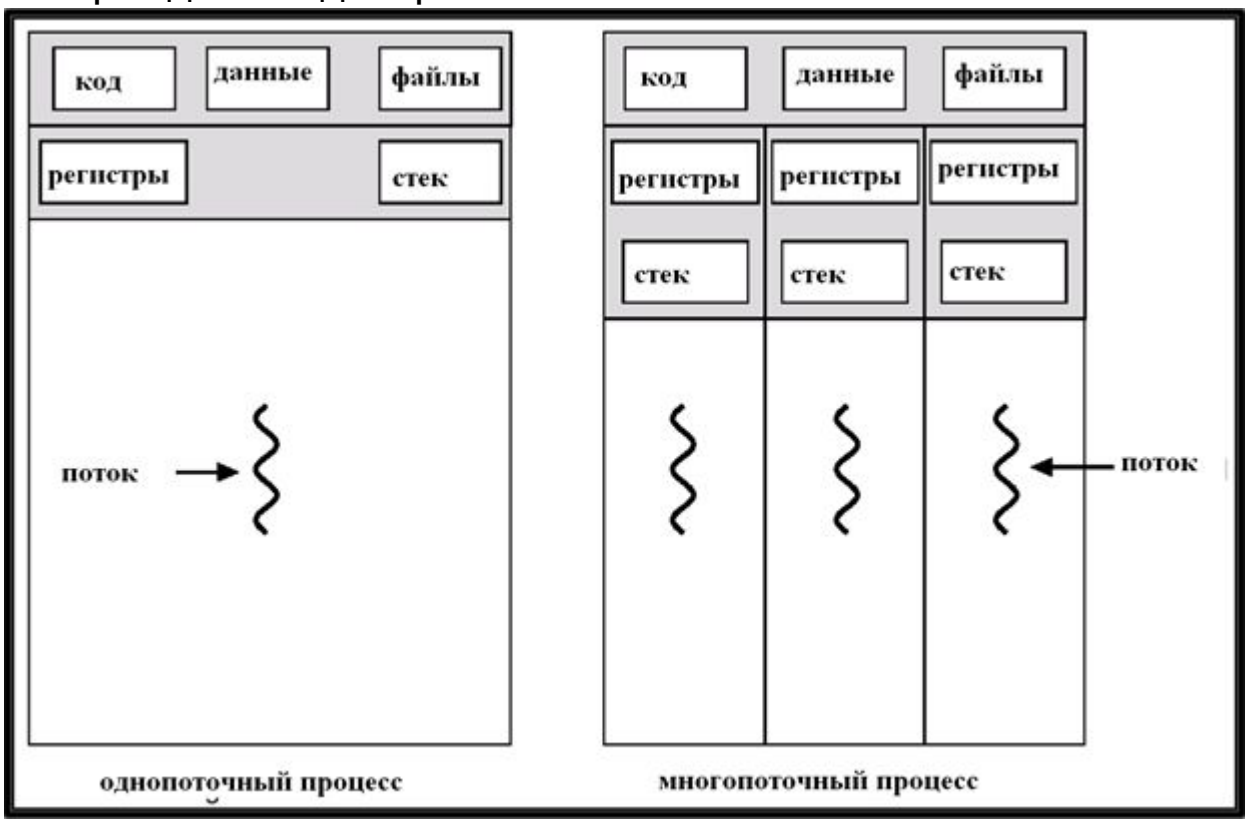
Организация потоков в Java

Процессы и потоки

Процесс — это экземпляр программы, который запускается независимо от остальных, у него есть собственное адресное пространство.
Поток – это одна из веток процесса. Все потоки разделяют адресное пространство породившего их процесса и имеют доступ к одним данным.
Один из потоков – **«главный»** начинает выполняться первым при запуске Java-программы. Главный поток создается автоматически с именем main и приоритетом 5 по умолчанию.
От него порождаются дочерние потоки.



Классы для работы с потоками

Класс *Thread* предназначен для создания нового потока.
Он определяет следующие основные конструкторы :

```
Thread()
Thread(Runnable object)
Thread(Runnable object, String name)
Thread(String name)
```

где *name* - имя, присваиваемое потоку, *object* - экземпляр объекта *Runnable* .
Если имя не присвоено, система сгенерирует уникальное имя в виде *Thread-N*, где *N* - целое число. Для создания потока можно использовать также интерфейс Runnable

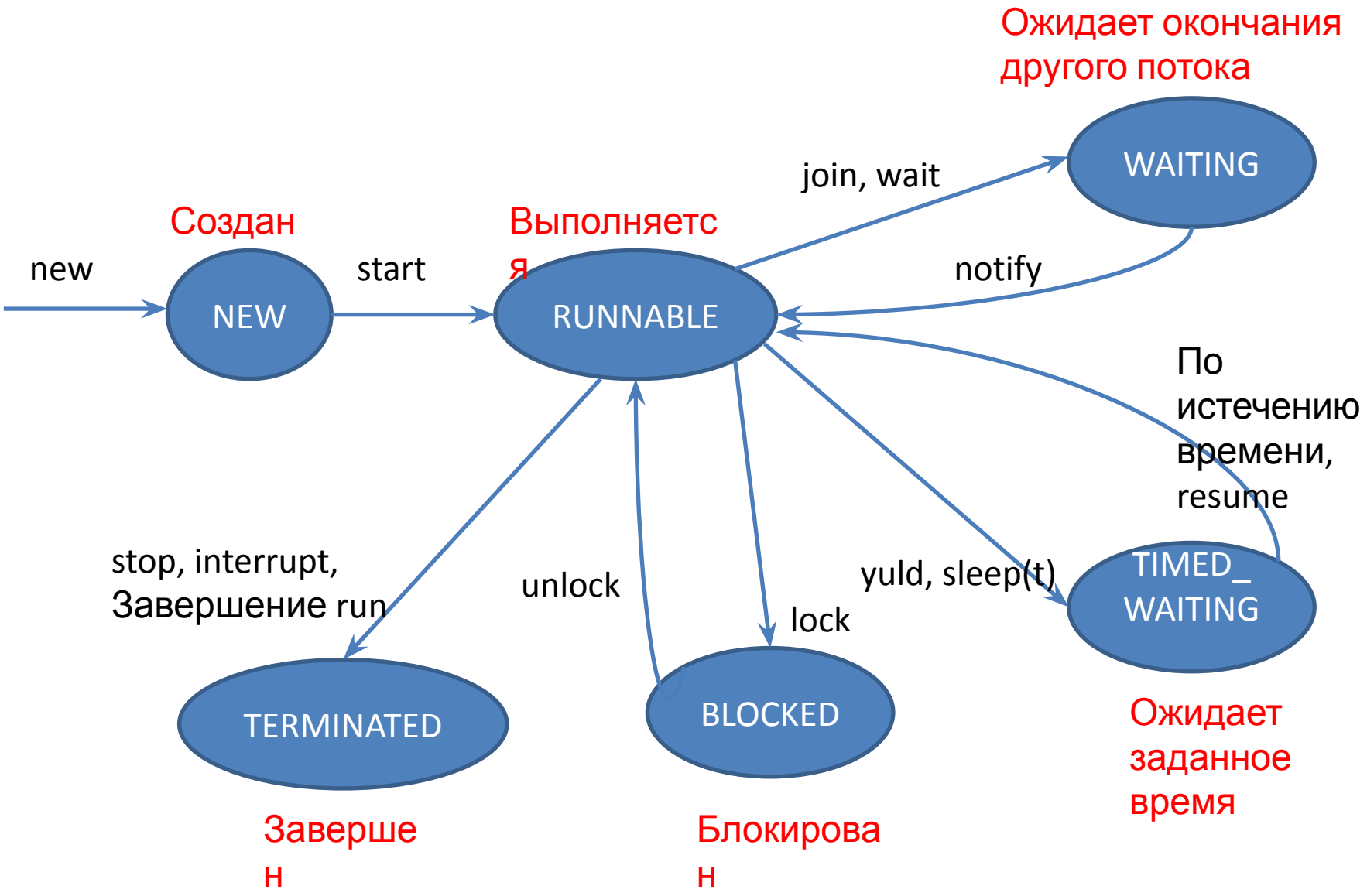
```
public interface Runnable {
public  abstract void run();
}
```

http://www.javaportal.ru/java/class/Thread.html

Методы управления потоками

static Thread.currentThread() - получить текущий поток выполнения
getName() - получить имя потока
getPriority() - получить приоритет потока
getState() – получить состояние потока
isAlive() - определить, выполняется ли поток
void start() - начинает выполнение потока
final void stop() - заканчивает выполнение потока
static void sleep(long msec) - прекращает выполнение потока на указанное количество млсек
static void yield() - приостановка потока, чтобы передать ресурсы процессора другому потоку
void join() – ожидание завершения потока

Состояние потока



Создание потоков Java

Запустить новый поток можно двумя способами:

1. Создать потомка класса Thread и переопределить его метод run()
2. Создать объект класса Thread, передав ему в конструкторе класс, реализующий интерфейс Runnable. Этот интерфейс содержит метод run(), который будет выполняться в новом потоке.

```
//Создание 1-го потока путем расширения класса Thread
class One extends Thread {
        // точка входа 1-го потока
        public void run() {
                . . . . . . . . . . . . . . . . .
                выполнение 1-го потока  . . . . . . . . . . . . . . . .   }  }
//Создание 2-го потока путем реализации интерфейса Runnable
        class Two implements Runnable {
        // точка входа 2-го потока
        public void run() {
                 . . . . . . . . . . . . . . . . .
                выполнение 2-го потока  . . . . . . . . . . . . . . . .   }  }
// запуск программы
        public class OneTwo {
        public static void main(String args[]) {
        // создание экземпляров классов
        One c = new One() ;    Runnable r = new Two() ;
        Thread t = new Thread(r) ; // передача объекта Runnable классу Thread
        . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
        // запуск потоков
        c.start() ;
        t.start() ;
}  }
```

Определение состояния потоков

```
class MyRun implements Runnable {
Thread t,t1;
private int sec;
MyRun(int sec) {
this.sec = sec;
// Главный поток
t1=Thread.currentThread();
}
public void run() {
// Дочерний поток
t=Thread.currentThread();
for (int i = 0; i < 1; i++) {
try {
System.out.println(t.getName()+t.getState());

System.out.println(t1.getName()+t1.getState());
Thread.sleep(sec);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
}
```

```
public class MyThread {
    public static void main(String[] args) throws Exception {
      Runnable runnable = new MyRun( 300);
      Thread thread = new Thread(runnable);
      Thread t=Thread.currentThread();
// Состояние главного потока
System.out.println(t.getName()+t.getState());
 // Состояние дочернего потока
   System.out.println(thread.getName()+thread.getState());
 // Запуск дочернего потока
thread.start();
 // Состояние дочернего потока
System.out.println(thread.getName()+thread.getState());
   for (int i = 0; i < 2; i++) {
      System.out.println("Ожидает-"+t.getName());
 Thread.sleep(100); // Задержка главного потока
      }
thread.join(); // Ожидание завершения дочернего
потока
System.out.println(thread.getName()+thread.getState());
      System.out.println(t.getName()+t.getState());
       }
  }
```

Модификатор **Volatile**

- Применяется в многопоточных приложениях
- Используется только с переменными
- Может использоваться со static переменными
- Нет локальных копий
- Не используется с final переменными
- Значение переменной, объявленной как volatile, измененное одним потоком, асинхронно меняется и для других потоков

Использование **Volatile**

```
public class InfiniteLoop implements Runnable {
   private  boolean flag = true;
// private  volatile boolean flag = true;
public void run() {
            while (flag) {
       }
      System.out.println("finished");
       }
      public static void main(String[] args)
         throws InterruptedException {
      InfiniteLoop loop = new InfiniteLoop();
      Thread t = new Thread(loop);
      t.start();
      Thread.sleep(1000);
      loop.flag = false;
   }
}
```

Пример создания потока путем расширения класса Thread

Incremenator — поток, который каждую секунду прибавляет или вычитает единицу из значения статической переменной Program.mValue. Incremenator содержит два закрытых поля – mIsIncrement и mFinish. То, какое действие выполняется, определяется булевой переменной mIsIncrement — если оно равно true, то выполняется прибавление единицы, иначе — вычитание. Завершение потока происходит, когда значение mFinish становится равно true.

```
class Incremenator extends Thread {
        private volatile boolean mIsIncrement = true;
        private volatile boolean mFinish = false;

        public void changeAction() //Меняет действие на противоположное
                { mIsIncrement = !mIsIncrement; }

        public void finish() //Инициирует завершение потока
                { mFinish = true; }

public void run() {
        System.out.print(this); // Вывод имени потока
        System.out.print("Значение Program.mValue = ");
        do { if(!mFinish) //Проверка на необходимость завершения раз в секунду
        { if(mIsIncrement) Program.mValue++; //Инкремент
        else Program.mValue--; //Декремент
//Вывод текущего значения переменной
System.out.print(Program.mValue + " "); }
         else return; //Завершение потока
        try{ Thread.sleep(1000); //Приостановка потока на 1 сек.
        }catch(InterruptedException e){} }
        while(true);
} }
```

Продолжение примера

```
public class Program {
 //Переменая, которой оперирует инкременатор
   public static int mValue = 0;
      static Incremenator mInc; // Обявление ссылки  на дочерний поток

   public static void main(String[] args)
   {       mInc = new Incremenator();    //Создание дочернего потока
           mInc.start();  //Запуск потока

      //Троекратное изменение действия инкременатора с интервалом в i*2 секунд
      for(int i = 1; i <= 3; i++)
      {
         try{
            Thread.sleep(i*2*1000); //Ожидание в течении i*2 сек.
         }catch(InterruptedException e){}

         mInc.changeAction(); //Переключение действия через 2, 4 и 6 сек.
      }
           mInc.finish(); //Инициация завершения дочернего потока
   }
}
```

Thread1

Завершение работы потоков

Завершить работу потока можно следующими тремя способами:

- Поток завершиться, когда закончит выполнение метода run().

- Поток может создать исключительное состояние или ошибку, которые не удастся перехватить.

- Поток поток может вызвать один из нерекомендуемых методов stop() и destroy() . Понятие «нерекомендуемые» означает, что эти методы существуют, но их не следует использовать. при принудительной остановке (приостановке) потока совершенно непонятно, что делать с ресурсами.

Главный поток является последним выполняющимся потоком. Программа завершается, когда главный поток останавливается.

Прерывание потока

Класс Thread содержит в себе скрытое булево поле, которое называется флагом прерывания.

Установить этот флаг можно вызвав метод interrupt() потока (посылает уведомление о прерывании).

Проверить же, установлен ли этот флаг, можно двумя способами.

Первый способ — вызвать метод bool isInterrupted() объекта потока, он возвращает состояние флага прерывания и оставляет этот флаг нетронутым.

Второй способ — вызвать статический метод bool Thread.interrupted().

Он возвращает состояние флага и сбрасывает его, и его вызов возвращает значение флага прерывания того потока, из которого он был вызван. Поэтому этот метод вызывается только изнутри потока и позволяет потоку проверить своё состояние прерывания.

У методов, приостанавливающих выполнение потока, таких как sleep(), wait() и join() есть одна особенность — если во время их выполнения будет вызван метод interrupt() этого потока, они, не дожидаясь конца времени ожидания, сгенерируют исключение InterruptedException.

Пример прерывания потока

```
class Incremenator extends Thread {
private volatile boolean mIsIncrement = true;
public void changeAction() //Меняет действие на противоположное
{ mIsIncrement = !mIsIncrement; }
public void run() {
do { if(!Thread.interrupted()) //Проверка прерывания
{ if(mIsIncrement) Program.mValue++; //Инкремент
else Program.mValue--; //Декремент
//Вывод текущего значения переменной
System.out.print(Program.mValue + " "); }
else return; //Завершение потока
try{ Thread.sleep(1000); //Приостановка потока на 1 сек.
    }catch(InterruptedException e){System.out.print("interrupt");
return; //Завершение потока после прерывания } }
while(true); } }
```

Продолжение примера

```
class Program {
//Переменая, которой оперирует инкременатор
public static int mValue = 0;
static Incremenator mInc; //Объект побочного потока
public static void main(String[] args) {
mInc = new Incremenator(); //Создание потока
System.out.print("Значение = ");
mInc.start(); //Запуск потока
//Троекратное изменение действия инкременатора
//с интервалом в i*2 секунд
for(int i = 1; i <= 3; i++) {
try{ Thread.sleep(i*2*1000); //Ожидание в течении i*2 сек.
    }catch(InterruptedException e){}
mInc.changeAction(); //Переключение действия }
mInc.interrupt(); //Прерывание  потока }
}
```

Диспетчеризация потоков

Планировщик определяет, какой поток должен запуститься, основываясь на номер приоритета, назначенный каждому потоку.

Приоритет потока может принимать значения от 1 до 10.

По умолчанию, значение приоритета для потока является Thread.NORM_PRIORITY, которому соответствует значение 5.

Так же доступны две других static переменных: Thread.MIN_PRIORITY, значение 1, и Thread.MAX_PRIORITY значение 10.

Метод getPriority() может использоваться для получения текущего значения приоритета соответствующего потока.

Установить приоритет можно методом

setPriority(int newPriority)

Статический метод Thread.yield() можно использовать для того чтобы принудить планировщик выполнить другой поток, который ожидает своей очереди.

Приоритеты потоков

```
class MyThread implements Runnable{
        public int count=0;
        private int prioriti;
        public MyThread(int prioriti ){
        this.prioriti=prioriti;
        }
        public void run() {
         Thread t=Thread.currentThread();
           System.out.println(t.getName());
           System.out.println(t.getPriority());
           t.setPriority(prioriti);
           System.out.println(t.getPriority());
           for(int i=0;i<1000000;i++){
        count++;
        }
           System.out.println(count);
        }
        }
public class Solution {
        public static void main(String[] args){
        //передаем параметр отвечающий за приоритет
            Runnable r2=new MyThread(Thread.MIN_PRIORITY);
           Thread p2 = new Thread(r2,"MIN");
           p2.start();
           Runnable r1=new MyThread(Thread.MAX_PRIORITY);
           Thread p1 = new Thread(r1,"MAX");
           p1.start();      }
           }
```