JAVA언어로 구현하는 스레드/소켓 통신 프로그래밍

작업 스레드 생성과 실행

```
1  package threadExam.createthread;
2
3  import java.awt.Toolkit;
4
5  public class BeepPrintExample1 {
6      public static void main(String[] args) {
7          Toolkit toolkit = Toolkit.getDefaultToolkit();
8          for(int i=0; i<5; i++) {
9              toolkit.beep();
10             try { Thread.sleep(500); } catch(Exception e) {}
11         }
12
13         for(int i=0; i<5; i++) {
14             System.out.println("띵");
15             try { Thread.sleep(500); } catch(Exception e) {}
16         }
17     }
18 }
```

작업 스레드 생성과 실행

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

1. Thread 클래스로부터 직접 생성

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

```
how1)
Runnable beeptask = new BeepTask();
Thread thread = new Thread(beepTask);
```

```
how2)
Thread thread = new Thread(new Runnable(){
        public void run(){
                스레드가 실행할 코드;
        }
});
```

Thread.start();

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

2. Thread 하위 클래스로부터 생성

```
1  package threadExam.createthread;
2
3  import java.awt.Toolkit;
4
5  public class BeepThread extends Thread {
6      @Override
7      public void run() {
8          Toolkit toolkit = Toolkit.getDefaultToolkit();
9          for(int i=0; i<5; i++) {
10             toolkit.beep();
11             try { Thread.sleep(500); } catch(Exception e) {}
12         }
13     }
14 }
```

```java
package threadExam.createthread;
import java.awt.Toolkit;
public class BeepPrintExample3 {
    public static void main(String[] args) {
        //how1
        Thread thread = new BeepThread();

        //how2
        /*Thread thread = new Thread() {
            @Override
            public void run() {
                Toolkit toolkit = Toolkit.getDefaultToolkit();
                for(int i=0; i<5; i++) {
                    toolkit.beep();
                    try { Thread.sleep(500); } catch(Exception e) {}
                }
            }
        };*/

        thread.start();

        for(int i=0; i<5; i++) {
            System.out.println("띵");
            try { Thread.sleep(500); } catch(Exception e) {}
        }
    }
}
```

<termin
띵 띵 띵 띵 띵

작업 스레드 생성과 실행

- 스레드의 이름
  - 메인 스레드 이름: main
  - 작업 스레드 이름 (자동 설정) : Thread-n

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

  - 작업 스레드 이름 변경

```
thread.setName("스레드  이름");
```

  - 코드 실행하는 현재 스레드 객체의 참조 얻기

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

※ 스레드 이름 예제

```java
package threadExam.threadName;

public class ThreadA extends Thread {
    public ThreadA() {
        setName("ThreadA");
    }

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

```java
package threadExam.threadName;

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

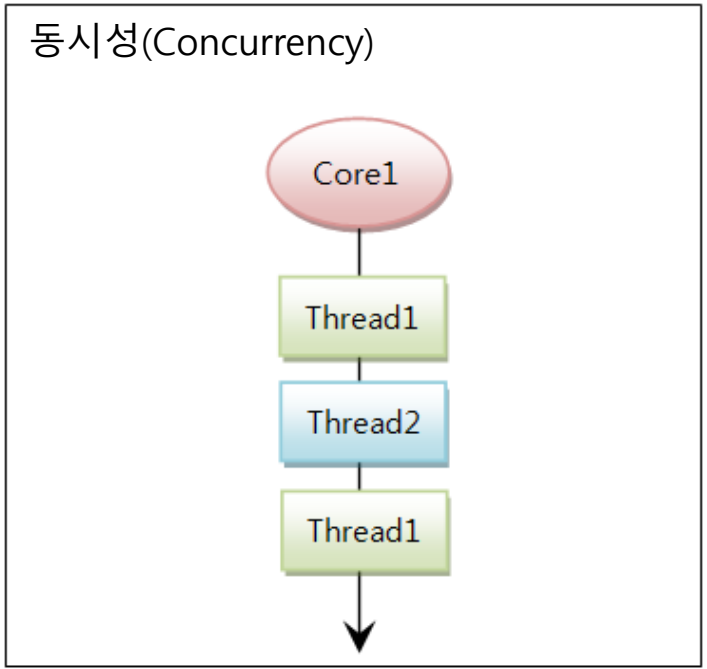
※ 스레드 이름 예제

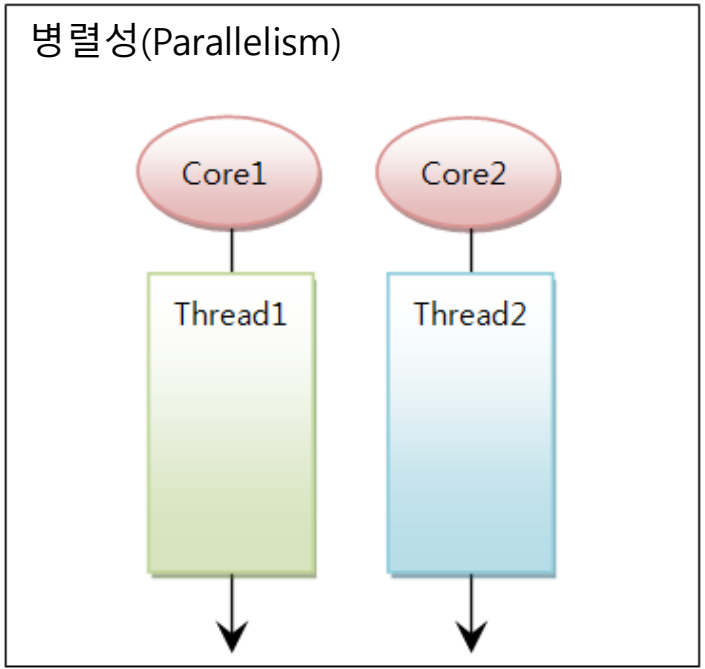
```java
package threadExam.threadName;

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

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

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

```
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
```

스레드 우선 순위

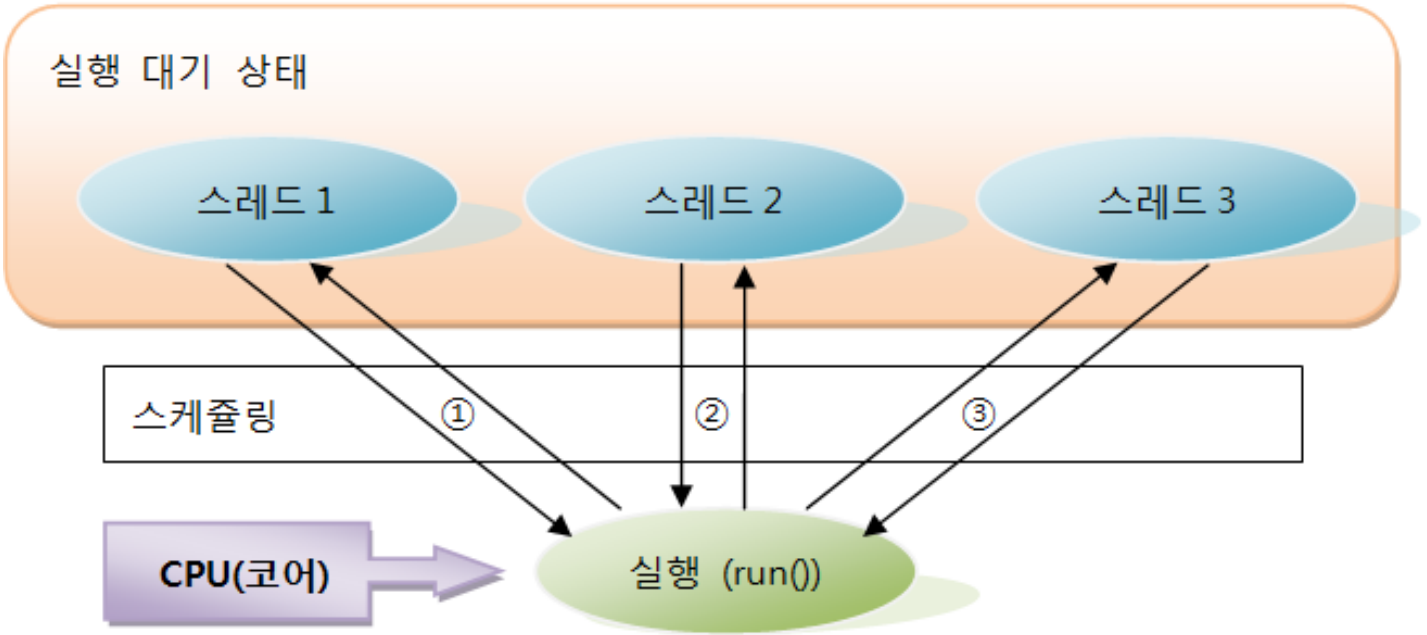
- 동시성과 병렬성
  - 동시성(Concurrency)
    - 멀티 작업을 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질
  - 병렬성(Parallelism)
    - 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질





스레드 우선 순위

- 스레드 스케줄링
  - 스레드의 개수가 코어의 수보다 많을 경우
    - 스레드를 어떤 순서로 동시성으로 실행할 것인가 결정→스레드 스케줄링
    - 스케줄링 의해 스레드들은 번갈아 가며 run() 메소드를 조금씩 실행



스레드 우선 순위

- 자바의 스레드 스케줄링
  - 우선 순위(Priority) 방식과 순환 할당(Round-Robin) 방식 사용

1. 우선 순위 방식 (코드로 제어 가능)
   - 우선 순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링
   - 1~10까지 값을 가질 수 있으며 기본은 5

2. 순환 할당 방식 (코드로 제어할 수 없음)
   - 시간 할당량(Time Slice) 정해서 하나의 스레드를 정해진 시간만큼 실행

스레드 우선 순위

- 스레드 우선 순위
  - 스레드들이 동시성을 가질 경우 우선적으로 실행할 수 있는 순위
  - 우선 순위는 1(낮음)에서부터 10(높은)까지로 부여
    - 모든 스레드들은 기본적으로 5의 우선 순위를 할당
  - 우선 순위 변경 방법

```
thread.setPriority(우선순위);
```

```
thread.setPriority(Thread.MAX_PRIORITY);
thread.setPriority(Thread.NORM_PRIORITY);
thread.setPriority(Thread.MIN_PRIORITY);
```

※ 스레드 우선 순위 예제

```
1  package threadExam.priority;
2
3  public class CalcThread extends Thread {
4      public CalcThread(String name) {
5          setName(name);
6      }
7
8      public void run() {
9          for(int i=0; i<2000000000; i++) {
10         }
11         System.out.println(getName());
12     }
13 }
```

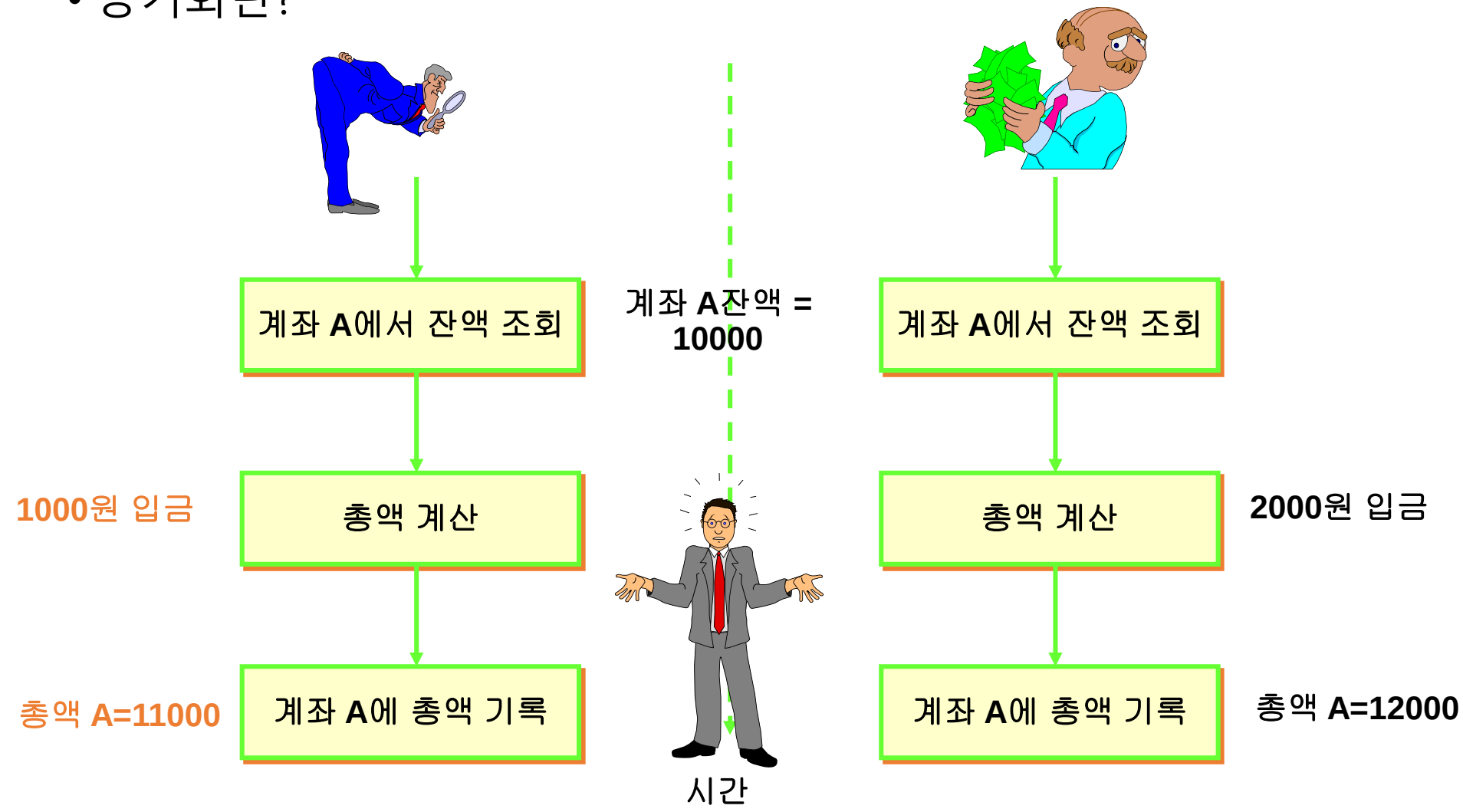
※ 스레드 우선 순위 예제

```java
package threadExam.priority;

public class PriorityExample {
    public static void main(String[] args) {
        for(int i=1; i<=10; i++) {
            Thread thread = new CalcThread("thread" + i);
            if(i != 10) {
                thread.setPriority(Thread.MIN_PRIORITY);
            } else {
                thread.setPriority(Thread.MAX_PRIORITY);
            }
            thread.start();
        }
    }
}
```

```
<terminated> Priori
thread10
thread3
thread7
thread4
thread6
thread5
thread1
thread9
thread2
thread8
```

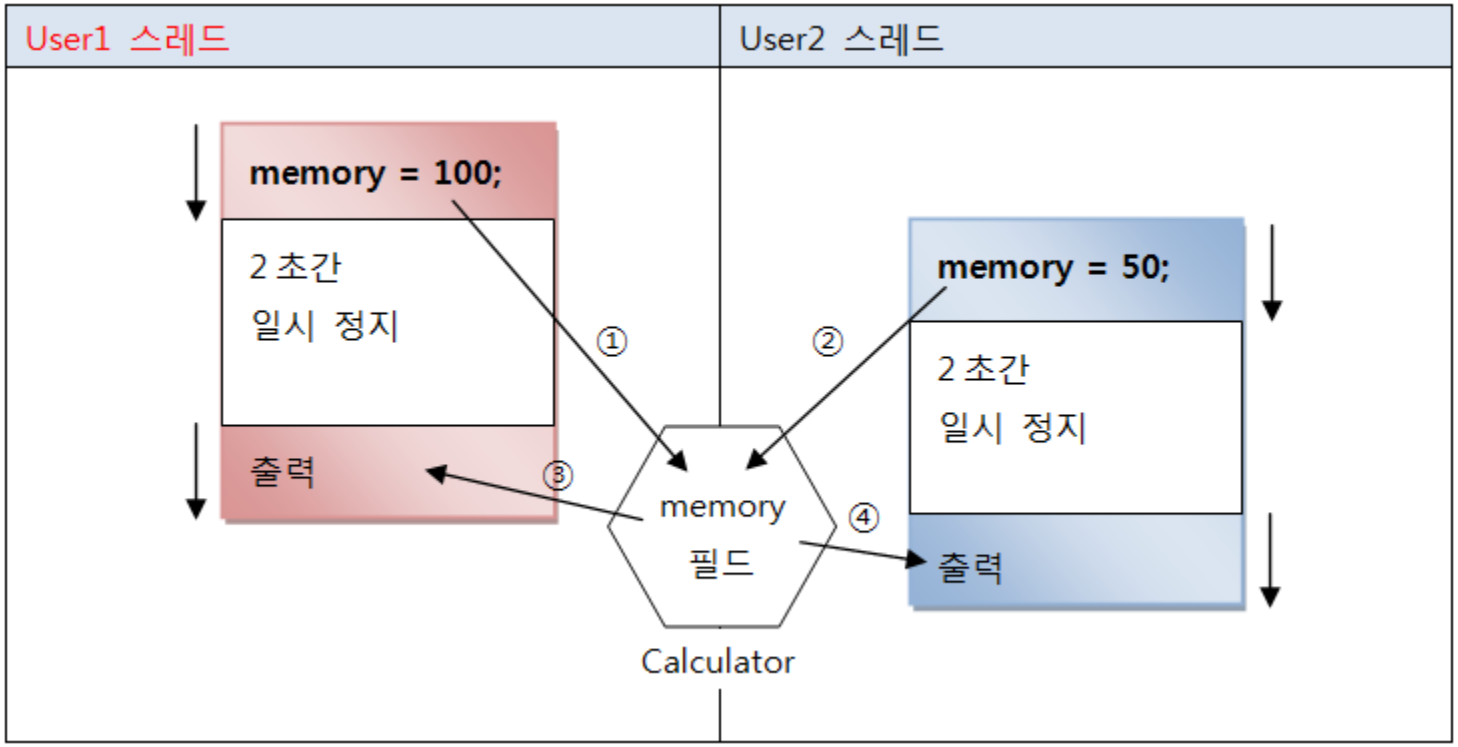
동기화 메소드와 동기화 블록

- 동기화란?



동기화 메소드와 동기화 블록

- 공유 객체를 사용할 때의 주의할 점
  - 멀티 스레드가 하나의 객체를 공유해서 생기는 오류



※ 공유객체 사용시 문제점 예제

```
1  package threadExam.unsynchronized;
2
3  public class Calculator {
4      private int memory;
5
6      public int getMemory() {
7          return memory;
8      }
9
10     public 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()
16                                 + ": " +  this.memory);
17     }
18 }
```

※ 공유객체 사용시 문제점 예제

```
1  package threadExam.unsynchronized;
2
3  public class User1 extends Thread {
4      private Calculator calculator;
5
6      public void setCalculator(Calculator calculator) {
7          this.setName("User1");
8          this.calculator = calculator;
9      }
10
11     public void run() {
12         calculator.setMemory(100);
13     }
14 }
```

※ 공유객체 사용시 문제점 예제

```
1  package threadExam.unsynchronized;
2
3  public class User2 extends Thread {
4      private Calculator calculator;
5
6      public void setCalculator(Calculator calculator) {
7          this.setName("User2");
8          this.calculator = calculator;
9      }
10
11     public void run() {
12         calculator.setMemory(50);
13     }
14 }
```

※ 공유객체 사용시 문제점 예제

```
1  package threadExam.unsynchronized;
2
3  public class MainThreadExample {
4      public static void main(String[] args) {
5          Calculator calculator = new Calculator();
6
7          User1 user1 = new User1();
8          user1.setCalculator(calculator);
9          user1.start();
10
11         User2 user2 = new User2();
12         user2.setCalculator(calculator);
13         user2.start();
14     }
15 }
```

```
<terminated> MainTh
User2: 50
User1: 50
```

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 함
  - 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
   ...
}
```

동기화 메소드와 동기화 블록

- 동기화 메소드
  - 동일한 객체에 대하여 여러 스레드의 중첩 실행을 방지할 때 사용하는 메소드
    - 동일한 객체에 대하여 한 스레드가 동기화 메소드를 실행하고 있으면, 다른 스레드는 그 메소드를 실행할 수 없음
  - 자바에서의 동기화 메소드 사용
    - "synchronized"로 선언
    - 스레드가 어떤 객체의 동기화 메소드를 호출하면 그 객체는 lock됨
    - 또 다른 스레드가 동일 객체의 동기화 메소드를 호출하면, 그 스레드는 lock이 해제될 때까지 블록됨

    ⇒ "상호 배제"

※ 동기화 예제 1

```
1  package threadExam.Synchronized;
2
3  public class Calculator {
4      private int memory;
5
6      public int getMemory() {
7          return memory;
8      }
9
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()
16                            + ": " +  this.memory);
17     }
18 }
```

※ 동기화 예제 1

```
1  package threadExam.Synchronized;
2
3  public class User1 extends Thread {
4      private Calculator calculator;
5
6      public void setCalculator(Calculator calculator) {
7          this.setName("User1");
8          this.calculator = calculator;
9      }
10
11     public void run() {
12         calculator.setMemory(100);
13     }
14 }
```

※ 동기화 예제 1

```java
package threadExam.Synchronized;

public class User2 extends Thread {
    private Calculator calculator;

    public void setCalculator(Calculator calculator) {
        this.setName("User2");
        this.calculator = calculator;
    }

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

※ 동기화 예제 1

```
1  package threadExam.Synchronized;
2
3  public class MainThreadExample {
4      public static void main(String[] args) {
5          Calculator calculator = new Calculator();
6
7          User1 user1 = new User1();
8          user1.setCalculator(calculator);
9          user1.start();
10
11         User2 user2 = new User2();
12         user2.setCalculator(calculator);
13         user2.start();
14     }
15 }
```

```
<terminated> MainThrea
User1: 100
User2: 50
```