AP Practice Questions

Q1.
Create an interface Rotator having a method ‘rotate’. Create an abstract class ‘Cleaner’ having an abstract ‘clean’ method. Have class ‘VacuumCleaner’ extend ‘Cleaner’. ‘VacuumCleaner’ should have a string variable “modelID”, and a static immutable field denoting the total number of models manufactured (set it to 10000). Create an inner class named ‘motor’ in ‘VacuumCleaner’ and have it implement and override the rotate method. The VacuumCleaner should have a ‘motor’ attribute and a method named ‘clean’, which uses the motor’s implemented ‘rotate’ method. Now, instantiate a ‘VacuumCleaner’ variable and save it using serialization. Create another variable of type Cleaner and deserialize and assign the stored VacuumCleaner to it. Use the methods of the initial and final variables.

Q2.
Create 3 JUnet tests for finding if an element exists in a sorted ArrayList using binary search.

Q3.
Extend Q2 by parallelizing the search by:

- Implementing Runnable
- Using Threads

Take user input for a low-point (*low*) and a high-point (*high*) for generating an ArrayList of integers from *low* to *high*, and take user input for the number to search for. Do proper error handling. Ensure that *low* is smaller than *high.* If this isn’t satisfied, throw a custom-defined *LimitsError* with an appropriate message.

Q4.
Create a class ‘TicketDispenser’, which creates ‘Tickets’ having a unique ticketID (can be through a trivial logic) for attractions ‘rollercoaster ride’, ‘bumper cars’, and ‘ferris wheel’. Only one ‘TicketDispenser’ should exist at any moment of time. Use design patterns as required.

Q5.
Implement Parallel Fibonacci using ForkJoinPool (solution present in the lecture slides). Look at both RecursiveTask and RecursiveAction implementations.

Soln for Q1:

```
abstract class Cleaner {
    abstract void clean();
}

interface Rotator{
    public void rotate();
}

class VacuumCleaner extends Cleaner implements java.io.Serializable{
    static final long serialVersionUID = 42L;


    private static int numUnits = 1000;
    private final Motor motor;
    private String modelID;

    public VacuumCleaner(String modelID){
        this.motor = new Motor();
        this.modelID = modelID;
    }

    class Motor implements Rotator, java.io.Serializable{
        static final long serialVersionUID = 20L;
        @Override
        public void rotate() {
            System.out.println("The motor is rotating");
        }
    }

    @Override
    void clean() {
        this.motor.rotate();
        System.out.println("VacuumCleaner is cleaning");
    }

    public int getnumUnits() {
        return numUnits;
    }

    public String getModelID() {
        return modelID;
    }
}

public class MainQ1 {

    public static void main(String[] args) throws IOException, ClassNotFoundException{
        VacuumCleaner cleaner_1 = new VacuumCleaner("Dykon");
        // serialization
        ObjectOutputStream out = null;
        try{
```

```
            out = new ObjectOutputStream(new FileOutputStream("cleaner.txt"));
            out.writeObject(cleaner_1);
        }
        finally{
            out.close();
        }

        //deserialization
        ObjectInputStream in = null;
        Cleaner cleaner_2 = null;
        try{
            in = new ObjectInputStream(new FileInputStream("cleaner.txt"));
            cleaner_2 = (VacuumCleaner) in.readObject();
        }
        finally{
            in.close();
        }


        //printing
        System.out.format("For cleaner_1: %s\n", cleaner_1.getModelID());
        cleaner_1.clean();

        // The below line will give an error because getModelID is not defined for Cleaner
        // System.out.format("For cleaner_2: %s\n", cleaner_2.getModelID());
        System.out.println("\nFor cleaner_2:");
        cleaner_2.clean();

    }
}
```

Soln for Q2:

```
class elementFinder{
   private ArrayList<Integer> arr;
   public elementFinder(ArrayList<Integer> arr){
       this.arr = arr;
   }
   public boolean findElement(int element){
       int start = 0, end = arr.size();
       while (start < end){
           int mid = start + (end - start)/2;
           if (arr.get(mid) == element){
               return true;
           }
           else if(arr.get(mid) > element){
               // search in lower bracket
               end = mid-1;
           }
           else{
               // search upper bracket
               start = mid+1;
           }
       }
       if(arr.get(start) == element){
           return true;
       }
       return false;
   }
}

public class findElementTest{

   @Test
   public void Test1(){

       ArrayList<Integer> testlist = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 1323, 4433));
       elementFinder finder = new elementFinder(testlist);
       int element = 5;

       boolean response = finder.findElement(element);
       assertFalse(response);  // same as assertEquals(response, false);
   }

   @Test
   public void Test2(){
       ArrayList<Integer> testlist = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 1323, 4433));
       elementFinder finder = new elementFinder(testlist);
       int element = 4;

       boolean response = finder.findElement(element);
       assertTrue(response);  // same as assertEquals(response, true);
   }

   @Test
   public void Test3(){
       ArrayList<Integer> testlist = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 340, 1323, 4433));
```

```
        elementFinder finder = new elementFinder(testlist);
        int element = 1;

        boolean response = finder.findElement(element);
        assertTrue(response);  // same as assertEquals(response, true);
    }
}


class MainQ2 {
    public static void main(String[] args) {
        Result result =
                JUnitCore.runClasses(findElementTest.class);
        for (Failure failure: result.getFailures()){
            System.out.println(failure.toString());
        }
        System.out.println(((Result) result).wasSuccessful());
    }
}
```

Soln for Q3 a)

```
class LimitsError extends Exception{
    LimitsError(String message){
        super(message);
    }
}

class elementFinderRunnable implements Runnable{
    private ArrayList<Integer> arr;
    private int toFind;
    private int low, high;

    private boolean result = false;

    public elementFinderRunnable(ArrayList<Integer> arr, int low, int high, int toFind){
        this.arr = arr;
        this.toFind = toFind;
        this.low = low;
        this.high = high;
    }

    @Override
    public void run() {
        int start = this.low, end = this.high;
        boolean elementFound = false;       // to check if it has been found

        while (start < end){
            int mid = start + (end - start)/2;
            if (arr.get(mid) == this.toFind){
                this.result = true;
                elementFound = true;
                break;
            }
            else if(arr.get(mid) > this.toFind){
                // search in lower bracket
                end = mid-1;
            }
            else{
                // search upper bracket
                start = mid+1;
            }
        }
        // if it hasn't been found
        if (elementFound == false){
            if(arr.get(start) == this.toFind){
                this.result = true;
            }
            else{
                this.result = false;
            }
        }
    }

    boolean getResult(){
        return this.result;
```

```
    }
}

public class MainQ3a {
    public static void main(String[] args) throws InterruptedException {

        Scanner s = new Scanner(System.in);

        int low, high;
        while (true){
            try{
                // for the lower limit
                while(true){
                    try{
                        System.out.print("Please enter the start of the array: ");
                        low = Integer.parseInt(s.next());
                        break;
                    }
                    catch (Exception e){
                        System.out.println(e.getMessage());
                        System.out.println("Please enter a valid integer\n");
                    }
                }

                // for the upper limit
                while(true){
                    try{
                        System.out.print("Please enter the end of the array: ");
                        high = Integer.parseInt(s.next());
                        break;
                    }
                    catch (Exception e){
                        System.out.println(e.getMessage());
                        System.out.println("Please enter a valid integer\n");
                    }
                }

                // check that the low point of the array is lower than the high point
                if (low >= high){
                    throw new LimitsError("The limit should be set properly!");
                }

                break;
            }
            catch (LimitsError e){
                System.out.println(e.getMessage());
            }
        }


        ArrayList<Integer> findElement = new ArrayList<Integer>();
        for (int i = low; i < high; i++){
            findElement.add(i);
        }

        // input the number that we have to search for
        int toFind;
```

```
        while(true){
            try{
                System.out.print("Please enter the number you want to find: ");
                toFind = Integer.parseInt(s.next());
                break;
            }
            catch (Exception e){
                System.out.println(e.getMessage());
                System.out.println("Please enter a valid integer\n");
            }
        }

        s.close();

        int mid = findElement.size()/2;
        elementFinderRunnable left = new elementFinderRunnable(findElement, 0, mid,
toFind);
        elementFinderRunnable right = new elementFinderRunnable(findElement, mid+1, findElement.size()-1, toFind);

        Thread t1 = new Thread(left);
        Thread t2 = new Thread(right);

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        boolean res = left.getResult() || right.getResult();

        System.out.format("The search result is: %s", res);
    }
}
```

Soln for Q3b)

```
class elementFinderThread extends Thread{
    private ArrayList<Integer> arr;
    private int toFind;
    private int low, high;

    private boolean result = false;

    public elementFinderThread(ArrayList<Integer> arr, int low, int high, int toFind){
        this.arr = arr;
        this.toFind = toFind;
        this.low = low;
        this.high = high;
    }

    @Override
    public void run() {
        int start = this.low, end = this.high;
        boolean elementFound = false;       // to check if it has been found

        while (start < end){
            int mid = start + (end - start)/2;
            if (arr.get(mid) == this.toFind){
                this.result = true;
                elementFound = true;
                break;
            }
            else if(arr.get(mid) > this.toFind){
                // search in lower bracket
                end = mid-1;
            }
            else{
                // search upper bracket
                start = mid+1;
            }
        }
        // if it hasn't been found
        if (elementFound == false){
            if(arr.get(start) == this.toFind){
                this.result = true;
            }
            else{
                this.result = false;
            }
        }
    }

    boolean getResult(){
        return this.result;
    }
}
```

```java
class LimitsError extends Exception{
    LimitsError(String message){
        super(message);
    }
}

public class MainQ3b {
    public static void main(String[] args) throws InterruptedException {

        Scanner s = new Scanner(System.in);

        int low, high;
        while (true){
            try{
                // for the lower limit
                while(true){
                    try{
                        System.out.print("Please enter the start of the array: ");
                        low = Integer.parseInt(s.next());
                        break;
                    }
                    catch (Exception e){
                        System.out.println(e.getMessage());
                        System.out.println("Please enter a valid integer\n");
                    }
                }

                // for the upper limit
                while(true){
                    try{
                        System.out.print("Please enter the end of the array: ");
                        high = Integer.parseInt(s.next());
                        break;
                    }
                    catch (Exception e){
                        System.out.println(e.getMessage());
                        System.out.println("Please enter a valid integer\n");
                    }
                }

                // check that the low point of the array is lower than the high point
                if (low >= high){
                    throw new LimitsError("The limit should be set properly!");
                }

                break;
            }
            catch (LimitsError e){
                System.out.println(e.getMessage());
            }
        }
```

```
        ArrayList<Integer> findElement = new ArrayList<Integer>();
        for (int i = low; i < high; i++){
            findElement.add(i);
        }

        // input the number that we have to search for
        int toFind;
        while(true){
            try{
                System.out.print("Please enter the number you want to find: ");
                toFind = Integer.parseInt(s.next());
                break;
            }
            catch (Exception e){
                System.out.println(e.getMessage());
                System.out.println("Please enter a valid integer\n");
            }
        }
        s.close();

        int mid = findElement.size()/2;

        elementFinderThread left = new elementFinderThread(findElement, 0, mid, toFind);
        elementFinderThread right = new elementFinderThread(findElement, mid+1, findElement.size()-1,
toFind);

        left.start();
        right.start();

        left.join();
        right.join();

        boolean res = left.getResult() || right.getResult();

        System.out.format("The search result is: %s", res);
    }
}
```

Note the difference between the two implementations: Runnable is an interface, while Thread is an abstract class. There was no change in the LimitsError thus defined.

Soln for Q4: We will use Singleton and Factory design patterns.

```
class TicketDispenser {
    private static TicketDispenser dispenser = null;

    private int sNo = 0;

    // implementation of the Singleton pattern
    public static TicketDispenser getInstance(){
        if (dispenser == null){
            dispenser = new TicketDispenser();
        }
        return dispenser;
    }

    private TicketDispenser(){
        System.out.println("Creating a ticket dispenser");
    }


    public Ticket dispenseTicket(String type){
        if (type == null){
            return null;
        }
        else if (type == "bumperCars"){
            this.sNo+=1;
            return new BumperCarsTicket(sNo);
        }
        else if (type == "rollerCoaster"){
            this.sNo+=1;
            return new RollerCoasterTicket(sNo);
        }
        else if (type == "ferrisWheel"){
            this.sNo+=1;
            return new FerrisWheelTicket(sNo);
        }
        else{
            return null;
        }
    }
}

abstract class Ticket {
    private final int sNo;

    public Ticket(int sNo){
        this.sNo = sNo;
    }

    public int getsNo() {
        return sNo;
    }
}
```

```java
public class BumperCarsTicket extends Ticket{
    public BumperCarsTicket(int sNo){
        super(sNo);
    }

    @Override
    public String toString() {
        return String.format("S.No: %d for bumper cars", super.getsNo());
    }
}

public class FerrisWheelTicket extends Ticket{
    public FerrisWheelTicket(int sNo){
        super(sNo);
    }

    @Override
    public String toString() {
        return String.format("S.No: %d for Ferris wheel ride", super.getsNo());
    }
}

class RollerCoasterTicket extends Ticket {
    public RollerCoasterTicket(int sNo){
        super(sNo);
    }

    @Override
    public String toString() {
        return String.format("S.No: %d for Rollercoaster ride", super.getsNo());
    }

}

public class MainQ4 {
    public static void main(String[] args) {
        TicketDispenser dispenser = TicketDispenser.getInstance();

        Ticket t1 = dispenser.dispenseTicket("bumperCars");
        Ticket t2 = dispenser.dispenseTicket("rollerCoaster");
        Ticket t3 = dispenser.dispenseTicket("ferrisWheel");

        System.out.format("Ticket 1: %s\n", t1.toString());
        System.out.format("Ticket 2: %s\n", t2);
        System.out.format("Ticket 3: %s\n", t3);
    }
}
```