Laborator 3 – Bazele Programarii Orientate pe Obiect

1. Consideratii teoretice ajutatoare pentru implementarea temelor

Operatori relaţionali. Operatorii relationali standard funcţionează pe valori de tip primitiv (int, double, char, float). O parte dintre acestia, si anume operatorii == (test de egalitate) şi != (test de inegalitate) pot fi utilizaţi și pentru a compara referinţe la obiecte. Rezultatul oricărei comparaţii este o valoare de tip boolean (true sau false).

Operator	Semnificație
<	mai mic
<=	mai mic sau egal
==	Egalitate
>=	mai mare sau egal
>	mai mare
!=	inegalitate (diferit de)

In Java, operatorii de comparare nu pot exprima inegalităţi duble. De exemplu o expresie de forma 0 < x < 10 este incorecta in limbajul Java. Pentru a exprima o astfel de inegalitate in Java e necesar sa scriem a două comparaţii, si anume 0 < x && x <10.

Testul de egalitate: ==, equals(), şi compareTo(), compare()

Daca dorim sa testam **tipuri primitive** de date testul de egalitate/ inegalitate se face cu operatorul ==, respective !=.
Exemplu:

```
int a=3;
int b =5;
if (a == b)
        System.out.println("cele doua numere sunt egale");
if(a != b)
        System.out.println("cele doua numere sunt diferite");
```

Daca este vorba de tipuri **referinta** (obiecte) avem urmatoarele variante pentru testarea egalitatii:

a) Comparatia cu operatorii ==, != :
   Daca a, b sunt doua obiecte, a == b, a!= b compară referinţele spre obiecte.
b) Comparatia cu metoda **equals**
   Daca a, b sunt doua obiecte, metoda a.**equals**(b) compară obiectele prin prisma valorilor atributelor lor. Dacă există valori referință ca si atribute ale clasei a caror instante sunt obiectele a, respectiv b, acestea se compară la rândul lor prin equals. Dacă metoda **equals** nu este definita pentru o clasă utilizator atunci aceasta se comportă ca şi operatorul ==
c) Comparatia cu metoda **compareTo**() disponibila in interfata **Comparable**
   a.**compareTo**(b) - compară valori. Clasa utilizator trebuie să aibă implementată interfaţa **Comparable** . Toate clasele Java având relaţia de ordine naturală au implementată această interfaţă (String, Double, BigInteger, ...) Face excepţie clasa BigDecimal care poate produce rezultate imprevizibile. Obiectele **Comparable** pot fi utilizate de metodele **Collections sort**() şi de structurile de date sortate implicit (d.e. TreeSet, TreeMap).
d) Comparatia cu metoda **compare**() disponibila in interfata **Comparator**

compare(a, b) - compară valori. Este disponibilă numai dacă este implementată interfaţa **Comparator** care nu este specifică pentru clasele Java. De obicei se foloseşte pentru a defini un obiect de comparare care va fi transmis metodelor **Collections sort**() sau structurilor de date ordonate. Interfața Comparator definește o metodă de **compare** (arg1, arg2) cu două argumente care reprezintă obiecte comparate și funcționează similar cu metoda Comparable.compareTo ().

Compararea referinţelor la obiecte folosind operatorii == şi != . Dacă doriţi să comparaţi obiecte, trebuie să ştiţi că operatorul == se foloseşte pentru a afla dacă ele sunt acelaşi obiect, iar operatorul != pentru a afla daca este vorba de un obiect diferit. Practic, aceşti operatori compară două valori pentru a vedea dacă ele referă acelaşi obiect. De obicei vreţi să aflaţi dacă obiectele au aceeaşi valoare, şi nu dacă două obiecte constituie o referinţă spre acelaşi obiect. In acest caz se foloseste metoda **equals**() care testeaza dacă ele sunt obiecte diferite, dar au aceeaşi valoare.

Spre exemplu, if (name == "ana")

Aici, rezultatul este **true** dacă *name* este o referinţă către acelaşi obiect spre care face referinţă şi "*ana*", altfel este false

Compararea valorilor obiectelor utilizând metoda equals(). Metoda **equals** compara valorilor obiectelor se si returneaza o valoare de tip boolean (true daca valorile celor doua obiecte sunt egale, false altfel). Mai multe clase, cum este si cazul clasei **String** definesc metoda **equals**() pentru a compara valori ale obiectelor.

Daca vrei sa comparam valorile obiectelor exemplu de mai sus poate fi rescris astfel:

if (name.**equals**(”ana"))

In acest caz se compară valori, nu referințe.

Interfaţa **Comparable**. Metoda **equals** precum şi operatorii == şi != sunt prevăzuţi pentru testul de egalitate/inegalitate, dar nu oferă o metodă de a testa valori înrudite. Unele clase (d.e. String şi alte clase având relaţia de ordine naturală) au implementată interfaţa **Comparable**, care defineşte metoda **compareTo** . Va trebui să implementaţi Comparable în clasele proprii dacă aveţi nevoie să le utilizaţi în metodele **Collections.sort()** sau **Arrays.sort**(). Clasa **String** oferă şi metode de comparaţie care nu fac diferenţă între literele mari şi mici.

Interfaţa **Comparable** face parte din pachetul java.lang și metoda **compareTo** (Object o). Metoda primeste un singur parametru repezetand obiectul de comparat. Metoda returneza valoarea 0 dacă argumentul este un șir egal din punct de vedere lexicografic cu acest șir; o valoare mai mică de 0 dacă argumentul este un șir lexicografic mai mare decât acest șir; și o valoare mai mare decât 0 dacă argumentul este un șir lexicografic mai mic decât acest șir.

Exemplu de sortare a unor obiectelor de tip Customer prin implementarea interfetei Comparable [1]:

```
public class Author implements Comparable<Author> {

 String firstName;
 String lastName;
 String bookName;
 Author(String first, String last, String book){
            this.firstName = first;
            this.lastName = last;
```

```
            this.bookName = book;
 }

 @Override
 /*
  * This is where we write the logic to sort. This method sort
  * automatically by the first name in case that the last name is
  * the same.
  */
 public int compareTo(Author au){
    /*
     * Sorting by last name. compareTo should return < 0 if this(keyword)
     * is supposed to be less than au, > 0 if this is supposed to be
     * greater than object au and 0 if they are supposed to be equal.
     */
    int last = this.lastName.compareTo(au.lastName);
    //Sorting by first name if last name is same d
    return last == 0 ? this.firstName.compareTo(au.firstName) : last;
 }
}
```

```
import java.util.ArrayList;
import java.util.Collections;
public class SortAuthByNames{
   public static void main(String args[]){
      // List of objects of Author class
      ArrayList<Author> al=new ArrayList<Author>();
      al.add(new Author("Henry","Miller", "Tropic of Cancer"));
      al.add(new Author("Nalo","Hopkinson", "Brown Girl in the Ring"));
      al.add(new Author("Frank","Miller", "300"));
      al.add(new Author("Deborah","Hopkinson", "Sky Boys"));
      al.add(new Author("George R. R.","Martin", "Song of Ice and Fire"));

      /*
       * Sorting the list using Collections.sort() method, we
       * can use this method because we have implemented the
       * Comparable interface in our user defined class Author
       */
      Collections.sort(al);
      for(Author str:al){
                System.out.println(str.firstName+" "+ str.lastName+" "+"Book: "+str.bookName);
      }
   }
}
```

Iesirea va fi:

```
Deborah Hopkinson Book: Sky Boys
Nalo Hopkinson Book: Brown Girl in the Ring
George R. R. Martin Book: A Song of Ice and Fire
Frank Miller Book: 300
Henry Miller Book: Tropic of Cancer
```

Interfata Comparator. Interfața Comparator este utilizată pentru a ordona obiectele din clasa definită de utilizator. Această interfață face parte din pachetul java.util și conține 2 metode de tip **compare** (Object obj1, Object obj2) și o metoda **equals** (Element obiect). Folosind comparatorul, putem sorta elementele dintr-o lista de obiecte instanta a calei definite de utilizator de pe baza datelor membru ale clasei (atribute).

Exemplu de sortare a unor obiectelor de tip stundet prin implementarea interfetei Comparator [2]

```
import java.util.*;
import java.lang.*;
import java.io.*;
// A class to represent a student.
class Student
{    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name, String address)
    {
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    // Used to print student details in main()
    public String toString()
     {
        return this.rollno + " " + this.name + " " + this.address;
     }
}

class Sortbyroll implements Comparator<Student>
{
    // Used for sorting in ascending order of roll number
    public int compare(Student a, Student b)
     {
        return a.rollno - b.rollno;
     }
}

class Sortbyname implements Comparator<Student>
{
    // Used for sorting in ascending order of name
    public int compare(Student a, Student b)
```

```
    {
      return a.name.compareTo(b.name);
    }
}


// Driver class
class Main
{
   public static void main (String[] args)
   {
      ArrayList<Student> ar = new ArrayList<Student>();
      ar.add(new Student(111, "bbbb", "london"));
      ar.add(new Student(131, "aaaa", "nyc"));
      ar.add(new Student(121, "cccc", "jaipur"));

      System.out.println("Unsorted");
      for (int i=0; i<ar.size(); i++)
         System.out.println(ar.get(i));

     Collections.sort(ar, new Sortbyroll());

      System.out.println("\nSorted by rollno");
      for (int i=0; i<ar.size(); i++)
         System.out.println(ar.get(i));

      Collections.sort(ar, new Sortbyname());

      System.out.println("\nSorted by name");
      for (int i=0; i<ar.size(); i++)
         System.out.println(ar.get(i));
   }
}
```

Output:

```
Unsorted
111 bbbb london
131 aaaa nyc
121 cccc jaipur

Sorted by rollno
111 bbbb london
121 cccc jaipur
131 aaaa nyc

Sorted by name
131 aaaa nyc
111 bbbb london
121 cccc jaipur
```

Probleme de rezolvat la laborator:

1. Scrieti un program java care citeste de la consola un numar de secunde si intoarce numarul maxim de ore, de minute, de secunde care este echivalent cu numarul initial de secunde.

   Exemplu: 7384 secunde este echivalent cu 2 ore, 3 minute si 4 secunde.

   Observatie: Pentru citirea de la tastatura se poate folosi clasa Scanner din Java.

```
Scanner s = new Scanner(System.in);
int time  = s.nextInt();
```

2. Scrieti un program Java care citeste de la tastatura doua numere intregi **a**, **b** si calculeaza cmmdc si cmmmc dintre cele doua numere. Pentru aceasta se va defini o clasa in care vor fi definite doua metode: una pt calculul cmmdc, iar alta pentru calculul cmmmc.

   Observatie:

   Algoritmul lui Euclid pentru calcularea **cmmdc** a doua numere**:** Pentru două numere **a** şi **b** se atribuie lui **b** restul împărţirii lui **a** la **b**, iar lui **a** vechea valoare a lui **b.** Rezolvarea problemei se bazează pe condiţia **b≠0**. Se repetă procesul de împărţire **« *până când* » *r=0***.
   Paşii algoritmului sunt:
   P1) Se împarte **a** la **b** şi se obţine restul restul **r (r ←a mod b)**
   P2) Se execută operaţiile de atribuire **a←b; b←r;**
   P3) Dacă **b ≠ 0**, atunci se revine la pasul 1, altfel **cmmdc←a.**

   Exemplu: dacă **a =36** şi **b=24**, calculul se desfăşoară astfel:
   1) se calculează restul **r ← 36 mod 24 ; => r ← 12;**
   2) se fac atribuirile a **←24, b←12**
   3) **cum b** ≠ 0 se calculează din nou restul **r ←24 mod 12; => r ←** 0;
   4) se fac atribuirile **a←12, b←0,**
   5) cum **b=0 => cmmdc ←a;=>cmmdc ← 12** (ultimul rest nenul)

   cmmmc - Cel mai mic multiplu comun a doua sau mai multe numere este cel *mai mic numar* natural diferit de 0 care se divide cu numerele date. Pentru a gasi cel mai mic multiplu comun a mai multor numere, se descompun numerele in factori primi si se face ***produsul factorilor primi comuni si necomuni luati o singura data, la puterea cea mai mare.***

   1) Calcularea cmmmc se bazează pe calculul cmmdc.
   2) Daca notam cu cmmdc(a,b) cel mai mare divizor comun al numerelor a şi b si cu cmmmc(a,b) cel mai mic multiplu comun al numerelor a şi b atunci :
   3) Produsul a doua numere naturale este egal cu produsul dintre cel mai mare divizor comun si cel mai mic multiplu comun.

      a*b= cmmdc(a,b)*cmmmc(a,b)

   4) din aceasta formula rezulta ca **cmmmc(a,b)=(a*b)/cmmdc(a,b)**
   5) pentru a calcula cmmmc(a,b) avem nevoie să determinăm cmmdc(a,b)

6) in cazul in care ambele numere au valoarea 0 cmmmc(a,b) nu se poate calcula

3. Scrieti un program java care efectueaza operatii pe numere complexe.
   a. Creați clasa ComplexNumber.java. Aceasta va avea două campuri: re si img ambele de tip float, si doi contructori (unul cu parametrii, iar altul fara parametrii). Clasa va implementa metode de tip get si set .
   b. Creați clasa Operations.java. Această clasă va implementa operațiile de adunare, scadere, înmulțire si impartire, modulo pentru numere complexe. De asemenea se vor define metodele **equals (**compara doua numere complexe folosind ca si criteriu de comparatie partea reala si partea imaginara a numerelor complexe; returneaza **true** daca cele doua numere complexe comparate sunt egale, altfel **false)** si **toString (**permite obţinerea unui **String** cu informaţii despre obiectul pe care este apelată**)**. Definiți în clasă Operations câte o metodă pentru fiecare operație;
   c. Creati o clase de test. In clasa de test creati un meniu care ii permite utilizatorului sa introduca cele doua numere complexe de la tastatura si sa selecteze operatia care doreste sa se efectueze pe ele. Utilizatorul poate iesi din aplicatie cand alege optiune exit din meniu.

   Observatie: Impartirea a doua numere complexe se face dupa modelul:

$$\frac{a+bi}{c+di} = \frac{(a+bi)(c-di)}{(c+di)(c-di)} = \frac{(ac+bd)+i(bc-ad)}{c^2+d^2} = \frac{ac+bd}{c^2+d^2} + i \cdot \frac{bc-ad}{c^2+d^2}.$$

4. Scrieți un program în care fiind dată o variabilă care stochează o referință către un String, determină:
   - numărul de consoane și vocale
   - indicii pentru o vocală introdusă din linia de comandă

Nota ajutatoare:

Folositi-va de metodele definite in casa String pentru implementarea acestui task (ex. char[] toCharArray() – metoda care converteste un string intr-un arry de caractere)

5. Implementati o aplicatie java in care creați urmatoarele clasele:
   - Clasa Student cu atributele *name* de tip String si *year* de tip Integer. Clasa defineste constructori, metoda toString, equals (metoda equals va compara doi studenti dupa nume si anul de studiu) si implementeaza interfata Comparable**.**
   - Clasa **Course** cu atributele title de tip String, description de tip String, students - array de obiecte de tip Student.In clasa **Course** se va defini o metoda filterbyYear care returneaza o listă de studenți care sunt intr-un an dat ca parametru.Lista de studenti returnata va fi sortata dupa numele studentului.
   - O clasa **Test** cu metoda main. In interiorul metodei main creati un obiect **Course** și mai multe obiecte de tip **Student**. Populați obiectul **Course** cu obiecte de tip **Student**.Afișați toți studenții din anul dat ca parametru ordonati dupa nume. Creați două obiecte **Student** cu aceleași date în ele. Afișați rezultatul folosirii **equals**() între ele.
6. Implementati o aplicatie java pentru lucru cu numere rationala.
   - Creați clasa NumarRational.java. Aceasta va avea două campuri: numitor si numarator si doi contructori (unul cu parametrii, iar altul fara parametrii).
   - Creați clasa Operatii.java. Această clasă va implementa operațiile de adunare, scadere, inmultire si impartire, afisare, citire numere rationale. Tot timpul fractia retinuta de un obiect trebuie sa fie ireductibila.
   - Se va crea o clasa de **Test** care va defini o interfata din consola care preia datele si apeleaza metoda dorita.

Bibliografie

1. https://beginnersbook.com/2017/08/comparable-interface-in-java-with-example/
2. https://www.geeksforgeeks.org/comparator-interface-java/