Day 25 - 09th Aug 2025

Task 1:

Actual complete code is in the code for reference folder →OOAD → Bridge Method

Part of the code -main.java

public class Square extends Shape{

private int s;

// ExcalidrawAPI excalidrawAPI;

Square(int s,ExcalidrawAPI excalidrawAPIobj ) {

super(excalidrawAPIobj);

// super(ExcalidrawAPI excalidrawAPI); // need to check

this.s = s;

}

@Override

void draw() {

excalidrawAPI.drawSquare(s);

}

}

**Answer**:

// Implementor

interface ExcalidrawAPI { void drawSquare(int side); }

// Abstraction

abstract class Shape {

protected final ExcalidrawAPI excalidrawAPI;

protected Shape(ExcalidrawAPI api){ this.excalidrawAPI = api; }

abstract void draw();

}

// Refined Abstraction

public class Square extends Shape {

private final int s;

public Square(int s, ExcalidrawAPI api){ super(api); this.s = s; }

@Override void draw(){ excalidrawAPI.drawSquare(s); }

}

// Concrete Implementors

class ExcalidrawV1 implements ExcalidrawAPI {

public void drawSquare(int side){ System.out.println(&quot;V1 square: &quot;+side); }

}

class ExcalidrawV2 implements ExcalidrawAPI {

public void drawSquare(int side){ System.out.println(&quot;V2 square: &quot;+side); }

}

// Client

class Main {

public static void main(String[] args){

Shape sq1 = new Square(50, new ExcalidrawV1());

Shape sq2 = new Square(80, new ExcalidrawV2());

sq1.draw(); sq2.draw();

}

}

Task 02

Composite

**Answer**:

// Component

interface Graphic { void draw(); }

// Leaf

class Dot implements Graphic {

private final int x,y;

Dot(int x,int y){ this.x=x; this.y=y; }

public void draw(){ System.out.println(&quot;Dot(&quot;+x+&quot;,&quot;+y+&quot;)&quot;); }

}

// Composite

class CompoundGraphic implements Graphic {

private final List&lt;Graphic&gt; children = new ArrayList&lt;&gt;();

public void add(Graphic g){ children.add(g); }

public void remove(Graphic g){ children.remove(g); }

public void draw(){ for(Graphic g: children) g.draw(); }

}

// Client

class CompositeDemo {

public static void main(String[] args){

CompoundGraphic scene = new CompoundGraphic();

scene.add(new Dot(1,2));

scene.add(new Dot(3,4));

scene.draw();

}

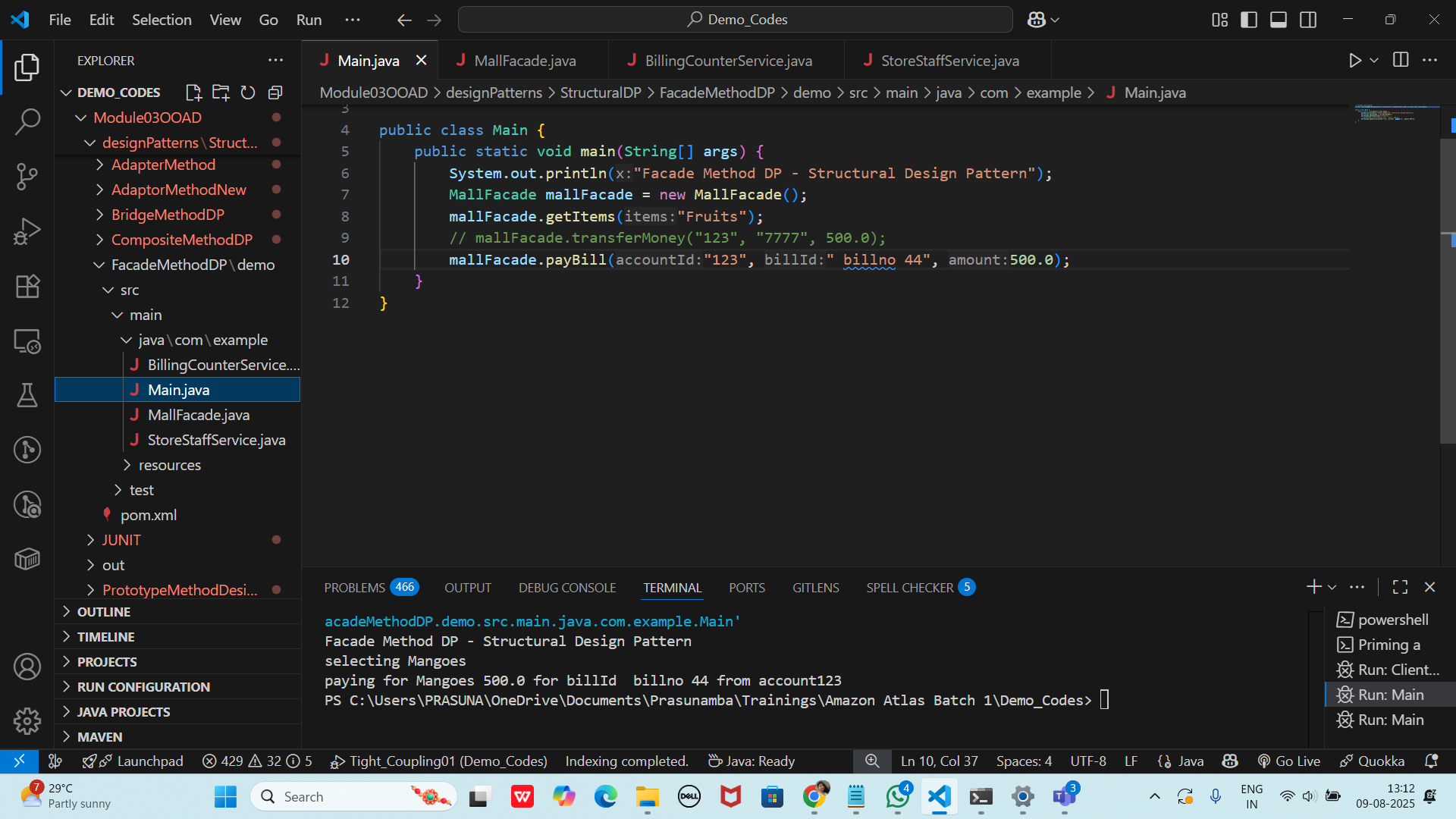
}

Actual complete code is in the code for reference folder →OOAD →

Task 03:

Facade

Actual complete code is in the code for reference folder →OOAD →



**Answer**:

// Subsystems

class AuthService { boolean login(String u,String p){ return true; } }

class PaymentGateway { void pay(double amt){ System.out.println(&quot;Paid &quot;+amt); } }

class InvoiceService { void generate(){ System.out.println(&quot;Invoice generated&quot;); } }

// Facade

class CheckoutFacade {

private final AuthService auth = new AuthService();

private final PaymentGateway pay = new PaymentGateway();

private final InvoiceService inv = new InvoiceService();

public void checkout(String user,String pass,double amount){

if(auth.login(user,pass)){ pay.pay(amount); inv.generate(); }

}

}

// Client

class FacadeClient {

public static void main(String[] args){ new CheckoutFacade().checkout(&quot;u&quot;,&quot;p&quot;,499.0); }

}

Answer:

interface Renderer { void drawCircle(int r); }

abstract class ShapeB { protected final Renderer r; ShapeB(Renderer r){ this.r=r; } abstract void

draw(); }

class CircleB extends ShapeB { private final int radius; CircleB(int radius, Renderer r){ super(r);

this.radius=radius; } void draw(){ r.drawCircle(radius); } }

class SvgRenderer implements Renderer { public void drawCircle(int r){

System.out.println(&quot;&lt;circle r=&#39;&quot;+r+&quot;&#39;/&gt;&quot;); } }

class BridgeSample { public static void main(String[] a){ new CircleB(10, new

SvgRenderer()).draw(); } }

Home Task 02:

//��

Flyweight

Answer:

// Flyweight

class BulletType {

private final String sprite; // intrinsic

BulletType(String sprite){ this.sprite = sprite; }

public void draw(int x,int y){ System.out.println(&quot;draw &quot;+sprite+&quot; at &quot;+x+&quot;,&quot;+y); }

}

// Flyweight Factory

class BulletFactory {

private static final Map&lt;String,BulletType&gt; cache = new HashMap&lt;&gt;();

public static BulletType get(String sprite){ return cache.computeIfAbsent(sprite,

BulletType::new); }

}

// Context

class Bullet {

private final int x,y; // extrinsic

private final BulletType t; // shared

Bullet(int x,int y,String sprite){ this.x=x; this.y=y; this.t = BulletFactory.get(sprite); }

void render(){ t.draw(x,y); }

}

Home Task 03:

Decorator

Answer:

// Component

interface Notifier { void send(String msg); }

// Concrete Component

class BasicNotifier implements Notifier { public void send(String msg){

System.out.println(&quot;Send: &quot;+msg); } }

// Base Decorator

abstract class NotifierDecorator implements Notifier {

protected final Notifier wrap;

protected NotifierDecorator(Notifier wrap){ this.wrap = wrap; }

}

// Concrete Decorators

class EmailDecorator extends NotifierDecorator {

public EmailDecorator(Notifier w){ super(w); }

public void send(String msg){ wrap.send(msg); System.out.println(&quot;Email: &quot;+msg); }

}

class SmsDecorator extends NotifierDecorator {

public SmsDecorator(Notifier w){ super(w); }

public void send(String msg){ wrap.send(msg); System.out.println(&quot;SMS: &quot;+msg); }

}

// Client

class DecoratorDemo {

public static void main(String[] a){

Notifier n = new SmsDecorator(new EmailDecorator(new BasicNotifier()));

n.send(&quot;Order shipped&quot;);

}

}

Link updated excalidraw - 18.09

<https://excalidraw.com/#json=KDKTMT2ib13Az0HOJjGAy,_AQWr8Z-iFGmfbG97kEC4g>

// 7989523779