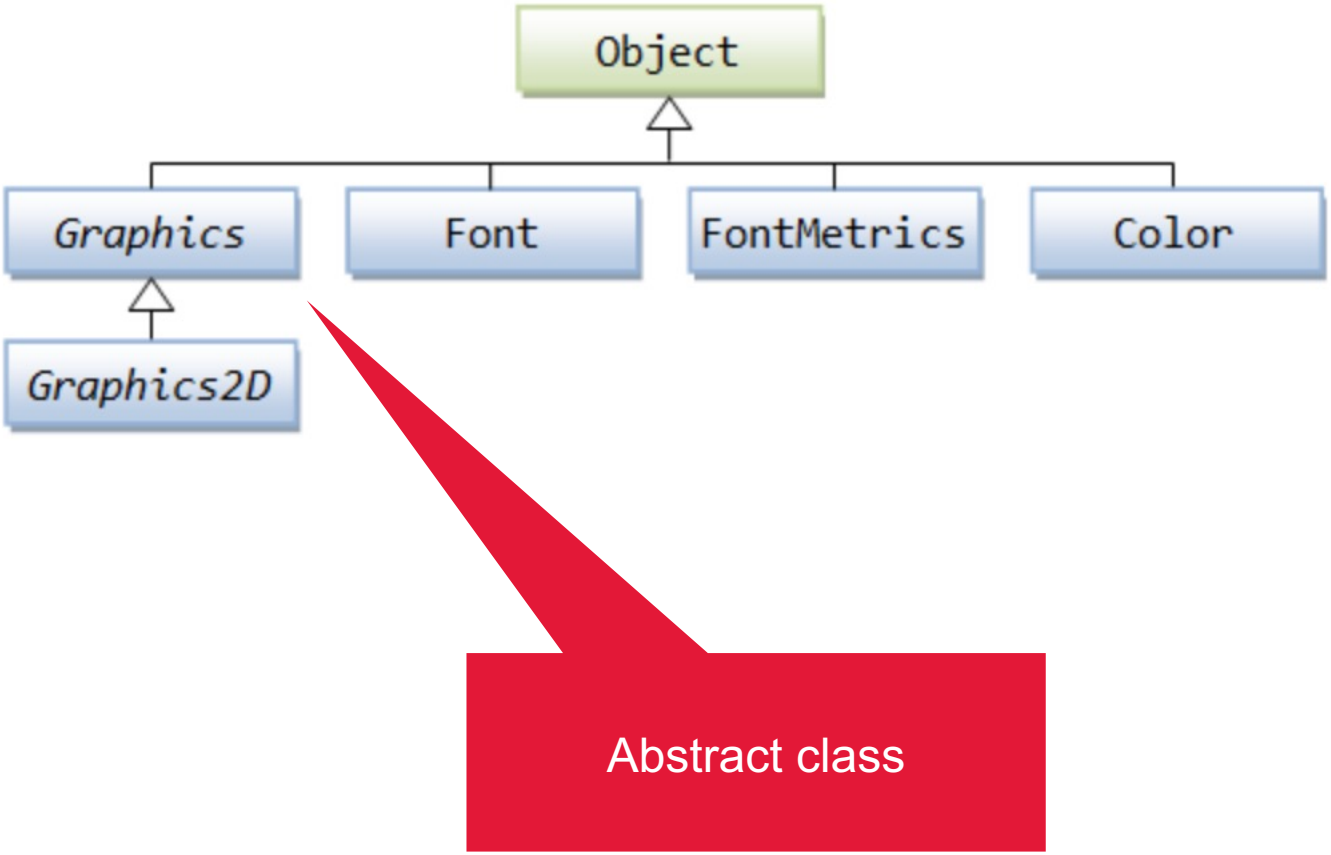
EECS 1720
Building Interactive Systems

Lecture 20 :: Abstract Classes, Threads and Animation

YORK UNIVERSITÉ UNIVERSITY

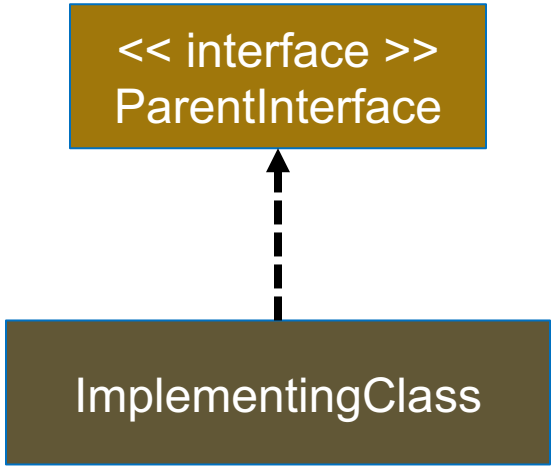
Abstract classes

Graphics/Graphics2D



Interface vs. Abstract Class:

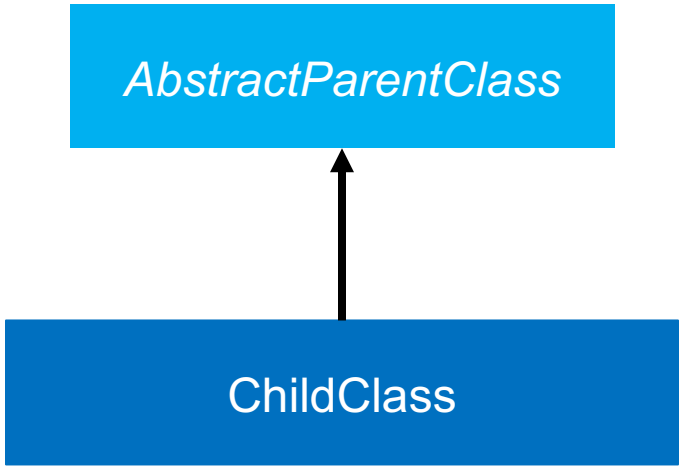
Interfaces



Used when **no method** implementation can be defined (usually used to enforce a set of behaviours to be implemented)

** a class may implement multiple interfaces

Abstract Classes



Used when **some methods can** be defined, while **some cannot** (usually used to enforce both partial structure + partial behaviours)

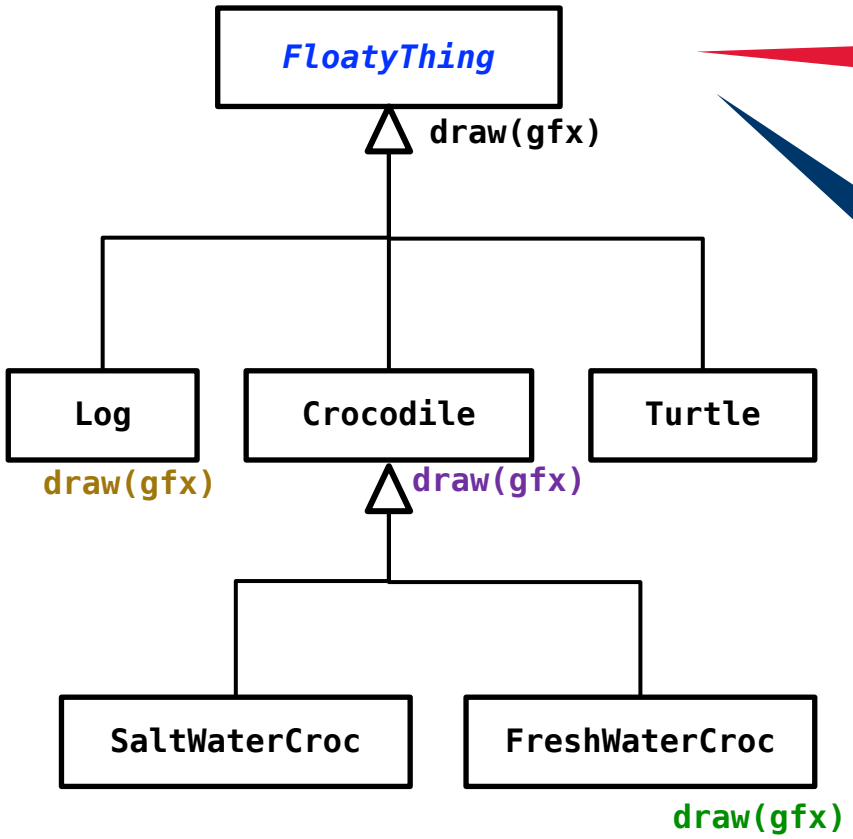
** a class may only inherit from one parent (whether they be abstract or concrete)

Abstract Classes

- an abstract class provides a partial definition of a class
  - the "partial definition" contains everything that is common to all of the subclasses
  - the subclasses complete the definition

  - an abstract class can define fields and methods
    - subclasses *inherit* these
  - an abstract class can define constructors
    - subclasses *must call* these
  - an abstract class can declare abstract methods
    - subclasses *must define* these (unless the subclass is also abstract)

FloatyThing as *abstract*?

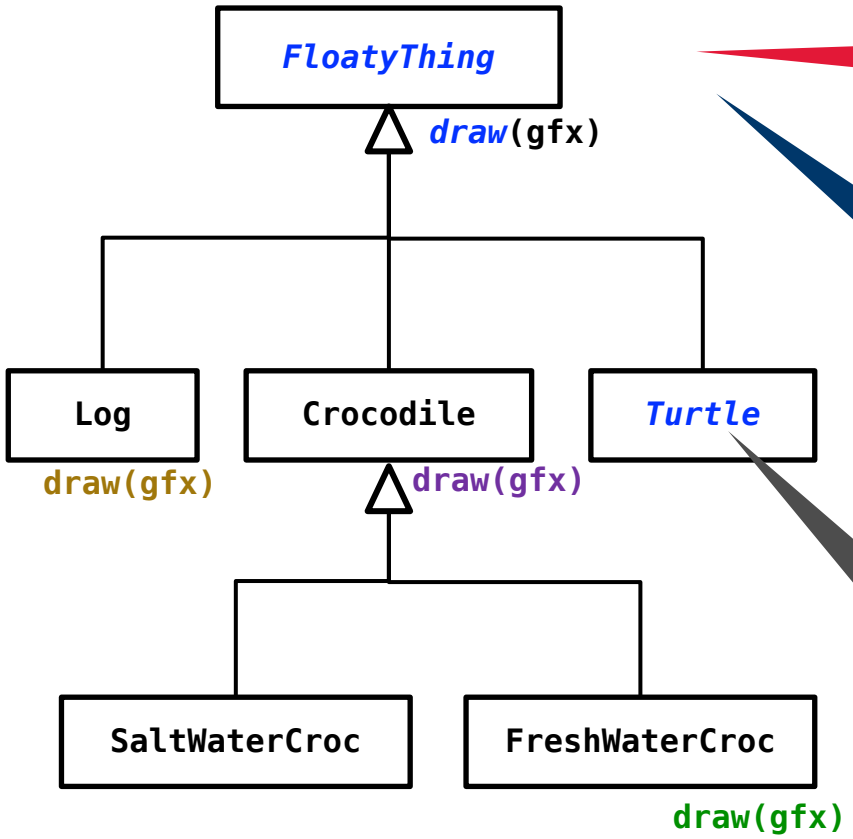


Means that we cannot instantiate FloatyThing objects

Can choose to have an implementation of draw(..) or to declare this as abstract (and force subclasses to implement)

FloatyThing references work same way as for interfaces/parent classes.. abstract methods are declared (so may be invoked if pointing to an instance of a concrete subclass)

FloatyThing as *abstract*?



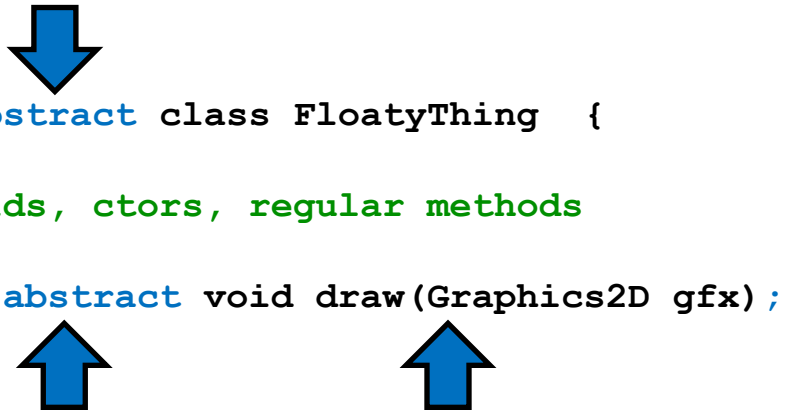
Means that we cannot instantiate FloatyThing objects

Can choose to have an implementation of draw(..) or to declare this as abstract (and force subclasses to implement)

If draw(..) is abstract in parent, and a subclass does NOT implement the method, then it too must be abstract

Abstract Methods

- an abstract base class can declare, *but not define*, zero or more abstract methods



```
public abstract class FloatyThing  {

  // fields, ctors, regular methods

  public abstract void draw(Graphics2D gfx);
}
```

- the base class is saying "all **`FloatyThing's`** can provide a **`void`** method that describes how to draw itself, but only the subclasses know enough to implement the method"
- If a subclass does not implement an abstract method, then it too must be declared as abstract (abstract class + abstract method)

Why have an abstract class (vs. interface)?

- When we want to encode partial state and some methods can have definition, we would use an abstract class.
- This allows us to impose behaviour on the children classes (through the abstract class parent)
  - in lab 6 for example, most of the structure and behaviour of the tetriminoes in a game of tetris are defined by the Block class (an abstract class)
  - if there are any methods that are declared abstract, then they are not given any definition (this is left to the class extending an abstract class)
- Interfaces also impose behaviour (by requiring methods to be defined by the implementing class)
  - typically we prefer using interfaces when it is just a set of behaviours (method signatures) we want to impose, and not class fields and structure (defined fields and methods)

Concrete vs. Abstract classes

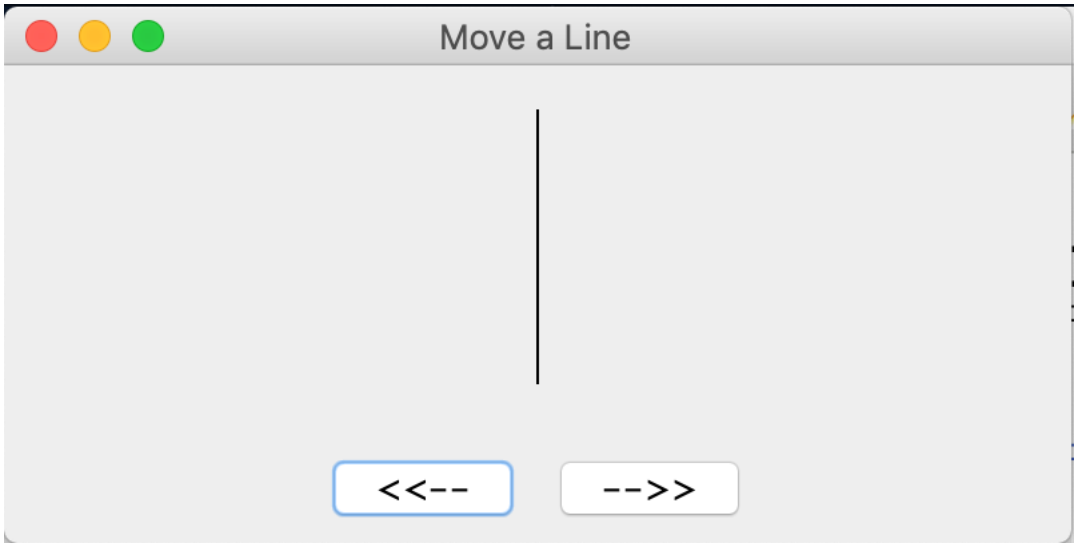
- Abstract
  - has keyword “abstract” in the class header
  - may/may not have some methods also declared “abstract”
  - if class or method declared abstract, then we CANNOT instantiate the class (create objects from it) – though it can be used as a type (reference) to hold/refer to any concrete children instances
- Concrete
  - a class that is not abstract
  - this means it has everything defined (no abstract keyword in class header or methods, and fully defined set of methods)
  - can be instantiated, as there is no danger of attempting to access fields or invoke a method that is not defined (which would otherwise cause a NullPointerException)

ANIMATION

& Timers …

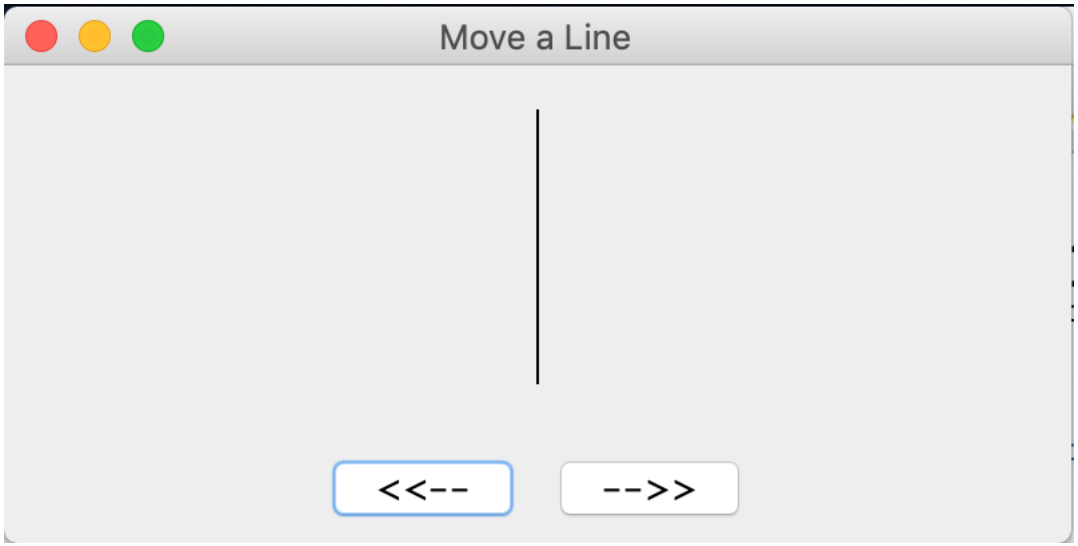
YORK UNIVERSITÉ UNIVERSITY

Previously..



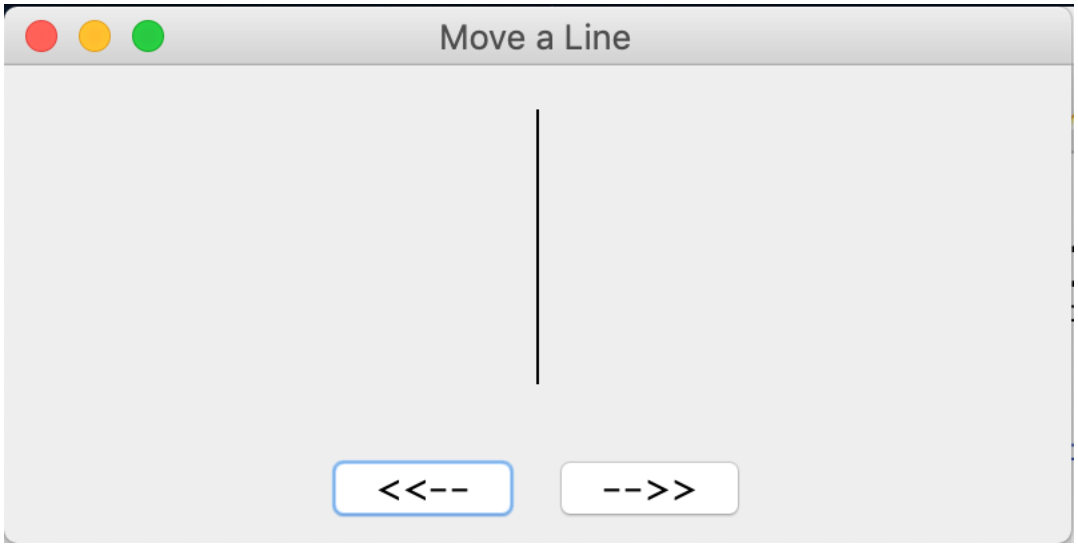
- We looked at an example of moving an object on the screen (e.g. line)
  - Object was a vertical line (starting at x1,y1 and finishing at x2,y2)
  - Moving (left/right) achieved by manual mouse clicks/key presses
    - that spawned mouse/mouse motion events or key events

Previously..



- Handlers (MouseListener, MouseMotionListener and KeyListener) translate interaction events into positional updates
  - i.e. subtract or add to x1, x2

Previously..



- Handlers explicitly invoke **repaint()** method on the ”canvas” component
  - we made a class “DrawCanvas” which was a subclass of JPanel
  - **repaint()** ultimately calls paintComponent(Graphics g) method which was overridden within our DrawCanvas class

So what if we wanted that line to move automatically?

- i.e. without a user intervening with a mouse/key press, or interaction via some GUI control?

- i.e. how would we ANIMATE the line?

Animation

- To achieve animation, you need to:
  - regularly fire off an event that will be caught and handled by an appropriate listener
  - the listener would then update the state of the moving object + call repaint

- Useful: **`javax.swing.Timer`** class
  - constructor:

```
public Timer(int delay, ActionListener listener)
```

What does Timer do?

- Basically fires off an ActionEvent after a given delay
- Then keeps firing off an ActionEvent at regular intervals (separated by the delay)
- These events get routed to the ActionListener we register with the timer object through its constructor

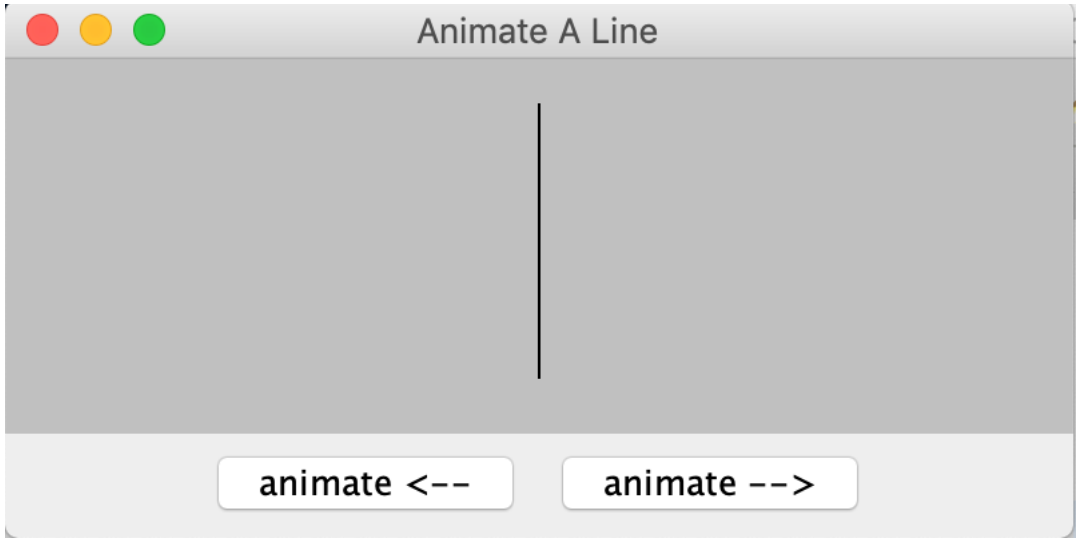
- A Timer can be started and stopped:

```
Timer t = new Timer(…);
t.start();
t.stop();
```

Example

```
// delay in milliseconds
int delay = 500;

// assume this is within an existing class (i.e. inner)
public class updateListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent evt) {
        // ......
    }
};

// Start and run the task at regular delay new
Timer t = Timer(delay, new updateListener());
t.start();
```

Example (Animate a Line)



- Buttons should initiate automatic motion of the line
- Line should animate until it is about to go off canvas (then stop)
- Add spacebar as a key event to stop the animation

First…

- Class fields:
  - Include a Timer instance
  - Include a field to indicate the type of motion (LEFT/RIGHT) and STATIONARY

```
// Define constants for directional movement
private static final int STATIONARY = 0;
private static final int MOVE_LEFT = 1;
private static final int MOVE_RIGHT = 2;


// timer
private Timer animateTimer;
private int animateHow;     // will use above constants
```

Then…

- Inside button listener:
  - Set the value of **animateHow**
  - i.e. if clicking left button, set animateHow to LEFT
  - i.e. if clicking right button, set animateHow to RIGHT
  - Start the timer
- Inside update listener (registered to Timer):
  - Do what we previously did in the button listener – i.e. set new position of x1,x2
  - If position about to go off edge of canvas, stop timer
- Inside key listener:
  - Add a case for the key VK_SPACE (to stop the timer)

Using same framework as SimpleMoveALine

```
// Constructor to set up the GUI components and event handlers

public SimpleAnimateALine(String title) {

    super(title);

    // setup the timer and moveDirection (delay in milliseconds)
    int delay = 100;
    this.animateTimer = new Timer(delay, new UpdateListener());
    this.animateHow = SimpleAnimateALine.STATIONARY;

    // Set up a panel for the buttons
    JPanel btnPanel = new JPanel(new FlowLayout());
    this.btnLeft = new JButton("animate <--");
    this.btnRight = new JButton("animate -->");
    ButtonListener actionHandler = new ButtonListener();

    // remaining constructor unchanged from
    // SimpleMoveALine example in previous lecture

    // ...
```

ButtonListener (modified)

```
private class ButtonListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {

        JButton clicked = (JButton) e.getSource();

        if (clicked == btnLeft) {
        // no longer need to update position here
        // set animateHow to MOVE_LEFT
        animateHow = SimpleAnimateALine.MOVE_LEFT;
        animateTimer.start();
        }

        if (clicked == btnRight) {
        // set animateHow to MOVE_RIGHT
        animateHow = SimpleAnimateALine.MOVE_RIGHT;
        animateTimer.start();
        }
        canvas.repaint();
        requestFocus();
    }
}
```

Timer's listener:

```
private class UpdateListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        // dont really need to check source of event

        switch (animateHow) {
            case SimpleAnimateALine.MOVE_LEFT:
            System.out.println("moving left");
            x1 -= 10;
            x2 -= 10;
            break;
        case SimpleAnimateALine.MOVE_RIGHT:
            System.out.println("moving right");
            x1 += 10;
            x2 += 10;
            break;
        default:
        // do nothing if anything other that MOVE_LEFT or MOVE_RIGHT
        }
        canvas.repaint();
        requestFocus();
    }
}
```

Modifying Timer's listener (to detect screen edge)

```
// inside actionPerformed's switch

case SimpleAnimateALine.MOVE_LEFT:
    System.out.println("moving left");
    x1 -= 10;
    x2 -= 10;
    if ((x1-10) < 0 ) {
        animateHow = SimpleAnimateALine.STATIONARY;
        animateTimer.stop();
        System.out.println("stopped");
    }
    break;

case SimpleAnimateALine.MOVE_RIGHT:
    System.out.println("moving right");
    x1 += 10;
    x2 += 10;
    if ((x1+10) > SimpleAnimateALine.CANVAS_WIDTH ) {
        animateHow = SimpleAnimateALine.STATIONARY;
        animateTimer.stop();
        System.out.println("stopped");
    }
    break;
// ...
```

YORK UNIVERSITÉ UNIVERSITY

Key Listener (modified)

```
@Override
public void keyPressed(KeyEvent e) {

    switch(e.getKeyCode()) {
    case KeyEvent.VK_LEFT:
        x1 -= 10;
        x2 -= 10;
        canvas.repaint();
        break;
    case KeyEvent.VK_RIGHT:
        x1 += 10;
        x2 += 10;
        canvas.repaint();
        break;
    case KeyEvent.VK_SPACE:
        if (animateTimer.isRunning()) {
            animateHow = SimpleAnimateALine.STATIONARY;
            animateTimer.stop();
            System.out.println("stopped");
        }
        break;
    }
}
```

THREADS

So, what is a Timer ??

- Timer is actually a **thread**

- Normally, a program runs a single thread (main)
  - Simply speaking, a thread is a process running in memory
  - Typical applications (that have a main method), run as a single process in memory
  - They run sequentially (from start to finish, from the main method)

Threaded programs

- Java supports *single-thread* as well as *multi-thread* programs

- A *single-thread* program has
  - a single entry point (the main() method), and
  - a single exit point (e.g. main() finishing or System.exit(0))

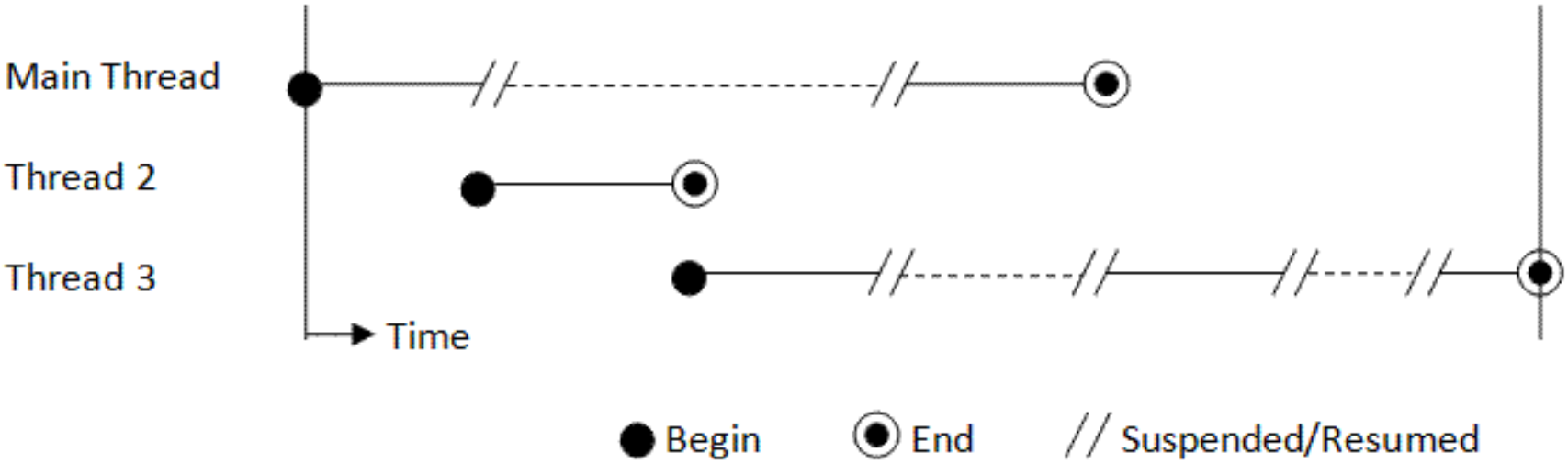
- A *multi-thread* program has
  - an initial entry point (the main() method), followed by
  - many entry and exit points, which are run concurrently with the main().

- "*concurrency*" refers to doing multiple tasks at the same time

A *thread*, also called a *lightweight process*, is a single sequential flow of programming operations, with a definite beginning and an end. During the lifetime of the thread, there is only a single point of execution.

A thread by itself is not a program because it cannot run on its own. Instead, it runs within a program.

Example: a program with 3 threads running under a single CPU:



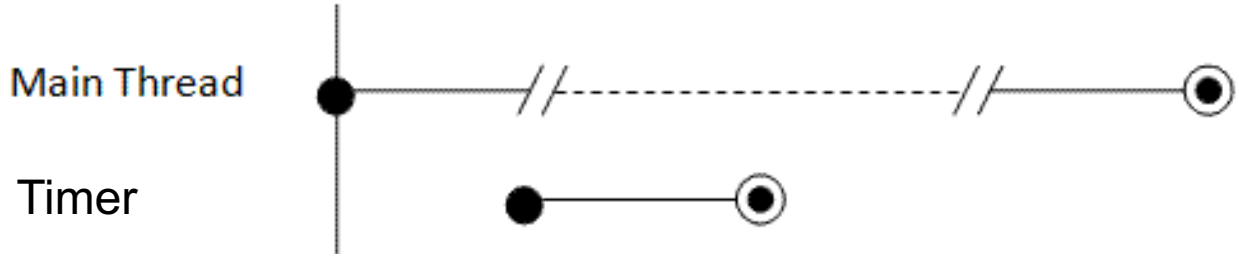
https://www3.ntu.edu.sg/home/ehchua/programming/java/J5e_multithreading.html

How do these run on CPU?

- CPU typically shares its resources with the threads
- Each thread shares memory and resources with the application
- Each thread is allocated time on the CPU (time slicing), and it runs.. Then is paused, while other threads are run

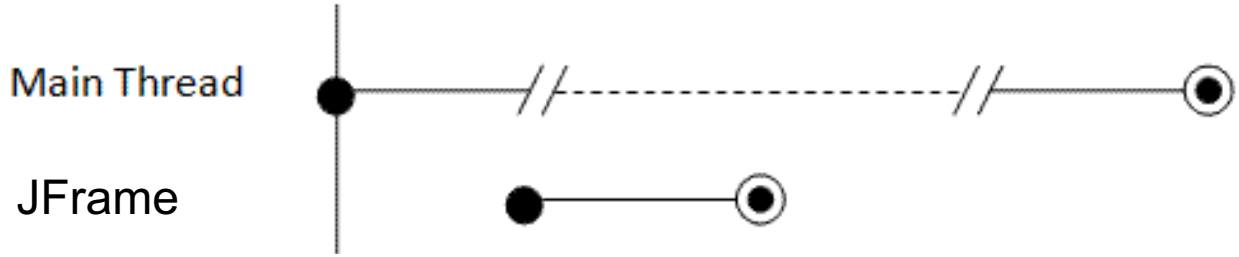
- This happens very quickly… and it appears as though all these threads are running in parallel (i.e. running concurrently)

Timer as a thread



- When Timer is created, it uses a thread that runs in the background
- Its execution may be paused (stop), and restarted (start)
- We can check if it is in a running state
- And a timer specifically generates an ActionEvent at regular intervals (with a specified delay in between)

JFrame has a thread



- When a JFrame is created, it is usually running the window in a thread…
- This is why the window can keep operating while other applications and processes are also operating …
- We can re-run our code (that uses a JFrame), and it will run a second window… both windows run “concurrently”
- Each window monitors for events (window/gui)

- A typical Java program runs in a single process, and is not interested in multiple processes.
- Within the process, it often uses multiple threads to to run multiple tasks concurrently.

- A standalone Java application starts with a single thread (called *main thread*) associated with the main() method.
- This *main thread* can then start new user threads.

How to create a new thread:

Method 1:

- Extend a subclass from the superclass **Thread**
  - override the **run()** method to specify the running behavior of the thread.
  - Create an instance and invoke the **start()** method, which will call-back the **run()** on a new thread

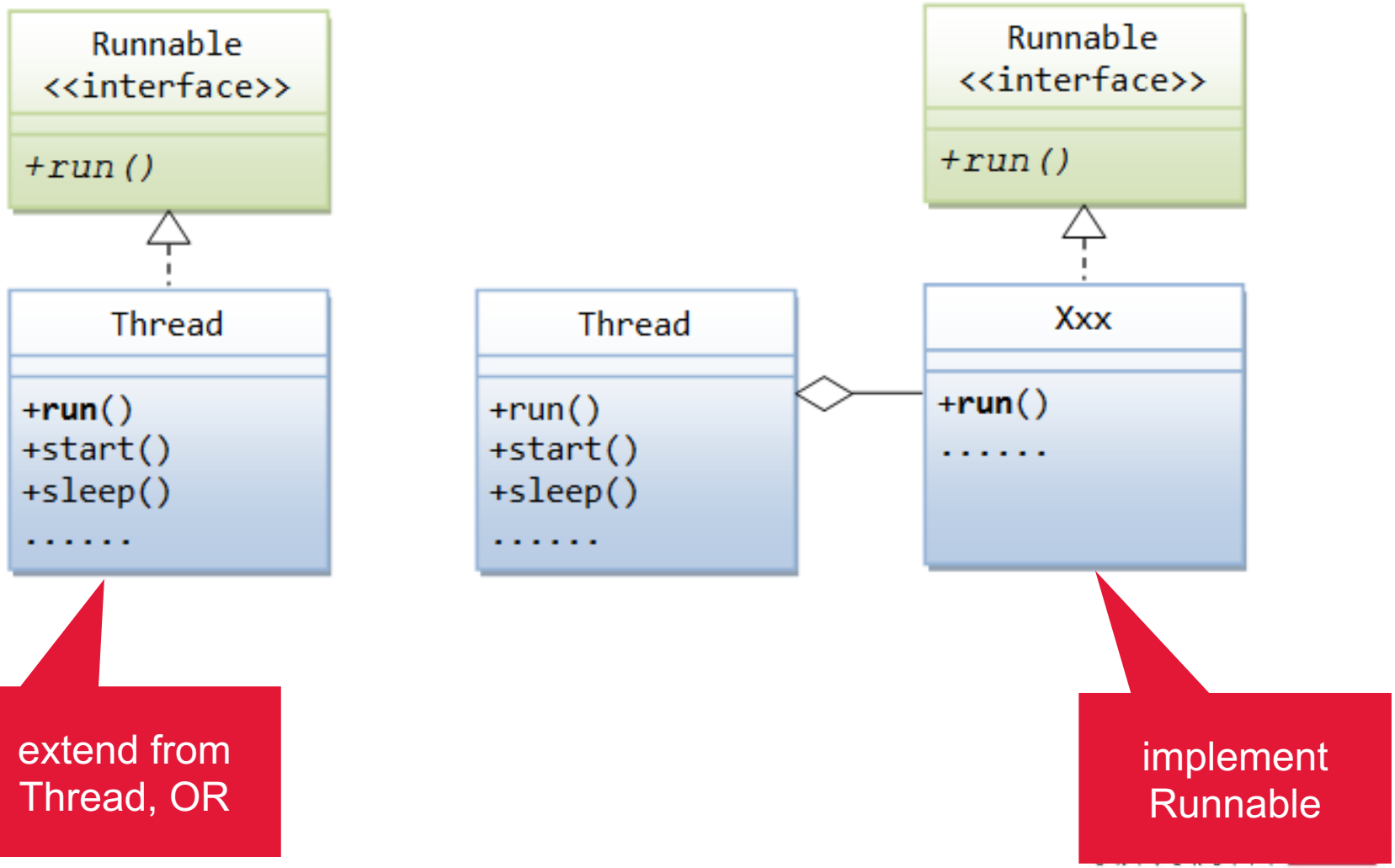
We will focus on this approach

How to create a new thread:

Method 2:

- Create a class that implements the **Runnable** interface
  - provide the implementation of the abstract method run() to specify the running behavior of the thread.
  - Construct a new Thread instance using the constructor with a Runnable object
    - A Thread can be linked to a Runnable object at creation
  - invoke the start() method, which will call back run() on a new thread.

Thread & Runnable



Method 1:

```
class MyThread extends Thread {

    // constructors, other variables and methods ......


    // override the run() method
    @Override
    public void run() {

        // Thread's running behavior

    }

}
```

Method 1:

```
public class Client {

    public static void main(String[] args) {
        // ......

        // Start a new thread
        MyThread t1 = new MyThread();
        t1.start();                          // calls run()

        // ......

        // Start another thread
        (new MyThread()).start();

        // ......
    }

}
```

YORK UNIVERSITÉ UNIVERSITY

Example:

```
public class MyThread extends Thread {

    private String name;

    public MyThread(String name) {
        // constructor
        this.name = name;
    }

    // Override the run() method
    @Override
    public void run() {
        for (int i = 1; i <= 5; ++i) {
            System.out.println(name + ": " + i);
            yield();
        }
    }
}
```

Each MyThread will attempt to count to 5, but yields after each iteration of the loop

Example

```
public class TestMyThread {

    public static void main(String[] args) {

        // create 3 threads and start them
        Thread[] threads = {
                new MyThread("Thread 1"),
                new MyThread("Thread 2"),
                new MyThread("Thread 3")   };

        for (Thread t : threads) {
            t.start();
        }

    }

}
```

Output

Thread 1: 1
Thread 3: 1
Thread 1: 2
Thread 2: 1
Thread 1: 3
Thread 3: 2
Thread 2: 2
Thread 3: 3
Thread 1: 4
Thread 1: 5
Thread 3: 4
Thread 3: 5
Thread 2: 3
Thread 2: 4
Thread 2: 5

Output not always the same

Thread class methods

// start and define run behaviour of a thread
public void **start()**
public void **run()**

// pause thread for period of time
public static **sleep(long millis)** throws InterruptedException
public static **sleep(long millis, int nanos)** throws InterruptedException

// interrupt execution of a thread, or allow it to be interrupted (if other threads)
public void **interrupt()**
public static void **yield()**

// check if thread is alive, or change its priority (so it gets CPU more)
public boolean **isAlive()**
public void **setPriority(int p)**

- **A deeper discussion and use of threads is <u>out of the scope</u> of this course..**

- Try to use **Timer** where possible to do what you need ☺

- Can also use multiple timers if you like (instantiate more than one with different delays and connected to different handlers)

In general.. you can do a lot with Timer …

And may not need to make your own special threads

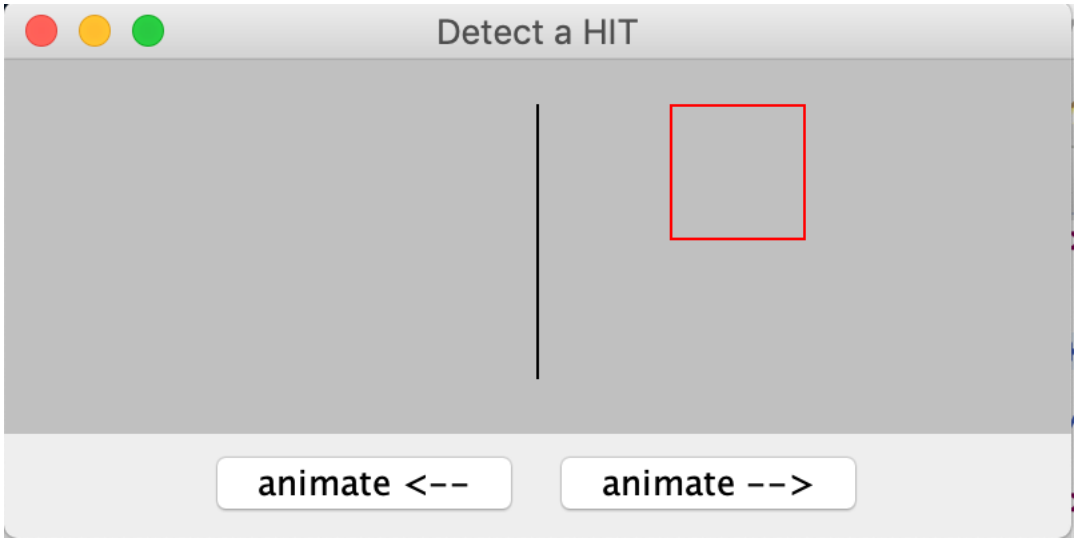
Example: Checking for collisions (common test within a game)

- If a collision happens.. can check if game should end or continue (then can branch to appropriate code)

Hit test (collision test)

- If using geometry, can be done with bounding box (bounds) of an object…. Rectangle
- Can test if a point, line or other rectangle “intersects” with a particular rectangle of interest

Example



- Allow line to animate (move left or right)
- Detect when it collides with the red rectangle (obstacle or enemy)

Example

```
// CLASS FIELD (add other object to collide with)
private Rectangle colliderBoundingBox;


// Inside constructor (create this Rectangle)
this.colliderBoundingBox = new
java.awt.Rectangle(x1+CANVAS_WIDTH/8, y1, CANVAS_WIDTH/8,
CANVAS_WIDTH/8);
```

DrawCanvas (modified)

```
private class DrawCanvas extends JPanel {

    @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        setBackground(CANVAS_BACKGROUND);

        g.setColor(LINE_COLOR);
        g.drawLine(x1, y1, x2, y2); // Draw the line

        g.setColor(Color.RED);

        g.drawRect(    (int)colliderBoundingBox.getX(),
                       (int)colliderBoundingBox.getY(),
                       (int)colliderBoundingBox.getWidth(),
                       (int)colliderBoundingBox.getHeight(). );

    }
}
```

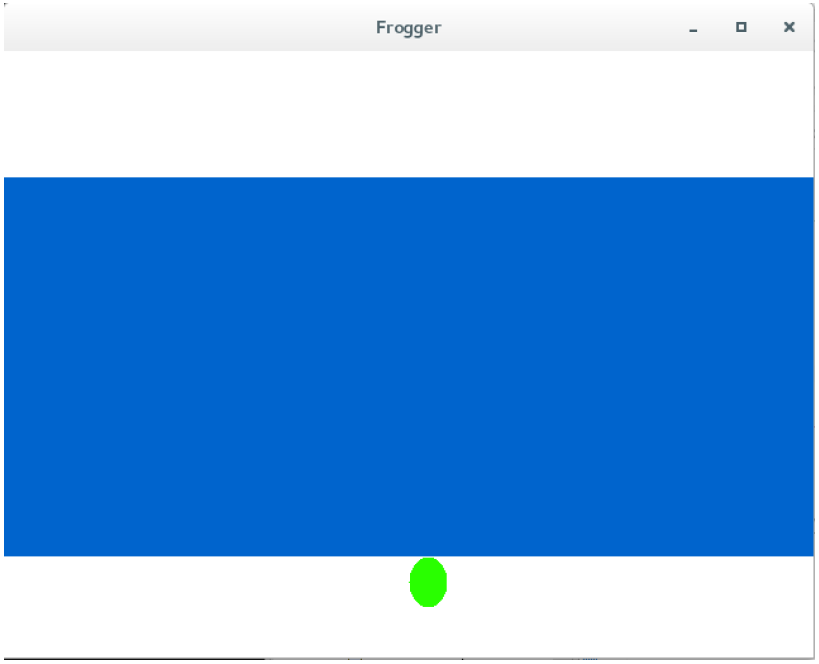
UpdateListener (modified)

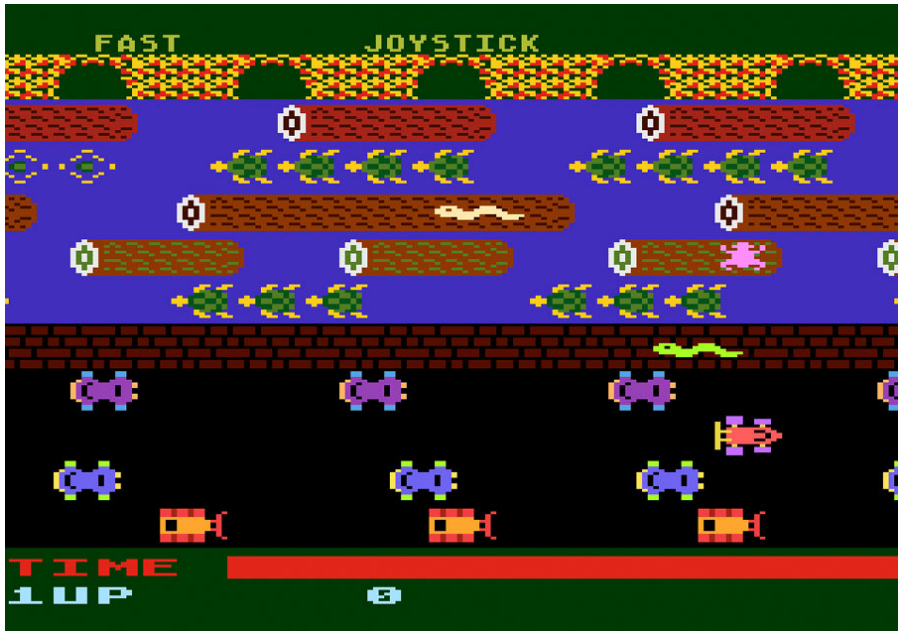
```
private class UpdateListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        switch (animateHow) {
          // not shown (as before)
        }

        if (collided()) {
            System.out.println("HIT DETECTED !!");
            System.out.println("Game Over Dude");
            animateHow = SimpleAnimateAndCollide.STATIONARY;
            animateTimer.stop();
        }

        canvas.repaint();
        requestFocus();
    }
}
```

Hit Test: collided()

```
public boolean collided() {

    boolean hit = false;

    if ( colliderBoundingBox.contains(new Point.Double(x1,y1)) ||
          colliderBoundingBox.contains(new Point.Double(x2,y2))    )
    {

        hit = true;
    }

    return hit;
}
```

Frogger revisited?

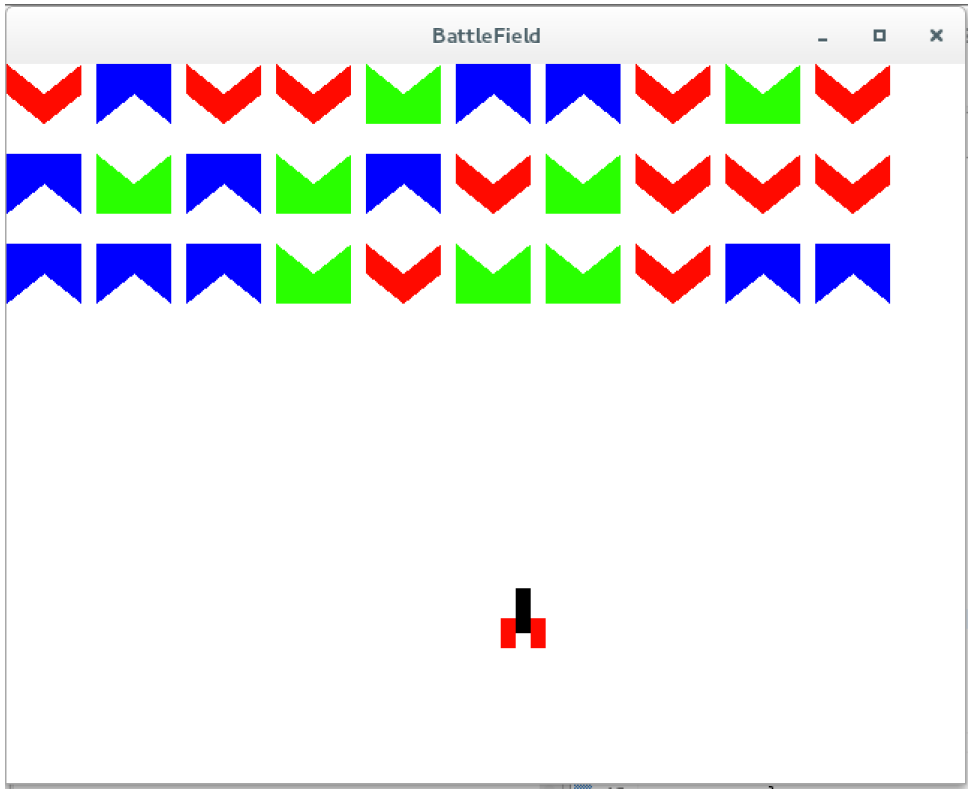




Now you should be able to
Test if bounds (frog) intersects
with bounds of water

And animate other objects
(logs, cars, turtles)

Space invaders?



Move hero; Move aliens
Fire bullet from random alien
Animate bullet (test if hits hero)

Fire bullet from hero
Animate bullet (test if hits aliens)
HINT (Single Timer to control all)