INDIAN ASSOCIATION FOR THE CULTIVATION OF SCIENCE

SCHOOL OF MATHEMATICAL & COMPUTATIONAL SCIENCES

PG - I (SMCS), Autumn Semester 2022-23

Instructor: **Debarshi Kumar Sanyal** Assignment Number: **6**

Course: **COM 4111: OOP with C++** Date: **(Nov 7,14,15), 2022**

Problem 1 Declare an abstract class `Shape` and derive classes `Rectangle`, `Circle`, and `Triangle` from it. The classes should have member functions to compute the area and perimeter. Implement a function `void compute(Shape*)` that takes a pointer of type `Shape*` and prints the area and perimeter of the object that the pointer points to. [`Shape.cpp`]

Problem 2 Suppose you have a class `ShapeCreator` that has a public method `Shape* create(string s);` where `s` is the name of the shape and can be one of `rectangle`, `circle`, and `triangle`. Depending on the shape name `s`, `create(s)` constructs a concrete shape as described in **Problem 1** and returns it. If the shape name is different from the aforementioned names, a `ShapeNotSupported` exception is thrown. Write a function that takes a shape name from the user and calls the function `compute(Shape*)` described in **Problem 1**. [`ShapeCreator.cpp`]

Problem 3 Consider an abstract class `Icon` that has a member function `Icon* clone()` which copies the current object and returns it. Suppose you have several sub-classes of `Icon`, e.g., `MenuIcon`, `SearchIcon`, `OpenIcon`, `CloseIcon`. You have another class `IconStore` that contains a list of icons. For example, the element 0 of the list is a menu icon, element 1 is a search icon, etc. Now suppose you have a client that requests an icon from the `IconStore` by specifying an icon number. The `IconStore` should clone the appropriate icon (making a call like `iconList[i]->clone()` where `i` is the input icon number) and return it to the client. Implement the appropriate classes in C++. [`IconStore.cpp`]

Problem 4 Consider an abstract class `Image` from which you derive two sub-classes: `RealImage` that contains an image file name, height and width of the image, and an appropriate matrix whose entries are integers between 0 and 255 filled (on demand) from the image file, and another class `ProxyImage` that contains the image file name, height and width of the image, and a pointer to an object of `RealImage`. A client creates an instance of `RealImage` and uses it to instantiate `ProxyImage`. When the client asks the `ProxyImage` for the height and/or width of the image, the proxy can answer but when the client asks for the image matrix or tries to perform some operation on the image matrix, the request is delegated to the instance of `RealImage`. Model these classes in C++. [`ProxyImage.cpp`]

Problem 5 Suppose you have 4 classes: `AI`, `Neural`, `Symbolic`, and `NeuroSymbolic` where the second and third classes are derived from the first class, and the fourth class is a sub-class of the second and the third classes. The class `Symbolic` has a member function `reason()` while the class `Neural` has a member function `train`. The class `AI` has a public data member `bool usesKG`. Design the above classes in C++. Do your classes support the following client code?

```
int main() {
  NeuroSymbolic* ai_ns = new NeuroSymbolic();
  ai_ns->reason();
  AI* ai = ai_ns;
  bool b = ai->usesKG;
}
```

[`MultipleInheritance.cpp`]

Problem 6 Suppose you have a client as follows:

```
void printArea(Figure* f) {
  cout<<"Area = "<<f->area();
}
int main() {
  Figure* f1 = new RectangularFigure();
  Figure* f2 = new CircularFigure();
  printArea(f1);
  printArea(f2);
}
```

Now recollect that you already have a `Shape` class hierarchy as described in **Problem 1**. How will you reuse this existing class hierarchy to support your new client code? [`Adapter.cpp`]

Problem 7 Suppose you want to model a sentence using the class `Sentence` and a paragraph using the class `Paragraph` in C++. A paragraph is a sequence of sentences. There is a member function `print()` in each of the classes: it is used to print a sentence and when called on a paragraph, it prints the constituent sentences in order. Each class has also has a member function `changeCase(toCase)` that is used to change the case of a `Sentence` object or to change the case of all sentences in a

`Paragraph` object (depending upon the class whose member it is). Thus, the two classes share many member functions but their functionalities differ reflecting a part-whole relationship. Design a suitable class structure to model the above scenario. [`Composite.cpp`]

Problem 8 Design a `Stack` class in C++ with exceptions like `Overflow` and `Underflow`. [`Stack.cpp`]

Problem 9 Design a templatized class `Vector` that has private members `T* elem` and `int size`. One of the constructors should take the size of the vector as an argument; if the size is negative, it should throw an exception `NegativeSize`. The class should have public member functions `T& operator[](int i)`, `const T& operator[](int i) const`, `int getSize() const`, `void print() const`, etc. The member functions must throw appropriate exceptions when required. [`TemplateVector.cpp`]

Problem 10 Write a function template to sort an array of objects of type `T` using selection sort algorithm. Use it to sort an array of `int`s, an array of `char`s, an array of strings, and an array of student records according to their aggregate marks where a student record is an instance of a structure that contains at least the student's roll number, name, and aggregate marks. [`Visitor.cpp`]

Problem 11 Suppose you are designing various commands for an application. You decide to model a command using an abstract class `Command`, from which you will derive concrete classes such as `DisplayFileCommand` (to display the contents of a text file), `HelpCommand` (to display help message), and `ShowTimeCommand` (to display the current time). Every command has a member function `execute()` that performs the desired operation. Every command also has a member function `log()` that writes to a shared log file that it has been called. A client using the application will create an instance of a concrete sub-class of `Command` to perform the desired operation. For example, the client may write code like the following:

```
int main() {
  Command* c1 = new ShowTimeCommand();
  c1->execute();
  c1 = new DisplayFileCommand("./streetnames.txt");
  c1->execute();
  c1 = new HelpCommand();
  c1->execute();
  c1 = new ShowTimeCommand();
  c1->execute();
  //Now display the shared log file.
  //It should display the commands in the order they were called.
}
```

[`Command.cpp`]

Problem 12

1. Suppose you have a class hierarchy where the base class is `Member` and its (publicly derived) sub-classes are `PremiumMember` and `OrdinaryMember`. It models the user base of a recreation club. Some of the `protected` data members of `Member` are `membershipId`, `name`, and `joiningDate`. Each sub-class has a static constant data member `monthlyCharges`. Some of the `public` member functions of `Member` are `getName()`, `getMembershipId()`, `displayAnnualCharges()` and `displayDetails()`. Design the class hierarchy in C++.

2. Suppose you want to add new operations to each sub-class. But you are not allowed to make too many changes in the above class hierarchy. So you model the new operations using a new class hierarchy whose abstract base class is `Visitor`. A `Visitor` has pure virtual methods `void visit(PremiumMember* m)` and `void visit(OrdinaryMember* m)`. The `visit()` method performs some operation on the object passed to it as parameter.

```
class Visitor {
    //...
  public:
    //...
    virtual void visit(PremiumMember* pm) = 0;
    virtual void visit(OrdinaryMember* om) = 0;
};
```

Now suppose you have a subclass `InvitationVisitor` in which you override the above pure virtual methods.

```
class InvitationVisitor {
    //...
  public:
    //...
    virtual void visit(PremiumMember* pm) {
      cout<<"Hello "<<pm->getName()<<", We have a special holiday package
      for you! A one-week tour of the Big Island of Hawaii with a unique
      5-day cruise itinerary!";
    }
    virtual void visit(OrdinaryMember* om) {
      cout<<"Hello "<<om->getName()<<", Pack your bags for the Andaman Islands.
      We have a special 2-day tour package for you!";
    }
};
```

A new member function is also added to the class `Member`:

```
class Member {
    //...
  public:
    //...
```

```
        void acceptVisitor(Visitor* v){
          v->visit(this);
        }
    };
```

The client may be written as follow:

```
int main() {
  Member* m = new PremiumMember(1000, "Suresh", "05-03-2022");
  Visitor* visitor = new InvitationVisitor();
  m->accept(visitor); //should print out the invitation!
  //...
}
```

You may declare other concrete sub-classes of `Visitor` to encapsulate other operations, and then create their instances. You may use them to perform specific operations on an instance of `PremiumMember` or `OrdinaryMember`.

[`Visitor.cpp`]