11 - Introduction to Animation

Computer Science Department
California State University, Sacramento

Overview

- **Frame-based Animation**
- **Timers**
- **Moving Images**
- **Self-drawing and Self-animating Objects**
- **Computing Animated Location**
- **Collision Detection and Response**

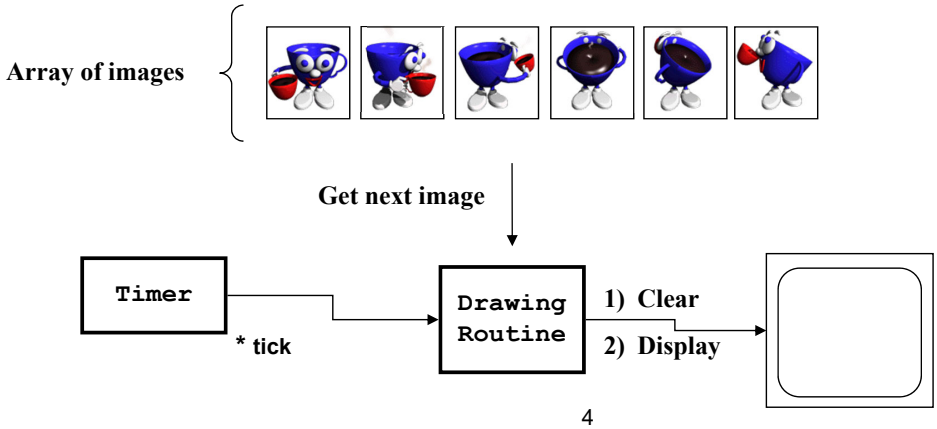
Frame-Based Animation

- Similar images shown in rapid succession imply movement

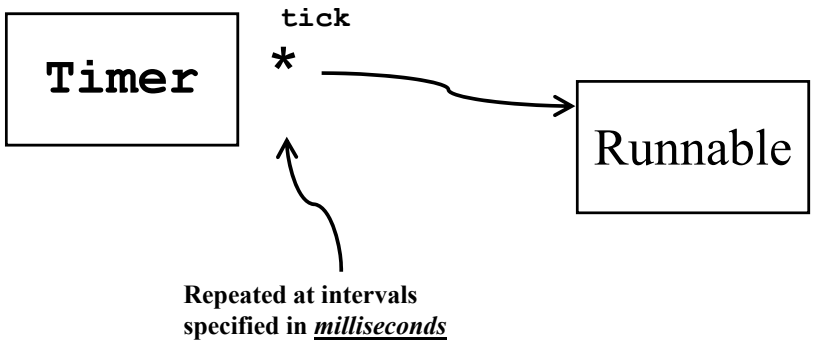
Image credit: Graphic Java: Mastering the JFC (3rd ed.), David Geary

Frame-Based Animation (cont.)

- Basic implementation structure:
  - Read images into an array
  - Use a Timer to invoke repeated "drawing"
  - Each "draw" outputs the "next" image



CN1 `UITimer` Class



CN1 `UITimer` Class (cont)

- Its constructor accepts a runnable to invoke on each tick: `UITimer(Runnable r)`
- It must be linked to a specific form:

```
schedule(int timeMillis, boolean repeat, Form bound)
```

- It is invoked on the CodenameOne main thread rather than on a separate thread.
- It is different from Java Swing **`Timer`** which generates action events in every tick…
- No need to start the timer (**`schedule()`** starts it), use **`cancel()`** to stop it.

CN1 `UITimer` Class (cont)

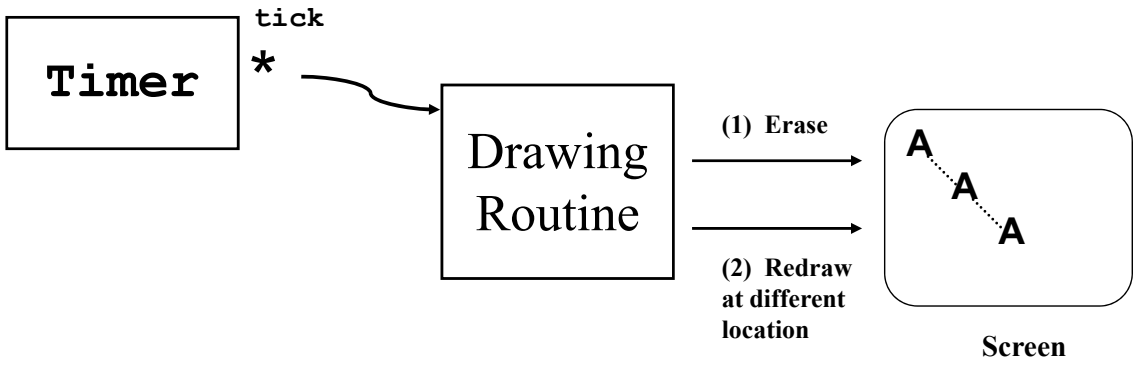
- **Runnable attached to the timer must implement interface *Runnable* (built-in CN1 interface):**

```
interface Runnable
        {
                public void run ();
        }
```


Using the UITimer

```
/** This class creates and binds the Timer to the form and provides a runnable (which is
 * the form itself) for the Timer.  The runnable draws graphical shapes of random sizes at
 * random locations. */
public class TimerGraphics extends Form implements Runnable {
  private TimerGraphicsContainer myContainer;
  public TimerGraphics() {
    // ...code here to initialize the form which uses border layout...
    // create a container on which to do graphics; put it in the center
    myContainer = new TimerGraphicsContainer();
    add(BorderLayout.CENTER, myContainer);
    // create timer and provide a runnable (which is this form)
    UITimer timer = new UITimer(this);
    // make the timer tick every second and bind it to this form
    timer.schedule(1000, true, this);}
    // entered when the Timer ticks
  public void run() {
    myContainer.repaint();}
}
public class TimerGraphicsContainer extends Container{
  public void paint(Graphics g){
    super.paint();
    g.setColor(ColorUtil.BLACK);
    int iShapeX = myRNG.nextInt(getWidth()); // shape location (relative to the
    int iShapeY = myRNG.nextInt(getHeight());// the origin of the container)
    int xSize = myRNG.nextInt (50);
    int ySize = myRNG.nextInt (25);
    // draw a random-sized rounded corner rectangle at a random location
    g.drawRoundRect(getX()+ iShapeX, getY()+ iShapeY,xSize,ySize,20,10);}
}
```

Animation via Image Movement

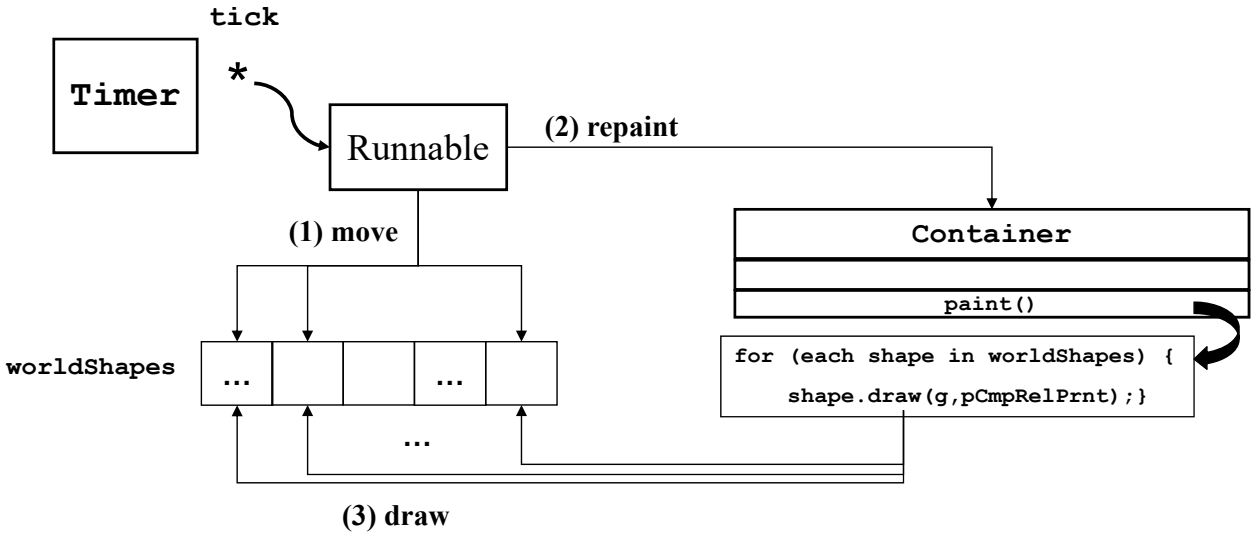


Animation Example

```
/*This time instead of drawing shapes of random sizes at random locations,
 * we will draw the same image (a simple filled shape) that moves on a path.
 * The form is the same as above example except that the tick would happen every 100 ms… */
public class AnimationContainer extends Container {
  private int currentX = 0, currentY = 0 ; // image location (relative to the origin
                                           //of the component)
  private int incX = 3, incY = 3 ;         // amount of movement
  private int size = 20 ;

  // update the image on the container
  public void paint(Graphics g) {
    super.paint (g);

    // draw the image (a simple filled rounded corner rect) at the current location.
    g.setColor(ColorUtil.BLACK);
    g.fillRoundRect(getX()+currentX, getY()+currentY, size, size, 20, 10);

    // update the image position for the next draw
    currentX += incX ;
    currentY += incY ;

    // reverse the movement direction if the image reaches an edge
    if ( (currentX+size >= getWidth()) || (currentX < 0) )
      incX = -incX ;
    if ( (currentY+size >= getHeight()) || (currentY < 0) )
      incY = -incY ;
  }
}
```

"Self-Animating" Objects

- Objects should be responsible for their own drawing and movement



"Self-Animation" Example

```
/** A form containing a collection of "self drawing objects". */
public class SelfDrawerAnimationForm extends Form implements Runnable {
  private SelfAnimationContainer myContainer ;
  public SelfDrawerAnimationFrame() {
    //...code here to initialize the frame with a BorderLayout
    // create a world containing a self-drawing object
    Vector<WorldObject> theWorld = new Vector<WorldObject>();
    theWorld.add( new WorldObject() );
    //create a container on which the world will be drawn
    myContainer = new SelfAnimationContainer(theWorld) ;
    add(BorderLayout.CENTER, myContainer);
    // create a Timer and schedule it
    UITimer timer = new UITimer (this);
    timer.schedule(15, true, this);
  }
  // called for each timer tick: tells object to move itself, then repaints the container
  public void run () {
    Dimension dCmpSize = new Dimension(myContainer.getWidth(),
                                       myContainer.getHeight());
    for (WorldObject obj : theWorld) {
      obj.move(dCmpSize);
    }
    myContainer.repaint();
}
```

"Self-Animation" Example (cont.)

```
/** This class defines an object which knows how to "move" itself, given a container
 *  with boundaries, and knows how to "draw" itself given a Graphics object and container
 *  coordinates relative to its parent.*/
public class WorldObject {
  private int currentX = 0, currentY = 0 ; // the object's current location (relative to
                                           // the origin of the component)
  private int incX = 3, incY = 3 ;         // amount of movement on each move
  private int size = 35 ;                  // object size

  // create the image to be used for this object
  Image theImage = null;
  public WorldObject(){
    try {// you should copy happyFace.png directly under the src directory
      theImage = Image.createImage("/happyFace.png");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
  // move this object within the specified boundaries
  public void move (Dimension dCmpSize) {
    // update the object position
    currentX += incX ;
    currentY += incY ;
    // reverse the next movement direction if the location has reached an edge
    if ( (currentX+size >= dCmpSize.getWidth()) || (currentX < 0) )
      incX = -incX ;
    if ( (currentY+size >= dCmpSize.getHeight()) || (currentY < 0) )
      incY = -incY ;
  }
```

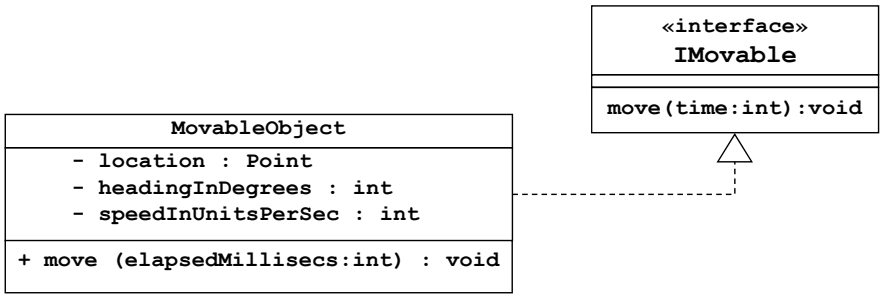
…continues…

"Self-Animation" Example (cont.)

```
  // draw the representation of this object using the received Graphics context
  public void draw(Graphics g, Point pCmpRelPrnt) {
    g.drawImage(theImage, pCmpRelPrnt.getX()+currentX,
                          pCmpRelPrnt.getY()+currentY, size, size);
  }
} //end of WorldObject class
-------------------------------------------------------------------------
/** A container which which redraws its world object(s) each time
 *  the container is repainted.
 */
public class SelfAnimationContainer extends Container {

  private Vector<WorldObject> theWorld ;
  public DisplayPanel (Vector<WorldObject> world) {
    theWorld = world ;
  }
  public void paint(Graphics g) {
    super.paint(g);
    Point pCmpRelPrnt = new Point(getX(),getY());
    for (WorldObject obj : theWorld) {
      obj.draw(g, pCmpRelPrnt) ;
  }
}
```

Computing Animated Location

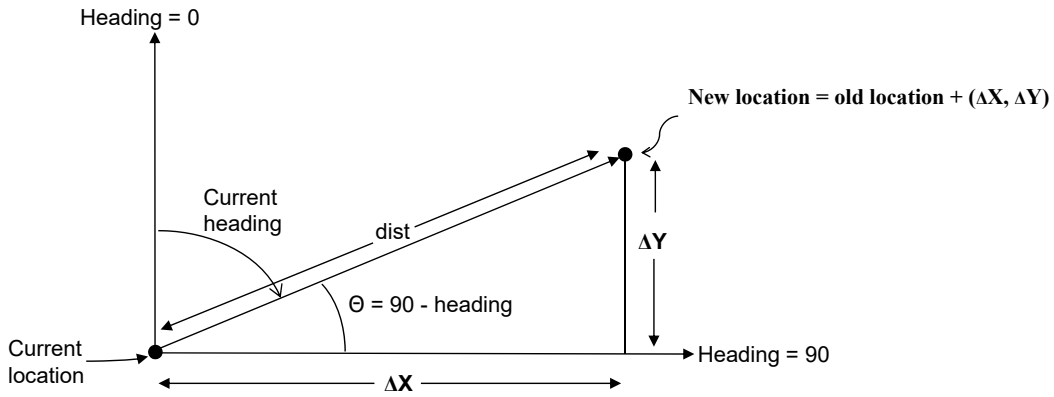
- Consider a "moveable object" defined as:



- Calling **`move()`** instructs the object to update its location, determined by
  - How long it has been moving from its current location
  - Its current heading and speed

Computed Animated Location (cont.)

Computing a new location:



$$dist = rate \times time \quad = \quad \text{speedInUnitsPerSecond} \times \frac{\text{elapsedMilliSecs}}{1000}$$

$$\cos\theta = \frac{\Delta X}{dist};\ \text{so}\quad \Delta X = \cos\theta \times dist.\quad \text{Likewise, } \Delta Y = \sin\theta \times dist$$

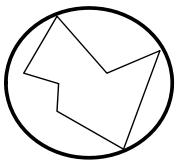
Collision Detection

- **Moving objects require:**
  - ***Detecting collisions***
  - ***Dealing with (responding to) collisions***
- ***Detection == determining overlap***
  - ***Complicated by "shape"***

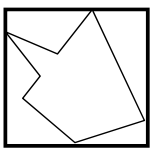
Collision Detection (cont.)

Simplification: "bounding volumes"

- ***Areas* in the 2D case**



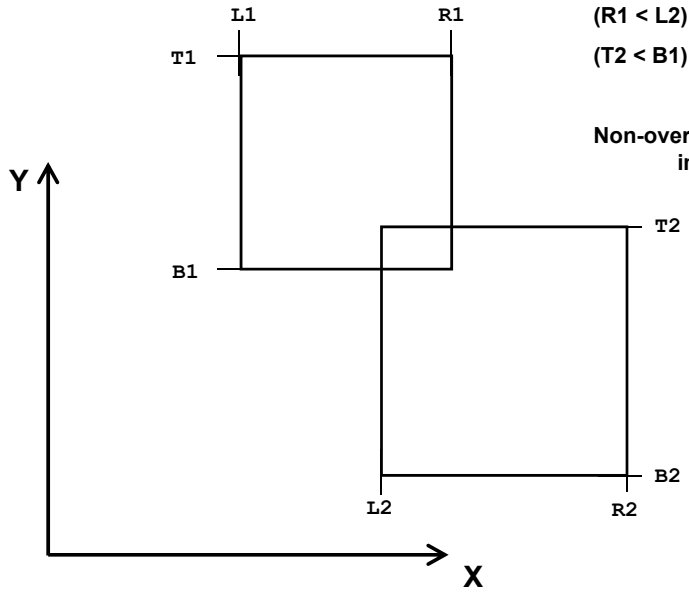
Bounding Circle



Bounding Rectangle

Collision Detection (cont.)

Bounding rectangle collisions



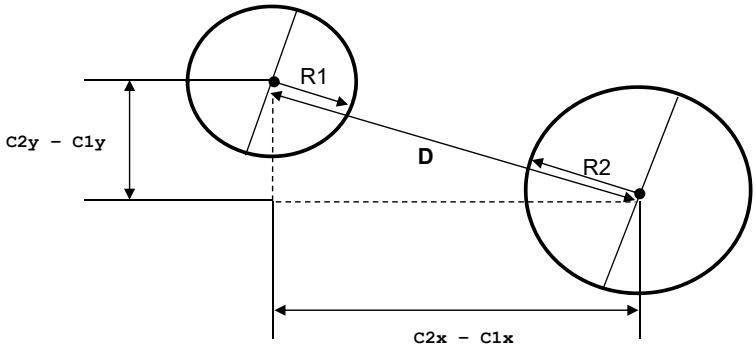
(R1 < L2) OR (L1 > R2) → No Left/Right overlap

(T2 < B1) OR (T1 < B2) → No Top/Bottom overlap

Non-overlap in *either* *(i.e., one or the other or BOTH)*
implies *no collision*

Collision Detection (cont.)

Bounding circle collisions



$D^2 = (C2y - C1y)^2 + (C2x - C1x)^2$

$D \leq (R1+R2)$ → colliding (requires calculating sqrt)

Also, $D^2 \leq (R1+R2)^2$ → colliding (no sqrt)

Collision Response

- **Application-dependent**
  - **Modify heading**
  - **Change appearance**
  - **Delete (explode?)**
  - **Update application state (e.g. "score points")**
  - **Other …**

Collision Response (cont.)

- **`ICollider` interface**

```
public interface ICollider {
   public boolean collidesWith(ICollider otherObject);
   public void handleCollision(ICollider otherObject);
}
```

  - **`collidesWith()`: apply appropriate *detection* algorithm**
  - **`handleCollision()`: apply appropriate *response* algorithm**

Collider Example

```
/** A form with self drawing objects. A Timer instructs the objects to move and
 *  a container to redraw the objects. On collision, an object changes color.  */
public class CollisionForm extends Form implements Runnable {
  private CollisionContainer myContainer;
  private Vector<RoundObject> theWorld ;

  public CollisionForm() {
    // code here to initialize the form...

    theWorld = new Vector<RoundObj>();

    // create a container on which the world objects will be drawn
    myContainer = CollisionContainer(theWorld) ;
    this.add(BorderLayout.CENTER,myContainer);
    // create a Timer to invoke move and repaint operations
    UITimer timer = new UITimer (this);
    timer.schedule(15, true, this);
    // create a world containing objects
    Dimension worldSize = new Dimension(myContainer.getWidth(),
                                        myContainer.getHeight());
    addObjects(worldSize);
  }

  private void addObjects(Dimension worldSize) {
    theWorld.addElement(new RoundObj(Color.red, worldSize));
    theWorld.addElement(new RoundObj(Color.blue, worldSize));
    // ...code here to add additional world objects...
  }
  ...continued...
```


Collider Example (cont.)

```
// this method is entered on each Timer tick; it moves the objects, checks for collisions
// and invokes the collision handler, then repaints the display panel.
public void run () {
  // move all the world objects
  Iterator iter = theWorld.iterator();
  while(iter.hasNext()){
    ((IMovable) iter.next()).move();
  }
  // check if moving caused any collisions
  iter = theWorld.iterator();
  while(iter.hasNext()){
    ICollider curObj = (ICollider)iter.next(); // get a collidable object
    // check if this object collides with any OTHER object
    Iterator iter2 = theWorld.iterator();
    while(iter2.hasNext()){
      ICollider otherObj = (ICollider) iter2.next(); // get a collidable object
       // check for collision
      if(otherObj!=curObj){// make sure it's not the SAME object
        if(curObj.collidesWith(otherObj)){
          curObj.handleCollision(otherObj);
        }
      }
    }
  }
  myContainer.repaint(); // redraw the world
}
} //end class CollisionForm
```

Collider Example (cont.)

```
/** This class defines an object which knows how to "move" and "draw" itself, and
 *  how to determine whether it collides with another object, and provides a method
 *  specifying what to if it is instructed to handle a collision with another object.
 *  (In this case collision changes the color of the object.)  */
public class RoundObj implements IMovable, IDrawable, ICollider {
  private static Random worldRNG = new Random();    // random number generator
  public void move () { ... }
  public void draw(Graphics g, Point pCmpRelPrnt) { ... }
  // Use bounding circles to determine whether this object has collided with another
  public boolean collidesWith(ICollider obj) {
    boolean result = false;
    int thisCenterX = this.xLoc + (objSize/2); // find centers
    int thisCenterY = this.yLoc + (objSize/2);
    int otherCenterX = obj.getX() + (objSize/2);
    int otherCenterY = obj.getY() + (objSize/2);
    // find dist between centers (use square, to avoid taking roots)
    int dx = thisCenterX - otherCenterX;
    int dy = thisCenterY - otherCenterY;
    int distBetweenCentersSqr = (dx*dx + dy*dy);
    // find square of sum of radii
    int thisRadius = objSize/2;
    int otherRadius = objSize/2;
    int radiiSqr = (thisRadius*thisRadius + 2*thisRadius*otherRadius
                                          + otherRadius*otherRadius);
    if (distBetweenCentersSqr <= radiiSqr) { result = true ; }
    return result ;
  }
```

…continues…

Collider Example (cont.)

```
    ...
    // defines this object's response to a collision with otherObject
    public void handleCollision(ICollider otherObject) {
      // change my color by generating three random colors
      color = (ColorUtil.rgb(worldRnd.nextInt(256),
                             worldRnd.nextInt(256),
                             worldRnd.nextInt(256)));
    }
    // ...additional required interface methods here...
  } // end class RoundObject
  ------------------------------
  /** A container which redraws its object(s) each time it is repainted. */
  public class CollisionContainer extends Container {
    Vector<RoundObj> theWorld ;
    public CollisionContainer (Vector<RoundObj> aWorld) {
      theWorld = aWorld ;
    }
    public void paint (Graphics g) {
      super.paint(g);
      Point pCmpRelPrnt = new Point(getX(), getY());
      RoundObj next;
      Iterator iter = theWorld.iterator();
      while(iter.hasNext()){
        next = (RoundObj) iter.next();
        next.draw(g, pCmpRelPrnt);
      }
    }
  }
```