|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **National University of Computer and Emerging Sciences, Lahore Campus** | | | | |
| C:\Users\saif\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\final design.jpg | **Course:** | **Software Design and Analysis** | **Course Code:** | **CS-3004** |
| **Program:** | **BS (Computer Science)** | **Semester:** | **Fall 2024** |
| **Duration:** | **40 Minutes** | **Total Marks:** | **20** |
| **Quiz Date:** | **27-Nov-24** | **Roll No.** |  |
| **Section:** | **BCS-5C** | **Name:** |  |
|  |  |  |  |
|  |  | | | |

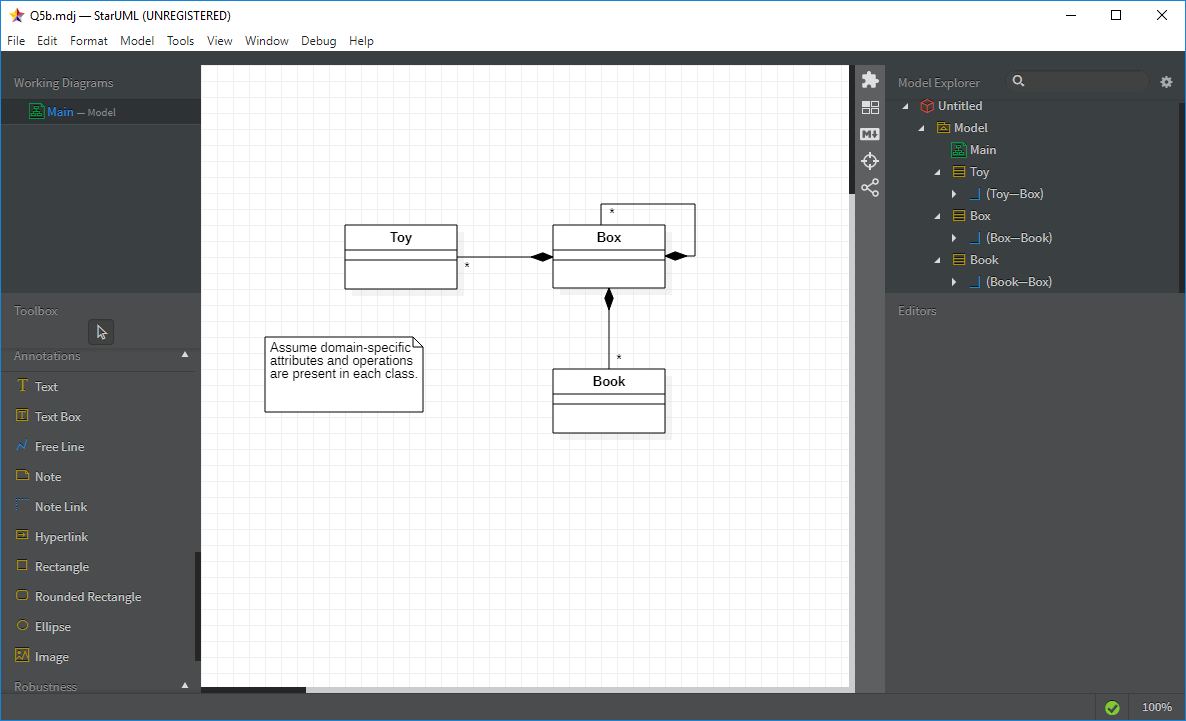
**Question 1)** You have two types of data processors: XMLProcessor and TextProcessor. Both processors have same implementation for methods **readData** and **saveData**, but they have a different implementation for **processing** the data. Identify the design pattern that could be applied

You are required to refactor the following code to enforce the workflow steps that each processor first reads the data, then processes the data and finally saves the data **(10 Marks)**

|  |  |
| --- | --- |
| // XML Processor  class XMLProcessor {  public:  void readData() {  cout << "Reading data" << endl;  }  void processXMLData() {  cout << "Processing XML data" << endl;  }  void saveData() {  cout << "Saving data" << endl;  }  };  // Text Processor  class TextProcessor {  public:  void readData() {  cout << "Reading data" << endl;  }  void processTextData() {  cout << "Processing Text data" << endl;  }  void saveData() {  cout << "Saving data" << endl;  }  }; | **Solution: Template Method Pattern**  // Abstract base class  class DataProcessor {  public:  // Template method defining the sequence of operations  void process() {  readData();  processData();  saveData();  }  protected:  // Common step with default implementation  void readData() {  cout << "Reading data" << endl;  }  // Common step with default implementation  void saveData() {  cout << "Saving data" << endl;  }  // Pure virtual function for the unique step that each subclass will implement  virtual void processData() = 0;  };  // XML Processor subclass  class XMLProcessor : public DataProcessor {  protected:  // Implement the unique processing for XML  void processData() override {  cout << "Processing XML data" << endl;  }  };  // Text Processor subclass  class TextProcessor : public DataProcessor {  protected:  // Implement the unique processing for Text  void processData() override {  cout << "Processing Text data" << endl;  }  }; |

**Q2)** Improve the following design using an appropriate **design pattern**: **(10 marks)**

Note: All three classes represent gifts sold by a gift shop.



**Design Pattern Used: Composite**

