Main.java | SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | Patient.java | module-info.java

```java

public class InvalidPatientIdException extends Exception{
    public InvalidPatientIdException(String message) {
        super(message);
    }
}

```

Main.java | SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java

```java
public class Patient {
    private String patientId;
    private String patientName;
    private String contactNumber;
    private String dateOfVisit;
    private String patientAddress;

    public String getPatientId() {
        return patientId;
    }
    public void setPatientId(String patientId) {
        this.patientId = patientId;
    }
    public String getPatientName() {
        return patientName;
    }
    public void setPatientName(String patientName) {
        this.patientName = patientName;
    }
    public String getContactNumber() {
        return contactNumber;
    }
    public void setContactNumber(String contactNumber) {
        this.contactNumber = contactNumber;
    }
    public String getDateOfVisit() {
        return dateOfVisit;
    }
    public void setDateOfVisit(String dateOfVisit) {
        this.dateOfVisit = dateOfVisit;
    }
    public String getPatientAddress() {
        return patientAddress;
    }
    public void setPatientAddress(String patientAddress) {
        this.patientAddress = patientAddress;
    }

}
```

Main.java | SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```java
import java.util.List;
import java.util.ArrayList;
import java.util.stream.Stream;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.regex.*;
import java.util.stream.Collectors;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class PatientUtility {
    public List <Patient> fetchPatient(String filePath) {
        List<Patient> patients = new ArrayList<>();
        try {
            File register = new File(filePath);
            Scanner reader = new Scanner(register);
            while(reader.hasNextLine()) {
                Patient p = new Patient();
                String[] infos = reader.nextLine().split(",");
                try {
                    if (isValidPatientId(infos[0])) {
                        p.setPatientId(infos[0]);
                        p.setPatientName(infos[1]);
                        p.setContactNumber(infos[2]);
                        p.setDateOfVisit(infos[3]);
                        p.setPatientAddress(infos[4]);
                        patients.add(p);
                    }
                }
                catch(InvalidPatientIdException e1) {
                    System.out.println(e1.getMessage());
                }
            }
            reader.close();
        }
        catch(FileNotFoundException e) {}
        return patients;
```

Main.java | SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```
                }
                catch(InvalidPatientIdException e1) {
                    System.out.println(e1.getMessage());
                }
            }
            reader.close();
        }
        catch(FileNotFoundException e) {}
        return patients;
    }

    public boolean isValidPatientId (String patientId)throws InvalidPatientIdException {
        Pattern p = Pattern.compile("^WM_[A-Z][0-9]{2}$");
        Matcher m = p.matcher(patientId);
        boolean isValid = m.matches();
        if (!isValid) {
            throw new InvalidPatientIdException(patientId + " is an Invalid Patient Id.");
        }
        return isValid;
    }
    public List<Patient> retrievePatientRecords_ByDateOfVisit(Stream<Patient> patientStream, String fromDate, String toDate) {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MM-yyyy");
        return patientStream
        .filter((p) -> {
            try {
                Date start = simpleDateFormat.parse(fromDate);
                Date end = simpleDateFormat.parse(toDate);
                Date current = simpleDateFormat.parse(p.getDateOfVisit());
                return start.compareTo(current) * current.compareTo(end) >= 0;
            }
            catch(ParseException e) {}
            return false;
        }).collect(Collectors.toList());
    }
    public Stream<Patient> retrievePatientRecords_ByAddress(Stream<Patient> patientStream, String address) {
        return patientStream.filter(p -> address.equals(p.getPatientAddress()));
    }
}

```

Main.java | *SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```java
import java.lang.reflect.Method;
import java.util.logging.Logger;
import java.util.logging.Level;
/**
* @author TJ
* This class is used to verify if the Code Skeleton is intact and not modified by participants thereby ensuring smooth auto evaluation*
*/
public class SkeletonValidator {
    public SkeletonValidator() {
        validateClassName("Patient");
        validateClassName("PatientUtility");
        validateClassName("InvalidPatientIdException");
        validateMethodSignature("fetchPatient:java.util.List,isValidPatientId:boolean,retrievePatientRecords_ByDateOfVisit:java.util.List,"
                + "retrievePatientRecords_ByAddress:java.util.stream.Stream","PatientUtility");
    }

    private static final Logger LOG = Logger.getLogger("SkeletonValidator");

    protected final boolean validateClassName(String className) {
        boolean iscorrect = false;
        try {
            Class.forName(className);
            iscorrect = true;
            LOG.info("Class Name " + className + " is correct");
        }
        catch (ClassNotFoundException e) {
            LOG.log(Level.SEVERE, "You have changed either the " + "class name/package. Use the correct package "
                    + "and class name as provided in the skeleton");
        }
        catch (Exception e) {
            LOG.log(Level.SEVERE, "There is an error in validating the " + "Class Name. Please manually verify that the "
                    + "Class name is same as skeleton before uploading");
        }
        return iscorrect;
    }

    protected final void validateMethodSignature(String methodWithExcptn, String className) {
        Class cls = null;
        try {
```

Main.java | *SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```
38    protected final void validateMethodSignature(String methodWithExcptn, String className) {
39        Class cls = null;
40        try {
41            String[] actualmethods = methodWithExcptn.split(",");
42            boolean errorFlag = false;
43            String[] methodSignature;
44            String methodName = null;
45            String returnType = null;
46
47            for(String singleMethod : actualmethods) {
48                boolean foundMethod = false;
49                methodSignature = singleMethod.split(":");
50
51                methodName = methodSignature[0];
52                returnType = methodSignature[1];
53                cls = Class.forName(className);
54                Method[] methods = cls.getMethods();
55                for (Method findMethod : methods) {
56                    if (methodName.equals(findMethod.getName())) {
57                        foundMethod = true;
58                        if (!(findMethod.getReturnType().getName().equals(returnType))) {
59                            errorFlag = true;
60                            LOG.log(Level.SEVERE, " You have changed the " + "return type in '" + methodName + "' method. Please stick to the " + "skeleton provided");
61                            }
62                        else {
63                            LOG.info("Method signature of " + methodName + " is valid");
64                        }
65
66                    }
67                }
68                if (!foundMethod) {
69                    errorFlag = true;
70                    LOG.log(Level.SEVERE, " Unable to find the given public method " + methodName + ". Do not change the " + "given public method name. " + "Verify it with the ske
71                }
72            }
73            if (!errorFlag) {
74                LOG.info("Method signature is valid");
75            }
76
```

with the skeleton");

Main.java | *SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```
45              String returnType = null;
46
47              for(String singleMethod : actualmethods) {
48                  boolean foundMethod = false;
49                  methodSignature = singleMethod.split(":");
50
51                  methodName = methodSignature[0];
52                  returnType = methodSignature[1];
53                  cls = Class.forName(className);
54                  Method[] methods = cls.getMethods();
55                  for (Method findMethod : methods) {
56                      if (methodName.equals(findMethod.getName())) {
57                          foundMethod = true;
58                          if (!(findMethod.getReturnType().getName().equals(returnType))) {
59                              errorFlag = true;
60                              LOG.log(Level.SEVERE, " You have changed the " + "return type in '" + methodName + "' method. Please stick to the " + "skeleton provided");
61                              }
62                          else {
63                              LOG.info("Method signature of " + methodName + " is valid");
64                          }
65
66                      }
67                  }
68                  if (!foundMethod) {
69                      errorFlag = true;
70                      LOG.log(Level.SEVERE, " Unable to find the given public method " + methodName + ". Do not change the " + "given public method name. " + "Verify it with the ske
71                  }
72              }
73              if (!errorFlag) {
74                  LOG.info("Method signature is valid");
75              }
76
77          }
78          catch (Exception e) {
79              LOG.log(Level.SEVERE, " There is an error in validating the " + "method structure. Please manually verify that the " + "Method signature is same as the skeleton before
80          }
81      }
82  }
83
```

before uploading");

Main.java | *SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```java
import java.util.Scanner;
import java.util.List;
import java.util.stream.Stream;

public class Main {

    public static void main(String[] args) {

        // CODE SKELETON - VALIDATION STARTS
        // DO NOT CHANGE THIS CODE

        new SkeletonValidator();

        // CODE SKELETON - VALIDATION ENDS

        Scanner sc = new Scanner(System.in);
        PatientUtility pUtil = new PatientUtility();
        System.out.println("Invalid Patient Id are:");

        List<Patient> patients = pUtil.fetchPatient("PatientRegister.txt");
        System.out.println("Retrieve Patient Details\n1. By Date of Visit\n2. By Address\nEnter your choice:");
        int choice = sc.nextInt();
        sc.nextLine();

        if (choice == 1) {
            System.out.println("Enter the start date");
            String fromDate = sc.nextLine();
            System.out.println("Enter the end date");
            String toDate = sc.nextLine();
            patients = pUtil.retrievePatientRecords_ByDateOfVisit(patients.stream(), fromDate, toDate);
            if (!patients.isEmpty()) {
                    for (Patient p : patients) {
                        System.out.printf("%s %s %s %s %s\n", p.getPatientId(), p.getPatientName(), p.getContactNumber(), p.getDateOfVisit(), p.getPatientAddress());
                    }
            }
            else {
                System.out.println("No patient records available during this interval");
            }
        }
```

Main.java | *SkeletonValidator.java | PatientUtility.java | InvalidPatientIdException.java | *Patient.java | module-info.java

```
16
17          Scanner sc = new Scanner(System.in);
18          PatientUtility pUtil = new PatientUtility();
19          System.out.println("Invalid Patient Id are:");
20
21          List<Patient> patients = pUtil.fetchPatient("PatientRegister.txt");
22          System.out.println("Retrieve Patient Details\n1. By Date of Visit\n2. By Address\nEnter your choice:");
23          int choice = sc.nextInt();
24          sc.nextLine();
25
26          if (choice == 1) {
27              System.out.println("Enter the start date");
28              String fromDate = sc.nextLine();
29              System.out.println("Enter the end date");
30              String toDate = sc.nextLine();
31              patients = pUtil.retrievePatientRecords_ByDateOfVisit(patients.stream(), fromDate, toDate);
32              if (!patients.isEmpty()) {
33                      for (Patient p : patients) {
34                          System.out.printf("%s %s %s %s %s\n", p.getPatientId(), p.getPatientName(), p.getContactNumber(), p.getDateOfVisit(), p.getPatientAddress());
35                      }
36              }
37              else {
38                  System.out.println("No patient records available during this interval");
39              }
40          }
41          else if(choice == 2) {
42              System.out.println("Enter the address");
43              String address = sc.nextLine();
44              Stream<Patient> pStream = pUtil.retrievePatientRecords_ByAddress(patients.stream(), address);
45              pStream.forEach(p -> System.out.printf("%s %s %s %s %s\n", p.getPatientId(), p.getPatientName(), p.getContactNumber(), p.getDateOfVisit(), p.getPatientAddress()));
46          }
47          else {
48              System.out.println("Invalid Option");
49          }
50          sc.close();
51      }
52
53  }
54
```