LAB #2

Thanh-Sach LE
LTSACH@hcmut.edu.vn

March 30, 2020

1 Compile and run project "DSA2020_GUI"

I have shared will you a project named **"DSA2020_GUI"** in the cloud folder[1] shared with you at the begining of the course.

Question 1

Do the following tasks:

1. Compile the project
2. Explore the code (with helps from Mr. LAM and Mr. PHU) and try to understand it.

2 Add features to "DSA2020_GUI"

2.1 Class SpaceMapping

Question 2

Using the document accompanied with material[a] to create class **SpaceMapping** in package **geom** in **"DSA2020_GUI"** .

[a] File: **SpaceMapping.pdf**

2.2 Print Mouse's Location

Question 3

Do the tasks below:

1. Create an instance of **SpaceMapping** in class **WorkingPanel** .
2. Allow classWorkingPanel to implement methods defined in interface **ComponentListener** [a].
3. Register the instance of classWorkingPanel as a processor for ComponentListener's events.
4. Add code to methods defined in **MouseMotionListener** and **MouseMotionListener** to show the mouse's position as in Figure 2.

[a] java.awt.event.ComponentListener

[1] Folder **Project/Project1** from: The Cloud Folder

2.2.1 Print Mouse's Location: Guideline

1. Create an instance of **SpaceMapping** in class **WorkingPanel**
   - declare and initialize a data field inside of **WorkingPanel** , see Line 5, Figure 1.
2. Allow classWorkingPanel to implement methods defined in interface **ComponentListener** [2].
   - implements interface **ComponentListener** for **WorkingPanel** , see Line 3, Figure 1. You **have to** add all methods defined in **ComponentListener** to class **WorkingPanel** .
   - add implementation for method componentResized , see Line 28, Figure 1. Please note that, you **need to** call this.spaceMapping.updateDevViewPort to update the viewport in the device. This viewport depends on the size of the working panel, so you need to change it accordingly.
3. Register the instance of classWorkingPanel as a processor for ComponentListener's events.
   - call addComponentListener(this) to register **WorkingPanel** as a lister to component's change.
4. Add code to methods defined in **MouseMotionListener** and **MouseMotionListener** to show the mouse's position as in Figure 2.
   - call method this.spaceMapping.device2Logic(e.getX(), e.getY()); to convert mouse position in device space to the logic space.
   - Format and print string, see method mouseMoved , mouseDragged in Figure 1 for more detail.

3 Basic geometry objects

This section helps you to create basic geometry objects as following:

3.1 Class GeomObject

This class is the parent for all geometrical objects. So, it contains common data fields, constants, and methods that should be supported by geometrical objects. Therefore, as you can see in Figure 3, **GeomObject** contains data field edgeColor to keep the color for drawing the objects, and faceColor to keep the color for filling the objects. It also contains method draw for drawing the object with given the graphics and the space mapping objects.

Question 4

Do the following tasks:

1. Create a package named **geom** and Class **GeomObject** as shown in Figure 3.
2. Fill the constructor of Class **GeomObject** to initialize Field edgeColor with a color[a] created from the following intensities: red = 20, green = 200, blue = 20;
3. Fill the constructor of Class **GeomObject** to initialize Field faceColor with red color.

[a] Please refer to Java Color for more information on using Java Color.

[2] java.awt.event.ComponentListener

```java
public class WorkingPanel extends JPanel
    implements MouseMotionListener, MouseListener,
    ItemListener, ActionListener, ComponentListener{

    SpaceMapping spaceMapping = new SpaceMapping();
    //here: more code
    public WorkingPanel(){
        this.setBorder(BorderFactory.createEtchedBorder());
        this.addMouseMotionListener(this);
        this.addMouseListener(this);
        this.addComponentListener(this);
    }
    @Override
    public void mouseDragged(MouseEvent e) {
        Point2D logPoint = this.spaceMapping.device2Logic(e.getX(), e.getY
            ());
        String message = String.format("mouseDragged: Device(x,y)=(%d,%d);
            Logic(x,y)=(%6.2f, %6.2f)", e.getX(), e.getY(), logPoint.getX(),
             logPoint.getY());
        MainFrame.infoPanel.println(message);
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        Point2D logPoint = this.spaceMapping.device2Logic(e.getX(), e.getY
            ());
        String message = String.format("mouseMoved: Device(x,y)=(%d,%d);
            Logic(x,y)=(%6.2f, %6.2f)", e.getX(), e.getY(), logPoint.getX(),
             logPoint.getY());
        MainFrame.infoPanel.println(message);
    }
    //here: similarly for other mouse events
    @Override
    public void componentResized(ComponentEvent e) {
        Dimension size = this.getSize();
        int xGap = 50, yGap = 20;
        this.spaceMapping.updateDevViewPort(xGap, size.width-2*xGap, yGap,
            size.height-2*yGap);
    }

    @Override
    public void componentMoved(ComponentEvent e) {}

    @Override
    public void componentShown(ComponentEvent e) {}

    @Override
    public void componentHidden(ComponentEvent e) {}
}
```

Figure 1: Class WorkingPanel: sample code

3.2 Class Point2D

Class **Point2D** is a subclass of **GeomObject** , it means that an instance of **Point2D** is a geometrical object. It overrides method draw to draw itself.

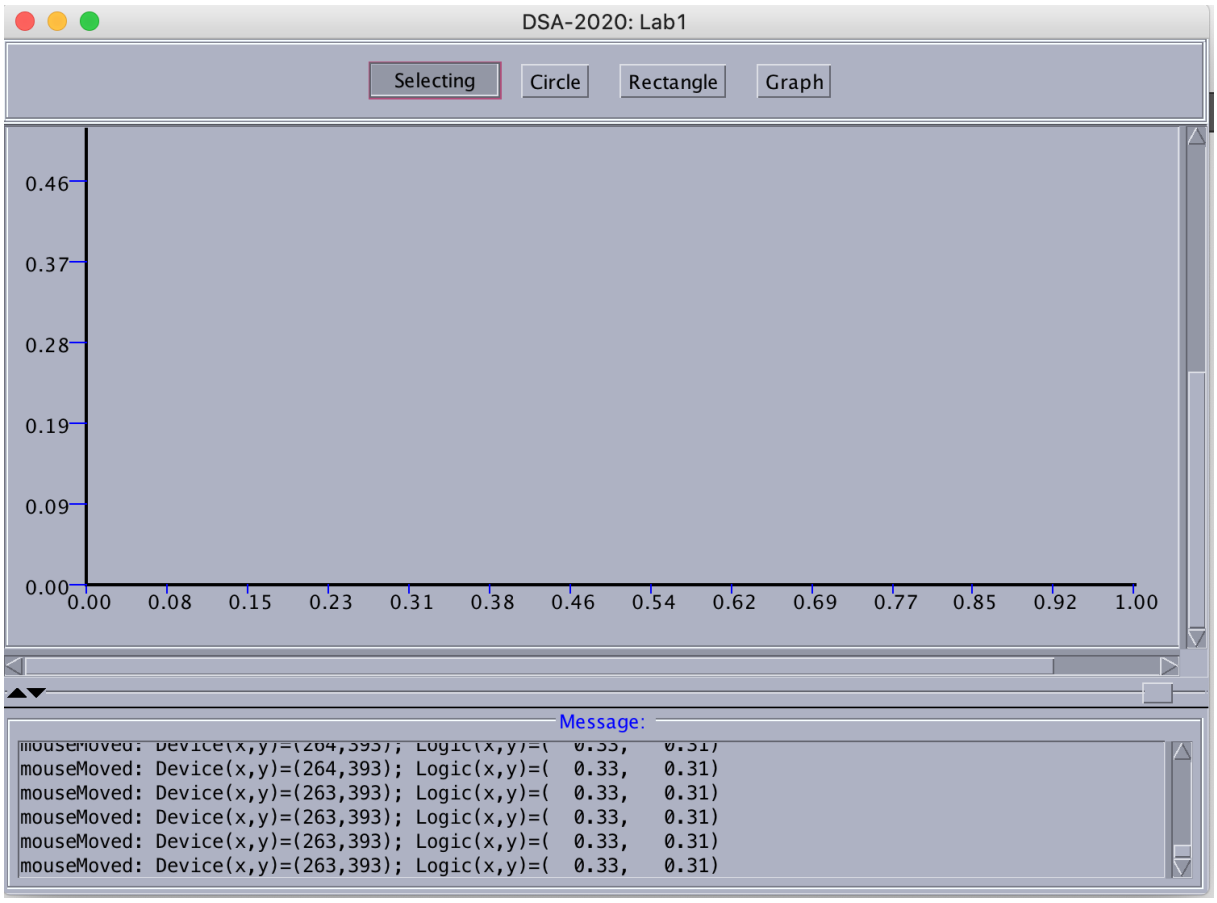

Figure 2: Demo application for Question 2: students are required to modify **"DSA2020_GUI"** to show the mouse's position in the device space and the logic space as shown in **message** areas of the GUI. Adding New button and drawing axis are exluded from Question 2.

```
1  package geom;
2  import java.awt.*;
3
4  public abstract class GeomObject {
5      protected Color edgeColor;
6      protected Color faceColor;
7      protected int line_weight = 1;
8      public GeomObject(){
9          /*Answer here*/
10     }
11     public abstract void draw(Graphics g, SpaceMapping mapper);
12 }
```

Figure 3: Class GeomObject

Question 5

Do the following tasks:

1. Create Class **Poin2D** in Package **geom** as shown in Figure 4. You need to fill the code at the location marked as **Reuse code from LAB1** with your answer for LAB1.
2. Modify the code to make Class **Point2D** be a subclass of Class **GeomObject** .
3. Fill method linear at the location marked as **/*answer here*/** to do:
   - create a point with y-coordinate computed by $y = a * x + b$
   - add this point to array "list" at index "idx".
4. Read the document of fillRect and then fill the code in method draw at the location marked as **/*answer here*/** .
5. Read document of fillOval then replace Line 53 in Figure 4 to show the point as a circle centering at the point and having $\text{radius} = POINT\_HALF\_SIZE$.

Please refer to Java Color for more information on using Java Color.

3.3 Drawing using java.awt.Graphics

In Java, to draw objects in 2D, we can use API supported by package **java.awt.Graphics** or **java.awt.Graphics2D** (Graphics2D is subclass of Graphics).

Students may have a question now. How do we obtain an instance of **Graphics** or **Graphics2D** ? The answer is simple, you need to step:

1. Create a class subclassed from class **javax.swing.JPanel** .
2. Override method paintComponent . paintComponent is called from Swing framework whenever it needs to redraw the component. It will pass an instance of Graphics to paintComponent . You can check class **WorkingPanel** in the project supplied to you.

API of **Graphics** and **Graphics2D** **only work with coordinates in a device space** . Therefore, in order to call draw , we need to pass an instance of class **SpaceMapping** . This class supports API for converting coordinates from the device to the logic space and vice versa.

In this Lab, points are drawn as a square of size $2 \times \text{POINT\_HALF\_SIZE} + 1$; where, POINT_HALF_SIZE is a static property of **Point2D** . As shown in Figure 4, the algoritm to draw a point is simple as explained in the following.

1. Line 46: "Graphics2D g2 = (Graphics2D) g;" It casts instance of Graphics to Graphics2D to have more functionalities for drawing with 2D objects.
2. Line 47: call method logic2device to convert logical coordinates to device coordinates, i.e., in unit of pixel.
3. Line 49-50: the result of previous step is the center of the square beging drawn for the point. So we need to shift to obtain the left-top corner of the square.
4. Line 52: set color of the square by using data field faceColor .
5. Line 53: perform the drawing by calling method fillRect of Graphics2D.

3.4 Graph

A graph can be a polyline (e.g. quadratic or linear line in Figure 5) or and sequence of dots (e.g. sin function in Figure 5). Graphs are usually used to visualize function or relation between variable x and y. Functions are defined in an infinite range, so you need viewport for viewing them. Therefore, class **Graph** must maintain the following information, see Figure 6 for an overview.

1. Viewport on the logic space for viewing graph. Remember, a viewport maintains four values: xMin, xMax, yMin, and yMax (see class **Viewport**).
2. List of points, look at sin function on Figure 5.
3. Mode of drawing; we draw as dots like sin function on Figure 5 or quadratic function on Figure 5.
4. Graph needs to extend from class **GeomObject** to inherit data fields and methods of geometrical objects.

Method copyPoints (Line 77-84): assigning array of points received from the parameter to the data field inside of Graph is not enough. A graph must maintain its viewport; so, it given a list of points, copyPoints must do two tasks as follows:

1. copies the point list from the parameter.
2. finds **the smallest rectangle** that contains all the point int the list. It does this by: (a) initialize the viewport to contain only the first, and (b) for each point in the list it enlarge the viewport to contain the point (see method addPoint in class **Viewport**).

Question 6

Do the following tasks:

1. Create class **Graph** from the source given partially in Figure 6, 7 and 8.
2. Try to understand the idea of method quaratic to generate a quaratic graph using parameters given.
3. Copy the idea of quadratic generation, fill code for method sin in Figure 7.

Question 7

Do the following tasks:

1. Add button **Graph** to toolbar, see Figure 5.
2. When users press on button **Graph** do:
   - Create a sin graph and a quaratic graph.
   - Draw it on the screen.

3.4.1 Drawing graph of functions: guideline

1. Add button **Graph** to toolbar, see Figure 5.
   - See Question 1 for how to do
2. When users press on button **Graph** do:
   - You have to place the code for drawing in method paintComponent of class **WorkingPanel** ;
   - but, you have to intialize the drawing from event processed for pressing button, i.e., in method actionPerformed .
   - From method actionPerformed you **can not** call paintComponent directly. Therefore, these two methods must communicate via common data fields in class **WorkingPanel** .

     (a) Declare two graphs (named sin , and quad) as data fields in **WorkingPanel** .

(b) intialize sin , and quad in actionPerformed by calling to Graph.sin and Graph.quadratic respectively.

(c) Inside of paintComponent , you check if sin / quad is not null, you call draw for sin and quad . Remember, you need to pass instance of SpaceMapping (data field of class **WorkingPanel**) to draw to allow graphs doing coordinate conversion.

```java
package geom;
import java.awt.*;
import java.util.Random;
public class Point2D /*answer here: Point2D is a subclass of GeomObject*/{
    public static int POINT_HALF_SIZE = 2;
    private double x;
    private double y;
    //Add setters and getters here: Reuse code from LAB1
    public Point2D(){
        this.x = 0; this.y = 0;
    }
    public Point2D(double x, double y){
        this.x = x; this.y = y;
    }
    public Point2D(Point2D oldPoint){
        this.x = oldPoint.x; this.y = oldPoint.y;
    }
    public Point2D clone(){
        return new Point2D(this.x, this.y);
    }
    public String toString(){
        return String.format("P(%6.2f, %6.2f)", this.x, this.y);
    }
    public static Point2D[] generate(int N, double min, double max){
        //Reuse code from LAB1
    }
    public static Point2D[] linear(int N, double a, double b, double xMin,
       double xMax){
        Point2D[] list = new Point2D[N];
        double step = (xMax - xMin)/(N-1); //xMax inclusive
        double x = xMin;
        for(int idx=0; idx < N; idx++){
            /*answer here*/
            x += step;
        }
        return list;
    }
    public static double distanceAB(Point2D a, Point2D b){
        //Reuse code from LAB1
    }
    public double distanceTo(Point2D point){
        //Reuse code from LAB1
    }

    @Override
    public void draw(Graphics g, SpaceMapping mapper) {
        Graphics2D g2 = (Graphics2D) g;
        Point2D point = mapper.logic2Device(this.getX(), this.getY() );

        int x = (int)point.getX() - POINT_HALF_SIZE;
        int y = (int)point.getY() - POINT_HALF_SIZE;

        g2.setColor(this.faceColor);
        g2.fillRect(/*anwser here*/);
    }
}
```

Figure 4: Class Point2D

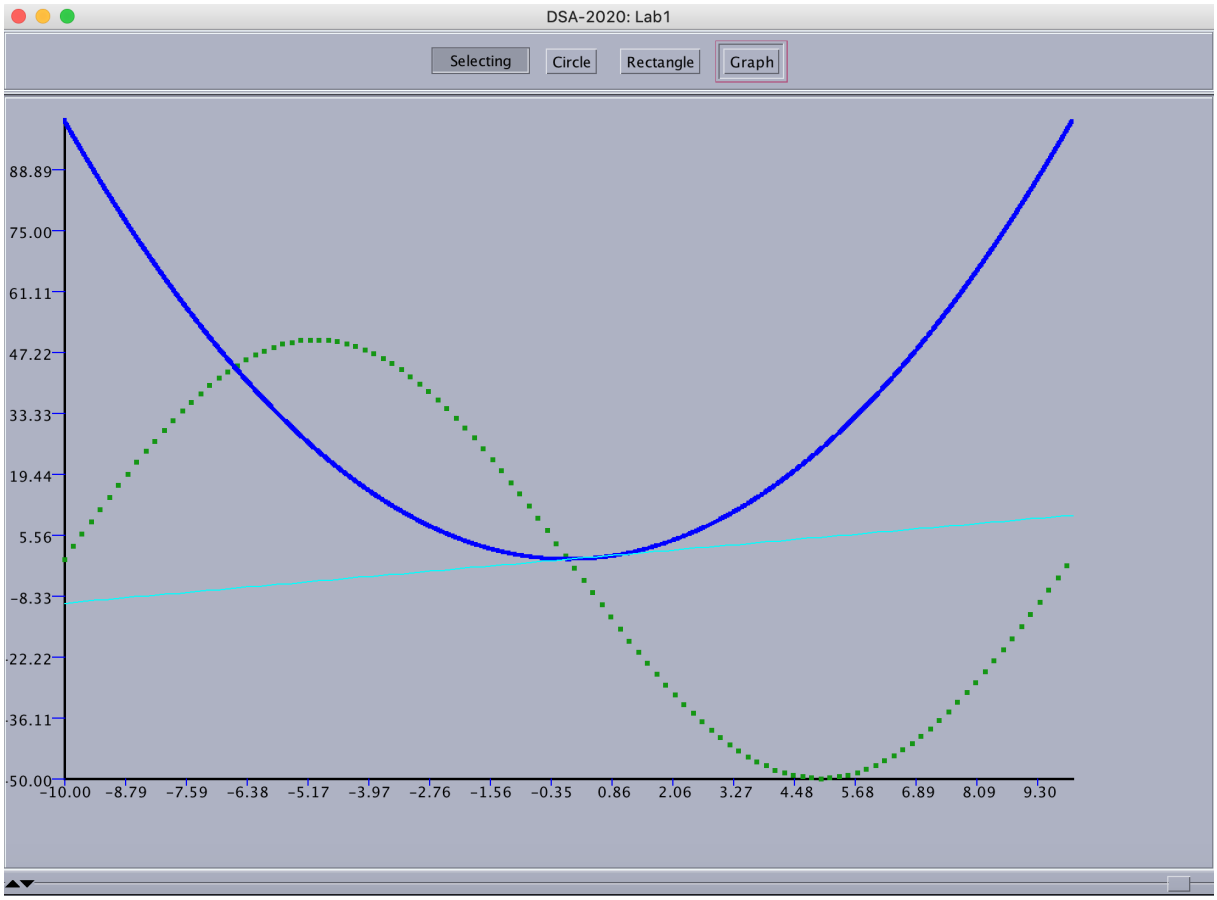

Figure 5: Axis and graph

```
package geom;
import java.awt.*
public class Graph extends GeomObject{
    public enum GraphMode {
        LINE,
        SCATTER
    };
    protected Viewport viewport = null;
    private Point2D[] points = null;
    private GraphMode mode = GraphMode.LINE;

    //setter and getter for mode
    public Graph(Point2D[] points, double xMin, double xMax, double yMin,
        double yMax){
        this.viewport = new Viewport(xMin, xMax, yMin, yMax);
        this.points = points;
    }
    public Graph(Point2D[] points, double xMin, double xMax, double yMin,
        double yMax, Color color){
        this.viewport = new Viewport(xMin, xMax, yMin, yMax);
        this.points = points;
        this.edgeColor = color;
    }
    private void copyPoints(Point2D[] points){
        this.points = points;
        //update viewport
        this.viewport = new Viewport(points[0].getX(), points[0].getX(),
        points[0].getY(), points[0].getY());
        for(int idx=0; idx < points.length; idx++)
        this.viewport.addPoint(points[idx]);
    }
    public Graph(Point2D[] points){
        copyPoints(points);
    }
    public Graph(Point2D[] points, Color color){
        copyPoints(points);
        this.edgeColor = color;
    }
    @Override
    public void draw(Graphics g, SpaceMapping mapper) {
    }
    public static Graph sin(double a, double xMin, double xMax, double step
        ){
    }
    public static Graph quadratic(double a, double b, double c, double xMin
        , double xMax, double
    }
}
```

Figure 6: Class Graph: an overview of methods and data fields

```
101 package geom;
102 import java.awt.*
103 public class Graph extends GeomObject{
104     //here: more code
105     public static Graph sin(double a, double xMin, double xMax, double step
        ){
106         /*answer here/
107     }
108     public static Graph quadratic(double a, double b, double c, double xMin
        , double xMax, double step){
109         int N = (int)((xMax - xMin)/step) + 1;
110         Point2D[] points = new Point2D[N];
111         double x = xMin;
112         for(int idx = 0; idx < N; idx++){
113         double y = a*x*x + b*x + c;
114         points[idx] = new Point2D(x, y);
115         x += step;
116         }
117         return new Graph(points);
118     }
119
120     //here: more code
121 }
```

Figure 7: Class Graph: Generating graph

```java
package geom;
import java.awt.*
public class Graph extends GeomObject{
    //here: more code
    @Override
    public void draw(Graphics g, SpaceMapping mapper) {
        Graphics2D g2 = (Graphics2D) g;
        //
        if(this.mode == GraphMode.LINE){
            if(this.points == null) return;
            int[] x = new int[this.points.length];
            int[] y = new int[this.points.length];
            for(int idx=0; idx < this.points.length; idx++){
                Point2D devPoint = mapper.logic2Device(this.points[idx]);
                x[idx] = (int)devPoint.getX();
                y[idx] = (int)devPoint.getY();
            }
            g2.setColor(this.edgeColor);
            Stroke style = new BasicStroke(this.line_weight);
            g2.setStroke(style);
            g2.drawPolyline(x, y, x.length);
        }
        else if(this.mode == GraphMode.SCATTER){
            for(int idx=0; idx < this.points.length; idx++){
                this.points[idx].draw(g, mapper);
            }
        }
        else{
            throw new UnsupportedOperationException("Not supported yet.");
        }
    }
    //here: more code
}
```

Figure 8: Class Graph: Drawing graph