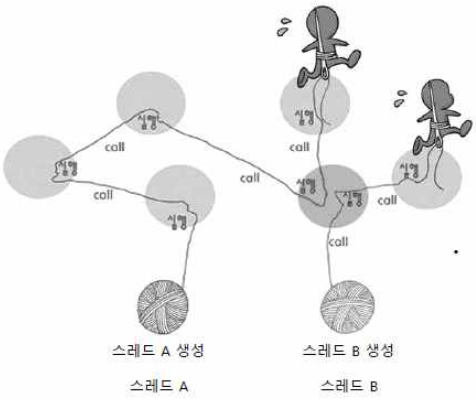
13차시 정리

// 멀티태스킹
// 하나의 응용프로그램이 여러 개의 작업(태스크)을 동시에 처리



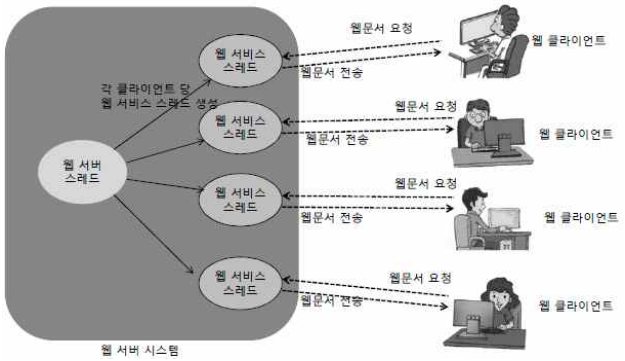
// 멀티태스킹 응용프로그램 사례
// 미디어 플레이어의 멀티테스킹 - 3개의 태스크 동시 실행
// 테트리스 게임의 멀티테스킹 - 3개의 태스크 동시 실행

// 스레드(thread) 개념과 실(thread)
// 마치 바늘이 하나의 실(thread)을 가지고 바느질하는 것과 자바의 스레드는 일맥상 통함

// 스레드와 멀티스레딩
// 스레드
// 사용자가 작성한 코드로서, JVM에 의해 스케줄링되어 실행되는 단위

// 자바의 멀티태스킹
// 멀티스레딩만 가능, 스레드는 JVM의 의한 실행 단위, 스케즐링 단위
// 하나의 응용프로그램은 여러 개의 스레드로 구성 가능

// 멀티스레딩의 효과
// 한 스레드가 대기하는 동안 다른 스레드 실행
// 프로그램 전체적으로 시간 지연을 줄임

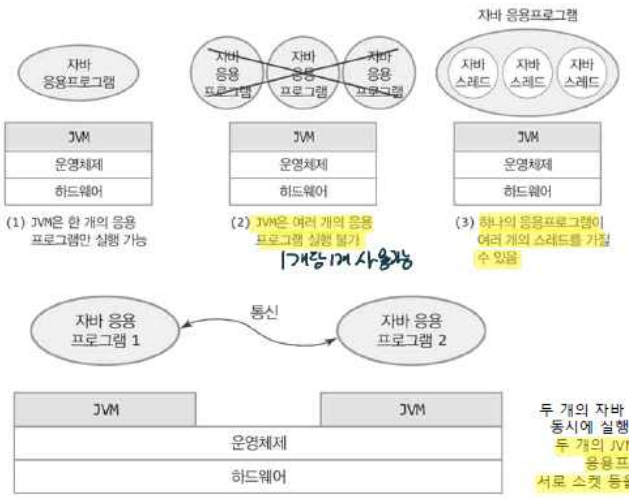


// 웹 서버의 멀티스레딩 사례

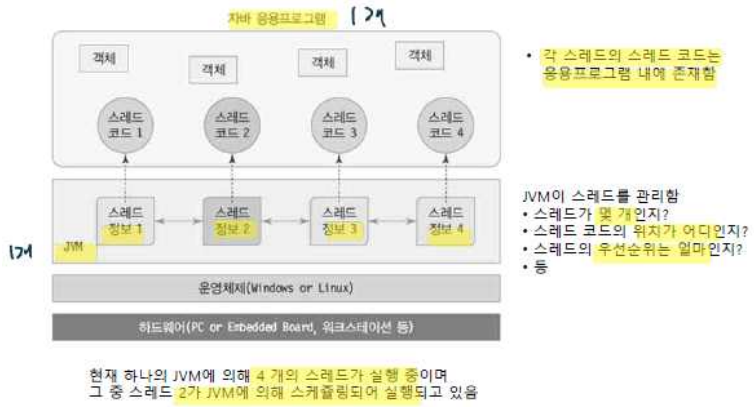
// 자바 스레드(Thread)란?
// 자바 스레드
// 자바 가상 기계(JVM)에 의해 스케쥴되는 실행 단위의 코드 블럭
// 스레드의 생명 주기는 JVM에 의해 관리됨
// JVM은 스레드 단위로 스케쥴링

// JVM과 멀티스레드의 관계
// 하나의 JVM은 하나의 자바 응용프로그램만 실행
// 예) 자바 응용프로그램이 시작될 때 JVM이 함께 실행됨, 자바 응용프로그램이 종료하면 JVM도 함께 종료함
// 하나의 응용프로그램은 하나 이상의 스레드로 구성 가능

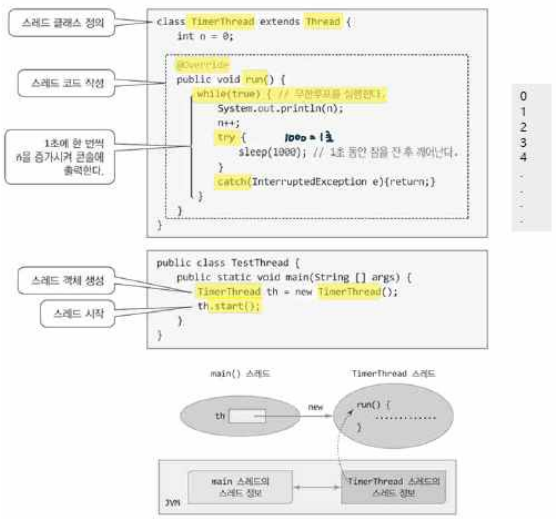
// JVM과 자바 응용프로그램, 스레드의 관계



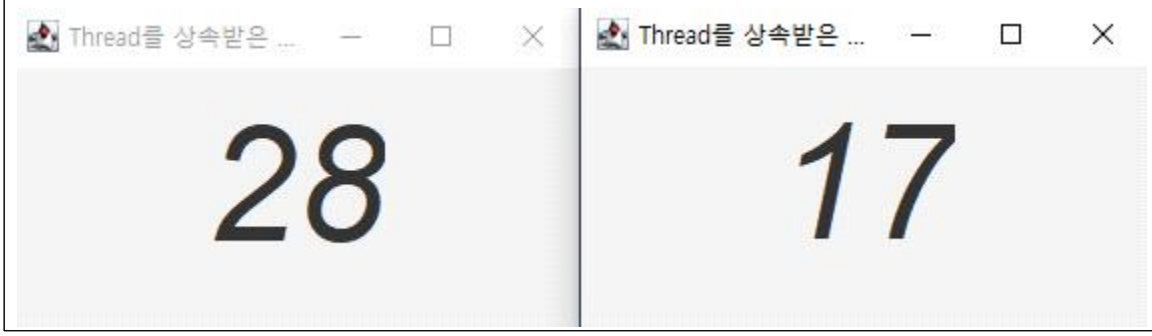
// 자바 스레드와 JVM



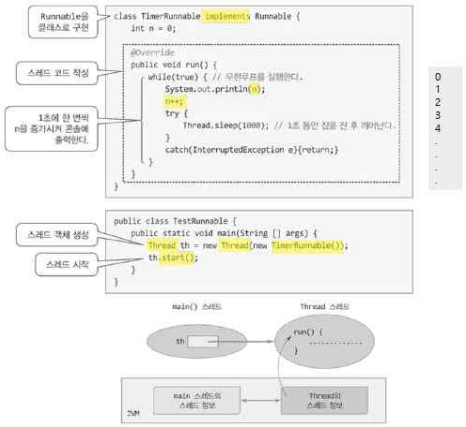
```
// 자바에서 스레드 만들기
// 스레드 실행을 위해 개발자가 하는 작업
// 스레드 코드 작성
// 스레드를 생성하고 스레드 코드를 실행하도록 JVM에게 요청

// 스레드 만드는 2 가지 방법
// 1. java.lang.Thread 클래스를 이용하는 경우
// 2. java.lang.Runnable 인터페이스를 이용하는 경우

// 1. Thread 클래스를 이용한 스레드 생성
// 스레드 클래스 작성, Thread 클래스 상속. 새 클래스 작성
// 스레드 코드 작성
// run() 메소드 오버라이딩
// run() 메소드를 스레드 코드라고 부름
// run() 메소드에서 스레드 실행 시작
//     class TimerThread extends Thread { -> 클래스 작성
//         ...
//         @Override
//         public void run() { // run() 오버라이딩 >- 메소드 오버라이딩
//             ...
//         }
//    }
// 스레드 객체 생성
//     TimerThread th = new TimerThread(); -> 객체 생성
// 스레드 시작
// start() 메소드 호출
// 스레드로 작동 시작, JVM에 의해 스케쥴되기 시작함
//    th.start();

// Thread를 상속받아 1초 단위로 초 시간을 출력하는 TimerThread 스레드 작성
```



```
import java.awt.*;
import javax.swing.*;

class TimerThread extends Thread {
	private JLabel timerLabel;

	public TimerThread(JLabel timerLabel) {
		this.timerLabel = timerLabel;
	}

	@Override
	public void run() {
		int n=0;
		while(true) {
			timerLabel.setText(Integer.toString(n));
			n++;
			try {
				Thread.sleep(1000);
			}
			catch(InterruptedException e) {
				return;
			}
		}
	}
}

public class ThreadTimerEx extends JFrame {
	public ThreadTimerEx() {
		setTitle("Thread를 상속받은 타이머 스레드 예제");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Container c = getContentPane();
		c.setLayout(new FlowLayout());

		JLabel timerLabel = new JLabel();

		timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));

		c.add(timerLabel);

		TimerThread th = new TimerThread(timerLabel);

		setSize(300,170);
		setVisible(true);

		th.start();
	}
	public static void main(String[] args) {
		new ThreadTimerEx();
	}
}
```



```
// 스레드 만들 때 주의 사항
// run() 메소드가 종료하면 스레드는 종료한다. - 스레드가 계속 살아있게 하려면 run() 메소드 내 무한루프 작성
// 한번 종료한 스레드는 다시 시작시킬 수 없다. - 다시 스레드 객체를 생성하고 start()를 호출해야 함
// 한 스레드에서 다른 스레드를 강제 종료할 수 있다. - 뒤에서 다룸

// Runnable 인터페이스로 스레드 만들기
// 1. 스레드 클래스 작성
// Runnable 인터페이스
// 구현하는 새 클래스 작성
// 2. 스레드 코드 작성
// run() 메소드 구현
// run() 메소드를 스레드 코드라고 부름
// run() 메소드에서 스레드 실행 시작
//          class TimerRunnable implements Runnable {, 1. 스레드 클래스 작성
//                  ...
//                  @Override
//                  public void run() { // run() 메소드 구현, 2. 스레드 코드 작성
//                          ...
//                  }
//          }
// 3. 스레드 객체 생성
//     Thread th = new Thread(new TimerRunnable());
// 4. 스레드 시작
// start() 메소드 호출
//     th.start();

// Runnable 인터페이스를 상속받아 1초 단위로 초 시간을 출력하는 스레드 작성
```




```
//
```

```
import java.awt.*;
import javax.swing.*;

class TimerRunnable implements Runnable {
	private JLabel timerLabel; public TimerRunnable(JLabel timerLabel) {
		this.timerLabel = timerLabel; }

	@Override
	public void run() {
		int n=0;
		while(true) {
			timerLabel.setText(Integer.toString(n));
			n++;
			try {
				Thread.sleep(1000);
			}
			catch(InterruptedException e) {
				return;
			}
		}
	}
}
public class RunnableTimerEx extends JFrame {
	public RunnableTimerEx() {
		setTitle("Runnable을 구현한 타이머 스레드 예제");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Container c = getContentPane();
		c.setLayout(new FlowLayout());

		JLabel timerLabel = new JLabel();
		timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));

		c.add(timerLabel);

		TimerRunnable runnable = new TimerRunnable(timerLabel);
		Thread th = new Thread(runnable);

		setSize(250,150);
		setVisible(true);

		th.start();
	}
	public static void main(String[] args) {
		new RunnableTimerEx();
	}
}
```

```
// 깜박이는 문자열을 가진 레이블 만들기
import java.awt.*;
import javax.swing.*;
class FlickeringLabel extends JLabel implements Runnable {
	private long delay;

	public FlickeringLabel(String text, long delay) {
		super(text);
		this.delay = delay;

		setOpaque(true);
		Thread th = new Thread(this);
		th.start();
	}
	@Override public void run() {
		int n=0;
		while(true) {
			if(n == 0)
				setBackground(Color.YELLOW);
			else
				setBackground(Color.GREEN);

			if(n == 0)
				n = 1;
			else
				n = 0;

			try {
				Thread.sleep(delay);
			}
			catch(InterruptedException e) {
				return;
			}
		}
	}
}
public class FlickeringLabelEx extends JFrame {
	public FlickeringLabelEx() {
		setTitle("FlickeringLabelEx 예제");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		Container c = getContentPane();
		c.setLayout(new FlowLayout()); // 깜박이는 레이블 생성

		FlickeringLabel fLabel = new FlickeringLabel("깜박",500); // 깜박이지 않는 레이블 생성
		c.add(fLabel);

		JLabel label = new JLabel("안깜박"); // 깜박이는 레이블 생성
		c.add(label);

		FlickeringLabel fLabel2 = new FlickeringLabel("여기도 깜박",300);
		c.add(fLabel2);

		setSize(300,150);
		setVisible(true);
	}
	public static void main(String[] args) {
		new FlickeringLabelEx();
	}
}
```

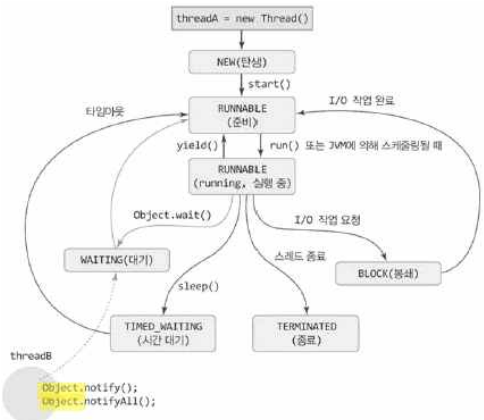
// 스레드 정보

필드	타입	내용
스레드 이름	스트링	스레드의 이름으로서 사용자가 지정
스레드 ID	정수	스레드 고유의 식별자 번호
스레드의 PC(Program Count)	정수	현재 실행 중인 스레드 코드의 주소
스레드 상태	정수	NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCK,TERMINATED 등 6개 상태 중 하나
스레드 우선순위	정수	스레드 스케줄링 시 사용되는 우선순위 값으로서 1~10 사이의 값이며 10이 최상위 우선순위
스레드 그룹	정수	여러 개의 자바 스레드가 하나의 그룹을 형성할 수 있으며 이 경우 스레드가 속한 그룹
스레드 레지스터 스택	메모리 블록	스레드가 실행되는 동안 레지스터들의 값

// 스레드 상태
// 스레드 상태 6 가지
// NEW
// 스레드가 생성되었지만 스레드가 아직 실행할 준비가 되지 않았음
// RUNNABLE
// 스레드가 현재 실행되고 있거나
// 실행 준비되어 스케쥴링을 기다리는 상태
// WAITING
// wait()를 호출한 상태 - 다른 스레드가 notify()나 notifyAll()을 불러주기를 기다리고 있는 상태
// 스레드 동기화를 위해 사용
// TIMED_WAITING
// sleep(n)을 호출하여 n 밀리초 동안 잠을 자고 있는 상태
// BLOCK
// 스레드가 I/O 작업을 요청하면 JVM이 자동으로 BLOCK 상태로 만듦
// TERMINATED
// 스레드가 종료한 상태

// 스레드 상태는 JVM에 의해 기록 관리됨

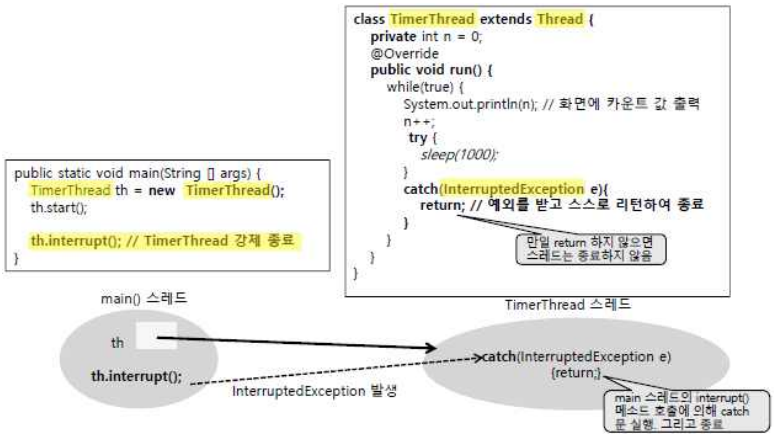
// 스레드 상태와 생명 주기

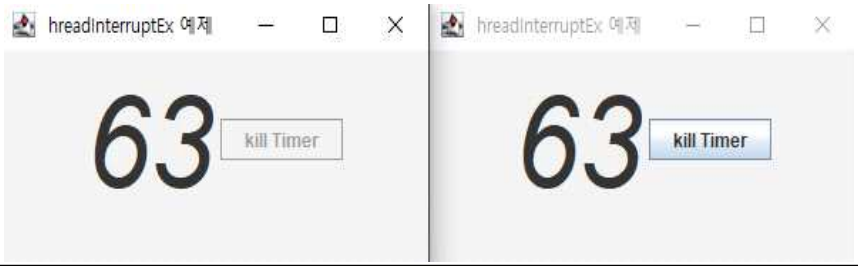


// 주의 : ** wait(), notify(), notifyAll()은 Thread의 메소드가 아니며 Object의 메소드임

```
// 스레드 우선순위와 스케쥴링

// 스레드의 우선순위
//     최대값 = 10(MAX_PRIORITY)
//     최소값 = 1(MIN_PRIORITY)
//     보통값 = 5(NORMAL_PRIORITY)

// 스레드 우선순위는 응용프로그램에서 변경 가능
//     void setPriority(int priority)
//     int getPriority()

// main() 스레드의 우선순위 값은 초기에 5
// 스레드는 부모 스레드와 동일한 우선순위 값을 가지고 탄생
// JVM의 스케쥴링 정책 : 철저한 우선순위 기반
//     가장 높은 우선순위의 스레드가 우선적으로 스케쥴링
//     동일한 우선순위의 스레드는 돌아가면서 스케쥴링(라운드 로빈 )

// main()을 실행하는 main 스레드
// main 스레드와 main() 메소드
// JVM은 응용프로그램을 실행을 시작할 때 스레드 생성 : main 스레드
// JVM은 main 스레드에게 main() 메소드 실행하도록 함 : main() 메소드가 종료하면 main 스레드 종료

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

                String name = Thread.currentThread().getName(); // 스레드 이름 얻기
                long id = Thread.currentThread().getId(); // 스레드 ID 얻기
                int priority = Thread.currentThread().getPriority(); // 스레드 우선순위 값 얻기
                Thread.State s = Thread.currentThread().getState(); // 스레드 상태 값 얻기

                System.out.println("현재 스레드 이름 = " + name);
                System.out.println("현재 스레드 ID = " + id);
                System.out.println("현재 스레드 우선순위 값 = " + priority);
                System.out.println("현재 스레드 상태 = " + s);
        }
}

// 출력
// 현재 스레드 이름 = main
// 현재 스레드 ID = 1
// 현재 스레드 우선순위 값 = 5
// 현재 스레드 상태 = RUNNABLE
```

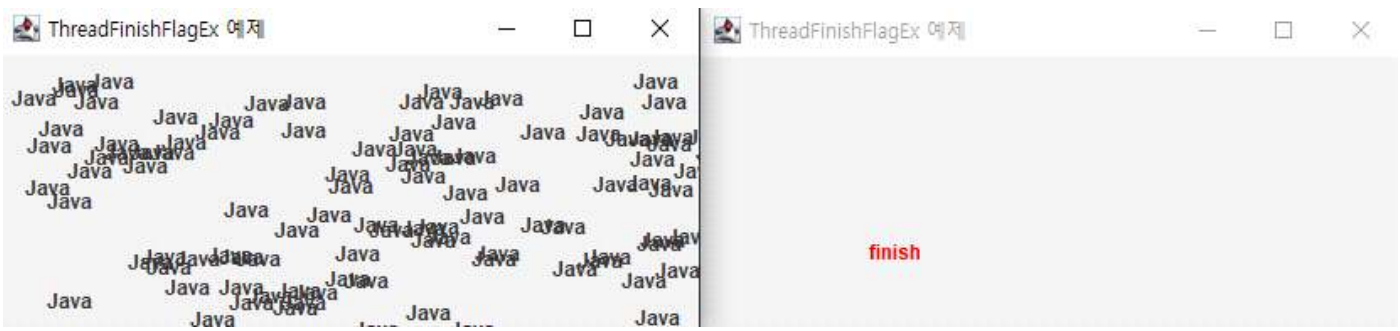
```
// 스레드 종료와 타 스레드 강제 종료
// 스스로 종료 : run() 메소드 리턴
// 타 스레드에서 강제 종료 : interrupt() 메소드 사용

// 타이머 스레드를 강제 종료시키는 응용프로그램
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class TimerRunnable implements Runnable {
	private JLabel timerLabel;
	public TimerRunnable(JLabel timerLabel) {
		this.timerLabel = timerLabel;
	}
	@Override public void run() {
		int n=0;
		while(true) {
			timerLabel.setText(Integer.toString(n));
			n++;
			try {
				Thread.sleep(1000); // 1초 동안 잠을 잔다.
			}
			catch(InterruptedException e) {
				return; // 예외가 발생하면 스레드 종료
			}
		}
	}
}
public class ThreadInterruptEx extends JFrame {
	private Thread th;
	public ThreadInterruptEx() {
		setTitle(" hreadInterruptEx 예제");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Container c = getContentPane();
		c.setLayout(new FlowLayout());

		JLabel timerLabel = new JLabel();
		timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));

		TimerRunnable runnable = new TimerRunnable(timerLabel);
		th = new Thread(runnable); // 스레드 생성
		c.add(timerLabel);
		// 버튼을 생성하고 Action 리스너 등록
		JButton btn =new JButton("kill Timer");
		btn.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				th.interrupt(); // 타이머 스레드 강제 종료
				JButton btn = (JButton)e.getSource();
				btn.setEnabled(false); // 버튼 비활성화
			}
		});
		c.add(btn);
		setSize(300,170);
		setVisible(true);
		th.start(); // 스레드 동작시킴
	}
	public static void main(String[] args) {
		new ThreadInterruptEx();
	}
}
```





```
// flag를 이용한 종료
// 스레드 A가 스레드 B의 flag를 true로 만들면, 스레드 B가 스스로 종료하는 방식

// 프로그램이 시작하자마자 0.3초 주기로 "Java" 문자열을 임의의 위치에 생성하는 스레드 생성하라
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class RandomThread extends Thread {
	private Container contentPane;
	private boolean flag=false; // 스레드의 종료 명령을 표시하는 플래그. true : 종료 지시

	public RandomThread(Container contentPane) {
		this.contentPane = contentPane;
	}

	void finish() { // 스레드 종료 명령을 flag에 표시
		flag = true;
	}

	@Override public void run() {
		while(true) {
			int x = ((int)(Math.random()*contentPane.getWidth()));
			int y = ((int)(Math.random()*contentPane.getHeight()));

			JLabel label = new JLabel("Java"); //새 레이블 생성

			label.setSize(80, 30);
			label.setLocation(x, y);

			contentPane.add(label);
			contentPane.repaint();

			try {
				Thread.sleep(300); // 0.3초 동안 잠을 잔다.
				if(flag==true) {
					contentPane.removeAll();

					label = new JLabel("finish");
					label.setSize(80, 30);
					label.setLocation(100, 100);
					label.setForeground(Color.RED);

					contentPane.add(label);
					contentPane.repaint();

					return; // 스레드 종료
				}
			}catch(InterruptedException e) {
				return;
			}
		}
	}
}
```

```
public class ThreadFinishFlagEx extends JFrame {
	private RandomThread th; // 스레드 레퍼런스

	public ThreadFinishFlagEx() {
		setTitle("ThreadFinishFlagEx 예제");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		Container c = getContentPane();
		c.setLayout(null);

		c.addMouseListener(new MouseAdapter() {
			@Override
			public void mousePressed(MouseEvent e) {
				th.finish(); // RandomThread 스레드 종료 명령
			}
		});

		setSize(300,200);
		setVisible(true);

		th = new RandomThread(c); // 스레드 생성
		th.start(); // 스레드 동작시킴
	}

	public static void main(String[] args) {
		new ThreadFinishFlagEx();
	}
}
```

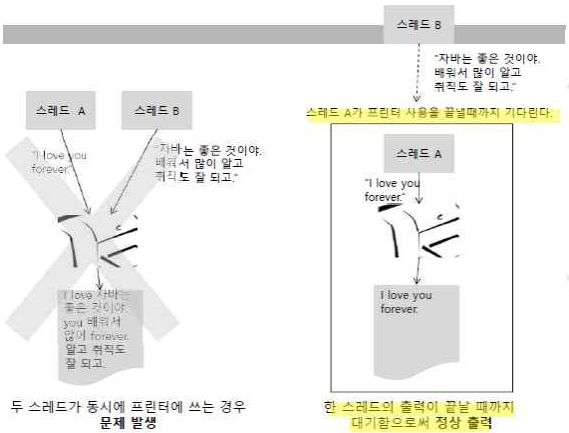


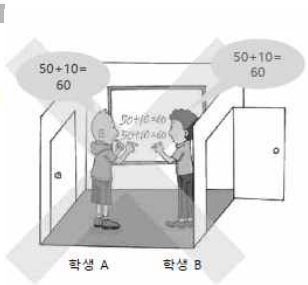
// 스레드 동기화(Thread Synchronization)

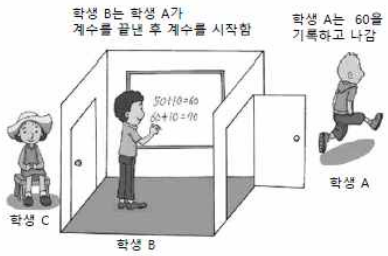
// 멀티스레드 프로그램 작성시 주의점
// 다수의 스레드가 공유 데이터에 동시에 접근하는 경우 - 공유 데이터의 값에 예상치 못한 결과 발생 가능

// 스레드 동기화
// 멀티스레드의 공유 데이터의 동시 접근 문제 해결책
// 공유 데이터를 접근하는 모든 스레드의 한 줄 세우기
// 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 대기 하도록 함

// 두 스레드의 프린터 동시 쓰기로 충돌하는 사례







// 스레드 동기화 기법

// 스레드 동기화
// 공유 데이터에 동시에 접근하는 다수의 스레드가 공유 데이터를 배타적으로 접근하기 위해 상호 협력(coordination)하는 것
// 동기화의 핵심 - 스레드의 공유 데이터에 대한 배타적 독점 접근 보장

// 자바에서 스레드 동기화를 위한 방법
// synchronized로 동기화 블록 지정
// wait()-notify() 메소드로 스레드 실행 순서 제어

// synchronized 블록 지정
// synchronized 키워드
// 한 스레드가 독점 실행해야 하는 부분(동기화 코드)을 표시하는 키워드 - 임계 영역(critical section) 표기 키워드
// 메소드 전체 혹은 코드 블록

// synchronized 블록에 대한 컴파일러의 처리
// 먼저 실행한 스레드가 모니터 소유 - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
// 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드 대기

```
synchronized void add() {
    int n = getCurrentSum();
    n+=10;
    setCurrentSum(n);
}
```
synchronized 메소드

```
void execute() {
    // 다른 코드들
    //
    synchronized(this) {
        int n = getCurrentSum();
        n+=10;
        setCurrentSum(n);
    }
    //
    // 다른 코드들
}
```
synchronized 코드 블록

// 아래 코드 설명
// SharedBoard의 add()를 스레드1이 실행하고 있는 동안,
// 스레드2가 호출하면 스레드2는 대기

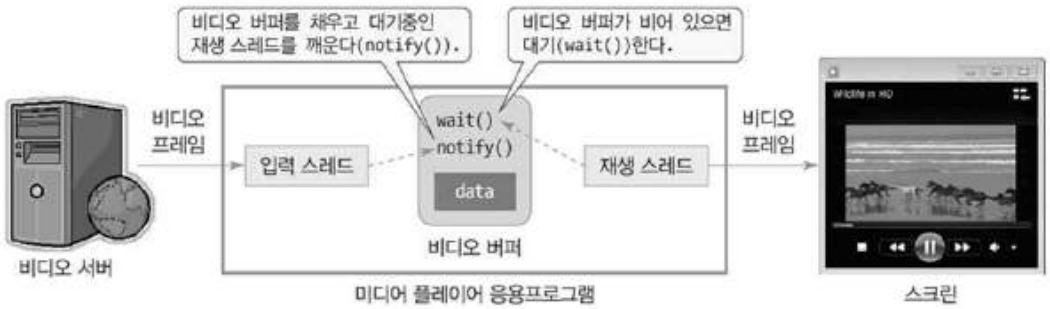
```
// 공유 집계판 사례를 코딩
public class SynchronizedEx {
	public static void main(String [] args) {
		SharedBoard board = new SharedBoard();

		Thread th1 = new StudentThread("kitae", board);
		Thread th2 = new StudentThread("hyosoo", board);

		th1.start();
		th2.start();
	}
}

class SharedBoard {
	private int sum = 0; // 집계판의 합

	synchronized public void add() {
		int n = sum;
		Thread.yield(); // 현재 실행 중인 스레드 양보
		n += 10; // 10 증가
		sum = n; // 증가한 값을 집계합에 기록
		System.out.println(Thread.currentThread().getName() + " : " + sum);
	}

	public int getSum() {
		return sum;
	}
}

class StudentThread extends Thread {
	private SharedBoard board; // 집계판의 주소

	public StudentThread(String name, SharedBoard board) {
		super(name);
		this.board = board;
	}
	@Override
	public void run() {
		for(int i=0; i<10; i++)
			board.add();
	}
}

// 출력
// kitae : 10
// kitae : 20
// kitae : 30
// kitae : 40
// kitae : 50
// kitae : 60
// kitae : 70
// kitae : 80
// kitae : 90
// kitae : 100
// hyosoo : 110
// hyosoo : 120
// hyosoo : 130
// hyosoo : 140
// hyosoo : 150
// hyosoo : 160
// hyosoo : 170
// hyosoo : 180
// hyosoo : 190
// hyosoo : 200
```

```
// 공유집계판 사례에서 synchronized 사용하지 않아 충돌로 인해 데이터에 오류가 발생한 경우
public class SynchronizedEx_Error {
	public static void main(String [] args) {
		SharedBoard board = new SharedBoard();

		Thread th1 = new StudentThread("kitae", board);
		Thread th2 = new StudentThread("hyosoo", board);

		th1.start();
		th2.start();
	}
}
class SharedBoard {
	private int sum = 0; // 집계판의 합

	synchronized public void add() {
		int n = sum;
		Thread.yield(); // 현재 실행 중인 스레드 양보
		n += 10; // 10 증가
		sum = n; // 증가한 값을 집계합에 기록
		System.out.println(Thread.currentThread().getName() + " : " + sum);
	}
	public int getSum() {
		return sum;
	}
}
class StudentThread extends Thread {
	private SharedBoard board; // 집계판의 주소

	public StudentThread(String name, SharedBoard board) {
		super(name);
		this.board = board;
	}
	@Override
	public void run() {
		for(int i=0; i<10; i++)
			board.add();
	}
}
// 결과 : 컴퓨터가 좋아, 스레드의 역량이 남아있어 정상적인 결과를 보임, 여러개 창을 띄우고 하면 오류가 남
// kitae : 10
// kitae : 20
// kitae : 30
// kitae : 40
// kitae : 50
// kitae : 60
// kitae : 70
// kitae : 80
// kitae : 90
// kitae : 100
// hyosoo : 110
// hyosoo : 120
// hyosoo : 130
// hyosoo : 140
// hyosoo : 150
// hyosoo : 160
// hyosoo : 170
// hyosoo : 180
// hyosoo : 190
// hyosoo : 200
```

// producer-consumer 문제와 동기화

// producer-consumer 문제
// producer : 공유 메모리에 데이터를 공급하는 스레드
// consumer : 공유 메모리의 데이터를 소비하는 스레드
// 문제의 본질 - producer와 consumer 가 동시에 공유 데이터를 접근하는 문제

// producer-consumer 문제 사례
// 미디어 플레이어 - producer:입력스레드, consumer:재생스레드, 공유데이터:비디오버퍼



// wait(), notify(), notifyAll()를 이용한 동기화

// 동기화 객체 : 두 개 이상의 스레드 동기화에 사용되는 객체

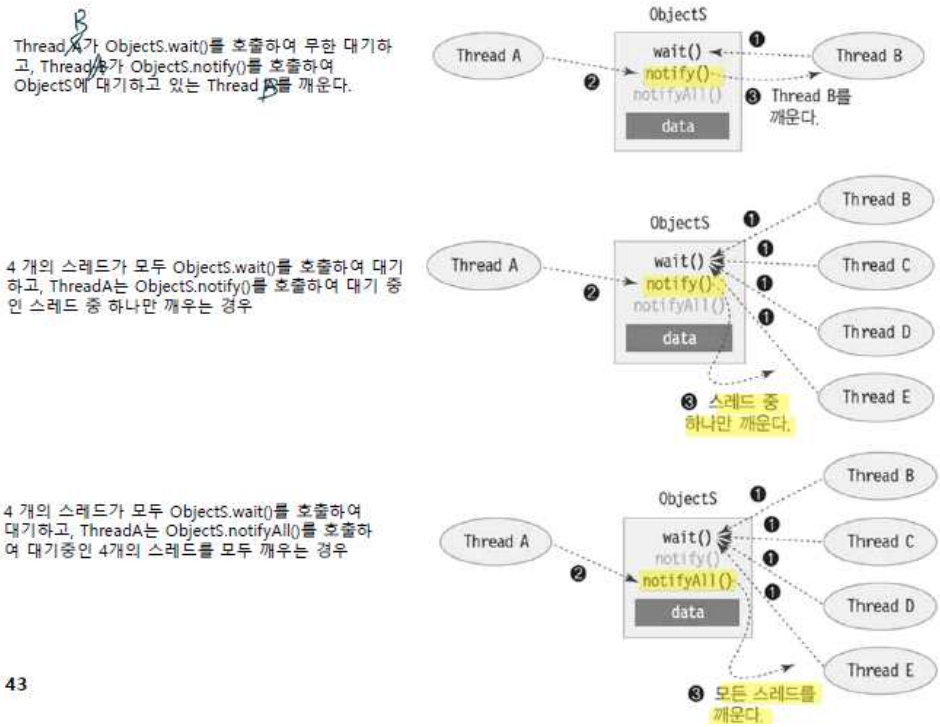
// 동기화 메소드
// wait()
// 다른 스레드가 notify()를 불러줄 때까지 기다린다.

// notify()
// wait()를 호출하여 대기중인 스레드를 깨우고 RUNNABLE 상태로 만든다.
// 2개 이상의 스레드가 대기중이라도 오직 한 스레드만 깨운다.

// notifyAll()
// wait()를 호출하여 대기중인 모든 스레드를 깨우고 모두 RUNNABLE 상태로 만든다.

// synchronized 블록 내에서만 사용되어야 함

// wait(), notify(), notifyAll()은 Object의 메소드
// 모든 객체가 동기화 객체가 될 수 있다.
// Thread 객체도 동기화 객체로 사용될 수 있다.



```
// wait(), notify()를 이용한 바 채우기
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class MyLabel extends JLabel {
	int barSize = 0; // 바의 크기
	int maxBarSize;

	MyLabel(int maxBarSize) {
		this.maxBarSize = maxBarSize;
	}

	public void paintComponent(Graphics g) {
		super.paintComponent(g);

		g.setColor(Color.MAGENTA);

		int width = (int)(((double)(this.getWidth())) /maxBarSize*barSize);
		if(width==0)
			return;

		g.fillRect(0, 0, width, this.getHeight());
	}

	synchronized void fill() {
		if(barSize == maxBarSize) {
			try {
				wait();
			}
			catch (InterruptedException e) {
				return;
			}
		}
		barSize++;
		repaint(); // 바 다시 그리기
		notify();
	}

	synchronized void consume() {
		if(barSize == 0) {
			try {
				wait();
			}
			catch (InterruptedException e){
				return;
			}
		}
		barSize--;
		repaint(); // 바 다시 그리기
		notify();
	}
}
```

```
class ConsumerThread extends Thread {
	MyLabel bar;

	ConsumerThread(MyLabel bar) {
		this.bar = bar;
	}

	public void run() {
		while(true) {
			try {
				sleep(200);
				bar.consume();
			}
			catch (InterruptedException e){
				return;
			}
		}
	}
}

public class TabAndThreadEx extends JFrame {
	MyLabel bar = new MyLabel(100);

	TabAndThreadEx(String title) {
		super(title);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		Container c = getContentPane();
		c.setLayout(null);

		bar.setBackground(Color.ORANGE);
		bar.setOpaque(true);
		bar.setLocation(20, 50);
		bar.setSize(300, 20);
		c.add(bar);

		c.addKeyListener(new KeyAdapter() {
			public void keyPressed(KeyEvent e) {
				bar.fill();
			}
		});

		setSize(350,200);
		setVisible(true);

		c.requestFocus();

		ConsumerThread th = new ConsumerThread(bar);
		th.start(); // 스레드 시작
	}

	public static void main(String[] args) {
		new TabAndThreadEx( "아무키나 빨리 눌러 바 채우기");
	}
}
```