Applied Software Engineering Assignment 3

Stefan M Ahmed

21359035

02/21

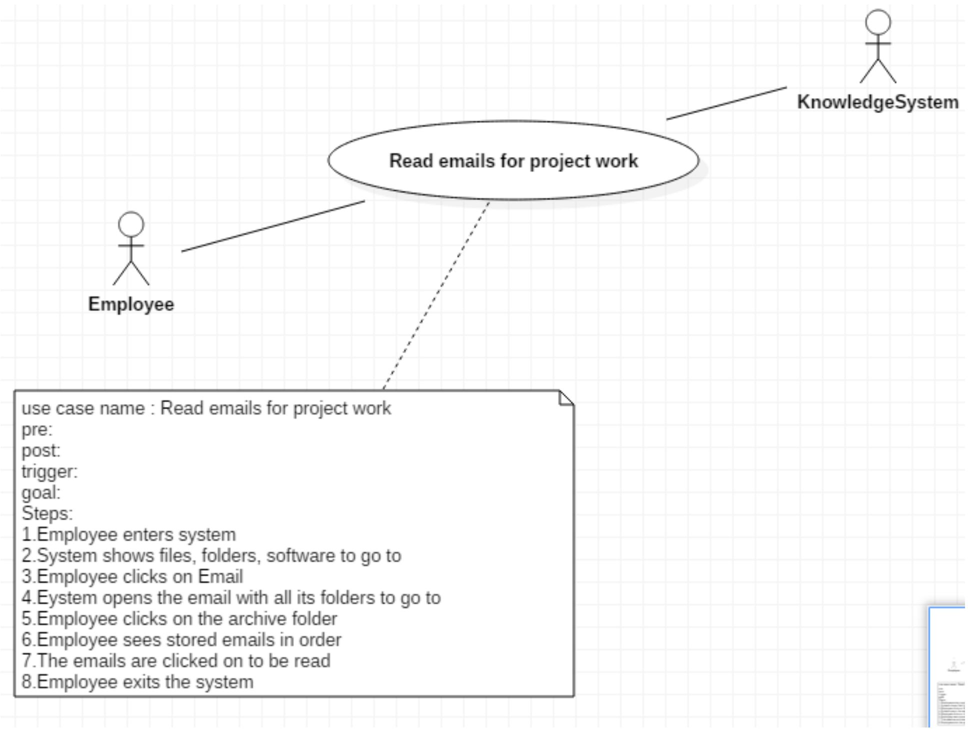
Intro

Hill and Knowlton the public relations firm focus their work on projects. Therefore, knowledge is worth them having to share with each other when doing the work on the project. Unfortunately, they were having problems with their knowledge system and needed to get a new one that employees would actually use. There was not much reason to use the current system. The data were not accurate enough or kept up to date. Finding ways to fix the problems from the reasons not to use it they managed to get a new system that they would actually use.

The system having software that allows data to be put in order and kept up to date. Emails can now be stores in order such as date. Emails of past projects can be stored in the archive for employees to read when given a project to do. The knowledge can then be shared by them to do their project tasks. In fact, all types of knowledge is stored in the knowledge system for them to get to and share.

Tasks

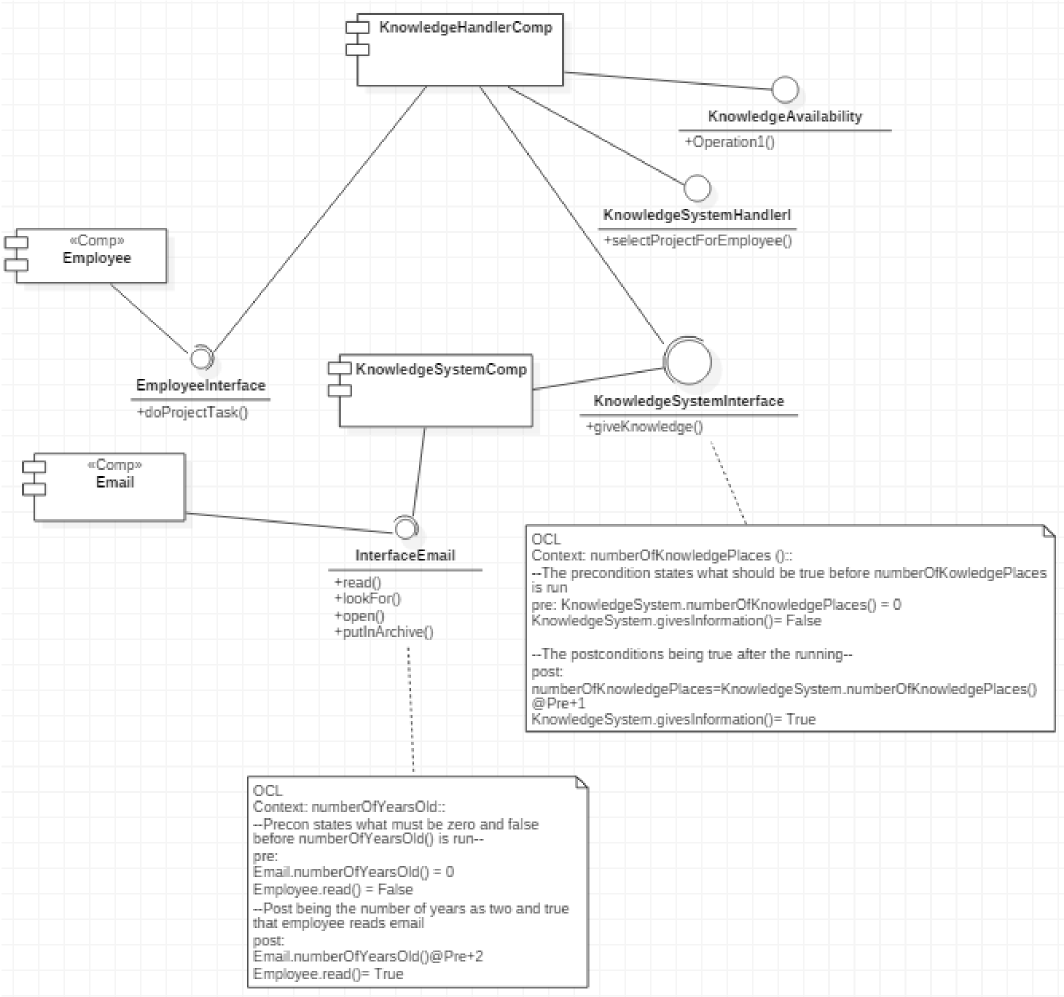
This coursework is a continuation of my read emails use case. Employees are the actors outside the system. The go into the system to go to the emails to read them. This is to help them do the project task when they are given the project to do. I must put the work I have done in the form of UML diagrams to the test by writing and running code for it in the programming language. The language I have decided to do it in is Java. This is basically the implementation task. I start with the UML diagrams including the class diagram:



use case name : Read emails for project work
pre:
post:
trigger:
goal:
Steps:
1.Employee enters system
2.System shows files, folders, software to go to
3.Employee clicks on Email
4.Eystem opens the email with all its folders to go to
5.Employee clicks on the archive folder
6.Employee sees stored emails in order
7.The emails are clicked on to be read
8.Employee exits the system

This use case with Employee outside the system. He or she reads the emails to do the project tasks when given the project to do. The emails are inside the system for the employee to go into to look through, open and read.

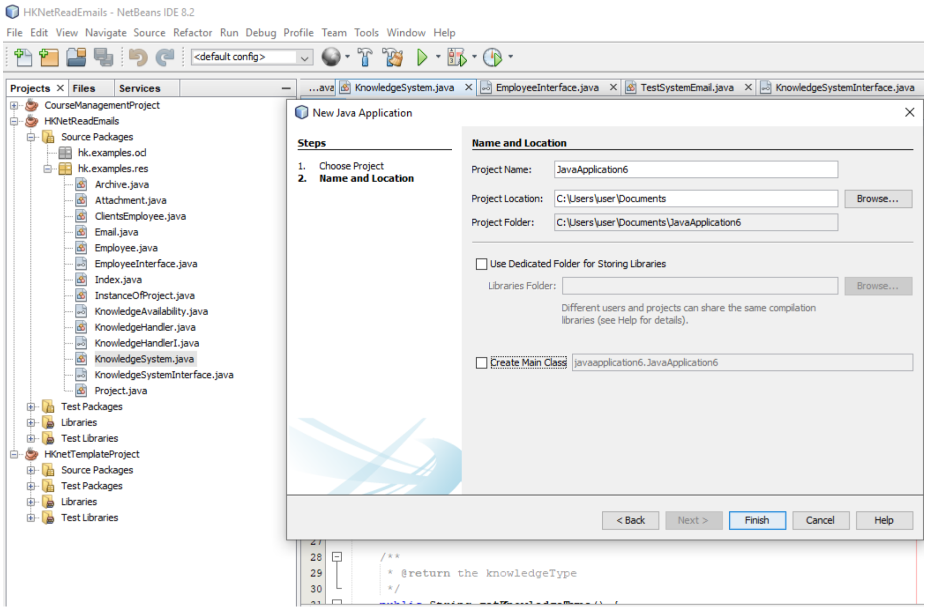
The class diagram showing the relationship between the classes for the use case. The use case is broken down into the classes. I am to write and run code in Java for these classes to test this use case.



```
OCL
Context: numberOfKnowledgePlaces ()::
--The precondition states what should be true before numberOfKowledgePlaces
is run
pre: KnowledgeSystem.numberOfKnowledgePlaces() = 0
KnowledgeSystem.givesInformation()= False

--The postconditions being true after the running--
post:
numberOfKnowledgePlaces=KnowledgeSystem.numberOfKnowledgePlaces()
@Pre+1
KnowledgeSystem.givesInformation()= True
```

```
OCL
Context: numberOfYearsOld::
--Precon states what must be zero and false
before numberOfYearsOld() is run--
pre:
Email.numberOfYearsOld() = 0
Employee.read() = False
--Post being the number of years as two and true
that employee reads email
post:
Email.numberOfYearsOld()@Pre+2
Employee.read()= True
```

Then there is the component diagram the class diagram puts into running. This means I am to write and run code to test the components for the classes. This includes the interface of the components which work with the methods of the classes.



Project Name: JavaApplication6

Project Location: C:\Users\user\Documents

Project Folder: C:\Users\user\Documents\JavaApplication6

☐ Use Dedicated Folder for Storing Libraries

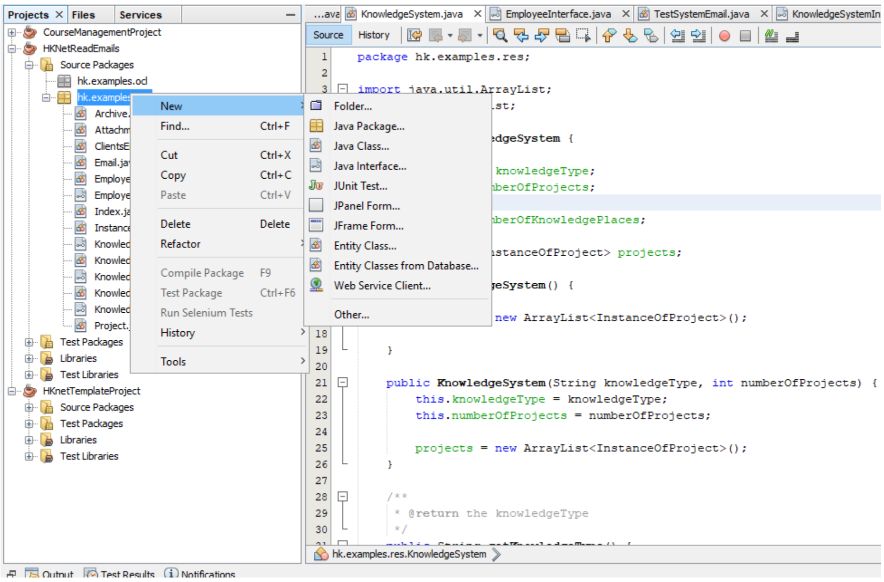
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class javaapplication6.JavaApplication6

```
    /**
     * @return the knowledgeType
     */
```

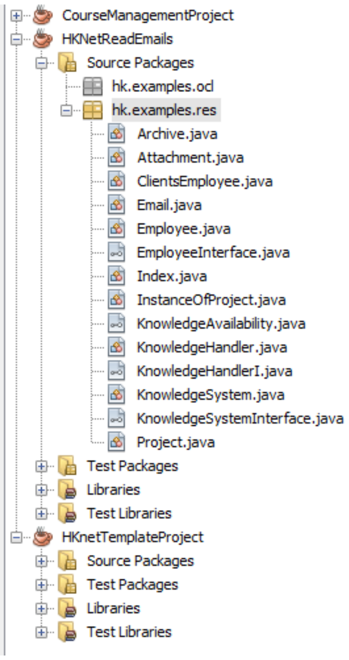
The programming begins with creating a new Java project on the IDE on NetBeans version 8.2. I Choose Java application, click next button and choose the right locations to keep the project. I decide to start the project without creating main classes for each file of my classes. The finish button is clicked to start the project.



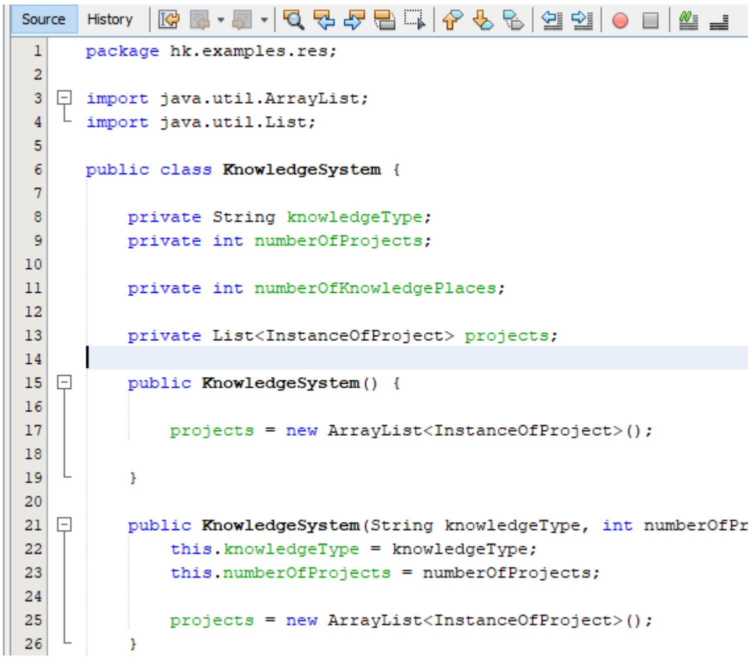
```
package hk.examples.res;

import java.util.ArrayList;
...st;

...dgeSystem {

...knowledgeType;
...berOfProjects;

...berOfKnowledgePlaces;

...stanceOfProject> projects;

...eSystem() {

...new ArrayList<InstanceOfProject>();

    }

    public KnowledgeSystem(String knowledgeType, int numberOfProjects) {
        this.knowledgeType = knowledgeType;
        this.numberOfProjects = numberOfProjects;

        projects = new ArrayList<InstanceOfProject>();
    }

    /**
     * @return the knowledgeType
     */
```

I can create classes in the folder by right clicking on the folder and just choosing Java Class. Give the class the right name with where it is located and just clock Finish.

KnowledgeSystem

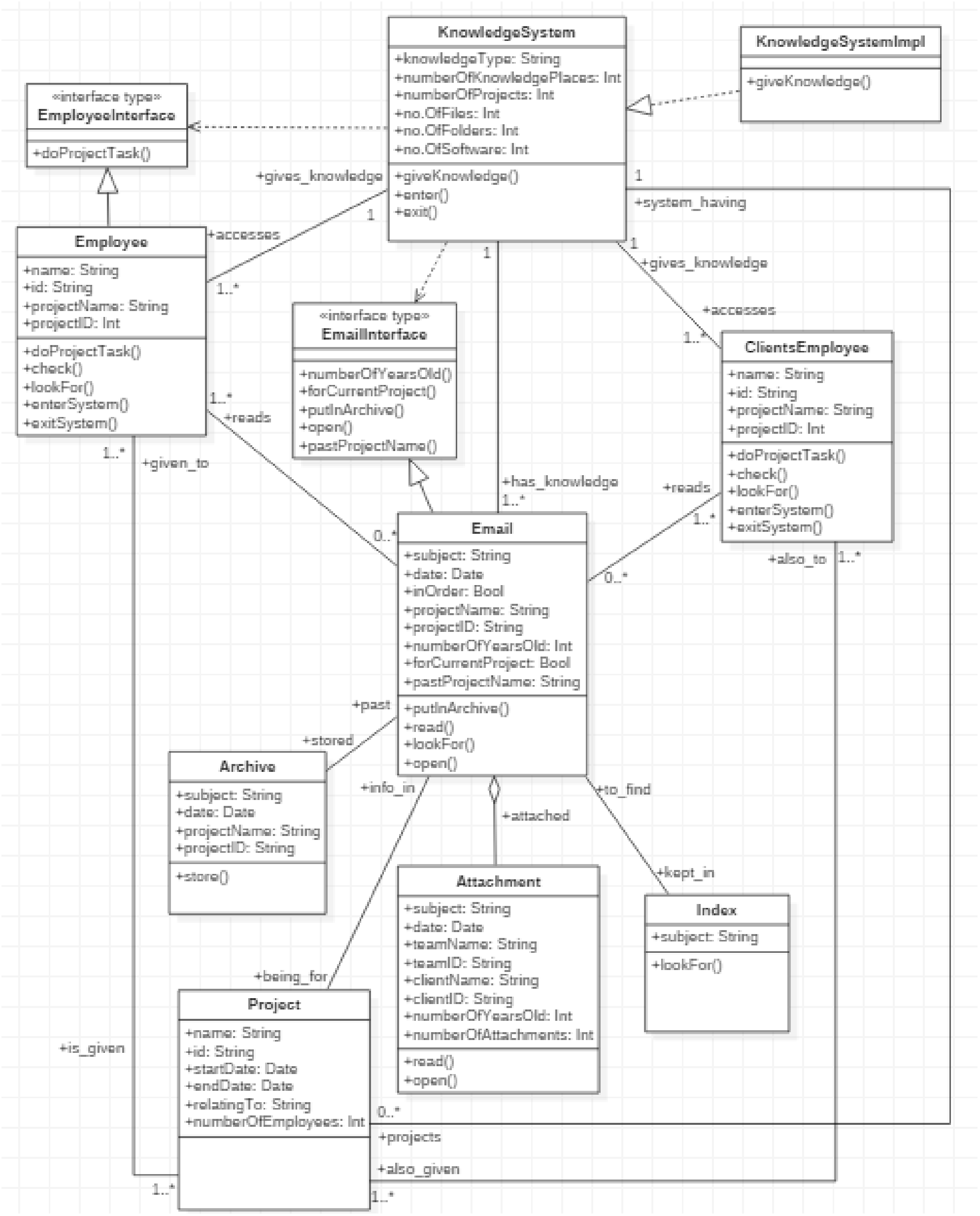


```
CourseManagementProject
HKNetReadEmails
    Source Packages
        hk.examples.ocl
        hk.examples.res
            Archive.java
            Attachment.java
            ClientsEmployee.java
            Email.java
            Employee.java
            EmployeeInterface.java
            Index.java
            InstanceOfProject.java
            KnowledgeAvailability.java
            KnowledgeHandler.java
            KnowledgeHandlerI.java
            KnowledgeSystem.java
            KnowledgeSystemInterface.java
            Project.java
    Test Packages
    Libraries
    Test Libraries
HKnetTemplateProject
    Source Packages
    Test Packages
    Libraries
    Test Libraries
```

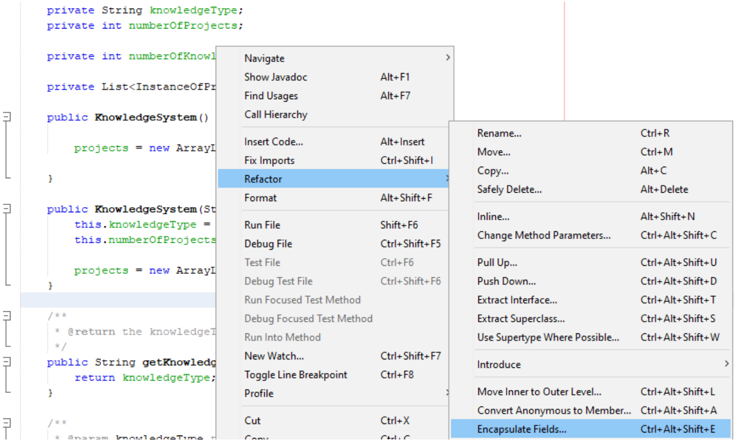


```java
package hk.examples.res;

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

public class KnowledgeSystem {

    private String knowledgeType;
    private int numberOfProjects;

    private int numberOfKnowledgePlaces;

    private List<InstanceOfProject> projects;

    public KnowledgeSystem() {

        projects = new ArrayList<InstanceOfProject>();

    }

    public KnowledgeSystem(String knowledgeType, int numberOfPr
        this.knowledgeType = knowledgeType;
        this.numberOfProjects = numberOfProjects;

        projects = new ArrayList<InstanceOfProject>();
    }
```

I now have my classes together in the folder. Starting with the KnolwedgeSystem class I have the attributes with their needed types String and int. The knowledgeType attribute being the type of knowledge which could be accounts, biographies or emails. The numberOfProjects is needed in the system so that is going to be a number which is int. This also means a list of projects is needed. Each attribute for the class are kept private. The methods are seen in the public.



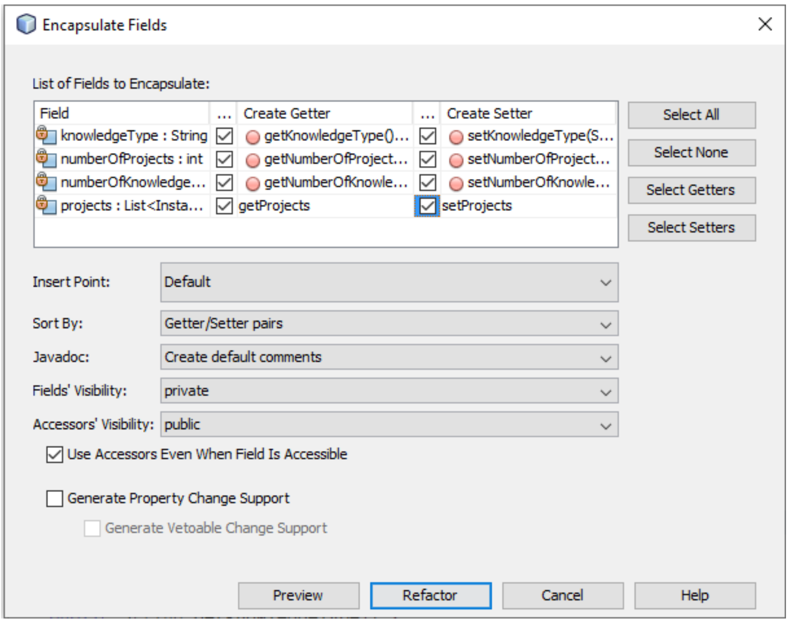
Looking at the class diagram between KnowledgeSystem and Project is the link. This is given a specific name "projects" that links it to the Project. This is the role name. Therefore "projects" is the list that is being created here. Created as an array where each information in the list is given in square brackets. I could have put this end point as "projects needed" and had that as the name of the list in the code but decided I want it to be "projects".



```
    private String knowledgeType;
    private int numberOfProjects;

    private int numberOfKnowl

    private List<InstanceOfPr

    public KnowledgeSystem()

        projects = new ArrayL

    }

    public KnowledgeSystem(St
        this.knowledgeType =
        this.numberOfProjects

        projects = new ArrayL
    }

    /**
     * @return the knowledgeT
     */
    public String getKnowledg
        return knowledgeType;
    }

    /**
     * @param knowledgeType t
```

Navigate
Show Javadoc Alt+F1
Find Usages Alt+F7
Call Hierarchy
Insert Code... Alt+Insert
Fix Imports Ctrl+Shift+I
Refactor
Format Alt+Shift+F
Run File Shift+F6
Debug File Ctrl+Shift+F5
Test File Ctrl+F6
Debug Test File Ctrl+Shift+F6
Run Focused Test Method
Debug Focused Test Method
Run Into Method
New Watch... Ctrl+Shift+F7
Toggle Line Breakpoint Ctrl+F8
Profile
Cut Ctrl+X
Copy Ctrl+C
Rename... Ctrl+R
Move... Ctrl+M
Copy... Alt+C
Safely Delete... Alt+Delete
Inline... Alt+Shift+N
Change Method Parameters... Ctrl+Alt+Shift+C
Pull Up... Ctrl+Alt+Shift+U
Push Down... Ctrl+Alt+Shift+D
Extract Interface... Ctrl+Alt+Shift+T
Extract Superclass... Ctrl+Alt+Shift+S
Use Supertype Where Possible... Ctrl+Alt+Shift+W
Introduce
Move Inner to Outer Level... Ctrl+Alt+Shift+L
Convert Anonymous to Member... Ctrl+Alt+Shift+A
Encapsulate Fields... Ctrl+Alt+Shift+E

From here putting the attributes together and the lines of code to make the lists I refactor them. Do this to create the methods by right clicking here going to Refactor and choosing Encapsulate Fields.



Encapsulate Fields

List of Fields to Encapsulate:

Field	...	Create Getter	...	Create Setter
knowledgeType : String	☑	getKnowledgeType()...	☑	setKnowledgeType(S...
numberOfProjects : int	☑	getNumberOfProject...	☑	setNumberOfProject...
numberOfKnowledge...	☑	getNumberOfKnowle...	☑	setNumberOfKnowle...
projects : List<Insta...	☑	getProjects	☑	setProjects

Insert Point: Default

Sort By: Getter/Setter pairs

Javadoc: Create default comments

Fields' Visibility: private

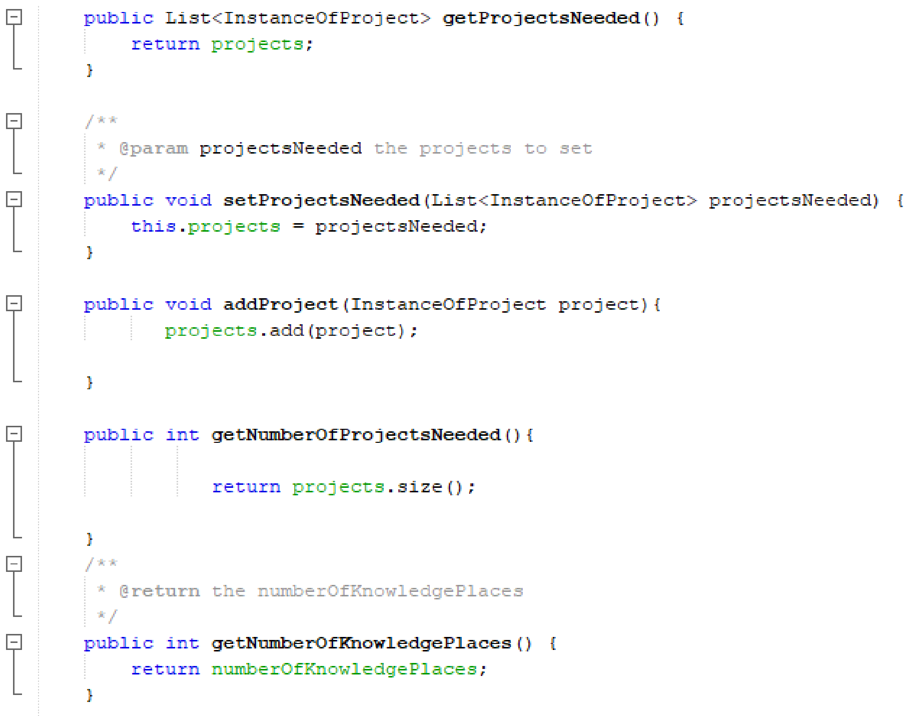
Accessors' Visibility: public

☑ Use Accessors Even When Field Is Accessible

☐ Generate Property Change Support

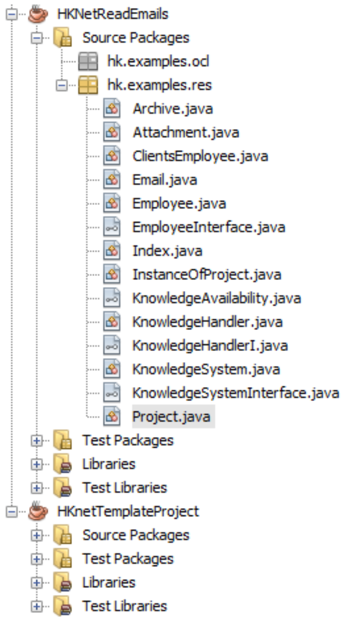
☐ Generate Vetoable Change Support

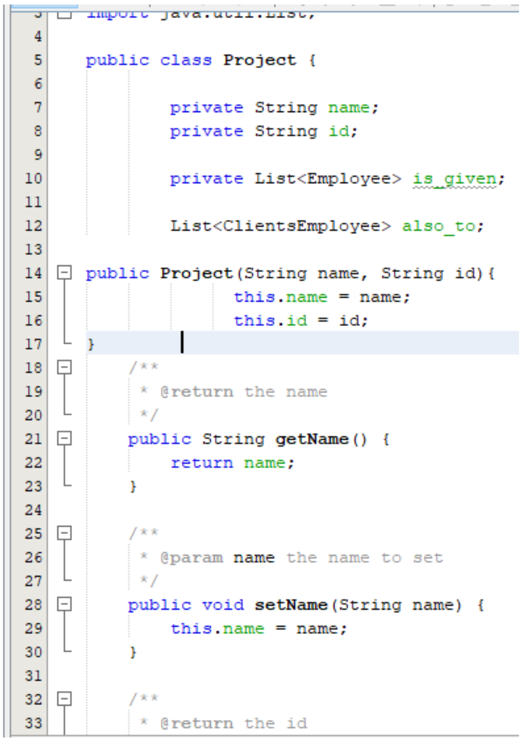
Encapsulating the fields is creating the methods that begin with get and set i.e. set and get methods. I tick the boxes for the methods to be able to bring the data that is needed from the KnowledgeSystem – the set methods. Do the same for the get methods to bring the data including those that bring data as lists.



```
public List<InstanceOfProject> getProjectsNeeded() {
    return projects;
}

/**
 * @param projectsNeeded the projects to set
 */
public void setProjectsNeeded(List<InstanceOfProject> projectsNeeded) {
    this.projects = projectsNeeded;
}

public void addProject(InstanceOfProject project){
        projects.add(project);

}

public int getNumberOfProjectsNeeded(){

            return projects.size();

}
/**
 * @return the numberOfKnowledgePlaces
 */
public int getNumberOfKnowledgePlaces() {
    return numberOfKnowledgePlaces;
}
```

Doing that creates the set and get methods automatically. So, there are now the methods to bring the number of projects available in the system. And then there is the list of Projects that are needed to be done available in the system. It is worth having the method to tell us the number of projects that are needed to be done. If there a project to that is needed that is not in the list, we can add it to the list by the add method.

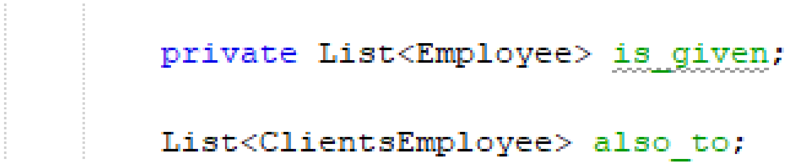




```
import java.util.List;

public class Project {

        private String name;
        private String id;

        private List<Employee> is_given;

        List<ClientsEmployee> also_to;

public Project(String name, String id){
            this.name = name;
            this.id = id;
}
    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the id
```

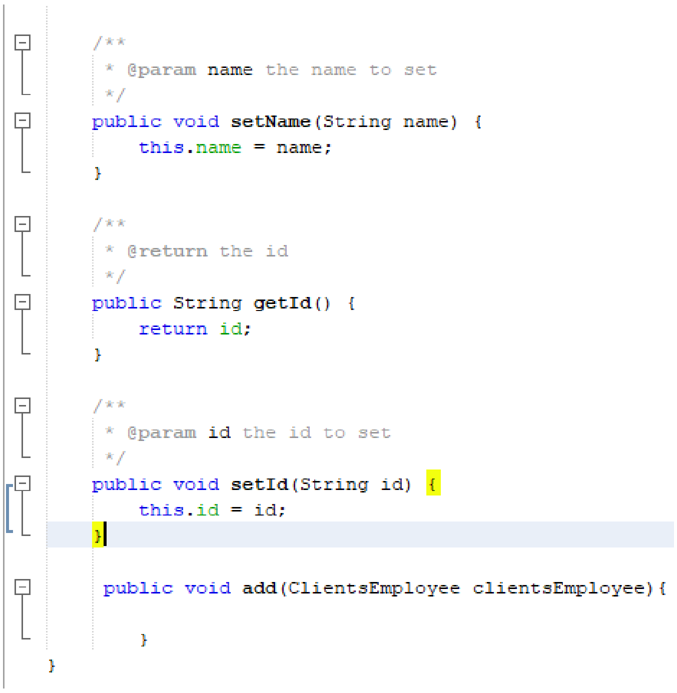
Project

For Project I have given its name and its id both as String and the line of code to make a list of the employees and their clients employees who are going to be given the project to do. After putting the attributes together I automatically put the set and get methods here by refactoring the fields.



```
private List<Employee> is_given;

List<ClientsEmployee> also_to;
```

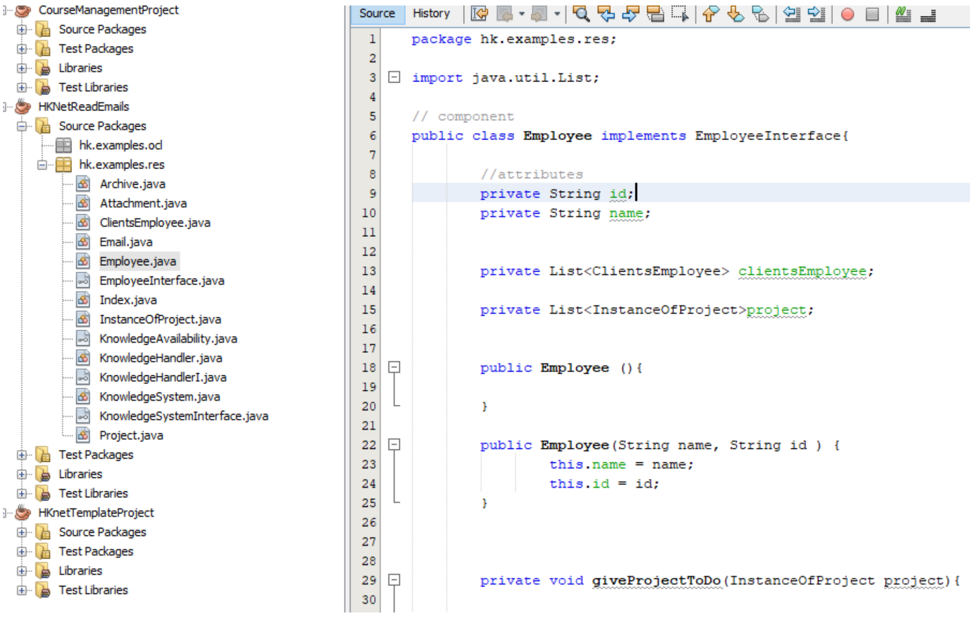
Here I am following the role names I have put in my class diagram. Between Project and Employee and Clients Employee the role names tell us if there is a reference and which class they are of type. So we are creating the lists of Employees and ClientsEmployees by the role names is_given and also_to which are in the diagram.



```
    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the id
     */
    public String getId() {
        return id;
    }

    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.id = id;
    }

     public void add(ClientsEmployee clientsEmployee){

        }
}
```

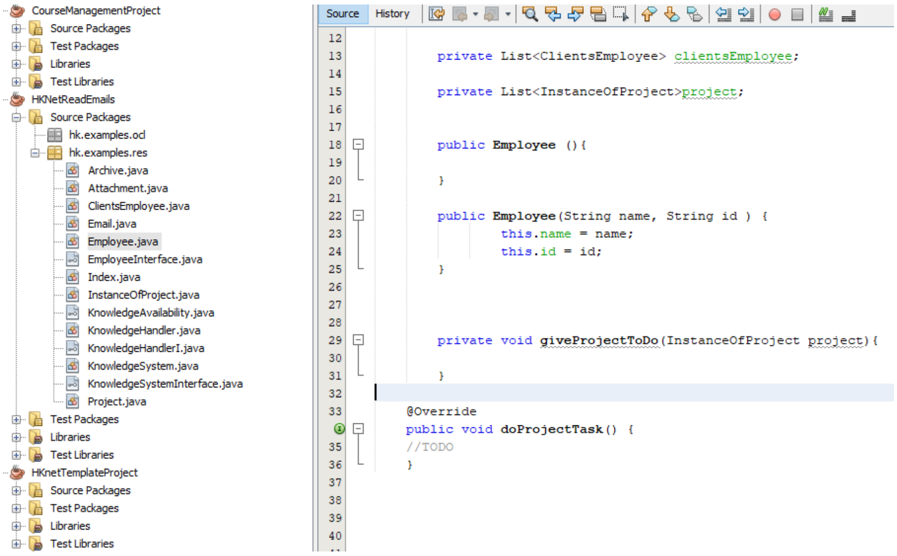
The add method I have given to add the list of Clients employees who are also given the project to do. This is put in uppercase letters next to it in lowercase letters as parameters to add the list to the project.

Employee



```
package hk.examples.res;

import java.util.List;

// component
public class Employee implements EmployeeInterface{

        //attributes
        private String id;
        private String name;


        private List<ClientsEmployee> clientsEmployee;

        private List<InstanceOfProject>project;


        public Employee (){

        }

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


        private void giveProjectToDo(InstanceOfProject project){

```

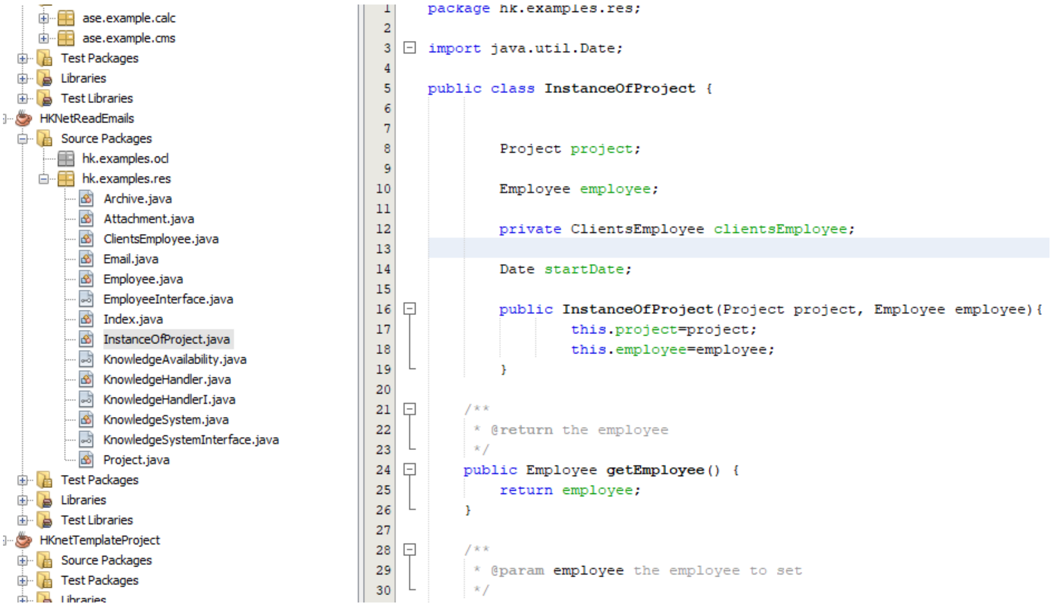
Employee starts with implementing its interface which I will come to in the next stage. It has its necessary attributes id and name of the employee being available as private. They are both string like the rest of the classes. It is worth creating lists here of the project since they will be assigned to a project. Their clients' employees are assigned to a project too, so it is worth having a list of them. The employee can then know their clients' employees have been given the same task to do.



```

        private List<ClientsEmployee> clientsEmployee;

        private List<InstanceOfProject>project;


        public Employee (){

        }

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


        private void giveProjectToDo(InstanceOfProject project){

        }

    @Override
    public void doProjectTask() {
    //TODO
    }
```

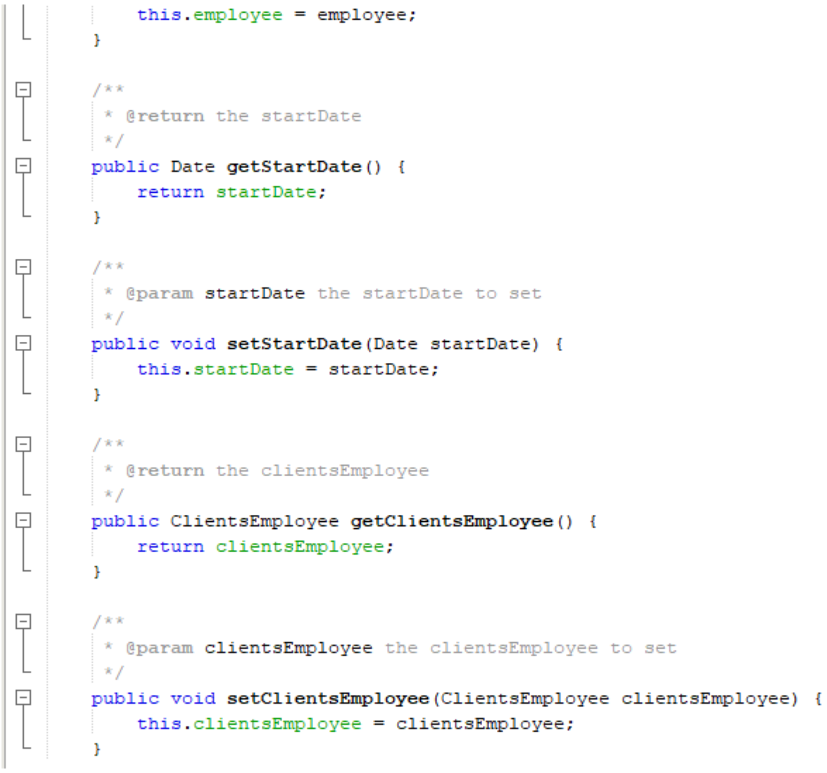
We have the method here which is to be seen in private. It has the parameters that refer to the project which is in the list of projects to be given to the employee to do. The list belonging to the InstanceOfProject class which is the parameter. The method that is public is the doProjectTask() as that is carried out as the Employee interface method.

InstanceOfProject



```java
package hk.examples.res;

import java.util.Date;

public class InstanceOfProject {


        Project project;

        Employee employee;

        private ClientsEmployee clientsEmployee;

        Date startDate;

        public InstanceOfProject(Project project, Employee employee){
                this.project=project;
                this.employee=employee;
        }

    /**
     * @return the employee
     */
    public Employee getEmployee() {
        return employee;
    }

    /**
     * @param employee the employee to set
     */
```

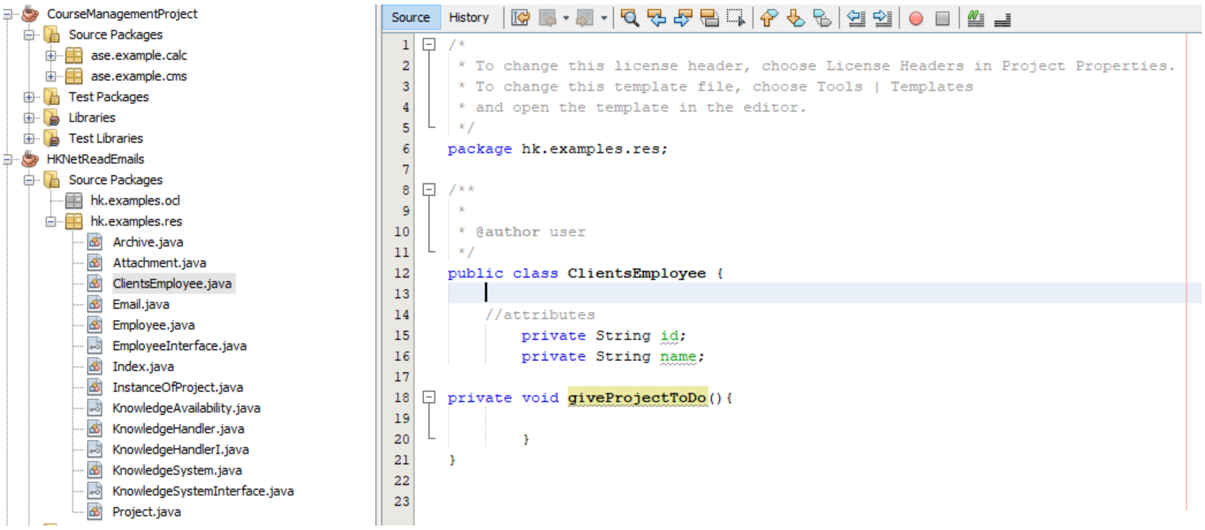
The InstanceOfProject is part of the project. The lines of code refer to that wen creating lists that are to do with the project. The classes that are needed for this instance of the project are put here, with their fields, including the Project itself. Having put them altogether I refactor them with their fields to list all the get and set methods.



```java
        this.employee = employee;
    }

    /**
     * @return the startDate
     */
    public Date getStartDate() {
        return startDate;
    }

    /**
     * @param startDate the startDate to set
     */
    public void setStartDate(Date startDate) {
        this.startDate = startDate;
    }

    /**
     * @return the clientsEmployee
     */
    public ClientsEmployee getClientsEmployee() {
        return clientsEmployee;
    }

    /**
     * @param clientsEmployee the clientsEmployee to set
     */
    public void setClientsEmployee(ClientsEmployee clientsEmployee) {
        this.clientsEmployee = clientsEmployee;
    }
```

The date the project starts is necessary. I have encapsulated its fields to list the method to bring the starting date i.e. the get method and the method that makes it available to be brought i.e. the set method.

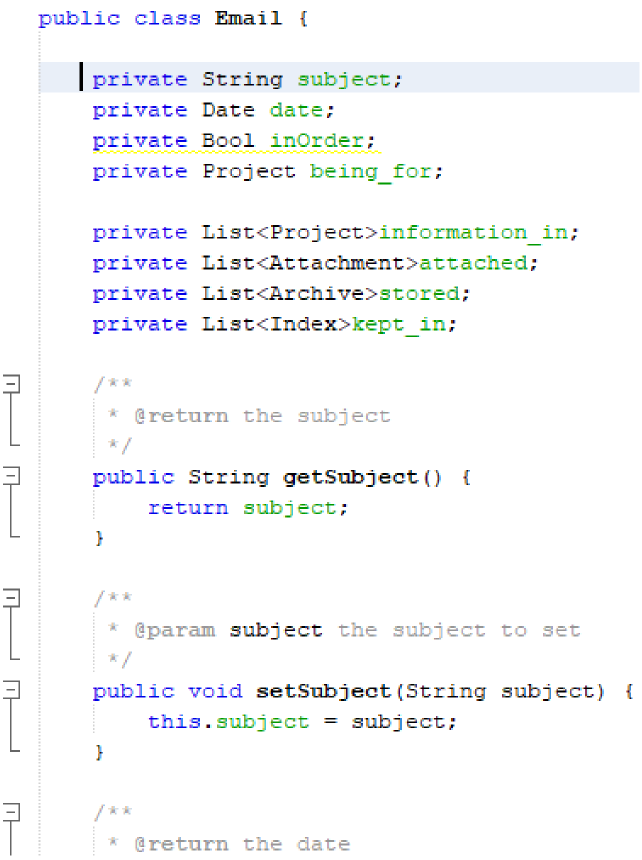
Clients Employee



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.res;

/**
 *
 * @author user
 */
public class ClientsEmployee {

    //attributes
        private String id;
        private String name;

private void giveProjectToDo(){

        }
}
```

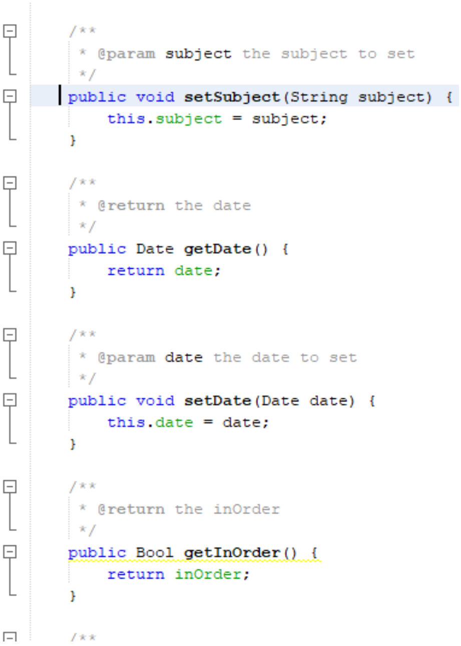
For the ClientsEmployee I have put the same parameters their name and their id as String. They are also given the project to do as their method.

Email



```
public class Email {

    private String subject;
    private Date date;
    private Bool inOrder;
    private Project being_for;

    private List<Project>information_in;
    private List<Attachment>attached;
    private List<Archive>stored;
    private List<Index>kept_in;

    /**
     * @return the subject
     */
    public String getSubject() {
        return subject;
    }

    /**
     * @param subject the subject to set
     */
    public void setSubject(String subject) {
        this.subject = subject;
    }

    /**
     * @return the date
```

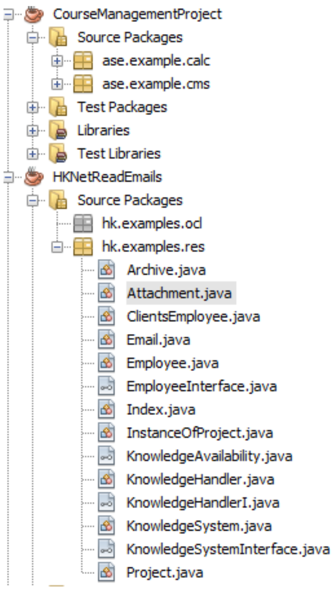
For the Email I have put several attributes and lists which are kept private. These attributes and lists are so far needed by the workers including Project, Archive and Index.



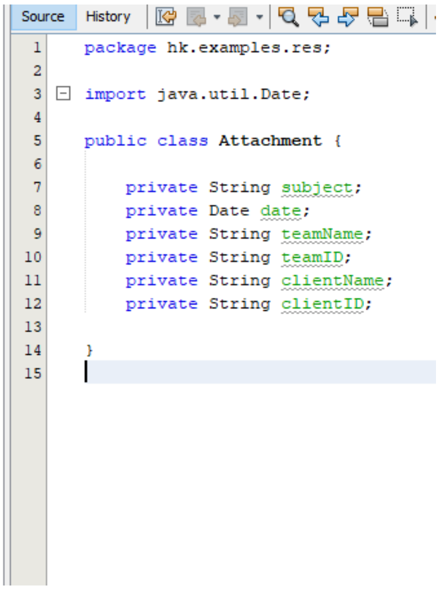
```
    /**
     * @param subject the subject to set
     */
    public void setSubject(String subject) {
        this.subject = subject;
    }

    /**
     * @return the date
     */
    public Date getDate() {
        return date;
    }

    /**
     * @param date the date to set
     */
    public void setDate(Date date) {
        this.date = date;
    }

    /**
     * @return the inOrder
     */
    public Bool getInOrder() {
        return inOrder;
    }

    /**
```

There are several attributes and lists to put together. I have just refactored their fields to automatically put the set and get methods of them. All these methods ae necessary for the workers to look for and read the emails of past projects to help them do the one they are doing at the moment. Also to check the date of the email and whether or not they are in order.

<u>Attachment</u>

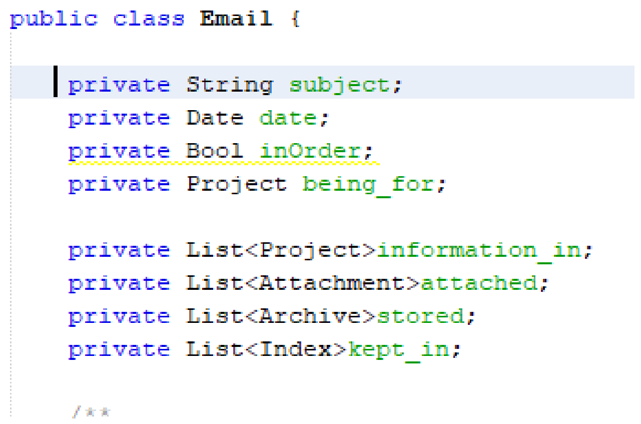


```
CourseManagementProject
    Source Packages
        ase.example.calc
        ase.example.cms
    Test Packages
    Libraries
    Test Libraries
HKNetReadEmails
    Source Packages
        hk.examples.ocl
        hk.examples.res
            Archive.java
            Attachment.java
            ClientsEmployee.java
            Email.java
            Employee.java
            EmployeeInterface.java
            Index.java
            InstanceOfProject.java
            KnowledgeAvailability.java
            KnowledgeHandler.java
            KnowledgeHandlerI.java
            KnowledgeSystem.java
            KnowledgeSystemInterface.java
            Project.java
```



```
package hk.examples.res;

import java.util.Date;

public class Attachment {

    private String subject;
    private Date date;
    private String teamName;
    private String teamID;
    private String clientName;
    private String clientID;

}

```

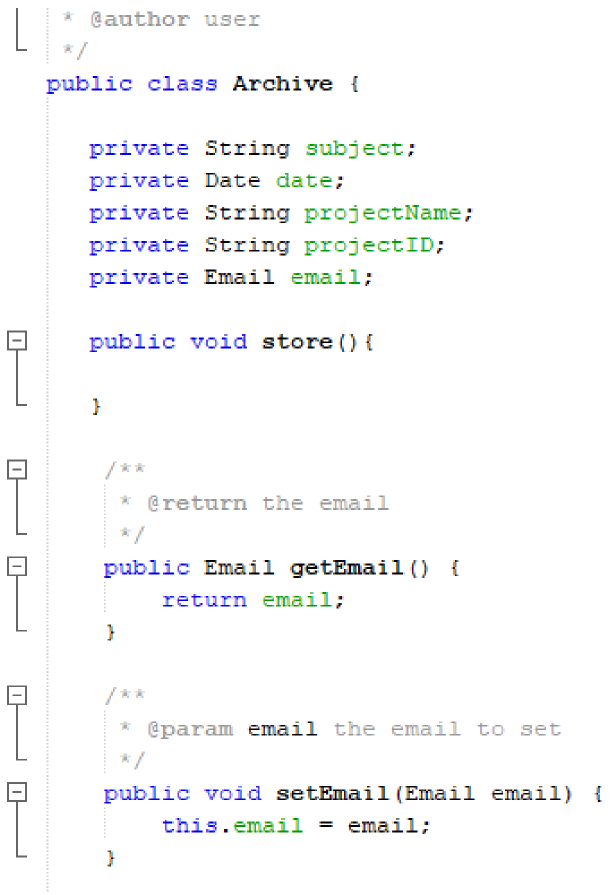
Attachment is an aggregation of Email. This means it is a class in the class Email. This means I wrote the code for it as shown above. Just the name with it's attributes that are needed such as the date and the names and IDs of the project teams and clients. The attachments are stored by client and by project team.



```
public class Email {

    private String subject;
    private Date date;
    private Bool inOrder;
    private Project being_for;

    private List<Project>information_in;
    private List<Attachment>attached;
    private List<Archive>stored;
    private List<Index>kept_in;

    /**
```

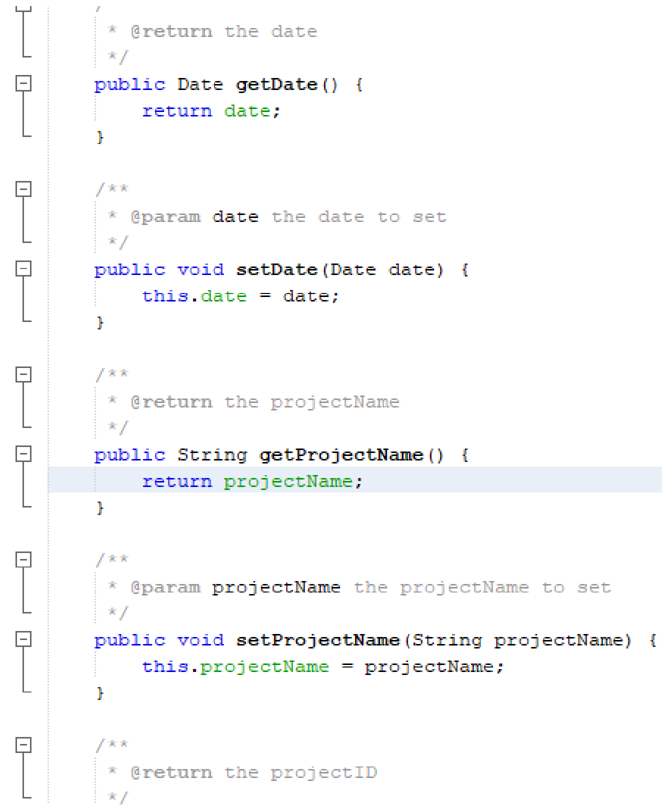
In Email I have put the line of code to create just a list of the attachments. Listed under the role name attached these attributes are needed to be known for each attachment in the list.

Archive



```
 * @author user
 */
public class Archive {

    private String subject;
    private Date date;
    private String projectName;
    private String projectID;
    private Email email;

    public void store(){

    }

     /**
      * @return the email
      */
     public Email getEmail() {
         return email;
     }

     /**
      * @param email the email to set
      */
     public void setEmail(Email email) {
         this.email = email;
     }
```

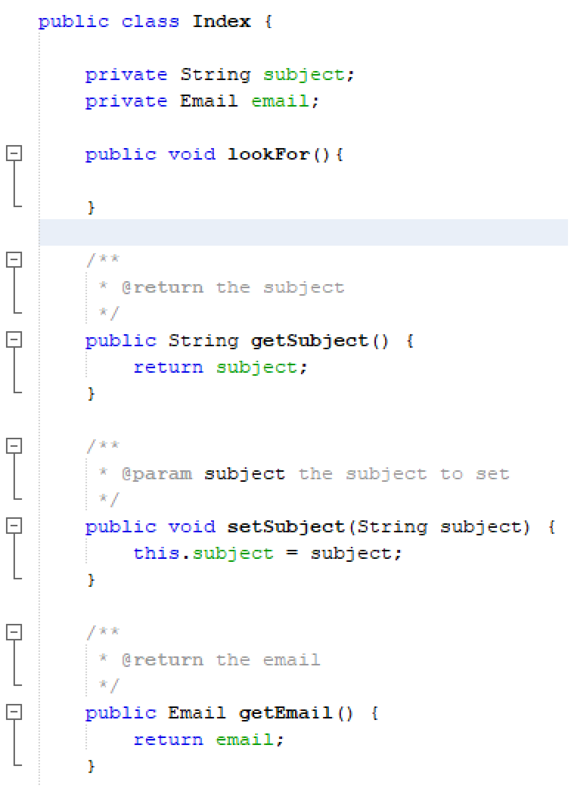
The Archive is associated with Email. This means putting its attributes that are kept private. The attributes are needed by the workers to read emails, in the archive, of projects done in the past. They need these types of information to do that. The method store() is available in the public which makes emails be stored in the archive. The Email itself is a private attribute because of the association.



```
    /
     * @return the date
     */
    public Date getDate() {
        return date;
    }

    /**
     * @param date the date to set
     */
    public void setDate(Date date) {
        this.date = date;
    }

    /**
     * @return the projectName
     */
    public String getProjectName() {
        return projectName;
    }

    /**
     * @param projectName the projectName to set
     */
    public void setProjectName(String projectName) {
        this.projectName = projectName;
    }

    /**
     * @return the projectID
     */
```

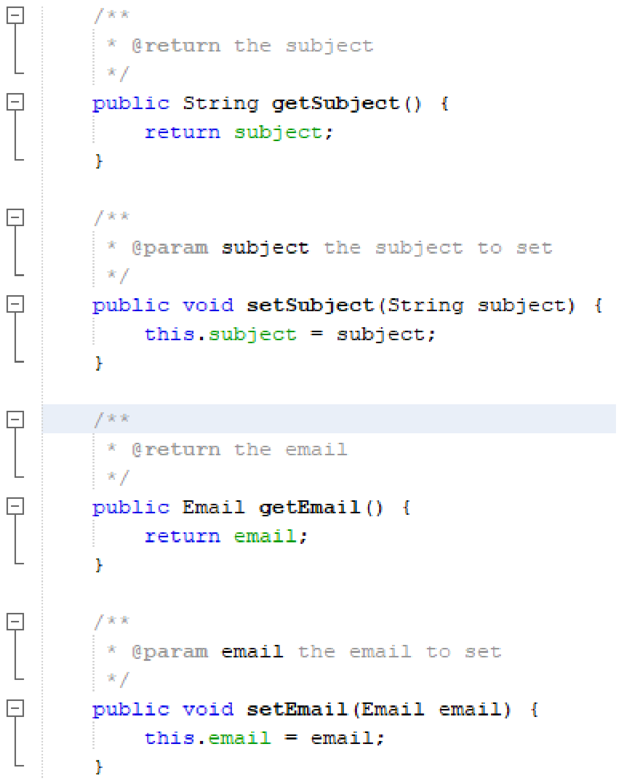
I have also refactored the attribute fields to automatically put the set and get methods. The methods to make the date available to be brought to be read plus those that make the project name and ID available to be brought to be checked.

Index



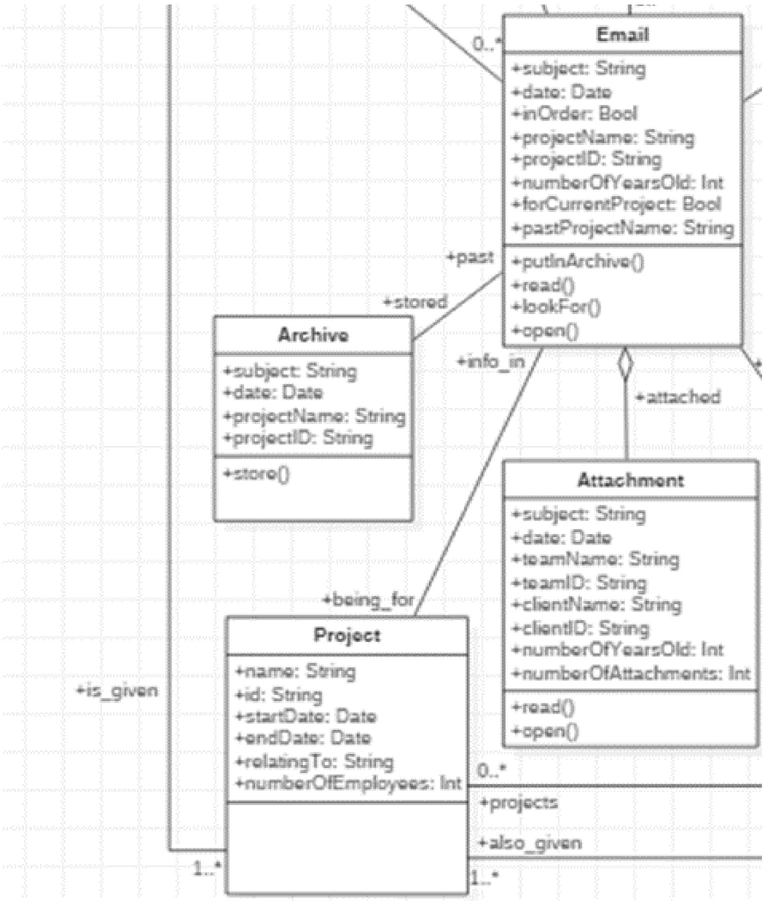
```
public class Index {

    private String subject;
    private Email email;

    public void lookFor(){

    }

    /**
     * @return the subject
     */
    public String getSubject() {
        return subject;
    }

    /**
     * @param subject the subject to set
     */
    public void setSubject(String subject) {
        this.subject = subject;
    }

    /**
     * @return the email
     */
    public Email getEmail() {
        return email;
    }
```

The same for Index. It has its attributes kept as private including Email because of the association. The method to make the subject of the emails be looked for, to find the email, is put next.

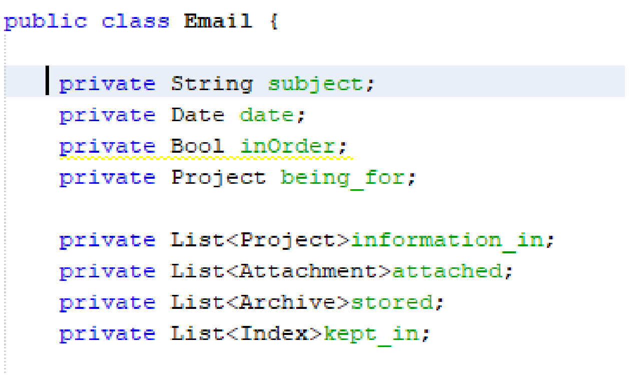


```
/**
 * @return the subject
 */
public String getSubject() {
    return subject;
}

/**
 * @param subject the subject to set
 */
public void setSubject(String subject) {
    this.subject = subject;
}

/**
 * @return the email
 */
public Email getEmail() {
    return email;
}

/**
 * @param email the email to set
 */
public void setEmail(Email email) {
    this.email = email;
}
```

The refactoring of the fields of the subject and email are done next. The set and get methods are automatically created to find the email in the index by looking at and clicking on its subject. The emails to be looked for are made available and then brought to open and read by the two methods.. Since each email is put in the index by subject the subject is made available and then brought to look up and click on by the two methods.



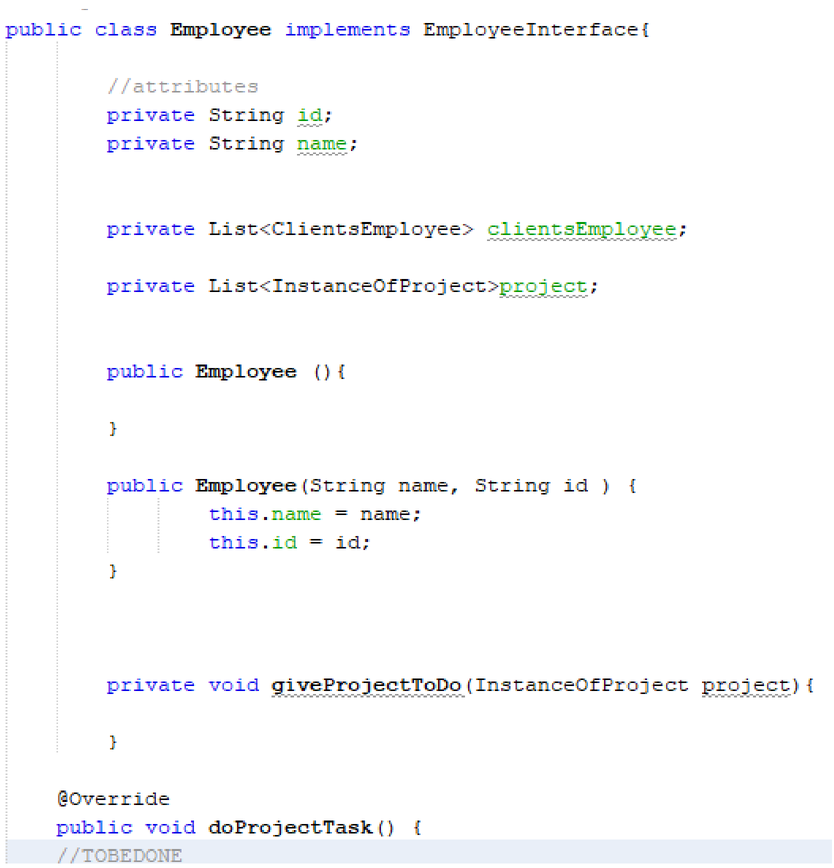
Again I follow what I have put in my class diagram. Between the Email and the Project we have the role name being_for which is of type Project - the Email being for the Project. The role name info_in is the foreign key the Project references the Email to. Therefore these role names are the end points. The end points that are worth having because I follow them to make this use case work in the form of this programming code.



```
public class Email {

    private String subject;
    private Date date;
    private Bool inOrder;
    private Project being_for;

    private List<Project>information_in;
    private List<Attachment>attached;
    private List<Archive>stored;
    private List<Index>kept_in;
```

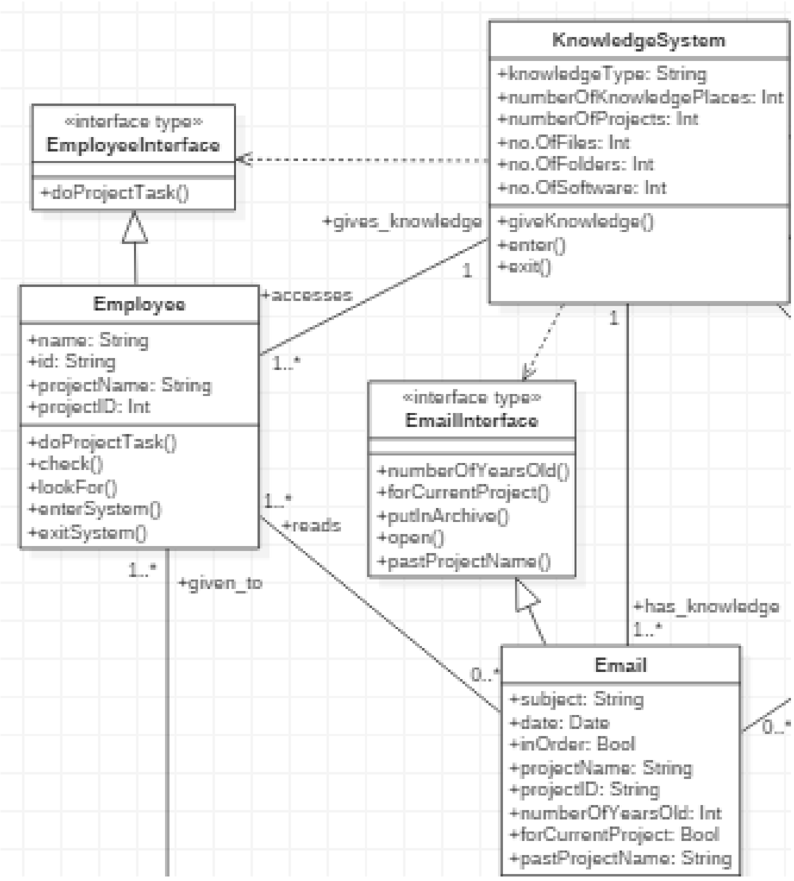
The Project in Email I have put at the beginning is the project the emails are being for. I then create the list of projects that have their information in the email. The lines of code to I have put to make this happen show I am following what my class diagram is saying.

Components Being Connected With Their Interfaces

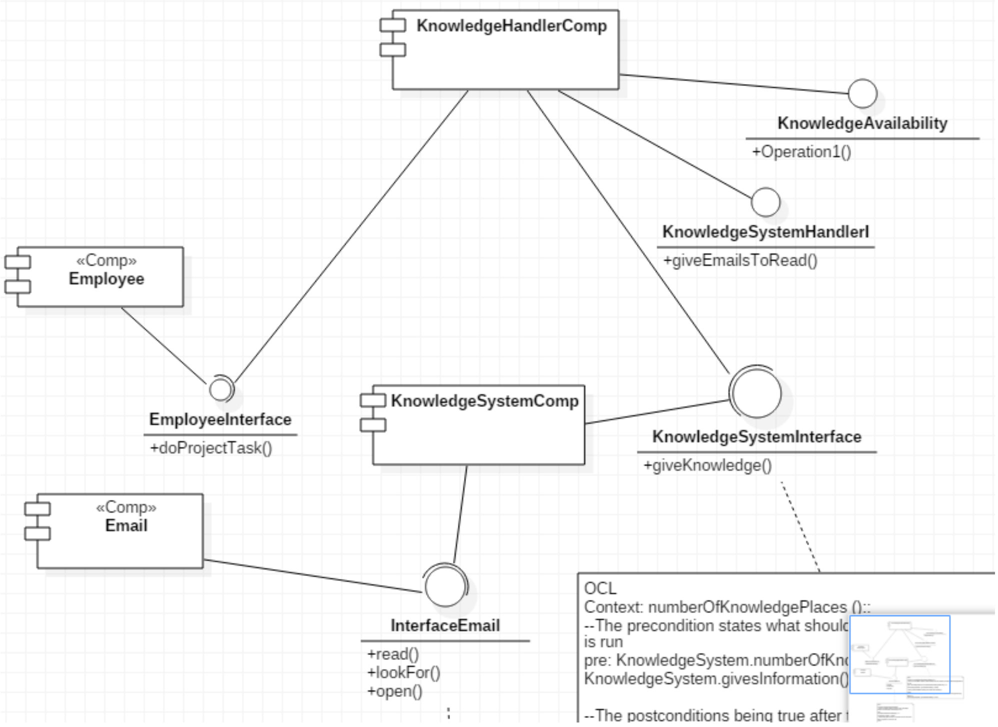


```
public class Employee implements EmployeeInterface{

        //attributes
        private String id;
        private String name;


        private List<ClientsEmployee> clientsEmployee;

        private List<InstanceOfProject>project;


        public Employee (){

        }

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



        private void giveProjectToDo(InstanceOfProject project){

        }

    @Override
    public void doProjectTask() {
    //TOBEDONE
```

The interface is what the user sees and does with such as look at the text box on the screen and type text and then click on the enter button. To have that with the Employee class I need to turn into a component. A component which has its interface for the user to see and do tasks with. Basically, this class puts it interface into doing the task. This is it implementing the interface.



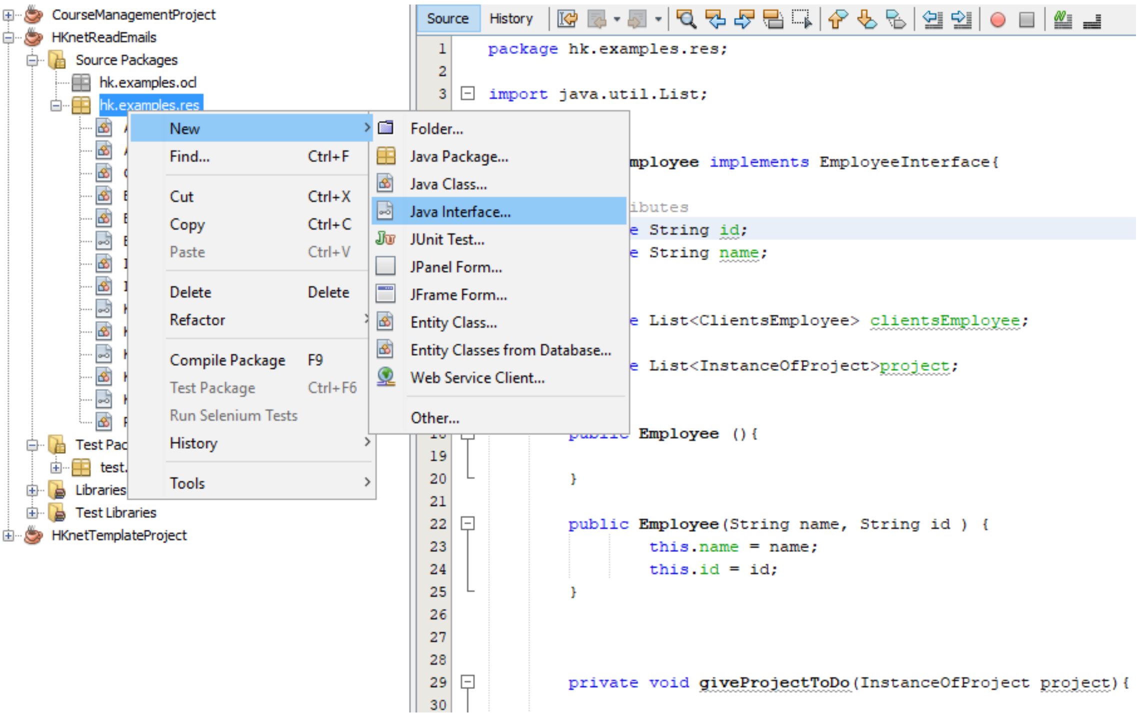
In the class diagram I have the Employee class implement its interface. This is by expanding it with its interface type. Its interface type has to have a method. In this case it has the method doProjectTask.



OCL
Context: numberOfKnowledgePlaces ()::
--The precondition states what should
is run
pre: KnowledgeSystem.numberOfKno
KnowledgeSystem.givesInformation()

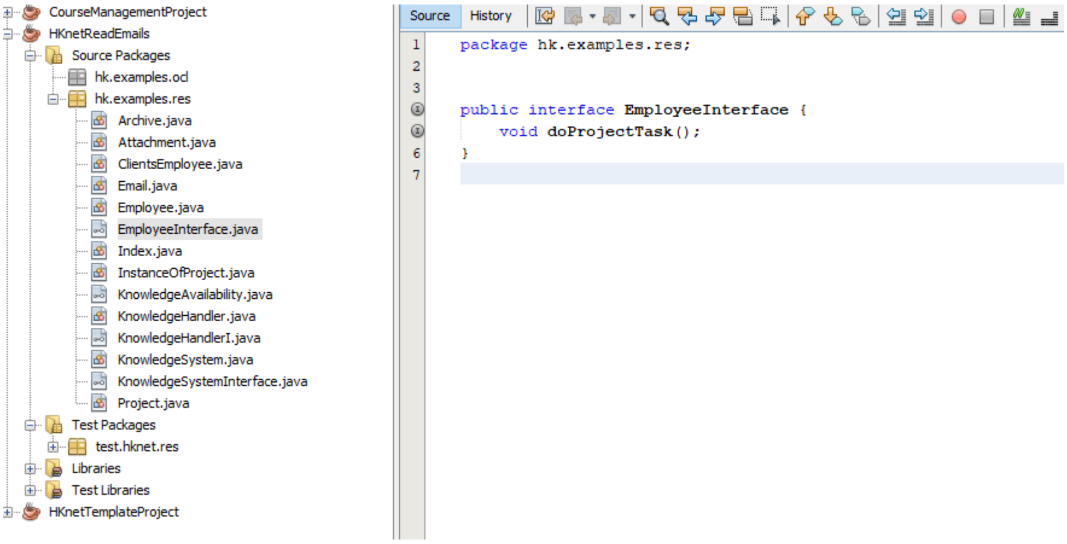
--The postconditions being true after t

The component diagram shows the Employee as a component with its interface that it puts into doing the task. Basically, it makes the project task be done by the employee. The employee looking at the computer screen and typing up the document to do the task. That is put as the method in the class diagram and here.



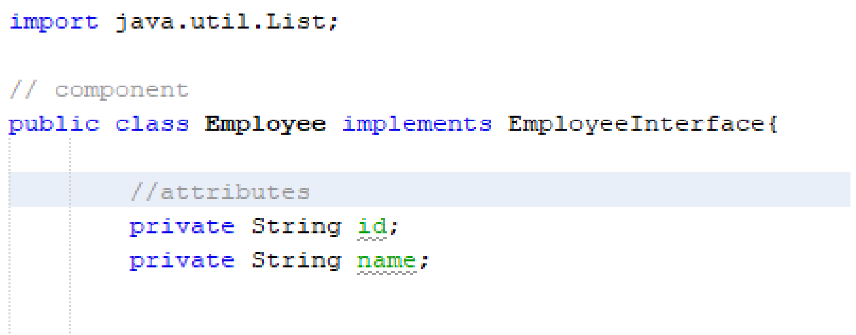
```
package hk.examples.res;

import java.util.List;

mployee implements EmployeeInterface{

ibutes
e String id;
e String name;

e List<ClientsEmployee> clientsEmployee;

e List<InstanceOfProject>project;

public Employee (){

}

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



private void giveProjectToDo(InstanceOfProject project){
```

To do the coding of the Employee interface I right click on my project source folder and select new Java Interface. Give the interface a class name as it is actually just a class I am creating. Make sure it is in the right folder and location and click finish.



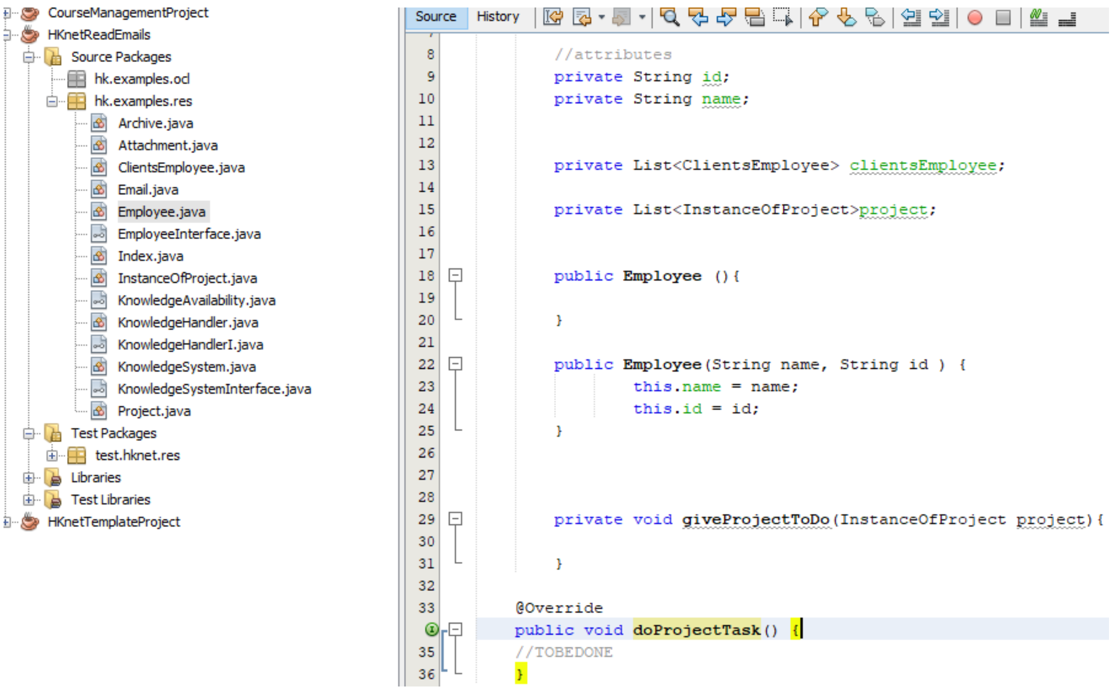
```
package hk.examples.res;


public interface EmployeeInterface {
    void doProjectTask();
}
```

I just make sure it states public interface as that is what it is instead of class. The name of the interface follows that with the method below it. Here the method is just a void method because it is being made to run by the same method in the class the interface is for i.e. implemented. This being the Employee class.



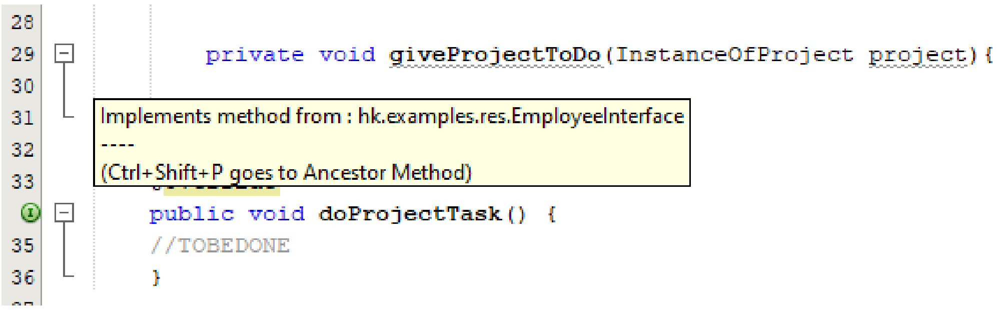
```
import java.util.List;

// component
public class Employee implements EmployeeInterface{

        //attributes
        private String id;
        private String name;
```

The Employee class putting the method into running in its interface means it now becomes a component in its class. I comment it as that state that it implements its interface.



```
        //attributes
        private String id;
        private String name;


        private List<ClientsEmployee> clientsEmployee;

        private List<InstanceOfProject>project;


        public Employee (){

        }

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



        private void giveProjectToDo(InstanceOfProject project){

        }

    @Override
    public void doProjectTask() {
    //TOBEDONE
    }
```

The doProjectTask method goes at the bottom as it is the one that puts the method in the interface class into running. This means it overrides that method being the same as that. It is annotated by @Override to indicate it implements the method.

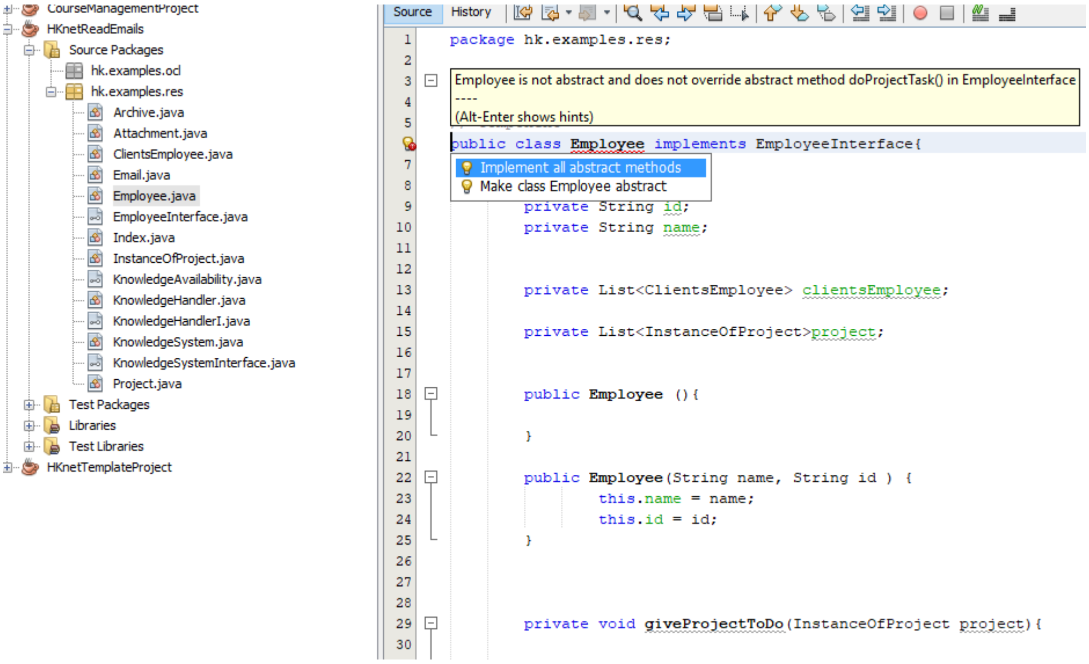


```
        private void giveProjectToDo(InstanceOfProject project){

    public void doProjectTask() {
    //TOBEDONE
    }
```

Implements method from : hk.examples.res.EmployeeInterface

(Ctrl+Shift+P goes to Ancestor Method)

By hovering the mouse pointer over the number of this line of code you see the message telling you the method in the interface class has been put into running. Clicking on the number shown as the green letter i takes you to the interface class to the method being implemented. The Override annotation makes this happen.

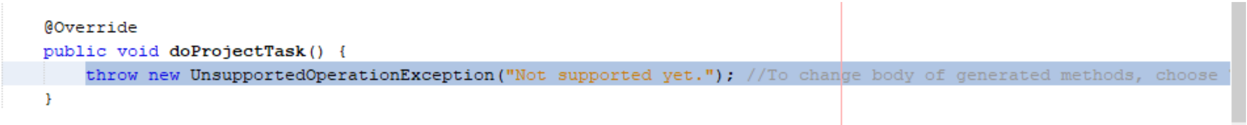


```
package hk.examples.res;
```

Employee is not abstract and does not override abstract method doProjectTask() in EmployeeInterface
----
(Alt-Enter shows hints)

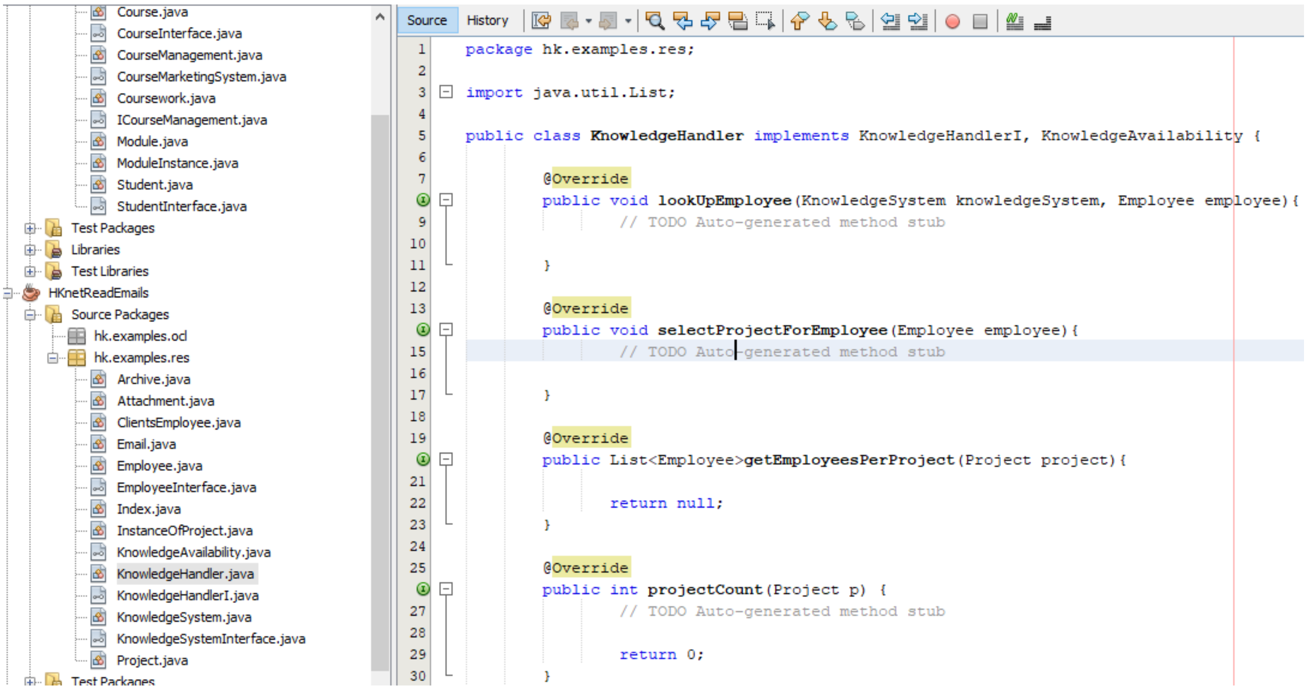
```
public class Employee implements EmployeeInterface{

        private String id;
        private String name;


        private List<ClientsEmployee> clientsEmployee;

        private List<InstanceOfProject>project;


        public Employee (){

        }

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



        private void giveProjectToDo(InstanceOfProject project){
```

Before putting the doProjectTask() method to override the interface method I must hover the mouse pointer over the number of the line of code “Public class Employee” and choose to ‘Implement all abstract methods’. This puts the method with the annotation indicating it is now overriding the one in the interface class.



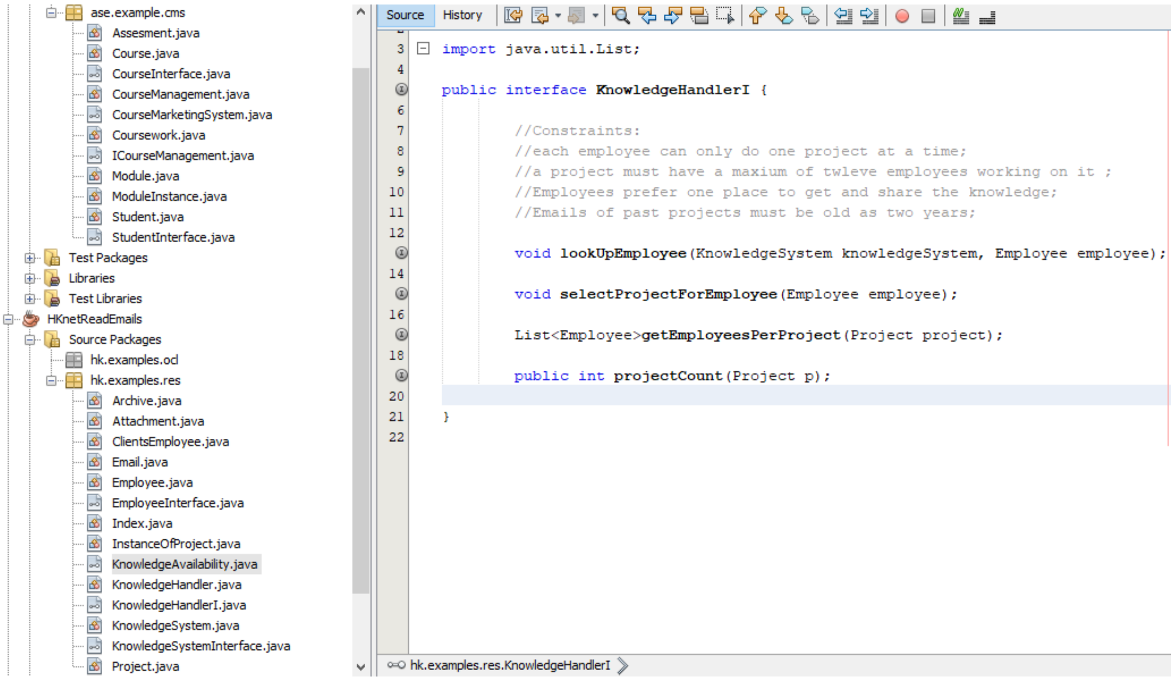
```
    @Override
    public void doProjectTask() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose
    }
```

Now this message I’ve highlighted is a call back message that appears by me choosing ‘Implement all abstract methods’. I decide to delete this as it is not needed to be kept here.



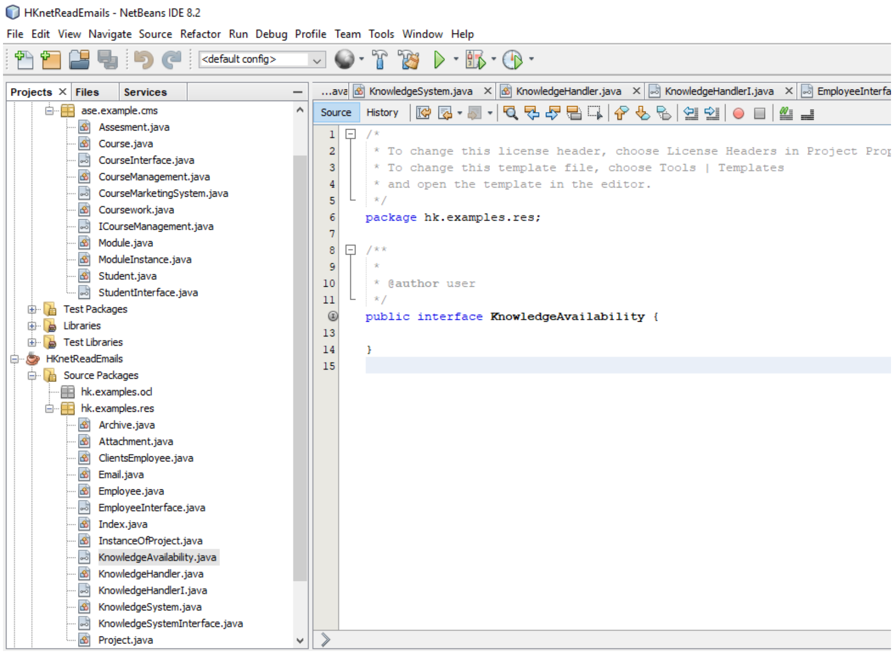
```java
package hk.examples.res;

import java.util.List;

public class KnowledgeHandler implements KnowledgeHandlerI, KnowledgeAvailability {

        @Override
        public void lookUpEmployee(KnowledgeSystem knowledgeSystem, Employee employee){
                // TODO Auto-generated method stub

        }

        @Override
        public void selectProjectForEmployee(Employee employee){
                // TODO Auto-generated method stub


        }

        @Override
        public List<Employee>getEmployeesPerProject(Project project){

                return null;
        }

        @Override
        public int projectCount(Project p) {
                // TODO Auto-generated method stub

                return 0;
        }
```

The next class becoming a component is called the KnowledgeHandler. This is connected to the KnowledgeSystem. It has methods together which are in public. These methods are the snippets of code that put the methods in KnowledgeHandlerI and KnowledgeAvailability into running. I have put one to create a list of employees for one project as in employees to be given one project to do.



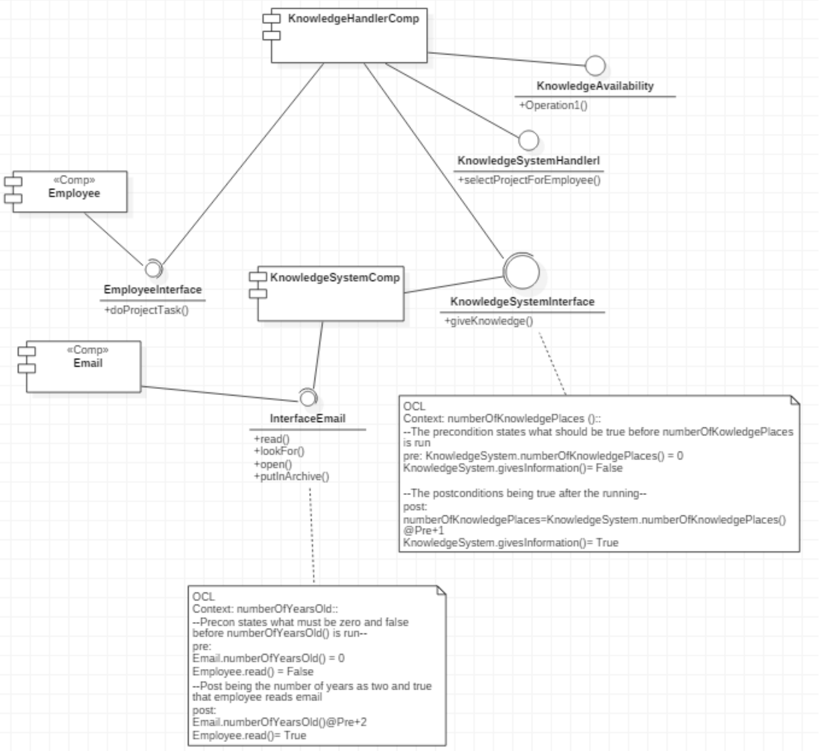
```java
import java.util.List;

public interface KnowledgeHandlerI {

        //Constraints:
        //each employee can only do one project at a time;
        //a project must have a maxium of twleve employees working on it ;
        //Employees prefer one place to get and share the knowledge;
        //Emails of past projects must be old as two years;

        void lookUpEmployee(KnowledgeSystem knowledgeSystem, Employee employee);

        void selectProjectForEmployee(Employee employee);

        List<Employee>getEmployeesPerProject(Project project);

        public int projectCount(Project p);

}
```

So the interface starts with it being public. It just has the methods altogether. Each method such as lookUpEmployee has the names of the classes, to refer to to be run, as their parameters. The class names being in lower case followed by the one in uppercase. These are basically constructors to create an instance of a class. These parameters are the method signatures. So the interface is the details of the employee being opened by clicking on a button on the screen to see which project they should be given to do. Done by entering the KnowledgeSystem to find a list of the details on the screen and look at them.



```
/*
 * To change this license header, choose License Headers in Project Proj
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.res;

/**
 *
 * @author user
 */
public interface KnowledgeAvailability {

}
```

For KnowledgeAvailability I could put the same methods here. Those that are to get data of projects and employees, that are available, from the KnowledgeSystem.

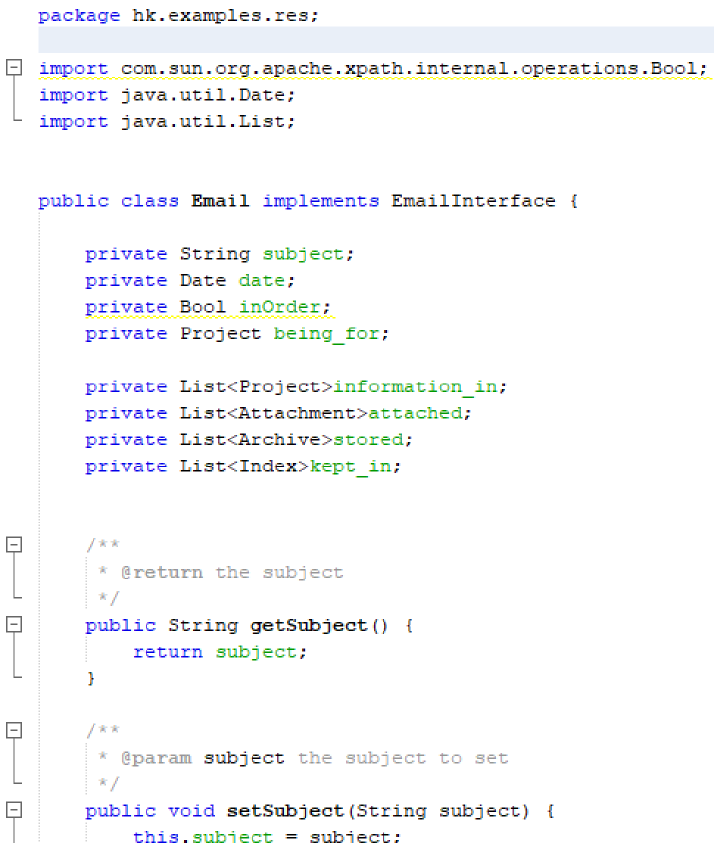


OCL
Context: numberOfKnowledgePlaces ()::
--The precondition states what should be true before numberOfKowledgePlaces is run
pre: KnowledgeSystem.numberOfKnowledgePlaces() = 0
KnowledgeSystem.givesInformation()= False

--The postconditions being true after the running--
post:
numberOfKnowledgePlaces=KnowledgeSystem.numberOfKnowledgePlaces()
@Pre+1
KnowledgeSystem.givesInformation()= True

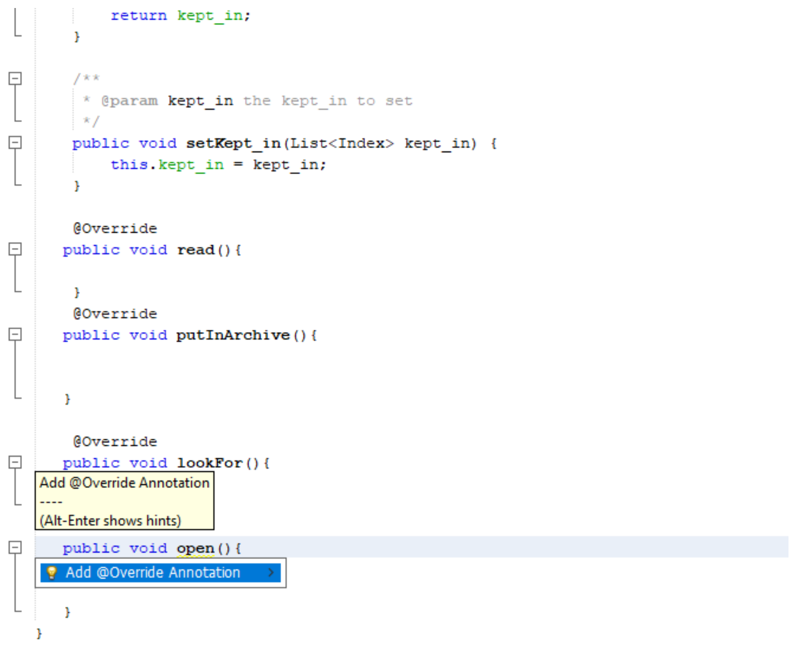
OCL
Context: numberOfYearsOld::
--Precon states what must be zero and false before numberOfYearsOld() is run--
pre:
Email.numberOfYearsOld() = 0
Employee.read() = False
--Post being the number of years as two and true that employee reads email
post:
Email.numberOfYearsOld()@Pre+2
Employee.read()= True

The component diagram shows the KnowledgeHandler component is connected to the KnowledgeSystem comp through the KnowledgeSystems interface. It shows KnowledgeHandler put the two interfaces into running. KnowledgeSystemHandlerI running the method selectProjectForEmployee. This shows that a component can implement more than one interface. I don't need to include these components in the class diagram so long as I have included them in this diagram.



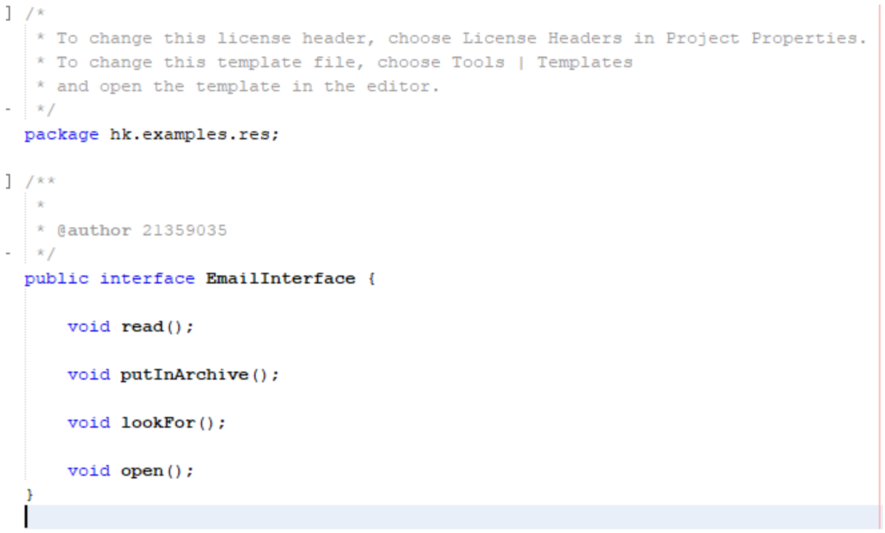
```
package hk.examples.res;

import com.sun.org.apache.xpath.internal.operations.Bool;
import java.util.Date;
import java.util.List;


public class Email implements EmailInterface {

    private String subject;
    private Date date;
    private Bool inOrder;
    private Project being_for;

    private List<Project>information_in;
    private List<Attachment>attached;
    private List<Archive>stored;
    private List<Index>kept_in;


    /**
     * @return the subject
     */
    public String getSubject() {
        return subject;
    }

    /**
     * @param subject the subject to set
     */
    public void setSubject(String subject) {
        this.subject = subject;
```

I follow the same steps for the Email interface. It is worth having the Email interface that the employee or clients employee looks through, clicks on the email message of a past project and reads on the screen.



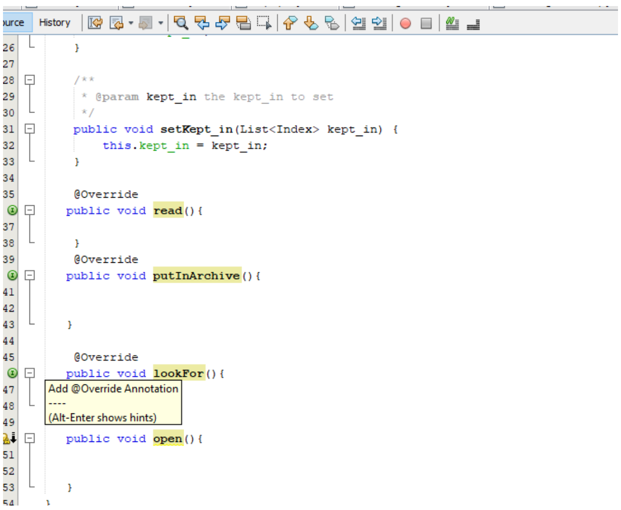
```
        return kept_in;
    }

    /**
     * @param kept_in the kept_in to set
     */
    public void setKept_in(List<Index> kept_in) {
        this.kept_in = kept_in;
    }

    @Override
   public void read(){

    }
    @Override
   public void putInArchive(){


   }

    @Override
   public void lookFor(){

   }

   public void open(){

   }
}
```

I put the methods to make the emails of past projects be read, looked for and clicked on to open on the screen. These are read(), lookFor, putInArchive() and open().



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.res;

/**
 *
 * @author 21359035
 */
public interface EmailInterface {

    void read();

    void putInArchive();

    void lookFor();

    void open();
}
```

I add them to the Email interface. I then check it has ‘Email implements EmailInterface’ correctly. Rather than choose ‘Implement all abstract methods’ I can just click on the line number of each method and choose to add the override notation.

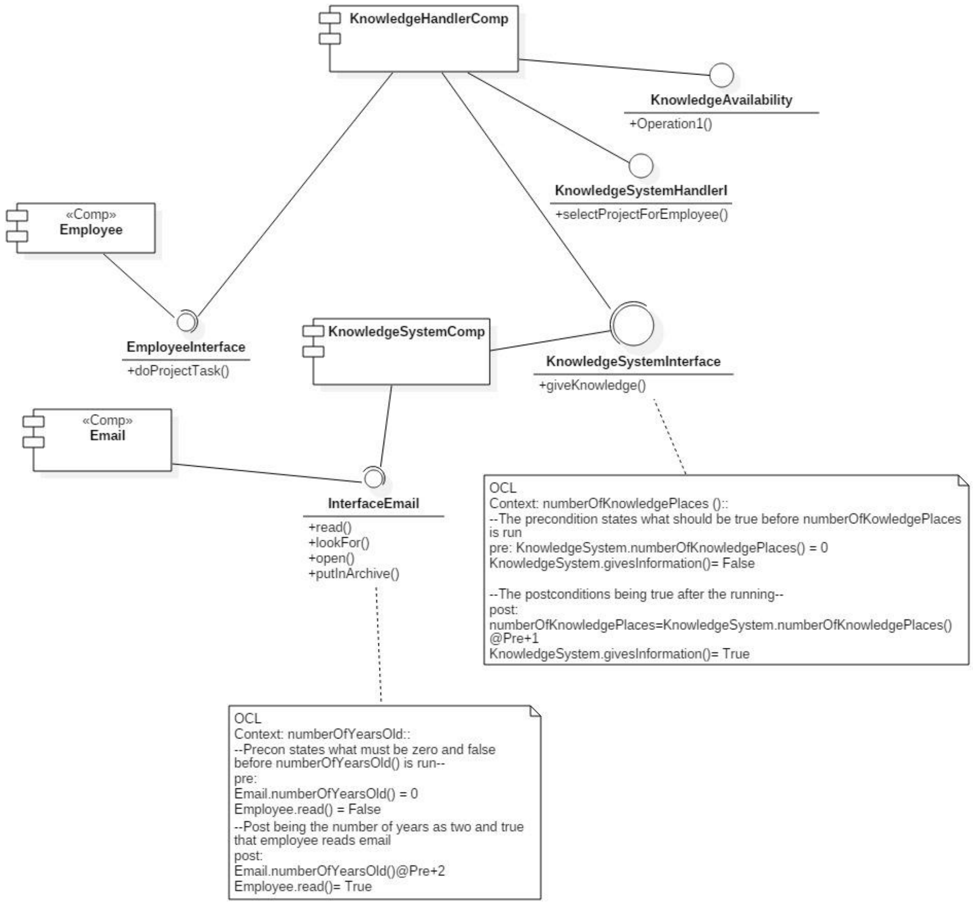


```
    }

    /**
     * @param kept_in the kept_in to set
     */
    public void setKept_in(List<Index> kept_in) {
        this.kept_in = kept_in;
    }

    @Override
   public void read(){

    }
    @Override
   public void putInArchive(){


   }

    @Override
   public void lookFor(){


   public void open(){


   }
}
```

Add @Override Annotation

(Alt-Enter shows hints)

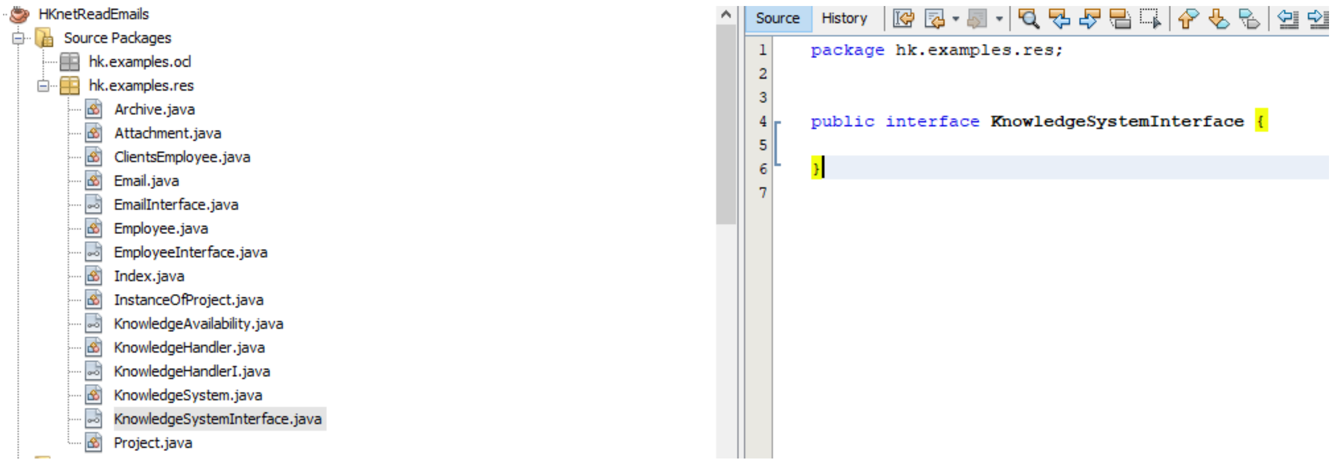
So I do that to add the override notation to each method. Each notation indicates they implement the same ones in the Email interface like the Employee and KnowledgeHandler classes do to theirs. The i on the number line of each method in the interface indicates they are being put into running.



```
OCL
Context: numberOfKnowledgePlaces ()::
--The precondition states what should be true before numberOfKowledgePlaces
is run
pre: KnowledgeSystem.numberOfKnowledgePlaces() = 0
KnowledgeSystem.givesInformation()= False

--The postconditions being true after the running--
post:
numberOfKnowledgePlaces=KnowledgeSystem.numberOfKnowledgePlaces()
@Pre+1
KnowledgeSystem.givesInformation()= True
```

```
OCL
Context: numberOfYearsOld::
--Precon states what must be zero and false
before numberOfYearsOld() is run--
pre:
Email.numberOfYearsOld() = 0
Employee.read() = False
--Post being the number of years as two and true
that employee reads email
post:
Email.numberOfYearsOld()@Pre+2
Employee.read()= True
```

It is also worth having Email implement its interface as a component since I have already put it in the component diagram. It is connected to the KnowledgeSystem component here. The Email class now becomes a component which I comment in the code. I could have Attachment, Archive and Index as components implementing their interfaces too but I didn't put them in the component diagram in the first place so I decided not to do so as I was short of time.



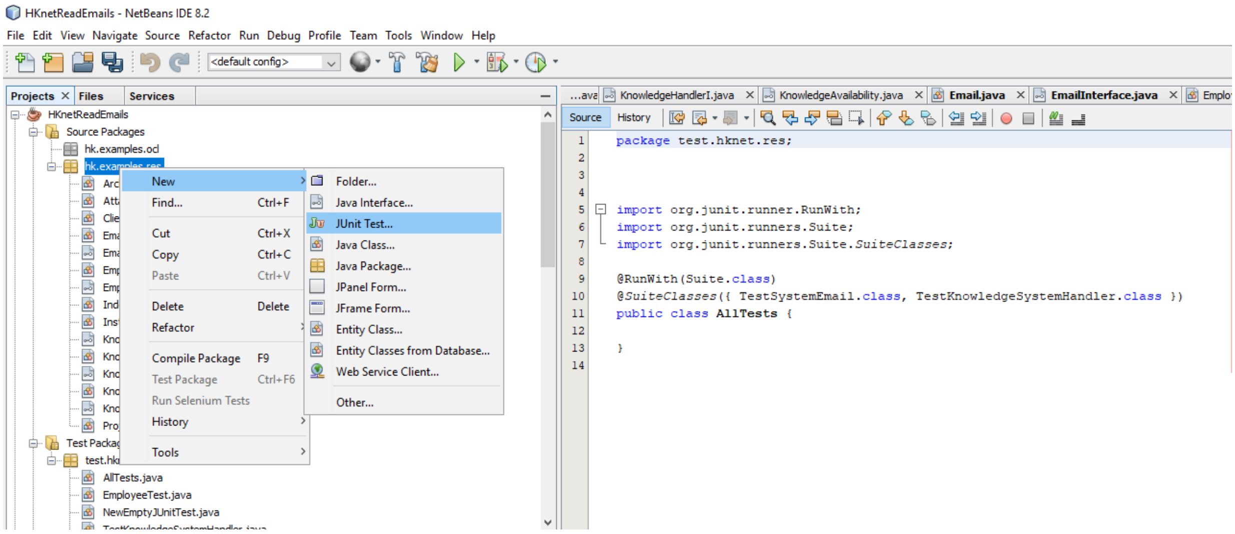
```
HKnetReadEmails
  Source Packages
    hk.examples.ocl
    hk.examples.res
      Archive.java
      Attachment.java
      ClientsEmployee.java
      Email.java
      EmailInterface.java
      Employee.java
      EmployeeInterface.java
      Index.java
      InstanceOfProject.java
      KnowledgeAvailability.java
      KnowledgeHandler.java
      KnowledgeHandlerI.java
      KnowledgeSystem.java
      KnowledgeSystemInterface.java
      Project.java
```

```java
package hk.examples.res;


public interface KnowledgeSystemInterface {

}
```

I have also created an interface for the KnowledgeSystem to put into running. I could add methods here to be implemented such as those in the component diagram for the interface of the KnowledgeSystem component. I was short of time to do so.

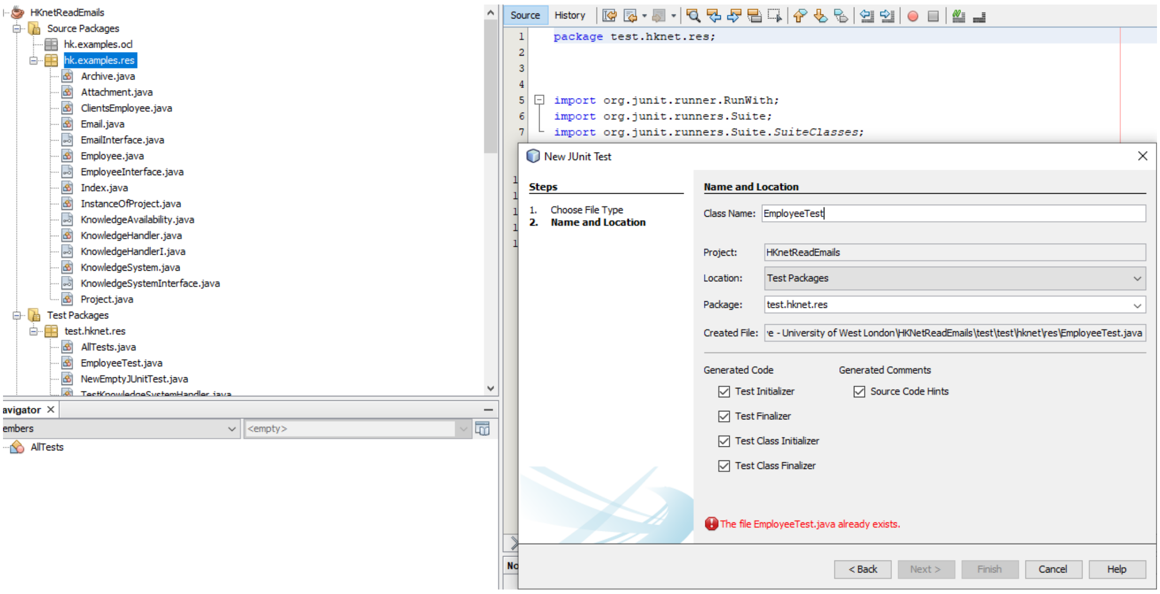
Unit Testing Of The Classes

These are the test files for testing programming code for the classes known as JUnit tests. The Java unit tests are the files in which we write and run code to see if the code we have written in the class files are correct. The code in the test files test the ones done in the files for the classes by being linked to them. I start off testing the ones I did like this:



```
package test.hknet.res;



import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ TestSystemEmail.class, TestKnowledgeSystemHandler.class })
public class AllTests {

}
```

I need to create a test file after creating this Java project. This is done by right clicking on the folder in the src packages and going to New and choosing JUnit Test. To create a test of the Employee class I call the file EmployeeTest.



```
package test.hknet.res;



import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
```

Name and Location

Class Name: EmployeeTest

Project: HKnetReadEmails

Location: Test Packages

Package: test.hknet.res

Created File: 'e - University of West London\HKNetReadEmails\test\test\hknet\res\EmployeeTest.java

Generated Code

☑ Test Initializer

☑ Test Finalizer

☑ Test Class Initializer

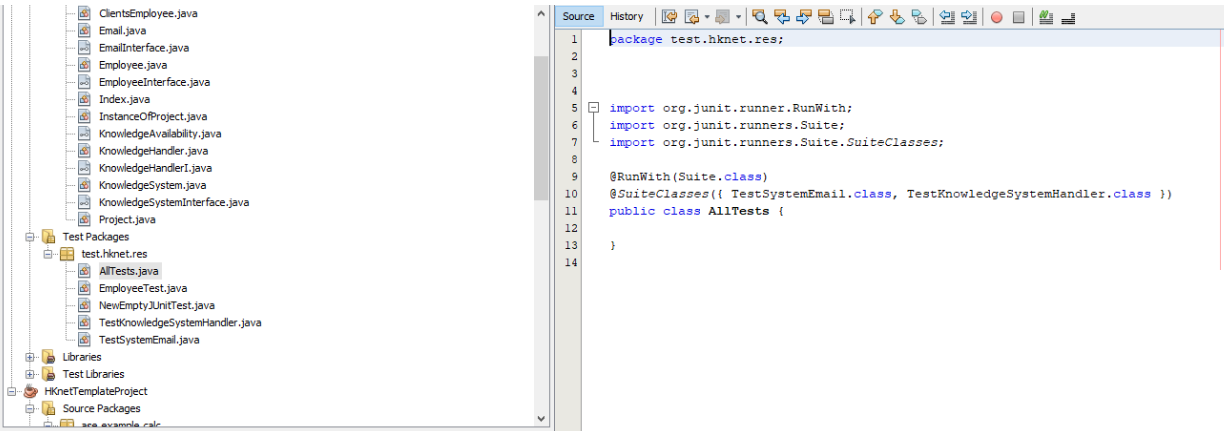
☑ Test Class Finalizer

Generated Comments

☑ Source Code Hints

The file EmployeeTest.java already exists.

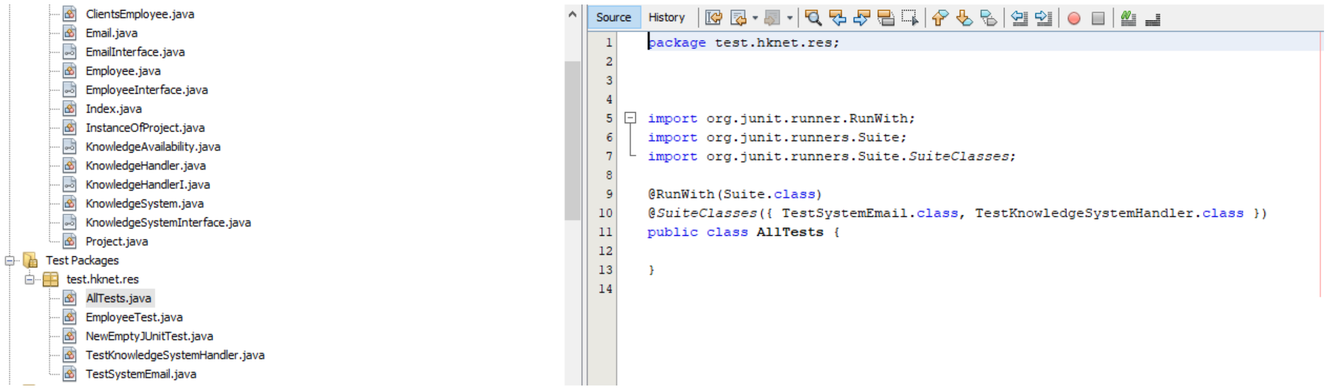
In the dialog box it is the class name that is the name of the test file which is EmployeeTest. Check it is in the right place which is Test Packages and the name of the package itself is correct and click Finish to create the test.



ClientsEmployee.java
Email.java
EmailInterface.java
Employee.java
EmployeeInterface.java
Index.java
InstanceOfProject.java
KnowledgeAvailability.java
KnowledgeHandler.java
KnowledgeHandlerI.java
KnowledgeSystem.java
KnowledgeSystemInterface.java
Project.java
Test Packages
test.hknet.res
AllTests.java
EmployeeTest.java
NewEmptyJUnitTest.java
TestKnowledgeSystemHandler.java
TestSystemEmail.java
Libraries
Test Libraries
HKnetTemplateProject
Source Packages
ace.example.calc

```
package test.hknet.res;


import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ TestSystemEmail.class, TestKnowledgeSystemHandler.class })
public class AllTests {

}
```

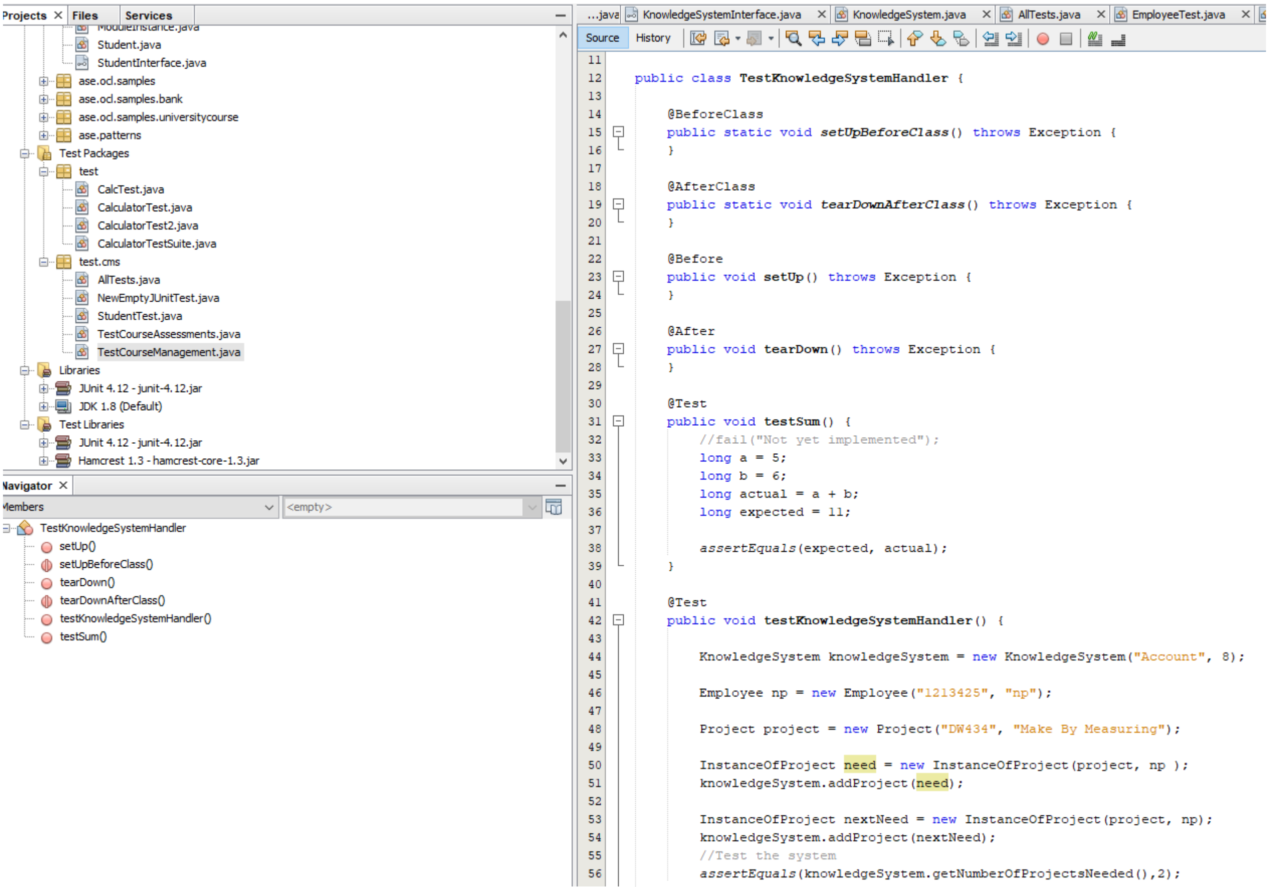
The tests are to be in the test package in the folder as the location Test Packages. I start with the file to test all the classes that are going to be tested in this location.



ClientsEmployee.java
Email.java
EmailInterface.java
Employee.java
EmployeeInterface.java
Index.java
InstanceOfProject.java
KnowledgeAvailability.java
KnowledgeHandler.java
KnowledgeHandlerI.java
KnowledgeSystem.java
KnowledgeSystemInterface.java
Project.java
Test Packages
test.hknet.res
AllTests.java
EmployeeTest.java
NewEmptyJUnitTest.java
TestKnowledgeSystemHandler.java
TestSystemEmail.java

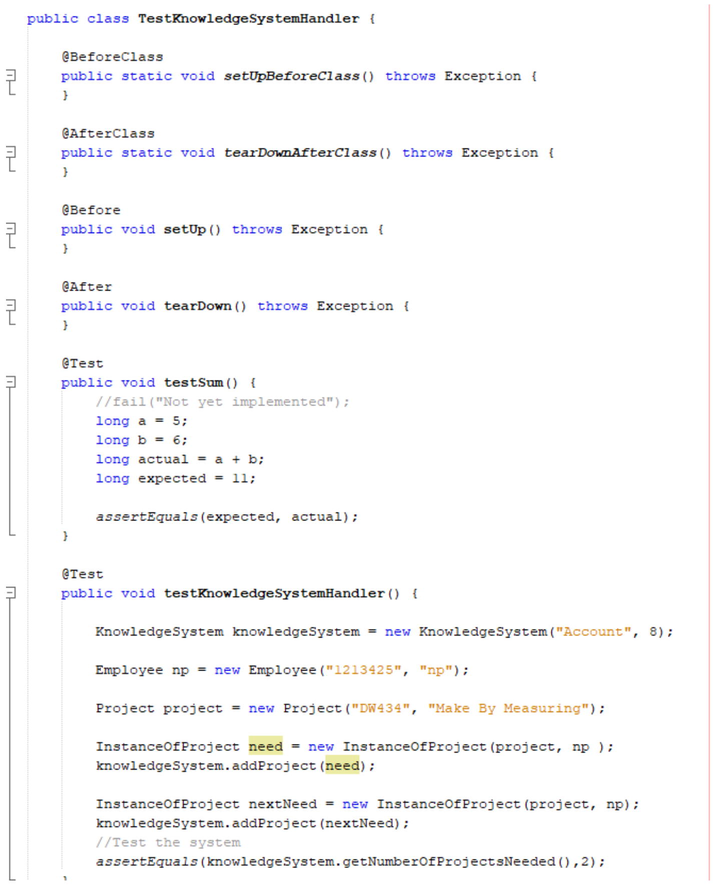
```
package test.hknet.res;


import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ TestSystemEmail.class, TestKnowledgeSystemHandler.class })
public class AllTests {

}
```

The line of code here makes the files test the classes by having the names of them together. Their names have the file extension .class instead of java for them to do the testing. Having checked the correct files are here I start to test the KnowledgeSytemTest file.



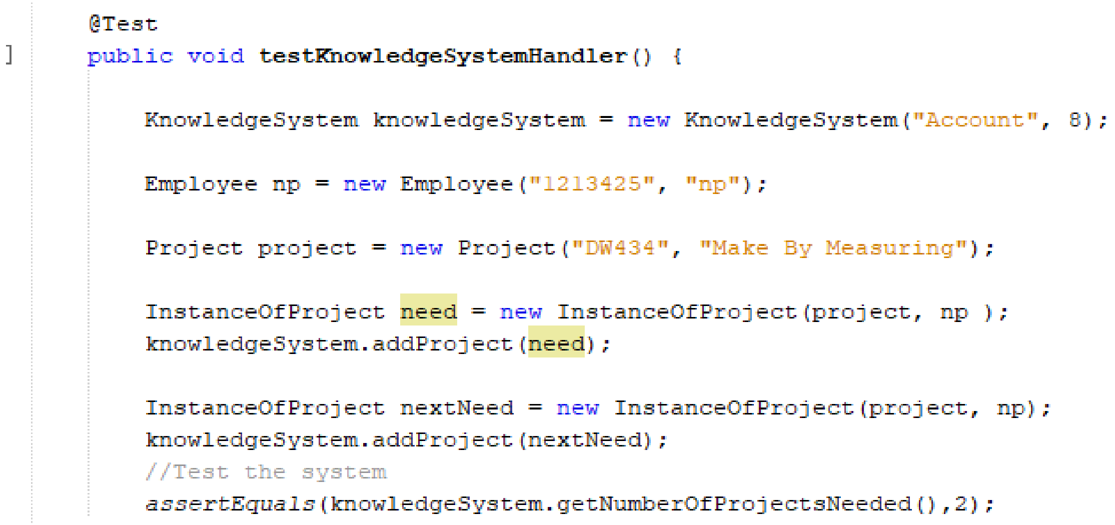
```java

public class TestKnowledgeSystemHandler {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testSum() {
        //fail("Not yet implemented");
        long a = 5;
        long b = 6;
        long actual = a + b;
        long expected = 11;

        assertEquals(expected, actual);
    }

    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Account", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),2);
```

Here the test file starts it being a public class like the files of the classes. What I must focus on here is not the annotated lines of code with the methods just below it. These have been made automatically as call-backs. They are not necessary. What I must focus on is the method testKnowledgeSystemHandler(). That does the test.



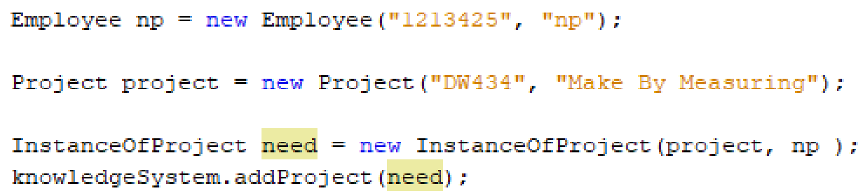
```
public class TestKnowledgeSystemHandler {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testSum() {
        //fail("Not yet implemented");
        long a = 5;
        long b = 6;
        long actual = a + b;
        long expected = 11;

        assertEquals(expected, actual);
    }

    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Account", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),2);
```

This method has the lines of code that links to the names of the classes, written as the code, in the files for the classes. It needs this to test all the programming code I have written to put my use case into running.



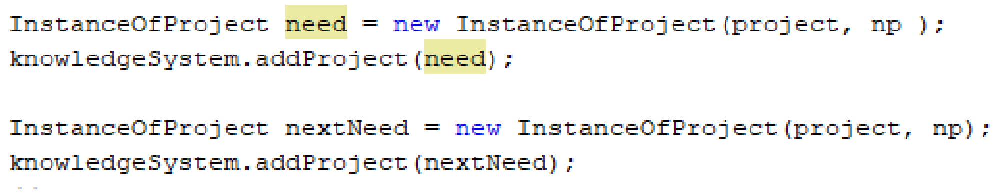
```
    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Account", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),2);
```

So I have here the KnowledgeSystem, Employee and Project. Although there is a one to many relationship between KnowledgeSystem and Employee I have put the InstanceOfProject here because many employees do many projects plus there could be many projects that need to be done.



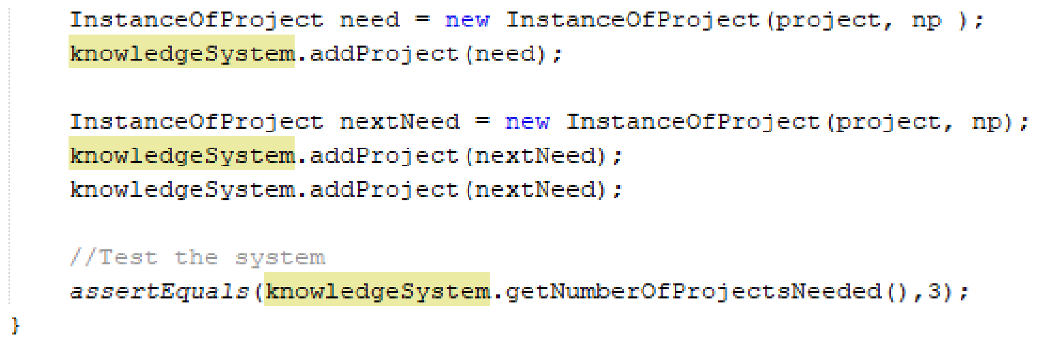
```
Employee np = new Employee("1213425", "np");

Project project = new Project("DW434", "Make By Measuring");

InstanceOfProject need = new InstanceOfProject(project, np );
knowledgeSystem.addProject(need);
```

The instance of project and the need which is the projects that are needed to be done. The word need being after InstanceOfProject. An example of this could be products that are made by measuring this year, for next year or from three years ago.



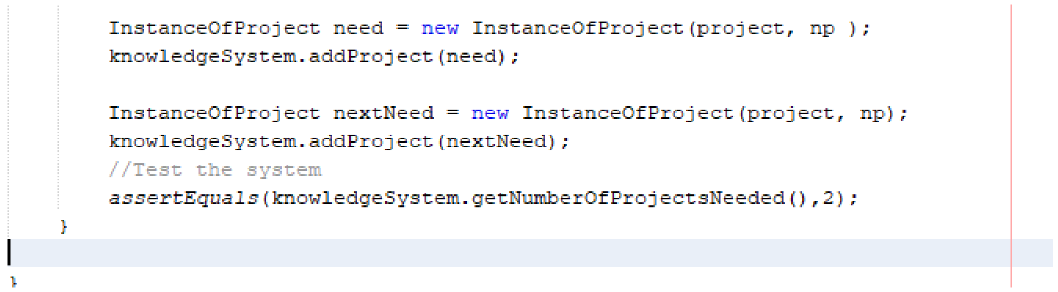
```
InstanceOfProject need = new InstanceOfProject(project, np );
knowledgeSystem.addProject(need);

InstanceOfProject nextNeed = new InstanceOfProject(project, np);
knowledgeSystem.addProject(nextNeed);
```

Then there is the next need being the need for the next project. If so, add the next project to the system. We expect the system to have two projects added to it. We put the same line of code again but with nextNeed to make this happen. To see if this happens I test this method code.



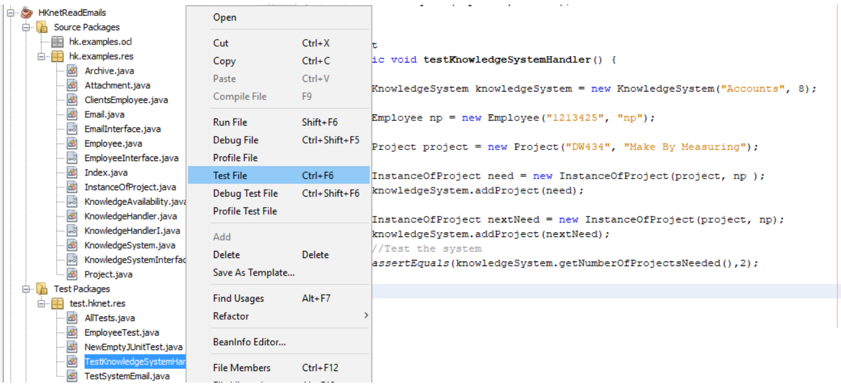
```
    InstanceOfProject need = new InstanceOfProject(project, np );
    knowledgeSystem.addProject(need);

    InstanceOfProject nextNeed = new InstanceOfProject(project, np);
    knowledgeSystem.addProject(nextNeed);
    knowledgeSystem.addProject(nextNeed);

    //Test the system
    assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),3);
}
```

There is the line that expects, in a pair of numbers, the right number to be the same as the left number in the parentheses (). This line begins with assertEquals because of that. We have that here to test the numbers of projects that are needed being added into the system. See if it works here.



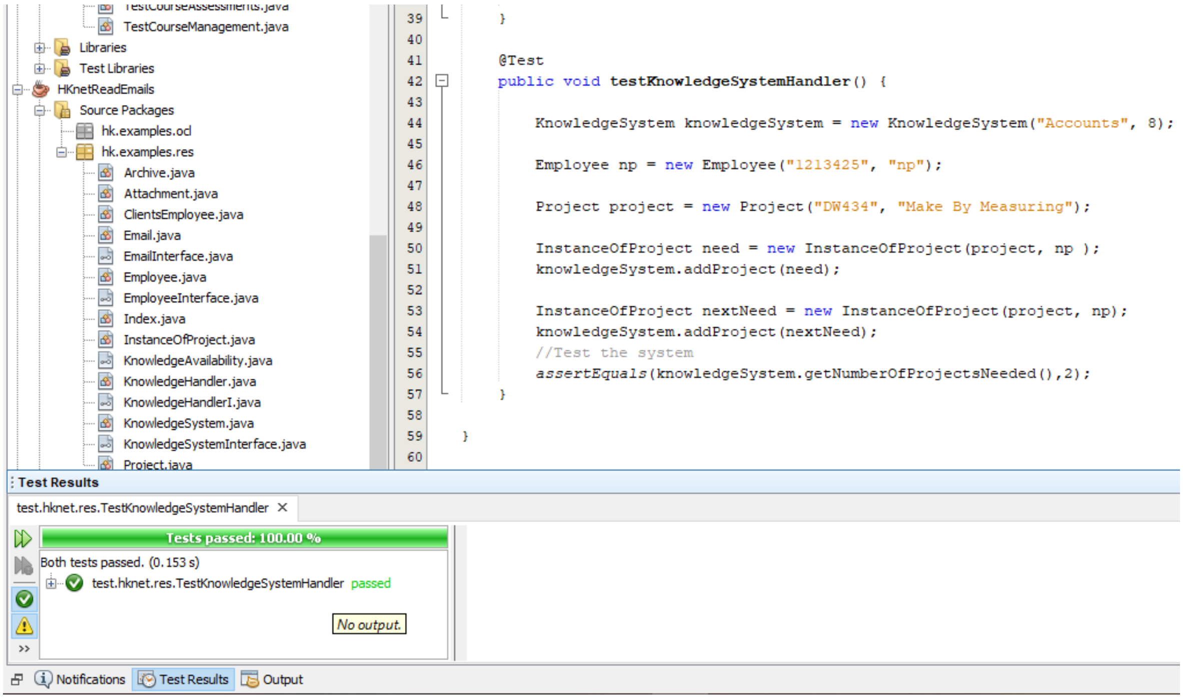
```
        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),2);
    }

}
```

So we have the KnowledgeSystem with the dot notation then the getNumberOfProjectsNeeded method from the KnowledgeSystem class. It is in a pair with the number of projects to being added such as two. This goes with the one line of code having "need" and one having nextNeed. I test this file now.



```
t
ic void testKnowledgeSystemHandler() {

KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accounts", 8);

Employee np = new Employee("1213425", "np");

Project project = new Project("DW434", "Make By Measuring");

InstanceOfProject need = new InstanceOfProject(project, np );
knowledgeSystem.addProject(need);

InstanceOfProject nextNeed = new InstanceOfProject(project, np);
knowledgeSystem.addProject(nextNeed);
//Test the system
assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),2);
```

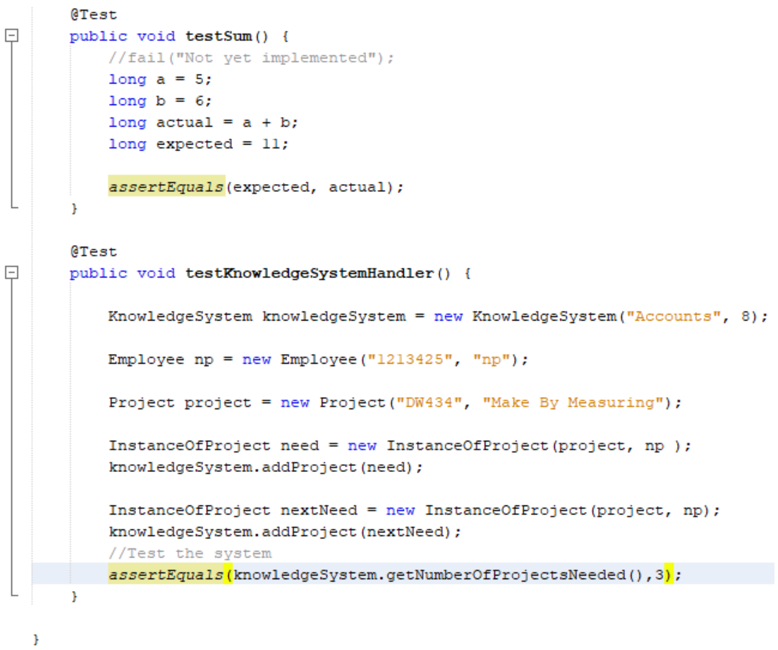
I just right click on the file listed in the Test Packages and select Test file.



```
    }

    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accounts", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),2);
    }

}
```

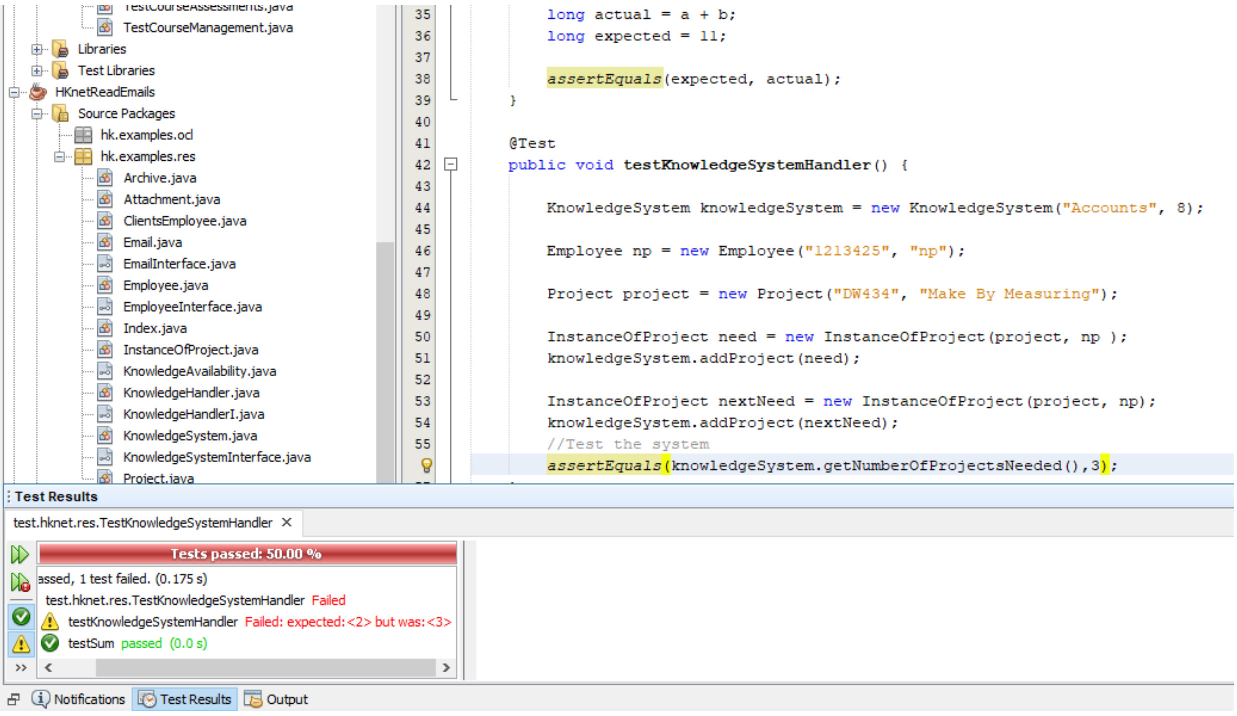
```
Tests passed: 100.00 %
Both tests passed. (0.153 s)
test.hknet.res.TestKnowledgeSystemHandler passed
No output.
```

The test passes. This tell us that the one line with need and having one line with nextNeed means two projects and be added to the system by putting number 2 in the pair in the parentheses of the assertEquals line.



```
    @Test
    public void testSum() {
        //fail("Not yet implemented");
        long a = 5;
        long b = 6;
        long actual = a + b;
        long expected = 11;

        assertEquals(expected, actual);
    }

    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accounts", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),3);
    }

}
```

To be sure these lines I put make the program work correctly I put number three in the parentheses and test this file.



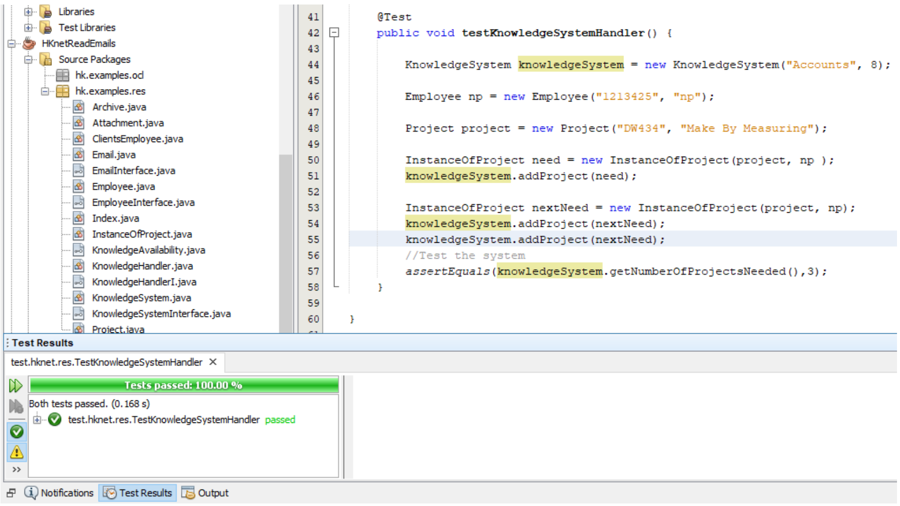
```
        long actual = a + b;
        long expected = 11;

        assertEquals(expected, actual);
    }

    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accounts", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),3);
```

Test Results

test.hknet.res.TestKnowledgeSystemHandler

```
Tests passed: 50.00 %
assed, 1 test failed. (0.175 s)
 test.hknet.res.TestKnowledgeSystemHandler Failed
  testKnowledgeSystemHandler Failed: expected:<2> but was:<3>
  testSum passed (0.0 s)
```

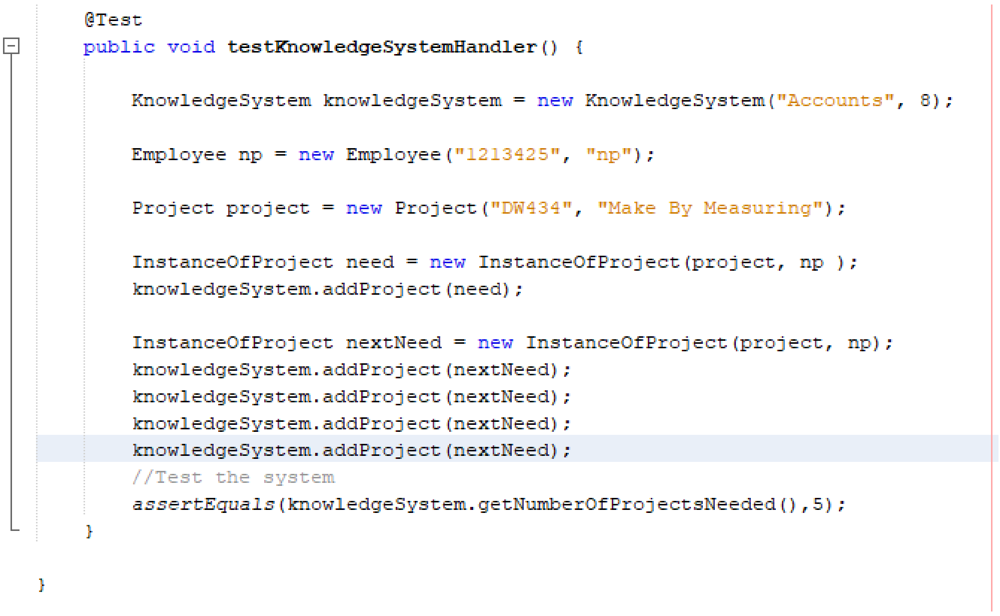
The test does not pass. Having the one line with need and the one line with nextNeed means only two projects are added. Not three. It mentions this in the test results. This shows the test is working. My class names with the dot notations and methods and the rest of the lines of code here in this test method are working correctly.



```
    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accounts", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),3);
    }

}
```

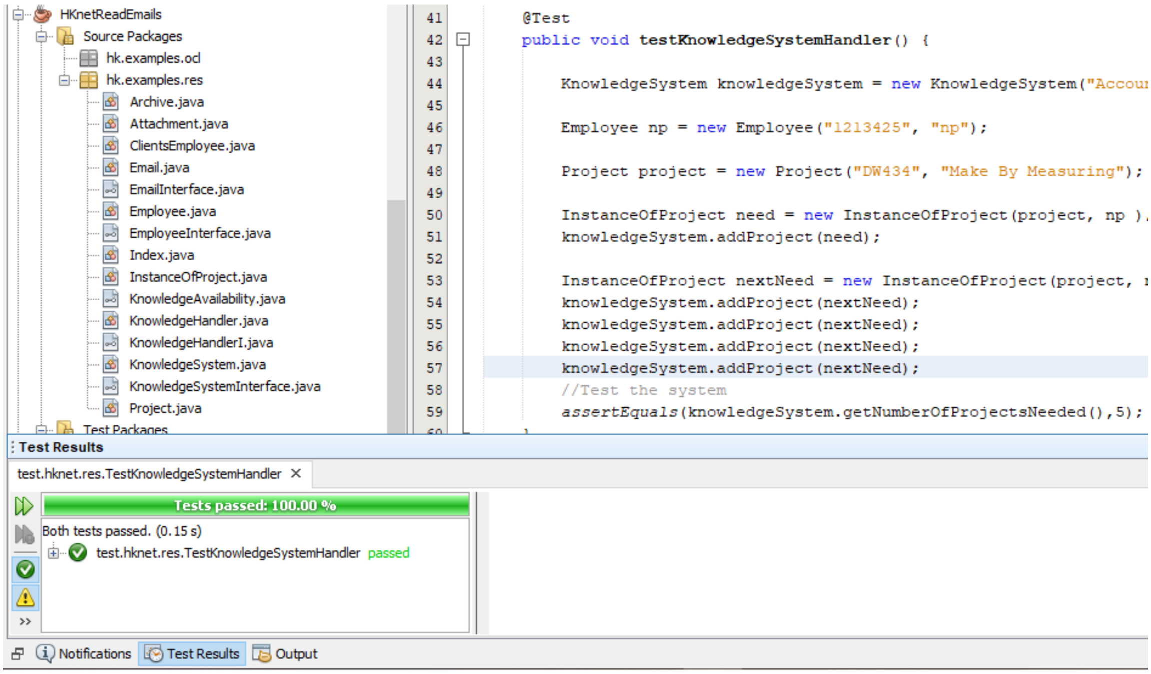
```
Tests passed: 100.00 %
Both tests passed. (0.168 s)
test.hknet.res.TestKnowledgeSystemHandler passed
```

To be sure again I add another line with nextNeed and keep three in the AssertEquals line. I run the file and the test passes. This test method does actually work. You add the same number of lines with “nextNeed”, as the number in the assertEquals parantheses, minus one. You leave the one line with “need” here.



```
    @Test
    public void testKnowledgeSystemHandler() {

        KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accounts", 8);

        Employee np = new Employee("1213425", "np");

        Project project = new Project("DW434", "Make By Measuring");

        InstanceOfProject need = new InstanceOfProject(project, np );
        knowledgeSystem.addProject(need);

        InstanceOfProject nextNeed = new InstanceOfProject(project, np);
        knowledgeSystem.addProject(nextNeed);
        knowledgeSystem.addProject(nextNeed);
        knowledgeSystem.addProject(nextNeed);
        knowledgeSystem.addProject(nextNeed);
        //Test the system
        assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),5);
    }

}
```

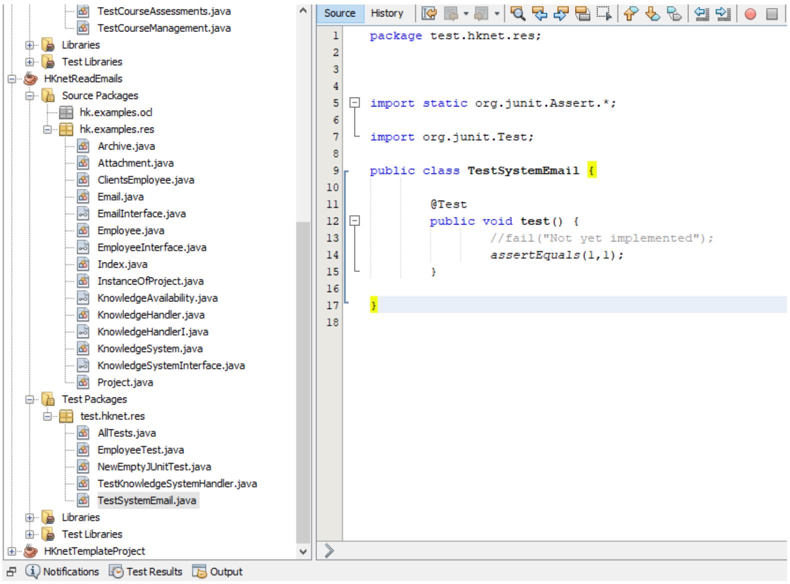
If five projects are needed in the system I add them to it by putting the number five in the parentheses. So I put the line with “nextNeed” four times this time.



```java
@Test
public void testKnowledgeSystemHandler() {

    KnowledgeSystem knowledgeSystem = new KnowledgeSystem("Accou

    Employee np = new Employee("1213425", "np");

    Project project = new Project("DW434", "Make By Measuring");

    InstanceOfProject need = new InstanceOfProject(project, np ).
    knowledgeSystem.addProject(need);

    InstanceOfProject nextNeed = new InstanceOfProject(project, 1
    knowledgeSystem.addProject(nextNeed);
    knowledgeSystem.addProject(nextNeed);
    knowledgeSystem.addProject(nextNeed);
    knowledgeSystem.addProject(nextNeed);
    //Test the system
    assertEquals(knowledgeSystem.getNumberOfProjectsNeeded(),5);
```

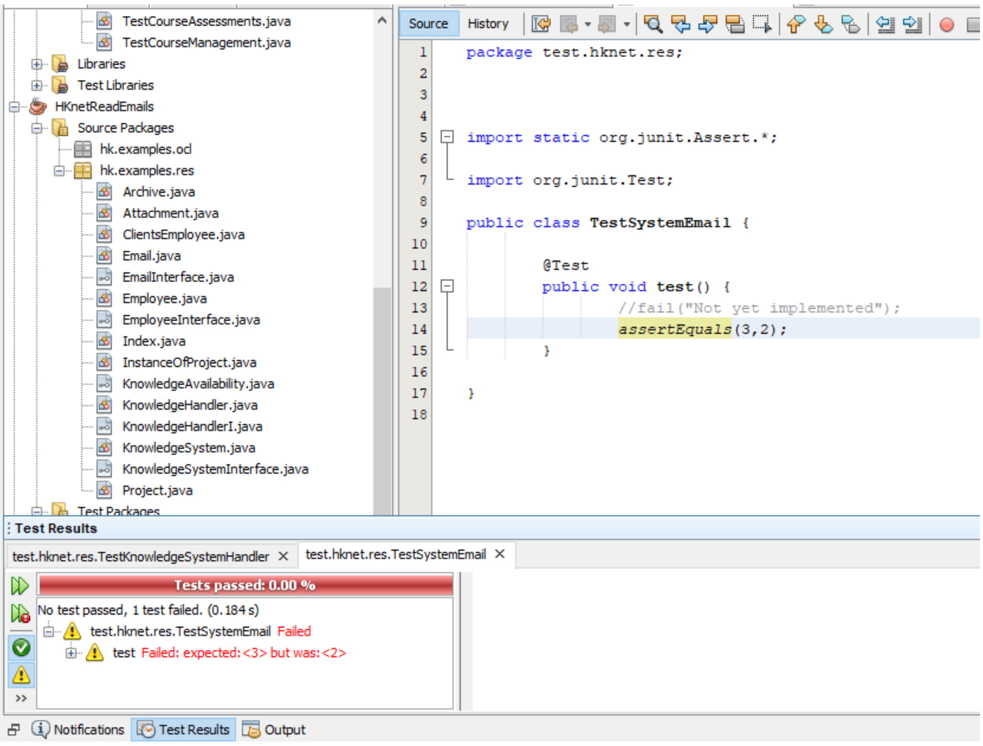
```
Tests passed: 100.00 %
Both tests passed. (0.15 s)
test.hknet.res.TestKnowledgeSystemHandler passed
```

Test the file and it passes as expected. The code I have put for my classes in my use case I have managed to run to make the use case work so far in this file.



```java
package test.hknet.res;



import static org.junit.Assert.*;

import org.junit.Test;

public class TestSystemEmail {

        @Test
        public void test() {
                //fail("Not yet implemented");
                assertEquals(1,1);
        }

}
```

In the file to test the SystemEmail we just have the line beginning with assertEquals. This seems to test the Email by the number being the same as the number on the left. The pair of numbers being in the parentheses of the line.



```
TestCourseAssessments.java
TestCourseManagement.java
Libraries
Test Libraries
HKnetReadEmails
  Source Packages
    hk.examples.od
    hk.examples.res
      Archive.java
      Attachment.java
      ClientsEmployee.java
      Email.java
      EmailInterface.java
      Employee.java
      EmployeeInterface.java
      Index.java
      InstanceOfProject.java
      KnowledgeAvailability.java
      KnowledgeHandler.java
      KnowledgeHandlerI.java
      KnowledgeSystem.java
      KnowledgeSystemInterface.java
      Project.java
  Test Packages
```

```java
package test.hknet.res;


import static org.junit.Assert.*;

import org.junit.Test;

public class TestSystemEmail {

        @Test
        public void test() {
                //fail("Not yet implemented");
                assertEquals(3,2);
        }

}
```

Test Results

test.hknet.res.TestKnowledgeSystemHandler ×
test.hknet.res.TestSystemEmail ×

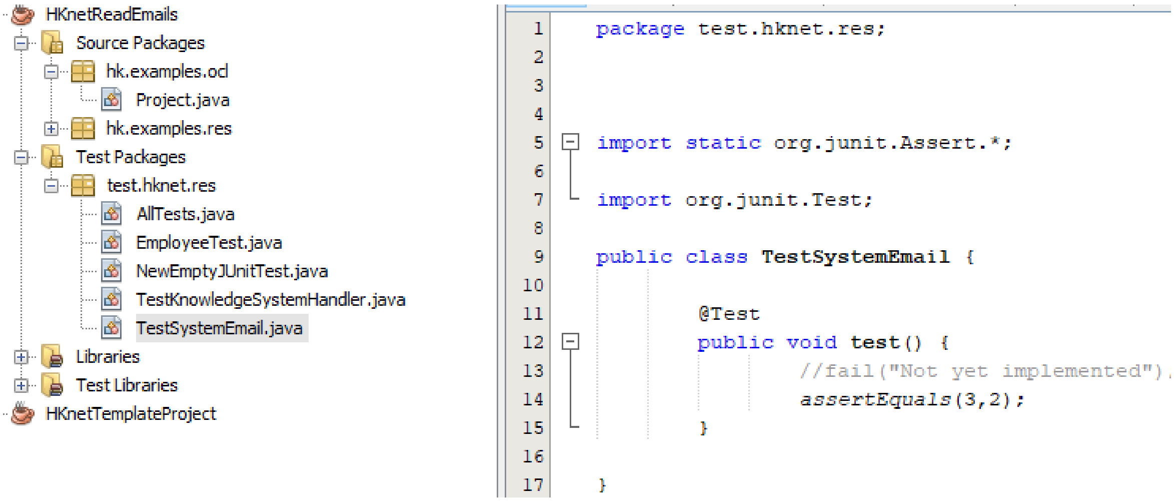
Tests passed: 0.00 %

No test passed, 1 test failed. (0.184 s)

test.hknet.res.TestSystemEmail Failed

test Failed: expected:<3> but was:<2>

If I put (3,2) as the pair and test the file it fails. The two on the right is meant to be three. So, this is the test of SystemEmail that is similar to the one for KnowledgeSystemHandler test. The assertEquals line is like the one in that test with the pair of numbers. It works with the number on the right being equal to the one on the left.



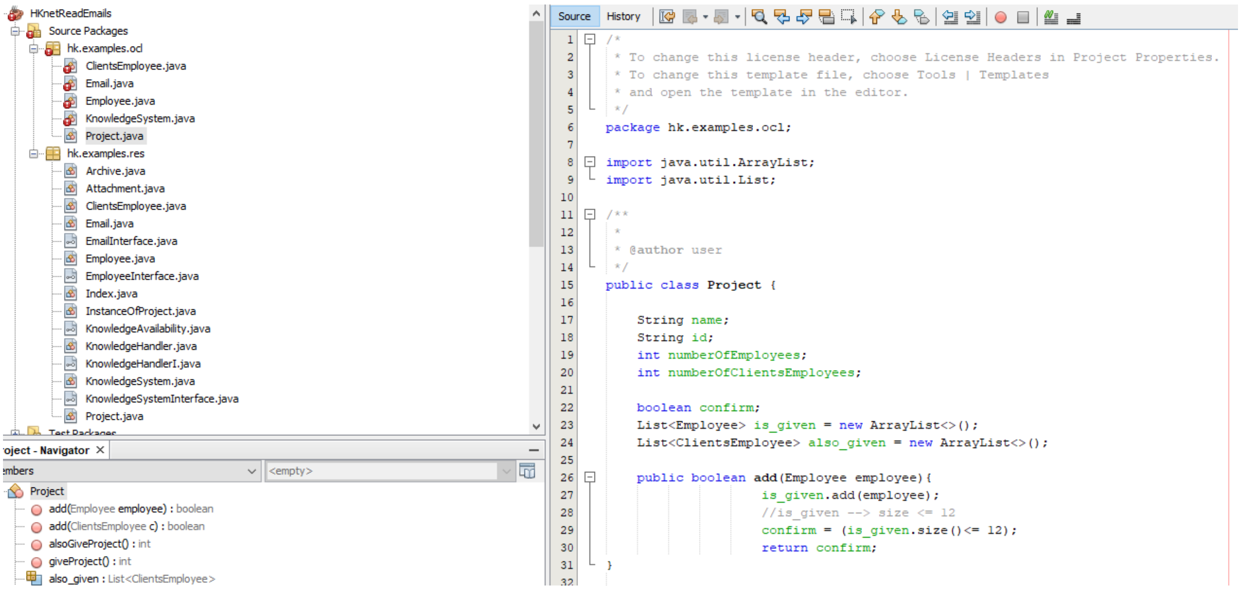
```
HKnetReadEmails
  Source Packages
    hk.examples.od
      Project.java
    hk.examples.res
  Test Packages
    test.hknet.res
      AllTests.java
      EmployeeTest.java
      NewEmptyJUnitTest.java
      TestKnowledgeSystemHandler.java
      TestSystemEmail.java
  Libraries
  Test Libraries
HKnetTemplateProject
```

```java
package test.hknet.res;


import static org.junit.Assert.*;

import org.junit.Test;

public class TestSystemEmail {

        @Test
        public void test() {
                //fail("Not yet implemented").
                assertEquals(3,2);
        }

}
```

I created the Junit test for Employee and an empty Junit in case I needed to test the Employee class I had done the programming code for. I could put code similar to those in the KnowledgeHandlerTest including dot notations, methods, names of classes, links to the code in my class files and parentheses. Put that in the empty Junit and Employee to see if I have made them work with the use case. I could have the line beginning with assertEquals having method and a number as a pair to be equal. Have this to do the test of Employee and another class in the empty Junit test.

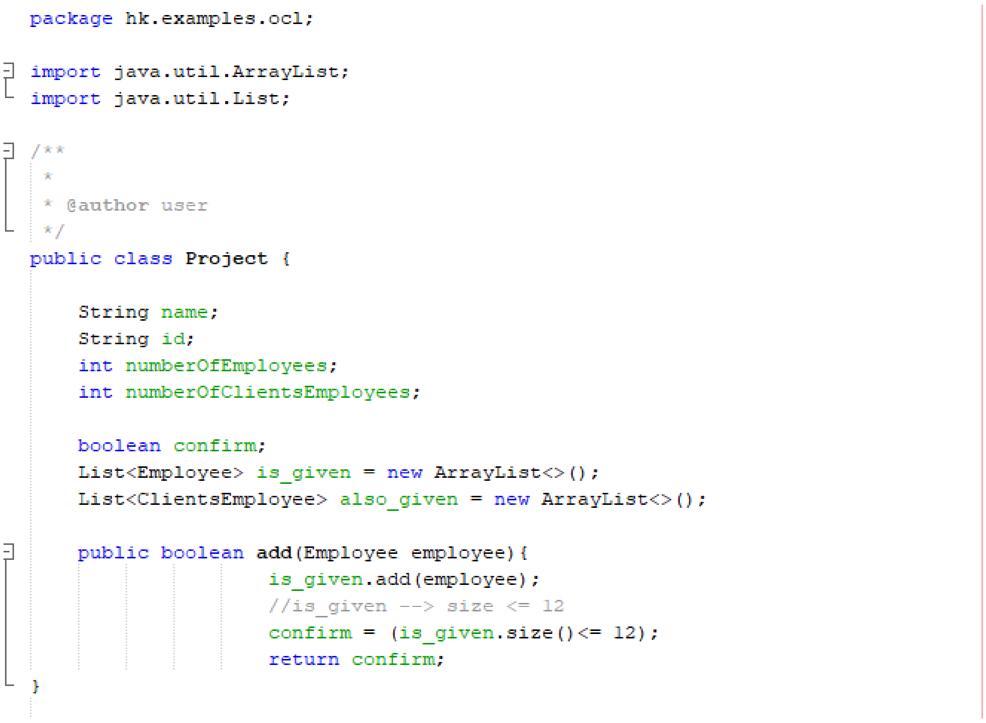
Tests For The Constraints – Object Constraint Language

In a business or workplace, we have limits to the we do our jobs or the number of data that is available to look at. These become the business rules and logic which are basically the constraints. We have language to set these limits i.e., constraints which is in constraint language. This is known as Object Constrain Language. It is mentioned as OCL for short. I have written language to set the limits I have come up with in my Read Emails use case. I have put it together with the Java programming code to see if can work as code to write and run a program. I start with the Project OCL class:



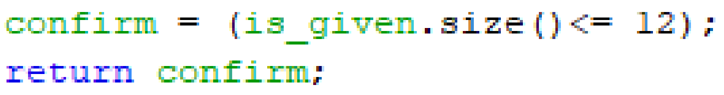
```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.ocl;

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

/**
 *
 * @author user
 */
public class Project {

    String name;
    String id;
    int numberOfEmployees;
    int numberOfClientsEmployees;

    boolean confirm;
    List<Employee> is_given = new ArrayList<>();
    List<ClientsEmployee> also_given = new ArrayList<>();

    public boolean add(Employee employee){
                    is_given.add(employee);
                    //is_given --> size <= 12
                    confirm = (is_given.size()<= 12);
                    return confirm;
}
```

In the project I have listed its attributes at the top without them need to be in private or in view of the public. I need to create two lists. These are the Employee and the ClientsEmployees who are given the project to do. I follow the role names I have put in my class diagram to make the lists.



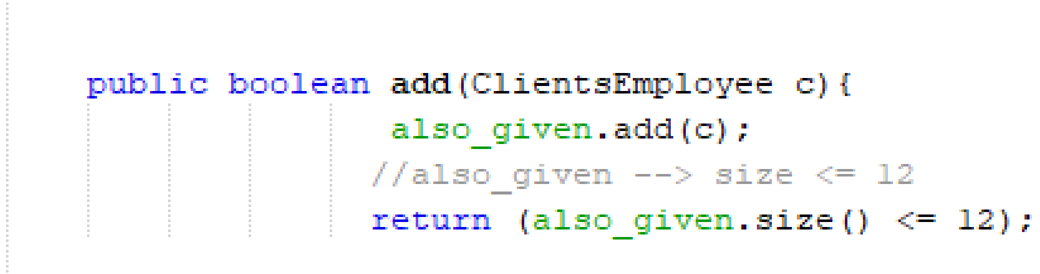
```
package hk.examples.ocl;

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

/**
 *
 * @author user
 */
public class Project {

    String name;
    String id;
    int numberOfEmployees;
    int numberOfClientsEmployees;

    boolean confirm;
    List<Employee> is_given = new ArrayList<>();
    List<ClientsEmployee> also_given = new ArrayList<>();

    public boolean add(Employee employee){
                    is_given.add(employee);
                    //is_given --> size <= 12
                    confirm = (is_given.size()<= 12);
                    return confirm;
}
```

The number of employees who are given the same project to do are to the list. They are added until there is a maximum of twelve of them. I start with the method add with Employee as parameters since I am adding them. Following what I put in my class diagram I add the line with the dot notation to add those who are given the project. Then there is the constraint (OCL) part as a comment.



```
confirm = (is_given.size()<= 12);
return confirm;
```

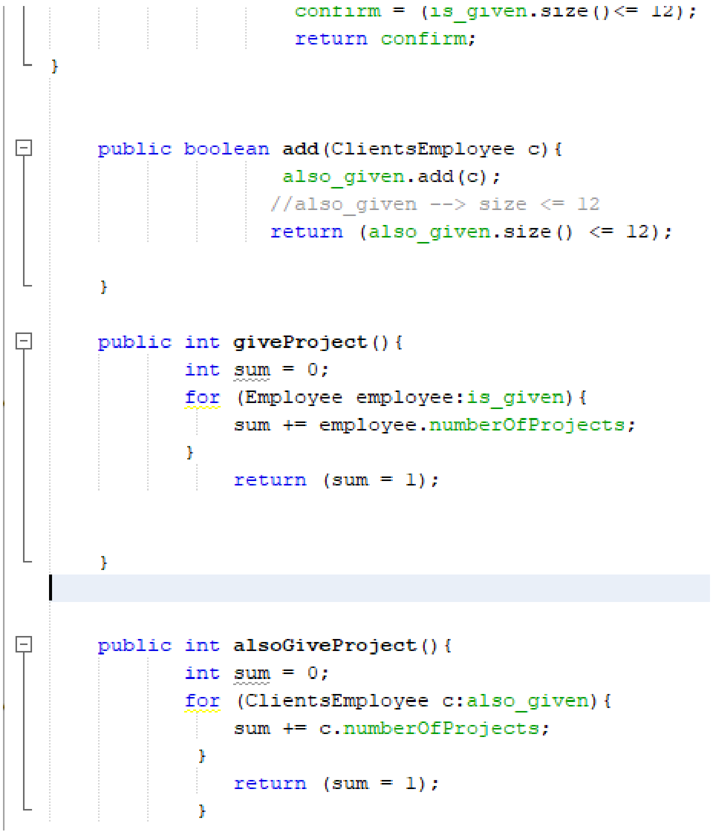
The variable confirm I have declared as Boolean comes into this method. That is given the OCL part in the Java code. The line same as the comment but in parentheses and with size having parentheses to make the method work. I then have return to make this variable return a Boolean value. The value is expected to be Yes, Y, T or True that the number of employees given the same project to do is less than on equal to twelve. No, N, F or false if it is more than that.



```
    public boolean add(ClientsEmployee c){
                    also_given.add(c);
                   //also_given --> size <= 12
                   return (also_given.size() <= 12);
```

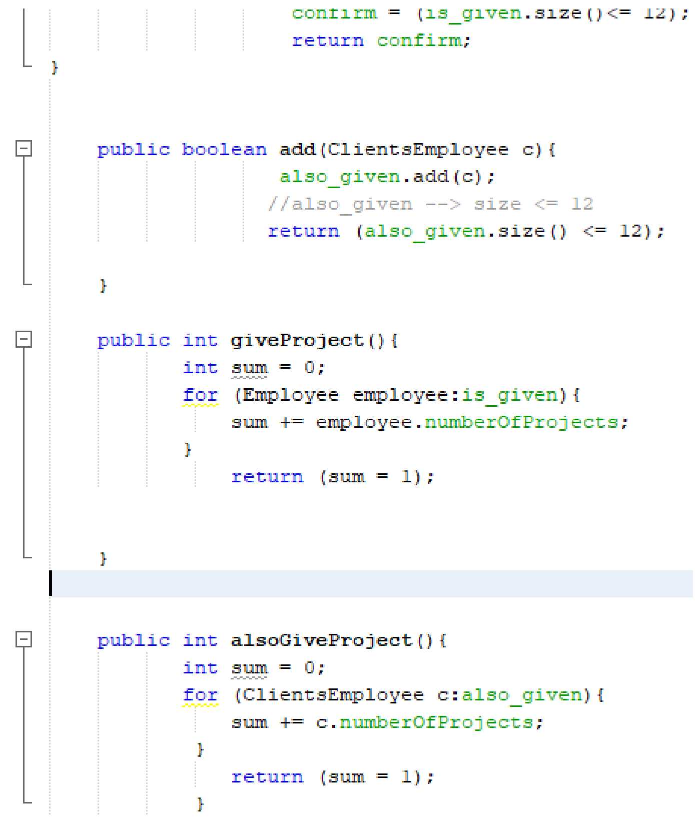
-

I then write the same method for the ClientsEmployee. It is like the one for Employee but works without the variable confirm. It just returns a Boolean saying if the number of clients' employees, who are given the same project to do, is less than or equal to twelve or more than that.



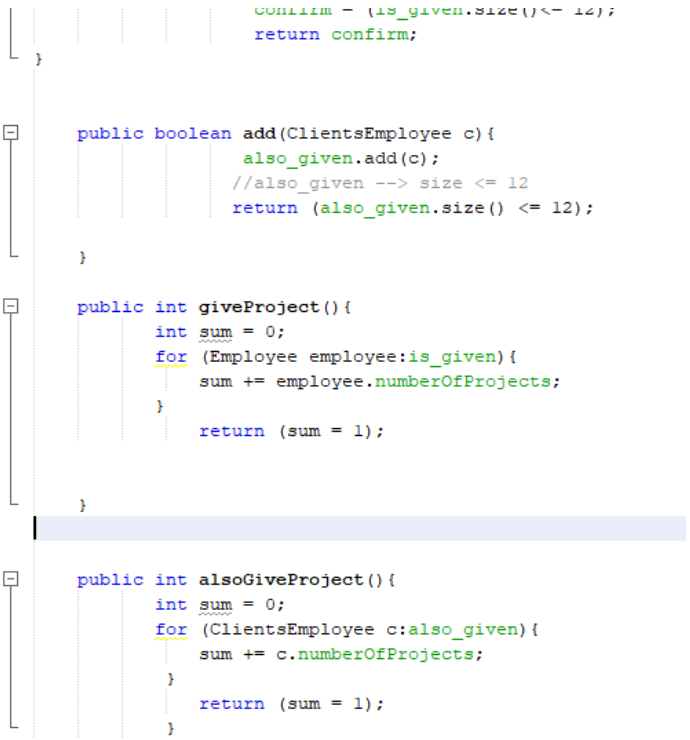
```
                    confirm = (is_given.size()<= 12);
                    return confirm;
    }


        public boolean add(ClientsEmployee c){
                       also_given.add(c);
                      //also_given --> size <= 12
                      return (also_given.size() <= 12);

        }

        public int giveProject(){
                int sum = 0;
                for (Employee employee:is_given){
                    sum += employee.numberOfProjects;
                }
                    return (sum = 1);


        }

        public int alsoGiveProject(){
                int sum = 0;
                for (ClientsEmployee c:also_given){
                    sum += c.numberOfProjects;
                 }
                    return (sum = 1);
                 }
```

The other methods giveProject and alsoGiveProject are for loops for the number of projects each employee or clients' employee is given to do. Starting with the variable sum of being a number that is first set to zero. The sum is added in the loop to the number of projects the Employee can do at one time.



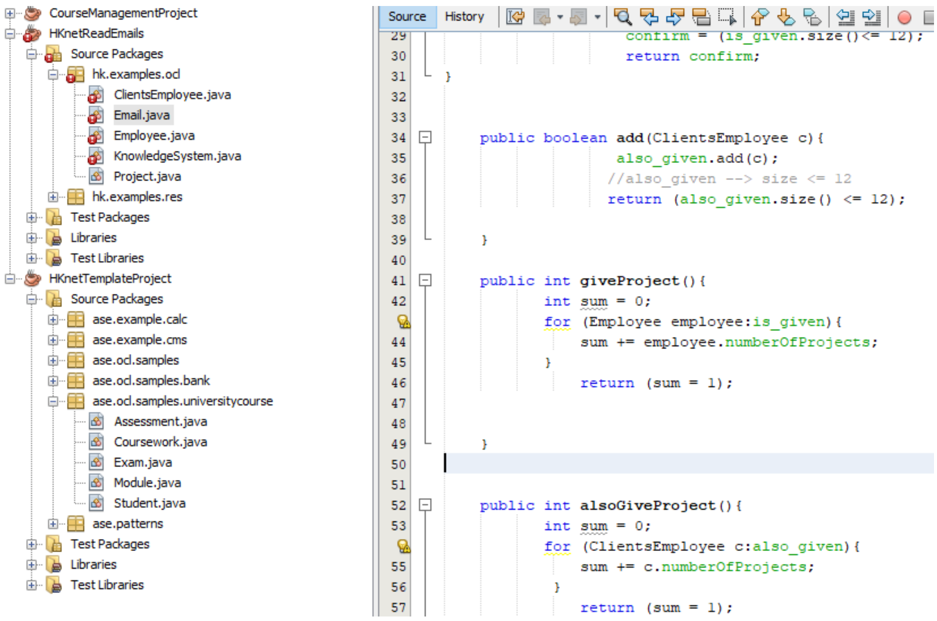
```
                    confirm = (is_given.size()<= 12);
                    return confirm;
    }


        public boolean add(ClientsEmployee c){
                       also_given.add(c);
                      //also_given --> size <= 12
                      return (also_given.size() <= 12);

        }

        public int giveProject(){
                int sum = 0;
                for (Employee employee:is_given){
                    sum += employee.numberOfProjects;
                }
                    return (sum = 1);


        }

        public int alsoGiveProject(){
                int sum = 0;
                for (ClientsEmployee c:also_given){
                    sum += c.numberOfProjects;
                 }
                    return (sum = 1);
                 }
```

The loop works by having the method signature as a link to the Employee with a link to the role name. These are in the parentheses. The sum is added in the loop to the number of projects until it is one.



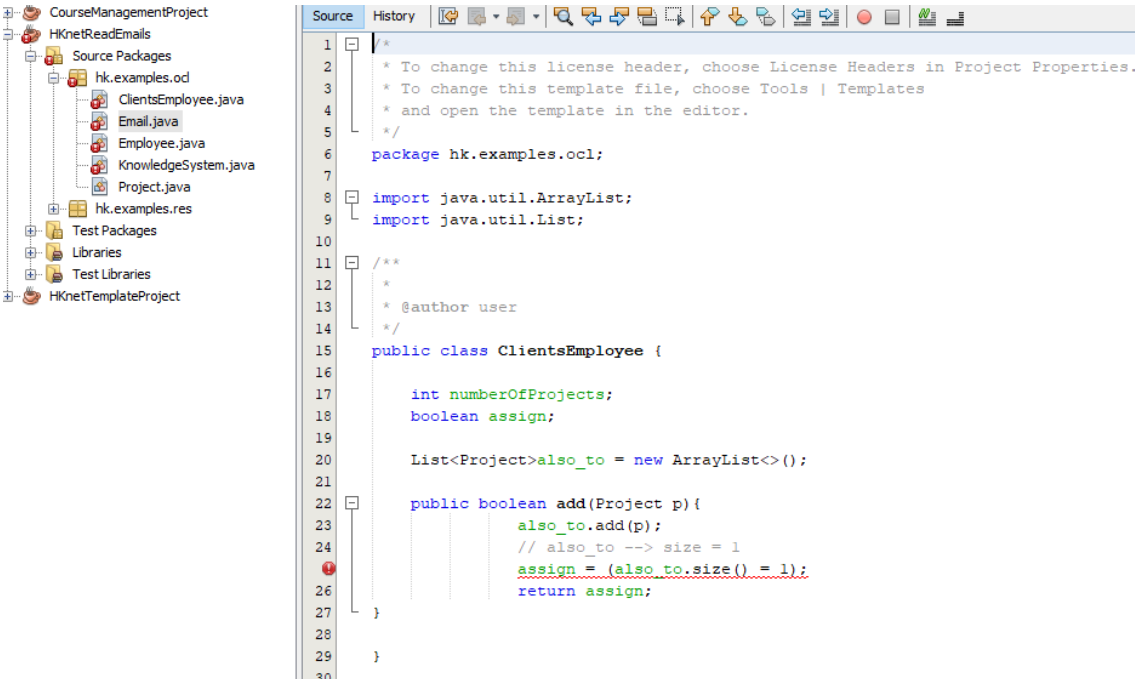
```
            confirm = (is_given.size()<= 12);
            return confirm;
}


    public boolean add(ClientsEmployee c){
             also_given.add(c);
            //also_given --> size <= 12
            return (also_given.size() <= 12);


    }

    public int giveProject(){
        int sum = 0;
        for (Employee employee:is_given){
            sum += employee.numberOfProjects;
        }
            return (sum = 1);


    }


    public int alsoGiveProject(){
        int sum = 0;
        for (ClientsEmployee c:also_given){
            sum += c.numberOfProjects;
         }
            return (sum = 1);
         }
```

Works by the sum adding or if it is equal to the number of projects the Employee is to do. The dot notation putting the class with its attribute numberOfProjects. I have then added the return statement to return the number of projects as one.



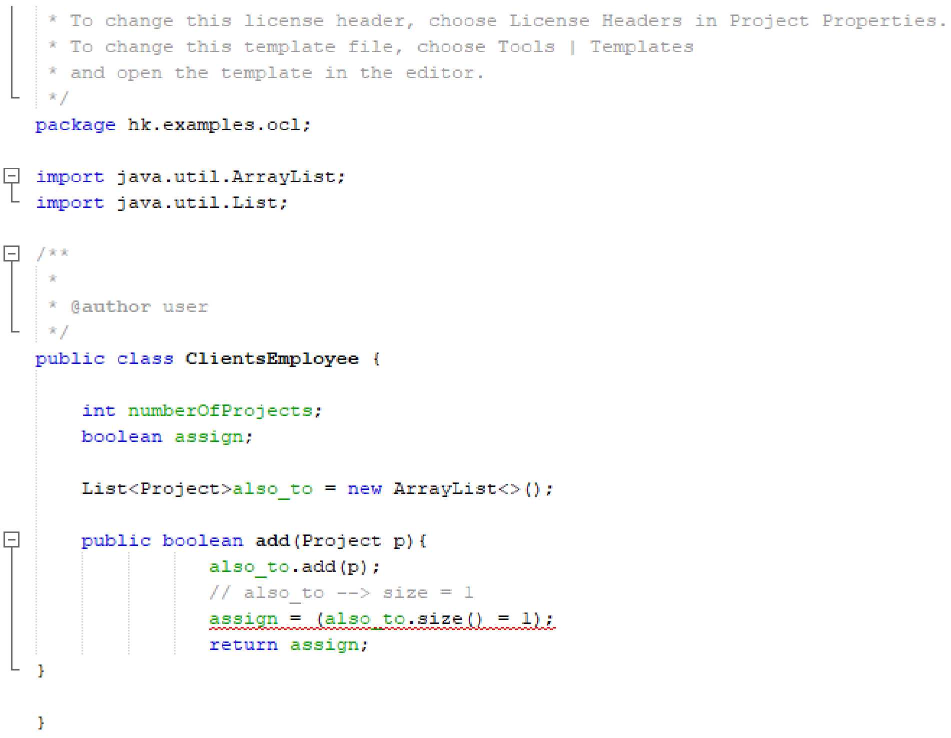
```
                        confirm = (is_given.size()<= 12);
                        return confirm;
}


    public boolean add(ClientsEmployee c){
                     also_given.add(c);
                    //also_given --> size <= 12
                    return (also_given.size() <= 12);

    }

    public int giveProject(){
            int sum = 0;
            for (Employee employee:is_given){
                sum += employee.numberOfProjects;
            }
                return (sum = 1);


    }


    public int alsoGiveProject(){
            int sum = 0;
            for (ClientsEmployee c:also_given){
                sum += c.numberOfProjects;
             }
                return (sum = 1);
```

I have done the same for ClientsEmployee. The same lines to have the number of projects being returned as one. I have put the link to the ClientsEmployee with the class itself and a c. And then the link to the role name. These are in the for loop. The dot notation putting the ClientsEmployee as c and the number together.



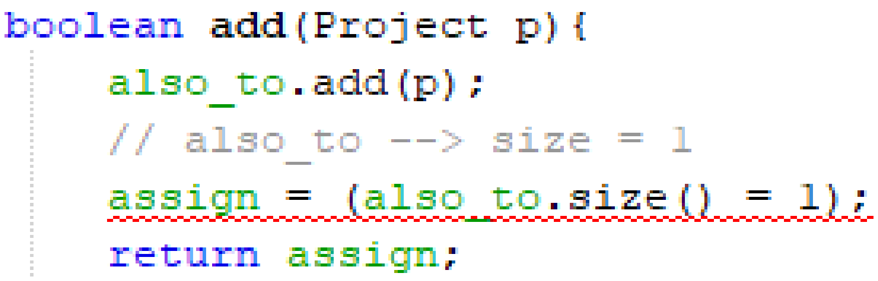
```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.ocl;

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

/**
 *
 * @author user
 */
public class ClientsEmployee {

    int numberOfProjects;
    boolean assign;

    List<Project>also_to = new ArrayList<>();

    public boolean add(Project p){
                also_to.add(p);
                // also_to --> size = 1
                assign = (also_to.size() = 1);
                return assign;
}

}
```

In the ClientsEmployee OCL class I have listed attribute numberOfProjects being of type number i.e., int and not viewed as private. I have declared the variable assign as Boolean. Like the Project OCL class I have created a list of the projects. This is expected to be one project. So I have put the add method to add one project.



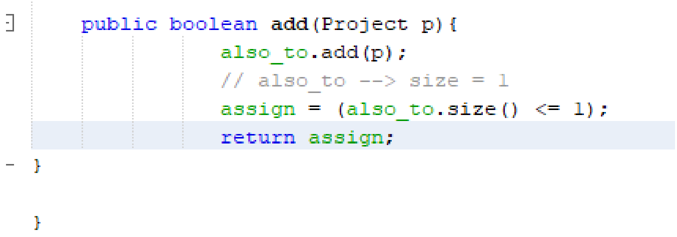
```
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.ocl;

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

/**
 *
 * @author user
 */
public class ClientsEmployee {

    int numberOfProjects;
    boolean assign;

    List<Project>also_to = new ArrayList<>();

    public boolean add(Project p){
                 also_to.add(p);
                 // also_to --> size = 1
                 assign = (also_to.size() = 1);
                 return assign;
}

}
```

Because I was short of time, I have not corrected the errors in the code. If it was correct you can get an idea from it that the method returns a Boolean value saying whether or not one project has been added. It is just like the add methods in the Project OCL class but with Project being added to the list instead of Employee and Clients' employee.



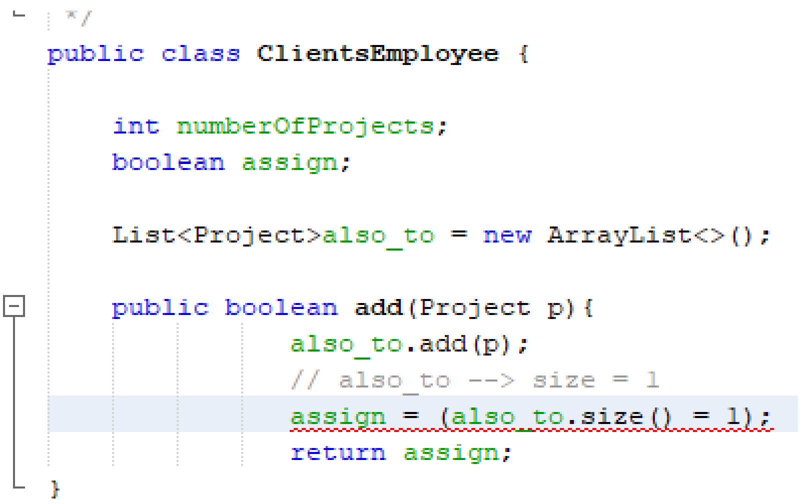
```
boolean add(Project p){
    also_to.add(p);
    // also_to --> size = 1
    assign = (also_to.size() = 1);
    return assign;
```

It has the role name put together with add and the link p to Project, by the dot notation. And then the OCL part commented out. The variable assign makes the method Boolean by returning the Boolean value. It is given the OCL part in parentheses. The return statement makes it do so.



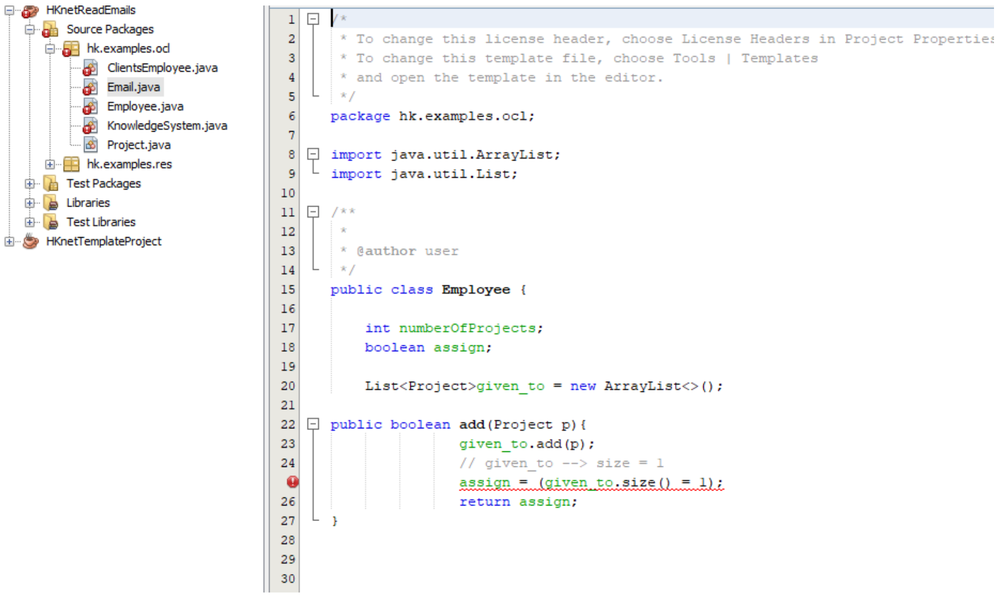
```
    public boolean add(Project p){
                    also_to.add(p);
                    // also_to --> size = 1
                    assign = (also_to.size() <= 1);
                    return assign;
}

}
```

However the error here is the size of the collection being one rather than less than or equal to one. If I put less than or equal to one sign it corrects the error in this code.



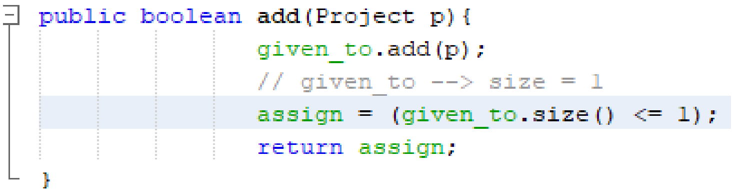
```
 */
public class ClientsEmployee {

    int numberOfProjects;
    boolean assign;

    List<Project>also_to = new ArrayList<>();

    public boolean add(Project p){
                    also_to.add(p);
                    // also_to --> size = 1
                    assign = (also_to.size() = 1);
                    return assign;
}
```

The number of projects is only one here which is why I am keeping the signs as equal to one. Problem is this leaves the code with the errors but I know how to put it right.



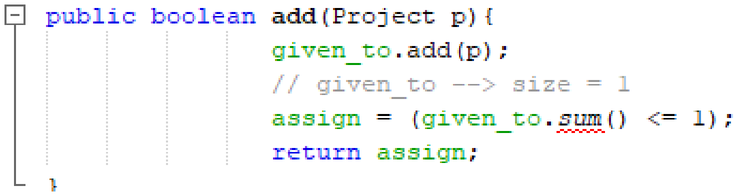
```
/*
 * To change this license header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.ocl;

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

/**
 *
 * @author user
 */
public class Employee {

    int numberOfProjects;
    boolean assign;

    List<Project>given_to = new ArrayList<>();

public boolean add(Project p){
                given_to.add(p);
                // given_to --> size = 1
                assign = (given_to.size() = 1);
                return assign;
}
```

I do the same for Employee. It just has the other role name. I can correct the same code here which has the same error by putting the less than sign with the equals. As I was short of time, I have not managed to correct the same error.



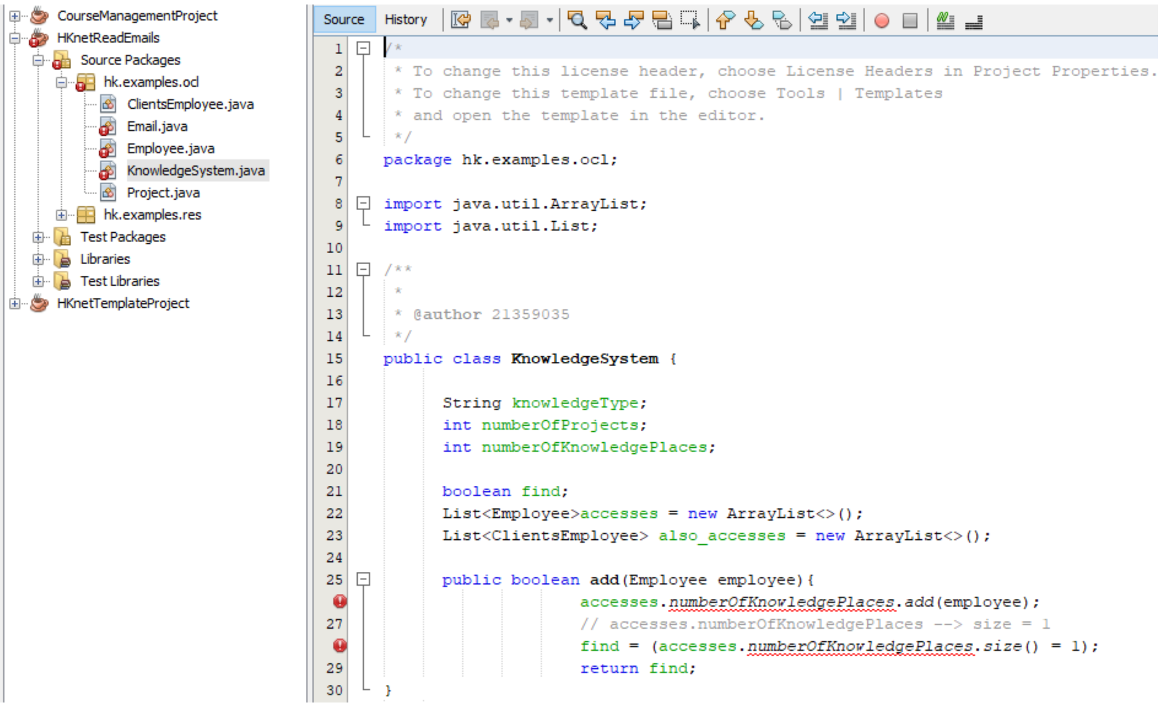
```
public boolean add(Project p){
                given_to.add(p);
                // given_to --> size = 1
                assign = (given_to.size() <= 1);
                return assign;
}
```

If I add the less than sign the error goes. But I leave it as equals one with the error because the number of projects is only one. The other OCL operation for collections added together, as a total, as a sum rather than the size of them could be put here. The operation given as sum().



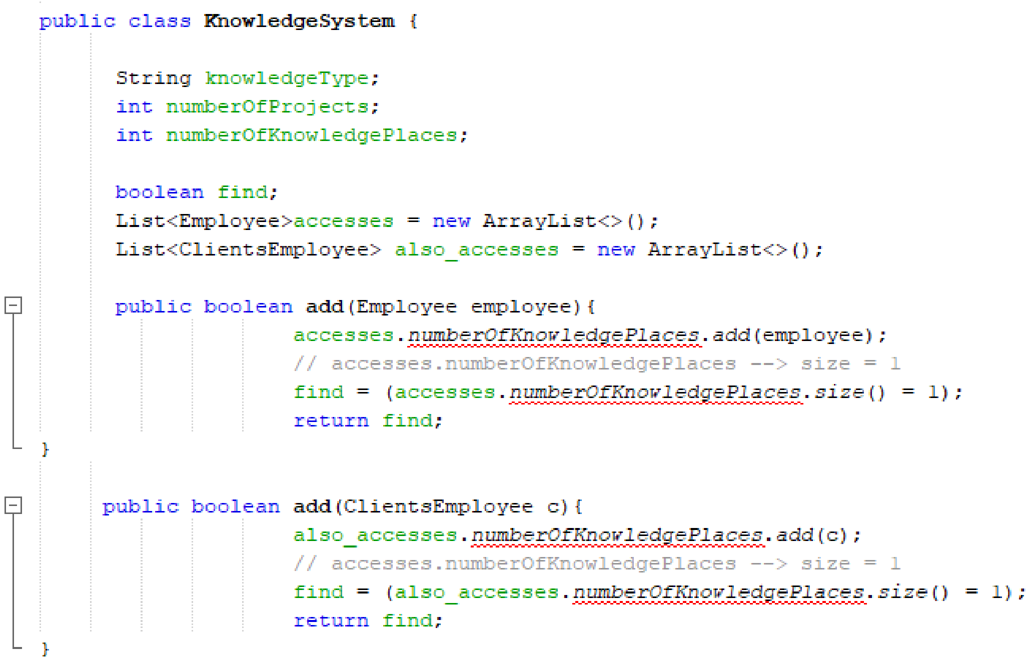
```
public boolean add(Project p){
                given_to.add(p);
                // given_to --> size = 1
                assign = (given_to.sum() <= 1);
                return assign;
}
```

If I put that here in the code it still makes an error. I leave it as size because the collection of projects here is the size rather than the total of them added together.



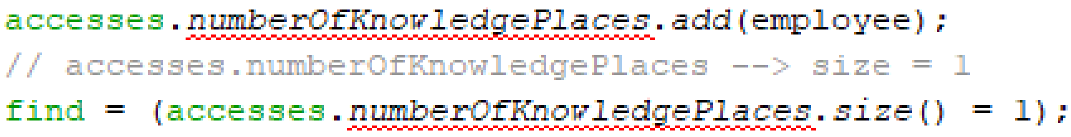
```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.ocl;

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

/**
 *
 * @author 21359035
 */
public class KnowledgeSystem {

        String knowledgeType;
        int numberOfProjects;
        int numberOfKnowledgePlaces;

        boolean find;
        List<Employee>accesses = new ArrayList<>();
        List<ClientsEmployee> also_accesses = new ArrayList<>();

        public boolean add(Employee employee){
                accesses.numberOfKnowledgePlaces.add(employee);
                // accesses.numberOfKnowledgePlaces --> size = 1
                find = (accesses.numberOfKnowledgePlaces.size() = 1);
                return find;
}
```

For KnowledgeSystem I have put its three attributes not visible as private like for the rest of the classes. Like the OCL for the other classes I declare a variable of data type Boolean called find. I then create the lists of Employee and Clients' Employee. The lists of them to find and share their knowledge in one place when given the project to do and asked to read the emails of past projects.



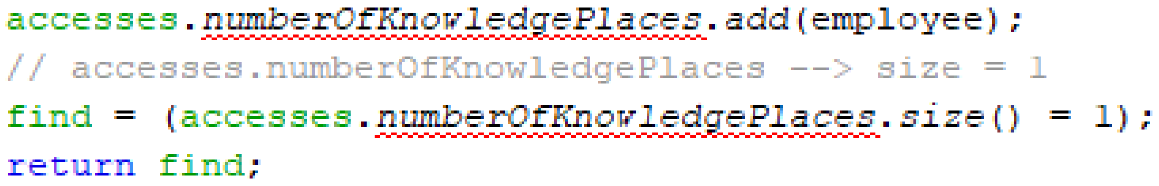
```
public class KnowledgeSystem {

        String knowledgeType;
        int numberOfProjects;
        int numberOfKnowledgePlaces;

        boolean find;
        List<Employee>accesses = new ArrayList<>();
        List<ClientsEmployee> also_accesses = new ArrayList<>();

        public boolean add(Employee employee){
                        accesses.numberOfKnowledgePlaces.add(employee);
                        // accesses.numberOfKnowledgePlaces --> size = 1
                        find = (accesses.numberOfKnowledgePlaces.size() = 1);
                        return find;
}

       public boolean add(ClientsEmployee c){
                        also_accesses.numberOfKnowledgePlaces.add(c);
                        // accesses.numberOfKnowledgePlaces --> size = 1
                        find = (also_accesses.numberOfKnowledgePlaces.size() = 1);
                        return find;
}
```

I put the add method similar to the ones I did in Project. Because I was short of time I did not manage to correct the errors in the code. It does give ideas of the limits to the number of places the workers prefer to go to for getting and sharing their knowledge.



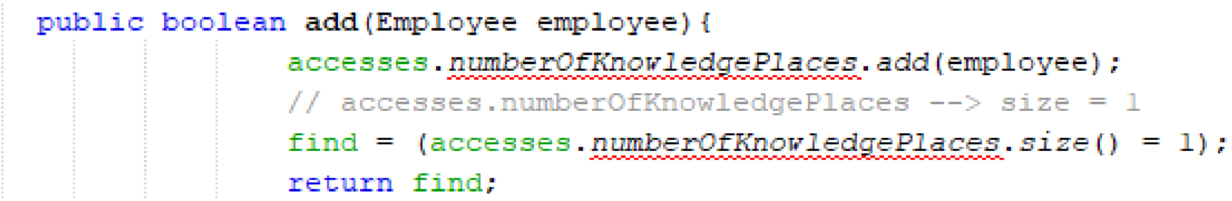
```
accesses.numberOfKnowledgePlaces.add(employee);
// accesses.numberOfKnowledgePlaces --> size = 1
find = (accesses.numberOfKnowledgePlaces.size() = 1);
```

The language for the limit is put as the role name and the class attribute name together with the dot notation. And then the adding of the employee. This is the OCL line I have written in Java. The OCL part is below that as a comment.



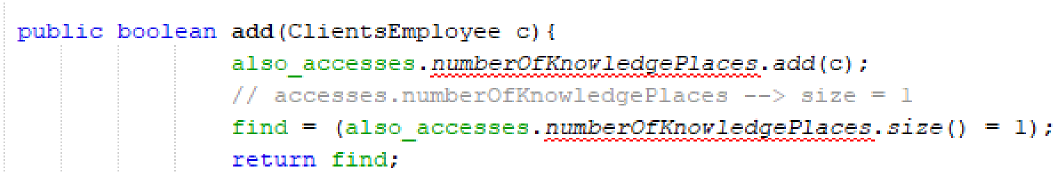
```
accesses.numberOfKnowledgePlaces.add(employee);
// accesses.numberOfKnowledgePlaces --> size = 1
find = (accesses.numberOfKnowledgePlaces.size() = 1);
return find;
```

The variable find makes the limit work as programming code. The variable having the role name, class attribute and size put together by the dot notation. So size is the operation in OCL for the collection of places of knowledge.



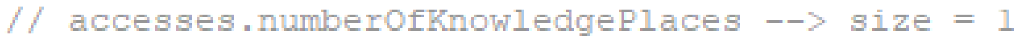
```
public boolean add(Employee employee){
                accesses.numberOfKnowledgePlaces.add(employee);
                // accesses.numberOfKnowledgePlaces --> size = 1
                find = (accesses.numberOfKnowledgePlaces.size() = 1);
                return find;
```

Like the add method code for the other classes it has parentheses to make the limit work. Then there is the equal to one. I let it be equal one rather than equal to or less than because the limit is one place to get and share the knowledge. Although this is not the only reason the code has errors.



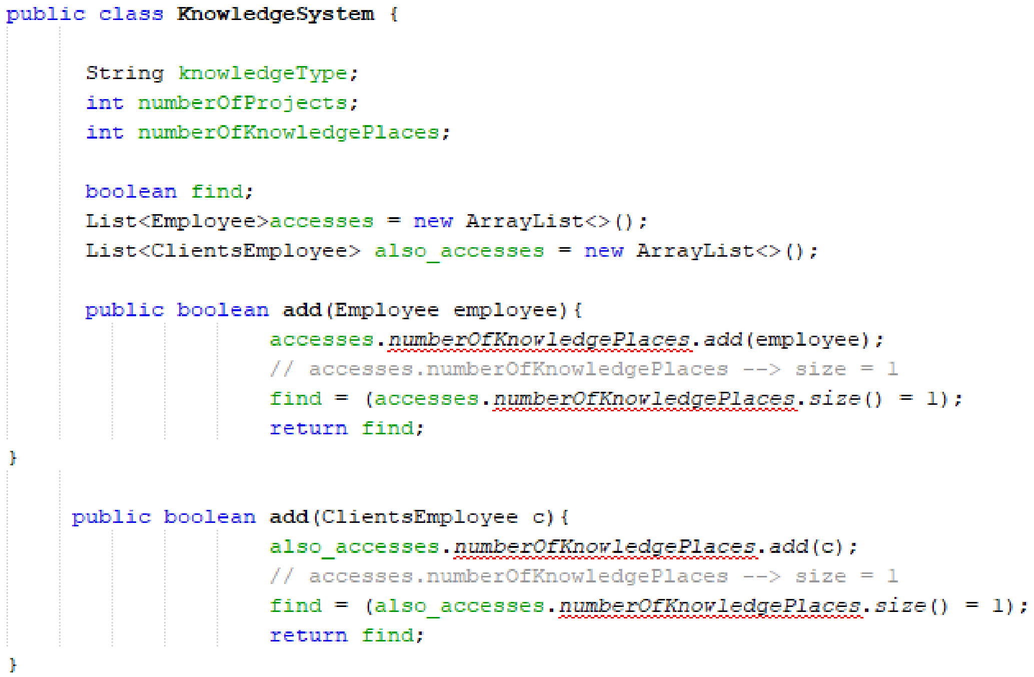
```
public boolean add(ClientsEmployee c){
                also_accesses.numberOfKnowledgePlaces.add(c);
                // accesses.numberOfKnowledgePlaces --> size = 1
                find = (also_accesses.numberOfKnowledgePlaces.size() = 1);
                return find;
```

I do the same for ClientsEmployee. I put the link c in its method signature and the other role name. So they are both accessing the system to share the knowledge.



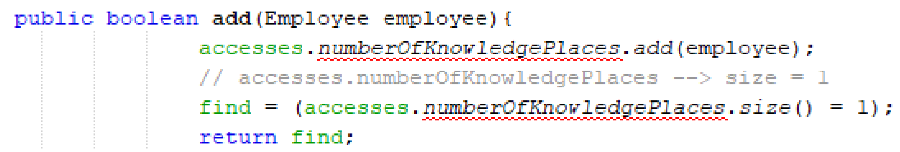
```
// accesses.numberOfKnowledgePlaces --> size = 1
```

The same OCL part here I comment is known as the invariant in OCL. This being the role name and the class attribute put together by the dot notation. The dashed arrow pointing to the operation which is the one for collections such as size or sum being equal to the limit. The limit which is usually a number. This is done by following what I have put in my class diagram.

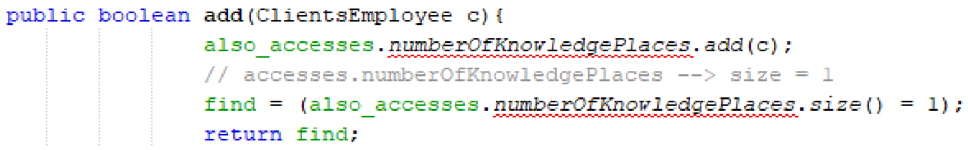


```
public class KnowledgeSystem {

    String knowledgeType;
    int numberOfProjects;
    int numberOfKnowledgePlaces;

    boolean find;
    List<Employee>accesses = new ArrayList<>();
    List<ClientsEmployee> also_accesses = new ArrayList<>();

    public boolean add(Employee employee){
                accesses.numberOfKnowledgePlaces.add(employee);
                // accesses.numberOfKnowledgePlaces --> size = 1
                find = (accesses.numberOfKnowledgePlaces.size() = 1);
                return find;
}

    public boolean add(ClientsEmployee c){
                also_accesses.numberOfKnowledgePlaces.add(c);
                // accesses.numberOfKnowledgePlaces --> size = 1
                find = (also_accesses.numberOfKnowledgePlaces.size() = 1);
                return find;
}
```

The context here is the class the limit is for which is KnowledgeSystem. It is likely to be that rather than Employee or Clients' Employee. These classes and the Project are the context I have given on their OCL class files.

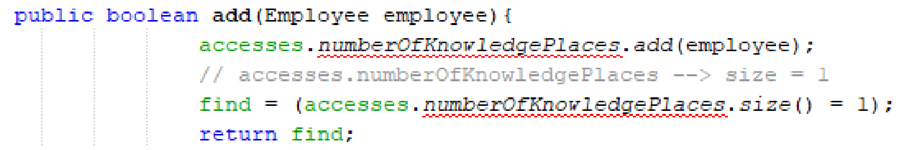


```
public boolean add(Employee employee){
            accesses.numberOfKnowledgePlaces.add(employee);
            // accesses.numberOfKnowledgePlaces --> size = 1
            find = (accesses.numberOfKnowledgePlaces.size() = 1);
            return find;
```

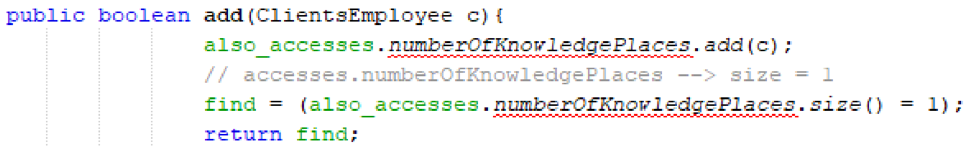


```
public boolean add(ClientsEmployee c){
            also_accesses.numberOfKnowledgePlaces.add(c);
            // accesses.numberOfKnowledgePlaces --> size = 1
            find = (also_accesses.numberOfKnowledgePlaces.size() = 1);
            return find;
```

So the invariant for this limit I put as Java programming code given to the variable find. It is put as Java code by replacing the dashed arrow with the dot notation. The operation size must have parentheses to make the limit work. The limit is equal to one place of knowledge.



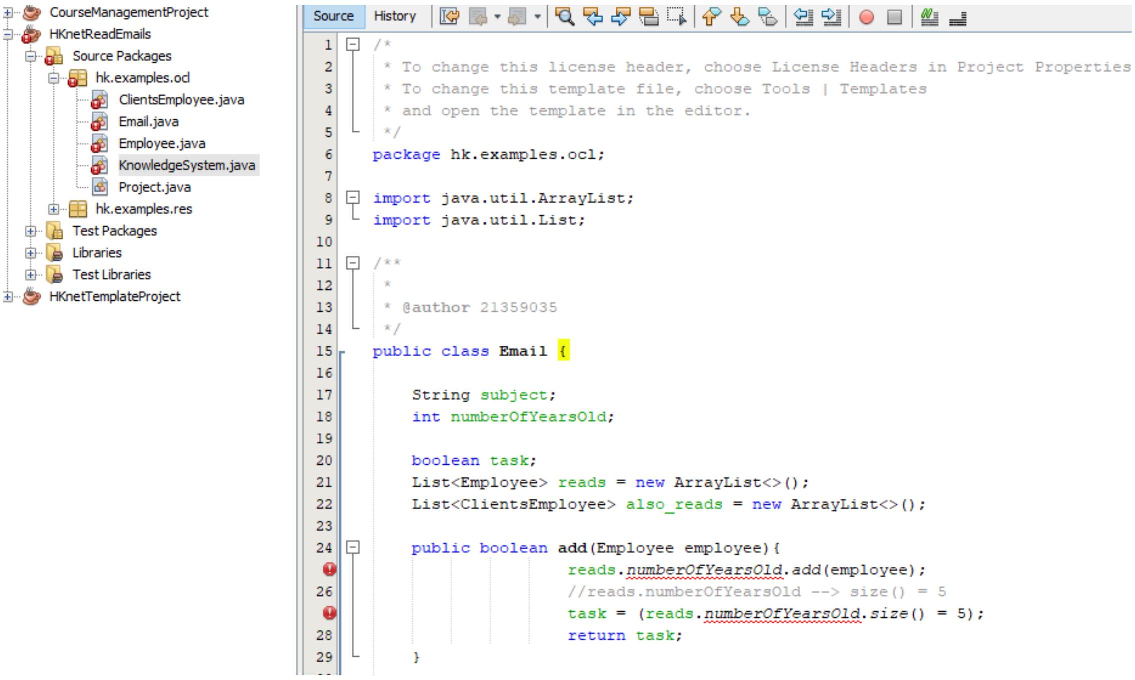
```
public boolean add(Employee employee){
            accesses.numberOfKnowledgePlaces.add(employee);
            // accesses.numberOfKnowledgePlaces --> size = 1
            find = (accesses.numberOfKnowledgePlaces.size() = 1);
            return find;
```



```
public boolean add(ClientsEmployee c){
            also_accesses.numberOfKnowledgePlaces.add(c);
            // accesses.numberOfKnowledgePlaces --> size = 1
            find = (also_accesses.numberOfKnowledgePlaces.size() = 1);
            return find;
```

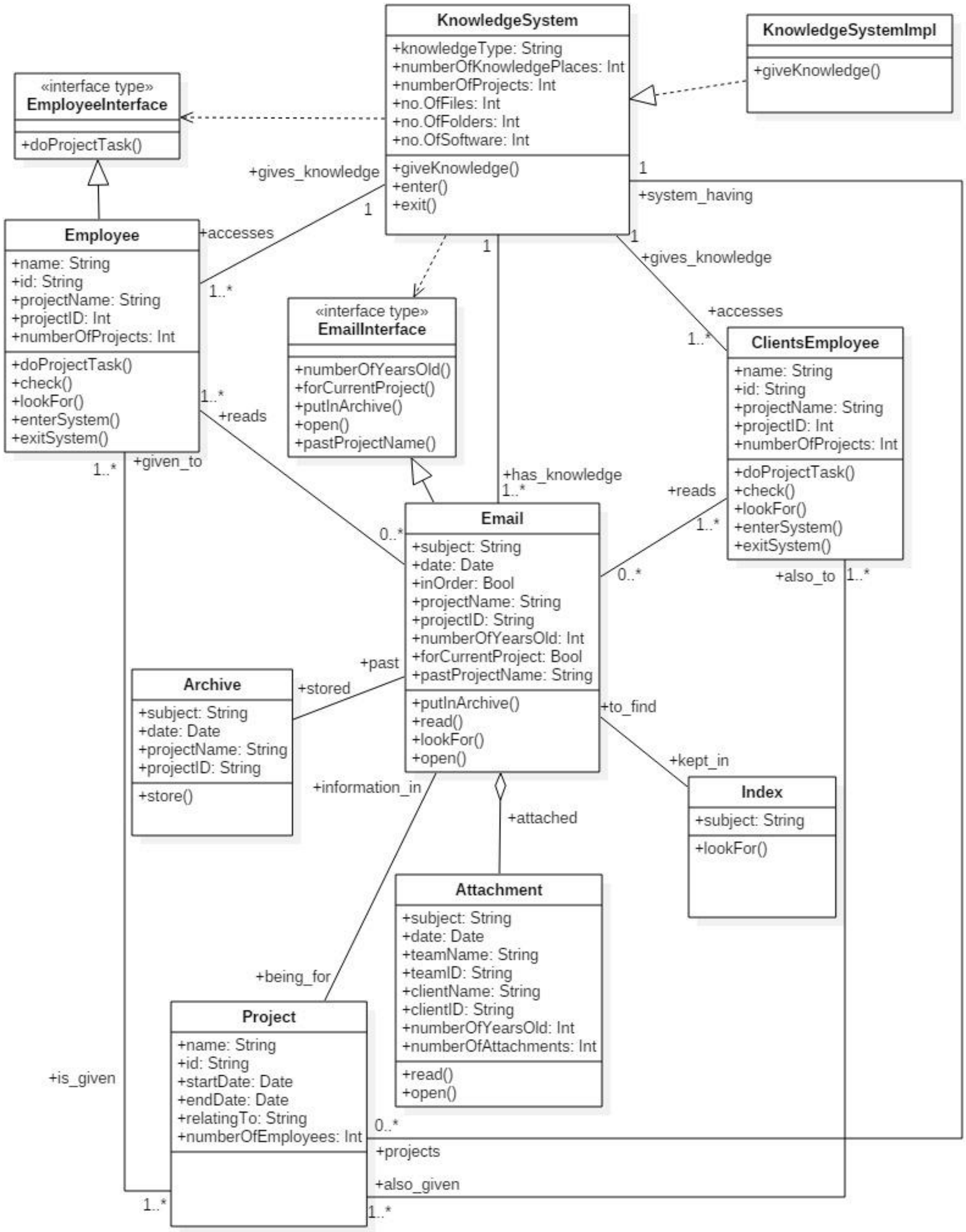
The same errors are here in this method. I can add a less than sign, but the limit is equal to one rather than less than or equal to.

Email

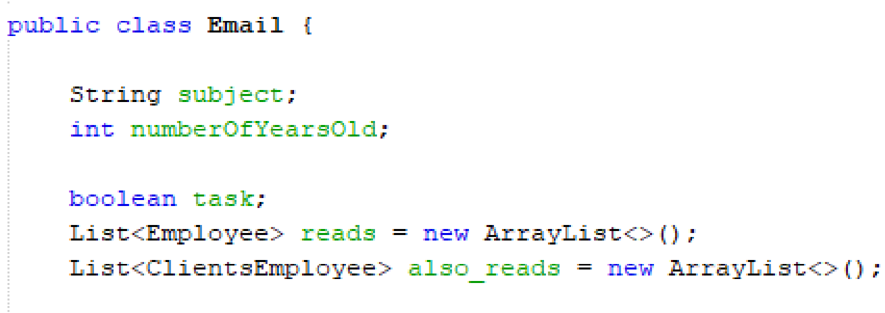


```java
/*
 * To change this license header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hk.examples.ocl;

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

/**
 *
 * @author 21359035
 */
public class Email {

    String subject;
    int numberOfYearsOld;

    boolean task;
    List<Employee> reads = new ArrayList<>();
    List<ClientsEmployee> also_reads = new ArrayList<>();

    public boolean add(Employee employee){
                    reads.numberOfYearsOld.add(employee);
                    //reads.numberOfYearsOld --> size() = 5
                    task = (reads.numberOfYearsOld.size() = 5);
                    return task;
    }
```

In Email I have done the same as I have for all the classes. Its two attributes are listed without being private. These are needed for the limit here which is the number of years old an email is. I declare variable to return data as a Boolean.

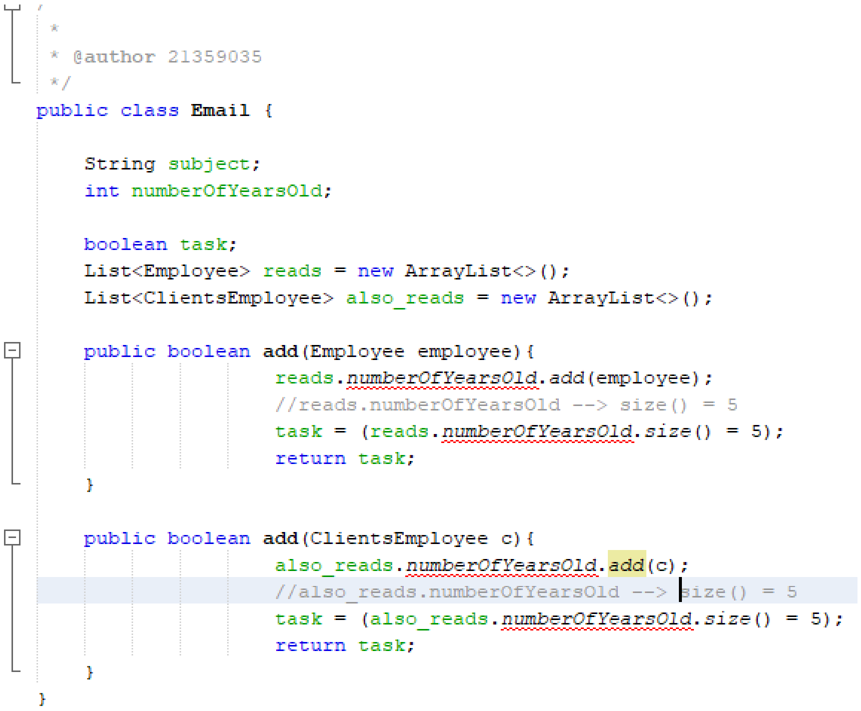


I have also had to edit my class diagram. Edit it including the role names to follow what it says. I do this to put my lines written in Object Constraint Language with the Java programming code to set the limits.



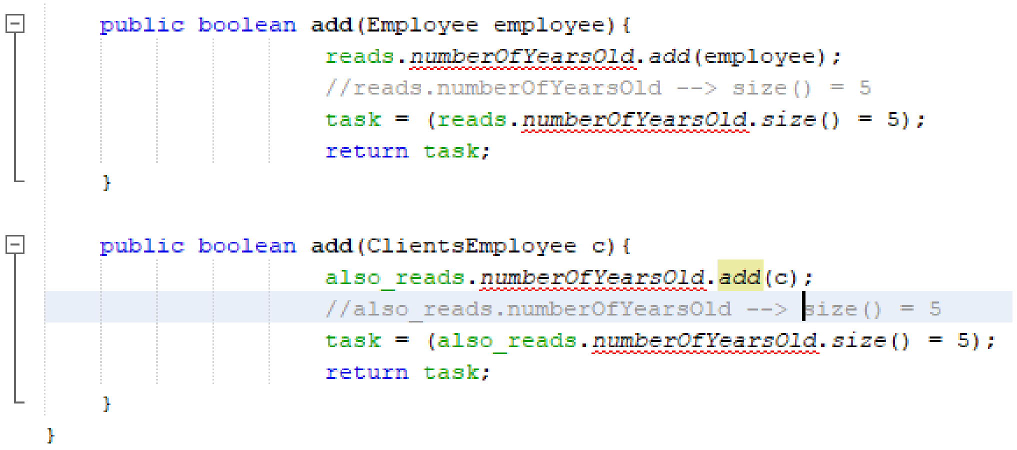
```
public class Email {

    String subject;
    int numberOfYearsOld;

    boolean task;
    List<Employee> reads = new ArrayList<>();
    List<ClientsEmployee> also_reads = new ArrayList<>();
```

I create the same lists. The employee and clients' employee are added to the list to read the emails when given the project to do. So I have my role name reads and also_reads for the OCL and Java programming code.



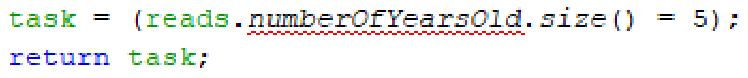
```
/
 *
 * @author 21359035
 */
public class Email {

    String subject;
    int numberOfYearsOld;

    boolean task;
    List<Employee> reads = new ArrayList<>();
    List<ClientsEmployee> also_reads = new ArrayList<>();

    public boolean add(Employee employee){
                    reads.numberOfYearsOld.add(employee);
                    //reads.numberOfYearsOld --> size() = 5
                    task = (reads.numberOfYearsOld.size() = 5);
                    return task;
    }

    public boolean add(ClientsEmployee c){
                    also_reads.numberOfYearsOld.add(c);
                    //also_reads.numberOfYearsOld --> size() = 5
                    task = (also_reads.numberOfYearsOld.size() = 5);
                    return task;
    }
}
```

I have ended up with the same errors here since the code is similar that in all the classes I have done the OCL coding for. It follows the same type of code for the methods. This includes the line for the variable to return a value that is Boolean.

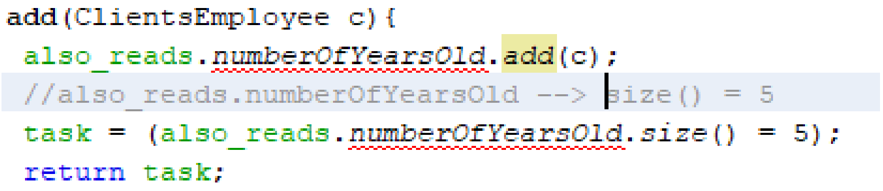


```
    public boolean add(Employee employee){
                    reads.numberOfYearsOld.add(employee);
                    //reads.numberOfYearsOld --> size() = 5
                    task = (reads.numberOfYearsOld.size() = 5);
                    return task;
    }

    public boolean add(ClientsEmployee c){
                    also_reads.numberOfYearsOld.add(c);
                    //also_reads.numberOfYearsOld --> size() = 5
                    task = (also_reads.numberOfYearsOld.size() = 5);
                    return task;
    }
}
```

However, it gives you an idea of the way the coding in Java programming language works with the OCL operations and dot notations in each line. I was short of time to correct the errors.

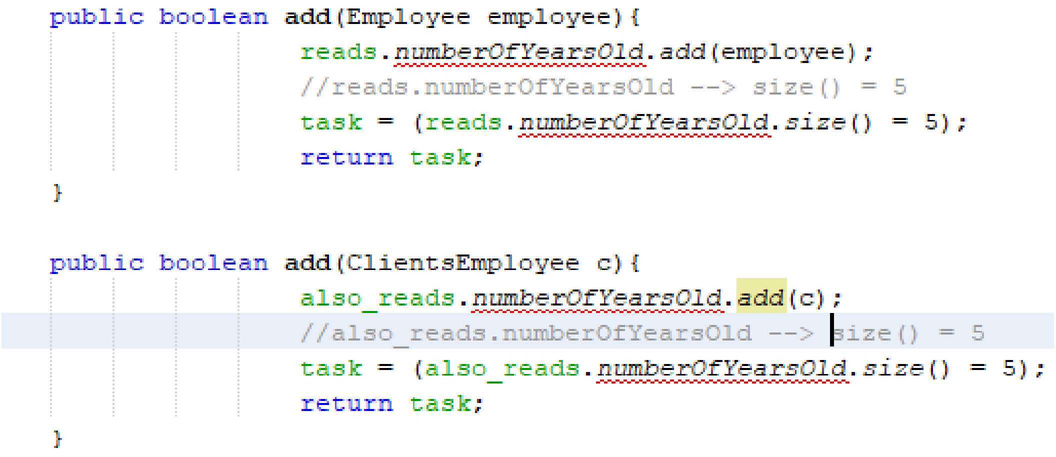


```
task = (reads.numberOfYearsOld.size() = 5);
return task;
```



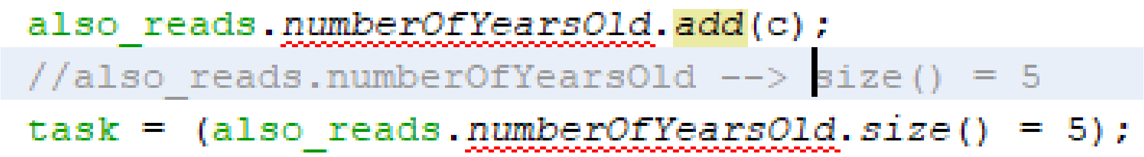
```
add(ClientsEmployee c){
 also_reads.numberOfYearsOld.add(c);
 //also_reads.numberOfYearsOld --> size() = 5
 task = (also_reads.numberOfYearsOld.size() = 5);
 return task;
```

I have put the role name reads and also_reads for both workers with the Email attribute which is the number of years old an email is and the operation size. They are all together with the dot notation to make the limit of years old an email is to be five.



```
public boolean add(Employee employee){
                reads.numberOfYearsOld.add(employee);
                //reads.numberOfYearsOld --> size() = 5
                task = (reads.numberOfYearsOld.size() = 5);
                return task;
}

public boolean add(ClientsEmployee c){
                also_reads.numberOfYearsOld.add(c);
                //also_reads.numberOfYearsOld --> size() = 5
                task = (also_reads.numberOfYearsOld.size() = 5);
                return task;
}
```

I was going to have the OCL operation sum() for the collection of years. I changed it to size as it seems to be a collection that can go up or down. The number of years going up or down rather than it being a value that stays the same. I.e., a fixed value that is a total of a collection of numbers being added.



```
also_reads.numberOfYearsOld.add(c);
//also_reads.numberOfYearsOld --> size() = 5
task = (also_reads.numberOfYearsOld.size() = 5);
```

The limit of an email being as old as five years is made by the role name reads with the number of years old an email being read is put together by the dot notation. It then is joined with the OCL operation by the dot in one line. The method add in the other line to add the class to the list for this limit to work.

The constraints I have come up with in my Read emails use case are clearly given below:

- Each employee can only do only one project at a time

- There must be a maximum of twelve employees working on the same project

- Employees prefer one place to get and share the knowledge

- Emails of past projects must be old as five years