

# CS-259 Data Structures with Java

Project 2: Breakout v1:

*Dumb Impervious Blocks*



Instructor: **Joel Castellanos**

e-mail: [joel@unm.edu](mailto:joel@unm.edu)

web: <http://cs.unm.edu/~joel/>

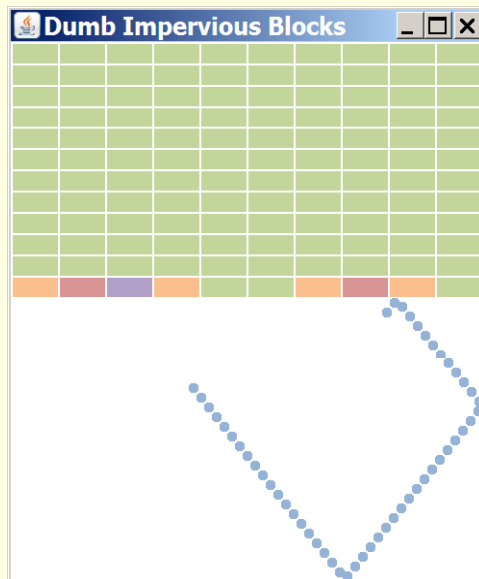
Course website:

<http://cs.unm.edu/~joel/cs259/>



10/7/2009

## Breakout: Project 2, Milestone 1



Due Tuesday at Midnight

- Ball always bounces with the angle of incidence equal to the angle of reflection.
- Ball always bounces when it meets any surface.
- Blocks change color when struck starting with sage and rotating through pumpkin, sand red, wisteria and back to sage.
- The ball leaves a history trail 50 balls long.

2

## Project 2, Milestone 1: Grading Rubric

- 3 Points: Adheres to CS-259 coding standard.
- 3 Points: Adheres the given class and field names.
- 2 Points: The correct number of blocks are drawn in the correct locations and the correct size.
- 2 Points: Frame resizing is handled correctly.
- 3 Points: The ball bounces correctly.
- 3 Points: Blocks change color when struck starting rotating through 4 colors.
- 2 Points: The ball leaves a history trail 50 balls long which is correctly updated.
- 2 Points: A correctly calculated "Average Time Between Turns" is output when the window is closed and this number is no greater 50 ms/turn.

3

## Dumb Impervious Blocks: Class Structure

You program must consist of two files. It may contain other non-public classes:

### ■ Breakout.java

```
public class Breakout extends JFrame implements  
    ComponentListener, ActionListener,  
    WindowListener
```

This JFrame instantiates BreakoutDraw, handles resizing (ComponentListener calls BreakoutDraw.setBounds), handles timer events (ActionListener calls BreakoutDraw.nextTurn()) and window closing events (WindowListener).

### ■ BreakoutDraw.java

```
public class BreakoutDraw extends JPanel
```

This JPanel does all the graphics, figures out ball collisions, and keeps track of how many times each brick is hit.

4

## Breakout JFrame: Size and Resize

- Whenever the user resizes the JFrame, your code must resize the JPanel, and restart the “game” with appropriately sized boxes. **Caution:** Do not call `addComponentListener(this)` to the JFrame’s constructor until *after* the constructor has finished instantiating `BreakoutDraw`. The *very last* thing the constructor should do is start the timer.
- Breakout must contain the fields:

```
private static final int TIMER_DELAY = 20; //millisec
public static final int MIN_WIDTH   = 280; //pixels
public static final int MIN_HEIGHT  = 425; //pixels
```

If the user resizes the JFrame smaller than these minimum dimensions, then Breakout must set the JFrame’s size to equal these dimensions.

5

## Breakout JFrame: Timer and Time/Turn

- “Dumb Impervious Blocks”, (aka Breakout: milestone 1) must run in a single thread.
- The JFrame must include a `javax.swing.Timer` set to fire every 20 milliseconds.
- When the user clicks the JFrame’s close box:
  - Breakout must calculate and display in the Java console “Average Time Between Turns” defined as the total elapsed milliseconds from the JFrame’s constructor to the user click in the close box divided by the number of times the timer fired.
  - This number must be no greater than 50 milliseconds per turn when run on xbox.cs.unm.edu (Quad-Core AMD Opteron x86, 64-bit system running Linux).
  - Then, the program must exit.

6

## BreakoutDraw JPanel: Ball Fields

Each of these fields must be defined:

```
private static final Color BALL_COLOR =
    new Color(149,179,215); //denim

private final int BALL_DIAMETER = 11; //pixels
private final int BALL_HISTORY_COUNT = 50;

private Point[] ballHistory =
    new Point[BALL_HISTORY_COUNT];
//Instantiate all 50 of these java.awt.Point objects ONLY ONCE.
//Do this in BreakoutDraw's constructor with bogus values:
//(-100,-100). During each turn, update the values of
//ballHistory[historyIdx].x & ballHistory[historyIdx].y
//from the deleted position to the new position.

private int historyIdx = 0; //index into ballHistory of the
//oldest ball, and index where the newest ball is added.
```

7

## Initial Ball Position and Velocity

At the start and after each frame resize, the ball should be set to an initial position and velocity:

- Initial position of the ball must be a random position along the bottom edge of the panel.
- The ball's initial vertical velocity must be +10 pixels/turn. Throughout the "game" the ball's vertical velocity should always be  $\pm 10$  pixels/turn.
- The ball's initial horizontal velocity, must be a random, between [-10, 10], but not 0 pixels/turn.

8

## BreakoutDraw JPanel: Block Fields

You may choose your own colors, but each of these fields must be defined

```
private final int BLOCK_GAP = 2;      //Pixels
private final int BLOCK_ROWS = 12;
private final int BLOCK_COLUMNS = 10;

private final int BLOCK_HEIGHT = 20; //Pixels
private int blockWidth, blockBoarder;

private static final Color[] BLOCK_COLORS =
{ new Color(194,214,155), //sage
  new Color(250,191,141), //pumpkin
  new Color(217,149,148), //sand red
  new Color(178,161,199) //wisteria
};
private int[][] blocks = new
    int[BLOCK_COLUMNS][BLOCK_ROWS];
    //Initialize all values in this array to 1, and increment when a block's
    //value with it is hit.
```

9

## Remove Oldest From End of List: v1

```
public static void main(String[] args)
{ int[] hist = {-100, -100, -100, -100};
  int ballX = 0;
  for(int i=0; i<15; i++)
  { ballX += 5;
    System.out.println("Erase " +
      hist[3]+ ", Draw " + ballX);
    hist[3] = hist[2];
    hist[2] = hist[1];
    hist[1] = hist[0];
    hist[0] = ballX;
  }
}
```

```
Erase -100, Draw 5
Erase -100, Draw 10
Erase -100, Draw 15
Erase -100, Draw 20
Erase 5, Draw 25
Erase 10, Draw 30
Erase 15, Draw 35
Erase 20, Draw 40
Erase 25, Draw 45
Erase 30, Draw 50
Erase 35, Draw 55
Erase 40, Draw 60
Erase 45, Draw 65
Erase 50, Draw 70
Erase 55, Draw 75
```

This works, but is slow, especially if the `hist` array has many elements. Each turn, each element is moved back one space, to delete the oldest from the back and make room for the new location at the front. This is a  $O(n)$  algorithm.

10

## Remove Oldest From End of List: v2

```
public static void main(String[] args)
{
    int[] hist = {-100, -100, -100, -100};
    int ballX = 0;
    int histIdx = 0; //index of oldest.
    for(int i=0; i<15; i++)
    {
        ballX += 5;
        System.out.println(histIdx +
            ") Erase " + hist[histIdx]+
            ", Draw " + ballX);
        hist[histIdx] = ballX;
        histIdx = (histIdx+1) % hist.length;
    }
}
```

```
0) Erase -100, Draw 5
1) Erase -100, Draw 10
2) Erase -100, Draw 15
3) Erase -100, Draw 20
0) Erase 5, Draw 25
1) Erase 10, Draw 30
2) Erase 15, Draw 35
3) Erase 20, Draw 40
0) Erase 25, Draw 45
1) Erase 30, Draw 50
2) Erase 35, Draw 55
3) Erase 40, Draw 60
0) Erase 45, Draw 65
1) Erase 50, Draw 70
2) Erase 55, Draw 75
```

`histIdx` is the index of the oldest value, the value that needs to be removed. Its value is outputted, then the new value is added in its place. Finally, `histIdx` is updated to index next oldest value. This is a **constant time** algorithm.

11

## General Hints on Dumb Impervious Blocks

- Erase the ball by first switching the draw color to the background color. Then draw the circle you want to erase. Remember to switch the draw color back to the ball color before drawing the new ball.
- When something is not working, add `System.out.println()` to figure out:
  1. Is the code you think is not working is even getting executed?
  2. Is the control flow what you expect it to be?
  3. Do intermediate values have values that you expect them to have?
- Try setting the starting size of the frame to the smallest allowed size. With a small frame, the ball bounces sooner. It is easier to look at printed debug output of every step if there are a small number of steps.
- Minimize use of `new`.

12

## Dumb Impervious Blocks Design Decisions

- Why create final fields such as  
`int BALL_DIAMETER?`
- Why not make `BALL_DIAMETER` static?
- Why not make `ballHistory` static?
- Why make the JFrame listen to timer events rather than the draw panel?

13

## Quiz ▲: What Happens when Executed?

```
1) import java.awt.Point;
2) public class HelloWorld
3) { private Point p;
4)   public HelloWorld()
5)   { Point p = new Point(1,2);
6)   }
7)   public static double vecLen(Point p)
8)   { return Math.sqrt(p.x*p.x + p.y*p.y);
9)   }
10)  public static void main(String[] args)
11)  { HelloWorld bob = new HelloWorld();
12)    bob.p.x = 3;
13)    bob.p.y = 4;
14)    System.out.println(vecLen(bob.p));
15)  }
16)}
```

- a) Null Pointer in line 12.
- b) Null Pointer in line 8.
- c) Cannot make static reference to non-static field p in line 12.
- d) Cannot make static reference to non-static field p in line 8.
- e) Output: 5.0

14

## Fixing the Error in Quiz ▲

```
1) import java.awt.Point;
2) public class HelloWorld
3) { private Point p;
4)   public HelloWorld()
5)   { //Point p = new Point(1,2);
6)     p = new Point(1,2);
7)   }
8)   public static double vecLen(Point p)
9)   { return Math.sqrt(p.x*p.x + p.y*p.y);
10)  }
11)  public static void main(String[] args)
12)  { HelloWorld bob = new HelloWorld();
13)    bob.p.x = 3;
14)    bob.p.y = 4;
15)    System.out.println(vecLen(bob.p));
16)  }
17)}
```

Now, the pointer to p in the constructor is the same as the class instance variable p.

Output: 5.0

15

## Quiz ✦: What Happens when Executed?

```
1) import java.awt.Point;
2) public class HelloWorld
3) { private Point[] p = new Point[2];
4)
5)   public static void main(String[] args)
6)   { HelloWorld sue = new HelloWorld();
7)     sue.p[0].x = 2;
8)     sue.p[0].y = 3;
9)     sue.p[1].x = 5;
10)    sue.p[1].y = 7;
11)    System.out.println(
12)      sue.p[0].x * sue.p[1].x);
13)  }
14)}
```

- a) Constructor for HelloWorld cannot be found in line 6.
- b) Null Pointer in line 7.
- c) Array index out of bounds in line 7.
- d) Array index out of bounds in line 9.
- e) Output: 10

16

## Fixing Error in Quiz ✦

```
1) import java.awt.Point;
2) public class HelloWorld
3) { private Point[] p = new Point[2];
4)
5)     public static void main(String[] args)
6)     { HelloWorld sue = new HelloWorld();
7)       sue.p[0] = new Point(2,3);
8)       sue.p[1] = new Point();
9)
10)      //sue.p[0].x = 2;
11)      //sue.p[0].y = 3;
12)      sue.p[1].x = 5;
13)      sue.p[1].y = 7;
14)      System.out.println(
15)          sue.p[0].x * sue.p[1].x);
16) }
```

Allocates space for a pointer to an array and allocates space for 2 pointers to Point objects.

Instantiates the two Point objects and sets p[0] and p[1] to point to these new objects.

Leaving this code uncommented would not change the output.

Output: 10

17

## Quiz: What Could **Not** Cause this Error?

The following line of code causes a null pointer exception.

```
mypoint[3].x = 3;
```

Which of the following could NOT be the cause of the error:

- a) The pointer `mypoint` was declared but not instantiated.
- b) The pointer `mypoint[3]` was not instantiated.
- c) A different pointer in the same program with the name `mypoint` may have been instantiated.
- d) The pointer `mypoint` or the pointer `mypoint[3]` may both have been instantiated, but later one of them was reassigned to a value of null.
- e) The pointer `mypoint` was instantiated with an array size less than or equal to 3.

18

## Quiz: What are the Possible Causes

The following line of code causes a null pointer exception.

```
fu.p[7].x = fu.c.y;
```

From this, it can be concluded that:

- a) `fu` is a null pointer.
- b) Either `p[7]` or `c` is a null pointer.
- c) Either `fu`, `p` or `c` is a null pointer.
- d) Either `fu`, `p`, `p[7]` or `c` is a null pointer.
- e) Either `fu`, `p`, `p[7]`, `x`, `c` or `y` is a null pointer.