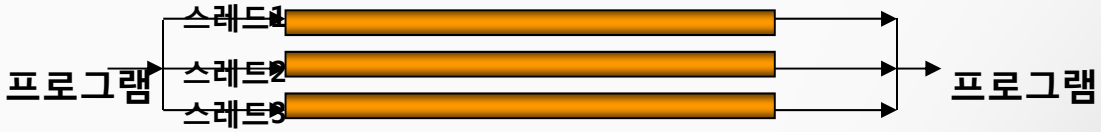
10. 쓰레드(Thread)

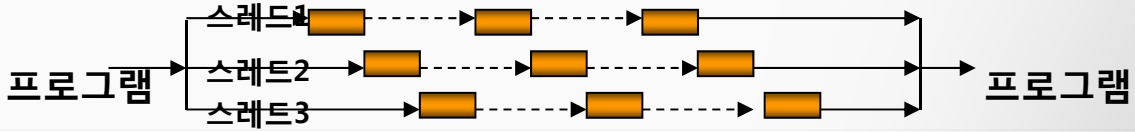
쓰레드 개요

쓰레드

- 순차적으로 동작하는 문장들의 단일 집합
- 경량(lightweight) 프로세스
- 다중 스레드
  - 하나의 프로세스(프로그램)에 하나 이상의 스레드를 생성하여 실행
- 자바는 스레드를 지원하기 위해 java.lang.Thread 클래스 제공
  - 다수개의 CPU를 가진 컴퓨터에서 다중 스레드의 실행



  - 한 개의 CPU를 가진 컴퓨터에서 다중 스레드의 실행



Thread 클래스와 스레드의 생명주기

- **JDK는 스레드를 지원하는 java.lang.Thread 클래스 제공**
- **스레드를 생성하기 위해 사용**
- **4개의 생성자를 제공**

```
Thread()
Thread(String s)
Thread(Runnable r)  // Runnable 인터페이스를 사용하여 스레드 생성
Thread(Runnable r, String s)
```

이때, r은 Runnable 인터페이스를 사용한 클래스의 객체이며,
s는 Thread 클래스 객체임

쓰레드 생성과 사용

❖ 쓰레드를 생성하는 2가지 방법

- Thread 클래스로부터 직접 상속받아 쓰레드를 생성

- Runnable 인터페이스를 사용하는 방법
  - 현재의 클래스가 이미 다른 클래스로부터 상속 받고 있는 경우

쓰레드의 생성과 사용

❖ Thread 클래스 이용

- Thread 클래스로부터 직접 상속 받아 쓰레드를 생성
- Thread 클래스에서 제공되는 run() 메소드를 오버라이딩하여 쓰레드의 동작을 기술

```
class ThreadA extends Thread  {  // Thread 클래스로부터 상속
  .........
  public void run() {
   ....  // 상위 클래스인 Thread 클래스의 run() 메소드를 오버
   ....  //라이딩하여 쓰레드가 수행하여야 하는 문장들을 기술
  }
  ....
}
```

```
ThreadA ta = new ThreadA();
ta.start();  // 쓰레드 객체를 생성하여 쓰레드를 시작시킨다
```

쓰레드의 생성과 사용

❖ Runnable 인터페이스 이용

- 현재의 클래스가 이미 다른 클래스로부터 상속을 받고 있다면 Runnable 인터페이스를 사용하여 쓰레드를 생성할 수 있음.

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

- Runnable 인터페이스는 JDK에 의해 제공되는 라이브러리 인터페이스임.
- Runnable 인터페이스에는 run() 메소드만 정의되어 있음.

쓰레드의 생성과 사용

❖ **이미 Applet 클래스로부터 상속을 받고 있으므로 인터페이스 이용**

```
class RunnableB extends Applet implements Runnable {
  ……...
    public void run() {
      ……..// Runnable 인터페이스에 정의된 run() 메소드를
      ……..// 오버라이딩하여 쓰레드가 수행할 문장들을 기술한다
    }
 …..
```

쓰레드의 생성과 사용

❖ **Runnable 인터페이스를 이용하여 쓰레드를 생성하는 방법**

```
RunnableB rb = new RunnableB(); // 객체 rb 생성
Thread tb = new Thread(rb);  // rb를 매개변수로 하여 쓰레드 객체 tb를 생성
tb.start();  // 쓰레드 시작
```

❖ **아래와 같이 생성**

```
RunnableB rb = new RunnableB();
new Thread(rb).start();  // 쓰레드 객체를 생성하여 바로 시작
```

❖ **두 가지 생성방법은 동일한 효과를 줌**

쓰레드의 생성과 사용

```
1  class ThreadTest extends Thread {
2    public void run() {
3      try {
4        for (int i=0 ; i<10 ; i++) {
5          Thread.sleep(200);
6          System.out.println("자바 :" + i);
7        }
8      }
9      catch(InterruptedException e ) {
10     System.out.println(e);
11     }
12   }
13 }
14
15 class ThreadfromThread {
16   public static void main(String args[]) {
17     ThreadTest t = new ThreadTest();
18     t.start();
19     // 스레드를 생성하고 시작시킨다
20   }
```

Thread 클래스로부터 생성

쓰레드의 생성과 사용

```
1  class RunnableTest implements Runnable {
2    public void run() {
3      try {
4        for (int i=0 ; i<10 ; i++) {
5          Thread.sleep(200);
6          System.out.println("자바 :" + i);
7        }
8      }
9      catch(InterruptedException e ) {
10     e.printStackTrace();
11     }
12   }
13 }
14
15 class ThreadfromRunnable {
16   public static void main(String args[]) {
17     RunnableTest r = new RunnableTest();
18     // Runnable 인터페이스 객체로 r을 생성
19     Thread t = new Thread(r);
20     // r을 매개 변수로 하여 스레드 객체 t를 생성
21     t.start();
22   }
23 }
```

Runnable 인터페이스로부터 생성

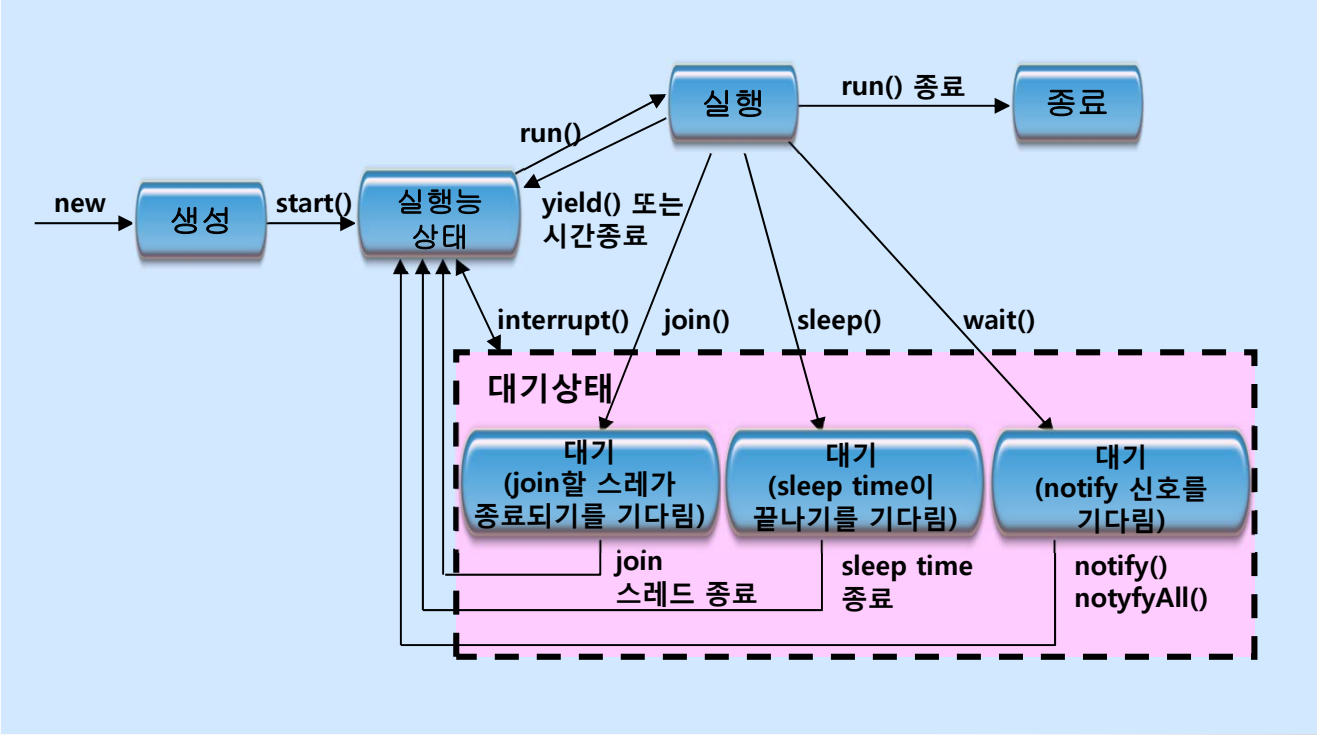
Thread 클래스의 메소드

메소드 이름	설명
static void sleep(long msec) throws InterruptedException	msec에 지정된 밀리초(milliseconds) 동안 대기
String getName()	스레드의 이름을 반환
void setName(String s)	스레드의 이름을 s로 설정
void start()	스레드를 시작시킨다. run() 메소드를 호출
int getPriority()	스레드의 우선 순위를 반환
void setPriority(int p)	스레드의 우선 순위를 p값으로 설정
boolean isAlive()	스레드가 시작되었고 아직 끝나지 않았으면 true를 그렇지 않으면 false를 반환
void join() throws InterruptedException	스레드가 끝날 때까지 대기
void run()	스레드가 실행할 부분을 기술하는 메소드 하위 클래스에서 오버라이딩 되어야 한다
void suspend()	스레드가 일시 정지된다. resume()에 의해 다시 시작될 수 있다
void resume()	일시 정지된 스레드를 다시 시작시킨다

this

Thread 클래스와 스레드의 생명주기

❖ Thread의 생명주기



스레드 우선 순위

- **스레드에 우선 순위를 부여하여 우선 순위가 높은 스레드에게 실행의 우선권을 부여할 수 있다**
  - setPriority() 메소드를 이용하여 우선 순위 부여
  - 우선 순위를 지정하기 위한 상수 제공

static final int MAX_PRIORITY	우선순위 10
static final int MIN_PRIORITY	우선순위 1
static final int NORM_PRIORITY	우선순위 5

쓰레드 우선 순위

```
1  class PriorityTest extends Thread {
2
3    public PriorityTest(String str) {
4      super(str);
5    }
6
7    public void run() {
8      try {
9        for(int i=0; i<5; i++) {
10         Thread.sleep(10);
11         System.out.println( i + getName
12         // getName()과 getPriority() 메
13         // 객체명 없이 바로 메소드 명을
14       }
15     }
16
17     catch(InterruptedException e) {
18       System.out.println(e);
19     }
20   }
21 }
```

```
0 : 첫번째 스레드 우선순위 : 1
0 : 세번째 스레드 우선순위 : 1
0 : 두번째 스레드 우선순위 : 1
1 : 첫번째 스레드 우선순위 : 1
1 : 두번째 스레드 우선순위 : 1
1 : 세번째 스레드 우선순위 : 1
2 : 첫번째 스레드 우선순위 : 1
2 : 세번째 스레드 우선순위 : 1
2 : 두번째 스레드 우선순위 : 1
3 : 첫번째 스레드 우선순위 : 1
3 : 두번째 스레드 우선순위 : 1
3 : 세번째 스레드 우선순위 : 1
4 : 첫번째 스레드 우선순위 : 1
4 : 세번째 스레드 우선순위 : 1
4 : 두번째 스레드 우선순위 : 1
```

```
>java ThreadPriority 1 1
```

쓰레드 우선 순위

```
23 class ThreadPriority {
24   public static void main(String args[]) {
25     PriorityTest t1 = new PriorityTest(" : 첫번째 스레드");
26     PriorityTest t2 = new PriorityTest(" : 두번째 스레드");
27     PriorityTest t3 = new PriorityTest(" : 세번째 스레드");
28
29     int priority_t1 = Integer.parseInt(args[0]);
30     int priority_t2 = Integer.parseIn
31
32     t1.setPriority(priority_t1);  //
33     t2.setPriority(priority_t2);
34     t3.setPriority(Thread.MIN_PRIORIT
35
36     t1.start();
37     t2.start();
38     t3.start();
39   }
40 }
```

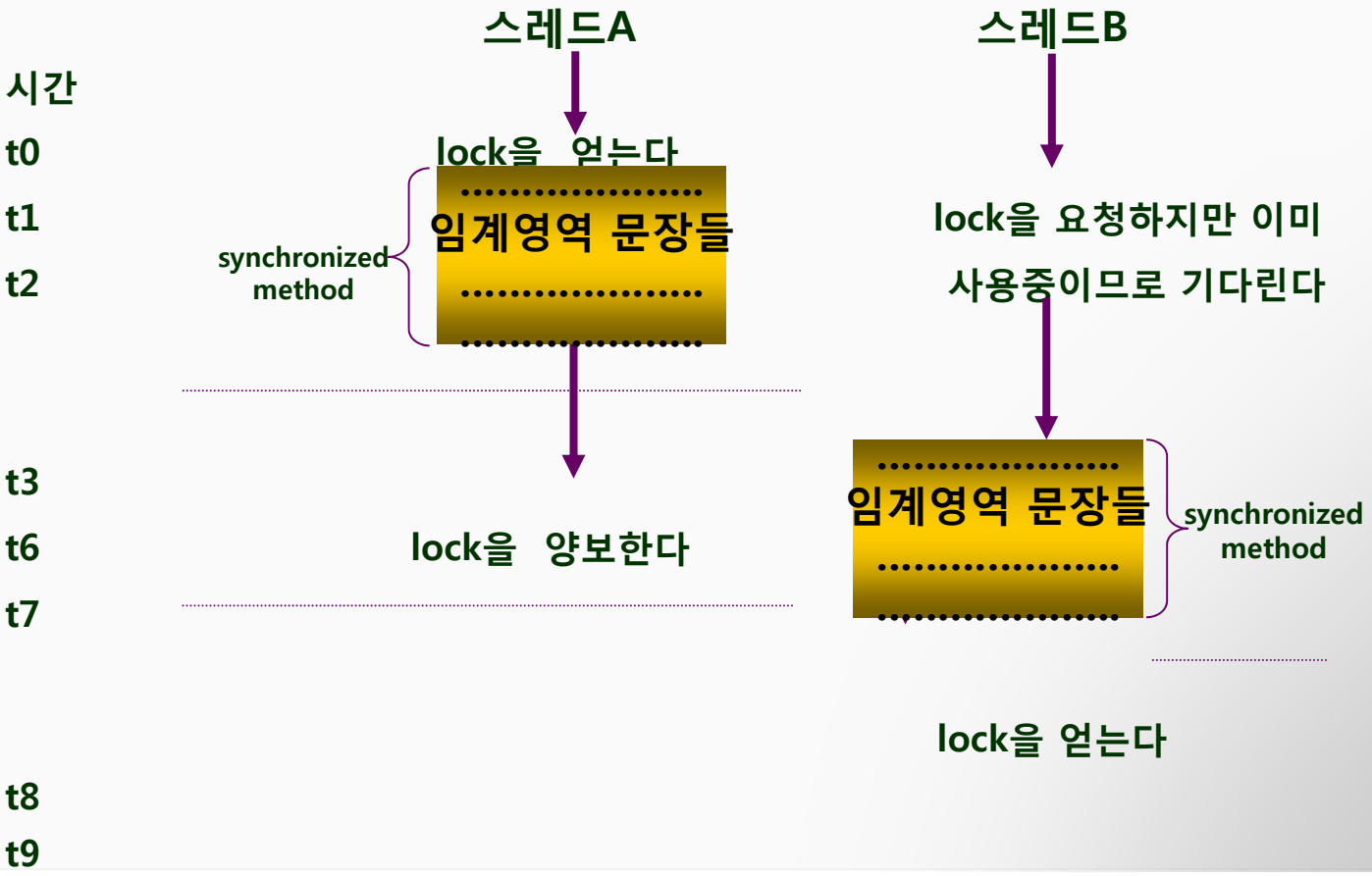
```
>java ThreadPriority  10  3
```

```
0 : 첫번째 스레드 우선순위 : 10
0 : 두번째 스레드 우선순위 : 3
0 : 세번째 스레드 우선순위 : 1
1 : 두번째 스레드 우선순위 : 3
1 : 첫번째 스레드 우선순위 : 10
1 : 세번째 스레드 우선순위 : 1
2 : 첫번째 스레드 우선순위 : 10
2 : 두번째 스레드 우선순위 : 3
2 : 세번째 스레드 우선순위 : 1
3 : 첫번째 스레드 우선순위 : 10
3 : 두번째 스레드 우선순위 : 3
3 : 세번째 스레드 우선순위 : 1
4 : 첫번째 스레드 우선순위 : 10
4 : 두번째 스레드 우선순위 : 3
4 : 세번째 스레드 우선순위 : 1
```

쓰레드 우선 순위

- **대부분의 응용 프로그램에서 다수개의 스레드가 공유할 수 있는 부분이 요구된다**
- **공유부분은 상호 배타적으로 사용되어야 한다**
- **임계영역(critical section)**
  - 상호배타적으로 사용되는 공유부분
  - 자바는 한 순간에 하나의 스레드만 실행할 수 있는 synchronized method 제공
  - 한 스레드가 synchronized method를 수행 중이면 다른 스레드는 대기

동기화(Synchronization)



동기화(Synchronization)

```
1  class Account {
2    private int total = 0;
3    synchronized void deposit(int amount) {
4    // 임계영역을 지정하는 synchronized 메소드
5      total += amount;
6    }
7    int gettotal() {
8      return total;
9    }
10 }
11
12 class Customer extends Thread {
13   Account account1;
14   Customer(Account account) {
15   // Customer 객체를 생성하는 생성자의 매개변수로 Account 객체를
16   // 받아들여 현재의 account1 객체에 연결(같은 주소를 가진다)
17     this.account1 = account;
18   }
19   public void run() {
20     try {
21       for(int i = 0; i < 200 ; i++) {
22         System.out.println("성금자 " + getName() + " : " + i +"번째");
23         account1.deposit(1000);
24         sleep(2);
25         if (account1.gettotal() >= 500000) break;
26       }
27     }
28     catch(Exception e) {
29       System.out.println(e);
30     }
31   }
32 }
```

동기화(Synchronization)

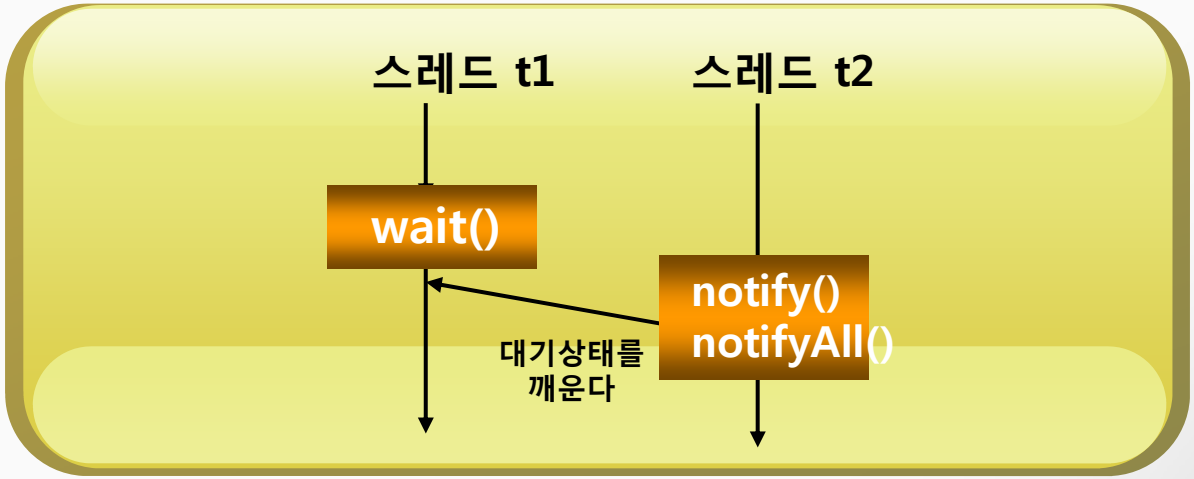
- **java.lang.Object 클래스에서는 스레드 사이의 통신을 위해 3개의 메소드를 제공**
  - wait() 메소드 : 스레드 수행중 이 메소드를 만나면 가지고 있는 lock을 양보하고 대기상태로 들어간다

```
void wait() throws InterruptedException
void wait(long msec) throws InterruptedException
void wait(long msec, int nsec) throws InterruptedException
        msec와 nsec는 대기 시간을 의미
```

```
notify() : 대기 상태의 스레드중에서 하나의 스레드를 깨운다
notifyAll() : 대기 상태의 모든 스레드를 깨운다
```

동기화를 이용한 스레드 사이의 통신

❖ **wait() 메소드와 notify(), notifyAll() 메소드의 동작**



동기화를 이용한 스레드 사이의 통신

```
1  class Producer extends Thread {
2    private Buffer blank;
3    public Producer(Buffer blank) {
4      this.blank = blank ;
5    }
6    public void run() {
7      for (int i=0; i<10;i++) {
8        blank.put(i);
9        // 생산자 스레드는 계속해서 put() 메소드 호출
10       try {
11         sleep((int)(Math.random() * 100 ));
12       } catch (InterruptedException e) { }
13     }
14   }
15 }
16
17 class Consumer extends Thread {
18   private Buffer blank;
19   public Consumer(Buffer c) {
20     blank = c;
21   }
22   public void run() {
23     int value = 0;
24     for (int i = 0 ; i < 10 ; i++ ) {
25       value = blank.get();
26       // 소비자 스레드는 계속 get() 메소드를 호출
27     }
28   }
```

동기화를 이용한 스레드 사이의 통신

```
class Buffer {
  private int contents;
  private boolean available = false;
  // flag 역할을 하는 available 변수의 초기값을 false로 설정
  // 처음에는 생산자가 먼저 데이터를 가져다 놓아야 한다

  public synchronized int get() {
  // 임계영역을 지정하는 메소드. 버퍼로부터 데이터를 get
    while (available == false ) {
      try {
        wait();
      } catch (InterruptedException e) {}
    }
    System.out.println("소비자########## : 소비 " + contents);
    notify();
    available = false;

    // 데이터를 반환하기전에 put()에 들어가기위해
    // 기다리는 메소드를 깨운다
    return contents;
  }
  public synchronized void put(int value) {
  // 임계영역을 지정하는 메소드. 버퍼에 데이터를 put
    while (available == true ) {
      try {
        wait();
      } catch (InterruptedException e) {}
    }
    contents = value;
    System.out.println("생산자########## : 생산 " + contents);
    notify();
    available = true;
    // 값을 저장하고 get() 메소드에 들어가기 위해
    // 대기하고 있는 스레드중에 하나를 깨운다
  }
}
```

동기화를 이용한 스레드 사이의 통신

```
68  public class ProducerConsumer {
69    public static void main(String args[]) {
70    Buffer c = new Buffer();
71    Producer p1 = new Producer(c);
72    Consumer c1 = new Consumer(c);
73    // 생산자와 소비자 스레드 객체를 생성
74    // 버퍼 객체 c를 매개변수로 생성한다
75    p1.start() ;
76    c1.start() ;
77    }
78  }
```

```
---------- Java Execution --------
생산자########## : 생산 0
소비자########## : 소비 0
생산자########## : 생산 1
소비자########## : 소비 1
생산자########## : 생산 2
소비자########## : 소비 2
생산자########## : 생산 3
소비자########## : 소비 3
생산자########## : 생산 4
소비자########## : 소비 4
생산자########## : 생산 5
소비자########## : 소비 5
생산자########## : 생산 6
소비자########## : 소비 6
생산자########## : 생산 7
소비자########## : 소비 7
생산자########## : 생산 8
소비자########## : 소비 8
생산자########## : 생산 9
소비자########## : 소비 9

출력 완료 (0초 경과) - 정상 종료
```

“Learn by studying examples”
“Learn by hand programming”