

LAB MANUAL

**Course: CSC303 Mobile Application Development**



**Department of Computer Science**

**COMSATS University Islamabad, Abbottabad Campus**

Table of Contents

[LAB 01: Environment setup & understanding 5](#_Toc85291369)

[Objective 5](#_Toc85291370)

[Scope 5](#_Toc85291371)

[Useful Concepts 5](#_Toc85291372)

[Exercises 6](#_Toc85291373)

[Exercises 7](#_Toc85291374)

[LAB 02: Basic Application Development 9](#_Toc85291375)

[Objective 9](#_Toc85291376)

[Scope 9](#_Toc85291377)

[Useful Concepts 9](#_Toc85291378)

[Lab Tasks 9](#_Toc85291379)

[Exercises 9](#_Toc85291380)

[LAB 03: Basic UI components and widget 10](#_Toc85291381)

[Objective 10](#_Toc85291382)

[Scope 10](#_Toc85291383)

[Useful Concepts 10](#_Toc85291384)

[Lab Tasks 10](#_Toc85291385)

[Exercises 10](#_Toc85291386)

[Objective 11](#_Toc85291387)

[Scope 11](#_Toc85291388)

[Useful Concepts 11](#_Toc85291389)

[Lab Tasks 11](#_Toc85291390)

[Exercises 11](#_Toc85291391)

[LAB 05: UI Layouts and Advanced UI Components 12](#_Toc85291392)

[Objective 12](#_Toc85291393)

[Scope 12](#_Toc85291394)

[Useful Concepts 12](#_Toc85291395)

[Lab Tasks 12](#_Toc85291396)

[Exercises 12](#_Toc85291397)

[LAB 06: Sessional 1 Exam 13](#_Toc85291398)

[LAB 07: Fragments 14](#_Toc85291399)

[Objective 14](#_Toc85291400)

[Scope 14](#_Toc85291401)

[Useful Concepts 14](#_Toc85291402)

[Lab Tasks 14](#_Toc85291403)

[Exercises 14](#_Toc85291404)

[LAB 08: Application Security and Permissions 15](#_Toc85291405)

[Objective 15](#_Toc85291406)

[Scope 15](#_Toc85291407)

[Useful Concepts 15](#_Toc85291408)

[Lab Tasks 15](#_Toc85291409)

[Exercises 15](#_Toc85291410)

[Objective 16](#_Toc85291411)

[Scope 16](#_Toc85291412)

[Useful Concepts 16](#_Toc85291413)

[Lab Tasks 16](#_Toc85291414)

[Exercises 16](#_Toc85291415)

[LAB 10: Multithreading 17](#_Toc85291416)

[Objective 17](#_Toc85291417)

[Scope 17](#_Toc85291418)

[Useful Concepts 17](#_Toc85291419)

[Lab Tasks 17](#_Toc85291420)

[Exercises 17](#_Toc85291421)

[LAB 11: Broadcast Receivers 18](#_Toc85291422)

[Objective 18](#_Toc85291423)

[Scope 18](#_Toc85291424)

[Useful Concepts 18](#_Toc85291425)

[Lab Tasks 18](#_Toc85291426)

[Exercises 18](#_Toc85291427)

[LAB 12: Sessional 2 Exam 19](#_Toc85291428)

[LAB 13: Services 20](#_Toc85291429)

[Objective 20](#_Toc85291430)

[Scope 20](#_Toc85291431)

[Useful Concepts 20](#_Toc85291432)

[Lab Tasks 20](#_Toc85291433)

[Exercises 20](#_Toc85291434)

[LAB 14 Sensors and Third-party APIs 21](#_Toc85291435)

[Objective 21](#_Toc85291436)

[Scope 21](#_Toc85291437)

[Useful Concepts 21](#_Toc85291438)

[Lab Tasks 21](#_Toc85291439)

[Exercises 21](#_Toc85291440)

[LAB 15 Cross-Platform Development 22](#_Toc85291441)

[Objective 22](#_Toc85291442)

[Scope 22](#_Toc85291443)

[Useful Concepts 22](#_Toc85291444)

[Lab Tasks 22](#_Toc85291445)

[Exercises 22](#_Toc85291446)

[Objectives 23](#_Toc85291447)

# LAB 01: Environment setup & understanding

## Objective

The objective of this lab is to understand how you can attach additional responsibilities to an object dynamically via restructuring its representation using the Decorator design pattern. In second session of the lab we learn about Factory Method that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

## Scope

The scope of this lab activity is the student’s ability to:

* Restructure the representation of an object with the help of Decorator to attach additional responsibility dynamically to an object.
* Provides an interface for creating objects in a superclass and allow subclasses to alter the type of the object.

## Useful Concepts

### Decorator Implementation Guidelines

1. Make sure your business domain can be represented as a primary component with multiple optional layers over it.
2. Figure out what methods are common to both the primary component and the optional layers. Create a component interface and declare those methods there.
3. Create a concrete component class and define the base behavior in it.
4. Create a base decorator class. It should have a field for storing a reference to a wrapped object. The field should be declared with the component interface type to allow linking to concrete components as well as decorators. The base decorator must delegate all work to the wrapped object.
5. Make sure all classes implement the component interface.
6. Create concrete decorators by extending them from the base decorator. A concrete decorator must execute its behavior before or after the call to the parent method (which always delegates to the wrapped object).
7. The client code must be responsible for creating decorators and composing them in the way the client needs.

### Factory Method Implementation Guidelines

1. Make all products follow the same interface. This interface should declare methods that make sense in every product.
2. Add an empty factory method inside the creator class. The return type of the method should match the common product interface.
3. In the creator’s code find all references to product constructors. One by one, replace them with calls to the factory method, while extracting the product creation code into the factory method.
4. You might need to add a temporary parameter to the factory method to control the type of returned product.
5. At this point, the code of the factory method may look pretty ugly. It may have a large switch operator that picks which product class to instantiate. But don’t worry, we’ll fix it soon enough.
6. Now, create a set of creator subclasses for each type of product listed in the factory method. Override the factory method in the subclasses and extract the appropriate bits of construction code from the base method.
7. If there are too many product types and it doesn’t make sense to create subclasses for all of them, you can reuse the control parameter from the base class in subclasses.
8. For instance, imagine that you have the following hierarchy of classes: the base Mail class with a couple of subclasses: AirMail and GroundMail; the Transport classes are Plane, Truck and Train. While the AirMail class only uses Plane objects, GroundMail may work with both Truck and Train objects. You can create a new subclass (say TrainMail) to handle both cases, but there’s another option. The client code can pass an argument to the factory method of the GroundMail class to control which product it wants to receive.
9. If, after all of the extractions, the base factory method has become empty, you can make it abstract. If there’s something left, you can make it a default behavior of the method.

## Exercises

### Activity 1 (Remote teaching)

Login in to subexpert.com with your student account and then watch the following interactive videos that demonstrates the Decorator and Factory Method.

* For Decorator Pattern
  + https://www.subexpert.com/CourseLectures/OnTopic/Design-Patterns/Decoratoy
* For Factory Method Pattern
  + https://www.subexpert.com/CourseLectures/OnTopic/Design-Patterns/Factory-Method

**Note:** In case of physical labs this activity is optional but recommended.

### Activity 2

The text book motivational code example for window/scrolling scenario is available on the following link. You need to run the version in Java which is quite simple.

* + https://en.wikipedia.org/wiki/ Decorator\_pattern

### Activity 3

Run one another very simple example about the Decorator pattern related to decorating geometrical shapes from the following link:

* + https://www.tutorialspoint.com/design\_pattern/decorator\_pattern.htm

### Activity 4

Imagine that you’re working on a notification library which lets other programs notify their users about important events. You need to first focus on the problem in this context and then understand its solution using the Decorator pattern from the following link:

* + <https://refactoring.guru/design-patterns/decorator>

**For Factory Method**

### Activity 5

Run the example on

* + https://github.com/sshpk/dp\_fall20/tree/master/FactoryMethodFA20

### Activity 6

Also run the following examples of factory method:

* + [Design Patterns: Factory Method in Java (refactoring.guru)](https://refactoring.guru/design-patterns/factory-method/java/example)
  + [Factory method pattern - Wikipedia](https://en.wikipedia.org/wiki/Factory_method_pattern) Modify the C# example into Java

## Exercises

Update the example with following additions:

**Home Work 1**:

Modify the example in Activity 3 to convert the output to "Shape : Circle with Red color" when it is decorated and "Shape: Circle" when not decorated.

**Home Work 2:**

For example, in Activity 3, Add one more decorator ThickBorderDecorator, which will decorate the shape with thick border.

**Home Work 3**:

Print shapes without decoration, with red color, with thick border only and then with both red color and thick border decoration in the Demo class.

**Home Work 4**:

For Activity 5, Add one another TriangleGeometry class to provide the Triangle factory method.

**Home Work 5**:

Instead of using four classifiers i.e. (Geometry, SquareGeometry, CircleGeometry etc.), We can implement factory method with a single method with any polymorphic behavior by passing it the type of object we need and then the method returns the desired object. Provide this single factory method implementation for the same example.

. **Home Work 6**:

Think on how can you utilize the Factory Method in your final year project and provide its implementation. (Optional)

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 02: Basic Application Development

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 03: Basic UI components and widget

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 04: Activity, Intent, and Intent filters

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 05: UI Layouts and Advanced UI Components

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 06: Sessional 1 Exam

**Purpose**

The purpose of this lab is to conduct the first sessional exam based on the activities conducted so far.

**Tasks**

The tasks will be decided by the respective course instructor/lab tutor.

# LAB 07: Fragments

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 08: Application Security and Permissions

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 9: Data Storage & Content Providers

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 10: Multithreading

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

* Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 11: Broadcast Receivers

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

* Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 12: Sessional 2 Exam

**Objectives**

The purpose of this lab is to conduct the second sessional exam based on the activities conducted so far.

**Tasks**

The tasks will be decided by the respective course instructor/lab tutor.

# LAB 13: Services

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 14 Sensors and Third-party APIs

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 15 Cross-Platform Development

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 16: Final Exam

# Objectives

The purpose of this lab is to conduct the final exam based on the activities conducted throughout the semester.

**Lab Tasks**

The tasks will be decided by the respective course instructor/lab tutor.