**Лабораторная работа 7**

**РАЗРАБОТКА АППЛЕТОВ С ИСПОЛЬЗОВАНИЕМ**

**ПОТОКОВ И ИХ СИНХРОНИЗАЦИИ**

**Цель работы:** научиться разрабатывать апплеты с использованием потоков.

**Варианты заданий(любой на выбор):**

1. Составить программу вывода на экран дисплея схематичного изобра­жения велосипедиста. При запуске программы велосипедист начинает движе­ние, вращая ногами педали велосипеда.
2. Составить программу вывода на экран дисплея схематичного изобра­жения человека. При запуске программы человек начинает идти, размахивая в такт движения руками.
3. Составить программу вывода в верхней части экрана дисплея изобра­жения облака. При запуске программы облако начинает двигаться, и из него начинает идти дождь. При этом размер облака постепенно уменьшается.
4. Составить программу вывода в верхнюю часть экрана дисплея изо­бражения тучи, а в нижнюю часть экрана дисплея - емкость для воды. При за­пуске программы начинает идти дождь. При этом размер тучи уменьшается, а емкость наполняется водой.
5. Составить программу вывода в верхнюю часть экрана дисплея изо­бражения тучи, а в нижнюю часть экрана дисплея - сугроб. При запуске про­граммы начинает идти снег. При этом размер тучи уменьшается, а сугроб рас­тет.
6. Составить программу вывода на экран дисплея изображения летящего самолета.
7. Составить программу вывода на экран дисплея изображения пушки. В правой части экрана появляется и исчезает случайным образом мишень. Нажа­тием кнопки производится выстрел из пушки. Момент попадания фиксируется в виде взрыва.
8. Составить программу вывода в верхней части экрана дисплея движу­щегося слева направо парусника с постоянной скоростью. Ее значение всякий раз задается генератором случайных чисел. В нижней части экрана дисплея расположена пушка. При нажатии кнопки происходит выстрел торпедой с по­стоянной скоростью. При попадании торпеды в пушку смоделировать взрыв парусника и его исчезновение. При промахе парусник достигает правой грани­цы экрана дисплея и начинает движение сначала с новой постоянной скоро­стью.
9. Составить программу вывода изображения циферблата механических часов с секундной, минутной и часовой стрелками. Организовать срабатывание будильника в заданное время.
10. Составить программу вывода на экран дисплея песочных часов. При нажатии кнопки моделируется процесс падения песчинок, уменьшение уровня песка в верхней части колбы и увеличение в нижней части колбы.
11. Составить программу вывода на экран дисплея треугольника. При нажатии клавиши «курсор вправо» треугольник вращается по часовой стрелке, клавиши «курсор влево» - против часовой стрелки.
12. Составить программу вывода на экран дисплея схематичного изобра­жения бабочки. При нажатии кнопки бабочка начинает полет, взмахивая крыльями.

**Многопоточное программирование**

В отличие от большинства других машинных языков, *Java* обеспечивает встроенную поддержку для многопоточного программирования. Многопоточ­ная программа содержит две и более части, которые могут выполняться одно­временно, конкурируя друг с другом. Каждая часть такой программы называет­ся потоком, а каждый поток определяет отдельный путь выполнения (в после­довательности операторов программы).

Потоки существуют в нескольких состояниях. Поток может быть в со­стоянии выполнения. Может находиться в состоянии готовности к выполне­нию, как только он получит время *CPU.* Выполняющийся поток может быть приостановлен, что временно притормаживает его действие. Затем приостанов­ленный поток может быть продолжен (возобновлен) с того места, где он был остановлен. Поток может быть блокирован в ожидании ресурса. В любой мо­мент выполнение потока может быть завершено. После завершения поток не может быть продолжен.

**Приоритеты потоков**

*Java* назначает каждому потоку приоритет, который определяет порядок обработки этого потока относительно других потоков. Приоритеты потоков - это целые числа, которые определяют относительный приоритет одного над другим. Приоритет потока используется для того, чтобы решить, когда переключаться от выполняющегося потока к другому. Это называется переключением контекста.

Потоку можно назначить приоритет от 1 (константа MIN\_PRIORITY) до 10 (MAX\_PRIORITY) с помощью метода *setPriority(),* получить значение приоритета можно помощью метода *getPriority().*

Правила переключения контекста следующие:

* поток может добровольно отказаться от управления;
* поток может быть приостановлен более приоритетным потоком.

**Класс Thread. Интерфейс Runnable**

Многопоточная система Java построена на классе *Thread,* его методах и связанном с ним интерфейсе *Runnable. Thread* инкапсулирует поток выполнения. Так как невозможно непосредственно обращаться к внутреннему состоя­нию потока выполнения, то взаимодействие с ним осуществляется через его полномочного представителя - экземпляр (объект) класса *Thread,* который его породил. Чтобы создать новый поток, выбранная программа должна будет или расширять класс *Thread,* или реализовывать интерфейс *Runnable.*

Класс *Thread* определяет несколько методов, которые помогают управ­лять потоками. В табл. 1 содержится описание методов класса *Thread*

Методы класса Thread

|  |  |
| --- | --- |
| Метод | Значение |
| getName () | Получить имя потока |
| getPriority() | Получить приоритет потока |
| isAlive () | Определить, выполняется ли еще поток |
| join () | Ждать завершения потока |
| run() | Указать точку входа в поток |
| sleep () | Приостановить поток на определенный период времени |
| start () | Запустить поток с помощью вызова метода *run()* |

**Создание потоков. Жизненный цикл потока**

Программа всегда имеет один поток, который создается при выполнении программы. В программе этот поток запускается в начале метода *main().* В ап­плете браузер является основным потоком. Программа создает поток в допол­нение к потоку выполнения, который его создает. Для создания дополнительно­го потока используется объект класса *Thread.* Каждый дополнительный поток, который задает программа, представлен объектом класса *Thread* или подкласса *Thread.* Если программа имеет три дополнительных потока, то необходимо соз­дать три таких объекта.

Для начала выполнения потока вызывается метод *start()* объекта *Thread.* Код, который выполняется в новом потоке, всегда является методом с именем *rип(),* который является открытым *(public),* не имеет аргументов и не возвраща­ет значение. Потоки, отличные от основного потока программы, всегда запус­каются методом *rип()* объекта, который представляет поток.

При реализации интерфейса *Runnable* необходимо определить его един­ственный метод *rип().*

Программа, которая создает три потока, схематически проиллюстрирова­на на рис. 2.1. Приведенная иллюстрация показывает метод *main(),* создающий три потока. Однако необязательно так должно быть всегда. Любой поток может создавать дополнительные потоки.

**Синхронизация потоков**

Основное различие между потоками и процессами состоит в том, что процессы защищены от воздействия друг на друга средствами операционной системы (каждый процесс выполняется в своем адресном пространстве). Ис­пользование потоков, лишенных подобной защиты, позволяет быстро запускать новые потоки и способствует их производительности. Однако здесь есть и от­рицательный эффект - любой из потоков может получить доступ и даже внести изменения в данные, которые другой поток считает принадлежащими только ему. Решение этой проблемы состоит в синхронизации потоков. Ситуация, ко­гда много потоков, обращающихся к некоторому общему ресурсу, начинают мешать друг другу, очень часта. Например, когда два потока записывают ин­формацию в файл/обьекг/поток. Синхронизация кода реализуется двумя основ­ными способами.

1. Если критическим участком является метод, то можно просто указать ключевое слово *synchronized* в объявлении метода, т.е.

*synchronized void ту Method ( ) {...}*

Эквивалентный код можно представить в виде

*void myMethod( )*

*{synchronized (this)*

*{ }}*

2. Если нет доступа к классу, в котором объявлен метод, то для его син­хронизации можно использовать следующий прием:

*synchonized (object)*

*{ // операторы критического участка, в том числе и вызовы метода }*

Здесь *object* - ссылка на объект, который нужно синхронизировать, т.е. на объект, элементом которого является вызываемый метод. При синхронизации одного оператора фигурные скобки можно опускать.

В Java кроме использования блока *synchonized* разработаны и эффектив­ные средства межпроцессового взаимодействия. Например, метод

*public final void wait () throws InterruptedException:*

осуществляет перевод вызывающего потока в режим ожидания, пока некото­рый другой поток не введет *notify().*

Существуют и другие варианты метода, например:

*public final void wait (long timeout) throws InterruptedException;*

осуществляет задержку на определенное время.

Метод

*public final void notify();*

«пробуждает» на том же объекте первый поток, который вызвал ожидание - *wait( );*

Следующий метод

*public final void notifyAll();*

пробуждает все потоки, для которых вызван *wait(),* и первым будет выполнять­ся поток с наибольшим приоритетом.

**Пример** программы с использованием потоков: демонстрирует применение потоков в апплете. Создается апплет, в разных потоках осуществляется осуществляется движение строки, квадрата и овала, а также зарисовка апплета.

Листинг AppletThreadSainple.java

import java.awt.\*;

import java.applet.\*;

//класс апплета, который реализует интерфейс Runnable

public class AppletThreadSample extends Applet implements Runnable {

private Thread T; //создать объект потока

//объявление переменных

private ShapeString m\_ShapeString = null; //для строки

private ShapeOval m\_ShapeOval = null; //для овала

private ShapeRect m\_ShapeRect = null; //для квадрата

public void run() { //реализация метода run, точка входа в поток

setBackground(Color.yellow); //фон апплета зарисовывается желтым

while (true){ //бесконечный цикл

repaint(); //перерисовка апплета или вызов метода paint

try{

T.sleep(10); //приостановка апплета на 10 миллисекунд

}

catch (Interrupted Except ion e){ }

}

}

public void init() {//метод инициализации апплета

Т = new Thread(this); //создание потока и привязка его к текущему классу

T.start(); //запуск потока (вызывается run)

//создание объектов

m\_ShapeString = new ShapeString();

m\_ShapeOval= new ShapeOval();

m\_ShapeRect= new ShapeRect();

}

public void paint(Graphics g) { //метод прорисовки апплета

//прорисовка строки

g.drawString("This is ShapeString",

m\_ShapeString.x\_String.m\_ShapeString.y\_String);

//прорисовка квадрата

g.setColor(Color.red);

g.drawRect(m\_ShapeRect.x\_Rect,m\_ShapeRect.y\_Rect,

m\_ShapeRect.w\_Rect,m\_ShapeRect.h\_Reel);

//прорисовка овала

g.setColor(Color.CYAN);

g.fillOval(m\_ShapeOval.x\_Oval,m\_ShapeOval.y\_Oval,

m\_ShapeOval.w\_Oval.m\_ShapeOval.h\_Oval);

}

//класс ShapeString. реализующий интерфейс Runnable

class ShapeString implements Runnable {

Thread T;

int x\_String, у\_String; //координаты строки

public ShapeString(){ //конструктор

T = new Thread(this); //создание объекта Thread

//установление начальных координат строки

Х\_String= 100; у\_String= 100;

T.start(); //запуск потока (вызов метода run)

}

public void run(){ //метод run

for;;){

x\_String+=15; //изменение координаты строки

try{

T.sleep( 1000);//приостановка работы потока на 1000 миллисекунд

}

catch (InterruptedException е){}

}

}

}

//класс ShapeRect реализующий интерфейс Runnable

class ShapeRect implements Runnable {

Thread T;

int x\_Rect,у\_Rect,w\_Rect.h\_Rect; //координаты и размеры квадрата

public ShapeRect(){ //конструктор

T = new Thread(this); //создание объекта Thread

//установление начальных координат квадрата

х\_Rect=350;y\_Rect=50;w\_Rect=100;h\_Rect=100;

T.start();//3anycк потока (вызов метода run)

}

public void run(){ //метод run

for;;){

x Rect-=15; //изменение координаты квадрата

try{

Т sleep(500); //приостановка работы потока

}

catch (Interrupted Exception e){}

}

}

}

//класс ShapeOval реализующий интерфейс Runnable

class ShapeOval implements Runnable{

Thread T;

int x\_Oval, у\_Oval,w\_Oval,h\_Oval; //координаты и размеры овала

public ShapeOval(){ //конструктор

T = new Thread(this); //создание объекта Thread

//установление начальных координат овала

X\_Oval=30: y\_Oval=30:w\_Oval= 100:h\_Oval=90;

T.start(); //запуск потока (вызов метода run)

}

public void run() {//метод run

for(;;){//изменение координат овала

x\_Oval+=8;

у\_Oval+=7;

try{

T.sleep(100); //приостановка работы потока на 100 миллисекунд

}

catch (Interrupted Exception e){ }

} } } }

