Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления **Thread.State**:

NEW – поток создан, но еще не запущен;

RUNNABLE – поток выполняется;

BLOCKED – поток блокирован;

WAITING – поток ждет окончания работы другого потока;

TIMED_WAITING – поток некоторое время ждет окончания другого потока;

TERMINATED — поток завершен.

Получить значение состояния потока можно вызовом метода **getState()**.

Поток переходит в состояние "неработоспособный" (**WAITING**) вызовом методов **wait()**, **suspend()** (deprecated-метод) или методов ввода/вывода, которые предполагают задержку. Для задержки потока на некоторое время (в миллисекундах) можно перевести его в режим ожидания (**TIMED_WAITING**) с помощью методов **sleep(long millis)** и **wait(long timeout)**, при выполнении которого может генерироваться прерывание **InterruptedException**. Вернуть потоку работоспособность после вызова метода **suspend()** можно методом **resume()** (deprecated-метод), а после вызова метода **wait()** – методами **notify()** или **notifyAll()**. Поток переходит в "пассивное" состояние (**TERMINATED**), если вызваны методы **interrupt()**, **stop()** (deprecated-метод) или метод **run()** завершил выполнение. После этого, чтобы запустить поток еще раз, необходимо создать новый объект потока. Метод **interrupt()** успешно завершает поток, если он находится в состоянии "работоспособность". Если же поток неработоспособен, то метод генерирует исключительные ситуации разного типа в зависимости от способа остановки потока.

Интерфейс **Runnable** не имеет метода **start()**, а только единственный метод **run()**. Поэтому для запуска такого потока, как **Walk**, следует создать объект класса **Thread** и передать объект **Walk** его конструктору. Однако при прямом вызове метода **run()** поток не запустится, выполнится только тело самого метода.

Методы **suspend()**, **resume()** и **stop()** являются deprecated-методами и запрещены к использованию, так как они не являются в полной мере "потокобезопасными".

Управление приоритетами и группы потоков

Потоку можно назначить приоритет от **1** (константа **MIN_PRIORITY**) до **10** (**MAX_PRIORITY**) с помощью метода **setPriority(int prior)**. Получить значение приоритета можно с помощью метода **getPriority()**.

// пример # 3 : демонстрация приоритетов: PriorityRunner.java: PriorThread.java

```
package chapt14;

public class PriorThread extends Thread {
     public PriorThread(String name){
          super(name);
     }
     public void run(){
```

```
        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.