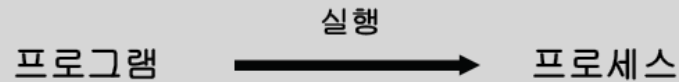


Java의 정석

제 12 장

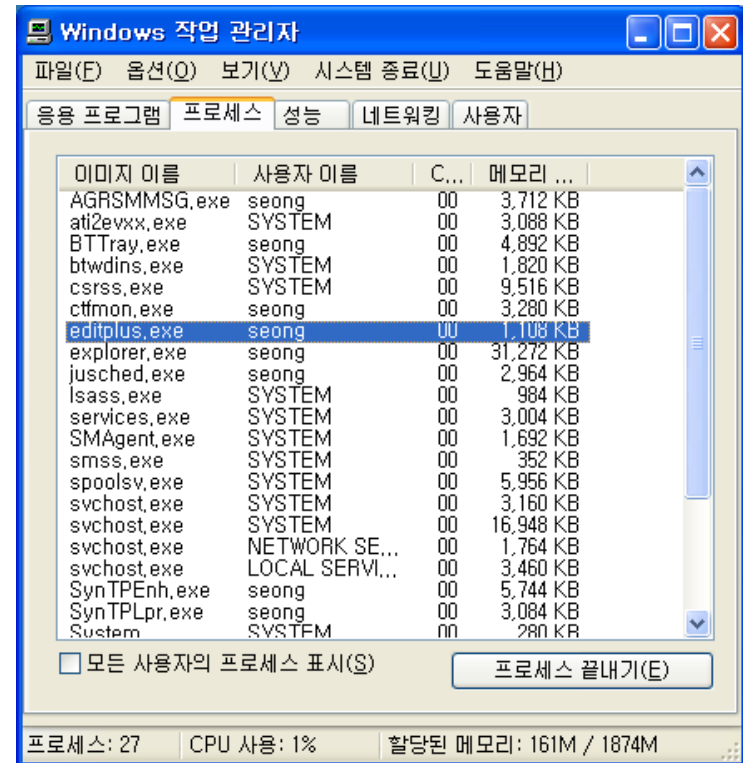
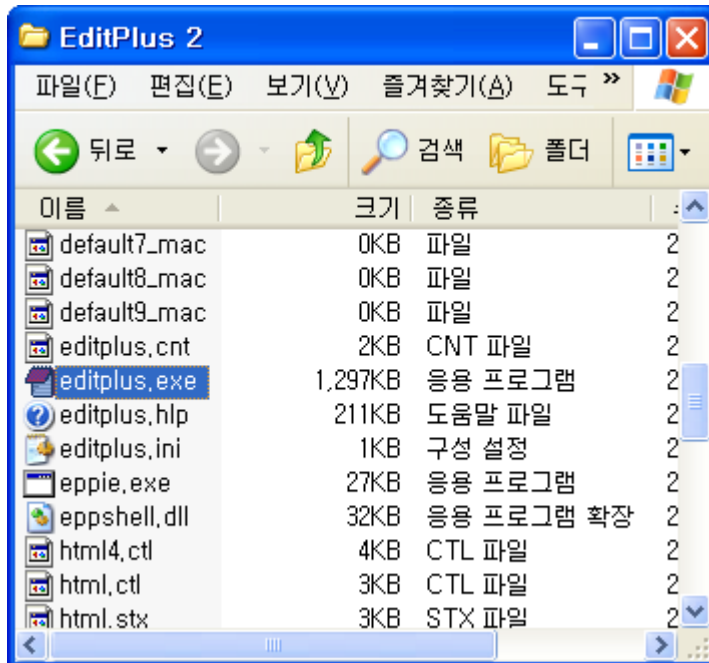
쓰레드(thread)

1.1 프로세스와 쓰레드(process & thread) (1)



▶ 프로그램 : 실행 가능한 파일(HDD)

▶ 프로세스 : 실행 중인 프로그램(메모리)



1.1 프로세스와 쓰레드(process & thread) (2)

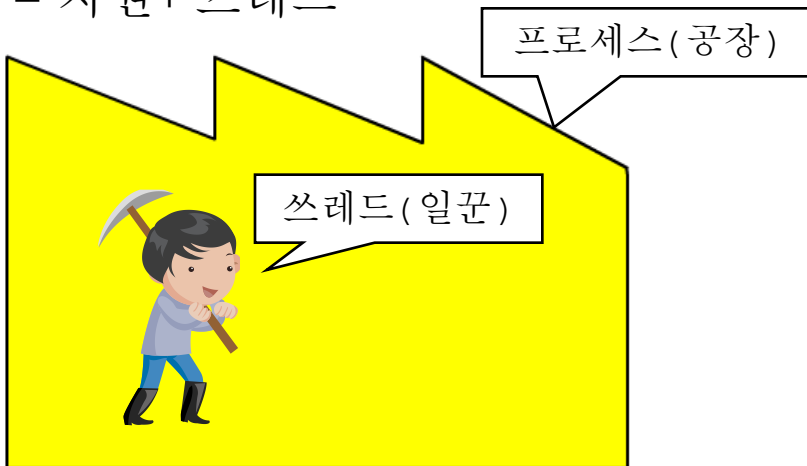
- ▶ 프로세스 : 실행 중인 프로그램, 자원(resources)과 쓰레드로 구성
- ▶ 쓰레드 : 프로세스 내에서 실제 작업을 수행.

모든 프로세스는 하나 이상의 쓰레드를 가지고 있다.

프로세스 : 쓰레드 = 공장 : 일꾼

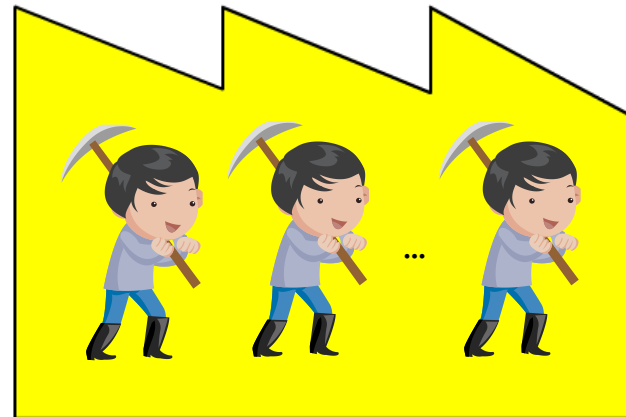
- ▶ 싱글 쓰레드 프로세스

= 자원 + 쓰레드



- ▶ 멀티 쓰레드 프로세스

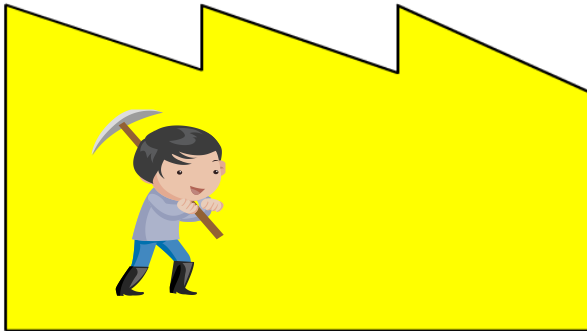
= 자원 + 쓰레드 + 쓰레드 + ... + 쓰레드



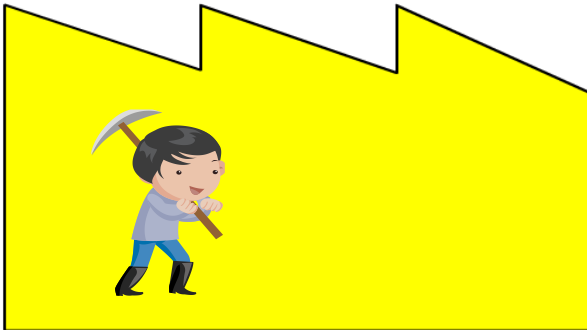
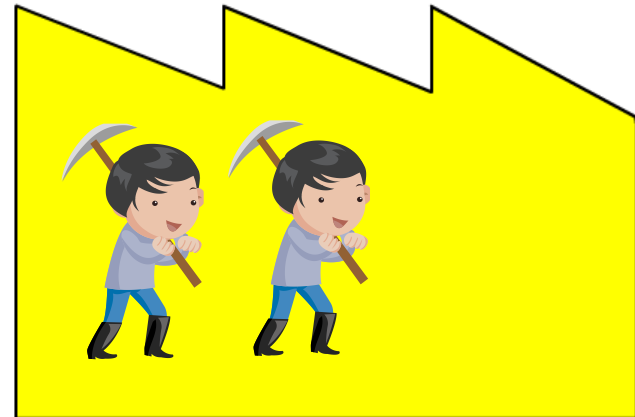
1.2 멀티프로세스 vs. 멀티쓰레드

“하나의 새로운 프로세스를 생성하는 것보다
하나의 새로운 쓰레드를 생성하는 것이 더 적은 비용이 든다.”

- 2 프로세스 1 쓰레드 vs. 1 프로세스 2 쓰레드



VS.



1.3 멀티쓰레드의 장단점

“많은 프로그램들이 멀티쓰레드로 작성되어 있다.

그러나, 멀티쓰레드 프로그래밍이 장점만 있는 것은 아니다.”

장점	<ul style="list-style-type: none">- 자원을 보다 효율적으로 사용할 수 있다.- 사용자에게 대한 응답성(responsiveness)이 향상된다.- 작업이 분리되어 코드가 간결해 진다. <p>“여러 모로 좋다.”</p>
단점	<ul style="list-style-type: none">- 동기화(synchronization)에 주의해야 한다.- 교착상태(dead-lock)가 발생하지 않도록 주의해야 한다.- 각 쓰레드가 효율적으로 고르게 실행될 수 있게 해야 한다. <p>“프로그래밍할 때 고려해야 할 사항들이 많다.”</p>

1.4 쓰레드의 구현과 실행

1. Thread클래스를 상속

```
class MyThread extends Thread {  
    public void run() { /* 작업내용 */ } // Thread클래스의 run()을 오버라이딩  
}
```

2. Runnable인터페이스를 구현

```
class MyThread implements Runnable {  
    public void run() { /* 작업내용 */ } // Runnable인터페이스의 추상메서드 run()을 구현  
}
```

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

```
ThreadEx1_1 t1 = new ThreadEx1_1();
```

```
Runnable r = new ThreadEx1_2();
```

```
Thread t2 = new Thread(r); // 생성자 Thread(Runnable target)
```

```
// Thread t2 = new Thread(new ThreadEx1_2());
```

1.5 start() & run()

```
class ThreadTest {  
    public static void main(String args[]) {  
        MyThread t1 = new MyThread();  
        t1.start();  
    }  
}
```

```
class MyThread extends Thread {  
    public void run() {  
        //...  
    }  
}
```

1. Call stack

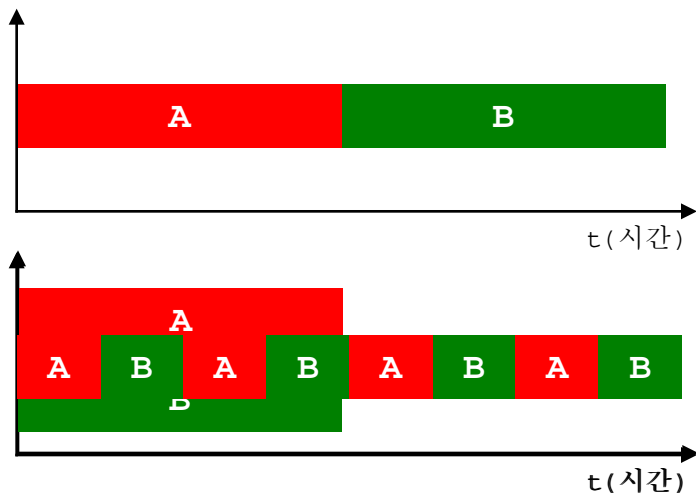


1.6 싱글쓰레드 vs. 멀티쓰레드(1)

▶ 싱글쓰레드

```
class ThreadTest {
    public static void main(String args[]){
        for(int i=0;i<300;i++) {
            System.out.println("-");
        }

        for(int i=0;i<300;i++) {
            System.out.println("|");
        }
    } // main
}
```



▶ 멀티쓰레드

```
class ThreadTest {
    public static void main(String args[]){
        MyThread1 th1 = new MyThread1();
        MyThread2 th2 = new MyThread2();
        th1.start();
        th2.start();
    }
}

class MyThread1 extends Thread {
    public void run() {
        for(int i=0;i<300;i++) {
            System.out.println("-");
        }
    } // run()
}

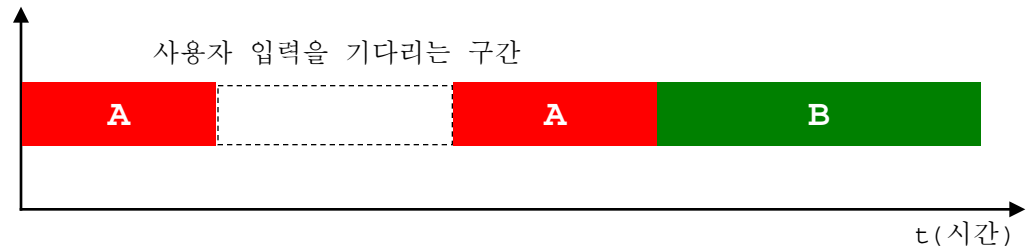
class MyThread2 extends Thread {
    public void run() {
        for(int i=0;i<300;i++) {
            System.out.println("|");
        }
    } // run()
}
```


1.6 싱글쓰레드 vs. 멀티쓰레드(2)

```
class ThreadEx6 {
    public static void main(String[] args){
        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요.");
        System.out.println("입력하신 값은 " + input + "입니다.");

        for(int i=10; i > 0; i--) {
            System.out.println(i);
            try { Thread.sleep(1000); } catch (Exception e) {}
        }
    } // main
}
```

▶ 싱글쓰레드

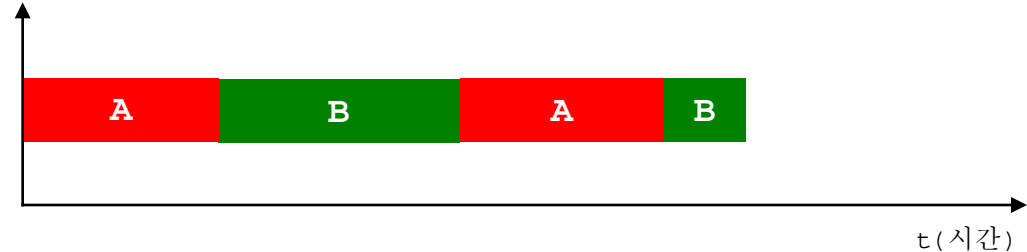


```
class ThreadEx7 {
    public static void main(String[] args){
        ThreadEx7_1 th1 = new ThreadEx7_1();
        th1.start();

        String input = JOptionPane.showInputDialog("아무 값이나 입력하세요.");
        System.out.println("입력하신 값은 " + input + "입니다.");
    }
}

class ThreadEx7_1 extends Thread {
    public void run() {
        for(int i=10; i > 0; i--) {
            System.out.println(i);
            try { sleep(1000); } catch (Exception e) {}
        }
    } // run()
}
```

▶ 멀티쓰레드



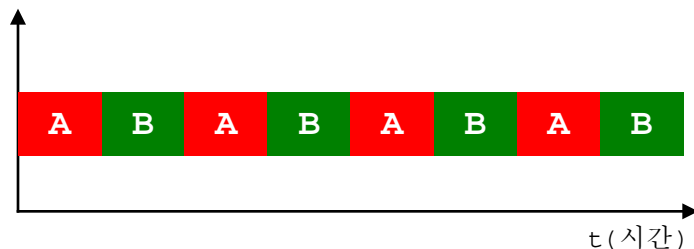
1.7 쓰레드의 우선순위(priority of thread)

“작업의 중요도에 따라 쓰레드의 우선순위를 다르게 하여
특정 쓰레드가 더 많은 작업시간을 갖도록 할 수 있다.”

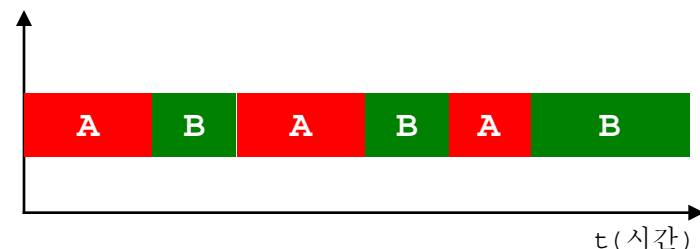
```
void setPriority(int newPriority) : 쓰레드의 우선순위를 지정한 값으로 변경한다.  
int getPriority() : 쓰레드의 우선순위를 반환한다.
```

```
public static final int MAX_PRIORITY = 10 // 최대우선순위  
public static final int MIN_PRIORITY = 1  // 최소우선순위  
public static final int NORM_PRIORITY = 5 // 보통우선순위
```

▶ 우선순위가 같은 경우



▶ A의 우선순위가 높은 경우



1.8 쓰레드 그룹(ThreadGroup)

- 서로 관련된 쓰레드를 그룹으로 묶어서 다루기 위한 것
- 모든 쓰레드는 반드시 하나의 쓰레드 그룹에 포함되어 있어야 한다.
- 쓰레드 그룹을 지정하지 않고 생성한 쓰레드는 'main쓰레드 그룹'에 속한다.
- 자신을 생성한 쓰레드(조상 쓰레드)의 그룹과 우선순위를 상속받는다.

생성자 / 메서드	설 명
ThreadGroup(String name)	지정된 이름의 새로운 쓰레드 그룹을 생성한다.
ThreadGroup(ThreadGroup parent, String name)	지정된 쓰레드 그룹에 포함되는 새로운 쓰레드 그룹을 생성한다.
int activeCount()	쓰레드 그룹에 포함된 활성상태에 있는 쓰레드의 수를 반환한다.
void destroy()	쓰레드 그룹과 하위 쓰레드 그룹까지 모두 삭제한다. 단, 쓰레드 그룹이나 하위 쓰레드 그룹이 비어있어야 한다.
int getMaxPriority()	쓰레드 그룹의 최대우선 순위를 반환한다.
String getName()	쓰레드 그룹의 이름을 반환한다.
ThreadGroup getParent()	쓰레드 그룹의 상위 쓰레드그룹을 반환한다.
void interrupt()	쓰레드 그룹에 속한 모든 쓰레드를 interrupt한다.
boolean isDaemon()	쓰레드 그룹이 데몬 쓰레드그룹인지 확인한다.
boolean isDestroyed()	쓰레드 그룹이 삭제되었는지 확인한다.
void list()	쓰레드 그룹에 속한 쓰레드와 하위 쓰레드그룹에 대한 정보를 출력한다.
boolean parentOf(ThreadGroup g)	지정된 쓰레드 그룹의 상위 쓰레드그룹인지 확인한다.
void setDaemon(boolean daemon)	쓰레드 그룹을 데몬 쓰레드그룹으로 설정/해제한다.
void setMaxPriority(int pri)	쓰레드 그룹의 최대우선 순위를 설정한다.

1.9 데몬 쓰레드(daemon thread)

- 일반 쓰레드(non-daemon thread)의 작업을 돕는 보조적인 역할을 수행.
- 일반 쓰레드가 모두 종료되면 자동적으로 종료된다.
- 가비지 컬렉터, 자동저장, 화면자동갱신 등에 사용된다.
- 무한루프와 조건문을 이용해서 실행 후 대기하다가 특정조건이 만족되면 작업을 수행하고 다시 대기하도록 작성한다.

`boolean isDaemon()` - 쓰레드가 데몬 쓰레드인지 확인한다. 데몬 쓰레드이면 `true`를 반환한다.

`void setDaemon(boolean on)` - 쓰레드를 데몬 쓰레드로 또는 사용자 쓰레드로 변경한다.

* `setDaemon(boolean on)`은 반
그렇지 않으면 `IllegalThreadS`

```
public void run() {  
    while(true) {  
        try {  
            Thread.sleep(3 * 1000); // 3초마다  
        } catch (InterruptedException e) {}  
  
        // autoSave의 값이 true이면 autoSave()를 호출한다.  
        if(autoSave) {  
            autoSave();  
        }  
    }  
}
```

1.10 쓰레드의 실행제어

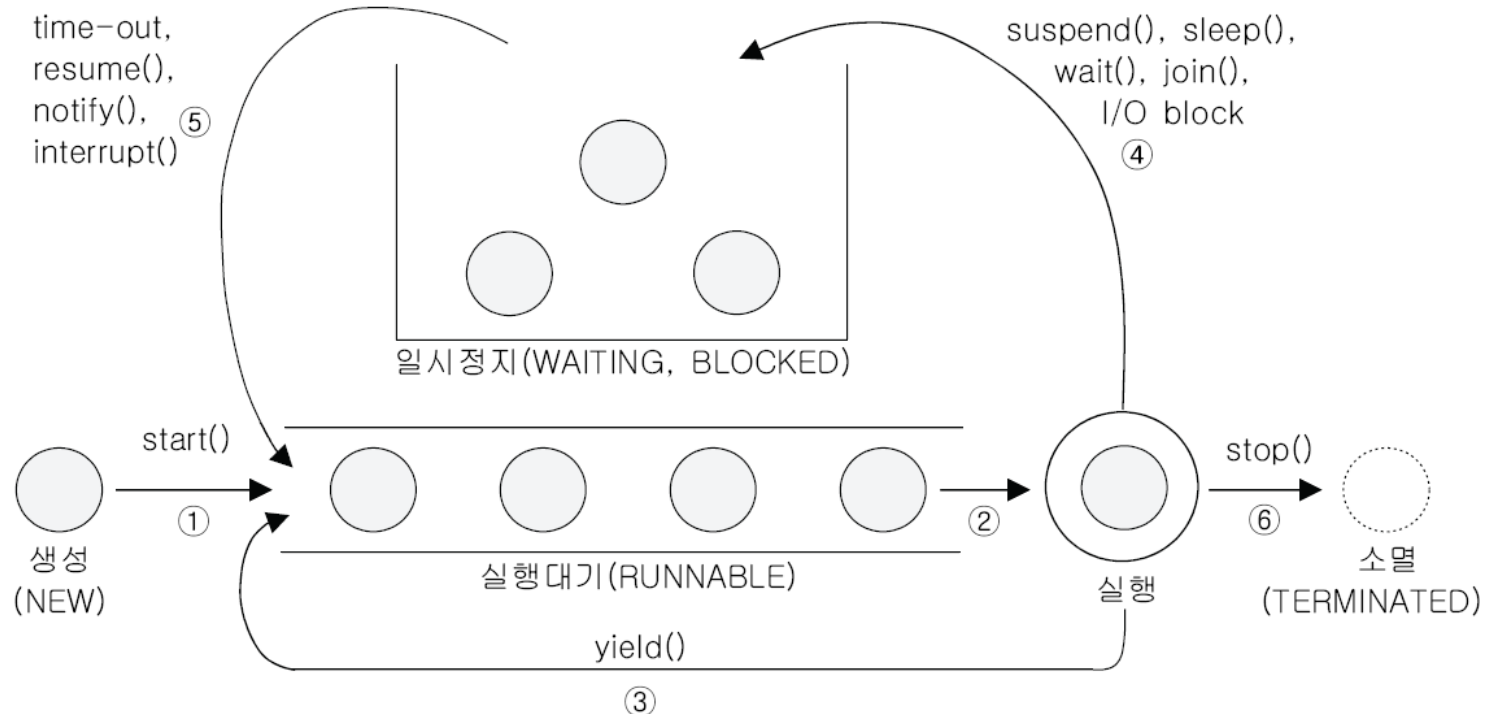
생성자 / 메서드	설 명
void interrupt()	sleep()이나 join()에 의해 일시정지상태인 쓰레드를 실행대기상태로 만든다. 해당 쓰레드에서는 InterruptedException이 발생함으로써 일시정지상태를 벗어나게 된다.
void join() void join(long millis) void join(long millis, int nanos)	지정된 시간동안 쓰레드가 실행되도록 한다. 지정된 시간이 지나거나 작업이 종료되면 join()을 호출한 쓰레드로 다시 돌아와 실행을 계속한다.
void resume()	suspend()에 의해 일시정지상태에 있는 쓰레드를 실행대기상태로 만든다.
static void sleep(long millis) static void sleep(long millis, int nanos)	지정된 시간(천분의 일초 단위)동안 쓰레드를 일시정지시킨다. 지정한 시간이 지나고 나면, 자동적으로 다시 실행대기상태가 된다.
void stop()	쓰레드를 즉시 종료시킨다. 교착상태(dead-lock)에 빠지기 쉽기 때문에 deprecated되었다.
void suspend()	쓰레드를 일시정지시킨다. resume()을 호출하면 다시 실행대기상태가 된다.
static void yield()	실행 중에 다른 쓰레드에게 양보(yield)하고 실행대기상태가 된다.

[표 12-2] 쓰레드의 스케줄링과 관련된 메서드

* resume(), stop(), suspend()는 쓰레드를 교착상태로 만들기 쉽기 때문에 deprecated되었다.

1.11 쓰레드의 상태(state of thread)

상태	설명
NEW	쓰레드가 생성되고 아직 start()가 호출되지 않은 상태
RUNNABLE	실행 중 또는 실행 가능한 상태
BLOCKED	동기화블럭에 의해서 일시정지된 상태(lock이 풀릴 때까지 기다리는 상태)
WAITING, TIMED_WAITING	쓰레드의 작업이 종료되지는 않았지만 실행가능하지 않은(unrunnable) 일시정지상태. TIMED_WAITING은 일시정지시간이 지정된 경우를 의미한다.
TERMINATED	쓰레드의 작업이 종료된 상태



1.12 쓰레드의 실행 제어 – Example1

```
class MyThreadEx18 implements Runnable {
    boolean suspended = false;
    boolean stopped = false;

    Thread th;

    MyThreadEx18(String name) {
        th = new Thread(this, name);
    }

    public void run() {
        while(!stopped) {
            if(!suspended) {
                /*
                 *      작업수행
                 */
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {}
            } // if
        } // while
    }
}
```

```
public void suspend() {
    suspended = true;
}

public void resume() {
    suspended = false;
}

public void stop() {
    stopped = true;
}

public void start() {
    th.start();
}
```

1.12 쓰레드의 실행 제어 – Example2

```
public void run() {
    while(true) {
        try {
            Thread.sleep(10 * 1000); // 10초를 기다린다.
        } catch (InterruptedException e) {
            System.out.println("Awaken by interrupt().");
        }

        gc(); // garbage collection을 수행한다.
        System.out.println("Garbage Collected. Free Memory : "
                           + freeMemory());
    }
}
```

```
for(int i=0; i < 20; i++) {
    requiredMemory = (int) (Math.random() * 10) * 20;

    // 필요한 메모리가 사용할 수 있는 양보다 적거나 전체 메모리의 60%이상을 사용했을 경우 gc를 깨운다.
    if(gc.freeMemory() < requiredMemory
        || gc.freeMemory() < gc.totalMemory() * 0.4) {
        gc.interrupt(); // 잠자고 있는 쓰레드 t1을 깨운다.
    }

    gc.usedMemory += requiredMemory;
    System.out.println("usedMemory:" + gc.usedMemory);
}
```


1.13 쓰레드의 동기화 - synchronized

- 한 번에 하나의 쓰레드만 객체에 접근할 수 있도록 객체에 락(lock)을 걸어서 데이터의 일관성을 유지하는 것.

1. 특정한 객체에 lock을 걸고자 할 때

```
synchronized(객체의 참조변수) {  
    //...  
}
```

2. 메서드에 lock을 걸고자할 때

```
public synchronized void calcSum() {  
    //...  
}
```

```
public synchronized void withdraw(int money) {  
    if(balance >= money) {  
        try {  
            Thread.sleep(1000);  
        } catch(Exception e) {}  
  
        balance -= money;  
    }  
}
```



```
public void withdraw(int money) {  
    synchronized(this) {  
        if(balance >= money) {  
            try {  
                Thread.sleep(1000);  
            } catch(Exception e) {}  
  
            balance -= money;  
        }  
    } // synchronized(this)  
}
```

1.13 쓰레드의 동기화 - Example

```

class Account {
    int balance = 1000;

    public void withdraw(int money) {
        if(balance >= money) {
            try { Thread.sleep(1000); } catch(Exception e) {}
            balance -= money;
        }
    } // withdraw
}

```

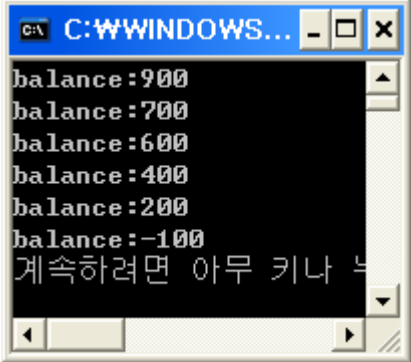
```

class RunnableEx24 implements Runnable {
    Account acc = new Account();

    public void run() {
        while(acc.balance > 0) {
            // 100, 200, 300중의 한 값을 임의로 선택해서 출금
            int money = (int)(Math.random() * 3 + 1) * 100;
            acc.withdraw(money);
            System.out.println("balance:" + acc.balance);
        }
    } // run()
}

```

▶ Synchronized 없을 때

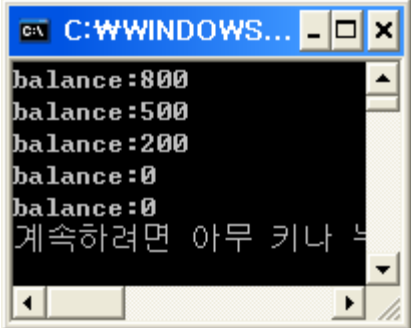


```

C:\WINDOWS...
balance:900
balance:700
balance:600
balance:400
balance:200
balance:-100
계속하려면 아무 키나 누르십시오 . . .

```

▶ Synchronized 있을 때



```

C:\WINDOWS...
balance:800
balance:500
balance:200
balance:0
balance:0
계속하려면 아무 키나 누르십시오 . . .

```

1.14 쓰레드의 동기화 – wait(), notify(), notifyAll()

- 동기화의 효율을 높이기 위해 wait(), notify()를 사용.
- Object클래스에 정의되어 있으며, 동기화 블록 내에서만 사용할 수 있다.
- ▶ wait() – 객체의 lock을 풀고 해당 객체의 쓰레드를 waiting pool에 넣는다.
- ▶ notify() – waiting pool에서 대기중인 쓰레드 중의 하나를 깨운다.
- ▶ notifyAll() – waiting pool에서 대기중인 모든 쓰레드를 깨운다.

```
class Account {
    int balance = 1000;

    public synchronized void withdraw(int money) {
        while(balance < money) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }

        balance -= money;
    } // withdraw

    public synchronized void deposit(int money) {
        balance += money;
        notify();
    }
}
```

1.14 쓰레드의 동기화 - Example2

```

class TypingGameEx5 extends Frame
{
    int speed    = 500;    // 단어가 떨어지는 속도
    int interval = 2 * 1000; // 새로운 단어가 나오는 간격

    int score = 0;
    int life  = 3;
    int curLevel = 0;

    String[][] data = {
        {"태연", "유리", "윤아", "효연", "수영"},
        {"태연", "유리", "윤아", "효연", "수영"},
        {"태연", "유리", "윤아", "효연", "수영"},
        {"태연", "유리", "윤아", "효연", "수영"},
    };

    final Level[] LEVEL = {
        new Level(500, 2000, 1000, data[0]),
        new Level(250, 1500, 2000, data[1]),
        new Level(120, 1000, 3000, data[2]),
        new Level(100, 500, 4000, data[3]),
    };

    public void levelUp() {
        virusGrp.interrupt();
        Level lvl = getLevel(++curLevel);

        lbLevel.setText("Level:" + curLevel);
        words.clear();
        screen.clear();
        showLevel(curLevel);

        speed    = lvl.speed;
        interval = lvl.interval;
    }

    //...
}

```

```

class VirusThread extends Thread {
    public VirusThread(ThreadGroup group, String name) {
        super(group, name);
    }

    public void run() {
        int rand = (int)(Math.random()*5);

        int oldValue = 0;
        int virusTime = 10 * 1000;

        switch(rand) {
            case 0:
                speed = speed / 2;
                break;
            case 1:
                interval = interval / 2;
                break;
            case 2:
                speed = speed * 2;
                break;
            case 3:
                interval = interval * 2;
                break;
            case 4:
                words.clear();
                break;
        } // switch

        delay(virusTime);

        speed = LEVEL[curLevel].speed;
        interval = LEVEL[curLevel].interval;
    } // end of run()
}

```

1.14 쓰레드의 동기화 – Example2-1 improved

```

class TypingGameEx5 extends Frame {
    int speed    = 500;    // 단어가 떨어지는 속도
    int interval = 2 * 1000; // 새로운 단어가 나오는 간격

    int score = 0;
    int life  = 3;
    int curLevel = 0;

    String[][] data = {
        {"태연", "유리", "윤아", "효연", "수영"},
        {"태연", "유리", "윤아", "효연", "수영"},
        {"태연", "유리", "윤아", "효연", "수영"},
        {"태연", "유리", "윤아", "효연", "수영"},
    };

    final Level[] LEVEL = {
        new Level(500, 2000, 1000, data[0]),
        new Level(250, 1500, 2000, data[1]),
        new Level(120, 1000, 3000, data[2]),
        new Level(100, 500, 4000, data[3]),
    };

    public synchronized int getCurLevel() {
        return curLevel;
    }

    public synchronized void levelUp() {
        virusGrp.interrupt();
        Level lvl = getLevel(++curLevel);

        lblLevel.setText("Level:" + curLevel);
        words.clear();
        screen.clear();
        showLevel(curLevel);

        speed    = lvl.speed;
        interval = lvl.interval;
    }
    //...
}

class VirusThread extends Thread {
    public VirusThread(ThreadGroup group, String name) {
        super(group, name);
    }

    public void run() {
        int rand = (int) (Math.random() * 5);

        int oldValue = 0;
        int virusTime = 10 * 1000;

        switch(rand) {
            case 0:
                speed = speed / 2;
                break;
            case 1:
                interval = interval / 2;
                break;
            case 2:
                speed = speed * 2;
                break;
            case 3:
                interval = interval * 2;
                break;
            case 4:
                words.clear();
                break;
        } // switch

        delay(virusTime);

        int curLevel = getCurLevel();
        speed = LEVEL[curLevel].speed;
        interval = LEVEL[curLevel].interval;
    } // end of run()
}

```