentOperatorRankOne();`	
Task 2.2 `void testAssignmentOperatorRankTwo();`	
Task 2.2 `void testCoutStreamOperatorRankOne();`	
Task 2.2 `void testCoutStreamOperatorRankTwo();`	
Task 2.2 `void testCinStreamOperatorRankOne();`	
Task 2.2 `void testCinStreamOperatorRankTwo();`	
Task 2.2 `void testIndexOperatorRankOne();`	
Task 2.2 `void testIndexOperatorRankTwo();`	
Task 2.3	

Assignment Marking Rubrics:

Level	Task 1.1, 1.2, 1.3, 1.4 (10 marks each); Task 2.2 operator (10 marks each); Task 2.2 unit test (10 marks each)
5	All requirement addressed, programs run without errors in validation testing.
4	Missing 1 requirements, the rest of the function runs without errors.
3	Missing 2 requirements, the rest of the function runs without errors.
2	Missing more than 3 requirements, the program didn’t run properly.
1	Less than 2 requirements addressed, the program didn’t run properly.
0	The program is irrelevant to the solution