16장
멀티스레딩

박숙영
blue@sookmyung.ac.kr

16장의 목표

1. 스레드를 사용하여서 독립적으로 실행되는 코드를 만들 수 있나요?
2. 스레드의 우선순위를 조정하거나 종료시킬 수 있나요?
3. 여러 개의 스레드가 함께 실행될 때, 발생하는 문제를 피할 수 있나요?
4. 여러 개의 스레드를 사용하여 작업을 빠르게 수행할 수 있나요?

멀티태스킹

- 멀티 태스킹(muli-tasking)는 여러 개의 애플리케이션을 동시에 실행하여서 컴퓨터 시스템의 성능을 높이기 위한 기법이다.

그림 16-1 • 병렬 처리의 예

스레드란?

- 다중 스레딩(multi-threading)은 하나의 프로그램이 동시에 여러 가지 작업을 할 수 있도록 하는 것
- 각각의 작업은 스레드(thread)라고 불린다.

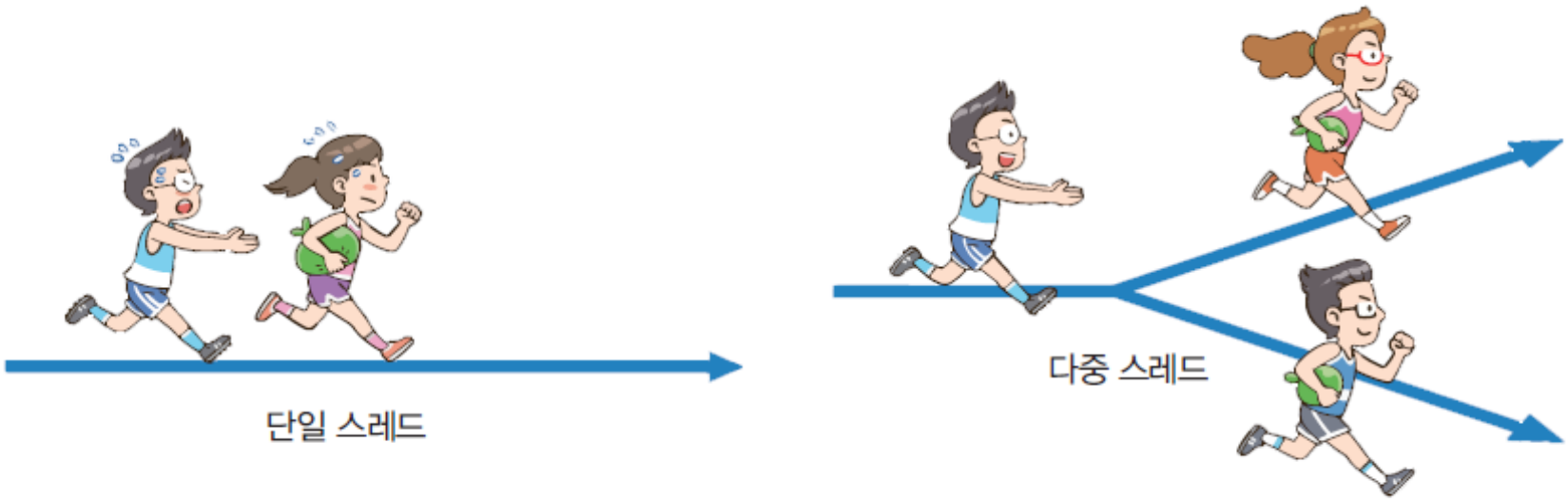


그림 16-2 • 다중 스레드의 개념

프로세스와 스레드

- 프로세스(process): 자신만의 데이터를 가진다.
- 스레드(thread): 동일한 데이터를 공유한다.

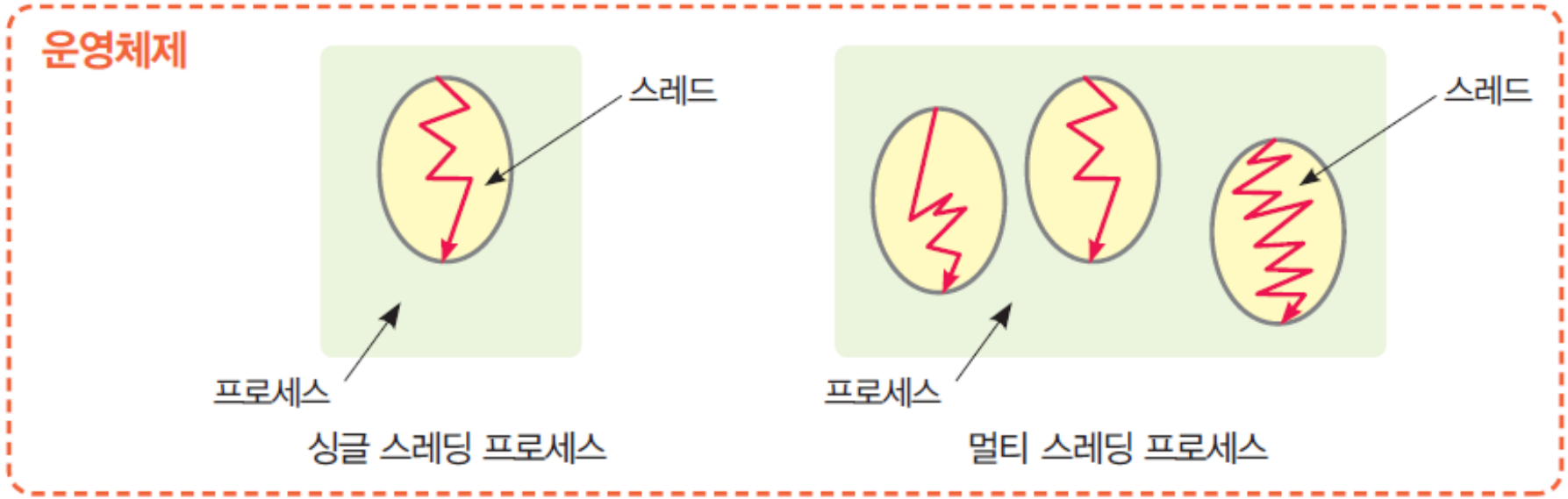


그림 16.2 스레드는 하나의 프로세스 안에 존재한다.

스레드를 사용하는 이유

- 프로그램을 보다 빠르게 실행하기 위하여 멀티 스레딩을 사용한다.
- 최근의 CPU는 속도가 매우 빠르며 여러 개의 코어가 포함되어 있기 때문에, 하나의 스레드로는 모든 코어를 이용할 수 없다.
- 멀티스레딩을 사용하면 여러 코어를 최대한 활용할 수 있다.

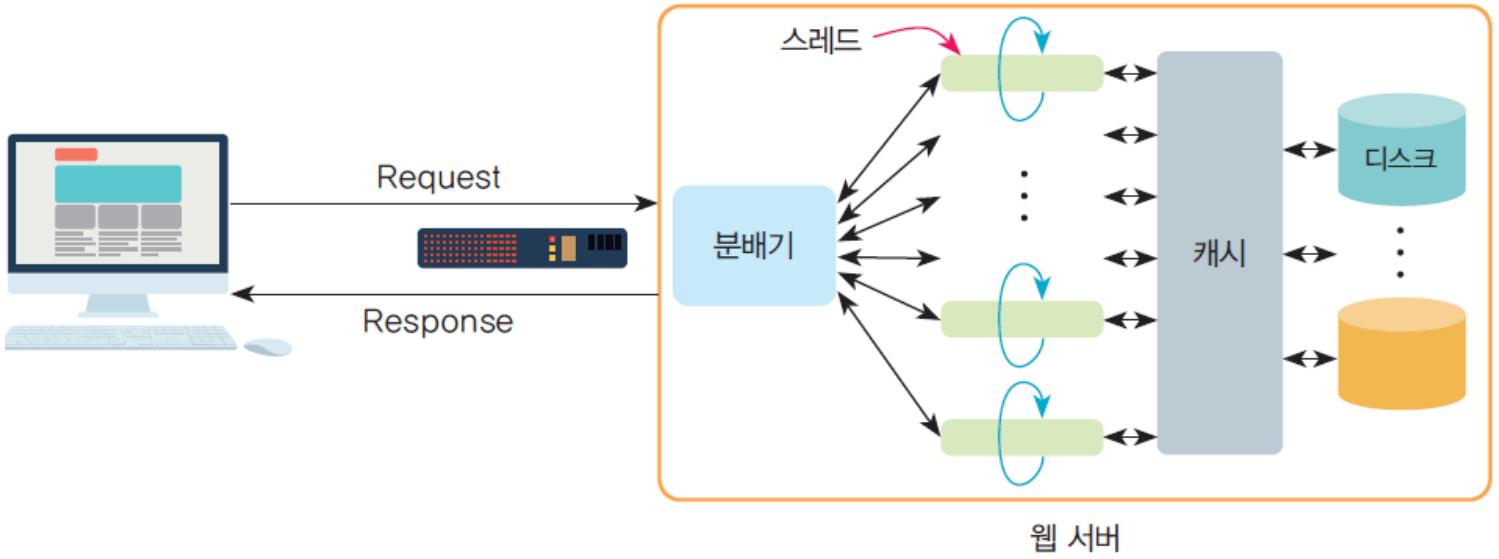


그림 16.3 웹 서버에서의 스레드 이용

멀티스레딩의 문제점

- 여러 스레드들이 같은 데이터를 공유하게 되면 동기화라고 하는 까다로운 문제가 발생하기 때문이다.

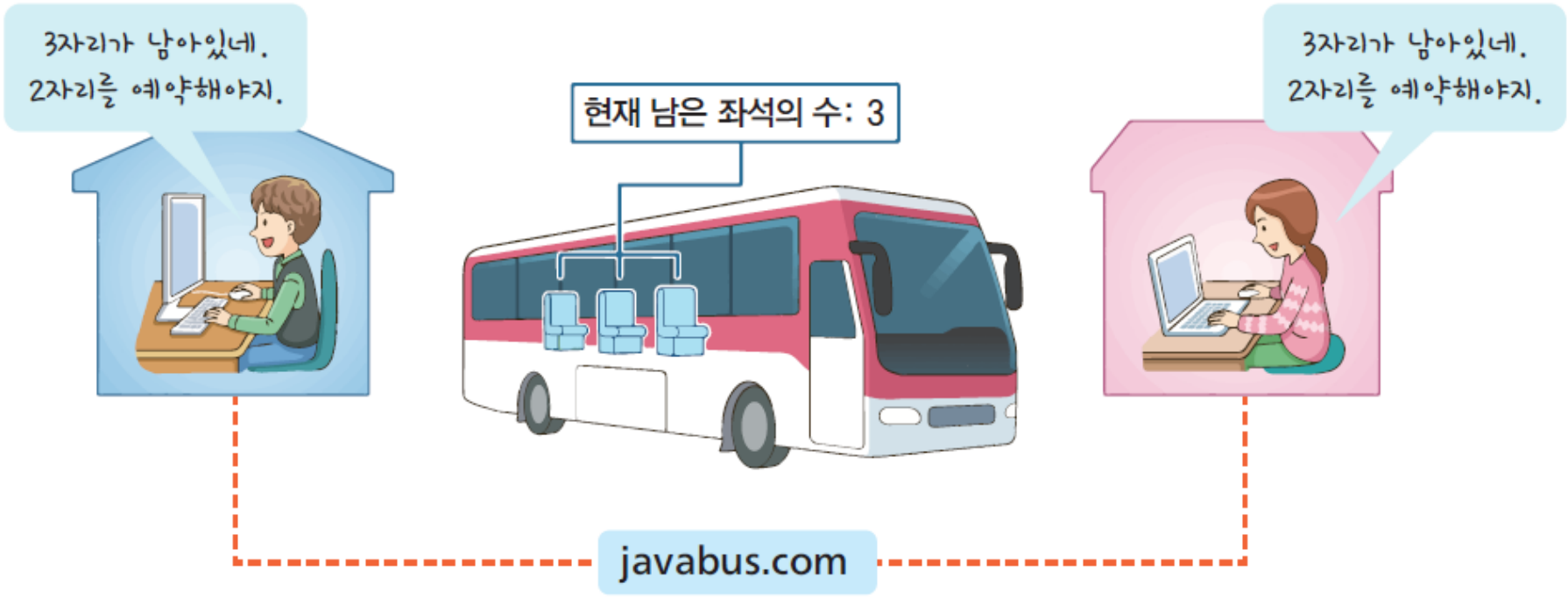


그림 16.4 동기화 문제

스레드 생성과 실행

- 스레드는 Thread 클래스가 담당한다.

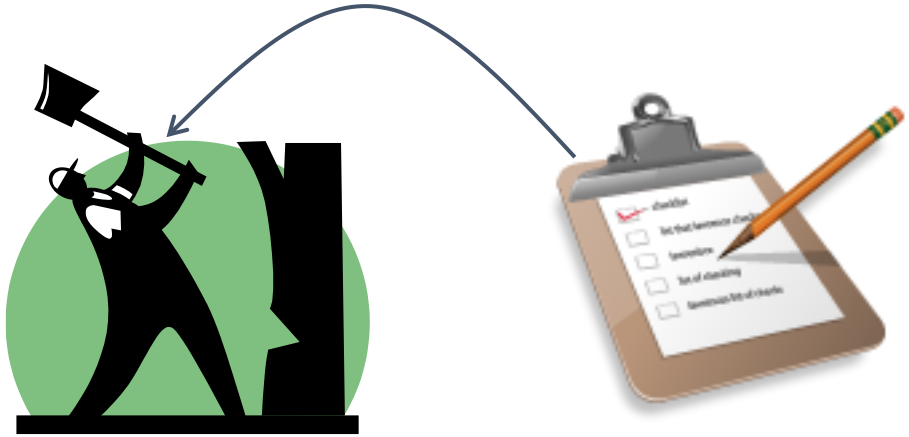
```
Thread t = new Thread(); // 스레드 객체를 생성한다.
t.start();               // 스레드를 시작한다.
```

Thread 클래스

메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String name)	이름이 name인 Thread 객체를 생성한다
static int activeCount()	현재 활동 중인 스레드의 개수를 반환한다.
String getName()	스레드의 이름을 반환한다.
intgtPriority()	스레드의 우선 순위를 반환한다.
void interrupt()	현재의 스레드를 중단한다.
boolean isInterrupted()	현재의 스레드가 중단될 수 있는지를 검사한다.
void setPriority(int priority)	스레드의 우선 순위를 지정한다.
void setName(String name)	스레드의 이름을 지정한다.
static void sleep(int milliseconds)	현재의 스레드를 지정된 시간만큼 재운다.
void run()	스레드가 해야 하는 작업을 이 메소드 안에 위치시킨다. 스레드가 시작될 때 호출된다.
void start()	스레드를 시작한다.
static void yield()	현재 스레드를 다른 스레드에 양보하게 만든다.

스레드를 생성하는 방법

- Thread 클래스를 상속하는 방법
  - Thread 클래스를 상속받은 후에 run() 메소드를 재정의한다.
  - run() 메소드 안에 작업을 기술한다.
  - Thread 객체를 생성하고
  - start()를 호출하여서 스레드를 시작한다.



Thread 객체 = 일꾼

Runnable 객체 = 작업의 내용

- Runnable 인터페이스를 구현하는 방법
  - Runnable 인터페이스를 구현한 클래스를 작성한다.
  - run() 메소드를 작성한다.
  - Thread 객체를 생성하고 이때 Runnable 객체를 인수로 전달한다.
  - start()를 호출하여서 스레드를 시작한다.

Thread 클래스를 상속하는 방법

```
class MyThread extends Thread {                                    // ①
        public void run() {                                                  // ②
                for (int i = 0; i <=10; i++)
                        System.out.print(i + " ");
        }
}

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

                Thread t = new MyThread();        // ③
                t.start();                                                   // ④
        }
}
```

```
0 1 2 3 4 5 6 7 8 9 10
```

Runnable 인터페이스를 구현하는 방법

```
class MyRunnable implements Runnable {                    // ①
        public void run() {                                                    //②
                for (int i = 0; i <=10; i++)
                        System.out.print(i + " ");
        }
}

public class MyRunnableTest {
        public static void main(String args[]) {
                Thread t = new Thread(new MyRunnable());          //③
                t.start();                                        //        ④
        }
}
```

```
0 1 2 3 4 5 6 7 8 9 10
```

예제: 스레드 2개 만들어보기

- 0부터 10까지 세는 스레드를 두 개 만들어보자. 2개의 스레드가 실행되면서 스레드의 출력이 섞이는 것을 알 수 있다.

```
A0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9
B10 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
```

```
class MyRunnable implements Runnable {
        String myName;
        public MyRunnable(String name) {
                myName = name;
        }
        public void run() {
                for (int i = 0; i <=10; i++)
                        System.out.print(myName + i + " ");
        }
}

public class TestThread {
        public static void main(String[] args) {
                Thread t1 = new Thread(new MyRunnable("A"));
                Thread t2 = new Thread(new MyRunnable("B"));
                t1.start();
                t2.start();
        }
}
```

람다식을 이용한 스레드 작성

- Java 8 버전부터 추가된 람다식은 스레드 프로그래밍에도 많이 활용된다.

```
public class LambdaTest {

        public static void main(String args[]) {

                Runnable task = () -> {
                        for (int i = 0; i <=10; i++)
                                System.out.print(i + " ");
                };

                new Thread(task).start();
        }
}
```

```
0 1 2 3 4 5 6 7 8 9 10
```

예제: 그래픽 버전 카운터 만들어보기

- 스윙 컴포넌트를 사용하여서 0부터 10까지 1초 단위로 카운트다운하는 애플리케이션을 그래픽 모드로 작성하여 보자.

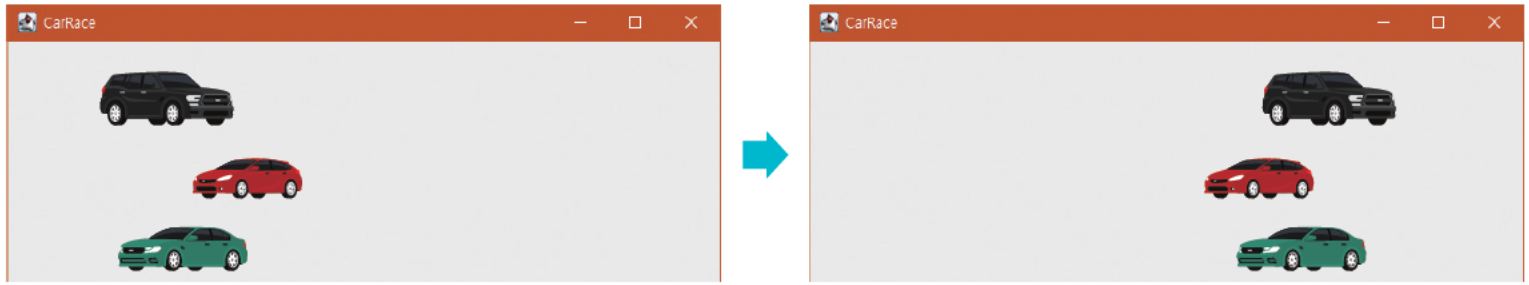

```
public class CountDownTest extends JFrame {

        private JLabel label;

        class MyThread extends Thread {

                public void run() {
                        for (int i = 0; i <=10; i++) {
                                try {
                                        Thread.sleep(1000);
                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }
                                label.setText(i + "");
                        }
                }
        }
```

예제: 그래픽 버전 카운터 만들어보기

```
        public CountDownTest() {
                setTitle("카운트다운");
                setSize(400, 150);
                label = new JLabel("0");
                label.setFont(new Font("Serif", Font.BOLD, 100));
                add(label);
                setVisible(true);
                (new MyThread()).start();
        }

        public static void main(String[] args) {
                CountDownTest t = new CountDownTest();
        }
}
```

Lab: 자동차 경주 게임 만들기

- 3대의 자동차는 이미지를 나타내는 3개의 레이블로 구현된다. 스레드를 사용하여서 0.1초에 한 번씩 난수의 값만큼 자동차의 위치를 변경해보자.



Sol: 자동차 경주 게임 만들기

```
public class CarGame extends JFrame {
        class MyThread extends Thread {
                private JLabel label;
                private int x, y;
                public MyThread(String fname, int x, int y) {
                        this.x = x;
                        this.y = y;
                        label = new JLabel();
                        label.setIcon(new ImageIcon(fname));
                        label.setBounds(x, y, 100, 100);
                        add(label);
                }
                public void run() {
                        for (int i = 0; i < 200; i++) {
                                x += 10 * Math.random();
                                label.setBounds(x, y, 100, 100);
                                repaint();
                                try {
                                        Thread.sleep(100);
                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }
```

Sol: 자동차 경주 게임 만들기

```
        public CarGame() {
                setTitle("CarRace");
                setSize(600, 200);

                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setLayout(null);

                (new MyThread("car1.gif", 100, 0)).start();
                (new MyThread("car2.gif", 100, 50)).start();
                (new MyThread("car3.gif", 100, 100)).start();
                setVisible(true);
        }

        public static void main(String[] args) {
                CarGame t = new CarGame();
        }
}
```

스레드 상태

- New 상태 - Thread 클래스의 인스턴스는 생성되었지만 start() 메소드를 호출하기 전이라면 스레드는 New 상태에 있다.
- Runnable 상태 - start() 메소드가 호출되면 스레드는 실행 가능한 상태가 된다. 하지만 아직 스레드 스케줄러가 선택하지 않았으므로 실행 상태는 아니다.
- Running 상태 - 스레드 스케줄러가 스레드를 선택하면, 스레드는 실행 중인 상태가 된다.
- Blocking 상태 - 스레드가 아직 살아 있지만, 여러 가지 이유로 현재 실행할 수 없는 상태이다.
- Terminated 상태 - 스레드가 종료된 상태이다. 스레드의 run() 메소드가 종료되면 스레드도 종료된다.

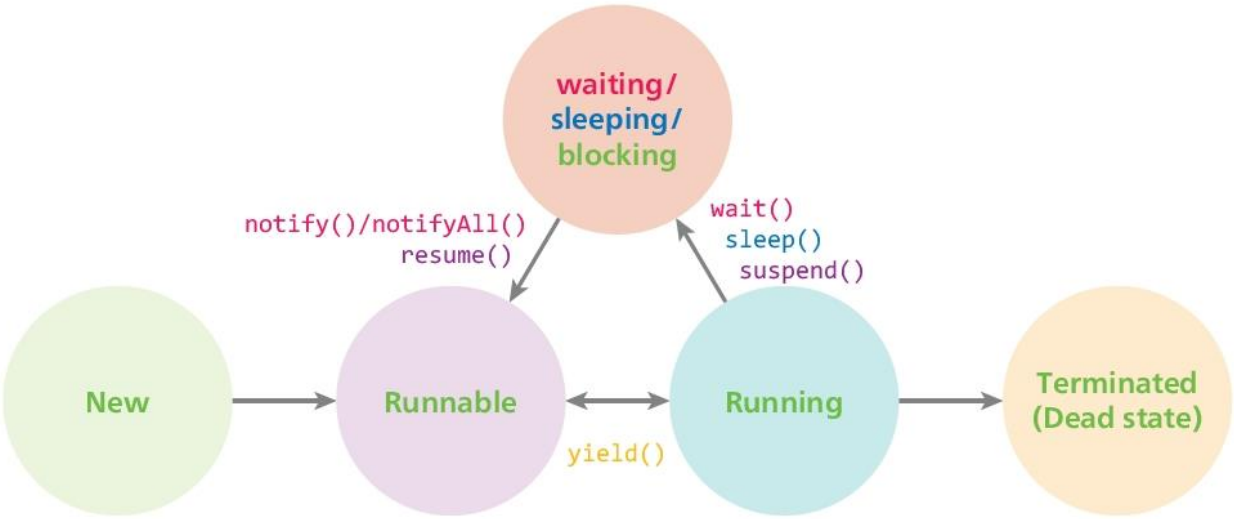


그림 16.5 스레드의 상태

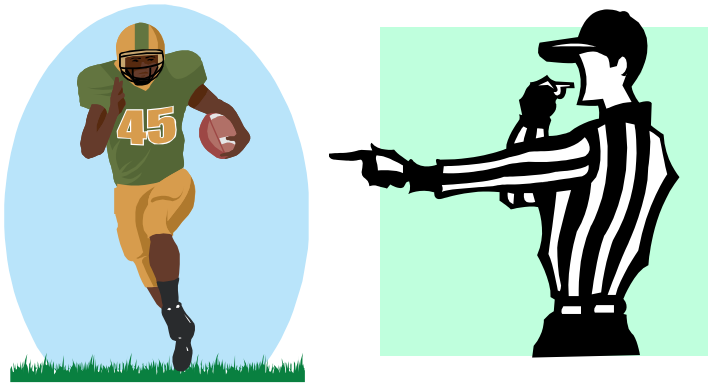
생성 상태와 실행 가능 상태

- 생성 상태
  - Thread 클래스를 이용하여 새로운 스레드를 생성
  - start()는 생성된 스레드를 시작
  - stop()은 생성된 스레드를 멈추게 한다.

- 실행 가능 상태
  - 스레드가 스케줄링 큐에 넣어지고 스케줄러에 의해 우선순위에 따라 실행

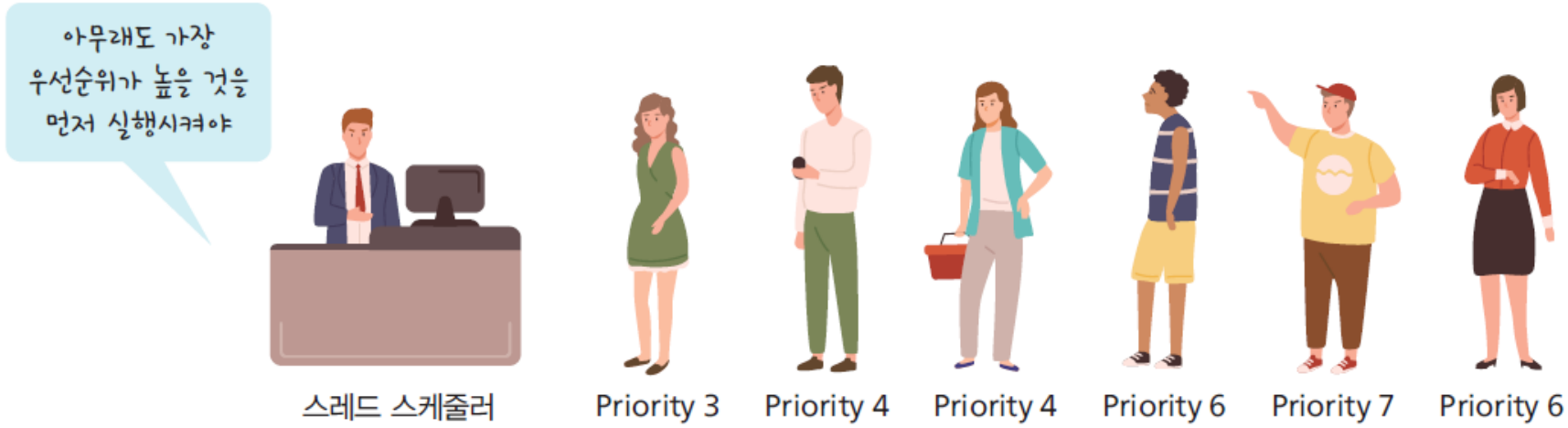
실행 중지 상태

- 실행 가능한 상태에서 다음의 이벤트가 발생하면 실행 중지 상태로 된다.
  - 스레드나 다른 스레드가 suspend()를 호출하는 경우
  - 스레드가 wait()를 호출하는 경우
  - 스레드가 sleep()을 호출하는 경우
  - 스레드가 입출력 작업을 하기 위해 대기하는 경우



스레드 스케줄링

- 대부분의 경우 스레드 스케줄러는 선점형 스케줄링과 타임 슬라이싱을 사용하여 스레드들을 스케줄링한다. 그러나 어떤 스케줄링을 선택하느냐는 자바 가상 머신에 의하여 결정된다.



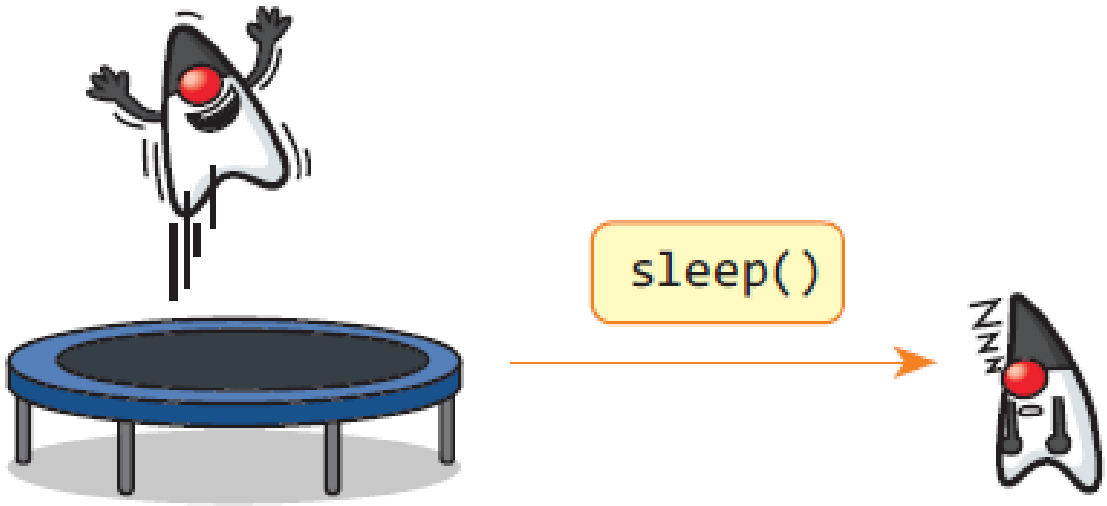
스레드 우선순위

- 우선순위는 1에서 10 사이의 숫자로 표시된다. 스레드의 기본 우선 순위는 NORM_PRIORITY이다. MIN_PRIORITY의 값은 1이고 MAX_PRIORITY의 값은 10이다.
  - 정적 정수 MIN_PRIORITY(1)
  - 정적 정수 NORM_PRIORITY(5)
  - 정적 정수 MAX_PRIORITY(10)

- void setPriority(int newPriority): 현재 스레드의 우선 순위를 변경한다.
- getPriority(): 현재 스레드의 우선 순위를 반환한다.

sleep()

- Thread 클래스의 sleep() 메소드는 지정된 시간 동안 스레드를 재우기 위하여 사용된다. 스레드가 수면 상태로 있는 동안, 인터럽트되면 InterruptedException이 발생한다



예제: 4초 간격으로 메시지 출력

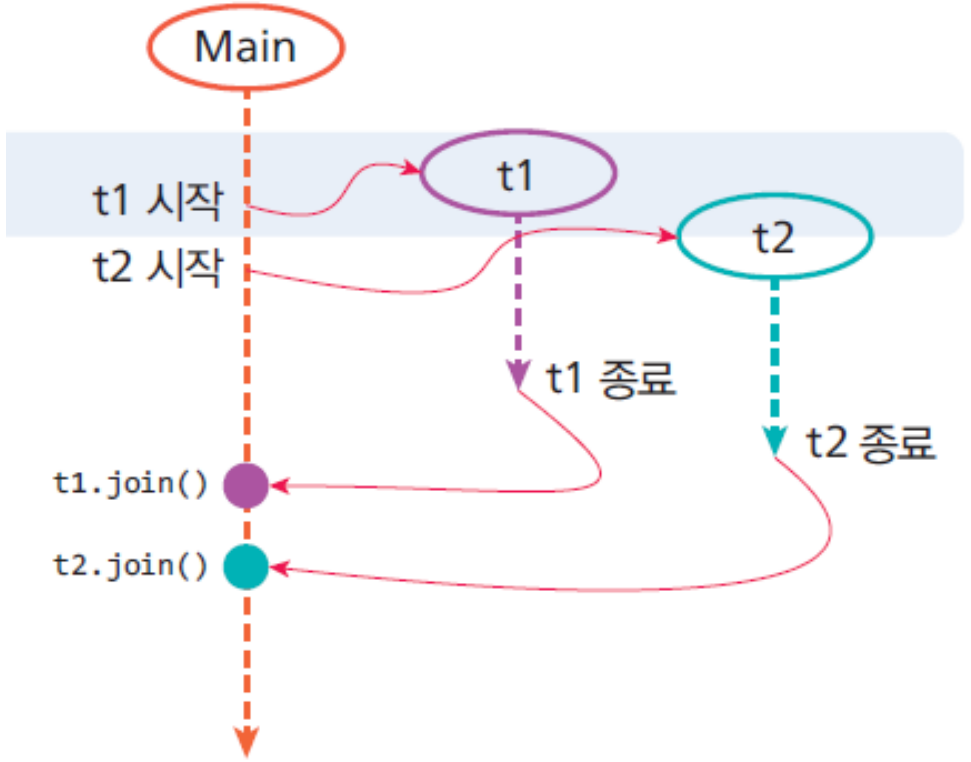
- sleep()을 이용하여 4초 간격으로 메시지를 출력하는 예제를 작성하여 보자.

```
public class SleepTest {
        public static void main(String args[]) throws InterruptedException {
                String messages[] = { "Pride will have a fall.",
                "Power is dangerous unless you have humility.",
                "Office changes manners.",
                "Empty vessels make the most sound." };

                for (int i = 0; i < messages.length; i++) {
                        Thread.sleep(4000);
                        System.out.println(messages[i]);
                }
        }
}
```

```
Pride will have a fall.
Power is dangerous unless you have humility.
Office changes manners.
Empty vessels make the most sound.
```

join()

- join() 메소드는 스레드가 종료될 때까지 기다리는 메소드이다. 즉, 특정 스레드가 작업을 완료할 때까지 현재 스레드의 실행을 중지하고 기다리는 것이다.



예제:

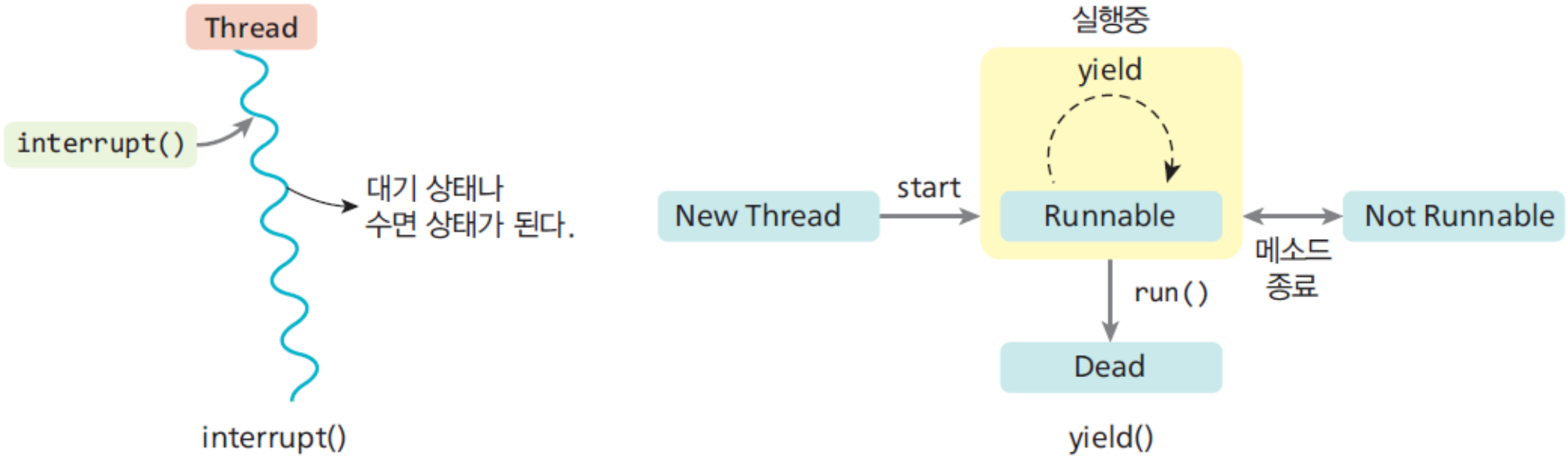
- 2개의 스레드 t1과 t2를 만들어서 t1이 종료되기를 기다렸다가 t2를 시작해보자.

```
public class JoinTest extends Thread {
        public void run() {
                for (int i = 1; i <= 3; i++) {
                        System.out.println(getName() + " "+i);
                }
        }
        public static void main(String args[]) {
                JoinTest t1 = new JoinTest();
                JoinTest t2 = new JoinTest();
                t1.start();
                try {
                        t1.join();
                } catch (Exception e) {
                        System.out.println(e);
                }
                t2.start();
        }
}
```

```
Thread-0 1
Thread-0 2
Thread-0 3
Thread-1 1
Thread-1 2
Thread-1 3
```

인터럽트(interrupt)와 yield()

- 인터럽트(interrupt)는 하나의 스레드가 실행하고 있는 작업을 중지하도록 하는 메커니즘이다.
- yield()는 CPU를 다른 스레드에게 양보하는 메소드이다.



예제: 스레드 스케줄링

- 아래는 앞에서 등장하였던 그래픽 카운터 프로그램이다. 옆의 버튼을 누르면 0부터 10까지 세는 카운터가 중지되도록 한다.

예제: 스레드 스케줄링

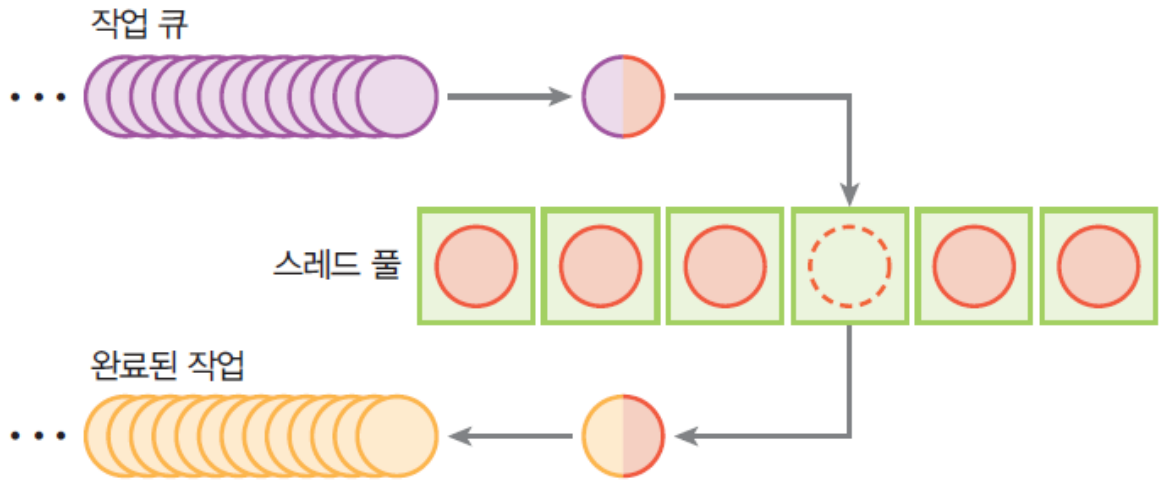
```
public class CountDownTest extends JFrame {
        private JLabel label;

        Thread t;

        class Counter extends Thread {
                public void run() {
                        for (int i = 0; i <= 10; i++) {
                                try {
                                        Thread.sleep(1000);
                                } catch (InterruptedException e) {
                                        return;
                                }
                                label.setText(i + "");
                        }
                }
        }
```

예제: 스레드 스케줄링

```
        public CountDownTest() {
                setTitle("카운트다운");
                setSize(400, 150);
                getContentPane().setLayout(null);
                label = new JLabel("0");
                label.setBounds(0, 0, 384, 111);
                label.setFont(new Font("Serif", Font.BOLD, 100));
                getContentPane().add(label);

                JButton btnNewButton = new JButton("카운터 중지");
                btnNewButton.setBounds(247, 25, 125, 23);

                btnNewButton.addActionListener(e -> t.interrupt());
                getContentPane().add(btnNewButton);
                setVisible(true);
                t = new Counter();
                t.start();
        }
        public static void main(String[] args) {
                CountDownTest t = new CountDownTest();
        }
}
```

자바 스레드 풀

- 스레드 풀(thread pool)은 미리 초기화된 스레드들이 모여 있는 곳이다. 일반적으로 스레드가 저장된 컬렉션의 크기는 고정되어 있지만 반드시 그런 것은 아니다. 스레드 풀의 동일한 스레드를 사용하여 N개의 작업을 쉽게 실행할 수 있다. 스레드의 개수보다 작업의 개수가 더 많은 경우 작업은 FIFO 큐에서 기다려야 한다.



자바 스레드 풀

- Java 5부터 자바 API는 Executor 프레임워크를 제공한다. 스레드 풀을 사용하면, 개발자는 Runnable 객체를 구현하고 ThreadPoolExecutor로 보내기만 하면 된다

```
class MyTask implements Runnable {
        private String name;

        public MyTask(String name) {
                this.name = name;
        }
        public String getName() {               return name;       }
        public void setName(String name) {  this.name = name;}
        @Override
        public void run() {
                try {
                        System.out.println("실행중 : "+name);
                        Thread.sleep((long)(Math.random() * 1000));
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

자바 스레드 풀

```
public class ThreadPoolTest
{
    public static void main(String[] args)
    {
        ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newFixedThreadPool(2);

        for (int i = 1; i <= 5; i++)
        {
            MyTask task = new MyTask("작업 " + i);
            System.out.println("작업 생성 : " + task.getName());
            executor.execute(task);
        }
        executor.shutdown();
    }
}
```

```
작업 생성 : 작업 1
작업 생성 : 작업 2
작업 생성 : 작업 3
실행중 : 작업 1
작업 생성 : 작업 4
실행중 : 작업 2
작업 생성 : 작업 5
실행중 : 작업 3
실행중 : 작업 4
실행중 : 작업 5
```

스레드 사용시 주의해야 할 점

- 스레드들은 동일한 데이터를 공유하기 때문에 매우 효율적으로 작업할 수 있다. 하지만 동일한 메모리를 사용하기 때문에 2가지의 문제가 발생할 수 있다.
  - 스레드 간섭(thread interference)
  - 메모리 불일치 문제(consistency problem)

- 동기화(synchronization)
  - 한 번에 하나의 스레드 만이 공유 데이터를 접근할 수 있도록 제어하는 것이 필요

그림 16.6 공유된 자원에서의 동기화 문제

동기화의 기본 해법

- 밀폐된 방 안에 자원을 놓고 한 번에 하나의 스레드만 방문을 열고 사용할 수 있게 하는 것이다. 하나의 스레드의 작업이 끝나면 다음 스레드가 사용할 수 있도록 한다.

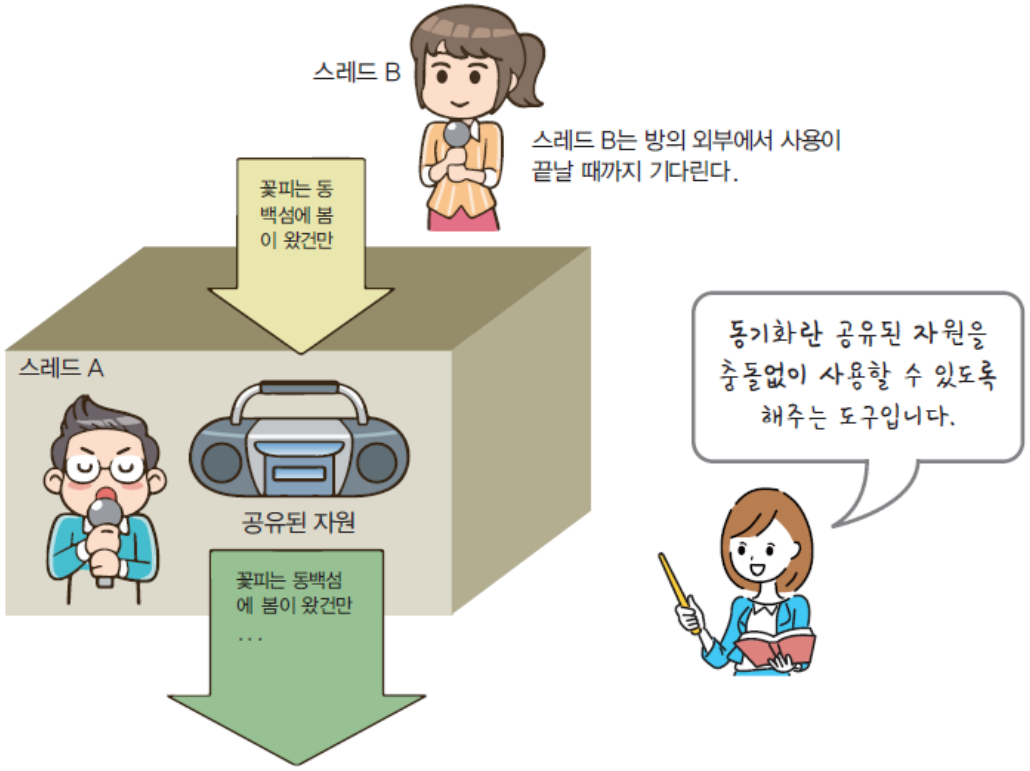


그림 16.7 공유된 자원에서의 동기화 문제 해결 방법

어떤 문제가 발생할 수 있는가?

- 정수 배열을 출력하는 클래스를 다음과 같이 작성하여 사용한다고 가정하자.

```
class Printer {
        void print(int[] arr) {
                for (int i = 0; i < arr.length; i++) {
                        System.out.print(arr[i]+" ");
                        Thread.sleep(100);
                }
        }
}
```

어떤 문제가 발생할 수 있는가?

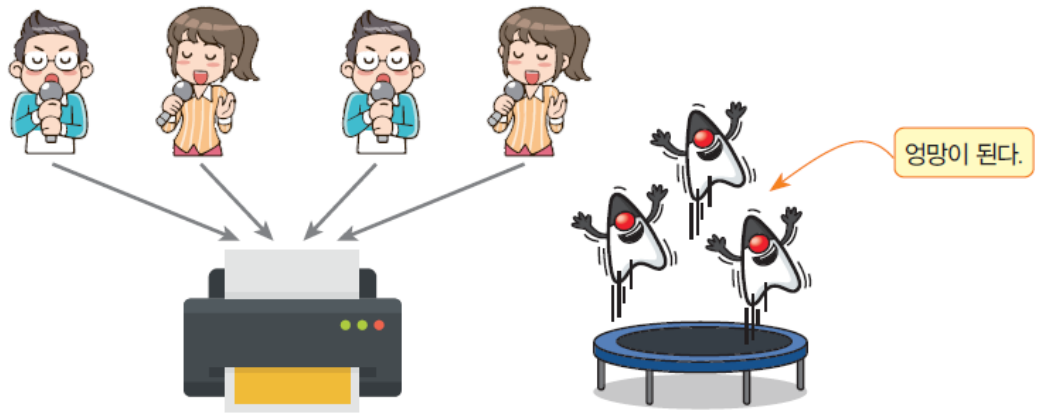
```
class MyThread1 extends Thread {
        Printer prn;
        int[] myarr = { 10, 20, 30, 40, 50 };

        MyThread1(Printer prn) {        this.prn = prn;     }
        public void run() {             prn.print(myarr);   }
}

class MyThread2 extends Thread {
        Printer prn;
        int[] myarr = { 1, 2, 3, 4, 5 };

        MyThread2(Printer prn) {        this.prn = prn;     }
        public void run() {             prn.print(myarr);   }
}
```

어떤 문제가 발생할 수 있는가?

```
public class TestSynchro {
        public static void main(String args[]) {
                Printer obj = new Printer();
                MyThread1 t1 = new MyThread1(obj);
                MyThread2 t2 = new MyThread2(obj);
                t1.start();
                t2.start();
        }
}
```

```
1 10 20 2 30 3 4 40 5 50
```

```
10 1 2 20 3 30 4 40 5 50
```

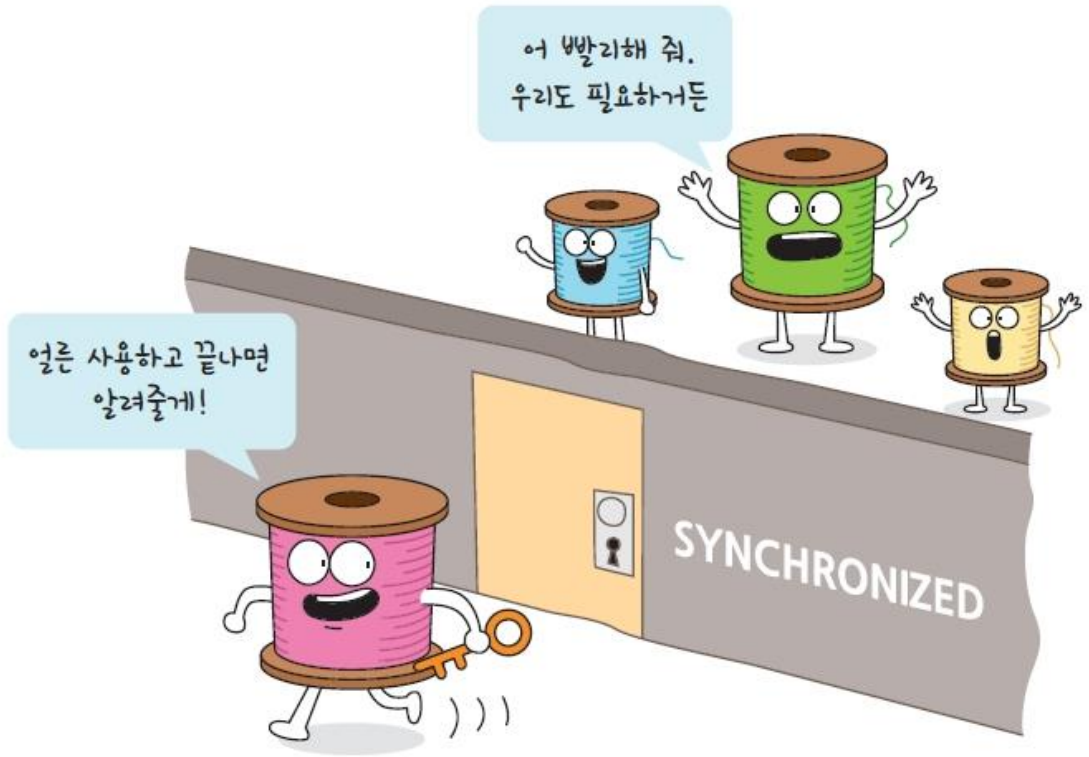


자바에서의 동기화 방법

- 자바에서는 다음과 같은 3가지의 방법을 제공하고 있다.
  - 동기화 메소드(synchronized method)
  - 동기화 블록(synchronized block)
  - 정적 동기화(static synchronization)

동기화 방법

- 동기화는 락(lock) 또는 모니터(monitor)로 알려진 방법을 사용하여 구축된다.



동기화 예제

- 동기화 메소드를 사용하여 이전 예제를 수정해보자. 다음과 같이 synchronized 키워드를 메소드 앞에 붙여주면 된다.

```
class Printer {
        synchronized void print(int[] arr) {
                ...
        }
}
```

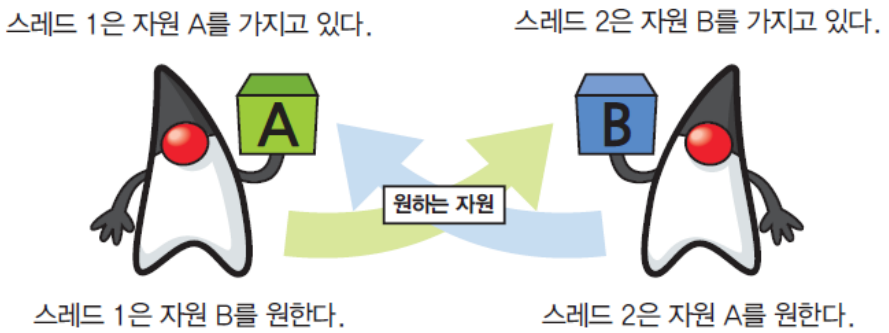
```
10 20 30 40 50 1 2 3 4 5
```

```
class Printer {
        void print(int[] arr) throws Exception {
                synchronized(this){
                    for (int i = 0; i < arr.length; i++) {
                            System.out.print(arr[i]+" ");
                            Thread.sleep(100);
                    }
                }
        }
}
```

메소드 앞에 synchronized 키워드를 붙이기도 하지만, 메소드 전체 코드가 아닌 부분 코드만 동기화 할 경우 synchronized 블록으로 설정하면 됩니다.

교착 상태, 기아 상태

- 동일한 자원을 접근하려고, 동기화를 기다리면서 대기하는 스레드들이 많아지면 자바 가상 머신이 느려지거나 일시 중단되기도 한다. 이것을 교착 상태(deadlock), 기아(starvation)과 라이브락(livelock)이라고 한다.



예제:

```
public class DeadLockTest {
  public static void main(String[] args) {
    final String res1 = "Gold";
    final String res2 = "Silver";

    Thread t1 = new Thread(()-> {
        synchronized (res1) {
         System.out.println("Thread 1: 자원 1 획득");
         try { Thread.sleep(100);} catch (Exception e) {}
         synchronized (res2) {
          System.out.println("Thread 1: 자원 2 획득");
         }
    });
```

```
    Thread t2 = new Thread(()-> {
         synchronized (res2) {
          System.out.println("Thread 2: 자원 2 획득");
          try { Thread.sleep(100);} catch (Exception e) {}
          synchronized (res1) {
           System.out.println("Thread 2: 자원 1 획득");
          }
    });

    t1.start();
    t2.start();
  }
}
```

```
Thread 1: 자원 1 획득
Thread 2: 자원 2 획득
```

교착상태의 해결 방법 하나

- 간단하게 잠금 순서를 변경하는 기법만 살펴보자

```
Thread t2 = new Thread(()-> {
        synchronized (res1) {
            System.out.println("Thread 2: 자원 1 획득");
            try { Thread.sleep(100);} catch (Exception e) {}
            synchronized (res2) {
                System.out.println("Thread 2: 자원 2 획득");
            }
        }
});
```

```
Thread 1: 자원 1 획득
Thread 1: 자원 2 획득
Thread 2: 자원 1 획득
Thread 2: 자원 2 획득
```

스레드간의 조정

- 만약 두 개의 스레드가 데이터를 주고 받는 경우에 발생

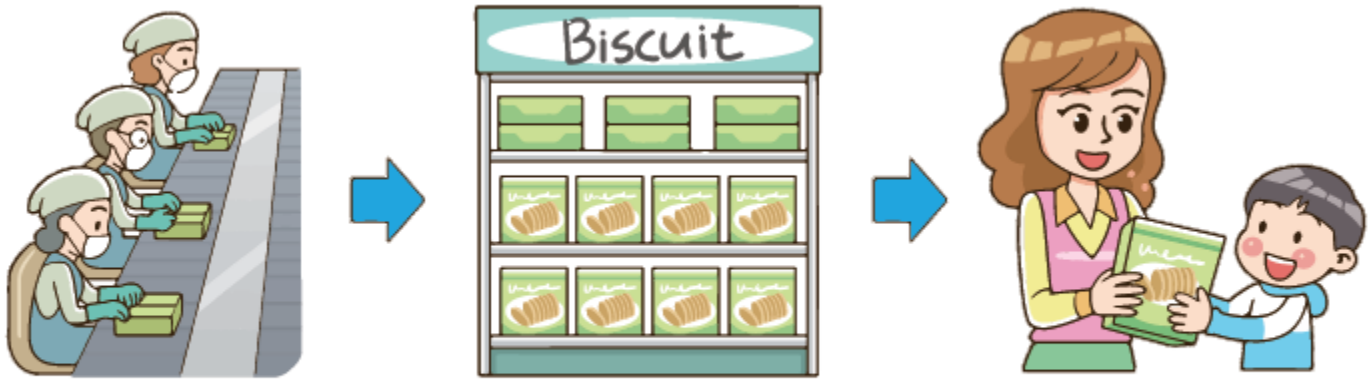


그림 16-5 • 생산자와 소비자 문제

2가지의 방법

좋지 못한 방법(polling)

좋은 방법(wait & notify)

wait()와 notify()

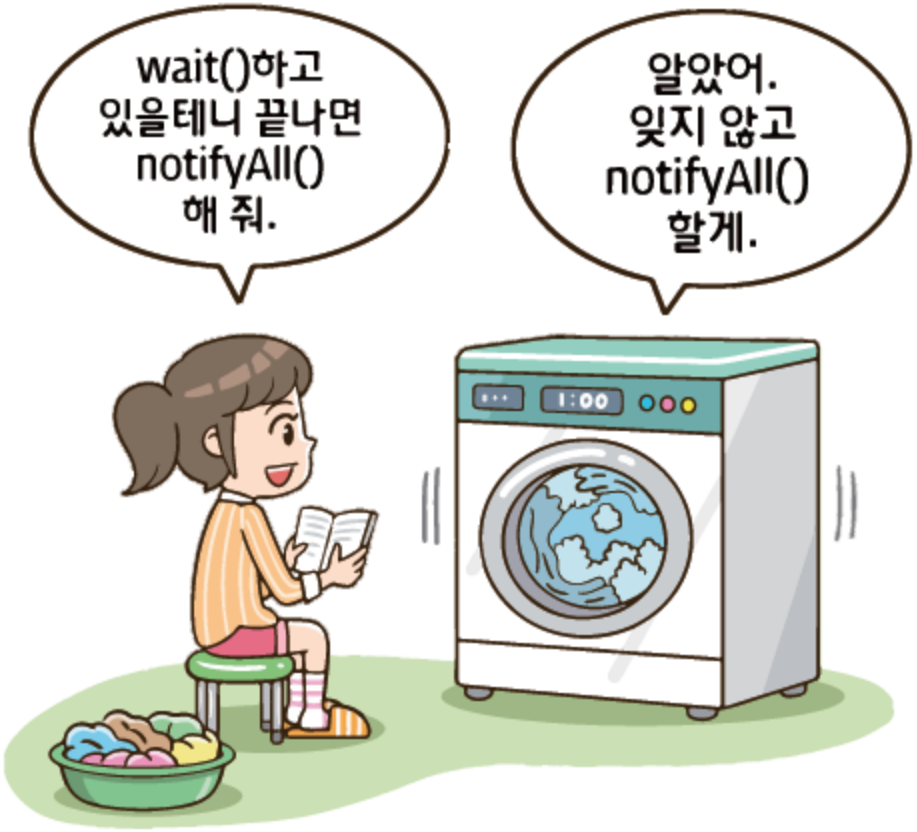
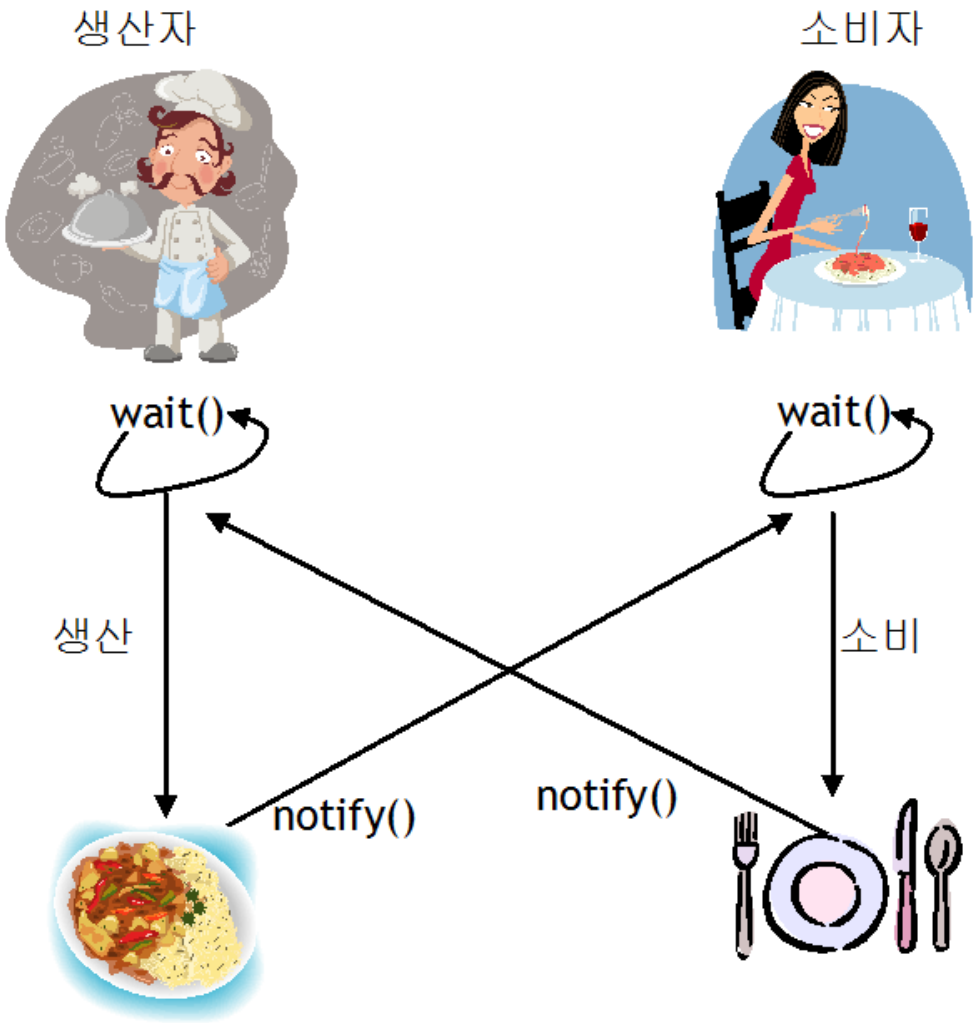


그림 16-6 • wait() notify()

생산자/소비자 문제에 적용



Buffer 클래스

```
class Buffer {
        private int data;
        private boolean empty = true;
        public synchronized int get() {
                while (empty) {
                        try {
                                wait();
                        } catch (InterruptedException e) {
                        }
                }
                empty = true;
                notifyAll();
                return data;
        }
        public synchronized void put(int data) {
                while (!empty) {
                        try {
                                wait();
                        } catch (InterruptedException e) {
                        }
                }
                empty = false;
                this.data = data;
                notifyAll();
        }
}
```

생산자

```
class Producer implements Runnable {
        private Buffer buffer;

        public Producer(Buffer buffer) {
                this.buffer= buffer;
        }
        public void run() {
                for (int i = 0; i < 10; i++) {
                        buffer.put(i);
                        System.out.println("생산자: " + i + "번 케익을 생산하였습니다.");
                        try {
                                Thread.sleep((int) (Math.random() * 100));
                        } catch (InterruptedException e) {
                        }
                }
        }
}
```

소비자

```
class Consumer implements Runnable {
    private Buffer buffer;

    public Consumer(Buffer drop) {
        this.buffer= drop;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            int data = buffer.get();
            System.out.println("소비자: " + data + "번 케익을 소비하였습니다.");

            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) {
            }
        }
    }
}
```

실행 결과

```
public class ProducerConsumerTest {
        public static void main(String[] args) {
                Buffer buffer = new Buffer();

                (new Thread(new Producer(buffer))).start();
                (new Thread(new Consumer(buffer))).start();
        }
}
```

```
생산자: 0번 케익을 생산하였습니다.
소비자: 0번 케익을 소비하였습니다.
생산자: 1번 케익을 생산하였습니다.
소비자: 1번 케익을 소비하였습니다.
...
생산자: 9번 케익을 생산하였습니다.
소비자: 9번 케익을 소비하였습니다.
```

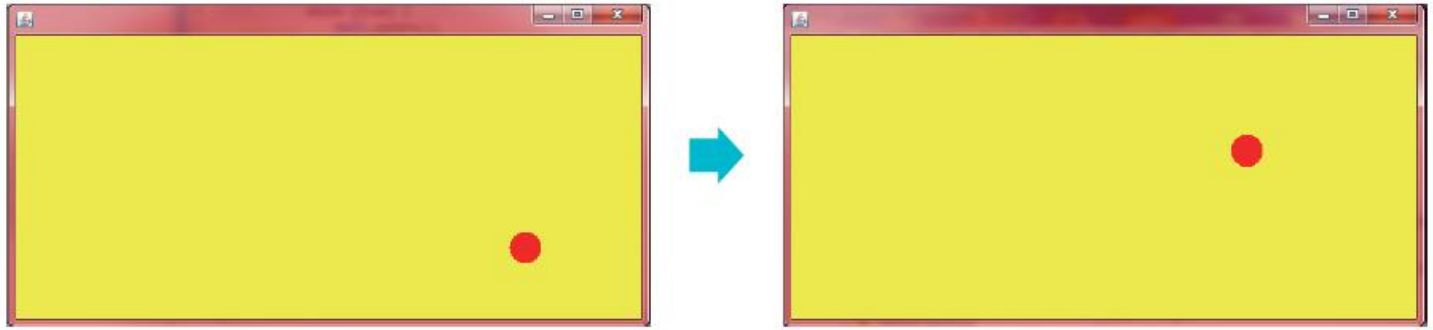
만약 wait()를 사용하지 않는다면

```
class Buffer {
        private int data;
        public synchronized int get() {
                return data;
        }
        public synchronized void put(intdata) {
                this.data = data;
        }
}
```

```
생산자: 0번 케익을 생산하였습니다.
소비자: 0번 케익을 소비하였습니다.
소비자: 0번 케익을 소비하였습니다.
소비자: 0번 케익을 소비하였습니다.
생산자: 1번 케익을 생산하였습니다.
소비자: 1번 케익을 소비하였습니다.
...
```

Lab: 공 움직이기

- 스레드를 이용하여 화면에서 공을 움직이는 프로그램을 작성해보자. 스레드의 작업을 지정할 때 람다식을 이용해보자.



Sol: 공 움직이기

```
class Ball {
        private int x = 100;
        private int y = 100;
        private int size = 30;
        private int xSpeed = 10;
        private int ySpeed = 10;

        public void draw(Graphics g) {
                g.setColor(Color.RED);
                g.fillOval(x, y, size, size);
        }
        public void update() {
                x += xSpeed;
                y += ySpeed;
                if ((x + size) > MyPanel.BOARD_WIDTH - size || x < 0) {
                        xSpeed = -xSpeed;
                }
                if ((y + size) > MyPanel.BOARD_HEIGHT - size || y < 0) {
                        ySpeed = -ySpeed;
                }
        }
}
```

Sol: 공 움직이기

```
public class MyPanel extends JPanel {
        static final int BOARD_WIDTH = 600;
        static final int BOARD_HEIGHT = 300;
        private Ball ball = new Ball();

        public MyPanel() {
                this.setBackground(Color.YELLOW);
                Runnable task = () -> {
                        while (true) {
                                ball.update();
                                repaint();
                                try {
                                        Thread.sleep(10);
                                } catch (InterruptedException ignore) {
                                }
                        }
                };
                new Thread(task).start();
        }
```

Sol: 공 움직이기

```
        @Override
        protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                ball.draw(g);
        }

        public static void main(String[] args) {
                JFrame frame = new JFrame();
                frame.setSize(MyPanel.BOARD_WIDTH, MyPanel.BOARD_HEIGHT);
                frame.add(new MyPanel());
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
        }
}
```