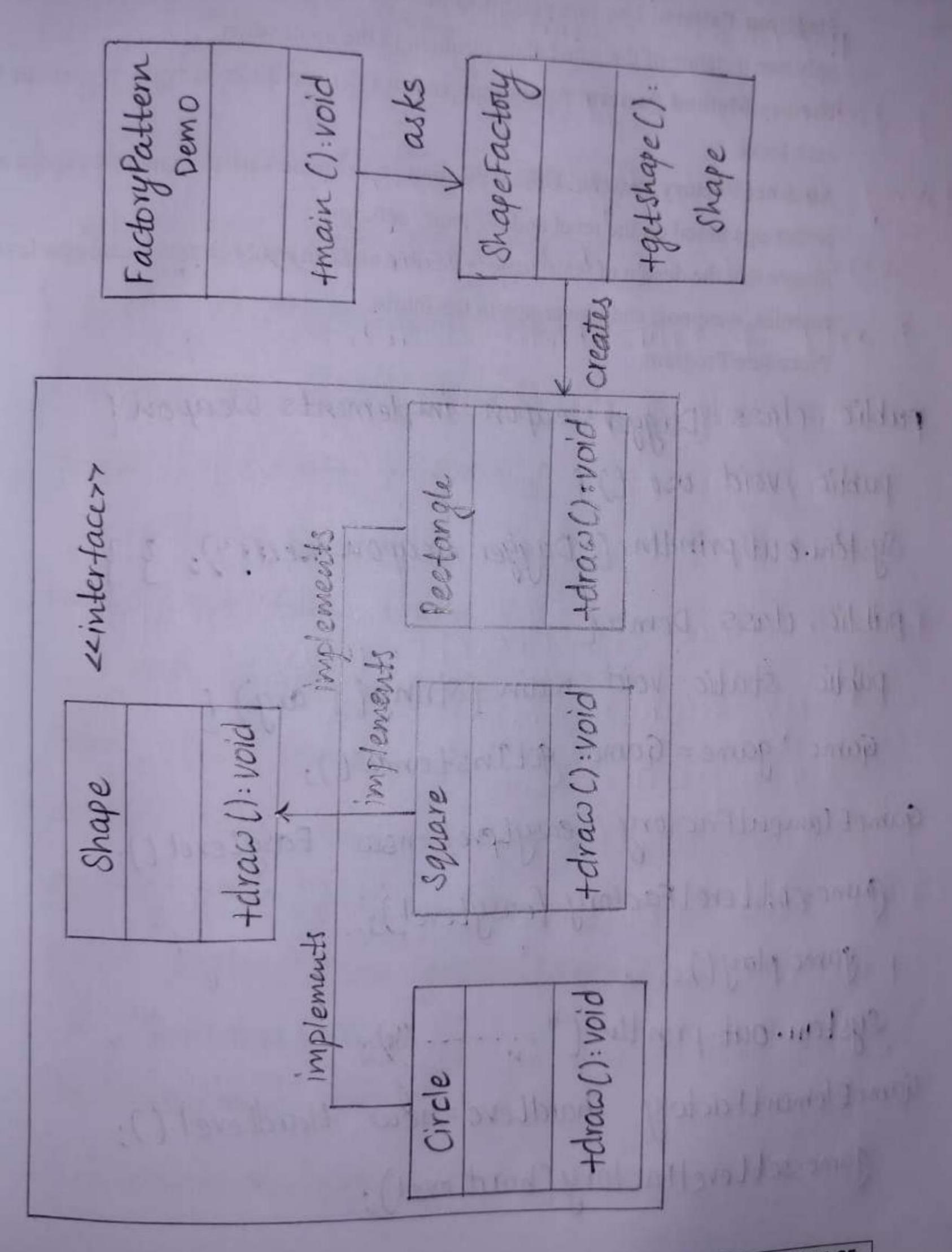
Experiment#	
Date	Student ID
	Student Name

2) Draw the UML Relationship Diagram for Factory Design Pattern for any customized scenario.



		ACADEMIC YEAR: 2024-25
Course Title	Advanced Object-Oriented Programming	Page 25
Course Code	23CS2103A & 23CS2103E	Page

erns s

(Factor)

In-Lab:

1) Develop a game application with multiple levels and varying difficulty at Implement the following design patterns to manage different aspects of the game Singleton Pattern: Use this pattern to manage the game state, ensuring that the only one instance of the game state throughout the application.

Factory Method Pattern: Apply this pattern to create different types of enemieach level.

Abstract Factory Pattern: Utilize this pattern to create various types of weapon power-ups based on the level and difficulty settings.

Ensure that the design of your game is flexible and can easily accommodate newle enemies, weapons, and power-ups in the future.

Procedure/Program:

public class Dagger weapon implements weapon {

public void use () {

System.out.println ("Dagger weapon used!"); 3 }

public class Demo {

public static void main (String[] args) {

Game game = Game gctInstance();

GameElementFactory easyLevel = new EasyLevel();

game.setLevelFactory (easyLevel);

System.out.println ("----");

GameElementFactory hardLevel=new HardLevel();

game.setLevelFactory(hardLevel);

Course little	Advanced Object-Orio
Course Code	23CS2103A & 23CS2103E Advanced Object-Oriented Programming
	ACADEMIC YEAR:
	Page 26

Experiment Date

public

y publ

re

b pu

publ

Public

Public

Public

if

instan

y re

Course Title Course Code Experiment# Student ID

Date Student Name

ty setting game:

that there

enemies f

weapons a

e new level

public class Easylevel implements Game Element Factory [

public Enemy create Enemy () {

return new Goblin Enemy ();

Je public Weapon createweapon() (
return new Daggerweapon();

game-play ();

y public PowerUp create PowerUp () [
return new HealthPowerUp (): 3 3

public interface Weapon (
void use (); 3

public interface Enemy [

void attack (); z

Public class game {

private static game instance;

private game () { }

Public static game get Instance () [

if (instance == null) [

instance = new game ();

3 return instance; z

		ACADEMIC YEAR: 2024-23
Course Title	Advanced Object-Oriented Programming	Page 27
Course Code	23CS2103A & 23CS2103E	Page

AR: 2024-2

Student ID Experiment# Student Name Date

public void set Level Factory (Game Element Factory factory) [

this factory = factory; public void play(){

Enemy = factory. creati Enemy ();

Weapon we apon = factory. create weapon ();

PowerUp powerup = factory.createPowerUp ();

enemy. attack ();

weapon use ();

powerup. activate (); } }

public interface Game Element Factory [

Enemy create Enemy ();

Weapon create We apon();

Powerup create Power Up (); }

public class Goblin Enemy implements Enemy { public void attack () {

System.out.println ("Goblin enemy attack!");

Advanced Object-Oriented Programming **Course Title** 23CS2103A & 23CS2103E Course Code ACADEMIC YEAR Page | 28

Experiment#

Date

public class public En return 1 public h return 3 public Pa return public clas Public Vi System-out-p public class Public voic System-out-Public inter

Course Title Advanced Course Code 23CS2103

void acti

void

y public in

Experiment# Student ID

Date Student Name

public class Hardlevel implements gameElement Factory 1 public Enemy create Enemy () (return new orc Enemy(); 3 public Weapon create weapon () { return new swordweapon (); & public Power up create Power up () [return new ShieldPower Up (); 4 4 public class HealthPowerUp implements PowerUp (public void activate () { System.out.println ("Health power-up activated"); 33 public class OrcEnemy implements Enemy L public void attack () [System.out-println ("orc enemy attack!"); 33 Public interface Power Up 1 Void-activate (); J' public interface Weapon { void use ();

		WEAR: 2024-25
Course Title	Advanced Object-Oriented Programming	ACADEMIC YEAR: 2024-23
Course Code	23CS2103A & 23CS2103E	Page 129

Experiment# Student ID

Oute Student Name

✓ Data and Results:

Enemy Created: Easy Enemy Weapon created: Gun Power-up created: Health Power-up

✓ Analysis and Inferences:

Both the Factory and Abstract Factory design patterns focus on encapsulating object creation and enhancing thexibility in object instantiation. Factory pattern is suitable when you want to create different instances of a single type.

Experiment#
Date

IVA-VOCE

in Java's

clas

provid

insta

single

for n

2) Discuss

Both

Factory type of

Abstro

familie.

3) Mention

reduced

such a

overus

ourse Title ourse Code

Experiment#	Student ID
Date	Student Name

VIVA-VOCE Questions (In-Lab):

- 1) Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?

 classes that manage global state, resources or provide utility functions are good candidates for singleton pattern. Singleton pattern ensures a single instance with controlled access, making it suitable for managing shared resources and configurations.
- 2) Discuss the difference between Factory and Abstract Factory design patterns.

Both patterns abstract the object creation process, Factory Method pattern is best for creating a single type of object with varying implementations, whereas Abstract Factory pattern is ideal for creating families of related objects that need to work together.

3) Mention the pros and cons of Factory Design pattern

It offers improved encapsulation, flexibility and reduced code duplication. It introduces complexities such as increased codebase complexity and potential overuse.

		TO ALC YEAR: 2024-25
Course Title	Advanced Object-Oriented Programming	ACADEMIC YEAR: 2024-23
Course Code	23CS2103A & 23CS2103E	Page

EAR: 2024-2

Post-Lab:

1) Design and implement a ride-sharing application that allows users to request ride various types of vehicles (cars, bikes, scooters). Utilize the Factory Method par create vehicle instances, the Abstract Factory pattern to implement different par methods (credit card, PayPal, cash), and the Singleton pattern to manage authentication securely. Provide a detailed example demonstrating the interaction these patterns within the application.

Procedure/Program:

```
public class Bike implements Vehicle {

public void request Ride () {

System out println ("Requesting a Bike ride!!!");

} }

public class Bike Factory implements Vehicle Factory {

Public Vehicle create Vehicle () {

return new Bike ();
```

Course Title	Advanced Object On	
Course Code	Advanced Object-Oriented Programming 23CS2103A & 23CS2103E	10
	TOCSZ103E	ACADEMIC YEAR:
		Page 32

ourse

Experiment#

Date

Student ID

Student Name

3 3 public class Car implements Vehicle [

public void requestRide()[

System out println ("Requesting a Car Ride!!!");

3 3 public class CarFactory implements VehicleFactory[

public Vehicle CreateVehicle()[

return neω Car();

} }

public class Credit Card Factory implements Payment Method [

Public void pay (double amount) [

System out printh ("Paid \$" + amount +" using a credit card.!");

public class DemoMain {

public static void main (string [] args) {

UserAuthentication authentication = UserAuthentication

getInstance();

boolean isAuthenticate = authentication authenticateUser

("username", "password");

if (isAuthenticate) {

Course Title Advanced Object-Oriented Programming ACADEMIC YEAR: 2024-25

Course Code 23CS2103A & 23CS2103E

est rides for nod pattern rent payment nanage us

R: 2024-

	Student ID
Experiment#	Student Name
Date	

Vehicle Factory carfactory = new CarFactory (); Vehicle car = car Factory. create Vehicle (); car request Ride (); Vehicle Factory bike Factory = new Bike Factory (); Vehicle bike = bike Factory. create Vehicle (); bike request Ride ();

Vehicle Factory Scooter Factory = new Scooter Factory (); Vehicle Scooter = scooter Factory. createvehicle (); scooter-requestRide();

Payment Method Factory credit Card Factory=new CreditCardFactory();

Payment Method credit CardPayment = credit CardFactory. createPaymentMethod();

credit Card Payment pay (20.0);

Payment MethodFactory payPalFactory= new PayPalFactory ();

Payment Method PayPalPayment = payPalFactory. create Payment Method (); PayPalPayment · pay (15.0);

Jelse (

Course Title	Advanced Object-Ovice
Course Code	Advanced Object-Oriented Programming 23CS2103A & 23CS2103E ACADEMIC YEAR
	ACADEMIC YEAR
	Page 34

Experiment#

Date

System-ou request a y public void J public Payment public class public Pa return public class

AU SALVES OF THE STREET	
ourse Title	Advanced O
ourse Code	23CS2103A 8
	-2COLLUSA (

Public v

System-out pri

Student ID Experiment# Student Name Date

System-out println ("Authentication tailed Unable to request a ride or make a payment. 4); The same of the sa

I public interface Payment Method [void pay (double amount);

3 public interface Payment MethodFactory (Payment Method createPaymentMethod ();

public class PayPalFactory implements Payment Method

public Payment Method create Payment Method () [return new PayPalPayment ();

implements Payment Method [public class PayPalFactory Public void pay (double amount) (System-out-println ("Paid \$" + amount +" using PayPal.!")

	ACADEMIC YEAR: 202
Course Title	Advanced Object-Oriented Programming Page 35
Course Code	23CS2103A & 23CS2103E

Student ID Student Name Experiment# Date

public class Scooter implements Vehicle public void requestRide () (System-out-println ('Requesting a scooter ride!!!").

class scooterfactory implements Vehicle Factory [

public Vehicle create_Vehicle () { return new Scooter ();

J' public class UserAuthentication (

private static UserAuthentication instance;

private UserAuthentication () []

public static UserAuthentication getInstance () {

if (instance == null)

instance = new User Authentication ();

return instance;

3 public boolean authenticate User (String username, String password) [

return true; }

xperiment#

Pate

public inte void requ 3 public i Vehicle

	ourse Code	23CS210
10		
EAR.		

ourse Title

Advanced Object

Course Title	Advanced Object-Oriented Programming 23CS2103A & 23CS2103E	-12
Course Code	23CS2103A & 23CS2103E	ACADEMIC YEAR:
		Page 36

Experiment#

Date

Student ID

Student Name

public interface Vehicle [

void request Ride ();

} public interface Vehicle Factory [

Vehicle create_Vehicle ();

}

the party and and stranger weeking the goal in

Lines with stated to anyth humblin aminant

The street of th

the first state dealth design to and and

Course Title Advanced Object-Oriented Programming ACADEMIC YEAR: 2024-25

Course Code 23CS2103A & 23CS2103E

Experiment# Student Name

Date

✓ Data and Results:

Riding a car.
Paying \$10.0 with credit card

✓ Analysis and Inferences:

Factory Method pattern separate the process of creating different types of vehicles from client code. Abstract Factory Pattern separates the creation of payment methods from the rest of application logic.

Evaluator Remark (if Any):	
	Marks Secured:out of 50
	Signature of the residual

xperiment#

ate

Structural Design

m/Objective: To a

e real-time scenario

scription: The str

e-Requisites: Class

ols: Eclipse IDE fo

e-Lab:

terns.

Draw the UMI adapter scenarion produces either between mobile

Socket

-get voltage
():int

ourse Title Advanced
ourse Code 23CS2103