CIS 1200 Final Exam December 22, 2022

Benjamin C. Pierce and Swapneel Sheth, instructors

SOLUTIONS

1. **OCaml and Java Concepts** (20 points total)

   Indicate whether the following statements are true or false.

   (a) True □ False ☒

   ```
   let rec iterate (f: 'a -> 'a option) (zero: 'a) (count: int) : 'a =
     begin match f zero with
     | None -> zero
     | Some one -> let res = iterate f one (count + 1) in
                   print_endline (string_of_int count);
                   res
     end
   ```

   The function shown above is tail recursive.

   (b) True ☒ False □

   In OCaml, if a given **sig** has five methods defined, it is possible for the **struct** that implements it to have more than five methods.

   (c) True □ False ☒

   In the OCaml ASM, stack bindings are mutable by default whereas in the Java ASM, they are immutable by default.

   (d) True □ False ☒

   The advantage of enforcing invariants like the Binary Search Tree invariant is that they eliminate the need for testing (because functions like `insert` and `lookup` are guaranteed to be correct).

   (e) True □ False ☒

   In OCaml, every mutable reference could refer to `None`.

   (f) True ☒ False □

   In our OCaml singly-linked queue implementations, one advantage over the in-built `list` is that we can efficiently add items to the end of the queue.

(g) True ☒ False □

In Java, whenever you implement the `Comparable` interface, you should also override the `equals` method compatibly.

(h) True □ False ☒

In Java, every subclass class must call the superclass constructor explicitly.

(i) True □ False ☒

In Java, every `Exception` must either be caught (with a **`try-catch`** block) or declared (via **`throws`**) in the method signature.

(j) True ☒ False □

In Java, the static type of a variable can be the same as the dynamic class of the object the variable refers to.

2. **OCaml: Binary Search Trees and Higher Order Functions** (14 points total)

Recall the definitions of generic `transform` function for lists and of generic binary trees, which are given in Appendix A.

(a) (2 points) Which of the trees below **satisfy** the binary search tree invariant? (Mark all that apply.)

```
3              7            4             3
 \            / \          / \           / \
  7          5   3        2   5         2   4
   \                       \           / \   \
    9                       3         1   4   8
```

☒ Tree (a) □ Tree (b) ☒ Tree (c) □ Tree (d)

Valid trees are (a) and (c). (b) has 3 to the right of 7, and (d) has 4 to the left of 3 (4 is also repeated twice).

Grading scheme: +0.5 points per correct answer above.

(b) (3 points) Consider the following modification to the `transform` function that now takes in a tree as input.

```
let rec transform_tree (f: 'a -> 'b) (t: 'a tree) : 'b tree =
  begin match t with
  | Empty -> Empty
  | Node(lt, x, rt) -> Node(transform_tree f lt, f x,
     transform_tree f rt)
  end
```

If the following tree `t` is provided as input to the code shown below...

```
t =      4
        / \
       2   6
          / \
         5   8
```

... what will be the resulting output tree `t1`? Draw it below.

```
let t1 = transform_tree (fun x -> 3 * x) t
```

```
t1 =         12
            / \
           6  18
              / \
             15  24
```

Grading scheme: +0.5 attempted the problem OR +2 partially correct OR +3 completely correct

(c) (3 points) Does the `transform_tree` function preserve BST invariants? That is, if the input to the function is a BST and **any** function `f` of type `int->int`, will it *always* return a valid BST as output? (Choose one.)

□ Yes. If you chose yes, explain why.

☒ No. If you chose no, provide an example of a function `f` as input using the tree `t`. (For the function, you can choose to write code or describe it in words. If it's the latter, please be as precise and accurate as possible.)

Using the tree `t` above, the following function will violate the BST invariants.

```
transform_tree (fun x -> 10) t
```

Grading scheme: +1 Selected "no" AND (+1 for partially correct function OR +2 completely correct function)

(d) (3 points) Consider the following higher order function that takes in a tree as input.

```
let rec mystery (f: 'a -> bool) (t: 'a tree) : 'a tree =
  begin match t with
  | Empty -> Empty
  | Node(lt, x, rt) -> if f x
                       then Node(mystery f lt, x, mystery f rt)
                       else Empty
  end
```

If the same tree `t` is provided as input to the code shown below...

```
t =      4
        / \
       2   6
          / \
         5   8
```

...what will be the resulting output tree `t2`? Draw it below.

```
let t2 = mystery (fun x -> x < 6) t
```

```
t2 =         4
            /
           2
```

Grading scheme: +0.5 attempted the problem OR +2 partially correct OR +3 completely correct

(e) (3 points) Does the `mystery` function preserve BST invariants? That is, if the input to the function is a BST and *any* function `f` of type `int->bool`, will it *always* return a valid BST as output? (Choose one.)

☒ Yes. If you chose yes, explain why.

The function will always preserve invariants since either it keeps the nodes in the tree or replaces them with `Empty` (and removes all children in the latter case).

□ No. If you chose no, provide an example of a function `f` as input using the tree `t`. (For the function, you can choose to write code or describe it in words. If it's the latter, please be as precise and accurate as possible.)

Grading scheme: +1 Selected "yes" AND (+1 for partially correct explanation OR +2 completely correct explanation)

3. **Java Design Problem** (33 points total)

Step 1: Understand the problem For Homework 7 and 8, you worked with `Iterator`s in Java that iterated over a *single* source of data (such as one CSV file). For this design problem, we will create two new kinds of iterators, called `SequenceIterator` and `MergeIterator` that can each draw from *two* sources of data.

A `SequenceIterator` is built from two other iterators, say `first` and `second`. Its `next` method will return items from `first` (by calling `first.next()`) till `first` becomes empty (`first.hasNext()` returns `false`); then it will return items from `second` until it, too, becomes empty.

A `MergeIterator` is also built from two iterators, say `first` and `second`. Its `next` method will first call `first.next()` and return its result, assuming `first.hasNext()` is `true`; the next item it returns is the one returned by calling `second.next()`, assuming `second.hasNext()` is `true`. The third call to `next` returns the next item returned from `first.next()`, and so on, continuing to alternate between the two. If one of the iterators has no more items left, it will use the other iterator for the rest.

For example, if `first` is an iterator over the array `{1, 2, 3}` and `second` is an iterator over the array `{4, 5}`, then...

- Calling `next()` on an iterator obtained from **`new`** `SequenceIterator(first,second)` will return `1`, `2`, `3`, `4`, and `5`, after which `hasNext()` will return **`false`**.
- Calling `next()` on an iterator obtained from **`new`** `MergeIterator(first,second)` will return `1`, `4`, `2`, `5`, and `3`, after which `hasNext()` will return **`false`**.

These examples are written out as JUnit tests on page 9.

(No questions on this page.)

Step 2: Design the interfaces We are considering two classes here—`SequenceIterator` and `MergeIterator`—that both implement the `Iterator<Integer>` interface.

Recall that an `Iterator` is an object that yields a sequence of elements. The Javadocs for the `Iterator<E>` interface are given in Appendix B.

We should also think a bit about the circumstances under which they can raise exceptions.

(a) (4 points) Based on the `Iterator` interface, is it possible for the `next()` method of a sequence or merge iterator to throw an `IOException` (either intentionally or accidentally)?

□ Yes ☒ No

In one sentence, explain your answer:
`IOException`s are checked exceptions in Java and must be either declared in the method signature (which it's not in this case) or handled with a `try/catch`.

(b) (4 points) Based on the interface, is it possible for the `next()` method of a sequence or merge iterator to throw a `NullPointerException` (either intentionally or accidentally)?

☒ Yes □ No

In one sentence, explain your answer:
`NullPointerException`s are unchecked exceptions in Java and don't need to be declared or caught.

Grading scheme: +1.5 Selected the correct checkbox AND (+1 for partially correct explanation OR +2.5 completely correct explanation)

Step 3: Write test code for `SequenceIterator` and `MergeIterator` One benefit of using the `Iterator` interface is that we can create iterators from other datatypes in Java (without needing to use the file system). Here are two example test cases written in this style.

```
@Test
public void testSequenceHasNextAndNext() {
    Integer[] firstElts = {1, 2, 3};
    Integer[] secondElts = {4, 5};
    Iterator<Integer> first = Arrays.asList(firstElts).iterator();
    Iterator<Integer> second = Arrays.asList(secondElts).iterator();

    SequenceIterator sequenced = new SequenceIterator(first, second);
    assertTrue(sequenced.hasNext());
    assertEquals(1, sequenced.next());
    assertEquals(2, sequenced.next());
    assertEquals(3, sequenced.next());
    assertEquals(4, sequenced.next());
    assertEquals(5, sequenced.next());
    assertFalse(sequenced.hasNext());
    assertFalse(first.hasNext());
    assertFalse(second.hasNext());
}
```

```
@Test
public void testMergeHasNextAndNext() {
    Integer[] firstElts = {1, 2, 3};
    Integer[] secondElts = {4, 5};
    Iterator<Integer> first = Arrays.asList(firstElts).iterator();
    Iterator<Integer> second = Arrays.asList(secondElts).iterator();

    MergeIterator merged = new MergeIterator(first, second);
    assertTrue(merged.hasNext());
    assertEquals(1, merged.next());
    assertEquals(4, merged.next());
    assertEquals(2, merged.next());
    assertEquals(5, merged.next());
    assertEquals(3, merged.next());
    assertFalse(merged.hasNext());
    assertFalse(first.hasNext());
    assertFalse(second.hasNext());
}
```

(No questions on this page.)

(a) (4 points) Fill in the blanks in the following test so that all the assertions pass. Each line beginning `assert`__________ must be completed with either `True` or `False`. The other blanks should be filled with numbers.

```
@Test
public void mergeSame() {
    Integer[] elts = {1, 2, 3};
    Iterator<Integer> iter = Arrays.asList(elts).iterator();

    MergeIterator merged = new MergeIterator(iter, iter);

    assertEquals(1, merged.next());

    assertEquals(2, iter.next());

    assertEquals(3, merged.next());

    assertFalse(merged.hasNext());

    assertFalse(iter.hasNext());
}
```

(b) (6 points) Again, fill in the blanks so that all the assertions pass.

```
@Test
public void nestedMerge() {
    Integer[] firstElts = {1, 2};
    Iterator<Integer> first = Arrays.asList(firstElts).iterator();

    Integer[] secondElts = {3, 4};
    Iterator<Integer> second = Arrays.asList(secondElts).iterator
       ();

    Integer[] thirdElts = {5, 6};
    Iterator<Integer> third = Arrays.asList(thirdElts).iterator();

    MergeIterator merged12 = new MergeIterator(first, second);
    MergeIterator merged123 = new MergeIterator(merged12, third);

    assertEquals(1, merged123.next());

    assertEquals(5, merged123.next());

    assertEquals(3, merged123.next());

    assertEquals(6, merged123.next());

    assertEquals(2, merged123.next());

    assertTrue(merged12.hasNext());

    assertFalse(third.hasNext());
}
```

Step 4: Implement `MergeIterator` (15 points)

Complete the code for `MergeIterator`. Your implementation should satisfy the `Iterator<Integer>` interface.

Hint: You might want to think about what *invariant* the state of your iterator maintains.

```
public class MergeIterator implements Iterator<Integer> {

    private Iterator<Integer> first;
    private Iterator<Integer> second;
    private boolean isFirst;

    // you can assume first and second are not null
    public MergeIterator (Iterator<Integer> first, Iterator<Integer>
       second) {
        this.first = first;
        this.second = second;
        isFirst = true;
    }

    @Override
    public boolean hasNext() {
        return (first.hasNext() || second.hasNext());
    }

    @Override
    public Integer next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }

        if (!first.hasNext()) {
            return second.next();
        } else if (!second.hasNext()) {
            return first.next();
        } else if (isFirst) {
            isFirst = !isFirst;
            return first.next();
        } else {
            isFirst = !isFirst;
            return second.next();
        }
    }
}
```

4. **Java Subtyping and Dynamic Dispatch** (24 points total)

This problem refers to three interfaces and several classes that might appear in program about Animals. You can find them in Appendix C.

(a) (2 points) Which lines of code are example uses of subtype polymorphism in Java? (Mark all that apply.)

☒ Line 84 ☐ Line 85 ☐ Line 86 ☐ Line 87 ☒ Line 89

(b) (2 points) Which lines of code are example uses of parametric polymorphism (i.e., generics) in Java? (Mark all that apply.)

☐ Line 84 ☐ Line 85 ☐ Line 86 ☐ Line 87 ☒ Line 89

(c) (4 points)

```
____________________ winter = new Dolphin();
```

Which type can be correctly used for the declaration of `winter` above? (Mark all that apply.)

☒ `Animal` ☐ `Flyer` ☐ `Penguin` ☒ `Swimmer`
☒ `Mammal` ☒ `Dolphin` ☐ `Bat` ☒ `Object`

Which of the following lines is legal Java code that will not cause any compile-time (i.e., type checking) or run-time errors?

If it is legal code, check the "Legal Code" box and answer the questions that follow it. If it is not legal, check one of the "Not Legal" options and explain why.

You can assume each option below is independent and written after line 91 in the `main` method (as shown in the Appendix).

(d) (3 points)

```
Animal dog = new Mammal();
```

□ Legal Code

A. The static type of `dog` is `Animal`.

B. The dynamic class of `dog` is `Mammal`.

□ Not Legal — Will compile, but will throw an `Exception` when run

☒ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

Cannot instantiate an Abstract Class.

(e) (3 points)

```
Mammal dolphin = new Dolphin();
System.out.println(dolphin.commonName());
```

☒ Legal Code

The code above will print (Choose all that apply.)

□ "Mammal"

☒ "Dolphin"

□ Not Legal — Will compile, but will throw an `Exception` when run

□ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

(f) (3 points)

```
Flyer bat = new Bat();
bat.echoLocate();
```

□ Legal Code

The code above will print (Choose all that apply.)

□ "Bat"

□ "Fly, bat, fly!"

□ "<<<eeek>>>"

□ Not Legal — Will compile, but will throw an `Exception` when run

☒ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

The `echoLocate()` *method isn't defined for* `Flyer`*.*

(g) (3 points)

```
Swimmer swim = new Dolphin();
Animal animal = (Animal) swim;
System.out.println(animal.distinguishingFeature());
```

☒ Legal Code

A. The static type of `animal` is `Animal`.

B. The dynamic class of `animal` is `Dolphin`.

C. The code above will print (Choose all that apply.)

☒ "Being in Titanic"

□ "Hair"

□ "Tuxedo Feather Pattern"

□ Not Legal — Will compile, but will throw an `Exception` when run

□ Not Legal — Will not compile

Reason for not legal (in either of the two illegal cases above):

–

(h) (4 points)

```
Swimmer swim = new ___???___();
Animal animal = (Dolphin) swim;
System.out.println(animal.distinguishingFeature());
```

What can be used on the first line (instead of the `???`) so that the code successfully compiles *but throws an exception when run*? (Mark all that apply.)

□ `Swimmer` □ `Bat` ☒ `Penguin` □ `Object`

Grading scheme: For (a, b), +0.5 for one correct response OR +1 for two correct responses or +1.5 for 3-4 correct responses OR +2 for all correct responses

Grading scheme: For (c, i), +0.5 for each option correctly answered

Grading scheme: For Illegal Code (d, f), +1.5 Selected correct option AND (+0.5 for partially correct reason OR +1.5 for completely correct reason)

Grading scheme: For (e), +1.5 Selected correct option AND +1.5 for print option correctly answered

Grading scheme: For (g), +1 Selected correct option AND +0.5 for static type correctly answered and +1 for dynamic class correctly answers AND +0.5 for print option correctly answered

Grading scheme: For (h), +1 for each option correctly answered

5. **Java Swing Programming** (29 points total)

Appendix D shows the code for a simplified version of the `PaintE` application that was demoed in lecture. The following questions use this code to test your understanding of both Swing and Java programming idioms.

(a) (2 points) The class `PointMode` defined on line 8 implements the Swing interface `MouseMotionListener`.

True ☒ False □

(b) (2 points) How will the program's behavior change if we delete the call to `canvas.repaint()` on line 67? (Select one.)

□ Drawing points will work fine, but lines will only appear after another point is entered (by going back to Point mode and clicking in the drawing area).

☒ Drawing points will work fine, and lines will be drawn as usual after the mouse is released, but the "preview" behavior of line drawing will stop working.

□ Nothing at all will ever be displayed – just a blank window.

□ The initial GUI will be displayed, but no shapes will ever be drawn.

□ No change in behavior.

(c) (3 points) How many instances of the class `PointMode` are created during a whole run of the program? (Select one.)

☐ None.

☐ At most one.

☒ Exactly one.

☐ One for every time the user enters a line (by clicking the drawing canvas twice while in line-drawing mode).

☐ Something else: ____

(d) (3 points) How many instances of the class `LineStartMode` are created during a whole run of the program? (Select one.)

☐ None.

☐ At most one.

☐ Exactly one.

☐ One for every time the user enters a line (by clicking the drawing canvas twice while in line-drawing mode).

☒ One for every time the user enters a line (by clicking the drawing canvas twice while in line-drawing mode), *plus* one when the program starts running.

☐ Something else: ____

(e) (3 points) How many instances of the class `LineEndMode` are created during a whole run of the program? (Select one.)

☐ None.

☐ At most one.

☐ Exactly one.

☒ One for every time the user enters a line (by clicking the drawing canvas twice while in line-drawing mode).

☐ One for every time the user enters a line (by clicking the drawing canvas twice while in line-drawing mode), *plus* one when the program starts running.

☐ Something else: ____

(f) (8 points) At the moment, lines and points are always drawn in black. If we wanted to give the user the ability to draw in multiple colors, what would we need to *add* or *change* in the existing code? (Just summarize the changes briefly in English – no need to actually write anything in Java. Make sure to consider how this change would affect the fields of the `PaintE` class, the construction of the GUI, and the behavior of GUI elements.)

Answer: We would need to

- *add a new field to hold the current color*
- *change lines 11, 28, and 34 to refer to this field*
- *add a new group of radio buttons around line 121, one for each color*
- *select one of the buttons at the beginning by calling its* `doClick` *method*

(g) (8 points)

Suppose we wanted to let the user draw rectangles in addition to lines and points. The "look and feel" should be similar to adding lines: when in Rectangle mode, the user can click, drag, and release the mouse to add a new rectangle to the picture; while dragging, a preview of the rectangle will be drawn. What do we need to add to the code in Appendix D to implement this new feature? (Just describe the additions in English—no need to write any Java code.)

Answer: We would need to add

- *a new class RectangleShape that looks exactly like LineShape except that it paints a rectangle instead of a line.*
- *two new inner classes, RectangleStartMode and RectangleEndMode, that are exactly like LineStartMode and LineEndMode except that "Line" is replaced by "Rectangle" everywhere*
- *a new JRadioButton similar to the ones for points and lines, replacing* `new lineStartMode()` *with* `new RectangleStartMode()`

Scratch Space

Use this page for work that you do not want us to grade. If you run out of space elsewhere in the exam and you ***do*** *want to put something here that we should grade, make sure to put a clear note in the normal answer space for the problem in question.*

A OCaml Code

Binary Trees

```
(* The type of generic binary trees. *)
type 'a tree =
  | Empty
  | Node of 'a tree * 'a * 'a tree
```

Higher-order Function: Transform

```
let rec transform (f: 'a -> 'b) (l: 'a list): 'b list =
  begin match l with
  | [] -> []
  | hd :: tl -> (f hd) :: (transform f tl)
  end
```

B Iterator Interface

`interface` `Iterator<E>`

`boolean` `hasNext()`

Returns **`true`** if the iteration has more elements. (In other words, returns true if `next()` would return an element rather than throwing an exception.)

- **Returns:** **`true`** if the iteration has more elements

`E next()`

Returns the next element in the iteration.

- **Returns:** the next element in the iteration
- **Throws:** `NoSuchElementException` - if the iteration has no more elements

C Java Code for Subtyping

```java
interface Animal {
    String commonName();

    String distinguishingFeature();

    Boolean isWarmBlooded();
}

interface Swimmer extends Animal {
    void swim();
}

interface Flyer extends Animal {
    void fly();
}

abstract class Mammal implements Animal {

    public String commonName() {
        return "Mammal";
    }

    public String distinguishingFeature() {
        return "Hair";
    }

    public Boolean isWarmBlooded() {
        return true;
    }
}

class Penguin implements Swimmer {

    public void swim() {
        System.out.println("swimming!");
    }

    public String commonName() {
        return "Adelie";
    }

    public String distinguishingFeature() {
        return "Tuxedo Feather Pattern";
    }

    public Boolean isWarmBlooded() {
        return true;
    }
}

class Bat extends Mammal implements Flyer {

    public String commonName() {
        return "Bat";
    }

    public void fly() {
        System.out.println("Fly, bat, fly!");
    }

    public void echoLocate() {
        System.out.println("<<<eeek>>>");
    }
}

class Dolphin extends Mammal implements Swimmer {

    public String commonName() {
        return "Dolphin";
    }

    public String distinguishingFeature() {
        return "Being in Titanic";
    }

    public void swim() {
        System.out.println("Swim, swim!");
    }
}

public class Subtyping {

    public static void main(String[] args) {
        Animal penguin = new Penguin();
        printAnimalInfo(penguin);
        Bat bat = new Bat();
        bat.echoLocate();

        List<Animal> animals = new LinkedList<>();
        animals.add(penguin);
        animals.add(bat);
    }

    public static void printAnimalInfo(Animal a) {
        System.out.println("Common Name: " + a.commonName());
        System.out.println("Characterized by: " + a.distinguishingFeature());
    }
}
```

D Java Code for Paint

```java
public class PaintE {
    private final Canvas canvas = new Canvas();
    private final List<Shape> shapes = new LinkedList<>();
    private Shape preview;

    interface Mode extends MouseListener, MouseMotionListener { }

    class PointMode extends MouseAdapter implements Mode {
        public void mouseReleased(MouseEvent e) {
            Point p = e.getPoint();
            shapes.add(new PointShape(Color.BLACK, new BasicStroke(3), p));
        }
    }

    class LineStartMode extends MouseAdapter implements Mode {
        public void mousePressed(MouseEvent e) {
            mode = new LineEndMode(e.getPoint());
        }
    }

    class LineEndMode extends MouseAdapter implements Mode {
        Point lineStart;
        LineEndMode(Point p) {
            lineStart = p;
        }
        public void mouseDragged(MouseEvent arg0) {
            Point p = arg0.getPoint();
            preview = new LineShape(Color.BLACK, new BasicStroke(3),
                                    lineStart, p);
        }
        public void mouseReleased(MouseEvent arg0) {
            mode = new LineStartMode();
            Point p = arg0.getPoint();
            shapes.add(new LineShape(Color.BLACK, new BasicStroke(3),
                                     lineStart, p));
            lineStart = null;
            preview = null;
        }
    }

    private Mode mode = null;
    private class Canvas extends JPanel {
        public void paintComponent(Graphics gc) {
            super.paintComponent(gc);
            for (Shape s : shapes) {
                s.draw(gc);
            }
            if (preview != null) {
                preview.draw(gc);
            }
        }

        public Dimension getPreferredSize() {
            return new Dimension(600, 400);
        }
    }

    private class Mouse extends MouseAdapter {
        public void mousePressed(MouseEvent arg0) {
            mode.mousePressed(arg0);
            canvas.repaint();
        }

        public void mouseDragged(MouseEvent arg0) {
            mode.mouseDragged(arg0);
            canvas.repaint();
        }

        public void mouseReleased(MouseEvent arg0) {
            mode.mouseReleased(arg0);
            canvas.repaint();
        }
    }
    private JRadioButton makeShapeButton(
                             ButtonGroup group,
                             JPanel modeToolbar,
                             String name,
                             final Mode buttonMode) {
        JRadioButton b = new JRadioButton(name);
        group.add(b);
        modeToolbar.add(b);
        b.addActionListener(e -> {
                mode = buttonMode;
                preview = null;
            });
        return b;
    }

    private JPanel createModeToolbar() {
        JPanel modeToolbar = new JPanel();

        ButtonGroup group = new ButtonGroup();
        JRadioButton point =
                makeShapeButton(group, modeToolbar, "Point",
                    new PointMode());
        JRadioButton line =
                makeShapeButton(group, modeToolbar, "Line",
                    new LineStartMode());
        point.doClick();   // Simulate a click on the Point button

        JButton quit = new JButton("Quit");
        modeToolbar.add(quit);
        quit.addActionListener(e -> System.exit(0));
        return modeToolbar;
    }

    public PaintE() {
        JFrame frame = new JFrame();
        frame.setLayout(new BorderLayout());
        Mouse mouseListener = new Mouse();
        canvas.addMouseListener(mouseListener);
        canvas.addMouseMotionListener(mouseListener);
        frame.add(canvas, BorderLayout.CENTER);
        frame.add(createModeToolbar(), BorderLayout.PAGE_END);
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(PaintE::new);
    }
}
```