Problem Description

A research lab that uses highly toxic substances requires the development of a system that will detect toxic leaks and raise an alarm to the appropriate emergency staff.

The emergency system consists of a number of sensor subsystems. Each sensor subsystem is capable of detecting a specific toxic leak. Dealing with each kind of toxic leak requires specific skills. Hence the company wishes to contact different sets of people depending on the kind of emergency.

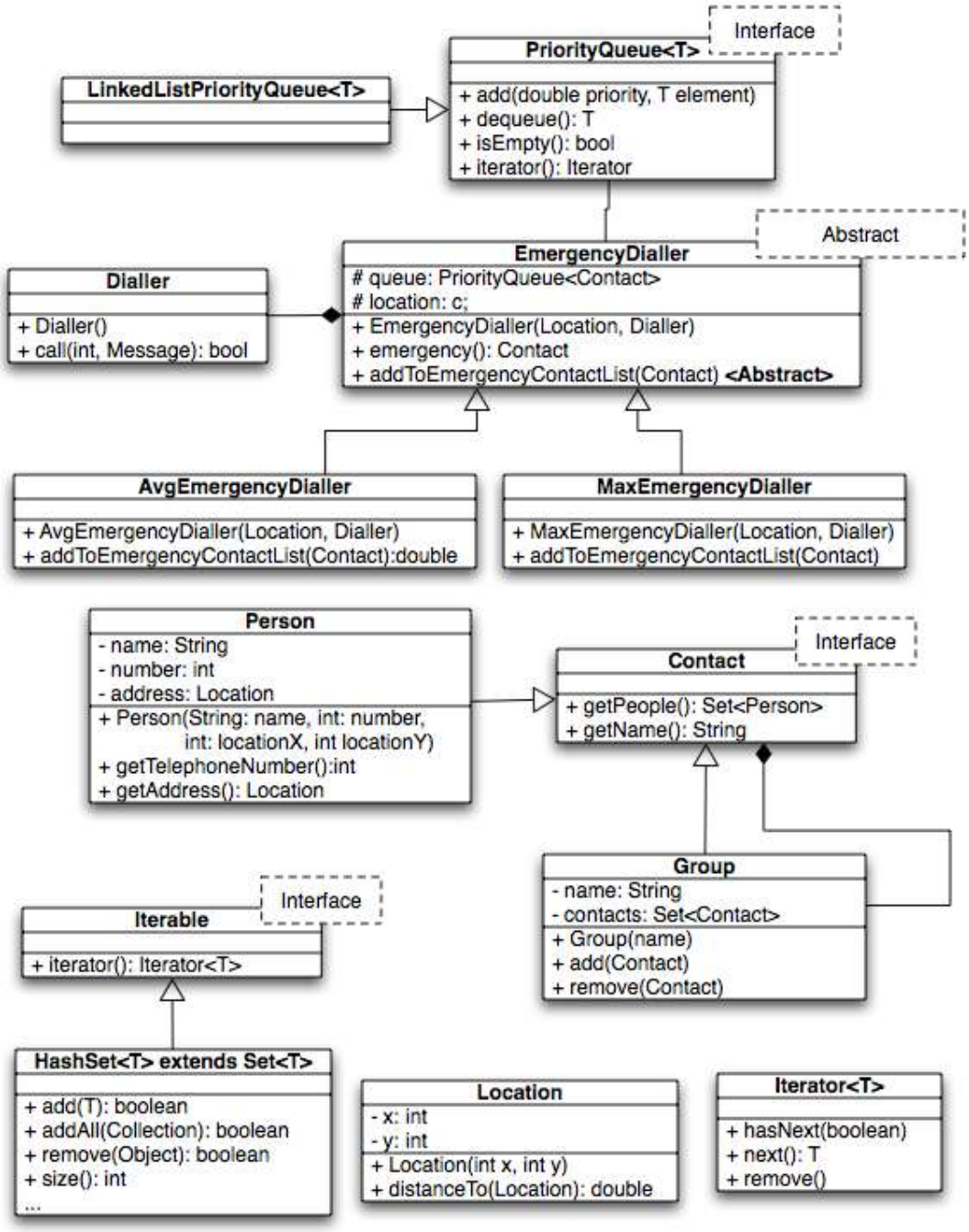
For security reasons, it is crucial that the system notify successfully at least one person in the case of an emergency. Consequently, to avoid the risk of not finding a contact capable of resolving the emergency, the sets of people associated with each kind of emergency tend to be quite large.

However, to react in an organised fashion, the company wishes the system to designate only one contact as the emergency coordinator. Hence, the emergency system is to implement a priority queue of contacts. If the first contact on the queue is unavailable or unreachable, then the second contact of the queue is called. If the second contact is not available, then the third contact is used, and so on. The priority that determines the order in which contacts will be called is the distance of the contact address from the location of the leak.

Another requirement of the system is that a contact can be a group (a specialised team who all need to be present) of people. If this were the case, the system will call each person that belongs to the group and will consider the call to the group successful if all the calls to people in the group were successful. The criterion for determining the priority of a contact group is expected to change depending on the kind of emergency. At least two criteria are anticipated. The first is that the priority of the queue is based on the *average* distance of the people in the group to the location of the emergency. For example if a group has 3 members whose distances from their addresses to the emergency location is 4, 6, and 5, then the priority of the group is 5. The second criterion is that priority of the queue is based on the *maximum* distance of the people in the group to the location of the emergency. Consequently, the priority of the group described earlier is 6.

The emergency system interacts with a complex communication system that the company has in place. The communication system is accessed via a **Dialler** class which is capable (via the **call** method) of sending a message to a person and returning a boolean value that indicates whether the person has received and accepted the message that was sent.

The following class diagram has been produced as a partial design for the system. Note that the class diagram is not complete, it complements the information provided in this problem statement and in the Specific Tasks section shown below.



Activities

Your task is to implement the emergency system in Java. You are given a number of files that will help you develop the system. You are provided with:

- The source code of classes **Location**, **Dialler**, **Person, InputReader** and **Node<E>** .
- Compiled versions (i.e. class files) of all classes you will have to develop. These compiled versions can be used to test your code or as stubs while

you are developing other classes.

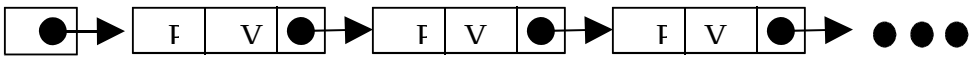
- Compiled test programs for each class you are to develop. You **must** make sure that each class that you have developed passes its tests in addition to satisfying the requirements set out in this document. The tests for a class named **X** can be executed by running the main method of the class **TestX**. For instance, tests for class **Person** are provided in class **TestPerson**. Note that for most tests, we only provide the class file and not the source file.

Notes:

- Refer to the exam coversheet for explanation of how to access the provided source and class files.
- All classes of this system (both classes provided and classes you are to write) are not to be declared as part of any package.

Specific Tasks

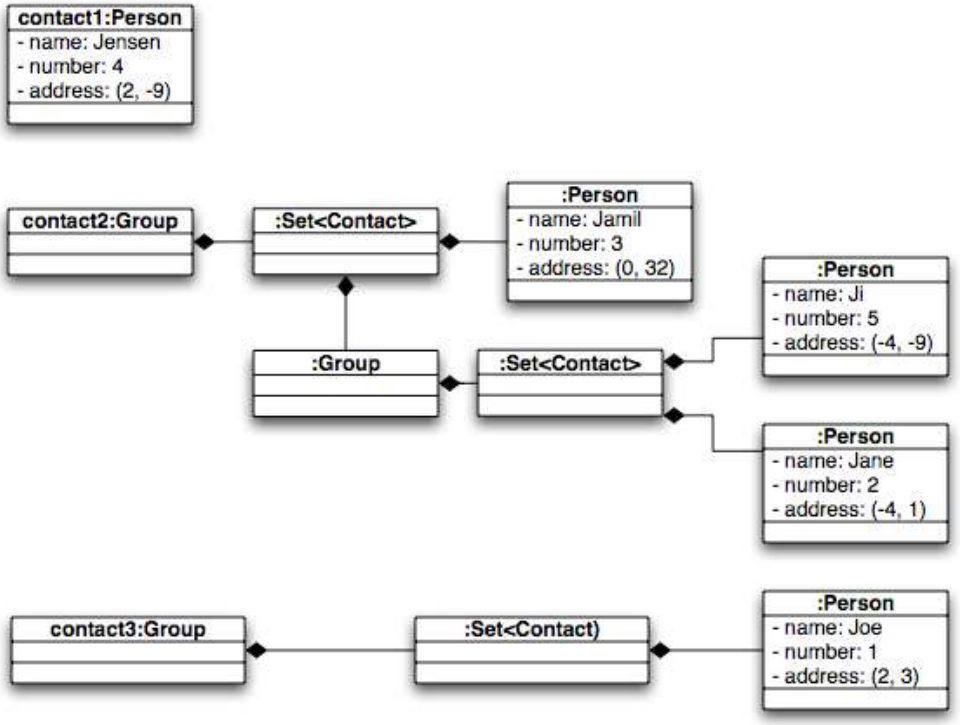
1. Implement interface **Contact**, and class **Group**. Note that a group can contain another group. The specification of method **getPeople** is that it returns a set containing all the person objects that are (either directly or indirectly) in the group. You may assume that cycles of group membership (such as a group being added to itself) will not occur.
2. a) Implement the **EmergencyDialler** class. This class will be used by the sensor subsystems. Each sensor subsystem will use an **EmergencyDialler** object that has been pre-configured with the priority queue that corresponds to the specific leak that the sensor subsystem detects. The configuration of an **EmergencyDialler** is performed using the method **addToEmergencyContactList** which given a contact calculates its priority and adds it into the priority queue accordingly. The **emergency** method is called by the sensor subsystem to trigger the calling of a contact in the queue. The **emergency** method returns a reference to the Contact that was successfully contacted. The **EmergencyDialler** object will call people using a **Dialler** object for which a reference is given when constructing the **EmergencyDialler** object. The calls shall be done according to the requirements described above.
   b) Implement the class **AvgEmergencySystem**. This class implements the criteria based on averages for determining the priority of a group.
3. Write the interface **PriorityQueue<T>** that extends the Java.lang interface **Iterable**. Specify for each method its post-condition with appropriate exceptions.
4. Consider the following data structure:



Each entry in the queue has a priority (P) of basic type double, and value

(V) of type Contact. The data structure for stopping the entry in the queue is provided to you by means of the class **Node<E>** which includes standard accessor and mutator methods.
Assume the priority order to be defined as follows: the lower the value P of an entry, the higher the priority of that Value.
Using the above data structure implement the class **LinkedListPriorityQueue<T>** that implements the interface **PriorityQueue<T>**
Note that you may leave the method **remove** of the PriorityQueue **Iterator** class as a method that does not do anything (i.e. it simply returns without removing anything from the queue)

5. Write a Java program as a main method in a class called **BuildEmergencyDialler**. The program should create an **EmergencyDialler** that uses the priority calculation criterion based on averages and that has a contact queue that stores the following three contacts: **contact1** of type **Person** and **contact2** and **contact3** of type **Group.** The contents of the groups can be deduced from the following object diagram.
Note that the contacts should be hard coded into your program(i.e. your program should not read any input from the keyboard)



The five parts carry, respectively, 20%, 25%, 5% , 45% and 5% of the marks.