Review Problems for CS111 EXAM 1

The first exam is coming up. The exam is open notes: you may refer to any handouts, your notes, and your assignments, but you may not refer to anyone else's materials nor any materials from previous semesters of CS111. . You may not use a computer during the exam.

This handout includes some problems adapted from previous exams that you may find helpful in studying for the exam. These problems are not necessarily indicative of the kinds of problems you may be given on your exam or the length of your exam, but they do cover much of the material you are expected to know for the exam.

Solutions to these problems will be posted. You will learn more if you refrain from consulting them until you have solved the problems on your own.

Problem 1: Buggle World Execution

Consider the two Java classes in Fig. 1.

```
public class DoItWorld extends BuggleWorld
{
     public void run ()
     {
          DoItBuggle dewey = new DoItBuggle();         // run statement 1
          int n = 5;                                   // run statement 2
          dewey.setPosition(new Location(n, n - 2));   // run statement 3 *
          dewey.brushUp();                             // run statement 4
          dewey.doit(Color.green, n - 1);              // run statement 5 *
          dewey.doit(Color.blue, n + 1);               // run statement 6 *
          dewey.forward();                             // run statement 7
          dewey.brushDown();                           // run statement 8
          dewey.forward(3);                            // run statement 9 *
     }
}

class DoItBuggle extends Buggle

{
     public void doit (Color c, int n)
     {
          Color oldColor = this.getColor();
          this.setColor(c);
          this.forward(n);
          this.brushDo‘wn();
          this.backward(n-2);
          this.brushUp();
          this.backward(2);
          this.left();
          this.setColor(oldColor);
     }
}
```

Figure 1: Two Java classes.

Suppose that the `run()` method is invoked on an instance of `DoItWorld` which has a 10×10 grid of cells. In the four grids on the following page, show the state of the grid directly *after* the execution of each of the statements in the `run()` method body marked with a `*`.

In each grid, you should show the following:

1. Draw buggle `dewey` as a triangle pointing in the direction that the buggle is facing.
2. Indicate the current color of the buggle by putting the *first letter* of the color name inside the triangle (e.g. `B` for blue, `G` for green, etc.).
3. Indicate the color of each non-white grid cell by putting the *first letter* of the color name inside the cell (e.g. `B` for blue, `G` for green, etc.).

`DoItWorld` grid after the
execution of `run()` statement 3

	1	2	3	4	5	6	7	8	9	10
10										
9										
8										
7										
6										
5										
4										
3										
2										
1										

`DoItWorld` grid after the
execution of `run()` statement 5

	1	2	3	4	5	6	7	8	9	10
10										
9										
8										
7										
6										
5										
4										
3										
2										
1										

`DoItWorld` grid after the
execution of `run()` statement 6

	1	2	3	4	5	6	7	8	9	10
10										
9										
8										
7										
6										
5										
4										
3										
2										
1										

`DoItWorld` grid after the
execution of `run()` statement 9

	1	2	3	4	5	6	7	8	9	10
10										
9										
8										
7										
6										
5										
4										
3										
2										
1										

Problem 2: Debugging

The class declarations in Fig. 2 contain (at least) **10 errors** (syntax errors and type errors).

```
public class ExamBuggleWorld extends BuggleWorld      // line 1
{                                                     // line 2
     public void run ()                               // line 3
     {                                                // line 4
          Color c = Color.cyan();                     // line 5
          int n = 4                                   // line 6
          ExamBuggle emma = ExamBuggle();             // line 7
          emma.mystery1(c, n);                        // line 8
          emma.mystery1(3, Color.red);                // line 9
          boolean answer = emma.mystery2();           // line 10
          this.mystery3();                            // line 11
     }                                                // line 12
}                                                     // line 13
                                                      // line 14
class ExamBuggle extends Buggle                       // line 15
{                                                     // line 16
     public void mystery1(Color c, int n1)            // line 17
     {                                                // line 18
          n2 = n1 + 1;                                // line 19
          this.setColor(Color.c);                     // line 20
          forward(n2);                                // line 21
          this.dropBagel();                           // line 22
                                                      // line 23
     public boolean mystery2()                        // line 24
     {                                                // line 25
          this.isOverBagel();                         // line 26
     }                                                // line 27
                                                      // line 28
     public mystery3()                                // line 29
     {                                                // line 30
          this.dropBagel();                           // line 31
    }                                                 // line 32
}                                                     // line 33
```

Figure 2:

In the table on the next page, for each of 10 errors in different lines of the above program give:

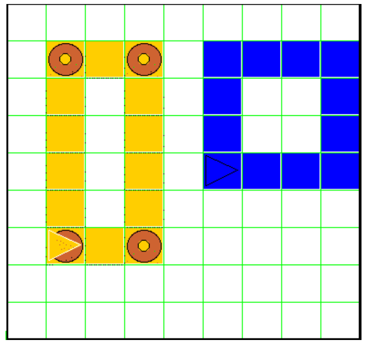
1. the line number of the error,
2. a *brief* description of the error, and
3. a corrected version of the line (i.e., with the error fixed).

You may list the errors in *any* order. You do *not* have to list them in the order in which they occur in the program.

Error #	**Line #**	**Brief description of error**	**Corrected line**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Problem 3: Buggle Methods

A class of Buggles enjoys doing window treatments. They call themselves `Windowers`. In `WindowWorld`, `wendy` and `winifred` each do a window treatment:



```
public class WindowWorld extends BuggleWorld {

    public void run()
    {
        Windower wendy = new Windower();          // line  1
        Windower winifred = new Windower();       // line  2

        wendy.setPosition(new Location(2, 3));    // line  3
        wendy.setColor(Color.orange);             // line  4
        wendy.forward(2);                         // line  5
        wendy.dropBagel();                        // line  6
        wendy.left();                             // line  7
        wendy.forward(5);                         // line  8
        wendy.dropBagel();                        // line  9
        wendy.left();                             // line 10
        wendy.forward(2);                         // line 11
        wendy.dropBagel();                        // line 12
        wendy.left();                             // line 13
        wendy.forward(5);                         // line 14
        wendy.dropBagel();                        // line 15
        wendy.left();                             // line 16

        winifred.setPosition(new Location(6, 5)); // line 17
        winifred.setColor(Color.blue);            // line 18
        winifred.forward(3);                      // line 19
        winifred.left();                          // line 20
        winifred.forward(3);                      // line 21
        winifred.left();                          // line 22
        winifred.forward(3);                      // line 23
        winifred.left();                          // line 24
        winifred.forward(3);                      // line 25
        winifred.left();                          // line 26
     }
}
```

a Assume there is a `Windower` class, which extends `Buggle`. Capture the repeated pattern of code in the `run()` method above by creating a single method named `decorateWindow()` that produces the same window treatments that `wendy` and `winifred` created above in lines 3–16 and 17–26. You may assume that your `decorateWindow()` method is being defined in the `Windower` class. Your method should take 5 parameters that provide the following information:

- a location specifying the position of the window's lower left corner,
- color of the window,
- width of the window (number of cells),
- height of the window (number of cells),
- and a boolean value that says whether the window corners should be decorated with bagels.

Assume an infinite grid, i.e., you don't have to worry about whether your windows will fit in the `BuggleWorld` grid.

b Below, write the two invocations of your `decorateWindow()` method that will replace lines 3–16 and lines 17–26 in the `run()` method:

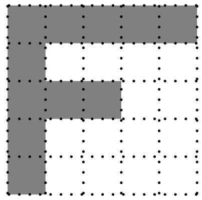
- *invocation to replace lines 3–16:*

- *invocation to replace lines 17–26:*

Problem 4: Writing Methods

Suppose that `LetterWorld` is a subclass of `PictureWorld` that supplies you with a method named `f()` with the following contract:

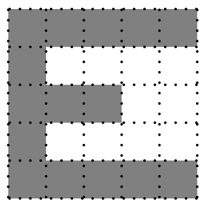
> `public Picture f (Color c)`
> Returns a picture of the letter "F" in color `c`, as shown below.



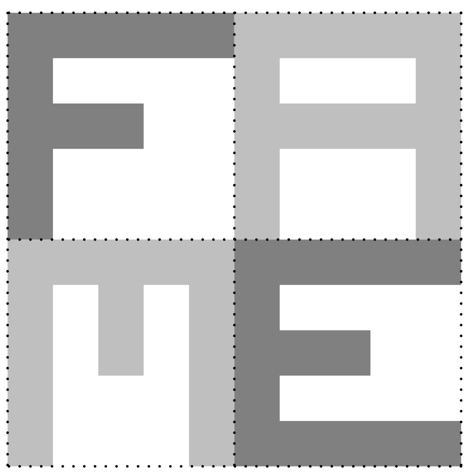
> The dotted lines indicate the boundaries of the unit square, and are <u>not</u> part of the picture. The letter is a solid color `c` and does <u>not</u> have any boundary line drawn in a separate color.

On the next page your task is to write <u>two</u> methods:

1. A method named `e()` that takes a single color parameter and returns the following picture of the letter "E" in that color.



2. A method named `fame()` that takes two color parameters and returns the following picture:



The "F" and "E" have the color of the first parameter, while the "A" and "M" have the color of the second parameter.

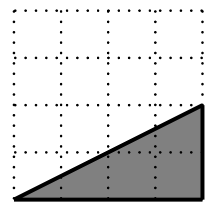
You may assume that both methods are defined in the `LetterWorld` class, and so may use the `f()` method in addition to the methods in the `PictureWorld` contract (e.g., `clockwise90()`, `flipDiagonally()`, `above()`, etc.). You may assume that the `fourPics()` and `fourSame()` methods defined in class and on the problem sets are also available. Your `fame()` method may use your `e()` method, which you may assume works correctly (even if your definition of `e()` is actually incorrect or missing).

Put your definition of the `e()` *method here.*

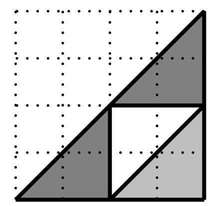
Put your definition of the `fame()` *method here.*

Problem 5: A Picture Method

Suppose that `TriangleWorld` is a subclass of `PictureWorld` that supplies you with a method named `wedge` with the following contract:

> `public Picture wedge (Color w)`
> Returns a picture of a black-bordered wedge of color `w`, as shown to the right. (The dotted lines indicate the grid of the unit square, and are *not* part of the picture.)
>
> 

At the bottom of this page, your task is to write a method named `threeTriangles()` that takes two color parameters and returns the picture to the right, which contains three black-bordered isosceles triangles: the lower-left and upper-right ones with a color specified by the first parameter and the lower-right one with the color specified by the second parameter. You may assume that the `threeTriangles()` method is defined within the `TriangleWorld` class, and so may use the `wedge` method in addition to the methods in the `PictureWorld` contract (e.g., `empty`, `clockwise90`, `flipDiagonally`, `beside`, etc.). You must *not* use the `Poly` class for constructing polygons. You must *not* use `fourPics()` or any methods (other than `wedge()`) not defined in the `PictureWorld` contract.



Partial credit will be awarded for writing a correct skeleton of the `threeTriangles()` method and for getting *some* of the triangles in the correct positions with the correct colors.

Hints: (1) Each of the isosceles triangles should be an appropriately transformed wedge picture; (2) You may define local variables of type `Picture` within your method; (3) *Think carefully* — the problem is trickier than it might first seem.

Put your definition of the `threeTriangles` *method here.*

Problem 6: Booleans and Conditionals

a Bud Lojack has written the following method in a rather unclear programming style:

```
public boolean isColdAndHeadingNorth ()
{
    if (getColor().equals(Color.blue)) {
        if (getHeading().equals(Direction.NORTH)) {
          return true;
        } else {
          return false;
        }
    } else if (!getColor().equals(Color.blue)) {
        return false;
    } else if (!getHeading().equals(Direction.NORTH)) {
        return false;
    } else {
        return true;
    }
}
```

Rewrite Bud's method in a much clearer style.

b Define a `Buggle` method named `isBoxedIn()` that has no parameters and returns `true` if a buggle is in a cell surrounded by walls on all four sides, and otherwise returns `false`. The final state of the buggle when `isBoxedIn()` returns should be the same as the state of the buggle when `isBoxedIn()` is invoked. You may not use recursion or iteration in your solution, but you may define auxiliary methods if you wish.

Problem 7: Java Execution Model in BuggleWorld

Consider the following two class definitions:

```
public class RelayRaceWorld extends BuggleWorld
{
    public void run()
    {
        RelayRunner r1 = new RelayRunner();
        RelayRunner r2 = new RelayRunner();
        RelayRunner r3 = new RelayRunner();

        r2.setColor(Color.green);
        r3.setColor(Color.blue);
        r1.firstLeg(3, r2, r3);
    }
}

class RelayRunner extends Buggle
{
    public void firstLeg(int length, RelayRunner next, RelayRunner last)
    {
        this.forward(length);
        next.setPosition(this.getPosition());
        next.secondLeg(length, last);
    }

    public void secondLeg(int length, RelayRunner next)
    {
        this.forward(length);
        next.setPosition(this.getPosition());
        next.thirdLeg(length);
    }

    public void thirdLeg(int length)
    {
        this.forward(length);
    }
}
```

The Java Execution Model diagram on the next page shows the state of the program after evaluating the first line of the `run()` method. Show the diagram after the completion of the `run()` method. Include in Object Land all instances of the `RelayRunner` class that are created during the execution. Also include all execution frames opened during the execution of the `run()` method. You may abbreviate references to instances of the `Location`, `Direction`, and `Color` classes as ovals surrounding appropriate identifying information (as shown in the JEM skeleton).

Object Land

RelayRaceWorld RRW

[redacted]

RelayRunner RR1

`position`	1,1
`heading`	EAST
`color`	red
`brushDown?`	true

Execution Land

`RRW.run()`

`this` RRW `r1` RR1

`RelayRunner r1 = ~~new RelayRunner()~~ RR1;`

`RelayRunner r2 = new RelayRunner();`

`RelayRunner r3 = new RelayRunner();`

`r2.setColor(Color.green);`

`r3.setColor(Color.blue);`

`r1.firstLeg(3,r2,r3);`

Problem 8: Invocation Trees

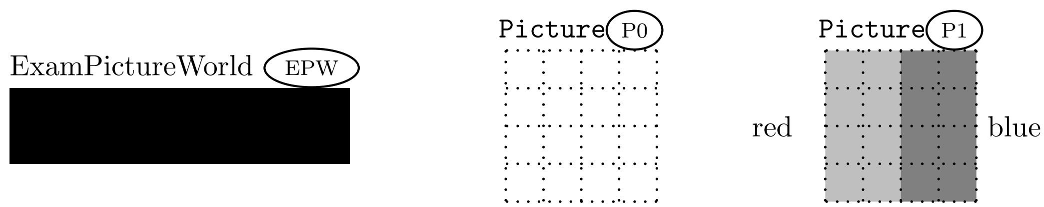
```
public class ExamPictureWorld extends PictureWorld {

    public Picture meth1 (Picture a) {
        Picture b = beside(a, empty());
        return overlay(meth2(b), b);
    }

    public Picture meth2 (Picture c) {
        return clockwise90(above(c, empty(), 0.75));
    }
}
```

Figure 3: A sublcass of `PictureWorld`.

Consider the subclass of `PictureWorld` shown in Fig. 3. Supose that: EPW is an instance of `ExamPictureWorld`, P0 is a `Picture` instance denoting the empty picture, P1 is a `Picture` instance denoting the rightmost picture below:



The dashed grid lines are <u>not</u> part of the pictures. They indicate coordinates within pictures. The colors names are <u>not</u> part of picture P1. They indicate the color of the two rectangles. Each of the two rectangles is a solid color <u>without</u> any separately colored border.

On the next page, you are to draw an invocation tree that models the instance method invocation EPW`.meth1(`P1`)`. In the area labeled **Execution Land**, you should draw an invocation tree that contains the following eight nodes, arranged appropriately into a tree. You should use each node exactly once.

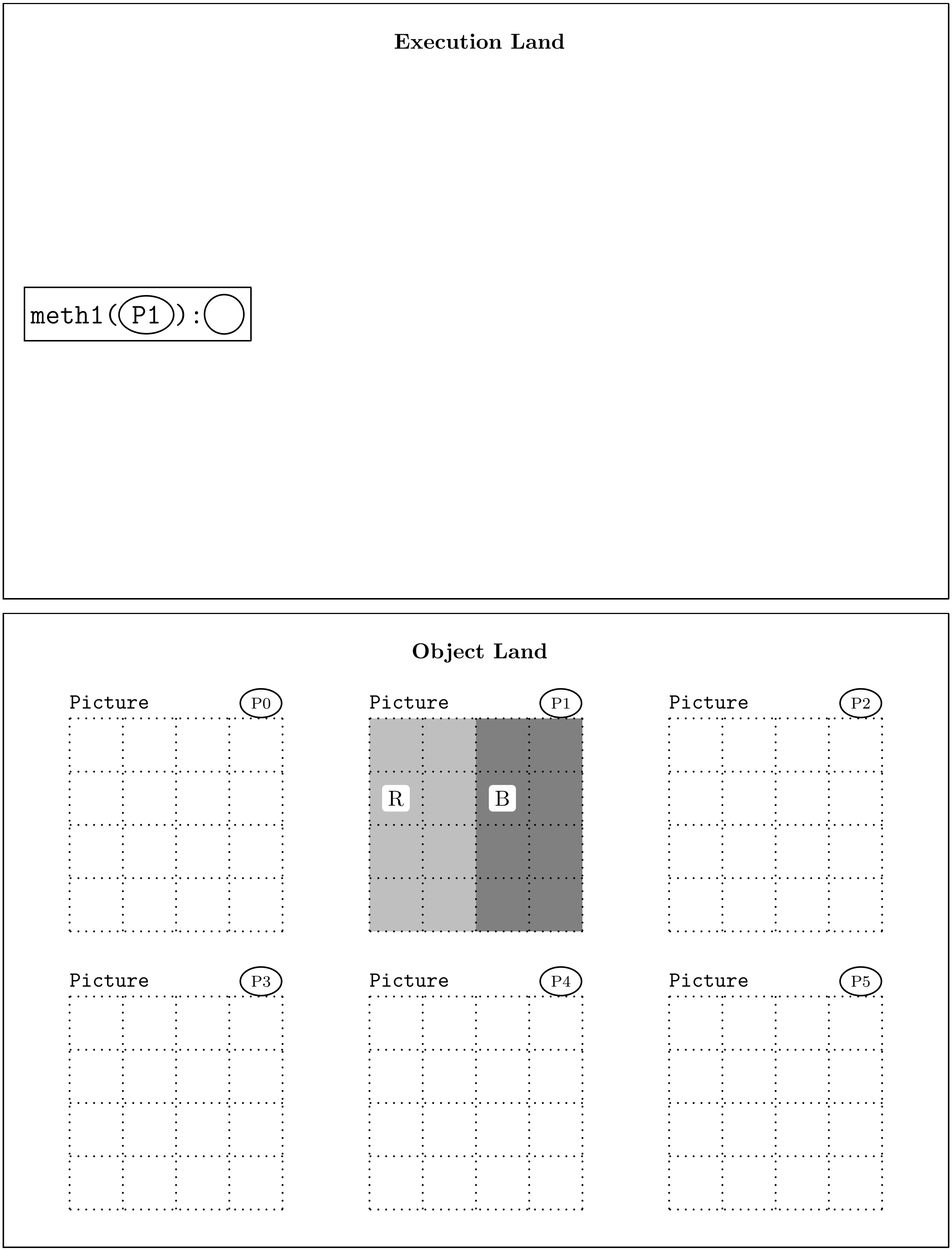
`above(◯,◯,◯):◯` `clockwise90(◯):◯` `empty():◯` `meth2(◯):◯`

`beside(◯,◯):◯` `empty():◯` `meth1(◯):◯` `overlay(◯,◯):◯`

The empty circles in the nodes are skeletons for object references that you should fill in with one of the labels P0, P1, P2, P3, P4, or P5 to refer to the appropriate Picture instance in Object Land (see below). A circle enclosed by parentheses is a reference to an actual argument of the method invocation. A circle appearing after a colon is a reference to the result of the method invocation. The root of the invocation tree is the `meth1()` node, which has already been drawn for you, and whose actual argument has been filled in (you need to fill in its result).

In the area labeled **Object Land** are the skeletons for the six `Picture` instances that are used during the execution. The pictures labeled P0 and P1 have already been drawn for you; you should draw the pictures for P2, P3, P4, and P5. In each picture, you should label red areas with the letter `R` and blue areas with the letter `B`. All other areas are presumed to be white.

(Note: for simplicity, the receiver object EPW for each of the method invocations has been omitted. This instance has also been omitted from Object Land.)


Execution Land
meth1(P1):
Object Land
Picture P0
Picture P1
R
B
Picture P2
Picture P3
Picture P4
Picture P5

Problem 9: Applications and Class Methods

Below, write from scratch a complete application named `MaxOps`. The `MaxOps` class should contain the following four class methods:

- A class method named `max2` that takes two integer arguments and returns the larger of the two. E.g., `MaxOps.max2(17,23)` and `MaxOps.max2(23,17)` should both return `23`. Note: do *not* use `Math.max` in your definition of `max2`. Instead, use a conditional statement.

- A class method named `max8` that takes eight integer arguments and returns the largest of the eight. E.g. `MaxOps.max8(23, 17, -273, 4, 37, 42, 0, -40)` should return `42`.

- A class method named `testMax8` that takes eight integer arguments, and displays all eight integers along with their maximum. E.g. `MaxOps.testMax8(23, 17, -273, 4, 37, 42, 0, -40)` should display `The max of 23, 17, -273, 4, 37, 42, 0, and -40 is 42`.

- A class method named `main` that is the entry point to the application. Invoking the application (via `java MaxOps`) should invoke `testMax8` on the integers 23, 17, -273, 4, 37, 42, 0, and -40.

Put your definition of the `MaxOps` *class here.*