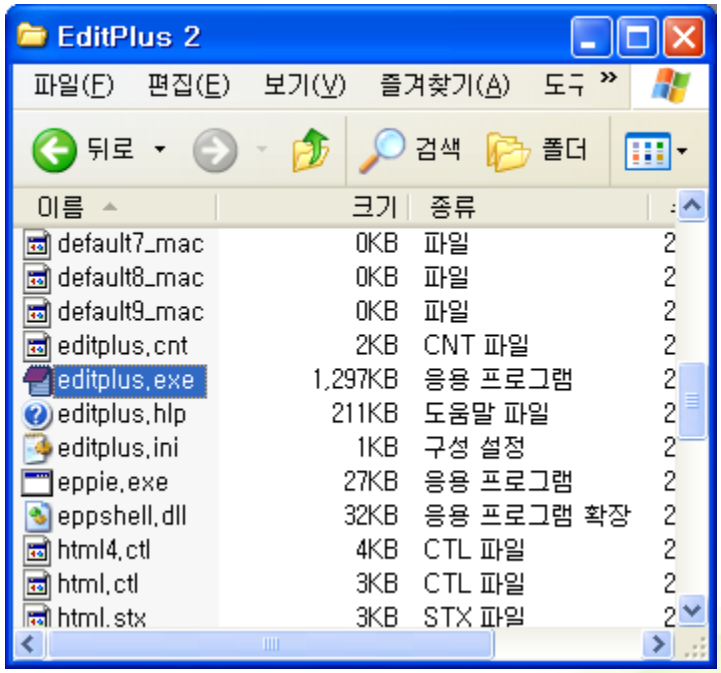
스 레 드(Thread)

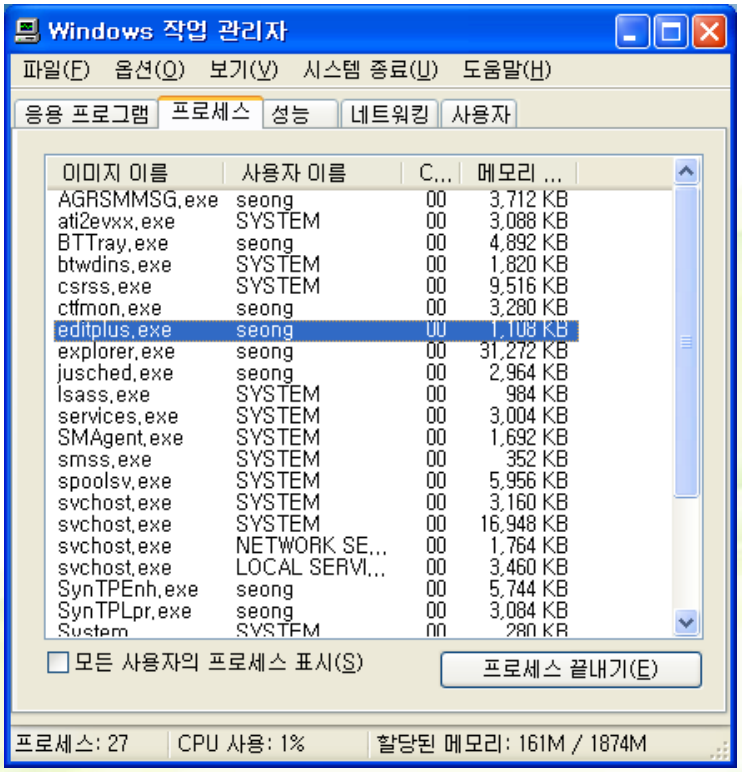
강사 : 강병준

프로세스와 쓰레드(process & thread)

▶ 프로그램 : 실행 가능한 파일(HDD)



▶ 프로세스 : 실행 중인 프로그램(메모리)



자바의 스레드

▶ 스레드란?

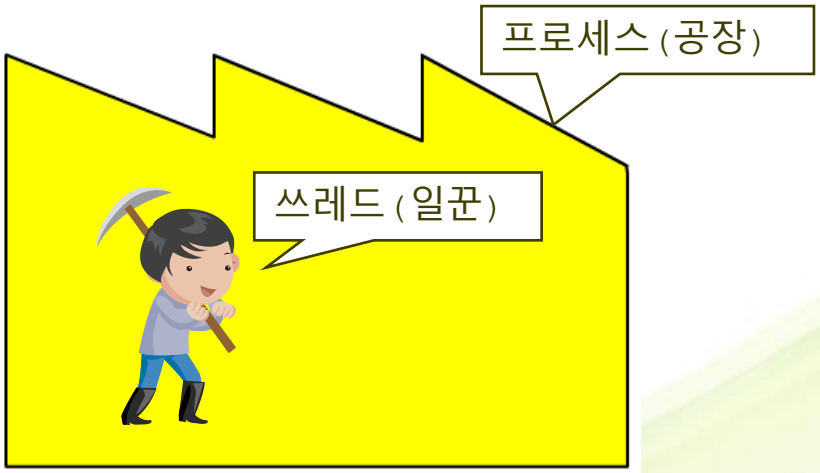
- 하나의 프로그램이 동시에 여러 개의 일을 수행할 수 있도록 해 주는것
- 순차적으로 동작하는 문장들의 단일 집합
- 경량(lightweight) 프로세스
- 다중 스레드
  - ✓ 하나의 프로세스(프로그램)에 하나 이상의 스레드를 생성하여 실행할 때
- 자바는 스레드를 지원하기 위해 java.lang.Thread 클래스 제공

프로세스와 쓰레드(process & thread)

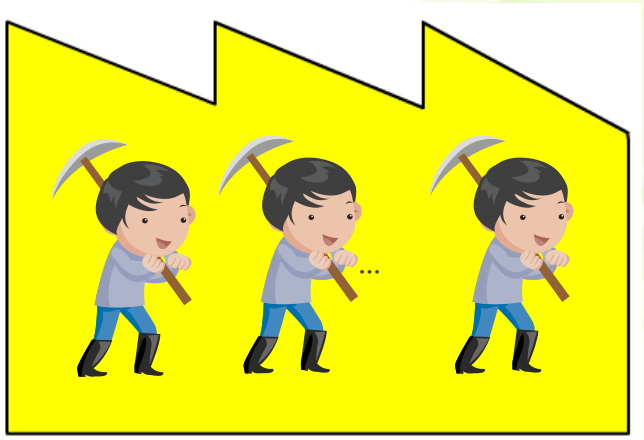
- 프로세스 : 실행 중인 프로그램, 자원(resources)과 쓰레드로 구성
- 쓰레드 : 프로세스 내에서 실제 작업을 수행.
  모든 프로세스는 하나 이상의 쓰레드를 가지고 있다.

프로세스 : 쓰레드 = 공장 : 일꾼

- 싱글 쓰레드 프로세스
  = 자원+쓰레드



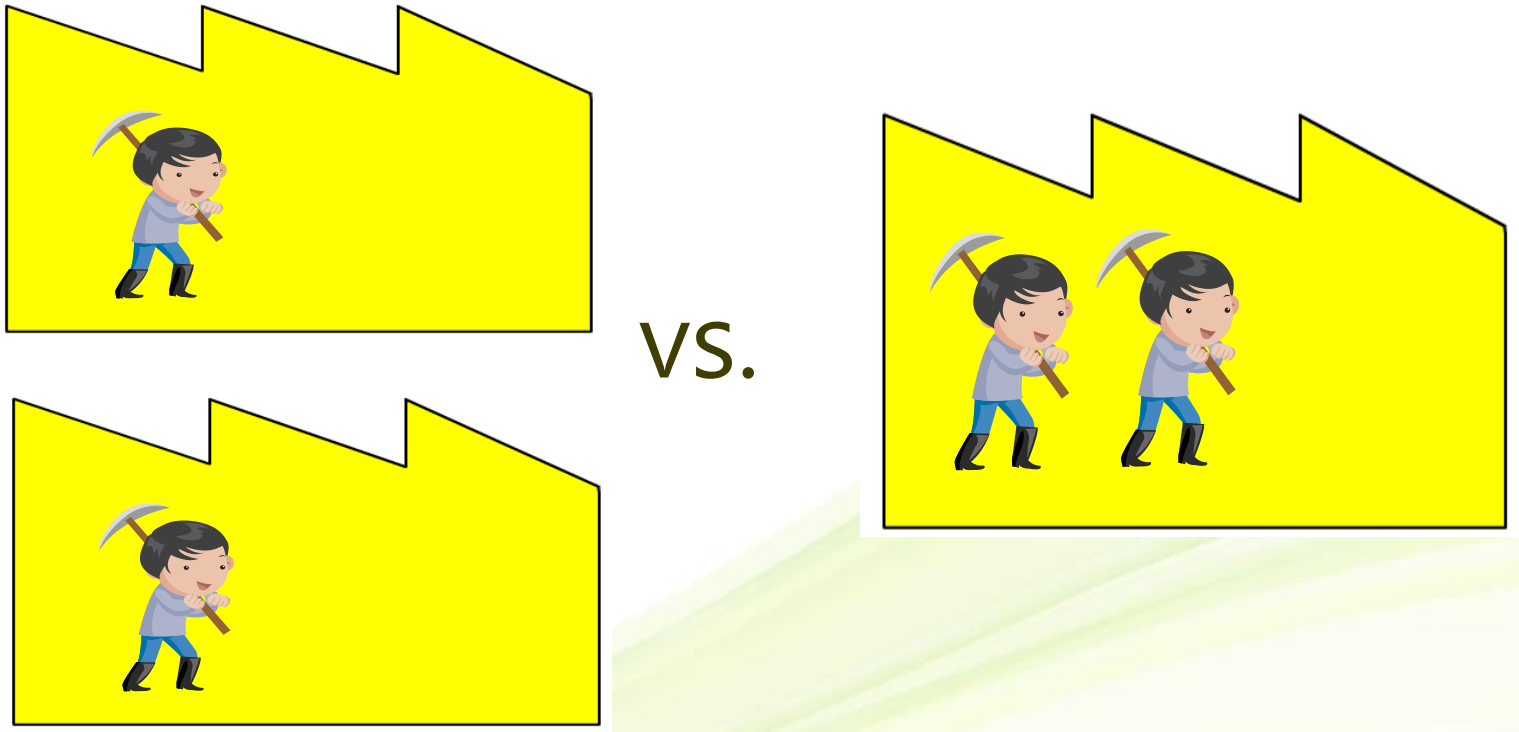
- 멀티 쓰레드 프로세스
  = 자원+쓰레드+쓰레드+…+쓰레드



멀티프로세스 vs. 멀티쓰레드

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

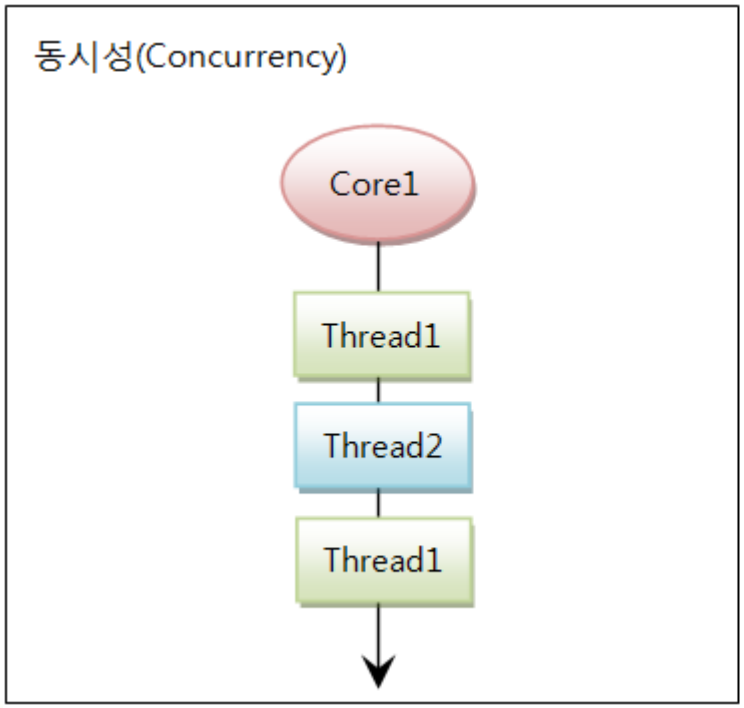
- 2 프로세스 1 쓰레드 vs. 1 프로세스 2 쓰레드

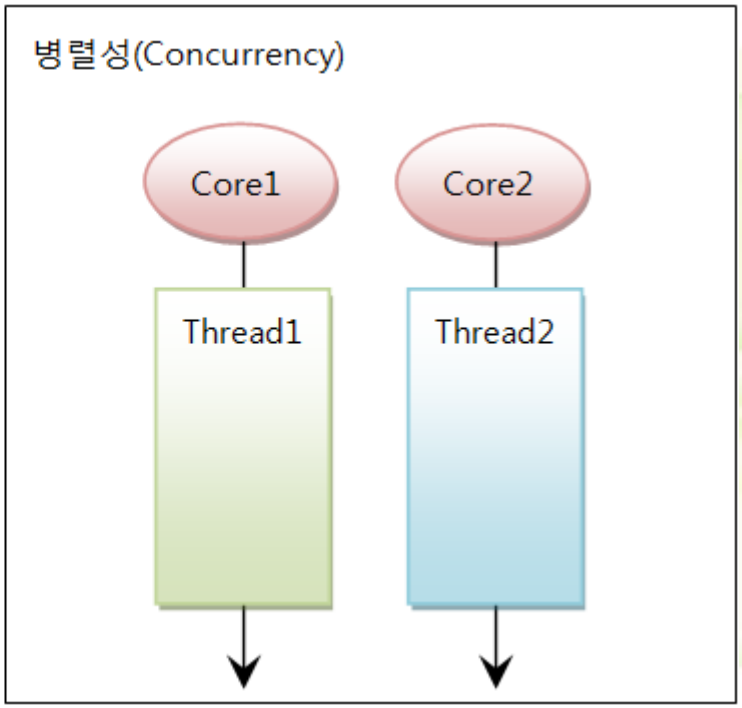


멀티쓰레드

❖ 동시성과 병렬성

- 동시성
  - 멀티 작업 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질
- 병렬성
  - 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질





멀티쓰레드의 장단점

**"많은 프로그램들이 멀티쓰레드로 작성되어 있다.
그러나, 멀티쓰레드 프로그래밍이 장점만 있는 것은 아니다."**

장점	- 자원을 보다 **효율적**으로 사용할 수 있다. - 사용자에 대한 **응답성(responseness)이 향상**된다. - 작업이 분리되어 **코드가 간결**해 진다. **"여러 모로 좋다."**
단점	- 동기화(synchronization)에 주의해야 한다. - 교착상태(dead-lock)가 발생하지 않도록 주의해야 한다. - 각 쓰레드가 효율적으로 고르게 실행될 수 있게 해야 한다. **"프로그래밍할 때 고려해야 할 사항들이 많다."**

Thread 클래스

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

Thread()

Thread(String *s*)

Thread(Runnable *r*)

Thread(Runnable *r*, String *s*)

Thread 클래스의 메서드

void sleep(long *msec*) throws InterruptedException	*msec*에 지정된 밀리초(milliseconds) 동안 대기
void sleep(long *msec*, int *nsec*) throws InterruptedException	*msec*에 지정된 밀리초+*nsec*에 지정된 나노초(nanoseconds) 동안 대기
String getName()	스레드의 이름을 반환
void setName(String *s*)	스레드의 이름을 *s*로 설정
void start()	스레드를 시작시킨다. run() 메소드를 호출
int getPriority()	스레드의 우선 순위를 반환
void setPriority(int *p*)	스레드의 우선 순위를 *p*값으로 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.

스레드의 생성

- 스레드를 생성하는 2가지 방법
  - ✓ Thread 클래스로부터 직접 상속받아 스레드를 생성
  - ✓ Runnable 인터페이스를 사용하는 방법(현재의 클래스가 이미 다른 클래스로부터 상속 받고 있는 경우)

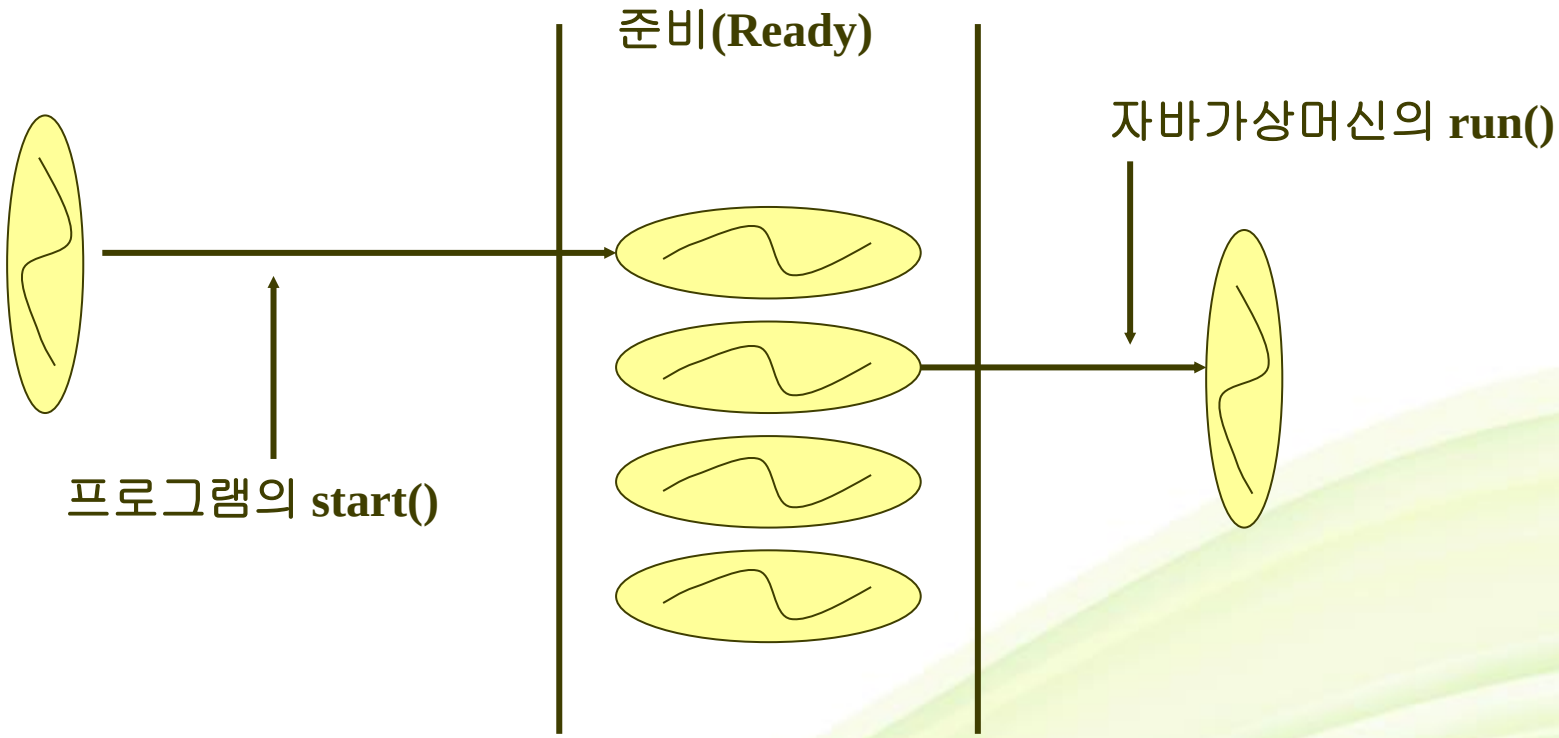
Thread 클래스 이용

- Thread 클래스를 상속받은 뒤, 해당 스레드에서 지원하고 싶은 코드를 run()메서드에서 오버라이딩 해준다.
- 해당 스레드를 생성한 후, 스레드 객체의 start()메서드를 호출한다.

```
class ThreadA extends Thread  {  // Thread 클래스로부터 상속
  .........
  public void run() {
    ....  // 상위 클래스인 Thread 클래스의 run() 메서드를 오버
    ....  //라 이딩하여 스레드가 수행하여야 하는 문장들을 기술
   }
  .......
}
```

```
ThreadA ta = new ThreadA();
ta.start();
// 스레드 객체를 생성하여 스레드를
// 시작시킨다
```

run()메서드와 start()메서드의 관계



ThreadTest.java

```
class ThreadTest extends Thread{
    ThreadTest(String s){
        super(s);
    }
    public void run(){
        for(int i=0;i<100;i++)
            System.out.println(getName()+" "+i);
        }
public static void main(String ar[]){
        ThreadTest a=new ThreadTest("th1");
        ThreadTest b=new ThreadTest("th2");
        a.start();
        b.start();
        System.out.println("End");
    }
}
```

Runnable 인터페이스 이용

- 현재의 클래스가 이미 다른 클래스로부터 상속을 받고 있다면 Runnable 인터페이스를 사용하여 스레드를 생성할 수 있다
- Runnable 인터페이스는 JDK에 의해 제공되는 라이브러리 인터페이스이다
- Runnable 인터페이스에는 run() 메소드만 정의되어 있다

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

▶ Runnable 인터페이스 이용

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

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

Runnable 인터페이스 이용

- **Runnable** 인터페이스를 이용하여 스레드를 생성하는 방법

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

또는

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

```
class A extends Thread {//단일상속
    public void run() {//main Method 이다.
            System.out.println("A");         }
}
class B extends Object implements Runnable {//다중상속
    public void run() {//main Method 이다.
            System.out.println("B");         }
}
public class Exam_01 {
    public static void main(String[] ar) {
            System.out.println("main start!");
            A ap = new A();
            ap.start();//Thread 호출          //ap.run();//Method 호출
            B bp = new B();
            Thread th = new Thread(bp);
            th.start();//Thread 호출          //bp.run();//Method 호출
            for(int i = 1; i < 100; i++) {
                    System.out.print(i);
                    if(i % 10 == 0) System.out.println();
                    else System.out.print("\t");
            }
     }
}
```

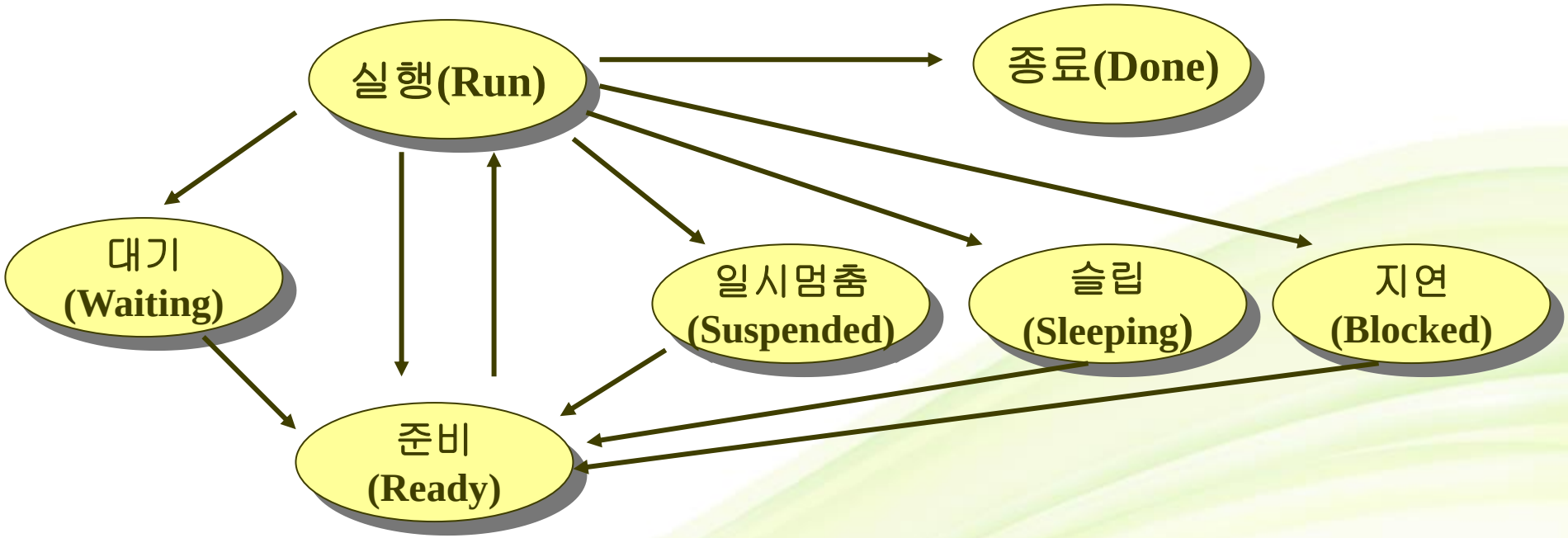
```
class RunnableExam01  implements Runnable {
    public void run( ){
            int a=0;
            System.out.println(">> run 메소드 진입 <<");
            while(true){
                try{       Thread.sleep(200);      //0.3초간 대기
                }catch(InterruptedException e){   e.printStackTrace();   }
                System.out.println(Thread.currentThread( ).getName( )+" : " + ++a);
                if(a>=5)    break;
            }
            System.out.println(">> run 메소드 종료 <<");
     }
}
public class Ex01 {
     public static void main(String[] args) {
            RunnableExam01 r1=new RunnableExam01();
            RunnableExam01 r2=new RunnableExam01();
            RunnableExam01 r3=new RunnableExam01();
            Thread t1=new Thread(r1, "첫 번째 스레드");
            Thread t2=new Thread(r2, "두 번째 스레드");
            Thread t3=new Thread(r3, "세 번째 스레드");
             t1.start();      t2.start();                t3.start();
      }
}
```

```
class ThreadEx1 {
    public static void main(String args[]) {
            ThreadEx1_1 t1 = new ThreadEx1_1();
            Runnable r = new ThreadEx1_2();
            Thread t2 = new Thread(r);      // 생성자 Thread(Runnable target)
            t1.start();                t2.start();
    }
}
class ThreadEx1_1 extends Thread {
    public void run() {
            for(int i=0; i < 5; i++) {
                    System.out.println(getName()); // 조상인 Thread의 getName()을 호출
            }
    }
}
class ThreadEx1_2 implements Runnable {
     public void run() {
            for(int i=0; i < 5; i++) { // Thread.currentThread() - 현재 실행중인 Thread를 반환
                System.out.println(Thread.currentThread().getName());
            }
    }
}
```

스레드의 제어

▶ 스레드의 상태도

- 모든 스레드는 상태도에 나타난 상태 중의 한 군데에 반드시 속하게 되고, 메소드 호출이나 상황에 의해 화살표 방향으로 상태가 변화하게 된다.



스레드의 제어

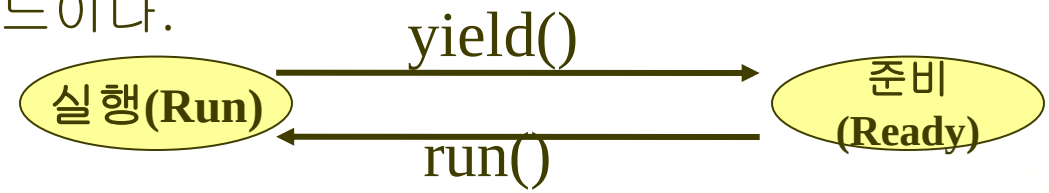
스레드의 상태

- **실행(Run)**: run()메소드 내의코드를 실행하고 있는 상태
- **대기**: 실행을 기다리고 있는 상태
  - ✓ **대기(Waiting)**: 뒤에서 배울 동기화(Synchronized)에 의한 대기상태
  - ✓ **슬립(Sleeping)**: CPU의 점유를 중지하고, 아무것도 안하고 있는 상태
  - ✓ **일시멈춤(Suspended)**: 일시중지 상태
  - ✓ **지연((Blocked)**: 입출력 메서드 등의 호출로 인해서, 메서드의 종료가 일어날 때까지 스레드가 대기하고 있는 상태
- **준비(Ready)**: 실행상태에 들어가기 위해, 준비하고 있는 상태. 스레드 스케줄러에 의해 선택된 스레드가 실행 상태에 들어가게 된다.
- **종료(Done)**: run() 메서드나 stop()메서드에 의해 스레드의 실행이 완료된 상태

스레드의 제어

▶메서드 호출에 의한 스레드 제어

- start() 메서드: 스레드는 준비상태에 들어간다
- run()메서드: 스레드가 살아있는 동안 계속해서 실행하는 코드를 갖고 있으며, 스레드가 실행상태에 있는동안 수행된다.
- yield()메서드: 다른 스레드에게 실행상태를 양보하고, 자신은 대기상태로 바꾸는 스레드이다.

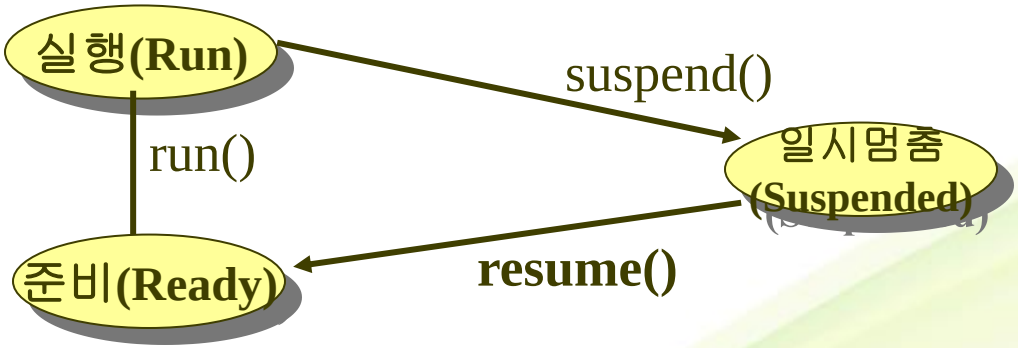


- sleeping()메서드: 실행상태에서 슬립상태로 들어감.
  - ✓ 보통 슬립메서드는 만분의 일초 단위의 슬립 시간을 지정하는데, 이 시간동안에 스레드는 아무동작도 하지 않으며, 시간이 지나면, 스레드는 준비상태로 들어가게 된다.

스레드의 제어

▶메서드 호출에 의한 스레드 제어

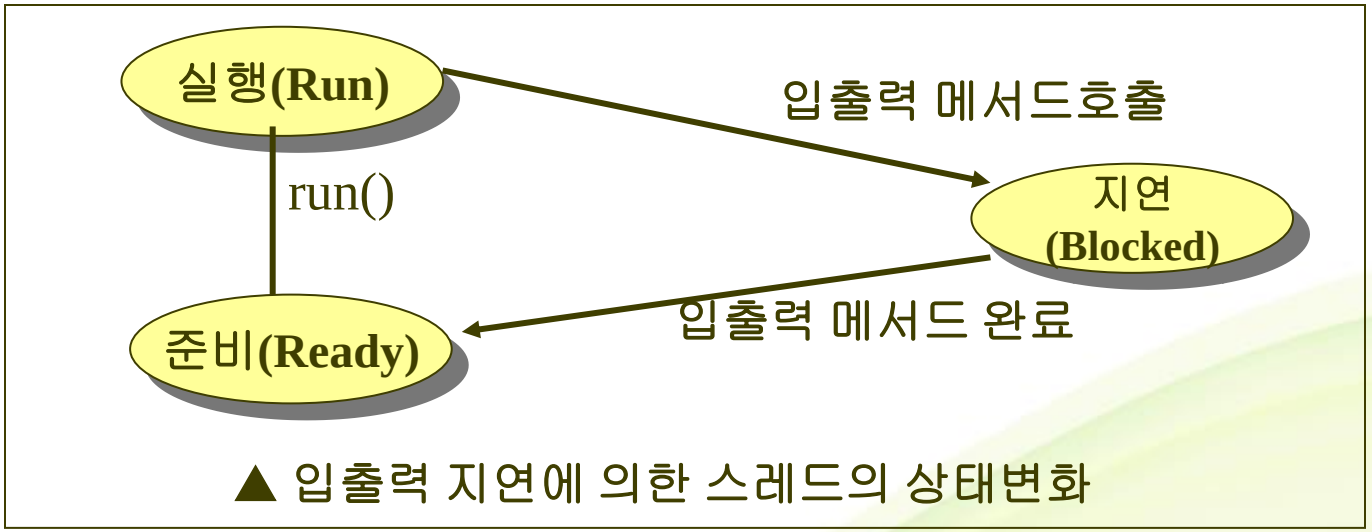
- Suspend()와 resume()메서드: 스레드의 실행을 잠시 중단하고, 다시 실행시키는 제어를 하는데 사용된다.
  - ✓ Suspend()메서드: 일시멈춤상태로 들어가게 된다.
  - ✓ resume()메서드: 스레드가 준비상태로 들어가서 스레드 스케줄러의 선택을 기다리게 된다.



- stop()메서드: 실행상태에서 강제로 종료 상태 쓰레드로 바꾼다. 이때 스레드는 실행이 중지되고, 종료된 스레드가 된다.
  - ✓ 스레드가 종료된 후에는 해당 스레드를 절대로 다시 실행시킬수 없다. 다시 실행시키려면, 스레드를 새로 생성하는 수밖에는 없다.

지연과 대기

- **지연이란?** 스레드가 입출력과 관련된 메서드를 호출할때, 이 메서드가 완료될 때까지 스레드가 들어가 있는 상태



▲ 입출력 지연에 의한 스레드의 상태변화

- **대기?** Wait()메서드에 의해 특정 이벤트를 기다리는 상태.
  - ✓ 아무때나 호출이 불가능하고, 여러 개의 스레드가 개입된 동기화 코드 내에서만 호출이 가능.

join()메서드

특정한 쓰레드가 실행된 후에 다른 쓰레드가 실행

```
import java.util.ArrayList;
class BeforeThread1 extends Thread{
    public BeforeThread1(String name) { super(name); }
    private void addCar(){
        System.out.println("addCar");          JoinTest2.carList.add("그랜저");
        JoinTest2.carList.add("소나타");      JoinTest2.carList.add("K9");
        JoinTest2.carList.add("SM7");
    }
    public void run() {
        System.out.println(currentThread().getName() + "실행");
        try{ sleep(1000); } catch(InterruptedException e){ e.printStackTrace(); }
        addCar();
    }
}
```

```
class AfterThread1 extends Thread{
    public AfterThread1(String name) {super(name);}
    public void run() {
        System.out.println(currentThread().getName() + "실행");
        ArrayList<String> carList = JoinTest2.carList;
        for(int i=0;i<carList.size();i++){
            System.out.println(carList.get(i));
        }
    }
}
public class JoinTest2 {
    public static ArrayList<String> carList = new ArrayList<String>();
    public static void main(String[] args) {
        BeforeThread1 beforeThread = new BeforeThread1("beforeThread");
        AfterThread1 afterThread = new AfterThread1("afterThread");
        beforeThread.start();
        try{   beforeThread.join();
        }   catch(InterruptedException e)  {  e.printStackTrace();  }
        afterThread.start();
    }
}
```

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 비어있어야한다.
int getMaxPriority()	쓰레드 그룹의 최대우선순위를 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 최대우선순위를 설정한다.

쓰레드 우선순위

상수 : MIN_PRIORITY, MAX_PRIORITY, NORM_PRIORITY

```
class ImportantThread1 extends Thread{
    public ImportantThread1(String name) {   super(name);    }
   public void run() {
            for(int i=1;i<=15;i++){  System.out.println(currentThread().getName());          }
   }
}
class NotImportantThread1 extends Thread{
    public NotImportantThread1(String name) {    super(name);          }
    public void run() {
        for(int i=1;i<=15;i++){              System.out.println(currentThread().getName());   }
   }
}
public class PriorityTest2 {
    public static void main(String[] args) {
            ImportantThread1 it = new ImportantThread1("중요한 작업");
            NotImportantThread1 nit = new NotImportantThread1("중요하지 않은 작업");
            it.setPriority(Thread.MAX_PRIORITY);   nit.setPriority(Thread.MIN_PRIORITY);
            nit.start();              it.start();
    }
}
```

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 true를 반환한다.
void setDaemon(boolean on)  - 쓰레드를 데몬 쓰레드로 또는 사용자 쓰레드로 변경한다.
                              (매개변수 on의 값을 true로 지정하면 데몬 쓰레드가 된다.)
```

* setDaemon(boolean on)은 반드시 start()를 호출하기 전에 실행되어야 한다.
  그렇지 않으면 IllegalThreadStateException이 발생한다.

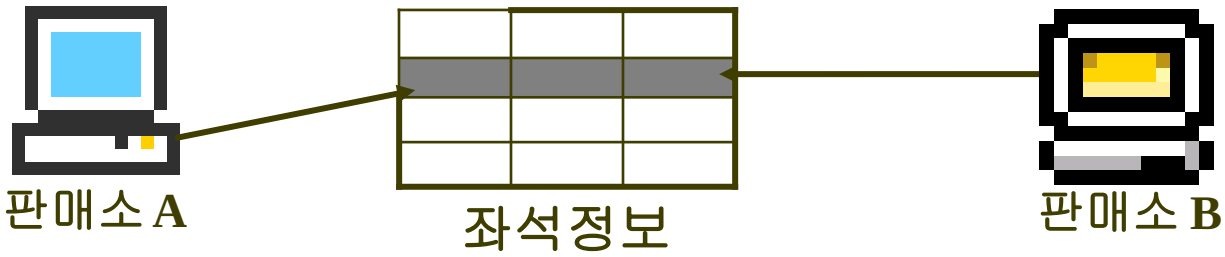
```
public class Daemo extends Thread{
   public static void main(String[] args) {
        Daemo t = new Daemo();
        t.setDaemon(true);// 이 부분이 없으면 종료되지 않는다.
        t.start();
        for(int i=1; i <= 20; i++) {
             try{ Thread.sleep(1000);
             } catch(InterruptedException e) {}
            System.out.println(i);
        }
        System.out.println("프로그램을 종료합니다.");
    }
    public void run() {
       while(true) {
            try { Thread.sleep(3 * 1000);// 3초마다
            } catch(InterruptedException e) {}
           System.out.println("작업파일이 자동저장되었습니다.");
      }
   }
}
```

동기화(Synchronization)

- **동기화(Synchronization)란?** 하나의 자원을 여러 태스크가 사용하려 할 때에, 한 시점에서 하나의 태스크만이 사용할 수 있도록 하는 것
- **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 대기한다

동기화(Synchronization)

▼동기화 예 - 좌석예매 시스템



네트워크를 이용하여 여러매표소에서 좌석이 예매될 때, 매표소들 간에 좌석 예매시스템의 좌석 정보를 동시에 접근할 수 있다면, 같은 좌석을 여러 사람에게 중복되게 예매할 수도 있다. 따라서, 이렇게 여러 태스크들이 하나의 정보에 접근 할때에는 , 한번에 하나씩의 태스크만이 접근 할 수 있도록 해야 한다.

동기화 메소드와 동기화 블록

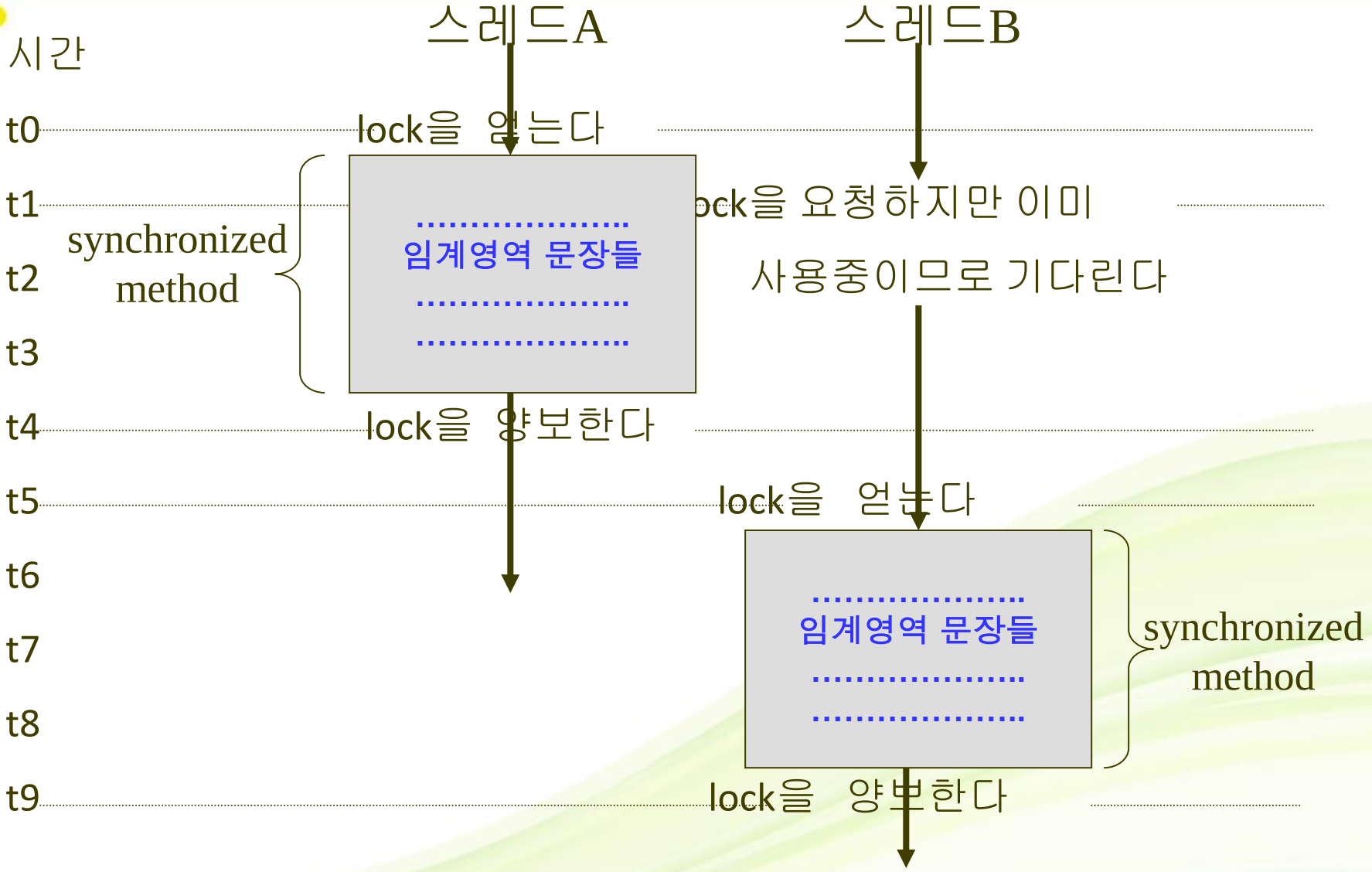
- 동기화 메소드 및 동기화 블록 – synchronized
  - 단 하나의 스레드만 실행할 수 있는 메소드 또는 블록
  - 다른 스레드는 메소드나 블록이 실행이 끝날 때까지 대기해야
  - 동기화 메소드

```
public synchronized void method() {
    임계 영역;  //단 하나의 스레드만 실행
}
```

  - 동기화 블록

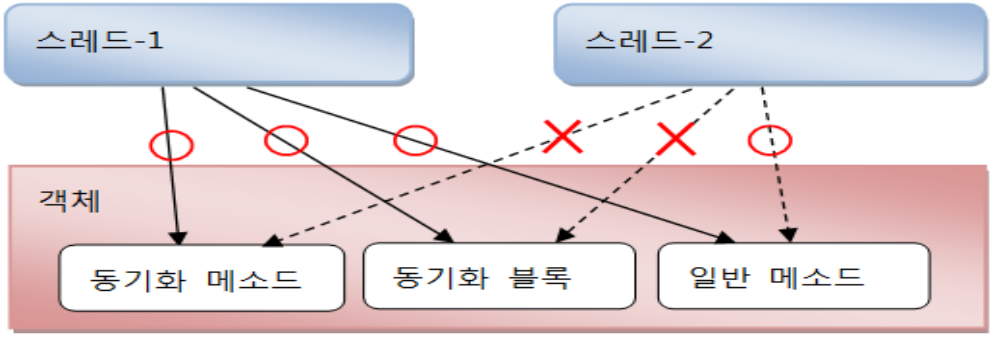
```
public void method () {
   //여러 스레드가 실행 가능 영역
   ...
   synchronized(공유객체) {
      임계 영역 //단 하나의 스레드만 실행
   }
   //여러 스레드가 실행 가능 영역
   ...
}
```

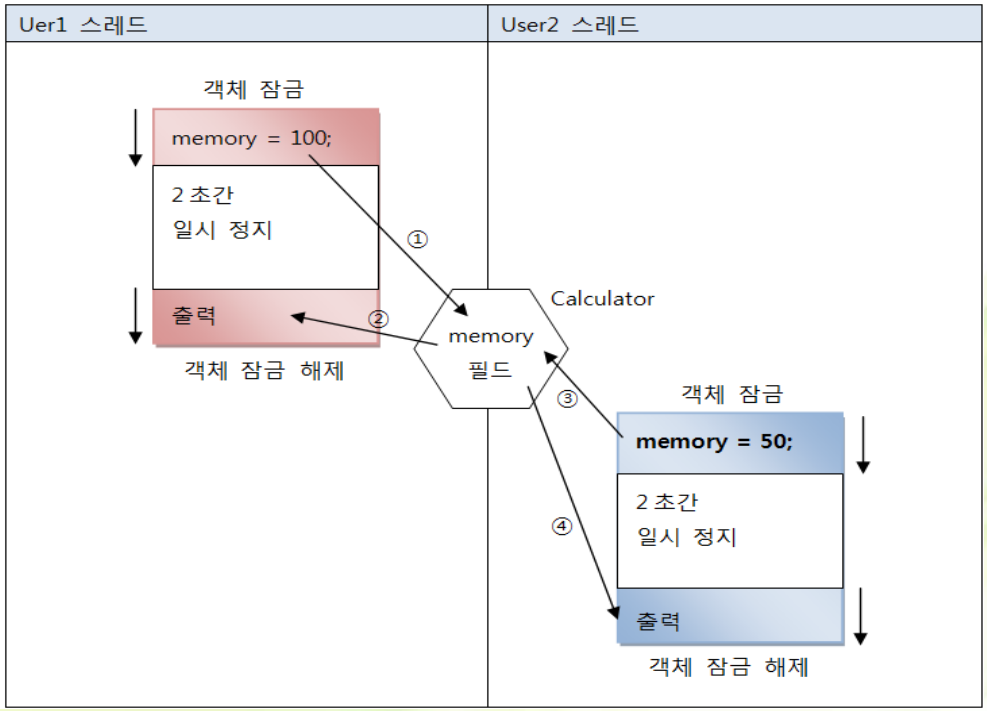
synchronized method 수행



동기화 메소드와 동기화 블록

❖ 동기화 메소드 및 동기화 블록





동기화 메소드

```
public class Account {
        private int total;
        public Account(int total) {    this.total = total;    }
        synchronized void deposit(int amt, String name) {
                total += amt;
                System.out.println(name+"입금 : " + amt);
        }
        synchronized void withdraw(int amt, String name) {
                if (amt <= total) {
                        total-= amt;
                        System.out.println(name+"출금 : "+amt);
                }       else System.out.println("돈 떨어졌어");
        }
        void print() {
                System.out.println("현재 잔액 : " + total);
        }
}
```

동기화 메소드

```
public class AccountUser extends Thread {
        boolean flag;  Account act; String name;
        public AccountUser(String name, Account act) {
                super(name);  this.act = act;  this.name = name;
        }
        public void run() {
                for(int i=0; i< 5; i++) {
                        int amt = (int)(Math.random()*10000) + 1;
                        if (flag) act.deposit(amt,name);
                        else act.withdraw(amt,name);
                        flag = !flag;   act.print();
                }
        }
}
```

동기화 메소드

```
public class AccountEx {
        public static void main(String[] args) {
                Account act = new Account(10000);
                AccountUser au1 = new AccountUser("수지", act);
                AccountUser au2 = new AccountUser("원빈", act);
                AccountUser au3 = new AccountUser("설현", act);
                AccountUser au4 = new AccountUser("강동원", act);
                AccountUser au5 = new AccountUser("하니", act);
                au1.start();
                au2.start();
                au3.start();
                au4.start();
                au5.start();
        }
}
```

동기화 블록

```
public class Account2 {
        private int total;
        public Account2(int total) {
                this.total = total;
        }
        void deposit(int amt, String name) {
                total += amt;
                System.out.println(name+"입금 : " + amt);
        }
        void withdraw(int amt, String name) {
                if (amt <= total) {
                        total-= amt;
                        System.out.println(name+"출금 : "+amt);
                }       else System.out.println("돈 떨어졌어");
        }
        void print() {
                System.out.println("현재 잔액 : " + total);
        }
}
```

동기화 블록

```
public class AccountUser2 extends Thread {
        boolean flag;
        Account2 act;
        String name;
        public AccountUser2(String name, Account2 act) {
                super(name);
                this.act = act;
                this.name = name;
        }
        public void run() {
                for(int i=0; i< 5; i++) {
                        synchronized (act) {
                                int amt = (int)(Math.random()*10000) + 1;
                                if (flag) act.deposit(amt,name);
                                else act.withdraw(amt,name);
                        }
                        flag = !flag;
                        act.print();
                }
        }
}
```

동기화 블록

```
public class AccountEx2 {
        public static void main(String[] args) {
                Account2 a2 = new Account2(10000);
                AccountUser2 au1 = new AccountUser2("원빈", a2);
                AccountUser2 au2 = new AccountUser2("이민우", a2);
                AccountUser2 au3 = new AccountUser2("공유", a2);
                AccountUser2 au4 = new AccountUser2("하니", a2);
                AccountUser2 au5 = new AccountUser2("설현", a2);
                au1.start();
                au2.start();
                au3.start();
                au4.start();
                au5.start();
        }
}
```

연습문제

1. 이 코드를 Runnable인터페이스를 구현하도록 변경하시오.

```
class Ex01 {
    public static void main(String args[]) {
        Thread1 th1 = new Thread1();
        th1.start();
    }
}

class Thread1 extends Thread {
     public void run() {
          for(int i=0; i < 300; i++) {
                System.out.print('-');
          }
     }
}
```