

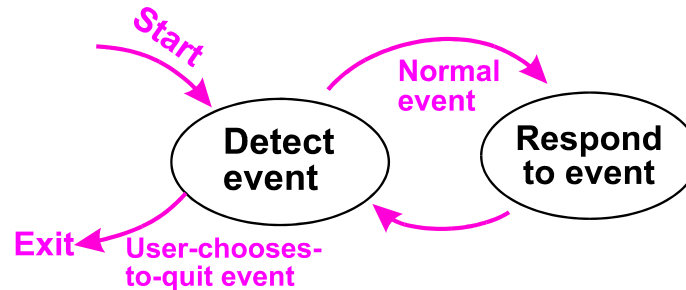
# Topic 9: Graphics programming

1. Events, controls, views
2. Bitmap and vector graphics
3. Polymorphic collections of shapes

## 1. Events, controls, views

- *Browsers* and most other apps are *interactive*: alternate input and output
- *Command-line environments*: URL line in browser, Google prompt, DOS or UNIX prompt
- *GUI events*: Windows/Mac/Xwindows
- *Features*: Icons, menus, dialog boxes, windows, buttons, scrollers, check boxes
- *Common feature*: User generates *events*, e.g., clicks, drags, keystrokes, timeouts
- One form of interaction in browser: hyperlinks
- Another is embedding of event-based JavaScript programs in HTML files

## Event-driven programming



- An *event* is normally the user's input
- Examples of events: keypress, menu choice, mouse click

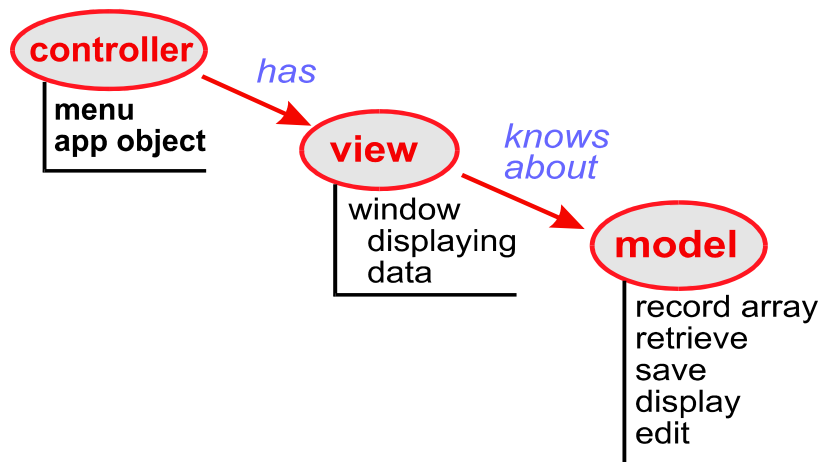
## Application classes

- Used in Windows and Java programming
- A user-interface library defines a general-purpose application class
- Application programmer defines a class that *inherits* from library class, extends its features
- Application programmer may focus on special purpose of application rather than on user-interface details

## Model-view-controller architecture

<b>Kind of class</b>	<b>Example</b>
<i>Model</i>	Array of database records Spreadsheet cells in linked- list grid
<i>View</i>	Window Button
<i>Controller</i>	Menu Instance of application class

## Model, view, and controller classes



## Windows

- A window is a rectangular *view* that is displayed graphically
- Windows are used to display data, including part of a document, graphic, or file
- To enable user to interactively manage what is displayed, controls such as scrollers are part of window objects

## 2. Bitmap and vector graphics

- A drawing may be rendered as a *bitmap*, pixel by pixel, or as instructions (*vector graphics*)
- Bitmap file formats: *TIFF*, *BMP*, *PNG*
- Java graphics packages:
  - *java.awt*: Active Windowing Toolkit
  - *java.io*: input/output classes
- **Graphics classes**: *Font*, *Graphics*, *Picture*, *graphicsEnvironment*, *Color*, *Graphics2D*, *Line2D*, *FontMetrics*, *Pixel*

## Drawing a bitmap

- A drawing may be rendered using *getPixel* and *setColor*
- To draw a red diagonal line:

```
Pixel px = null;
int y = 0;
Picture pic = new Picture();
pic.show();
for (int x = 10; x < this.getHeight-10; x++)
{
    px = this.getPixel(x,y);
    y = 0.6 * x;
    px.setColor(Color.red);
}
pic.repaint();
```

## Creating colors in Java

- *Color* is a class, whose instances may be assigned as the values of *Pixel* objects
- Instances of *Color* have three components: red, green, blue, each in 0..255
- *Color(0,0,0)* is constant *Color.black*;  
*Color(255,255,255)* is *Color.white*

## Vector graphics representation

- *Vector* is as opposed to *bitmap*
- Whereas bitmaps store a representation of each pixel, vector representations store a description with instructions on how to draw object
- *Example*: a line segment or rectangle may be represented by four *ints*
- Vector representations have advantages: more easily edited, shorter
- *Formats*: Illustrator, XML, SVG, CDR

## Java drawing methods

- *drawLine(x1, y1, x2, y2)* draws a line segment from location  $(x1, y1)$  to  $(x2, y2)$  in color set by *setColor()*
- Other shape outline drawing methods: *drawRect*, *drawOval*, *drawArc*, each with parameters  $x, y, w, h$
- *drawArc* also has *startAngle*, *arcAngle* parms
- Methods to draw filled shapes: *fillRect*, *fillOval*, *fillArc*
- *drawPolygon*, *fillPolygon* have parameters *xArray*, *yArray*, and *numPoints*

## The *java.awt.Graphics2D* class

- A class derived from *Graphics*
- Features not possessed by *Graphics* methods:
  - Each shape is an object
  - Set brush width
  - Enable broken lines
  - Rotate, translate, scale, shear
  - Gradient or textured fill
  - Control of effect of overlapping
  - Clipping
  - Curve smoothing

## Drawing text

- *Method* (from *java.awt.Graphics*):  
*drawString(String s, int x, int y)*, where *x, y* specify leftmost point and vertical baseline of string
- Font and color are as previously set by *setFont, setColor*
- *Font* class has *name, style, size* attributes

### 3. Polymorphic collections of shapes

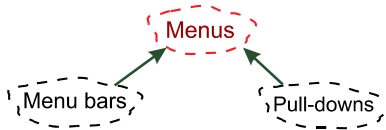
- Inheritance
- Polymorphism
- A collection of graphical objects

## Inheritance and object-oriented design

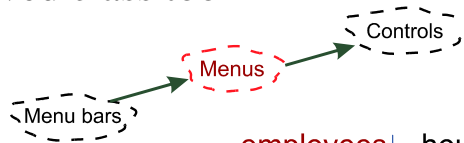
- A class that embodies a concept that is a *subcategory* of some other concept (class) may inherit from that class
- The *is-a* or *kind-of* relationship is an inheritance relationship
- Base class encapsulates a more general category
- Derived class = subclass = descendant
- Base class = superclass = ancestor

## A base class may...

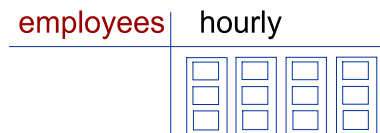
- have multiple derived classes



- be a derived class too



- have no instances



## Polymorphism uses inheritance

- Application programmer who uses application-class library writes a derived class that redefines the base-class event handler
- Each element of a list of base-class pointers points to a derived-class object
- *Example:* processing a mixed payroll of hourly and salaried employees

## A collection of graphical objects

- *Concept*: an expandable collection of descriptions of shapes or other objects, of different classes, each of which calls a *draw* method
- *Java features used*: inheritance, polymorphism
- *Implementation*:
  - Linked list of references to objects of base class, e.g., *Shape*
  - Each object is of a derived class, e.g. *Rectangle*, *Triangle*, *Arrow*, *Oval*
  - Draw is a *virtual method* defined only in the base-class definitions

## References

Cay Horstmann. *Big Java*, 3<sup>rd</sup> Ed. Wiley, 200\_.

Mark Guzdial, Barbara Ericson. *Introduction to Computing and Programming with Java*. Pearson Prentice Hall, 2007.