Five-Minute Review

1. What are *enumerated types*? How can we represent enumerations in Java?
2. What is *character arithmetic*?
3. What is a string, conceptually?
4. How do we check strings for equality?
5. What are the rules governing the order of expression evaluation?

Programming – Lecture 10

Detecting Bugs

Object-Oriented Graphics (Chapter 9)

- **`acm.graphics`** package
- **`GCanvas`**
- Encapsulated Coordinates
- **`GMath, GObjects`**
- Interfaces
- **`GLabel`**, **`GRect`**, **`GOval`**, **`GLine`**, **`GArc`**
- **`GImage`**, **`GPolygon`**, **`GCompound`**
- Graphical Object Decomposition

Detecting Bugs

println-debugging

Using *debugger* (in IDE, e.g. Eclipse)

Test class, *unit testing*

Assertions

*Experience has shown that writing **assertions** while programming is one of the **quickest and most effective** ways to detect and correct bugs.*
As an added benefit, assertions serve to document the inner workings of your program, enhancing maintainability.

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

Debugging with Eclipse

- Set breakpoints simply by double clicking left of line number
- To start debugger, click on "bug icon"
- Can inspect current variables
- Can control further execution:
  - Step Into (F5)
  - Step Over (F6)
  - Step Return (F7)
  - Resume (F8)

`assert` *condition* **`;`**

`assert` *condition* **`:`** *error-string* **`;`**

- Must also enable in Eclipse:
  Preferences → Java → Installed JREs → Edit → Default VM arguments: **`-ea`**
- Disabled assertions have (almost/usually) no performance penalty
  https://stackoverflow.com/a/40919125/2308035
- Should be used for checking invariants, preconditions (of non-public methods), postconditions
- Should be used instead of comments

Invariants

Invariant: something expected to hold at certain point.

Before assertions, invariants typically stated as a comment:

```
int n = countSomething();
if (n % 2 == 0) {
  ...
} else {
  // It must be n % 2 == 1
  ...
}
```

Problem: no check that assumption holds

Consider e.g. negative `n`

Invariant in Java: `n == n / d * d + n % d` for all int's `n`, `d`

Invariants – With Assertions

```
int n = countSomething();
if (n % 2 == 0) {
  ...
} else {
  assert n % 2 == 1 :
  "n % 2 = " + n % 2 + ", not 1!";
  ...
}
```

Preconditions

Preconditions must hold *before* some computation, e.g., on method arguments

For *non-public methods*, use *assertions* to check input arguments:

```
private void setInterval(int i) {
  assert i > 0 && i <= 1000/MAX_RATE :
    i + " is not legal interval";
  ...
}
```

Preconditions

For *public* methods, don't use assertion, but *throw exception* upon invalid input arguments:

```
public void setRate(int rate) {
  if (rate <= 0 || rate > MAX_RATE)
    throw new IllegalArgumentException(
      "Illegal rate: " + rate);
  setInterval(1000/rate);
}
```

Postconditions

Postconditions must hold *after* some computation, e.g., on method return value

```
public int countSomething() {
   ...
   assert count >= 0 :
     "got negative count!";
   return count;
}
```

Aside 1: Hoare Logic

- Hoare logic (a.k.a. Floyd-Hoare logic) to formally analyze computer programs
- Key component is the *Hoare triple*, of form { Precondition } Command { Postcondition }
- Set of reasoning rules, such as

$$\frac{\{B \wedge P\}\ S\ \{Q\} \quad , \quad \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{endif}\ \{Q\}}$$

- Can prove *partial correctness*: if program terminates, then it produces correct result

C.A.R. Hoare, An axiomatic basis for computer programming, *Communications of the ACM*, 12 (10): 576–580, Oct. 1969

Aside 2: Design by Contract

Clients and implementors of piece of software (e.g., a method) agree on *contract*, specifying

- Preconditions – obligations on clients
- Postconditions – obligations on implementors
- (Class) invariants – obligations on both

Bertrand Meyer, Applying Design by Contract, *Computer* (IEEE), 25 (10),Oct. 1992

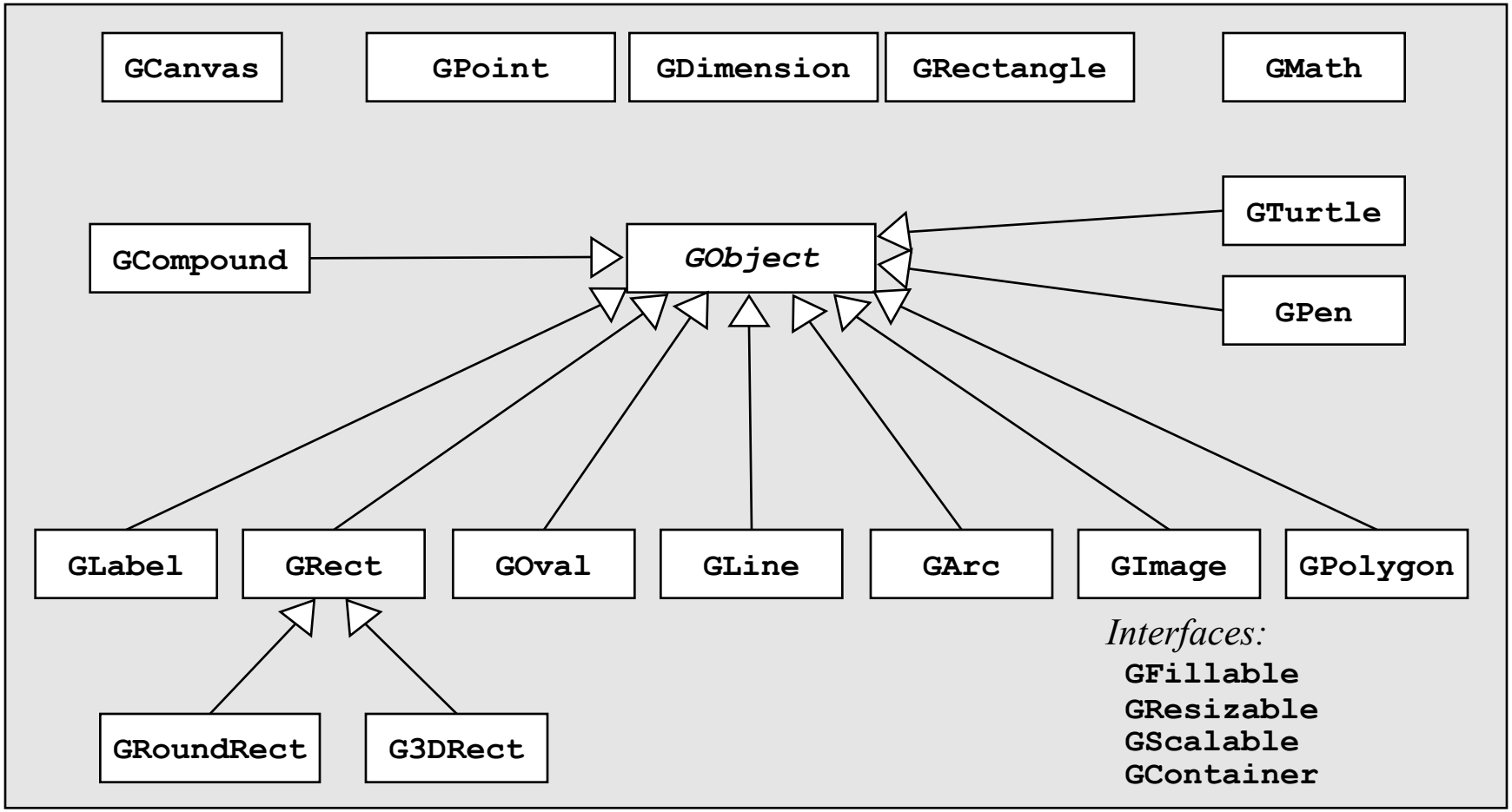
Summary Assertions

- Assertions are a very effective mechanism for detecting bugs early, both during development and when deployed
- Assertions are per default disabled, in which case they carry no run-time overhead
- Common uses of assertions are the checking of invariants, preconditions, postconditions
- Assertions should *not* replace input checks on public methods, which should be performed irrespective of whether assertions are enabled or not

Recall: Stacking Order

Wikipedia / marcel4995 / CC

`acm.graphics` Package



`GObject` is *abstract class*

Calls to these methods are *forwarded* from **`GraphicsProgram`** to **`GCanvas`** :

`add(`*object*`)`**	Adds the object to the canvas at the front of the stack
`add(`*object*`, `***x*`, `***y*`)`**	Moves the object to (*x*, *y*) and then adds it to the canvas
`remove(`*object*`)`**	Removes the object from the canvas
`removeAll()`	Removes all objects from the canvas
`getElementAt(`*x*`, `***y*`)`**	Returns the frontmost object at (*x*, *y*), or **`null`** if none
`getWidth()`	Returns the width in pixels of the entire canvas
`getHeight()`	Returns the height in pixels of the entire canvas
`setBackground(`*c*`)`**	Sets the background color of the canvas to *c*.

Methods in **`GraphicsProgram`** only

`pause(`*milliseconds*`)`**	Pauses the program for the specified time in milliseconds
`waitForClick()`	Suspends the program until the user clicks the mouse

Encapsulated Coordinates

`GPoint(`*x*`, `*y*`)`**

`GDimension(`*width*`, `*height*`)`**

`GRectangle(`*x*`, `*y*`, `*width*`, `*height*`)`**

`getX`, **`getY`**, **`getWidth`**, **`getHeight`**

`GMath` Class

`GMath.sinDegrees(`*theta***`)`**	Returns the sine of *theta,* measured in degrees
`GMath.cosDegrees(`*theta***`)`**	Returns the cosine of *theta*
`GMath.tanDegrees(`*theta***`)`**	Returns the tangent of *theta*
`GMath.angle(`*x*`, `*y***`)`**	Returns the angle in degrees formed by the line connecting the origin to the point (x, y)
`GMath.angle(`x_0**`, `**y_0**`, `**x_1**`, `**y_1**`)`**	Returns the angle in degrees formed by the line connecting the points (x_0, y_0) and (x_1, y_1)
`GMath.distance(`*x*`, `*y***`)`**	Returns the distance from the origin to (x, y)
`GMath.distance(`x_0**`, `**y_0**`, `**x_1**`, `**y_1**`)`**	Returns the distance from (x_0, y_0) to (x_1, y_1)
`GMath.toRadians(`*degrees***`)`**	Converts an angle from degrees to radians
`GMath.toDegrees(`*radians***`)`**	Converts an angle from radians to degrees
`GMath.round(`*x***`)`**	Returns the closest **`int`** to x

Methods Common to `GObjects`

`setLocation(`*x*`, `***y*`)`**	Resets the location of the object to the specified point
`move(`*dx*`, `***dy*`)`**	Moves the object *dx* and *dy* pixels from its current position
`movePolar(`*r*`, `***theta*`)`**	Moves the object *r* pixel units in direction *theta*
`getX()`	Returns the *x* coordinate of the object
`getY()`	Returns the *y* coordinate of the object
`getWidth()`	Returns the horizontal width of the object in pixels
`getHeight()`	Returns the vertical height of the object in pixels
`contains(`*x*`, `***y*`)`**	Returns **`true`** if the object contains the specified point
`setColor(`*c*`)`**	Sets the color of the object to the **`Color`** *c*
`getColor()`	Returns the color currently assigned to the object
`setVisible(`*flag*`)`**	Sets the visibility flag (**`false`** = invisible, **`true`** = visible)
`isVisible()`	Returns **`true`** if the object is visible
`sendToFront()`	Sends the object to the front of the stacking order
`sendToBack()`	Sends the object to the back of the stacking order
`sendForward()`	Sends the object forward one position in the stacking order
`sendBackward()`	Sends the object backward one position in the stacking order

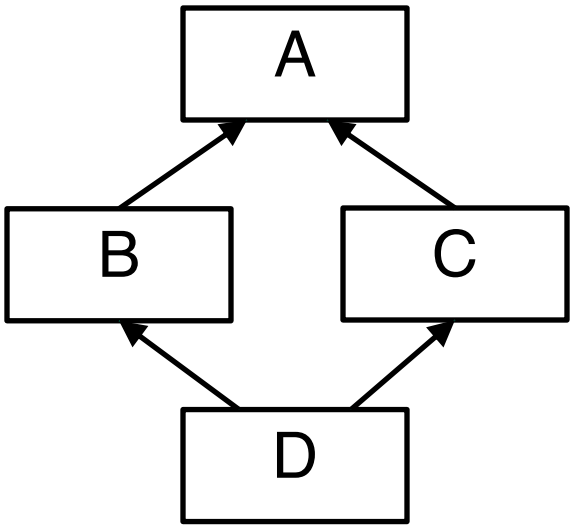
Aside: Multiple Inheritance

- *Multiple inheritance* (*Mehrfachvererbung*): a class has *multiple* superclasses
- Allowed in some languages (e.g. C++)
- Conceptually tricky – consider diamond problem (next slide)
- **No multiple inheritance** in Java
- In Java, classes have exactly one parent class, except **`java.lang.Object`**, which has none

https://en.wikipedia.org/wiki/Multiple_inheritance

Diamond Problem

- class A implements method foo()
- classes B and C inherit from A, both override foo()
- class D inherits from B and C
- **Problem:** which foo() should D use?



https://www.javabrahman.com/java-8/java-8-multiple-inheritance-conflict-resolution-rules-and-diamond-problem/

Interfaces

- *Interface*: conceptually, collection of method signatures
- Some functionality of multiple inheritance: classes can *implement* several interfaces
- Generally, interface (unlike superclass) does not implement methods itself
- Since Java 8, interface can implement default and static methods (again: diamond problem)

```
public class GRect extends GObject
   implements GFillable, GResizable, GScalable
```

Interfaces in `acm.graphics`

GFillable (**GArc**, **GOval**, **GPolygon**, **GRect**)

setFilled(*flag* **)**	Sets the fill state for the object (**false** = outlined, **true** = filled)
isFilled()	Returns the fill state for the object
setFillColor(*c* **)**	Sets the color used to fill the interior of the object to *c*
getFillColor()	Returns the fill color

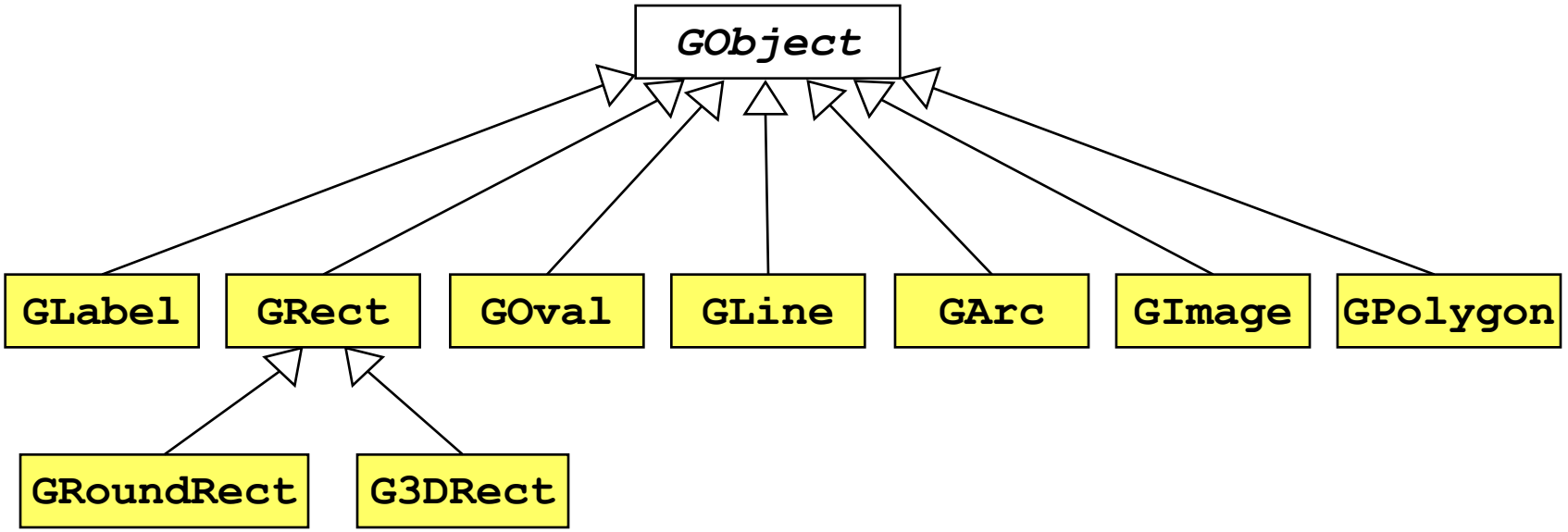
GResizable (**GImage**, **GOval**, **GRect**)

setSize(*width* **,** *height* **)**	Sets the dimensions of the object as specified
setBounds(*x* **,** *y* **,** *width* **,** *height* **)**	Sets the location and dimensions together

GScalable (**GArc**, **GCompound**, **GLine**, **GImage**, **GOval**, **GPolygon**, **GRect**)

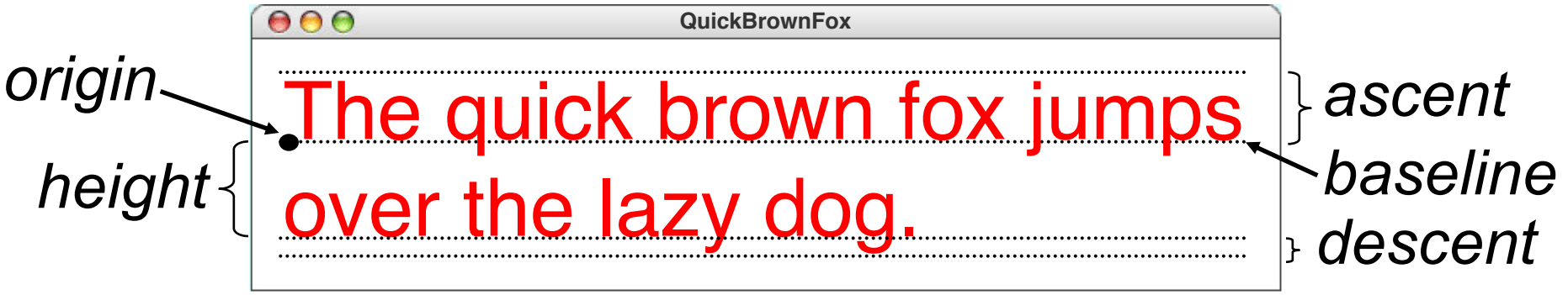
scale(*sf* **)**	Scales both dimensions of the object by *sf*
scale(*sx* **,** *sy* **)**	Scales the object by *sx* horizontally and *sy* vertically

Shape Classes



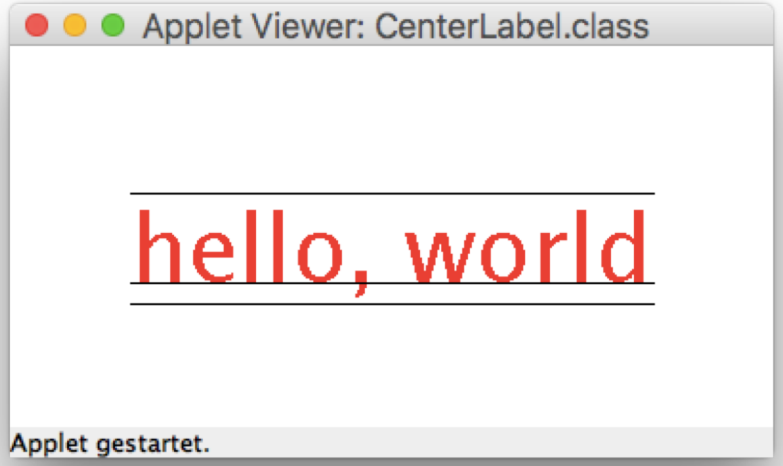
Each corresponds to method in **`Graphics`** class in **`java.awt`** package

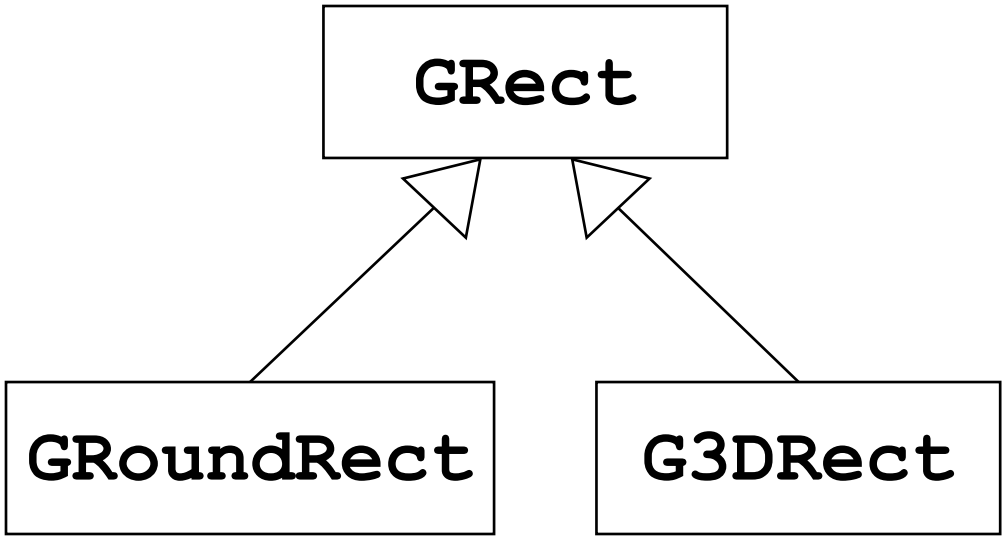
```
public class HelloProgram extends
GraphicsProgram {
   public void run() {
      GLabel label = new
        GLabel("hello, world", 100, 75);
      label.setFont("SansSerif-36");
      label.setColor(Color.RED);
      add(label);
   }
}
```


QuickBrownFox
origin
height
The quick brown fox jumps
over the lazy dog.
ascent
baseline
descent

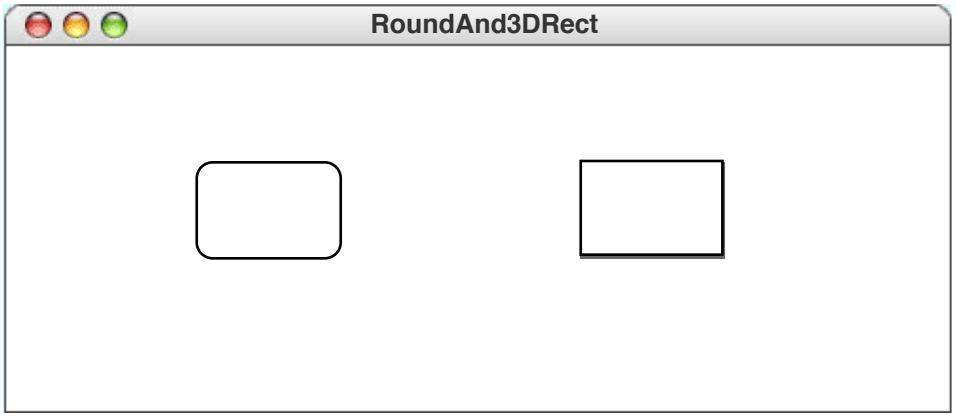
```
public class CenterLabel extends GraphicsProgram {
 public void run() {
    setSize(400, 200);
    GLabel label = new GLabel("hello, world");
    label.setFont("SansSerif-48");
    label.setColor(Color.RED);

    double x = (getWidth() - label.getWidth()) / 2;
    double x1 = x + label.getWidth();
    double y = (getHeight() + label.getAscent()) / 2;
    double y1 = y - label.getAscent();
    double y2 = y + label.getDescent();

    add(label, x, y);
    add(new GLine(x, y, x1, y));
    add(new GLine(x, y1, x1, y1));
    add(new GLine(x, y2, x1, y2));
  }
}
```

Note: Most characters have smaller height than ascent

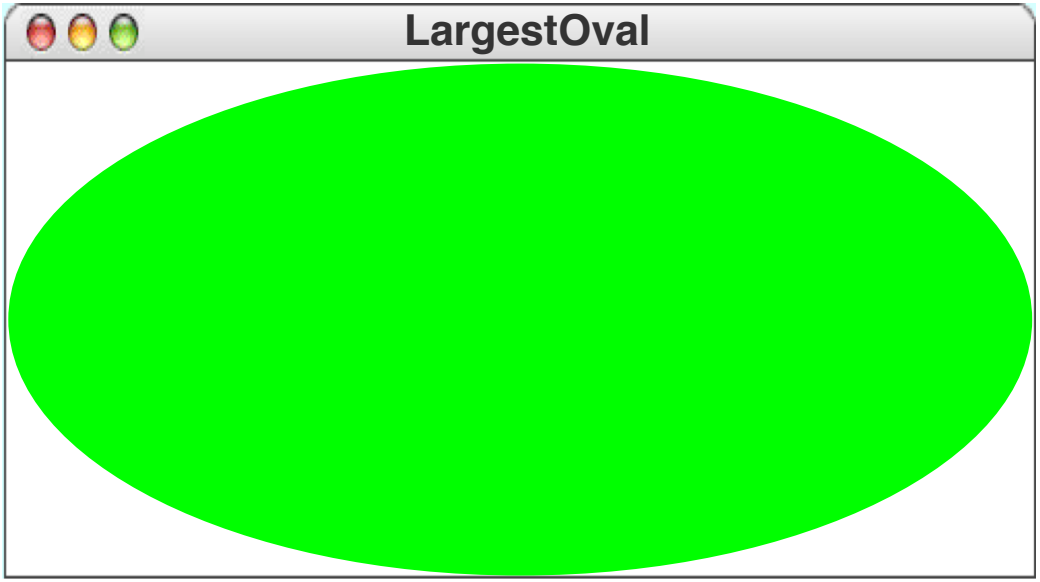





```
add(new GRoundRect(100, 60, 75, 50));
add(new G3DRect(300, 60, 75, 50));
```

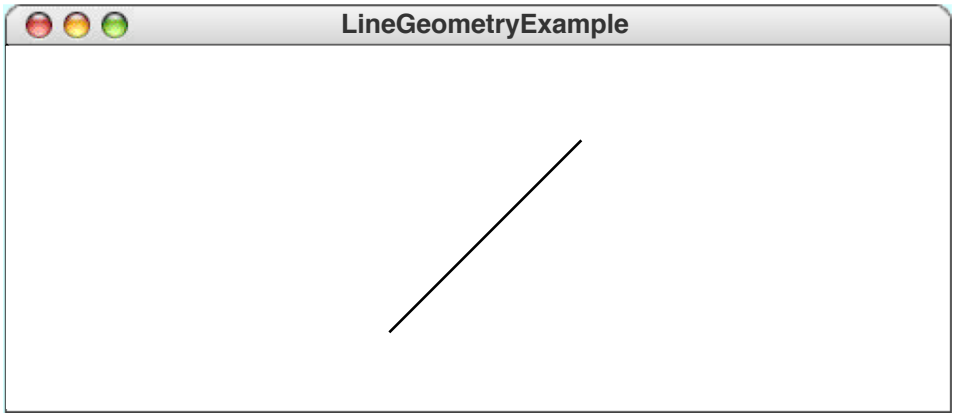


```
public void run() {
    GOval oval = new
      GOval(getWidth(), getHeight());
    oval.setFilled(true);
    oval.setColor(Color.GREEN);
    add(oval, 0, 0);
}
```

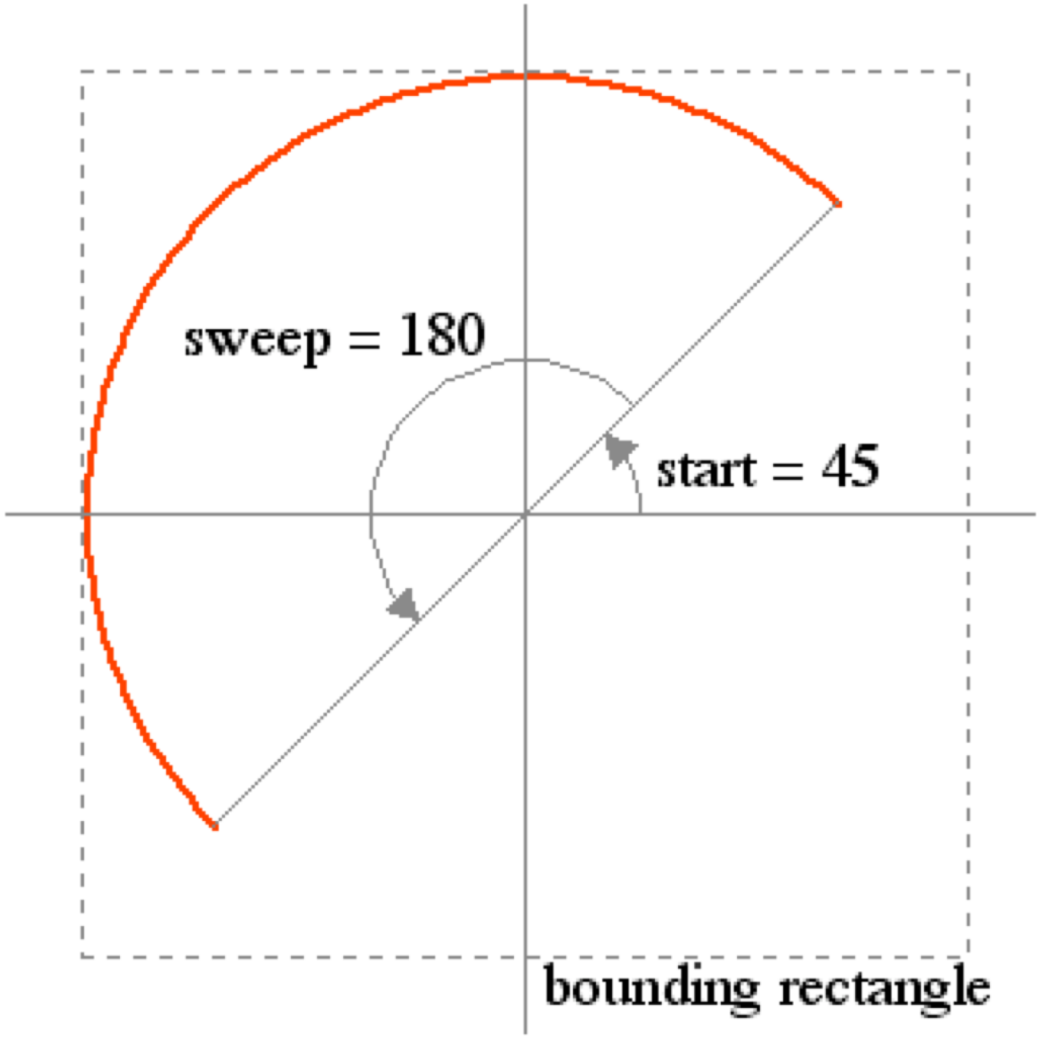


Setting Points in `GLine`

```
public void run() {
   GLine line = new GLine(0, 0, 100, 100);
   add(line);
   line.setLocation(200, 50);
   line.setStartPoint(200, 150);
   line.setEndPoint(300, 50);
}
```



```
GArc(double width, double height,
     double start, double sweep)
```



Filled Arcs

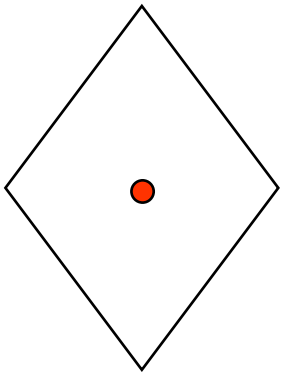
```
public void run() {
    GArc arc = new GArc(0, 0,
      getWidth(), getHeight(), 0, 90);
    arc.setFilled(true);
    add(arc);
}
```

```
public void run() {
   add(new GImage("EarthFromApollo17.jpg"));
   addCitation("Courtesy NASA/JPL-Caltech");
}
```

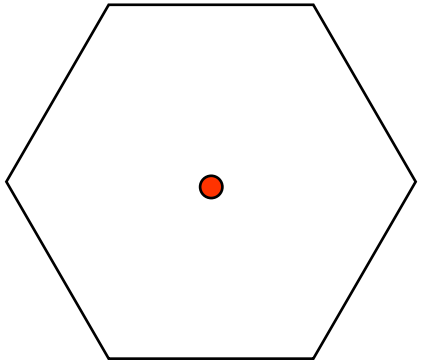
GPolygon

Polygons, vertices, edges

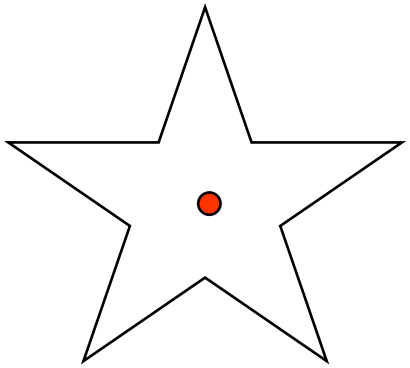
Reference points



diamond



regular hexagon



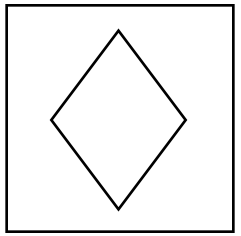
five-pointed star

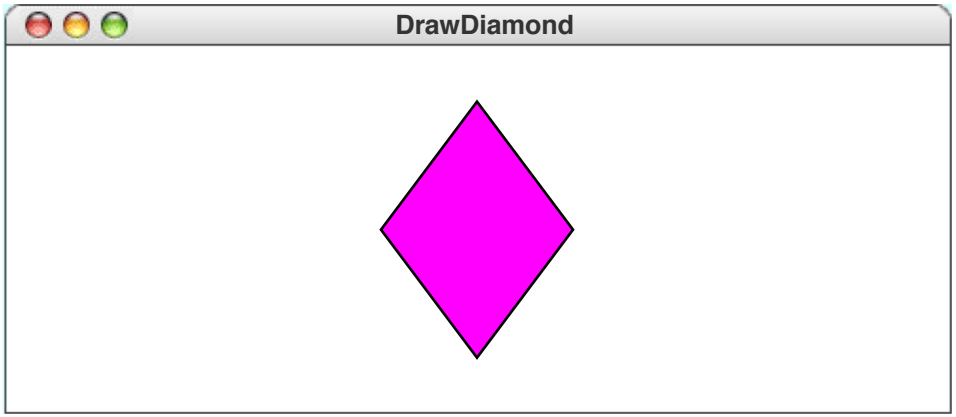
Drawing a Diamond (`addVertex`)

```
public void run() {
```

```
private GPolygon createDiamond(double width, double height) {
   GPolygon diamond = new GPolygon();
   diamond.addVertex(-width / 2, 0);
   diamond.addVertex(0, -height / 2);
   diamond.addVertex(width / 2, 0);
   diamond.addVertex(0, height / 2);
   return diamond;
}
```

diamond





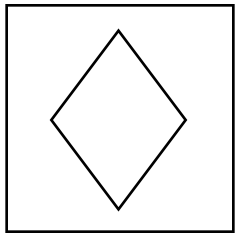
skip simulation

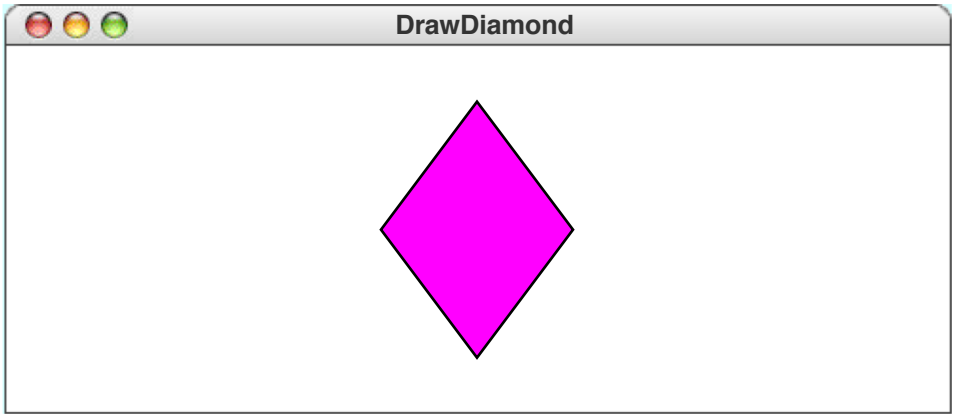
Drawing a Diamond (`addEdge`)

```
public void run() {
```

```
private GPolygon createDiamond(double width, double height) {
   GPolygon diamond = new GPolygon();
   diamond.addVertex(-width / 2, 0);
   diamond.addEdge(width / 2, -height / 2);
   diamond.addEdge(width / 2, height / 2);
   diamond.addEdge(-width / 2, height / 2);
   diamond.addEdge(-width / 2, -height / 2);
   return diamond;
}
```

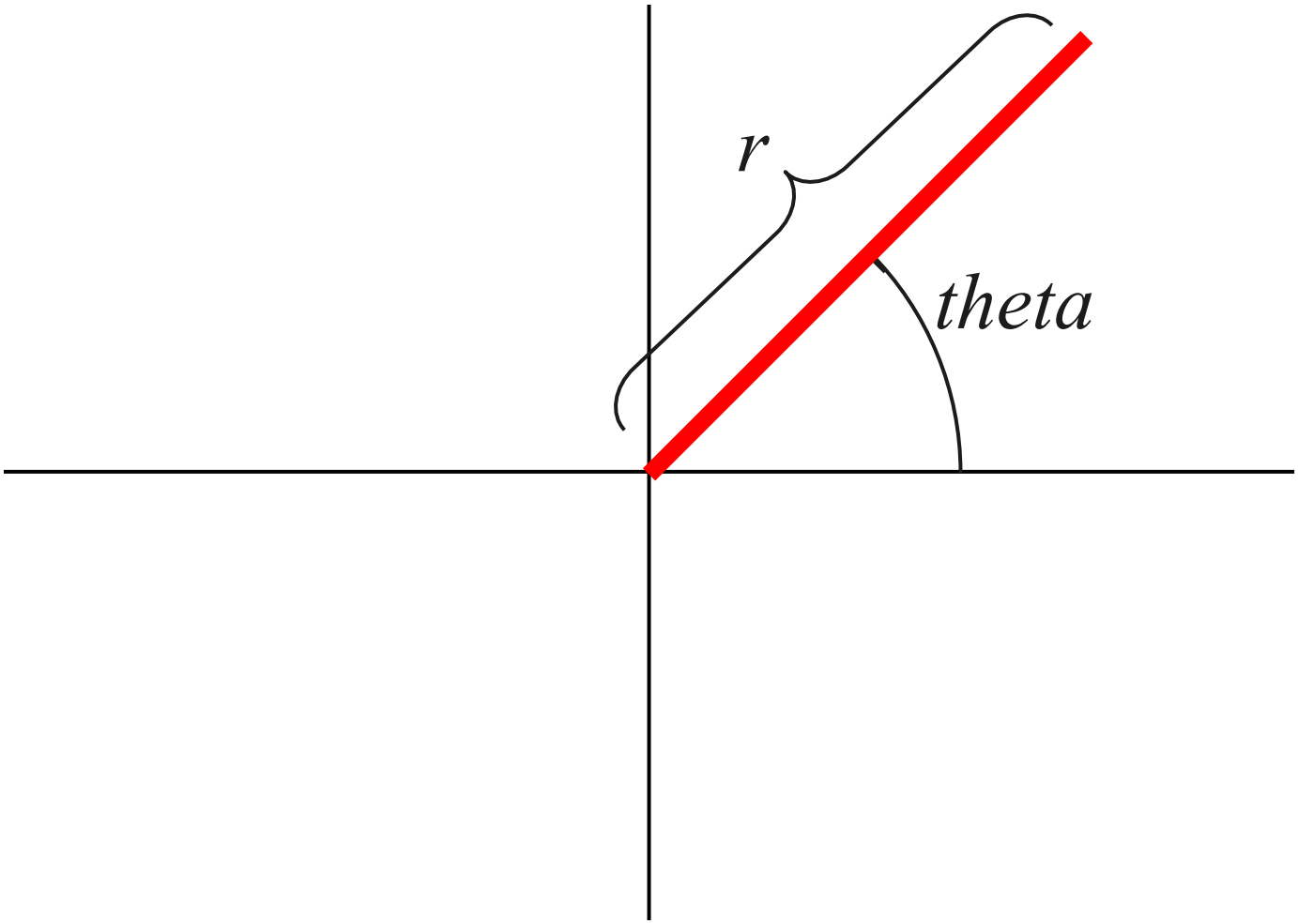
diamond





skip simulation

`addPolarEdge(`*r*`, `*theta*`)`

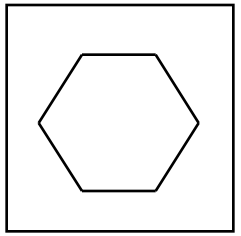


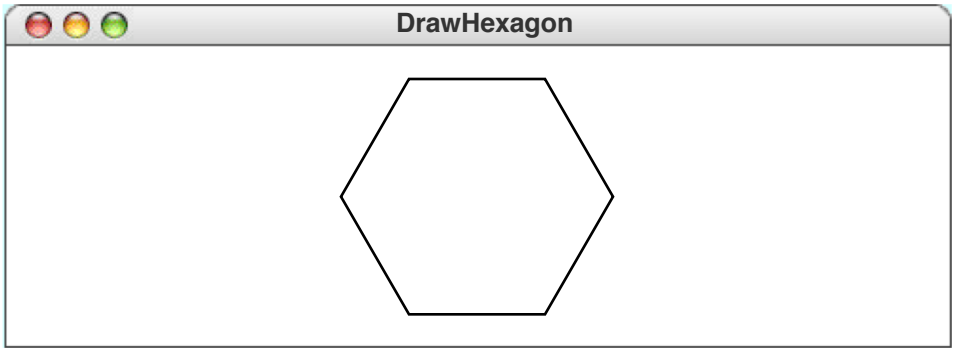
Hexagon

```
public void run() {
```

```
private GPolygon createHexagon(double side) {
   GPolygon hex = new GPolygon();
   hex.addVertex(-side, 0);
   int angle = 60;
   for (int i = 0; i < 6; i++) {
      hex.addPolarEdge(side, angle);
      angle -= 60;
   }
   return hex;
}
```

hex





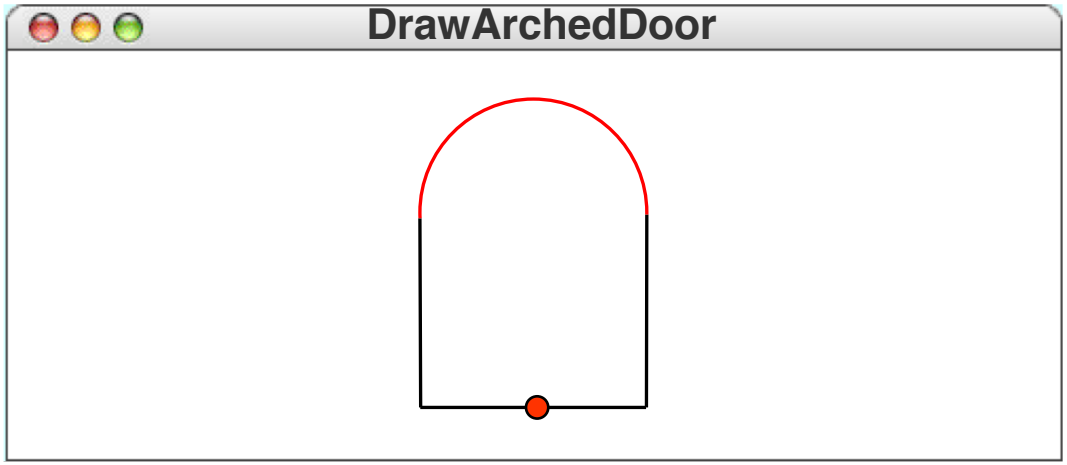
skip simulation

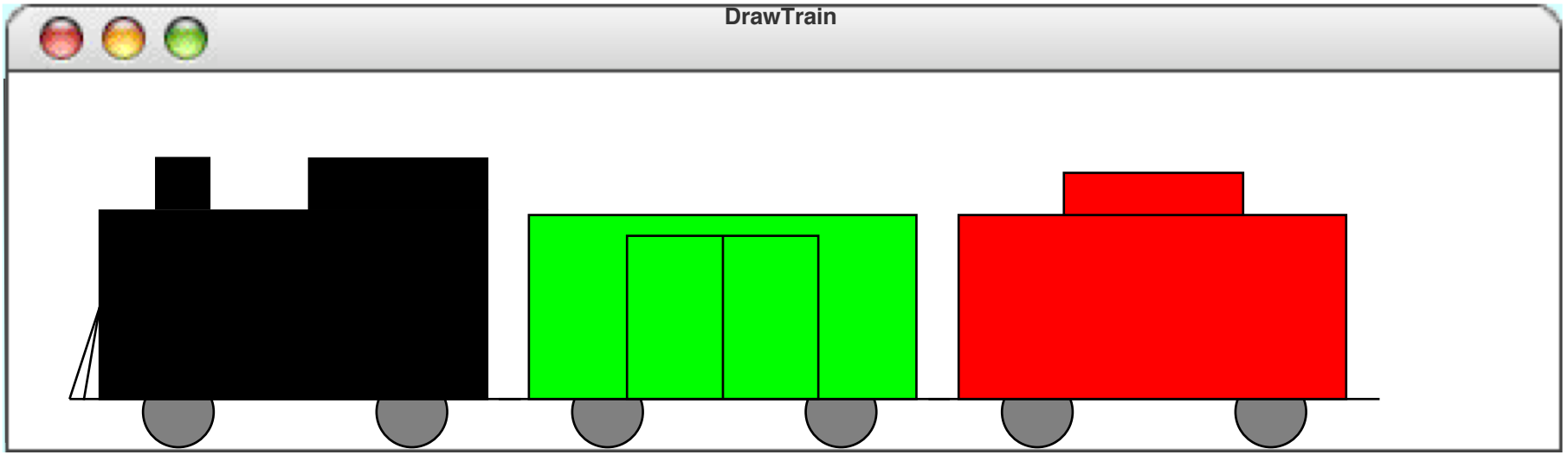
Defining `GPolygon` Subclasses

```
public class GHexagon extends GPolygon {

   public GHexagon(double side) {
      addVertex(-side, 0);
      int angle = 60;
      for (int i = 0; i < 6; i++) {
         addPolarEdge(side, angle);
         angle -= 60;
      }
   }

}
```

polygon`.addArc(`*width*`,` *height*`,` *start*`,` *sweep*`);`

```
public class GArchedDoor extends GPolygon {
  public GArchedDoor(double width,
                     double height) {
    double lengthOfVerticalEdge =
        height - width / 2;
    addVertex(-width / 2, 0);
    addEdge(width, 0);
    addEdge(0, -lengthOfVerticalEdge);
    addArc(width, width, 0, 180);
    addEdge(0, lengthOfVerticalEdge);
  }
}
```





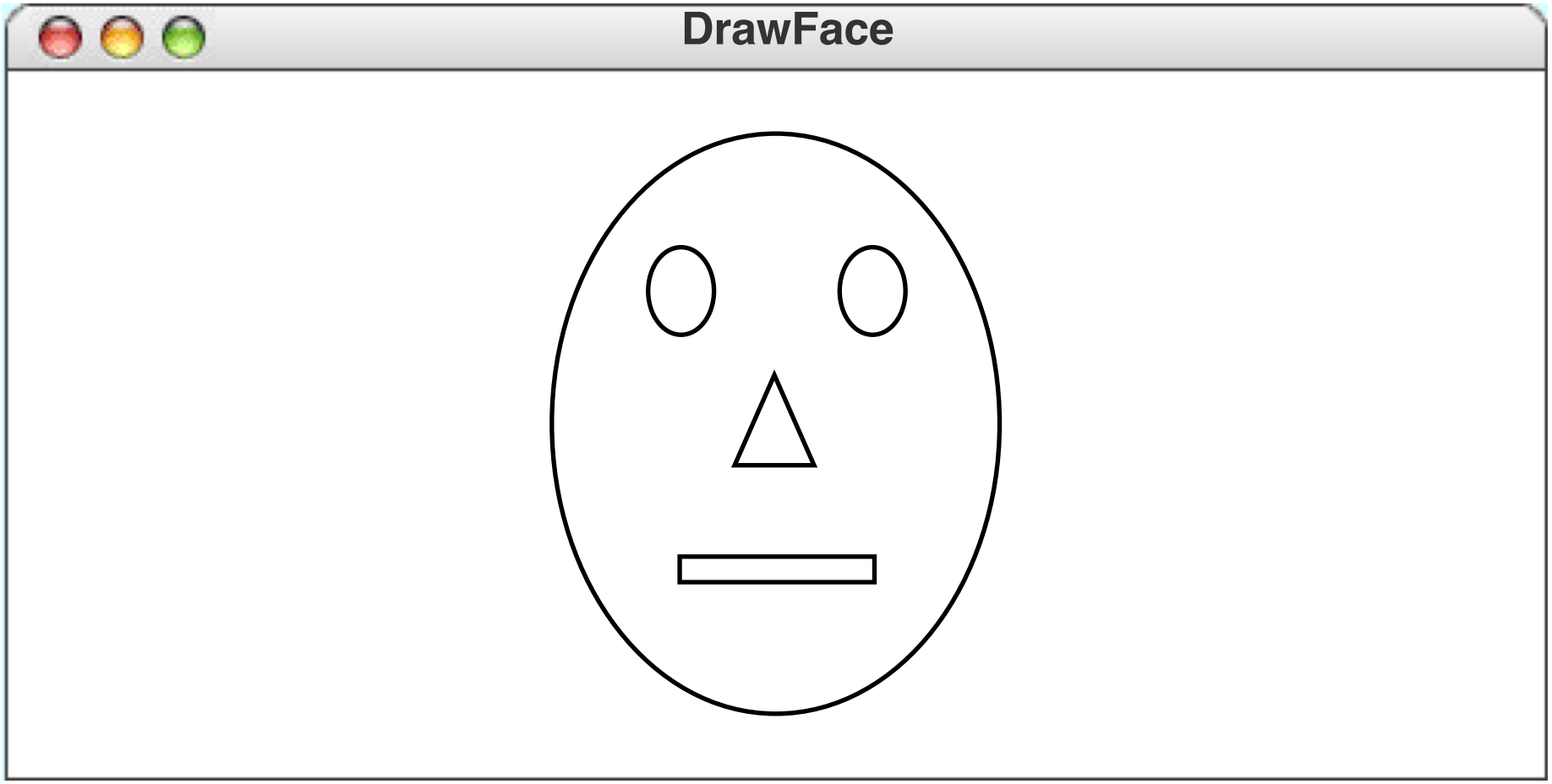
How to achieve this?

So far: method decomposition

Now: object decomposition

GCompound

Conceptually: **GCanvas** (can add objects to it)
+ **GObject** (can add it to a canvas)



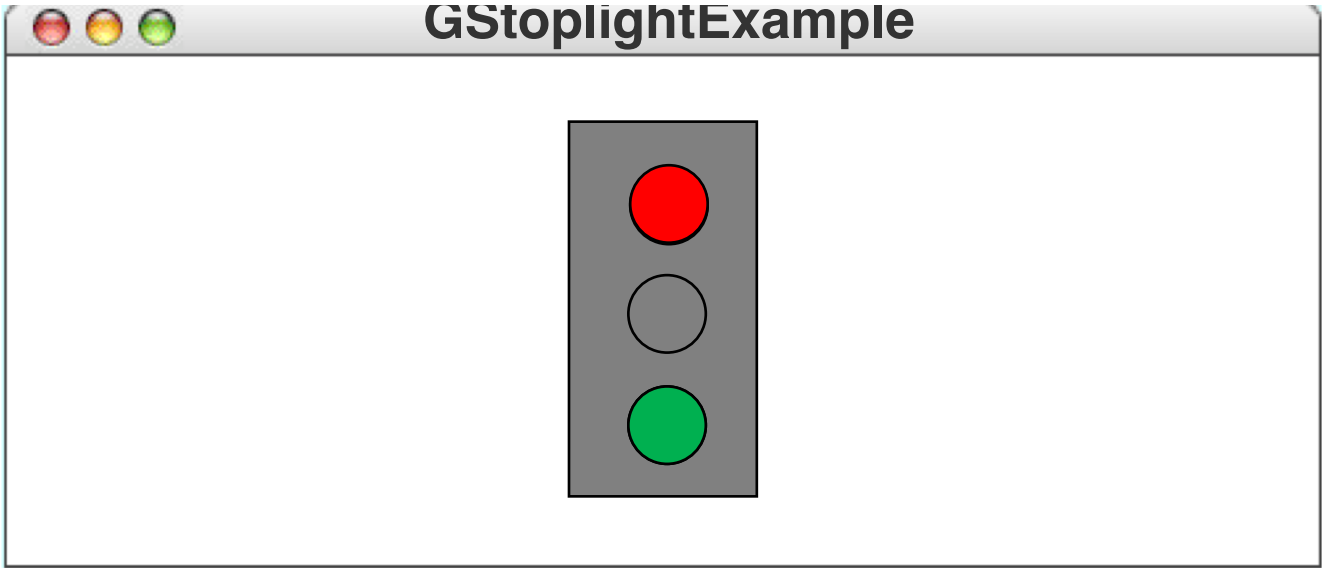
GFace

```
import acm.graphics.*;

/** Defines a compound GFace class */
public class GFace extends GCompound {

   /* Constants specifying feature size as a fraction of the head size */
   private static final double EYE_WIDTH    = 0.15;
   private static final double EYE_HEIGHT   = 0.15;
   private static final double NOSE_WIDTH   = 0.15;
   private static final double NOSE_HEIGHT  = 0.10;
   private static final double MOUTH_WIDTH  = 0.50;
   private static final double MOUTH_HEIGHT = 0.03;

   /* Private instance variables */
   private GOval head;
   private GOval leftEye, rightEye;
   private GPolygon nose;
   private GRect mouth;
```

```
   /** Creates a new GFace object with the specified dimensions */
   public GFace(double width, double height) {
      head = new GOval(width, height);
      leftEye = new GOval(EYE_WIDTH * width, EYE_HEIGHT * height);
      rightEye = new GOval(EYE_WIDTH * width, EYE_HEIGHT * height);
      nose = createNose(NOSE_WIDTH * width, NOSE_HEIGHT * height);
      mouth = new GRect(MOUTH_WIDTH * width, MOUTH_HEIGHT * height);
      add(head, 0, 0);
      add(leftEye, 0.25 * width - EYE_WIDTH * width / 2,
                   0.25 * height - EYE_HEIGHT * height / 2);
      add(rightEye, 0.75 * width - EYE_WIDTH * width / 2,
                    0.25 * height - EYE_HEIGHT * height / 2);
      add(nose, 0.50 * width, 0.50 * height);
      add(mouth, 0.50 * width - MOUTH_WIDTH * width / 2,
                 0.75 * height - MOUTH_HEIGHT * height / 2);
   }

   /* Creates a triangle for the nose */
   private GPolygon createNose(double width, double height) {
      GPolygon poly = new GPolygon();
      poly.addVertex(0, -height / 2);
      poly.addVertex(width / 2, height / 2);
      poly.addVertex(-width / 2, height / 2);
      return poly;
   }

}
```

Specifying Behavior of a **GCompound**

```
public void run() {
   GStoplight stoplight =
       new GStoplight();
   add(stoplight, getWidth() / 2,
       getHeight() / 2);
   stoplight.setState(Color.RED);
}
```



GStoplight

```
/**
 * Defines a GCompound subclass that displays a stoplight.  The
 * state of the stoplight must be one of the Color values RED,
 * YELLOW, or GREEN.
 */
public class GStoplight extends GCompound {

   /* Private constants */
   private static final double FRAME_WIDTH = 50;
   private static final double FRAME_HEIGHT = 100;
   private static final double LAMP_RADIUS = 10;

   /* Private instance variables */
   private Color state;
   private GOval redLamp;
   private GOval yellowLamp;
   private GOval greenLamp;
```

GStoplight

```
/** Creates a new Stoplight object, which is initially GREEN */
 public GStoplight() {
    GRect frame = new GRect(FRAME_WIDTH, FRAME_HEIGHT);
    frame.setFilled(true);
    frame.setFillColor(Color.GRAY);
    add(frame, -FRAME_WIDTH / 2, -FRAME_HEIGHT / 2);
    double dy = FRAME_HEIGHT / 4 + LAMP_RADIUS / 2;
    redLamp = createFilledCircle(0, -dy, LAMP_RADIUS);
    add(redLamp);
    yellowLamp = createFilledCircle(0, 0, LAMP_RADIUS);
    add(yellowLamp);
    greenLamp = createFilledCircle(0, dy, LAMP_RADIUS);
    add(greenLamp);
    setState(Color.GREEN);
 }
```

GStoplight

```
/** Sets the state of the stoplight */
public void setState(Color color) {
   if (color.equals(Color.RED)) {
      redLamp.setFillColor(Color.RED);
      yellowLamp.setFillColor(Color.GRAY);
      greenLamp.setFillColor(Color.GRAY);
   } else if (color.equals(Color.YELLOW)) {
      redLamp.setFillColor(Color.GRAY);
      yellowLamp.setFillColor(Color.YELLOW);
      greenLamp.setFillColor(Color.GRAY);
   } else if (color.equals(Color.GREEN)) {
      redLamp.setFillColor(Color.GRAY);
      yellowLamp.setFillColor(Color.GRAY);
      greenLamp.setFillColor(Color.GREEN);
   }
   state = color;
}

/** Returns the current state of the stoplight */
public Color getState() {
   return state;
}
```

GStoplight

```
   /* Creates a filled circle centered at (x, y) with radius r */
   private GOval createFilledCircle(double x, double y, double r) {
      GOval circle = new GOval(x - r, y - r, 2 * r, 2 * r);
      circle.setFilled(true);
      return circle;
   }
}
```

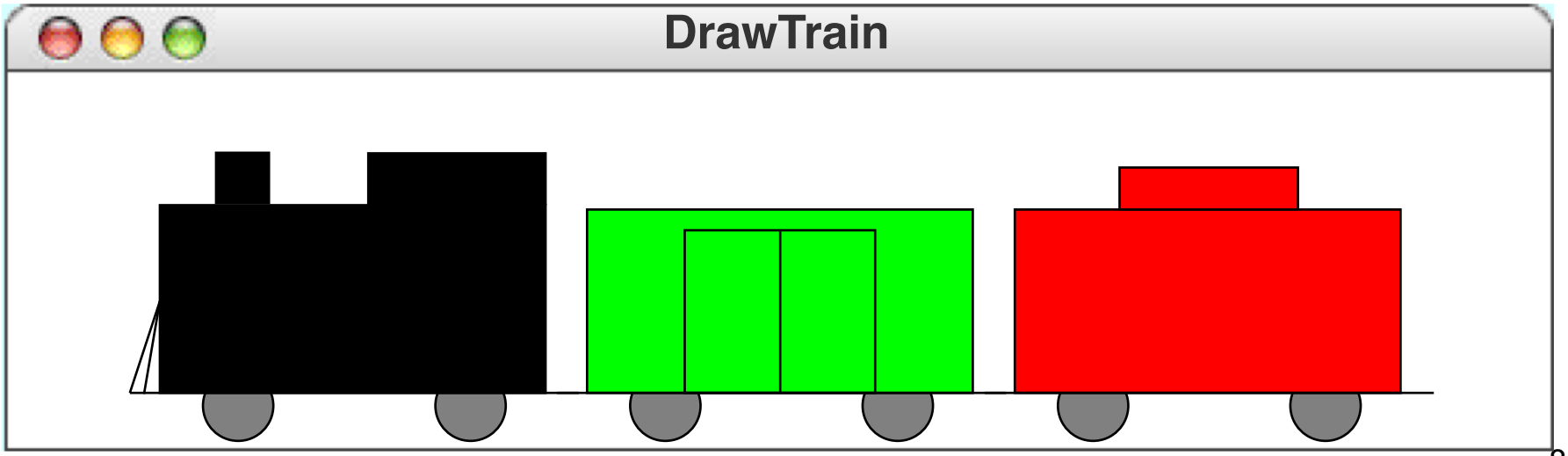
`GCompound` Coordinate System

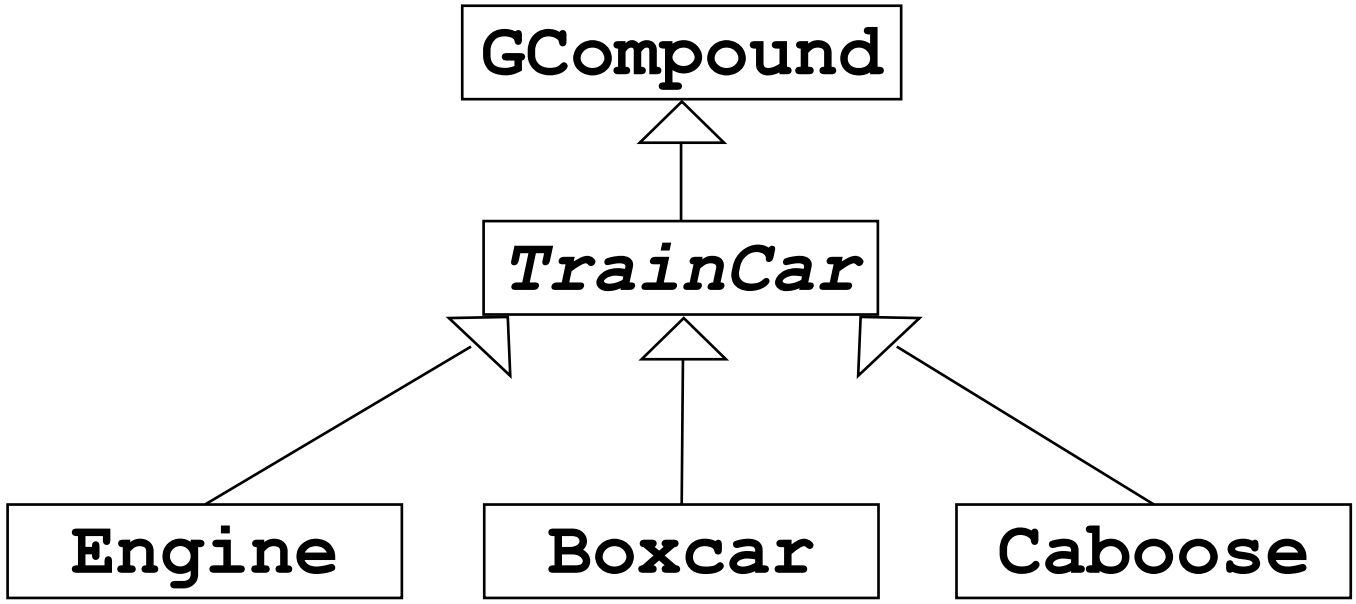
`getCanvasPoint(`*x*`, `***y*`)`**
Converts local point (*x*, *y*) to canvas coordinates

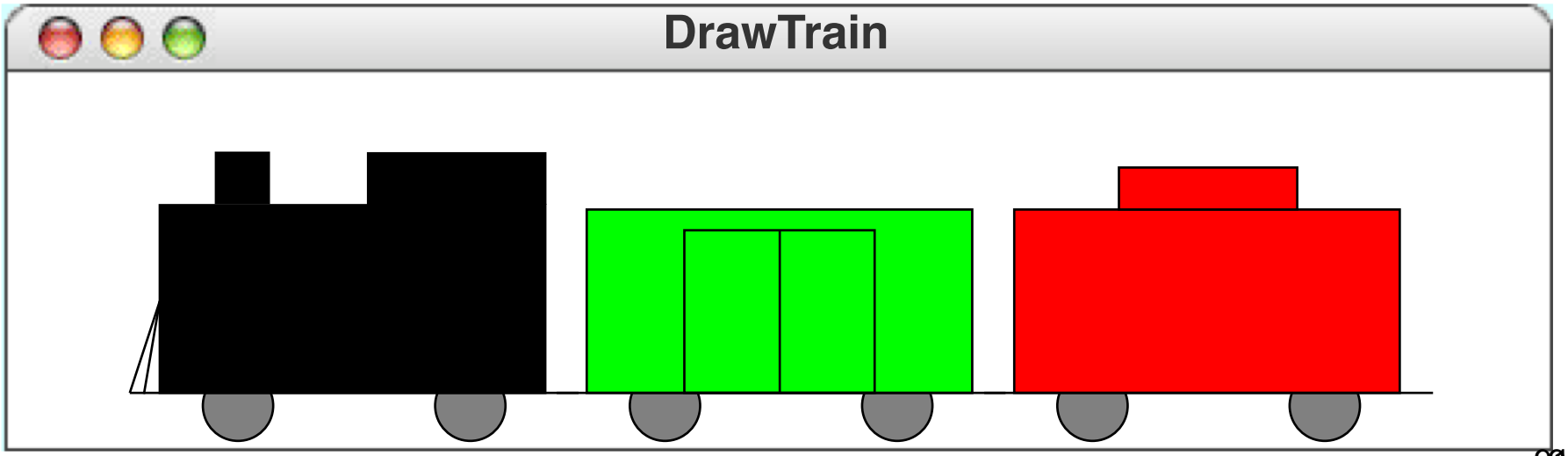
`getLocalPoint(`*x*`, `***y*`)`**
Converts canvas point (*x*, *y*) to local coordinates

Graphical Object Decomposition

- **`GCompound`** supports decomposition in domain of graphical objects
- Before (Chapter 5): decomposed train cars into hierarchy of *methods*
- Now: decompose into hierarchy of *classes*




GCompound
TrainCar
Engine
Boxcar
Caboose


DrawTrain

TrainCar

```
import acm.graphics.*;
import java.awt.*;

/** This abstract class defines what is common to all train cars */
public abstract class TrainCar extends GCompound {

   /* Private constants */
   protected static final double CAR_WIDTH = 75;
   protected static final double CAR_HEIGHT = 36;
   protected static final double CAR_BASELINE = 10;
   protected static final double CONNECTOR = 6;
   protected static final double WHEEL_RADIUS = 8;
   protected static final double WHEEL_INSET = 16;
```

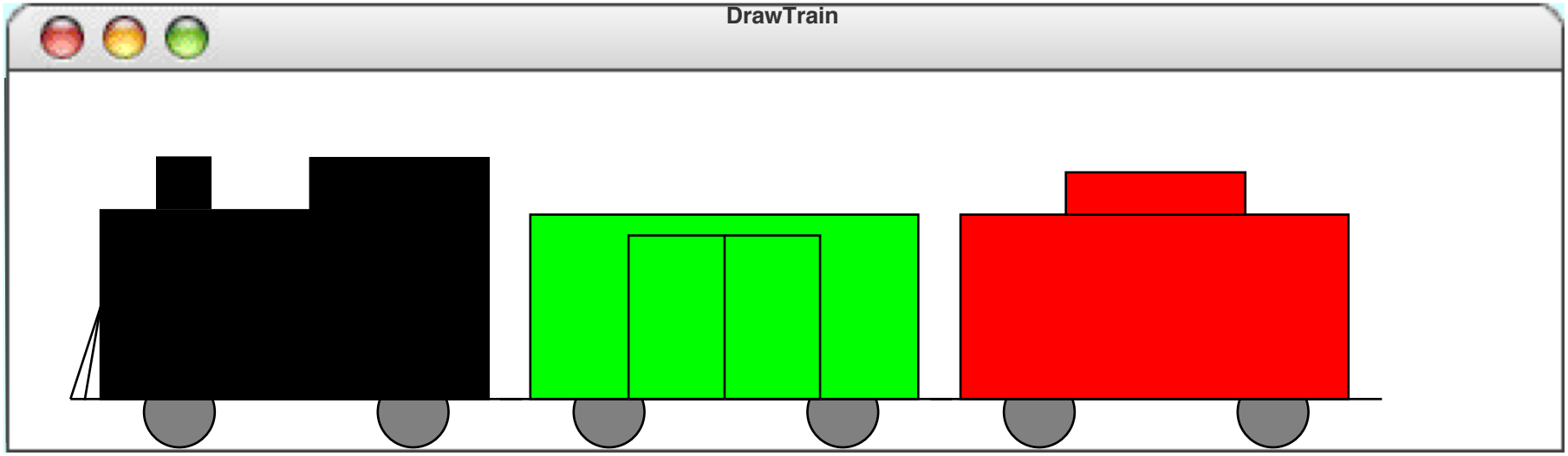
```
/**
 * Creates the frame of the car using the specified color.
 * @param color The color of the new train car
 */
   public TrainCar(Color color) {
      double xLeft = CONNECTOR;
      double yBase = -CAR_BASELINE;
      add(new GLine(0, yBase, CAR_WIDTH + 2 * CONNECTOR, yBase));
      addWheel(xLeft + WHEEL_INSET, -WHEEL_RADIUS);
      addWheel(xLeft + CAR_WIDTH - WHEEL_INSET, -WHEEL_RADIUS);
      double yTop = yBase - CAR_HEIGHT;
      GRect r = new GRect(xLeft, yTop, CAR_WIDTH, CAR_HEIGHT);
      r.setFilled(true);
      r.setFillColor(color);
      add(r);
   }

   /* Adds a wheel centered at (x, y) */
   private void addWheel(double x, double y) {
      GOval wheel = new GOval(x - WHEEL_RADIUS, y - WHEEL_RADIUS,
                              2 * WHEEL_RADIUS, 2 * WHEEL_RADIUS);
      wheel.setFilled(true);
      wheel.setFillColor(Color.GRAY);
      add(wheel);
   }
}
```

Boxcar

```
/**
 * This class represents a boxcar.  Like all TrainCar subclasses,
 * a Boxcar is a graphical object that you can add to a GCanvas.
 */
public class Boxcar extends TrainCar {

   /* Dimensions of the door panels on the boxcar */
   private static final double DOOR_WIDTH = 18;
   private static final double DOOR_HEIGHT = 32;


/**
 * Creates a new boxcar with the specified color.
 * @param color The color of the new boxcar
 */
   public Boxcar(Color color) {
      super(color);
      double xRightDoor = CONNECTOR + CAR_WIDTH / 2;
      double xLeftDoor = xRightDoor - DOOR_WIDTH;
      double yDoor = -CAR_BASELINE - DOOR_HEIGHT;
      add(new GRect(xLeftDoor, yDoor, DOOR_WIDTH, DOOR_HEIGHT));
      add(new GRect(xRightDoor, yDoor, DOOR_WIDTH, DOOR_HEIGHT));
   }

}
```

Nesting Compound Objects

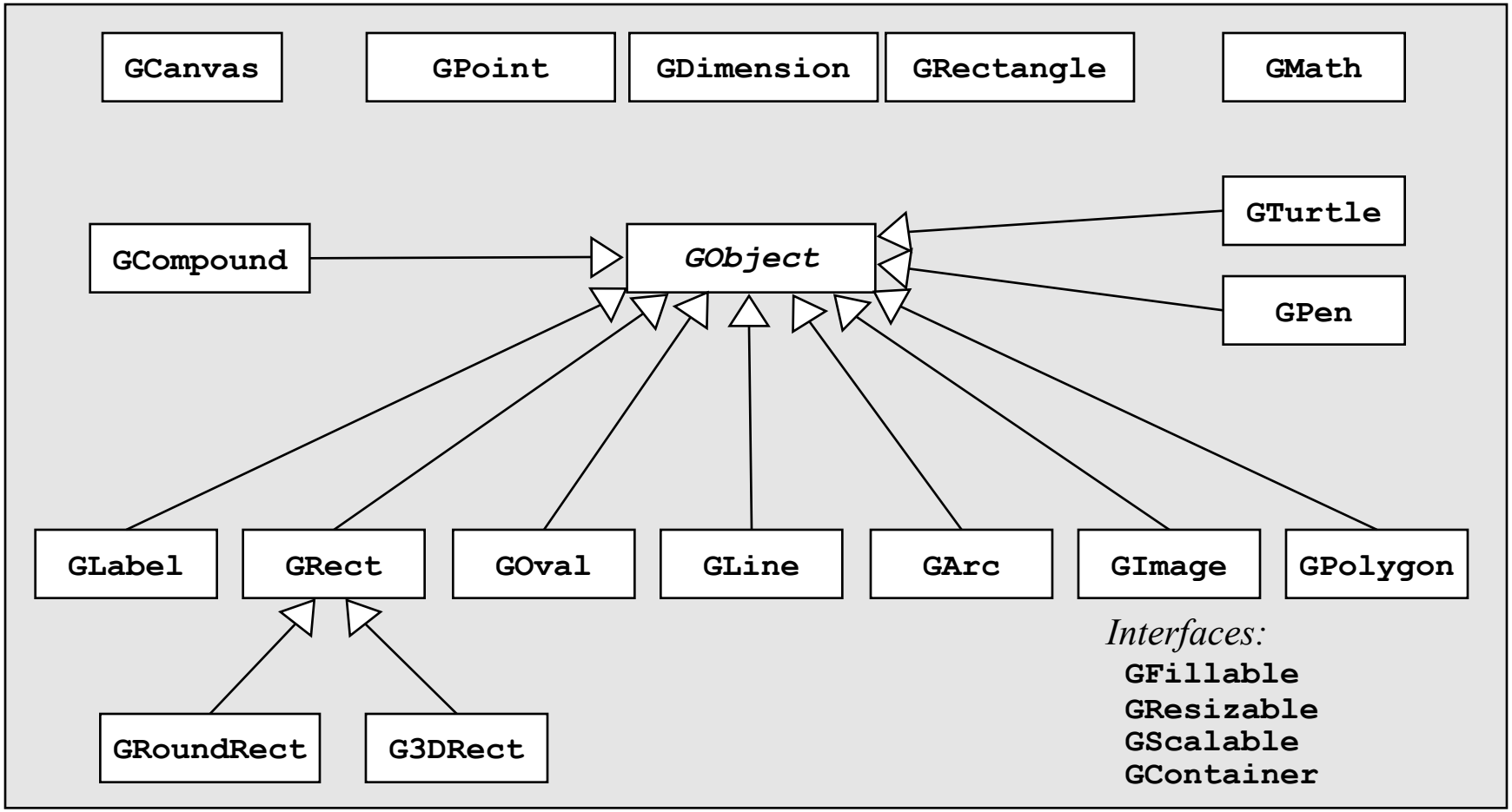
```
Train train = new Train();
train.append(new Engine());
train.append(new Boxcar(Color.GREEN));
train.append(new Caboose());
```



Train

```
/** This class defines a GCompound that represents a train. */
public class Train extends GCompound {

   /**
    * Creates a new train that contains no cars.  Clients can add
    * cars at the end by calling append.
    */
   public Train() {
      /* No operations necessary */
   }

   /**
    * Adds a new car to the end of the train.
    * @param car The new train car
    */
   public void append(TrainCar car) {
      double width = getWidth();
      double x = (width == 0) ? 0 : width - TrainCar.CONNECTOR;
      add(car, x, 0);
   }

}
```

Summary



Object decomposition is important design strategy