```
        for (int i = 0; i < 71; i++){
            System.out.println(getName() + " " + i);
            try {
                sleep(1);//попробовать sleep(0);
            } catch (InterruptedException e) {
                System.err.print("Error" + e);
            }
        }
    }
}
package chapt14;

public class PriorityRunner {
    public static void main(String[] args) {
        PriorThread min = new PriorThread("Min");//1
        PriorThread max = new PriorThread("Max");//10
        PriorThread norm = new PriorThread("Norm");//5
        min.setPriority(Thread.MIN_PRIORITY);
        max.setPriority(Thread.MAX_PRIORITY);
        norm.setPriority(Thread.NORM_PRIORITY);
        min.start();
        norm.start();
        max.start();
    }
}
```

Поток с более высоким приоритетом в данном случае, как правило, монополизирует вывод на консоль.

Потоки объединяются в группы потоков. После создания потока нельзя изменить его принадлежность к группе.

```
ThreadGroup tg = new ThreadGroup("Группа потоков 1");
Thread t0 = new Thread(tg, "поток 0");
```

Все потоки, объединенные группой, имеют одинаковый приоритет. Чтобы определить, к какой группе относится поток, следует вызвать метод **getThreadGroup()**. Если поток до включения в группу имел приоритет выше приоритета группы потоков, то после включения значение его приритета станет равным приоритету группы. Поток же со значением приоритета более низким, чем приоритет группы после включения в оную, значения своего приоритета не изменит.

Управление потоками

Приостановить (задержать) выполнение потока можно с помощью метода **sleep(**время задержки**)** класса **Thread**. Менее надежный альтернативный способ состоит в вызове метода **yield()**, который может сделать некоторую паузу и позволяет другим потокам начать выполнение своей задачи. Метод **join()** блокирует работу потока, в котором он вызван, до тех пор, пока не будет закончено выполнение вызывающего метод потока.

```
// пример # 4 : задержка потока: JoinRunner.java
package chapt14;

class Th extends Thread {
     public Th(String str) {
          super();
          setName(str);
     }
     public void run() {
          String nameT = getName();
          System.out.println("Старт потока " + nameT);
          if ("First".equals(nameT)) {
               try {
                    sleep(5000);
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
               System.out.println("завершение потока "
                                                  + nameT);
          } else if ("Second".equals(nameT)) {
               try {
                    sleep(1000);
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
             System.out.println("завершение потока "
                                                  + nameT);
          }
     }
}
public class JoinRunner {
     public static void main(String[] args) {
          Th tr1 = new Th("First");
          Th tr2 = new Th("Second");
          tr1.start();
          tr2.start();
          try {
             tr1.join();
             System.out.println("завершение main");
          } catch (InterruptedException e){
                    e.printStackTrace();
          }
          /* join() не дает работать потоку main до окончания выполнения
          потока tr1 */
     }
}
```

Возможно, будет выведено:

```
Старт потока First
```

```
Старт потока Second
завершение потока Second
завершение потока First
завершение main
```

Несмотря на вызов метода **join()** для потока **tr1**, поток **tr2** будет работать, в отличие от потока **main**, который сможет продолжить свое выполнение только по завершении потока **tr1**.

Вызов метода **yield()** для исполняемого потока должен приводить к приостановке потока на некоторый квант времени, для того чтобы другие потоки могли выполнять свои действия. Однако если требуется надежная остановка потока, то следует использовать его крайне осторожно или вообще применить другой способ.

```
// пример # 5 : задержка потока: YieldRunner.java
package chapt14;

public class YieldRunner {
     public static void main(String[] args) {
          new Thread() {
             public void run() {
                    System.out.println("старт потока 1");
                    Thread.yield();
                    System.out.println("завершение 1");
               }
          }.start();
          new Thread() {
             public void run() {
                    System.out.println("старт потока 2");
                    System.out.println("завершение 2");
               }
          }.start();
     }
}
```

В результате может быть выведено:

```
старт потока 1
старт потока 2
завершение 2
завершение 1
```

Активизация метода **yield()** в коде метода **run()** первого объекта потока приведет к тому, что, скорее всего, первый поток будет остановлен на некоторый квант времени, что даст возможность другому потоку запуститься и выполнить свой код.

Потоки-демоны

Потоки-демоны работают в фоновом режиме вместе с программой, но не являются неотъемлемой частью программы. Если какой-либо процесс может выполняться на фоне работы основных потоков выполнения и его деятельность за-