CS390 Fundamental Programming Practices
Final Examination

Date : 12.17.2020 **Max.Marks : 40** **Time : 2.00 Hours**

Student ID _____ Name _____

Part – 1 (10)	Prog – 2 (10)	Prog – 3 (10)	Prog- 4 (10)	Total (40)

Problem : 1 ArrayList and Comparator[API] **[10 Points]**

You have fully implemented Staff and Teacher classes which implements EmployeeData Interface. There is EmployeeSort.java file which contains an ArryList collections of EmployeeData and one unimplemented method with the given highlighted signature and you must implement to sort the list using Comparator interface. The implementation space and problem requirement mentioned in the next page. First look at the given implemented codes.

public static void empSort(List<EmloyeeData> list, String sort_type);

Note: Comparator Implementation may be Inner class or Anonymous or Separate class or Lambdas. Choice is yours.

//Implemented class and Interface

```
public interface EmployeeData {
        double getSalary();
        String getName();
}
public class Staff implements EmployeeData {
        public String name;
        public double salary;
    public Staff(String name, double salary) {
        this.name = name;
        this.salary = salary;
        }
        @Override
     public String getName() {
                return name;
        }
        @Override
        public double getSalary() {
                return salary;
        }
```

```
public class Teacher implements EmployeeData{
        private String name;
        private double salary;
        private double bonus;
   public Teacher(String name, double salary,
double bonus) {
                this.name = name;
                this.salary = salary;
                this.bonus = bonus;
        }
        @Override
        public String getName() {
                return name;
        }
        @Override
        public double getSalary() {
                this.salary +=this.bonus;
                return salary;
        }
```

```
        @Override
        public String toString() {
                return "Staff [" + name + ", " +
salary + "]";
        }
}
```

```
        @Override
        public String toString() {
                return "Teacher [" + name + ", " +
salary + "]";
        }
}
```

```
// Test Class
public class EmployeeSort {
        public static void main(String[] args) {
                List<Teacher> list1 = new ArrayList<>();
                list1.add(new Teacher("David",90000,400));
                list1.add(new Teacher("Abigail",85000,600));
                list1.add(new Teacher("Michael",95000,700));
                list1.add(new Teacher("Glen",75000,900));

                List<Staff> list2 = new ArrayList<>();
                list2.add(new Staff("Saly",78000));
                list2.add(new Staff("Becky",89000));
                list2.add(new Staff("Alan",95000));

                List<EmployeeData> employees = new ArrayList<>();
                employees.addAll(list1);
                employees.addAll(list2);
                // calling empSort() to sort and print the list by Employee name
                empSort(employees,"BYNAME");
                // calling empSort() to sort and print the list by Employee salary
                empSort(employees,"BYSALARY");
        }
}
```

Hint : Syntax of Comparator	**Hint : Syntax of sort from Collections**

public interface Comparator<T> {
int compare(T o1, T o2);
}
	Collections.sort(List<T> list, Comparator<T> ob);

/*Task: Write your implementation to sort the EmployeeData list based on the input of user choice as String Sort type.

If your sort_type value is BYNAME, then sort and print the list by using name field.

If your sort_type value is BYSALARY, then sort and print the list by using salary field.

Your implementation should not be consist with equals.*/

```
public static void empSort(List<EmployeeData> list, String sort_type){
        // Write your implementation here
```

}

Problem -2 – Linked List **[10 Points]**

You have given DoublyLinkedList class with a header instance field and a constructor. You have one implemented addFirst() method. Look at the given code and write the two unimplemented methods.

a. void addLast(String s);
b. String deleteFirst();

Note: No data should be placed on header. Either header.next will be null or points the first node.

```
public class DoublyLinkedList {
        Node header;
        DoublyLinkedList(){
                header = new Node(null, null, null);
        }
class Node {
                Node previous;
                String value;
                Node next;
                Node(Node previous, String value, Node next){
                        this.previous = previous;
                        this.value = value;
                        this.next = next;
                }}
public void addFirst(String item){
        Node n = new Node(header,item,header.next);
          if(header.next != null){
             header.next.previous = n;
        }
          header.next = n; }
public String deleteFirst(){  // [ 4 Points]
/* You should not remove the header. Delete the first node which connects with the header.
   If the list has no elements throw NoSuchElementException. Return the deleted Node value.*/
```

```
}
```

public void addLast(String item){// [6 Points]

/* Add an element in the end of the list. Your logic must work if the list has no elements, one element or more elements. Do the necessary null check to get rid of NullPointerException*/

}

Problem-3 [User Implementation of Stack using Array] [10 Points]

Implement the stack behaviors such as push, pop, peek and isempty.

```
class ArrayStack {
        private Object a[];
        private int top;

       public ArrayStack(int n) {
                a = new Object[n];
                top = -1;
        }

        /*Insert an item into the Stack if stack gets full make use of given resize(). Do the
          necessary null check to avoid NullPointerException */
        public void push(Object item) {  // [ 3 Points ]
```

}

// Remove the element from the Stack and return the deleted item. If the queue has no elements throw EmptyStackException()

public Object pop() { // [3 Points]

}

/* Get the peek element from the Stack.
If the queue has no elements throw EmptyStackException()*/

public Object peek() { // [2.5 Points]

}

// Write your logic to check the stack is empty or not
public boolean isEmpty() { // [1.5 Points]

```
}
  private void resize(){
                System.out.println("resizing");
                int len = a.length;
                int newlen = 2*len;
                Object[] temp = new Object[newlen];
                a = Arrays.copyOf(a, newlen);
        }
        public String toString(){
                StringBuilder sb = new StringBuilder("[");
                for(int i = 0; i <top; ++i){
                        sb.append(a[i]+", ");
                }
                sb.append(a[top]+"]");
                return sb.toString();
        }}
```

Problem – 4 – Hash Table / Hash Map **[10 Points]**

You have fully implemented Employee class given below.

```
public class Employee {
        private String ssn;
        private String name;
        private double salary;

        public Employee(String ssn, String name, double aSalary) {
                this.name = name;
                this.ssn = ssn;
                this.salary = aSalary;
        }
        public String getName() {
                return name;
        }
        public String getSsn() {
                return ssn;
        }
        public double getSalary() {
                return salary;
        }
        @Override
        public String toString() {
                return "Employee [ssn=" + ssn + ", name=" + name + ", salary=" + salary + "]";
```

```
    }
}
```

Test Class given below.

```
public class TestEmployee {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee("234-23-4455", "Joe", 3450));
        list.add(new Employee("221-45-9990", "Mike",  5500));
        list.add(new Employee("876-99-7654", "Chelle",  8000));
        list.add(new Employee("564-66-6767", "Tom",  4000));
        list.add(new Employee("344-4-6654", "Anne", 3450));
        list.add(new Employee("123-45-6745", "Dan",  8000));
        list.add(new Employee("435-09-3425", "Bruen",  5000));

        HashMap<String,Employee> emap = convertHash(list);
        System.out.println(emap);
        List<String> name = names(emap);
        System.out.println("Names : " + name);
    }
```

Task 1: Need to implement the given method. **[5 Points]**

public static HashMap<String, Employee> convertHash(List<Employee> emps);

Read the Employee list and convert that into the HashMap which takes Employee SSN as a key and Employee object as Value for the HashMap and return the converted HashMap collection.

Example: If the Employee list contains the following items

"234-23-4455", "Joe", 3450
"221-45-9990", "Mike", 5500
“876-99-7654", "Chelle", 8000

The output of the HashMap with the below result

Key as String(SSN)	**Values as Employee object**
"234-23-4455"	"234-23-4455", "Joe", 3450
"221-45-9990"	"221-45-9990", "Mike", 5500
“876-99-7654"	“876-99-7654", "Chelle", 8000

```
public static HashMap<String, Employee> convertHash(List<Employee> emps){
// Write your implementation here
```

}

Task 2: Return the names as a list from the HashMap, the employee who got more than 5000 salary. **[5 Points]**

Example: The method needs to return `[Chelle, Mike, Dan]` for the HashMap input.

public static List<String> names(HashMap<String, Employee> map){

}