1. Platformă online de găzduire web GIT. Timer-ul.

Git este un sistem de control al versiunilor distribuit gratuit, open-source cod sursă, conceput pentru a lucra rapid și eficient cu orice proiecte - de la mici la foarte mari. Git este ușor de învățat și ocupă puțin spațiu și are performanțe fulgerătoare. Este superior unor astfel de SCM-uri (Supply Managementul lanțului – trad. Supply Chain Management - instrumente precum Subversion, CVS, Perforce și ClearCase, cu caracteristici precum local low-cost ramificații, zone de depozitare convenabile și fluxuri de lucru multiple.

1.1 Ramificare și contopire

Caracteristica Git care îl face cu adevărat să iasă în evidență de aproape toți ceilalți alte SCM-uri, este modelul de ramificare. Git permite și încurajează creația mai multe ramuri locale, care pot fi complet independente unele de altele prieten. Crearea, îmbinarea și ștergerea acestor ramuri durează doar câteva secunde.

Aceasta înseamnă că puteți face lucruri precum:

• Comutare de context fără contact. Creați o ramură pe care să o încercați orice idee, fă câteva comite, întoarce-te la locul de unde ramificație a fost făcută, aplicați remedierea, întoarceți-vă unde au fost efectuate experimente și combinate.

• Linii de cod pentru jocuri de rol. Ai o ramură care conține întotdeauna doar ce trimis la producție, altul, în care munca este fuzionată pentru testare și câteva ramuri mici pentru munca de zi cu zi.

• Flux de lucru bazat pe funcții. Creați filiale noi pentru fiecare funcție nouă la care lucrați, astfel încât să puteți face fără probleme comuta între ele și apoi șterge fiecare ramură când funcția vor fi îmbinate cu linia principală.

• Experimente unice. Creați un fir pentru experimentare, înțelegeți asta nu funcționează și șterge-l, părăsește-ți jobul și nimeni nu o va mai vedea ea (chiar dacă în acest timp ai promovat alte ramuri).

1.2 Distribuire

Una dintre cele mai utile caracteristici ale oricărui SCM distribuit este inclusiv Git, este distribuția sa. Aceasta înseamnă că în loc de efectuați o „verificare” a vârfului codului sursă curent, efectuați o „clonă” întregul depozit.

1.3 Copii de rezervă multiple

Aceasta înseamnă că, chiar dacă utilizați un flux de lucru centralizat, fiecare utilizator are în esență o copie de rezervă completă a serverului principal.

Fiecare dintre aceste copii poate fi ridicată pentru a înlocui serverul principal în caz de defecțiune sau deteriorare. În esență, nu există un singur punct de eșec în Git decât dacă există o singură copie a depozitului.

1.4 Flux de lucru

Datorită naturii distribuite a lui Git și sistemului excelent de ramificare este posibil să se implementeze cu relativă uşurinţă un număr practic infinit procesele de lucru.

1.5 Flux de lucru în stil subversiune (Subversion)

Fluxul de lucru centralizat este foarte comun, mai ales în rândul utilizatorilor, care au trecerea dintr-un sistem centralizat. Git nu te va lăsa să realizați push dacă cineva a făcut un push de la ultima extracție, deci centralizat funcționează modelul în care toți dezvoltatorii realizează push pe același server pur si simplu minunat.

1.6 Instalarea Git

Descărcați programul de instalare (https://git-scm.com/downloads), rulați programul de instalare,selectați opțiunile de care aveți nevoie (sau lăsați totul așa cum este, dacă nu prea mult înțelegeți), așteptați finalizarea procesului de instalare și ați terminat.

1.7 Utilizare Git

În folderul de lucru în care se află fișierele sursă ale aplicației/proiectului, prin terminal sau linia de comandă, utilizați comanda git init (documentația oficială - https://git-scm.com/docs/git-init) pentru a inițializa depozitul local, după ce Git va urmări modificările la fișierele sursă și le va remedia modificările - utilizați comanda git commit (documentație oficială - https://git scm.com/docs/git-commit).

În viitor, dacă trebuie să sincronizați modificările în comun pentru toate depozitele - utilizați comanda git push

2 . Planificarea activităţii proceselor.

1.1. Timer-ul

Timer-ul este un dispozitiv care notifică periodic aplicaţia că a expirat o anumită perioadă de timp prestabilit. Programul specifică intervalul de timp pe care îl trimite timerului indicând intervalul de expirare. Timerul este important pentru efectuarea multor aplicaţii.

1.1.2. Utilizarea timerului

Pentru a ataşa un timer unui program trebuie să utilizăm clasa Timer, care oferă facilitatea de a planifica diverse acţiuni pentru a fi realizate la un anumit moment de către aplicaţia, care rulează.

Acţiunile unui obiect de tip Timer sânt implementate ca instanţe ale clasei TimerTask şi pot fi programate pentru o singură execuţie sau pentru execuţii repetate la intervale regulate de timp. Pentru folosirea unui timer trebue executaţi următorii paşi:

- crearea unei sub-clase acţiune a clasei TimerTask şi suprascrierea metodei run() ce va conţine acţiunea planificată. Pot fi folosite și clase anonime;
- crearea unui fir de execuţie prin instanţierea clasei Timer;
- crearea unui obiect de tip acţiune;
- planificarea la execuţie a obiectuluii de tip acţiune, folosind metoda schedule() din clasa Timer.

Metoda de planificare pe care o putem utiliza are următoarele formate(este supraîncarcată):

```
schedule(TimerTask task, Date time)
schedule(TimerTask task, long delay, long period)
schedule(TimerTask task, Date time, long period)
scheduleAtFixedRate(TimerTask task, long delay, long period)
scheduleAtFixedRate(TimerTask task, Date time, long period)
```

unde:

task - descrie acţiunea ce se va executa;

delay- reprezintă întârzierea faţă de momentul curent după care va începe execuţia;

time - momentul exact la care va începe execuţia;

period - intervalul de timp dintre două execuţii.

Metodele de planificare se împart în două categorii:

- schedule() - planificare cu întârziere fixă: dacă dintr-un anumit motiv acţunea este întârziată, următoarele acţiuni vor fi şi ele întârziate în consecinţă;
- scheduleAtFixedRate() - planificare cu număr fix de rate: dacă acţiunea este întârziată, următoarele acţiuni vor fi executate mai repede, astfel încât numărul total de acţiuni dintr-o perioadă de timp să fie tot timpul acelaşi.

Un timer se va opri la expirarea metodei sale run(), dar tot odată poate fi oprit forţat, folosind metoda cancel(). După oprire acesta nu va mai putea fi folosit pentru planificarea altor acţiuni. De asemenea, prin metoda System.exit() se opresc forţat toate firele de execuţie şi se termina aplicaţia curentă.

Exemplul 1: folosirea claselor Timer şi TimerTask

```
import java.util .*;
import java.awt .*;
class Atentie extends TimerTask {
public void run () {
Toolkit.getDefaultToolkit(.beep();
System.out.print(".");
}}
class Alarma extends TimerTask {
public String mesaj ;
public Alarma(String mesaj) {
this.mesaj = mesaj ;
}
public void run() {
System.out.println( mesaj );
}}
public class TestTimer {
public static void main(String args []) {
// Setăm o acţiune repetitivă, cu rată fixă
final Timer t1 = new Timer ();
t1.scheduleAtFixedRate(
new Atentie(), 0, 1*1000) ;
// Clasă anonima pentru o altă acţiune
Timer t2 = new Timer ();
t2. schedule ( new TimerTask() {
public void run() {
System.out.println("Au trecut 10 secunde");
// Oprim primul timer
t1. cancel();
}}, 10*1000) ;
// Setăm o acţiune pentru ora 13:15
Calendar calendar = Calendar.getInstance();
calendar.set( Calendar . HOUR_OF_DAY, 13);
calendar.set( Calendar .MINUTE, 15);
calendar.set( Calendar .SECOND, 0);
Date ora = calendar.getTime();
Timer t3 = new Timer();
t3.schedule(new Alarma "Ora mesei!"), ora );
}}
```

Rezultatul realizării programului:

```
init:
deps-jar:
Compiling 1 source file to C:\Documents and Settings\User\JavaApplication2\build\classes
compile-single:
```

run-single:
.....Au trecut 10 secunde
Ora mesei!

Exemplul următor contine o fereastră cu trei butoane şi cu trei obiecte de tip Timer. Fiecare obiect Timer controlează un buton, adică la trecerea intervalului specific fiecărui timer se incrementează numărul de pe butonul corespunzător. La crearea obiectelor de tip Timer specificăm intervalul de timp la care acţionează respectiv şi obiectul listener, adică componenta care este notificată. Timerul trimite mesajul actionPerformed() după trecerea intervalului de timp. Obiectul Timer trebuie pornit prin accesarea metodei start().

Exemplul 2: Realuzarea unei fereastre cu trei butoane şi cu trei obiecte de tip Timer.

```
import javax.swing.JFrame;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.Timer;
import java.awt.event.ActionEvent;

public class MyWindow extends JFrame implements ActionListener {
        private JButton b[];
        private Timer t[];

        public MyWindow() {
                final int n = 3;
                int i;
   getContentPane().setLayout( new java.awt.FlowLayout() );
                b = new JButton[ n ];
                for(i=0;i<n;i++){
             b[i] = new JButton(Integer.toString( i ));
    getContentPane().add( b[ i ] );
                }
   addWindowListener( new java.awt.event.WindowAdapter(){
    public void windowClosing
(java.awt.event.WindowEvent e )
        {
        setVisible( false );
        System.exit( 0 );
                        }});
                t = new Timer[ n ];
                for(i=0;i<n;i++){
    t[ i ] = new Timer( (i+1)*100, this );
        t[ i ].start();
                }}
   public static void main(String[] args) {
                MyWindow w = new MyWindow();
                w.setBounds(1,1,400,300);
                w.setVisible( true );
        }

        public void actionPerformed(ActionEvent arg0) {
   if( arg0.getSource() == t[0] ){
       System.out.println( "Button: "+0);
```

```
        b[0].setLabel(Integer.toString(
        Integer.parseInt(b[0].getLabel())+1));
                  }
                  else
          if( arg0.getSource() == t[1] ){
         System.out.println( "Button: "+1);
        b[1].setLabel( Integer.toString(
      Integer.parseInt(b[1].getLabel())+1 ));
                        }
                        else
              if( arg0.getSource() == t[2] ){
         System.out.println( "Button: "+2);
        b[2].setLabel( Integer.toString(
      Integer.parseInt(b[2].getLabel())+1 ));

                        }}}
```

Rezultatul realizării programului este prezentat în figura 1.1.

În exemplul următor se creează un MP3 Player în Java, folosind obiectul Jslider şi obiectul Timer.

Fig 1.1. Rezultatul executării exemplului 2.

Exemplul 3: MP3 Player în Java folosind obiectul Jslider şi obiectul Timer.

```
import java.awt.BorderLayout;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.JFrame;
import javax.swing.JSlider;
public class JavaSlider{
   public static void main(String []args){
      // Declarare variabile
      final int valoareMinima = 0;
      final int valoareMaxima = 100;
      final int valoareInitiala = 0;
      final int valoareDeCrestere = 10;
      // Construim fereastra
      JFrame fereastra = new JFrame("Fereastra Mea");
      // Construim si adaugăm slider-ul
      final JSlider sliderNou = new JSlider(JSlider.HORIZONTAL,
valoareMinima, valoareMaxima, valoareInitiala);
      sliderNou.addChangeListener(null);
      sliderNou.setMinorTickSpacing(10);
```

```
      sliderNou.setPaintTicks(true);
      fereastra.getContentPane().add(sliderNou,
BorderLayout.CENTER);
      // Proprietăţile ferestrei
      fereastra.pack();
      fereastra.setVisible(true);
    fereastra.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Crearea timer-ul
      int delay = 0;
      int period = 1000; // 1000 milisecunde = 1 secunda
      Timer timer = new Timer();
      timer.scheduleAtFixedRate(new TimerTask(){
         int valoareCurenta = 0;
         public void run(){
//Codul care se va executa la fiecare //secundă.
               if (valoareCurenta < valoareMaxima){
                  valoareCurenta = valoareCurenta +
valoareDeCrestere;
                  sliderNou.setValue(valoareCurenta);
               }
               else{
                  this.cancel();
               }}
      }, delay, period);
   }}
```

Se va afişa un Jslider, iar cu ajutorul unui `Timer` vom mişca slider-ul timp de 10 secunde.

Rezultatul realizării programului este prezentat în figura 1.2. La începutul programului sânt declarate 4 variabile care se folosesc în administrarea slider-ului. O valoare minimă, o valoare maximă şi o valoare initială servesc pentru a crea obiectul JSlider. Dupa ce a fosr construit acest obiect, se setează distanta dintre liniile riglei de sub slider, fiind 10 la număr, după care ecestea sânt afişate.

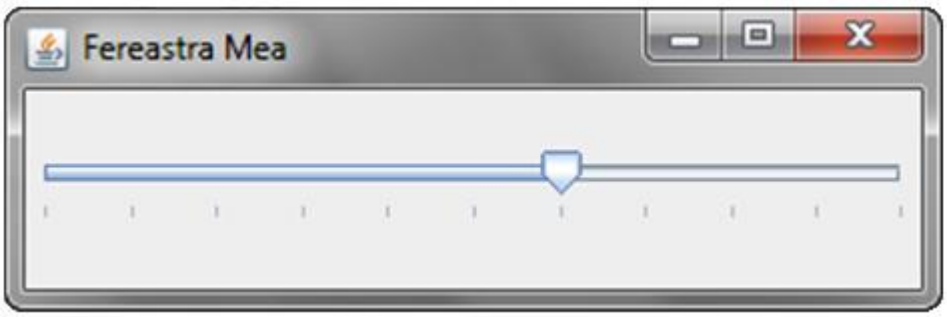

Fig. 1.2. Rezultatul executării exemplului 3

În continuare, după afisarea ferestrei, s-a construit timerul. În cadrul clasei Timer am declarat o nouă variabilă, valoareCurenta, care indică ce valoare are sliderul la un moment dat. Prin comanda sliderNou.setValue(valoareCurenta) setăm valoarea sliderului.

Un fir de execuţie (*thread*) este similar acestor programe secvenţiale, în sensul ca are un început, o secvenţă de execuţii şi un sfârşit şi în orice moment pe durata execuţiei firului există un singur punct de execuţie.

Însă un fir nu este el insuşi un program, deoarece nu poate fi executat de sine stătător. În schimb, firul este executat (rulează) într-un program. Posibilitatea utilizării mai multor fire de

execuţie într-un singur program, rulând (fiind executate) in acelaşi timp şi realizând diferite sarcini, este numită multithreading.

Lucrare de laborator 1

Tema lucrării: Elaborarea unui mecanizm de planificare a activităţii proceselor pe GIT.

Obiectivele lucrării:

- Însuşirea modalităţilor de creare a mecanizmelor de planificare;
- Însuşirea modalităţilor de prelucrare a mecanizmului de planificare;
- Utilizarea platformei Git

Numărul de ore alocate: 4 ore.

Scopul lucrării: Obținerea cunoștințelor practice în GitHub, utilizând posibilitățile de a modifica apicațiile realizate.

Exemplu de realizare:

```
import java.awt.*;
import java.util.Calendar;
import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;

class SoundPlayer extends TimerTask {

   @Override
   public void run() {
      Toolkit.getDefaultToolkit().beep();
      System.out.println("Sound played");
   }
}

class Message extends TimerTask {
   String msg;

   public Message(String msg) {
      this.msg = msg;
   }

   @Override
   public void run() {

      System.out.println(msg);
   }
}

public class TimerApp {
   public static void main(String[] args) {
      SoundPlayer soundPlayer = new SoundPlayer();
      Message message = new Message("Salut, Salut, Salut!!! ");
```

```
        Timer soundTimer = new Timer();
        Timer messageTimer = new Timer();

        soundTimer.scheduleAtFixedRate(soundPlayer, 0, 2000);
        messageTimer.schedule(new TimerTask() {
           @Override
           public void run() {
              System.out.println("5 seconds have passed");
              soundTimer.cancel();
           }
        }, 5000);

        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.HOUR_OF_DAY, 18);
        calendar.set(Calendar.MINUTE, 24);
        calendar.set(Calendar.SECOND, 0);

        Date date = calendar.getTime();

        Timer clock = new Timer();
        clock.schedule(message, date);
        System.out.println("Start");

     }
  }
```

Rzultatul realizării:

```
run:
Start
Sound played
Sound played
Sound played
5 seconds have passed
Salut, Salut, Salut!!!
```

Sarcina lucrării:

Creaţi o aplicaţie cu unu sau mai multe taimere utilizînd clasele Timer și TimerTask în diferite moduri:

1. Să reacționeze la un anumit interval de timp.
2. Să reacționeze la un anumit timp.
3. Să reacționeze cu o perioadă indicată.
4.

Etapele de realizare a lucrării:

1. Creați un cont pe GitHub (https://github.com/) (dacă nu aveți unul) și trimiteți numele de utilizator la linkul de profil.

2. Instalați mediul Git pe dispozitivul dvs., pregătiți un folder pentru aplicație și inițializați depozitul local din acest folder (comanda git init).

3. Implementați o aplicație simplă (din ***Sarcina lucrării***). În echipă decideți cine ce părți a aplicației va implementa.

4. După ce ați primit instrucțiuni de la profesor, introduceți modificările în ramura dvs. separată de pe GitHub. Efectuați commit pentru modificări în codul aplicației (comanda git commit).

5. La finalizarea lucrării, elaborați un raport care trebuie să conțină - numele, prenumele, grupa, sarcina și opțiunea dvs. pentru implementarea sarcinii, scurtă descriere, un link către codul sursă de pe GitHub. Salvați raportul în format PDF sau WORD și trimiteți pe ELSE.

Întrebări de verificare:

1. Ce este GIT?
2. Componentele de bază în GIT?
3. Funcțiile de bază a lui GIT?
4. Dați definiția unui Timer.
5. Pentru ce este folosită metoda schedule() și din ce clasă Java ea vine?
6. Enumerați pașii care trebuie urmați pentru crearea unui timer.
7. Care este diferența dintre metoda schedule() și scheduleAtFixedRate()?
8. Cînd se oprește executarea unui timer?
9. Ce metode se folosesc pentru oprirea forțată a unui timer?

Criterii de evaluare:

1. Înregistrarea pe GitHub și crearea a inui GIT local pe calculatorul gazdă.
2. Crearea şi iniţializarea timer-ului.
3. Creare a minimum 3 fire de execuție în aplicație.
4. Crearea interfeţei programului.
5. Corectitudinea codului - verificarea dacă codul este corect, fără erori de funcționare.
6. Respectarea instrucțiunilor și cerințelor - verificarea corectitudenii cerințelor sarcinii, cum ar fi numărul de timer-e și fire de execuție.
7. Optimizarea codului - Evaluarea eficienței codului în utilizarea resurselor și evitarea codului redundant.
8. Respectarea termenului de susținere - evaluarea punctajului în funcție de punctualitate, dacă lucrarea a fost predată în termenul stabilit.
9. Evaluarea cunoștințelor - explicațiile oferite despre procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.
10. Utilizarea de către student a IA.

Pentru obținerea notei 5 - 6 sunt obligatorii criteriile 1,2,5,6,8,9

Pentru obținerea notei 7 - 8 sunt obligatorii criteriile 1,2,3,5,6,8,9

Pentru obținerea notei 9 - 10 sunt obligatorii criteriile 1-9

Dacă a fost utilizat criteriul 10 nota este scăzută cu 2 baluri, numai dacă studentul sa lămurit în funcționarea codului. În caz contrar lucrare de laborator nu este susținută.

Lista de literatură recomandată:

1. - https://git-scm.com/docs/git-push) accesat pe 5.01.25
2. - https://ocw.cs.pub.ro/courses/uso/laboratoare/laborator-08/git-intro accesat la 5.01.25
3. - https://docs.oracle.com/javase/8/docs/api/java/util/Timer.html accesat pe 20.10.24
4. - https://www.geeksforgeeks.org/java-util-timer-class-java/ accesat pe 20.10.24