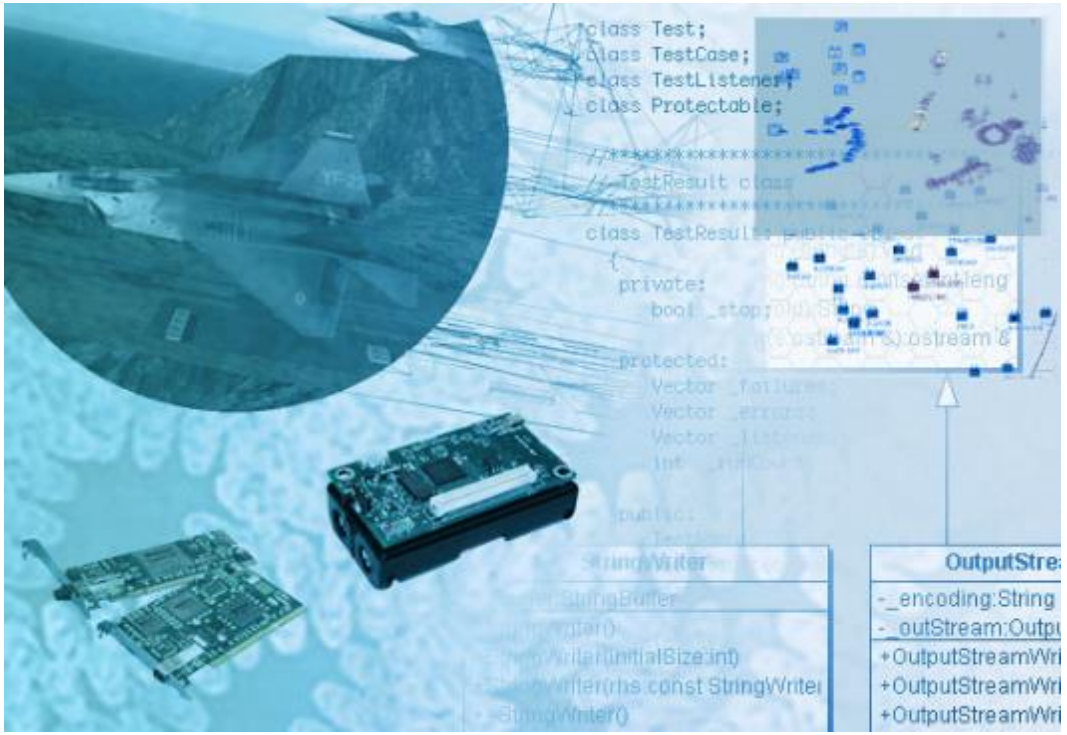
CSYE 6200

CONCEPTS OF OBJECT-ORIENTED DESIGN

SESSION 8

MARK G. MUNSON

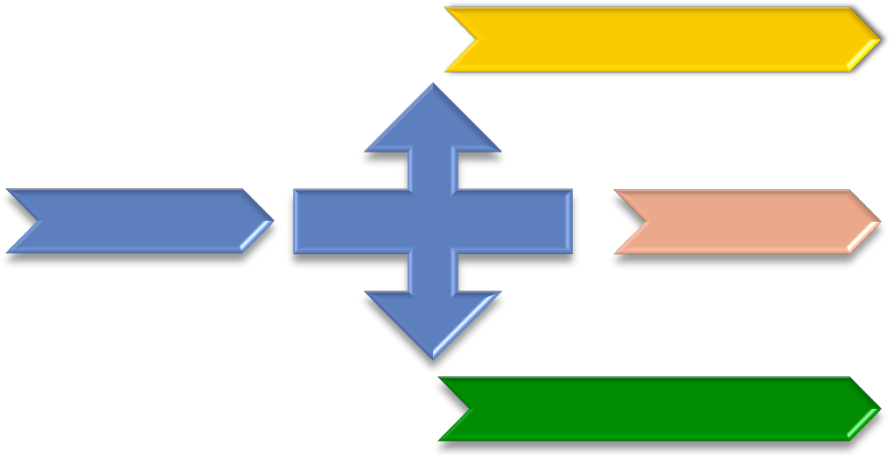


ADMINISTRATION

- **Mar 15th – Session 8: Concurrency, Timers, Threads**
  - Package: Jar
  - Concurrency
  - Design Pattern: Observer
- **Mar 22nd – Session 9: Inner and Anonymous Classes**
  - Enumeration
  - Design Pattern: Command
  - UML: Sequence Diagram
  - Assignment #5a target due date (not collected)
- **Mar 29th – Session 10: Communication/Networking**
  - Design Pattern: Abstract Factory
  - HTTP Networking
  - Lambdas
  - Assignment 5b diagram review
- **Apr 5th – Session 11: Complex Development/Deployment**
  - Libraries and Packages
  - GUI / GIS Development
  - Quiz 3
- **Apr 12th – Session 12: Extra**
  - Assignment $5abc due
- **Apr 19th – Session 13: Full Review**
- **Apr 26th – Final Exam (Online proctored)**

THE LECTURE

- **Recap**
- **Archive**
- **Concurrency**
  - Timers
  - Threads
    - Inherited
    - Runnable
    - Synchronized
- **Design Pattern: Observer**
- **Assignment 5**
  - UI Event Subscribing
  - Algorithm
  - Simulation Thread Control
  - User Interface Design
- **Swing (cont.)**



RECAP

JAR FILES

JAVA ARCHIVES

TAPE ARCHIVES → JAR

- **Tape archiving, combined with zip compression, results in the tar command for single file tape archiving (.tar)**
- **Like tar, multiple Java files may be combined and compressed into a single Java ARchive (.jar) file using the jar utility**

```
CSYE6200>  jar cvf csye6200.jar –C ./classes .
```

c = create
v = verbose
f = file

The output jar file

Change directory to 'classes'

Select all files in the directory

JAR

- **A standard Jar file is just a .zip file that contains either your .java files and/or your .class files**

```
project/classes/
                edu/neu/csye6200/io/
                                    FileTest.class
                                    FileWriterTest.class
```

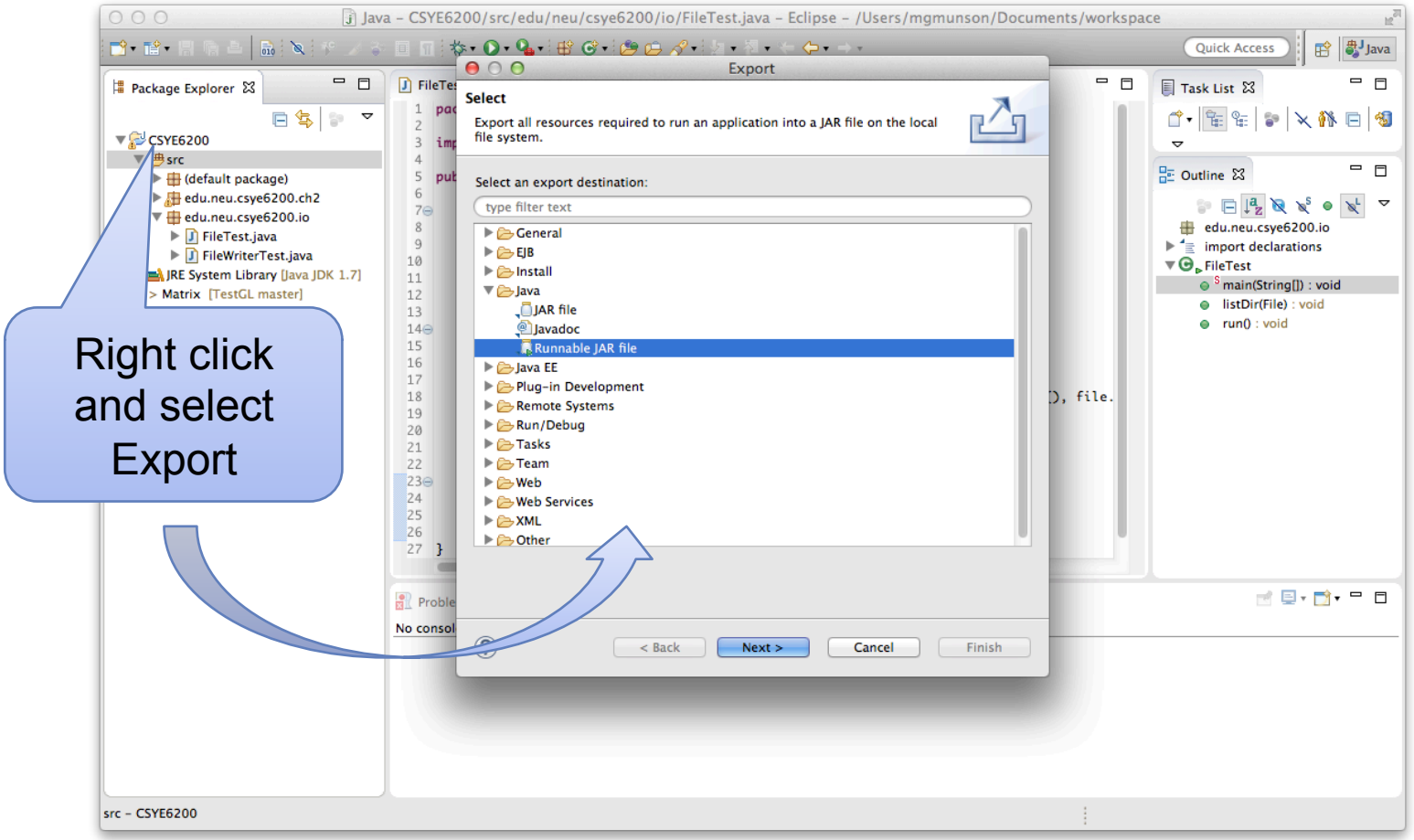
- **A runnable Jar file contains .class files, and also contains a special manifest meta-file, which specifies the start class**

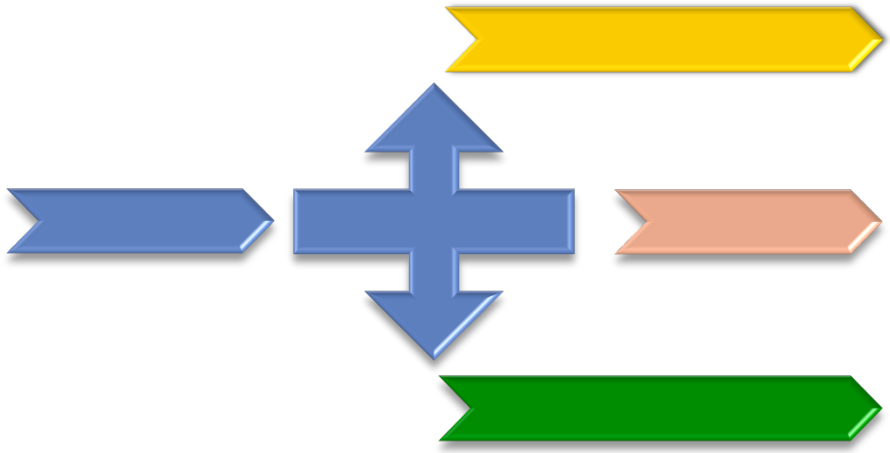
```
META-INF/MANIFEST.MF
```

```
Manifest-Version: 1.0
Class-path: .
Main-Class: edu.neu.csye6200.io.FileTest
```

JAR

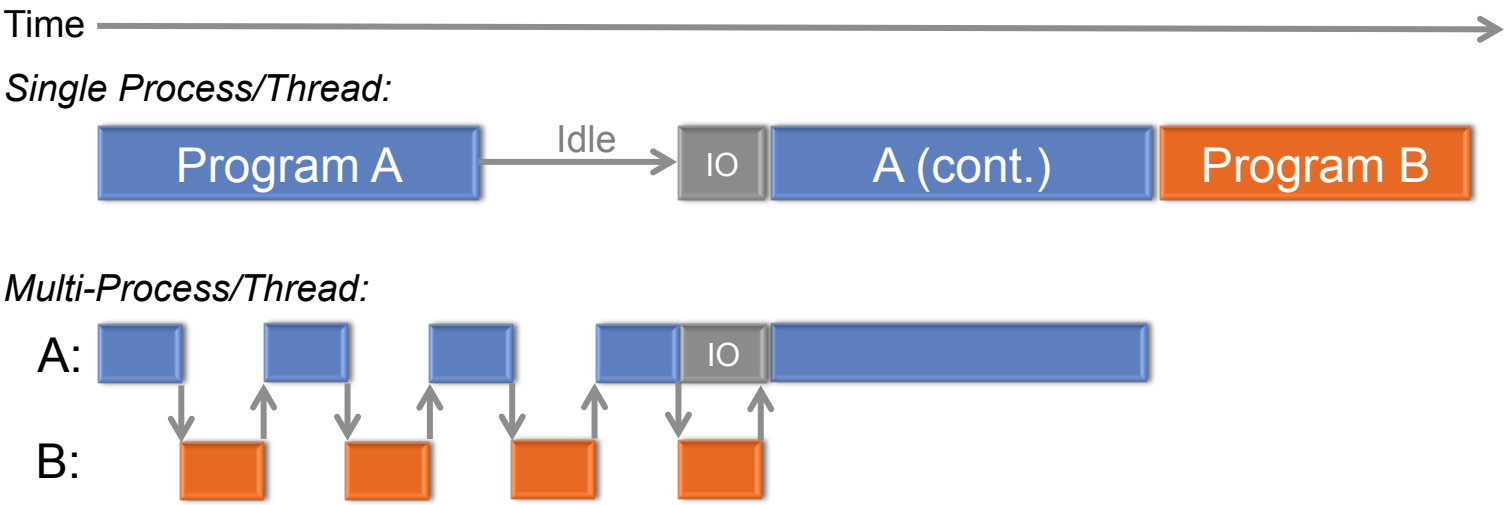
In Eclipse, you may create a jar file using the Export option





CONCURRENCY

TIME SHARING



On computer systems, each running program occupies a Process space

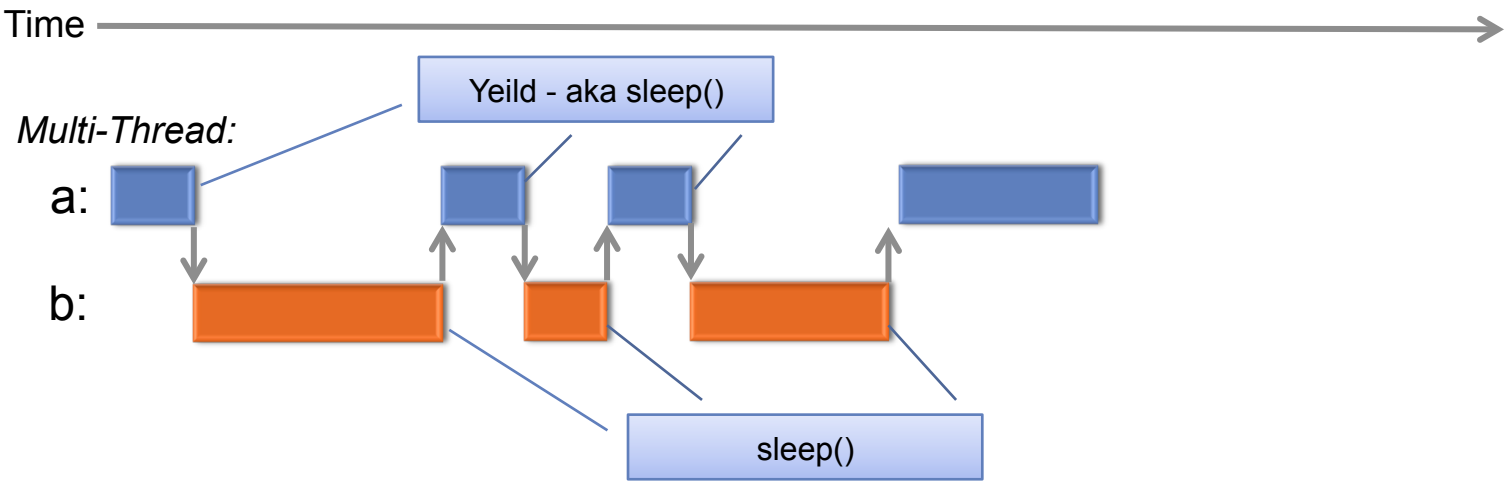
Generally speaking each process runs independently from any others

Time sharing permits two processes to run at the same time

PREEMPTIVE

- **Processes are built by the Operating System (OS)**
  - They are powerful tools
  - They are expensive (time wise) to create
  - They may synchronize with other processes and compete for common resources
  - An 'Idle' process is running that consumes time when nothing else needs to be done
- **Processes are usually pre-emptive**
  - The OS interrupts a program's execution and bookmarks the current state – control is then given to another process
  - The OS steals away time without the process being aware
    - The only indication that this happens is the clock suddenly advances
  - **No action is required by the process**

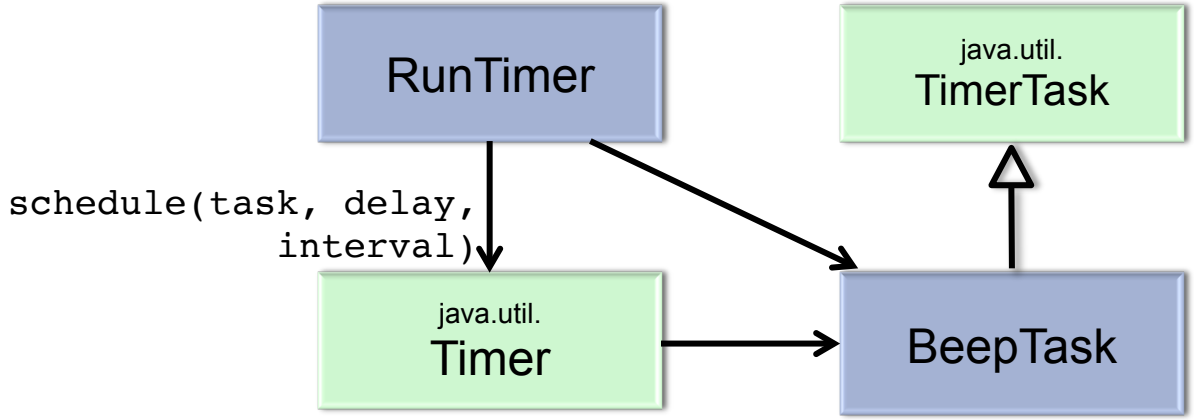
COOPERATIVE



- **Processes are expensive to run, so within each process, separate light-weight threads may be created that agree to cooperate**
- **Each thread will agree to 'sleep' on a regular basis, so that other threads may wake up and do work**
- **Threads have less power, and usually only work with other threads in the same process space**

TIMERS

TIMERS



- **Often you want to execute code periodically**
- **A Thread will work, but it's difficult to control the timing**
- **Instead of extending Thread or implementing Runnable, you can extend TimerTask**
- **A Timer will schedule a TimerTask that wakes on a periodic basis**

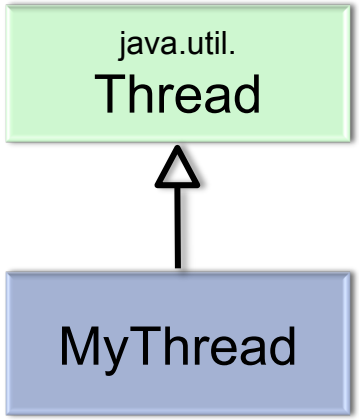
TIMER DEMO

```
java.util.TimerTask;
@override
public void run(){}
```

```
import java.util.Timer;
Timer t = new Timer();
t.schedule(new RemindTask(), 3000L, 1000L)
```

THREADS

THREAD

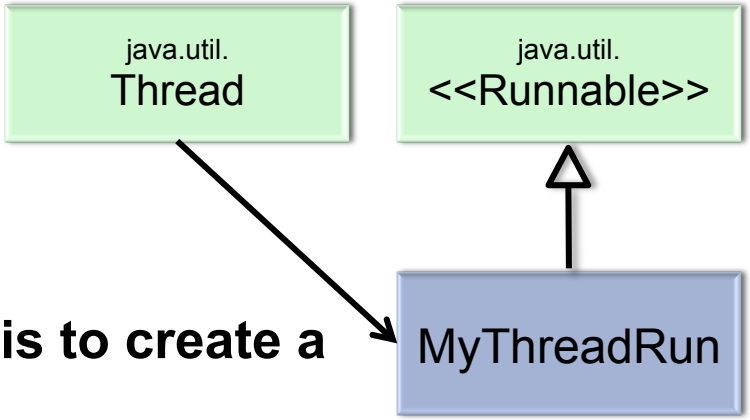


- **One simple way to create concurrency is to create a class that inherits directly from Thread**
  - Override the run() method
  - Execution begins when start() is called
  - Your class will expose all Thread controls

```
package edu.neu.csye6200.conc;

import java.util.Thread;

public class MyThread extends Thread {
…
    @override
    public void run() {
        // your code starts here
    }
…
}
```

THREAD DEMO

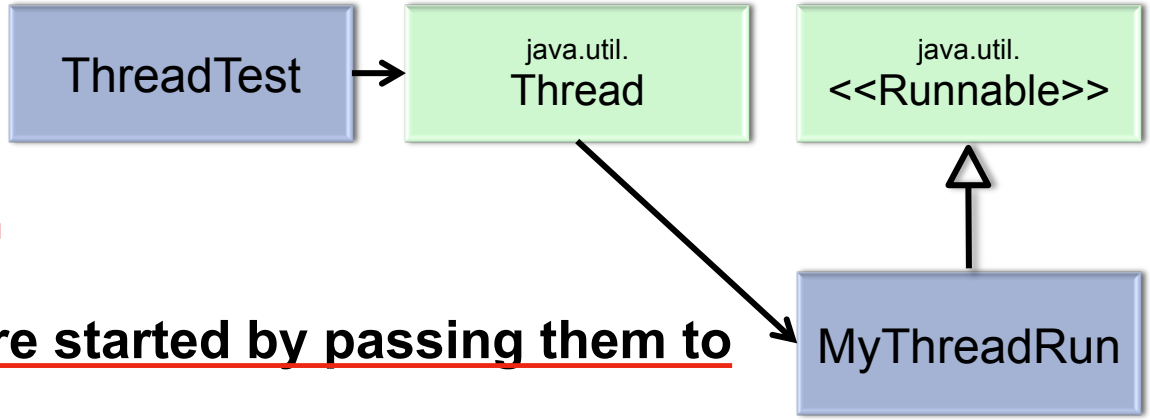
RUNNABLE



- **Another way to create concurrency is to create a class that inherits from Runnable**
  - Override the run() method
  - Execution begins when thread.start() is called

```
package edu.neu.csye6200.conc;

import java.util.Runnable;

public class MyThreadRun implements Runnable {
…
    @override
    public void run() {
        // your code starts here
    }
…
}
```

STARTING RUNNABLE



- **Runnable classes are started by passing them to a thread**
  - Execution begins when thread.start() is called
  - Main thread continues running

```
package edu.neu.csye6200.conc;

import java.util.Thread;

public class TestThread {
   private Thread thread = null;
…
   public TestThread() {
      MyThreadRun myThreadRun = new MyThreadRun();
         thread = new Thread(myThredRun);
         thread.start(); // exec myThreadRun.run();
      // main thread continues…
   }
…
}
```

RUNNABLE DEMO

SEMAPHORES

With parallel and multi-threaded applications, you need to control access to shared resources and classes/methods that aren't "thread safe"

- Processes use Semaphore signals
- Threads use Mutex (Mutual exclusion) locks to control access
- In Java, the `synchronized` keyword provides equivalent functionality to the Mutex

THREAD SAFE

- Often a class or method may accesses internal variables in a way that can lead to errors if two threads of execution are performing concurrent operations.
- These methods must be altered to correct this flaw, or they must be properly marked and documented so that a programmer will protect them from concurrent access.
- A thread-safe class or method is able to safely work with multiple threads accessing it at the same time.

SYNCHORIZED

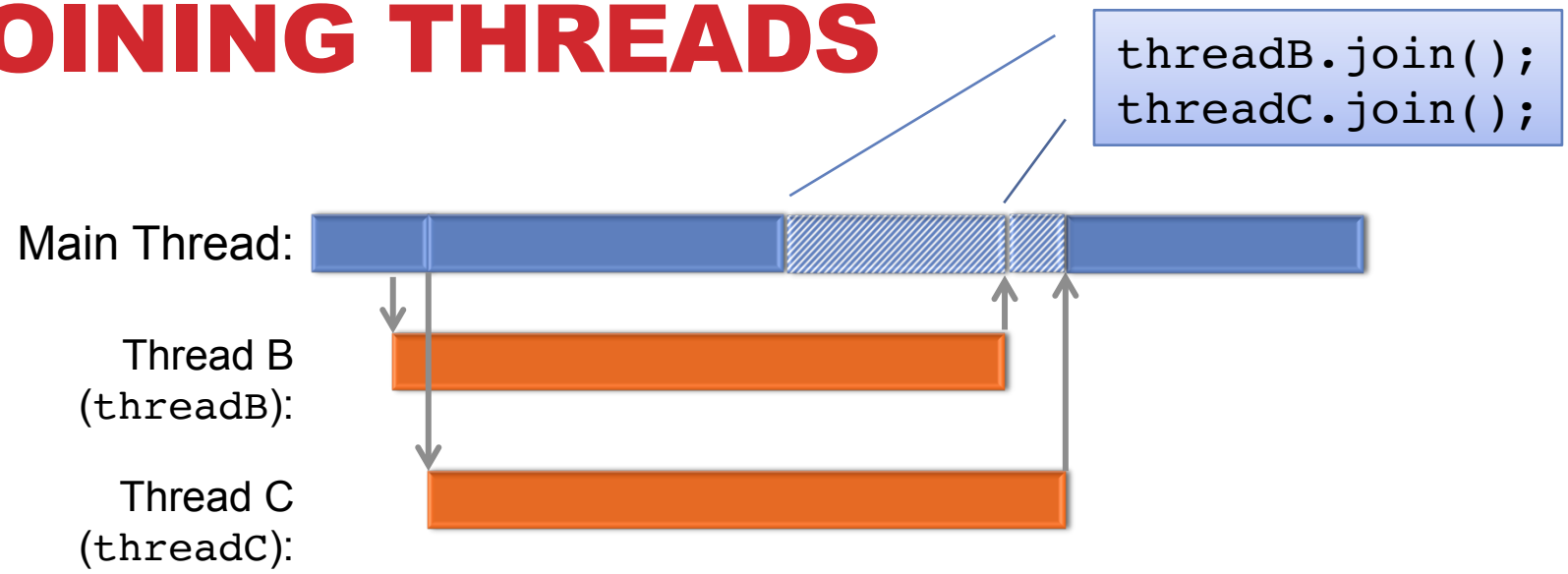
Methods my be preceded by the synchronized keyword to prevent multiple threads from calling the method at the same time

```
public synchronized int updateList() { … }
```

When external code needs to be accessed by more than one thread and each should wait on the other, the synchornized keyword can be used with an object lock:

```
synchronized(object) {
// synchronized code
}
```

JOINING THREADS



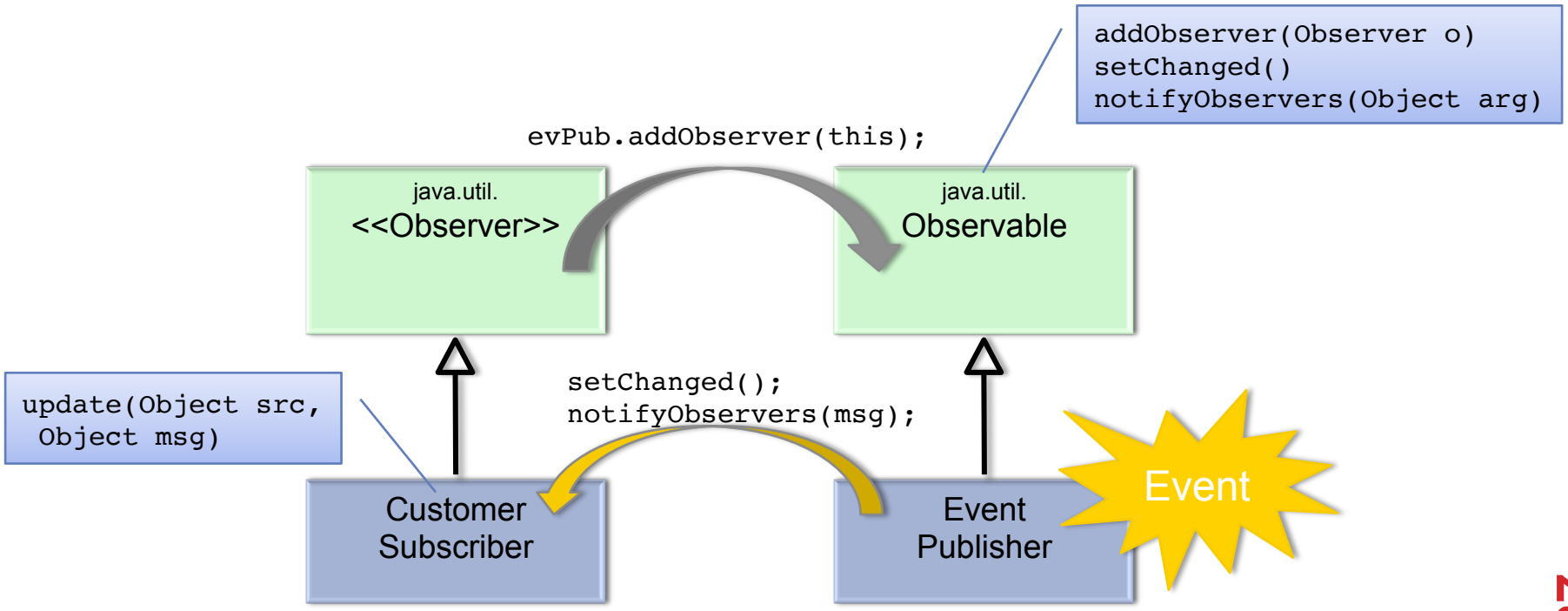
- **When multiple threads are created, the main thread may be rejoined with the other executing threads before proceeding**
- **Each call to join will block until the specified thread completes execution**

INTRO TO OBSERVER

DESIGN PATTERNS

DESIGN PATTERN: OBSERVER

- Often in event driven systems, it is useful for an Object to notify other Objects when a change has occurred
- The Observer Design Pattern permits a class to register as an observer for changes in another class



OBSERVER

- **When a change is experience in an Observable class, it may notify an observing class through a notify action:**

```
package edu.neu.csye6200.obs;

import java.util.Observable;

public class EventPublisher extends Observable {
…
   public void doAction() { // tell our subscribing friends
     setChanged();
     notifyObservers(new Message(Event.UPDATE));
   }
…
}
```

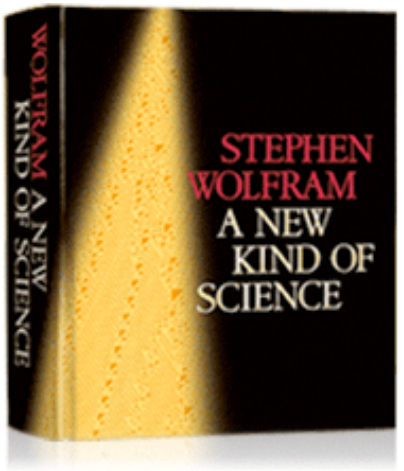
OBSERVER

- For a class to be notified of changes, it must first add itself as an Observer to an class that is Observable.

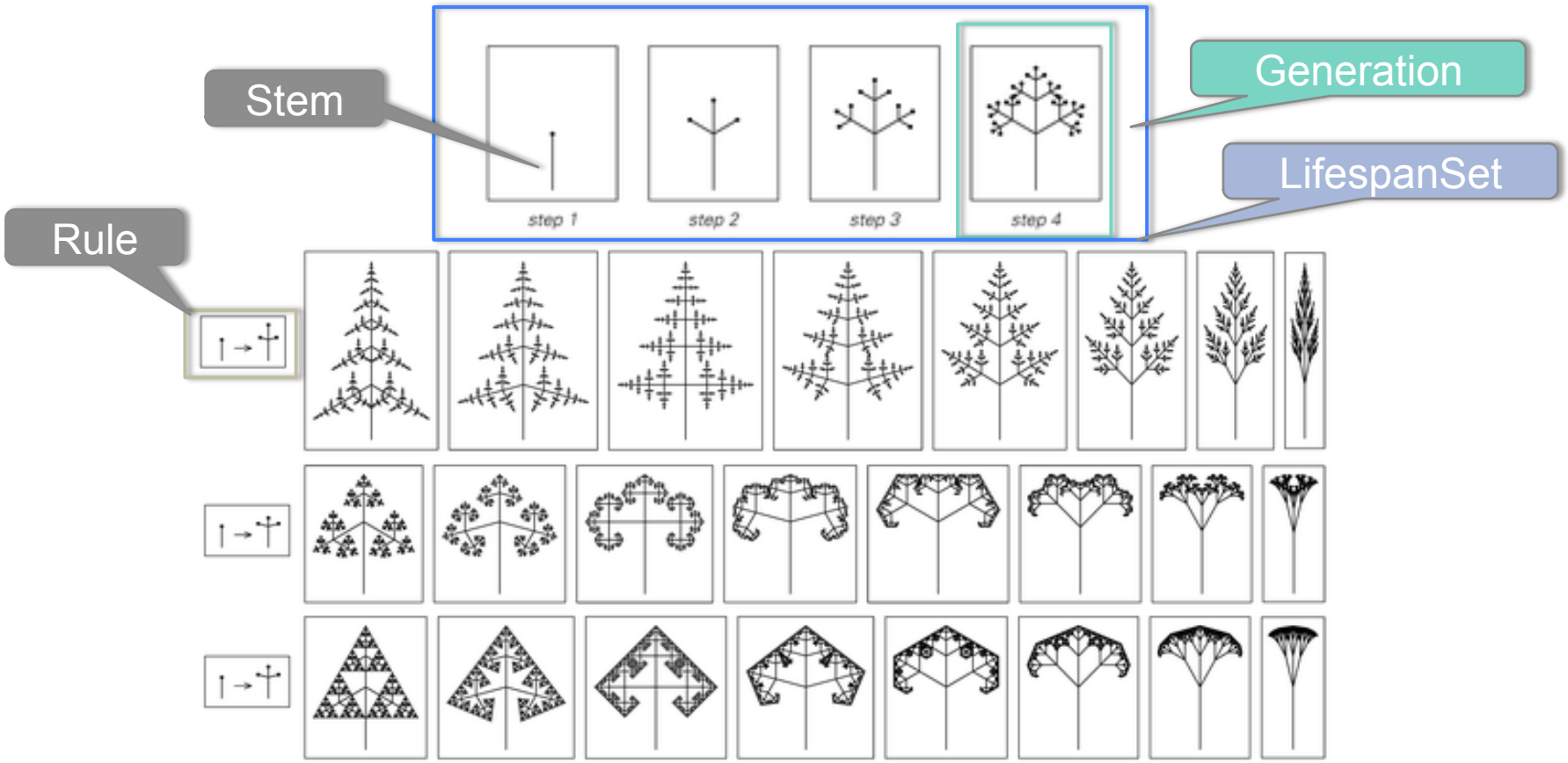
```
package edu.neu.csye6200.obs;

import java.util.Observable;
import java.util.Observer;

public class CustomerSubscriber implements Observer {
…
    public void update(Observable src, Object arg1) {
    // src is the sending Observable class
    // arg1 is the object that is sent as a message
    }
…
}
```

DRONE DELIVERY SIMULATION

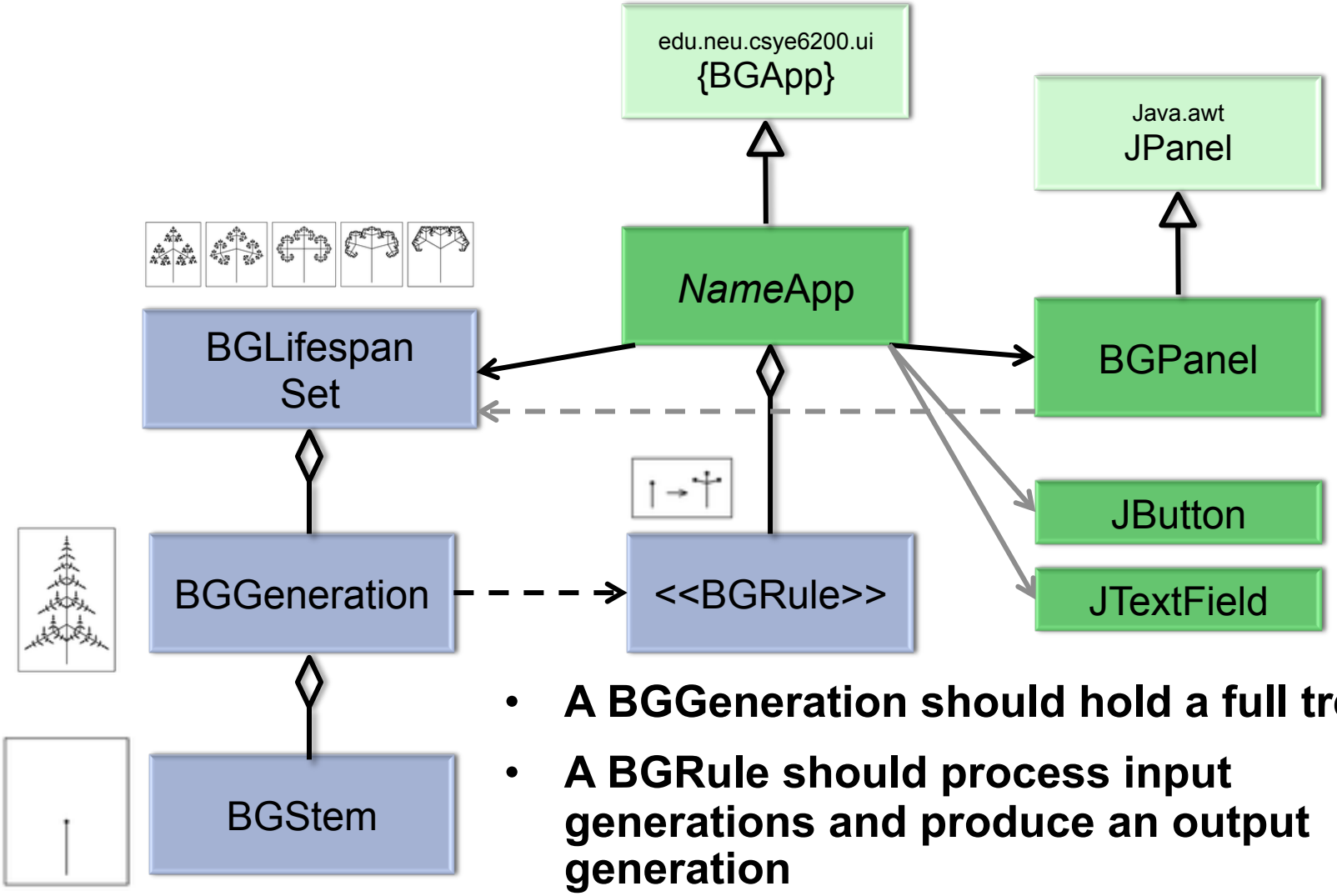
COMPUTATIONAL SCIENCES



- Chapter 8: Implications for Everyday Systems
  - Subsection: Growth of Plants and Animals



BG APP STATIC CLASS DIAGRAM



- **A BGGeneration should hold a full tree/leaf**
- **A BGRule should process input generations and produce an output generation**

SIMULATION THREAD CONTROL

GUI - SWING

GUI EVENTS

- **GUI programs are a good example of event driven programming**
  - User mouse actions on controls
  - Keyboard actions
  - Window resizing

NEXT WEEK

JABG Read:

- Ch. 12 Enumerations pp. 422-430

Design Pattern: Command

- **Assignment #5 Biological Growth [multi-part]**
  - Part A – Biological Growth (Part A Target Due date: Mar. 22nd) *Not Collected*
    - Create a package called edu.neu.csye6200.bg
    - Create supporting classes to perform Biological Growth calculations
    - Recommended tasks:
      - Create a Stem class which contains an array of child Stem instances plus length, direction, age, etc.
      - Create a BGRule class which can extend an existing Stem, and possibly create new generation of stems based on a Stem growth
      - Create a BGGenerationSet (aka BGLifespanSet) that holds multiple BG generations and can call the BGRule class repeatedly to "grow" successive generations.
        Question: will you keep copies of prior generations?
      - Add interfaces to collect growth data and generate statistics
      - Demonstrate valid growth
    - Simulation
      - Set initial conditions
      - Add a run() method that begins execution of a simulation loop
      - Simulation loop: Advance time by a unit amount and re-evaluate the drone position/motion
      - Add instrumentation points to gather desired statistics