**5.1.Crearea excepţiilor**

Fiecare caz excepţional este în conformitate cu anumite clase. În cazul în care clasa corespunzătoare nu există, atunci ea poate fi creată. Excepţii sunt moştenite de la clasa de bază **Throwable**  şi clasele sale derivate **Error** şi **Exception** din pachetul **java.lang.**

În Java atunci când o excepţie este generată întotdeauna este creat un obiect de tip corespunzător, care trebuie să fie prelucrat de către un anumit bloc de cod. Atunci când se creează o metodă prelucrarea excepţiei poate fi inclusă în codul, care poate genera această metodă.

Există trei moduri de a crea excepţiile :

- blocuri **try – catch ;**

- declaraţia excepţiei în secţiunea **throws**  a metodei şi transmiterea apelantului

- capturarea excepţiei, convertirea la altă clasă şi repetarea apelului

Examinare primului mod are loc în următorul exemplu. La I/O se foloseşte o clasa IOException. De exemplu:

**public** **void** doRead(){

**try**{

System.in.read(buffer);

}

**catch** (IOException e) {

String err = e.toString();

System.out.println(err);

}}

Activarea clasei IOException este generată de metoda **read():**

**public int read() throws IOException**

Metoda **doRead()** în blocul try apelează metoda **read(),** care încearcă să introducă datele din **System.in** în **buffer**. Dacă este o eroare de intrare, realizarea trece la bloca **catch**, care prelucrează excepţia, în caz contrar blocul **catch** nu este realizat. Atunci cînd blocul **try-catch** este omis, compilatorul va indica o eroare.

A doua abordare poate fi demonstrat în acelaşi exemplu.

Dacă o metodă poate genera o excepţie, pe care nu o prelucrează, atunci o transmite pentru prelucrare altei metode, şi trebuie să anunţe acest comportament prin utilizarea cuvîntului cheie **throws**, astfel încât metodele apelate să se poată proteja de aceste excepţii.

Însă, metoda poate conţine blocul **try-catch**, sau poate sa nu-l conţină. De exemplu, metoda doRead() poate fi declarată:

**public** **void** doRead() **throws** IOException{

System.in.read(buffer);

}

Va prelucra excepţia metoda care va activa metoda **doRead()**:

**public** **void** myDoRead(){

**try** {

doRead();

}

**catch** (IOException e){

String err = e.toString();

System.out.println(err); }}

A treia metodă va fi examinată mai jos la crearea excepţiilor personale.

O metodă poate prelucra câteva excepţii.

/ \* Exemplu # 1: prelucrarea a 2 tipuri de excepţii: \* /

**class** TwoException {

**public** **static** **void** main(String[] args) {

**try** {

**int** a = (**int**)(Math.random() \* 2);

System.out.println("a = " + a);

**int** c[] = { 1/a };

c[1] = 1;

}

**catch**(ArithmeticException e){

System.out.println("divizare la 0"+ e);

}

**catch**(ArrayIndexOutOfBoundsException e) {

System.out.println("depăşeşte limitelle tabloului: " + e);

}

System.out.println

("după blocul try/catch");}}

Excepţia *"*divizare la 0*"* apare atunci când iniţializarea elementului din matrice a = 0. În caz contrar, (a = 1) se genereayă o excepţie “depăşeşte limitele tabloului” atunci când încearcă să atribuie o valoare pentru al doilea element al matricei **c** [], care conţine doar un element.

Clasele derivate de excepţii trebuie să fie plasate înainte de orice clasă părinte a lor, în caz contrar clasele de bază vor prinde aceste excepţii. De exemplu:

**catch (Exception e ){}**/\* clasa de bază **Exception** prinde toate excepţiile \* /

**catch (ArithmeticException e) {}** / \* nu poate fi activat, astfel încât se produce o eroare de compilare \* /

Operatorul **try** poate fi inclus în alt **try**. Dacă operatorul **try** nu conţine **catch,** care corespunde excepţiei date, căutarea va fi continuată pe urmărotul nivel, şi vor fi inspectate secţiunile **catch** din următorul nivel **try.**

/\* exemplu # 2 : blocurile incluse try-catch \*/

**class** MultiTryCatch {

**public** **static** **void** main(String[] args) {

**try** {//bloc exterior

**int** a = (**int**)(Math.random() \* 2);

System.out.println("a = " + a);

**try** {//bloc interior

**int** c[] = { 1/a };

c[1] = 1;

} **catch**(ArrayIndexOutOfBoundsException e) {

System.out.println(

"depăşeşte limitele tabloului: " + e);

}}

**catch**(ArithmeticException e){

System.out.println("divizarea la 0"+ e);

}}}

In rezultatul execuţiei pentru a = 0 va fi aruncată excepţia **ArithmeticException**, însă blocul de prelucrare potrivit **try/catch** este extern pentru locul de generare a excepţiei. Acest bloc va fi utilizat pentru prelucrarea excepţiei care a avut loc.

* 1. **Оperatorul throw**

Operatorul **throw** se utilizeauă pentru generarea excepţiilor. În acest caz se utiliyeayă obiectul clasei **Throwable sau obiectul clasei derivate**, sau referinţile lor. Sintacsa instrucţiunii **throw**:

**throw Obiect*Throwable;***

După realizarea acestui operator, realiuarea codului programului se întrerupe. Cel mai apropiat bloc **try** găseşte realizarorul de excepţie necesar **catch**. Dacă este găsit , începe realiuarea lui, altfel este controlat următorul bloc înclus **try**.

În exemplul următor se crează un obiect excepţie şi apoi operatorul **throw** generează excepţia prelucrată în **catch** în cere la rîndul său este generată o altă excepţie.

// Exemplu # 3 : generarea excepţiilor //

**public** **class** ThrowGeneration {

**static** **void** throwGen() {

**try** {

**throw** **new** ClassCastException("demonstratia");

}

**catch** (ClassCastException e) {

System.out.println

("exceptia metodei throwGen()");

**throw** e;//generarea a unei exceptii

}}

**public** **static** **void** main(String[] args) {

**try** {

throwGen();

} **catch**(ClassCastException e) {

System.out.print(

"generarea metodei in afara metodei: " + e);

}}}

Metoda **throwGen()** crează obiectul clasei **ClassCastException** şi aruncă o excepţie. Prelucrarea excepţiei spune că o excepţie este aruncaăt, şi apoi o generează, astfel excepţia **ClassCastException** este transferată la rutina de tratare a excepţiei în metoda **main(),** în caz contrar, compilatorul ar cere tratarea exceptiilor în metodă sau decizia de această excepţie prin instrucţiunea **throw**.

Daca metoda genereaza exceptia cu ajutorul operatorului **throw,** blocul **catch** lipseste, atunci tipul clasei de excepţii trebuie să fie specificat în operatorul **throws** la declararea metodei pentru toate tipurile de excepţii, în afară de excepţiile clasei **RuntimeException.**

/ / Exemplu # 4: Utilizarea throws //

**public** **class** ThrowsSample {

**static** **void** method() **throws** IllegalAccessException {

System.out.println("In interiorul metodei");

**throw**

**new** IllegalAccessException(

"demonstrarea exceptiei");

}

**public** **static** **void** main(String[] args) {

**try** {

method();

}

**catch** (IllegalAccessException e) {

System.out.println("throw: " + e);

}}}

Uneori apare necesitatea de a efectua anumite acţiuni, indiferent dacă o excepţie a avut loc sau nu. În acest caz se foloseşte bloc **finally**, care este executat dupa blocu **try** sau **catch**. De exemplu:

**try** {/ \* codul care poate arunca o excepţie \* /}

**catch (Exception e)** {/ \* prelucrarea excepţie \* /}

**finally** {/ \* este executat sau după try, sau după catch \* /}

Fiecărui bloc **try** trebuie să-i corespundă cel puţin un bloc **catch** sau **finally**. Blocul **finally** este adesea folosit pentru a închiderea fişierelor şi eliberarea resurselor luate pentru uz temporar la începutul metodei. Codul blocului **finally** este executat, chiar dacă în faţa lui sunt instrucţiunile de returnare **return, break, continue**. De exemplu:

/ \* Exemplu # 5: de execuţie a blocului îfinally: \* /

**class** SampleFinally {

**static** **void** procA() {

**try** {

System.out.println("in metoda procA()");

**Throw** **new** RuntimeException(

"demonstratia exceptiei");

} **finally** {

System.out.println(

"blocul finally a metodei procA()");

} }

**static** **int** procB() {

**try** {

System.out.println("in metoda procB()");

**return** 1;

} **finally** {

System.out.println(

"blocul finally a metodei procB()");

**return** 0;

} }

**public** **static** **void** main(String[] args) {

**try** {

procA();

}

**catch** (Exception e) {/\*executia \*/}

**int** i = procB();} }

În metoda **procA()** din cauza excepţiei are loc ieşirea forţată din blocul **try**, dar înainte de a părăsi metoda, se realizează **finally**. Metoda **procB()** î-şi finalizază lucru realizînd operatorul return din blocul **try**, dar, înainte de a ieşi este executat de blocul **finally.**

* 1. **Exceptiile proprii**

Avem posibilitatea de a crea proprii excepţii ca o clasă derivată a clasei Exception şi apoi să le folosim în prelucrarea situaţiilor care apar la activarea metodei. De exemplu:

/\* exemplu # 6 : metodă, care generează o excepţie creată de programator: \*/

**public** **class** MyBasic {

**static** **int** result(**int** i)

**throws** MyResultException {

**try** {

i = 1 / i;

**throw** **new** MyResultException("Excepţia proprie");

}

**catch** (ArithmeticException e) {

**throw** **new** MyResultException(

"MyResultException bazată pe o altă excepţie ", e);

}}

**public** **static** **void** main(String[] args) {

**try** {

**int** a = result(8);

}

**catch** (MyResultException e) { System.out.println(e.toString());}}}

Atunci când este imposibil să se calculeze valoarea necesară se generează obiestul **ArithmeticException,** realizatorul căruia la rândul său generează exscepţia **MyResultException,** folosită ca o excepţie proprie, care are două argumente. Primul - un mesaj care poate fi afişat în fluxul de eroare, şi al doilea - o excepţie, care aduce la apelul excepţiei proprii. Acest cod arată cum se poate salva informaţii în o excepţiile proprii. Avantajul în acest caz este că dacă metoda activată va dori să ştie motivul real de apel pentru **MyResultException,** ea doar trebuie să apeleze metoda **getHiddenException().** Acesta permite metodei apelate să deschidă sau să lucreze cu o excepţie specifică sau este destul de prelucrat **MyResultException**  .

/\* exemplu # 7 :exceptie proprie: \*/

**public** **class** MyResultException **extends** Exception{

**private** Exception \_myException;

**public** MyResultException(String error, Exception e){

**super**(error);

\_myException = e;

}

**public** MyResultException(String error){

**super**(error);

}

**public** Exception getHiddenException(){

**return**(\_myException);

}}

**Lucrare de laborator nr. 5**

**1. Tema lucrării:**

Crearea excepţiilor

**2.**  **Scopul lucrării:**

* Însuşirea modalităţilor de creare şi realizare a excepţiilor în Java;

**3. Etapele de realizare:**

* 1. Realizarea mai multor tipuri de excepţii;
  2. Realizarea excepţiilor standarte ;
  3. Crearea excepţiilor poprii ;
  4. Crearea interfeţii programului;
  5. Prezentarea lucrării.

**4. Exemplu de realizare:**

import java.lang.RuntimeException;

import java.util.Stack;

public class JFrame extends javax.swing.JFrame {

public JFrame() {

initComponents();

}

@SuppressWarnings("unchecked")

private void initComponents() {

jScrollPane1 =

new javax.swing.JScrollPane();

jTextArea1 = new javax.swing.JTextArea();

jButton1 = new javax.swing.JButton();

jLabel1 = new javax.swing.JLabel();

setDefaultCloseOperation(

javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jTextArea1.setColumns(20);

jTextArea1.setRows(5);

jTextArea1.addKeyListener(

new java.awt.event.KeyAdapter() {

public void keyPressed(

java.awt.event.KeyEvent evt) {

jTextArea1KeyPressed(evt);

} }); jScrollPane1.setViewportView(jTextArea1);

jButton1.setText("Verifica");

jButton1.addMouseListener(

new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt){

jButton1MouseClicked(evt);

} });

jLabel1.setForeground(

new java.awt.Color(255, 51, 51));

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup( layout.createParallelGroup(

javax.swing.GroupLayout.Alignment.LEADING) .addGroup(layout.createSequentialGroup()

.addContainerGap().addGroup(layout.

createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING).addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 457, Short.MAX\_VALUE).addComponent(jButton1, javax.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jLabel1,javax.swing.

GroupLayout.PREFERRED\_SIZE, 392, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap()) );

layout.setVerticalGroup(layout.

createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING).addGroup (layout.createSequentialGroup().addGap(32, 32, 32).addComponent (jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 77, javax.swing.GroupLayout.PREFERRED\_SIZE) .addGroup(layout.createParallelGroup(

javax.swing.GroupLayout.Alignment.LEADING) .addGroup(layout.createSequentialGroup()

.addGap(33, 33, 33) .addComponent(jButton1)) .addGroup(layout.createSequentialGroup().addGap(18, 18, 18).addComponent(jLabel1))) .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)));

pack();

}// </editor-fold>

private void jButton1MouseClicked(java.awt.event.

MouseEvent evt) {

jLabel1.setText("");

Calcul c = new Calcul();

String s = "";

String s2 = jTextArea1.getText();

try{

boolean k = true;

while(k){

int l = c.funct(jTextArea1.getText()); s=""+s+jTextArea1.getText()+"\n"; jTextArea1.setText(""+c.afisare\_stack());

if (l==1){

k=false; s=""+s+jTextArea1.getText()+"\n";

}

c.c.removeAllElements();

}

jTextArea1.setText(s);

}

catch(Exception e ){

jLabel1.setText("Verificati datele introduse");

} }

private void jTextArea1KeyPressed(java.awt.event.KeyEvent evt) {try{ if(evt.getKeyCode()>=48 && evt.getKeyCode()<=57||evt.getKeyCode()==32){ jLabel1.setText(""+evt.getKeyChar());

}

else

throw new RuntimeException();

}

**catch**(RuntimeException e){

jLabel1.setText("Nu ati introdus numar !");

} }

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(

new Runnable() {

public void run() {

new JFrame().setVisible(true);

} }); }

private javax.swing.JButton jButton1;

private javax.swing.JLabel jLabel1;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JTextArea jTextArea1;

}

class Calcul {

Stack c = new Stack();

public int funct(String text){

String s[] = text.split(" ");

int j = 1;

for(int i=0;i<s.length;i=i+2,j=j+2){

if(j<s.length){

c.add((Integer.parseInt(s[i])+

Integer.parseInt(s[j])));

}

else{

if(i<s.length)

c.add(s[i]);

} }

return c.size();

}

public String afisare\_stack(){

String s="";

int j=1;

for(int i=0;i<c.size();i++,j++){

s = ""+s+c.elementAt(i);

if(j!=c.size())

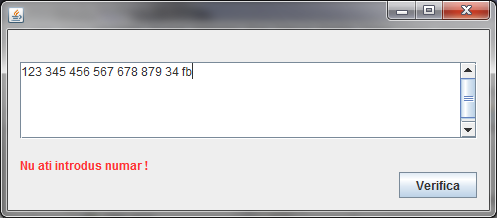
s=""+s+" ";

}

return s;

}}

Rezultatul realizării programului:



**5. Probleme propuse:**

De realizat variantele date folosind exceptiile.

1. Sa se citeasca cite un rind dintr-un fişier şi să se înscrie in stivă. Apoi rîndurile din stivă de înscris în fişier în ordine inversă.
2. Introduceţi un număr, sub forma de cifre în stivă. Afişaţi numărul la care cifrele sunt inversate.
3. Sa se creeze o stivă complectată cu cîmpurile înregistrării. Organizaţi acces direct la elementele înregistrării.
4. Să se creeze şi comlecteze două stive , şi să se schimbe informaţia intre stive cu locul.
5. Definiţi clasa Stack. Întroduceţi o secvenţă de caractere şi

afisaţ-o în ordine inversă

1. Creati o clasa Queue. Introduce o serie de siruri de caractere, să se verifece daca string-ul dat este unic in coadă.
2. Folosind clasa Set, pentru un şir de numere întregi să se creeze metode care determină concatinarea şi intersecţia multimii.
3. Se întroduc N parametri de la consolă. Aceşti parametri -sunt elemente ale vectorului detip double, şi in baza vectorului – se creaza clasa cu obiectul DoubleVector. Apoi, programul afisiaza la consola valorile vectorului în formă: Vector: 2.3 5.0 7.3.
4. Listele I (1..N) şi U (1..N), conţin rezultatele a N măsurări de current şi tensiune pentru o rezistenţă necunoscută R. Apreciaţi valuarea aproximativa a lui R.
5. Efectuaţi o sumare pereche pentru orice secvenţă finită de numere, după cum urmează: în prima etapă, se adună căte două numerele. La a doua etapă, se adună rezultatele primei etape şi etc. până când nu rămîne un număr. Dacă în orice etapă, dacă rămîne număr fără pereche, el trece în etapa următoare.
6. Efectuaţi înmultirea a două polinoame de grad fix, în cazul în care coeficienţii polinoamelor sunt stocaţi în liste.
7. Ne utilizind facilităţile obiectelor adăugătoare, aranjati elementele negative ale listei la sfirsit, iar cele pozitive - la inceputul acestei liste, păstrînd ordinea.