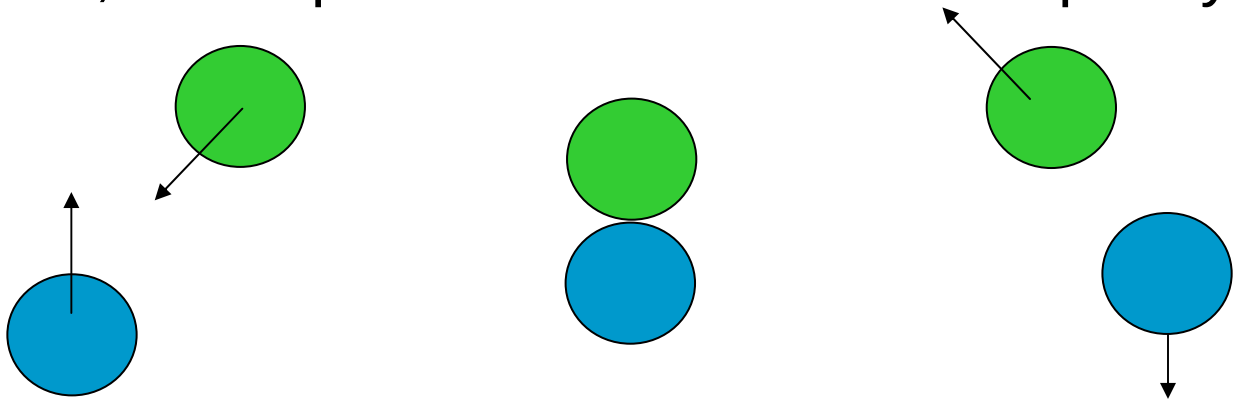
ECE 462
Object-Oriented Programming using C++ and Java

Lecture 19

Yung-Hsiang Lu
yunglu@purdue.edu

Can Java Be Faster Than C++?

- Yes, it is possible.
- C++ may create too many temporary objects.
- example: vector addition
- why to study vector addition? why is it important?
- vector operations are the foundation in many games for calculation collisions. In a complex game with many objects, it is important to detect collision quickly.



addvector1.cpp

```
vector<double> operator + (vector<double> vec1,   vector<double> vec2)
{
  unsigned int vecindex = 0;
  unsigned int veclength = vec1.size();
  vector<double> vdest;
  while (vecindex < veclength)
    {
      vdest.push_back(vec1[vecindex] + vec2[vecindex]);
      vecindex ++;
    }
  return vdest;
}
```

```
void addVectors(const vector<double> * vec1,
                const vector<double> * vec2,
                vector<double> * vecdest, int length)
{
  for (int vecindex = 0; vecindex < length; vecindex ++)
    {
      vecdest[vecindex] = vec1[vecindex] + vec2[vecindex];
    }
}
// caller
  vector<double> vsrc1[NUMBER_VECTOR];
  vector<double> vsrc2[NUMBER_VECTOR];
  vector<double> vdest[NUMBER_VECTOR];
  addVectors(vsrc1, vsrc2, vdest, NUMBER_VECTOR);
```

```
// addvector2.cpp
vector<double> operator + (const vector<double> & vec1,
                           const vector<double> & vec2)

// addvector3.cpp
class DVector
{
  int length;
  double * element;
public:
  DVector() { } // needed to create an array of objects
  DVector(int l)
  {
    length = l;
    element = new double[length];
  }
```

```
DVector(const DVector & vecorig)
{
  length = vecorig.length;
  element = new double[length];
  for (int vecindex = 0; vecindex < length; vecindex ++)
    { element[vecindex] = vecorig.element[vecindex]; }
}
void init(int l)
{
  length = l;
  element = new double[length];
}
DVector add(const DVector & vec2) const
{
  DVector vdest(length);
```

```
  for (int vecindex = 0; vecindex < length; vecindex ++)
    {
    vdest.element[vecindex] =
      element[vecindex] + vec2.element[vecindex];
    }
  return vdest;
}
DVector & operator = (const DVector & vecorig)
{
  if (this == & vecorig) { return * this; }
  delete [] element;
  length = vecorig.length;
  element = new double[length];
  for (int vecindex = 0; vecindex < length; vecindex ++)
    { element[vecindex] = vecorig.element[vecindex]; }
  return * this;
}
```

```
~ DVector()
   { delete [] element; }
  void print()
  {
    int vecindex = 0;
    while (vecindex < length)
      {
      cout << element[vecindex] << " ";
      vecindex ++;
     }
    cout << endl;
  }
  void assign(int vecindex, double val)
  { element[vecindex] = val; }
  friend void addVectors(DVector * vec1, DVector * vec2,
                         DVector * vecdest, int numVector);
};
```

```
void addVectors(DVector * vec1, DVector * vec2,
                DVector * vecdest, int numVector)
{
  for (int vecindex = 0; vecindex < numVector; vecindex ++)
    {
      vecdest[vecindex] = vec1[vecindex].add(vec2[vecindex]);
    }
}
void printVectors(DVector * vec, int numVector, string message = "")
{
  cout << message << endl;
  for (int vecindex = 0; vecindex < numVector; vecindex ++)
    { vec[vecindex].print(); }
}
int main(int argc, char * argv[])
{
      .....
```

```
// create vectors
DVector vsrc1[NUMBER_VECTOR];
DVector vsrc2[NUMBER_VECTOR];
DVector vdest[NUMBER_VECTOR];
for (int vecindex = 0; vecindex < NUMBER_VECTOR; vecindex ++)
 {
   vsrc1[vecindex].init(DIMENSION);
   vsrc2[vecindex].init(DIMENSION);
   vdest[vecindex].init(DIMENSION);
   for (int dimindex = 0; dimindex < DIMENSION; dimindex ++)
  {
    vsrc1[vecindex].assign(dimindex, 0.69 * (vecindex + dimindex));
    vsrc2[vecindex].assign(dimindex, 2.7 * (vecindex - dimindex));
  }
 }
addVectors(vsrc1, vsrc2, vdest, NUMBER_VECTOR);
```

```
// addvector4.cpp
void addVector(double * vec1,  double * vec2, double * vdest,  int dimension)
{
  int vecindex = 0;
  while (vecindex < dimension)
    {
      vdest[vecindex] = vec1[vecindex] + vec2[vecindex];
      vecindex ++;
    }
}
void addVectors(double * * vec1, double * * vec2, double * * vecdest,
              int numVector, int dimension)
{
  for (int vecindex = 0; vecindex < numVector; vecindex ++)
    {
      addVector(vec1[vecindex], vec2[vecindex], vecdest[vecindex], dimension);
    }
}
```

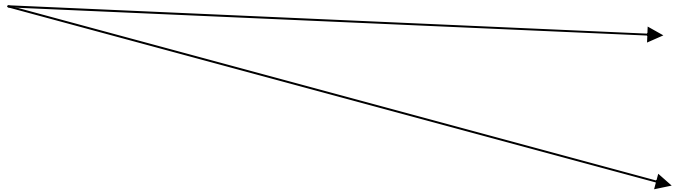
```
// addvector.java
class addvector
{
    public static double[] addVector(double [] vec1,  double [] vec2)
    {
     int vecindex = 0;
     int veclength = vec1.length;
     double [] vdest = new double[veclength];
     while (vecindex < veclength)
        {
            vdest[vecindex] = vec1[vecindex] + vec2[vecindex];
            vecindex ++;
        }
     return vdest;
    }
    public static void addVectors(final double[][] vec1,  final double[][] vec2,
                                  double [][] vecdest)
```

```
{
 for (int vecindex = 0; vecindex < vec1.length; vecindex ++)
     {    vecdest[vecindex] = addVector(vec1[vecindex],  vec2[vecindex]);    }
}
public static void printVector(final double[] vec)
{
 int vecindex = 0;
 while (vecindex < vec.length)
     {
         System.out.print(vec[vecindex] + " ");
         vecindex ++;
     }
 System.out.println("");
}
public static void printVectors(final double[][] vec,   String message)
{
```

```
  System.out.println(message);
  for (int vecindex = 0; vecindex < vec.length; vecindex ++)
      { printVector(vec[vecindex]); }
}
public static void main( String[] args )
{
  ......
  // create vectors
  double [][] vsrc1 = new double[NUMBER_VECTOR][DIMENSION];
  double [][] vsrc2 = new double[NUMBER_VECTOR][DIMENSION];
  double [][] vdest = new double[NUMBER_VECTOR][DIMENSION];
  addVectors(vsrc1, vsrc2, vdest);
```

Overhead of Java synchronized

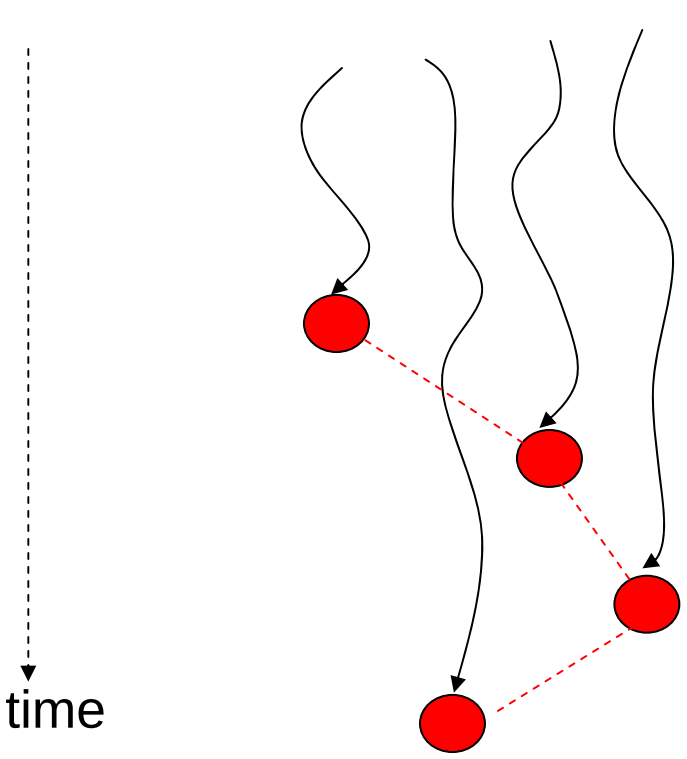
```
synchronized foo() {
...
}
```



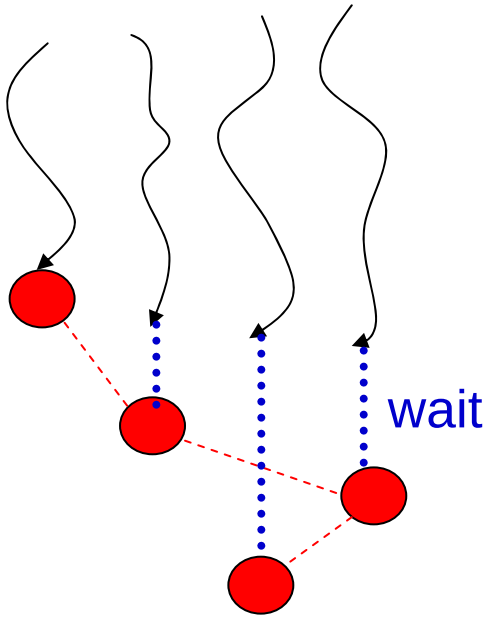
```
foo() {
acquire_lock();
...
release_lock();
}
```

Use Synchronization Efficiently

To take advantage of multi-thread, calling synchronized methods should occur rarely (and at different time).



synchronized

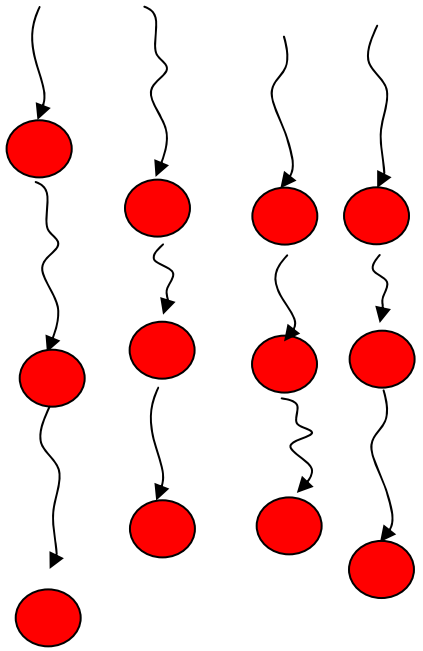


If many threads call a synchronized method at the same time, the program degenerates to a sequential program

Daily Sales Summary

- Your company has four stores in town, open 9AM-9PM.
- Each store manager has to report the total sales amount ($) to the company before the manager leaves.
- The company has one (or more) phone for store managers to call.
- Solution one:

```
for each store
    for each sold item
        call company to report the sales
        hang up
        // if the company has only one phone,
        // the reports are synchronized
```

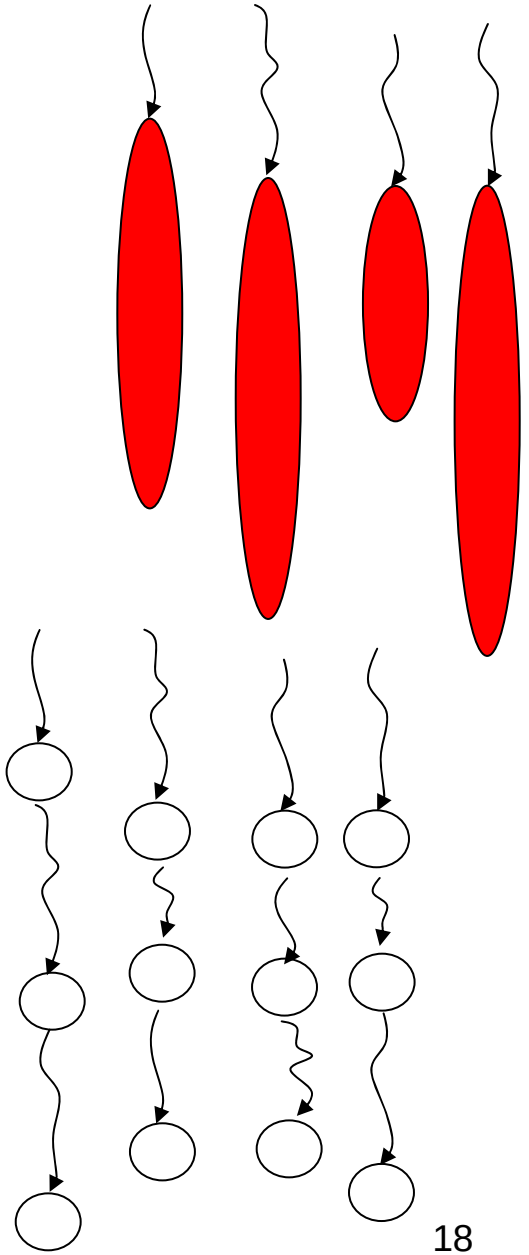


- Solution two:

```
for each store
    for each sold item
        call company to report the sales
        continue the call until all items are reported
        // if the company has only one phone,
        // the reports are synchronized
```

- Solution three:

```
// install four phones in the company, four people
// keep updating the sum on a whiteboard
for each store
    for each sold item
        call company to report the sales
        continue the call until all items are reported
        // no synchronization
        // lead to incorrect answer
```



- Solution four:

  for each store

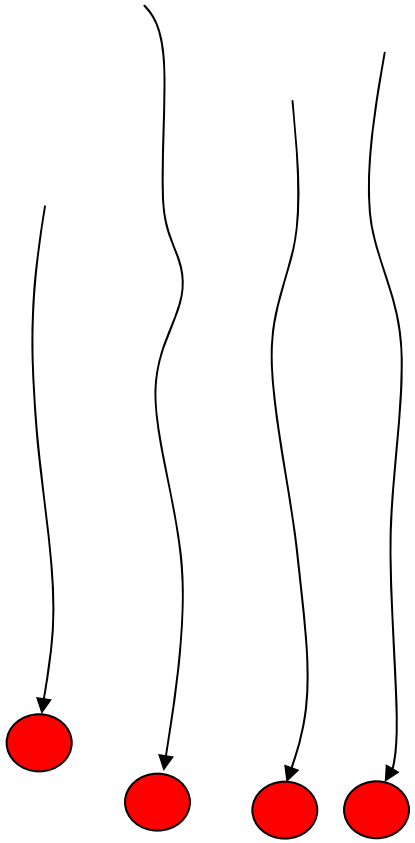
  for each sold item

  add the item to the sum ***in the store***

  call company to report the sales

  // if the company has only one phone,

  // the reports are synchronized



```
import java.util.*;
class SharedObject
{
    private long total;
    public SharedObject() { reset(); }
    synchronized public void syncIncr() { total ++; }
    synchronized public void syncIncr(long sub)
    { total += sub; }
    public void incr() { total ++; } // no synchronized
    synchronized public void reset() { total = 0; }
    synchronized public  String toString()
    { return new String("Total = " + total); }
    public static long NUMBER_ITERATION;
}
class SyncAlways extends Thread
{
```

```
private SharedObject sobj;
    public void run()
    {
     for (long iter = 0; iter < SharedObject.NUMBER_ITERATION; iter ++)
         { sobj.syncIncr(); }
    }
    public SyncAlways(SharedObject obj)
    { sobj = obj; }
}
class NoSync extends Thread
{
    private SharedObject sobj;
    public void run()
    {
     for (long iter = 0;  iter < SharedObject.NUMBER_ITERATION;  iter ++)
         { sobj.incr(); }
    }
```

```
 public NoSync(SharedObject obj)
    { sobj = obj; }
}
class SyncLast extends Thread
{
    private SharedObject sobj;
    private long subtotal = 0;
    public void run()
    {
     for (long iter = 0; iter < SharedObject.NUMBER_ITERATION; iter ++)
         { subtotal ++;  }
     sobj.syncIncr(subtotal);
    }
    public SyncLast(SharedObject obj)
    { sobj = obj; }
}
```

```
class CompareSync
{
    public static void main( String[] args )
    {
        long NUMBER_ITERATION = 10;
     int NUMBER_THREAD = 4;
        if (args.length > 0)
            { NUMBER_ITERATION = Integer.parseInt(args[0]); }
        if (args.length > 1)
            { NUMBER_THREAD = Integer.parseInt(args[1]); }
        if (NUMBER_ITERATION < 10) { NUMBER_ITERATION = 10; }
        if (NUMBER_THREAD < 4) { NUMBER_THREAD = 4; }
      System.out.println("Iteration = " + NUMBER_ITERATION / 100000 +
                           " (millions) " + "Thread = " +  NUMBER_THREAD);
      SharedObject.NUMBER_ITERATION = NUMBER_ITERATION;
      SharedObject sobj = new SharedObject();
      SyncLast [] slt = new SyncLast[NUMBER_THREAD];
```

```
for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
    { slt[tcnt] = new SyncLast(sobj); }
  long time1 = System.currentTimeMillis();
try {
    for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
        { slt[tcnt].start(); }
    for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
        { slt[tcnt].join(); }
} catch (InterruptedException ie) {
    System.out.println("exception caught");
}
  long time2 = System.currentTimeMillis();
  System.out.println("\nSyncLast   = " + 1e-3 * (time2 - time1));
 System.out.println("SyncLast   " + sobj);
sobj.reset();
System.gc();
SyncAlways [] saws = new SyncAlways[NUMBER_THREAD];
```

```
for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
    { saws[tcnt] = new SyncAlways(sobj); }
time1 = System.currentTimeMillis();
try {
    for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
        { saws[tcnt].start(); }
    for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
        { saws[tcnt].join(); }
} catch (InterruptedException ie) {
    System.out.println("exception caught");
}
time2 = System.currentTimeMillis();
  System.out.println("\nSyncAlways = " + 1e-3 * (time2 - time1));
System.out.println("SyncAlways " + sobj);
sobj.reset();
System.gc();
NoSync [] nosyn = new NoSync[NUMBER_THREAD];
```

```
for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
    { nosyn[tcnt] = new NoSync(sobj); }
time1 = System.currentTimeMillis();
try {
    for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
        { nosyn[tcnt].start(); }
    for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
        { nosyn[tcnt].join(); }
} catch (InterruptedException ie) {
    System.out.println("exception caught");
}
time2 = System.currentTimeMillis();
  System.out.println("\nNoSync     = " + 1e-3 * (time2 - time1));
System.out.println("NoSync     " + sobj);
sobj.reset();
System.gc();
long total = 0;
```

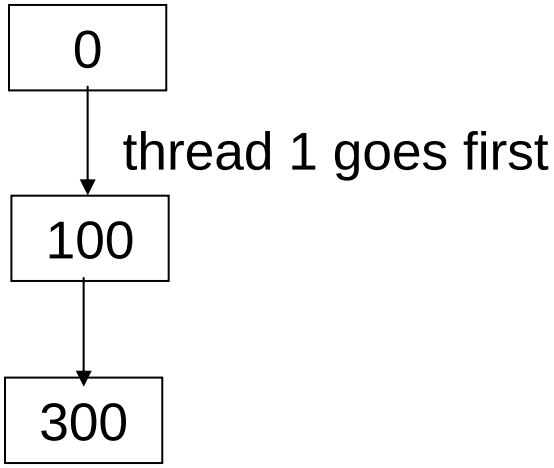
```
    time1 = System.currentTimeMillis();
    for (long iter = 0; iter < NUMBER_ITERATION; iter ++)
       {
           for (int tcnt = 0; tcnt < NUMBER_THREAD; tcnt ++)
              { sobj.incr(); }
       }
    time2 = System.currentTimeMillis();
      System.out.println("\nSequential = " + 1e-3 * (time2 - time1));
    System.out.println("Sequential " + sobj);
   }
}
```

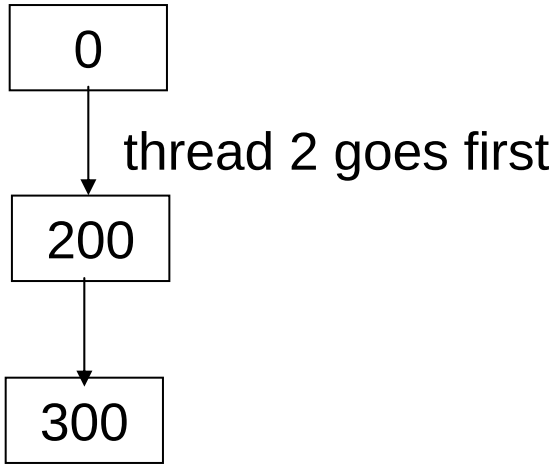
Design Principles

- Frequent synchronization leads to serialization of the threads — problems in solution 1
- Keep each critical section (synchronized method) short — problems in solution 2
- Mutual exclusion needed when a shared object (company's daily sale) may be modified by multiple threads (store managers) — problems in solution 3
- Use local variables (subtotal) to store temporary results before calling a synchronized method — solution 4

Synchronization

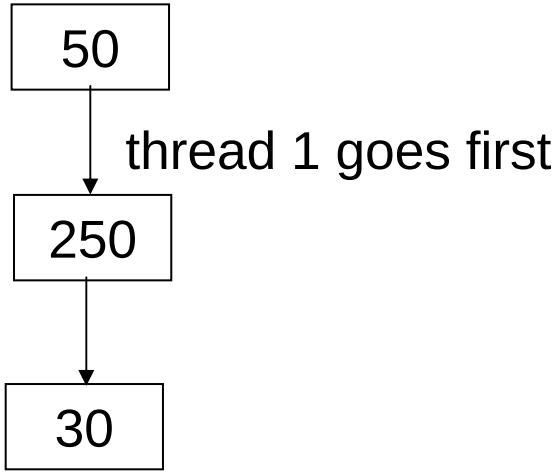
- Synchronization guarantees serialization to update objects; it does **not** guarantee the order of execution.
- Suppose the shared object's initial value is 0.
- thread 1 adds 100 and thread 2 adds 200.
- The correct result is 300.
- The intermediate result can be 100 or 200.

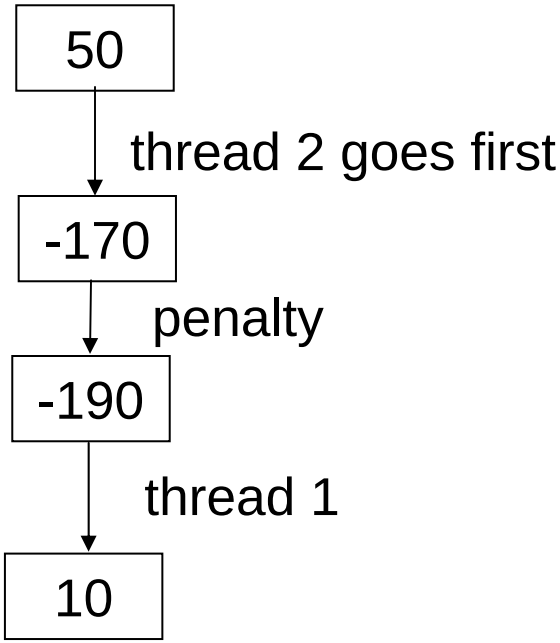




Different Results
(Even With Synchronization)

- Suppose a bank charges $20 penalty for overdraft.
- An account has the initial value of $50.
- Thread 1 deposits $200 to the account.
- Thread 2 withdraws $220 from the account.





Java Container Classes

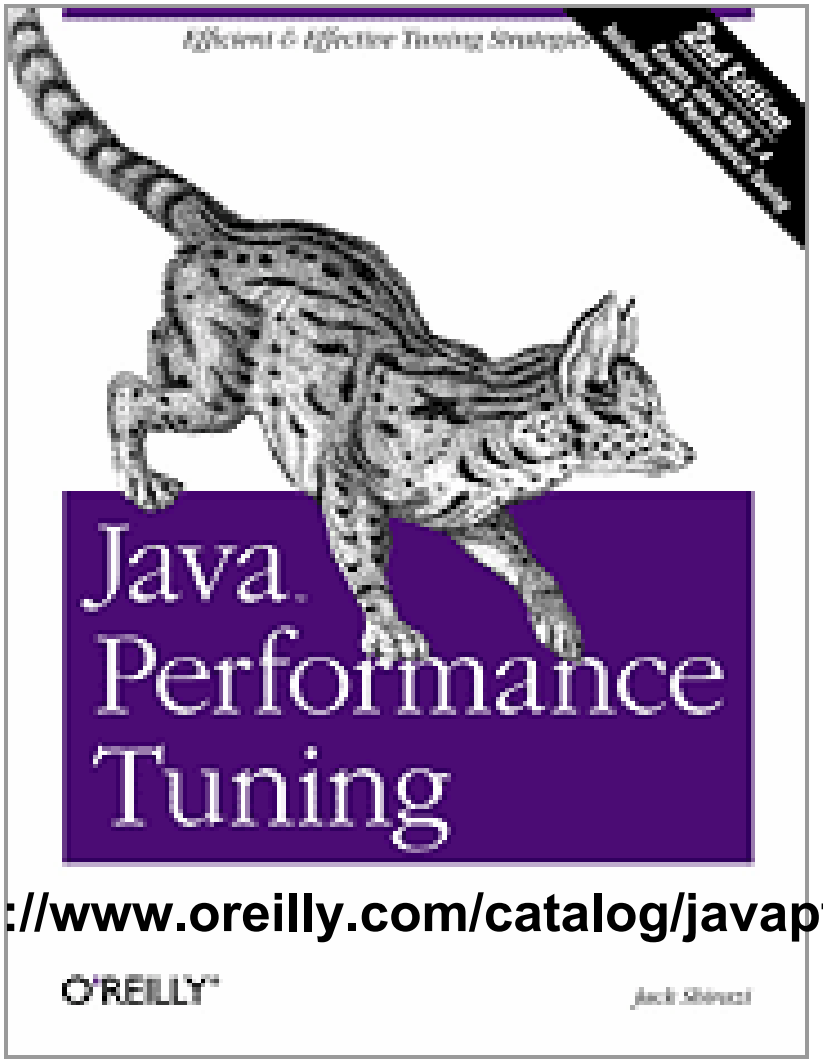
- Similar container classes may have substantially different performance.
- Be aware of the performance penalty for the additional, sometimes unnecessary, functionalities.
- Example: Java Map (key $\rightarrow$value) and SortedMap Interface
  - HashMap
  - TreeMap: keys are sorted
  - HashTable (synchronized)
  - IdentityHashMap

General Rules

- If you know the size (even as an estimation), it is usually much faster to set the size right away, instead of waiting for the objects to resize itself, possibly multiple times.
- For example, HashMap has a constructor that allows you to specify the initial size (with default load factor = 0.75).
- Java automatically enlarges the map if

  # element > size * load factor
- Smaller load factor: faster, need more space
- Using a prime number is likely to make it faster.

More Techniques for Performance



(http://www.oreilly.com/catalog/javapt2/)

```
// java -Xms256m -Xmx512m SetSpeed 1000000
import java.util.*;
class SetSpeed
{
    public static void main( String[] args )
    {
     int NUMBER_ELEMENT = 10;
     if (args.length > 0)
        { NUMBER_ELEMENT = Integer.parseInt(args[0]); }
     if (NUMBER_ELEMENT < 4) { NUMBER_ELEMENT = 4; }
     System.out.println("NUMBER_ELEMENT = " + NUMBER_ELEMENT);
     Random rd = new Random(0);
     Set <String> hSet = new HashSet<String>();
     Set <String> tSet = new TreeSet<String>();
     rd.setSeed(0);
     long time1 = System.currentTimeMillis();
```

```
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
    { hSet.add(Integer.toString(rd.nextInt())); }
long time2 = System.currentTimeMillis();
System.out.println("hSet add = " + 1e-3 * (time2 - time1));
rd.setSeed(0);
time1 = System.currentTimeMillis();
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
    { tSet.add(Integer.toString(rd.nextInt())); }
time2 = System.currentTimeMillis();
System.out.println("tSet add = " + 1e-3 * (time2 - time1));
// compare time to look up
rd.setSeed(0);
boolean found;
time1 = System.currentTimeMillis();
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
    { found = hSet.contains(Integer.toString(rd.nextInt()));  }
time2 = System.currentTimeMillis();
System.out.println("hSet contains = " + 1e-3 * (time2 - time1));
```

```
rd.setSeed(0);
time1 = System.currentTimeMillis();
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
   { found = tSet.contains(Integer.toString(rd.nextInt()));  }
time2 = System.currentTimeMillis();
System.out.println("tSet contains = " + 1e-3 * (time2 - time1));

// get the first ten elements
System.out.println("\nHashSet:");
Iterator hIter = hSet.iterator();
for (int iter = 0; iter < 10; iter ++)
   {
      if (hIter.hasNext())
         { System.out.println(hIter.next()); }
   }
```

```
    System.out.println("\nTreeSet:");
    Iterator tIter = tSet.iterator();
    for (int iter = 0; iter < 10; iter ++)
        {
            if (tIter.hasNext())
                { System.out.println(tIter.next()); }
        }
    }
}
```

```
// java -Xms256m -Xmx512m MapSpeed 1000000
import java.util.*;
class MapSpeed
{
    public static void main( String[] args )
    {
     int NUMBER_ELEMENT = 10;
     if (args.length > 0)
         { NUMBER_ELEMENT = Integer.parseInt(args[0]); }
     if (NUMBER_ELEMENT < 4) { NUMBER_ELEMENT = 4; }
     System.out.println("NUMBER_ELEMENT = " + NUMBER_ELEMENT);
     // create keys, outside insert and lookup so that we do not
     // spend too much on creating keys and values
     String [] hKey = new String[NUMBER_ELEMENT];
     String [] hValue = new String[NUMBER_ELEMENT];
     Random rd = new Random(0);
     Set <String> keySet = new HashSet<String>();
```

```
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
   {
      boolean repeat = true;
      do
         {
               hKey[iter] = new String(Integer.toString(rd.nextInt()));
               if (keySet.contains(hKey) == false)
                  { repeat = false; }
         } while (repeat);
      keySet.add(hKey[iter]);
      hValue[iter] = new String(Integer.toString(rd.nextInt()));
   }
/*
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
   {   System.out.println("(" + hKey[iter] + "," +  hValue[iter] + ")");    }
*/
// compare time to insert (key,value) pairs
HashMap<String, String> hMap = new HashMap<String, String>();
```

```
long time1 = System.currentTimeMillis();
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
   { hMap.put(hKey[iter], hValue[iter]); }
long time2 = System.currentTimeMillis();
System.out.println("hMap put = " + 1e-3 * (time2 - time1));
TreeMap<String, String> tMap = new TreeMap<String, String>();
time1 = System.currentTimeMillis();
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
   { tMap.put(hKey[iter], hValue[iter]);  }
time2 = System.currentTimeMillis();
System.out.println("tMap put = " + 1e-3 * (time2 - time1));
// compare time to look up keys
boolean found;
time1 = System.currentTimeMillis();
for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
   { found = hMap.containsKey(hKey[iter]);  }
time2 = System.currentTimeMillis();
System.out.println("hMap containsKey = " + 1e-3 * (time2 - time1));
```

```
    time1 = System.currentTimeMillis();
    for (int iter = 0; iter < NUMBER_ELEMENT; iter ++)
       { found = tMap.containsKey(hKey[iter]);  }
    time2 = System.currentTimeMillis();
    System.out.println("tMap containsKey = " + 1e-3 * (time2 - time1));
    time1 = System.currentTimeMillis();
    for (int iter = 0; iter < (NUMBER_ELEMENT / 1000); iter ++)
       { found = hMap.containsValue(hValue[iter]);  }
    time2 = System.currentTimeMillis();
    System.out.println("hMap containsValue = " + 1e-3 * (time2 - time1));
    time1 = System.currentTimeMillis();
    for (int iter = 0; iter < (NUMBER_ELEMENT / 1000); iter ++)
       { found = tMap.containsValue(hValue[iter]);  }
    time2 = System.currentTimeMillis();
    System.out.println("tMap containsValue = " + 1e-3 * (time2 - time1));
   }
}
```

ECE 462
Object-Oriented Programming using C++ and Java

Lecture 20

Yung-Hsiang Lu

yunglu@purdue.edu

Lab 11 Correct Synchronization and Performance Comparison

3D Graphics in C++ and Java

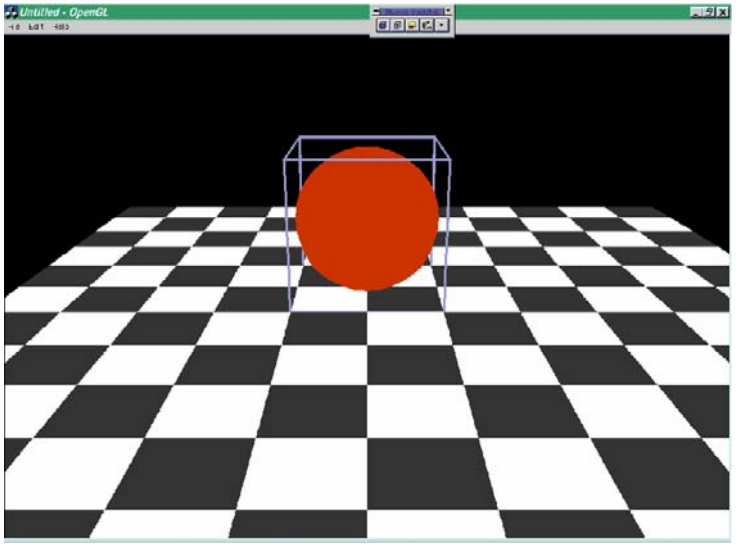
- libraries for 3D graphics:
  - C++

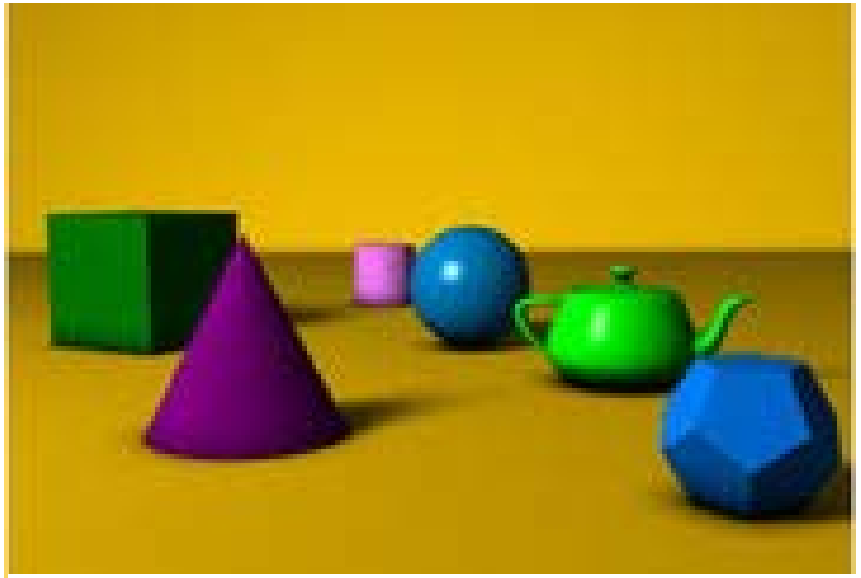
    G3D, http://g3d-cpp.sourceforge.net/index.html

    IrrLicht, http://irrlicht.sourceforge.net/index.html
  - Java 3D, http://java.sun.com/products/java-media/3D/

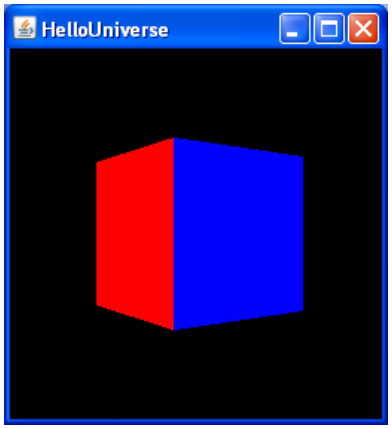
Show 3D on 2D Screen

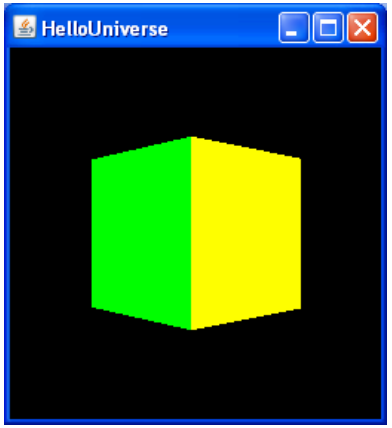
- project 3D on 2D: farther objects look smaller
- calculate depth: a near non-transparent object blocks the visibility of a farther object
- show lighting and shading

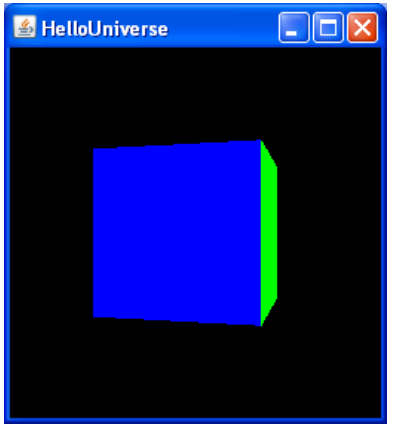




```
// HelloUniverse.java
// http://fivedots.coe.psu.ac.th/~ad/jg/code/index.html
// 2006 Sun Microsystems, Inc.
// Simplified by Andrew Davison, ad@fivedots.coe.psu.ac.th, June 2007
/* A Java 3D example that displays a spinning 3D colored cube.

   Compilation:
     javac HelloUniverse.java

   Execution:
     java HelloUniverse

   If something goes wrong, it means you haven't installed
   Java 3D (or Java) correctly.
 */
import java.awt.*;
import javax.swing.*;
import com.sun.j3d.utils.universe.*;
```







```
import com.sun.j3d.utils.geometry.ColorCube;
import javax.media.j3d.*;
import javax.vecmath.*;
public class HelloUniverse extends JFrame
{
  public HelloUniverse()
  {
    // create a Swing panel inside the JFrame
    JPanel p = new JPanel();
    p.setLayout( new BorderLayout() );
    p.setPreferredSize( new Dimension(250, 250) );
    getContentPane().add(p, BorderLayout.CENTER);
    // add the 3D canvas to panel
    Canvas3D c3d = createCanvas3D();
    p.add(c3d, BorderLayout.CENTER);
    // configure the window (JFrame)
    setTitle("HelloUniverse");
```

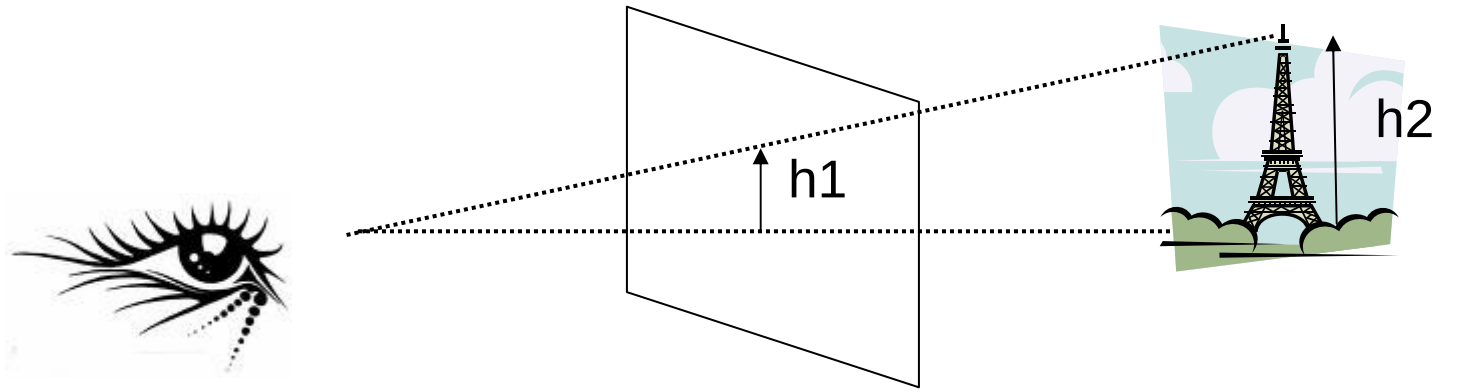
```
    setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    pack();
    setVisible(true);
  }  // end of HelloUniverse()
  private Canvas3D createCanvas3D()
  /* Build a 3D canvas holding a SimpleUniverse which contains
     our 3D scene (a rotating colored cube) */
  {
    // get the preferred graphics configuration for the default screen
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
    // create a Canvas3D using the preferred configuration
    Canvas3D c3d = new Canvas3D(config);
    // create a simple universe
    SimpleUniverse univ = new SimpleUniverse(c3d);
    // move the camera back a bit so the cube can be seen
    univ.getViewingPlatform().setNominalViewingTransform();
    // ensure at least one redraw every 5 ms
    univ.getViewer().getView().setMinimumFrameCycleTime(5);
```

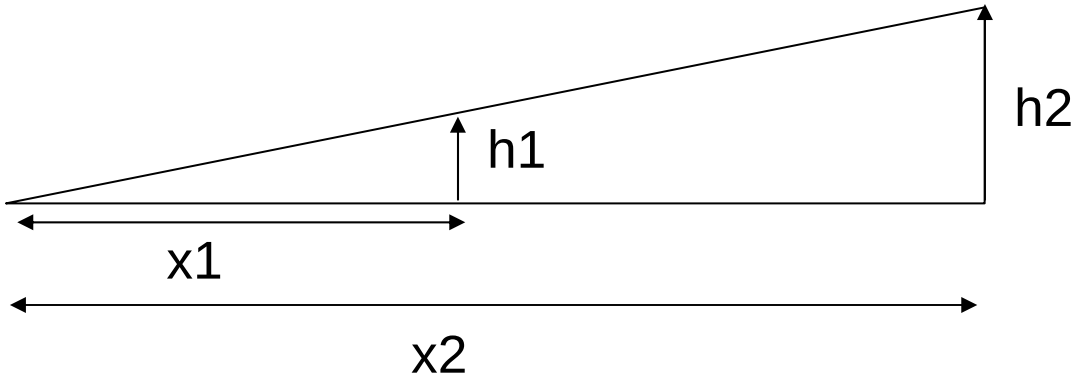
```
    // add the scene to the universe
    BranchGroup scene = createSceneGraph();
    univ.addBranchGraph(scene);
    return c3d;
  }  // end of createCanvas3D()
  public BranchGroup createSceneGraph()
  /* The scene graph is:
         scene ---> tg ---> colored cube
                    |
                    ---> rotator
  */
  {
    BranchGroup scene = new BranchGroup();
    /* Create a TransformGroup node. Enable its TRANSFORM_WRITE
       capability so it can be affected at run time */
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    scene.addChild(tg);   // add to the scene
```

```
  // connect a coloured cube to the TransformGroup
  tg.addChild( new ColorCube(0.4) );
  /* Create a rotation behaviour (a rotation interpolator) which will
     make the cube spin around its y-axis, taking 4 secs to do one
     rotation. */
  Transform3D yAxis = new Transform3D();
  Alpha rotationAlpha = new Alpha(-1, 4000);   // 4 secs
  RotationInterpolator rotator =
            new RotationInterpolator(rotationAlpha, tg,
                              yAxis, 0.0f, (float) Math.PI*2.0f);
  rotator.setSchedulingBounds(
                    new BoundingSphere( new Point3d(0,0,0), 100.0) );
  scene.addChild(rotator);    // add to the scene
  // optimize the scene graph
  scene.compile();
  return scene;
} // end of createSceneGraph()
```

```
  // -------------------------------------------------------
  public static void main(String args[])
  {  new HelloUniverse();  }
} // end of HelloUniverse class
```

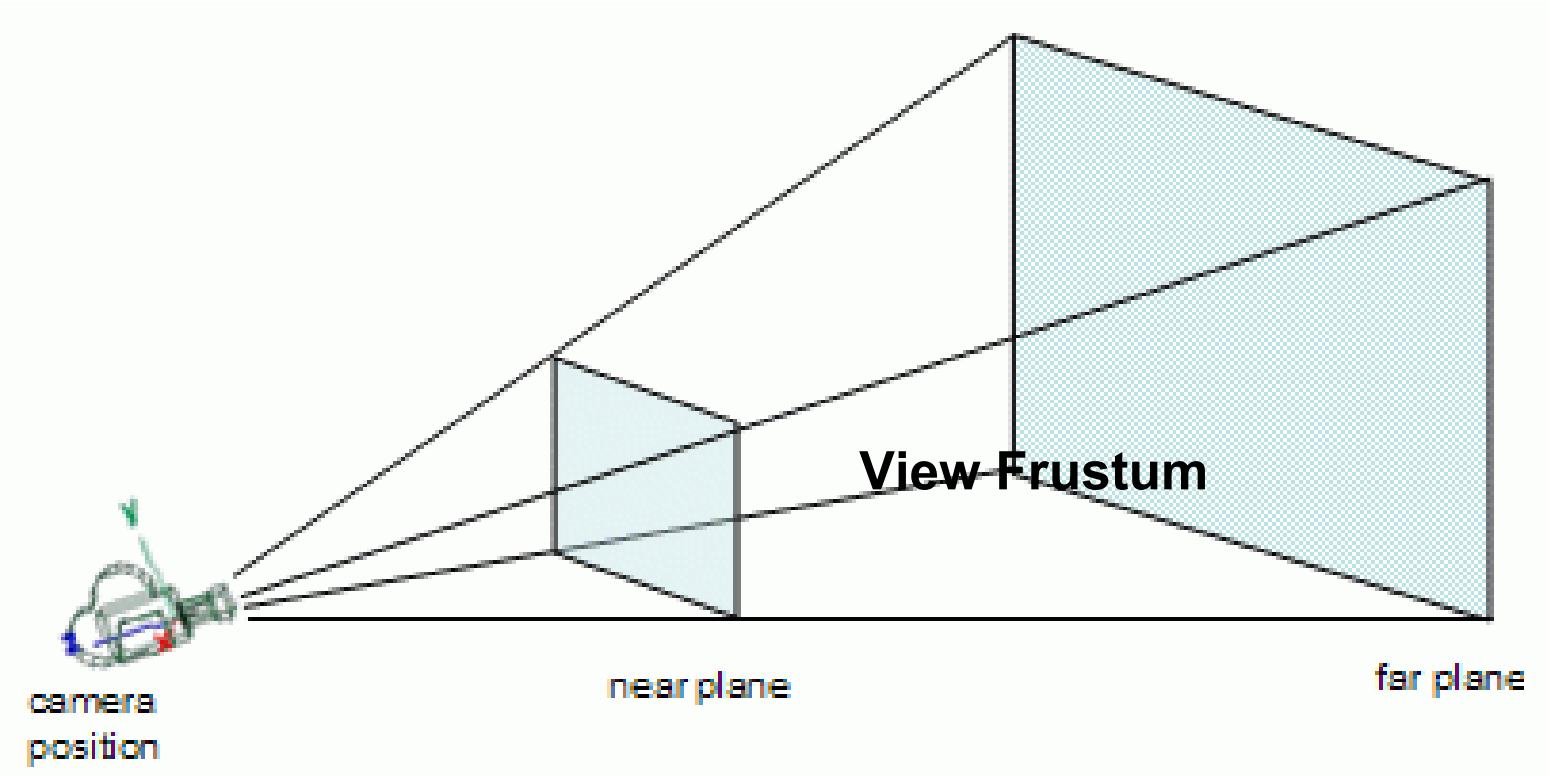
Calculate Projection on Screen

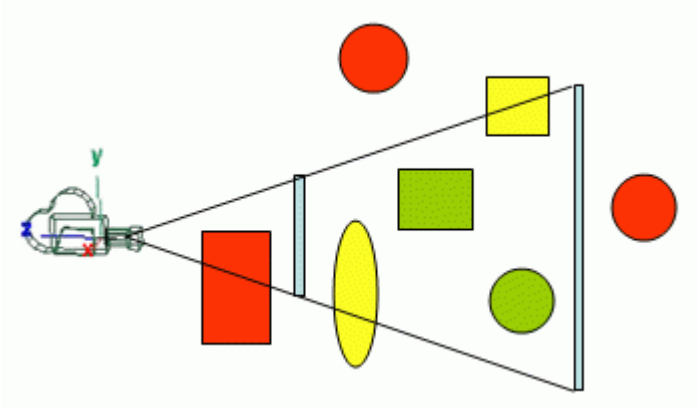




$$\frac{h1}{x1} = \frac{h2}{x2} \Rightarrow h1 = x1\frac{h2}{x2}$$

A farther object (larger x2) looks shorter (smaller h1).

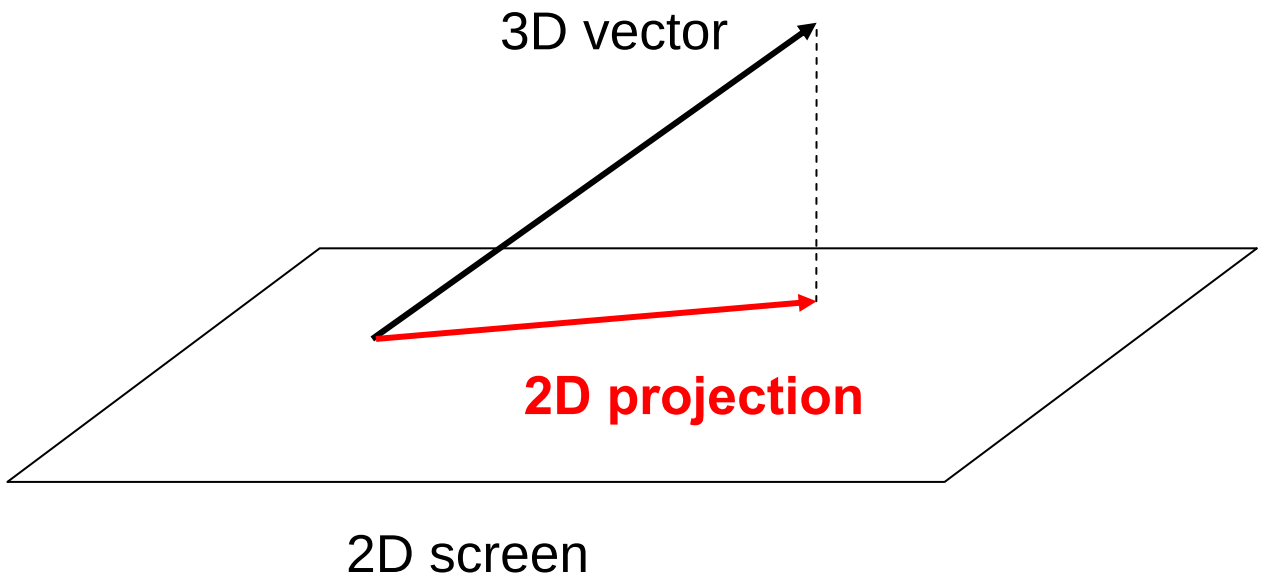




Yellow and green objects are rendered because they are (partially) inside the frustum. The orange objects are not rendered. The green sphere is invisible because it is behind the yellow object.

Projection

3D? It is a **projection** of 3D objects on a 2D screen.



If a vector is (x, y, z), its 2D projection
on the X-Y plane is (x,y)

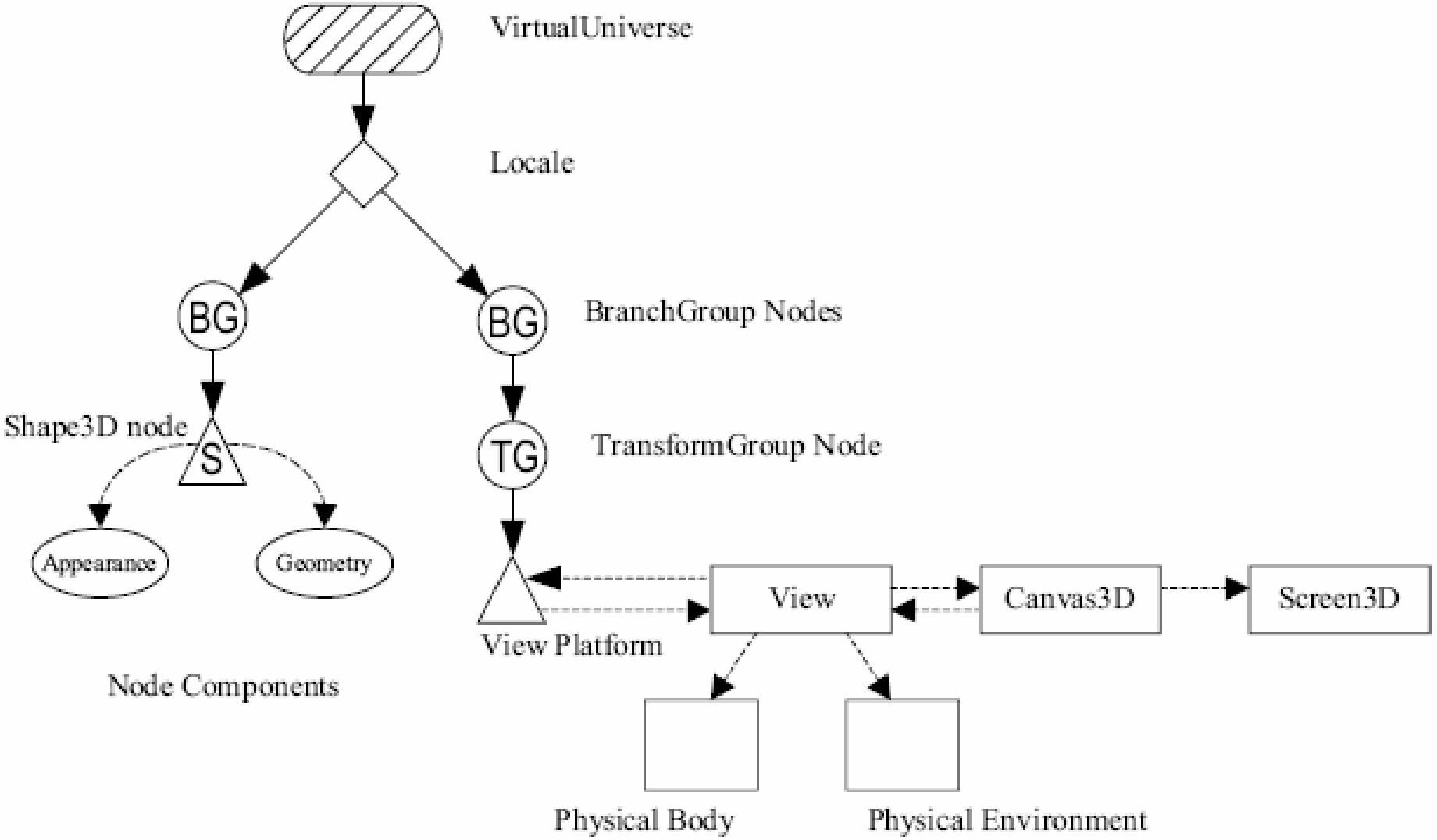
Why is 3D More Complex

- The image on the screen depends on many factors
  - location of the viewer
  - locations of the objects, including their relative depths to the viewer
  - shape of the objects
  - locations of the lights
  - surface materials of the objects, reflective (such as polished metal) or absorptive (such as dark-color cloth)
  - motion
  - ...
- This course is not about graphics so we will explain only the basic programming concepts.

Using Java 3D

- tutorial: http://java.sun.com/developer/onlineTraining/java3d/
- “virtual universe”: the world where all geometric objects reside and the source for rendering
- graph scene: populated with instances (i.e. objects) from Java 3D classes, including geometry, sound, light ...
- A graph scene is often constructed as a tree
- “locale”: landmark to determine the locations of objects

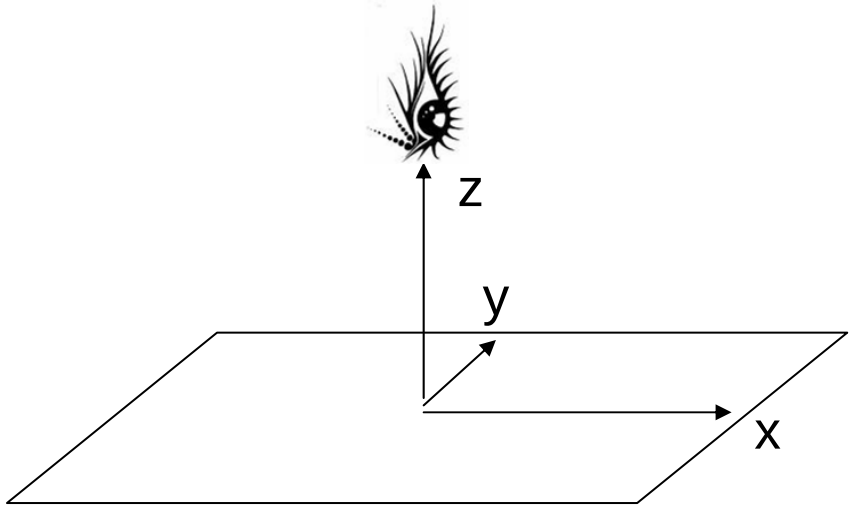
- (Other 3D libraries have similar concepts.)


VirtualUniverse
Locale
BG
BG
BranchGroup Nodes
Shape3D node
S
TG
TransformGroup Node
Appearance
Geometry
View
Canvas3D
Screen3D
View Platform
Node Components
Physical Body
Physical Environment

Steps for a 3D Program

1. create a Canvas3D object (C3D)
2. create a VirtualUniverse object
3. create a Locale object and attach it to the VirtualUniverse object
4. construct a **view branch** graph
   - create a View object
   - create a ViewPlatform object (VP)
   - create a PhysicalBody object (PB)
   - create a PhysicalEnvironment object (PE)
   - attach VP, PB, PE, and C3D to the View object

5. construct content branch graph(s)
6. compile branch graph(s)
7. insert subgraphs into the Locale



default Java viewing plane:
x-y, centered at (0, 0, 0).
The eye looks toward negative Z.

3D Transformation

- Let [x, y, z, 1] represent a 3-D location (x,y,z), called homogeneous coordinates
- Transformation:
  - translation, i.e. move it to (x+dx, y+dy, z+dz)
  - scaling, move it (ax, by, cz)
  - rotation θ along z axis (x *cos*θ - y *sin*θ, x *sin*θ + y *cos*θ, z)
- All transformations can be expressed by matrices.

- translation

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ z + dz \\ 1 \end{bmatrix}$$

- scaling

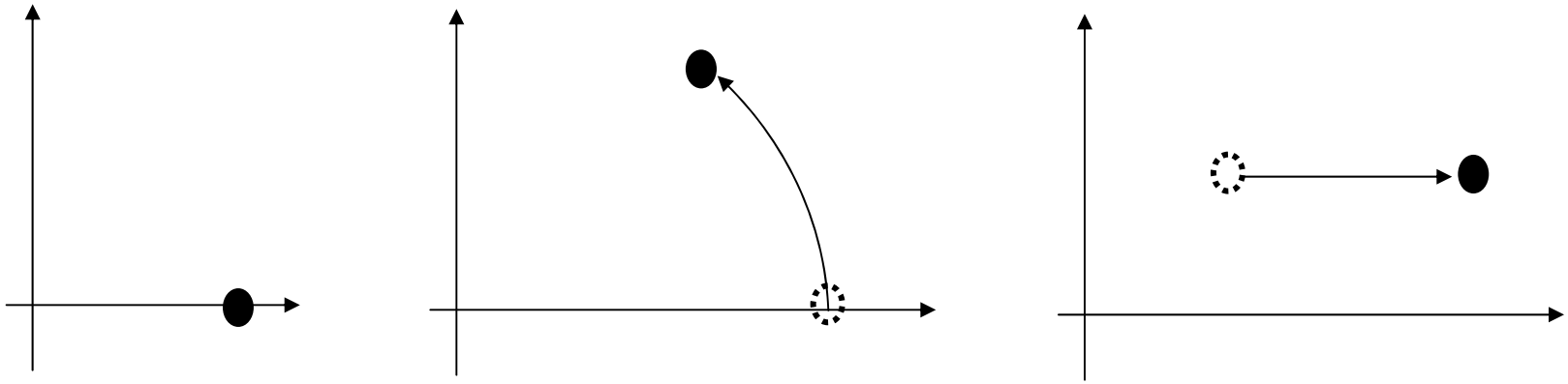
$$\begin{bmatrix} a & 0 & 0 & dx \\ 0 & b & 0 & dy \\ 0 & 0 & c & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ cz \\ 1 \end{bmatrix}$$

- rotation along z

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & dx \\ \sin\theta & \cos\theta & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \\ 1 \end{bmatrix}$$

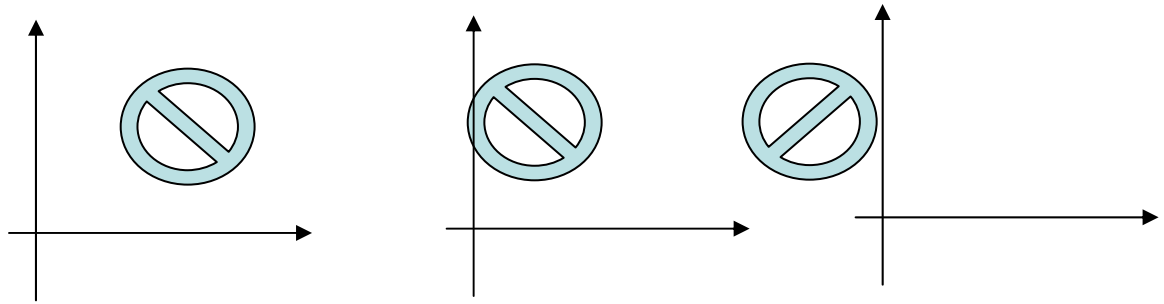
Transformation *not* Commutative

- $T_1\ T_2 \neq T_2\ T_1$ in general
- (3,0) translate (3,0) then rotate $45^o \Rightarrow$ (4.2, 4.2)
- (3,0) rotate 45^o then translate (3,0) $\Rightarrow$ (5.1, 2.1)

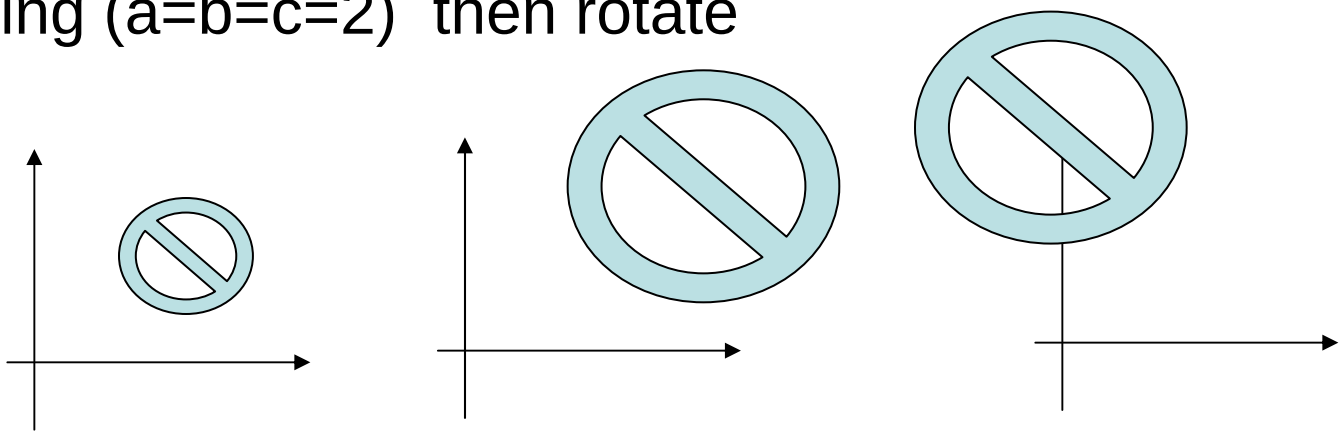


Combine Transformation

- translation then scaling (mirror, a= -1, b= c= 1)

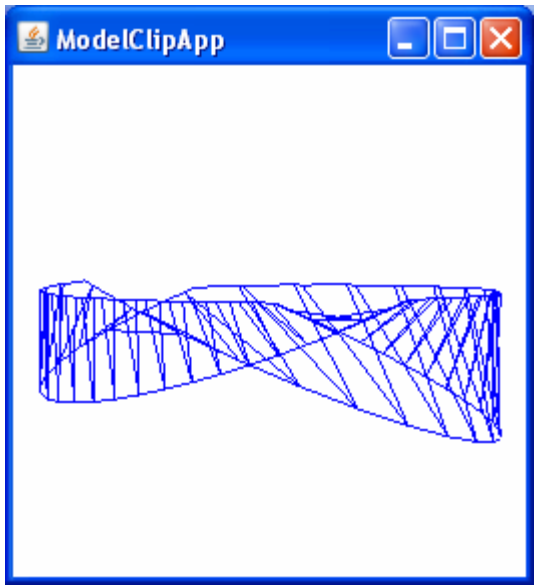


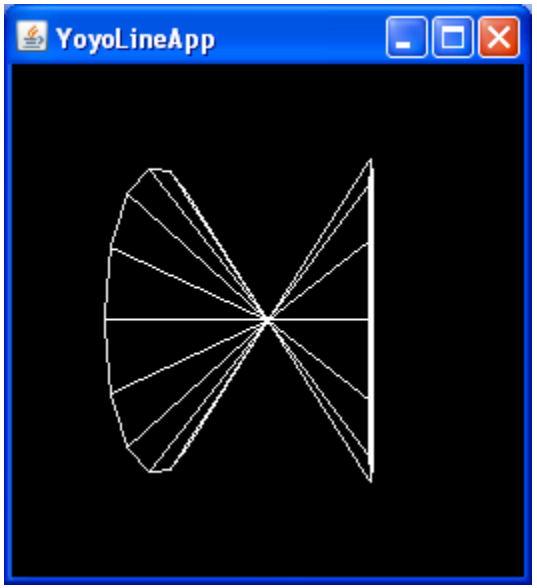
- scaling (a=b=c=2) then rotate



Creating Geometric Objects

- use basic shapes, such as box, cone, cylinder, or spheres
- specify vertex coordinates for points, lines, and polygon surfaces





Create Geometric Shape
by Specifying Coordinates

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
public class TwistStripApp extends Applet {
    ////////////////////////////////////////////////
    //
    // create Twist visual object
    //
```

```
public class Twist extends Shape3D{
    ////////////////////////////////////////
    //
      // create twist subgraph
    //
      public Twist() {
          this.setGeometry(createGeometry());
          this.setAppearance(createAppearance());
      } // end of twist constructor
      Geometry createGeometry(){
          TriangleStripArray twistStrip;
       Color3f blue  = new Color3f(0.0f, 0.0f, 1.0f);
          // create triangle strip for twist
          int N = 80;
          int stripCounts[] = {N};
          twistStrip = new TriangleStripArray(N,
            TriangleStripArray.COORDINATES  | TriangleStripArray.COLOR_3,
            stripCounts
```

```
    );
    double a;
    int v;
    for(v = 0, a=0.0; v < N; v+=2, a=v*2.0*Math.PI/(N-2)){
       twistStrip.setCoordinate(v, new Point3d(
              0.7*Math.sin(a)+0.2*Math.cos(a),
              0.3*Math.sin(a),
              0.7*Math.cos(a)+0.2*Math.cos(a)));
       twistStrip.setCoordinate(v+1, new Point3d(
              0.7*Math.sin(a)-0.2*Math.cos(a),
              -0.3*Math.sin(a),
              0.7*Math.cos(a)-0.2*Math.cos(a)));
       twistStrip.setColor(v, blue);
       twistStrip.setColor(v+1, blue);
     }
    return twistStrip;
  }
```

```
    // create Appearance for Twist Strip
    //
    // this method creates the default Appearance for the
    // twist strip.  The commented line of code containting
    // the setCullFace will fix the problem of half of the
    // Twisted Strip disappearing.
    Appearance createAppearance(){
        Appearance twistAppear = new Appearance();
        PolygonAttributes polyAttrib = new PolygonAttributes();
        // polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);
        twistAppear.setPolygonAttributes(polyAttrib);
        return twistAppear;
    }
} // end of class Twist
////////////////////////////////////////////////
//
// create scene graph branch group
//
```

```
public BranchGroup createSceneGraph() {
    BranchGroup contentRoot = new BranchGroup();
    // Create the transform group node and initialize it to the
    // identity. Add it to the root of the subgraph.
    TransformGroup objSpin = new TransformGroup();
    objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    contentRoot.addChild(objSpin);
    Shape3D twist = new Twist();
    objSpin.addChild(twist);
    // Duplicate the twist strip geometry and set the
    // appearance of the new Shape3D object to line mode
    // without culling.
    // Add the POLYGON_FILLED and POLYGON_LINE strips
    // in the scene graph at the same point.
    // This will show the triangles of the original Mobius strip that
    // are clipped.  The PolygonOffset is set to prevent stitching.
```

```
PolygonAttributes polyAttrib = new PolygonAttributes();
polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);
polyAttrib.setPolygonMode(PolygonAttributes.POLYGON_LINE);
polyAttrib.setPolygonOffset(0.001f);
Appearance polyAppear = new Appearance();
polyAppear.setPolygonAttributes(polyAttrib);
objSpin.addChild(new Shape3D(twist.getGeometry(), polyAppear));
Alpha rotationAlpha = new Alpha(-1, 16000);
RotationInterpolator rotator =
        new RotationInterpolator(rotationAlpha, objSpin);
// a bounding sphere specifies a region a behavior is active
// create a sphere centered at the origin with radius of 1
BoundingSphere bounds = new BoundingSphere();
rotator.setSchedulingBounds(bounds);
objSpin.addChild(rotator);
// make background white
```

```
    Background background = new Background(1.0f, 1.0f, 1.0f);
    background.setApplicationBounds(bounds);
    contentRoot.addChild(background);
    // Let Java 3D perform optimizations on this scene graph.
    contentRoot.compile();
    return contentRoot;
} // end of CreateSceneGraph method of TwistStripApp
// Create a simple scene and attach it to the virtual universe
public TwistStripApp() {
    setLayout(new BorderLayout());
    GraphicsConfiguration config =
      SimpleUniverse.getPreferredConfiguration();
    Canvas3D canvas3D = new Canvas3D(config);
    add("Center", canvas3D);
    BranchGroup scene = createSceneGraph();
    // SimpleUniverse is a Convenience Utility class
    SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
```

```
      // This will move the ViewPlatform back a bit so the
      // objects in the scene can be viewed.
        simpleU.getViewingPlatform().setNominalViewingTransform();
        simpleU.addBranchGraph(scene);
    } // end of TwistStripApp constructor
    //  The following method allows this to be run as an application
    public static void main(String[] args) {
        System.out.println("TwistStripApp - Java 3D API");
        System.out.println("This program demonstrates back face culling.");
        System.out.print("In this program two visual objects rotate, ");
        System.out.println("one wireframe and one solid surface.");
        System.out.print("The wire frame is visible only when components");
        System.out.println(" of the surface are culled.");
        Frame frame = new MainFrame(new TwistStripApp(), 256, 256);
    } // end of main method of TwistStripApp
} // end of class TwistStripApp
```

Programming Assignment 4

- You decide what to do in this assignment.
- Part 1: design
  - explain the purpose of the program
  - present the **design with sufficient details**
  - focus on the interfaces (public) and the interactions (messages and sequences)
  - use UML to express the design
  - **must use concurrency** through (1) multiple threads, (2) network communication, (3) both
  - SIMD (single instruction multiple data) not allowed

- example: extend PA3 so that (1) two people can play through the Internet (2) spectators can connect to the program and watch the game
  - one person can play against the demo mode
  - the second person can request to play against the first player
- Part 1 needs to provide a **quantitative** analysis of the program. This analysis will be used to evaluate the accuracy of the design.
- Source code is not required for Part 1.
- You decide how to grade 50% of Part 2.

- Part 2: implement the design
  - It has to closely match the design in Part 1 by using (1) number of classes, (2) number of public members, (3) number of objects. Twenty percent (20%) difference is allowed.
  - It has to include the analysis of test coverage. You need to know which line of code has not be executed.

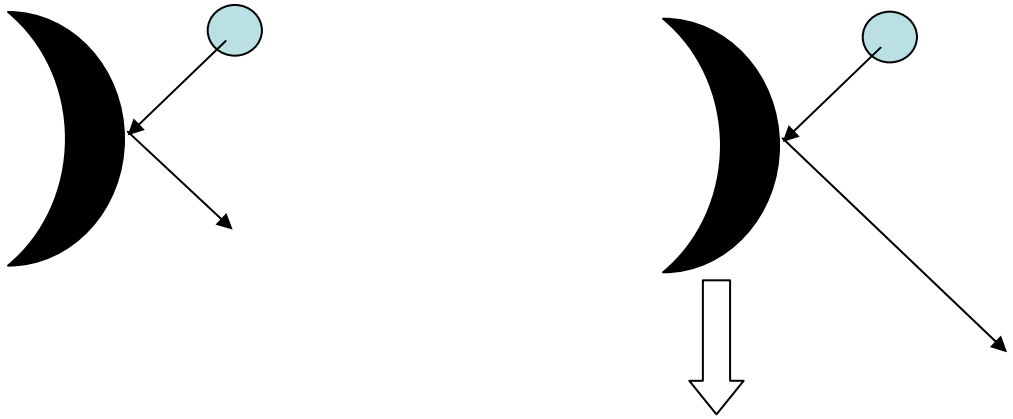
- This will be a good experience to design a software project. **Software ≠ Programming**
- **Software needs design and planning.**

Object-Oriented Design

- four key concepts: class, encapsulation, inheritance, polymorphism
- extract commonalities and create base classes (class and inheritance)
- reuse code and create incremental changes (inheritance and polymorphism)
- think of objects and their interactions (interface design)
- ask objects what to do (encapsulation)
- allow objects to “change their minds” (encapsulation)

“What If”

- evaluate the amount of code to modify if a change is made in the specification
- what if, in PA1, the ball’s speed is affected by the paddle’s speed?



- what if, in PA1, the ball’s speed increases as the score is higher?

“What if”

- what if, in PA1, the ball’s color changes from time to time?
- what if, in PA1, the bricks can move at the moment when the ball hits the paddle?
- what if, in PA1, the length of the paddle can change?

- Object-oriented design (encapsulation) is supposed to help you minimize the amount of code modification.

Inheritance and Polymorphism

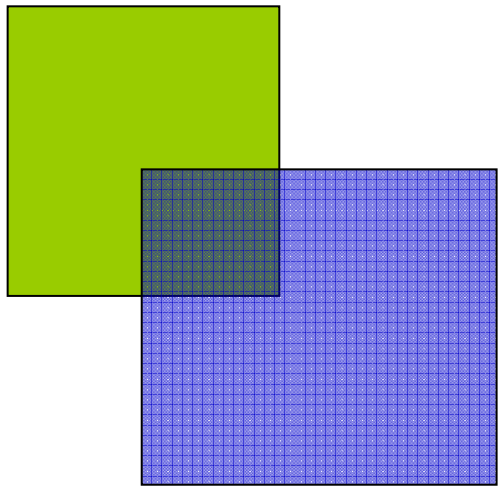
- what if, in PA2, another piece of squares is added?
- what if, in PA2, when a yellow square is above a green square, the green square disappears and the yellow square drops?
- what if, in PA2, only clockwise rotation is allowed?
- what if, in PA2, the pieces' speed increase as the score is higher?
- what if, in PA2, each square is actually a small image not a single color?

Design Tradeoff

- tradeoff between flexibility, performance, and reuse
- before writing “the most general” reusable code, develop a plan for potential scenarios to reuse
- profile program performance. often performance can be improved substantially by changing only a few places while keep the rest general and reusable
- build a “layered” approach: lower-layer (foundation) classes are more likely to be reused than higher-layer classes
- unless there is a strong evidence of performance requirement, reuse what is available from library

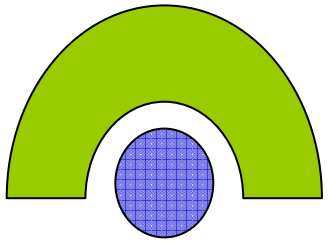
Bad Design

```
// collision detection
if ((ob1.getMaxX() > obj2.getMinX()) &&
    (obj1.getMaxX() < obj2.getMaxX()) &&
    (obj1.getMaxY() > obj2.getMinY()) &&
    (obj1.getMaxY() < obj2.getMaxY()))
{
    delete obj1;
    delete obj2;
}
```



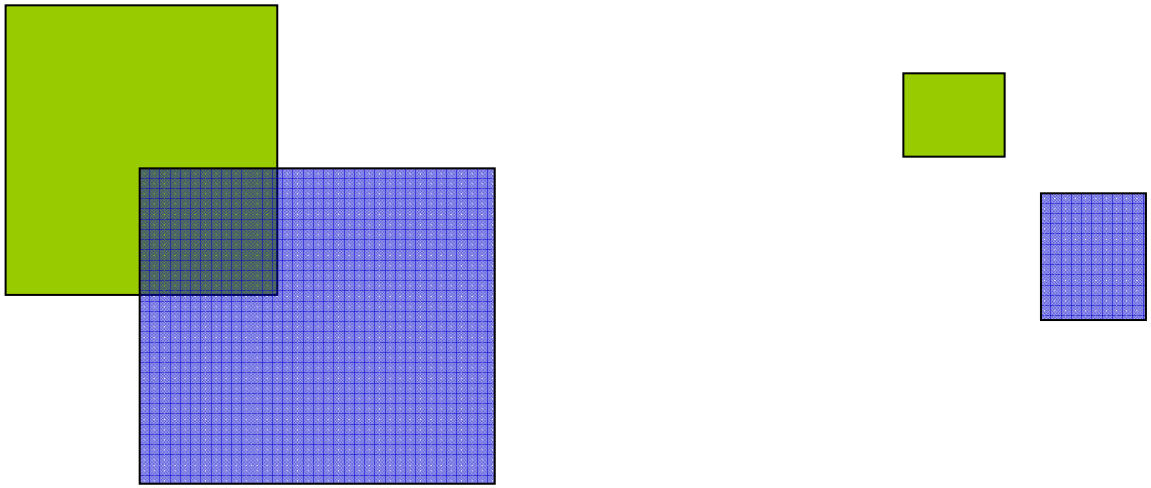
should “ask objects”

what if the two objects are not squares?



Bad Design

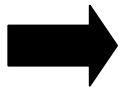
- what if, upon collision, the two objects become smaller?



Probably Bad Design

- If there is a large “switch-case” block, the whole block probably should be converted to using polymorphism.

```
switch (object.role) {
case TEACHER:
        ...
        break;
case STUDENT:
        ...
        break;
}
```



```
object1 = new Teacher(...);
object2 = new Student(...);
...
object1.method1();
object2.method1();
```

Testing in Object-Oriented Design

- A “well-designed” object-oriented program is harder to test **externally** because of encapsulation.
- A typical object-oriented program should have some private attributes that keep the internal states of objects.
- A well-designed object-oriented program, once fully tested, is harder to break externally because the internal states cannot be arbitrarily changed.
- example: ball’s location in PA1

```
x += dx;
y += dy;
```
} the only way to change the ball’s location

```
ball.x = 30;
ball.y = -5;
```
} not allowed

Test Plan

- A test plan is needed to test (any) program, especially for object-oriented program due to limited visibility and controllability.
- Using many cout (or printf, or System.out.println) is ineffective and inefficient.
- We will discuss this topic more in the next lecture.