Componente specifice interfețelor grafice, gestionarea evenimentelor

Graphical user interfaces components, events handling

Obiective:

- implementarea intefețelor grafice;
- utilizarea mecanismelor de gestionare a evenimentelor

Objectives:

- implementing graphical user interfaces;
- using the event handling mechanisms;

Componente Java

Componentele sunt blocuri funcţionale care servesc la construirea interfeţelor grafice destinate utilizatorilor. Există 3 categorii de componente:

· componente vizuale care servesc direct la construirea GUI (butoane, liste de selecţie, etc.)
· componente de tip container care servesc la organizarea spaţială a celor din prima categorie (sunt derivate direct din clasa Container)
· componente de tip meniu

Există câteva metode de bază care sunt comune oricărei clase care gestionează componente (mai puţin meniurile), indiferent de locul în care se încadrează acestea în categorizarea de mai sus. Aceste metode sunt moştenite din clasa *java.awt.Component*. (https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html)

- *Dimension getsize()*

=> returnează un obiect de tip *Dimension* care conţine lăţimea şi înălţimea componentei

- *void setBackground(Color c)*

=> setează culoarea de fundal a componentei

- *void setForeground(Color c)*

=> setează culoarea de faţadă a componentei (de obicei culoarea textului care este afişat pe componenta)

- *void setFont(Font f)*

=> setează fontul cu care vor fi afişate caracterele care compun textul afişat pe componentă

- *void setEnabled(boolean b)*

=> dacă parametrul primit este *false*, componenta este afişată cu tonuri de gri şi nu răspunde acţiunilor utilizatorului (este dezactivată)

- *void setSize(Dimension d)* sau *void setSize(int width, int height)*

=> setează mărimea (lăţimea şi înălţimea) componentei curente

- *void setBounds(int x, int y, int width, int height)*

=> setează mărimea (lăţimea şi înălţimea) şi poziţia pe ecran (colţul din stânga sus) a componentei curente

- *void setVisible(boolean b)*

=> specifică dacă componenta curentă este vizibilă sau nu

- *etc.*

Clasele de tip meniu sunt derivate din *java.awt.MenuComponent*.
(https://docs.oracle.com/javase/8/docs/api/java/awt/MenuComponent.html)

Button https://docs.oracle.com/javase/8/docs/api/java/awt/Button.html

Instanţele acestei clase reprezintă butoane care pot fi ”apăsate” de către utilizator. Pe butoate pot fi afişate etichete (informaţie textuală) setate în momentul apelării constructorului clasei sau ulterior.

Când un buton este apăsat, generează un *ActionEvent*.

Canvas https://docs.oracle.com/javase/8/docs/api/java/awt/Canvas.html

Componentele de acest tip nu au aspect şi comportament implicit. Obiectele de acest tip sau instanțiate din subclase ale *Canvas* servesc la definirea unor zone de desenare, a unor zone de lucru pentru a grupa componentele incluse, etc.

Obiectele de tip *Canvas* trimit evenimente de tip *MouseEvent*, *MouseMotionEvent*, *KeyEvent.*

Checkbox https://docs.oracle.com/javase/8/docs/api/java/awt/Checkbox.html

O astfel de componentă este un buton cu două stări (bifat sau ne-bifat).

Prin intermediul metodelor

boolean getState() şi
void setState(boolean state)

se poate citi, respectiv scrie starea butonului.

Dacă mai multe butoane de acest tip sunt grupate prin intermediul unei componente de tip *CheckboxGroup*, ele capătă funcţionalităţi de butoane radio (apăsarea unui buton deselectează orice alt buton apăsat).

Exemplu:

```
CheckboxGroup cbg = new CheckboxGroup();
add(new Checkbox(”Articol 1”, false, cbg));
add(new Checkbox(”Articol 2”, false, cbg));
add(new Checkbox(”Articol 3”, true, cbg));
```

Dintr-o astfel de grupare, se poate identifica care buton este apăsat folosind metoda

Checkbox getSelectedCheckbox()

şi se poate seta componenta bifata cu funcţia

void setSelectedCheckbox(Checkbox newSelection)

Componentele de tip *Checkbox* trimit evenimente de tip *ItemEvent*.

Choice https://docs.oracle.com/javase/8/docs/api/java/awt/Choice.html

Componentele de tip *Choice* oferă utilizatorului posibilitatea selectării unui articol dintr-o lista expandabilă de opţiuni posibile. Lista de opţiuni poate fi populată prin apelarea repetată a funcţiei *addItem()*.

Exemplu:

```
Choice ch = new Choice();
ch.add(”Articol 1”);
ch.add(”Articol 2”);
ch.add(”Articol 3”);
```

Mecanismele de selectare şi de citire a articolelor selectate sunt similare cu cele prezentate la *CheckboxGroup*. Componentele *Choice* generează evenimente de tip *ItemEvent.*

FileDialog https://docs.oracle.com/javase/8/docs/api/java/awt/FileDialog.html

Această clasă pune la dispoziţie mecanismul de creare a unei ferestre de dialog dedicată salvării sau selectării unui fişier. Aspectul acestei ferestre diferă mult în funcţie de sistemul de operare pe care rulează mediul Java.

Există 2 constante simbolice: *FileDialog.LOAD* şi *FileDialog.SAVE* care codează modul de funcţionare al ferestrei de dialog.

După ce utilizatorul a specificat un fişier sau un director, numele acestuia poate fi citit cu funcţiile

String getFile()
String getDirectory()

Exemplu:

```
FileDialog fd = new FileDialog(f, ”Alege un fişier”, FileDialog.LOAD);
fd.setVisible(true);
System.out.println(”A fost ales fişierul: ”+fd.getFile());
```

Label https://docs.oracle.com/javase/8/docs/api/java/awt/Label.html

Acest tip de componente este cel mai simplu din punct de vedere al implementării sau funcţionalităţilor oferite. Un obiect *Label* conţine o informaţie textuala care poate fi afişată pe ecran.

Metodele pentru setarea şi preluarea textului etichetei sunt

void setText(String newText) şi
String getText()

List https://docs.oracle.com/javase/8/docs/api/java/awt/List.html

O listă este o colecţie de texte, aranjate vertical unele sub altele. Dacă lista conţine mai multe elemente decât poate afişa, va fi dotată automat cu o componentă de tip *scrollbar* vertical.

Clasa *List* are o mulţime de metode care ajuta programatorul să controleze conţinutul, iar dintre acestea mai importante ar fi:

void add(String text) => adaugă un nou rând în listă
void add(String text, int index) => adaugă un nou rând în listă, pe poziţia specificată
String getItem(int index) => returnează textul de pe rândul primit ca parametru
int getItemCount() => returnează numărul de articole din listă
int getRows() => returnează numărul de rânduri vizibile din listă
int getSelectedIndex() => returnează indexul elementului selectat
int[] getSelectedIndexes() => returnează un şir de indecşi ale elementelor selectate (dacă lista permite selecţia multiplă)
String getSelectedItem() => returnează textul elementului selectat
String[] getSelectedItems() => returnează un şir de tip *String* care conţine textele elementelor selectate (în listele cu selecţie multiplă)

Obiectele de tip *List* generează evenimente de tip *ItemEvent*.

ScrollPane https://docs.oracle.com/javase/8/docs/api/java/awt/ScrollPane.html

Acest tip de componentă poate conţine orice altă componentă, care poate depăşi limitele *ScrollPane*-ului. În funcţie de politica aleasă, componenta *ScrollPane* poate pune la dispoziţie scrollbar-uri verticale şi orizontale care să ajute utilizatorul în vizualizarea completă a conţinutului.

Exemplu:

```
ScrollPane sp = new ScrollPane();
Label eticheta = new Label("AAABBBCCCDDDEEEFFFGGGHHHIIIJJJ");
eticheta.setFont(new Font("Şerif", Font.BOLD, 80));
sp.add(eticheta );
```

Scrollbar https://docs.oracle.com/javase/8/docs/api/java/awt/Scrollbar.html

Acest tip de componentă serveşte la ajustarea poziţiei conţinuturilor unor componente, în cazul în care acesta este mai mare ca dimensiune decât spaţiul pus la dispoziţie. Poziţia implicită a unui *scrollbar* este verticală.

TextField* şi *TextArea
https://docs.oracle.com/javase/8/docs/api/java/awt/TextField.html
https://docs.oracle.com/javase/8/docs/api/java/awt/TextArea.html

O componentă de tip *TextField* este folosită pentru afişarea textelor (editabile sau nu de către utilizator) pe o singură linie. Câmpurile *TextArea* pot avea mai multe linii de text.

Pentru a citi/seta un text în cadrul componentei se folosesc metodele

String getText(), *String getSelectedText()* şi
Void setText(String newText).

Metoda *void setEditable(boolean editable)* specifică dacă textul componentei respective poate fi modificat de utilizator sau nu.
În cadrul componentelor de tip *TextArea* se pot introduce caractere *escape*, cum ar fi *n* sau \t.

Exemplu:

```
TextField tf1 = new TextField(5);
tf1.setFont(new font("SansSerif", Font.PLAIN, 8));
tf1.setText("abcdefg");

int val = 7;

TextArea ta1 = new TextArea(3, 20);
ta1.setText("aaa\nbbb\nccc"+val);
ta1.append(tf1.getText());
```

Componente de tip *Container*

Componentele de tip container sunt următoarele:

- *Applet*
- *Frame*
- *Panel*
- *Dialog*

Gestionarea (tratarea) evenimentelor - Delegarea evenimentelor (*Delegation Event Model*)

Conceptul care stă în spatele acestui model de tratare a evenimentelor are următorul principiu: *o sursă* (*event source*) generează un eveniment şi îl trimite către unul sau mai mulţi ascultători (*listeners*). În acest mod, ascultătorii aşteaptă în permanenţă apariţia unui eveniment şi odată ce îl primeşte, efectuează toate acţiunile prevăzute prin cod şi apoi intră din nou în starea de aşteptare.

Un *eveniment* reprezintă o schimbare a stării unui obiect sursă (de evenimente). Evenimentele pot apărea de regulă ca urmare a acţiunilor operatorului uman care interacţionează cu componentele grafice ale unei aplicaţii Java (apăsări de butoane, introduceri de text în zonele dedicate, schimbări ale articolelor selectate dintr-o listă, redimensionări de ferestre, mişcări ale mouse-ului,etc.). Există şi situaţii în care evenimentele apar independent de acţiunile operatorului uman (de exemplu la expirarea timpului setat pe un cronometru, anumite componente grafice pot suferi modificări controlate prin cod).

O *sursă de evenimente* este un obiect care de regulă este o instanţă a unei clase care implementează componentele grafice de interacţiune cu utilizatorul (butoane, ferestre, meniuri, etc.). Când aceste componente îşi schimba starea, ele generează evenimente care apoi sunt procesate de ascultătorii specifici care le sunt ataşaţi. Fiecare tip posibil de eveniment are o metodă specifică care permite ataşarea acestuia de o componentă în vederea monitorizării şi prelucrării ulterioare a evenimentului apărut.

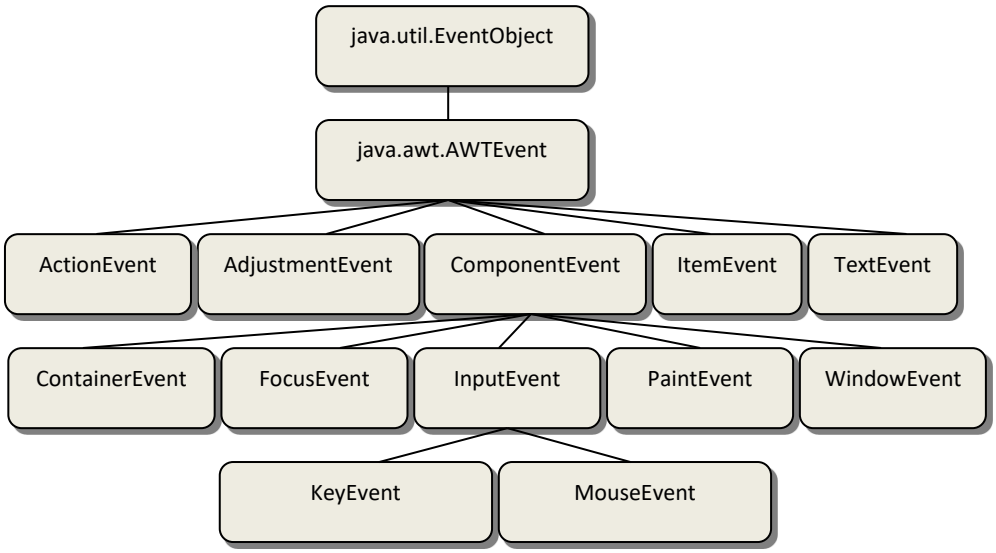
Forma generală a unei astfel de metode este următoarea:

Sintaxa:

public void addTipListener(TipListener obiect)
public void removeTipListener(TipListener el)

Clase care se ocupa de gestionarea evenimentelor

Există o mare varietate de clase care se ocupa cu tratarea evenimentelor şi care reprezintă baza mecanismelor în discuţie. Ele încapsulează o serie de mijloace uşor de accesat care permit gestionarea evenimentelor specifice.



Clasa *java.util.EventObject*

Clasa de bază a întregii ierarhii de mai sus este *EventObject*.

Metoda constructor

- *EventObject(Object src)*

=> *src* reprezintă sursa evenimentului

Metode membre principale

- *Object getSource( )*

=> returnează sursa evenimentului

- *String toString( )*

=> returnează echivalentul textual al evenimentului

Clasa *java.awt.AWTEvent*

Este derivată din *java.util.EventObject* iar toate clasele derivate din ea se ocupă cu evenimente date de componentele de tip AWT (*Abstact Window Toolkit*) de interacţiune cu utilizatorul.

Metode membre principale

- *int getID( )*

=> returnează ID-ul evenimentului curent. Prin codificarea de tip *int* a ID-ului se specifică exact natura evenimentului. Există o serie de constante simbolice asociate acestor ID-uri pentru o utilizare mai facilă (de ex. *MouseEvent.MOUSE_DRAGGED*, etc.).

Sub-clasele *java.awt.AWTEvent* sunt prezentate în continuare.

- *ActionEvent*

=> se ocupă cu evenimentele care apar la apăsarea butoanelor, la selectarea unui articol dintr-o listă sau dintr-un meniu

- *AdjustmentEvent*

=> se ocupă cu evenimentele care apar la manipularea scroll-bar-urilor

- *ComponentEvent*

=> se ocupă cu evenimentele care apar când o componentă este ascunsă/făcută vizibilă, mişcată sau redimensionată

- *ContainerEvent*

=> se ocupă cu evenimentele care apar când o componentă este adăugată sau scoasă dintr-un container

- *FocusEvent*

=> se ocupă cu evenimentele care apar când o componentă primeşte sau pierde focus-ul

- *InputEvent*

=> clasa abstractă de bază pentru toate clasele care tratează evenimentele de intrare

- *ItemEvent*

=> se ocupă cu evenimentele care apar la selectarea unei căsuţe de selecţie sau a unui articol dintr-o listă

- *KeyEvent*

=> se ocupă cu evenimentele care apar la apăsarea tastelor

- *MouseEvent*

=> se ocupă cu evenimentele care apar la acţionarea mouse-ului (mişcare, click, dublu-click, etc.)

- *PaintEvent*

=> se ocupă cu evenimentele care apar la desenarea unei componente pe ecran

- *TextEvent*

=> se ocupă cu evenimentele care apar la modificarea componentelor de tip text

- *WindowEvent*

=> se ocupă cu evenimentele care apar la acţionarea ferestrei unui program (iconificare, de-iconificare, etc.)

Interfețe de tip Listener

- *ActionListener*

=> defineşte metoda *public void actionPerformed(ActionEvent ae)* pentru a trata evenimente de tip acţiune

=> metoda de asociere cu o sursă de evenimente este *addActionListener()*

- *AdjustmentListener*

=> defineşte metoda *public void adjustmentValueChanged(AdjustmentEvent ae)* pentru a trata evenimente de tip ajustare a dimensiunilor grafice
=> metoda de asociere cu o sursă de evenimente este *addAdjustmentListener()*

- *ComponentListener*

=> defineşte metodele *public void componentHidden(ComponentEvent ce), public void componentMoved(ComponentEvent ce), public void componentResized(ComponentEvent ce), public void componentShown(ComponentEvent ce),* pentru a trata evenimente de tip redimensionare, ascundere/afişare sau mutare efectuate pe componente de interacţiune cu utilizatorul
=> metoda de asociere cu o sursă de evenimente este *addComponentListener()*

- *ContainerListener*

=> defineşte metoda *public void componentAdded(ContainerEvent ce)* pentru a trata evenimente legate de adăugarea/scoaterea componentelor din containere
=> metoda de asociere cu o sursă de evenimente este *addContainerListener()*

- *FocusListener*

=> defineşte metodele *public void focusGained(FocusEvent fe)* şi *public void focusLost(FocusEvent fe)* pentru a trata evenimente de primire sau pierdere a focus-ului
=> metoda de asociere cu o sursă de evenimente este *addFocusListener()*

- *ItemListener*

=> defineşte metoda *public void itemStateChanged(ItemEvent ae)* pentru a trata evenimente de schimbare a stării sursei de evenimente
=> metoda de asociere cu o sursă de evenimente este *addItemListener()*

- *KeyListener*

=> defineşte metodele *public void keyPressed(KeyEvent ke), public void keyReleased(KeyEvent ke), public void keyTyped(KeyEvent ke),* pentru a trata evenimente de tastatură
=> metoda de asociere cu o sursă de evenimente este *addKeyListener()*

- *MouseListener*

=> defineşte metodele *public void mouseClicked(MouseEvent me), public void mouseEntered(MouseEvent me), public void mouseExited(MouseEvent me), public void mousePressed(MouseEvent me), public void mouseDragged(MouseEvent me)* pentru a trata evenimente de mouse
=> metoda de asociere cu o sursă de evenimente este *addMouseListener()*

- *MouseMotionListener*

=> defineşte metodele *public void mouseDragged(MouseEvent me), public void mouseMoved(MouseEvent me)* pentru a trata evenimente de mişcare a mouse-ului
=> metoda de asociere cu o sursă de evenimente este *addMouseMotionListener()*

- *MouseWheelListener*

=> defineşte metoda *public void mouseWheelMoved(MouseWheelEvent mwe* pentru a trata evenimente de mişcare a rotii mouse-ului
=> metoda de asociere cu o sursă de evenimente este *addMouseWheelListener()*

- *TextListener*

=> defineşte metoda *public void textValueChanged(TextEvent te)* pentru a trata evenimente de schimbare a conţinutului componentelor de tip text
=> metoda de asociere cu o sursă de evenimente este *addTextListener()*

- *WindowFocusListener*

=> defineşte metodele *public void windowGainedFocus(WindowEvent we), public void windowLostFocus(WindowEvent we)* pentru a trata evenimente de primire/pierdere a focus-ului pe componente de tip *Window*
=> metoda de asociere cu o sursă de evenimente este *addWindowFocusListener()*

- *WindowListener*

=> defineşte metodele *public void windowActivated(WindowEvent we), public void windowClosed(WindowEvent we), public void windowClosing (WindowEvent we), public void windowDeactivated(WindowEvent we), public void windowDeiconified(WindowEvent we), public void windowIconified (WindowEvent we), public void*

windowOpened(WindowEvent we) pentru a trata evenimente legate de componente de tip *Window*
=> metoda de asociere cu o sursă de evenimente este *addWindowListener()*

Clase de tip *Adapter*

Mecanismul de folosire a interfeţelor de tip *listener* poate fi dificil uneori, datorită faptului că unele interfeţe prevăd mai multe metode decât ar putea fi necesare într-un anumit program. De exemplu, este posibil ca un cod să aibă nevoie doar de metoda de iconificare a unei ferestre din interfaţa *WindowListener*. Dacă clasa nu defineşte cel puţin o implementare vidă şi pentru restul metodelor din interfaţă (lucru inutil dar necesar), ea trebuie declarată abstractă şi deci este neinstanțiabilă.

Nume clasă	**Nume interfaţă**
ComponentAdapter	*ComponentListener*
ContainerAdapter	*ContainerListener*
FocusAdapter	*FocusListener*
KeyAdapter	*KeyListener*
MouseAdapter	*MouseListener*
MouseMotionAdapter	*MouseMotionListener*
WindowAdapter	*WindowListener*

Gestionarea (tratarea evenimentelor) - Crearea de componente capabile să îşi gestioneze evenimentele (*Inheritance Event Model with Masks*)

Alt mecanism prevăzut de mediul Java pentru tratarea evenimentelor se referă la derivarea claselor care controlează componentele grafice de interacţiune cu utilizatorul şi forţarea obiectelor instanțiate din clasele derivate să se ocupe de evenimentele pe care le generează. Acest lucru se realizează prin suprascrierea metodelor care delegă evenimentele la ascultători (*listeners*).

Metodele care trebuie suprascrise sunt în legătură cu o serie de constante simbolice numite *măşti* care identifică în mod unic fiecare tip de eveniment. Tabelul de corespondenţă dintre măşti şi numele metodelor este dat mai jos.

Masca	**Numele metodei**
AWTEvent.ACTION_EVENT_MASK	*public void processActionEvent(ActionEvent ae)*
AWTEvent.ADJUSTMENT_EVENT_MASK	*public void processAdjustmentEvent(AdjustmentEvent ae)*
AWTEvent.COMPONENT_EVENT_MASK	*public void processComponentEvent(ComponentEvent ce)*
AWTEvent.CONTAINER_EVENT_MASK	*public void processContainerEvent(ContainerEvent ce)*
AWTEvent.FOCUS_EVENT_MASK	*public void processFocusEvent(FocusEvent fe)*
AWTEvent.ITEM_EVENT_MASK	*public void processItemEvent(ItemEvent ie)*
AWTEvent.KEY_EVENT_MASK	*public void processKeyEvent(KeyEvent ae)*
AWTEvent.MOUSE_EVENT_MASK	*public void processMouseEvent(MouseEvent me)*
AWTEvent.MOUSE_MOTION_EVENT_MASK	*public void processMouseMotionEvent(MouseMotionEvent me)*
AWTEvent.TEXT_EVENT_MASK	*public void processTextEvent(TextEvent te)*
AWTEvent.WINDOW_EVENT_MASK	*public void processWindowEvent(WindowEvent we)*

Lucru individual

1. Creați o aplicație ce preia din două câmpuri text (*TextField())* numele vostru și grupa din care faceți parte și afișează această informație într-o etichetă de culoare RGB(122,123,129).

==========

2. Creați o aplicație Java ce include un 3 elemente de tip slider, prin intermediul cărora se poate seta culoarea unui pătrat de dimensiunea 100x100.

==========

3. Scrieți o aplicație Java ce include un formular de înregistrare ca și student la un curs online. Formularul include informații referitoare la nume, prenume, an de studii, facultatea, finanțare taxă/buget și cursul dorit. Anul de studii, facultatea și cursul sunt disponibile ca și listă de opțiuni, iar finanțarea este de tip checkbox. Într-un câmp de tip TextArea afișați informația completată de student ca urmare a apăsării butonului de înregistrare.

==========

4. Implementați un applet Java care desenează un labirint din pătrate. Unul din pătrate (intrarea în labirint) este colorat distinct. Folosiți cursorul pentru a deplasa pătratul înspre ieșire. Orice apăsare necorespunzătoare de tastă resetează poziția pătratului colorat și îl aduce în poziția inițială. Să se afișeze un mesaj corespunzător atunci când labirintul a fost rezolvat.

* În loc de cursor folosiți poziția mouse-ului pentru a parcurge labirintul. Atingerea marginilor labirintului sau ieșirea din acesta, resetează jocul.

==========

5. Creați a aplicație ce simulează aruncarea a două zaruri. Aplicația va include un buton cu eticheta ”Aruncă” și va desena cele două zaruri în fereastra grafică.

* Împărțiți fereastra în două zone, astfel încât să permită un joc între doi utilizatori. Cel care obține suma zarurilor mai mare câștigă runda. Contorizați numărul de partide câștigate, pierdute și terminate la egalitate pentru fiecare dintre cei doi utilizatori.

** Eliminați butoanele și aruncați zarurile atunci când utilizatorul dă click în zona sa de joc.

==========

6. Creați o aplicație ce măsoară timpul de reacție al utilizatorului. Aplicația generează aleator 10 cercuri de culoare roșie și neagră. Applet-ul conține două butoane cu etichetele ROȘU și NEGRU. Utilizatorul trebuie să apese butonul corect. La final aplicația va afișa numărul de răspunsuri corecte și timpul mediu de reacție al utilizatorului (folosiți metoda *System.currentTimeMillis()*).

==========

7. Să se deseneze o serie de fulgi de zăpadă. Desenai fractalul Koch până la o dimensiune specificată ca și parametru HTML. Detalii despre fractal găsiți aici: http://en.wikipedia.org/wiki/Koch_snowflake. Animația trebuie să prezinte fiecare iterație a fractalului.

Individual work

1. Create an application which takes from two text fields your name and the group you are part of and displays this info in a label colored in RGB(122,123,129).

==========

2. Write a Java application which includes 3 sliders which enables the user to set the color of a 100x100 square.

==========

3. Write a Java application which includes a sign-up form for an online course. The form includes information regarding the name, surname, year of study, faculty, financing (tax/budget) and the course. The year of study, faculty and course are drop-down lists, and the financing is a check-box field. În a TextArea field print the filled-in information after the Sign-up button is pressed.

==========

4. Implement a Java applet which draws a maze made up of squares. One of the squares (the entry points) is colored in a distinct color. Use the cursor to guide the square to the exit. Any mistake resets the position of the square and sets it back to its original position. Display a message when the square reached the exit point.

* Instead of the cursor, use the mouse's position to navigate through the maze. Touching the edges of the maze or exiting it, resets the game.

==========

5. Write an application which simulates a dice roll. The application will include a button labelled "Roll dice" which triggers the event of drawing two dice in the app's window.

* Split the window into 2 areas, so that two players can play the game. The players with the highest dice score wins the round. Count the number of games won, lost and finished with a draw by each player.

** Remove the buttons and roll the dice when the user clicks its game area.

==========

6. Build an application which measures the user's response time. The application randomly generates 10 red and black circles. The application includes two buttons labelled RED and BLACK. The user must press the correct button. At the end of the game, the app will display the number of correct answers and the user's average reaction time (use the method System.currentTimeMillis()).

==========

7. You need to draw several snowflakes. Draw the Kock Snowflake fractal up to a dimension specified in the HTML file. Details of the fractal can be found here: http://en.wikipedia.org/wiki/Koch_snowflake. The animation should display each iteration of the fractal.