Ring.java > ...

```java
import java.util.Scanner;

public class Ring {
    Run | Debug
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println(x:"Enter the number of processes: ");
        int num = in.nextInt();

        Rr[] proc = new Rr[num];

        // Initialize processes
    // This code block is initializing the processes. It creates an array of Rr objects with a size of
    // `num` (which is the number of processes entered by the user), and then prompts the user to enter
    // the ID of each process. It sets the index of each process to its corresponding index in the
    // array, sets the state of each process to "active", and sets the value of `f` (which is used as a
    // flag during the election process) to 0 for each process.
        for (int i = 0; i < num; i++) {
            proc[i] = new Rr();
            proc[i].index = i;
            System.out.println("Enter the ID of process " + (i + 1) + ": ");
            proc[i].id = in.nextInt();
            proc[i].state = "active";
            proc[i].f = 0;
        }

        // Sort processes based on ID
      // This code block is sorting the `proc` array of `Rr` objects based on the `id` field of each
      // object. It uses a bubble sort algorithm, where it compares adjacent elements in the array and
      // swaps them if they are in the wrong order. The outer loop iterates `num - 1` times, and the
      // inner loop iterates `num - 1` times as well. The `if` statement inside the inner loop checks
      // if the `id` of the current element is greater than the `id` of the next element. If it is,
      // then it swaps the two elements using a temporary variable `temp`. This process continues until
      // the array is sorted in ascending order based on the `id` field.
        for (int i = 0; i < num - 1; i++) {
            for (int j = 0; j < num - 1; j++) {
                if (proc[j].id > proc[j + 1].id) {
```

Ring.java > ...

```
36                  if (proc[j].id > proc[j + 1].id) {
37                      Rr temp = proc[j];
38                      proc[j] = proc[j + 1];
39                      proc[j + 1] = temp;
40                  }
41              }
42          }
43
44          // Print the sorted processes
45          // This code block is printing out the sorted processes in the `proc` array of `Rr` objects. It
46          // uses a `for` loop to iterate through each element in the array, and prints out the index of
47          // the element (`i`), the `id` field of the `Rr` object at that index (`proc[i].id`), and a
48          // space character. The output is formatted as `[index] id `, where `index` is the index of the
49          // process in the array, and `id` is the ID/name of the process.
50          for (int i = 0; i < num; i++) {
51              System.out.print("[" + i + "] " + proc[i].id + " ");
52          }
53
54          // Select last process as coordinator
55          proc[num - 1].state = "inactive";
56          System.out.println("\nProcess " + proc[num - 1].id + " selected as coordinator");
57
58          // This code block is implementing a loop that repeatedly prompts the user to choose between two
59          // options: initiating an election or quitting the program. It uses a `while` loop with a
60          // condition of `true`, which means that the loop will continue indefinitely until it is
61          // explicitly broken out of using a `return` statement.
62          while (true) {
63              System.out.println(x:"\n1. Election\n2. Quit");
64              int ch = in.nextInt();
65
66              // Reset flags
67              for (int i = 0; i < num; i++) {
68                  proc[i].f = 0;
69              }
70
71              switch (ch) {
72                  case 1:
```

Ring.java > ...

```java
switch (ch) {
    case 1:
        System.out.println(x:"Enter the process number that initializes the election: ");
        int init = in.nextInt();
        int temp2 = init;
        int temp1 = init + 1;
        int i = 0;

        while (temp2 != temp1) {
            if (temp1 == num) {
                temp1 = 0;
            }
            if ("active".equals(proc[temp1].state) && proc[temp1].f == 0) {
                System.out.println("Process " + proc[init].id + " sends a message to Process " + proc[temp1].id);
                proc[temp1].f = 1;
                init = temp1;
                i++;
            }
            temp1++;
        }

        System.out.println("Process " + proc[init].id + " sends a message to Process " + proc[temp1].id);
        int max = -1;

        // Find maximum ID for coordinator selection
        for (int j = 0; j < i; j++) {
            if (max < proc[j].id) {
                max = proc[j].id;
            }
        }

        // Select coordinator and update states
        System.out.println("Process " + max + " selected as coordinator");
        for (int k = 0; k < num; k++) {
            if (proc[k].id == max) {
                proc[k].state = "inactive";
            }
```

Ring.java > ...

```
93                      int max = -1;
94
95                      // Find maximum ID for coordinator selection
96                      for (int j = 0; j < i; j++) {
97                          if (max < proc[j].id) {
98                              max = proc[j].id;
99                          }
100                     }
101
102                     // Select coordinator and update states
103                     System.out.println("Process " + max + " selected as coordinator");
104                     for (int k = 0; k < num; k++) {
105                         if (proc[k].id == max) {
106                             proc[k].state = "inactive";
107                         }
108                     }
109                     break;
110                 case 2:
111                     System.out.println(x:"Program terminated.");
112                     in.close();
113                     return;
114                 default:
115                     System.out.println(x:"Invalid response.");
116                     break;
117             }
118         }
119     }
120 }
121
122 class Rr {
123     public int index; // To store the index of the process
124     public int id; // To store the ID/name of the process
125     public int f;
126     public String state; // Indicates whether the process is active or inactive
127 }
128
129
```

```
Enter the number of processes:
5
Enter the ID of process 1:
1
Enter the ID of process 2:
2
Enter the ID of process 3:
3
Enter the ID of process 4:
4
Enter the ID of process 5:
5
[0] 1 [1] 2 [2] 3 [3] 4 [4] 5
Process 5 selected as coordinator

1. Election
2. Quit
1
Enter the process number that initializes the election:
3
Process 4 sends a message to Process 1
Process 1 sends a message to Process 2
Process 2 sends a message to Process 3
Process 3 sends a message to Process 4
Process 3 selected as coordinator

1. Election
2. Quit
1
Enter the process number that initializes the election:
2
Process 3 sends a message to Process 4
Process 4 sends a message to Process 1
Process 1 sends a message to Process 2
Process 2 sends a message to Process 3
Process 3 selected as coordinator
```

```
1. Election
2. Quit
2
Program terminated.
```