CSC 133

Object-Oriented Computer Graphics Programming

Interactive Techniques I

Dr. Kin Chung Kwan

Spring 2023

Computer Science Department
California State University, Sacramento

SACRAMENTO STATE

Last Lecture

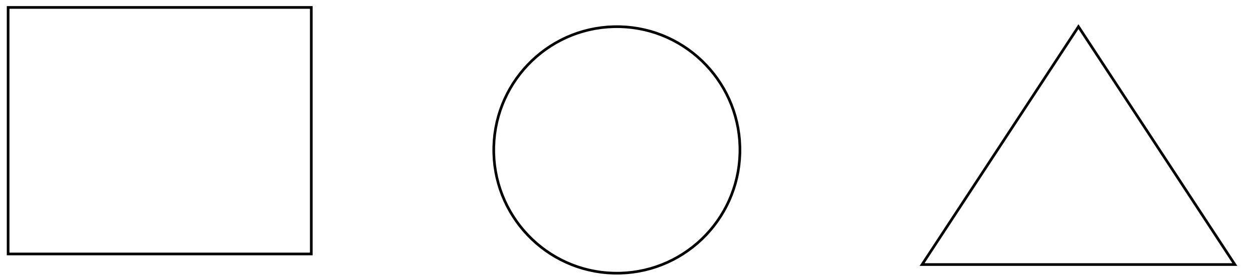
- UI Component
  - Button, label, checkbox, etc.
  - Form
  - Containers: Sub-area of a form

- Event
  - Trigger when interact with components
  - Auto call **`actionperformed()`**

Graphics

- How to draw 2D graphic in CN1?

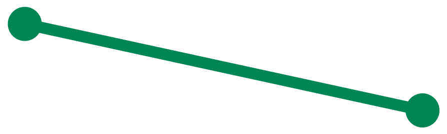
- Not pasting images
  - But drawing rectangles/circles/triangles



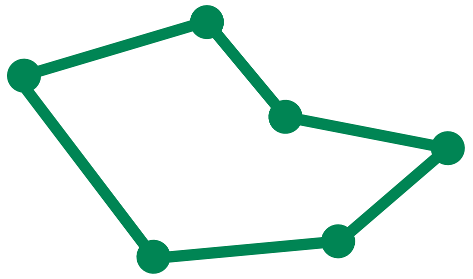
Basic Graphics Elements

Vertex / Point
- (x, y)

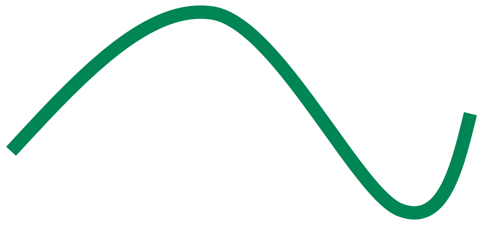
Line
- Two points



Polygon
- Multiple closed lines

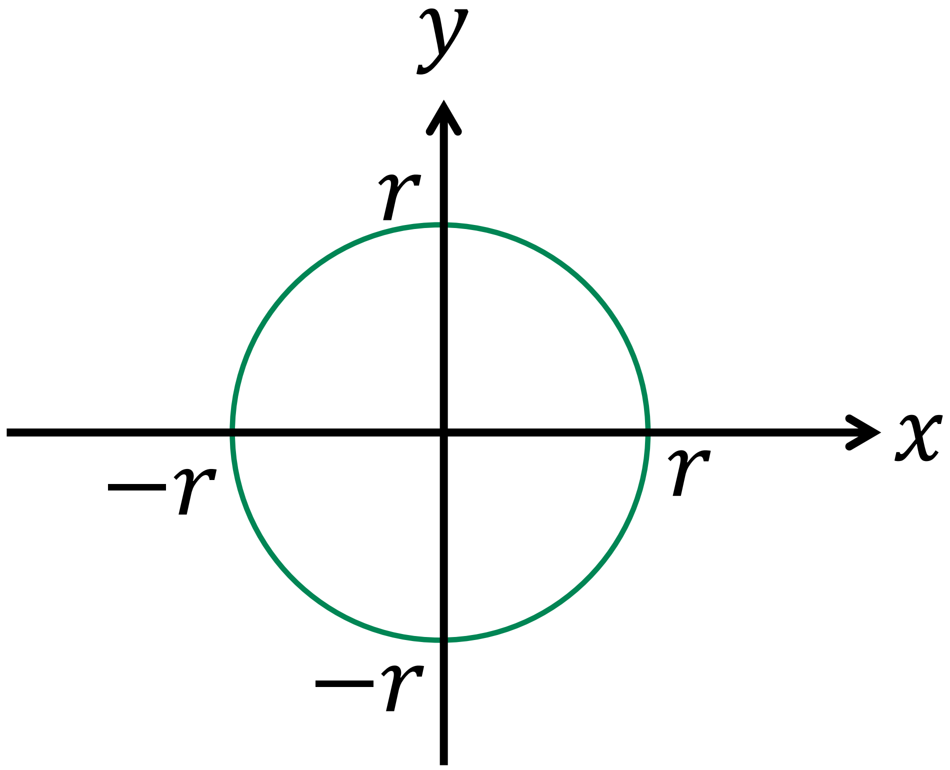


Curve
- Equations



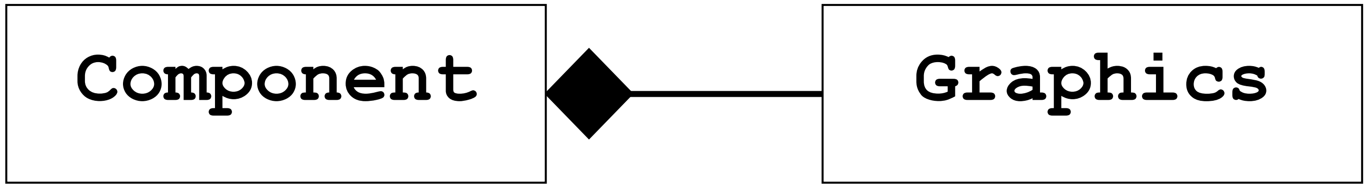
Circle

$$\boldsymbol{x}^2 + \boldsymbol{y}^2 = \boldsymbol{r}^2$$



Component Graphics

- Every `Component` has a `Graphics` object



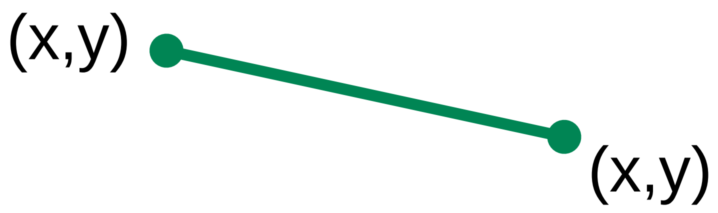
- `Graphics` objects handle the drawing

Graphics Class

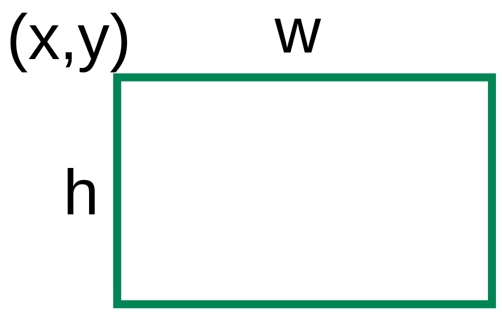
- Contain methods to draw on their components
- `drawLine (int x1, int y1, int x2, int y2)`
- `drawRect (int x, int y,  int width, int height)`
- `fillRect (int x, int y,  int width, int height)`
- `drawArc (int x, int y,  int width, int height,`
  `int startAngle, int arcAngle)`
- `fillArc(int x, int y, int width, int height,`
  `int startAngle, int arcAngle)`
- `drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`
- `drawString (String str,  int x,  int y)`
- `setColor (int RGB)`
- Etc.

Methods

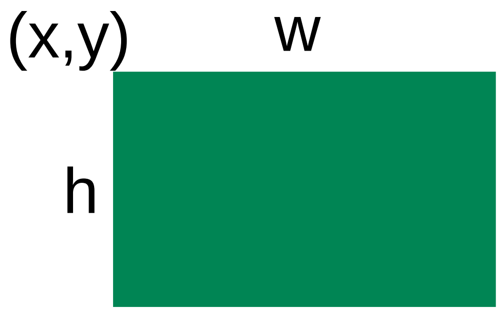
drawLine



drawRect



fillRect

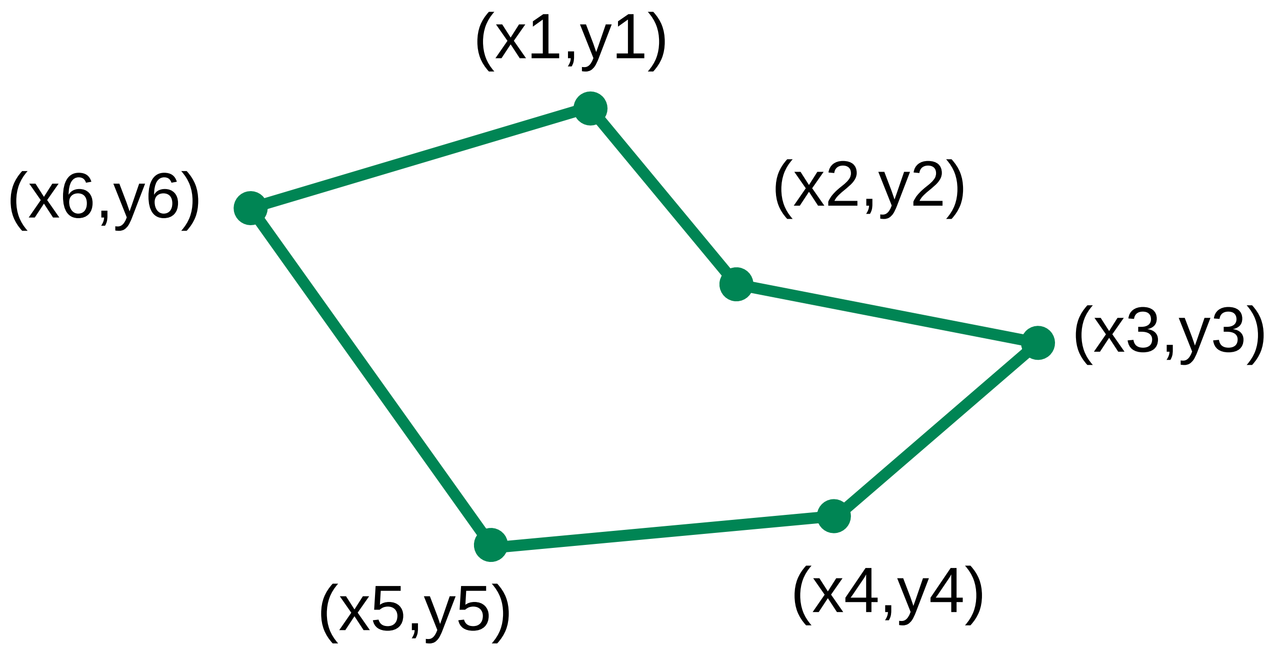


drawString

Text

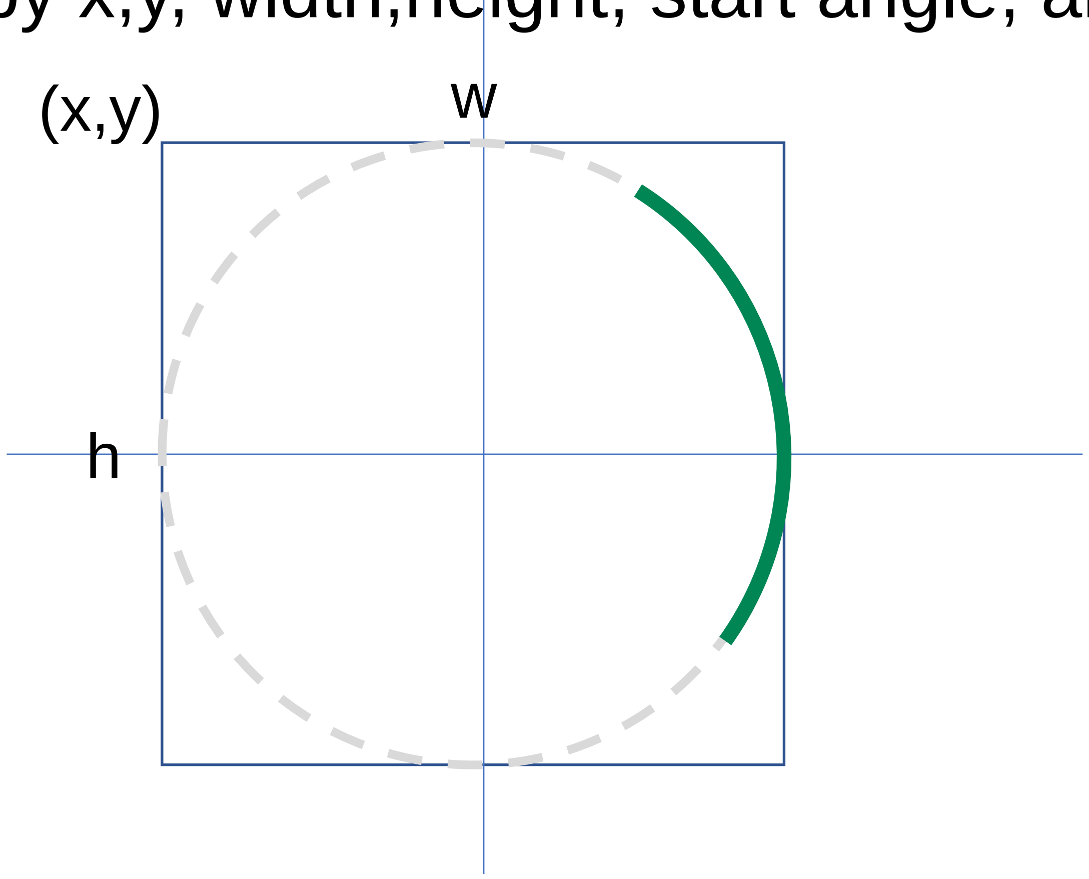
drawPolygon

- Defined by (x1,x2,…,xn) and (y1,y2,…,yn)



drawArc

- Defined by x,y, width,height, start angle, angle



Reference to `Graphics`

- To call these drawing methods
  - get `Graphics` object of a component
  - How?

- ~~`getGraphics()?`~~
  - Not supported in CN1

- Component repainting mechanism
  - `repaint()` & `paint()`

Repaint

- Every `Component` has `repaint()` method
  - Update component’s appearance
- Can be called automatically or manually
  - Opening App for the first time
  - User switched back to the app while multi-tasking among different apps
  - Changing styles such as `setBgColor(int RGB)`

Paint

- `Component` also has `paint()` method
  - `paint()` is responsible for the actual drawing
  - E.g., drawing line

  - `repaint()` update the component by calling `paint()` method with `Graphics`

Paint vs Repaint

paint()	**repaint()**
- Actually drawing	- Update drawing
- By using graphics obj	- By calling paint()
- Override it	- Call it
- Never call it directly	- Call it directly

Drawing Graphics

- **Override** `paint()` method in the component
  - `public void paint(Graphics g);`

- Provide a `Graphics` object for you to draw

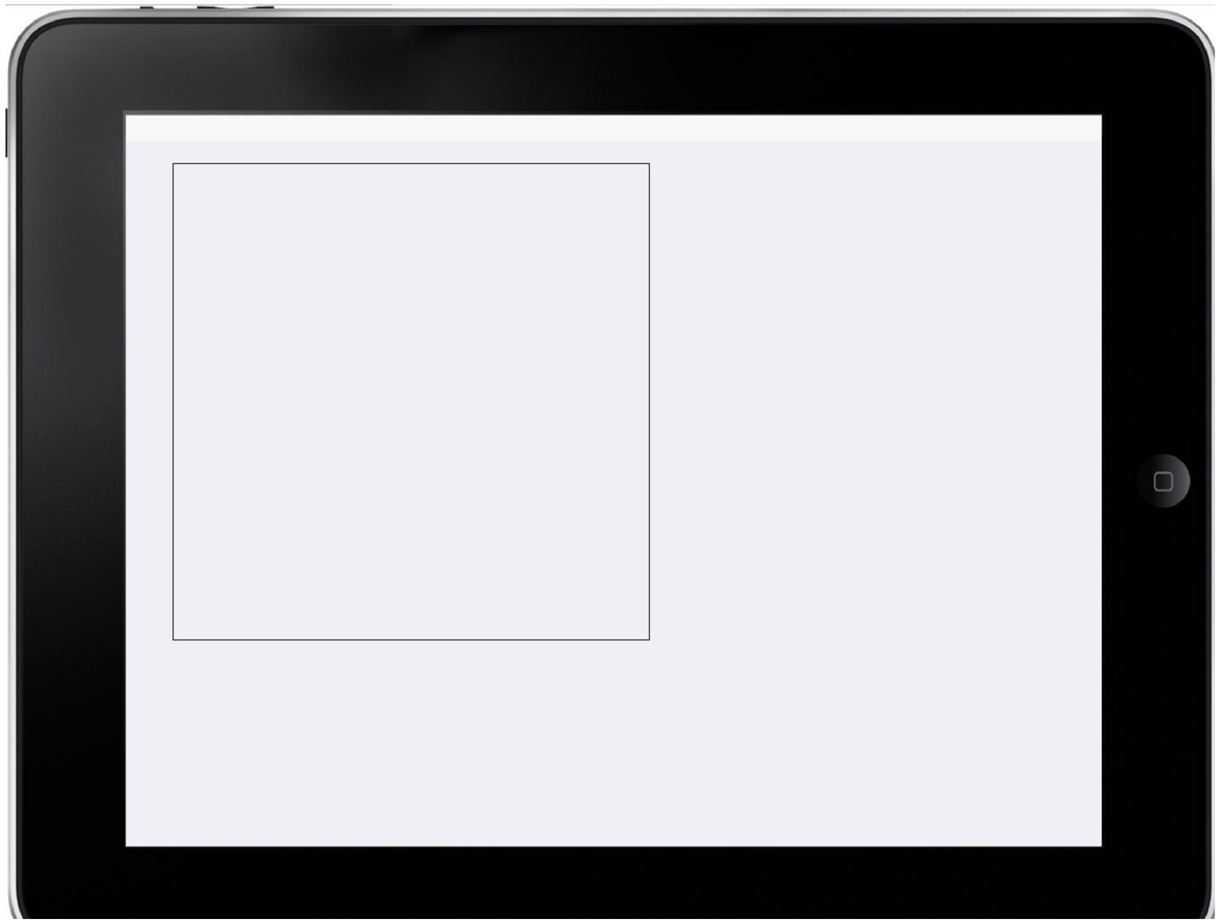
- Remember to call `super.paint()`
  - Performs other important operations
  - Necessary for drawing

Example

```
public class MyForm extends Form {
  public MyForm() {
     show();
  }
  public void paint(Graphics g){
     super.paint(g);
     g.drawRect(100, 100, 1000, 1000);
  }
}
```

Result

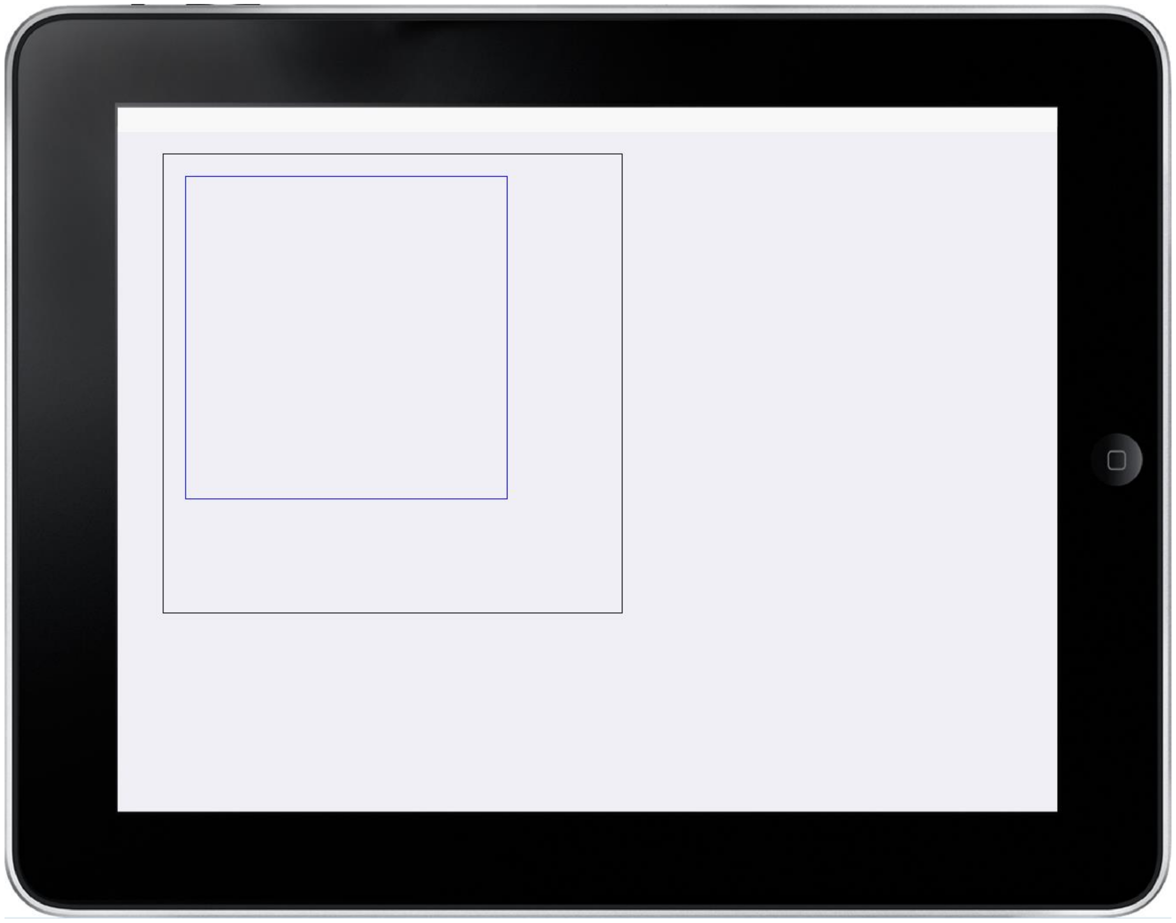
```
g.drawRect(100, 100, 1000, 1000);
```



Further Example

```
public class MyForm extends Form {
  public MyForm() {
     show();
  }
  public void paint(Graphics g){
    super.paint(g);
    g.setColor(ColorUtil.Black);
    g.drawRect(100, 100, 1000, 1000);
    g.setColor(ColorUtil.BLUE);
    g.drawRect(150, 150, 700, 700);
  }
}
```

Result



Code Order

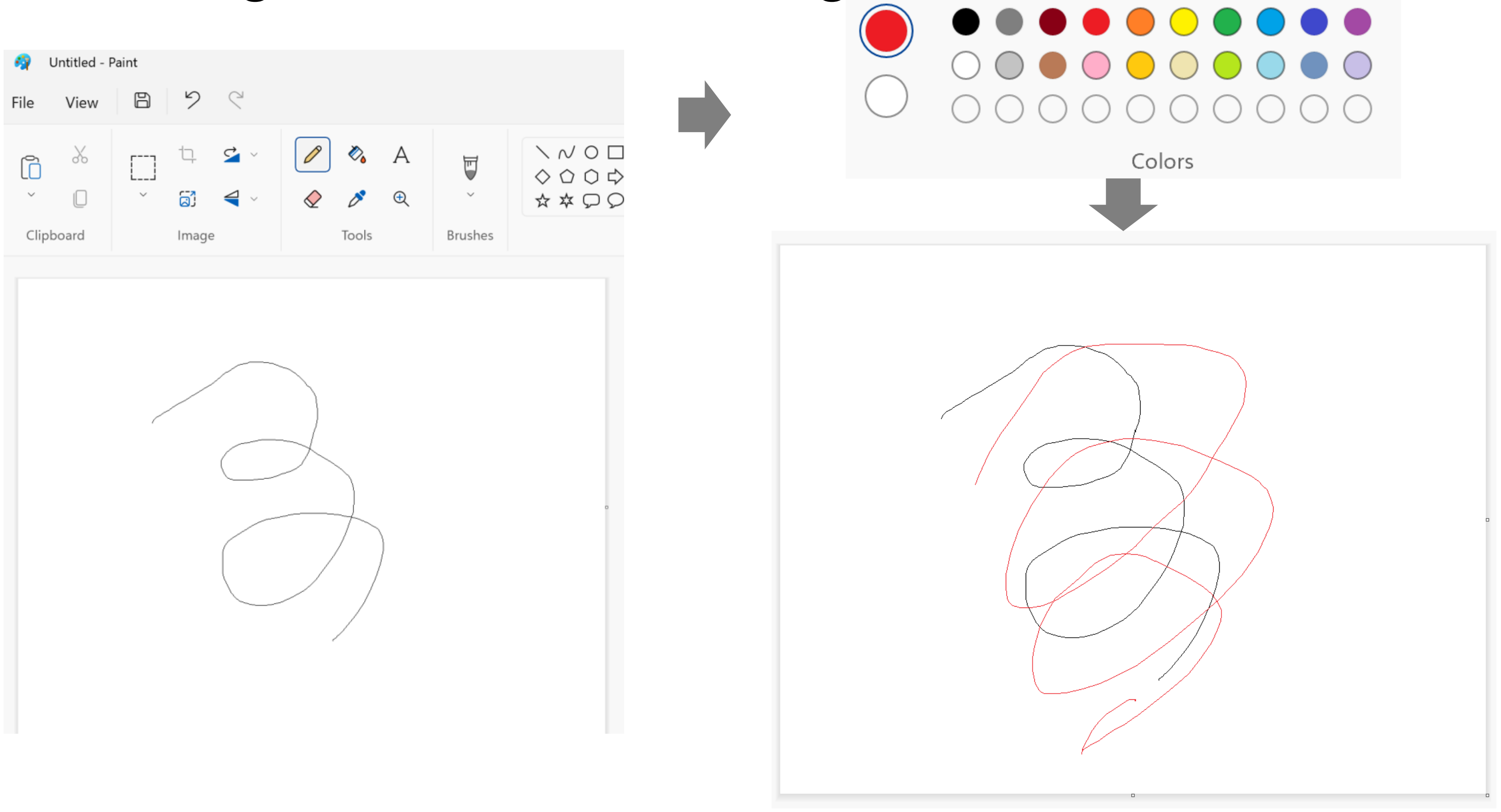
- Running from top to bottom

```
super.paint(g);
g.drawRect(100, 100, 1000, 1000);
g.setColor(ColorUtil.BLUE);
g.drawRect(150, 150, 700, 700);
```

- Changing properties affects the following draws

Similar to Painting Software

- Setting affect the following draws



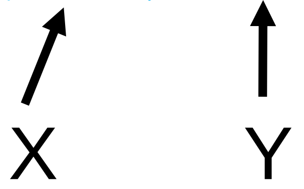
Java VS CN1

- Java AWT/Swing component has `getGraphics()` method
  - returns `Graphics` object of the component.

- CN1 **does not** have this method.
  - Get `Graphics` object by overriding `paint()`

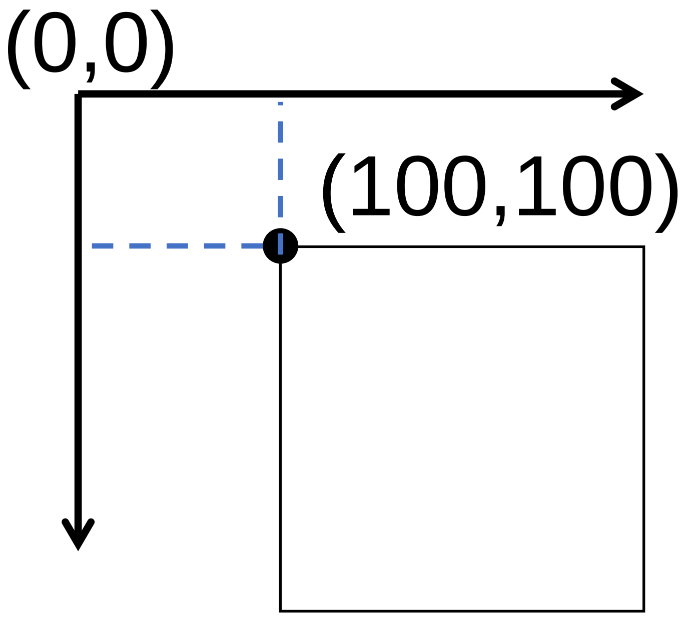
Coordinates

```
g.drawRect(100, 100, 1000, 1000);
```



- X,Y coordinate
- Origin
  - **top-left** corner

- But where?



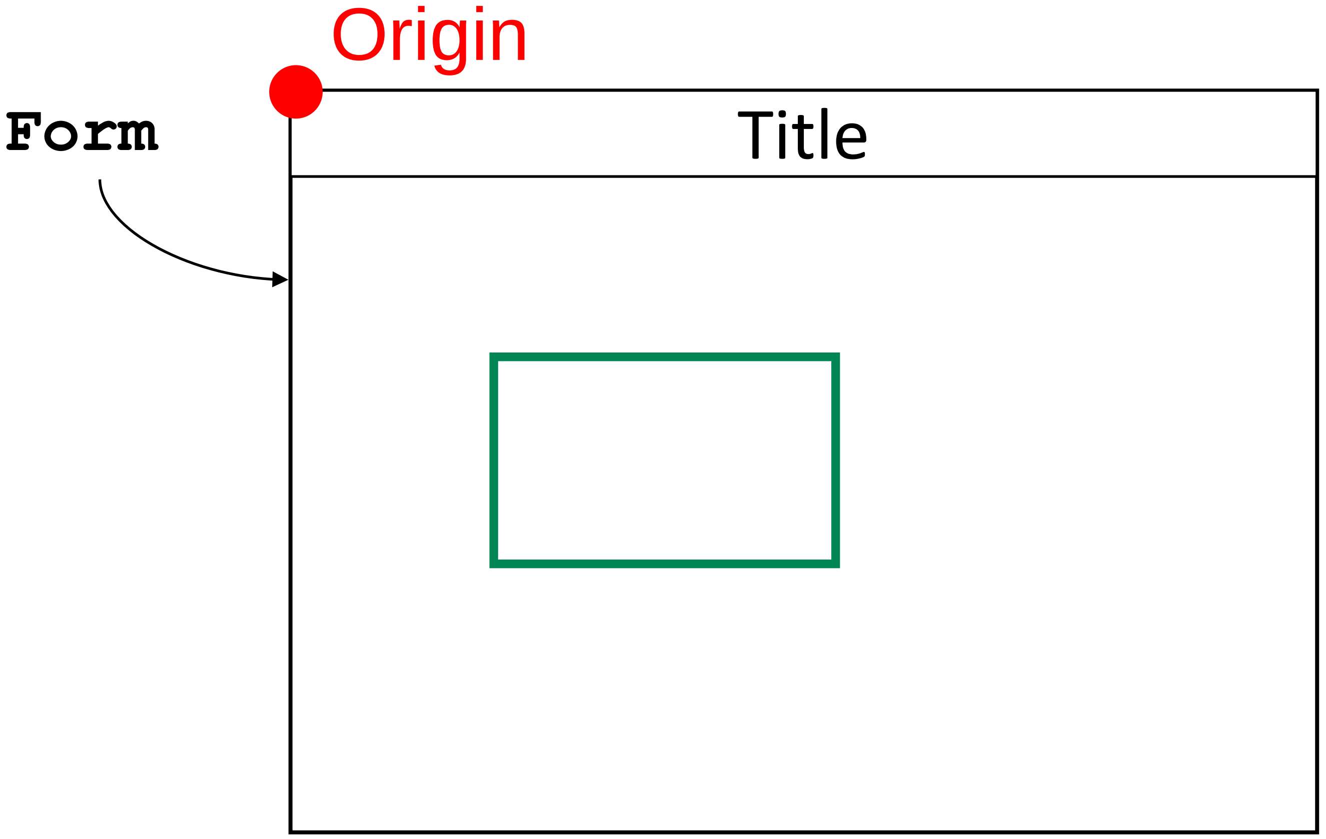
Drawing Coordinates

- Relative to the component's **parent's** "origin"
  - Not the component's origin
  - Not the screen

- Parent is the container that holds the component.
  - In form: screen
  - A component in a form: the content pane of form
  - A component in a container: the container

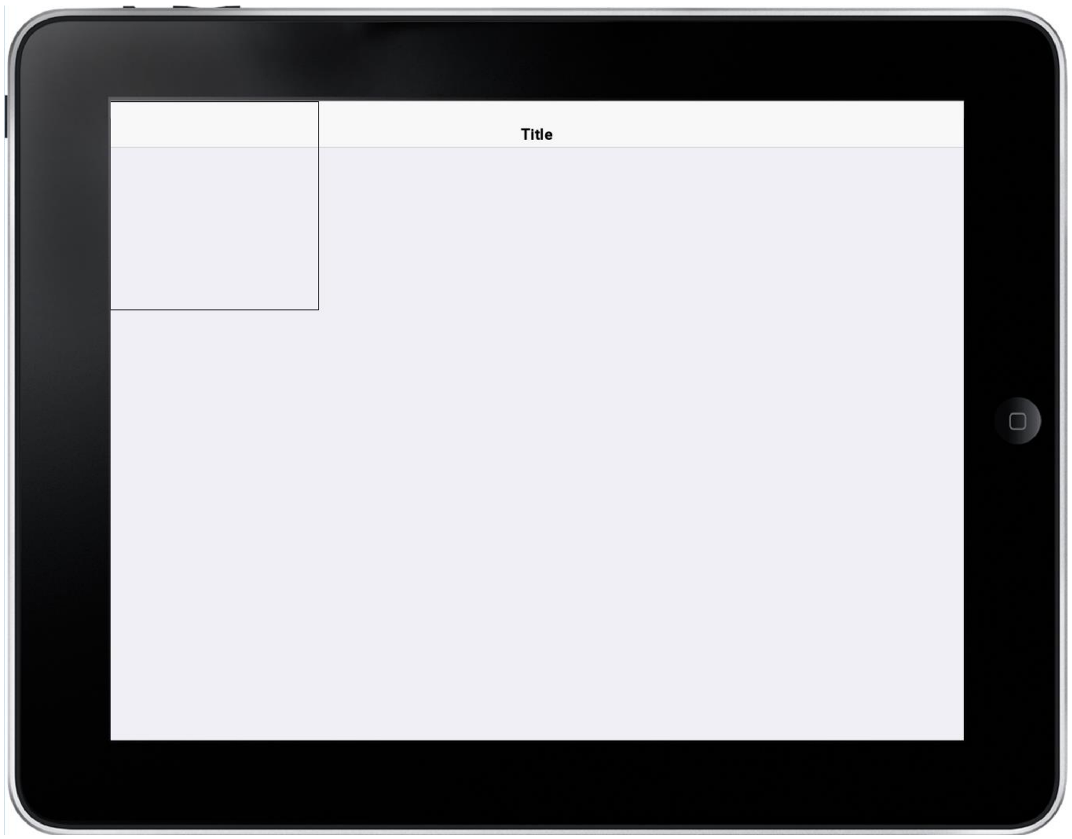
Parent

- If we draw on the **Form** directly



Draw on Form

```
g.drawRect(0,0, 500, 500);
```

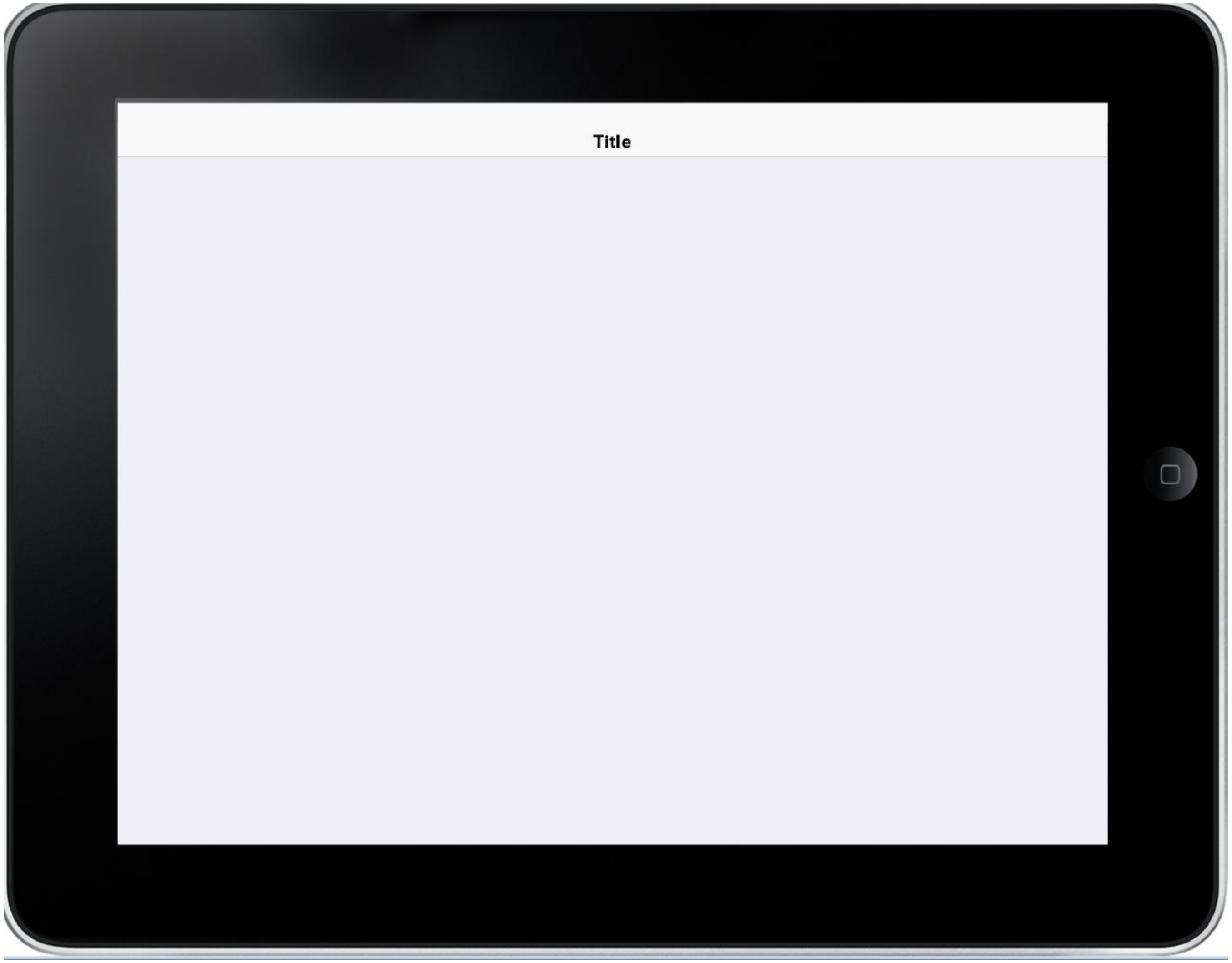


Draw on Container

```
public class MyForm extends Form {
  public MyForm() {
    super("Title");
    add(new MyContainer());
    show();
  }
}
================
public class MyContainer extends Container {
   public void paint(Graphics g){
    super.paint(g);
    g.setColor(ColorUtil.BLACK);
    g.drawRect(0,0, 500, 500);
   }
}
```

Result

- ???



Size of Container

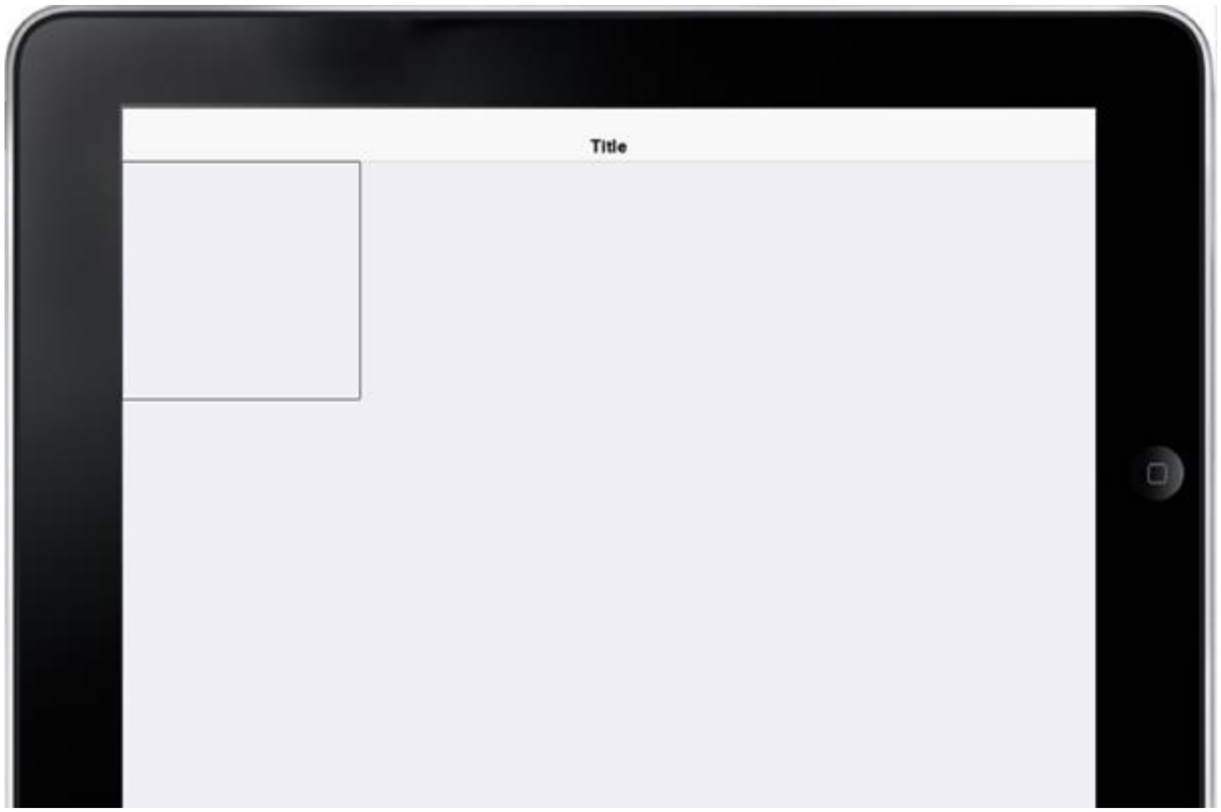
- The size of container will not be adjusted automatically to fit the content in `paint()`

- You need to set the size of the container

```
@Override
protected Dimension calcPreferredSize(){
    return new Dimension(1000, 1000);
}
```

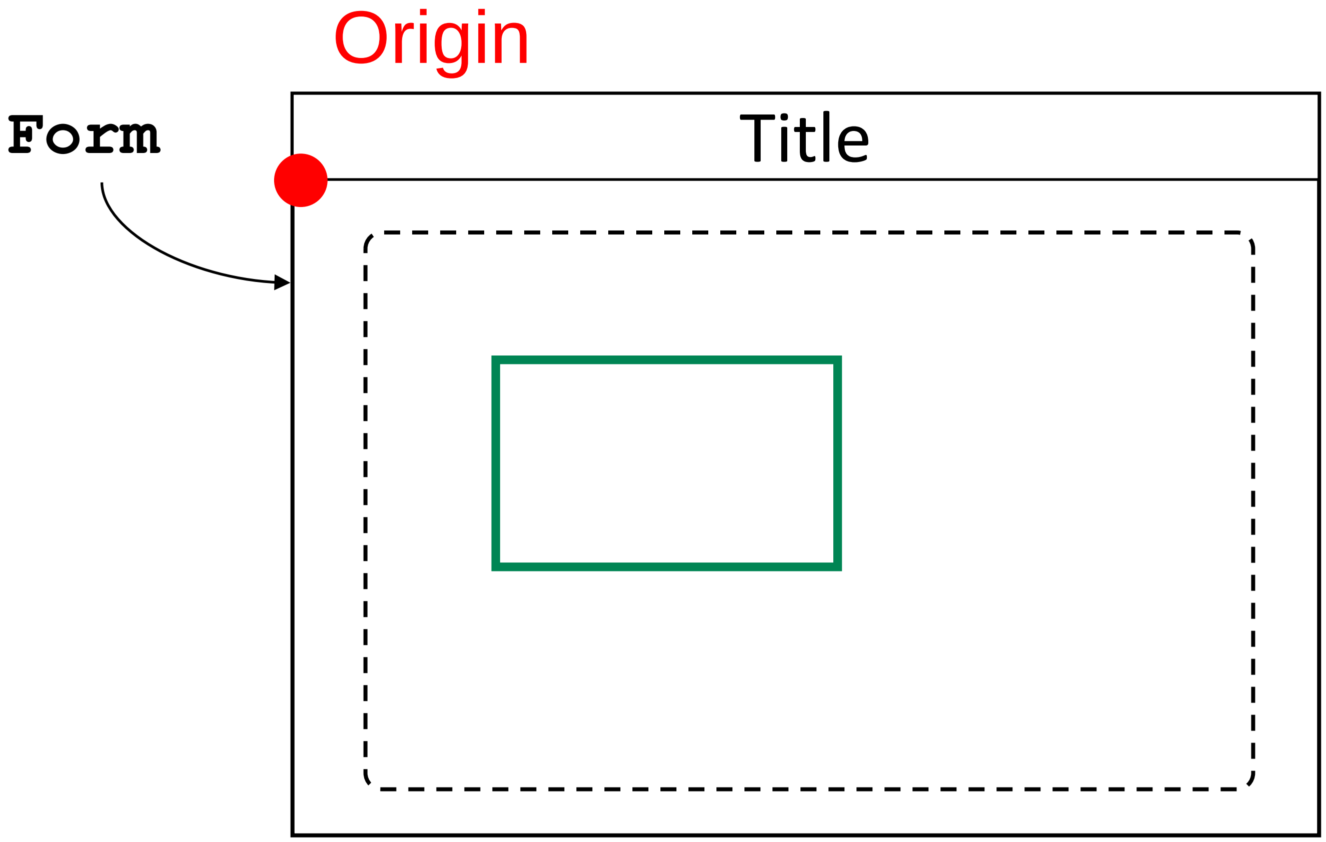
Draw on Container

- `g.drawRect(0,0, 500, 500);`



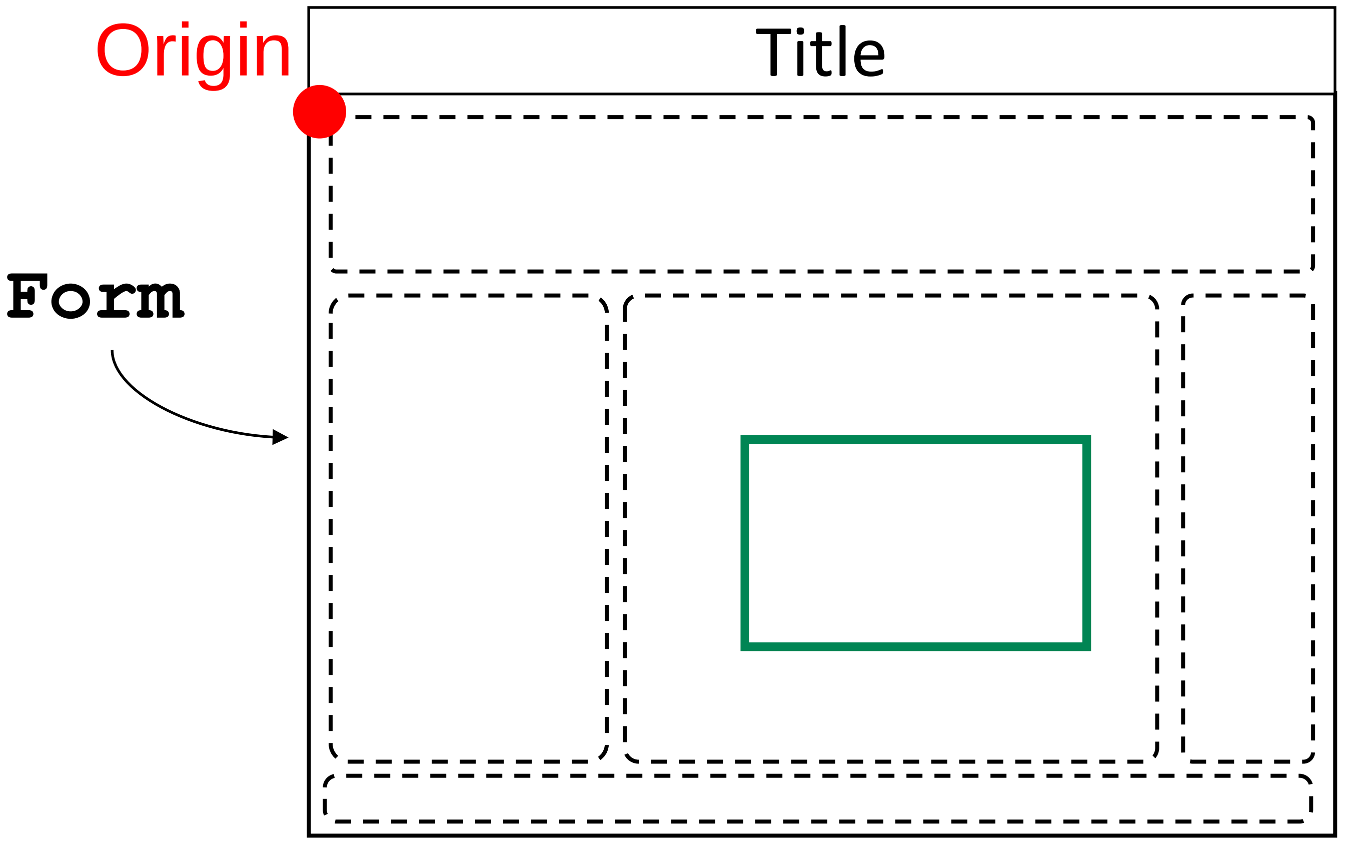
Parent

- If we draw in a container on `Form`



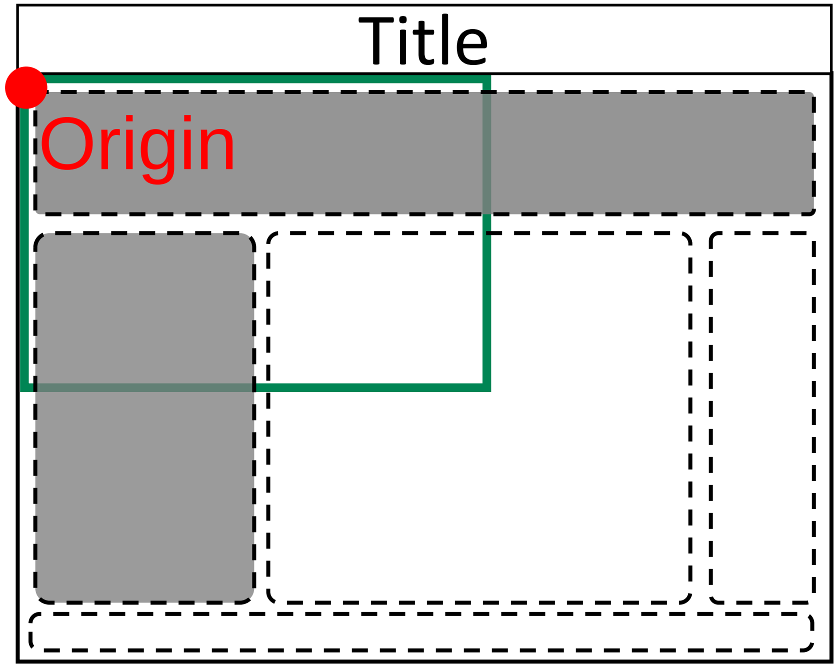
Problem

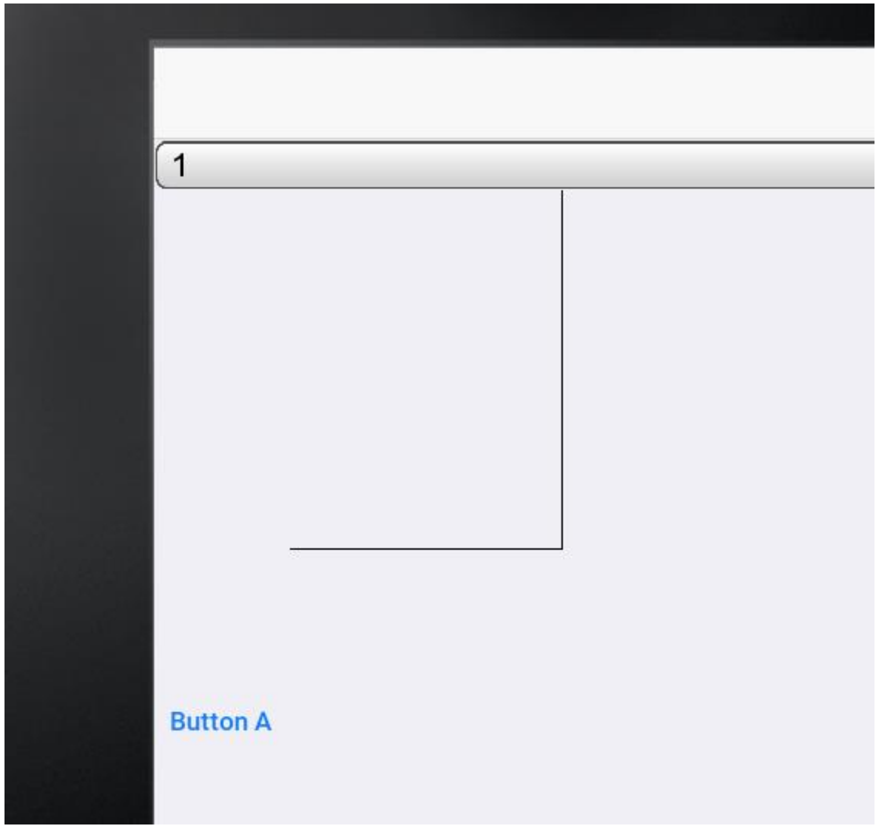
- If we draw in a center container on `Form`



Problems

- `g.drawRect(0,0, 500, 500);`
  - The rectangle is clipped



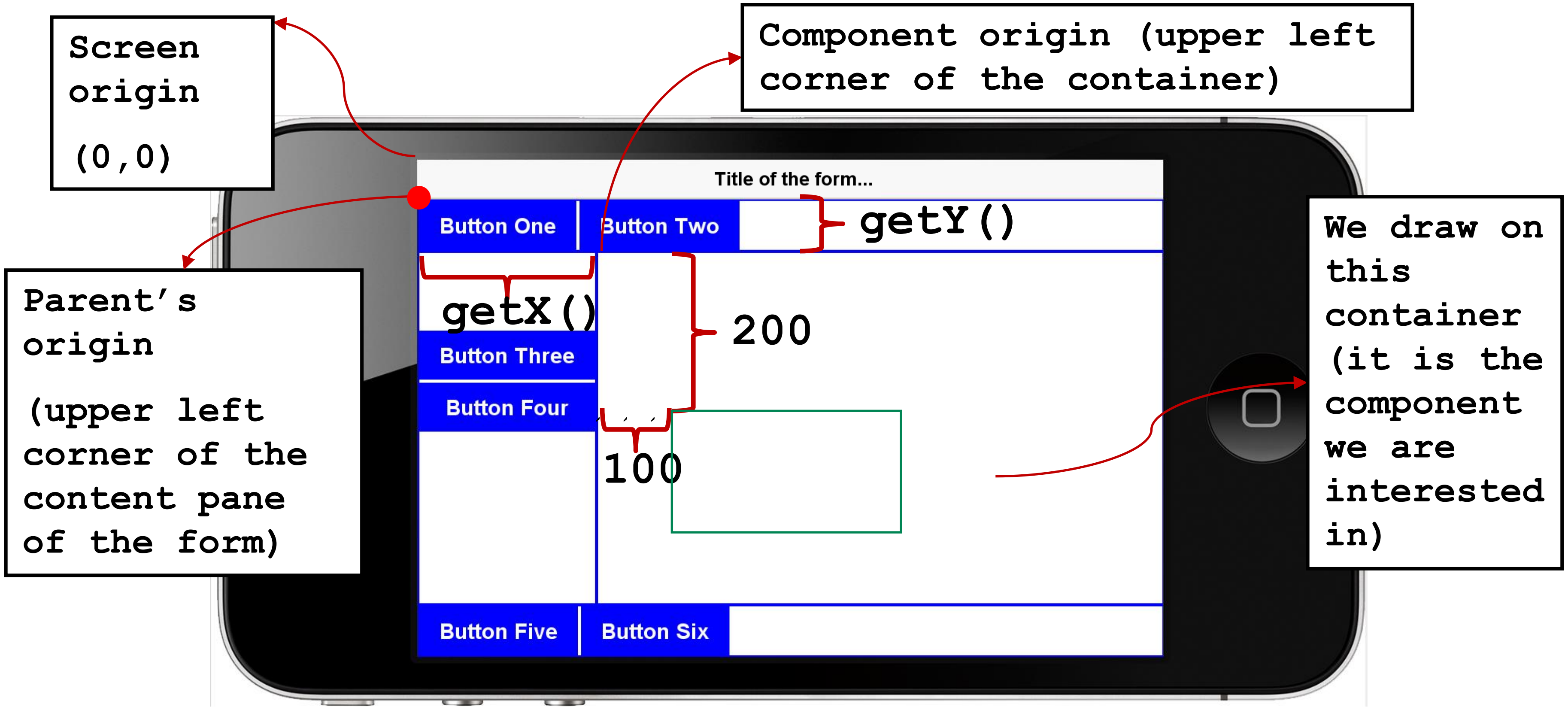


Component’s Origin

- `getX()/getY()` methods of `Component`
- Return the component’s origin location
- **Relative** to its parent’s origin location.

Drawing from Origin

```
drawRect(getX()+100,getY()+200,w,h)
```


Screen origin
(0,0)
Component origin (upper left corner of the container)
Title of the form...
Button One
Button Two
getY()
getX()
200
Button Three
Button Four
100
Button Five
Button Six
Parent's origin
(upper left corner of the content pane of the form)
We draw on this container (it is the component we are interested in)

Importance of getX()/getY()

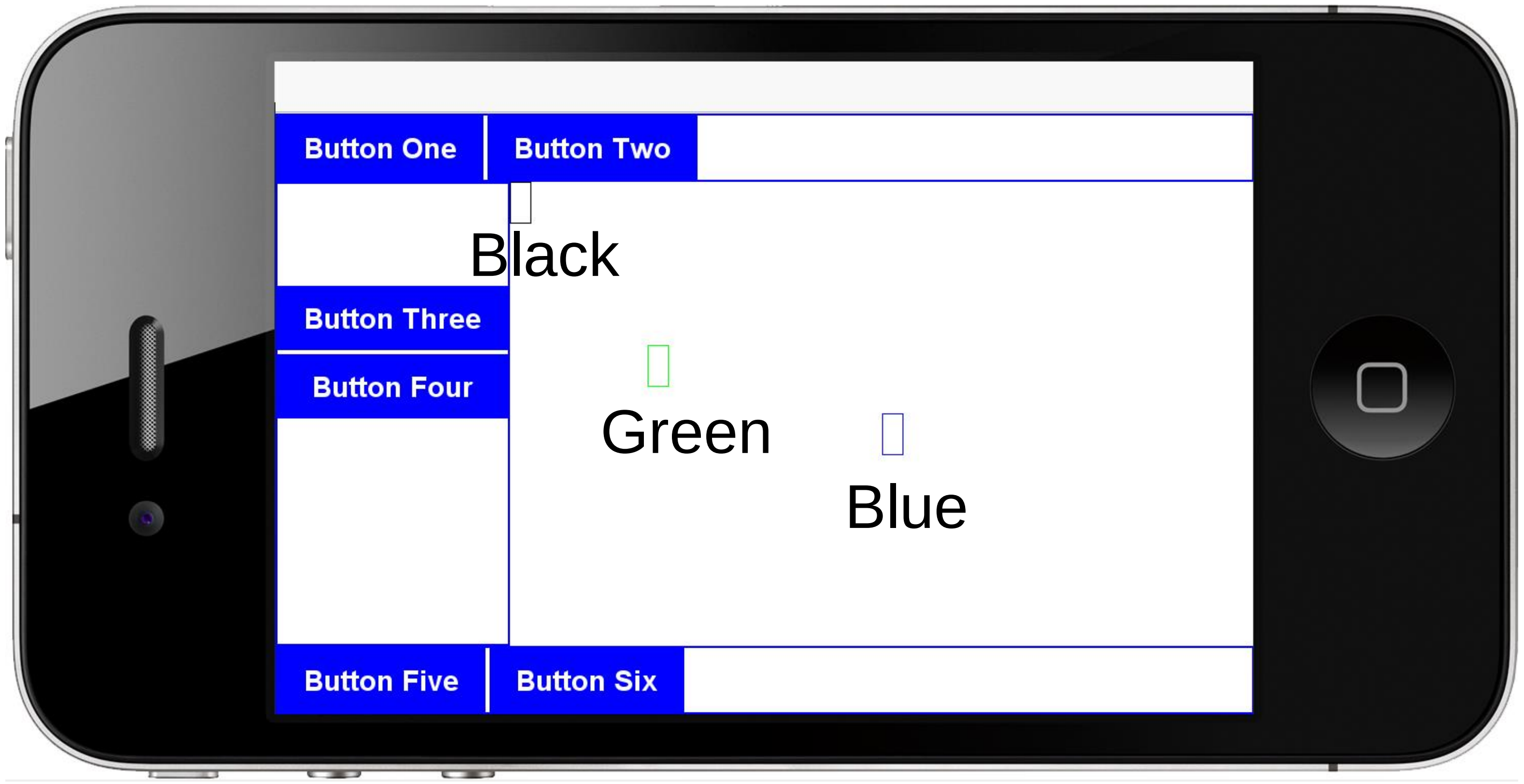
Draw a rectangle in the middle of **`Container`**.

If we have the following **`paint()`** method:

```
super.paint(g);
int w = getWidth();     int h = getHeight();
g.setColor(ColorUtil.BLACK);
g.drawRect(getX(), getY(), 20, 40);
g.setColor(ColorUtil.GREEN);
g.drawRect(w/2, h/2, 20, 40);
g.setColor(ColorUtil.BLUE);
g.drawRect(getX()+w/2, getY()+h/2, 20, 40);
```

Result

Only the blue rectangle would appear in the center of the `CustomContainer`…



Non-working Example

- Never save the `Graphics` object and use it in another method to draw things!
- Otherwise:
  - Drawn things would vanish the next time `repaint()` is called …
  - Drawn things would be located in wrong positions…

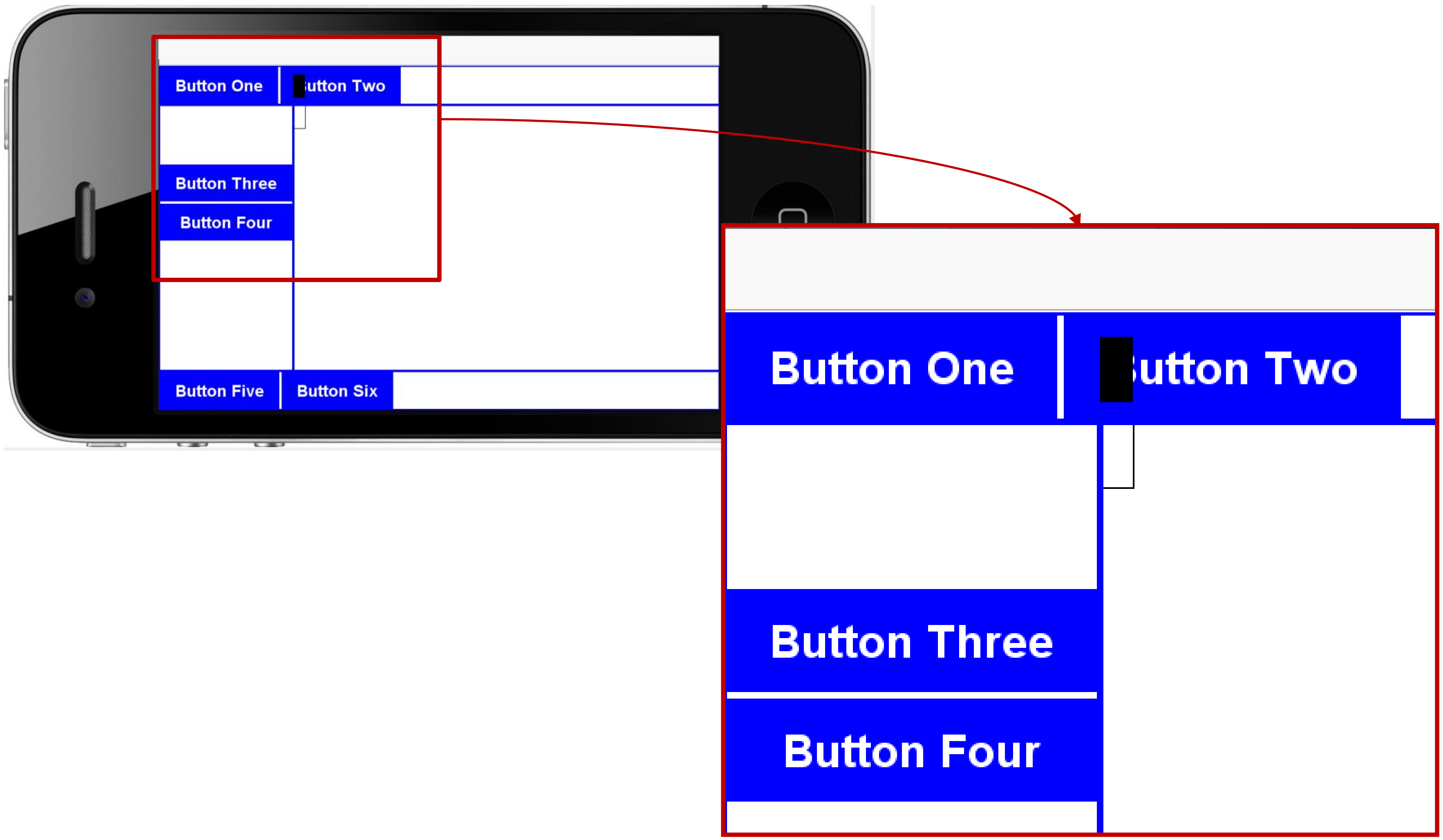
Non-working example

```
public class NonWorkingGraphics extends Form implements
ActionListener{
CustomContainer myCustomContainer = new CustomContainer();
 public NonWorkingGraphics() {
  //... [use border layout and add north, east, south containers (each
  //include two styled buttons)]
  buttonOne.addActionListener(this);
  this.add(BorderLayout.CENTER, myCustomContainer);
 }
 public void actionPerformed(ActionEvent evt) {
  myCustomContainer.drawObj();
 }
}
```

Non-working example (cont.)

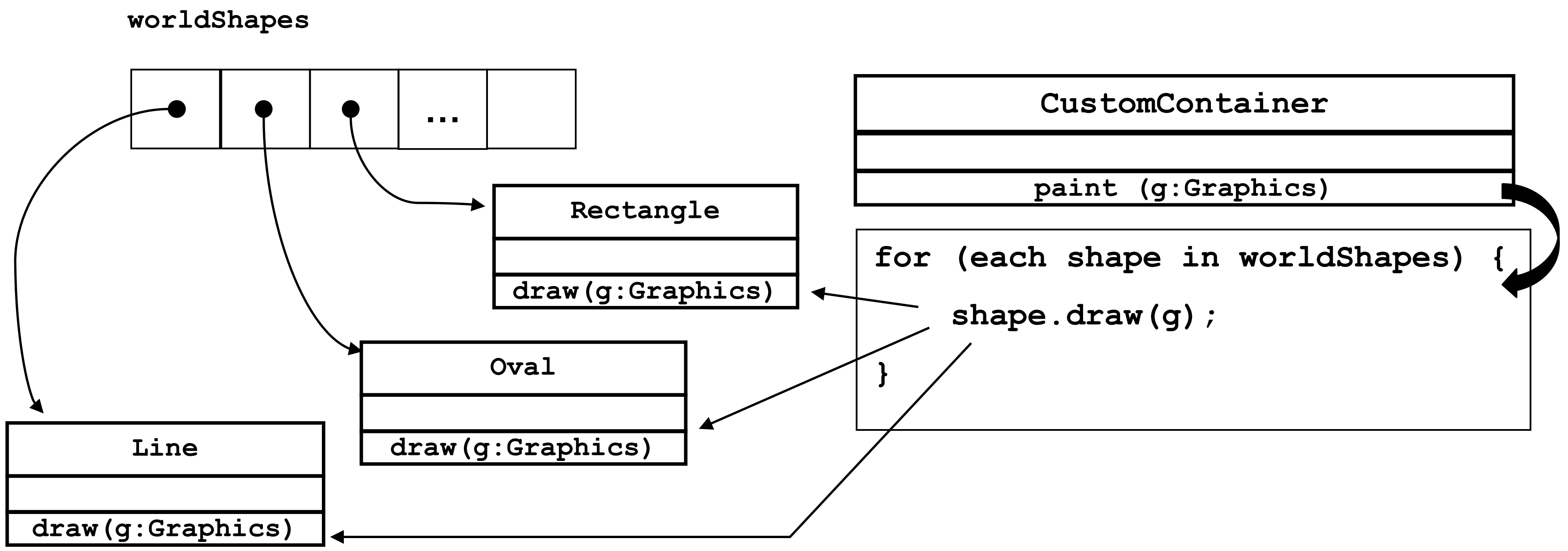
```
public class CustomContainer extends Container{
 private Graphics myGraphics;
 public void paint(Graphics g){
  super.paint(g);
  myGraphics = g;
  myGraphics.setColor(ColorUtil.BLACK);
  //empty rectangle appears in the CORRECT place (at the origin of this)
  myGraphics.drawRect(getX(), getY(), 20, 40);
 }
 public void drawObj(){
  repaint();
  myGraphics.setColor(ColorUtil.BLACK);
  //filled rectangle appears in the WRONG place and disappears next time
  //repaint() is called
  myGraphics.fillRect(getX(), getY(), 20, 40);
 }
}
```

Non-working example (cont.)



Maintaining Graphical State

- Must assume `repaint()` will be invoked
  - Must *keep track of objects you want displayed*
  - Redisplay them in paint()

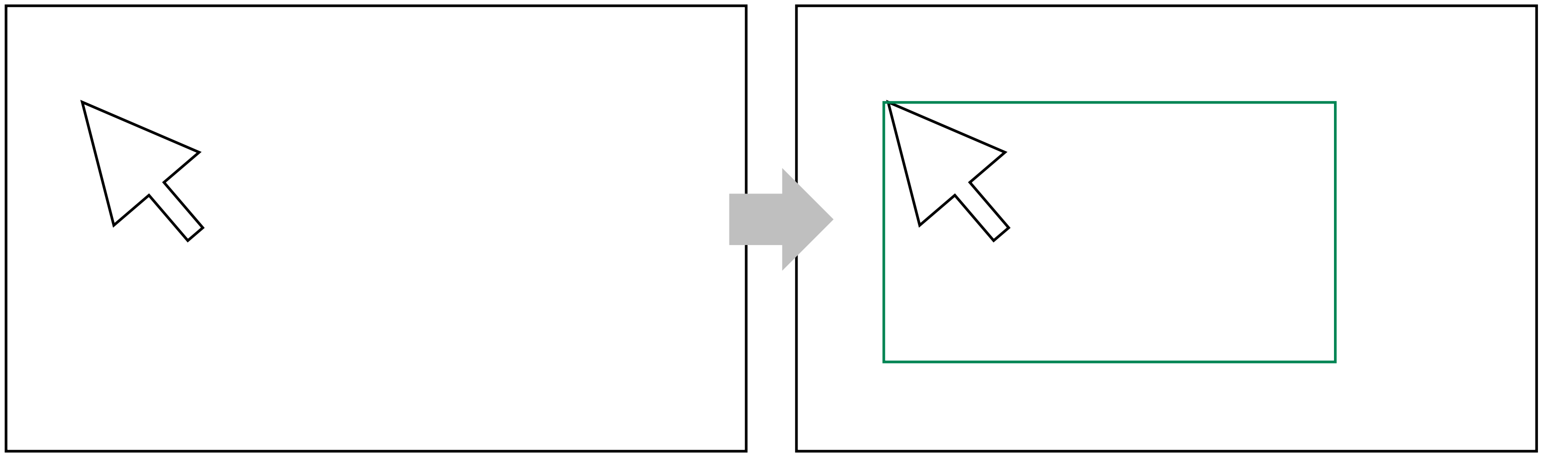


```
for (each shape in worldShapes) {
    shape.draw(g);
}
```

Simple Drawer App

Rectangle Drawer

- An app that allow users to stamp a rectangle



- We need to handle Pointer Event

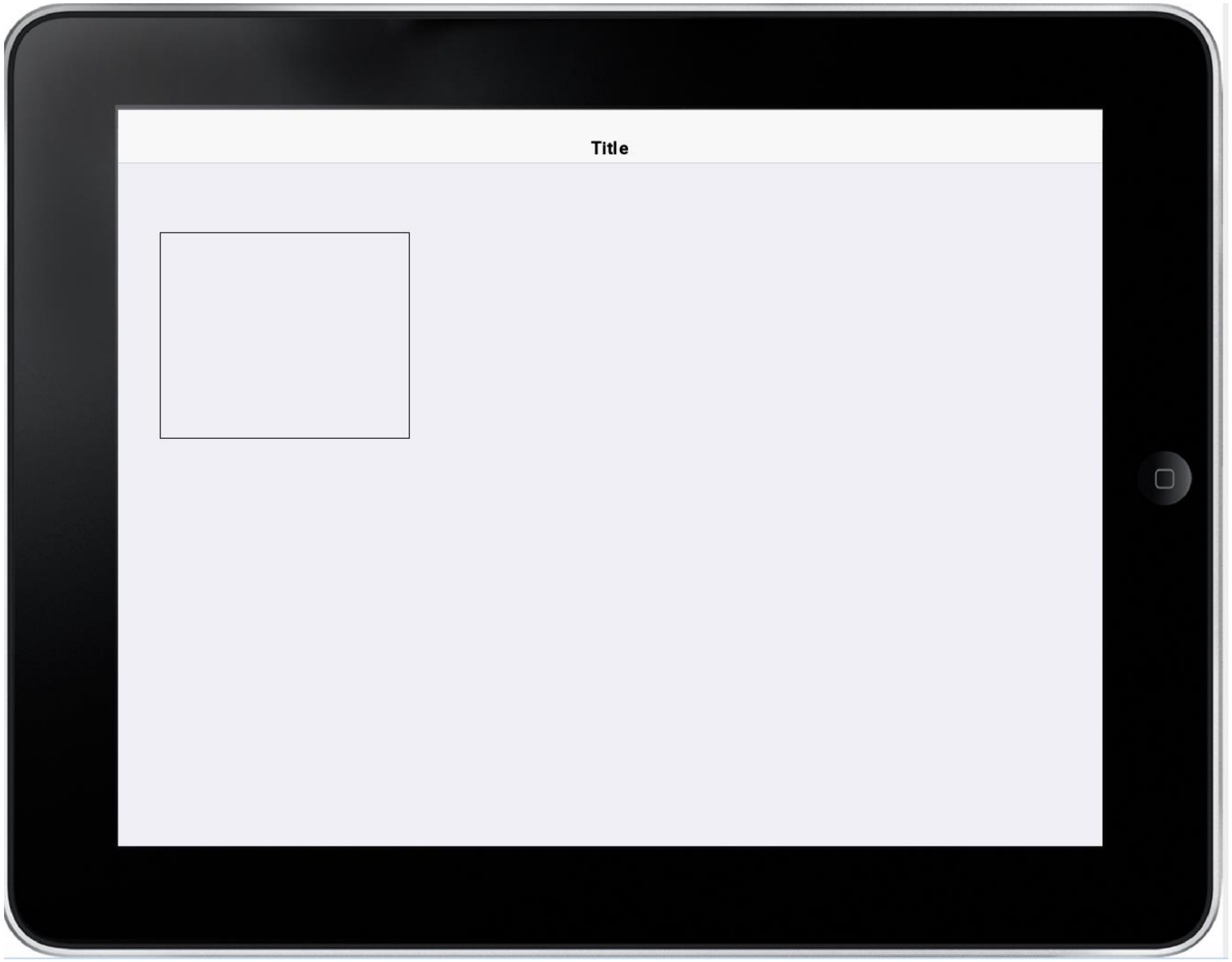
Pointer Methods

```
/* Center container of the form is a PointerContainer which
extends from Container */
public class PointerListenerForm extends Form{
  public PointerListenerForm() {
    PointerContainer myPointerContainer = new
      PointerContainer();
    this.add(BorderLayout.CENTER,myPointerContainer);
    ... }
}
------
/* We can override the pointer methods in the Container */
public class PointerContainer extends Container{
  public void pointerPressed(int x,int y){...}
  public void pointerReleased(int x,int y){...}
  public void pointerDragged(int x,int y){...}
}
```

Code

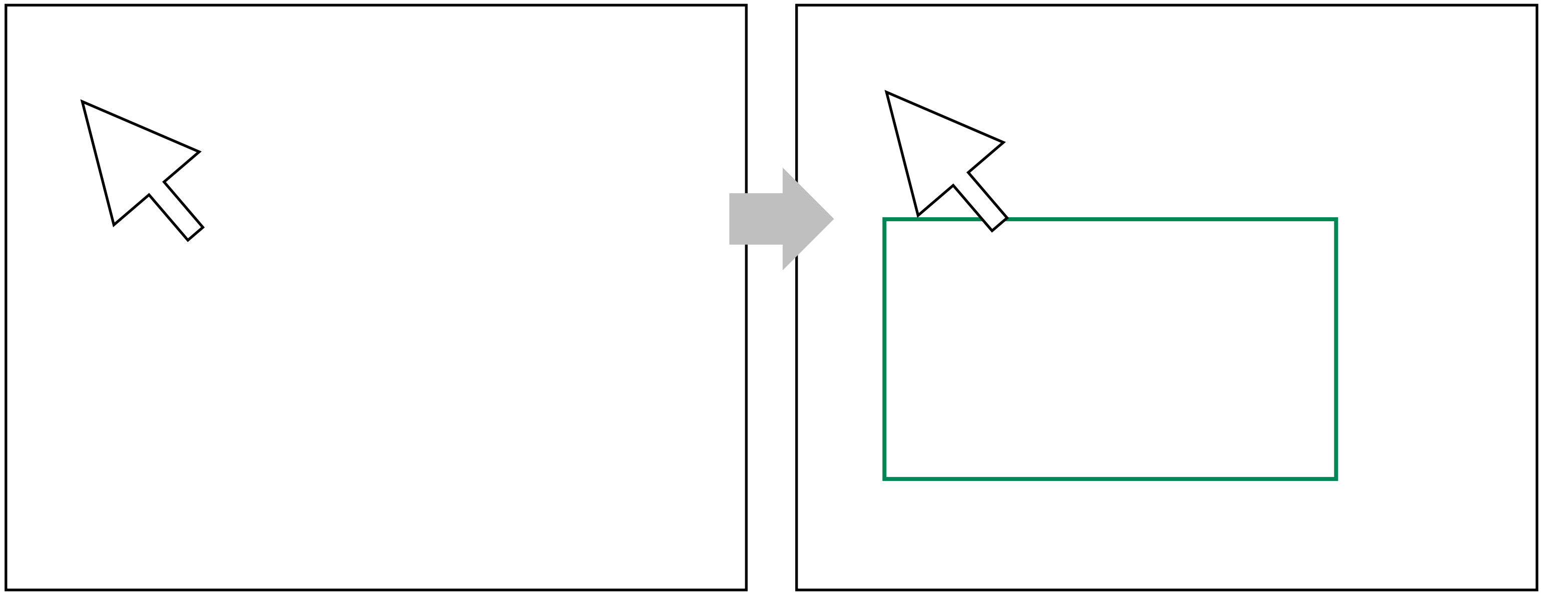
```
public class MyContainer extends Container {
  int startX = 0, startY = 0;
  public void paint(Graphics g){
    super.paint(g);
    g.setColor(ColorUtil.BLACK);
    g.drawRect(startX, startY, 300,200);
  }
  public void pointerPressed(int x,int y){
        startX = x;    startY = y;    repaint();
  }
  public Dimension calcPreferredSize() {
     return new Dimension(1000, 1000);
  }
}
```

Results



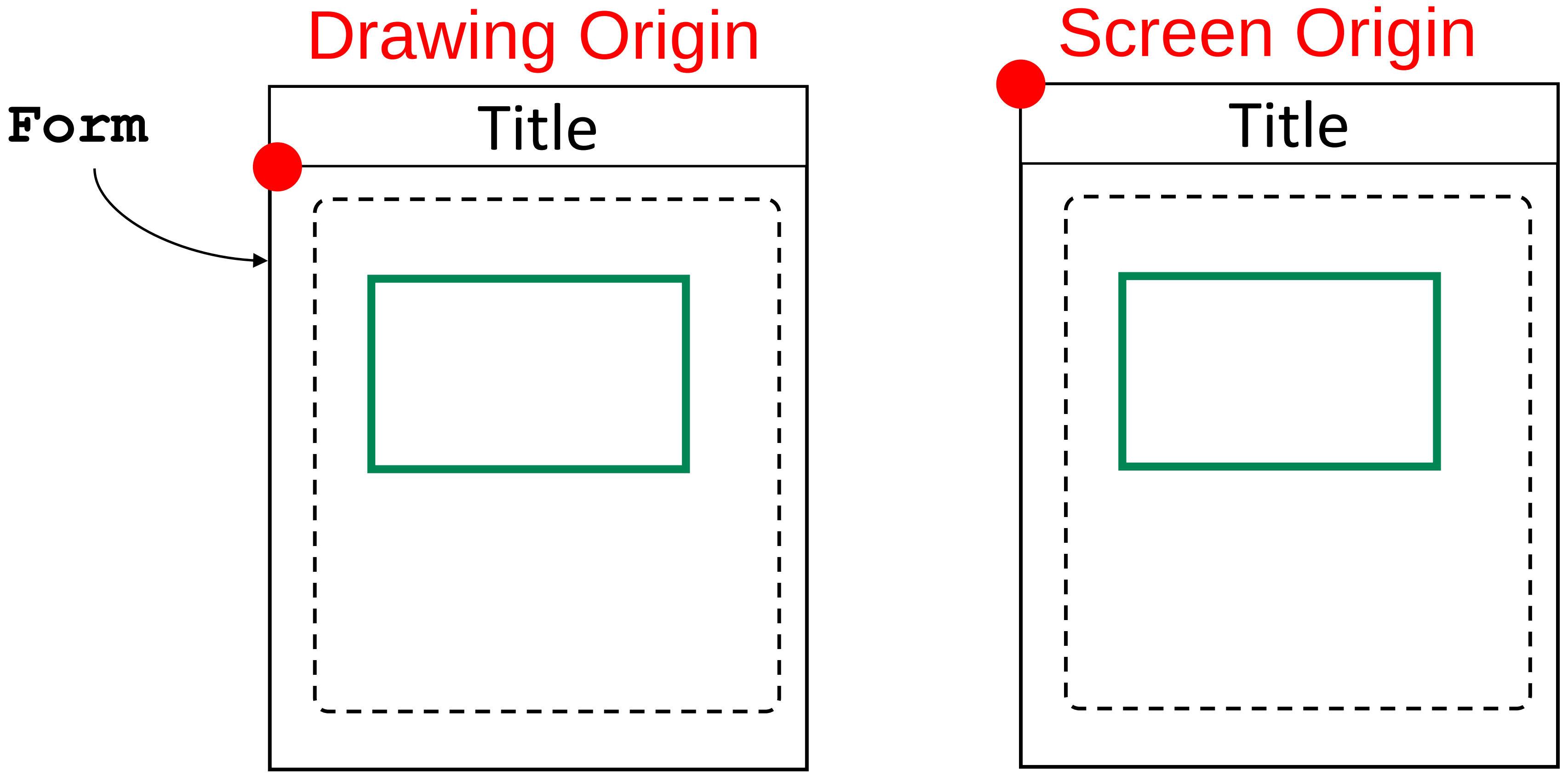
Wait…

- Why is the rectangle offset?



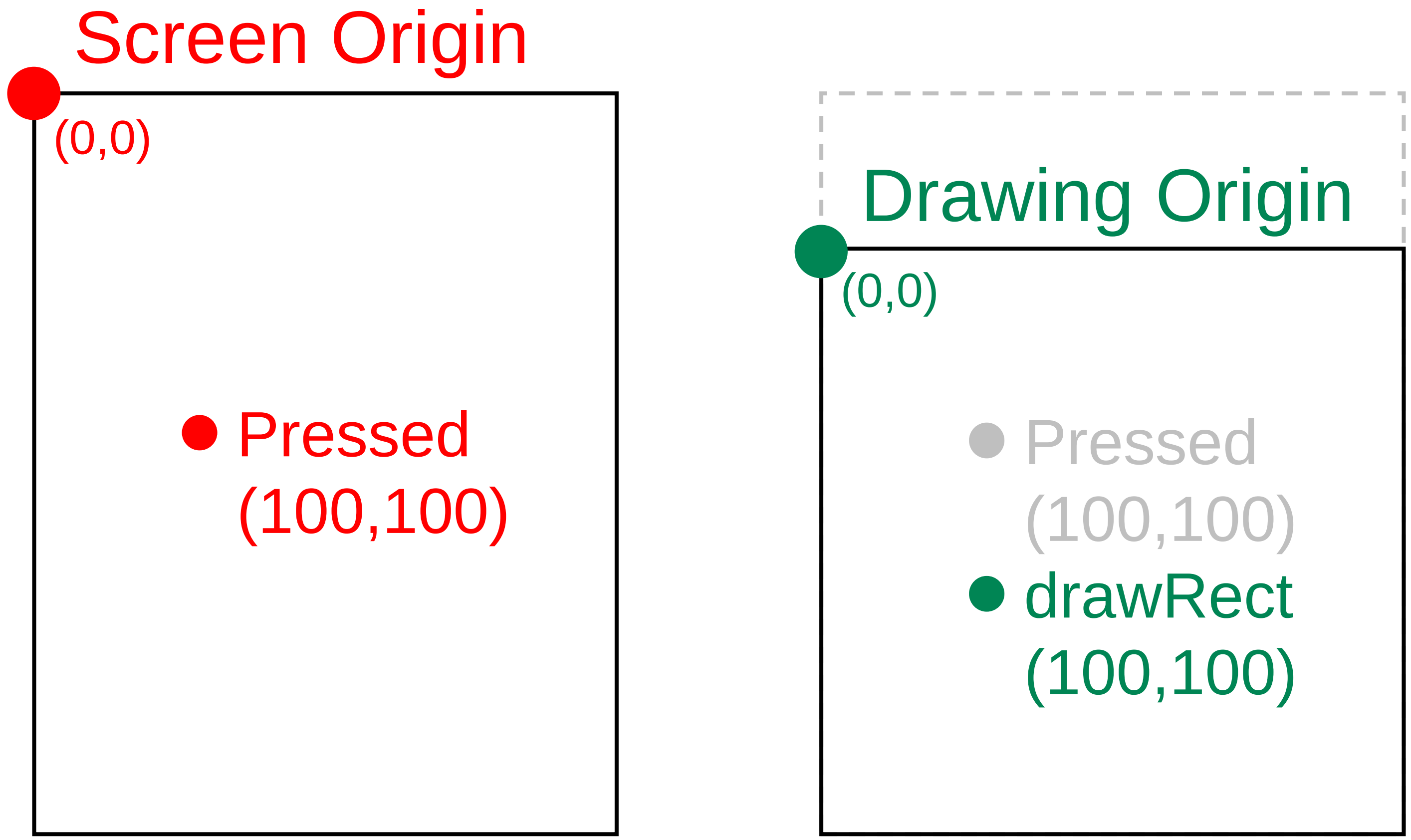
Remember

- Drawing and pointer origin are different



Drawing Problem

- If we draw with different origin



Coordinate Convertion

- Convert screen coordinate to parent coordinate

- `getAbsoluteX()` and `getAbsoluteY()` methods of the parent container.
  - Get screen location of the origin

- You can get the parent using `getParent()` method of the component.

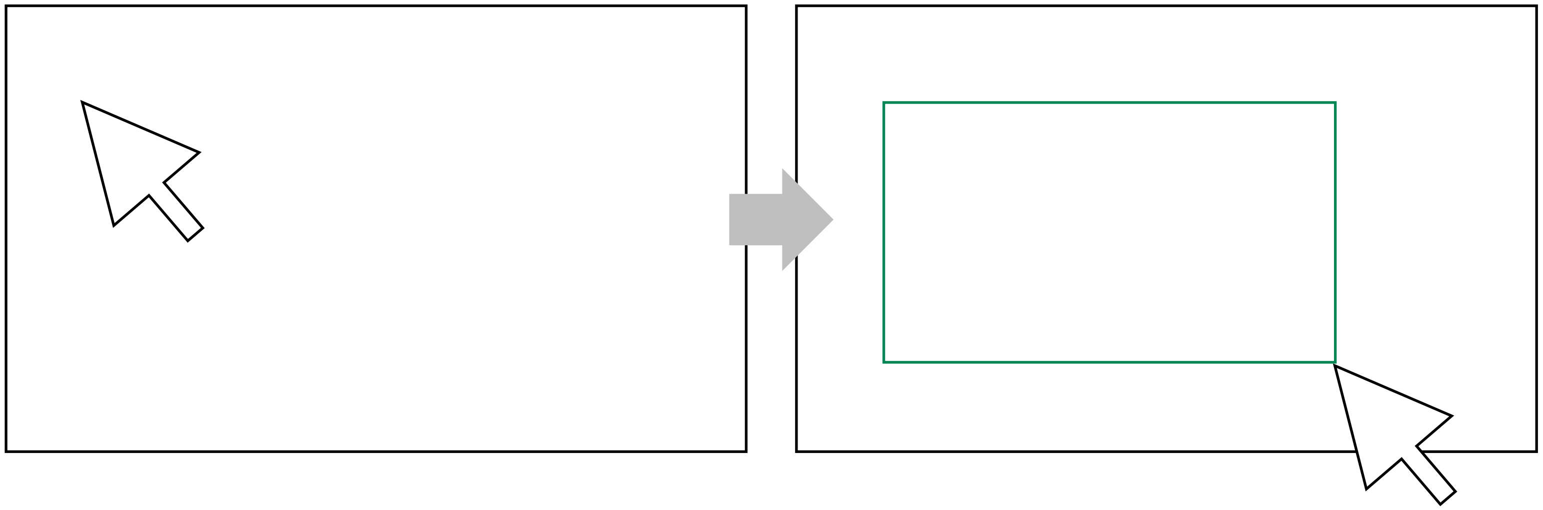
Updated Code

```
public class MyContainer extends Container {
  int startX = 0, startY = 0, endX = 0, endY = 0;
  public void paint(Graphics g){
    super.paint(g);
    g.setColor(ColorUtil.BLACK);
    g.drawRect(startX - getParent().getAbsoluteX(),
               startY - getParent().getAbsoluteY(), 300,200);
  }
  public void pointerPressed(int x,int y){
        startX = x;   startY = y;   repaint();
  }
  public Dimension calcPreferredSize() {
     return new Dimension(1000, 1000);
  }
}
```

But stamping is boring

Rectangle Drawer

- An app that allow users to draw a rectangle



- Record the pressed and released position

Updated Code

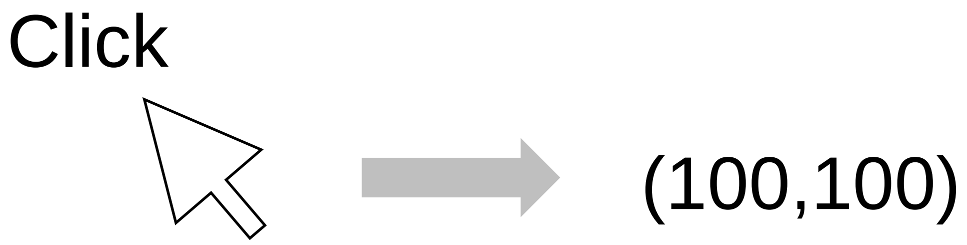
```
public class MyContainer extends Container {
  int startX = 0, startY = 0, endX = 0, endY = 0;
  public void paint(Graphics g){
    super.paint(g);
    g.setColor(ColorUtil.BLACK);
    g.drawRect(startX - getParent().getAbsoluteX(),
               startY - getParent().getAbsoluteX(),
   endX - startX, endY - startY);
  }
  public void pointerPressed(int x,int y){ startX = x; startY = y;}
  public void pointerReleased(int x,int y){
        endX = x; endY = y;
        repaint();
  }
  public Dimension calcPreferredSize() {
     return new Dimension(1000, 1000);
  }
}
```

Positioning Technique

- Pointing
- Rubber Band Technique
- Constraints
- Dragging
- Grid
- Gravity Field

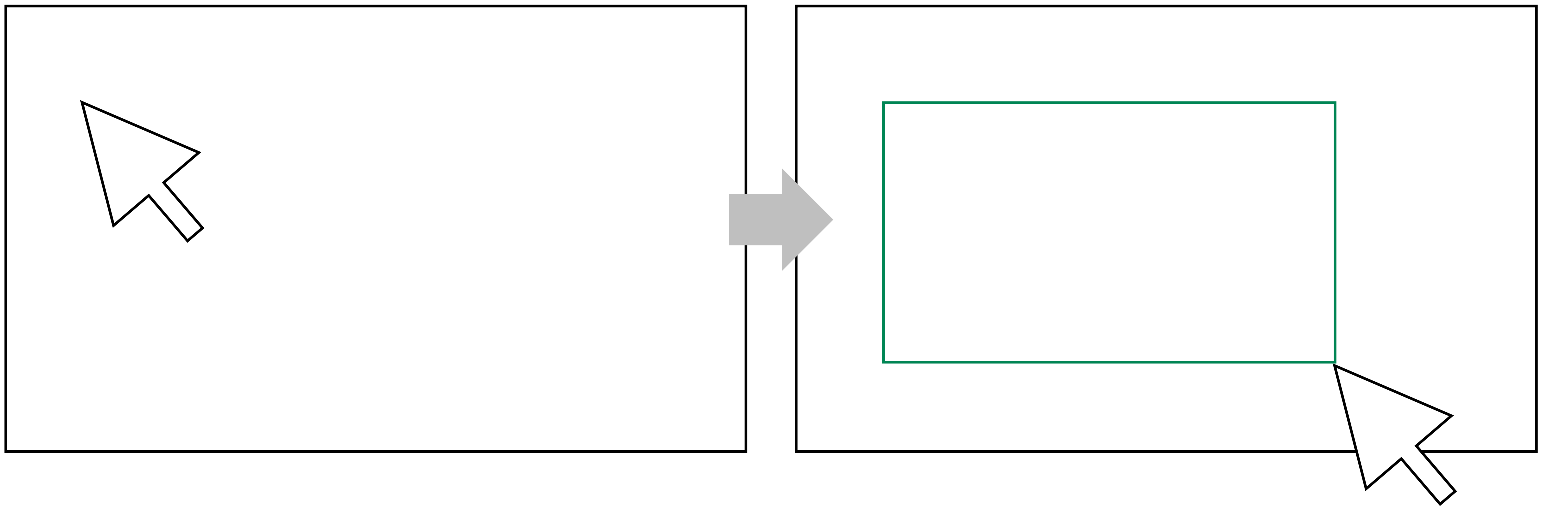
Pointing

- Click to locate the absolute position
  - E.g., Stamping



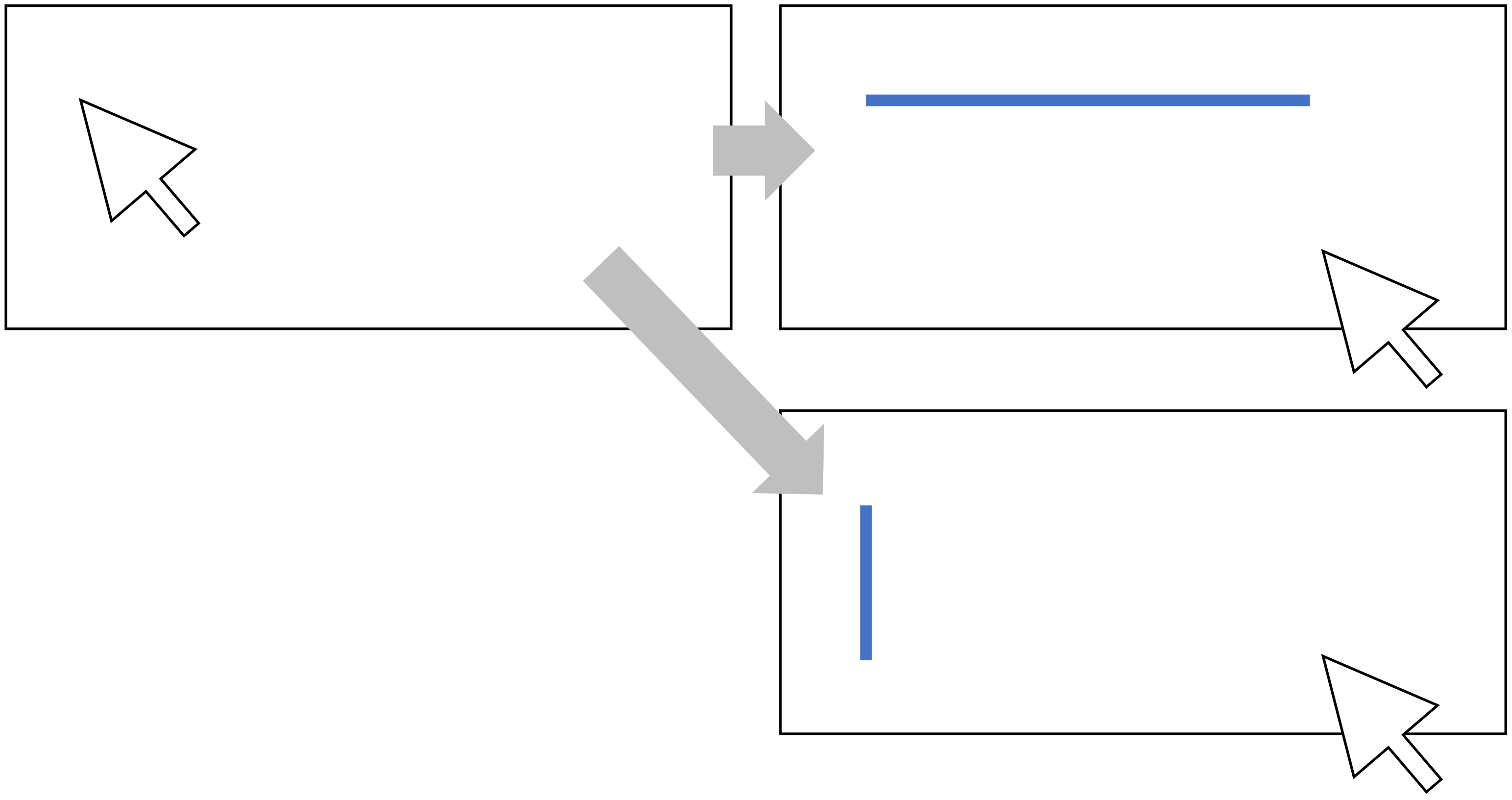
Rubber Band Technique

- Define a shape by dragging from a point



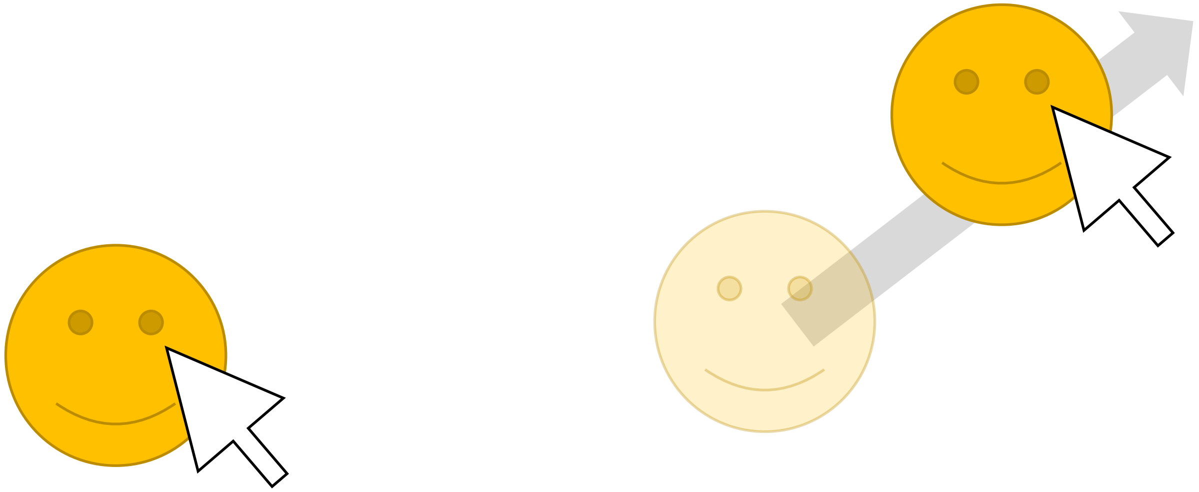
Constraints

- Restrict the direction of the shape



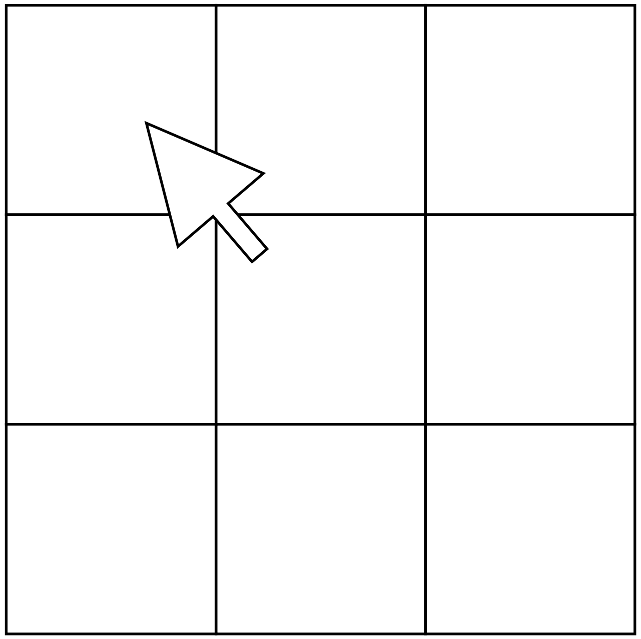
Dragging

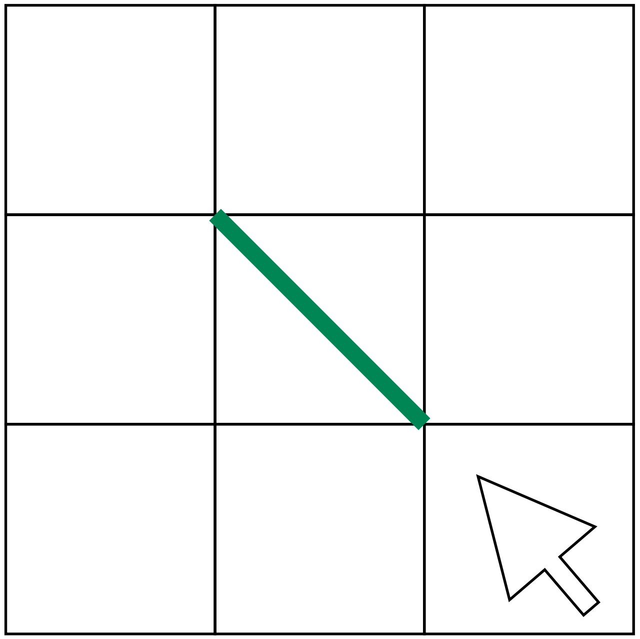
- A method to changing position



Grids

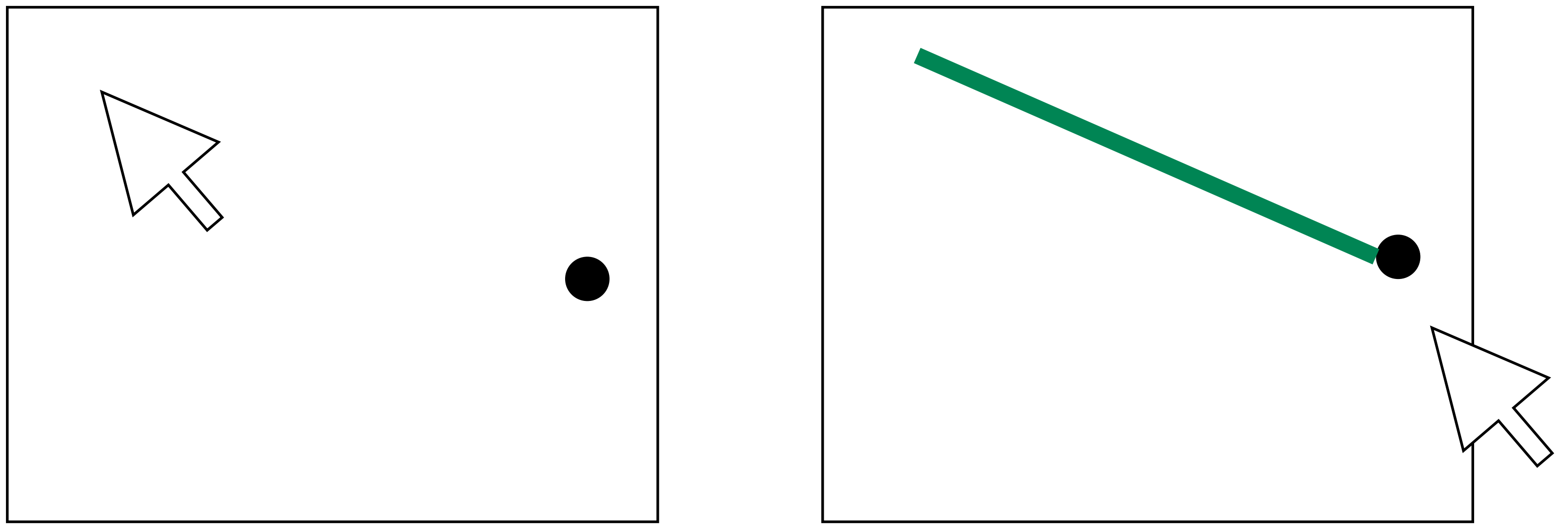
- Position is rounded to the nearest intersection of two grid lines





Gravity Field

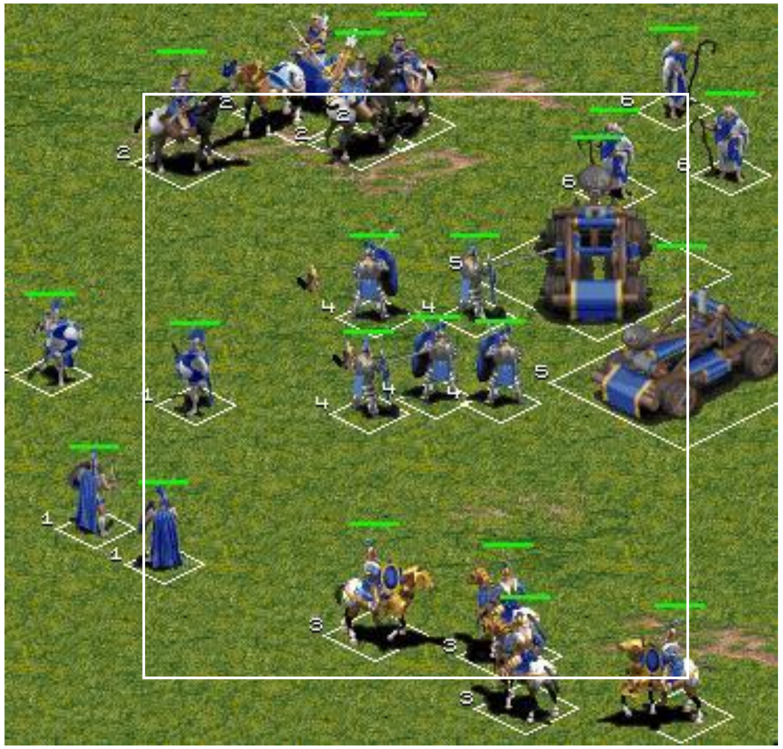
- Or **Snapping**
  - Position is moved to the nearest objects if it is close enough



Why do we need to draw rectangle?

Very Useful

- Selecting object
- E.g., game objects
- Next lecture!


Age of Empires, ©Microsoft

Any Questions