

Name: MINH DUC NGUYEN

CSC 143 Winter 2021

LAB 02: Objects, Inheritance-Interface

MATRIX OBJECT

SquareMatrix class:

Constructors:

SquareMatrix(int size): generates an identity matrix of that size.

```
/*Constructor accepts the size of the matrix
and generates an identity matrix of that size.*/

public SquareMatrix(int size) {
    this.size = size;
    matrix = new int[size][size];
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            matrix[i][j] = (i == j)? 1:0;
        }
    }
}
```

SquareMatrix(int[][] matrix): represents the n x n matrix

```
/*Constructor accepts a 2D array that is assigned to the stored 2D array
that represents the n x n matrix.*/

public SquareMatrix(int[][] matrix) {
    int row, col;
    row = matrix.length;
    col = matrix[0].length;

    if (row != col)
        throw new IllegalArgumentException("It must be n x n matrix");

    this.matrix = matrix;
    this.size = row;
}
```

Methods:

Add: add two matrices

```
public SquareMatrix add(SquareMatrix other) {
    int[][] sum = new int[size][size];
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            sum[i][j] = matrix[i][j] + other.matrix[i][j];
        }
    }
    return new SquareMatrix(sum);
}
```

Subtract: subtract two matrices

```
public SquareMatrix subtract(SquareMatrix other) {
    int[][] subtract = new int[size][size];
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            subtract[i][j] = matrix[i][j] - other.matrix[i][j];
        }
    }
    return new SquareMatrix(subtract);
}
```

Multiply: multiply two matrices

```
public SquareMatrix multiply(SquareMatrix other) {
    int[][] multiply = new int[size][size];
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int n = 0; n < size; n++) {
                multiply[i][j] += matrix[i][n] * other.matrix[n][j];
            }
        }
    }
    return new SquareMatrix(multiply);
}
```

Equals: check if two matrices are identical whether or not

```
public boolean equals(Object obj) {
    if (obj instanceof SquareMatrix) {
        SquareMatrix other = (SquareMatrix) obj;
        return Arrays.deepEquals(matrix, other.matrix);
    } else
        return false;
}
```

isDiagonal and isIdentity: check if the matrix is diagonal or identity

```
public boolean isDiagonal() {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i != j && matrix[i][j] != 0)
                return false;
        }
    }
    return true;
}

public boolean isIdentity() {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (i == j && matrix[i][j] != 1)
                return false;
            else if (i != j && matrix[i][j] != 0)
                return false;
        }
    }
    return true;
}
```

toString: represent the matrix

```
@Override
public String toString() {
    StringBuilder result = new StringBuilder("\n");
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[i].length; j++) {
            result.append(matrix[i][j]).append("\t");
        }
        result.append("\n");
    }
    return result.toString();
}
```

MainProgram class:

Methods:

read and write methods: to read the data in the input file, and write the output into a new output file

```
public static SquareMatrix read(String location) throws IOException {
    File file = new File(location);
    Scanner input = new Scanner(file);
    int size = input.nextInt();
    int[][] matrix = new int[size][size];

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            matrix[i][j] = input.nextInt();
        }
    }
    input.close();
    return new SquareMatrix(matrix);
}

public static void write(String location, StringBuilder builder) throws IOException {
    File file = new File(location);
    PrintWriter writer = new PrintWriter(file);

    writer.println(builder);
    writer.close();
}
```

Main():

First, read the given data in the input files.

Then, use StringBuilder to append the matrices, and test the operations to make sure everything works correctly.

After that, use write method to put all the things above into the output file.

```
public static void main(String[] args) throws IOException{
    String location = "data" + File.separator + "matrix.txt";
    String otherLocation = "data" + File.separator + "other_matrix.txt";
    String outputLocation = "data" + File.separator + "output.txt";

    // Create object to append string output of matrix
    StringBuilder builder = new StringBuilder("\n");

    // TODO : Create matrices for testing
    SquareMatrix m1 = read(location);
    SquareMatrix m2 = read(otherLocation);

    // TODO : Test operations of Square Matrices
    builder.append("PROGRAM TEST\n");
    builder.append("\nMatrix:" + m1.toString());
    builder.append("\nOther Matrix:" + m2.toString());
    builder.append("\nMatrix + Other Matrix:" + m1.add(m2));
    builder.append("\nMatrix - Other Matrix:" + m1.subtract(m2));
    builder.append("\nMatrix * Other matrix:" + m1.multiply(m2));
    builder.append("\nCheck if Matrix is diagonal: " + m1.isDiagonal() + "\n");
    builder.append("\nCheck if Matrix is identity: " + m1.isIdentity() + "\n");
    builder.append("\nCheck if Other Matrix is diagonal: " + m2.isDiagonal() + "\n");
    builder.append("\nCheck if Other Matrix is identity: " + m2.isIdentity() + "\n");
    builder.append("\nCheck if two matrices are equal: " + m1.equals(m2));

    // write to output file
    write(outputLocation, builder);
}
```

OUTPUT: output of SquareMatrix Program test

```
output.txt - Notepad
File Edit Format View Help

PROGRAM TEST

Matrix:
4      3      6
5      4      3
8      5      4

Other Matrix:
1      2      1
2      1      2
1      2      1

Matrix + Other Matrix:
5      5      7
7      5      5
9      7      5

Matrix - Other Matrix:
3      1      5
3      3      1
7      3      3

Matrix * Other matrix:
16     23     16
16     20     16
22     29     22

Check if Matrix is diagonal: false
Check if Matrix is identity: false
Check if Other Matrix is diagonal: false
Check if Other Matrix is identity: false
Check if two matrices are equal: false
```

INHERITANCE/POLYMORPHISM

Shape class: It is an abstract class

Constructors and getter methods:

```
public abstract class Shape {
    private double x;
    private double y;

    public Shape() {
        this( x: 0.0, y: 0.0);
    }

    public Shape(double x, double y) {
        setLocation(x,y);
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public void setLocation(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Methods:

Equals: check if two shapes' contents are equal or not

```
public boolean equals(Object obj) {
    if (obj instanceof Shape) {
        Shape other = (Shape) obj;
        return Objects.equals(other.x, x)
            && Objects.equals(other.y, y);
    } else
        return false;
}
```

Abstract methods perimeter and area: return the perimeter and area of the shape (specific shapes declared in subclasses)

testShape: print out the perimeter and area of the shape (along with other shapes' information, in subclasses)

toString: represent the shape's data

```
public abstract double perimeter();

public abstract double area();

public void testShape() {
    System.out.println("Perimeter: " + perimeter());
    System.out.println("Area: " + area());
}

public String toString() {
    return String.format("%.1f, %.1f", x, y);
}
```

Circle class: extends the Shape class

Constructors and getter method:

```
public class Circle extends Shape {
    private double radius;

    public Circle() {
        this.radius = 1;
    }

    public Circle(double radius) throws IllegalArgumentException {
        if (radius >= 0) {
            this.radius = radius;
        } else {
            throw new IllegalArgumentException("Radius cannot be negative");
        }
    }

    public double radius() {
        return radius;
    }
}
```

Methods: Override the methods in abstract class Shape, with detailed operations

```
@Override
public boolean equals(Object obj) {
    if (obj instanceof Circle) {
        Circle other = (Circle) obj;
        return Objects.equals(other.radius, this.radius);
    } else
        return false;
}

@Override
public double area() {
    return Math.PI * radius * radius;
}

@Override
public double perimeter() {
    return Math.PI * 2 * radius;
}

@Override
public void testShape() {
    System.out.println("Circle ");
    System.out.println("Radius: " + radius());
    super.testShape();
}

@Override
public String toString() { return String.format("%.1f", radius); }
```

Rectangle class: extends the Shape class

Constructors and getter methods:

```
public class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle() {
        this.width = 1;
        this.height = 1;
    }

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double height() {
        return height;
    }

    public double width() {
        return width;
    }
}
```


Methods: Override the methods in the abstract class Shape, with detailed operations

```
@Override
public boolean equals(Object obj) {
    if (obj instanceof Rectangle) {
        Rectangle other = (Rectangle) obj;
        return Objects.equals(other.width, width)
            && Objects.equals(other.height, height);
    } else {
        return false;
    }
}

@Override
public double area() {
    return width * height;
}

@Override
public double perimeter() {
    return 2 * (width + height);
}

@Override
public void testShape() {
    System.out.println("Rectangle ");
    System.out.println("Width: " + width() + "\tHeight: " + height());
    super.testShape();
}
```

toString:

```
@Override
public String toString() {
    return String.format("%.1f, %.1f", width, height);
}
```

Triangle class: extends the Shape class

Constructors and getter methods:

```
public class Triangle extends Shape{
    private double a;
    private double b;
    private double c;

    public Triangle() {
        this(a: 1, b: 1, c: 1);
    }

    public Triangle(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public double getC() {
        return c;
    }
}
```

Methods: Override the methods in the abstract class Shape, with detailed operations

```
@Override
public boolean equals(Object obj) {
    if (obj instanceof Triangle) {
        Triangle other = (Triangle) obj;
        return Objects.equals(other.a, a)
            && Objects.equals(other.b, b)
            && Objects.equals(other.c, c);
    } else
        return false;
}

@Override
public double area() {
    double s = (a + b + c) / 2;
    return Math.sqrt(s * (s - a) * (s - b) * (s - c));
}

@Override
public double perimeter() {
    return a + b + c;
}

@Override
public void testShape() {
    System.out.println("Triangle ");
    System.out.println("Side a: " + getA() + "\tSide b: " + getB() + "\tSide c: " + getC());
    super.testShape();
}
```

MainProgram class:

Test the above classes' operations

```
package shapes;

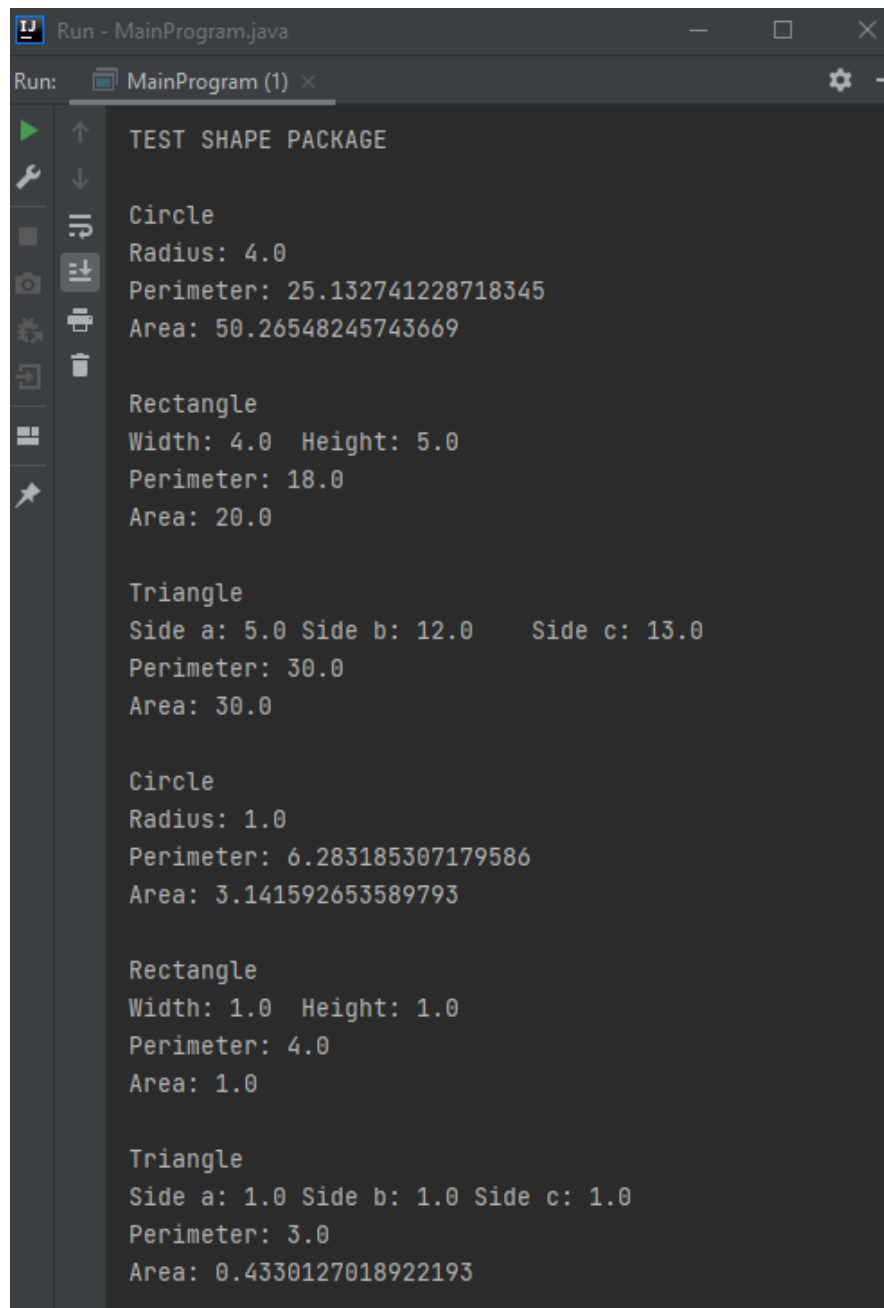
import java.util.ArrayList;

public class MainProgram {
    public static void main(String[] args) {
        //TODO: Create All Classes and Update with test methods
        ArrayList<Shape> shapeList = new ArrayList<>();

        shapeList.add(new Circle( radius: 4));
        shapeList.add(new Rectangle( width: 4, height: 5));
        shapeList.add(new Triangle( a: 5, b: 12, c: 13));
        shapeList.add(new Circle());
        shapeList.add(new Rectangle());
        shapeList.add(new Triangle());

        System.out.println("\nTEST SHAPE PACKAGE");
        for (Shape shape : shapeList) {
            System.out.println();
            shape.testShape();
        }
    }
}
```

OUTPUT:



The screenshot shows a Java IDE's Run window for a file named 'MainProgram.java'. The output is as follows:

```
Run: MainProgram (1) ×

TEST SHAPE PACKAGE

Circle
Radius: 4.0
Perimeter: 25.132741228718345
Area: 50.26548245743669

Rectangle
Width: 4.0 Height: 5.0
Perimeter: 18.0
Area: 20.0

Triangle
Side a: 5.0 Side b: 12.0 Side c: 13.0
Perimeter: 30.0
Area: 30.0

Circle
Radius: 1.0
Perimeter: 6.283185307179586
Area: 3.141592653589793

Rectangle
Width: 1.0 Height: 1.0
Perimeter: 4.0
Area: 1.0

Triangle
Side a: 1.0 Side b: 1.0 Side c: 1.0
Perimeter: 3.0
Area: 0.4330127018922193
```

GUI / INHERITANCE

Truck class: It is an abstract class

Constructors and Data Fields:

```
public abstract class Truck {  
  
    protected Color    color;  
    protected Graphics pen;  
    protected int      size;  
    protected int      startX;  
    protected int      startY;  
  
    //TODO: Update Class, you are expected to figure this out!  
  
    public Truck(Graphics pen) {  
        this.pen = pen;  
    }  
  
    public Truck(Graphics pen, Color color) {  
        this.pen = pen;  
        this.color = color;  
        System.out.println("Truck");  
        setLocation( x: 25, y: 25);  
    }  
}
```

Methods:

drawWheel, drawWindow: to draw parts of the truck

move: move graphics object to a different location in GUI Window

setLocation: set the starting location of the object

setSize: set size to scale and draw graphics object to

```
protected void drawWheel(int outerX, int outerY, int innerX, int innerY, int length) {
    pen.setColor(Color.BLACK);
    pen.fillOval(outerX, outerY, length, length);
    pen.setColor(Color.LIGHT_GRAY);
    pen.fillOval(innerX, innerY, width: length / 2, height: length / 2);
}

protected void drawWindow(int x, int y, int width, int height) {
    pen.setColor(Color.LIGHT_GRAY);
    pen.fillRect(x, y, width, height);
}

public void move(int dx, int dy) {
    this.startX += dx;
    this.startY += dy;
}

public void setLocation(int x, int y) {
    this.startX = x;
    this.startY = y;
}

public int setSize(int size) {
    this.size = size;
    return size;
}
```

draw and drawToScale: abstract methods of abstract class Truck

```
public abstract void draw();

public abstract void drawToScale(int length);
```

Boxed class: extends the Truck class

Constructors: use the color YELLOW to make the “Boxed” Truck

```
public class Boxed extends Truck {  
  
    //TODO: Update Class, you are expected to figure this out!  
    protected static final Color DEFAULT_COLOR = new Color( r: 250, g: 185, b: 0);  
  
    public Boxed(Graphics pen) {  
        super(pen, DEFAULT_COLOR);  
    }  
  
    public Boxed(Graphics pen, Color color) {  
        super(pen, DEFAULT_COLOR);  
        System.out.println("Boxed");  
    }  
}
```

Methods:

Override the draw method in Truck, with detailed actions, using graphics object

```
@Override  
public void draw(){  
    int outerXLeft = startX + 10;  
    int outerXRight = startX + 70;  
    int outerY = startY + 40;  
    int innerXLeft = startX + 15;  
    int innerXRight = startX + 75;  
    int innerY = startY + 45;  
    int windowX = startX + 70;  
    int windowY = startY + 10;  
  
    drawBody( width: 100, height: 50);  
    drawWheel(outerXLeft, outerY, innerXLeft, innerY, length: 20); // left wheel  
    drawWheel(outerXRight, outerY, innerXRight, innerY, length: 20); // right wheel  
    drawWindow(windowX, windowY, width: 30, height: 20);  
}  
  
private void drawBody(int width, int height){  
    pen.setColor(DEFAULT_COLOR);  
    pen.fillRect(startX, startY, width, height);  
}
```

Override the drawToScale method in Truck, resize the "Boxed" truck with graphics object

```
@Override
public void drawToScale(int length) {

    setSize(length);

    int outerXLeft = startX + size / 10;           // x for left wheel
    int outerXRight = startX + 7 * size / 10;       // x for right wheel
    int outerY = startY + 2 * size / 5;             // y for wheels
    int innerXLeft = startX + 3 * size / 20;         // x for left hubcap
    int innerXRight = startX + 3 * size / 4;         // x for right hubcap
    int innerY = startY + 9 * size / 20;            // y for hubcaps
    int windowX = startX + 7 * size / 10;
    int windowY = startY + size / 10;
    int windowWidth = 3 * size / 10;

    drawBody(size, height: size / 2);
    drawWheel(outerXLeft, outerY, innerXLeft, innerY, length: size / 5);
    drawWheel(outerXRight, outerY, innerXRight, innerY, length: size / 5);
    drawWindow(windowX, windowY, windowWidth, height: size / 6);
}
```

FlatBed: extends the Truck class

Constructors: use the color LIGHT BLUE to make the "FlatBed" Truck

```
public class FlatBed extends Truck {

    //TODO: Update Class, you are expected to figure this out!
    protected static final Color DEFAULT_COLOR = new Color(125, 180, 250);

    public FlatBed(Graphics pen) {
        super(pen, DEFAULT_COLOR);
    }

    public FlatBed(Graphics pen, Color color) {
        super(pen, DEFAULT_COLOR);
        System.out.println("Flat Bed");
    }
}
```


Methods:

Override the draw method in the abstract class Truck, with helper method drawFlatBedBody, to draw the FlatBed Truck using graphic objects.

```
@Override
public void draw(){
    int outerXLeft = startX + 10;
    int outerXRight = startX + 70;
    int outerY = startY + 5;
    int innerXLeft = startX + 15;
    int innerXRight = startX + 75;
    int innerY = startY + 10;
    int windowX = startX + 60;
    int windowY = startY - 15;

    drawFlatBedBody(startX, startY);
    drawWheel(outerXLeft, outerY, innerXLeft, innerY, length: 20); // left wheel
    drawWheel(outerXRight, outerY, innerXRight, innerY, length: 20); // right wheel
    drawWindow(windowX, windowY, width: 30, height: 15);
}

private void drawFlatBedBody(int x, int y){
    pen.setColor(DEFAULT_COLOR);
    pen.fillRect(x, y, width: 100, height: 15);
    pen.fillRect(x: x + 50, y: y - 20, width: 50, height: 30);
}
```

Override the drawToScale method in Truck, with the helper method drawResizeBody, to resize the "FlatBed" truck with graphics object

```
private void drawResizedBody(int x, int y){
    pen.setColor(DEFAULT_COLOR);
    pen.fillRect(x, y, size, height: size / 6);
    pen.fillRect(x: x + size / 2, y: y - size / 4, width: size / 2, height: 7 * size / 20);
}

@Override
public void drawToScale(int length) {
    setSize(length);

    int outerXLeft = startX + size / 10; // x for left wheel
    int outerXRight = startX + 7 * size / 10; // x for right wheel
    int outerY = startY + size / 20; // y for wheels
    int innerXLeft = startX + 3 * size / 20; // x for left hubcap
    int innerXRight = startX + 3 * size / 4; // x for right hubcap
    int innerY = startY + size / 10; // y for hubcaps
    int windowX = startX + 6 * size / 10;
    int windowY = startY - 3 * size / 20;
    int windowWidth = 7 * size / 20;

    drawResizedBody(startX, startY);
    drawWheel(outerXLeft, outerY, innerXLeft, innerY, length: size / 5);
    drawWheel(outerXRight, outerY, innerXRight, innerY, length: size / 5);
    drawWindow(windowX, windowY, windowWidth, height: size / 6);
}
```

Display class: already be given by instructor

MainProgram class:

testBoxedTruck: Update the test method for Boxed Truck

```
public static void testBoxedTruck(Display display, Truck truck) throws Exception {  
    // Pause for a bit  
    display.pause( milliseconds: 200);  
  
    // TODO : draw truck at specific location  
    truck.setLocation( x: 50, y: 100);  
    // TODO : move truck to a different location and draw  
    truck.move( dx: 0, dy: -100);  
    // TODO : resize Truck at specific location  
    truck.setLocation( x: 250, y: 50);  
    truck.drawToScale( length: 300);  
    // Pause for a bit  
    display.pause( milliseconds: 200);  
}
```

TestFlatBedTruck: Update the test method for FlatBed Truck

```
public static void testFlatBedTruck(Display display, Truck truck) throws Exception {  
  
    // Pause for a bit  
    display.pause( milliseconds: 200);  
  
    // TODO : draw truck at specific location  
    truck.setLocation( x: 50, y: 315);  
    // TODO : move truck to a different location and draw  
    truck.move( dx: 0, dy: -100);  
    // TODO : resize Truck at specific location  
    truck.setLocation( x: 250, y: 300);  
    truck.drawToScale( length: 300);  
    // Pause for a bit  
    display.pause( milliseconds: 200);  
}
```

Main: create Boxed and FlatBed Truck objects, and do the above tests

```
public static void main(String[] args) throws Exception {

    Display display = new Display( width: 600, height: 500);
    display.setBackground(Color.GRAY);

    // Get Graphics Pen Object for Drawing in Window
    Graphics pen = display.getGraphics();

    // TODO : Create a Boxed Truck for display
    Boxed boxedTruck = new Boxed(pen);

    boxedTruck.draw();
    testBoxedTruck(display,boxedTruck);

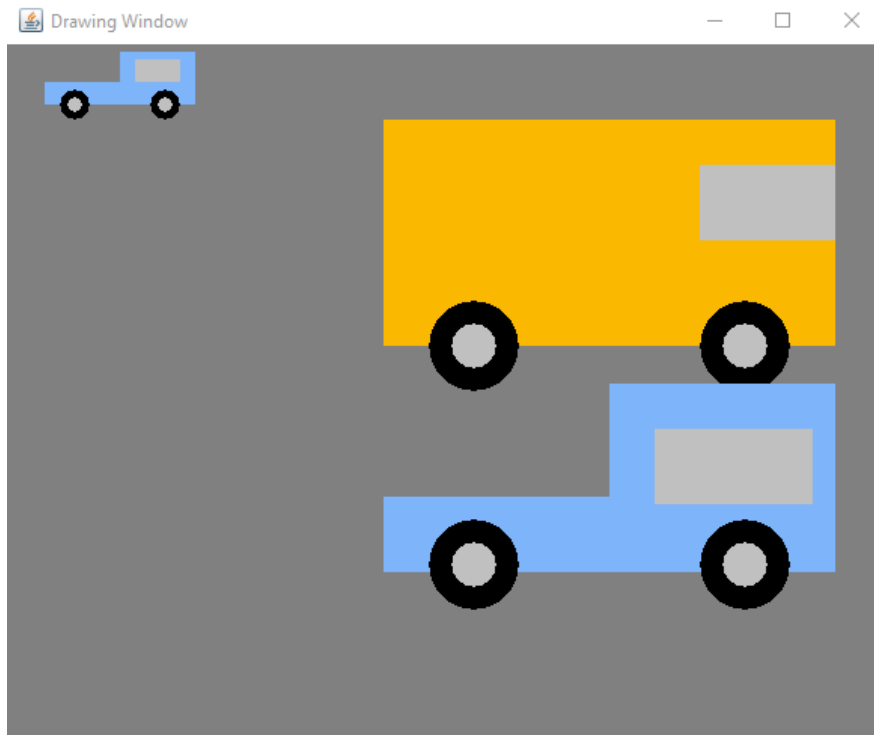
    // clear graphics (if you want to)

    // TODO : with same reference, to create a FlatBed Truck for display
    FlatBed flatBedTruck = new FlatBed(pen);

    flatBedTruck.draw();
    testFlatBedTruck(display,flatBedTruck);

}
```

OUTPUT:



INTERFACE:

Drawable Interface: create an interface with two methods draw and move

Draw: draw outlines of graphics object in GUI Window

Move: translate object to a different location

```
public interface Drawable {  
    void draw(Color color);  
    void move(int dx, int dy);  
}
```

Fillable Interface: create an interface with three methods and one constant color AQUA

Draw: draw outlines of graphics object using a color AQUA in GUI Window

Fill: draw a sequence on circles in GUI Window.

setLocation: set the starting location of circle

```
public interface Fillable {  
    Color AQUA = new Color( r: 0, g: 150, b: 225);  
    void draw();  
    void fill(Color color);  
    void setLocation(int i, int y);  
}
```

Circles class: implements the two above interfaces

Constructors:

```
public class Circles implements Drawable, Fillable{  
    private Color color;  
    private Graphics pen;  
    private int x;  
    private int y;  
  
    public Circles(Graphics _graphics) {  
        pen = _graphics;  
        color = new Color( r: 255, g: 255, b: 255);  
        x = 30;  
        y = 5;  
    }  
}
```

Methods: Override the methods in the interfaces using graphic objects

```
public void setLocation(int x, int y) {
    this.x = x;
    this.y = y;
}

@Override
public void draw() {
    move( dx: 80, dy: 80);
    color = Fillable.AQUA;
    draw(color);
}

@Override
public void draw(Color color) {
    pen.setColor(color);
    for (int i = 1; i < 11; i++) {
        pen.drawOval(x, y, width: 20 * i, height: 20 * i);
    }
}

@Override
public void fill(Color color) {
    move( dx: 50, dy: 50);
    pen.setColor(color);
    for (int i = 1; i < 11; i++) {
        pen.fillOval(x, y, width: 10 * i, height: 10 * i);
    }
}

@Override
public void move(int dx, int dy) {
    x += dx;
    y += dy;
}
```

Display class: already be given

MainProgram class:

testDrawable: create circles of a "Drawable" Type

```
7
8 @ public static void testDrawable(Display display, Drawable circles) throws Exception {
9
10     // TODO : Move to location and draw Yellow Circles
11     circles.move( dx: 50, dy: 0);
12     circles.draw(new Color( r: 255, g: 255, b: 0));
13     // TODO : Move to location and draw RED Circles
14     circles.move( dx: 0, dy: 100);
15     circles.draw(new Color( r: 255, g: 0, b: 0));
16     // TODO : Move to location and draw Circles with Fillable AQUA
17     circles.move( dx: 0, dy: 100);
18     circles.draw(Fillable.AQUA);
19     display.pause( milliseconds: 500);
20 }
```

testFillable: create circles of a "Fillable" Type

```
2  @ public static void testFillable(Display display, Fillable filledCircles) throws Exception {
3
4      //TODO: Set Location back to location (50, 50)
5      filledCircles.setLocation(x: 0, y: 0);
6      display.pause( milliseconds: 500);
7
8      // TODO : Draw fillable RED, WHITE and BLUE Circles
9      filledCircles.fill(new Color( r: 255, g: 0, b: 0));
10     filledCircles.fill(new Color( r: 255, g: 255, b: 255));
11     filledCircles.fill(new Color( r: 0, g: 0, b: 255));
12     filledCircles.draw();
13     // TODO : Draw circles with Fillable AQUA Color
14     display.pause( milliseconds: 500);
15 }
```

Main: create objects and do the above tests

```
public static void main(String[] args) throws Exception{
    Display display = new Display( width: 600, height: 500);
    display.setBackground(Color.DARK_GRAY);

    // Get Graphics Pen Object for Drawing in Window
    Graphics pen = display.getGraphics();

    // TODO : Create Circles of a Drawable Type
    Drawable drewCircles = new Circles(pen);

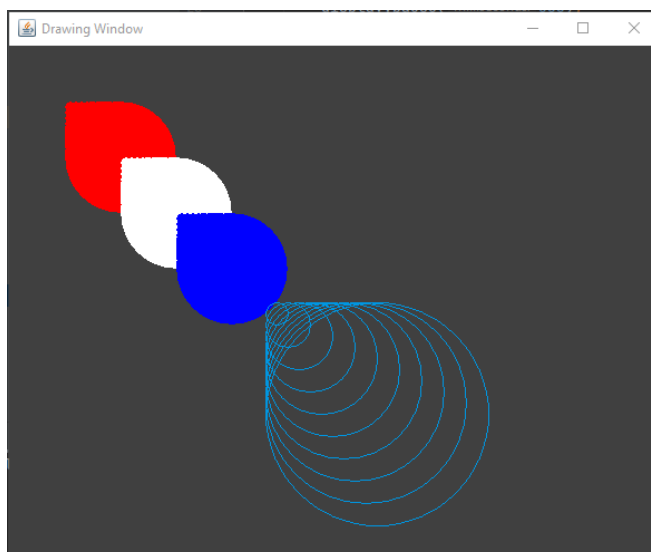
    // TODO : Draw Drawable Circles
    testDrawable(display,drewCircles);

    // clear display if needed (uncomment)
    display.clear();

    // TODO : Create Circles of a Fillable Type
    Fillable filledCircles = new Circles(pen);

    // TODO : Draw Fillable Circles
    testFillable(display,filledCircles);
}
```

OUTPUT:



ZYBOOKS:

SeatReservationFrame class: create a program to manage the seat reservation. We can enter the seat number, customers' names, and the amount paid. Then it will add this information to the table to save the data. Also, it will notice if the seat has already been added, or is not in the table.

Below is the original program (on ZyBooks):

Screenshot:

The screenshot shows a window titled "Seat reservation" with a table of reservations and input fields below it.

Seat Number	First Name	Last Name	Amount Paid
0	Martha	Hobson	45
1	John	Boraxle	60
2	Gene	Baxter	7000
3	Javier	Georges	65
4	Kyle	Ferrani	57

Below the table are input fields for "Seat Number:", "First Name:", "Last Name:", and "Amount Paid:". The "Seat Number" field contains "4", "First Name" contains "Kyle", "Last Name" contains "Ferrani", and "Amount Paid" contains "\$57". There are "Reserve" and "Quit" buttons.

Screenshot:

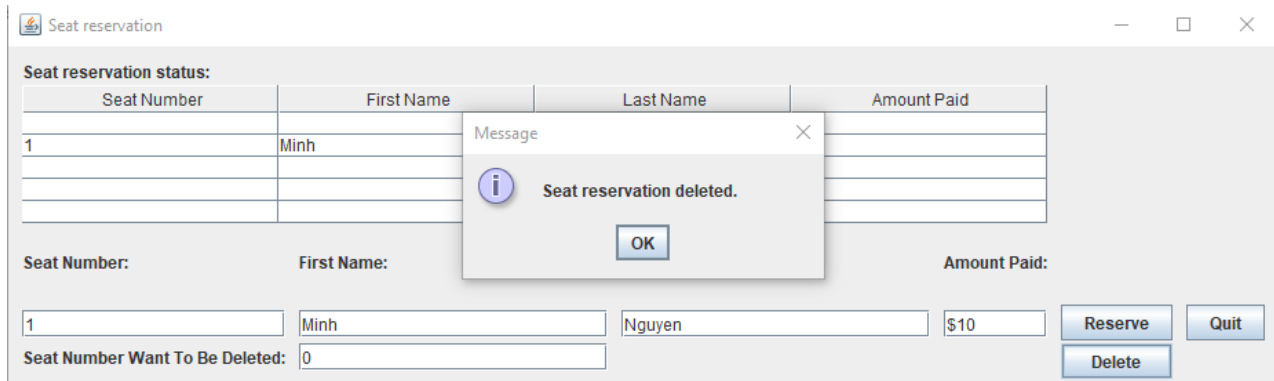
The screenshot shows the same window as the previous one, but with a "Message" dialog box open in the center. The dialog box has a Java logo and the text "Seat is not empty!". There is an "OK" button in the dialog box. The input fields in the background are now "Seat Number: 4", "First Name: John", "Last Name: Walker", and "Amount Paid: \$57".

Modify the SeatReservationFrame program to have an additional JFormattedTextField and JButton component for the purposes of deleting a particular seat reservation. The JFormattedTextField component should allow the user to enter the seat number that should be deleted, and the JButton should trigger the deletion.

Below is the program after adding the "Delete" feature

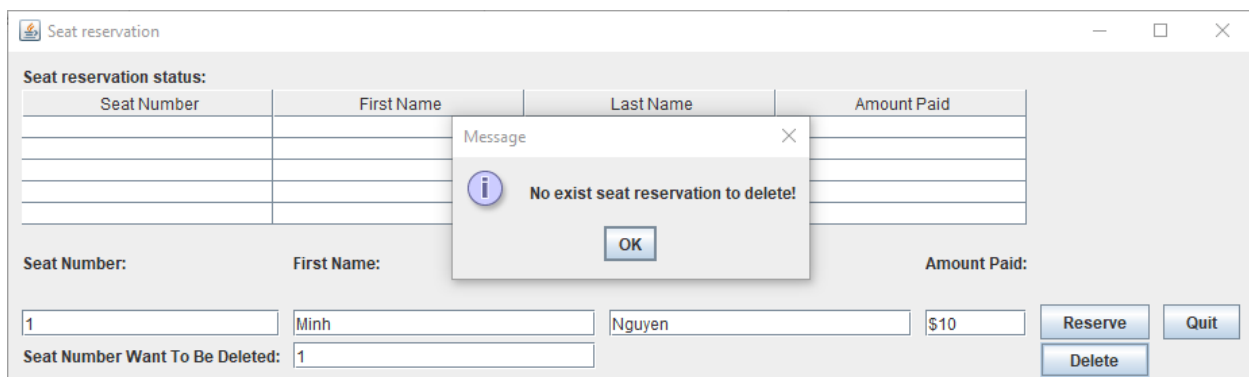
The screenshot shows the modified "Seat reservation" window. It has a table with 5 empty rows for reservations. Below the table are input fields for "Seat Number:", "First Name:", "Last Name:", and "Amount Paid:". The "Seat Number" field contains "0", "First Name" contains "John", "Last Name" contains "Doe", and "Amount Paid" contains "\$10". There are "Reserve" and "Quit" buttons. Below these is a new input field labeled "Seat Number Want To Be Deleted:" containing "0", and a "Delete" button.

After we enter the seat number want to be deleted, this seat's data is removed



The screenshot shows the 'Seat reservation' application window. A modal message box is displayed in the center with the text 'Seat reservation deleted.' and an 'OK' button. In the background, the application window has a title bar 'Seat reservation' and a table titled 'Seat reservation status:' with columns 'Seat Number', 'First Name', 'Last Name', and 'Amount Paid'. The table contains one row with '1', 'Minh', 'Nguyen', and '\$10'. Below the table, there are input fields for 'Seat Number:', 'First Name:', 'Last Name:', and 'Amount Paid:', each with a corresponding value. At the bottom, there is a 'Seat Number Want To Be Deleted:' field with the value '0'. On the right side, there are 'Reserve', 'Quit', and 'Delete' buttons.

If we try to delete an empty seat, there is a popup like this

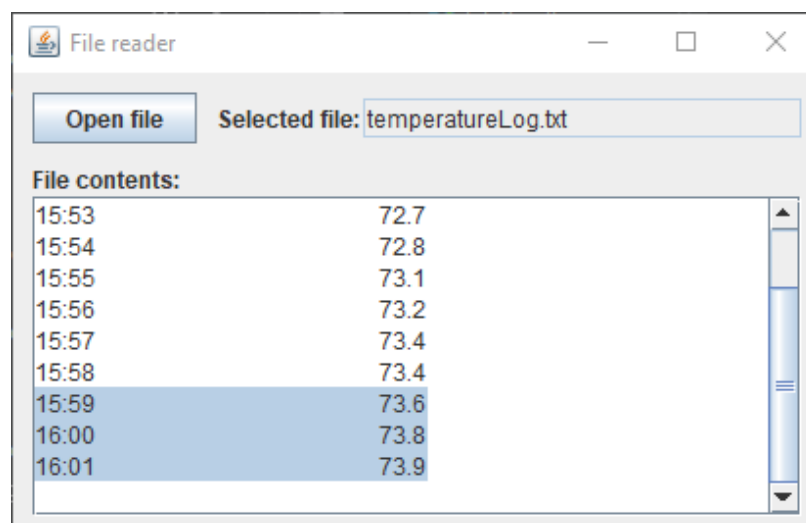


The screenshot shows the 'Seat reservation' application window. A modal message box is displayed in the center with the text 'No exist seat reservation to delete!' and an 'OK' button. In the background, the application window has a title bar 'Seat reservation' and a table titled 'Seat reservation status:' with columns 'Seat Number', 'First Name', 'Last Name', and 'Amount Paid'. The table is empty. Below the table, there are input fields for 'Seat Number:', 'First Name:', 'Last Name:', and 'Amount Paid:', each with a corresponding value. At the bottom, there is a 'Seat Number Want To Be Deleted:' field with the value '1'. On the right side, there are 'Reserve', 'Quit', and 'Delete' buttons.

Reading files with a GUI

The GUI let us search and open the "temperatureLog.txt" file that is located in the data folder. It helps us to search through the disks, folders, and files. Additionally, it has some other features, such as create a new folder, go back (Up One Level) when we go to the wrong directory. Also, it can change the view of folders, like view as list, or details. And it also allows user to search for the file by its name or type.

Modify the "temperatureLog.txt" file in the "data" folder to add 3 more values (highlighted) you made up.



The screenshot shows the 'File reader' application window. It has a title bar 'File reader' and a button 'Open file'. Below the button, there is a text field 'Selected file:' with the value 'temperatureLog.txt'. Below this, there is a section titled 'File contents:' with a list of data. The list has two columns: time and temperature. The data is as follows:

Time	Temperature
15:53	72.7
15:54	72.8
15:55	73.1
15:56	73.2
15:57	73.4
15:58	73.4
15:59	73.6
16:00	73.8
16:01	73.9

The last three rows (15:59, 16:00, 16:01) are highlighted in blue.