11/23/2016

Software Engineering Assignment

Design note

Student name : Nguyen Manh Tien
CLASS : 3C14 STUDENTID : 1401040206

TABLE OF CONTENTS

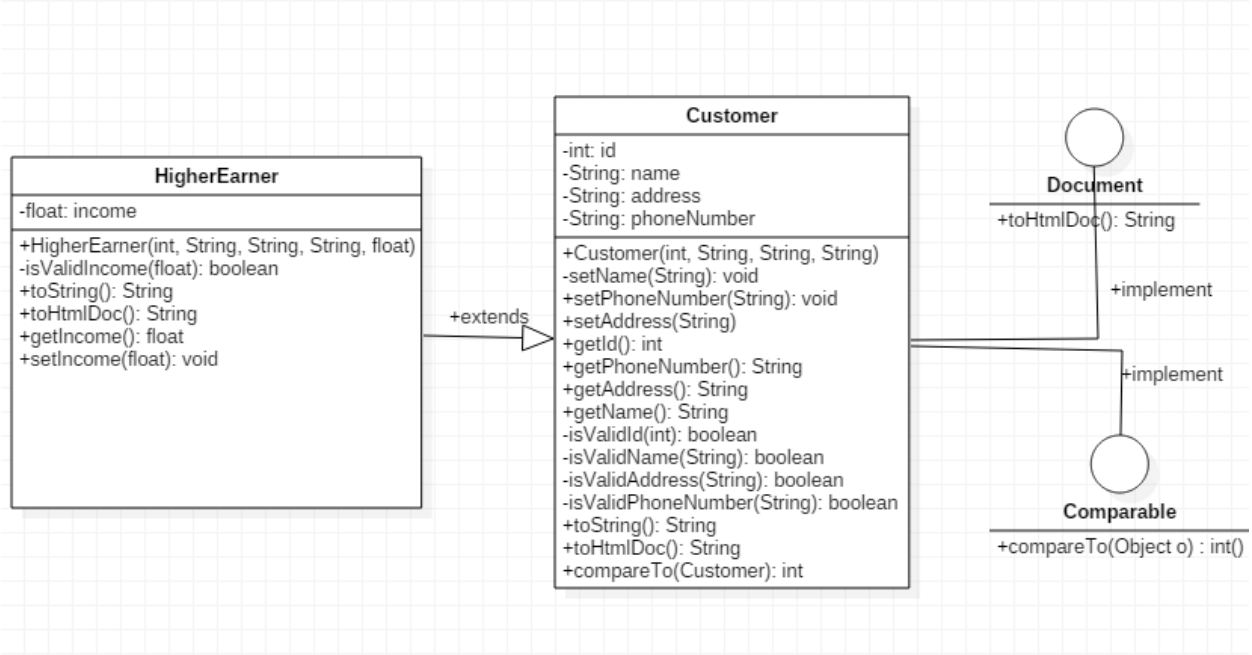
A. DESIGN NOTE

1. The purpose of using the Comparable and Document interface ? Is there another way of designing the same program functionality
(a) without using Comparable?
(b) without using Document?
Briefly explain why or why not under each case?

Answer :

- The purpose of using Comparable is to compare the Customer in SortedSet for storing Customer in SortedSet by order, by using method compareTo().
- The purpose of using Document is support for Kengine, it is necessary to convert Customer object into html for Kengine can execute by keyword searching. The interface contain method toHtmlDoc()which takes no argument and returns a String containing the text of a simple HTML document generated from the state of the current object.
- 2 way of without using Comparable and Document
  + Implement method compareTo() in Customer class
  + implement method toHtmlDoc() in Customer and HigherEarner classes.
- The reason we should not using 2 way above is using interface increases the flexibility of the code. It show reusable of java.

2. Draw a UML design class diagram of the type hierarchy used in the program. What is an advantage of designing types in this hierarchy?



- Designing types in this hierarchy give a general view about project, we can see connection between classes for code more easily. Moreover, design UML reduce time of developementt and easy to follow with larger and more elements program.

3. What is the purpose of using the SortedSet class in the program? Is it possible to develop the same program functionality without using this class? Briefly explain why or why not?

- Purpose of using SortedSet is to store Customers in order, it is same as a list.

- It is possible to develop the same program functionality without using this class.

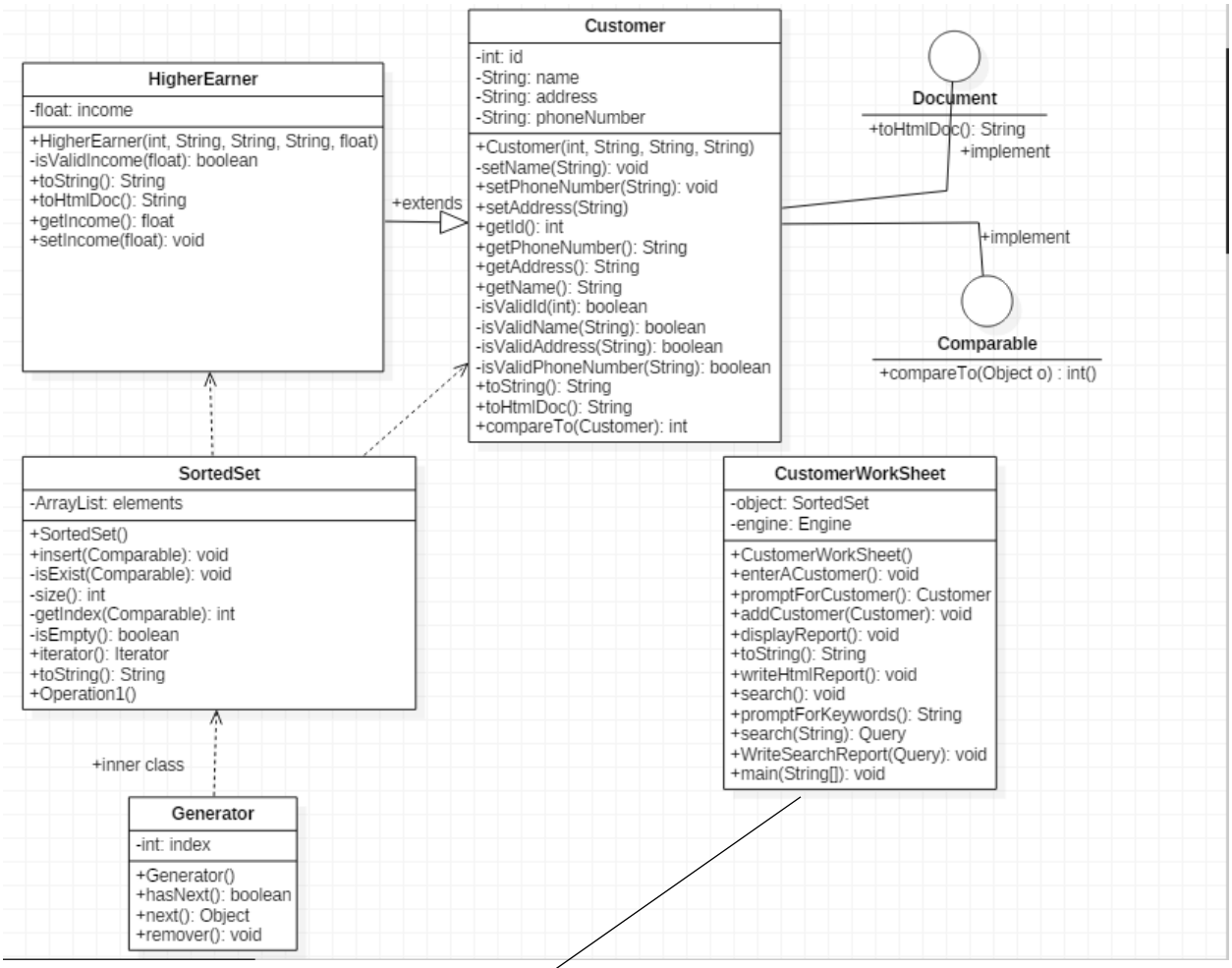
- Because in java we have an interface java.ulti.SortedSet can replace our SortedSet class. The SortedSet interface extends Set and declares the behavior of a set sorted in an ascending order. In addition to those methods defined by Set Several methods throw a NoSuchElementException when no items are contained in the invoking set. A ClassCastException is thrown when an object is incompatible with the elements in a set. A NullPointerException is thrown if an attempt is made to use a null object and null is not allowed in the set. Using this interface can replace SortedSet class.

4. The original KEngine library can only search for text documents using keywords. What makes it possible to use this component in your program to search for Customer objects using keywords?

It is possible to using method addDoc where parameter of Doc type called to search in CustomerWorkSheet class : engine.addDocs(new Doc(c.toHtmlDoc())).

6. **From the design class diagram, identify the implementation strategy that was used to build the application. Briefly discuss this strategy.**

5. **Draw a complete design class diagram of the application showing the classes and their dependencies.**


HigherEarner
-float: income
+HigherEarner(int, String, String, String, float)
-isValidIncome(float): boolean
+toString(): String
+toHtmlDoc(): String
+getIncome(): float
+setIncome(float): void
+extends
Customer
-int: id
-String: name
-String: address
-String: phoneNumber
+Customer(int, String, String, String)
-setName(String): void
+setPhoneNumber(String): void
+setAddress(String)
+getId(): int
+getPhoneNumber(): String
+getAddress(): String
+getName(): String
-isValidId(int): boolean
-isValidName(String): boolean
-isValidAddress(String): boolean
-isValidPhoneNumber(String): boolean
+toString(): String
+toHtmlDoc(): String
+compareTo(Customer): int
Document
+toHtmlDoc(): String
+implement
+implement
Comparable
+compareTo(Object o) : int()
SortedSet
-ArrayList: elements
+SortedSet()
+insert(Comparable): void
-isExist(Comparable): void
-size(): int
-getIndex(Comparable): int
-isEmpty(): boolean
+iterator(): Iterator
+toString(): String
+Operation1()
CustomerWorkSheet
-object: SortedSet
-engine: Engine
+CustomerWorkSheet()
+enterACustomer(): void
+promptForCustomer(): Customer
+addCustomer(Customer): void
+displayReport(): void
+toString(): String
+writeHtmlReport(): void
+search(): void
+promptForKeywords(): String
+search(String): Query
+WriteSearchReport(Query): void
+main(String[]): void
+inner class
Generator
-int: index
+Generator()
+hasNext(): boolean
+next(): Object
+remover(): void
Engine
Engine()
queryfirst(String): Query
queryMore(String): Query
findDoc(String): Doc
addDocs(String): Query
Sorting
quickSort(Vector)
Query
Query()
Query(WordTable, String)
keys(): String[]
size(): int
fetch(int): Doc
addKey(String)
addDoc(Doc)
Comparable
Comm
getDocs(): Iterator
TitleTable
Hashtable docs
TitleTable()
addDoc(Doc)
lookUp(String): Doc
WordTable
Hashtable table
WordTable()
isInteresting(String): boolean
lookUp(String): Vector
addDoc(Doc): Hashtable
Doc
Doc(String)
title(): String
body(): String
words(): Iterator
DocCnt
Doc d
int freqCount
Helpers
canon(String): String

B. SOURCE CODE

- **Packet fsis :**

Customer :

```
package fsis;
import static fsis.TextIO.putln;

/**
 * @author : Nguyen Manh Tien 3c14
 * @Overview : Customer is a person who has information about id, name,
phoneNumber,address.
 * @Attribute :
 * id           int
 * name         String
 * phoneNumber  String
 * address      String
 * @Objects :
 * @Abstract_properties: mutable(id) = false /\ optional(id) = false /\
length(id) = 10 /\ min = 1 /\ max = 9999999/\
 * mutable(name) = true /\ optional(name) = false /\ length(name) = 80 /\
 * mutable(phoneNumber) = true /\ optional(phoneNumber) = false /\ length =
10 /\
 * mutable(address) = true /\ optional(address) = false /\ length = 100.
 */

public class Customer implements Comparable<Customer>, Document {
    @DomainConstraint(type = "int", mutable = false, optional = false, length
= 10, min = 1, max = 9999999)
    protected int id;
    @DomainConstraint(type = "String", mutable = true, optional = false,
length = 80)
    protected String name;
    @DomainConstraint(type = "String", mutable = true, optional = false,
length = 10)
    protected String phoneNumber;
    @DomainConstraint(type = "String", mutable = true, optional = false,
length = 100)
    protected String address;

    /**
     * @effect if name and phone number is valid initialise Customer with
string input.
     */
    public Customer(int id, String name, String phoneNumber, String address)
{
        if (isvalidId(id) && isValidName(name) &&
isValidPhoneNumber(phoneNumber) && isValidAddress(address)) {
            this.id = id;
            this.name = name;
            this.phoneNumber = phoneNumber;
            this.address = address;
        } else {
            putln("invalid input");
```

```
    }

}

/**
 * @effect if name is valid
 * this.name = name
 * else
 * print error
 */
public void setName(String name) {
    if (isValidName(name)) {
        this.name = name;
    } else {
        putln("invalid name!");
    }
}

/**
 * @effect if phone number is valid
 * this.phoneNumber = phoneNumber
 * else
 * print error
 */
public void setPhoneNumber(String phoneNumber) {
    if (isValidPhoneNumber(phoneNumber)) {
        this.phoneNumber = phoneNumber;
    } else {
        putln("invalid phone number!");
    }
}

/**
 * @effect this.address = address
 */
public void setAddress(String address) {
    this.address = address;
}

/**
 * @effect return id
 */
public int getId() {
    return id;
}

/**
 * @effect return name
 */
public String getName() {
    return name;
}

/**
 * @effect return phoneNumber
 */
public String getPhoneNumber() {
```

```
        return phoneNumber;
    }

    /**
     * @effect return address
     */
    public String getAddress() {
        return address;
    }

    /**
     * @effect if 0 < id < 9999999
     * return true
     * else
     * return false
     */
    private boolean isvalidId(int id) {
        if (id > 0 && id < 9999999) {
            return true;
        }
        return false;
    }

    /**
     * @effect if name id valid
     * return true
     * else
     * return fasle
     */
    private boolean isValidName(String name) {
        if (name != null && !name.isEmpty() && name.length() <= 50) {
            return true;
        }
        return false;
    }

    /**
     * @effect if phone number is valid
     * return true
     * else
     * return false
     */
    private boolean isValidPhoneNumber(String phoneNumber) {
        if (phoneNumber != null && !phoneNumber.isEmpty() &&
phoneNumber.length() <= 10) {
            return true;
        }
        return false;
    }

    /**
     * if address is valid
     * return true
     * else
     * return false
     */
    private boolean isValidAddress(String address) {
```

```
        if (address != null && !address.isEmpty() && address.length() <= 100)
{
            return true;
        }
        return false;
    }

    /**
     * @effect return string from constructor
     */
    @Override
    public String toString() {
        return "Customer : " +
                "id = " + id +
                ", name = " + name +
                ", phoneNumber = " + phoneNumber +
                ", address = " + address;
    }

    @Override
    public String toHTMLDoc() {
        return " <html> " +
                "<head><title>Customer:" + this.name + "</title></head> " +
                "<body> " +
                this.id + " " + this.name + " " + this.phoneNumber + " " +
this.address +
                "</body></html> ";
    }

    @Override
    public int compareTo(Customer o) {
        return this.name.compareTo(o.name);
    }
}
```

CustomerWorkSheet:

```
package fsis;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.Iterator;

import static fsis.TextIO.*;

import kengine.*;

/**
 * @author dmle
 * @overview Represents a worksheet program that enables a user to create
customer objects,
 * display a report about them, and to search for objects of interest using
keywords.
 * @attributes objects    SortedSet<Customer>
```

```
 * engine     Engine
 * @abstract_properties mutable(objects)=true /\ optional(objects)=false /\
 * mutable(engine)=false /\ optional(engine)=false /\
 * size(objects) > 0 ->
 * (for all o in objects: o.toHtmlDoc() is in engine)
 */
public class CustomerWorkSheet {
    @DomainConstraint(type = "Collection", mutable = true, optional = false)
    private SortedSet objects;
    @DomainConstraint(type = "Engine", mutable = false, optional = false)
    private Engine engine;

    /**
     * @effects initialise this to include an empty set of objects and an
empty engine
     */
    public CustomerWorkSheet() {
        objects = new SortedSet();
        engine = new Engine();
    }

    /**
     * @effects invoke promptForCustomer to prompt the user to enter details
of
     * a customer, create a Customer object from these details and
     * invoke addCustomer to add the object to this.
     * <p>
     * If invalid details were entered then throws NotPossibleException.
     */
    public void enterACustomer() throws NotPossibleException {
        putln("enter a customer ");
        Customer customer = null;
        customer = this.promptForCustomer();
        if (customer == null) {
            throw new NotPossibleException(
                    " Invalid Customer to insert.");
        }else {
            this.addCustomer(customer);
            putln("Added: " + customer.toString());
        }

    }


    /**
     * @effects prompt the user whether to enter details for Customer or
HighEarner,
     * then prompt the user for the data values needed to create an object
     * for the selected type.
     * <p>
     * Create and return a Customer object from the entered data. Throws
     * NotPosibleException if invalid data values were entered.
     */
    public Customer promptForCustomer() throws NotPossibleException {
        try {
            int choice;
            do {
```

```
                putln("choose an option : ");
                putln("1. Input Customer information. ");
                putln("2. Input Higher Earner information.");
                choice = getlnInt();
            } while (choice != 1 && choice != 2);
            put("enter  id : ");
            int id = getlnInt();
            put("enter  name : ");
            String name = getln();
            put("enter phone number :");
            String phoneNumber = getln();
            put("enter address : ");
            String address = getln();
            if (choice == 1) {
                return new Customer(id, name, phoneNumber, address);
            } else {
                put("enter income :");
                Float income = TextIO.getlnFloat();
                return new HighEarner(id, name, phoneNumber, address,
income);
            }
        } catch (NotPossibleException ex) {
            putln(ex.getMessage());
            return null;
        }
    }

    /**
     * @effects add c to this.objects and
     * add to this.engine a Doc object created from c.toHtmlDoc
     */
    public void addCustomer(Customer customer) {
        this.objects.insert(customer);
        this.engine.addDoc(new Doc(customer.toHTMLDoc()));

    }

    /**
     * @modifies System.out
     * @effects if this.objects == null
     * prints "empty"
     * else
     * prints each object in this.objects one per line to the standard output
     * invoke writeHTMLReport to write an HTML report to file
     */
    public void displayReport() {
        if (this.objects == null) {
            putln("empty");
        } else {
            Iterator<Customer> iterator = this.objects.iterator();
            while (iterator.hasNext()) {
                Customer customer = iterator.next();
                putln(customer.toString());
            }
            this.writeHTMLReport();
            putln("Report written to file objects.html");
        }
```

```
    }

    /**
     * @effects if objects is empty
     * return "empty"
     * else
     * return a string containing each object in this.objects one per line
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        if (this.objects.isEmpty()) {
            return "empty";
        }
        Iterator<Customer> iterator = this.objects.iterator();
        while (iterator.hasNext()) {
            Customer customer = iterator.next();
            sb.append(customer.toString());
            sb.append("\n");
        }
        return sb.toString();
    }

    /**
     * @modifies objects.html (in the program directory)
     * @effects if this.objects is empty
     * write an HTML document to file with the word "empty" in the body
     * else
     * write an HTML document to file containing a table, each row of which
     * lists an object in this.objects
     * <p>
     * The HTML document must be titled "Customer report".
     */
    public void writeHTMLReport() {
        try (BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter("object.html"))) {
            if (this.objects.isEmpty()) {
                bufferedWriter.write("empty");
            } else {
                StringBuilder sb = new StringBuilder();
                sb.append("<html>");
                sb.append("<head><title>Customer report</title></head>");
                sb.append("<body>");
                sb.append("<table border=1>");
                sb.append("<tr><th>Id</th><th>Name</th><th>Phone
number</th><th>Address</th><th>Income</th></tr>");
                Iterator<Customer> iterator = this.objects.iterator();
                while (iterator.hasNext()) {
                    Customer customer = iterator.next();
                    sb.append("<tr>");
                    sb.append("<td>" + customer.getId() + "</td>");
                    sb.append("<td>" + customer.getName() + "</td>");
                    sb.append("<td>" + customer.getPhoneNumber() + "</td>");
                    sb.append("<td>" + customer.getAddress() + "</td>");
                    if (customer instanceof HighEarner) {
                        HighEarner highEarner = (HighEarner) customer;
                        sb.append("<td>" + highEarner.getIncome() + "</td>");
```

```
                    }
                    sb.append("</tr>");
                }
                sb.append("</table>");
                sb.append("</body></html>");
                bufferedWriter.write(sb.toString());
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    /**
     * @modifies System.out
     * @effects prompt the user to enter one or more keywords
     * if keywords != null AND keywords.length > 0
     * invoke operation search(String[]) to search using keywords,
     * <p>
     * if fails to execute the query
     * throws NotPossibleException
     * else
     * print the query string to the standard output.
     * invoke operation writeSearchReport(Query) to output the query to an
HTML file
     * else
     * print "no search keywords"
     */
    public void search() throws NotPossibleException {
        String[] keywords = promptForKeywords();
        if (keywords != null && keywords.length > 0) {
            try {
                putln("Searching for customers using keywords :"
+keywords.toString());
                Query query = search(keywords);
                putln(query.toString());
                writeSearchReport(query);
            } catch (NotPossibleException e) {
                throw new NotPossibleException("Fail to execute query");
            }
        } else {
            putln("no search keyword");
        }
    }

    /**
     * @effects prompt the user to enter some keywords and
     * return an array containing these or null if no keywords were entered
     */
    public String[] promptForKeywords() {
        putln("enter some keyword(separatered by space) : ");
        String keywords = getln();
        if (keywords != null || keywords.length() > 0) {
            return keywords.split(" ");
        }
        return null;
    }
```

```
    /**
     * @requires words != null /\ words.length > 0
     * @effects search for objects whose HTML documents match with the query
containing words
     * and return a Query object containing the result
     * <p>
     * If fails to execute query using words
     * throws NotPossibleException
     */
    public Query search(String[] words) throws NotPossibleException {
        Query query = null;
        try {
            query = this.engine.queryFirst(words[0]);
            for (int i = 1; i < words.length; i++) {
                query = this.engine.queryMore(words[i]);
            }
        } catch (NotPossibleException ex) {
            ex.printStackTrace();
        }
        return query;
    }

    /**
     * @requires query != null
     * @modifies search.html (in the program directory)
     * @effects write to file an HTML document containing the query keys and
a table,
     * each row of which lists a match
     * <p>
     * The HTML document must be titled "Search report".
     */
    public void writeSearchReport(Query query) {
        putln("Writing query report(sorted in descending order) to file
search.html...");
        try (BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter("search.html"))) {
            Iterator iterator = query.matchIterator();
            bufferedWriter.write("<html><title> Search Report
</title><body>Query: ");
            bufferedWriter.write(Arrays.toString(query.keys()) + "<br>
Results: " + "<br><table border = 1 > <th> Documents </th> <th>Sum
freqs</th>");
            while (iterator.hasNext()) {
                DocCnt docCnt = (DocCnt) iterator.next();
                Doc doc = docCnt.getDoc();
                bufferedWriter.write("<tr><td>" + doc.title() + "</td><td>" +
docCnt.getCount() + "</td></tr>");
            }
            bufferedWriter.write("</table> </body> </html>");
            putln("Report writed to file search.html ");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    /**
     * @effects initialise a CustomerWorkSheet
```

```
     * ask the users to create the five Customer objects
     * display report about the objects
     * ask the users to enter a keyword query to search for objects and
display
     * the result
     */
    public static void main(String[] args) {
        // initialise a CustomerWorkSheet
        putln("Initialising program...");
        CustomerWorkSheet worksheet = new CustomerWorkSheet();

        try {
            // ask user to create 5 test customer objects
            putln("\nCreating some customers...");
            int num = 5;
            for (int i = 0; i < num; i++) {
                putln("*----------------------------------------*");
                worksheet.enterACustomer();
            }
                putln("******************************************");

            // display report about the objects
            worksheet.displayReport();

            putln("******************************************");

            // ask the users to enter a keyword query to search for objects
and display
            // the result
            worksheet.search();
            putln("******************************************");

            // end

            putln("Good bye.");
        } catch (NotPossibleException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Document :

```
package fsis;
public interface Document {
    String toHTMLDoc();
}
```

HignerEarner :

```
package fsis;

import kengine.NotPossibleException;

/**
 * @author : Nguyen Manh Tien 3c14
 * @Overview : HigherEarner is a customer who has high income.
 * @Attribute :
 * id           int
 * name         String
 * phoneNumber  String
 * address      String
 * income       float
 * @Objects :
 * @Abstract_properties: mutable(id) = false /\ optional(id) = false /\
length(id) = 10 /\ min = 1 /\ max = 9999999/\
 * mutable(name) = true /\ optional(name) = false /\ length(name) = 80 /\
 * mutable(phoneNumber) = true /\ optional(phoneNumber) = false /\ length =
10 /\
 * mutable(address) = true /\ optional(address) = false /\ length = 100 /\
 * mutable(income) = true /\ optional = false /\ min = 10000000.
 */
public class HighEarner extends Customer {
    @DomainConstraint(type = "float", mutable = true, optional = false, min =
10000000)
    private float income;

    public HighEarner(int id, String name, String phoneNumber, String
address, float income) {
        super(id, name, phoneNumber, address);
        if (isValidIncome(income)) {
            this.income = income;
        }
        else {
            throw new NotPossibleException("HighEarner: invalid HighEarner");
        }
    }

    /**
     * @return this.income
     */
    public float getIncome() {
        return income;
    }


    /**
     * if income is valid
     * this.income = income
     * else
     * print err message
     */
    public void setIncome(float income) {
        if (isValidIncome(income)) {
```

```
            this.income = income;
        } else {
            System.err.println("invalid income !");
        }

    }

    /**
     * if income > 10000000
     * return true
     * else
     * return false.
     */
    private boolean isValidIncome(float income) {
        if (income > 10000000) {
            return true;
        }
        return false;
    }


    @Override
    /**
     * return a string contain infor of a HigherEarner
     */
    public String toString() {
        return "HigherEarner : " +
                "id = " + id +
                ", name = " + name +
                ", phoneNumber = " + phoneNumber +
                ", address = " + address +
                ", income = "+(int)income;
    }

    @Override
    public String toHTMLDoc() {
        return  " <html> " +
                "<head><title>Higher Earner : "+this.name+"</title></head> "
+
                "<body> " +
                this.id +" "+this.name+" "+this.phoneNumber+"
"+this.address+" "+(int)this.income+
                "</body></html> ";
    }
}
```

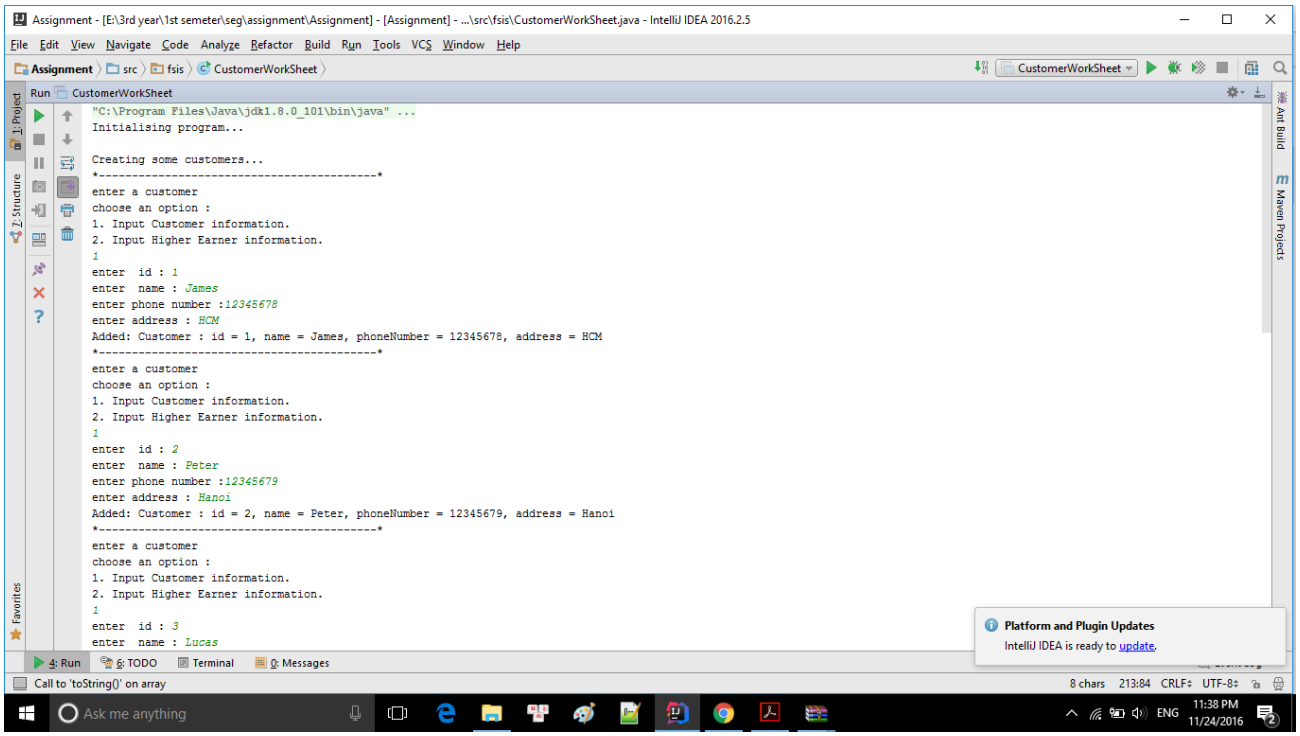
SortedSet :

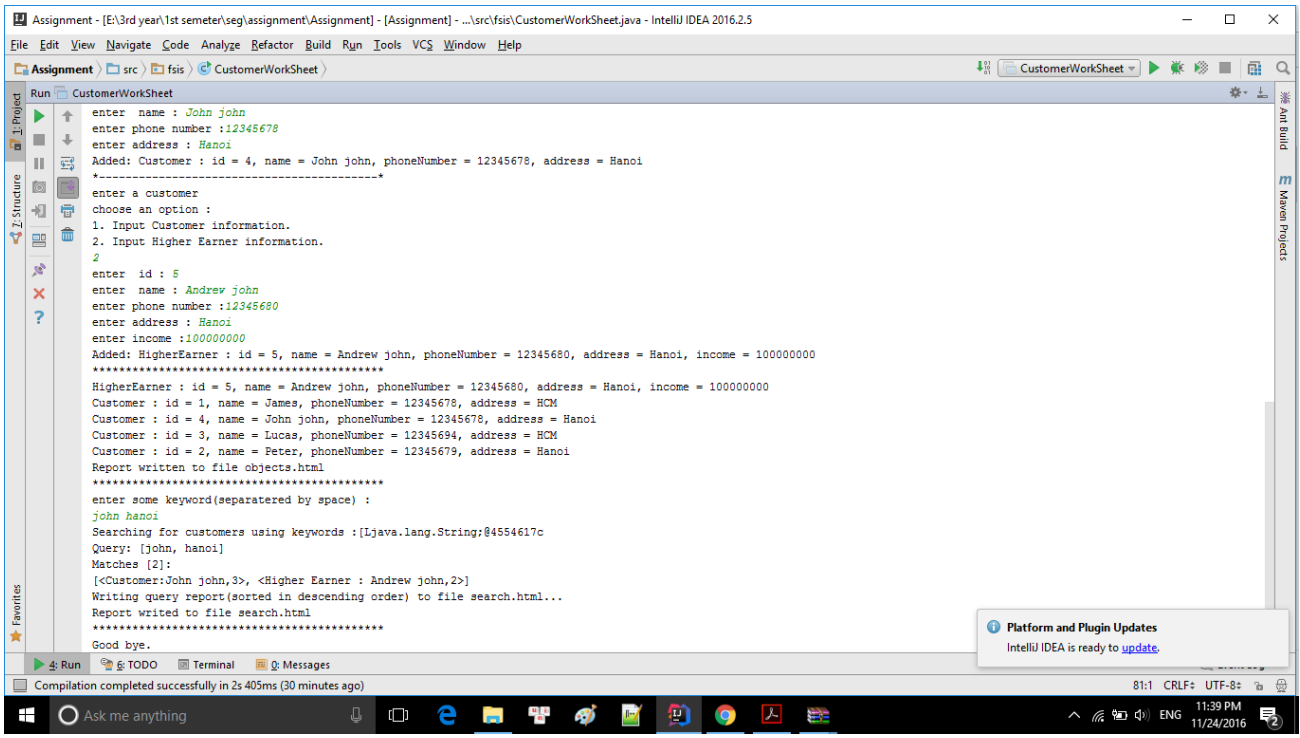
```
package fsis;

import java.util.*;


/**
 * @overview Sorted are mutable, unbounded sets of customer.
 * @attributes elements  ArrayList<Customer>
 * @object A typical Sorted object is c={c1,...,cn}, where c1,...,cn are
 * elements.
 * @abstract_properties optional(elements) = false /\ for all x in elements.
x
 * is integer /\ for all x, y in elements. x neq y
 */
public class SortedSet {
    @DomainConstraint(type = "ArrayList", optional = false)
    private ArrayList<Comparable> elements;

    /**
     * @effects initialise object SortedSet with elements is an Arraylist
     */
    public SortedSet() {
        elements = new ArrayList<Comparable>();
    }

    /**
     * @effects if elements is empty
     * add customer in this.elements.
     * else
     * add customer in this.elements and sort elements.
     */
    public void insert(Comparable customer) {
        if (elements.isEmpty()) {
            elements.add(customer);
        }
        else {
            Customer customerInput = (Customer) customer;
            Iterator iterator = iterator();
            while (iterator.hasNext()) {
                Customer custom = (Customer) iterator.next();
                if (custom.compareTo(customerInput) > 0 && isExist(customer)
== false) {
                    elements.add(getIndex(custom), customer);
                    customer = custom;
                }
            }

            if (isExist(customer) == false) {
                elements.add(customer);
            }
        }
    }
```

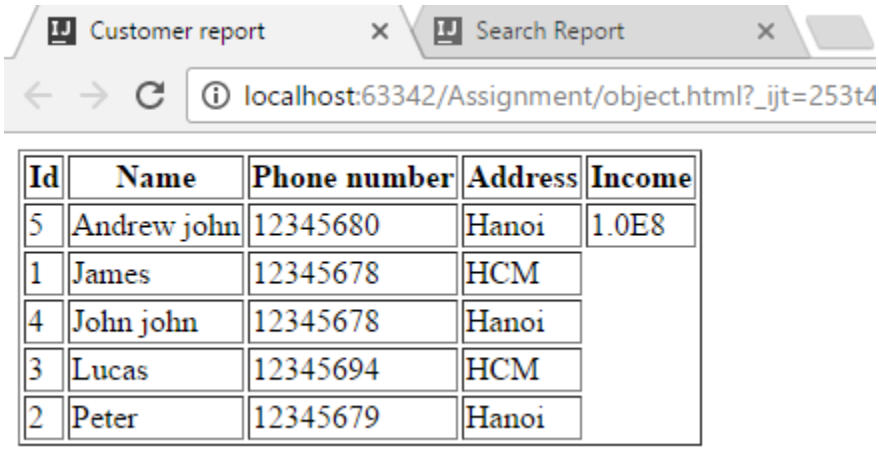
```java
/**
 * @effects if x is in this
 * return true
 * else
 * return false
 */
private boolean isExist(Comparable customer) {
    return (getIndex(customer) >= 0);
}

/**
 * @effects return the size of this
 */
private int size() {
    return elements.size();
}

/**
 * @effects if x is in this
 * return the index where x appears
 * else
 * return -1
 */
private int getIndex(Comparable customer) {
    for (int i = 0; i < elements.size(); i++) {
        if (customer.compareTo(elements.get(i)) == 0)
            return i;
    }
    return -1;
}

/**
 * if element is empty
 * return true
 * else
 * return false
 */
public boolean isEmpty() {
    return elements.isEmpty();
}

/**
 * @return a Iterator
 */
public Iterator iterator() {
    return new Generator();
}

@Override
public String toString() {
    if (size() == 0)
        return "List Cutomers:{ }";
    String s = "List Customers:{" + elements.get(0).toString();
    for (int i = 1; i < size(); i++) {
        s = s + " , " + elements.get(i).toString();
    }
```

```
        return s + "}";
    }

    private class Generator implements Iterator {
        private int index;

        public Generator() {
            index = -1;
        }

        public boolean hasNext() {
            return (index < size() - 1);
        }

        public Object next() throws NoSuchElementException {
            if (index < size() - 1) {
                index++;
                return elements.get(index);
            }
            throw new NoSuchElementException("No more elements");
        }

        public void remove() {
            // do nothing
        }
    }
}
```

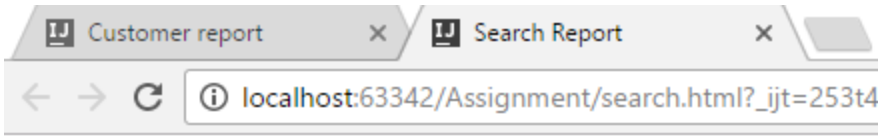
C. Demo :


Assignment - [E:\3rd year\1st semeter\seg\assignment\Assignment] - [Assignment] - ...\src\fsis\CustomerWorkSheet.java - IntelliJ IDEA 2016.2.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run CustomerWorkSheet
"C:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Initialising program...
Creating some customers...
---
enter a customer
choose an option :
1. Input Customer information.
2. Input Higher Earner information.
1
enter id : 1
enter name : James
enter phone number :12345678
enter address : HCM
Added: Customer : id = 1, name = James, phoneNumber = 12345678, address = HCM
---
enter a customer
choose an option :
1. Input Customer information.
2. Input Higher Earner information.
1
enter id : 2
enter name : Peter
enter phone number :12345679
enter address : Hanoi
Added: Customer : id = 2, name = Peter, phoneNumber = 12345679, address = Hanoi
---
enter a customer
choose an option :
1. Input Customer information.
2. Input Higher Earner information.
1
enter id : 3
enter name : Lucas
Platform and Plugin Updates
IntelliJ IDEA is ready to update.
4: Run 6: TODO Terminal 0: Messages
Call to 'toString()' on array
8 chars 213:84 CRLF UTF-8
Ask me anything
ENG 11:38 PM 11/24/2016


Assignment - [E:\3rd year\1st semeter\seg\assignment\Assignment] - [Assignment] - ...\src\fsis\CustomerWorkSheet.java - IntelliJ IDEA 2016.2.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run CustomerWorkSheet
enter name : John john
enter phone number :12345678
enter address : Hanoi
Added: Customer : id = 4, name = John john, phoneNumber = 12345678, address = Hanoi
---
enter a customer
choose an option :
1. Input Customer information.
2. Input Higher Earner information.
2
enter id : 5
enter name : Andrew john
enter phone number :12345680
enter address : Hanoi
enter income :100000000
Added: HigherEarner : id = 5, name = Andrew john, phoneNumber = 12345680, address = Hanoi, income = 100000000
****
HigherEarner : id = 5, name = Andrew john, phoneNumber = 12345680, address = Hanoi, income = 100000000
Customer : id = 1, name = James, phoneNumber = 12345678, address = HCM
Customer : id = 4, name = John john, phoneNumber = 12345678, address = Hanoi
Customer : id = 3, name = Lucas, phoneNumber = 12345694, address = HCM
Customer : id = 2, name = Peter, phoneNumber = 12345679, address = Hanoi
Report written to file objects.html
****
enter some keyword(separatered by space) :
john hanoi
Searching for customers using keywords :[Ljava.lang.String;@4554617c
Query: [john, hanoi]
Matches [2]:
[<Customer:John john,3>, <Higher Earner : Andrew john,2>]
Writing query report(sorted in descending order) to file search.html...
Report writed to file search.html
****
Good bye.
Platform and Plugin Updates
IntelliJ IDEA is ready to update.
4: Run 6: TODO Terminal 0: Messages
Compilation completed successfully in 2s 405ms (30 minutes ago)
81:1 CRLF UTF-8
Ask me anything
ENG 11:39 PM 11/24/2016



Id	Name	Phone number	Address	Income
5	Andrew john	12345680	Hanoi	1.0E8
1	James	12345678	HCM	
4	John john	12345678	Hanoi	
3	Lucas	12345694	HCM	
2	Peter	12345679	Hanoi	



Query: [john, hanoi]
Results:

Documents	Sum freqs
Customer:John john	3
Higher Earner : Andrew john	2