Project external documentation

Overview

This program consists of main body work in the power service class. It outputs the no. of people who are without service, the most damaged region, the order of most significant(impactful) hubs, rate at which people get service back, plan for vehicle to travel to hubs to fix, and to identify the areas who underserviced by the company to improve service in the future.

Files and external data

The program consists of the following file and databases.

- Powerservice.java – a class that manages the body of work as a whole. It includes the commits made and has methods to return various software components, busy classes, repeated bugs, broad features and experts
- Postalcode table – stores postalcode, people, area, hub count, estimated repair time of hub, list of hubs servicing the postal code
- distribHub table – stores hubid, location at which hub is present, areas servicdd by the hub and population covered by the hub
- Hubdamage table- stores hubid along with repair estimate and total time to repair the hub
- HubRepair table – stores hubid, empid who worked on the hub, repair time taken by employee, hubInService to indicate whether hub is repaired or not
- postalBypopulation table- stores postal code, no. of population in the hub covering the postalcode
- postalByarea - stores postalcode, area covering the hub by postalcode

Algorithms&Data structures and their relations to each other

To compute the people out of service(), calculate the no. of people in each hub by summing the postal codes population. For each hub , execute the query and get the information from the table to compute the total population who are out of service.

Damagedpostalcodes class has the information related to repair time needed for each postal code. The class instance values will be set from the sql query output .

HubImpact class also gets hubimpact value by taking the data from the sql query. The rate at which the service can be restored can be calculated by using the impact value of each hub.

For each postal code get the information of how many people living and area of the postal code through the query from database. Divide the hubs with that area /

population and compare with the other hubs to print the top underserved population.

Advantanges of using database:

- Using of databases speed up the execution process
- Reduces complexity of java program
- Improves understanding and modularity of the code.

Assumptions:

1. When repair estimate is less than repair time and hub is still not in service, which means the original repair estimate is wrong and user/employee will be updating it in the real-time. In the code, repair estimate will not be changed in this case assuming that the time to repair would be same as initial estimate.
2. Hub Repair estimate table is created in the system before creating HubDamage table
3. There won't be duplicate additions of rows to any of the db tables
4. createMaps() method would be called first while implementing any of the planning or reporting methods.
5. Data would be entered from the government census in real-time, sample data was taken and tested in the code.

Constraints:

- Vehicle can travel only in monotonous fashion
- Diagonal of the rectangle can be crossed more than once.
- Postal code can't be duplicate
- Postal codes can't be deleted once entered
- Data should be entered in an order to the system to avoid conflicting information

Error Reporting:

Exception handling is done using try catch blocks for all sql queries and connections. Any other errors will be returned from the methods without proceeding with the implementation.

Test Plan:

Input validation:

boolean addPostalCode (String postalCode, int population, int area)

- postal code is null
- int/float/boolean/double passed as parameter for postal code
- float/double/boolean/char/string passed as parameter for population
- float/double/boolean/char/string passed as parameter for area

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

- hubIdentifier is null
- int/float/boolean/double passed as parameter for hubIdentifier
- int/float/double/boolean/char/string passed as parameter for location
- int/float/double/boolean/char/string passed as parameter for servicedAreas

Void hubDamage (String hubIdentifier, float repairEstimate)

- hubIdentifier is null
- int/float/boolean/double passed as parameter for hubIdentifier
- int/double/boolean/char/string passed as parameter for repairEstimate

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

- employeeId is null
- int/float/boolean/double passed as parameter for hubIdentifier
- hubIdentifier is null
- int/float/boolean/double passed as parameter for employeeId
- int/double/boolean/char/string passed as parameter for repairTime
- int/float/double/char/string passed as parameter for inService

List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)
List<String> underservedPostalByPopulation (int limit)
List<String> underservedPostalByArea (int limit)
List<HubImpact> fixOrder (int limit)

- float/boolean/double/char/string passed as parameter for limit

List<Integer> rateOfServiceRestoration (float increment)

- int/double/boolean/char/string passed as parameter for increment

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

- int/float/boolean/double passed as parameter for startHub.
- float/double/boolean/char/string passed as parameter for maxDistance
- int/double/boolean/char/string passed as parameter for maxTime

Boundary conditions:

boolean addPostalCode (String postalCode, int population, int area)

- population exceeds int_maximum

- area exceeds int maximum
- postal code is duplicate
- postal code is single character
- postal code is empty string
- area is negative number
- population is negative number

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

- hubIdentifier is duplicate
- location is same for two hubidentifiers
- hubIdentifier is empty string
- servicedArea not available in the list of postal codes
- servicedAreas set is empty
- location set to origin
- hubidentifier has single character

Void hubDamage (String hubIdentifier, float repairEstimate)

- hubIdentifier is duplicate
- hubIdentifier is empty string
- repairEstimate is negative
- hubidentifier has single character

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

- hubIdentifier is duplicate
- employeeId is duplicate
- repairEstimate is negative
- hubIdentifier is empty string
- employeeId is empty string
- hubidentifier has single character
- employeeId has single character

List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)
List<String> underservedPostalByPopulation (int limit)
List<String> underservedPostalByArea (int limit)
List<HubImpact> fixOrder (int limit)

- limit is negative number
- limit is int_maximum

List<Integer> rateOfServiceRestoration (float increment)

- increment is negative
- increment exceeds maximum percentage

ControlFlow:

boolean addPostalCode (String postalCode, int population, int area)

- postal codes having duplicate values
- postal codes having unique values

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

- hubIdentifier values are unique
- hubIdentifer has duplicate values

Void hubDamage (String hubIdentifier, float repairEstimate)

- Inserting valid estimate hours
- Inserting invalid no. of hours

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

- hubIdentifier values are unique
- hubIdentifer has duplicate values
- employeeId values are unique
- employeeId has duplicate values
- repairTime is valid
- repairTime has invalid no. of hours
- repairTime greater than repairEstimate
- repairTime less than repairEstimate
- repairTime equal to repairEstimate

List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)
List<String> underservedPostalByPopulation (int limit)
List<String> underservedPostalByArea (int limit)
List<HubImpact> fixOrder (int limit)

- limit is valid number
- limit is invalid number
- No. of records less than limit no.
- output records having tie to list limit no. of them

List<Integer> rateOfServiceRestoration (float increment)

- increment is valid value
- increment having invalid value
- increment is very small
- increment is large value

DataFlow:

boolean addPostalCode (String postalCode, int population, int area)

- call before addDistributionHub()

- call before hubDamage()
- call before hubRepair()

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

- call before hubImpact object creation
- call after hubDamage()
- call after hubRepair()

Void hubDamage (String hubIdentifier, float repairEstimate)

- call before addDistributionHub()
- call after hubDamage()
- call before hubDamage()

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

- call before addPostalcode()
- call before addDistributionHub()
- call before hubDamage()

List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)
List<String> underservedPostalByPopulation (int limit)
List<String> underservedPostalByArea (int limit)

- call before addPostalcode() method

List<HubImpact> fixOrder (int limit)

- call before hubimpact helper class implementation
- call before getters and setters method call
- call before addPostalcode()
- call before addDistributionHub()

List<Integer> rateOfServiceRestoration (float increment)

- call before addPostalcode()
- call before addDistributionHub()
- call before hubRepair() method