18장. 멀티스레드 프로그래밍

학습 목표

- 멀티스레드 프로그램이란?
- 멀티스레드 프로그램의 작성 방법
- 스레드간의 커뮤니케이션 방법
- 스레드의 상태를 알아내는 방법

01. 멀티스레드 프로그램이란?

스레드란?

- 스레드(thread) : 프로그램의 실행 흐름

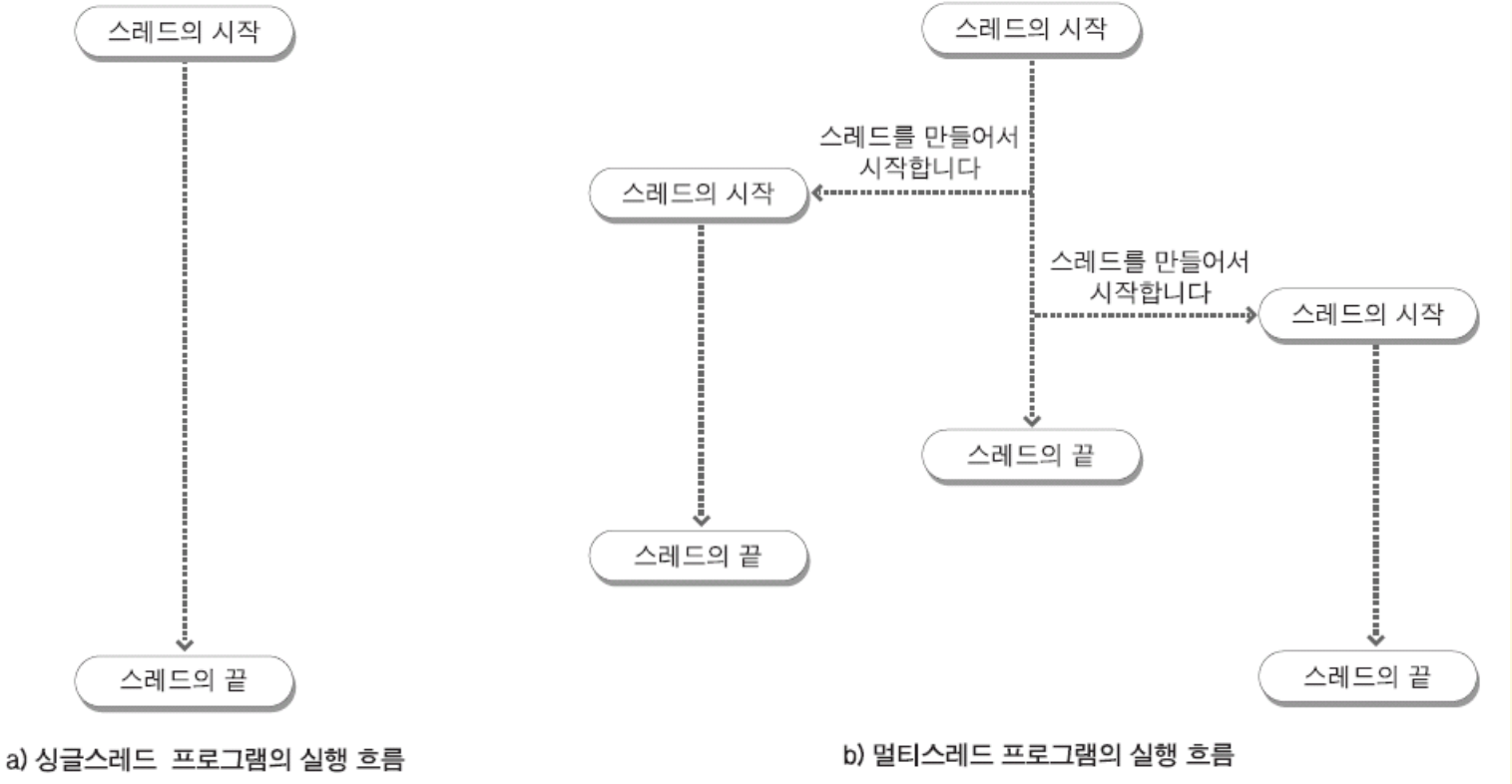
```
class Total {
    public static void main(String args[]) {
        int total = 0;
        for (int cnt = 0; cnt < 3; cnt++)
            total += cnt;
        System.out.println(total);
    }
}
```

- 싱글 스레드(single thread program) : 스레드가 하나뿐인 프로그램
- 멀티스레드 프로그램(multithread program) : 스레드가 둘 이상인 프로그램

01. 멀티스레드 프로그램이란?

싱글 스레드/멀티스레드 프로그램

- 작동 방식의 차이



a) 싱글스레드 프로그램의 실행 흐름

b) 멀티스레드 프로그램의 실행 흐름

02. 멀티스레드 프로그램의 작성 방법

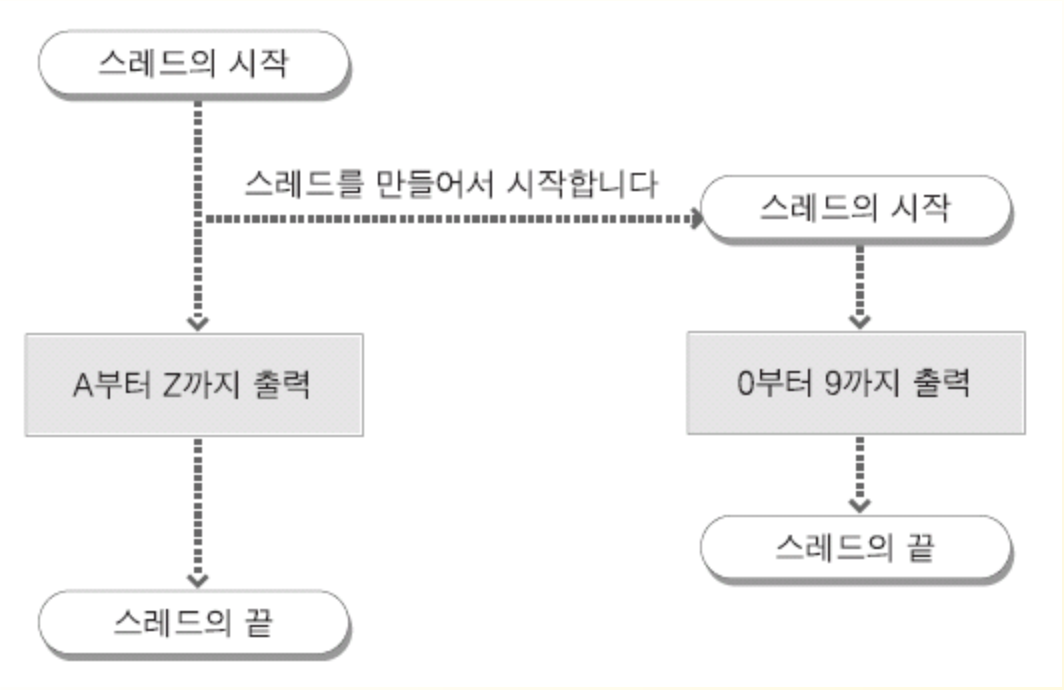
멀티클래스 프로그램의 작성 방법

- 다음 두 가지임
  - java.lang.Thread 클래스를 이용하는 방법
  - java.lang.Runnable 인터페이스를 이용하는 방법

02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- 지금부터 작성할 예제의 스레드 구성



02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

[예제 18-1] 알파벳과 숫자를 동시에 출력하는 멀티스레드 프로그램 (미완성)

main 메소드를 포함하는 클래스

```
class MultithreadExample1 {
    public static void main(String args[]) {


        for (char ch = 'A'; ch <= 'Z'; ch++)
            System.out.print(ch);
    }
}
```

스레드를 만들어서 시작하는 부분

숫자를 출력하는 스레드 클래스

02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- 스레드로 실행할 클래스의 선언 방법

  1) java.lang.Thread 클래스를 상속받는 클래스를 선언합니다.

```
class DigitThread extends Thread {
    ...
}
```

java.lang.Thread의 서브클래스로 선언

* java.lang.Thread 클래스와 서브클래스들을 **스레드 클래스(thread class)**라고 부름

02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- 스레드로 실행할 클래스의 선언 방법

  2) run 메소드 안에 스레드가 해야할 일을 명령문으로 써넣습니다.

```
class DigitThread extends Thread {
    public void run() {
        for (int cnt = 0; cnt < 10; cnt++)
            System.out.print(cnt);
    }
}
```

쓰레드가 해야할 일을
run 메소드 안에 씁니다

02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- 스레드를 만들어서 시작하는 방법

  1) 스레드 클래스의 객체를 생성합니다.

가능

```
Thread thread = new DigitThread();
```

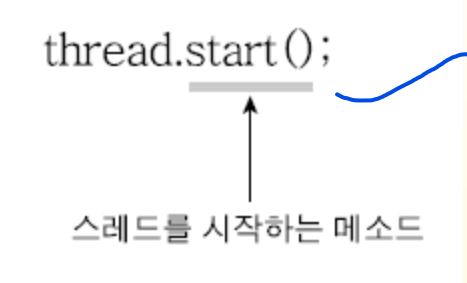
스레드를 생성하는 식

02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- 스레드를 만들어서 시작하는 방법

  2) 스레드 객체에 대해 start 메소드를 호출합니다.



02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

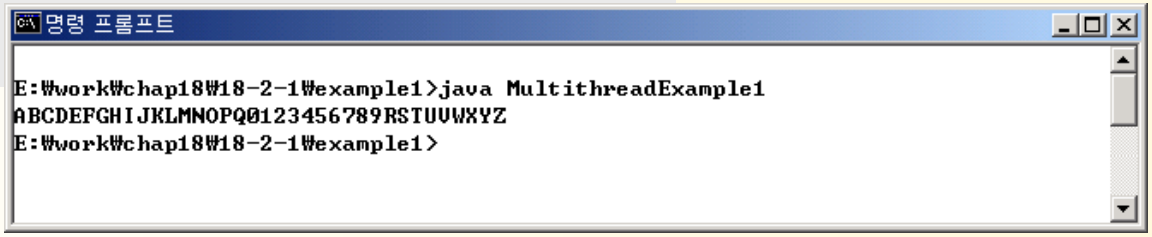
[예제 18-2] 알파벳과 숫자를 동시에 출력하는 멀티스레드 프로그램 (1)

main 메소드를 포함하는 클래스

```
1  class MultithreadExample1 {
2      public static void main(String args[]) {
3          Thread thread = new DigitThread();     // 스레드를 생성
4          thread.start();                        // 스레드를 시작
5          for (char ch = 'A'; ch <= 'Z'; ch++)
6              System.out.print(ch);
7      }
8  }
```

숫자를 출력하는 스레드 클래스

```
1  class DigitThread extends Thread {
2      public void run() {
3          for (int cnt = 0; cnt < 10; cnt++)
4              System.out.print(cnt);
5      }
6  }
```



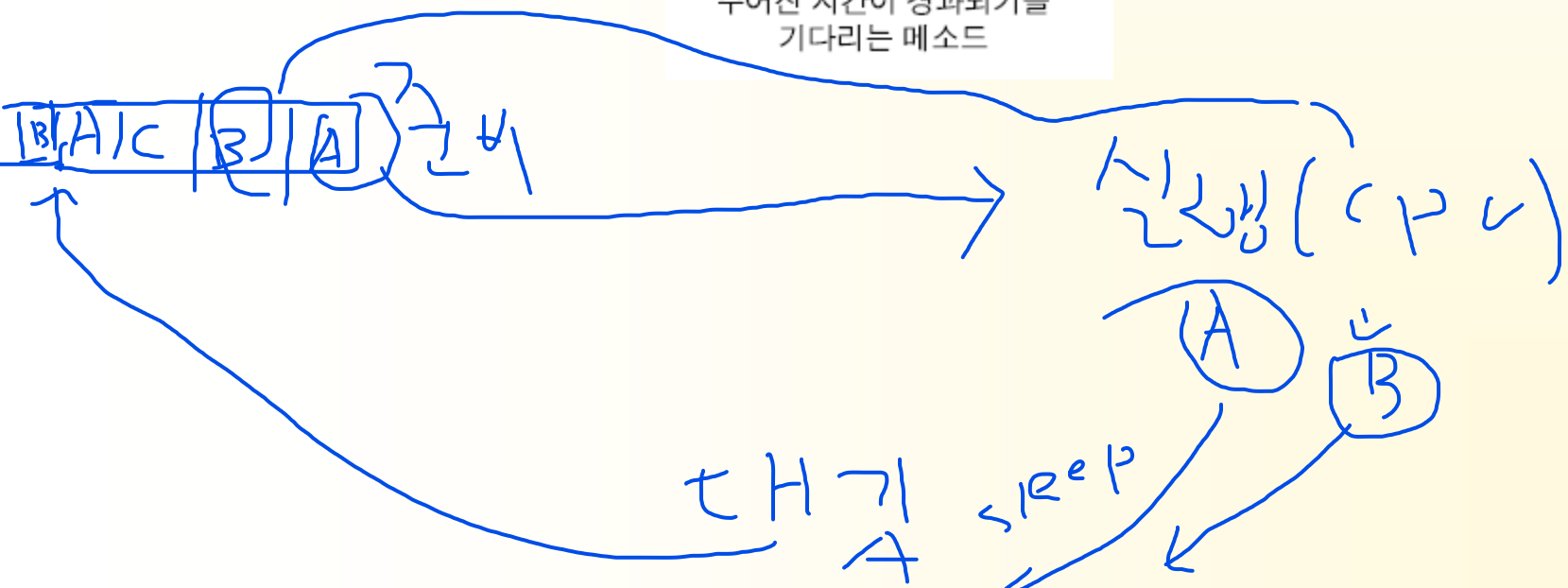
02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- java.lang.Thread 클래스의 sleep 메소드
  - 일정 시간이 경과되기를 기다리는 메소드

```
Thread.sleep(1000);
```

주어진 시간이 경과되기를 기다리는 메소드



02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

- java.lang.Thread 클래스의 sleep 메소드
  - InterruptedException의 처리 방법

반드시

```
try {
    Thread.sleep(1000);
}
catch (InterruptedException e) {
    System.out.println(e.getMessage());
}
```

sleep 메소드가 발생하는 InterruptedException을 처리합니다

02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

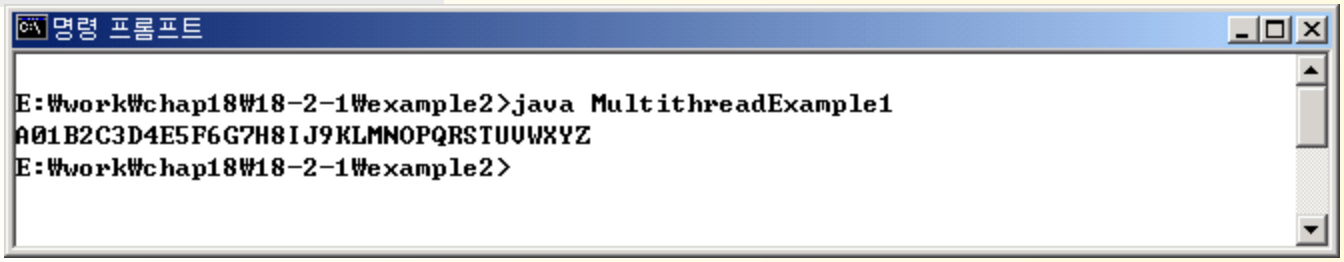
[예제 18-3] 알파벳과 숫자를 동시에 출력하는 멀티스레드 프로그램 (2)

main 메소드를 포함하는 클래스

```
1  class MultithreadExample1 {
2      public static void main(String args[]) {
3          Thread thread = new DigitThread();
4          thread.start();
5          for (char ch = 'A'; ch <= 'Z'; ch++) {
6              System.out.print(ch);
7              try {
8                  Thread.sleep(1000);
9              } catch (InterruptedException e) {
10                 System.out.println(e.getMessage());
11             }
12         }
13     }
14 }
```

숫자를 출력하는 스레드 클래스

```
1  class DigitThread extends Thread {
2      public void run() {
3          for (int cnt = 0; cnt < 10; cnt++) {
4              System.out.print(cnt);
5              try {
6                  Thread.sleep(1000);
7              } catch (InterruptedException e) {
8                  System.out.println(e.getMessage());
9              }
10         }
11     }
12 }
```



02. 멀티스레드 프로그램의 작성 방법

Thread 클래스를 이용한 멀티스레드 프로그램

[예제 18-4] 네 개의 스레드로 실행되는 멀티스레드 프로그램

main 메소드를 포함하는 클래스

```
1   class MultithreadExample2 {
2       public static void main(String args[]) {
3           Thread thread1 = new DigitThread();
4           Thread thread2 = new DigitThread();
5           Thread thread3 = new AlphabetThread();
6           thread1.start();
7           thread2.start();
8           thread3.start();
9       }
10  }
```

3개의 스레드를 생성해서 시작합니다

숫자를 출력하는 스레드 클래스

```
1   class DigitThread extends Thread {
2       public void run() {
3           for (int cnt = 0; cnt < 10; cnt++) {
4               System.out.print(cnt);
5               try {
6                   Thread.sleep(1000);
7               } catch (InterruptedException e) {
8                   System.out.println(e.getMessage());
9               }
10          }
11      }
12  }
```

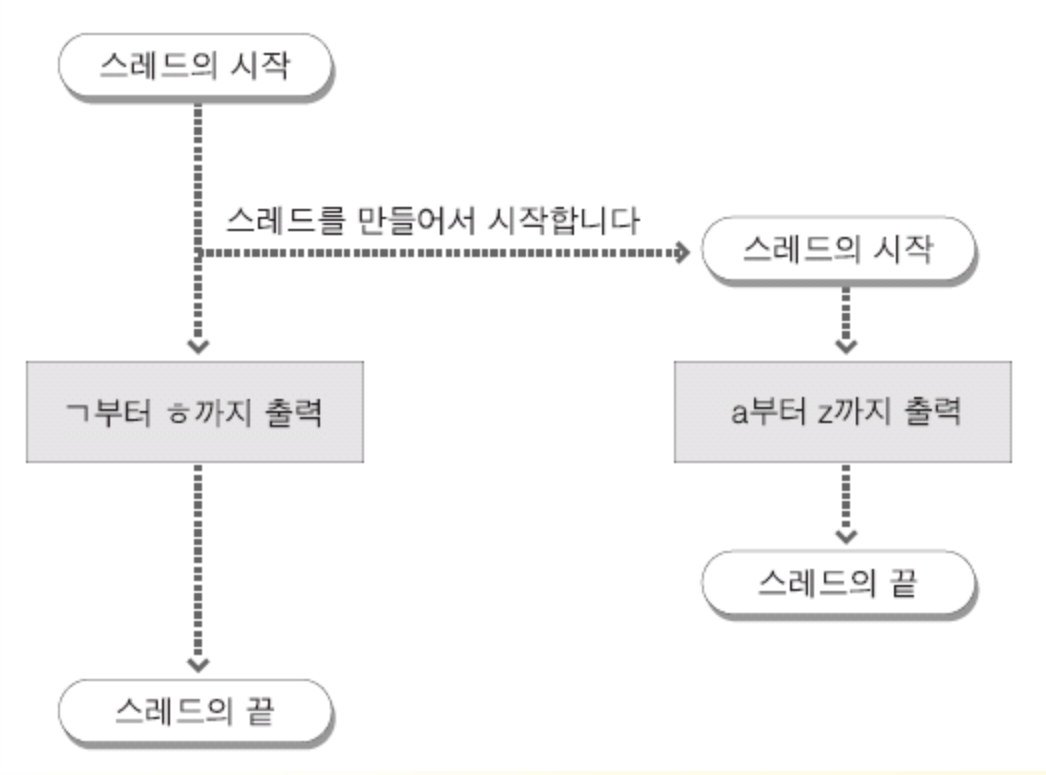
알파벳을 출력하는 스레드 클래스

```
1   class AlphabetThread extends Thread {
2       public void run() {
3           for (char ch = 'A'; ch <= 'Z'; ch++) {
4               System.out.print(ch);
5               try {
6                   Thread.sleep(500);
7               } catch (InterruptedException e) {
8                   System.out.println(e.getMessage());
9               }
10          }
11      }
12  }
```

02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 지금부터 작성할 예제의 스레드 구성



02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

[예제 18-5] 한글과 영문을 동시에 출력하는 멀티스레드 프로그램 (미완성)

main 메소드를 포함하는 클래스

```
1  class MultithreadExample3 {
2      public static void main(String args[]) {
3          ┌──────────────────────────────────────────┐
4          └──────────────────────────────────────────┘
5          char arr[] = { 'ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ',
                          'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ' };
6          for (char ch : arr)
7              System.out.print(ch);
8      }
9  }
```

스레드를 만들어서 시작하는 부분

영문 소문자를 출력하는 스레드 클래스

02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 스레드로 실행할 클래스의 선언 방법

  1) java.lang.Runnable 인터페이스를 구현하는 클래스를 선언합니다.

```
class SmallLetters implements Runnable {
    ...
}
```

java.lang.Runnable 인터페이스를 구현하는 클래스로 선언

02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 스레드로 실행할 클래스의 선언 방법

  2) run 메소드 안에 스레드가 해야할 일을 명령문으로 써넣습니다.

```
class SmallLetters implements Runnable {
    public void run() {
        for (char ch = 'a'; ch <= 'z'; ch++)
            System.out.print(ch);
    }
}
```

스레드가 해야할 일을 run 메서드 안에 씁니다

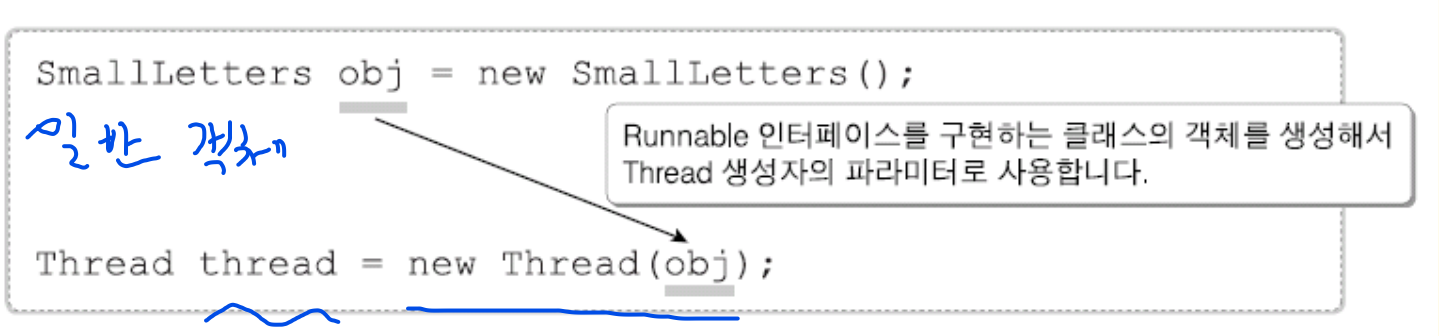
주의 : 이런 클래스는 스레드 클래스가 아님

02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 스레드를 만들어서 시작하는 방법

  1) 다음과 같은 방법으로 Thread 객체를 생성합니다.

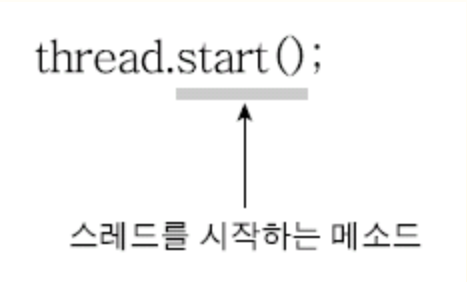


02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 스레드를 만들어서 시작하는 방법

  2) Thread 객체에 대해 start 메소드를 호출합니다.



02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 이용한 멀티스레드 프로그램

[예제 18-6] 한글과 영문을 동시에 출력하는 멀티스레드 프로그램 (완성)

main 메소드를 포함하는 클래스

```
1  class MultithreadExample3 {
2      public static void main(String args[]) {
3          Thread thread = new Thread(new SmallLetters());  // 스레드를 생성
4          thread.start();                                  // 스레드를 시작
5          char arr[] = { 'ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ',
                          'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ' };
6          for (char ch : arr)
7              System.out.print(ch);
8      }
9  }
```

영문 소문자를 출력하는 스레드 클래스

```
1  class SmallLetters implements Runnable {
2      public void run() {
3          for (char ch = 'a'; ch <= 'z'; ch++)
4              System.out.print(ch);
5      }
6  }
```

02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 사용해야만 하는 경우

[예제 18-7] 특정 패키지에 속하는 Numbers 클래스

```
1    package kr.co.bsw;  --------- 이 클래스는 bsw.co.kr이라는 도메인을 가진 회사에 속합니다
2    public class Numbers {
3        protected void list(int start, int end) {
4            for (int cnt = start; cnt <= end; cnt++) {
5                System.out.printf("(%d)", cnt);
6            }
7        }
8    }
```

다른 도메인을 가진 회사에서
이 클래스를 스레드로 활용하려면?

02. 멀티스레드 프로그램의 작성 방법

Runnable 인터페이스를 사용해야만 하는 경우

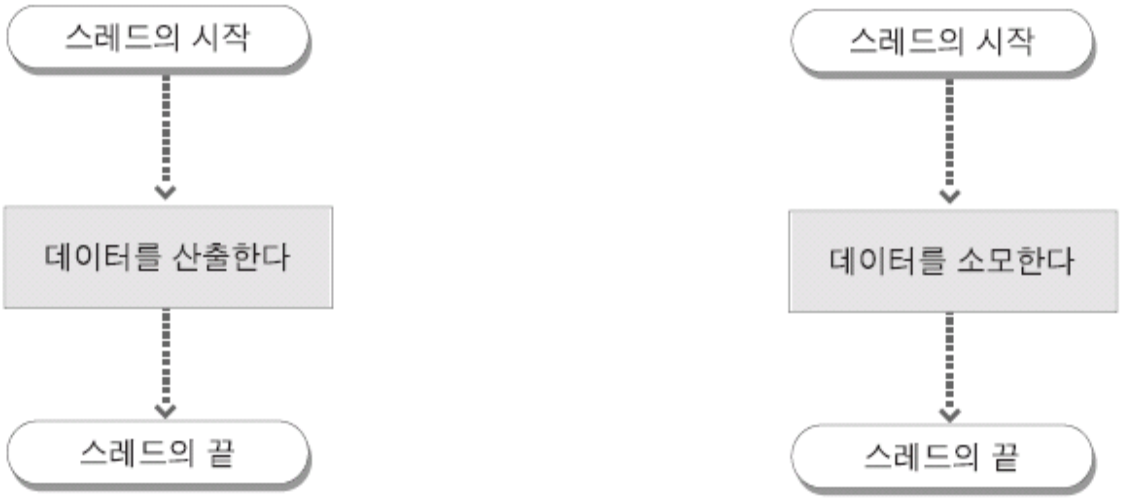
[예제 18-8] Runnable 인터페이스를 구현하는 Numbers의 서브클래스

```
1    package kr.co.asw; ---------- 이 클래스는 asw.co.kr이라는 도메인을 가진 회사에 속합니다
2    public class NumbersRunnable
                    extends kr.co.bsw.Numbers implements Runnable {
3        public void run() {
4            list(1, 30); ---------- 슈퍼클래스의 메소드 호출
5        }
6    }
```

03. 스레드간의 커뮤니케이션

스레드간 커뮤니케이션의 필요성

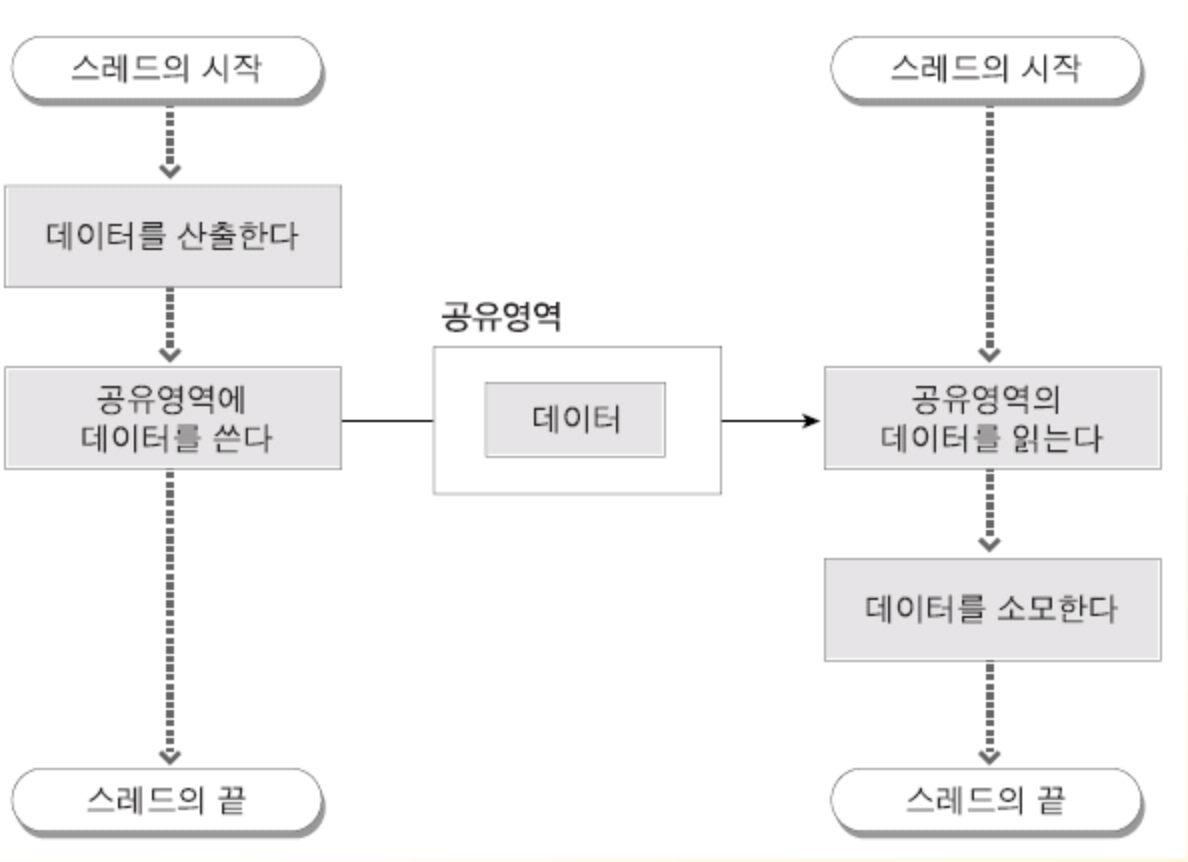
- 다음과 같은 두 스레드가 있다고 가정합시다.



03. 스레드간의 커뮤니케이션

스레드간 커뮤니케이션의 필요성

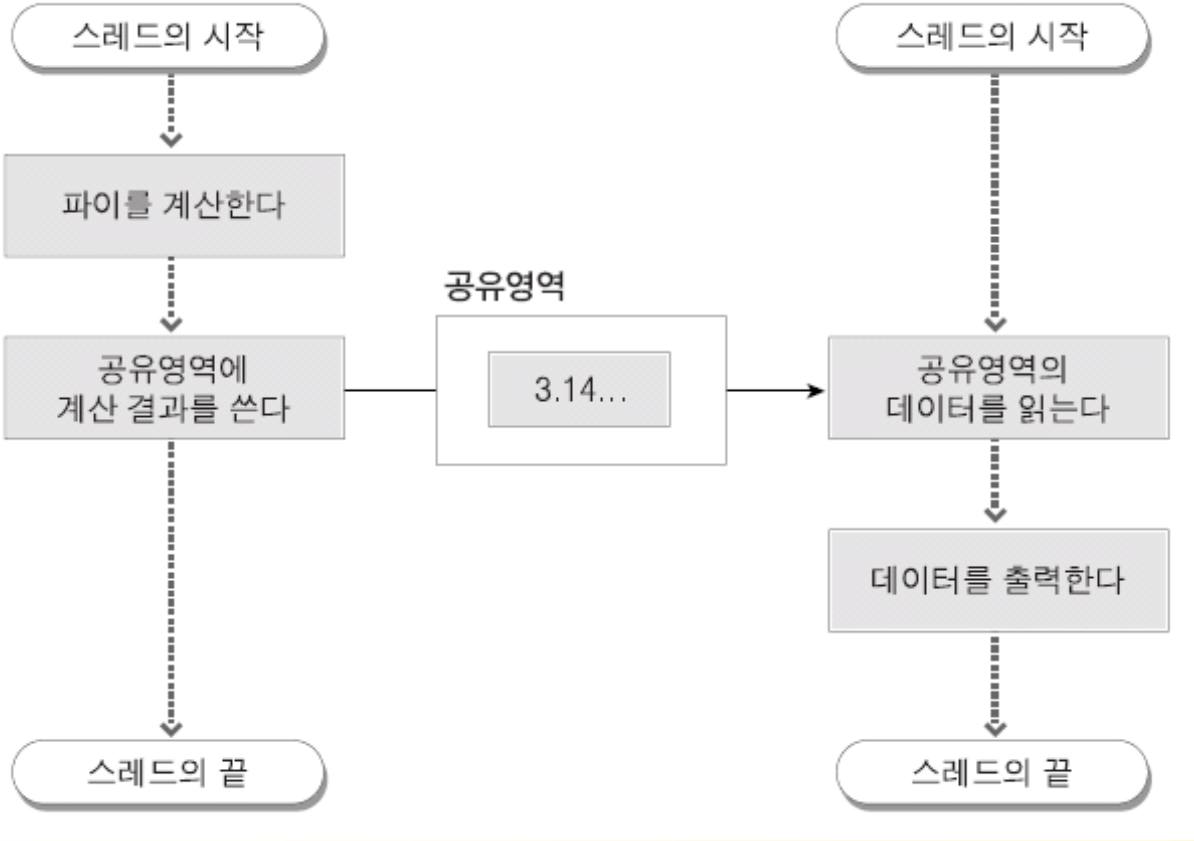
- 두 스레드가 데이터를 교환하는 기본적인 방법



03. 스레드간의 커뮤니케이션

스레드간의 데이터 교환

- 지금부터 작성할 예제의 스레드 구성



03. 스레드간의 커뮤니케이션

스레드간의 데이터 교환

- 공유 영역을 만드는 방법

  레퍼런스 타입으로 선언해야 여러 스레드가 참조값을 가지고 접근할 수 있습니다.

```
class SharedArea {    // 공유 영역을 표현하는 클래스
    double result;
}
```

공유 데이터를 저장할 필드

03. 스레드간의 커뮤니케이션

스레드간의 데이터 교환

[예제 18-9] 원주율 π를 계산해서 출력하는 멀티스레드 프로그램 (미완성)

main 메소드를 포함하는 클래스

```
1   class MultithreadExample4 {
2       public static void main(String args[]) {
3           CalcThread thread1 = new CalcThread();
4           PrintThread thread2 = new PrintThread();
5           SharedArea obj = new SharedArea();
6           thread1.sharedArea = obj;
7           thread2.sharedArea = obj;
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 객체를 생성해서 그 객체의 참조값을 두 스레드의 필드에 저장합니다

공유 영역 클래스

```
1   class SharedArea {
2       double result;
3   }
```

파이를 계산하는 스레드 클래스

```
1   class CalcThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           double total = 0.0;
5           for (int cnt = 1; cnt < 1000000000; cnt += 2)
6               if (cnt / 2 % 2 == 0)
7                   total += 1.0 / cnt;
8               else
9                   total -= 1.0 / cnt;
10          sharedArea.result = total * 4;
11      }
12  }
```

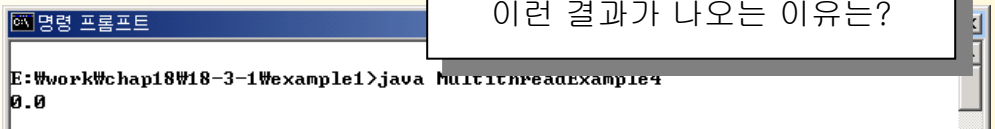
계산 결과를 공유 영역에 씁니다

파이를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           System.out.println(sharedArea.result);
5       }
6   }
```

공유 영역의 데이터를 출력합니다

이런 결과가 나오는 이유는?



02. 멀티스레드 프로그램의 작성 방법

스레드간의 데이터 교환

- 데이터 교환의 타이밍을 맞추는 방법
  - 가장 간단한 방법은 공유 영역 안에 **데이터 유무를 표시하는 필드**를 추가하는 것입니다.

```
class SharedArea {
    double result;
    boolean isReady;
}
```

공유 데이터가 쓰여졌는지 여부를 표현하는 필드

03. 스레드간의 커뮤니케이션

스레드간의 데이터 교환

[예제 18-10] 원주율 π를 계산해서 출력하는 멀티스레드 프로그램 (완성)

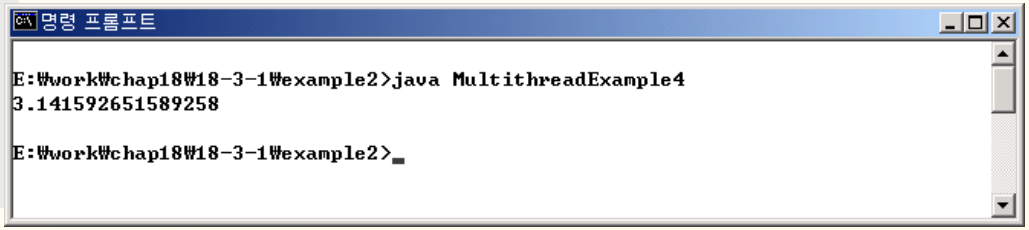
main 메소드를 포함하는 클래스

```
1   class MultithreadExample4 {
2       public static void main(String args[]) {
3           CalcThread thread1 = new CalcThread();
4           PrintThread thread2 = new PrintThread();
5           SharedArea obj = new SharedArea();
6           thread1.sharedArea = obj;
7           thread2.sharedArea = obj;
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 클래스

```
1   class SharedArea {
2       double result;
3       boolean isReady;  --------
4   }
```

공유 데이터가 쓰여졌는지 여부를 표현하는 필드. 디폴트 값은 false



파이를 계산하는 스레드 클래스

```
1   class CalcThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           double total = 0.0;
5           for (int cnt = 1; cnt < 1000000000; cnt += 2)
6               if (cnt / 2 % 2 == 0)
7                   total += 1.0 / cnt;
8               else
9                   total -= 1.0 / cnt;
10          sharedArea.result = total * 4;
11          sharedArea.isReady = true;  ----------
12      }
13  }
```

SharedArea 객체의 isReady 필드 값을 true로 설정합니다.

파이를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           while(sharedArea.isReady != true)
5               continue;
6           System.out.println(sharedArea.result);
7       }
8   }
```

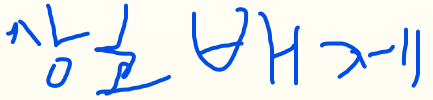
SharedArea 객체의 isReady 필드 값이 true가 될 때까지 루프를 반복합니다.

03. 스레드간의 커뮤니케이션

critical section의 동기화

- critical section
  - 스레드 실행 중에 다른 스레드로 제어가 넘어가면 문제를 일으킬 수 있는 부분
  - 주로 공유 데이터를 사용하는 부분임

- critical section의 **동기화(synchronization)**
  - critical section이 실행되는 동안 다른 스레드가 공유 데이터를 사용할 수 없도록 만드는 것



03. 스레드간의 커뮤니케이션

critical section의 동기화

[예제 18-11] 은행 계좌 클래스

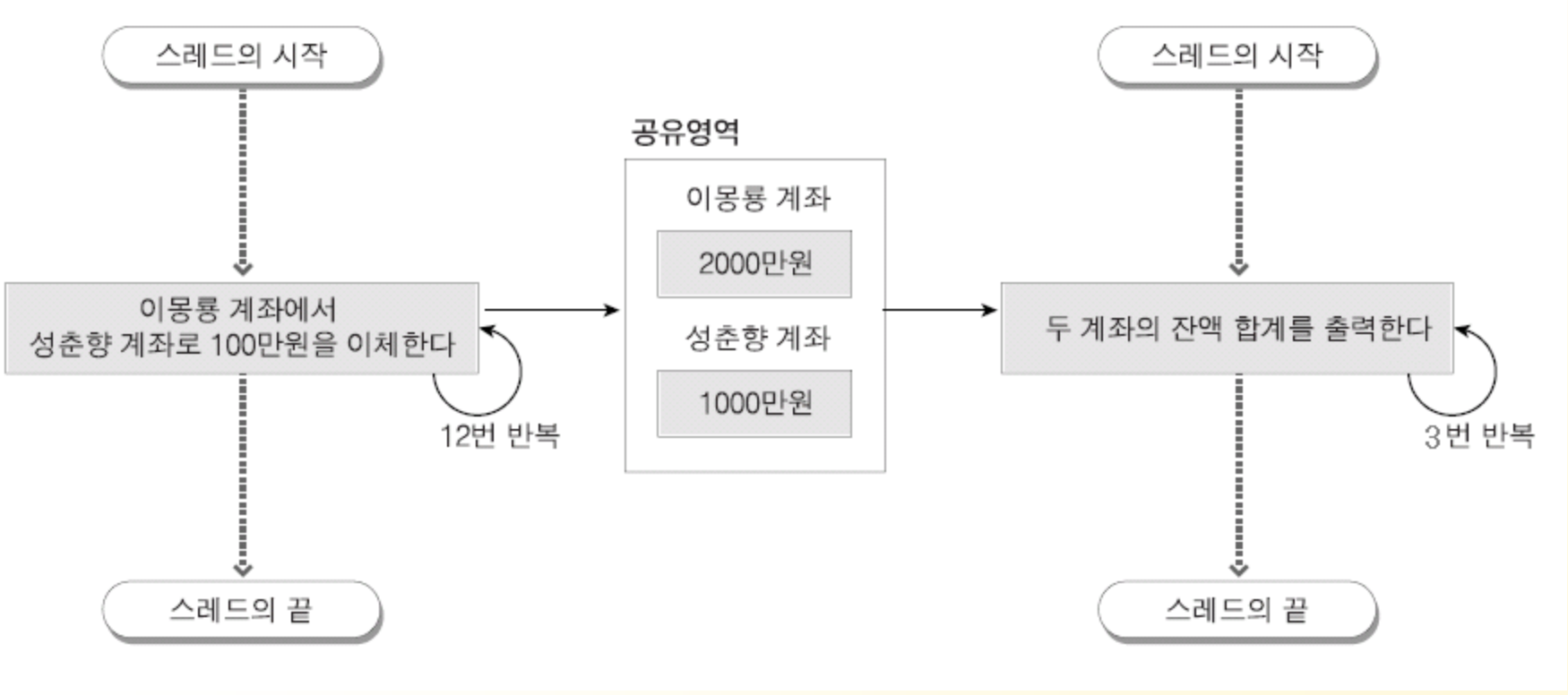
```
1    class Account {
2        String accountNo;    // 계좌번호
3        String ownerName;    // 예금주 이름
4        int balance;         // 잔액
5        Account(String accountNo, String ownerName, int balance) {
6            this.accountNo = accountNo;
7            this.ownerName = ownerName;
8            this.balance = balance;
9        }
10       void  deposit(int amount)  {
11           balance += amount;
12       }
13       int withdraw(int amount) {
14           if (balance < amount)
15               return 0;
16           balance -= amount;
17           return amount;
18       }
19   }
```

지금부터 작성할 예제에서 사용할 클래스

03. 스레드간의 커뮤니케이션

critical section의 동기화

- 지금부터 작성할 예제의 스레드 구성



03. 스레드간의 커뮤니케이션

critical section의 동기화

[예제 18-12] 계좌 이체와 잔액 합계 출력을 하는 멀티스레드 프로그램(미완성)

main 메소드를 포함하는 클래스

```
1   class MultithreadExample5 {
2       public static void main(String args[]) {
3           SharedArea area = new SharedArea();
4           area.account1 = new Account("111-111-1111", "이몽룡" , 20000000);
5           area.account2 = new Account("222-222-2222", "성춘향" , 10000000);
6           TransferThread thread1 = new TransferThread(area);
7           PrintThread thread2 = new PrintThread(area);
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 클래스

```
1   class SharedArea {
2       Account account1;   // 이몽룡의 계좌
3       Account account2;   // 성춘향의 계좌
4   }
```

계좌 이체를 수행하는 스레드 클래스

```
1   class TransferThread extends Thread {
2       SharedArea sharedArea;
3       TransferThread(SharedArea area) {   // 생성자
4           sharedArea = area;
5       }
6       public void run() {
7           for (int cnt = 0; cnt < 12; cnt ++) {
8               sharedArea.account1.withdraw(1000000);
9               System.out.print("이몽룡 계좌: 100만원 인출,");
10              sharedArea.account2.deposit(1000000);
11              System.out.println("성춘향 계좌: 100만원 입금");
12          }
13      }
14  }
```

계좌 잔액 합계를 출력하는 스레드 클래스

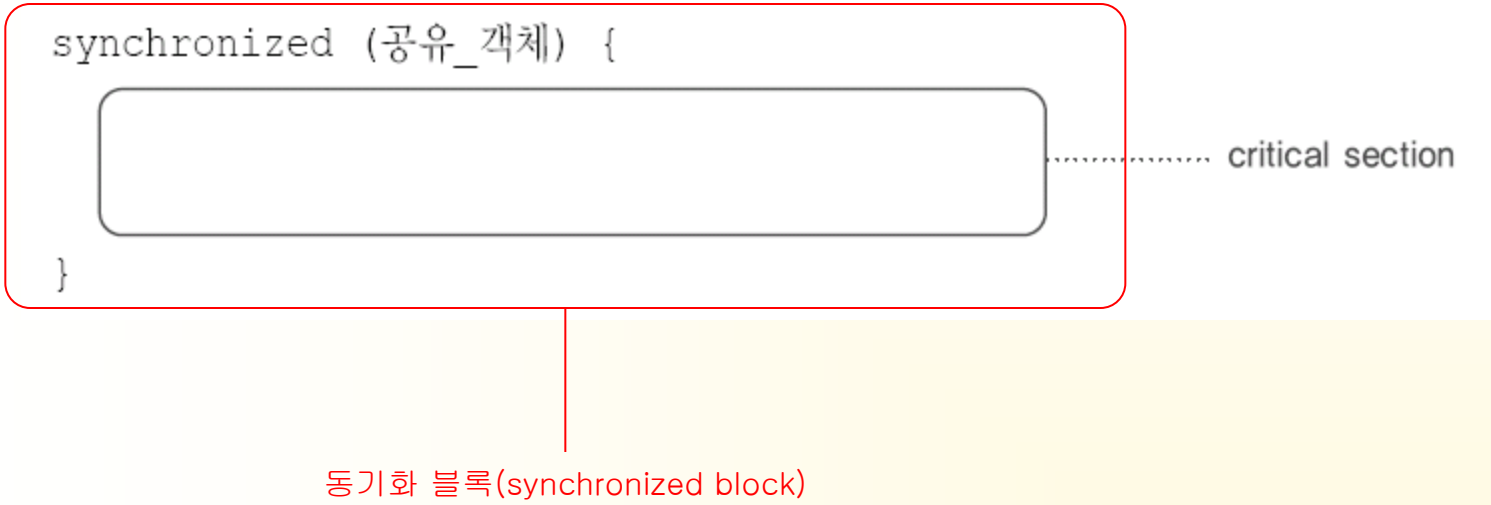
```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       PrintThread(SharedArea area) {   // 생성자
4           sharedArea = area;
5       }
6       public void run() {
7           for (int cnt = 0; cnt < 3; cnt ++) {
8               int sum = sharedArea.account1.balance +
                          sharedArea.account2.balance;
9               System.out.println("계좌 잔액 합계: " + sum);
10              try {
11                  Thread.sleep(1);
12              } catch (InterruptedException e) {
13                  System.out.println(e.getMessage());
14              }
15          }
16      }
17  }
```

critical section

03. 스레드간의 커뮤니케이션

critical section의 동기화

- critical section의 동기화 방법



03. 스레드간의 커뮤니케이션

critical section의 동기화

[예제 18-13] 계좌 이체와 잔액 합계 출력을 하는 멀티스레드 프로그램(1)

main 메소드를 포함하는 클래스

```
1   class MultithreadExample5 {
2       public static void main(String args[]) {
3           SharedArea area = new SharedArea();
4           area.account1 = new Account("111-111-1111", "이몽룡" , 20000000);
5           area.account2 = new Account("222-222-2222", "성춘향" , 10000000);
6           TransferThread thread1 = new TransferThread(area);
7           PrintThread thread2 = new PrintThread(area);
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 클래스

```
1   class SharedArea {
2       Account account1;   // 이몽룡의 계좌
3       Account account2;   // 성춘향의 계좌
4   }
```

[계좌 이체를 수행하는 스레드 클래스

```
1   class TransferThread extends Thread {
2       SharedArea sharedArea;
3       TransferThread(SharedArea area) {   // 생성자
4           sharedArea = area;
5       }
6       public void run() {
7           for (int cnt = 0; cnt < 12; cnt ++) {
8               synchronized (sharedArea) {
9                   sharedArea.account1.withdraw(1000000);
10                  System.out.print("이몽룡 계좌: 100만원 인출,");
11                  sharedArea.account2.deposit(1000000);
12                  System.out.println("성춘향 계좌: 100만원 입금");
13              }
14          }
15      }
16  }
```

동기화 블록

[계좌 잔액 합계를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       PrintThread(SharedArea area) {   // 생성자
4           sharedArea = area;
5       }
6       public void run() {
7           for (int cnt = 0; cnt < 3; cnt ++) {
8               synchronized (sharedArea) {
9                   int sum = sharedArea.account1.balance +
10                          sharedArea.account2.balance;
11                  System.out.println("계좌 잔액 합계: " + sum);
12              }
13              try {
14                  Thread.sleep(1);
15              } catch (InterruptedException e) {
16                  System.out.println(e.getMessage());
17              }
18          }
19      }
20  }
```

동기화 블록

03. 스레드간의 커뮤니케이션

critical section의 동기화

[예제 18-14] 계좌 이체와 잔액 합계 출력을 하는 멀티스레드 프로그램(2)

main 메소드를 포함하는 클래스

```
1   class MultithreadExample6 {
2       public static void main(String args[]) {
3           SharedArea area = new SharedArea();
4           area.account1 = new Account("111-111-1111", "이몽룡" , 20000000);
5           area.account2 = new Account("222-222-2222", "성춘향" , 10000000);
6           TransferThread thread1 = new TransferThread(area);
7           PrintThread thread2 = new PrintThread(area);
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 클래스

```
1   class SharedArea {
2       Account account1;   // 이몽룡의 계좌
3       Account account2;   // 성춘향의 계좌
4       void transfer(int amount) {    // 계좌 이체를 한다
5           synchronized (this) {
6               account1.withdraw(1000000);
7               System.out.print("이몽룡 계좌: 100만원 인출,");
8               account2.deposit(1000000);
9               System.out.println("성춘향 계좌: 100만원 입금");
10          }
11      }
12      int getTotal() {              // 잔액 합계를 구한다
13          synchronized (this) {
14              return account1.balance + account2.balance;
15          }
16      }
17  }
```

[계좌 이체를 수행하는 스레드 클래스

```
1   class TransferThread extends Thread {
2       SharedArea sharedArea;
3       TransferThread(SharedArea area) {   // 생성자
4           sharedArea = area;
5       }
6       public void run() {
7           for (int cnt = 0; cnt < 12; cnt ++) {
8               sharedArea.transfer(100); --------- 계좌 이체 메소드 호출
9           }
10      }
11  }
```

[계좌 잔액 합계를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       PrintThread(SharedArea area) {   // 생성자
4           sharedArea = area;
5       }
6       public void run() {
7           for (int cnt = 0; cnt < 3; cnt ++) {
8               int sum = sharedArea.getTotal(); --------- 잔액 합계 메소드 호출
9               System.out.println("계좌 잔액 합계: " + sum);
10              try {
11                  Thread.sleep(1);
12              } catch (InterruptedException e) {
13                  System.out.println(e.getMessage());
14              }
15          }
16      }
17  }
```

03. 스레드간의 커뮤니케이션

동기화 메소드

- 메소드를 동기화하는 방법
  - 메소드 선언 제일 앞에 synchronized 키워드를 쓰면 됩니다.

```
synchronized void transfer(int amount) {
    account1.withdraw(1000000);
    System.out.print("이몽룡 계좌: 100만원 인출,");
    account2.deposit(1000000);
    System.out.println("성춘향 계좌: 100만원 입금");
}
```

동기화 메소드

03. 스레드간의 커뮤니케이션

동기화 메소드

[예제 18-15] 계좌 이체와 잔액 합계 출력을 하는 멀티스레드 프로그램(3)

main 메소드를 포함하는 클래스

```
1   class MultithreadExample6 {
2       public static void main(String args[]) {
3           SharedArea area = new SharedArea();
4           area.account1 = new Account("111-111-1111", "이몽룡" , 20000000);
5           area.account2 = new Account("222-222-2222", "성춘향" , 10000000);
6           TransferThread thread1 = new TransferThread(area);
7           PrintThread thread2 = new PrintThread(area);
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 클래스

```
1   class SharedArea {
2       Account account1;   // 이몽룡의 계좌
3       Account account2;   // 성춘향의 계좌
4       synchronized void transfer(int amount) {
5           account1.withdraw(1000000);
6           System.out.print("이몽룡 계좌: 100만원 인출,");
7           account2.deposit(1000000);
8           System.out.println("성춘향 계좌: 100만원 입금");
9       }
10      synchronized int getTotal() {
11          return account1.balance + account2.balance;
12      }
13  }
```

동기화 메소드

동기화 메소드

[계좌 이체를 수행하는 스레드 클래스

```
1   class TransferThread extends Thread {
2       SharedArea sharedArea;
3       TransferThread(SharedArea area) {
4           sharedArea = area;
5       }
6       public void run() {
7          for (int cnt = 0; cnt < 12; cnt ++) {
8              sharedArea.transfer(100);
9          }
10      }
11  }
```

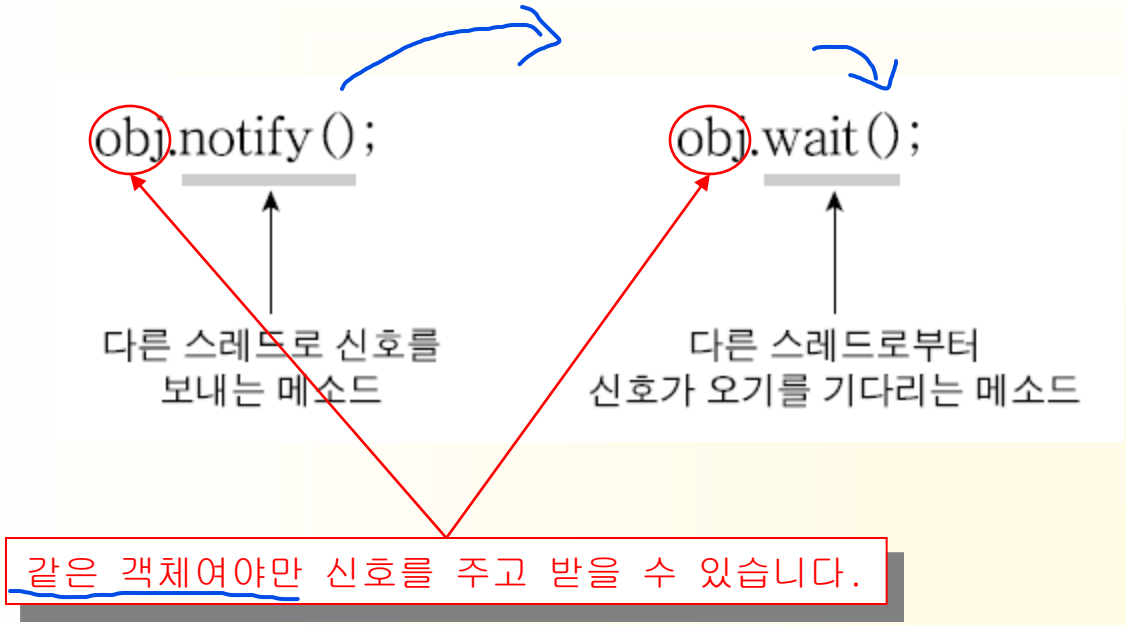
[계좌 잔액 합계를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       PrintThread(SharedArea area) {
4           sharedArea = area;
5       }
6       public void run() {
7          for (int cnt = 0; cnt < 3; cnt ++) {
8              int sum = sharedArea.getTotal();
9              System.out.println("계좌 잔액 합계: " + sum);
10             try {
11                 Thread.sleep(1);
12             } catch (InterruptedException e) {
13                 System.out.println(e.getMessage());
14             }
15         }
16      }
17  }
```

03. 스레드간의 커뮤니케이션

스레드간의 신호 전송

- 신호를 보내는 **notify 메소드**와 신호를 받는 **wait 메소드**의 호출 방법



03. 스레드간의 커뮤니케이션

스레드간의 신호 전송

- 신호를 주고 받는 notify 메소드와 wait 메소드

```
class CalcThread extends Thread {
    ...
    public void run() {
        synchronized(obj) {
            obj.notify();
        }
    }
}
```

신호를 보냅니다 →

```
class PrintThread extends Thread {
    ...
    public void run() {
        synchronized(obj) {
            obj.wait();
        }
    }
}
```

03. 스레드간의 커뮤니케이션

스레드간의 신호 전송

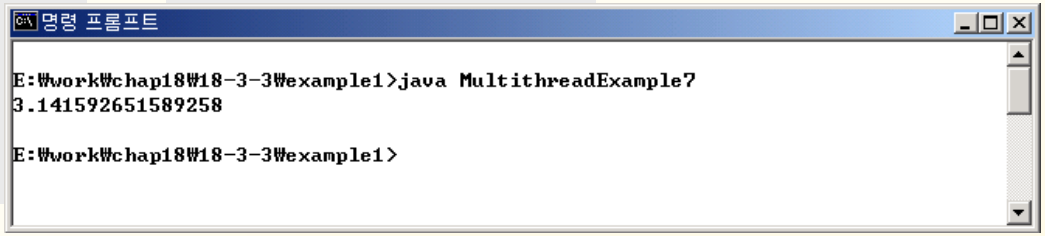
[예제 18-16] notify, wait 메소드의 사용 예를 보여주는 원주율 계산 프로그램

main 메소드를 포함하는 클래스

```
1   class MultithreadExample7 {
2       public static void main(String args[]) {
3           CalcThread thread1 = new CalcThread();
4           PrintThread thread2 = new PrintThread();
5           SharedArea obj = new SharedArea();
6           thread1.sharedArea = obj;
7           thread2.sharedArea = obj;
8           thread1.start();
9           thread2.start();
10      }
11  }
```

공유 영역 클래스

```
1   class SharedArea {
2       double result;
3       boolean isReady;
4   }
```



파이를 계산하는 스레드 클래스

```
1   class CalcThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           double total = 0.0;
5           for (int cnt = 1; cnt < 1000000000; cnt += 2)
6               if (cnt / 2 % 2 == 0)
7                   total += 1.0 / cnt;
8               else
9                   total -= 1.0 / cnt;
10          sharedArea.result = total * 4;
11          sharedArea.isReady = true;
12          synchronized (sharedArea) {
13              sharedArea.notify();   ------ 신호를 보냅니다.
14          }
15      }
16  }
```

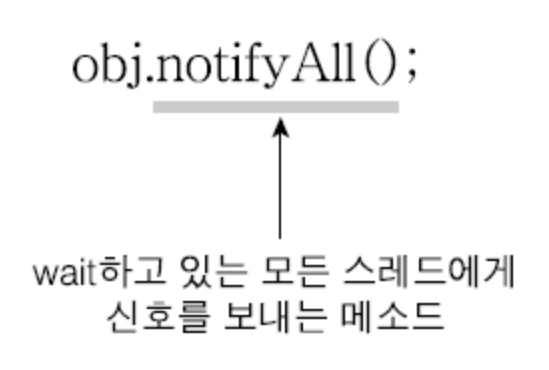
파이를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           if (sharedArea.isReady != true) {
5               try {
6                   synchronized (sharedArea) {
7                       sharedArea.wait();   ------ 신호를 받습니다.
8                   }
9               }
10              catch (InterruptedException e) {
11                  System.out.println(e.getMessage());
12              }
13          }
14          System.out.println(sharedArea.result);
15      }
16  }
```

03. 스레드간의 커뮤니케이션

스레드간의 신호 전송

- 대기하고 있는 모든 스레드로 신호를 보내는 notifyAll 메소드



03. 스레드간의 커뮤니케이션

스레드간의 신호 전송

[예제 18-17] notifyAll 메소드의 사용 예

main 메소드를 포함하는 클래스

```
1 class MultithreadExample8 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         SimplePrintThread thread3 = new SimplePrintThread();
6         LuxuryPrintThread thread4 = new LuxuryPrintThread();
7         SharedArea obj = new SharedArea();
8         thread1.sharedArea = obj;
9         thread2.sharedArea = obj;
10        thread3.sharedArea = obj;
11        thread4.sharedArea = obj;
12        thread1.start();
13        thread2.start();
14        thread3.start();
15        thread4.start();
16    }
17 }
```

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3     boolean isReady;
4 }
```

파이를 예쁘게 출력하는 스레드 클래스

```
1 class LuxuryPrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait(); -- 신호를 기다립니다.
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.println("*** π = " + sharedArea.result + " ***");
15    }
16 }
```

파이를 계산하는 스레드 클래스

```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11        sharedArea.isReady = true;
12        synchronized (sharedArea) {
13            sharedArea.notifyAll(); ---- 기다리고 있는 모든 스레드로 신호를 보냅니다.
14        }
15    }
16 }
```

파이를 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait(); -- 신호를 기다립니다.
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.println(sharedArea.result);
15    }
16 }
```

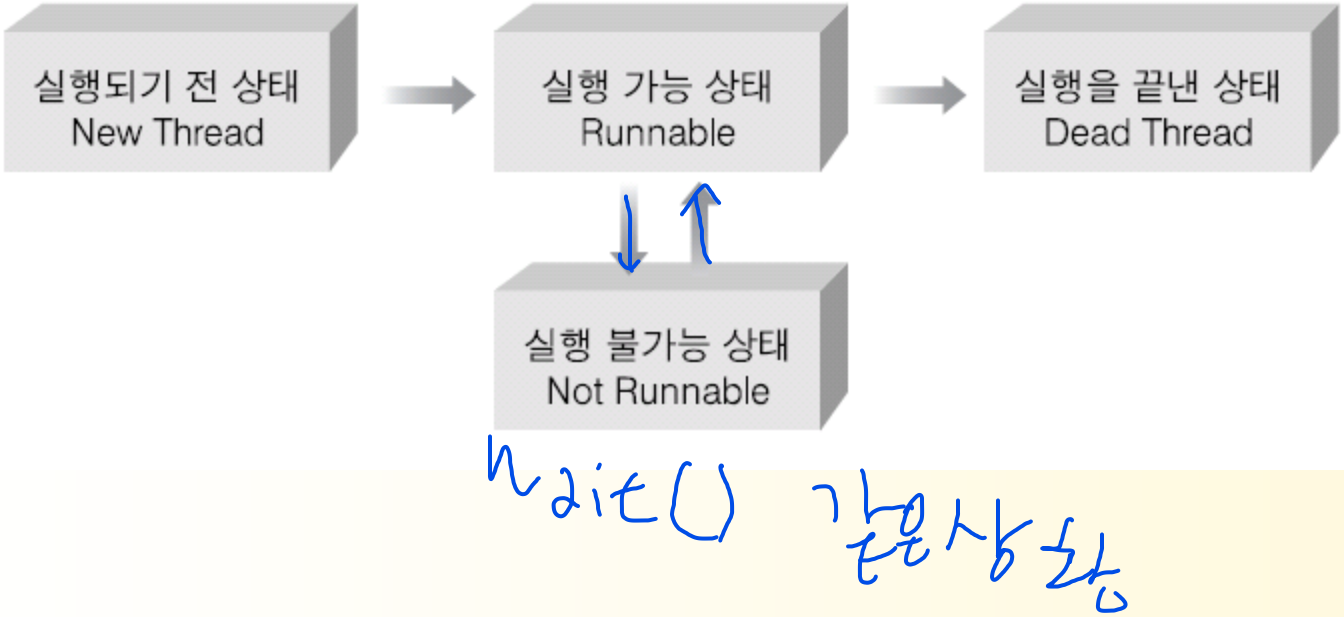
파이를 소수점 두자리까지 출력하는 스레드 클래스

```
1 class SimplePrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait(); -- 신호를 기다립니다.
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.printf("%.2f %n", sharedArea.result);
15    }
16 }
```

04. 스레드의 상태

스레드의 라이프 사이클

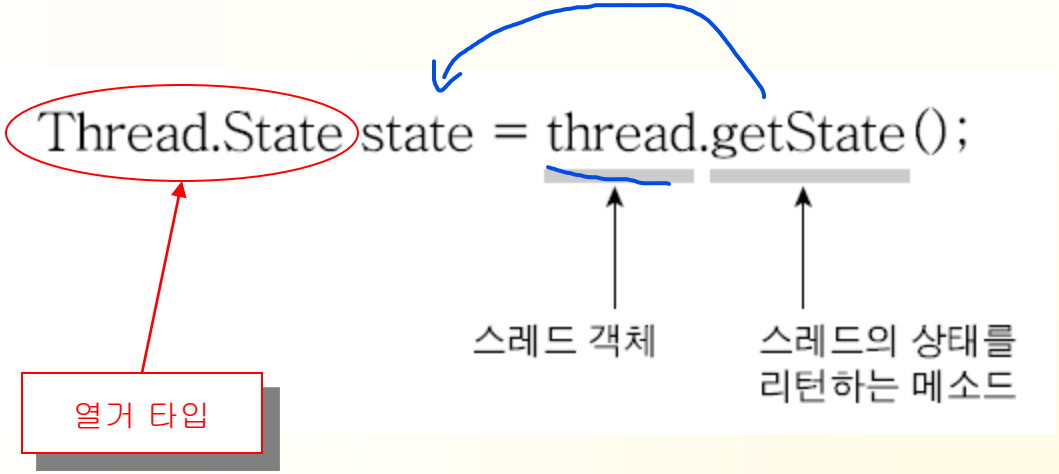
- 스레드의 라이프 사이클
  - 스레드가 생성되서 start 메소드를 호출하기 전까지의 상태
  - run 메소드 실행 중 상태 -> 다시 두 가지 상태로 나뉨
  - run 메소드 완료 후의 상태



04. 스레드의 상태

스레드의 상태를 알아내는 메소드

- Thread 클래스의 getState 메소드



04. 스레드의 상태

스레드의 상태를 알아내는 메소드

- 열거 타입 Thread.State의 열거값

열거 상수	의미하는 스레드의 상태
NEW	실행되기 전 상태
RUNNABLE	실행 가능 상태
WAITING	wait 메소드를 호출하고 있는 상태
TIMED_WAITING	sleep 메소드를 호출하고 있는 상태
BLOCKED	다른 스레드의 동기화 블록이나 동기화 메소드가 끝나기를 기다리고 있는 상태
TERMINATED	실행을 마친 상태

04. 스레드간의 상태

스레드의 상태를 알아내는 메소드

[예제 18-18] 모니터링 스레드가 추가된 원주율 계산 프로그램

main 메소드를 포함하는 클래스

```
1   class MultithreadExample9 {
2       public static void main(String args[]) {
3           CalcThread thread1 = new CalcThread();
4           PrintThread thread2 = new PrintThread();
5           MonitorThread thread3 = new MonitorThread(thread1);
6           SharedArea obj = new SharedArea();
7           thread1.sharedArea = obj;
8           thread2.sharedArea = obj;
9           thread1.start();
10          thread2.start();
11          thread3.start();
12      }
13  }
```

공유 영역 클래스

```
1   class SharedArea {
2       double result;
3       boolean isReady;
4   }
```

다른 스레드를 모니터링하는 스레드 클래스

```
1   class MonitorThread extends Thread {
2       Thread thread;
3       MonitorThread(Thread thread) {   // 생성자
4           this.thread = thread;
5       }
6       public void run() {
7           while (true) {
8               Thread.State state = thread.getState();
9               System.out.println("쓰레드의 상태: " + state);
10              if (state == Thread.State.TERMINATED)
11                  break;
12              try {
13                  Thread.sleep(2000);
14              } catch (InterruptedException e) {
15                  e.printStackTrace();
16              }
17          }
18      }
19  }
```

파이를 계산하는 스레드 클래스

```
1   class CalcThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           double total = 0.0;
5           for (int cnt = 1; cnt < 1000000000; cnt += 2)
6               if (cnt / 2 % 2 == 0)
7                   total += 1.0 / cnt;
8               else
9                   total -= 1.0 / cnt;
10          sharedArea.result = total * 4;
11          sharedArea.isReady = true;
12          synchronized (sharedArea) {
13              sharedArea.notify();
14          }
15      }
16  }
```

파이를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           if (sharedArea.isReady != true) {
5               synchronized (sharedArea) {
6                   try {
7                       sharedArea.wait();
8                   }
9                   catch (InterruptedException e) {
10                      System.out.println(e.getMessage());
11                  }
12              }
13          }
14          System.out.println(sharedArea.result);
15      }
16  }
```

명령 프롬프트

```
E:₩work₩chap18₩18-4>java MultithreadExample9
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
3.141592651589258
쓰레드의 상태: TERMINATED

E:₩work₩chap18₩18-4>_
```

04. 스레드간의 상태

스레드의 상태를 알아내는 메소드

[CalcThread를 모니터링하도록 수정된 예제 18-18]

main 메소드를 포함하는 클래스

```
1   class MultithreadExample9 {
2       public static void main(String args[]) {
3           CalcThread thread1 = new CalcThread();
4           PrintThread thread2 = new PrintThread();
5           MonitorThread thread3 = new MonitorThread(thread2);
6           SharedArea obj = new SharedArea();
7           thread1.sharedArea = obj;
8           thread2.sharedArea = obj;
9           thread1.start();
10          thread2.start();
11          thread3.start();
12      }
13  }
```

공유 영역 클래스

```
1   class SharedArea {
2       double result;
3       boolean isReady;
4   }
```

다른 스레드를 모니터링하는 스레드 클래스

```
1   class MonitorThread extends Thread {
2       Thread thread;
3       MonitorThread(Thread thread) {   // 생성자
4           this.thread = thread;
5       }
6       public void run() {
7           while (true) {
8               Thread.State state = thread.getState();
9               System.out.println("쓰레드의 상태: " + state);
10              if (state == Thread.State.TERMINATED)
11                  break;
12              try {
13                  Thread.sleep(2000);
14              } catch (InterruptedException e) {
15                  e.printStackTrace();
16              }
17          }
18      }
19  }
```

파이를 계산하는 스레드 클래스

```
1   class CalcThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           double total = 0.0;
5           for (int cnt = 1; cnt < 1000000000; cnt += 2)
6               if (cnt / 2 % 2 == 0)
7                   total += 1.0 / cnt;
8               else
9                   total -= 1.0 / cnt;
10          sharedArea.result = total * 4;
11          sharedArea.isReady = true;
12          synchronized (sharedArea) {
13              sharedArea.notify();
14          }
15      }
16  }
```

파이를 출력하는 스레드 클래스

```
1   class PrintThread extends Thread {
2       SharedArea sharedArea;
3       public void run() {
4           if (sharedArea.isReady != true) {
5               synchronized (sharedArea) {
6                   try {
7                       sharedArea.wait();
8                   }
9                   catch (InterruptedException e) {
10                      System.out.println(e.getMessage());
11                  }
12              }
13          }
14          System.out.println(sharedArea.result);
15      }
16  }
```

명령 프롬프트

```
E:₩work₩chap18₩18-4>java MultithreadExample9
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
3.141592651589258
쓰레드의 상태: TERMINATED

E:₩work₩chap18₩18-4>_
```