# 12

# 자바 스레드 기초

# 학습 목표

1. 멀티태스킹과 스레드의 개념 이해
2. Thread 클래스를 상속받아 자바 스레드 만들기
3. Runnable 인터페이스를 구현하여 자바 스레드 만들기
4. 스레드 종료 시키기
5. 스레드의 동기화 개념과 필요성 이해
6. synchronized로 간단한 스레드 동기화
7. wait()-notify()로 간단한 스레드 동기화

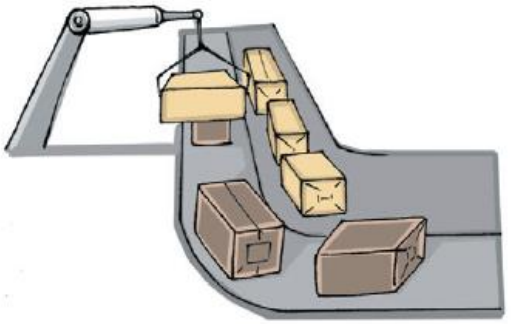
# 멀티태스킹(multi-tasking) 개념



- 멀티태스킹
  - 여러 개의 작업(태스크)이 동시에 처리되는 것

다림질하면서
전화 받는 주부

운전하면서 음악을
듣는 연수 양

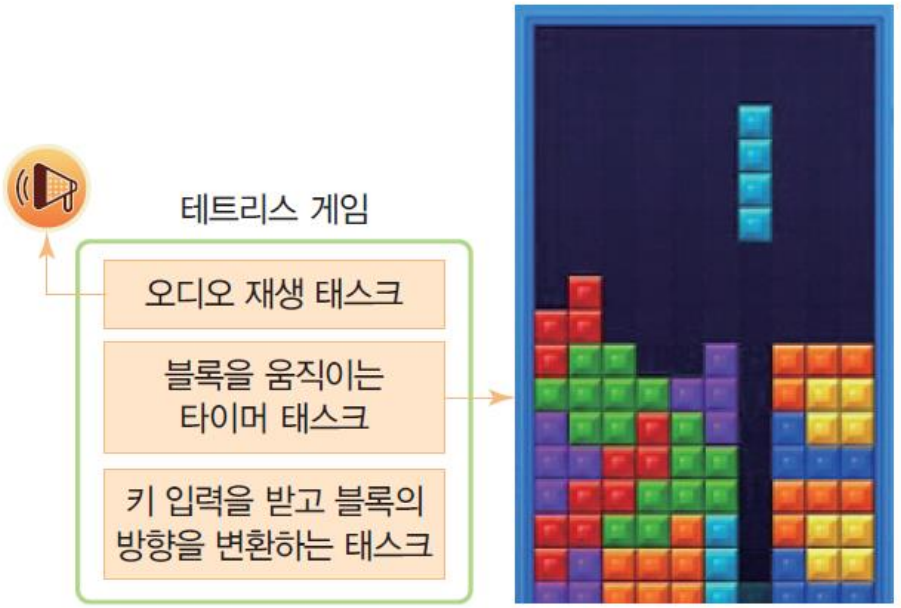


판독과 포장을
동시에 하는 기계

# 멀티태스킹 프로그램 사례




(a) 미디어 플레이어의 멀티태스킹




(b) 테트리스 게임의 멀티태스킹

# 스레드와 운영체제



- 스레드(thread)
  - 운영체제에 의해 관리되는 하나의 작업 혹은 태스크
  - 스레드와 태스크(혹은 작업)은 바꾸어 사용해도 무관

- 멀티스레딩(multi-threading)
  - 여러 스레드를 동시에 실행시키는 응용프로그램을 작성하는 기법

- 스레드 구성
  - 스레드 코드
    - 작업을 실행하기 위해 작성한 프로그램 코드
    - 개발자가 작성
  - 스레드 정보
    - 스레드 명, 스레드 ID, 스레드의 실행 소요 시간, 스레드의 우선 순위 등
    - 운영체제가 스레드에 대해 관리하는 정보

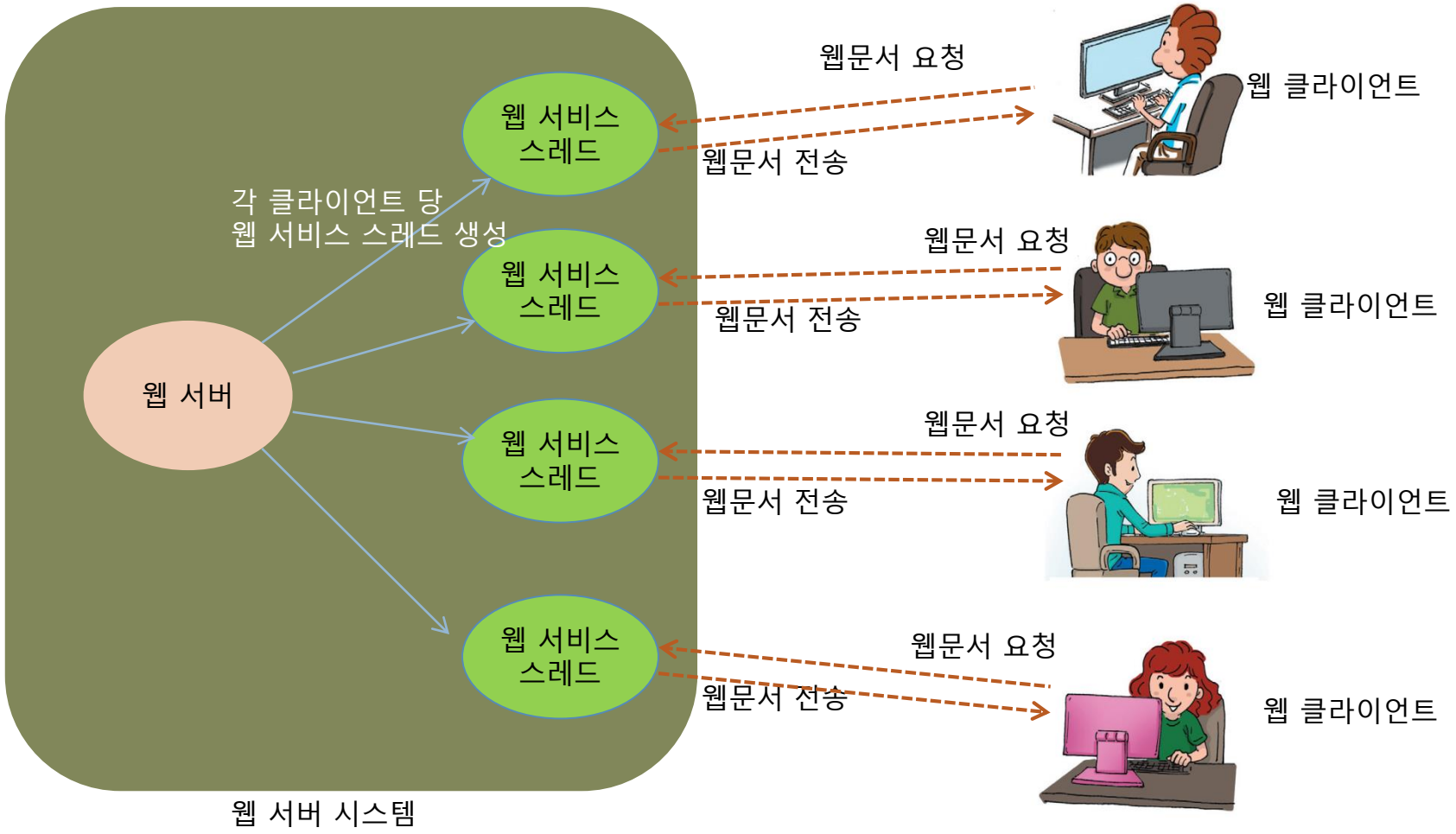
# 멀티태스킹과 멀티스레딩



- 멀티태스킹 구현 기술
  - 멀티프로세싱(multi-processing)
    - 하나의 응용프로그램이 여러 개의 프로세스를 생성하고, 각 프로세스가 하나의 작업을 처리하는 기법
    - 각 프로세스 독립된 메모리 영역을 보유하고 실행
    - 프로세스 사이의 문맥 교환에 따른 과도한 오버헤드와 시간 소모의 문제점
  - 멀티스레딩(multi-threading)
    - 하나의 응용프로그램이 여러 개의 스레드를 생성하고, 각 스레드가 하나의 작업을 처리하는 기법
    - 하나의 응용프로그램에 속한 스레드는 변수 메모리, 파일 오픈 테이블 등 자원으로 공유하므로, 문맥 교환에 따른 오버헤드가 매주 작음
    - 현재 대부분의 운영체제가 멀티스레딩을 기본으로 하고 있음

# 웹 서버의 멀티스레딩 사례



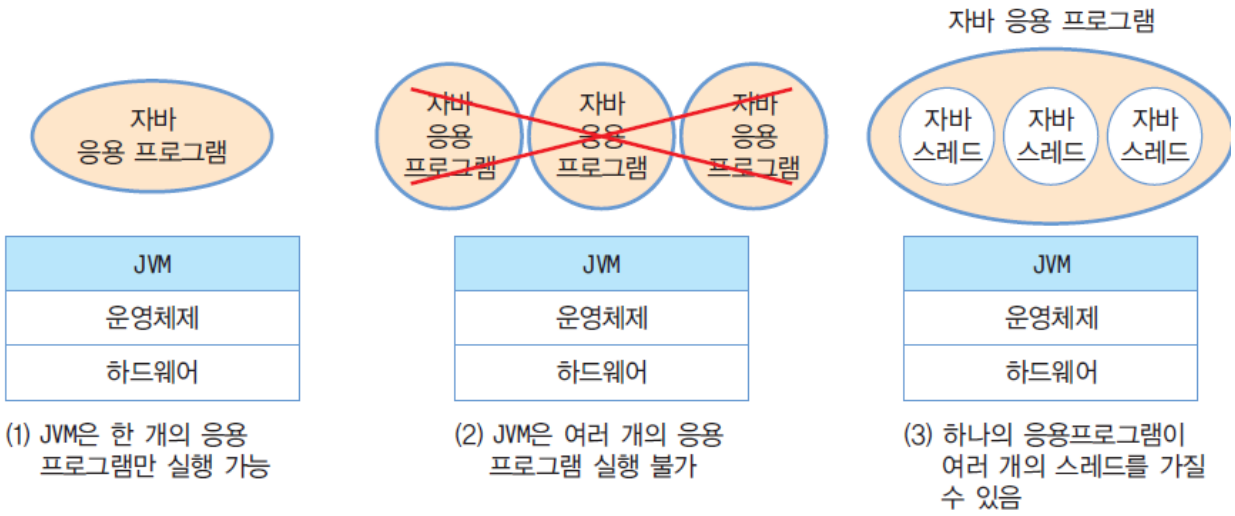


# 자바 스레드(Thread)와 JVM



- 자바 스레드
  - 자바 가상 기계(JVM)에 의해 스케쥴되는 실행 단위의 코드 블럭
  - 스레드의 생명 주기는 JVM에 의해 관리됨 : JVM은 스레드 단위로 스케쥴링

- JVM과 자바의 멀티스레딩
  - 하나의 JVM은 하나의 자바 응용프로그램만 실행
    - 자바 응용프로그램이 시작될 때 JVM이 함께 실행됨
    - 자바 응용프로그램이 종료하면 JVM도 함께 종료함
  - 응용프로그램은 하나 이상의 스레드로 구성 가능



# 자바 스레드 만들기



- 스레드 만드는 2 가지 방법
  - java.lang.Thread 클래스를 상속받아 스레드 작성
  - java.lang.Runnable 인터페이스를 구현하여 스레드 작성

# Thread 클래스를 상속받아 스레드 만들기(1)



Thread 클래스의 주요 메소드

| Thread의 메소드 | 내용 |
|---|---|
| Thread()<br>Thread(Runnable target)<br>Thread(String name)<br>Thread(Runnable target, String name) | 스레드 객체 생성<br>Runnable 객체인 target을 이용하여 스레드 객체 생성<br>이름이 name인 스레드 객체 생성<br>Runnable 객체를 이용하며, 이름이 name인 스레드 객체 생성 |
| void run() | 스레드 코드로서 JVM에 의해 호출된다. 개발자는 반드시 오버라이딩하여 스레드 코드를 작성하여야 한다. 이 메소드가 종료하면 스레드도 종료 |
| void start() | JVM에게 스레드 실행을 시작하도록 요청 |
| void interrupt() | 스레드 강제 종료 |
| static void yield() | 다른 스레드에게 실행을 양보한다. 이때 JVM은 스레드 스케줄링을 시행하며 다른 스레드를 선택하여 실행시킨다. |
| void join() | 스레드가 종료할 때까지 기다린다. |
| long getId() | 스레드의 ID 값 리턴 |
| String getName() | 스레드의 이름 리턴 |
| int getPriority() | 스레드의 우선순위 값 리턴. 1에서 10 사이 값 |
| void setPriority(int n) | 스레드의 우선순위 값을 n으로 변경 |
| Thread.State getState() | 스레드의 상태 값 리턴 |
| static void sleep(long mills) | 스레드는 mills 시간 동안 잔다. mills의 단위는 밀리초 |
| static Thread currentThread() | 현재 실행 중인 스레드 객체의 레퍼런스 리턴 |

# Thread 클래스를 상속받아 스레드 만들기(2)



- Thread를 상속받아 run() 오버라이딩
  - Thread 클래스 상속. 새 클래스 작성
  - run() 메소드 작성
    - run() 메소드를 스레드 코드라고 부름
    - run() 메소드에서 스레드 실행 시작
- 스레드 객체 생성
  - 생성된 객체는 필드와 메소드를 가진 객체일 뿐 스레드로 작동하지 않음
- 스레드 시작
  - start() 메소드 호출
    - 스레드로 작동 시작
    - 스레드 객체의 run()이 비로소 실행
    - JVM에 의해 스케쥴되기 시작함

```
class TimerThread extends Thread {
   ..........

   @Override
   public void run() { // run() 오버라이딩
      ..........
   }
}
```

```
TimerThread th = new TimerThread();
```

```
th.start();
```

# Thread를 상속받아 1초 단위로 초 시간을 출력하는 TimerThread 스레드 작성 사례



스레드 클래스 선언

스레드 코드 작성

1초에 한 번씩 n을 증가시켜 콘솔에 출력한다.

```
class TimerThread extends Thread {
    int n = 0;

    @Override
    public void run() {
        while(true) { // 무한루프를 실행한다.
            System.out.println(n);
            n++;
            try {
                sleep(1000); //1초 동안 잠을 잔 후 깨어난다.
            }
            catch(InterruptedException e){return;}
        }
    }
}
```

```
0
1
2
3
4
.
.
.
```

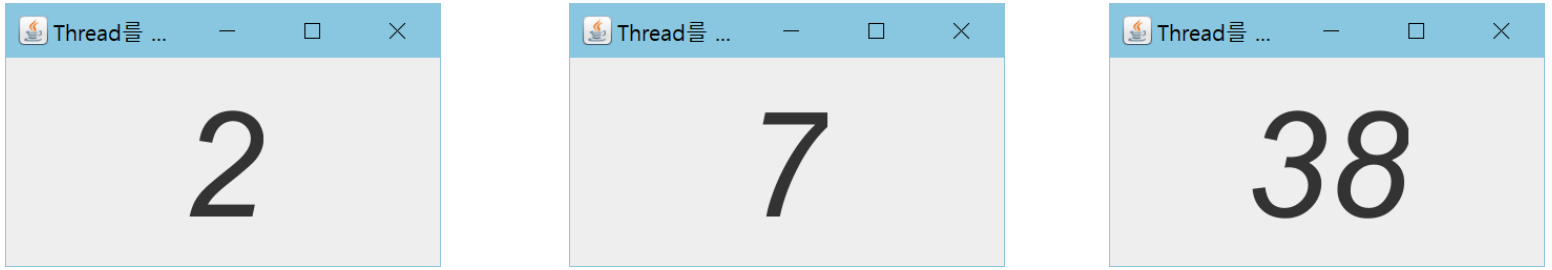
스레드 객체 생성

스레드 시작

```
public class TestThread {
    public static void main(String [] args) {
        TimerThread th = new TimerThread();
        th.start();
    }
}
```

# 예제 12-1 : Thread를 상속받아 1초 단위 타이머 스레드 만들기






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

class TimerThread extends Thread {
    private JLabel timerLabel; // 타이머 값이 출력되는 레이블
    public TimerThread(JLabel timerLabel) {
        this.timerLabel = timerLabel;
    }

    // 스레드 코드. run()이 종료하면 스레드 종료
    @Override
    public void run() {
        int n=0; // 타이머 카운트 값
        while(true) { // 무한 루프
            timerLabel.setText(Integer.toString(n));
            n++; // 카운트 증가
            try {
                Thread.sleep(1000); // 1초동안 잠을 잔다.
            }
            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());

            // 타이머 값을 출력할 레이블 생성
            JLabel timerLabel = new JLabel();
            timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));
            c.add(timerLabel);

            TimerThread th = new TimerThread(timerLabel);
            setSize(250,150);
            setVisible(true);
            th.start(); // 타이머 스레드의 실행을 시작하게 한다.
    }

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

# Runnable 인터페이스로 스레드 만들기



- Runnable 인터페이스 구현하는 새 클래스 작성
  - run() 메소드 구현
    - run() 메소드를 스레드 코드라고 부름
    - run() 메소드에서 스레드 실행 시작

```
class TimerRunnable implements Runnable {
   ..........................................................

   @Override
   public void run() { // run() 메소드 구현
      ..........................
   }
}
```

- 스레드 객체 생성

```
Thread th = new Thread(new TimerRunnable());
```

- 스레드 시작
  - start() 메소드 호출
    - 스레드로 작동 시작
    - 스레드 객체의 run()이 비로소 실행
    - JVM에 의해 스케쥴되기 시작함

```
th.start();
```

# 예제 12-2 : Runnable 인터페이스를 이용하여 1초 단위로 출력하는 타이머 스레드 만들기






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

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

    // 스레드 코드. run()이 종료하면 스레드 종료
    @Override
    public void run() {
        int n=0; // 타이머 카운트 값
        while(true) { // 무한 루프
            timerLabel.setText(Integer.toString(n));
            n++;
            try {
                Thread.sleep(1000); // 1초동안 잠을 잔다.
            }
            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());

        // 타이머 값을 출력할 레이블 생성
        JLabel timerLabel = new JLabel();
        timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));
        c.add(timerLabel); // 레이블을 컨텐트팬에 부착

        TimerRunnable runnable = new TimerRunnable(timerLabel);
        Thread th = new Thread(runnable); // 스레드 객체 생성
        setSize(250,150);
        setVisible(true);
        th.start(); // 타이머 스레드가 실행을 시작하게 한다.
    }
    public static void main(String[] args) {
        new RunnableTimerEx();
    }
}
```

# main 스레드



- main 스레드
  - JVM이 응용프로그램을 실행할 때 디폴트로 생성되는 스레드
    - main() 메소드 실행 시작
    - main() 메소드가 종료하면 main 스레드 종료

# 예제 12-3 : main 스레드 확인과 스레드 정보를 알아내는 코드



main() 메소드 내에서 현재 스레드 정보를 가진 Thread 객체를 알아내어 현재 실행중인 스레드에 관한 다양한 정보를 출력한다.

```
public class ThreadMainEx {
    public static void main(String [] args) {
        long id = Thread.currentThread().getId();
        String name = Thread.currentThread().getName();
        int priority = Thread.currentThread().getPriority();
        Thread.State s = Thread.currentThread().getState();

        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);
    }
}
```

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

# 스레드 종료와 타 스레드 강제 종료



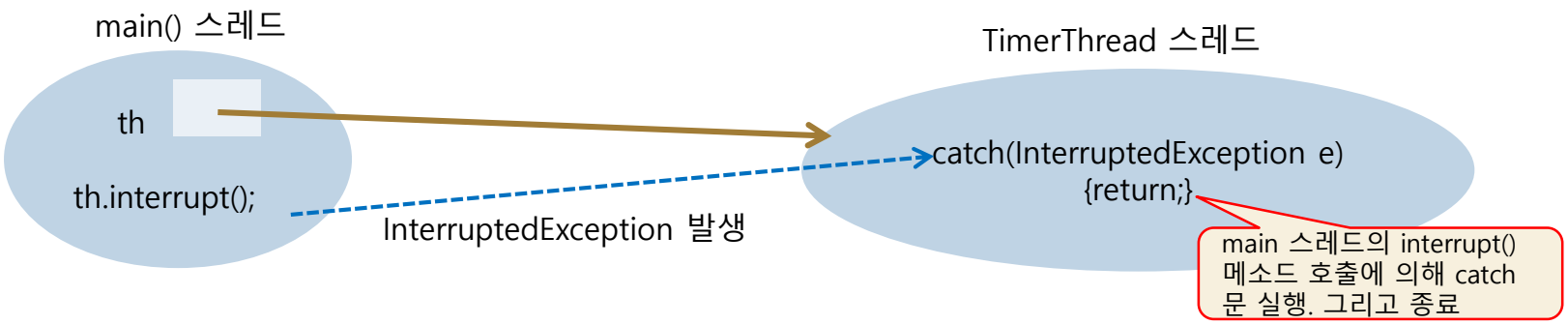
- 스스로 종료
  - run() 메소드 리턴
- 타 스레드에서 강제 종료
  - interrupt() 메소드 사용

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

    th.interrupt(); // TimerThread 강제 종료
}
```

```
class TimerThread extends Thread {
    int n = 0;
    @Override
    public void run() {
        while(true) {
            System.out.println(n); // 화면에 카운트 값 출력
            n++;
             try {
                sleep(1000);
            }
            catch(InterruptedException e){
                return; // 예외를 받고 스스로 리턴하여 종료
            }
        }
    }
}
```

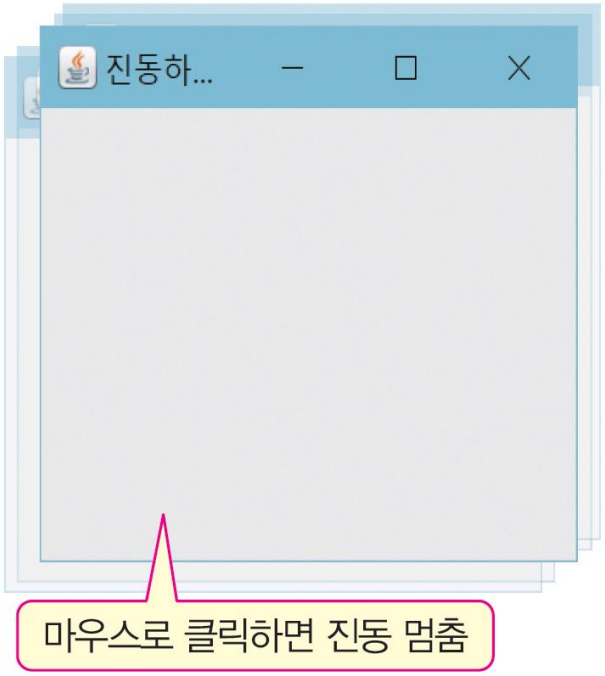
return 하지 않으면 스레드는 종료하지 않음



# 예제 12-4 : 진동하는 스레드와 스레드의 강제 종료



Runnable을 받은 스레드를 작성하여 프레임이 심하게 진동하도록 프로그램을 작성하라. 그리고 컨텐트팬에 마우스를 클릭하면 진동 스레드를 종료시켜 진동이 멈추도록 하라



# 예제 12-4 정답



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;

public class VibratingFrame extends JFrame implements Runnable
{
    private Thread th; // 진동하는 스레드
    public VibratingFrame() {
        setTitle("진동하는 프레임 만들기");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200,200);
        setLocation(300,300);
        setVisible(true);

        getContentPane().addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if(!th.isAlive()) return;
                    th.interrupt();
            }
        });

        th = new Thread(this); // 진동하는 스레드 객체 생성
        th.start(); // 진동 시작
    }
```

Runnable 인터페이스 구현. 프레임에 run() 메소드 반드시 작성 필요

프레임 객체가 Runnable 인터페이스를 구현한 객체이므로 this 가능

```
    @Override
    public void run() { // 프레임의 진동을 일으키기 위해
            // 20ms마다 프레임의 위치를 랜덤하게 이동
        Random r = new Random();
        while(true) {
            try {
                Thread.sleep(20); // 20ms 잠자기
            }
            catch(InterruptedException e){
                return; // 리턴하면 스레드 종료
            }
            int x = getX() + r.nextInt()%5; // 새 위치 x
            int y = getY() + r.nextInt()%5; // 새 위치 y
            setLocation(x, y); // 프레임의 위치 이동
        }
    }
    public static void main(String [] args) {
        new VibratingFrame();
    }
}
```

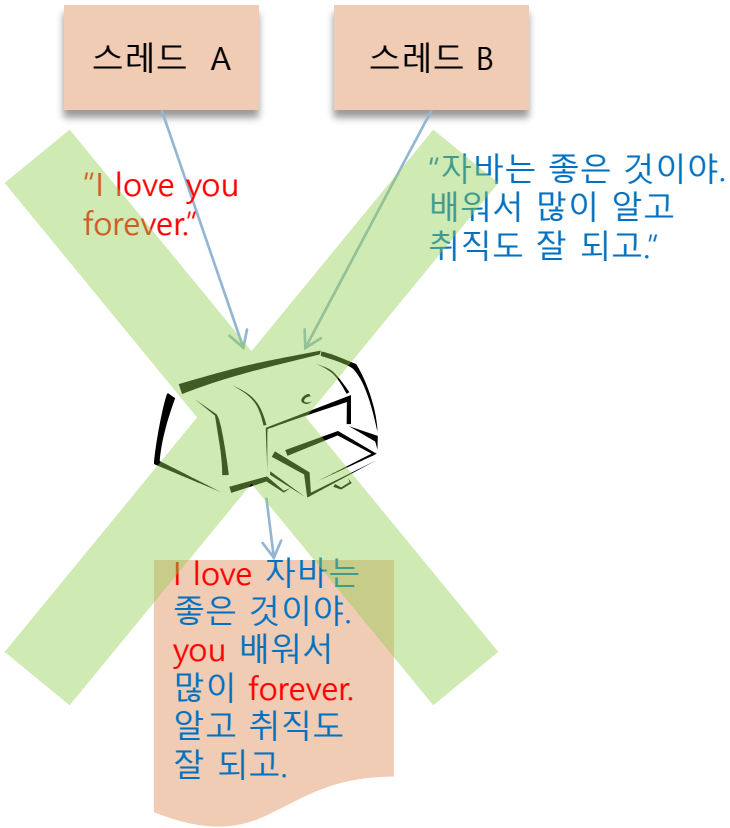
# 스레드 동기화(Thread Synchronization)



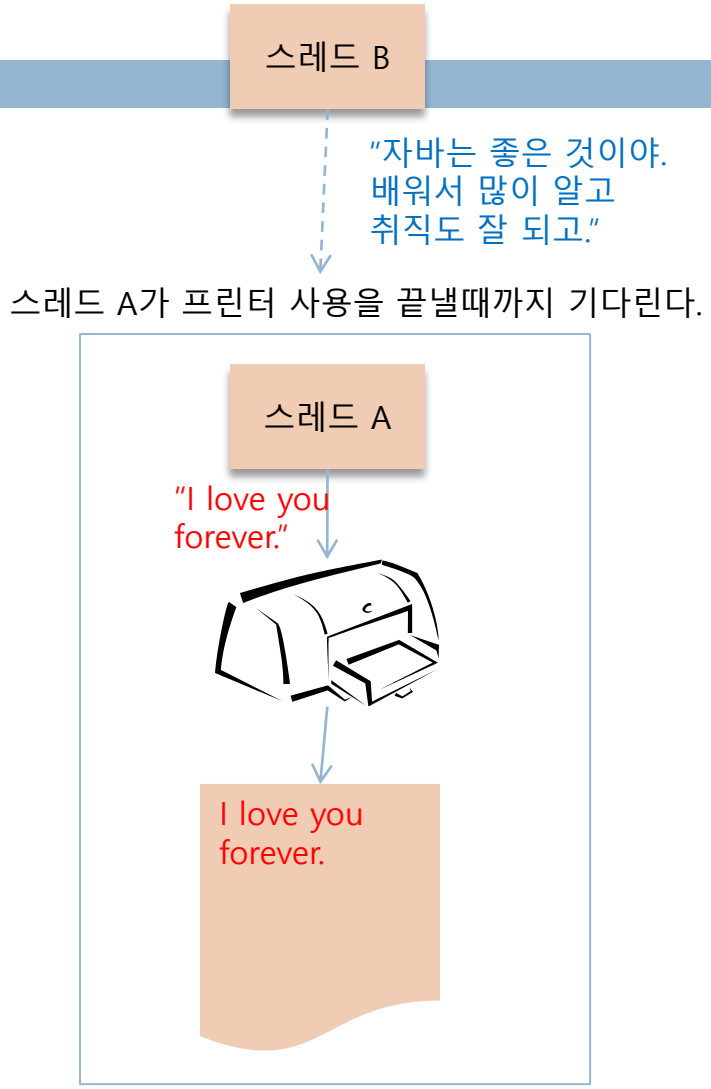
- 멀티스레드 프로그램 작성시 주의점
  - 다수의 스레드가 공유 데이터에 동시에 접근하는 경우
    - 공유 데이터의 값에 예상치 못한 결과 발생 가능
- 스레드 동기화
  - 동기화란?
    - 스레드 사이의 실행순서 제어, 공유데이터에 대한 접근을 원활하게 하는 기법
  - 멀티스레드의 공유 데이터의 동시 접근 문제 해결
    - 방법1) 공유 데이터를 접근하는 모든 스레드의 한 줄 세우기
    - 방법2) 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 대기 하도록 함
- 자바의 스레드 동기화 방법 - 2가지
  - synchronized 키워드로 동기화 블록 지정
  - wait()-notify() 메소드로 스레드의 실행 순서 제어

# 동기화의 필요성 - 두 스레드가 프린터에 동시 쓰기로 충돌하는 경우






두 스레드가 동시에 프린터에 쓰는 경우
**문제 발생**




한 스레드의 출력이 끝날 때까지 다른 스레드 대기함으로써
**정상 출력**

# synchronized 블록 지정



- synchronized 키워드
  - 스레드가 독점적으로 실행해야 하는 부분(동기화 코드)을 표시하는 키워드
    - 임계 영역(criitical section) 표기 키워드
  - synchronized 블록 지정 방법
    - 메소드 전체 혹은 코드 블록

- synchronized 블록이 실행될 때,
  - 먼저 실행한 스레드가 모니터 소유
    - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
  - 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드 대기

```
synchronized void print(String text) { // 동기화 메소드
    ...
    for(int i=0; i<text.length(); i++) // text의 각 문자 출력
        System.out.print(text.charAt(i));
    ...
}
```

synchronized 메소드

```
void execute(String text) {
    ...
    synchronized(this) { // 동기화 코드 블록
        ...
        for(int i=0; i<text.length(); i++)
            System.out.print(text.charAt(i));
            ...
    }
}
```

synchronized 코드 블록

# 예제 12-5 : 두 스레드가 공유 프린터 객체를 통해 동시에 출력하는 경우 동기화 - synchronized 블록 지정



```
public class SynchronizedEx {
    public static void main(String[] args) {
        SharedPrinter p = new SharedPrinter(); // 공유 데이터 생성
        String [] engText = { "Wise men say, ",
                              "only fools rush in",
                              "But I can't help, ",
                              "falling in love with you",
                              "Shall I stay? ",
                              "Would it be a sin?",
                              "If I can't help, ",
                              "falling in love with you" };
        String [] korText = { "동해물과 백두산이 마르고 닳도록, ",
                              "하느님이 보우하사 우리 나라 만세",
                              "무궁화 삼천리 화려강산, ",
                              "대한 사람 대한으로 길이 보전하세",
                              "남산 위에 저 소나무, 철갑을 두른 듯",
                              "바람서리 불변함은 우리 기상일세.",
                              "무궁화 삼천리 화려강산, ",
                              "대한 사람 대한으로 길이 보전하세" };
        Thread th1 = new WorkerThread(p, engText);//영문출력스레드
        Thread th2 = new WorkerThread(p, korText);//국문출력스레드

        // 두 스레드를 실행시킨다.
        th1.start();
        th2.start();
    }
}
```

```
// 두 WorkerThread 스레드에 의해 동시 접근되는 공유 프린터
class SharedPrinter {
    // synchronized를 생략하면
    // 한글과 영어가 한 줄에 섞여 출력되는 경우가 발생한다.
    synchronized void print(String text) {
        // Thread.yield();
        for(int i=0; i<text.length(); i++)
        System.out.print(text.charAt(i));
        System.out.println();
    }
}

// 스레드 클래스
class WorkerThread extends Thread {
    private SharedPrinter p; // 공유 프린터 주소
    private String [] text;
    public WorkerThread(SharedPrinter p, String[] text) {
        this.p = p; this.text = text;
    }

    // 스레드는 반복적으로 공유 프린터에 10번 접근 text[] 출력
    @Override
    public void run() {
        for (int i=0; i<text.length; i++) // 한 줄씩 출력
            p.print(text[i]); // 공유 프린터에 출력
    }
}
```

# 예제 12-5 실행 결과



한글이 한 줄 출력되든지 영문이 한 줄 출력되는 것이 정상이지만 synchronized가 생략된 print() 메소드가 두 스레드에 의해 동시 호출되면 두 스레드의 동기화가 이루어지지 않아서 한글과 영문이 섞여 출력된다.

```
Wise men say,
only fools rush in
But I can't help,
falling in love with you
Shall I stay?
Would it be a sin?
If I can't help,
falling in love with you
동해물과 백두산이 마르고 닳도록,
하느님이 보우하사 우리 나라 만세
무궁화 삼천리 화려강산,
대한 사람 대한으로 길이 보전하세
남산 위에 저 소나무, 철갑을 두른 듯
바람서리 불변함은 우리 기상일세.
무궁화 삼천리 화려강산,
대한 사람 대한으로 길이 보전하세
```

라인 31에 synchronized로 선언한 경우

```
Wise 동해물과 백두산이 마르고 닳도록, men say,
only fools rush in
But I can't help,
하느님이 보우하사 우리 나라 만세
falling in love with you
무궁화 삼천리 Shall I stay?
화려강산,
Would it be a sin?
대한 사람 대한으로 길이 보전하세
If I can't help,
남산 위에 저 소나무, 철갑을 두른 듯
falling in love바람서리 불변함은 우리 기상일세.
with you
무궁화 삼천리 화려강산,
대한 사람 대한으로
길이 보전하세
```

print() 메소드 충돌

print() 메소드 충돌

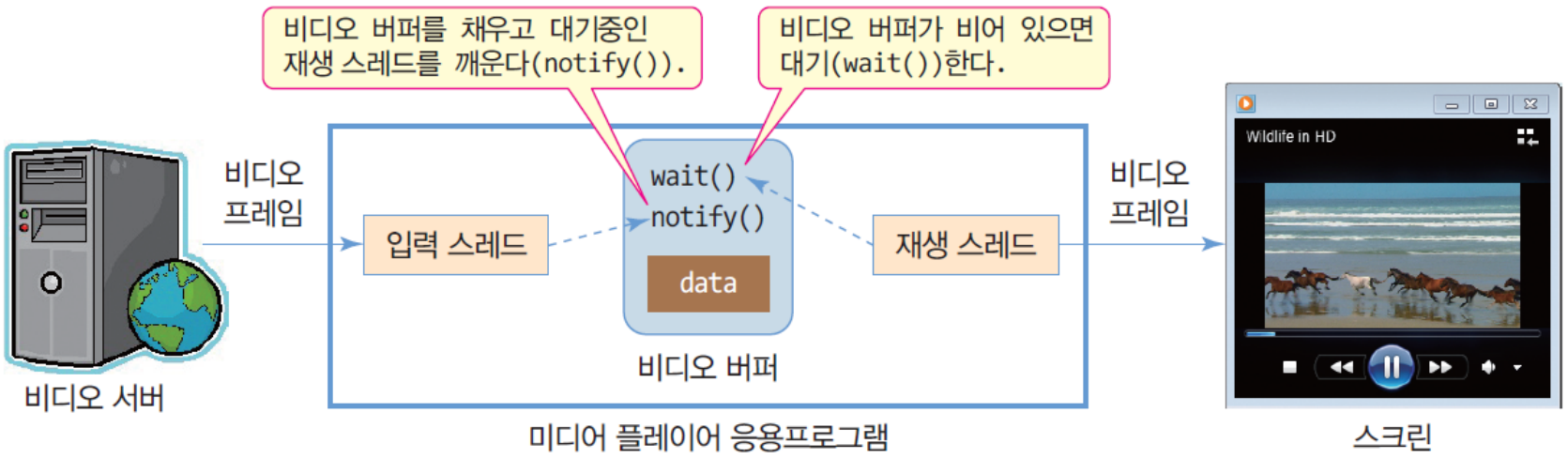
print() 메소드 충돌

라인 31에 synchronized를 생략한 경우

# wait()-notify()를 이용한 스레드 동기화



- wait()-notify()가 필요한 경우
  - 공유 데이터로 두 개 이상의 스레드가 데이터를 주고 받을 때
    - producer-consumer문제
- 동기화 메소드
  - wait() : 다른 스레드가 notify()를 불러줄 때까지 기다린다.
  - notify() : wait()를 호출하여 대기중인 스레드를 깨운다.
    - wait(), notify()는 Object의 메소드

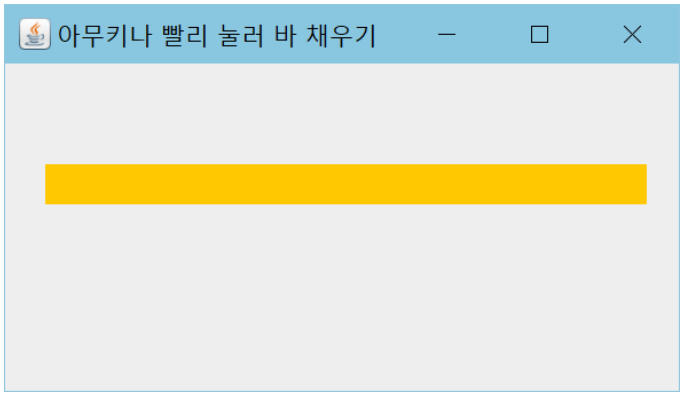


# 예제 12-6 : wait(), notify()를 이용한 바 채우기

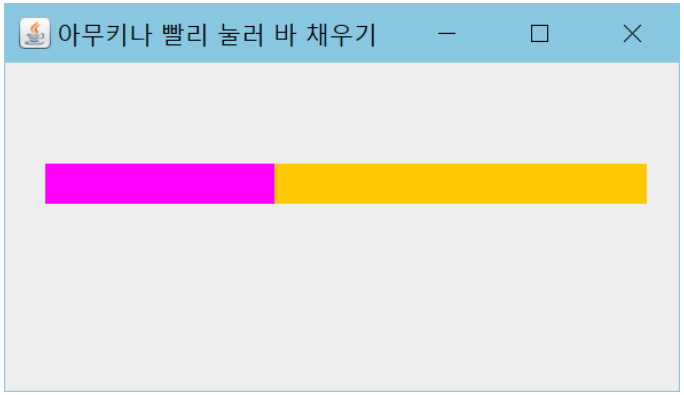


## 다음 설명과 같이 작동하는 스윙 프로그램을 작성하라.

아래 그림에는 스레드를 가진 bar가 있다. 아무 키나 누르면 bar에 마젠타색이 오른쪽으로 1/100씩 채워진다. 가만히 있으면 스레드에 의해 0.1초 간격으로 bar의 마젠타색을 1/100씩 감소시킨다. 키를 빨리 누르지 않으면 스레드의 감소 속도를 이기지 못한다. bar는 JLabel을 상속받은 MyLabel로 작성하고 MyLabel의 paintComponent() 메소드가 bar를 마젠타색으로 채우도록 하라.




초기 화면




키를 반복하여 빨리 누른 화면

# 예제 12-6 정답



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

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

    public 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 {
    private MyLabel bar;

    public ConsumerThread(MyLabel bar) {
        this.bar = bar;
    }
    @Override
    public void run() {
        while(true) {
            try {
                sleep(200);
                bar.consume();
            } catch (InterruptedException e)
                 { return; }
        }
    }
}
```

```
public class TabAndThreadEx  extends JFrame
{
    private MyLabel bar = new MyLabel(100);
    public 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.setFocusable(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");
    }
}
```