UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS214

Practical 1 Specifications

Release Date: 29-07-2025 at 11:00

Due Date: 12-08-2025 at 11:00

Total Marks: 120

Contents

1 General Instructions

- *Read the entire assignment thoroughly before you start coding.*
- This assignment should be completed in pairs.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- You can use any version of C++.
- You can import the following libraries:
  - vector
  - string
  - iostream
  - map
  - list
- You will upload your code with your main to FitchFork as proof that you have a working system.
- **If you meet the minimum FitchFork requirements (working code with at least 60% testing coverage), you will be marked by tutors during your practical session. Please book in advance (Instructions will follow on ClickUp).**
- **You will not be allowed to demo if you do not meet the minimum FitchFork requirements.**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.library.up.ac.za/plagiarism`. If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.

3 Mark Distribution

Activity	Mark
Task 1: UML Diagrams	30
Task 2: Implementing patterns	60
Task 3: FitchFork Testing Coverage	10
Task 4: Demo preparation	20
Total	**120**

Table 1: Mark Allocation

4 Overview and Background

Design tools have become a go-to medium for sharing ideas. Students use them for projects and presentations. Businesses and influencers rely on them for creating content. And many people use them for personal projects such as invitations and event flyers. However, most of today's design tools restrict access through paid subscriptions.

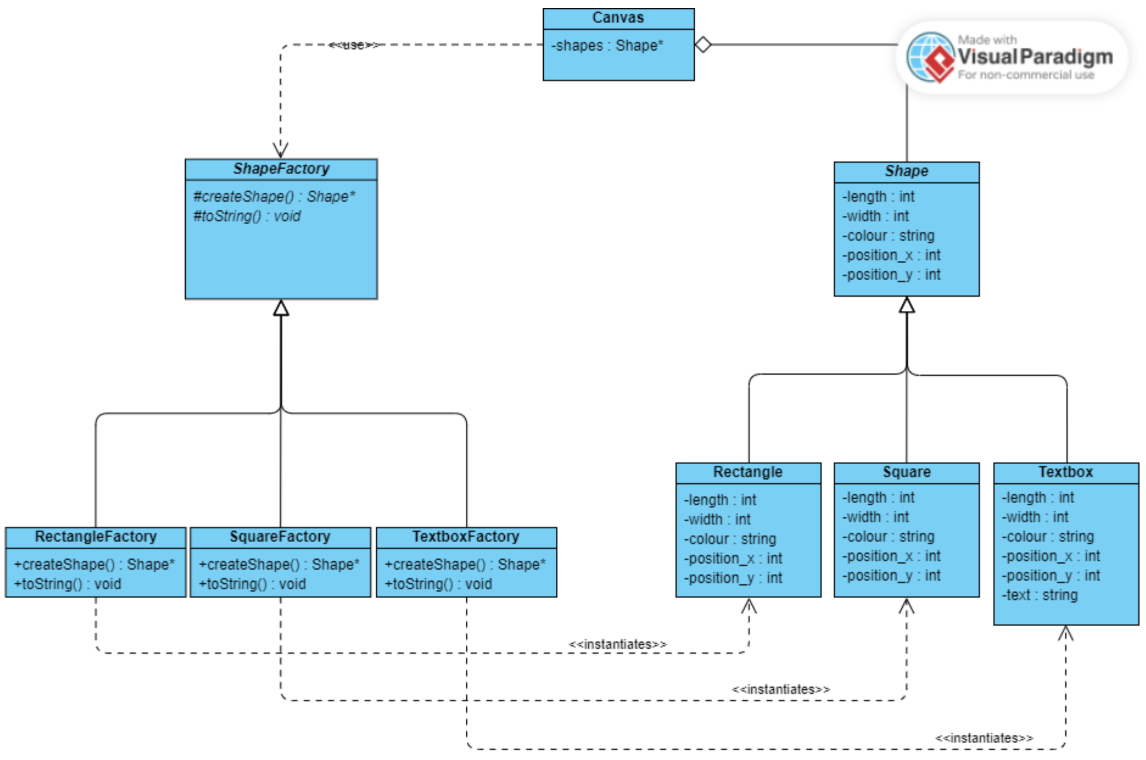
This practical introduces OpenCanvas: a free-to-use design platform that gives all users full access to every feature. As a startup, the application will begin with the basics including shapes and text, with more features added as the user base grows.

To ensure the platform is both flexible and scalable from the start, you will design the system using the following design patterns: Factory Method, Template Method, Memento, and Prototype.

Read the following sections carefully and complete the tasks that follow.

4.1 Factory Method

Different shape elements can be added to the canvas. These include Rectangles, Squares and Textboxes.



4.1.1 Classes and Attributes

1. **Canvas**
   Holds all of the shape elements added to the canvas to be displayed.

   - *Attributes*
     - shapes: Shape*

2. **ShapeFactory (Abstract Base Class)**

   - *Methods*
     - createShape() : Shape*
       Abstract method to be implemented by concrete factories.

3. **RectangleFactory, SquareFactory, TextboxFactory (Concrete Factory Classes)**

   - *Methods*
     - createShape() : Shape*
       Concrete implementation that instantiates and returns an object of the respective shape type (Rectangle, Square, Textbox) to be implemented by concrete factories.
     - toString() : string
       String representation of the shape type

4. **Shape(Abstract Product Class)**

   - *Attributes*
     - length : int
     - width : int
     - colour : string
     - positionX : int
     - positionY : int

   These are attributes to hold the general characteristics of a shape.

5. **Rectangle, Square, Textbox (Concrete Product Classes)**

   - Inherit from Shape
   - Only Textbox has an additional attribute text. This is the text that a textbox can contain.

4.1.2 Implementation Details

Factories

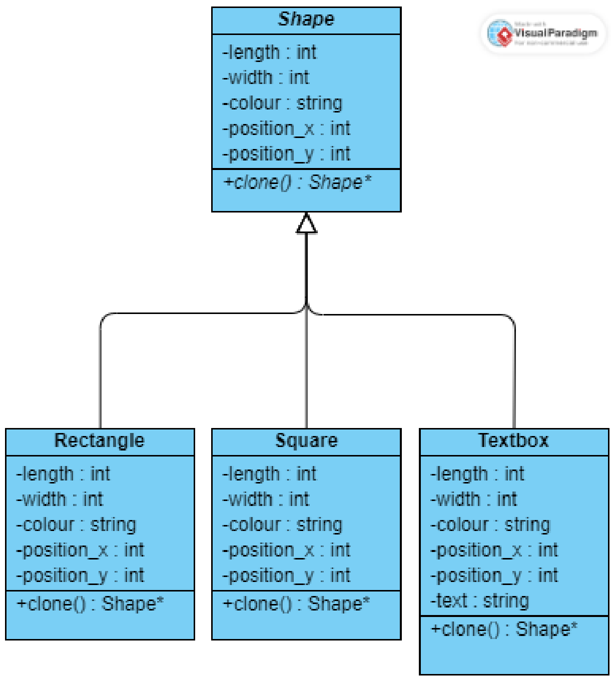
Each factory class should have methods to instantiate its corresponding shape type. For instance, RectangleFactory's createShape() would create and return an instance of Rectangle.

Shape Types

Implement the concrete shape classes with their specific characteristics. For example, a Textbox must have text, and a Square will have the same length and width.

4.2 Prototype

It is common practice to copy and paste shapes when duplicates are required. This is especially useful when diagrams compromise multiple of the same shape with the same attributes.



4.2.1 Classes and Attributes

1. **Shape(Abstract Product Class)**

   - *Attributes*
     - length : int
     - width : int
     - colour : string
     - positionX : int
     - positionY : int
   - *Methods*
     - clone() : Shape*
       Abstract method for cloning shape objects.

2. **Rectangle, Square, Textbox (Concrete Product Classes)**

   - Each class inherits all attributes from the Shape class.
   - Each class implements the +clone() : Shape* method, enabling the creation of new shape instances with identical attributes.

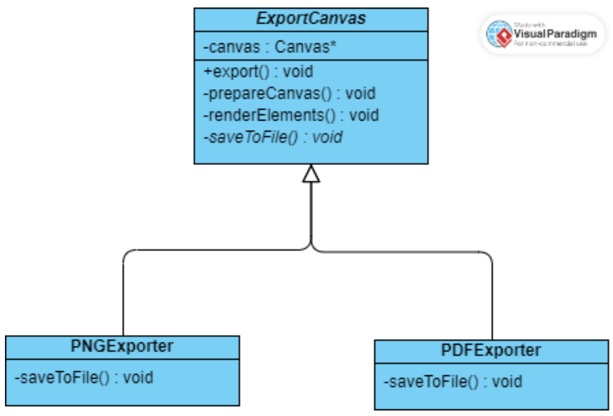
4.2.2 Implementation Details

Clone Method Implementation

Each concrete shape class must implement the clone() method. This method should create a new instance of the respective class and copy all attribute values from this object to the new object, ensuring a deep copy if needed.

4.3 Template Method

Users may either want to export their design as a PNG or a PDF depending on the content they have created. OpenCanvas caters to this functionality free of charge. To create these different file types, the same process is followed, but the final step differs when the saveToFile function is called.



4.3.1 Classes and Attributes

1. **ExportCanvas (Abstract Class):**

   - *Attributes*
     - canvas : Canvas*
   - *Methods*
     - export() : void
       A template method for exporting the canvas to a specific file format, that outlines the sequence of events. This calls abstract methods prepareCanvas(), renderElements(), and saveToFile()
     - prepareCanvas() : void, renderElements() : void, saveToFile() : void
       This method will be implemented differently by each concrete class.

2. **PNGExporter, PDFExporter (Concrete Classes)**

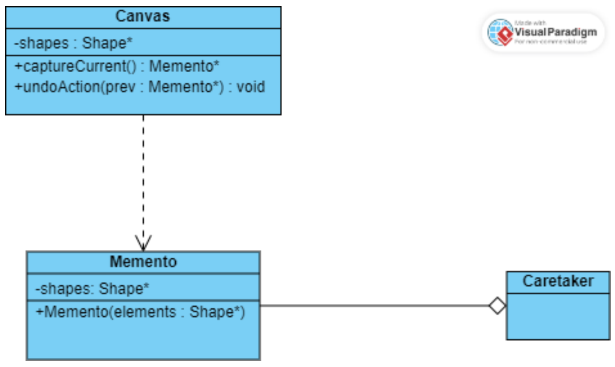
   - Each class inherits from ExportCanvas
   - Implements the abstract method in ExportCanvas

4.3.2 Implementation Details

- Each concrete exporter class (PNGExporter, PDFExporter) needs to provide their own implementation for prepareCanvas(), renderElements(), and saveToFile().
- The export() function should contain a sequence of calls to the following functions in this order:
  1. prepareCanvas()
  2. renderElements()
  3. saveToFile()

4.4 Memento

Instead of deleting a shape, it is a simple shortcut to just undo the last action created. This also ensures that the last item created is deleted and not some unintended item.


Canvas
-shapes : Shape*
+captureCurrent() : Memento*
+undoAction(prev : Memento*) : void
Memento
-shapes: Shape*
+Memento(elements : Shape*)
Caretaker
Visual Paradigm

4.4.1 Classes and Attributes

1. **Canvas**
   - *Attributes*
     - shapes: Shape*
   - *Methods*
     - captureCurrent() : Memento*
       Creates a memento containing a snapshot of its current state.
     - undoAction(prev: Memento*) : void
       Restores its state from the memento object.
2. **Memento**
   - *Attributes*
     - shapes: Shape*
   - *Methods*
     - Memento(elements: Shape*)
       Initializes the memento with the state of the shapes on the canvas.
3. **CareTaker**
   - *Responsibility*
     - Manages saving and restoring of Memento objects. This class can hold a stack, list, or another collection of mementos allowing the platform to revert to any previous state.

4.4.2 Implementation Details

Define Memento and Caretaker Classes:

- Memento Class:
  Implement the constructor to take the current state of the Canvas and store it. Ensure that all the attributes from Shapes are accurately copied to preserve the state.
- CareTaker Class:
  Manage a collection of Memento objects. Implement methods to add new mementos to the collection and retrieve them based on the platform's needs (the undo functionality).

Implement Methods in Canvas Class:

- captureCurrent():
  This method should create a new Memento object initialized with the current state of the Canvas object.
- undoAction(prev : Memento*):
  This method should take a Memento object and use it to restore the state of the Canvas object.

5 Task 1: UML Diagrams

In this task, you need to construct UML Class and Object diagrams for the given scenario.

5.1 Class Diagrams (20 Marks)

- Create a comprehensive UML class diagram that shows how the classes in the scenario interact and participate in various patterns.
- Ensure it includes all relevant classes, their attributes, and methods.
- Identify and indicate the design patterns each class is involved in, as well as their roles within these patterns. (Remember a class can participate in more than one pattern)
- Display all relationships between classes.
- Add any additional classes that are necessary for a complete representation.

5.2 Object Diagrams (10 Marks)

- Develop a UML object diagram representing your system at a specific point in time (of your choosing).
- Include the relationships and attribute values of the objects.
- There should be objects of at least 3 different classes and they cannot all be part of the Shape hierarchy.

6 Task 2: Implementation

In this task, you need to implement the design patterns described in the scenario.

- Use your UML class diagram as a guide to integrate the design patterns. Note that a **single class** can be part of **multiple patterns**.
- The provided scenario outlines the minimum requirements for this practical. Feel free to add any additional classes or functionalities to better demonstrate your understanding of the patterns and tasks.
- Thoroughly test your code (more details will be provided in the next task). Tutors will only mark code that runs.
- You may use arrays, vectors, or other relevant containers. Avoid using any libraries not mentioned in the general instructions, as your code must be compatible with FitchFork.
- Tutors may require you to explain specific functions to ensure you understand the pattern being implemented.

7 Task 3: FitchFork Testing Coverage

In this task, you are required to upload your completed practical to FitchFork. The FitchFork submission slot will open on 5 August 2025.

- Ensure that your code has at least 60% coverage in your testing main. This is necessary for demonstrating your work to the tutors.
- You will need to show your FitchFork submission with sufficient coverage to the tutor before being allowed to demo.
- You will be required to download your code from FitchFork for demonstration purposes.
- It might be useful to have two main files: a testing main for FitchFork and a demo main with a more user-friendly interface (e.g., a terminal menu) for demoing. This is just a suggestion, not a requirement. You are required to have at least a testing main. If you choose to have a demo main, upload it to FitchFork as well, but ensure it does not compile and run with make run.
- Name your testing main file **TestingMain.cpp**.
- If you create a demo main, name it **DemoMain.cpp** so it can be excluded from the coverage percentage. Note that you do not have to test your demo main.
- Additionally, upload a makefile that will compile and run your code with the command **make run**.

8 Task 4: Demo Preparation

You will have 5-7 minutes to demo your system and answer any questions from the tutors. Proper preparation is crucial.

- Ensure you have all relevant files open, such as UML diagrams and source code.
- Set up your environment so that you can easily run the code during the demo after downloading it.
- Be ready to demonstrate specific function implementations upon request.
- Practice your demo to make sure it fits within the allotted time and leaves time for questions.
- Prepare a brief overview (30 seconds) of your system to quickly explain its functionality and design.
- Make sure your testing and demo mains (if applicable) are ready to show their respective functionalities.
- Have a clear understanding of the design patterns used and be prepared to discuss how they are implemented in your code. Also, think about other potential use cases for the patterns.

- Be prepared to answer questions. If you do not know the answer, inform the marker and offer to return to the question later to avoid running out of time.

9 Submission Instructions

You will submit on both ClickUp and FitchFork.

- FitchFork submission:
  - Zip all files.
  - Ensure you are zipping the files and not the folder containing the files.
  - Upload to the appropriate slot on FitchFork well before the deadline, as no extensions will be granted.
  - Ensure that you have at least 60% coverage.
- ClickUp submission. You should submit the following in an archived folder:
  - Your UML diagrams (both as images and Visual Paradigm projects).
  - Your source code.