

12-1. 멀티 쓰레드

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 쓰레드
- 메인 쓰레드
- 작업 쓰레드 생성과 실행
- 동기화 메소드
- 키워드로 끝내는 핵심 포인트
- 확인문제

시작하기 전에

[핵심 키워드] : 프로세스, 멀티 쓰레드, 메인 쓰레드, 작업 쓰레드, 동기화 메소드

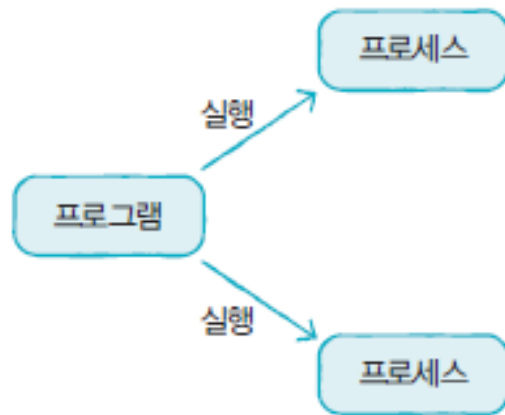
[핵심 포인트]

애플리케이션을 실행하면 운영체제로부터 실행에 필요한 메모리를 할당받아 애플리케이션이 실행되는데, 이것을 프로세스라 한다. 그리고 프로세스 내부에서 코드의 실행 흐름을 쓰레드라 한다. 애플리케이션 개발에 필수 요소인 쓰레드에 대해 살펴본다.

시작하기 전에

❖ 프로세스 (process)

- 실행 중인 하나의 애플리케이션
- 애플리케이션이 실행되면 운영체제로부터 실행에 필요한 메모리 할당받아 코드를 실행함
- 멀티 프로세스 역시 가능함



The screenshot shows the Windows Task Manager window titled '작업 관리자' (Task Manager). The '프로세스' (Processes) tab is selected. The table lists running processes, including foreground applications and background services.

이름	상태	9% CPU	24% 메모리
애플리케이션 (3)			
> 메모장		0.4%	2.2MB
> 메모장		0.1%	2.2MB
> 작업 관리자		1.3%	34.2MB
백그라운드 프로세스 (84)			
AcroTray(32비트)		0%	1.3MB
Activation Licensing Service(32...		0%	1.6MB
Adobe Acrobat Update Service(...		0%	0.9MB

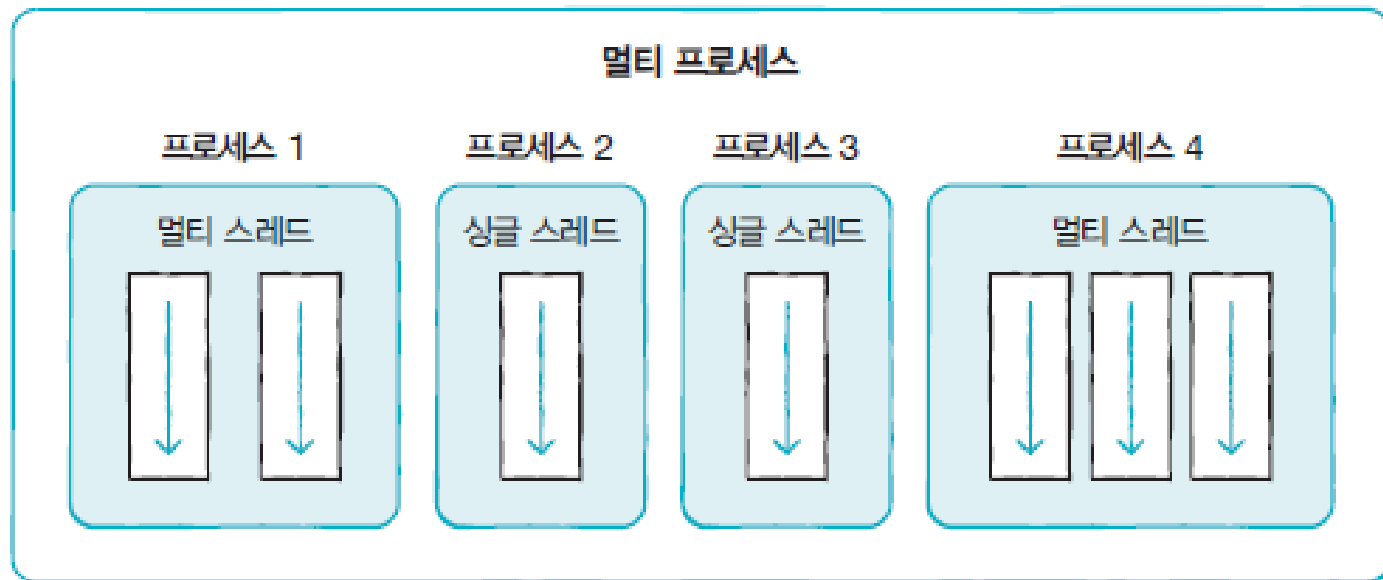
쓰레드

❖ 쓰레드 (thread)

- 한 가지 작업을 실행하기 위해 순차적으로 실행할 코드를 이어놓은 것
- 하나의 쓰레드는 하나의 코드 실행 이름

❖ 멀티 쓰레드 (multi thread)

- 하나의 프로세스로 두 가지 이상의 작업을 처리
- 데이터 분할하여 병렬로 처리하거나 다수 클라이언트 요청 처리하는 서버 개발하는 등의 용도
- 한 쓰레드가 예외 발생시킬 경우 프로세스 자체가 종료될 수 있음



메인 쓰레드

❖ 메인 쓰레드 (main thread)

- 모든 자바 애플리케이션은 메인 쓰레드가 main() 메소드 실행하면서 시작됨
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행

```
public static void main(String[] args) {  
    String data = null;  
    if(...) {  
    }  
    while(...) {  
    }  
    System.out.println("...");  
}
```

코드의 실행 흐름 → 쓰레드

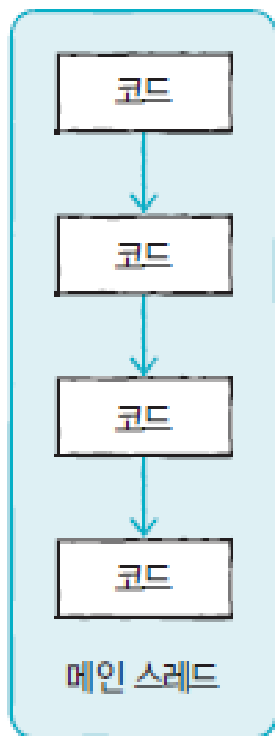


- 필요에 따라 작업 쓰레드들 만들어 병렬로 코드 실행 가능
- 멀티 쓰레드 애플리케이션에서는 실행 중인 쓰레드 하나라도 있으면 프로세스 종료되지 않음

메인 스레드

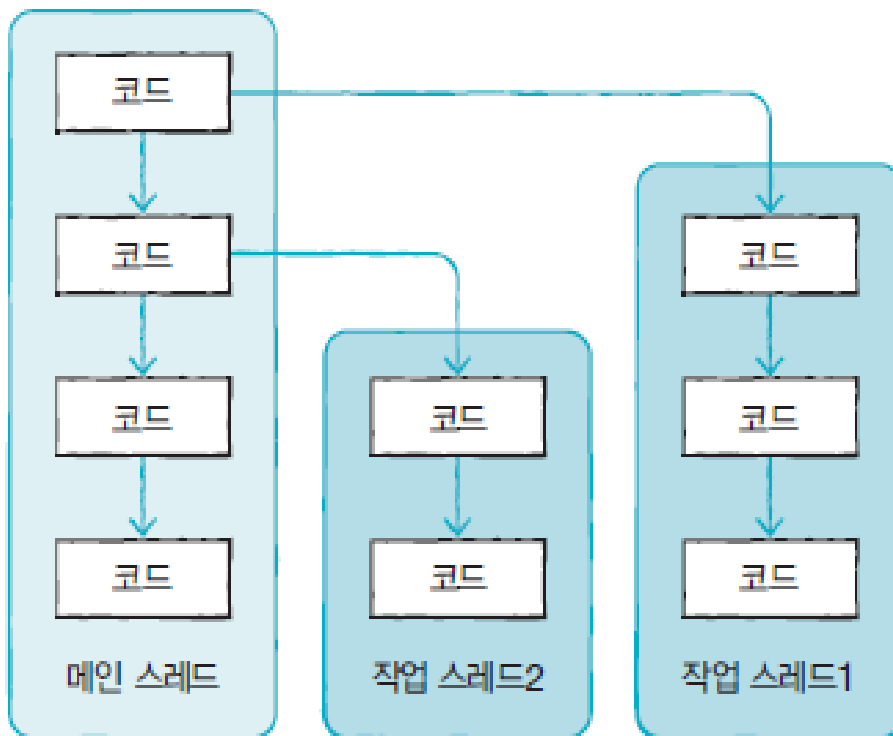
싱글 스레드 애플리케이션

프로세스



멀티 스레드 애플리케이션

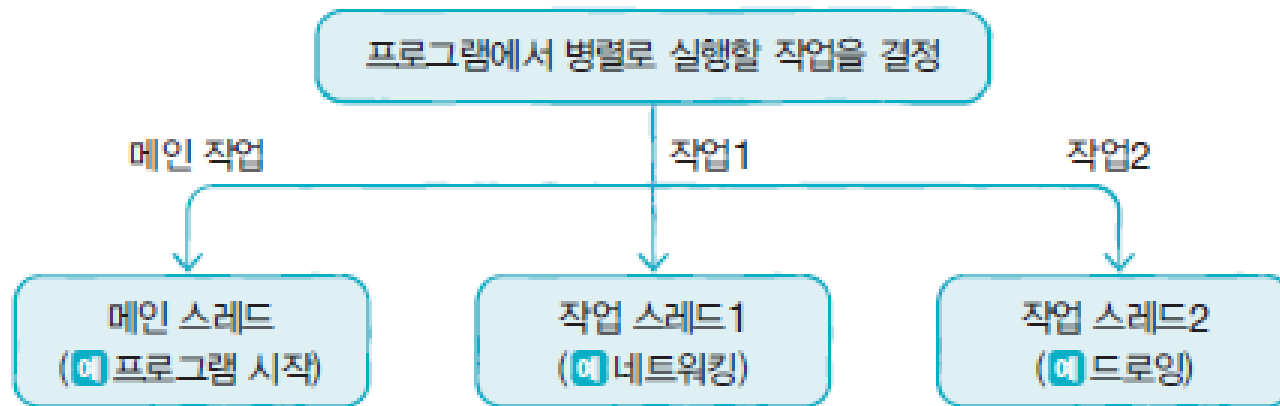
프로세스



작업 스레드 생성과 실행

❖ 작업 스레드

- 멀티 스레드로 실행하는 애플리케이션 개발하려면 몇 개의 작업을 병렬로 실행할지 우선 결정한 뒤 각 작업별로 스레드 생성해야
- 작업 스레드 역시 객체로 생성되므로 클래스 필요
 - Thread 클래스 상속하여 하위 클래스 만들어 사용할 수 있음



작업 스레드 생성과 실행

❖ Thread 클래스로부터 직접 생성

- Runnable을 매개값으로 갖는 생성자 호출

```
Thread thread = new Thread(Runnable target);
```

- 구현 객체 만들어 대입 필요

```
class Task implements Runnable {  
    public void run() {  
        스레드가 실행할 코드;  
    }  
}
```

- 구현 객체 매개값으로 Thread 생성자 호출하면 작업 스레드 생성됨

```
Runnable task= new Task();  
                        ↓  
Thread thread = new Thread(task);
```

작업 스레드 생성과 실행

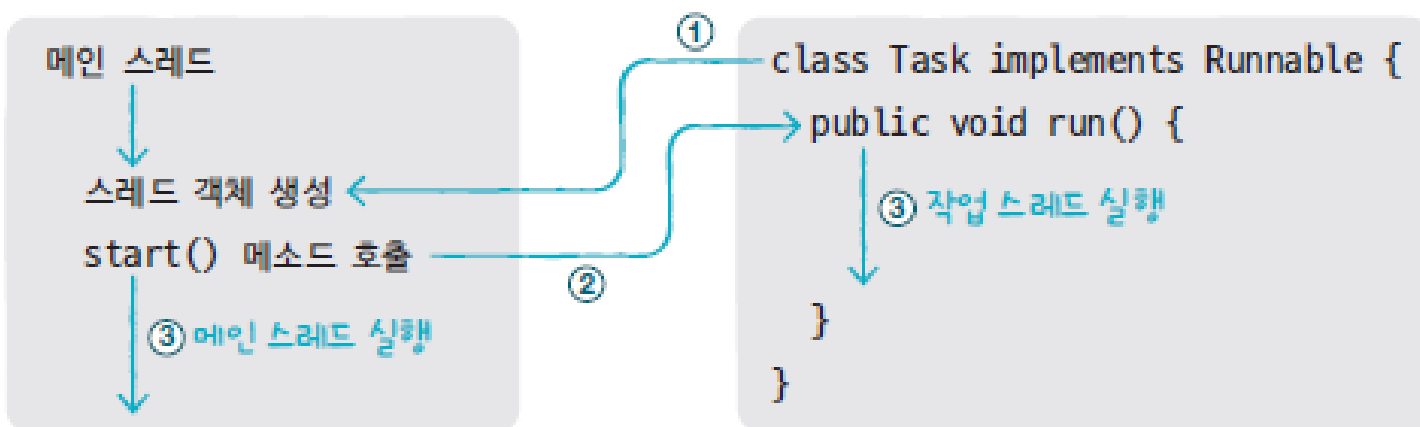
- Runnable 익명 객체를 매개값으로 사용하여 Thread 생성자 호출할 수도 있음

```
Thread thread = new Thread( new Runnable() {  
    public void run() {  
        스레드가 실행할 코드;  
    }  
});
```

← 익명 구현 객체

- start() 메소드 호출하면 작업 스레드 실행

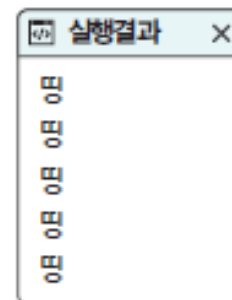
```
thread.start();
```



작업 스레드 생성과 실행

■ 예시 - 메인 스레드만 이용한 경우

```
01 package sec01.exam01;
02
03 import java.awt.Toolkit;
04
05 public class BeepPrintExample1 {
06     public static void main(String[] args) {
07         Toolkit toolkit = Toolkit.getDefaultToolkit(); ← Toolkit 객체 얻기
08         for(int i=0; i<5; i++) {
09             toolkit.beep(); ← 비프음 발생
10             try { Thread.sleep(500); } catch(Exception e) {}
11         } ↑ 0.5초간 일시 정지
12
13         for(int i=0; i<5; i++) {
14             System.out.println("띵");
15             try { Thread.sleep(500); } catch(Exception e) {}
16         } ↑ 0.5초간 일시 정지
17     }
18 }
```



작업 스레드 생성과 실행

- 예시 - 비프음을 들려주는 작업 정의 - Runnable 구현 클래스


```
01 package sec01.exam02;
02
03 import java.awt.Toolkit;
04
05 public class BeepTask implements Runnable {
06     public void run() {
07         Toolkit toolkit = Toolkit.getDefaultToolkit();
08         for(int i=0; i<5; i++) {
09             toolkit.beep();
10             try { Thread.sleep(500); } catch(Exception e) {}
11         }
12     }
13 }
```

← 스레드 실행내용

작업 스레드 생성과 실행

- 예시 - 비프음 - 메인 스레드와 작업 스레드가 동시에 실행

```
01 package sec01.exam02;
02
03 public class BeepPrintExample2 {
04     public static void main(String[] args) {
05         Runnable beepTask = new BeepTask();
06         Thread thread = new Thread(beepTask);
07         thread.start();
08
09         for(int i=0; i<5; i++) {
10             System.out.println("땡");
11             try { Thread.sleep(500); }
12                 catch(Exception e) {}
13         }
14     }
15 }
```



```
public void run() {
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    for(int i=0; i<5; i++) {
        toolkit.beep();
        try { Thread.sleep(500); } catch(Exception e) {}
    }
}
```

작업 스레드 생성과 실행

❖ Thread 하위 클래스로부터 생성

- Thread의 하위 클래스로 작업 스레드를 정의하면서 작업 내용을 포함
- 작업 스레드 클래스 정의하는 법

```
public class WorkerThread extends Thread {  
    @Override  
    public void run() {  
        스레드가 실행할 코드;  
    }  
}  
Thread thread = new WorkerThread();
```

← run() 메소드 재정의

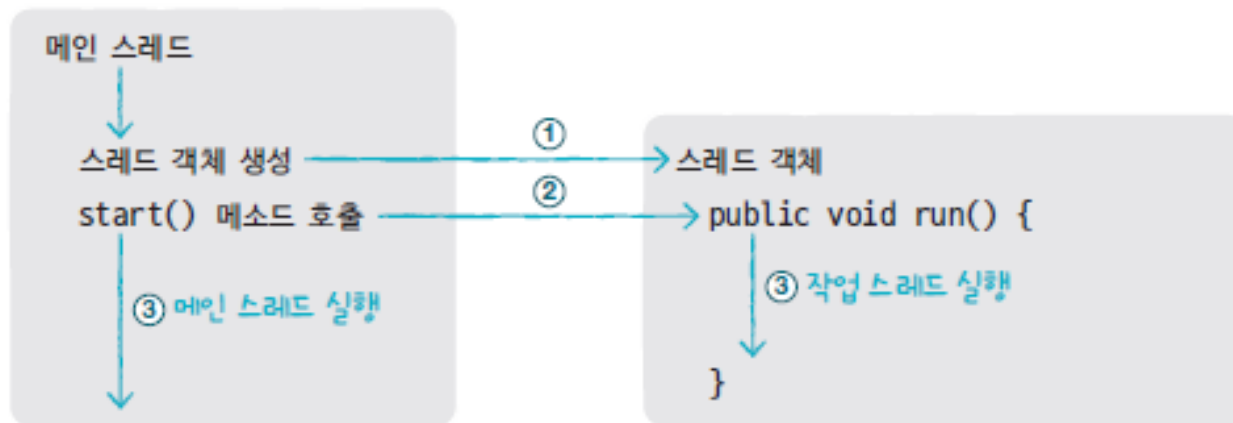
```
Thread thread = new Thread() {  
    public void run() {  
        스레드가 실행할 코드;  
    }  
};
```

← 익명 자식 객체

작업 스레드 생성과 실행

- 작업 스레드 객체 생성 후 start() 메소드 호출하면 run() 메소드가 실행

```
thread.start();
```



작업 스레드 생성과 실행

■ 예시 - 비프음을 들려주는 스레드

```
01 package sec01.exam04;
02
03 import java.awt.Toolkit;
04
05 public class BeepThread extends Thread {
06     @Override
07     public void run() {
08         Toolkit toolkit = Toolkit.getDefaultToolkit();
09         for(int i=0; i<5; i++) {
10             toolkit.beep();
11             try { Thread.sleep(500); } catch(Exception e) {}
12         }
13     }
14 }
```

← 스레드 실행 내용

작업 스레드 생성과 실행

- 예시 – 메인 스레드와 작업 스레드가 동시에 실행

```
01 package sec01.exam04;
02
03 public class BeepPrintExample4 {
04     public static void main(String[] args) {
05         Thread thread = new BeepThread();
06         thread.start();
07
08         for(int i=0; i<5; i++) {
09             System.out.println("띵");
10             try { Thread.sleep(500); }
11                 catch(Exception e) {}
12         }
13     }
14 }
```

메인 스레드

BeepThread

```
public void run() {
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    for(int i=0; i<5; i++) {
        toolkit.beep();
        try { Thread.sleep(500); } catch(Exception e) {}
    }
}
```

작업 스레드 생성과 실행

- 메인 스레드는 'main' 이름 가지며, 우리가 직접 생성한 스레드는 자동적으로 'Thread-n' 이름 설정됨

- setName() 메소드로 이름 변경 가능

```
thread.setName("스레드 이름");
```

- getName() 메소드로 스레드 이름 알 수 있음

```
thread.getName();
```

- currentThread() 메소드로 현재 스레드의 참조 얻을 수 있음

```
Thread thread = Thread.currentThread();
```

작업 스레드 생성과 실행

- 예시 – 메인 스레드 이름 출력 및 UserThread 생성 및 시작

```
01 package sec01.exam06;
02
03 public class ThreadNameExample {
04     public static void main(String[] args) {
05         Thread mainThread = Thread.currentThread(); ← 이 코드를 실행하는 스레드 객체 얻기
06         System.out.println("프로그램 시작 스레드 이름: " + mainThread.getName());
07
08         ThreadA threadA = new ThreadA(); ← ThreadA 생성
09         System.out.println("작업 스레드 이름: " + threadA.getName()); ← 스레드 이름 얻기
10         threadA.start(); ← ThreadA 시작
11
12         ThreadB threadB = new ThreadB(); ← ThreadB 생성
13         System.out.println("작업 스레드 이름: " + threadB.getName()); ← 스레드 이름 얻기
14         threadB.start(); ← ThreadB 시작
15     }
16 }
```

실행결과

```
프로그램 시작 스레드 이름: main
작업 스레드 이름: ThreadA
ThreadA가 출력한 내용
ThreadA가 출력한 내용
작업 스레드 이름: Thread-1
Thread-1가 출력한 내용
Thread-1가 출력한 내용
```

작업 스레드 생성과 실행

■ 예시 – ThreadA 클래스

```
01 package sec01.exam06;
02
03 public class ThreadA extends Thread {
04     public ThreadA() {
05         setName("ThreadA"); ← 스레드 이름 설정
06     }
07
08     public void run() {
09         for(int i=0; i<2; i++) {
10             System.out.println(getName() + "가 출력한 내용"); ← ThreadA 실행 내용
11         }
12     }
13 }
```

↑ 스레드 이름 얻기

작업 스레드 생성과 실행

■ 예시 – ThreadB 클래스

```
01 package sec01.exam06;
02
03 public class ThreadB extends Thread {
04     public void run() {
05         for(int i=0; i<2; i++) {
06             System.out.println(getName() + "가 출력한 내용");
07         }
08     }
09 }
```

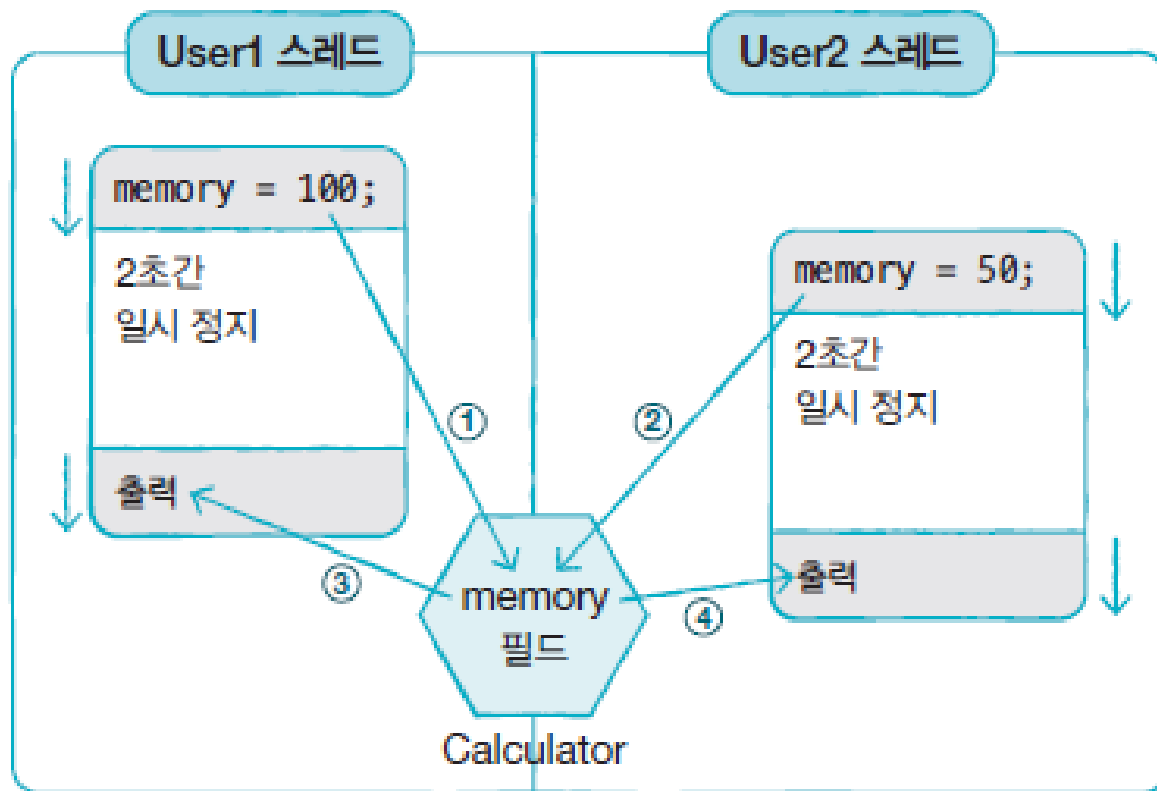
← ThreadB 실행 내용

↑ 스레드 이름 얻기

동기화 메소드

❖ 공유 객체를 사용할 때 주의할 점

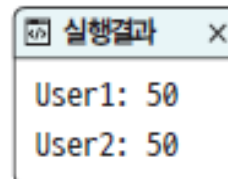
- 멀티 스레드 프로그램에서 스레드들이 객체 공유해서 작업해야 하는 경우 의도했던 것과 다른 결과 나올 수 있음



동기화 메소드

■ 메인 스레드가 실행하는 코드

```
01 package sec01.exam07;  
02  
03 public class MainThreadExample {  
04     public static void main(String[] args) {  
05         Calculator calculator = new Calculator();  
06  
07         User1 user1 = new User1(); ← User1 스레드 생성  
08         user1.setCalculator(calculator); ← 공유 객체 설정  
09         user1.start(); ← User1 스레드 시작  
10  
11         User2 user2 = new User2(); ← User2 스레드 생성  
12         user2.setCalculator(calculator); ← 공유 객체 설정  
13         user2.start(); ← User2 스레드 시작  
14     }  
15 }
```



동기화 메소드

■ 공유 객체

```
01 package sec01.exam07;
02
03 public class Calculator {
04     private int memory;
05
06     public int getMemory() {
07         return memory;
08     }
09
10     public void setMemory(int memory) { ← 계산기 메모리에 값을 저장하는 메소드
11         this.memory = memory; ← 매개값을 memory 필드에 저장
12         try {
13             Thread.sleep(2000); ← 스레드를 2초간 일시 정지시킴
14         } catch (InterruptedException e) {}
15         System.out.println(Thread.currentThread().getName() + ": " + this.memory);
16     }                                     스레드 이름 얻기                     메모리 값
17 }
```


동기화 메소드

■ User1 스레드

```
01 package sec01.exam07;
02
03 public class User1 extends Thread {
04     private Calculator calculator;
05
06     public void setCalculator(Calculator calculator) {
07         this.setName("User1"); ← 스레드 이름을 User1로 설정
08         this.calculator = calculator; ← 공유 객체인 Calculator를 필드에 저장
09     }
10
11     public void run() {
12         calculator.setMemory(100); ← 공유 객체인 Calculator의
13     }                                     메모리에 100을 저장
14 }
```

동기화 메소드

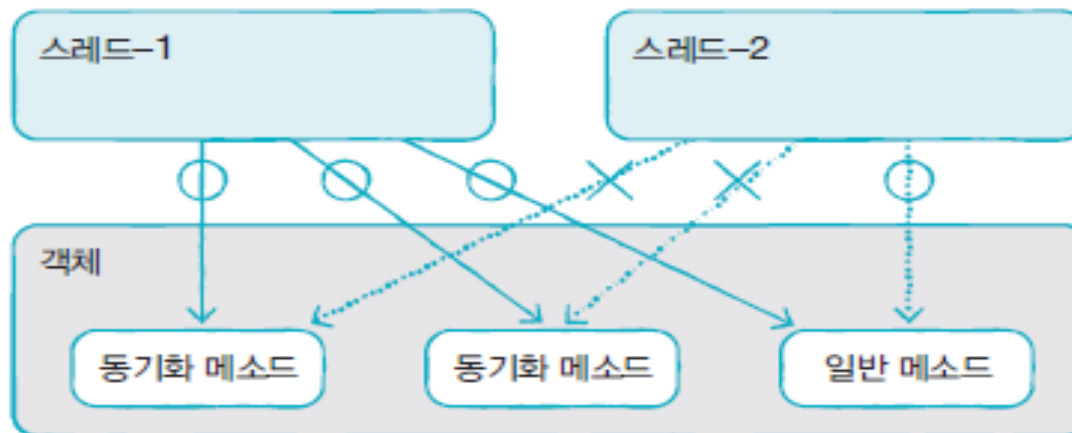
■ User2 쓰레드

```
01 package sec01.exam07;
02
03 public class User2 extends Thread {
04     private Calculator calculator;
05
06     public void setCalculator(Calculator calculator) {
07         this.setName("User2"); ← 스레드 이름을 User2로 설정
08         this.calculator = calculator; ← 공유 객체인 Calculator를 필드에 저장
09     }
10
11     public void run() {
12         calculator.setMemory(50); ← 공유 객체인 Calculator의
13     }                               메모리에 50을 저장
14 }
```

❖ 동기화 메소드

- 스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없게 하려면 스레드 작업 끝날 때까지 객체에 잠금 걸어야 함
- 임계 영역 (critical section) : 단 하나의 스레드만 실행할 수 있는 코드 영역
- 동기화 (synchronized) 메소드 : 스레드가 객체 내부의 동기화 메소드 실행하면 즉시 객체에 잠금 걸림

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



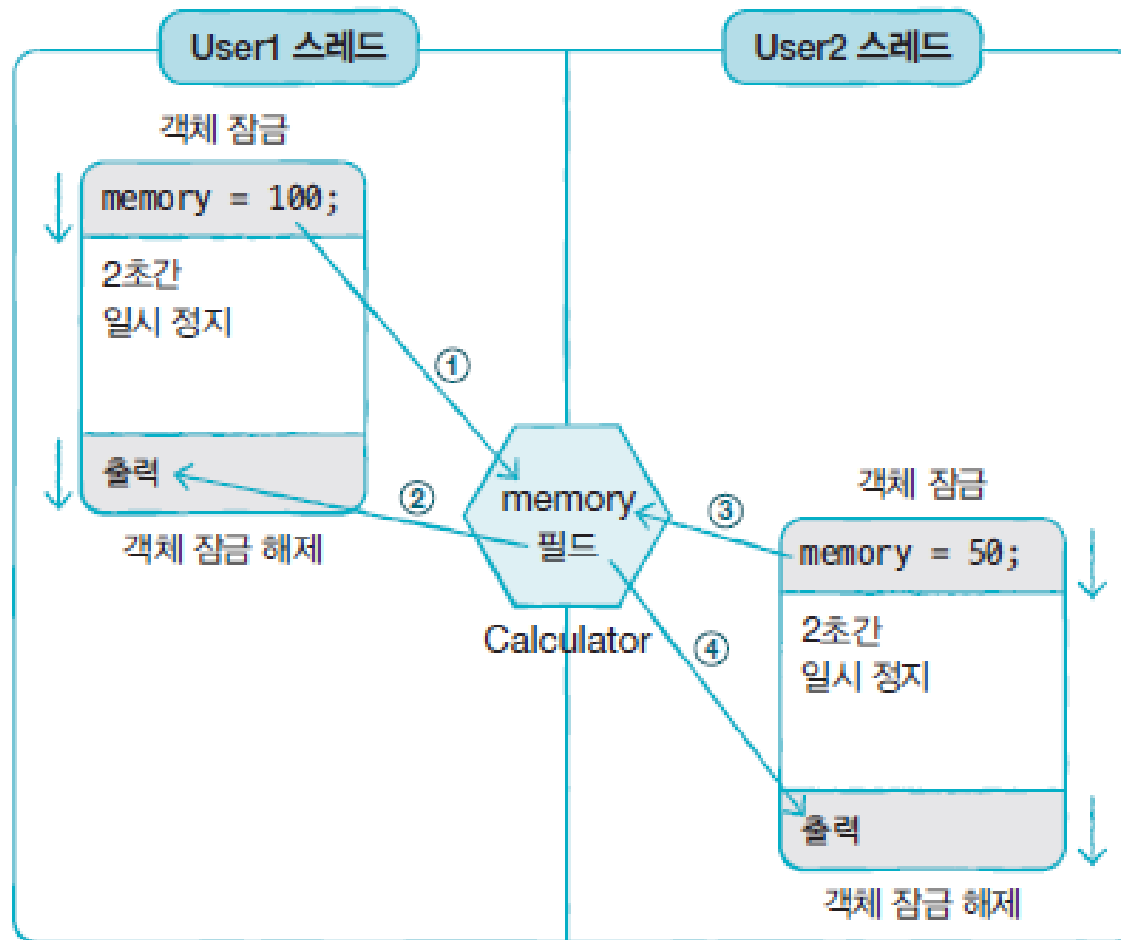
동기화 메소드

- 동기화 메소드로 수정된 공유 객체

```
01 package sec01.exam08;
02
03 public class Calculator {
04     private int memory;
05
06     public int getMemory() {
07         return memory;
08     }
09
10     public synchronized void setMemory(int memory) {
11         this.memory = memory;
12         try {
13             Thread.sleep(2000);
14         } catch (InterruptedException e) {}
15         System.out.println(Thread.currentThread().getName() + ": " + this.memory);
16     }
17 }
```

동기화 메소드

- MainThreadExample.java 실행하면 User1은 100, User2는 50



키워드로 끝내는 핵심 포인트

- **프로세스** : 애플리케이션 실행하면 운영체제로부터 실행에 필요한 메모리 할당 받아 애플리케이션 실행됨.
- **멀티 쓰레드** : 하나의 프로세스 내에 동시 실행하는 쓰레드가 2개 이상인 경우
- **메인 쓰레드** : 자바의 모든 어플리케이션은 메인 쓰레드가 `main()` 메소드 실행하면서 시작. `main()` 메소드의 첫 코드부터 아래로 순차 실행하고, `main()` 메소드의 마지막 코드 실행하거나 `return` 문 만나면 실행이 종료
- **작업 쓰레드** : 메인 작업 이외에 병렬 작업의 수만큼 생성하는 쓰레드. 객체로서 생성되기 때문에 클래스 필요. `Thread` 클래스를 직접 객체화해서 생성할 수도 있고, `Thread` 클래스를 상속해서 하위 클래스 만들어 생성할 수도 있음
- **동기화 메소드** : 멀티 쓰레드 프로그램에서 단 하나의 쓰레드만 실행할 수 있는 코드 영역을 임계 영역이라 함. 이를 지정하기 위해 동기화 메소드가 제공됨. 쓰레드가 객체 내부의 동기화 메소드 실행하면 즉시 객체에 잠금 걸어 다른 쓰레드가 동기화 메소드 실행하지 못하게 함

❖ 쓰레드에 대한 설명 중 틀린 것은 무엇입니까?

- 자바 애플리케이션은 메인 쓰레드가 `main()` 메소드를 실행한다
- 작업 쓰레드 클래스는 `Thread` 클래스를 상속해서 만들 수 있다
- `Runnable` 객체는 쓰레드가 실행해야 할 코드를 가지고 있는 객체라 볼 수 있다
- 쓰레드 실행을 시작하려면 `run()` 메소드를 호출해야 한다

❖ 동기화 메소드에 대한 설명 중 틀린 것은 무엇입니까?

- 동기화 메소드는 싱글 쓰레드 환경에서는 필요하지 않다
- 쓰레드가 동기화 메소드를 실행할 때 다른 쓰레드는 일반 메소드를 호출할 수 없다
- 쓰레드가 동기화 메소드를 실행할 때 다른 쓰레드는 다른 동기화 메소드를 호출할 수 없다
- 동기화 메소드 선언부에는 `synchronized` 키워드가 필요하다

Thank You !

혼자 공부하는 자바 (신용권 저)