CSCI 3901
FALL- 2022
FINAL PROJECT REPORT

1. Assumptions

a. As given in the assignment

i. Postal codes identifiers will not overlap
ii. Postal codes will be stable. Once created will not be destroyed.

b. Exception handling

i. IllegalArgumentException: will be thrown when there is an error along with an error message. Exception indicating that a method has received an argument that is invalid or inappropriate for this method's purposes.

ii. SqlException: will be thrown when there is an error along with an error message. An exception that provides information on a database access error or other errors.

iii. RunTimeException: Exceptions will be cached in the runtime exception and will be handled by printing error message.

c. **PowerService class related exceptions**

i. The constructor of the class PowerService will call the class JdbcConnection which is used to create the connection between the java program and database.

DataBase:jdbc:mysql://db.cs.dal.ca:3306?serverTimezone=UTC&useSSL=false
UserName: mpandey
Password: B00917801
Exception thrown: The constructor will throw runTimeException: This exception will be thrown when any error will occur with connecting the database.

ii. In each method of the PowerService class I will be throwing various database related exceptions to handle the errors in the most efficient way.

2. Database Design

a. For the project I have created total 5 tables. The tables are as follows:
   i. AddPostalCode
   ii. AddDistributionHub
   iii. ServiceArea
   iv. HubDamage
   v. HubRepair

b. For creating the tables I have attached the **tablesCreation.sql** file. Using this file, we can create the tables that are required for the project in the MySQL Workbench.
c. I have also attached **truncateTables.sql** and **dropTables.sql** which will be used to truncate and destroy the tables.
d. Explanation of each table:
    i. **AddPostalCode**: The AddPostalCode table contains the postal codes of the province along with their population and area.

    ii. **AddDistributionHub**: This table contains the HubIdentifier which is the primary key of the table. It also contains the coordinates of each hub under the column name Location.

    iii. **ServiceArea**: This table contains HubIdentifier which is the foreign key of the table. The foreign key references to the addDistributionHub table. It provides all the postal codes that are served by a particular hub.

    iv. **HubDamage**: This table contains HubIdentifier which is the foreign key references to AdddistributionHub table and along with that it also contains the estimated time which will be required to fix a given hub.

    v. **HubRepair**: HubRepair contains HubIdentifier which is the foreign key references to AdddistributionHub table. It also contains id of the employee which works on a hub and also contains the time he spend in repairing the hub. If the hub is fully recovered, then it will be in service (true) otherwise it will be false.

3. ER Diagram

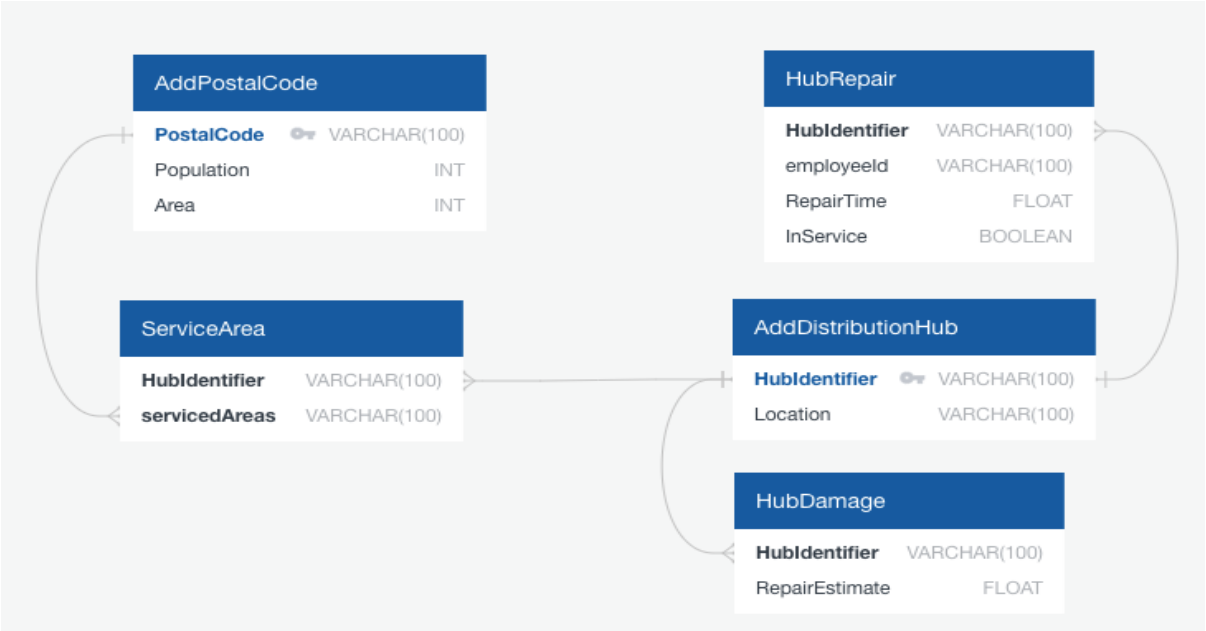

Figure 1. ER-DIAGRAM

4. Implementation Details:

a. PowerService

i. **PowerService:** This method is used to create the connection between database and the java program. If there is an error then it will throw runtime exception.

ii. **Addpostalcode:** This method is taking postalcode, population and area as its parameter. Using this method I am inserting the values into AddPostalCode table in database.

iii. **addDistributionHub:** This method takes hubidentifier, ocation(coordinates of the hub) and area in its parameter. Using this method, we are inserting values into 2 tables (AddDistributionTable and ServiceArea) so that it won't violate the first Normal Form. AddDistributionHub table contains hub identifier along with its coordinates and ServiceArea table consist of hub identifier and postal codes served by the hub.

iv. **hubDamage:** This method takes hub identifier and repair estimate as parameter and insert values in the HubDamage table.

v. **repairHub:** Repair hub gets hub identifier, employee id (who works on that hub), time taken by that employee and if the hub is fully repaired or not as parameter. This method is also used to update HubDamage as well as RepairHub table. It will update the RepairEstimate value by subtracting the (repair estimate of a hub) by sum of (time taken to fix that hub). If the hub is now fixed, then it will change the value of Inservice to true in RepairHub table.

vi. **peopleOutOfService:** This method is used to get the sum of the population of the province who are out of power. The value is then rounded using Math.round() function.

Query

```
SELECT
   SUM(a.populationServedByHub) AS pos
FROM
   (SELECT DISTINCT
      ServiceArea.HubIdentifier,
         ServiceArea.servicedAreas,
         AddPostalCode.Population,
         HubRepair.Inservice,
          c,
         AddPostalCode.Population / c AS populationServedByHub
   FROM
      ServiceArea, HubRepair, AddPostalCode, (SELECT DISTINCT
      servicedAreas, COUNT(HubIdentifier) AS c
   FROM  ServiceArea
   GROUP BY servicedAreas) sub  WHERE
      ServiceArea.servicedAreas = AddPostalCode.PostalCode
         AND sub.servicedAreas = ServiceArea.servicedAreas
         AND HubRepair.HubIdentifier = ServiceArea.HubIdentifier
```

```
AND HubRepair.InService = FALSE) a;
```

vii. **mostDamagedPostalCodes:** This will give the list of the most damaged postal codes. The most damaged postal code is which needs most repairs in order to get whole population to power. One hub can serve many postal codes. I have taken the sum of all the repair time of postal code when the hub associated to that postal code is fixed. Then the values is stored in the object of DamagedPostalCodes class. The values are stored in a list of data type DamagedPostalCodes.

Query

```
Select
ServiceArea.servicedAreas,
sum(HubDamage.RepairEstimate) as damaged from
ServiceArea left join HubDamage
on ServiceArea.HubIdentifier=HubDamage.HubIdentifier
group by ServiceArea.servicedAreas order by damaged desc ;
```

viii. **fixOrder:** The fix order is significance of a hub to fix is the number of people who regain service per hour of repair. I have taken the sum of all the population served by a hub divided by the repair estimate of that hub. Then the values is stored in the object of HubImpact class. The values are stored in a list of data type HubImapct.

Query

```
SELECT
a.HubIdentifier,
SUM(a.populationServedByHub)/HubDamage.RepairEstimate
ASimpact
      FROM
         (SELECT DISTINCT,
            ServiceArea.HubIdentifier,
                ServiceArea.servicedAreas,
                AddPostalCode.Population,
                HubRepair.Inservice,   c,    AddPostalCode.Population / c AS
populationServedByHub
         FROM   ServiceArea, HubRepair, AddPostalCode, (SELECT
DISTINCT
            servicedAreas, COUNT(HubIdentifier) AS c
         FROM
            ServiceArea   GROUP BY servicedAreas) sub
         WHERE    ServiceArea.servicedAreas = AddPostalCode.PostalCode
AND HubRepair.InService = false
              AND sub.servicedAreas = ServiceArea.servicedAreas  AND
HubRepair.HubIdentifier = ServiceArea.HubIdentifier   ) a , HubDamage
where a.HubIdentifier=HubDamage.HubIdentifier
               group by a.HubIdentifier order by impact desc;
```

ix. **rateOfServiceRestoration:** The rate of service restoration returns a list of the rate at which the 100% people in the province is restored to power.The increment is provided in the parameter by which we can increment.I have used list data structures to store population(which get power after a certain time) and repairEstimate(the time when a particular amount of population will get power).

x. **repairPlan:** We will start from a position (hub) and determine a hub that has greatest impact. We take the starting hub to the greatest impact hub. If the starting hub has the greatest impact, then we go to the next greatest impact hub. We go from starting hub to the max impact hub and it will form a rectangle then we consider the points that lie inside that rectangle. Now we will check for x and y monotone. We will check the impact of x and y monotone, and whichever has highest impact that will give us our optimal path.

xi. **underservedPostalByPopulation:**I have used the datastructure list of string data type to store and return the list of postal codes. The postal codes order in descending order is calculated by taking the Count of hub identifiers serving that postal code by population.

Query

```
SELECT DISTINCT
        ServiceArea.servicedAreas,
        AddPostalCode.Area,
        HubRepair.Inservice,
        c,
        AddPostalCode.Area / c AS populationServedByHub
  FROM
     ServiceArea, HubRepair, AddPostalCode, (SELECT DISTINCT
     servicedAreas, COUNT(HubIdentifier) AS c
  FROM
     ServiceArea
  GROUP BY servicedAreas) sub
  WHERE
     ServiceArea.servicedAreas = AddPostalCode.PostalCode
        AND sub.servicedAreas = ServiceArea.servicedAreas
        AND HubRepair.HubIdentifier = ServiceArea.HubIdentifier
        AND HubRepair.InService = FALSE
        order by populationServedByHub desc limit 3;
```

xii. **underServedPostalByArea:** I have used the datastructure list of string data type to store and return the list of postal codes. The postal codes order in descending order is calculated by taking the count of hub identifiers serving that postal code by Area.

Query

```
SELECT DISTINCT
        ServiceArea.servicedAreas,
        AddPostalCode.Area,
        HubRepair.Inservice,
        c,
        AddPostalCode.Area / c AS AreaServedByHub
```

```
FROM
   ServiceArea, HubRepair, AddPostalCode, (SELECT DISTINCT
   servicedAreas, COUNT(HubIdentifier) AS c
FROM
   ServiceArea
GROUP BY servicedAreas) sub
WHERE
   ServiceArea.servicedAreas = AddPostalCode.PostalCode
      AND sub.servicedAreas = ServiceArea.servicedAreas
      AND HubRepair.HubIdentifier = ServiceArea.HubIdentifier
      AND HubRepair.InService = FALSE
      order by AreaServedByHub desc limit 3;
```

b. **JdbcConnection:** It is a class whose constructor will be called from the PoweService class and used to connect the program with database.
   i. **createConnection:** this method is used to create the connection between java program and database.

c. **Point:** This class contains the coordinates x and y of each distribution hub. It has getters and setter for the coordinate values.
   i. **Getter:** getter methods are getx() and getY() used to retrieve the value of coordinates.
   ii. **Setter:** Used to set the value of x and y.
   iii. **loc():** This method is called from the AddDistributionHub method and used to fill the table.

d. **HubImpact:** This class have getter and setter of the hub and their impact. Using fixOrder method values of hub and impact will be set.
   i. **Getter:** Used to retrieve the values of hub and their impact. It has getHubId() and getImpact() method.
   ii. **Setter:** used to set values of hub and their estimate.

e. **DamagedPostalCodes:** This class have getter and setter of the postalcode and their repairtime. Using mostDamagedpostalCode method values of hub and impact will be set.
   i. **Getter:** Used to retrieve the values of postalCode and their repair time. It has getPostalCode() and getRepairEstimate() method.
   ii. **Setter:** Used to set values of postalcode and its repair estimate.

5. Sample Input and Output

a. Input and tables

```
public class Main {
    public static void main(String[] args) throws SQLException {
PowerService ps=new PowerService();


Point p=new Point(2,3);
Point p1=new Point(1,2);
Point p2=new Point(4,5);
Point p3=new Point(7,5);
Point p4=new Point(4,9);
Point p5=new Point(5,3);
```

```
Point p6=new Point(7,1);
Point p7=new Point(5,2);

        Set<String> postalCodes =new HashSet<>();
        postalCodes.add("P2");
        postalCodes.add("P3");

        Set<String> postalCodes1 =new HashSet<>();
        postalCodes1.add("P4");

        Set<String> postalCodes2 =new HashSet<>();
        postalCodes2.add("P1");
        postalCodes2.add("P2");

        Set<String> postalCodes3 =new HashSet<>();
         postalCodes3.add("P4");
         postalCodes3.add("P6");
         postalCodes3.add("P7");

         Set<String> postalCodes4 =new HashSet<>();
         postalCodes4.add("P6");
         postalCodes4.add("P8");
         postalCodes4.add("P1");

         Set<String> postalCodes5 =new HashSet<>();
         postalCodes5.add("P9");
         postalCodes5.add("P3");

         Set<String> postalCodes6 =new HashSet<>();
         postalCodes6.add("P2");
         postalCodes6.add("P4");
         postalCodes6.add("P8");

         Set<String> postalCodes7 =new HashSet<>();
         postalCodes7.add("P5");
         postalCodes7.add("P7");

         ps.addPostalCode( "P1",10 ,200 );
         ps.addPostalCode( "P2",30 ,100 );
         ps.addPostalCode( "P3",25 ,50 );
         ps.addPostalCode( "P4",15 ,250 );
         ps.addPostalCode( "P5",20 ,150 );
         ps.addPostalCode( "P6",10 ,100 );
         ps.addPostalCode( "P7",15 ,250 );
         ps.addPostalCode( "P8",35 ,10 );
         ps.addPostalCode( "P9",40 ,300 );

         ps.addDistributionHub("H1",p,postalCodes);
         ps.addDistributionHub("H2",p1,postalCodes1);
         ps.addDistributionHub("H3",p2,postalCodes2);
         ps.addDistributionHub("H4",p3,postalCodes3);
         ps.addDistributionHub("H5",p4,postalCodes4);
         ps.addDistributionHub("H6",p5,postalCodes5);
         ps.addDistributionHub("H7",p6,postalCodes6);
         ps.addDistributionHub("H8",p7,postalCodes7);

         ps.hubDamage("H1",10);
         ps.hubDamage("H2",20);
         ps.hubDamage("H3",5);
         ps.hubDamage("H4",25);
         ps.hubDamage("H5",30);
```

```
ps.hubDamage("H6",15);
ps.hubDamage("H7",10);
ps.hubDamage("H8",2);

ps.hubRepair("H2","E2",10,false);
ps.hubRepair("H1","E1",5,false);
ps.hubRepair("H3","E4",5,true);
ps.hubRepair("H2","E5",10,true);
ps.hubRepair("H4","E2",15,false);
ps.hubRepair("H5","E3",10,false);
ps.hubRepair("H6","E4",7,false);
ps.hubRepair("H5","E1",10,false);
ps.hubRepair("H7","E5",2,false);
ps.hubRepair("H8","E6",1,false);
ps.hubRepair("H8","E3",1,true);
```

i. AddPostalCode

PostalCode	Population	Area
P1	10	200
P2	30	100
P3	25	50
P4	15	250
P5	20	150
P6	10	100
P7	15	250
P8	35	10
P9	40	300

Figure 2: AddPostalCode

ii. AddDistributionHub()

HubIdentifier	Location
H1	2,3
H2	1,2
H3	4,5
H4	7,5
H5	4,9
H6	5,3
H7	7,1
H8	5,2

Figure 3: AddDistributionHub

iii. serviceArea

HubIdentifier	servicedAreas
H1	P2
H1	P3
H2	P4
H3	P1
H3	P2
H4	P4
H4	P6
H4	P7
H5	P1
H5	P6
H5	P8
H6	P3
H6	P9
H7	P2
H7	P4
H7	P8
H8	P5
H8	P7

Figure 3: ServiceArea

iv. HubDamage

HubIdentifier	RepairEstimate
H1	5
H2	0
H3	0
H4	10
H5	10
H6	8
H7	8
H8	0

Figure 4: HubDamage

v. HubRepair

HubIdentifier	employeeId	RepairTime	InService
H2	E2	10	1
H1	E1	5	0
H3	E4	5	1
H2	E5	10	1
H4	E2	15	0
H5	E3	10	0
H6	E4	7	0
H5	E1	10	0
H7	E5	2	0
H8	E6	1	1
H8	E3	1	1

Figure 5: HubRepair

b. **Output of reporting and planning methods**
   i. **peopleOutOfService**
   ii. **mostDamagedPostalCodes**
   iii. **fixOrder**
   iv. **rateOfServiceRestoration**
   v. **repairPlan**
   vi. **underservedPostalByPopulation**
   vii. **underServedPostalByArea**

```
System.out.println("peopleOutOfService"+ps.peopleOutOfService());
System.out.println(ps.mostDamagedPostalCodes(2).get(1).getPostalCode());
System.out.println(ps.fixOrder(5).get(1).getHubId());
System.out.println("rateOfServiceRestoration"+ps.rateOfServiceRestoration(0
.05f));
System.out.println("underservedPostalByPopulation"+ps.underservedPostalByPo
pulation(2));
System.out.println("underservedPostalByArea"+ps.underservedPostalByArea(3))
;
```

```
peopleOutOfService153
P8
H1
rateOfServiceRestoration[0, 0, 0, 0, 0, 8, 8, 8, 8, 8, 13, 13, 13, 21, 21, 21, 31, 31, 31, 41, 51]
underservedPostalByPopulation[P4, P6]
underservedPostalByArea[P8, P3, P2]

Process finished with exit code 0
```

6. **Test Cases for the Project**
   a. **<u>Input Validation Test Cases:</u>**
      i. **boolean addPostalCode (String postalCode, int population, int area)**

- Null provided for postalCode.
  - Returns False
- Illegal format provided for postalCode.
  - Returns False
- Duplicate value provided for postalCode
  - Returns False
- Valid value for postalCode provided.
  - Accepts the input and returns True
- Negative value provided for population.
  - Returns False
- Null provided for population.
  - Returns False

- Valid value for population provided.
  - Accepts the input and returns True
- Negative value provided for area.
  - Returns False
- Null provided for area.
  - Returns False
- Valid value for area provided.
  - Accepts the input and returns True

ii. **boolean addDistributionHub (String hubIdentifier, Point location, Set<String> servicedAreas)**

- Empty string provided for hubIdentifier.
  - Returns False
- Null provided for hubIdentifier.
  - Returns False
- Illegal format provided for hubIdentifier.
  - Returns False
- Duplicate value provided for hubIdentifier
  - Returns False
- Valid value for hubIdentifier provided.
  - Accepts the input.
- location is null.
  - Returns False
- location is an object of class Point
  - Accepts the input and return true.
- ServicedAreas contains data structure other than String
  - Returns false
- ServicedAreas does not contain any postal code
  - Returns false
- ServicedAreas contains at least one postal code
  - Returns true

iii. **Void hubDamage(String hubIdentifier,float repairEstimate)**

- Null provided for hubIdentifier.
  - Throws RuntimeException
- Illegal format provided for hubIdentifier.
  - Throws RuntimeException
- Valid value for hubIdentifier provided.
  - Accepts the input.
- Negative value provided for repairEstimate.
  - Throws RuntimeException
- Null provided for repairEstimate.
  - Throws RuntimeException
- Valid value for repairEstimate provided.
  - Accepts the input and returns True

iv. **Void hubRepair(String hubIdentifier, String employeeId, float repairTime, boolean inService)**

- Null provided for hubIdentifier.
  - Throws RuntimeException

- Illegal format provided for hubIdentifier.
  - Throws RuntimeException
- Duplicate value provided for hubIdentifier
  - Throws RuntimeException
- Valid value for hubIdentifier provided.
  - Accepts the input.
- Null provided for employeeId.
  - Throws RuntimeException
- Illegal format provided for employeeId.
  - Throws RuntimeException
- Valid value for employeeId provided.
  - Accepts the input.
- Duplicate value provided for employeeId
  - Throws RuntimeException
- Negative value provided for repairEstimate.
  - Throws RuntimeException
- Null provided for repairTime.
  - Throws RuntimeException
- Valid value for repairTime provided.
  - Accepts the input and returns True
- True/ False value passed for inService
  - Accepts the input
- Value other than true/ false passed for inService
  - Throws RuntimeException

v. **List<HubImpact> repairPlan (String startHub, int maxDistance, float maxTime)**

- Null provided for startHub.
  - Throws RuntimeException
- Illegal format provided for startHub.
  - Throws RuntimeException
- Valid value for startHub provided.
  - Accepts the input.
- Negative value provided for maxDistance.
  - Throws RuntimeException
- Null provided for maxDistance.
  - Throws RuntimeException
- Valid value for maxDistance provided.
  - Accepts the input
- Negative value provided for maxTime.
  - Throws RuntimeException
- Null provided for maxTime.
  - Throws RuntimeException
- Valid value for maxTime provided.
  - Accepts the input

b. **Boundary Test Cases**

i. **boolean addPostalCode (String postalCode, int population, int area)**

- PostalCode is an empty string.

        - Throws RuntimeError
    - Population is 0
        - Throws RuntimeError
    - Population is negative
        - Throws RuntimeError
    - Area passed is 0
        - Throws RuntimeError
    - Area passed is negative
        - Throws RuntimeError

ii. **boolean addDistributionHub (String hubIdentifier, Point location, Set<String> servicedAreas)**
    - HubIdentifier is an empty string
        - Throws RuntimeError
    - Location does not cantain any coordinates
        - Throws RuntimeError
    - ServicedAreas is an empty set
        - Throws RuntimeError

iii. **Void hubDamage (String hubIdentifier, float repairEstimate)**
    - HubIdetifier is an empty string
        - Throws RuntimeError
    - RepairEstimate is 0
        - Throws RuntimeError
    - RepairEstimate is in negative
        - Throws RuntimeError

iv. **Void hubRepair(String hubIdentifier, String employeeId, float repairTime, boolean inService)**
    - HubIdetifier is an empty string
        - Throws RuntimeError
    - RepairEstimate is 0
        - Throws RuntimeError
    - RepairEstimate is in negative
        - Throws RuntimeError
    - EmployeeId is an empty string
        - Throws RuntimeError
    - Duplicate EmployeeId entered
        - Throws RuntimeError

v. **int peopleOutOfService ()**
    - Returns an integer value
        - Accepted
    - Returns a value other than integer
        - Throws RuntimeError

vi. **List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)**
    - Call when mostDamagedPostalCodes is an empty list

vii. **List<HubImpact> fixOrder (int limit)**
- Call when fixOrder is an empty list

viii. **List<Integer> rateOfServiceRestoration (float increment)**
- Call when rateOfServiceRestoration is an empty list

ix. **List<HubImpact> repairPlan (String startHub, int maxDistance, float maxTime)**
- Call when repairPlan is an empty list

x. **List<String> underservedPostalByPopulation (int limit)**
- Call when underservedPostalByPopulation is an empty list

xi. **List<String> underservedPostalByArea (int limit)**
- Call when underservedPostalByArea is an empty list

c. **Control Flow Test Cases**

i. **boolean addPostalCode (String postalCode, int population, int area)**
- Constructor is called under normal conditions.
  - Instantiates the PowerService class.

ii. **boolean addDistributionHub (String hubIdentifier, Point location, Set<String> servicedAreas)**
- Reading the hubIdentifier string more than once.
  - Returns false
- ServicedAreas is a valid set
  - Returns true
- Trying to set duplicate values in set
  - Returns false
- ServicedAreas is not a valid set
  - Returns false

iii. **Void hubDamage (String hubIdentifier, float repairEstimate)**
- Reading the hubIdentifier string more than once.
  - Returns false

iv. **Void hubRepair(String hubIdentifier, String employeeId, float repairTime, boolean inService)**
- Trying to set duplicate employeeId
- InService has other than boolean value

v. **int peopleOutOfService ()**
- Return people out of service

vi. **List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)**

- Return postal codes in any order other than descending order
  - False
- Postal codes returned in descending order according to the limit set
  - True

vii. **List<HubImpact> fixOrder (int limit)**
- Return fixorder of hubs in any order other than descending order
  - Runtime error
- Fix order of hubs returned in descending order according to the limit set
  - List is returned

viii. **List<Integer> rateOfServiceRestoration (float increment)**
- List is returned of list of hours of repair effort.
  - List returned

ix. **List<HubImpact> repairPlan (String startHub, int maxDistance, float maxTime)**
-

x. **List<String> underservedPostalByPopulation (int limit)**
- List is returned of underservedPostalByPopulation.
  - List returned

xi. **List<String> underservedPostalByArea (int limit)**
- List is returned of underservedPostalByArea.
  - List returned

d. **<u>Data Flow Test Cases</u>**

- Call addPostalCode() method multiple times in a row
- Call adddistributionHub() method multiple times in a row
- Call hubDamage() method multiple times in a row
- Call hubRepair() method multiple times in a row
- Call hubDamage () method before calling addPostalCode () method
- Call hubDamage () method before calling adddistributionHub () method
- Call peopleOutofService () method before calling addPostalCode () method
- Call peopleOutofService () method before calling adddistributionHub () method
- Call mostDamagedpostalCodes() method before calling addPostalCode () method
- Call mostDamagedpostalCodes () method before calling adddistributionHub () method
- Call mostDamagedpostalCodes () method before calling hubRepair() method
- Call fixOrder() method before calling mostDamagedpostalCodes () method
- Call rateOfServiceRestoration() method before calling fixOrder() method
- Call repairplan() method multiple times in a row

- Call repairPlan() method before calling addPostalCode () method
- Call repairplan() method before calling adddistributionHub () method
- Call underservedPostalByPopulation() method before calling addPostalCode () method
- Call underservedPostalByPopulation() method before calling adddistributionHub() method
- Call underservedPostalByArea () method before calling addPostalCode () method
- Call underservedPostalByArea () method before calling adddistributionHub () method