Program 3 Preview

Dr. Demetrios Glinos
University of Central Florida

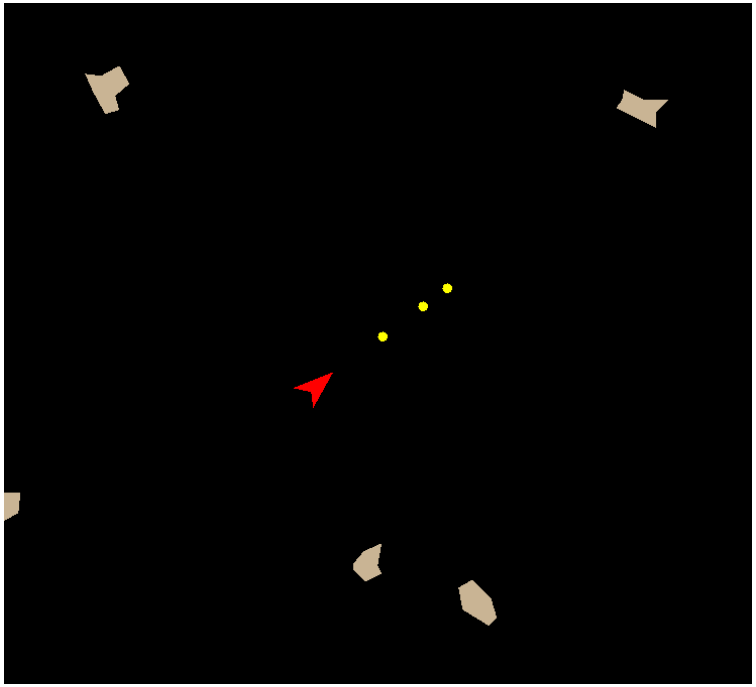
COP3330 – Object Oriented Programming

Outline

- Where We Are Headed
- Program 3 Task Overview
- Using the Javadocs
- Demo Program

Where We Are Headed

- A working Asteroids game in three assignments
  - A playing field with asteroids that move and rotate
  - A rocket of your own design that can move about in the asteroid field
  - Rocket can shoot missiles at asteroids to destroy them, scoring points
  - If the rocket is hit by an asteroid, the game ends



OOP Features

- Developing this game will involve these OOP features:
  - Inheritance — subclasses for asteroids, rocket, and missile
  - Polymorphism — interfaces for key actions and collision detection; also inherent polymorphism in the class hierarchy
  - Event handlers — for keyboard inputs and sound generation
  - Graphics — using convenience methods in external JAR file
  - Encapsulation — functionality will be in separate classes

Program 3 Task Overview

1. Create **Asteroid** class

2. Create **AsteroidField** test driver class

(*that's all for now – not the full game*)

- You will need to use **Blobz.jar** third-party JAR to run your program
  - Contains simulation engine and game graphics
  - Must configure your Eclipse project to use it
  - Must configure test folder on desktop to test

} *Full details for these are in the assignment description and the external libraries lecture and slide set*

Operating the Game GUI

- Start New Game
  - starts a fresh game with asteroids starting from offscreen in all directions
  - **Note:** if the game is paused, you must press stop button before you can restart
- Pause / Unpause
  - pauses and unpauses the game
- Stop
  - stops the game

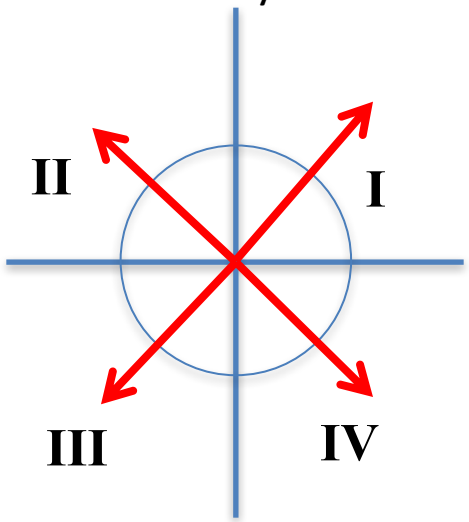
demo: asteroidfield

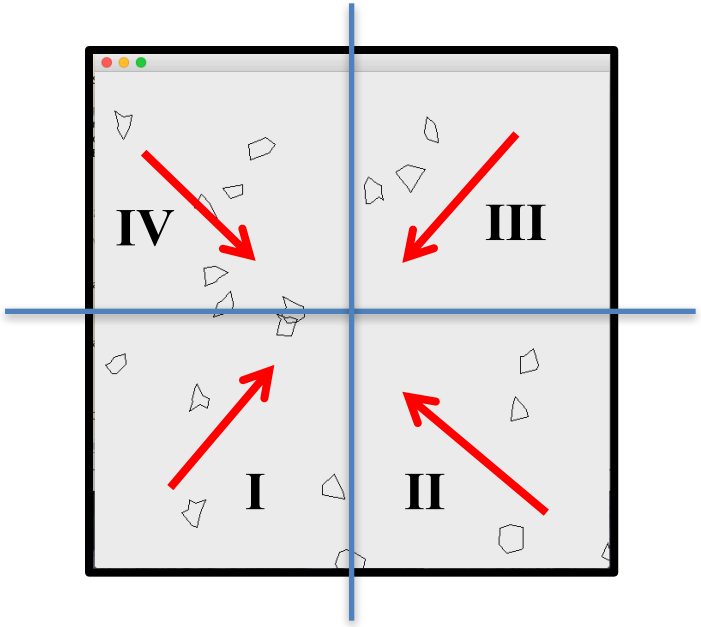
SandBox "flow" Mode

- Set via SandBox instance method:

```
setSandBoxMode(SandBoxMode.FLOW);
```

- In flow mode:
  - Blob can start anywhere off-screen and moves in a straight line
  - Once offscreen, restarts along edge of the corner corresponding to the quadrant of its velocity vector





Variable Frame Rate

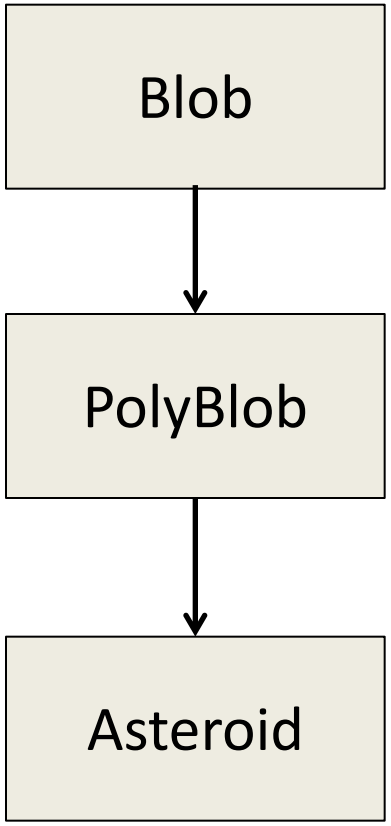
- Set via SandBox instance method:

  setFrame Rate(n);

  - where n = frame rate in frames per second

- Simulation update uses the difference between frame rate and time consumed during update

Blob Inheritance Hierarchy



- Circular object that can move
- Will flow or bounce according to how the sandbox is configured

- Polygon-shaped Blob
- Can also rotate

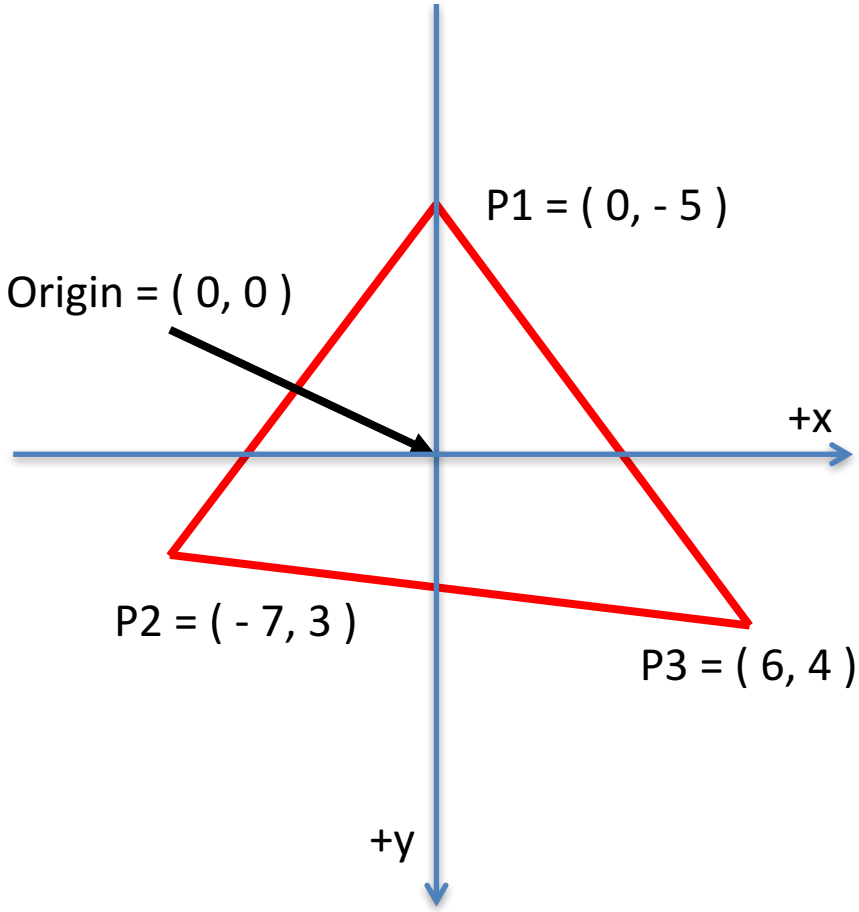
- Custom PolyBlob
- Self-configuring
  - 5 – 9 sides
  - 10 – 30 pixels in diameter

You will create the Asteroid class (extends PolyBlob)

PolyBlob Class

- Extends Blob
- Constructor: PolyBlob(int x, int y, double r)
  - Creates a default diamond-shaped blob at location (x,y)
- Instance methods you will need:
  - setPolygon(Point[] p)
    - points are the vertices of the polygon, relative to PolyBlob center
    - **Point** is a class in the JavaAPI (use theAPI Javadocs to see how to use it)
  - setRate(double r)
    - Sets the rotation rate (> 0 clockwise; < 0 counterclockwise)

Polygon Geometry



- Vertex coordinates are relative to the **origin**
  - i.e., the current location of the PolyBlob on the drawing surface

- To add this polygon to a PolyBlob:

```
private Point[] p = {
        new Point( 0, -5),    ← P1
        new Point( -7, 3 ),   ← P2
        new Point( 6, 4)      ← P3
}
setPolygon( p );
```

an instance method

What the Asteroid Class Must Do

- These requirements summarize (**but do not change**) what is in the assignment

- **Asteroid** class requirements:
  1. Extend the PolyBlob class
  2. Constructor must take only x- and y- ***velocity*** and ***rotation*** data input
     a. Invoke superclass constructor for offscreen initial ***location***
     b. Set the velocity vector based on the inputs
        – Use PolyBlob's setDelta method
        – Inputs should not have zero x- or y- velocity components
     c. Use PolyBlob's setPolygon instance method to define the shape:
        – Polygon must have between 5 and 9 sides (both values inclusive)
        – Vertices can be no closer than 5 pixels from origin
        – Vertices can be no farther than 15 pixels from origin

Asteroids Constructor

- The Asteroids class will have only a constructor (no other methods)
- Constructor must begin with invoking the PolyBlob constructor for flow mode (use x = -100 and y = -100, plus input rotation value)
- It must also explicitly set the velocity vector
- Here's some code you can use:

```
public class Asteroid extends PolyBlob {

   private static final Random random = new Random();

   public Asteroid( int idx, int jdx, double rot ) {

      // Construct a polyblob that starts offscreen,
      // using the input velocity and rotation values
      super( -100, -100, rot );
      super.setDelta(idx, jdx);
```

- Constructor must also determine the shape of the asteroid (use setPolygon)

Setting Polygon Vertices

- Follow this outline to create asteroids that do not have lines that cross:

1. Choose the number of **sides** (random between 5 and 9, both values inclusive
2. For each **vertex**, choose the **distance** from the origin (random between 5 and 15)
   - Note: an asteroid will have the same number of vertices as sides
3. Choose an **angle** for each vertex as follows:
   a. Divide **2π** by the number of sides to create a fixed-size **region** of size **regSize** that will contain each vertex,

      i.e., regSize = **2π** / (number_of_sides)

   b. For each region, choose a random angle using this formula:

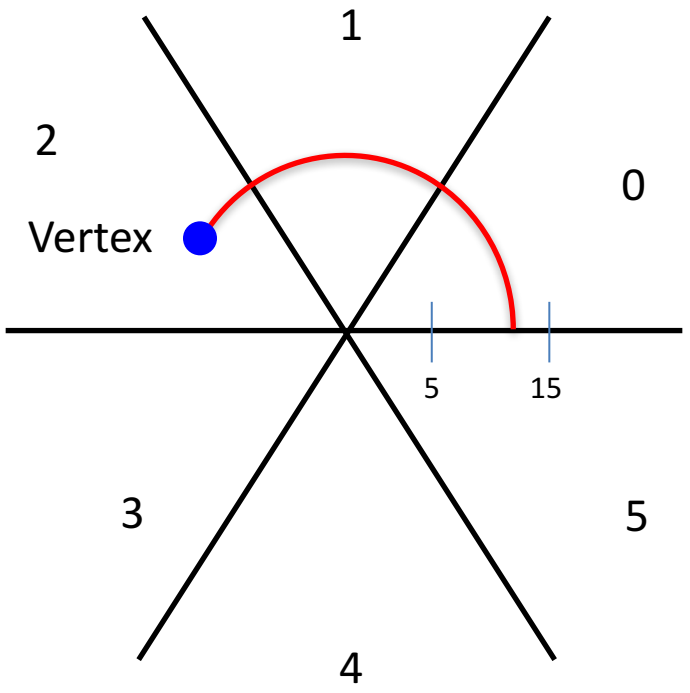
      angle[i] = (i * regSize) + (Math.random() * regSize);
4. For each vertex and associated angle, get the relative x and y coordinates from BlobUtils.rotatePoint(int distance, double angle)
5. Stuff the returned Point into a Point array to use for setPolygon()

Vertex Coordinates

angle[i] = (i * regSize) + (Math.random() * regSize);

Here is an example:

- Suppose we choose (randomly) to have 6 sides for a particular asteroid.
- Then, we will have 6 regions
- Each vertex will be randomly chosen in its own region
- So, we take a point on the x-axis at the appropriate distance from the origin (5 to 15 pixels) …
- … and we rotate it to the required angle …
- …according to equation above (same as Equation 3(b) on the previous slide)



Note: The size of each region is regSize

What the AsteroidField Class Must Do (1)

- These requirements summarize (**but do not change**) what is in the assignment

- **AsteroidField** class requirements:

  1. Must implement the BlobGUI interface
     - public class AsteroidField implements BlobGUI {

  2. Main method – use this code

```
public static void main( String[] args ) {
   new AsteroidField( Integer.parseInt( args[ 0 ] ) );
}
```

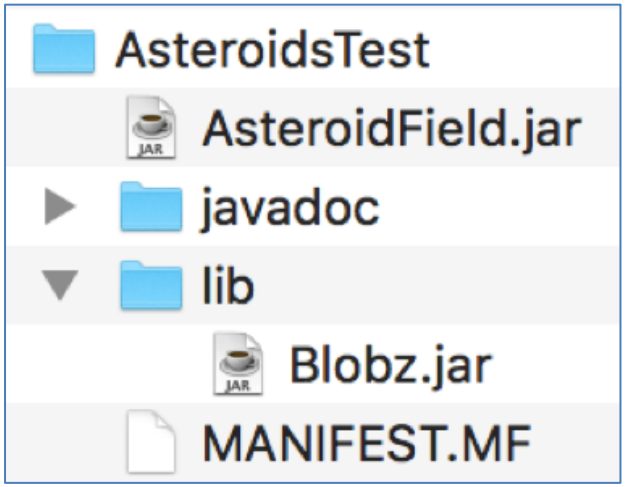
  3. Constructor
     - must take the int input argument and save it to a static int variable
     - Create a sandbox and configure it for flow mode with 15 frames per second
     - Initialize the sandbox using the instance method init(this)

What the AsteroidField Class Must Do (2)

4. public void generate() method (required)
    - Create as many asteroids as specified in the static variable saved by the constructor
    - For each asteroid:
        - Choose X and Y velocity components independently
            - each component a random int from -3 to +3, but not 0
            - Do not allow a zero velocity component for either X or Y
        - Don't need to choose an initial location
            - Asteroid construtor will set initial location to offscreen)
        - Choose a rotation value either +0.1 or -0.1, randomly, with equal probability (use only these 2 values -- no other values are allowed)
        - Create an asteroid with the chosen velocity and rotation values
        - Add the asteroid to the sandbox, either individually or as a collection

Program Export and Testing

- Follow the Test folder setup and Eclipse project setup sections of the assignment carefully
  - they include creating the custom manifest file MANIFEST.MF in your project
- Make sure the program runs correctly within Eclipse
- Export your AsteroidField.jar file into the test folder or copy your exported JAR to the test folder
  - must be executable
  - must use the custom manifest file
  - must contain your source code



Program 3 test folder

- Open a command window and navigate to your Test folder
- To run with 15 asteroids use the command: java –jar AsteroidField.jar 15

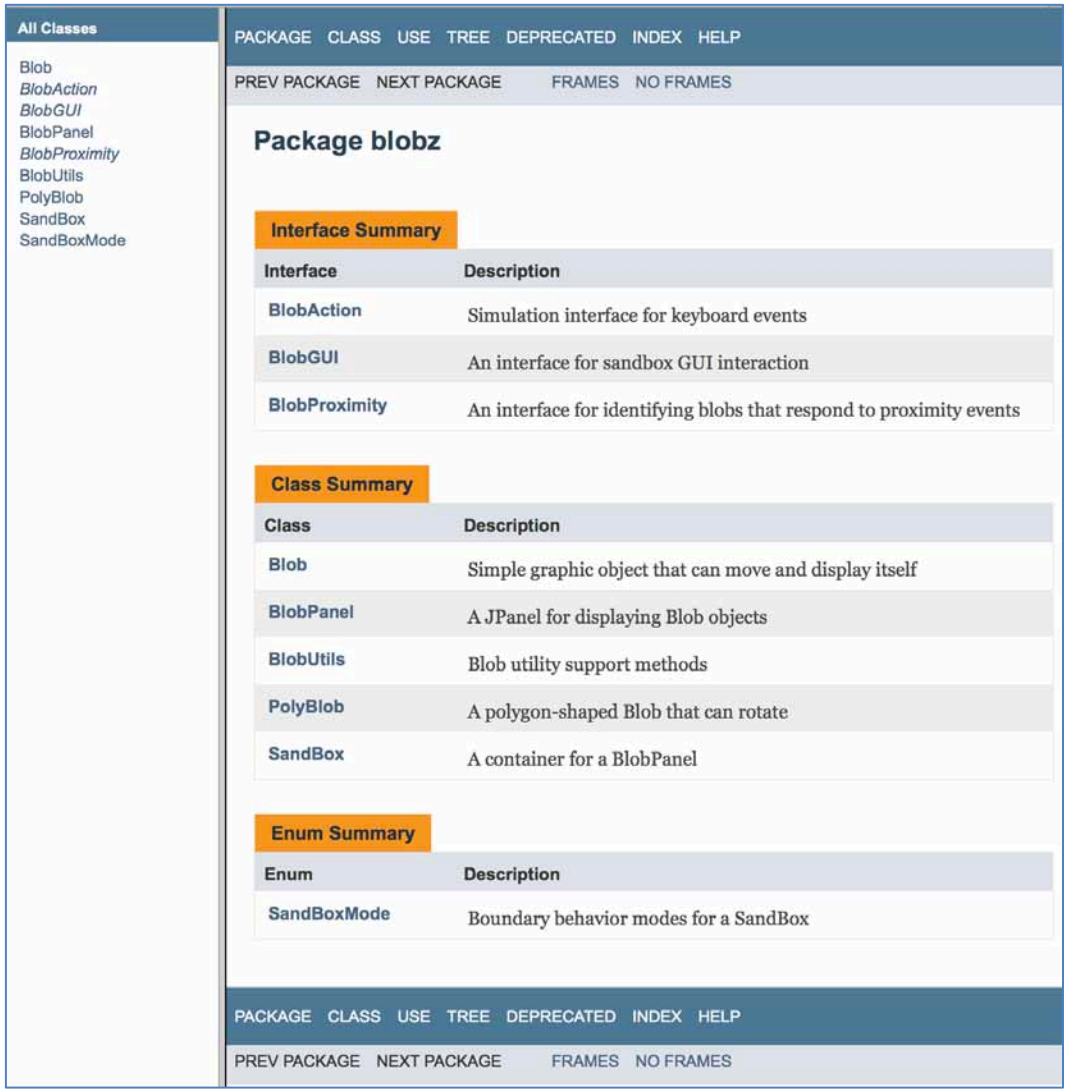
Assignment Logistics

- What to submit: An *executable JAR* named AsteroidField.jar
  - Source code must be in the JAR file you submit
    - 2 source files: Asteroid.java and AsteroidField.java (with author headers)
    - 2 class files: Asteroid.class and AsteroidField.class
  - The JAR file will also contain your custom MANIFEST.MF manifest file
  - Navigate to your test folder and use the command jar –tvf AsteroidField.jar to be sure your JAR file contains all of the required files above
  - Do NOT submit Blobz.jar
  - Blobz.jar and Javadocs are furnished with the assignment
  - Grading rubric is also included with the assignment

Using the Javadocs

- The Blobz.jar contains all classes in the package
- Classes you will need to examine:
  - SandBox
  - PolyBlob
  - Blob
  - BlobUtils
- Other Java API classes you will need
  - Random
  - Point
  - Dimension

demo: index.html



Demo Program

To run this on your own:

- Create a separate project in Eclipse
- Configure the project to use Blobz.jar
- Create a MANIFEST.MF file in the new project that sets the Main-Class attribute to BlobBounce and Class-Path to lib/Blobz.jar
- Run and confirm operation in Eclipse
- Optional: export to the AsteroidsTest folder and run from the command line

```
import blobz.Blob;
import blobz.BlobGUI;
import blobz.SandBox;
import java.awt.Dimension;

public class BlobBounce implements BlobGUI {

   private static SandBox sb;

   public BlobBounce() {
      sb = new SandBox();
      sb.init(this);
   }

   public static void main( String[] args ) {
      new BlobBounce();
   }

   @Override
   public void generate() {
      Dimension dim = sb.getPanelBounds();
      Blob b = new Blob( dim.width/2, dim.height/2, null );
      b.setSize( 30 );
      b.setDelta(3, 1);
      sb.addBlob( b );
   }
}
```

demo: bounce