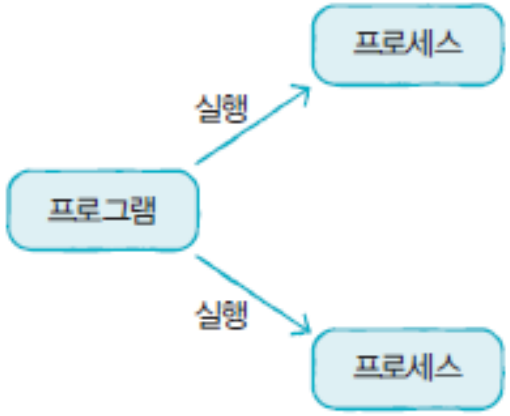
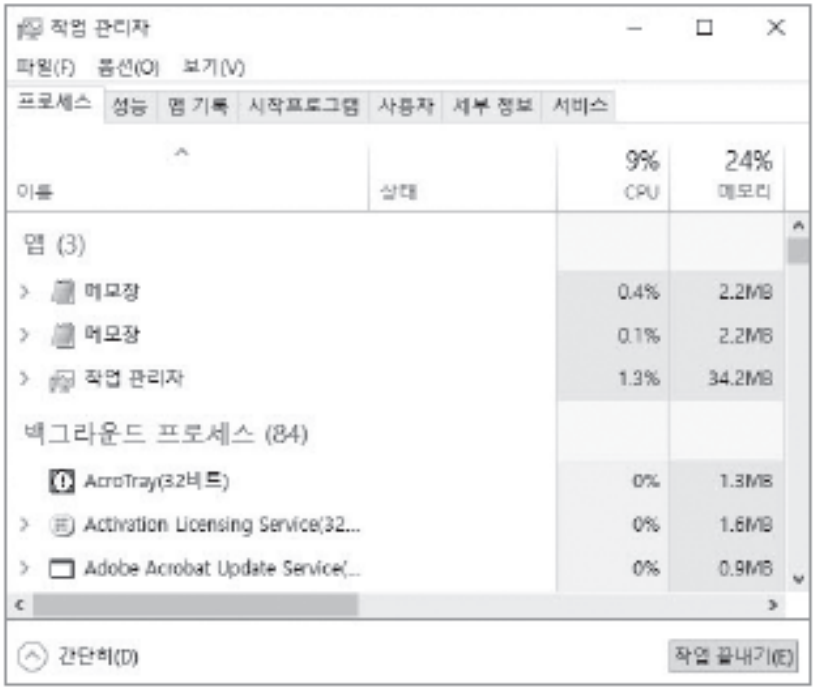
■ 스레드 (Thread)

※ 프로세스

- 실행 중인 하나의 애플리케이션
- 운영체제로부터 실행에 필요한 메모리를 할당받아 실행





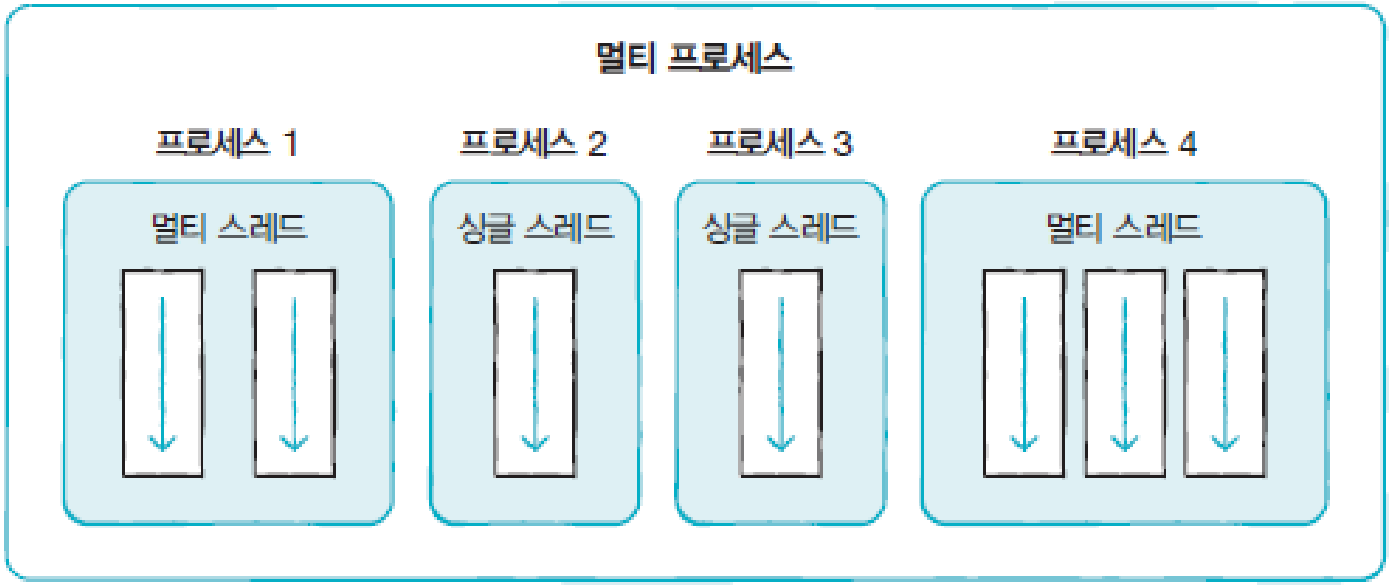
■ 스레드 (Thread)

● 한가지 작업을 실행하기 위해 순차적으로 실행되는 코드

● 1개의 스레드는 1개의 코드 실행

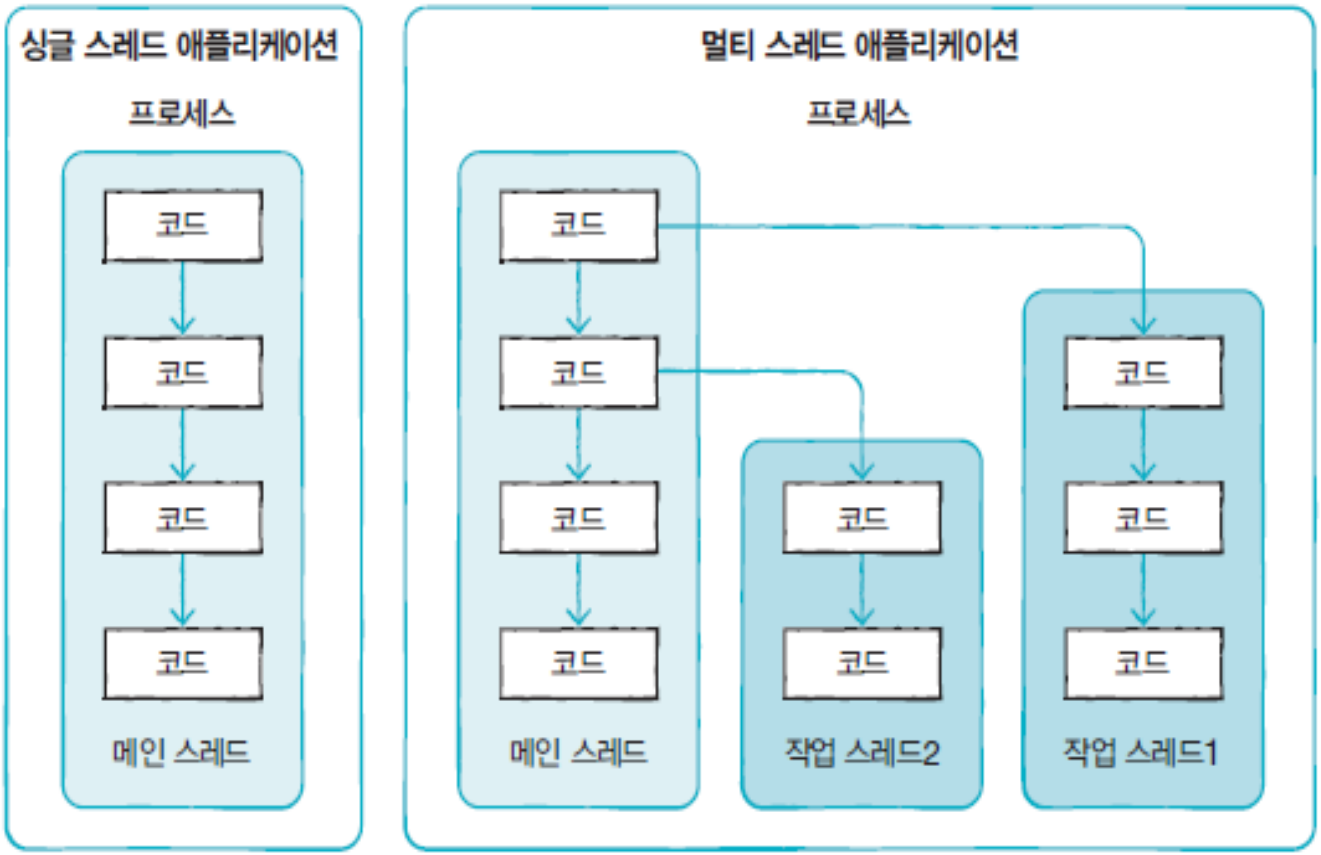
※ 멀티 스레드 (Multi Thread)

- 1개의 프로세스 내에서 2가지 이상의 작업을 처리
- 데이터를 분할하여 병렬처리, 다수의 요청을 처리하는 서버 등 개발 가능



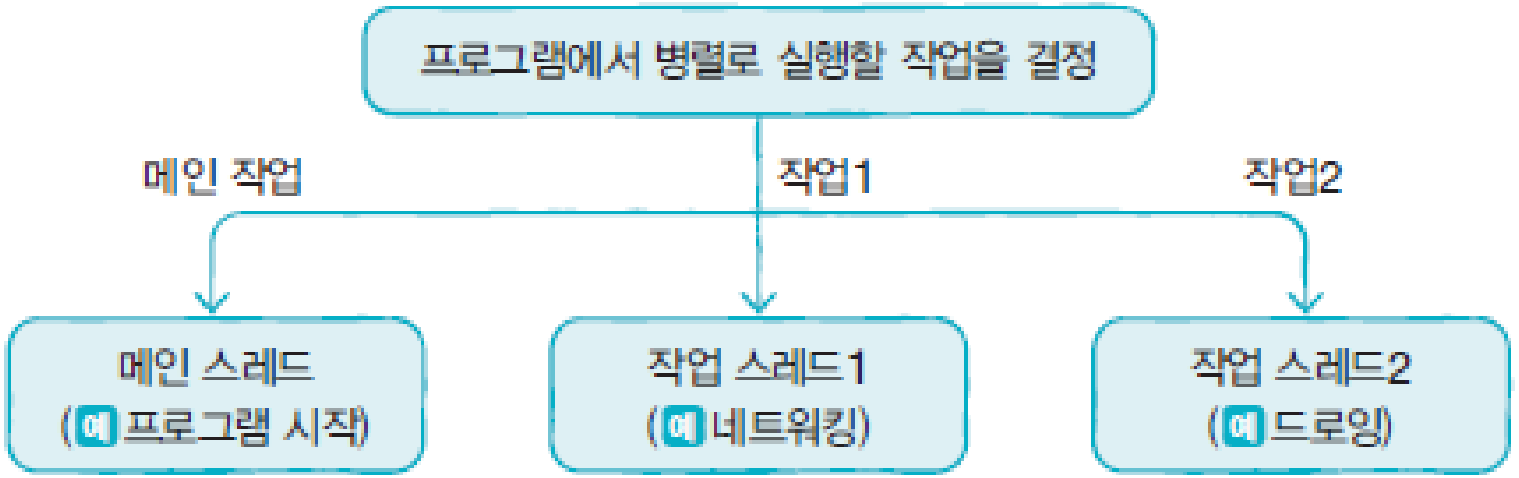
■ 메인 스레드 (Main Thread)

- main() 메소드가 실행되면 메인 스레드가 시작됨
- 필요에 따라 멀티 스레드를 만들어 병렬로 코드 실행 가능
- 멀티 스레드로 동작할 때 실행중인 스레드가 하나라도 있으면 프로세스는 종료되지 않음



■ 작업 스레드

- 멀티 스레드로 실행되는 애플리케이션을 개발하려면
  각 작업별 스레드를 생성해야 됨
- 작업 스레드는 Thread 클래스를 상속받는 클래스 형태로 작성

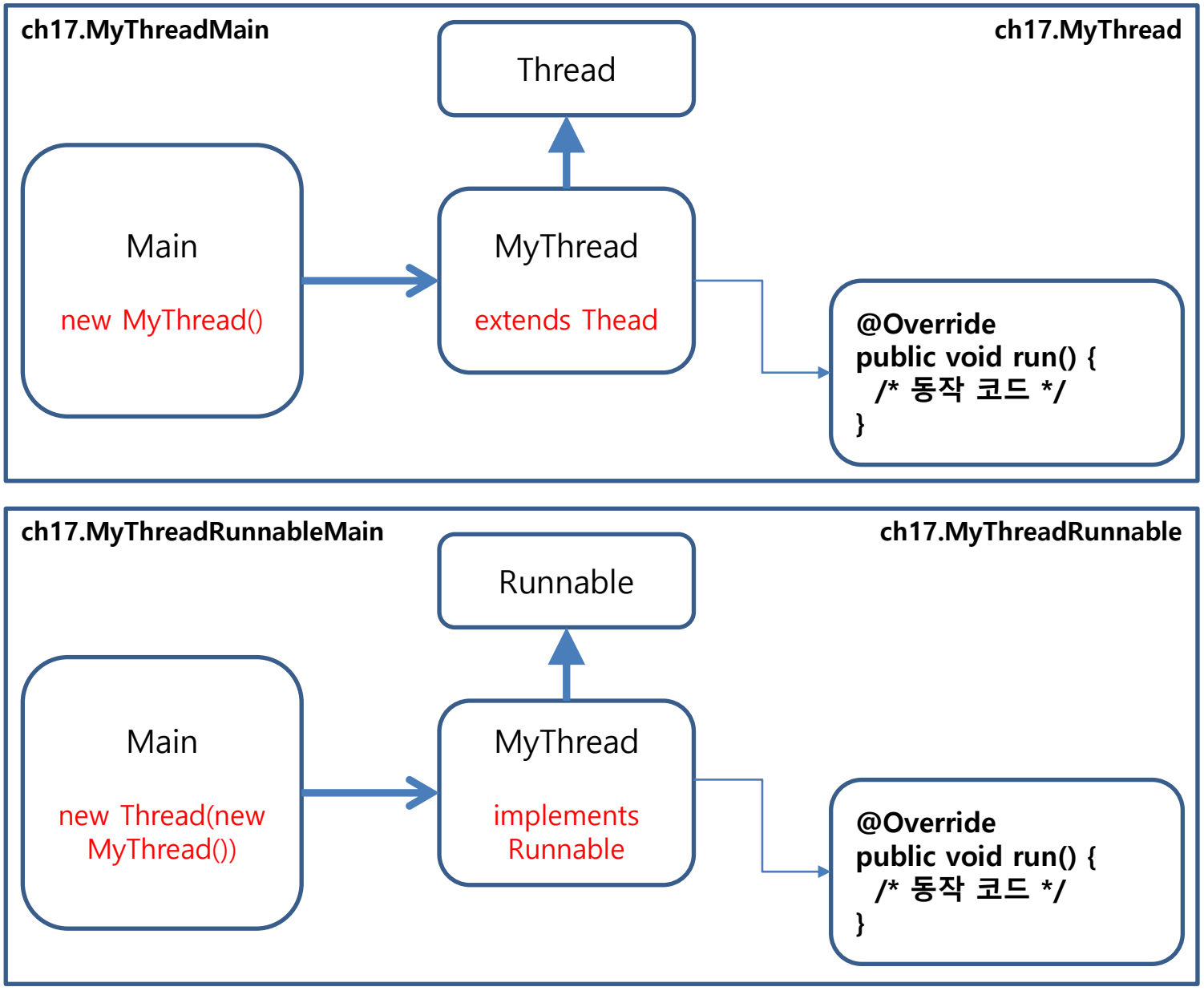



```
public class Main {
  public static void main(
    String[] args) { }
}
```

```
public class Networking
  extends Thread {

}
```

```
public class Drawing
  extends Thread {

}
```

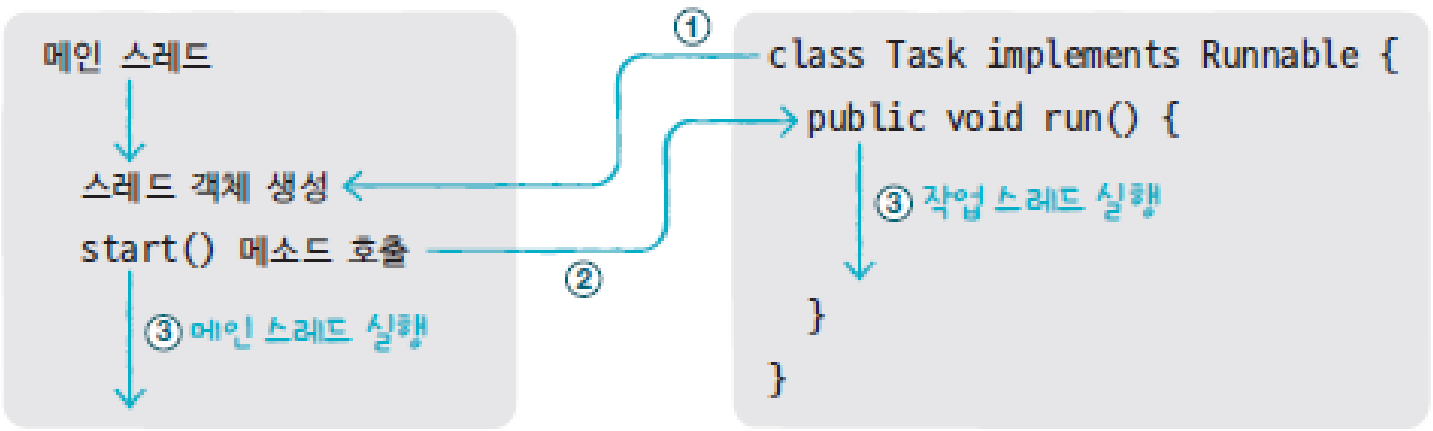
■ Thread 사용 (extends Thread / implements Runnable)



■ Thread 생성과 실행

- 객체 생성 (new OOO) 후 start() 메소드를 호출하면 작업 스레드 실행

```
thread.start();
```



■ Thread 사용 - 1 (extends Thread)

ch17.ThreadExam1

```
public class ThreadExam1 extends Thread {
  String name;

  ThreadExam1(String name) {
    this.name = name;
  }

  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println(name + " : " + i);
    }
  }
}
```

```
first : 0
first : 1
second : 0
first : 2
second : 1
second : 2
first : 3
second : 3
first : 4
```

ch17.ThreadExam1Main

```
ThreadExam1 t1 = new ThreadExam1("first");
ThreadExam1 t2 = new ThreadExam1("second");

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

■ Thread 사용 - 2 (implements Runnable)

ch17.ThreadExam2

```
public class ThreadExam2 implements Runnable {
  String name;

  ThreadExam1(String name) {
    this.name = name;
  }

  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println(name + " : " + i);
    }
  }
}
```

```
first : 0
second : 0
first : 1
first : 2
second : 1
first : 3
second : 2
first : 4
second : 3
```

ch17.ThreadExam2Main

```
ThreadExam2 t1 = new ThreadExam2("first");
ThreadExam2 t2 = new ThreadExam2("second");

Thread thread1 = new Thread(t1);
Thread thread2 = new Thread(t2);
thread1.start();
thread2.start();
```

■ Thread 상태 제어

- 주어진 시간 동안 일시 정지 (sleep)

```
try {
   Thread.sleep(1000);
} catch(InterruptedException  e) {
   // interrupt()  메소드가  호출되면  실행
}
```

  - 밀리 세컨드(1/1000) 단위로 지정

■ Thread 사용 - 3 (sleep)

ch17.ThreadExam3

```
public class ThreadExam3 extends Thread {
  @Override
  public void run() {
    for (int i = 1; i <= 10; i++) {
      try {
        System.out.println(i * 10 + "퍼센트 완료");
        sleep(500);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}
```

```
10퍼센트 완료
20퍼센트 완료
30퍼센트 완료
40퍼센트 완료
50퍼센트 완료
60퍼센트 완료
70퍼센트 완료
80퍼센트 완료
90퍼센트 완료
100퍼센트 완료
```

ch17.ThreadExam3Main

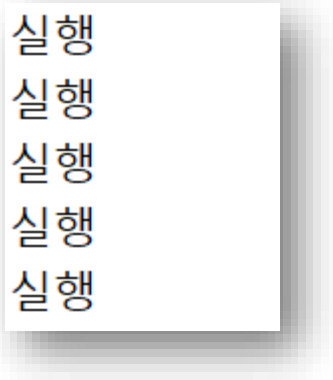
```
ThreadExam3 t = new ThreadExam3();
t.start();
```

■ 소리와 문자 출력 - Thread 미사용

ch17.BeepPrint1

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
for (int i = 0; i < 5; i++) {
  toolkit.beep();
  try {
    Thread.sleep(500);
  } catch (Exception e) {
    e.printStackTrace();
  }
}

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

비프음 5번 동작 후 문자 출력



■ 소리와 문자 출력 - Thread 사용

ch17.BeepPrint2

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
new Thread(new Runnable() {
  @Override
  public void run() {
    for (int i = 0; i < 5; i++) {
      toolkit.beep();
      try {
        Thread.sleep(500);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
}).start();

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

비프음 5번 동작과 함께 문자 출력

```
실행
실행
실행
실행
실행
```

■ 클릭된 위치에 이미지 생성 및 이동 - Thread 미사용

ch17.Balloon1

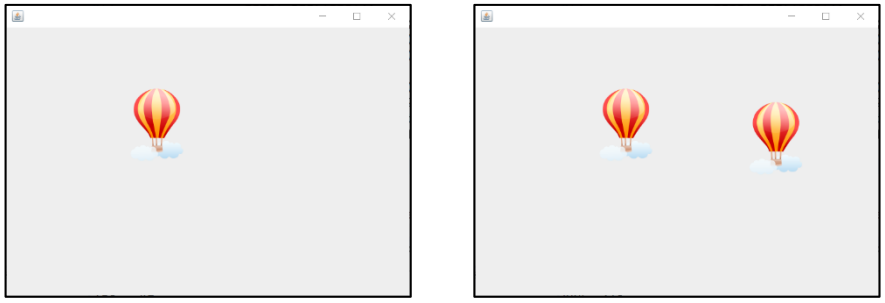
```
public static void makeBalloon(JPanel panel, int x, int y) {
  /* 이미지 생성 */
  ImageIcon icon = new ImageIcon("images/balloon.png");

  /* 이미지를 보여주기 위해서 JLabel 사용 */
  JLabel label = new JLabel(icon);

  /* 이미지 크기와 같게 JLabel 크기 지정 */
  label.setSize(icon.getIconWidth(), icon.getIconHeight());

  /* 이미지를 가지고 있는 JLabel을 JPanel로 추가 */
  panel.add(label);
  for(int i = 0; i < 10; i++) {
    /* 이미지가 보여질 위치 지정 */
    label.setLocation(x, y);
    try {
      /* 그래픽 표현 상태 변경 후 새로 고침을 해야 화면에 반영 */
      panel.repaint();
      Thread.sleep(200);
    } catch (InterruptedException e1) {
      e1.printStackTrace();
    }
    /* Y축 위치 변경 (위로 이동) */
    y -= 25;
  }
}
```

■ 클릭된 위치에 이미지 생성 및 이동 - Thread 미사용

ch17.Balloon1Main

```
JFrame j = new JFrame();
JPanel panel = new JPanel();
panel.addMouseListener(new MouseAdapter() {
  @Override
  public void mousePressed(MouseEvent e) {
    Balloon1.makeBalloon(panel, e.getX(), e.getY());
  }
});
j.add(panel);
j.setSize(700, 500);
j.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
j.setVisible(true);
```

클릭하여 이미지 생성 및 이동 완료 후 다음 동작 가능



■ 클릭된 위치에 이미지 생성 및 이동 - Thread 사용

ch17.Balloon2

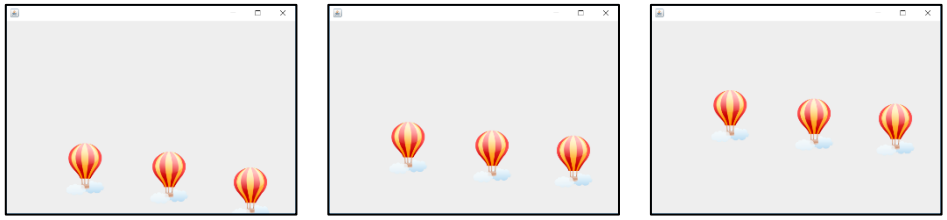
```
public static void makeBalloon(JPanel panel, int x, int y) {
  /* Thread 사용 */
  new Thread(new Runnable() {
    @Override
    public void run() {
      /* 내부 클래스에서 지역변수 사용 */
      int y2 = y;
      ImageIcon icon = new ImageIcon("images/balloon.png");
      JLabel label = new JLabel(icon);
      label.setSize(icon.getIconWidth(), icon.getIconHeight());
      panel.add(label);
      for (int i = 0; i < 10; i++) {
        label.setLocation(x, y2);
        try {
          panel.repaint();
          Thread.sleep(200);
        } catch (InterruptedException e1) {
          e1.printStackTrace();
        }
        y2 -= 25;
      }
    }
  }).start();
}
```

■ 클릭된 위치에 이미지 생성 및 이동 - Thread 사용

ch17.Balloon2Main

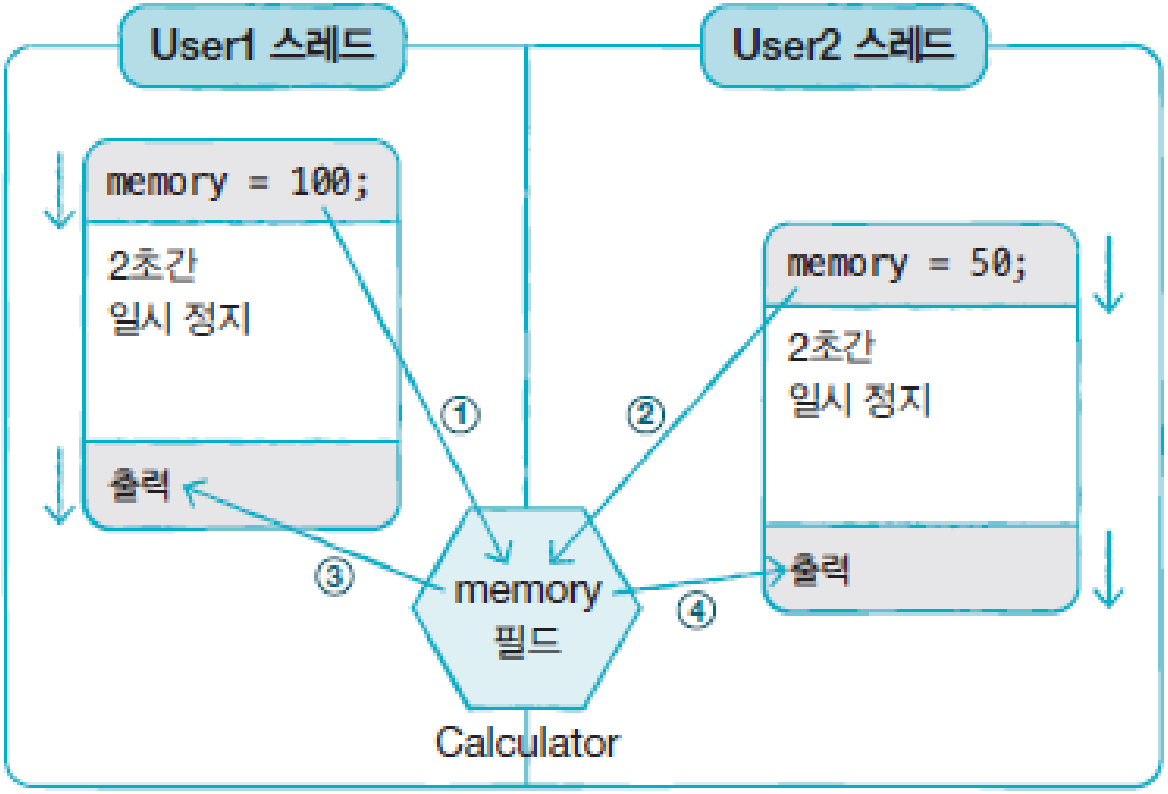
```
JFrame j = new JFrame();
JPanel panel = new JPanel();
panel.addMouseListener(new MouseAdapter() {
  @Override
  public void mousePressed(MouseEvent e) {
    Balloon2.makeBalloon(panel, e.getX(), e.getY());
  }
});
j.add(panel);
j.setSize(700, 500);
j.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
j.setVisible(true);
```

이전 작업과 상관없이 다음 동작 실행



■ Thread 동기화

- Thread는 하나의 자원(객체)를 공유하는 경우 주의



■ Thread 동기화 - 1

ch17.Sync1Main

```
Sync1Value value = new Sync1Value();

Sync1UserA userA = new Sync1UserA();
userA.setSyncValue(value);
userA.start();

Sync1UserB userB = new Sync1UserB();
userB.setSyncValue(value);
userB.start();
```

ch17.Sync1Value

```
private int memory;

public int getMemory() { return memory; }

public void setMemory(int memory) {
  this.memory = memory;
  try {
    Thread.sleep(2000);
  } catch(InterruptedException e) { e.printStackTrace(); }
  System.out.println(
      Thread.currentThread().getName() + " : " + this.memory);
}
```

■ Thread 동기화 - 1

ch17.Sync1UserA

```
public class Sync1UserA extends Thread {
  private Sync1Value syncValue1;

  public void setSyncValue(Sync1Value syncValue1) {
    this.setName("User1");
    this.syncValue1 = syncValue1;
  }

  @Override
  public void run() { syncValue1.setMemory(100); }
}
```

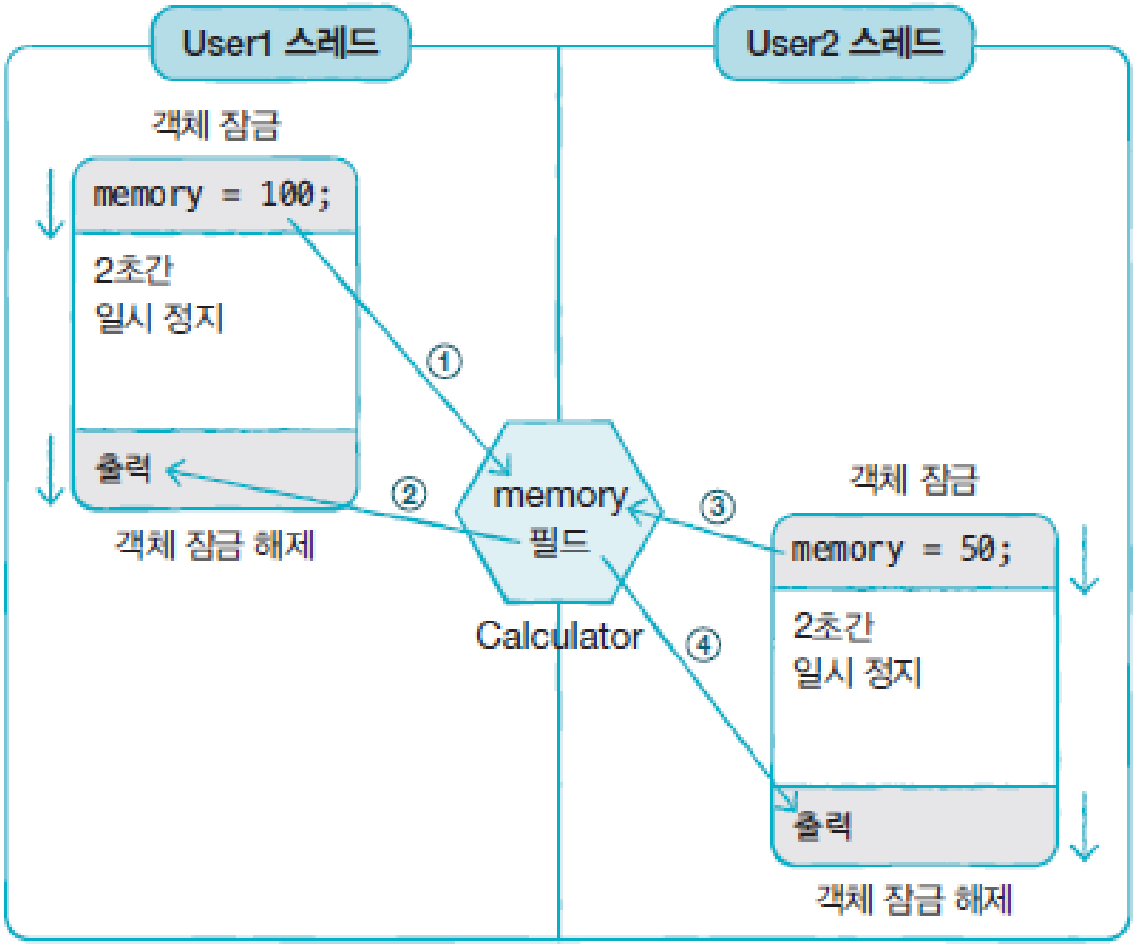
ch17.Sync1UserB

```
public class Sync1UserB extends Thread {
  private Sync1Value syncValue1;

  public void setSyncValue(Sync1Value syncValue1) {
    this.setName("User2");
    this.syncValue1 = syncValue1;
  }

  @Override
  public void run() { syncValue1.setMemory(50); }
}
```

```
User1 : 50
User2 : 50
```

■ Thread 동기화

● 하나의 Thread가 사용중인 객체는 다른 Thread가 접근할 수 없도록 잠금



■ Thread 동기화 - 2

ch17.Sync2Main

```
Sync2Value value = new Sync2Value();

Sync2UserA userA = new Sync2UserA();
userA.setSyncValue(value);
userA.start();

Sync2UserB userB = new Sync2UserB();
userB.setSyncValue(value);
userB.start();
```

ch17.Sync2Value

```
private int memory;

public int getMemory() { return memory; }

public synchronized void setMemory(int memory) {
  this.memory = memory;
  try {
    Thread.sleep(2000);
  } catch(InterruptedException e) { e.printStackTrace(); }
  System.out.println(
      Thread.currentThread().getName() + " : " + this.memory);
}
```

■ Thread 동기화 - 2

ch17.Sync2UserA

```
public class Sync2UserA extends Thread {
  private Sync2Value syncValue2;

  public void setSyncValue(Sync2Value syncValue2) {
    this.setName("User1");
    this.syncValue2 = syncValue2;
  }

  @Override
  public void run() { syncValue2.setMemory(100); }
}
```

ch17.Sync2UserB

```
public class Sync2UserB extends Thread {
  private Sync2Value syncValue2;

  public void setSyncValue(Sync2Value syncValue2) {
    this.setName("User2");
    this.syncValue2 = syncValue2;
  }

  @Override
  public void run() { syncValue2.setMemory(50); }
}
```

```
User1 : 100
User2 : 50
```

■ Thread 동기화 - 3

ch17.WashRoomMain

```
WashRoom room = new WashRoom();
WashRoomThread father = new WashRoomThread("father", room);
WashRoomThread mother = new WashRoomThread("mother", room);
WashRoomThread syster = new WashRoomThread("syster", room);
WashRoomThread brother = new WashRoomThread("brother", room);

father.start();
mother.start();
syster.start();
brother.start();
```

ch17.WashRoom

```
public synchronized void useRoom(String name) {
  System.out.println(name + " 입장");
  try {
    System.out.println(name + " 사용중");
    int random = new Random().nextInt(3) + 1;
    Thread.sleep(random * 1000);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  System.out.println(name + " 퇴장");
}
```

■ Thread 동기화 - 3

ch17.WashRoomThread

```
public class WashRoomThread extends Thread {
  private WashRoom room;
  private String name;

  public WashRoomThread(String name, WashRoom room) {
    this.name = name;
    this.room = room;
  }

  @Override
  public void run() {
    room.useRoom(name);
  }
}
```

```
father 입장
father 사용중
father 퇴장
brother 입장
brother 사용중
brother 퇴장
syster 입장
syster 사용중
syster 퇴장
mother 입장
mother 사용중
mother 퇴장
```