16장. 스레드

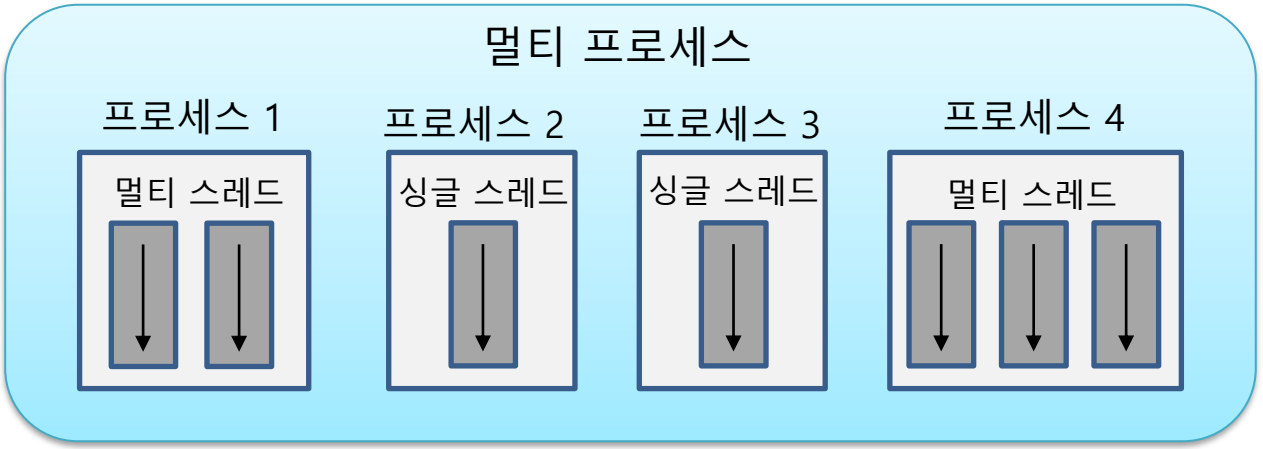
Thread

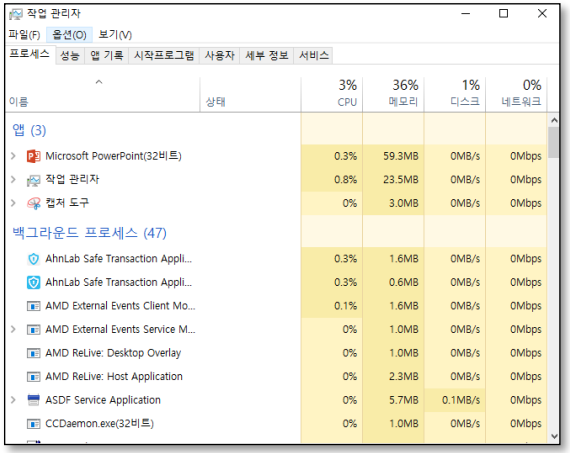
멀티 쓰레드

- **프로세스와 쓰레드**

프로세스(Process)
- 실행 중인 하나의 프로그램을 말한다.(하드디스크 -> 주기억장치)

멀티 태스킹(multi tasking)
- 두가지 이상의 작업을 동시에 처리하는 것.
- 멀티 프로세스 : 독립적으로 프로그램들을 실행하고 여러가지 작업처리.
- 멀티 스레드 : 한 개를 프로그램을 실행하고 내부적으로 여러 가지 작업 처리



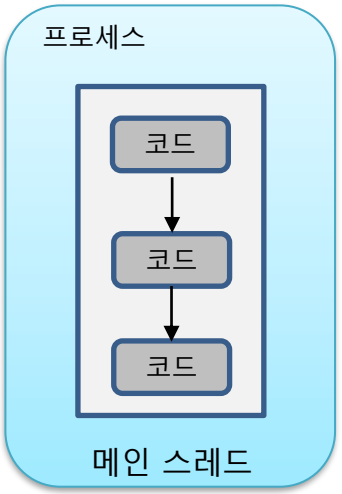


멀티 쓰레드

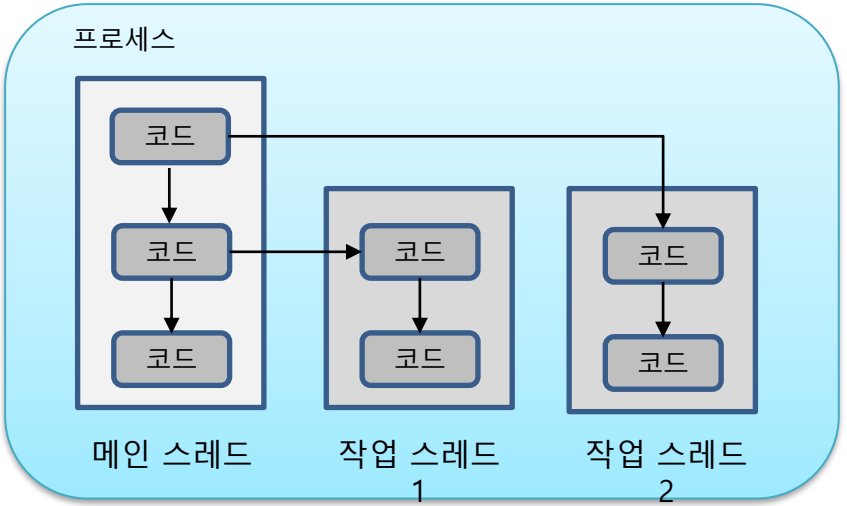
메인(main) 스레드

- 모든 자바 프로그램은 메인 스레드가 main() 메소드를 실행하면서 시작된다.
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행한다.
- main()메소드의 마지막 코드를 실행하거나, return문을 만나면 실행이 종료된다.

싱글 스레드 애플리케이션



멀티 스레드 애플리케이션



프로세스의 종료

- 싱글 스레드 : 메인 스레드가 종료되면 프로세스도 종료된다.
- 멀티 스레드 : 실행 중인 스레드가 하나라도 있다면, 프로세스는 종료되지 않는다.
  (메인 스레드가 작업스레드보다 먼저 종료되는 경우도 있다.)

Thread 클래스

작업 스레드 생성과 실행

- 자바에서는 작업 스레드도 객체로 생성되기 때문에 클래스가 필요하다.
- Java.lang.Thread 클래스를 직접 객체화하거나, Thread를 상속해서 하위 클래스를 만들어 생성

Class Thread

java.lang.Object
    java.lang.Thread

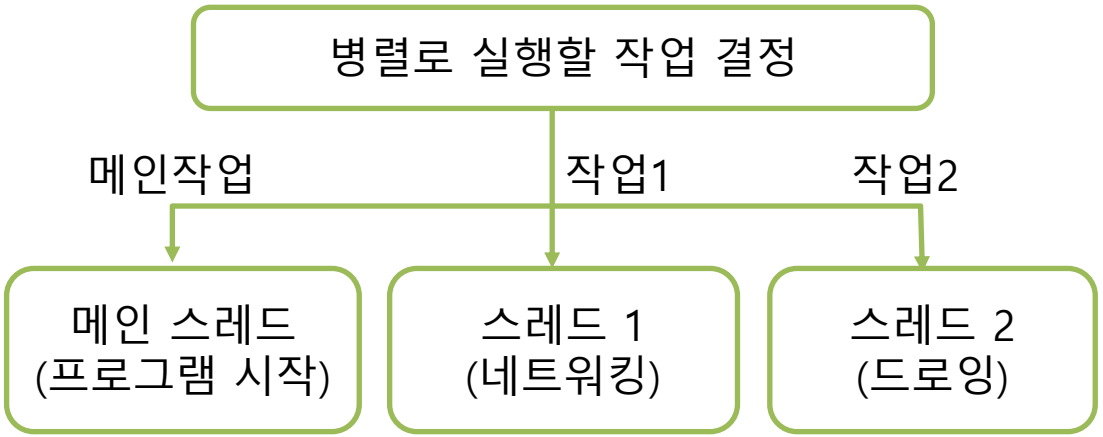
All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread

```
public class Thread
extends Object
implements Runnable
```



Thread 클래스

방법 1-1) Thread 클래스로 부터 직접 생성

Runnable은 스레드가 작업흘 실행할 때 사용하는 인터페이스이다.

Run() 메서드를 재정의해서 스레드가 실행할 코드를 가지고 있어야 함

```
class Task implements Runnable{

        @Override
        public void run(){
              //스레드가 실행할 코드;
        }
}
```

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

→ **thread.start()** 쓰레드 시작(실행)

Thread 클래스

방법 1-2) Runnable 익명 구현 객체를 매개값으로 사용

```
    Thread thread = new Thread(new Runnable(){

    @Override
    public void run() {
        //스레드가 실행할 코드
    }
};
```

Thread 클래스

방법 2) Thread 자식 클래스로 생성 (상속)

```
public class 스레드 클래스 extends Thread{
    …
}

Thread thread = new 스레드클래스();
```

→ **thread.start()**

쓰레드 시작(실행)

Thread 이름

Thread 이름

스레드는 자신의 이름을 가지고 있다. 메인 스레드는 'main'이라는 이름을 가지고 있고, 직접 생성한 스레드는 자동으로 Thread-n 이라는 이름으로 설정된다.

다른 이름으로 설정하고 싶다면 Thread 클래스의 setName() 메소드로 변경한다.

```
main 실행
Thread-0 실행
Thread-1 실행
chat-thread 실행
chat-thread 실행
```

Thread 이름

Thread 이름

```
public class ThreadNameTest {

    public static void main(String[] args) {
        //현재 이 코드를 실행하는 쓰레드의 정보 얻기
        Thread mainThread = Thread.currentThread();
        System.out.println(mainThread.getName() + " 실행");

        for(int i=0; i<2; i++) {
            //Thread(작업 스레드) 객체 생성
            Thread threadA = new Thread() {
                //run() 메서드 재정의 함
                @Override
                public void run() {
                    //getName() - threaA의 이름을 리턴
                    System.out.println(getName() + " 실행");
                }
            };
            threadA.start(); //실행 대기 상태임
        }
```

Thread 이름

Thread 이름

```
        for(int i=0; i<2; i++) {
            Thread chatThread = new Thread() {

                @Override
                public void run() {
                    System.out.println(getName() + " 실행");
                }

            };
            chatThread.setName("chat-thread");  //쓰레드 이름 변경
            chatThread.start();
        }
}
```

Thread 이름

1. Thread 이름을 설정한 경우

```
public class ThreadA extends Thread{

    public ThreadA() {
        setName("ThreadA");
    }

    @Override
    public void run() {
        for(int i=0; i<2; i++) {
            System.out.println(getName() + "가 출력한 내용");
        }
    }
}
```

Thread 이름

2. Thread 이름을 설정하지 않은 경우

```
public class ThreadB extends Thread{

    @Override
    public void run() {
        for(int i=0; i<2; i++) {
            System.out.println(getName() + "가 출력한 내용");
        }
    }
}
```

Thread 이름

```
public class ThreadTest {

    public static void main(String[] args) {
        Thread mainThread = Thread.currentThread();
        System.out.println("프로그램 시작 스레드 이름: " + mainThread.getName());
        System.out.println("=====================================");

        ThreadA threadA = new ThreadA();
        System.out.println("작업 스레드 이름: " + threadA.getName());

        threadA.run();
        System.out.println("=====================================");

        ThreadB threadB = new ThreadB();
        System.out.println("작업 스레드 이름: " + threadB.getName());

        threadB.run();
    }
}
```

멀티 쓰레드

메인 스레드만 이용한 경우

```
public class BeepPrintTest {
    public static void main(String[] args) {
        //메인 스레드만 실행
        //"띵" 문자를 5번 출력하기 -> 1초 대기 간격
        for(int i=0; i<5; i++) {
            System.out.println("띵");
            try {
                Thread.sleep(1000); //1000ms -> 1s(1초)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        //"띵" 소리를 5번 재생하기
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        for(int i=0; i<5; i++) {
            toolkit.beep();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

Module java.desktop
Package java.awt

Class Toolkit

java.lang.Object
    java.awt.Toolkit

public abstract class **Toolkit**
extends Object

멀티 쓰레드

비프음을 들려주는 작업스레드 정의

비프음을 발생시키면서 동시에 프린팅을 하고 싶다면 두 작업중 하나를 작업스레드에서 처리하도록 해야한다.

```
public class BeepTask implements Runnable{
    //Runnable 인터페이스를 구현한 BeepTask 클래스 생성
    //비프음을 재생하는 작업 정의
    @Override
    public void run() {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        for(int i=0; i<5; i++) {
            toolkit.beep();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

멀티 쓰레드

메인 스레드와 작업 스레드가 동시에 실행

```
public class BeepPrintTest2 {
    public static void main(String[] args) {
        //메인 스레드와 작업 스레드가 동시에 실행
        Runnable beepTask = new BeepTask();
        Thread thread = new Thread(beepTask);
        thread.start();  //쓰레드 시작(실행)

        for(int i=0; i<5; i++) {
            System.out.println("띵");
            try {
                Thread.sleep(1000); //1000ms -> 1s(1초)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
띵
띵
띵
띵
띵
```

멀티 쓰레드

익명 객체로 구현

```
public class BeepPrintTest3 {
    public static void main(String[] args) {
        //메인 스레드와 작업 스레드가 동시에 실행
        //익명 객체로 구현
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                Toolkit toolkit = Toolkit.getDefaultToolkit();
                for(int i=0; i<5; i++) {
                    toolkit.beep();
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        thread.start();  //스레드 시작

        for(int i=0; i<5; i++) {
            System.out.println("띵");
            try {
                Thread.sleep(1000); //1000ms -> 1s(1초)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

멀티 쓰레드

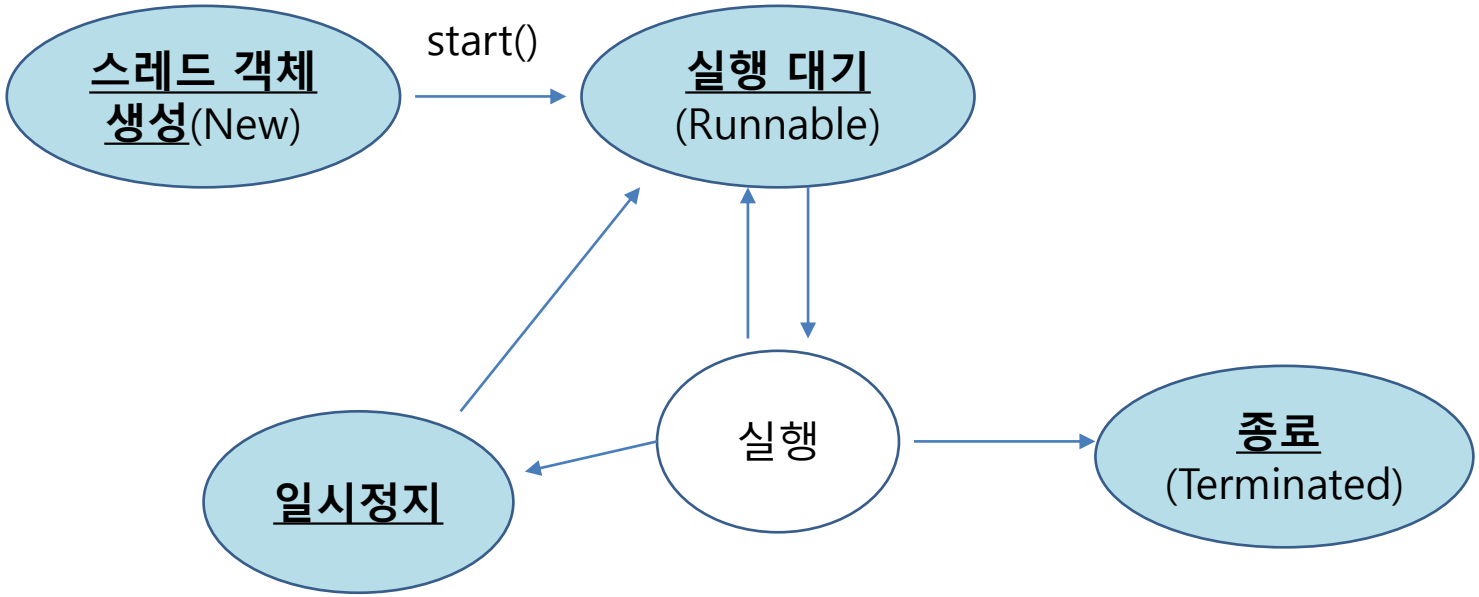
람다식으로 구현

```
public class BeepPrintTest4 {
    public static void main(String[] args) {
        //메인 스레드와 작업 스레드가 동시에 실행
        Thread thread = new Thread(()->{
            Toolkit toolkit = Toolkit.getDefaultToolkit();
            for(int i=0; i<5; i++) {
                toolkit.beep();
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        thread.start();

        for(int i=0; i<5; i++) {
            System.out.println("띵");
            try {
                Thread.sleep(1000); //1000ms -> 1s(1초)
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

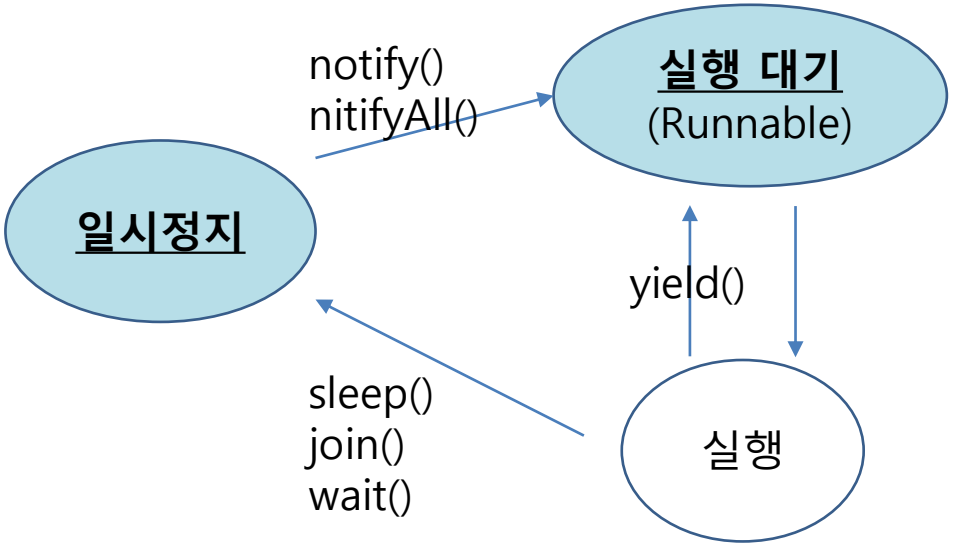
스레드 상태

스레드 객체를 생성하고 start() 메서드를 호출하면 곧바로 스레드가 실행되는 것이 아니라 실행 대기 상태(Runnable)가 된다. 실행 대기하는 스레드는 CPU 스케줄링에 따라 CPU를 점유하고 run() 메서드를 실행한다. 이때를 실행(Running) 상태라 한다.

이렇게 스레드는 실행 대기 상태와 실행 상태를 번갈아 가면서 자신의 run() 메서드를 조금씩 실행한다. 실행 상태에서 run() 메서드가 종료되면 더 이상 실행할 코드가 없기 때문에 스레드의 실행은 멈추게 된다. 이 상태를 종료 상태(Terminated)라고 함



스레드 상태



구분	메서드	설명
일시정지로 보냄	sleep(long millis)	주어진 시간 동안 스레드를 일시정지상태로 만듬
	join()	join() 메서드를 호출한 스레드는 일시 정지 상태가 됨 실행대기 상태가 되려면 join() 메서들를 가진 스레드가 종료되어야 함
	wait()	동기화 블록 내에서 스레드를 일시 정지 상태로 만듬

스레드 상태

구분	메서드	설명
일시정지로 보냄	sleep(long millis)	주어진 시간 동안 스레드를 일시정지상태로 만듬
	join()	join() 메서드를 호출한 스레드는 일시 정지 상태가 됨 실행대기 상태가 되려면 join() 메서들를 가진 스레드가 종료되어야 함
	wait()	동기화 블록 내에서 스레드를 일시 정지 상태로 만듬
일시 정지에서 벗어남	interrupt()	일시 정지 상태일 경우, InterruptedException을 발생시켜 실행 대기 상태 또는 종료 상태로 만듬
	notify() notifyAll()	wait() 메서드로 인해 일시 정지 상태인 스레드를 실행 대기 상태로 만듬
실행 대기로 보냄	yield()	실행 상태에서 다른 스레드에게 실행을 양보하고 실행 대기 상태가 됨

스레드 상태

메인 스레드는 sumThread를 호출하고 일시 정지 상태에 들어가고, sumThread가 최종 계산된 결과값을 산출하고 종료하면 다시 메인스레드는 결과값을 받아 출력함

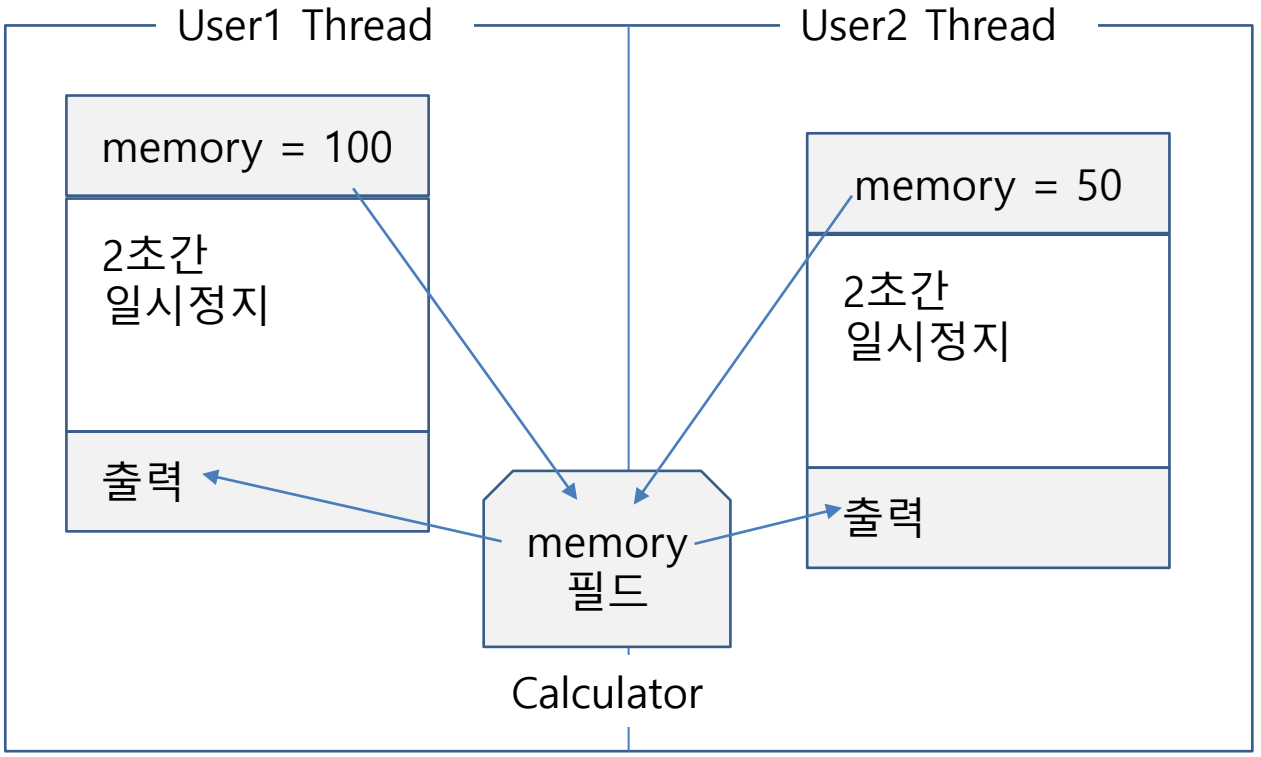
```
public class JoinExample {

    public static void main(String[] args) {

        SumThread sumThread = new SumThread();
        sumThread.start(); //메인스레드가 sumThread를 호출하고 일시 정지 상태로 들어감

        //생략하면 0이 나옴
        try { //sumThread가 계산을 수행함
            sumThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //메인스레드가 결과값을 출력하도록 호출
        System.out.println(sumThread.getSum());
    }
}
```

스레드 상태

```java
//Thread를 상속받은 클래스
public class SumThread extends Thread{
    private long sum;

    public long getSum() {
        return sum;
    }

    public void setSum(long sum) {
        this.sum = sum;
    }

    @Override
    public void run() { //1~100까지 더하기
        for(int i=1; i<=100; i++) {
            sum += i;
        }
    }
}
```

스레드 동기화

멀티 스레드는 하나의 객체를 공유해서 작업할 수도 있다. 이 경우, 다른 스레드에 의해 객체 내부 데이터가 쉽게 변경될 수 있기 때문에 의도했던 것과 다른 결과가 나올 수 있음



스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없도록 하려면 스레드 작업이 끝날때까지 객체에 잠금을 걸면 된다. 이를 위해 자바는 동기화(synchronized) 메서드를 제공함

스레드 동기화

```
public class Calculator {

    private int memory;

    public int getMemory() {
        return memory;
    }

    //공유하면 다른 쓰레드가 사용할 수 있음
    /*public void setMemory(int memory) {
        this.memory = memory;

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
        }
        System.out.println(Thread.currentThread().getName()
                + ":" + this.memory);
    }*/
```

스레드 동기화

```
//synchronized를 붙이면 동기화 메서드가 되면서 lock을 건다.
public synchronized void setMemory(int memory) {
    this.memory = memory;

    try {
        Thread.sleep(2000);  //2초간 일시 정지
    } catch (InterruptedException e) {
    }
    System.out.println(Thread.currentThread().getName()
            + ":" + this.memory);
}
```

동기화 하지 않은 경우

```
User1Thread:50
User2Thread:50
```

동기화 한 경우

```
User1Thread:100
User2Thread:50
```

스레드 동기화

```
public class SyncronizedTest {

    public static void main(String[] args) {
        //공유 객체 생성
        Calculator calculator = new Calculator();

        //User1Thread 객체 생성
        User1Thread user1Thread = new User1Thread();
        user1Thread.setCalculator(calculator);
        user1Thread.start();

        //User2Thread 객체 생성
        User2Thread user2Thread = new User2Thread();
        user2Thread.setCalculator(calculator);
        user2Thread.start();
    }
}
```

스레드 동기화

```
public class User1Thread extends Thread{

    private Calculator calculator;

    public User1Thread() {
        setName("User1Thread");
    }

    public void setCalculator(Calculator calculator) {
        this.calculator = calculator;
    }

    @Override
    public void run() {
        calculator.setMemory(100);
    }
}
```

스레드 동기화

```
public class User2Thread extends Thread{

    private Calculator calculator;

    public User2Thread() {
        setName("User2Thread");
    }

    public void setCalculator(Calculator calculator) {
        this.calculator = calculator;
    }

    @Override
    public void run() {
        calculator.setMemory(50);
    }
}
```