Разработка механизма планирования деятельности процессов с использованием таймера. Использование среды Git в группе.

1. Онлайн-платформа для веб-хостинга GIT. Таймер.

Git — это бесплатная система контроля версий с открытым исходным кодом, разработанная для быстрой и эффективной работы с любыми проектами — от небольших до очень крупных. Git прост в освоении, занимает мало места и отличается высокой производительностью. Он превосходит такие SCM (Supply Chain Management — управление цепочкой поставок) инструменты, как Subversion, CVS, Perforce и ClearCase, благодаря таким характеристикам, как локальные низкозатратные ветвления, удобные хранилища и множественные рабочие потоки.

1.1 Ветвление и слияние

Особенность Git, которая действительно выделяет его среди почти всех других SCM, — это модель ветвления. Git позволяет и поощряет создание нескольких локальных ветвей, которые могут быть полностью независимыми друг от друга. Создание, слияние и удаление этих ветвей занимает всего несколько секунд.
Это означает, что вы можете делать такие вещи, как:
• Бесконтактное переключение контекста. Создайте ветвь, на которой можно опробовать любую идею, сделайте несколько коммитов, вернитесь к месту, где была создана ветвь, примените исправление, вернитесь к месту, где были проведены эксперименты, и объедините все.
• Строки кода для ролевых игр. У вас есть ветвь, которая всегда содержит только то, что отправлено в производство, другая, в которой работа объединяется для тестирования, и несколько небольших ветвей для повседневной работы.
• Рабочий процесс, основанный на функциях. Создавайте новые ветки для каждой новой функции, над которой вы работаете, чтобы вы могли без проблем переключаться между ними, а затем удалять каждую ветку, когда функция будет объединена с основной линией.
• Единовременные эксперименты. Создайте ветку для экспериментов, поймите, что она не работает, и удалите ее, покиньте свою работу, и никто больше ее не увидит (даже если за это время вы продвинули другие ветки).

1.2 Распределение

Одной из самых полезных функций любого распределенного SCM, включая Git, является его распределение. Это означает, что вместо того, чтобы выполнять «проверку» текущей вершины исходного кода, вы выполняете «клонирование» всего репозитория.

1.3 Множественные резервные копии

Это означает, что даже если вы используете централизованный рабочий процесс, каждый пользователь фактически имеет полную резервную копию основного сервера.
Каждая из этих копий может быть использована для замены основного сервера в случае сбоя или повреждения. По сути, в Git нет единой точки отказа, если только не существует единственной копии репозитория.

1.4 Рабочий процесс

Благодаря распределенному характеру Git и отличной системе ветвления можно относительно легко реализовать практически бесконечное количество рабочих процессов.

1.5 Рабочий процесс в стиле Subversion

Централизованный рабочий процесс очень распространен, особенно среди пользователей, которые переходят с централизованной системы. Git не позволит вам выполнить push, если кто-то выполнил push с момента последнего извлечения, поэтому

централизованная модель, в которой все разработчики выполняют push на один и тот же сервер, работает просто великолепно.

1.6 Установка Git

Скачайте программу установки (https://git-scm.com/downloads), запустите программу установки, выберите нужные опции (или оставьте все как есть, если не очень хорошо разбираетесь), дождитесь завершения процесса установки и все готово.

1.7 Использование Git

В рабочей папке, где находятся исходные файлы приложения/проекта, через терминал или командную строку используйте команду git init (официальная документация - https://git-scm.com/docs/git-init) для инициализации локального репозитория, после чего Git будет отслеживать изменения в исходных файлах и фиксировать их — используйте команду git commit (официальная документация — https://git scm.com/docs/git-commit).
В дальнейшем, если необходимо синхронизировать изменения в общем репозитории для всех репозиториев, используйте команду git push

2 . Планирование деятельности процессов.

1.1. Таймер

Таймер — это устройство, которое периодически уведомляет приложение о том, что истек определенный заранее установленный период времени. Программа указывает таймеру интервал времени, указывая интервал истечения. Таймер важен для работы многих приложений.

1.1.2. Использование таймера

Чтобы привязать таймер к программе, необходимо использовать класс Timer, который предоставляет возможность планировать различные действия, которые должны быть выполнены в определенный момент времени запущенным приложением.

Действия объекта типа Timer реализуются как экземпляры класса TimerTask и могут быть запрограммированы для однократного выполнения или для повторяющихся выполнений через регулярные промежутки времени. Для использования таймера необходимо выполнить следующие шаги:

- создание подкласса действия класса TimerTask и переопределение метода run(), который будет содержать запланированное действие. Можно также использовать анонимные классы;
- создание потока выполнения путем инстанциирования класса Timer;
- создание объекта типа action;
- планирование выполнения объекта типа action с помощью метода schedule() из класса Timer.

Метод планирования, который мы можем использовать, имеет следующие форматы (он перегружен):

```
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)
```

где:

task - описывает действие, которое будет выполняться;

delay — представляет собой задержку по отношению к текущему моменту, после которой начнется выполнение;

time - точное время, когда начнется выполнение;

period - интервал времени между двумя выполнениями.

Методы планирования делятся на две категории:

• schedule() - планирование с фиксированной задержкой: если по какой-то причине действие задерживается, следующие действия также будут задержаны соответственно;

• scheduleAtFixedRate() — планирование с фиксированным количеством шагов: если действие задерживается, следующие действия будут выполняться быстрее, так что общее количество действий за период времени будет всегда одинаковым.

Таймер остановится по истечении времени его метода run(), но в то же время его можно принудительно остановить с помощью метода cancel(). После остановки его больше нельзя будет использовать для планирования других действий . Кроме того, с помощью метода System.exit() принудительно останавливаются все потоки выполнения и завершается текущее приложение.

Пример 1: использование классов Timer и 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 сообщение ;
public Alarma(String сообщение) {
this.mesaj = mesaj ;
}
public void run() {
System.out.println( сообщение );
}}
public class TestTimer {
public static void main(String args []) {
// Устанавливаем повторяющееся действие с фиксированной частотой
final Timer t1 = new Timer ();
t1.scheduleAtFixedRate(
new Внимание(), 0, 1*1000) ;
// Анонимный класс для другого действия
Timer t2 = new Timer ();
t2. schedule ( new TimerTask() {
public void run() {
System.out.println("Прошло 10 секунд");
// Останавливаем первый таймер
t1. cancel();
}}, 10*1000) ;
// Устанавливаем действие на 13:15
Calendar calendar =  Calendar.getInstance();
calendar.set( Calendar . HOUR_OF_DAY, 13);
calendar.set( Calendar .MINUTE, 15);
calendar.set( Calendar .SECOND, 0);
Дата time = calendar.getTime();
Timer t3 = new Timer();
t3.schedule(new Alarma "Время обеда!"), ora );
}}
```

Результат выполнения программы:

```
run-single:
.....Прошло  10 секунд
Время обеда!
```

Следующий пример содержит окно с тремя кнопками и тремя объектами типа Timer. Каждый объект Timer управляет одной кнопкой, то есть по истечении интервала, заданного для каждого таймера, увеличивается число на соответствующей кнопке. При создании объектов типа Timer мы указываем интервал времени, в течение которого он действует, и объект listener, то есть компонент, который получает уведомление. Timer отправляет сообщение actionPerformed() по истечении интервала времени. Объект Timer должен быть запущен путем вызова метода start().

Пример 2: Создание окна с тремя кнопками и тремя объектами типа 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&lt;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&lt;n;i++){
    t[ i ] = new Timer( (i+1)*100, this );
        t[ i ].start();
                }}
   public static void main(String[] args) {
                MyWindow w = new MyWindow();

                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 ));
 }}}
```

Результат выполнения программы представлен на рисунке 1.1.

В следующем примере создается MP3-плеер на Java с использованием объекта Jslider и объекта Timer.

Рис. 1.1. Результат выполнения примера 2.

Пример 3: MP3-плеер на Java с использованием объекта Jslider и объекта 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){
      // Объявление переменных
      final int минимальноеЗначение = 0;
      final int максимальноеЗначение = 100;
      final int начальноеЗначение = 0;
      final int значениеРоста = 10;
      // Создание окна
      JFrame окно = new JFrame("Мое окно");
      // Создаем и добавляем слайдер
      final JSlider новый ползунок =  new
JSlider(JSlider.HORIZONTAL, минимальное значение, максимальное
значение, начальное значение);
      sliderNou.addChangeListener(null);
      sliderNou.setMinorTickSpacing(10);
      sliderNou.setPaintTicks(true);
      окно.getContentPane().add(новый слайдер,
BorderLayout.CENTER);
      // Свойства окна
```

```
      окно.pack();
      окно.setVisible(true);
    окно.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Создание таймера
      int delay = 0;
      int period = 1000; // 1000 миллисекунд = 1 секунда
      Таймер timer =  new Timer();
      timer.scheduleAtFixedRate(new TimerTask(){
         int текущееЗначение = 0;
         public void run(){
//Код, который будет выполняться каждую //секунду.
               if (текущаяЗначение <  максимальноеЗначение){
          текущаяЗначение =  текущаяЗначение +  значениеРоста;
                  sliderNou.setValue(текущееЗначение);
               }
               else{
                  this.cancel();
               }} }, задержка, период);
   }}
```

Отобразится Jslider, и с помощью `Timer` мы будем перемещать слайдер в течение 10 секунд.

Результат выполнения программы представлен на рисунке 1.2. В начале программы объявлены 4 переменные, которые используются для управления слайдером. Минимальное значение, максимальное значение и начальное значение служат для создания объекта JSlider. После создания этого объекта устанавливается расстояние между линиями линейки под слайдером, которых насчитывается 10, после чего они отображаются.

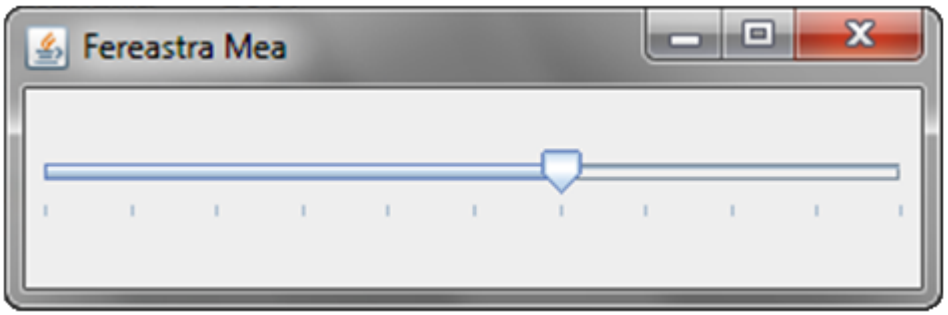
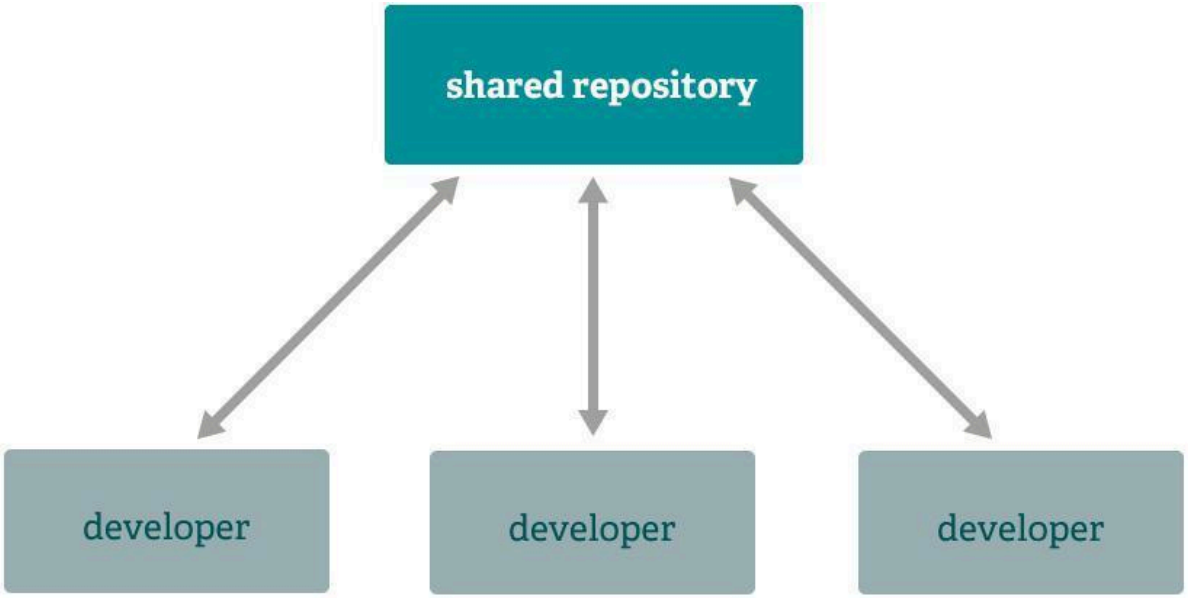


Рис. 1.2. Результат выполнения примера 3

Далее, после отображения окна, был построен таймер. В классе Timer мы объявили новую переменную, текущееЗначение, которая указывает, какое значение имеет слайдер в данный момент. С помощью команды sliderNou.setValue(текущееЗначение) мы устанавливаем значение слайдера.

Поток выполнения (*thread*) похож на эти последовательные программы в том смысле, что он имеет начало, последовательность выполнений и конец, и в любой момент во время выполнения потока существует только одна точка выполнения.

Однако поток сам по себе не является программой, поскольку не может выполняться самостоятельно. Вместо этого поток выполняется (работает) в программе. Возможность использования нескольких потоков выполнения в одной программе, работающих (выполняемых) одновременно и выполняющих различные задачи, называется многопоточностью.



Лабораторная работа я 1

Тема работы: Разработка механизма планирования деятельности процессов с использованием таймера. Использование среды Git в группе.

Цели работы:

- Освоение способов создания механизмов планирования;
- Освоение способов обработки механизма планирования;
- Использование платформы GitHub

Количество часов: 4 часа.

Цель работы: Получение практических знаний в GitHub, используя возможности изменения созданных приложений. Создание таймеров

Пример реализации:

```
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("Звук воспроизведен");
   }
}

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("Привет, привет, привет!!! ");
        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();
        Таймер clock =  new Timer();
        clock.schedule(message, date);
        System.out.println("Start");
    }
}
```

Результат выполнения:

```
run:
Start
Звук воспроизведен
Звук воспроизведен
Звук воспроизведен
Прошло 5 секунд
Привет, привет, привет!!!
```

Задание работы:

Создайте приложение с несколькими таймерами, используя классы Timer и TimerTask в различных режимах в группе:

1. Реагировать на определенный промежуток времени.
2. Реагировать на определенное время.
3. Реагировать с указанным периодом.

Этапы выполнения работы:

1. Создайте учетную запись на GitHub (https://github.com/) (если у вас ее еще нет) и отправьте имя пользователя по ссылке профиля.
2. Установите среду Git на вашем устройстве, подготовьте папку для приложения и инициализируйте локальный репозиторий в этой папке (команда git init).
3. Реализуйте простое приложение (из ***Задания работы***). В команде решите, кто какую часть приложения будет реализовывать.
4. После получения инструкций от преподавателя внесите изменения в свою отдельную ветку на GitHub. Сделайте коммит для изменений в коде приложения (команда git commit).
5. По завершении работы составьте отчет, который должен содержать: имя, фамилию, группу, задание и ваш вариант реализации задания, краткое описание, ссылку на исходный код на GitHub. Сохраните отчет в формате PDF или WORD и отправьте на ELSE.

Контрольные вопросы:

1. Что такое GIT?
2. Основные компоненты GIT?
3. Основные функции GIT?
4. Дайте определение таймеру.
5. Для чего используется метод schedule() и из какого класса Java он происходит?
6. Перечислите шаги, которые необходимо выполнить для создания таймера.
7. В чем разница между методами schedule() и scheduleAtFixedRate()?
8. Когда останавливается выполнение таймера?
9. Какие методы используются для принудительной остановки таймера?

Критерии оценки:

1. Регистрация на GitHub и создание локального GIT на хост-компьютере.
2. Создание и инициализация таймера.
3. Создание и инициализация 2 и более таймеров.
4. Создание интерфейса программы.
5. Корректность кода — проверка того, что код является корректным, без ошибок в работе.
6. Соблюдение инструкций и требований — проверка правильности требований задачи, таких как количество таймеров.
7. Оптимизация кода - оценка эффективности кода в использовании ресурсов и избегание кода.
8. Соблюдение срока сдачи - оценка баллов в зависимости от пунктуальности, сдана ли работа в установленный срок.
9. Оценка знаний — объяснения, данные о процессе выполнения работы, что может включать описание основных функций и использованной логики.
10. Использование ИИ студентом.

Для получения оценки 5-6 обязательны критерии 1, 2, 5, 8, 9.

Для получения оценки 7–8 обязательны критерии 1, 2, 3, 5, 6, 8, 9.

Для получения оценки 9–10 обязательны критерии 1–9.

Если был использован критерий 10, оценка снижается на 2 балла, только если студент разъяснил работу кода. В противном случае лабораторная работа не принимается.

Список литературы рекомендуется:

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