SUPPORTING DOCUMENT FOR LAB #2

Thanh-Sach LE
LTSACH@hcmut.edu.vn

March 30, 2020

1 Device space and Logic Space

1.1 Device space

Device space is a one for working with GUI-framework, for examples, Java Swing. This space has the following features and it is illustrated in Figure 1.

1. It is discrete, the coordinate is represented by **integer value** .
2. Its origin is always at the **left-top corner** of the GUI-component, for example JPanel.
3. The y-axis is **upside down** , so the y-coordinate is increased from the left-top corner down to the bottom of the screen.

Students can verify the above features with the application accompanied with this tutorial (source of the app. included).

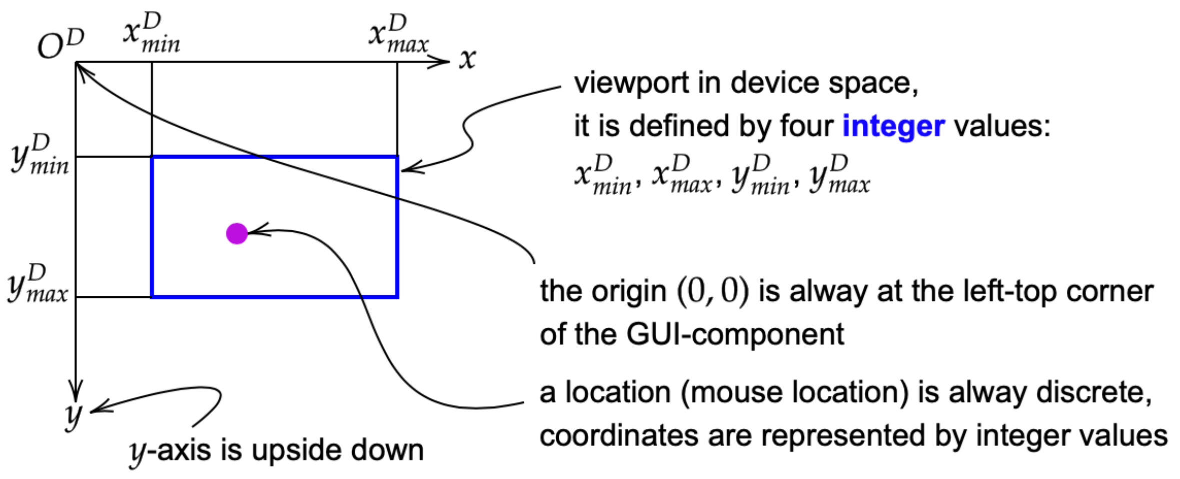

O^D
x^D_{min}
x^D_{max}
x
y^D_{min}
y^D_{max}
y
viewport in device space,
it is defined by four integer values:
$x^D_{min}, x^D_{max}, y^D_{min}, y^D_{max}$
the origin $(0, 0)$ is alway at the left-top corner
of the GUI-component
a location (mouse location) is alway discrete,
coordinates are represented by integer values
y-axis is upside down

Figure 1: Device space is **discrete**. Note that $x^D_{min} \leq x^D_{max}$ and $y^D_{min} \leq y^D_{max}$. x^D_{min} is on the left of x^D_{max} and y^D_{min} is **upper** than y^D_{max}

1.2 Logic space

Logic space is the one that all students have studied and used in mathematics, as shown in Figure 2. Some main features are as follows.

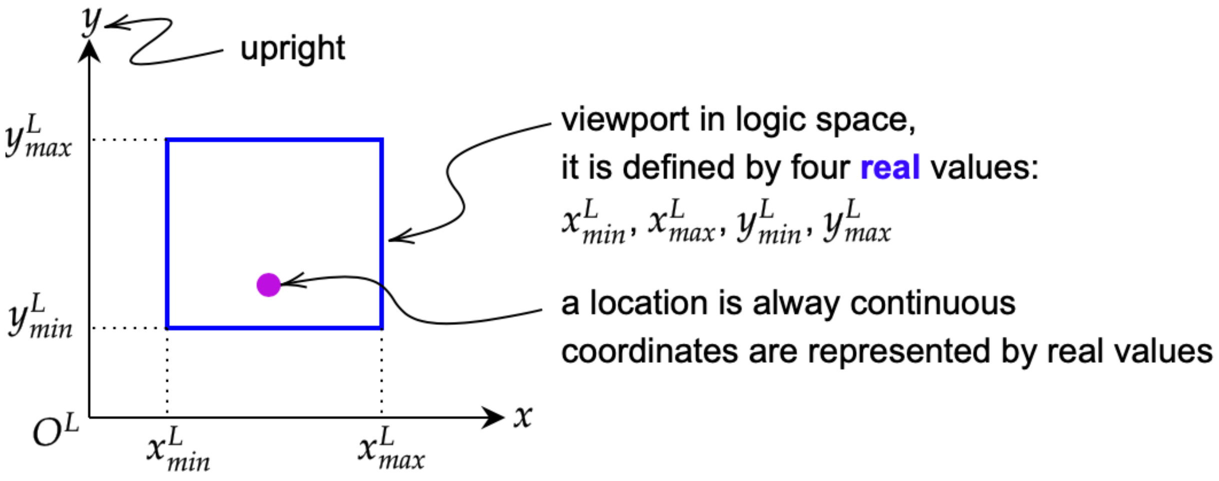
1. It is **continuous** , the coordinate is represented by **floating value** .
2. The y-axis is **upright** .

To create a drawing, the logic space is more general; your can specify your objects in a continuous space instead of a discrete one. It is also easier and more natural for drawing than the device space. For example, if you want to draw a graph, e.g. $\sin x$, you immediately think about and imagine about the logic space, the one with upright y-axis.

1.3 Viewport

Because the logic space is infinitive, so you need to show only an area inside that space. That area is usually **rectangular** and called a **viewport** . A viewport for the logical space is specified by **four floating values** : x^L_{min}, x^L_{max}, y^L_{min} and y^L_{max}.

The logic space is natural for logically thinking about the drawing, but if you want to show your drawing, you need the device space; it means that you need to show your drawing on a specific GUI component. Moreover, you need show your drawing on a specific area on the device space. That are is called a viewport on the device space. That viewport is specified by **four integer values** : x^D_{min}, x^D_{max}, y^D_{min} and y^D_{max}.



upright
viewport in logic space,
it is defined by four real values:
$x^L_{min}, x^L_{max}, y^L_{min}, y^L_{max}$
a location is alway continuous
coordinates are represented by real values
y
x
y^L_{max}
y^L_{min}
O^L
x^L_{min}
x^L_{max}

Figure 2: Logic space is **continuous**. Note that $x^D_{min} \leq x^D_{max}$ and $y^D_{min} \leq y^D_{max}$. x^D_{min} on the left of x^D_{max} and y^D_{min} is **lower** than y^D_{max}

2 Conversion between logic and device space

Logically, Pogramming API in graphic frameworks support users the following steps to create drawing.

1. Support API for users to create drawings on the logic space. Users make drawings by using the space as studied in mathematics. The device space is hidden from users; it means the conversion from the logic space to/from the device space is automated by frameworks.

2. The frameworks convert the drawings to the device space and perform actual drawing tasks on GUI-components. Therefore, if you are a provider of your graphic framework, as in this tutorial, you need to answer **where is on the device space to which a given point on the logic space convert?** ; this task is called **Logic2Device** or **L2D** conversion.

3. The end users uses mouse pointer to interact with the drawings, we can catch the mouse position in the device space. In order to change the graph, we need to answer **where is on the logic space to which a given device point convert?**; this task is called **Device2Logic** or **D2L** conversion.

   The above explantion means that we need to solve a problem or develope an algorithm to determine (x^D, y^D) from (x^L, y^L) and vice versa. Where, (x^D, y^D) is a point in the viewport of the device space; and (x^L, y^L) is a point in the viewport of the logic space, as shown in Figure 3.

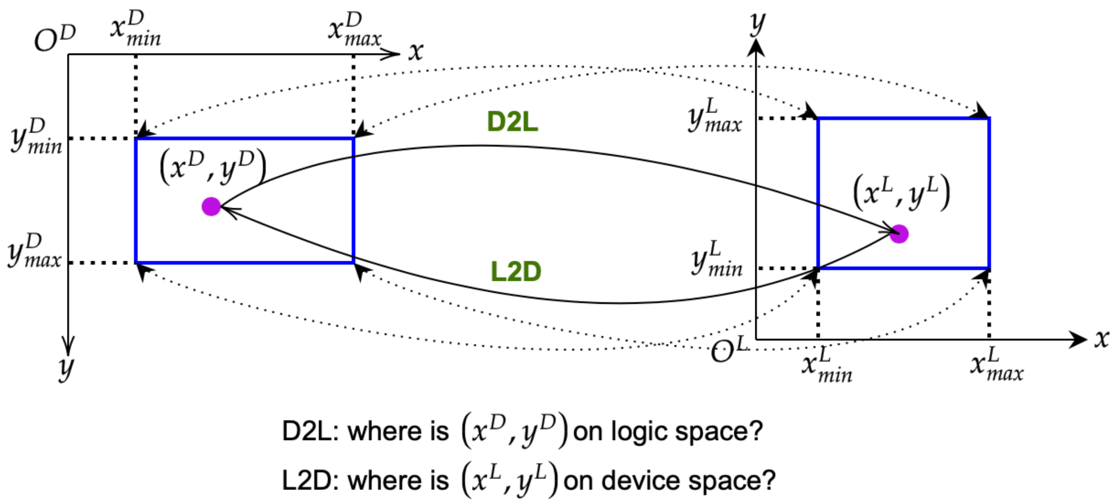


Figure 3: Conversion between space. We map the viewport on the logic device to the viewport on the device space and vice versa.

2.1 Scaling for matching two viewports

We have:

- The viewport on the device space, it is specified by: x^D_{min}, x^D_{max}, y^D_{min} and y^D_{max}
- The viewport on the logic space, it is specified by: x^L_{min}, x^L_{max}, y^L_{min} and y^L_{max}

We can compute the width and the height of the two viewports.

- The width of the viewport on the device space: $W^D = x^D_{max} - x^D_{min}$
- The height of the viewport on the device space: $H^D = y^D_{max} - y^D_{min}$
- The width of the viewport on the logic space: $W^L = x^L_{max} - x^L_{min}$
- The height of the viewport on the logic space: $H^L = y^L_{max} - y^L_{min}$

Viewport mapping, what does it means? We can think as you need to resize a viewport to match with the size of another viewport.

It means that W^D units (pixels) on the device space will be mapped to (corresponds to) W^L units on the logic space. So, **1 unit on the device will be mapped to $\frac{W^L}{W^D}$ units on the logic space** , this factor named as sxD2L; so, $\text{sxD2L} = \frac{W^L}{W^D}$. Moreover, in order to convert d units from device space to logic space, just multiply it with sxD2L; so d units are mapped to $d \times \text{sxD2L}$

In similar way, we have the following results

1. to convert 1 units on x-axis from device to logic, use factor sxD2L: $\text{sxD2L} = \frac{W^L}{W^D}$
2. to convert 1 units on y-axis from device to logic, use factor syD2L: $\text{syD2L} = \frac{H^L}{H^D}$
3. to convert 1 units on x-axis from logic to device, use factor sxL2D: $\text{sxL2D} = \frac{W^D}{W^L}$
4. to convert 1 units on y-axis from logic to device, use factor syL2D: $\text{syL2D} = \frac{H^D}{H^L}$

2.2 The coordinates of the origin of the logic space on the device space

When we perform the conversion, we map the viewport on the logic space to the viewport on the device space and vice versa, see Figure 3. Therefore, we need to map four corners of the two viewport coresspond-ing (**left-bottom to left-bottom** , **left-top to left-top** , **right-top to right-top** , and **right-bottom to right-bottom**).

We come to a question. Where is the location on the device space to which the origin of the logic space map? To answer this question, we try the following reasoning and combination with looking at Figure 3.

- The left-bottom of the viewport on the logic space is mapped to the left-bottom of the viewport on the device. It means $(x^L_{min}, y^L_{min}) \longrightarrow (x^D_{min}, y^D_{max})$; **please note that y^D_{max} instead of y^D_{min}** .
- On the logic space, the origin is displaced from the **left-bottom** of the viewport a vector $\vec{t^L}$ from the **left-bottom** to the origin, so $\vec{t^L} = [-x^L_{min}, -y^L_{min}]$ (L denotes the logic space).
- Conversion of $\vec{t^L}$ to the device by the following two steps:
  - scaling: to convert x^L_{min} and y^L_{min} to the device space; hence, we obtain $\vec{t^D}$; their coordinates are computed according to Equation (1) and (2)

$$\vec{t^D_x} = -\mathrm{sxL2D} \times x^L_{min} \qquad \text{(after scaling)} \tag{1}$$
$$\vec{t^D_y} = -\mathrm{syL2D} \times y^L_{min} \qquad \text{(after scaling)} \tag{2}$$

  - flipping: because y-axis on the device space is upside down (instead of upright as in the logic space), we need to do the reflection about the x-axis (also called flipping). The reflection is simple, just change the sign of y-coordinate; thereby, we obtain the coordinates of $\vec{t^D}$ Equation (3) and (4).

$$\vec{t^D_x} = -\mathrm{sxL2D} \times x^L_{min} \qquad \text{(after scaling and then reflection)} \tag{3}$$
$$\vec{t^D_y} = \mathrm{syL2D} \times y^L_{min} \qquad \text{(after scaling and then reflection)} \tag{4}$$

- Let O^{LinD} be the point on the device space which is mapped from the origin of the logic space. The coordinates of O^{LinD} is obtained by translating the **left-bottom** of the viewport on the device space by vector $\vec{t^D}$ obtained above; this translation is performed by Equation (5) and (6).

$$O^{\mathrm{LinD}}_x = x^D_{min} - \mathrm{sxL2D} \times x^L_{min} \tag{5}$$
$$O^{\mathrm{LinD}}_y = y^D_{max} + \mathrm{syL2D} \times y^L_{min} \tag{6}$$

2.3 Conversion from device to logic

Let $P(x^D, y^D)$ be the point in the device space; x^D and y^D are integer, they are the coordinates of pixel P on the deivce space. Now, we would like to know where is P on the logic space. Please note that P is still there on the device, but we actually want to know its coordinates in the logic space. What all we do is to change the system for measuring the coordinates. The changing is called conversion, it consists of three steps as follows:

1. **Scaling**: this step is to convert the unit of the device space (pixel) to the unit used in the logic space (metric). As explained in Section 2.1, we can use factor sxD2L and syD2L to convert for x- and y-coordinate. Thereby, after the scaling, the coordinates of P in the scaled system are (x^{scaled}, y^{scaled}) and computed according to Equation (7) and (8).

$$x^{scaled} = \mathrm{sxD2L} \times x^D \tag{7}$$
$$y^{scaled} = \mathrm{syD2L} \times y^D \tag{8}$$

2. **Reflection**: this step is to convert a coordinate measured by an upside down axis (y-axis of the device) to an upright axis (as in y-axis of the logic space). We do this task by changing the sign of the y-coordinate. Thereby, after the reflection, the coordinates of p in the reflected system are (x^{ref}, y^{ref}), see Euation (9) and (10). The reflected system is denoted as $O^{ref}xy$.

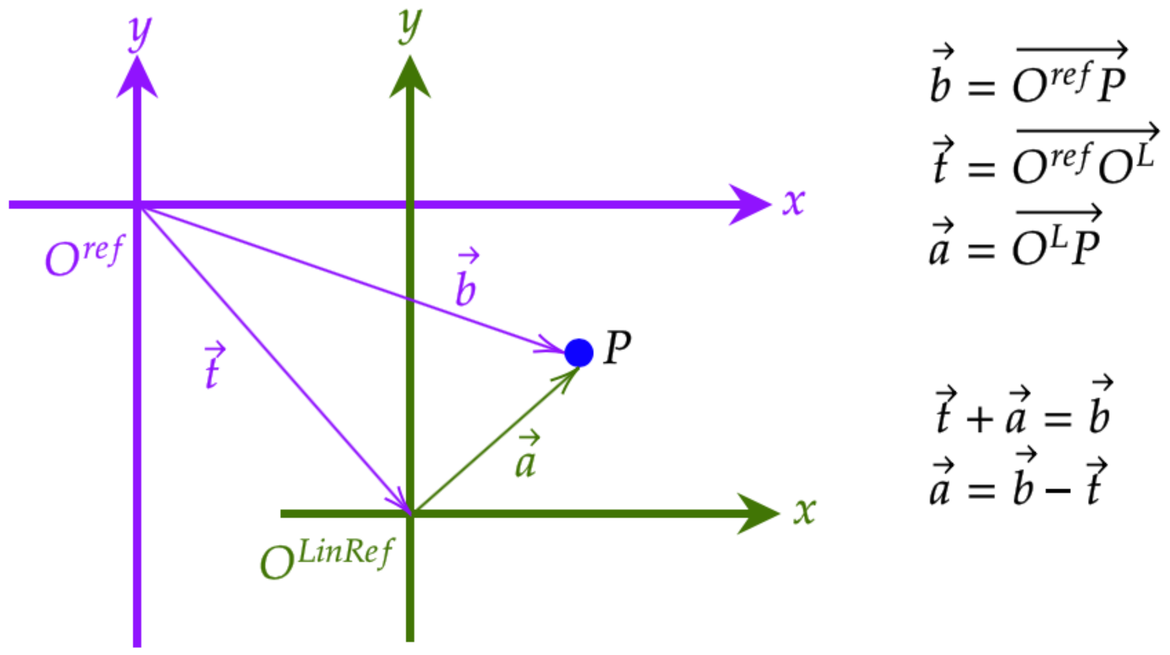
$$x^{ref} = \text{sxD2L} \times x^D \tag{9}$$
$$y^{ref} = -\text{syD2L} \times y^D \tag{10}$$

In Section 2.2, the location of the origin of the logical space on the device space is O^{LinD}. We change its coordinates to $O^{ref}xy$ by the two above steps (scaling and then reflection), we obtain O^{LinRef}; the x- and the y-coordinate of O^{LinRef} are computed using Equation (11) and (12).

$$O_x^{LinRef} = \text{sxD2L} \times O_x^{LinD} \tag{11}$$
$$O_y^{LinRef} = -\text{syD2L} \times O_y^{LinD} \tag{12}$$

3. **Translation**: As shown in Figure 4, the coordinates of P in the logic space is represented by vector $\vec{a}$. Using the **rule of vector addition** , we can infer $\vec{a} = \vec{b} - \vec{t}$ (see Equation (13) and (14); where, $\vec{b}$ represents for the coordinates of P in the reflected system, see Equation (9) and (10). and $\vec{t}$ reprents for the coordinates of the origin of the logic space in the reflected system, see Equation (11) and (12).

$$x^L = \text{sxD2L} \times x^D - O_x^{LinRef} \quad = \text{sxD2L} \times x^D - \text{sxD2L} \times O_x^{LinD} \tag{13}$$
$$y^L = -\text{syD2L} \times y^D - O_y^{LinRef} \quad = -\,\text{syD2L} \times y^D + \text{syD2L} \times O_y^{LinD} \tag{14}$$



y
y
x
x
O^{ref}
$\vec{b}$
$\vec{t}$
P
$\vec{a}$
O^{LinRef}
$\vec{b} = \overrightarrow{O^{ref}P}$
$\vec{t} = \overrightarrow{O^{ref}O^{L}}$
$\vec{a} = \overrightarrow{O^{L}P}$
$\vec{t} + \vec{a} = \vec{b}$
$\vec{a} = \vec{b} - \vec{t}$

Figure 4: Conversion from $O^{ref}xy$ system to O^Lxy ($\equiv O^{LinRef}xy$) system. $\vec{b}$ and $\vec{t}$ is the representation of point P and O^L in $O^{ref}xy$ respectively. $\vec{a}$ is the reprentation of point P in the logical space.

2.4 Conversion from logic to device

$$x^D = \text{sxL2D} \times x^L + O_x^{LinD} \tag{15}$$

$$y^D = -\text{syL2D} \times y^L + O_y^{LinD} \tag{16}$$

Using Equation (13) and (14), we can obtain Equation (15) and (16) for converting from the logic to the device space. Please using $\text{sxL2D} = 1/\text{sxD2L}$ and $\text{sxD2} = 1/\text{sxL2D}$ for transformation of equations.

3 Implementation

3.1 Data fields and constructors

In order to do the mapping, we need to maintain the viewport of the logic space and of the device space. Table 1 shows the relation between data fields used in Class **SpaceMapping** (see Figure 5) and their notation used in mathematical equations in previous sections.

As shown in Figure 5, Class **SpaceMapping** has a default constructor, it initializes the objects with default values for 8 data fields. Those data fields will be updated to actual values by methods updateDevViewPort and updateLogViewPort later.

Table 1: Data fields and their notation in mathematical equations

Index	**Data fields**	**Notation**
1	minDevX	x^D_{min}
2	maxDevX	x^D_{max}
3	minDevY	y^D_{min}
4	maxDevY	y^D_{max}
5	minLogX	x^L_{min}
6	maxLogX	x^L_{max}
7	minLogY	y^L_{min}
8	maxLogY	y^L_{max}

```
public class SpaceMapping {
    private int minDevX, maxDevX;
    private int minDevY, maxDevY;
    private double minLogX, maxLogX;
    private double minLogY, maxLogY;


    public SpaceMapping(){
    this.minDevX = 0;
    this.maxDevX = 800;
    this.minDevY = 0;
    this.maxDevY = 600;

    this.minLogX = 0;
    this.maxLogX = 1.0;
    this.minLogY = 0;
    this.maxLogY = 1.0;
}
//here: has more code (later)
}
```

Figure 5: Class SpaceMapping: data fields and contructors

3.2 Supporting methods

Some methods for supporting the conversion are shown in Figure 6, 7, and 8. Figure 6 contains methods for computing the width (prefixed with letter w) and the height (prefixed with letter h) of the viewports. Letter D and L denote for Device and Logic respectively.

Figure 7 contains methods for scaling units between space; they are sxL2D , syL2D , sxD2L , and syD2L . Method oLinD is to determine the coordinates of the origin of the logic space in the device space, see Equation (5) and (6).

Method step returns the distance in the logic space that corresponds to 5 pixels in the device space. This method helps to rasterize the logic space (it is continuous) to grid of cells that corresponds to n (n=5) pixels in the device space.

Figure 8 contains method for updating the viewports. Parameters of these methods can the limits (min and max on x- and y-axis) or the object of Class **Viewport** . This class is defined in Figure 11.

```
public class SpaceMapping {
    //data fields and constructors
    public double wL(){
        return this.maxLogX - this.minLogX;
    }
    public double hL(){
        return this.maxLogY - this.minLogY;
    }
    public double wD(){
        return this.maxDevX - this.minDevX;
    }
    public double hD(){
        return this.maxDevY - this.minDevY;
    }
    //here: has more code (later)
}
```

Figure 6: Class SpaceMapping: Supporting methods - PART 1

3.3 Conversion between Logic and Device

The conversion the coordinates from the device space to the logic space programed as in Figure 9, the code follows Equation (13) and (14).

Similarly, The conversion the coordinates from the logic space to the device space programed as in Figure 10, the code follows Equation (15) and (16).

```java
public class SpaceMapping {
    //data fields and constructors

    public double sxL2D(){
        double wL = this.maxLogX - this.minLogX;
        double wD = this.maxDevX - this.minDevX;
        return wD/wL;
    }
    public double syL2D(){
        double hL = this.maxLogY - this.minLogY;
        double hD = this.maxDevY - this.minDevY;
        return hD/hL;
    }
    public double sxD2L(){
        double wL = this.maxLogX - this.minLogX;
        double wD = this.maxDevX - this.minDevX;
        return wL/wD;
    }
    public double syD2L(){
        double hL = this.maxLogY - this.minLogY;
        double hD = this.maxDevY - this.minDevY;
        return hL/hD;
    }
    public Point2D oLinD(){
        return new Point2D(
            this.minDevX - this.minLogX*this.sxL2D(),
            this.maxDevY + this.minLogY*this.syL2D()
            );
    }
    public double step(){
        return 5.0*Math.min(this.sxD2L(), this.syD2L());
    }
    //here: has more code (later)
}
```

Figure 7: Class SpaceMapping: Supporting methods - PART II

```java
public class SpaceMapping {
    //data fields and constructors

    public void updateDevViewPort(int minDevX, int maxDevX, int minDevY,
        int maxDevY){
        this.minDevX = minDevX;
        this.maxDevX = maxDevX;
        this.minDevY = minDevY;
        this.maxDevY = maxDevY;
    }
    public void updateLogViewPort(double minLogX, double maxLogX, double
        minLogY, double maxLogY){
        this.minLogX = minLogX;
        this.maxLogX = maxLogX;
        this.minLogY = minLogY;
        this.maxLogY = maxLogY;
    }
    public void updateLogViewPort(Viewport vp){
        this.minLogX = vp.getxMin();
        this.maxLogX = vp.getxMax();
        this.minLogY = vp.getyMin();
        this.maxLogY = vp.getyMax();
    }
    //here: has more code (later)
}
```

Figure 8: Class SpaceMapping: Supporting methods - PART III

```java
public class SpaceMapping {
    //data fields and constructors
    //Supporting methods

    public Point2D device2Logic(Point2D devPoint){
        return device2Logic((int)devPoint.getX(), (int)devPoint.getY());
    }
    public Point2D device2Logic(int devX, int devY){
        double txD2L = this.oLinD().getX() * sxD2L();
        double tyD2L = - this.oLinD().getY() * syD2L(); // "-" means
            flipped
        double logX =  sxD2L()*devX - txD2L;
        double logY = -syD2L()*devY - tyD2L;
        return new Point2D(logX, logY);
    }
    //here: has more code (later)
}
```

Figure 9: Class SpaceMapping: Device to Logic conversion

```java
public class SpaceMapping {
    //data fields and constructors
    //Supporting methods
    public Point2D logic2Device(Point2D logPoint){
        return logic2Device(logPoint.getX(), logPoint.getY());
    }
    public Point2D logic2Device(double logX, double logY){
        double txL2D = this.oLinD().getX();
        double tyL2D = -this.oLinD().getY();
        double devX = sxL2D()*logX + txL2D;
        double devY = -syL2D()*logY - tyL2D;
        return new Point2D(devX, devY);
    }
}
```

Figure 10: Class SpaceMapping: Logic to Device Conversion

```java
public class Viewport {
    //Setter and Getter
    public double getyMin() { return yMin; }
    public void setyMin(double yMin) { this.yMin = yMin; }
    public double getyMax() { return yMax; }
    public void setyMax(double yMax) { this.yMax = yMax; }
    public double getxMin() { return xMin; }
    public void setxMin(double xMin) { this.xMin = xMin; }
    public double getxMax() { return xMax; }
    public void setxMax(double xMax) { this.xMax = xMax; }
    //Data fields
    private double xMin, xMax;
    private double yMin, yMax;

    //Constructors
    public Viewport(double xMin, double xMax, double yMin, double yMax){
        this.xMin = xMin; this.xMax = xMax;
        this.yMin = yMin; this.yMax = yMax;
    }
    public Viewport clone(){
        return new Viewport(this.xMin, this.xMax, this.yMin, this.yMax);
    }
    public void combine(Viewport vp){
        this.xMin = Math.min(this.xMin, vp.xMin);
        this.xMax = Math.max(this.xMax, vp.xMax);
        this.yMin = Math.min(this.yMin, vp.yMin);
        this.yMax = Math.max(this.yMax, vp.yMax);
    }
    public void addPoint(Point2D point){
        this.xMin = Math.min(this.xMin, point.getX());
        this.xMax = Math.max(this.xMax, point.getX());
        this.yMin = Math.min(this.yMin, point.getY());
        this.yMax = Math.max(this.yMax, point.getY());
    }
}
```

Figure 11: Class Viewport