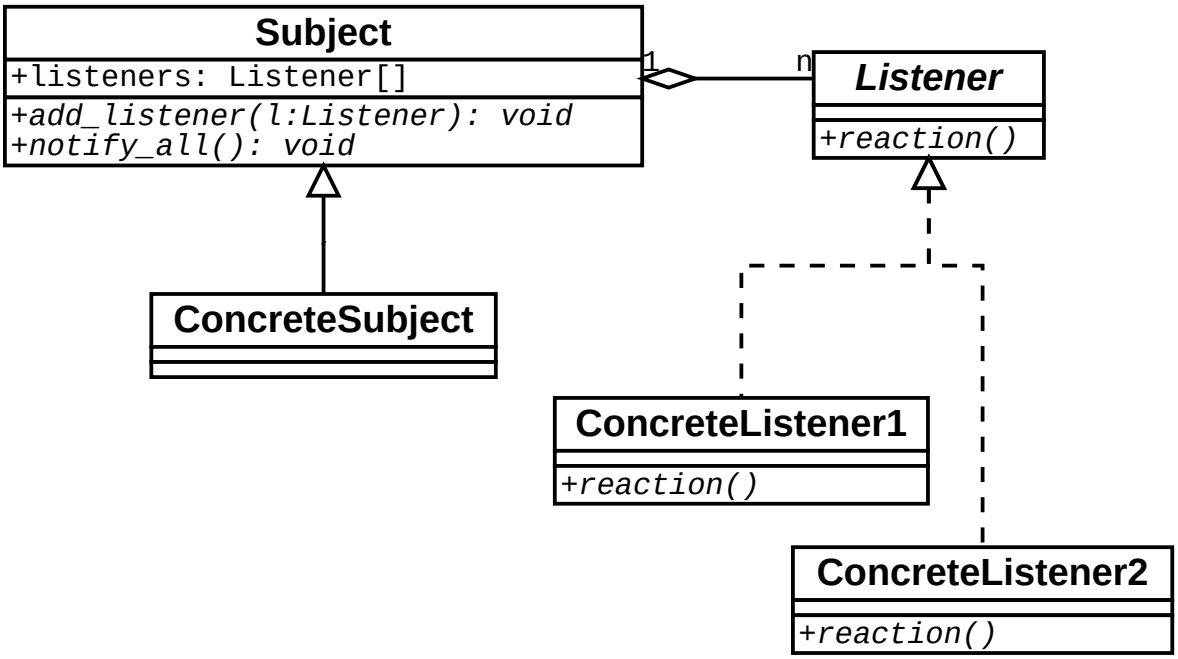
Review

- "Listeners", "Observers", "Callbacks", MVC



Applets

```
public class Myapplet extends Applet {
  public void init() { }
  public void start() { }
  public void stop() { }
  public void destroy() { }
  public void paint(Graphics g) {
    g.drawRect(50, 70, 40, 20);
    g.drawLine(35, 10, 90, 90);
    g.drawString("Some text here", 50, 70);
    g.drawString("Welcome to Java!!", 50, 60 );
    g.setColor(Color.green);
    g.fillOval(80, 120, 20, 50);
    int[] xs = { 50, 30, 80, 90};
    int[] ys = { 70, 30, 40, 10}
    g.drawPolygon(xs, ys, 4);
  }
}
```

Applets

```
public class MouseEvent {
  public Point getPoint() { ... }
}

public class Point {
  public double getX() { ... }
  public double getY() { ... }
}

public interface MouseListener {
  public void mouseClicked(MouseEvent e);
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
}
```

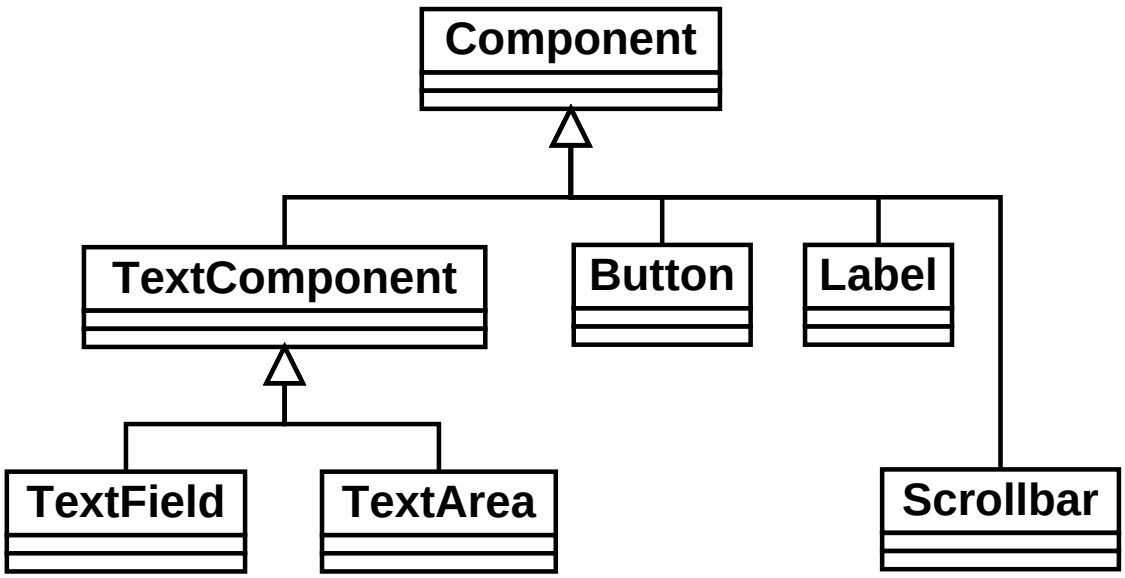
Applets

```
import java.applet.Applet;
import java.awt.*;
public class Myapplet extends Applet {
  Point p;
  public void init()
  {
    p = null;
    addMouseListener(new MyMouseListener(this));
  }
  public void set_point(Point p)
  {
    this.p = p;
  }
  public void paint(Graphics g) {
    if (p != null) {
      Circle b = new Circle((int)p.getX(),
                            (int)p.getY(), 10);
      b.draw(g);
    }
  }
}
```

Applets

```
import java.awt.event.*;
class MyMouseListener implements MouseListener {
  Myapplet applet;
  MyMouseListener(Myapplet a)
  {
    applet = a;
  }
  public void mouseClicked(MouseEvent e)
  {
    Point p = e.getPoint();
    applet.set_point(p);
    applet.repaint();
  }
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
}
```

Applets



- A component is a subject with listeners

```
public interface ActionListener {
  public void actionPerformed(ActionEvent event);
}
```

- A component may have several listeners
- A listener can listen to more than one component

Applets

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends Applet
    implements ActionListener {
  Button b;
  public void init()
  {
    b = new Button(“OK”);
    b.addActionListener(this);
    add(b);
  }
  public void actionPerformed(ActionEvent event)
  {
    Object source = event.getSource();
    // do something
  }
}
```

Applets

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends Applet
    implements ActionListener {
  Button b1, b2;
  public void init()
  {
    b1 = new Button(“OK”);
    b1.addActionListener(this);
    add(b1);
    b2 = new Button(“Cancel”);
    b2.addActionListener(this);
    add(b2);
  }
  public void actionPerformed(ActionEvent event)
  {
    Object source = event.getSource();
    if (source == b1) // do b1’s action
    else if (source == b2) // do b2’s action
  }
}
```

Applets

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends Applet {
  Button b1, b2;
  Button1Handler h1;
  Button2Handler h2;
  public void init()
  {
    b1 = new Button(“OK”);
    h1 = new Button1Handler();
    b1.addActionListener(h1);
    add(b1);
    b2 = new Button(“Cancel”);
    h2 = new Button2Handler();
    b2.addActionListener(h2);
    add(b2);
  }
}
```

Applets

```
public class Button1Handler
implements ActionListener {
  public void actionPerformed(ActionEvent event)
  {
    // do b1's action
  }
}
public class Button2Handler
implements ActionListener {
  public void actionPerformed(ActionEvent event)
  {
    // do b2's action
  }
}
```

GUIs

- Elements
  - Events and Listeners
  - Components (which have listeners)
  - Containers (which have components)
  - Layout managers
- Packages
  - AWT
  - Swing
  - etc.

Exception handling

- Errors and exceptions
- Separation of concerns

```
static int q(float f) {
  if (f < 10)
    System.out.println("Error, f <10, "+f);
  return f * 3 + 1;
}
static void p() {
  float n = Keyboard.readFloat();
  int m = q(n);
  System.out.println(m);
}
```

Exception handling

```
static int q(float f) {
  if (f < 10) {
    System.out.println("Error, f <10, "+f);
    return -1;
  }
  return f * 3 + 1;
}
static void p() {
  float n = Keyboard.readFloat();
  int m = q(n);
  if (m != -1)
    System.out.println(m);
  else
    System.out.println(“Error”);
}
```

Exception handling

```
static int q(float f) {
  if (f < 10) return -1;
  return f * 3 + 1;
}
static void p() {
  float n = Keyboard.readFloat();
  int m = q(n);
  if (m != -1)
    System.out.println(m);
  else
    System.out.println("Error"); //No Error info
}
```

Exception handling

```
static int q(float f) {
  if (f < -5) return -1;
  return f * 3 + 1;
}
static float r(float f) {
  if (f > 15) return -1;
  return f - 2;
}
static void p() {
  float n = Keyboard.readFloat();
  int m = q(r(n));
  if (m != -1)
    System.out.println(m);
  else
    System.out.println("Error"); //No Error info
}
// q(r(13)) = q(11) = 34
// q(r(16)) = q(-1) = -2 // wrong
// q(r(-6)) = q(-8) = -1
```

Exception handling

```
static int q(float f) {
  if (f < -5) return -1;
  return f * 3 + 1;
}
static float r(float f) {
  if (f > 15) return -1;
  return f - 2;
}
static void p() {
  float n = Keyboard.readFloat();
  int partial1 = r(n);
  if (partial1 == -1)
    System.out.println(“Error in r”);
  else {
    int partial2 = q(partial1);
    if (partial2 == -1)
      System.out.println("Error in q"); //No Error
    else
      System.out.println(partial2);
  }
}
```

Exception handling

```
class MyException extends Exception {
  String message;
  MyException(String m)
  {
    message = m
  }
  public String toString()
  {
    return “MyException occurred: “+message;
  }
}
```

Exception handling

```
static int q(float f) throws MyException
{
  if (f < -5)
    throw new MyException(“q: ”+f);
  return f * 3 + 1;
}
static float r(float f) throws MyException
{
  if (f > 15)
    throw new MyException(“r: ”+f);
  return f - 2;
}
```

Exception handling

```
static void p() {
  float n = Keyboard.readFloat();
  try {
    int m = q(r(n));
    System.out.println(m);
  }
  catch (MyException e) {
    System.out.println(e);
  }
}
```

Exception handling

- The *try-catch* statement:

```
try {
  statements;
}
catch (ExceptionSubclass1 e) {
  statements1;
}
catch (ExceptionSubclass2 e) {
  statements2;
}
.
.
.
```

- An exception is generated (raised) with the *throw* statement:

```
throw object;
```

where *object* is an instance of a subclass of `Exception` or `Throwable`

Exception handling

- A try-catch statement executes its default statements in sequence, and
  - If no exception is raised, then computation continues after the catch clauses
  - Otherwise, if an exception is raised, the sequence of statements is interrupted, and execution continues in the catch clause that matches the type of the exception
- After a catch clause finishes, computation continues after the try-catch. This is, the flow of control does not return to the point where the exception occurred.
- An exception which is not caught by a try-catch, is "propagated", i.e. it is raised again