CSc 133 Lecture Notes

10 - Interactive Techniques

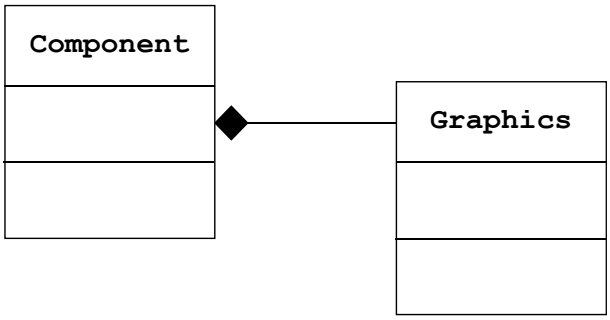
Computer Science Department
California State University, Sacramento

Overview

- `Graphics` Class (and object)
- Component Repainting, `paint()`
- Graphics State Saving
- Onscreen Object Selection

Component Graphics

- **Every `Component` contains an object of type `Graphics`**



- **`Graphics` objects know how to draw on the component**

`Graphics` Class

- **`Graphics`** objects 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);`**

    e.g., to draw a filled circle with radius r:

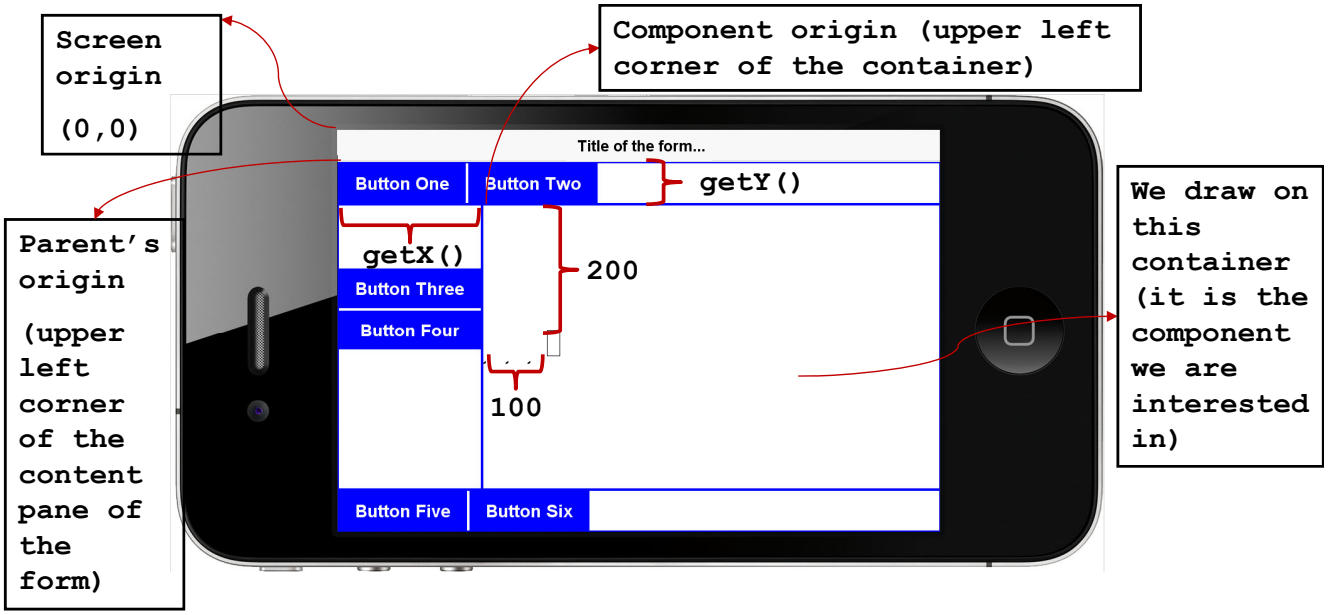
    `fillArc(x, y, 2*r, 2*r, 0, 360);`
  - **`drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`**

    e.g., you can draw a triangle using the drawPolygon()...
  - **`drawString (String str, int x, int y);`**
  - **`setColor (int RGB);`**
  - **`. . .`**

Drawing Coordinates

- Drawing coordinates (e.g., x/y in **`drawRect()`**) indicate the location of upper left corner of the shape that is being drawn.
- Drawing coordinates are **relative to the component's parent's "origin" (not the component's origin … it's parent's origin)**
- Parent is the container that holds the component. If we add a component (e.g. container) to a form, content pane of the form would be the parent of the component.
- Origin of the parent/component is at its upper left corner.
- **`getX()/getY()`** methods of **`Component`** return the component's origin location relative to its parent's origin location.

Drawing Coordinates (cont.)



So to draw a rectangle at 100 pixels right and 200 pixels down of the origin of the component:

```
drawRect(getX()+100, getY()+200, width, height)
```

Getting a reference to the `Graphics` object

- But how can we get a hold of **`Graphics`** object of a component to call the draw methods on it??
- "Component repainting" mechanism allows us to get a hold of this reference…

Component Repainting

- **Every `Component` has a `repaint()` method**
  - Tells a component to update its screen appearance
  - Called automatically whenever the component needs redrawing
    - e.g., app is opened for the first time, user switched back to the app while multi-tasking among different apps, a method such as **`setBgColor(int RGB)`** is called…
  - Can also be called manually by the application code to force a redraw

Component Repainting (cont.)

- **`Component`** also contain a method named **`paint()`**
  - **`repaint()` passes the `Graphics` object to the component's `paint()` method**
  - **`paint()`** is responsible for the actual drawing (using **`Graphics`**)
  - Never invoke **`paint()`** directly; always call **`repaint()`** since repaint() does other important operations…

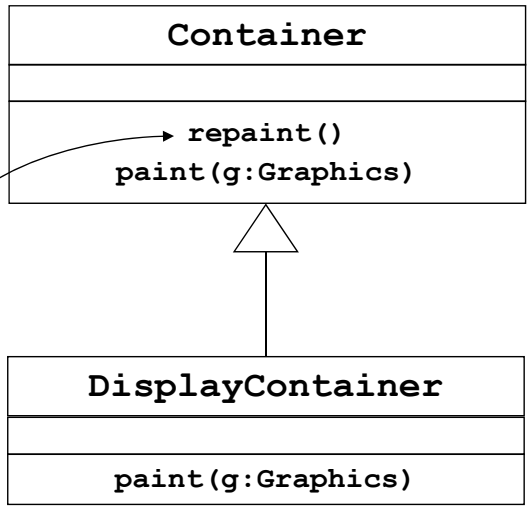
Differences between Java and CN1

- Java AWT/Swing component has **`getGraphics()`** method which returns **`Graphics`** object of the component.
- CN1 UI component does not have this method....
- Only way to get a hold of **`Graphics`** object is through overriding **`paint()`** method.

Overriding `paint()`

- Consider the following organization
  - Which **`paint()`** get invoked?

```
public class MyForm extends Form {
   private DisplayContainer myContainer;
   public MyForm() {
      ...
      myContainer = new DisplayContainer();
      ...
   }
   public void someMethod() {
      ...
      myContainer.repaint();
      ...
   }
}
```



Overriding `paint()` (cont.)

- Always perform the drawing in the overriden **`paint()`** method.
  - Never save the **`Graphics`** object and use it in another method to draw things! If you do so:
    - Drawn things would vanish the next time **`repaint()`** is called …
    - Drawn things would be located in wrong positions…
- The first line of the overriden **`paint()`** method **must** be **`super.paint()`** **!**
  - default **`paint()`** method performs other important operations necessary for updating component's screen appearance…

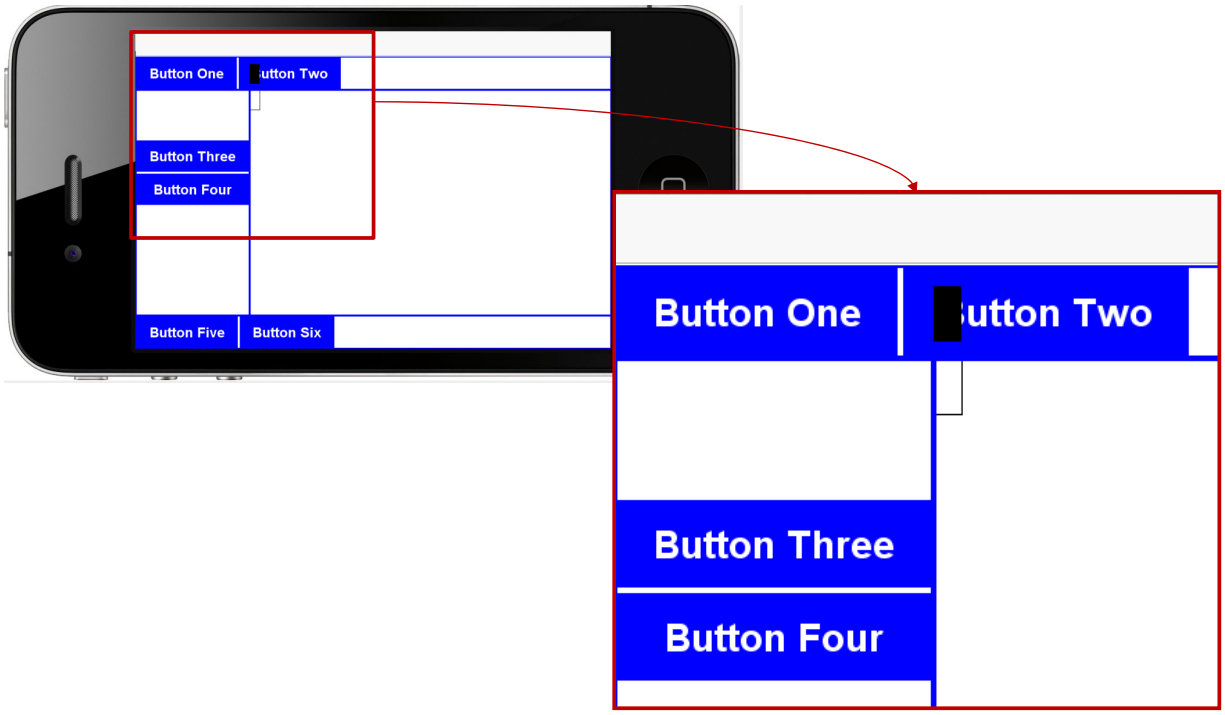
Non-working example

```
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.)



Importance of getX()/getY()

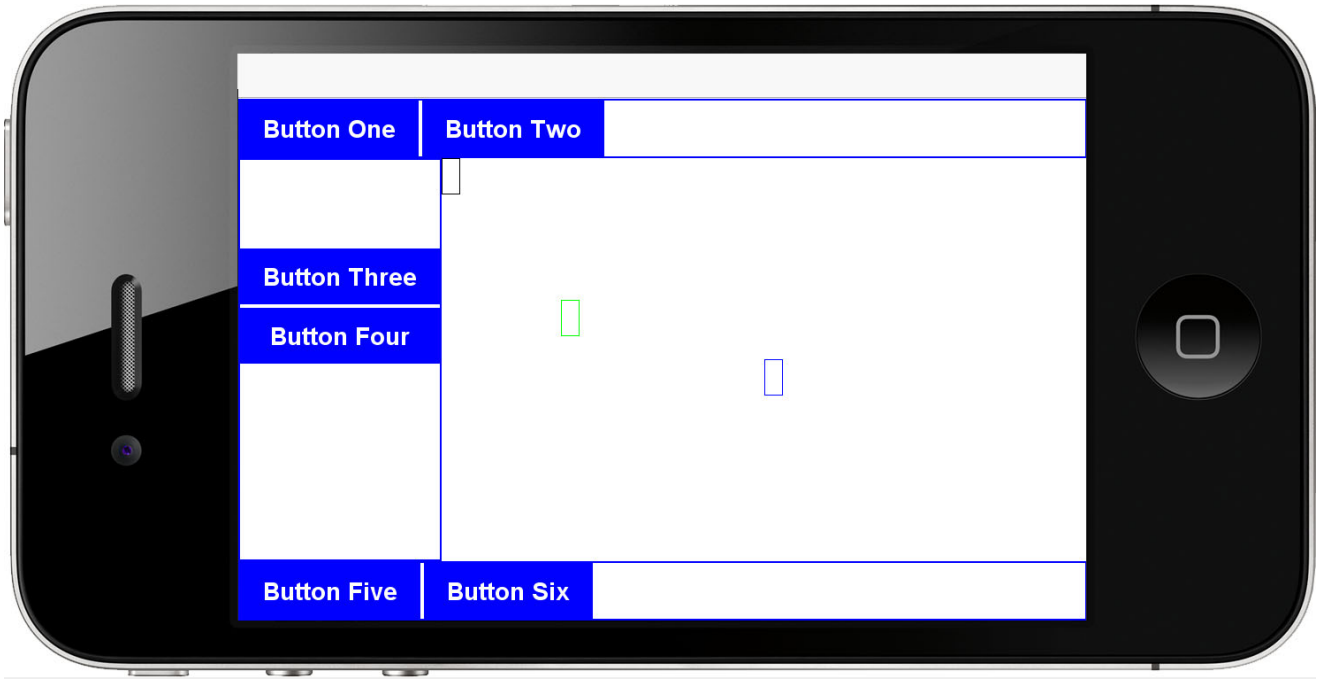
Assume we would like to draw a rectangle in the middle of **`CustomContainer`**.

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

```
public void paint(Graphics g){
 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);
}
```

Importance of getX()/getY() (cont.)

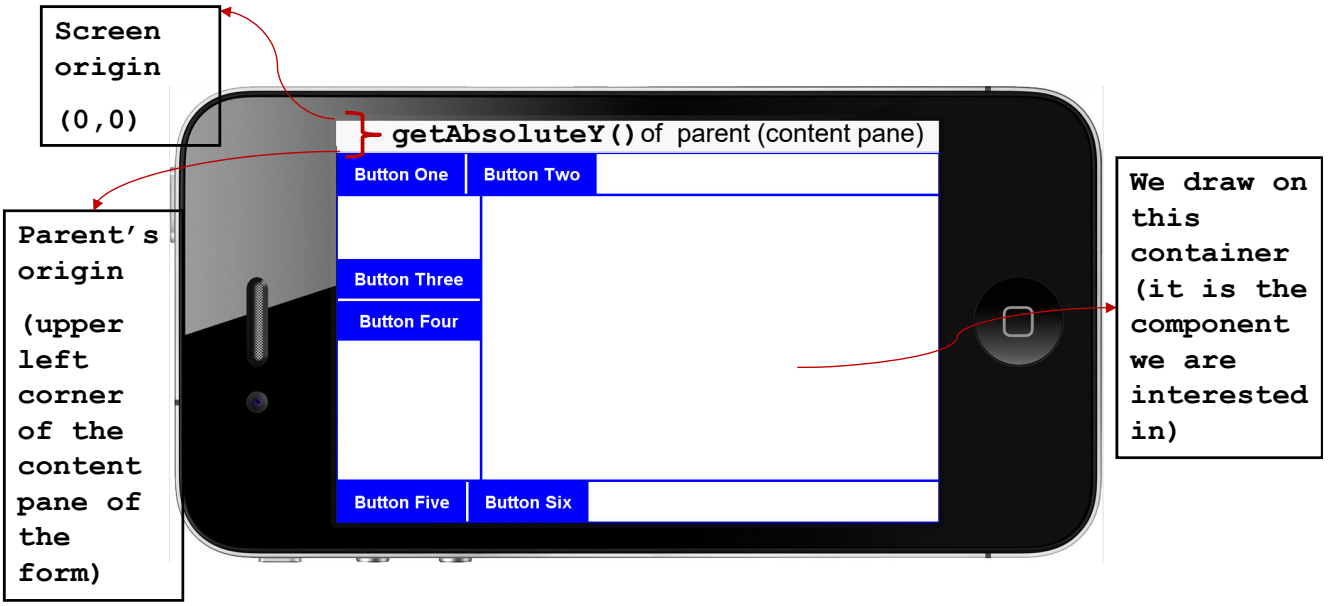
Only the blue rectangle would appear in the center of the **`CustomContainer`**…



Pointer Graphics

- We would like to draw a rectangle wherever the user presses on the **`CustomContainer`**.
- Pointer pressed gets coordinates relative to the screen origin (upper left corner of the screen).
- However draw methods expect coordinates relative to the component's parent's origin.
- You can convert screen coordinate to parent coordinate using **`getAbsoluteX()`** and **`getAbsoluteY()`** methods of the parent container.
- You can get the parent using **`getParent()`** method of the component.

Pointer Graphics (cont.)



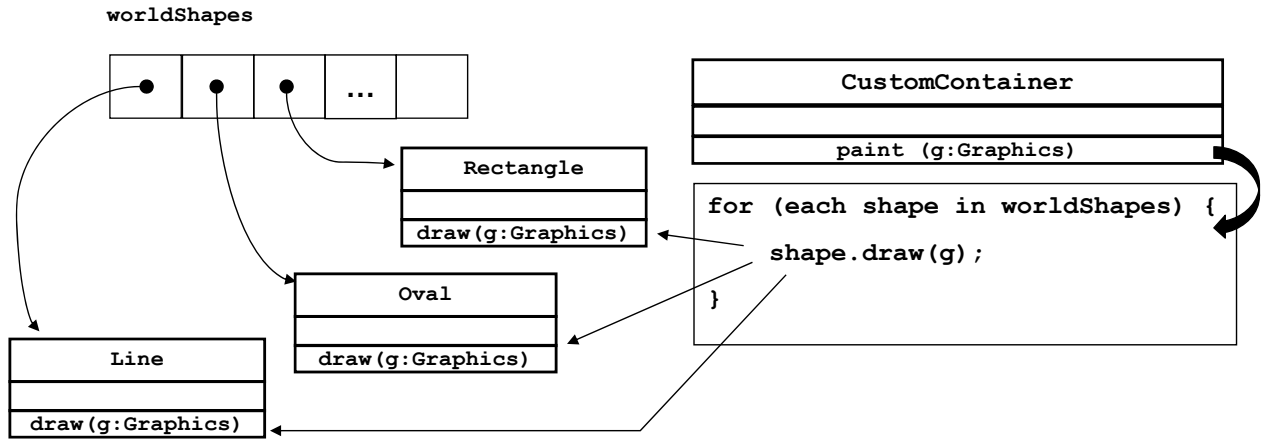
`getAbsoluteX()` of parent (content pane) is 0 in this example…

Pointer Graphics Example

```
public class CustomContainer extends Container{

  private int iPx = 0;
  private int iPy = 0;

  @Override
  public void paint(Graphics g){
    super.paint(g);
    g.setColor(ColorUtil.BLACK);
    //make the point location relative to the component's parent's origin
    //and then draw the rectangle (below un-filled rect would appear in the CORRECT location)
    g.drawRect(iPx-getParent().getAbsoluteX(),iPy-getParent().getAbsoluteY(),20,40);
    //below filled rect would appear in the WRONG location
    g.fillRect(iPx,iPy, 20,40);
  }

  @Override
  public void pointerPressed(int x,int y){
    //save the pointer pressed location
    //it is relative the to the screen origin
    iPx = x;
    iPy = y;
    repaint();
  }
}
```

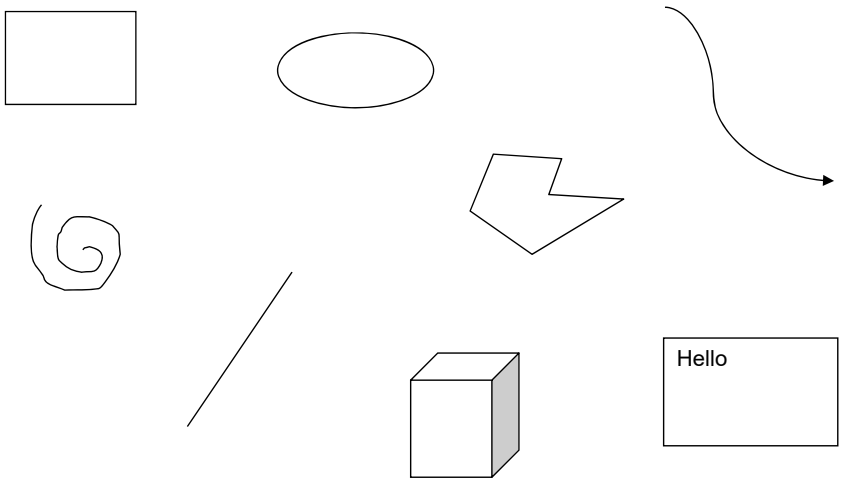
Maintaining Graphical State

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



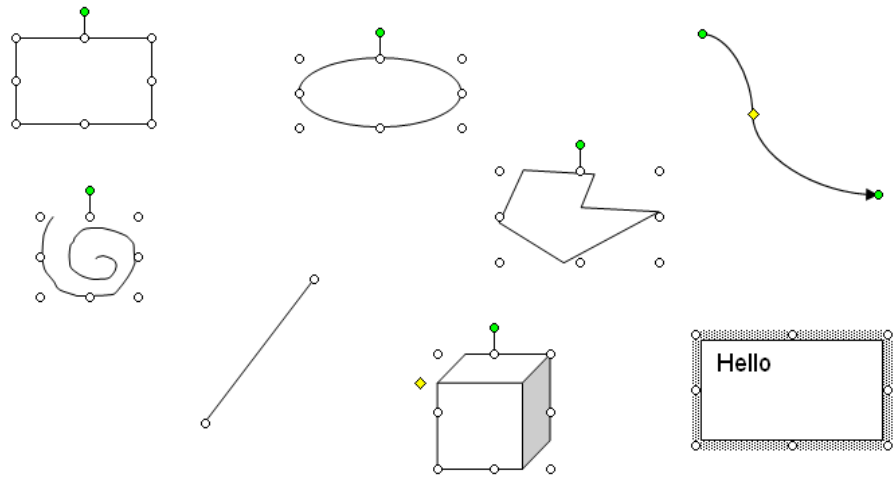
Object Selection

- Various *unselected* objects:



Object Selection (cont.)

- *Selected* versions of the same objects:

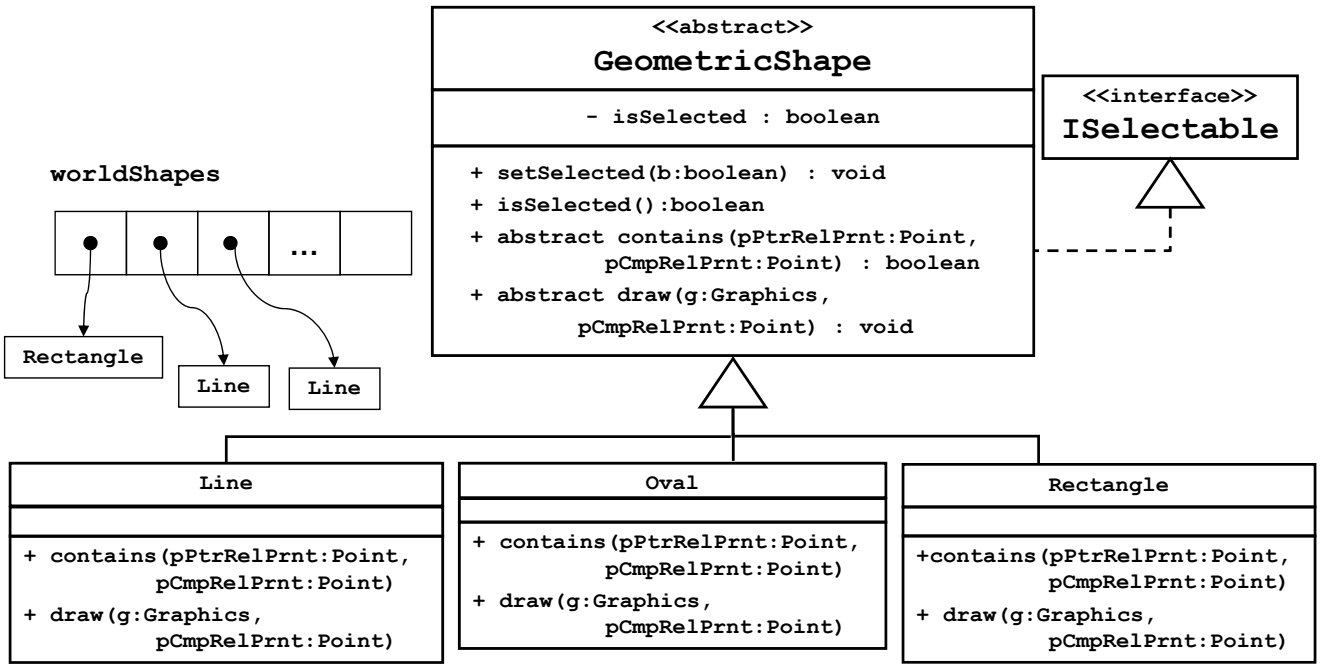


Defining "Selectability"

```
/** This interface declares the services (methods) provided
 *  by an object which is "Selectable" on the screen
 */
public interface ISelectable {

 // a way to mark an object as "selected" or not
 public void setSelected(boolean b);

 // a way to test whether an object is selected
 public boolean isSelected();
 // a way to determine if a pointer is "in" an object
 // pPtrRelPrnt is pointer position relative to the parent origin
 // pCmpRelPrnt is the component position relative to the parent origin
 public boolean contains(Point pPtrRelPrnt, Point pCmpRelPrnt);

 // a way to "draw" the object that knows about drawing
 // different ways depending on "isSelected"
 public void draw(Graphics g, Point pCmpRelPrnt);
}
```

Implementing Object Selection

(1) Expand objects to support selection



Implementing Object Selection (cont.)

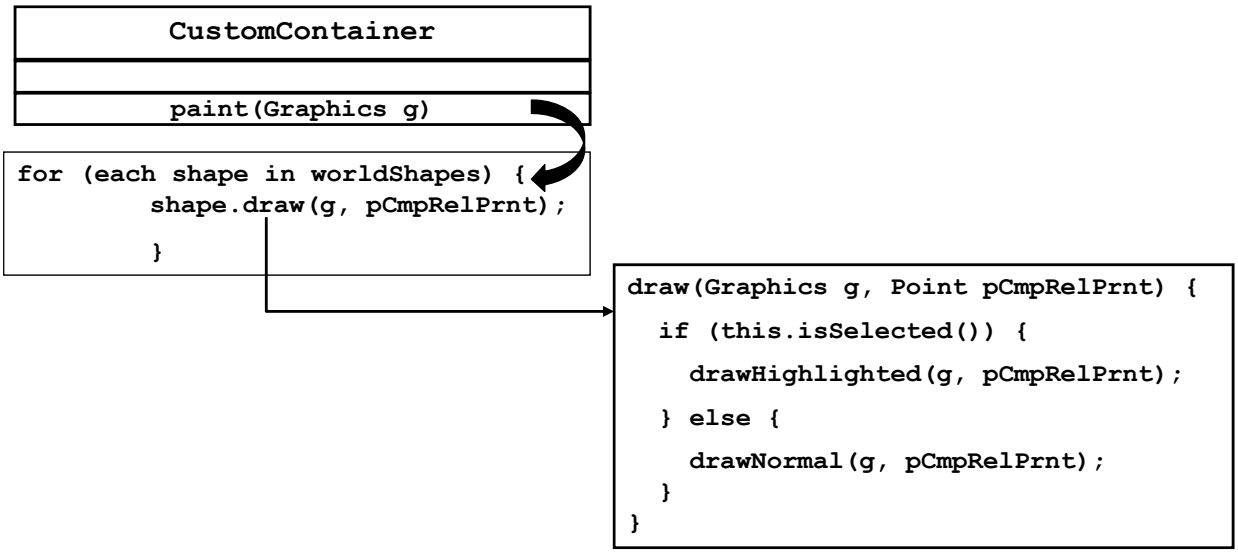
(2) On pointer pressed:

- Determine if pointer is “inside” any shape
  - if shape contains pointer, mark as “selected”
- Repaint container

```
//overriding pointerPressed() in CustomContainer
import com.codename1.ui.geom.Point;
void pointerPressed(int x, int y) {
//make pointer location relative to parent's origin
  x = x - getParent().getAbsoluteX();
  y = y - getParent().getAbsoluteY();
  Point pPtrRelPrnt = new Point(x, y);
  Point pCmpRelPrnt = new Point(getX(), getY());
  for (each shape in worldShapes) {
    if (shape.contains(pPtrRelPrnt, pCmpRelPrnt)) {
      shape.setSelected(true);
    } else {
      shape.setSelected(false);
    }
  }
  repaint();
}
```

Implementing Object Selection (cont.)

(3) Draw "selected" objects in different form



Object Selection Example

```
abstract public class GeometricShape implements ISelectable {
  private boolean isSelected;
  public void setSelected(boolean b) { isSelected = b; }
  public boolean isSelected() { return isSelected; }
  abstract void draw(Graphics g, Point pCmpRelPrnt);
  abstract boolean contains(Point pPtrRelPrnt, Point pCmpRelPrnt);
}
```

```
public class MyRect extends GeometricShape {
  //...[assign iShapeX and iShapeY to rect coordinates (upper left corner of rect
  //which is relative to the origin of the component) supplied in the constructor]
  public boolean contains(Point pPtrRelPrnt, Point pCmpRelPrnt) {
    int px = pPtrRelPrnt.getX(); // pointer location relative to
    int py = pPtrRelPrnt.getY(); // parent's origin
    int xLoc = pCmpRelPrnt.getX()+ iShapeX;// shape location relative
    int yLoc = pCmpRelPrnt.getY()+ iShapeY;// to parent's origin
    if ( (px >= xLoc) && (px <= xLoc+width)
      && (py >= yLoc) && (py <= yLoc+height) )
      return true; else return false;}
  public void draw(Graphics g, Point pCmpRelPrnt) {
    int xLoc = pCmpRelPrnt.getX()+ iShapeX;// shape location relative
    int yLoc = pCmpRelPrnt.getY()+ iShapeY;// to parent's origin
    if(isSelected())
      g.fillRect(xLoc, yLoc, width, height);
    else
      g.drawRect(xLoc, yLoc, width, height);}
}
```

```
public class ObjectSelectionForm extends Form {
  private Vector<GeometricShape> worldShapes = new Vector<GeometricShape>();
  public ObjectSelectionFrame() {
  // ...code here to initialize the form with a CustomContainer...
  //specify rect coordinates (relative to the origin of component), size, and color
    worldShapes.addElement(new MyRect(100, 100, 50, 50, ColorUtil.BLACK));
    worldShapes.addElement(new MyRect(200, 200, 100, 100, ColorUtil.GREEN));}
}
```

```
public class CustomContainer extends Container {
  //…assume we pass worldShapes to the constructor of CustomContainer
  public void paint(Graphics g) {
       super.paint(g);
       Point pCmpRelPrnt = new Point(getX(), getY());
       for(int i=0; i<worldShapes.size();i++)
            worldShapes.elementAt(i).draw(g, pCmpRelPrnt);}
  public void pointerPressed(int x, int y) {
       x = x - getParent().getAbsoluteX();
       y = y - getParent().getAbsoluteY();
       Point pPtrRelPrnt = new Point(x, y);
       Point pCmpRelPrnt = new Point(getX(), getY());
       for(int i=0;i<worldShapes.size();i++) {
         if(worldShapes.elementAt(i).contains(pPtrRelPrnt, pCmpRelPrnt))
           worldShapes.elementAt(i).setSelected(true);
         else
           worldShapes.elementAt(i).setSelected(false);
       }
       repaint(); }
}
```