```
                } catch (InterruptedException e) {
                    System.err.println(e);
                }
            }
        }
}
package chapt14;

public class WalkTalk {
    public static void main(String[] args) {
        // новые объекты потоков
        Talk talk  = new Talk();
        Thread walk = new Thread(new Walk());
        // запуск потоков
        talk.start();
        walk.start();

        //Walk w = new Walk(); // просто объект, не поток
        // w.run(); //выполнится метод, но поток не запустится!
    }
}
```

Использование двух потоков для объектов классов **Talk** и **Walk** приводит к выводу строк: `Talking Walking`. Порядок вывода, как правило, различен при нескольких запусках приложения.

Жизненный цикл потока

При выполнении программы объект класса **Thread** может быть в одном из четырех основных состояний: "новый", "работоспособный", "неработоспособный" и "пассивный". При создании потока он получает состояние "новый" (**NEW**) и не выполняется. Для перевода потока из состояния "новый" в состояние "работоспособный" (**RUNNABLE**) следует выполнить метод **start()**, который вызывает метод **run()** – основной метод потока.

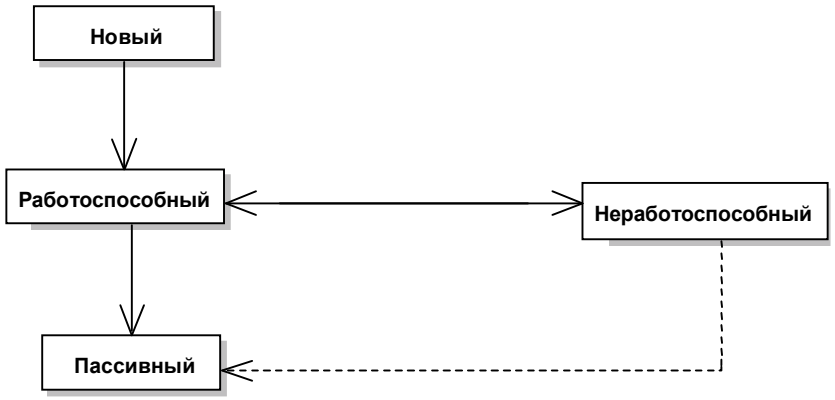

Рис. 14.1. Состояния потока

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(){
```