# Lab 8

| | Student Name | Student CSUSM ID | Contribution percentage |
|---|---|---|---|
| 1 | John Paul Evert | evert005 | 20 |
| 2 | Nathan Hefler | hefle005 | 20 |
| 3 | Sirena Murphree | murph135 | 40 |
| **4** | Paul Nguyen | nguye403 | 20 |

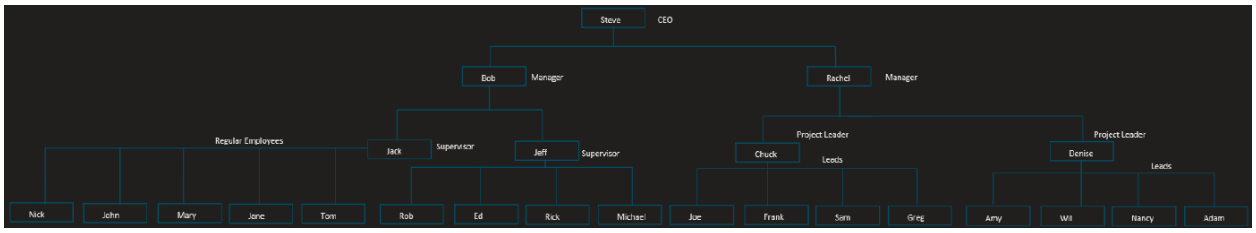
## Grading Rubrics (for instructor only):

| Criteria | 1. Beginning | 2. Developing | 3. Proficient | 4. Exemplary |
|---|---|---|---|---|
| **Modeling** | 0-14 | 15-19 | 20-24 | 25-30 |
| | | | | |
| **Program: functionality** ***correctness*** | 0-9 | 10-14 | 15-19 | 20 |
| | | | | |
| **Program: functionality** ***Behavior Testing*** | 0-9 | 10-14 | 15-19 | 20 |
| | | | | |
| **Program: quality** -> ***Readability*** | 0-2 | 3-5 | 6-9 | 10 |
| | | | | |
| **Program: quality** -> ***Modularity*** | 0-2 | 3-5 | 6-9 | 10 |
| | | | | |
| **Program: quality** -> ***Simplicity*** | 0-2 | 3-5 | 6-9 | 10 |
| | | | | |
| **Total Grade (100)** | | | | |

## SE 471 Software Architecture

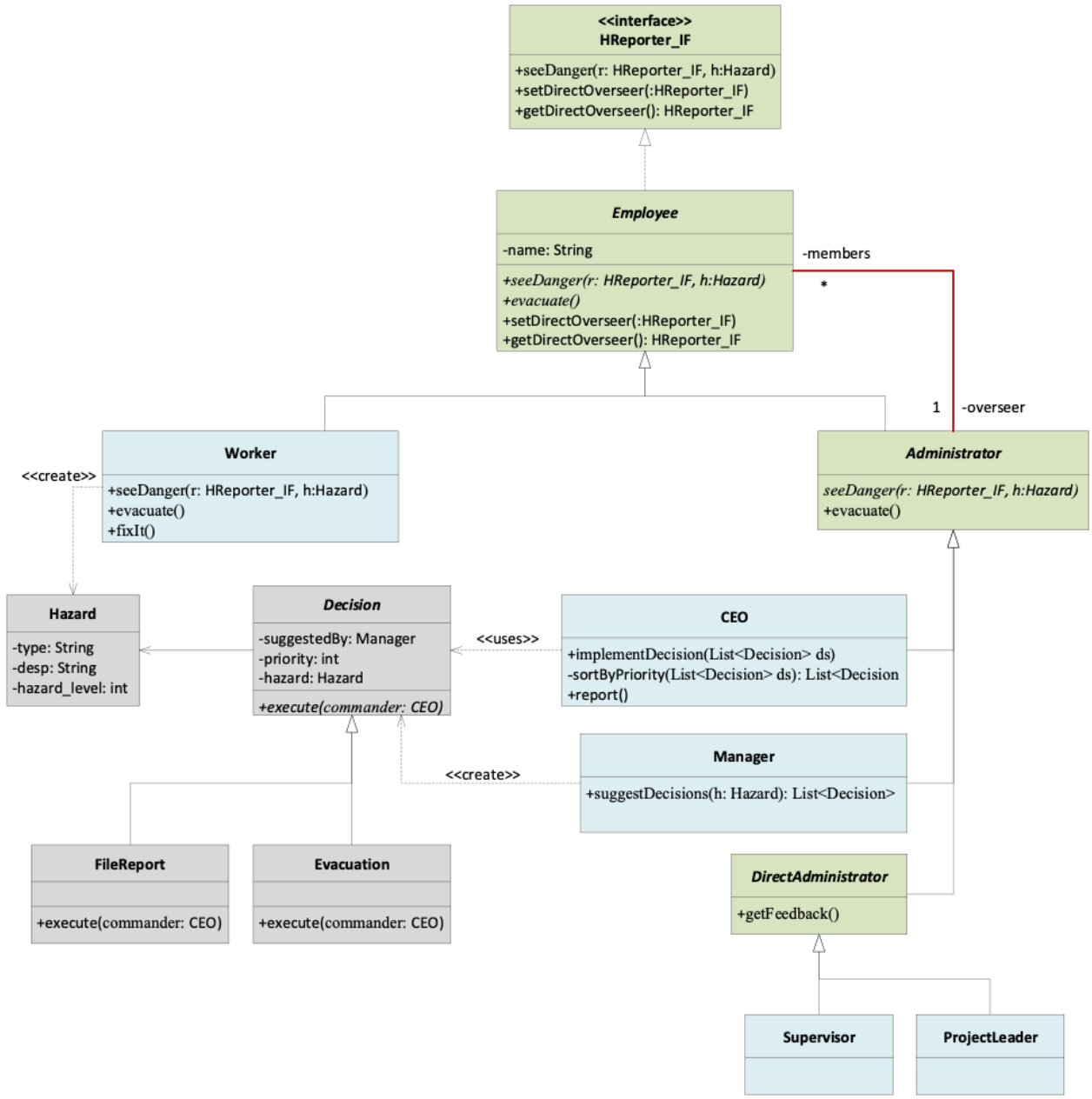
## Problems:

Company XYZ has a hazard control model. There are two supervisors Jack and Jeff, and two project leaders Chuck and Denise. Jack supervises regular employees (workers) John, Mary, Jane, Tom and Nick. Jeff supervises Rob, Ed, Rick and Michael. Chuck leads Joe, Frank, Sam and Greg. Denise leads Amy, Wil, Nancy and Adam. Bob is the manager of Jack and Jeff, while Rachel is the manager of Chuck and Denise. The CEO is Steve. Their relations are shown below.




When a regular employee (worker) identifies a danger, the issue is reported to his/her supervisor or project leader. The supervisor or leader will announce the trouble to all regular employees in his/her team and also report it to the manager in charge. The manager will collect feedbacks from his/her managed supervisors or leaders and contact the CEO if necessary. The CEO will collect decisions suggested by all managers. Eventually, the CEO picks the final decisions.

Below is an architecture design of the system.




Here is a scenario. A worker John observed a gas leak of a big tank and triggered the method "void seeDanger()" to report it to his supervisor. The supervisor ran "void seeDanger()" to tell all his team members to perform fixIt() and also inform his manager. The fixIt() method prints out a message like "The employee [name] is fixing it." The manager ran "void seeDanger()" to handle the danger by asking feedbacks from all supervisors/leaders under his management and contacting the CEO in case the feedbacks are all positive (true). Each supervisor or leader object has a "boolean getFeedback()" method, displaying "Feedback by [name]" and returning true if the hazard needs to be reported to the upper level of administration. The CEO ran "void

seeDanger()" to collect suggested decisions from the managers who performed their "suggestDecisions(h: Hazard): List<Decision>" method. The CEO makes up his final decisions by method "implementDecision(List<Decision> ds)".

After sorting all the decisions that he received by priorities, assume that the CEO always chooses to implement the first two decisions. Each decision has a method execute (:CEO). Assume there are two types of decisions. The first decision is of type Evacuation and the second decision is of type FileReport. The execute () of the FileReport decision simply displays "The city's environmental department is notified". The execute () of the Evacuation decision demands all employees in the company to evacuate. The evacuation is initiated by the CEO directly and the evacuation execution must start with all workers first, then supervisors or leaders, managers next, and finally the CEO. When a person's evacuate() method is called it displays "The employee [name] is evacuating".

## Solution:

- First, remember to zip the src folder of your project and submit the zip file to the ungraded assignment named "Lab8CodeSubmission". One submission from each team.
- Please explain the design pattern(s) being used in the design.
- Implement the design in Java. Paste a screenshot of a run of your program here.
- Also paste all you source code here.
- Save this report in PDF, and submit the pdf report to the graded assignment named "Lab8ReportSubmission". One submission from each team.

The design pattern being used in this design is the Chain of Responsibility. Chain of responsibility will have a worker see danger, and begin escalating the hazard notification up the chain of responsibility. At each level in the chain there is a different process defined in seeDanger that determines the next level of escalation.  At the top level CEO makes a final decision about how to handle the item escalated to them.

```
/Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=57798
:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -
classpath /Users/smurph/Documents/- - - - - - - - School - - - - - - - - - /
SE471 Software Architecture/ASSIGNMENT/SE471_LABS/SE471_LABS/
8_ChainResponseibility/out/production/8_ChainResponseibility src.Main
Worker John has observed something hazardous.
What type of hazard is this?
    [1] - Biological
    [2] - Chemical
    [3] - Physical
    [4] - Safety
    [5] - Ergonomic
    [6] - Psychosocial
1
On a scale from 1(low) - 10(high), how dangerous is this hazard?
9
Please provide a short description of this hazard:
Bodily fluids in pool
 -> John is fixing Biological Hazard - Bodily fluids in pool.
Jack instructs team members to fix Biological Hazard - Bodily fluids in
pool
 -> Nancy is fixing Biological Hazard - Bodily fluids in pool.
 -> Mary is fixing Biological Hazard - Bodily fluids in pool.
 -> Jane is fixing Biological Hazard - Bodily fluids in pool.
 -> Tom is fixing Biological Hazard - Bodily fluids in pool.

Feedback from Jeff is true.

Feedback from Jack is true.
Manager Bob, please suggest some decisions to the CEO regarding Biological
Hazard - Bodily fluids in pool.
Does the area need to be evacuated?
    [Yes] - Evacuate the area
    [No] - File an incident report
YES
How urgent is evacuating the area on a scale from 1(low) - 10(high)? 10
Would you like to make an alternate suggestion?
    [1] - Suggestion another decision
    [ANY] - Done
1
Does the area need to be evacuated?
    [Yes] - Evacuate the area
    [No] - File an incident report
No
How urgent does a report need to be filed on a scale from 1(low) - 10(high
)? 8
Would you like to make an alternate suggestion?
    [1] - Suggestion another decision
    [ANY] - Done
0
Your suggestions have been recorded.
Manager Rachel, please suggest some decisions to the CEO regarding
```

```
Biological Hazard - Bodily fluids in pool.
Does the area need to be evacuated?
    [Yes] - Evacuate the area
    [No] - File an incident report
no
How urgent does a report need to be filed on a scale from 1(low) - 10(high
)? 9
Would you like to make an alternate suggestion?
    [1] - Suggestion another decision
    [ANY] - Done
0
Your suggestions have been recorded.
Execute Evacuation Plan: suggested by Bob for Biological Hazard - Bodily
fluids in pool
Evacuating members first ...
Evacuating members first ...
Evacuating members first ...
Rob has been evacuated.
Ed has been evacuated.
Rick has been evacuated.
Michael has been evacuated.
Jeff has evacuated now that their members have evacuated.
Evacuating members first ...
Nancy has been evacuated.
John has been evacuated.
Mary has been evacuated.
Jane has been evacuated.
Tom has been evacuated.
Jack has evacuated now that their members have evacuated.
Bob has evacuated now that their members have evacuated.
Evacuating members first ...
Evacuating members first ...
Joe has been evacuated.
Frank has been evacuated.
Sam has been evacuated.
Greg has been evacuated.
Chuck has evacuated now that their members have evacuated.
Evacuating members first ...
Amy has been evacuated.
Will has been evacuated.
Nancy has been evacuated.
Adam has been evacuated.
Denise has evacuated now that their members have evacuated.
Rachel has evacuated now that their members have evacuated.
Steve has evacuated now that their members have evacuated.
Execute File Report: suggested by Rachel for Biological Hazard - Bodily
fluids in pool
The city's environmental department is notified

Process finished with exit code 0
```

```
/Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=57837
:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -
classpath /Users/smurph/Documents/- - - - - - - - School - - - - - - - - - /
SE471 Software Architecture/ASSIGNMENT/SE471_LABS/SE471_LABS/
8_ChainResponseibility/out/production/8_ChainResponseibility src.Main
Worker John has observed something hazardous.
What type of hazard is this?
    [1] - Biological
    [2] - Chemical
    [3] - Physical
    [4] - Safety
    [5] - Ergonomic
    [6] - Psychosocial
5
On a scale from 1(low) - 10(high), how dangerous is this hazard?
2
Please provide a short description of this hazard:
Mike Slouches
 -> John is fixing Ergonomic Hazard - Mike Slouches.
Jack instructs team members to fix Ergonomic Hazard - Mike Slouches
 -> Nancy is fixing Ergonomic Hazard - Mike Slouches.
 -> Mary is fixing Ergonomic Hazard - Mike Slouches.
 -> Jane is fixing Ergonomic Hazard - Mike Slouches.
 -> Tom is fixing Ergonomic Hazard - Mike Slouches.

Feedback from Jeff is false.

Feedback from Jack is false.
I fail to see why the CEO needs to be informed of this.

Process finished with exit code 0
```


# SE 471 Software Architecture

*Main.java*

```
package src;

import src.Personel.*;

public class Main {

   public static void main(String[] args) {
   // build management hierarchy
       // make employees
       Worker nick = new Worker ("Nick");
       Worker john = new Worker ("John");
       Worker mary = new Worker ("Mary");
       Worker jane = new Worker ("Jane");
       Worker tom = new Worker ("Tom");
       Worker rob = new Worker ("Rob");
       Worker ed = new Worker ("Ed");
       Worker rick = new Worker ("Rick");
       Worker michael = new Worker ("Michael");
       Worker joe = new Worker ("Joe");
       Worker frank = new Worker ("Frank");
       Worker sam = new Worker ("Sam");
       Worker greg = new Worker ("Greg");
       Worker amy = new Worker ("Amy");
       Worker will = new Worker ("Will");
       Worker nancy = new Worker ("Nancy");
       Worker adam = new Worker ("Adam");

       Supervisor jack = new Supervisor ("Jack");
       Supervisor jeff = new Supervisor ("Jeff");

       ProjectLeader chuck = new ProjectLeader ("Chuck");
       ProjectLeader denise = new ProjectLeader ("Denise");

       Manager bob = new Manager ("Bob");
       Manager rachel = new Manager ("Rachel");

       CEO steve = new CEO ("Steve");

       //link employees
       // Jack
       jack.addMember(nancy);
       jack.addMember(john);
       jack.addMember(mary);
       jack.addMember(jane);
       jack.addMember(tom);
       nick.setDirectOverseer(jack);
       john.setDirectOverseer(jack);
       mary.setDirectOverseer(jack);
```

```
jane.setDirectOverseer(jack);
tom.setDirectOverseer(jack);

// Jeff
jeff.addMember(rob);
jeff.addMember(ed);
jeff.addMember(rick);
jeff.addMember(michael);
rob.setDirectOverseer(jeff);
ed.setDirectOverseer(jeff);
rick.setDirectOverseer(jeff);
michael.setDirectOverseer(jeff);

//Bob
bob.addMember(jeff);
bob.addMember(jack);
jack.setDirectOverseer(bob);
jeff.setDirectOverseer(bob);

//Chuck
chuck.addMember(joe);
chuck.addMember(frank);
chuck.addMember(sam);
chuck.addMember(greg);
joe.setDirectOverseer(chuck);
frank.setDirectOverseer(chuck);
sam.setDirectOverseer(chuck);
greg.setDirectOverseer(chuck);

//Denise
denise.addMember(amy);
denise.addMember(will);
denise.addMember(nancy);
denise.addMember(adam);
amy.setDirectOverseer(denise);
will.setDirectOverseer(denise);
nancy.setDirectOverseer(denise);
adam.setDirectOverseer(denise);

//Rachel
rachel.addMember(chuck);
rachel.addMember(denise);
chuck.setDirectOverseer(rachel);
denise.setDirectOverseer(rachel);

//Steve
steve.addMember(bob);
steve.addMember(rachel);
bob.setDirectOverseer(steve);
rachel.setDirectOverseer(steve);
```

```
        john.seeDanger(null, null);
    }
}
```

---

```
//Decision.java
package src.Actions;

import src.Personel.CEO;
import src.Personel.Manager;

public abstract class Decision {

    /**
     * the manager that suggested the plan
     */
    protected Manager suggestedBy;

    /**
     * the priority level
     */
    protected int priority;

    /**
     * the hazard that is to be resolved by this plan
     */
    protected Hazard hazard;

    /**
     * constructor
     * @param suggestedBy
     * @param priority
     * @param hazard
     */
    public Decision(Manager suggestedBy, int priority, Hazard hazard) {
        this.suggestedBy = suggestedBy;
        this.priority = priority;
        this.hazard = hazard;
    }

    /**
     * execute this plan
     * @param commander
     */
    public abstract void execute(CEO commander);

    /**
     * get the priority of this decision
     * @return
```

```
     */
    public int getPriority() {
        return priority;
    }
}
```

---

```
//Evacuation.java
package src.Actions;

import src.Personel.CEO;
import src.Personel.Manager;

public class Evacuation extends Decision{
    /**
     * constructor
     * @param suggestedBy
     * @param priority
     * @param hazard
     */
    public Evacuation(Manager suggestedBy, int priority, Hazard hazard) {
        super(suggestedBy, priority, hazard);
    }

    /**
     * execute this plan
     *
     * @param commander
     */
    @Override
    public void execute(CEO commander) {
            System.out.printf("Execute Evacuation Plan: suggested by %s for %s\n",
suggestedBy.getName(), hazard.toString());
        commander.evacuate();
    }
}
```

---

```
//FileReport.java
package src.Actions;

import src.Personel.CEO;
import src.Personel.Manager;

public class FileReport extends Decision{
    /**
     * constructor
     *
     * @param suggestedBy
     * @param priority
     * @param hazard
     */
```

```
    public FileReport(Manager suggestedBy, int priority, Hazard hazard) {
        super(suggestedBy, priority, hazard);
    }

    /**
     * execute this plan
     *
     * @param commander
     */
    @Override
    public void execute(CEO commander) {
            System.out.printf("Execute File Report: suggested by %s for %s\n",
suggestedBy.getName(), hazard.toString());
        commander.report();
    }
}
```

---

```
//Hazard.java
package src.Actions;

public class Hazard {

    /**
     * type of hazard
     */
    private String type;

    /**
     * short description
     */
    private String desp;

    /**
     * the seriousness of the hazard
     */
    private int hazard_level;

    /**
     * constructor
     * @param type
     * @param desp
     * @param hazard_level
     */
    public Hazard(String type, String desp, int hazard_level) {
        this.type = type;
        this.desp = desp;
        this.hazard_level = hazard_level;
    }
```

```
    /**
     * get a string that describes this hazard
     * @return complete hazard type and description
     */
    public String toString(){
        return String.format( "%s Hazard - %s" , type, desp);
    }

    /**
     * get the description of the hazard
     * @return hazard description
     */
    public String getDesp() {
        return desp;
    }

    /**
     * get the seriousness of the hazard
     * @return hazard level
     */
    public int getHazard_level() {
        return hazard_level;
    }
}
```

---

```
//IReporterHazard.java
package src.Personel;

import src.Actions.Hazard;

public interface IReporterHazard {
    public void seeDanger(IReporterHazard reporter, Hazard hazard);
    public void setDirectOverseer(IReporterHazard director);
    public IReporterHazard getDirectOverseer();
}
```

---

```
//Employee.java
package src.Personel;

import src.Actions.Hazard;

public abstract class Employee implements IReporterHazard
{
    /**
     * employee name
     */
    protected String name;

    /**
```

```
    * employee's boss (direct overseer)
    */
   protected Administrator overseer;

   public Employee(String name) {
       this.name = name;
   }

   /**
        * Worker - triggered the method to report hazard to his overseer(supervisor)
    * Supervisor - tell all his team members to perform fixIt() and also inform their overseer(manager)
        * Manager - handle the danger by asking feedbacks from all supervisors/leaders under his management
    *            and contacting the CEO in case the feedbacks are all positive(T)
    * CEO - collect suggested decisions from the managers who performed their suggestDecisions method
    * @param reporter
    * @param hazard
    */
   public abstract void seeDanger(IReporterHazard reporter, Hazard hazard);

   /**
    *   Leave the immediate area
    */
   public abstract void evacuate();

   /**
    * set this employee's direct overseer
    * @param director boss
    */
   public void setDirectOverseer(IReporterHazard director){
       this.overseer = (Administrator)director;
   }

   /**
    * get this employee's direct overseer
    * @return IReporterHazard - boss
    */
   public IReporterHazard getDirectOverseer(){
       return overseer;
   }

   /**
    * get the name of this person
    * @return this person's name
    */
   public String getName(){
       return name;
```

```
    }
}//Administrator.java
package src.Personel;

import src.Actions.Hazard;
import java.util.ArrayList;
import java.util.List;

public abstract class Administrator extends Employee{
    /**
     * this admin's direct employee's
     */
    protected List<Employee> members;

    /**
     * constructor
     * @param name
     */
    public Administrator(String name) {
        super(name);
        members = new ArrayList<Employee>();
    }

    /**
     * add direct employee to this administrator's members
     * @param newTeamMember
     */
    public void addMember(Employee newTeamMember){
        members.add(newTeamMember);
    }

    /**
      * Supervisor - tell all his team members to perform fixIt() and also inform
their overseer(manager)
        * Manager -      handle the danger by asking feedbacks from all
supervisors/leaders under his management
     *            and contacting the CEO in case the feedbacks are all positive(T)
      * CEO -         collect suggested decisions from the managers who performed
their suggestDecisions method
     * @param reporter
     * @param hazard
     */
    @Override
    public abstract void seeDanger(IReporterHazard reporter, Hazard hazard);

    /**
     * Evacuates all members, then evacuates self
     */
    @Override
```

```
   public void evacuate() {
        System.out.println("Evacuating members first ... ");
        for(Employee e: members){
            e.evacuate();
        }
           System.out.println(name + " has evacuated now that their members have
evacuated.");
   }
}
```

---

```
//CEO.java
package src.Personel;

import src.Actions.Decision;
import src.Actions.Hazard;

import java.util.ArrayList;
import java.util.List;

public class CEO extends Administrator{
   /**
    * constructor
    *
    * @param name
    */
   public CEO(String name) {
        super(name);
   }

   /**
     * Supervisor - tell all his team members to perform fixIt() and also inform
their overseer(manager)
         * Manager -      handle the danger by asking feedbacks from all
supervisors/leaders under his management
    * and contacting the CEO in case the feedbacks are all positive(T)
     * CEO -         collect suggested decisions from the managers who performed
their suggestDecisions method
    *
    * @param reporter
    * @param hazard
    */
   @Override
   public void seeDanger(IReporterHazard reporter, Hazard hazard) {
        //collect suggestions
        List<Decision> gatheredDS = new ArrayList<>();
        for(Employee admin: members){
            if(admin instanceof Manager)
                gatheredDS.addAll(((Manager)admin).suggestDecisions(hazard));
        }
        implementDecision(gatheredDS);
```

```
    }

    /**
     * given a list of decisions, order the list by priority
     * @param ds a list of Decisions
     * @return an ordered list of Decisions
     */
    private List<Decision> sortByPriority(List<Decision> ds){
        List<Decision> orderedDS = new ArrayList<>();
        while(!ds.isEmpty()){
            Decision topPriority = ds.remove(0);
            for (Decision d: ds){
                if(d.getPriority() > topPriority.getPriority())
                    topPriority = d;
            }
            orderedDS.add(topPriority);
        }
        return orderedDS;
    }

    /**
     * given a list of decision the CEO sorts through them and executes some of the
decisions
     * @param ds a list of decisions
     */
    public void implementDecision(List<Decision> ds){
        ds = sortByPriority(ds);
        int dsToBeExecuted = 2;
        while (!ds.isEmpty() && dsToBeExecuted-- > 0)
            ds.remove(0).execute(this);
    }

    public void report(){
        System.out.println("The city's environmental department is notified");
    }
}
```

---

```
//Manager.java
package src.Personel;

import src.Actions.Decision;
import src.Actions.Evacuation;
import src.Actions.FileReport;
import src.Actions.Hazard;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Scanner;
```

```
public class Manager extends Administrator{
    /**
     * constructor
     * @param name
     */
    public Manager(String name) {
        super(name);
    }

    /**
   * Manager -   handle the danger by asking feedbacks from all supervisors/leaders under his management
     * and contacting the CEO in case the feedbacks are all positive(T)
     *
     * @param reporter
     * @param hazard
     */
    @Override
    public void seeDanger(IReporterHazard reporter, Hazard hazard) {
        //ask for feedback
        boolean tellCEO = true;
        for(Employee e: members){
            if(e instanceof DirectAdministrator)
                        tellCEO = ((DirectAdministrator)e).getFeedBack(hazard) && tellCEO;
        }

        if(tellCEO && overseer != null)
            overseer.seeDanger(this, hazard);
        else{
             System.out.println("I fail to see why the CEO needs to be informed of this.");
        }
    }

    /**
     * ask the manager to come up with some suggestions
     * @param hazard
     * @return List<Decision> this managers suggested decisions
     */
    public List<Decision> suggestDecisions(Hazard hazard){
        List<Decision> mySuggestions = new ArrayList<>();

        Scanner choice = new Scanner(System.in);
          System.out.printf("Manager %s, please suggest some decisions to the CEO regarding %s.\n", name, hazard.toString());
        do {
              System.out.println("Does the area need to be evacuated?\n\t[Yes] - Evacuate the area\n\t[No] - File an incident report");
```

```
                                                    boolean     shouldEvacuate     =
choice.next().toLowerCase(Locale.ROOT).indexOf('y') >= 0;

            System.out.printf("How urgent %s on a scale from 1(low) - 10(high)? ",
shouldEvacuate ? "is evacuating the area" : "does a report need to be filed");
           int priority = choice.nextInt();

              mySuggestions.add(shouldEvacuate ? new Evacuation(this, priority,
hazard) : new FileReport(this, priority, hazard));

                    System.out.println("Would you like to make an alternate
suggestion?\n\t[1] - Suggestion another decision\n\t[ANY] - Done");
       }while(choice.nextInt()==1);
       System.out.println("Your suggestions have been recorded.");

       return mySuggestions;
   }
}
```

---

```
//DirectAdministrator.java
package src.Personel;

import src.Actions.Hazard;

public abstract class DirectAdministrator extends Administrator{

   private final int HAZARD_LEVEL_THRESHOLD = 5;
   /**
    * constructor
    *
    * @param name
    */
   public DirectAdministrator(String name) {
       super(name);
   }

   /**
      * Each supervisor or leader object has a "boolean getFeedback()" method,
displaying "Feedback by [name]"
     * @return true if the hazard needs to be reported to the upper level of
administration
    */
   public boolean getFeedBack(Hazard hazard){
       boolean feedback = hazard.getHazard_level() >= HAZARD_LEVEL_THRESHOLD;
       System.out.printf("\nFeedback from %s is %b.\n", name, feedback);
       return feedback;
   }

   /**
```

```
     * Supervisor - tell all his team members to perform fixIt() and also inform
their overseer(manager)
         * Manager -      handle the danger by asking feedbacks from all
supervisors/leaders under his management
     * and contacting the CEO in case the feedbacks are all positive(T)
      * CEO -         collect suggested decisions from the managers who performed
their suggestDecisions method
     *
     * @param reporter
     * @param hazard
     */
    @Override
    public void seeDanger(IReporterHazard reporter, Hazard hazard) {
            System.out.printf("%s instructs team members to fix %s\n", name,
hazard.toString());
        for(Employee e: members){
            if(e != reporter){
                e.seeDanger(this, hazard);
            }
        }
        if(overseer != null)
            overseer.seeDanger(reporter, hazard);
    }
}
```

---

```
// Supervisor.java
package src.Personel;

public class Supervisor extends DirectAdministrator{
    /**
     * constructor
     *
     * @param name
     */
    public Supervisor(String name) {
        super(name);
    }
}
```

---

```
//ProjectLeader.java
package src.Personel;

public class ProjectLeader extends DirectAdministrator{
    /**
     * constructor
     *
     * @param name
     */
    public ProjectLeader(String name) {
        super(name);
```

```
    }

}
```

---

```
// Worker.java
package src.Personel;

import src.Actions.Hazard;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Worker extends Employee{
    public Worker(String name) {
        super(name);
    }

    /**
     * Worker -      triggered the method [seeDanger] to report hazard to his overseer(supervisor)
     * @param reporter
     * @param hazard
     */
    @Override
    public void seeDanger(IReporterHazard reporter, Hazard hazard)
    {
        if(reporter != null){
            fixIt(hazard);
        }else{
            hazard = documentHazard();
            fixIt(hazard);
            overseer.seeDanger(this, hazard);
        }
    }

    /**
     * create a new hazard
     * @return hazard
     */
    private Hazard documentHazard(){
        Scanner observationDetails = new Scanner(System.in);

        List<String> types = new ArrayList<String>(){
            {
                add("Biological");
                add("Chemical");
                add("Physical");
                add("Safety");
                add("Ergonomic");
```

```
                add("Psychosocial");
            }
        };

        System.out.printf("Worker %s has observed something hazardous.\n", name);
        System.out.println("What type of hazard is this?");
        int i = 1;
        for (String type: types){
            System.out.printf("\t[%d] - %s\n", i++, type);
        }
        int typeSelection = observationDetails.nextInt()-1;
        if(typeSelection<0 || typeSelection >= types.size())
              typeSelection = types.indexOf("Safety");// if they select something
weird default to safety hazard
        String type = types.get(typeSelection);

          System.out.println("On a scale from 1(low) - 10(high), how dangerous is
this hazard?");
        int level = observationDetails.nextInt();

        System.out.println("Please provide a short description of this hazard:");
        String desp = observationDetails.next();
        desp += observationDetails.nextLine();

        return new Hazard(type, desp, level);
    }

    /**
     * Announce that this worker has evacuated.
     */
    @Override
    public void evacuate() {
        System.out.println(name + " has been evacuated." );
    }

    /**
     * Announce that this worker is fixing the hazard
     * @param hazard the hazard being fixed
     */
    public void fixIt(Hazard hazard){
        System.out.printf(" -> %s is fixing %s.\n", name, hazard.toString());
    }
}
```