Lab 4 OOP

Exercise 1

Implement a class Person (name, cnp, address) and create 5 objects of this class with 2 objects with equal values. Then :

- Put it in an ArrayList. Use an Iterator to parse and display the list. Use a ListIterator to parse the list backwards and add an element at the middle of the list. Compare the objects using iterator.

Using Iterator

```
Iterator itr=set.iterator();
while(itr.hasNext()) {
    A my=(A)itr.next();
    System.out.println(my);
}
```

What can be done with a ListIterator?

- iterate backwards
- obtain the iterator at any point.
- add a new value at any point.
- set a new value at that point.

ListIterator Example

```
ListIterator<String> litr = null;
List<String> names = new ArrayList<String>();
names.add(“Anne");
names.add(“Bob");
//Obtaining list iterator
litr=names.listIterator();
System.out.println("Traversing the list in forward direction:");
while(litr.hasNext()){
    System.out.println(litr.next());
}
System.out.println("\nTraversing the list in backward direction:");
while(litr.hasPrevious()){
    System.out.println(litr.previous());
}
```

Adding elements on a certain position using ListIterator

```
//nextIndex and previousIndex return next and previous
//index from the current position in the list
System.out.println("Previous Index is : " + litr.previousIndex());
System.out.println("Next Index is : " + litr.nextIndex());
// Add an element just before the next element
litr.add(“my_new_element");
```

Other functionalities of ListIterator

```
// void remove() method removes the last element
//returned by next or previous methods
litr.remove();
//void set(Object o) method replaces the last element
   returned
//by next or previous methods. The set method can only be
   called
//if neither add or remove methods are called after last call of
//next or previous methods
litr.set("elem");
for(int i = 0; i < names.size(); i++)
   System.out.println(names.get(i));
```

Hash Set

- There are two methods that a class needs to override to make objects of that class work as hash map keys:

```
public int hashCode();
public boolean equals(Object o);
```

In hashing, the informational content of a key is used to determine a unique value, called its hash code. The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

• **hashCode()** method is where we put our hash function.

• **equals()** method returns true if two objects have the same hashCode().

• If two objects have the same hash code, they may be not equal.

Hash Set Example

```
class Dog{
        String color;
        public Dog(String s) { color = s; }
        //overridden methods
        public boolean equals(Object obj) {
                if (!(obj instanceof Dog))
                        return false;
                if (obj == this)
                        return true;
                return this.color.equals(((Dog) obj).color);
        }
        public int hashCode(){
                return color.length();
        }
}
```

Testing

Output:

true
We have 1 white dogs!
We have a white dog!

```
...
public static void main(String[] args) {
    HashSet<Dog> dogSet = new HashSet<Dog>();
    Dog d1 = new Dog("white");
    Dog d2 = new Dog("white");
    dogSet.add(d1);
    dogSet.add(d2);
    System.out.println(d1.equals(d2));
    System.out.println("We have " + dogSet.size() + " white dogs!");

    if(dogSet.contains(new Dog("white"))){
            System.out.println("We have a white dog!");
    }else{
            System.out.println("No white dog!");
    }
}
```

But if…

```
public int hashCode(){
   return (int) (color.length()+Math.random()*10);
}
```

Possible output:

We have 2 white dogs!

No white dog!

HashSet vs TreeSet

- **HashSet is much faster than TreeSet, but offers no ordering guarantees like TreeSet.**
- **TreeSet** guarantees that elements of set will be sorted (ascending, natural, or the one specified by you via its constructor) (implements SortedSet).

TreeSet example(I)

```
import java.util.Comparator;
import java.util.TreeSet;

public class TreeSetEg {
    public static void main(String a[]) {
    // By using name comparator (String comparison)
    TreeSet<Empl> nameComp = new TreeSet<>(new MyNameComp());
    nameComp.add(new Empl("Ionel", 3000));
    nameComp.add(new Empl("Oana", 6000));
    nameComp.add(new Empl("Andreea", 2000));
    nameComp.add(new Empl("Toma", 2400));
    for (Empl e : nameComp) {
            System.out.println(e);
    }
    }
}

class MyNameComp implements Comparator<Empl> {
    @Override
    public int compare(Empl e1, Empl e2) {
            return e1.getName().compareTo(e2.getName());
    }
}
```

TreeSet Example (II)

```
class Empl {
	private String name;
	private int salary;

	public Empl(String n, int s) {
		this.name = n;
		this.salary = s;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getSalary() {
		return salary;
	}

	public void setSalary(int salary) {
		this.salary = salary;
	}

	public String toString(){
		return "Name: "+this.name+"-- Salary: "+this.salary;
	}
}
```

Exercise 2

Implement a class Product (productName, description, ingredients, expiry date) and create 5 objects of this class with 2 objects with equal values. Then :

- Put it in a HashSet -> implement equals and hashCode

Exercise 3

Implement a class Car (brand, description, maxEngineSpeed) and create 5 objects of this class with 2 objects with equal values. Then :

- Put it in a TreeSet -> implement Comparable