CAPITOLUL 1

1.1. Clase, obiecte, constructori.

1.1.1. Clase

Clasele definesc structura şi comportamentul unui set de elemente din domeniul pentru care este dezvoltat software-ul modelului. Într-un obiect orientat sunt patru tipuri de relatii: dependenţă, generalizare, asociere, şi realizarea. Dependenţa este raportul de utilizare, indicând faptul că schimbarea starii obiectului al unei clase poate afecta obiectul altei clase pe care o utilizează. Generalizarea înseamnă că obiectele clasei derivate pot fi folosite ori de câte ori există obiecte a clasei de bază dar nu invers. Realizarea este raportul dintre clasificatori (clase, interfeţe), în care unul dintre ei descrie un alt clasificator. Asociaţia arată că obiectele dintr-o clasă se referă la obiecte din altă clasă, şi să reflectă între ele. Tipul "class" în Java poate conţine câmpuri de date, constructori, metode, clase interne şi interfeţe.

1.1.2. Constructori

Constructorul este o metodă care se activează în mod automat atunci când se crează o clasă şi poate efectua acţiuni asupra obiectului pe care îl iniţializează. Constructorul are acelaş nume ca si clasa. Constructorul nu returnează valori, dar poate avea parametri pentru a fi supraîncărcat.

```
/*Exemplu 1 de supraîncarcare a constructorului: */
class NewBook {
 private String title, publisher;
 private float price;
  public NewBook() {
    title = "NoTitle";
    }
public NewBook(String t,String pub,float p) {
    title = new String(t);
    publisher = pub;
    price = p;
    }
public static void main(String[] args) {
NewBook tips1;
tips1 = new NewBook();
// initializarea
NewBook tips2 =
new NewBook("Java2", "C++", 9.f);
}}
```

În cazul în care constructorul nu este definit în clasa, Java oferă un constructor implicit care iniţializează obiectul cu valori implicite.

1.1.3. Clase-Interfeţe

Pentru a nu pune în aplicare toate metodele din interfeţele corespunzătoare, atunci când se creează clasa - blocuri pentru evenimente, sunt folosite clase – Interfeţe. Aceste clase contin metode interfeţe, evenimente realizate pe care le extinde. În acest caz, se defineşte o nouă clasă care acţionează ca un bloc de evenimente realizate şi realizează doar evenimentele care necesită. De exemplu, clasa **MouseMotionAdapter** are două metode**: mouseDragged()** şi **mouseMoved()**. Esenta acestor metode sunt aceleaşi ca şi în interfaţa **MouseMotionListener**. Dacă există necesitatea doar în evenimentele de clic a mouse-ului, se extinde adaptorul realizînd override cu metoda **mouseDragged()** din această clasa. Evenimentul de clic a mouse-ului se realizeaza prin aplicarea metodei **mouseMove().** Pentru a crea o interfaţă grafică trebui să fie indicat un loc (fereastră), în care aceasta va fi afişată. Fereastra este un container care creaza o interfaţă cu utilizatorul. Contextul grafic este incapsulat în clasă şi este disponibil în două moduri:

- prin activarea metodelor **paint(), update();**
- prin valoarea returnata a metodei **getGraphics()** din clasa **Component.**

Evenimentele **FocusEvent** indică că componentele au fost înregistrate sau nu. Clasa **InputEvent** este clasa de bază pentru clasele **KeyEvent şi MouseEvent**. Evenimentul **WindowEvent** indică un program in care a fost activată una din sistemele de control a fereastrei. Următorul exemplu creează obiectul **MyMouseWithFrame** şi permite controlul său direct in metoda **main()**.

```
/ / Exemplu # 2 : utilizarea interfeţii:
import java.applet.*;
import java.awt.*;
 import java.awt.event.*;
 public class MyMouseWithFrame extends Frame implements
ActionListener{
Button button = new Button("Button");
 public String msg = "none";
public MyMouseWithFrame() {
addMouseListener(new MyMouseAdapter(this));
 setLayout(null);
setBackground(new Color(255, 255, 255));
 setForeground(new Color(0, 0, 255));
 button.setBounds(100, 100, 50, 20);
 button.addActionListener(this);
 add(button); }
public static void main(String[] args) {
MyMouseWithFrame myf=new MyMouseWithFrame();
 myf.addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent e) {
                    System.exit(0);
};});
 myf.setSize(new Dimension(250, 200));
 myf.setTitle("Frame - window");
 myf.setVisible(true);
 }
public void paint(Graphics g) {
 g.drawString(msg, 80, 50);
}
```

```
public void actionPerformed(ActionEvent e) {
     msg = "Button is pressed";
     repaint();
      }}
class MyMouseAdapter extends MouseAdapter {
     public MyMouseWithFrame mym;
public MyMouseAdapter(MyMouseWithFrame mym) {
     this.mym = mym;
     }
public void mousePressed(MouseEvent me) {
     mym.msg = "Mouse button is pressed";
     mym.repaint();
     }
public void mouseReleased(MouseEvent me){
mym.msg = "Mouse button is released";
     mym.repaint();}}
```

Rezultatul realizării programului:

Constructorul clasei foloseste **MyMouseWithFrame addMouseListener (new MyMouseAdapter (this))** pentru a înregistra evenimentele **mouse**. La crearea unui obiect de tip **MyMouseWithFrame**, metoda dată indică ca obiectul este interesat în prelucrarea anumitor evenimente. Clasa abstracta **MouseAdapter** este utilizata pentru a trata evenimente asociate cu **mouse** prin crearea blocului care conţine metodele **mousePressed(MouseEvent e), mouseReleased(MouseEvent e)**. Când un obiect generează evenimentul **WindowEvent**, obiectul **MyMouseWithFrame** analizează dacă este un eveniment **WindowClosing**. Dacă nu, obiectul **MyMouseWithFrame** îl ignorează. Dacă este evenimentul aşteptat, programul incepe procesul de realizare a activităţii sale. Clasa abstractă **WindowAdapter** este utilizată pentru primirea şi prelucrarea evenimentului. Clasa conţine metodele reîncărcare **windowActivated (WindowEvent e)**, apelată la activarea ferestrei şi **windowSlosing (WindowEvent e)** apelată la închiderea ferestrei.

1.2. Siruri de caractere

Java Library System conţine clasele **String** şi **StringBuffer**, care lucrează cu siruri de caractere, definite în pachetul din java. lang. Aceste clase sunt declarate ca ***final***.

1.2.1. Clasa String

Clasa **String** conţine constructorii **String (), String (String str.), String (char [] c), String (asciichar byte [])**. Aceşti constructori sunt folosiţi pentru a iniţializa obiectele din clasă.

Clasa String contine metode pentru a lucra cu şiruri de caractere:

concat(String s) sau **+** – concatenează două siruri;

equals(Object ob), equalsIgnoreCase(String s) – compară două şiuri cu şi fara inregistrare;

compareTo(String s), compareToIgnoreCase (String s) – compară două siruri cu şi fara inregistrare;

contentEquals(StringBuffer ob) – compararea sirului şi a conţinutului unui obiect de tip StringBuffer;

charAt(int n)– extraje dintr-un şir de caractere un caracter cu indicele specificat;

substring(int n, int m)- extrage dintr-un şir de caractere un subşir cu lungimea m-n, începând de la poziţia n;

length() – determina lungimea sirului;

valueOf(obiect) – transformarea unui obiect în şir de caractere;

toUpperCase()/ toLowerCase() – convertiază toate caracterele din şir în litere majuscule / litere minuscule;

replace(char c1, char c2) –înlocueşte toate apariţiile în sir a primului caracter cu al doilea;

getBytes(obiect), getChars(obiect) –sirul de caractere e transformat în tablou de octeţi sau caractere;

În următorul exemplu este descrisă o serie de numere întregi care sunt convertite în obiecte String, folosind metodele din această clasă.

```
/ / Exemplu # 3: utilizarea clasei:
public class DemoString {
public static void main(String[] args) {
          char s[] = { 'J', 'a', 'v', 'a' };
          int i = 2;
String str = new String(s); //str="Java"
          i = str.length(); //i=4
String num = String.valueOf(2); //num="2"
     str = str.toUpperCase(); //str="JAVA"
     num = str.concat(num); //num="JAVA2"
     str = str + "C";//str="JAVAC";
     char ch = str.charAt(2); //ch='V'
i = str.lastIndexOf('A');
//i=3 (-1 daca lipseste)
num = num.replace('2', 'H'); //num="JAVAH"
i = num.compareTo(str);
//i=5(impreuna cu simbolurile 'H' si 'C')
 str.substring(0, 3).toLowerCase(); //JAVA
}
```

Deoarece obiectul a fost transmis de referinţă, orice schimbare a obiectului în metodă ar trebui să fie transmis obiectului original, deoarece ambele referinţe sunt egale. Acest lucru nu se intamplă deoarece apelînd metoda **concat()**, se crează un nou obiect prin care se face referire. Acest obiect este returnat de către operatorul **return**, dar valoarea returnata nu este atribuită, deci toate modificările sunt pierdute. Dacă codul va fi modificat aşa cum se arată în comentarii, toate modificările obiectului realizate în metoda **changeStr()** vor fi salvate în obiectul declarat în **main().**

În următorul exemplu, se examinează modalităţi de stocare şi identificare a obiectelor utilizînd metoda **equals()**, se compară String-ul cu obiectului specificat a metodei **hashCode()**, care apreciauă codul de distribuire a obiectului.

```
/*Exemplu # 4: Utilizaţi equals(), hashCode():*/
EqualStrings.java */
public class EqualStrings {
   public static void main(String[] args) {
     String s1 = "Java";
     String s2 = "Java";
     String s3 = new String(s1);
System.out.println(s1 + "==" + s2 + " : " + (s1==s2));//true
System.out.println(s1 + "==" + s3 + " : " + (s1==s3));//false
System.out.println(s1 + " equals " + s2 + " : " +
s1.equals(s2));//true
System.out.println(s1 + " equals " + s3 + " : " +
s1.equals(s3));//true
System.out.println(s1.hashCode());
System.out.println(s2.hashCode());
System.out.println(s3.hashCode());
//toate valorile sunt egale!!!
   }}
```

Metoda **HashCode()** este metoda care returneaza codul de distribuire a obiectului, care este apreciat în conformitate cu principiul - diferite obiecte cu conţinut de acelaşi tip au diferite hash-coduri. Este posibil de a redefini referinţa de la obiect la literal, prin apelul metodei **intern()**.

```
    //Exemplu # 5 : utilizarea metodei intern ()
    public class DemoIntern {
    public static void main(String[] args) {
          String s1 = "Java";
          String s2 = new String("Java");
     System.out.println(s1 == s2); //false
          s2 = s2.intern();
     System.out.println(s1 == s2); //true
     }}
```

În următorul exemplu are loc sortarea sirurilor de caractere.

```
    / * Exemplu # 6: Sortare: */
    public class SortArray  {
      public static void main(String[] args)  {
    String a[] = {"  Vika", "Natasha  ", " Alina", "   Dima ",
"Denis  "};
    for(int j = 0; j < a.length; j++) a[j].trim();
    for(int j = 0; j < a.length; j++)
      for(int i = j + 1; i < a.length; i++)
         if(a[i].compareTo(a[j]) < 0)   {
          String t = a[j];
    a[j] = a[i];
    a[i] = t;
```

```
    }
   for(int i = 0; i < a.length; i++)
    System.out.print(a[i] + "   ");
       }}
```

Metoda **trim()** asigură eliminarea tuturor lăcunilor de la începutul şi sfîrsitul şirului. Metoda **CompareTo()**-efectuază o comparaţie a şirurilor după principiul Unicode.

1.2.2. Clasa StringBuffer

Clasa **StringBuffer** este asemănătoare cu clasa **String**, dar, spre deosebire, conţinutul şi mărimea obiectelor din clasa **StringBuffer** pot fi schimbate. Constructorul poate primi ca parametru un obiect **String** sau mărimea obiectului. Obiectele acestei clase pot fi transformate în clasa **String** prin metoda **toString()** sau folosind constructorul clasei **String**. Setarea lungimii buferului este efectuată de metoda **setLength(int n)**. Pentru a adăuga simboluri, valori de tipuri standarde, tablouri, şiruri de caractere, folosim metoda **append(parametri)**. Inserarea unui simbol, unui obiect sau a unui şir în poziţia indicată se realizează cu metoda **insert(opţiuni)**.

```
    /*Exemplu # 7: Crearea unui obiect StringBuffer şi a
proprietăţilor sale: */
public class DemoStringBuffer {
public static void main(String[] args) {
     StringBuffer sb = new StringBuffer();
System.out.println("lungimea-"+sb.length());
System.out.println("mărimea"+sb.capacity());
     //sb = "Java"
//error, numai pentru clasa String
          sb.append("Java");
     System.out.println("şirul ->" + sb);
System.out.println("lungimea-"+sb.length());
System.out.println("mărimea"+sb.capacity());
System.out.println("revers-"+sb.reverse());
          }}
Rezultatul codul de mai sus este:
Lungimea ->0
Size ->16
String ->JAVA
Lungime ->4
Size ->16
Revers ->avaJ
```

Folosind metoda **reverse()** se poate schimba rapid ordinea caracterelor in obiect. În cazul în care metoda activată pentru un obiect StringBuffer, produce modificări în conţinutul obiectului, nu se creează un obiect nou, ca în cazul cu obiecte din clasa **String**, dar modifică obiectul curent **StringBuffer**.

```
    /*Exemplu # 8: Modificarea obiectului StringBuffer: * /
public class RefStringBuffer {
     static void changeStr(StringBuffer s) {
          s.append(" Microsystems");
     }
```

```
public static void main(String[] args) {
StringBuffer str = new StringBuffer("Sun");
          changeStr(str);
          System.out.println(str);
     }}
```

Rezultatul realizării:

Sun Microsystems

Pentru clasa **StringBuffer** nu poate fi realizată suprascrierea a metodelor **equal()** şi **hashCode()**, adică compararea conţinutului a două obiecte nu este posibil, chiar şi codurile hash a tuturor obiectelor se apreciază exact ca şi în clasa Object.

```
    /*Exemplu # 9: compararea obiectelor StringBuffer şi codurile
hash * /
public class EqualsStringBuffer {
public static void main(String[] args) {
StringBuffer sb1 = new StringBuffer("Sun");
StringBuffer sb2 = new StringBuffer("Sun");
     System.out.println(sb1.equals(sb2));
     System.out.println(sb1.hashCode() ==sb2.hashCode());
     }}
```

Rezultatul acestui program va fi :

```
false
false
```

Lucrare de laborator nr. 1

1. Tema lucrării:

Clase, obiecte, constructori.

2. Scopul lucrării:

- Însuşirea modalităţilor de creare a claselor, obiectelor în Java;
- Însuşirea modalităţilor de prelucrare a şirurilor de caractere utilizând clasele de bază;

3. Etapele de realizare:

1) Crearea unor clase noi;
2) Crearea şi iniţializarea obiectelor
3) Utilizarea metodelor claselor Sting şi StringBuffer pentru modificarea şirurilor;
4) Crearea interfeţii programului;
5) Prezentarea lucrării.

4. Exemplu de realizare:

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
```

```
public class Main
{
public static void main(String[] args){
          String sentence = "";
          int wordLenght = 0;
          String myWord = "";
InputStreamReader is =
new InputStreamReader(System.in);
     BufferedReader bis =
new BufferedReader(is);
          try
{
System.out.println("Itrodu propoziţia: ");
          sentence = bis.readLine();
System.out.println("Introdu lungimia cuvântului  înlocuit");
          wordLenght = Integer.parseInt(bis.readLine());
System.out.println("Introdu cuvîntul care trebue înlocuit");
          myWord = bis.readLine();
}
catch (IOException e)
{
     e.printStackTrace();
}
     Text myText = new Text(myWord, sentence, wordLenght);
          myText.changeSentence();
System.out.println("Propoziţia nouă" + myText.getSentence());
}}
class Text
{
     private String mySentence;
     private int charNumber;
     private String wordToChange;
     private String newSentence = "1.";

public Text(String wordToChange,
     String mySentece, int charNumber) {
          this.mySentence = mySentece;
          this.wordToChange = wordToChange;
          this.charNumber = charNumber;
}
     public String getSentence()
{
          return newSentence;
}
     public void changeSentence()
{
          int firstPos = 0;
```

```
          int i;
for (i = 0; i < mySentence.length(); i++)
 {
     if (mySentence.charAt(i) == ' ')
{
     if (i - firstPos == charNumber)
{
     newSentence = newSentence.concat(wordToChange+" ");
          firstPos = i+1;
}
else
{
newSentence = newSentence.concat
(mySentence.substring(firstPos, i+1));
     firstPos=i+1;
}}
else if(i == mySentence.length()-1)
{
     if (i - firstPos == charNumber)
{
     newSentence = newSentence.concat(wordToChange+" ");
          firstPos = i+1;
}
else
{
     newSentence = newSentence.concat
(mySentence.substring(firstPos, i+1));
          firstPos=i+1;
                    }}}}}
```

Rezultatul realizării:

Îtrodu propoziţia:
Flori de tei apar în mai.
Introdu lungimea cuvântului înlocuit :
3
Introdu cuvântul spre înlocuire :
măr
Propoziţia noua este : 1. Flori de măr apar în măr.

5. **Probleme propuse:**

1. În fiecare cuvânt din textul dat simbolul de pe poziţia indicată a cuvântului de înlocuit cu un simbol dat. Dacă poziţia indicată este mai mare ca lungimea cuvântului, corecţia nu se face.
2. Fiecare litera din textul dat, sa fie înlocuită cu poziţia numerică din alfabet. Textul sa se scrie într-o singură linie cu doua spatii intre litere. In linia următoarea sub fiecare literă sa se indice poziţia.

3. În textul dat, cuvintele cu lungimea indicată, sa se înlocuiască cu un subşir specificat, lungimea căruia poate să nu coincidă cu lungimea cuvântului.
4. În textul dat după fiecare caracter indicat, să se introducă un subşir specificat.
5. După fiecare cuvânt din textul dat, care se termina cu un subşir specificat, sa se adauge cuvântul indicat.
6. În dependenţă de valoarea introdusă (0 sau 1), în textul dat de extras caracterul indicat, ori de câte ori apare, sau sa fie introdus după simbolul cu indicele specificat.
7. Extrageţi din textul dat toate cuvintele cu lungimea specificată, care încep cu o consoană.
8. Afişaţi de câte ori sunt repetate cuvintele care apar în textul dat.
9. Afişaţi, care litere, vocalele sau consoanele, sunt mai multe in fiecare propoziţie din textul dat.
10. Din textul dat găsiţi şi extrageţi numărul de cuvinte care încep şi se finalizează cu vocale.
11. De afişat fără repetare cuvântul din textul dat, care încep si se termina cu aceea literă.
12. De extras din textul dat primul subşir cu lungime maximă, care nu conţine litere.