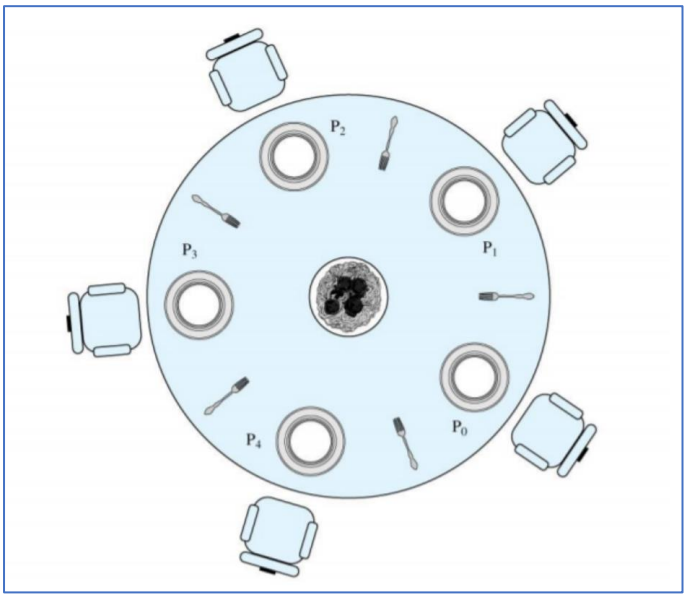
JAVA프로그래밍

식사하는철학자, 과제3(8.4, 8.5, 8.6(3))

2019305059

이현수

과제2 : 식사하는 철학자 문제



식사하는 철학자 문제

철학자 다섯이서 원형 식탁에 둘러앉아 생각에 빠지다가, 배고플 땐 밥을 먹는다. 그들의 양쪽엔 각각 젓가락 한 짠씩 놓여있고, 밥을 먹으려 할 때 다음의 과정을 따른다.

1. 왼쪽 젓가락부터 집어든다. 다른 철학자가 이미 왼쪽 젓가락을 쓰고 있다면 그가 내려놓을 때까지 생각하며 대기한다.
2. 왼쪽을 들었으면 오른쪽 젓가락을 든다. 들 수 없다면 1번과 마찬가지로 들 수 있을 때까지 생각하며 대기한다.
3. 두 젓가락을 모두 들었다면 일정 시간동안 식사를 한다.
4. 식사를 마쳤으면 오른쪽 젓가락을 내려놓고, 그 다음 왼쪽 젓가락을 내려놓는다.
5. 다시 생각하다가 배고프면 1번으로 돌아간다.

발생할 수 있는 문제

식사하는 철학자 문제는 **교착상태**의 대표적인 예제. 위의 상황을 프로그램으로 만들어 실행하면 잘 돌아가다가 어느순간 멈춰버림. 만약 모든 철학자가 동시에 배가 고파서 왼쪽 젓가락을 집어든다면 오른쪽 젓가락은 이미 자신의 우측에 앉은 철학자가 집어들었을 것이므로, 모두가 영원히 오른쪽 젓가락을 들지 못하게 된다. 그렇게 상단 과정의 2번에서 더이상 진행하지 못하고 철학자들은 영원히 생각에 빠지게 되는데, 이런 현상을 컴퓨터과학에선 교착상태(Deadlock)라고 한다.

또 포크를 집는 동작 혹은 놓는 동작이 여러 스레드가 동시에 접근하게 되면 공유자원을 의도치 않게 잘못된 동작을 하게 되는 경우가 있어서 스레드가 특정 동작에 있어서 특정 시간에 한 개의 스레드만 접근가능하게 만들어야 한다. 즉 임계영역을 만들어야 한다. 즉 **동기화문제**가 있다.

포크 = **공유자원**
철학자 = **프로세스**

해결방법

철학자 문제에서 젓가락에 1부터 5번까지 번호를 부여하고, 철학자들은 가장 낮은 번호를 우선 취하도록 강요하는 것이다. 1과 2번 젓가락 사이에 있는 철학자와 1과 5번 사이에 있는 철학자는 그의 오른쪽에 있는 젓가락을 잡는 것이 아니라 같은 젓가락 1번을 우선 잡아야 한다. 젓가락 1을 잡지 못한 철학자는 먼저 잡은 철학자가 젓가락을 놓을 때까지 기다려야 한다. 이러한 상황에서 데드락은 발생하지 않게 된다.

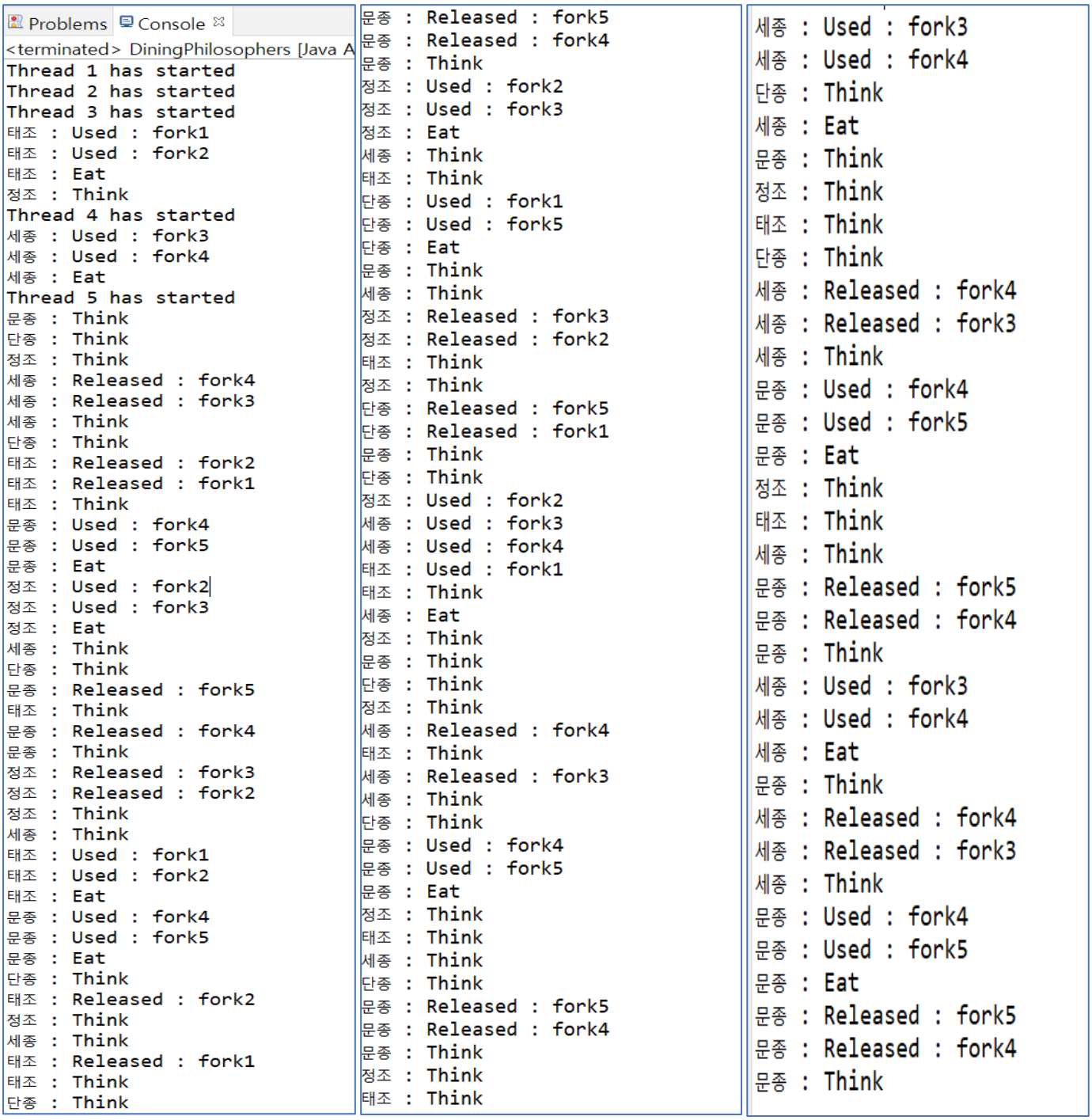
동기화 문제를 해결하기 위해서 `synchronized` 키워드를 사용해서 해결한다.

<코드>

```
class Philosopher extends Thread {
    private Fork leftfork;
    private Fork rightfork;
    private String name;
    private int rounds;

    public Philosopher ( String name, Fork left, Fork right, int rounds){
        this.name = name;
        leftfork = left;
        rightfork = right;
        this.rounds = rounds;
    }

    public void eat()
    {
        if(! leftfork.used){
            if(!rightfork.used){
                int temp1 = leftfork.take(name);
                if(temp1==0) {
                    think();
                    return;
                }
                int temp2 = rightfork.take(name);
                if(temp2==0) {
                    think();
                    return;
                }
                Log.msg(name + " : Eat");
                Log.Delay(1000);

                rightfork.release(name);
                leftfork.release(name);
            }
        }
        think();
    }

    public void think(){
            Log.msg(name + " : Think");
            Log.Delay(1000);
    }

    public void run(){
        for(int i=0; i<=rounds; i++){
            eat();
        }
    }
}

class Fork{
    public boolean used;
    public String fname;

    public Fork(String name){
        this.fname = name;
    }

    public synchronized int take(String name) {
        if(this.used) return 0;
        Log.msg (name+" : Used : " + fname );
        this.used = true;
        return 1;
    }

    public synchronized void release(String name) {
        Log.msg (name + " : Released : " + fname );
        this.used = false ;
    }
}

class Log{
    public static void msg(String msg){
        System.out.println(msg);
    }

    public static void Delay(int ms){
        try{
            Thread.sleep(ms);
        }
        catch(InterruptedException ex){ }
    }
}

```

```java
public class DiningPhilosophers{
    public static void main(String[] args){

        int rounds=10;

        Fork[] forks = new Fork[5];

        //initlize the forks
        for(int i=0; i< forks.length; i++){
            forks[i] = new Fork("fork"+(i+1));
        }

        Philosopher[] philosophers = new Philosopher[5];

        philosophers[0] = new Philosopher("태조", forks[0], forks[1], rounds);
        philosophers[1] = new Philosopher("정조", forks[1], forks[2], rounds);
        philosophers[2] = new Philosopher("세종", forks[2], forks[3], rounds);
        philosophers[3] = new Philosopher("문종", forks[3], forks[4], rounds);
        philosophers[4] = new Philosopher("단종", forks[0], forks[4], rounds);

        for(int i=0;i<philosophers.length;i++){
            Log.msg("Thread "+ (i+1) + " has started");
            Thread t= new Thread( philosophers[i]);
            t.start();
        }
    }
}
```

<실행결과>



```
Problems  Console
<terminated> DiningPhilosophers [Java A
Thread 1 has started
Thread 2 has started
Thread 3 has started
태조 : Used : fork1
태조 : Used : fork2
태조 : Eat
정조 : Think
Thread 4 has started
세종 : Used : fork3
세종 : Used : fork4
세종 : Eat
Thread 5 has started
문종 : Think
단종 : Think
정조 : Think
세종 : Released : fork4
세종 : Released : fork3
세종 : Think
단종 : Think
태조 : Released : fork2
태조 : Released : fork1
태조 : Think
문종 : Used : fork4
문종 : Used : fork5
문종 : Eat
정조 : Used : fork2
정조 : Used : fork3
정조 : Eat
세종 : Think
단종 : Think
문종 : Released : fork5
태조 : Think
문종 : Released : fork4
문종 : Think
정조 : Released : fork3
정조 : Released : fork2
정조 : Think
세종 : Think
태조 : Used : fork1
태조 : Used : fork2
태조 : Eat
문종 : Used : fork4
문종 : Used : fork5
문종 : Eat
단종 : Think
태조 : Released : fork2
정조 : Think
세종 : Think
태조 : Released : fork1
태조 : Think
단종 : Think
```

```
문종 : Released : fork5
문종 : Released : fork4
문종 : Think
정조 : Used : fork2
정조 : Used : fork3
정조 : Eat
세종 : Think
태조 : Think
단종 : Used : fork1
단종 : Used : fork5
단종 : Eat
문종 : Think
세종 : Think
정조 : Released : fork3
정조 : Released : fork2
태조 : Think
정조 : Think
단종 : Released : fork5
단종 : Released : fork1
문종 : Think
단종 : Think
정조 : Used : fork2
세종 : Used : fork3
세종 : Used : fork4
태조 : Used : fork1
태조 : Think
세종 : Eat
정조 : Think
문종 : Think
단종 : Think
정조 : Think
세종 : Released : fork4
태조 : Think
세종 : Released : fork3
세종 : Think
단종 : Think
문종 : Used : fork4
문종 : Used : fork5
문종 : Eat
정조 : Think
태조 : Think
세종 : Think
단종 : Think
문종 : Released : fork5
문종 : Released : fork4
문종 : Think
정조 : Think
태조 : Think
```

```
세종 : Used : fork3
세종 : Used : fork4
단종 : Think
세종 : Eat
문종 : Think
정조 : Think
태조 : Think
단종 : Think
세종 : Released : fork4
세종 : Released : fork3
세종 : Think
문종 : Used : fork4
문종 : Used : fork5
문종 : Eat
정조 : Think
태조 : Think
세종 : Think
문종 : Released : fork5
문종 : Released : fork4
문종 : Think
세종 : Used : fork3
세종 : Used : fork4
세종 : Eat
문종 : Think
세종 : Released : fork4
세종 : Released : fork3
세종 : Think
문종 : Used : fork4
문종 : Used : fork5
문종 : Eat
문종 : Released : fork5
문종 : Released : fork4
문종 : Think
```

Eat 순서 : 태조, 세종, 문종, 정조, 태조, 문종, 정조, 단종, 세종, 문종, 세종, 문종, 세종, 문종

실행결과 태조, 정조, 세종, 문종, 단종 모두 최소 한번씩 식사를 함.

<설명>

```
1  class Philosopher extends Thread {
2      private Fork leftfork;
3      private Fork rightfork;
4      private String name;
5      private int rounds;
6
7      public Philosopher ( String name, Fork left, Fork right, int rounds){
8          this.name = name;
9          leftfork = left;
10         rightfork = right;
11         this.rounds = rounds;
12     }
13
14     public void eat()
15     {
16         if(! leftfork.used){
17             if(!rightfork.used){
18                 int temp1 = leftfork.take(name);
19                 if(temp1==0) {
20                     think();
21                     return;
22                 }
23                 int temp2 = rightfork.take(name);
24                 if(temp2==0) {
25                     think();
26                     return;
27                 }
28                 Log.msg(name + " : Eat");
29                 Log.Delay(1000);
30
31                 rightfork.release(name);
32                 leftfork.release(name);
33             }
34         }
35         think();
36     }
37
38     public void think(){
39             Log.msg(name + " : Think");
40             Log.Delay(1000);
41     }
42
43     public void run(){
44         for(int i=0; i<=rounds; i++){
45             eat();
46         }
47     }
48 }
49
```

철학자를 의미하는 Philosopher 클래스를 만든다. 이 클래스는 Thread를 상속받아 스레드를 구현해주게 된다. Philosopher클래스는 총 4개의 필드를 가진다. Fork클래스 leftfork, rightfork를 필드로 가진다. 각각 철학자 기준 왼쪽, 오른쪽에 놓여져 있는 포크를 의미한다. String형 name필드는 철학자의 이름을 의미한다. 정수형 변수 rounds는 eat을 시도하는 횟수라고 보면된다.

(코드7~12) Philosopher의 매개변수 있는 생성자를 만든다. 그래서 4개의 필드를 모두 초기화 하는 작업을 한다.

(코드14~36) eat메소드에서는 이중 조건문을 통해 철학자의 왼쪽포크와 오른쪽포크가 사용중인지 체크한다. 만약 한 개라도 사용중 이라면 조건문 조건이 만족을 못해 think메소드가 실행된다. 만약 철학자가 식사하기 위한 왼쪽 포크와 오른쪽 포크 두개 모두 사용중이 아니라면 이중 조건문 조건이 만족이되 조건문 안의 코드가 실행된다. 왼쪽포크와 오른쪽포크 순서대로 take메소드를 통해 사용하고 1초동안 delay후 오른쪽, 왼쪽 포크 순서대로 release하게 된다. 근데 여기에 문제가 발생하게 된다. 스레드가 동시에 실행되기 때문에 식사를 하기위에 필요한 포크가 겹치는 철학자 둘이 동시에 이 조건문이 실행되 둘다 조건문의 조건이 만족이되 이중조건문 안의 코드를 동시에 실행된다고 했을 때 두개의 철학자가 동시에 같은 포크를 사용하는 오류가 발생하게 된다. 그래서 이때 take를 할 때 temp1, temp2 정수형 변수에 값을 반환받아 반약 변수에 0이 반환되면 누군가가 이미 이포크를 선점했다는 것을 의미해 think메소드를 호출하고 return;해 함수를 종료하게 된다.

(코드38~41) 철학자가 eat을 하지 못한경우 think를 하게된다. Log클래스의 msg메소드를 통해 관련 내용을 알려주고 Delay메소드를 통해 1초동안 해당 스레드를 sleep한다. 이것의 의미는 1초 후에 먹는걸 다시 시도한다는 의미이다.

(코드43~47) thread를 상속받았다. 그래서 run메소드를 오버라이딩해 구현해준다. for반복문을 통해 rounds+1만큼 eat메소드를 호출한다. 즉 rounds+1만큼 eat을 시도하는 것이다.

```
50 class Fork{
51     public boolean used;
52     public String fname;
53 
54     public Fork(String name){
55         this.fname = name;
56     }
57 
58     public synchronized int take(String name) {
59         if(this.used) return 0;
60         Log.msg (name+" : Used : " + fname );
61         this.used = true;
62         return 1;
63     }
64 
65     public synchronized void release(String name) {
66         Log.msg (name + " : Released : " + fname );
67         this.used = false ;
68     }
69 }
70 
```

Fork클래스는 포크를 의미한다. 여기서는 두개의 필드를 가진다. boolean자료형의 used 필드다. used는 포크가 철학자에 의해 사용되고 있냐 즉 사용자가 이 포크를 먹는 행위를 하기위해 들고있어나 들고 음식을 먹느냐의 의미이다. 그래서 그럴경우 true이고 그 상태가 아니면 false상태가 된다. String형 fname필드는 포크의 이름을 의미한다.

(코드54~56) Fork클래스의 매개변수 있는 생성자를 구현한다. String형 매개변수를 받아 fname필드를 초기화한다.

(코드58~63) public이고 int형을 받환하는 take메소드를 구현한다. 이 메소드는 String형 변수를 매개변수로 받는다. take함수는 이 포크를 매개변수로 받은 name이름을 가진 철학자가 사용을 한다는 동작을 의미한다. 그래서 메소드 안에서 used가 true인 경우 0을 리턴한다. 즉 이 메소드에 들어와서 포크가 사용중일경우 실패했다는 의미로 0을 리턴하고 그것이 아니면 used를 true로 만들고 성공의 의미로 1을 리턴한다. 이때 Log클래스에 있는 msg메소드를 사용해 name이름을 가진 철학자가 fname을 가진 포크를 take한다는 것을 출력해준다. 이때 여러 쓰레드 중 단 한개의 스레드만 이 함수를 실행할 수 있다는 동기화를 보장하기 위해서 synchronized 키워드를 사용해준다.

(코드65~68) release함수 역시 동기화를 보장하기 위해서 synchronized 키워드를 사용한다. 이함수는 public으로 take메소드와 다르게 void 즉 아무것도 반환하지 않고, String형 변수를 통해 행위를 하는 철학자 이름을 매개변수로 넘겨 받는다. 역시 Log클래스의 msg메소드를 사용해 한 행위를 출력해주고 used필드를 false로 바꾼다.

```
71 class Log{
72     public static void msg(String msg){
73         System.out.println(msg);
74     }
75 
76     public static void Delay(int ms){
77         try{
78             Thread.sleep(ms);
79         }
80         catch(InterruptedException ex){ }
81     }
82 }
83 
```

클래스 Log는 철학자, 포크 클래스에서 일어나는 일들을 출력하는 역할을 한다. 그래서 Log클래스는 두개의 메소드가 구현되있다. msg메소드와 Delay메소드이다. msg메소드는 매개변수로 받은 String형 msg를 출력해주는 역할을 하고 Delay메소드는 매개변수로 받은 정수형변수 ms만큼 Thread.sleep를 통해 sleep해준다. 이때 try블록에 구현이 되있고 예외를 처리하기 위해 catch블록 역시 구현되있다.

여기서 두개의 메소드 모두 public static 으로 구현되있다. 그래서 Log클래스 내에 있는 두개의 메소드를 사용할 때 Log클래스의 객체를 생성하지 않고 Log.메소드이름(...)형태로 사용하게 된다.

```
84 public class DiningPhilosophers{
85     public static void main(String[] args){
86 
87         int rounds=10;
88 
89         Fork[] forks = new Fork[5];
90 
91         //initlize the forks
92         for(int i=0; i< forks.length; i++){
93             forks[i] = new Fork("fork"+(i+1));
94         }
95 
96         Philosopher[] philosophers = new Philosopher[5];
97 
98         philosophers[0] = new Philosopher("태조", forks[0], forks[1], rounds);
99         philosophers[1] = new Philosopher("정조", forks[1], forks[2], rounds);
100        philosophers[2] = new Philosopher("세종", forks[2], forks[3], rounds);
101        philosophers[3] = new Philosopher("문종", forks[3], forks[4], rounds);
102        philosophers[4] = new Philosopher("단종", forks[0], forks[4], rounds);
103 
104        for(int i=0;i<philosophers.length;i++){
105            Log.msg("Thread "+ (i+1) + " has started");
106            Thread t= new Thread( philosophers[i]);
107            t.start();
108        }
109    }
110 }
```

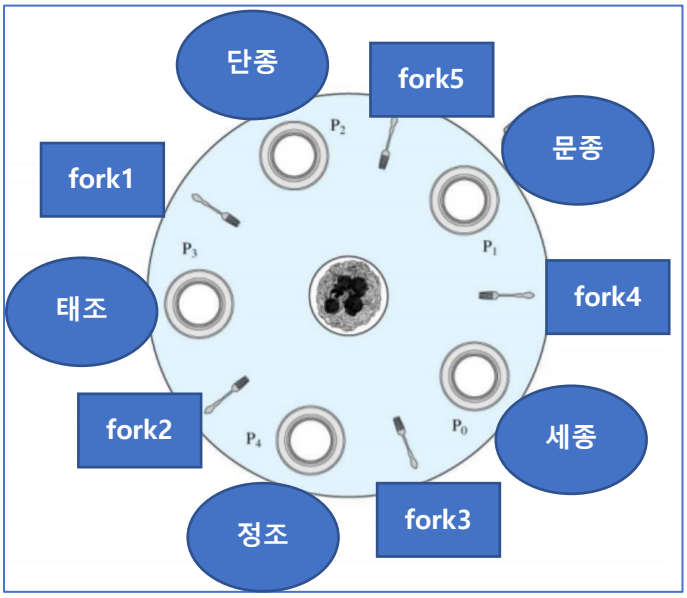
main함수에 rounds를 정해준다. 여기서는 10으로 해준다.

(코드89) 포크 5개가 필요하다. 그래서 Fork클래스 객체 5개를 배열로 만들어준다.

(코드92~94) 포크를 초기화하는 for반복문을 만든다. Forks를 인덱스로 접근해 5개의 객체에 대해서 포크이름

(코드96) 철학자를 의미하는 Philosopher객체 5개를 배열로 만들어야 한다.

(코드98~102) 5개의 철학자 객체를 실제로 생성한다. 객체를 생성할 때 매개변수로 넘겨줘야 할 것들은 철학자의 이름과 왼쪽 포트, 오른쪽포크의 각각의 객체, rounds 수이다. 여기서 철학자의 이름은 조선시대 왕이름으로 대신했다. 그리고 왼쪽, 오른쪽 포트는 아래 그림과 같이 설정해 주었다. 다만 마지막 철학자에게는 왼쪽포크와 오른쪽 포크의 위치를 바꾸어 주었다. 그 이유는 교착생태를 막기위함이다.



(코드104~108) for반복문을 통해 5개의 철학자 객체의 쓰레드를 실행시킨다. 이때 Log클래스의 msg메소드를 활용해 스레드가 실행된 다는 것을 알려준다.

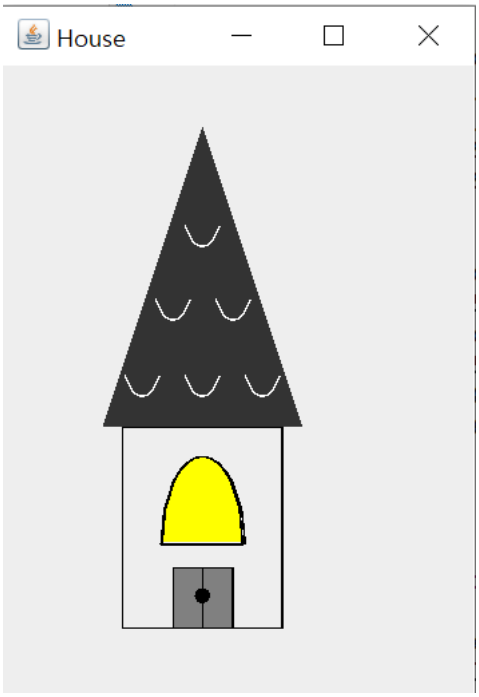
<문제>8.4

다음과 같은 그림이 되도록 본문에 있는 [예제 8.5]를 수정하시오.

<코드>

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

class HouseDrawingComponent extends JComponent{
    public void paintComponent(Graphics g) {
        //지붕그리기
        int[] xPnts = {100,50,150,100};
        int[] yPnts = {30,180,180,30};
        int nPnts = xPnts.length;
        g.fillPolygon(xPnts, yPnts, nPnts);

        //지붕 위의 기와 모양을 그린다
        g.setColor(Color.WHITE);
        g.drawArc(90, 50, 20, 40, 210, 120);
        g.drawArc(75, 87, 20, 40, 210, 120);
        g.drawArc(105, 87, 20, 40, 210, 120);
        g.drawArc(60, 125, 20, 40, 210, 120);
        g.drawArc(90, 125, 20, 40, 210, 120);
        g.drawArc(120, 125, 20, 40, 210, 120);

        //몸채를 그린다
        g.setColor(Color.BLACK);
        g.drawRect(60,180,80,100);

        //반원 모양의 창문을 그린다.
        g.setColor(Color.YELLOW);
        g.fillArc(80,195,40,85,0,180);

        //노란색 창문 주위를 2개의 검은색 선으로 그린다.
        g.setColor(Color.BLACK);
        g.drawArc(80, 195, 40, 85, 0, 180);
        g.drawArc(79,195,42,86,0,180);
        g.drawLine(79, 238, 120, 238);
        g.drawLine(79, 239, 120, 239);

        //문을 그린다.
        g.setColor(Color.GRAY);
        g.fillRect(85, 250, 30, 30);
        g.setColor(Color.BLACK);
        g.drawRect(85, 250, 30, 30);
        g.drawLine(100, 250, 100, 280);
        g.fillOval(96, 260, 8, 8);
    }
}

public class ch8_4 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(250,350);
        frame.setTitle("House");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        HouseDrawingComponent component = new HouseDrawingComponent();
        frame.add(component);
        frame.setVisible(true);
    }
}
```

<실행결과>



<설명>

JComponent를 상속받는 HouseDrawingComponent클래스를 만든다. 도형을 그리기 위해서는 반드시 Graphics클래스 객체가 필요하며 이것을 그래픽 객체라고 부른다. 그래픽 객체는 JComponent 클래스를 확장하여 얻을 수 있다. 여기서 paintComponent 메소드는 화면을 다시 그려야할 때 자바에 의해서 자동으로 호출되는 메소드로 매개변수에 그래픽 객체가 전달된다. 그래서 paintComponent메소드에 그리기 작업을 하면 화면에 나타난다. 그래서 paintComponent메소드를 재정의한다.

```
6           //지붕그리기
7           int[] xPnts = {100,50,150,100};
8           int[] yPnts = {30,180,180,30};
9           int nPnts = xPnts.length;
10          g.fillPolygon(xPnts, yPnts, nPnts);
11
```

삼각형의 지붕을 그려주는 코드이다. 지붕을 그리기 위한 x좌표와 y좌표 4개를 각각 xPnts와 yPnts 배열에 저장한다. 그리고 nPnts는 xPnts 배열의 길이 즉 4가 저장되게 된다. 그리고 fillPolygon에 매개변수로 xPnts, yPnts, nPnts를 넣어주어 지붕을 그려주게 된다.

```
12          //지붕 위의 기와 모양을 그린다
13          g.setColor(Color.WHITE);
14          g.drawArc(90, 50, 20, 40, 210, 120);
15          g.drawArc(75, 87, 20, 40, 210, 120);
16          g.drawArc(105, 87, 20, 40, 210, 120);
17          g.drawArc(60, 125, 20, 40, 210, 120);
18          g.drawArc(90, 125, 20, 40, 210, 120);
19          g.drawArc(120, 125, 20, 40, 210, 120);
20
```

검정색 지붕에 흰색으로 표시된 모양을 그리기 위한 코드이다. 먼저 setColor메소드를 통해 색을 흰색으로 바꾼다. 그리고 drawArc메소드를 통해 호를 그려준다. 14번째줄 코드를 보면 (90,50)위치에 너비:20, 높이:40인 흰색호를 그리는데 호를 그리기 위한 시작각은 210도이고 호의 중심각은 120이다. 그래서 6개의 drawArc메소드가 x, y 즉 첫번째, 두번째 매겨변수를 제외하고는 너비:20, 높이:40이고 호를 그리기 위한 시작각이 210도이고 호의 중심각이 120인 흰색 호를 그리는 것은 동일하다.

```
21          //몸채를 그린다
22          g.setColor(Color.BLACK);
23          g.drawRect(60,180,80,100);
24
```

setColor메소드를 통해 색을 검정색으로 바꾼다. 그리고 지붕아래 집의 몸체를 그려준다. drawRect메소드를 사용해 (60,180)부터 너비:80, 높이:100인 사각형을 검정색으로 그려준다.

```
25          //반원 모양의 창문을 그린다.
26          g.setColor(Color.YELLOW);
27          g.fillArc(80,195,40,85,0,180);
28
```

setColor메소드를 통해 색을 노란색으로 바꾼다. 반원 모양의 창문을 그린다. fillArc메소드를 통해 (80,195)좌표 부터 시작해서 너비:40, 높이:85 크기의 호를 그리기위한 시작각은 0도, 호의 중심각은 180도의 노란색으로 채워진 호를 그리게된다.

```
29          //노란색 창문 주위를 2개의 검은색 선으로 그린다.
30          g.setColor(Color.BLACK);
31          g.drawArc(80, 195, 40, 85, 0, 180);
32          g.drawArc(79,195,42,86,0,180);
33          g.drawLine(79, 238, 120, 238);
34          g.drawLine(79, 239, 120, 239);
35
```

setColor메소드를 통해 색을 검정색으로 바꾼다. drawArc메소드를 통해 (80,195)위치에 너비:40, 높이:85이면서 호를 그리기위한 시작각이 0도이고 호의 중심각이 180도의 호를 검정색 선으로 그려준다. 또 (79,195) 위치에 너비:42, 높이:86이면서 호를 그리기 위한 시작각이 0도이고 호의 중심각이 180도의 호를 검정색으로 그려주어 노란색 창문 주위에 2개의 검은색 선으로 그려준다.
drawLine메소드를 통해 (79,238)부터 (120,238)까지 검정색 선을 그려주고, 마찬가지로 (79,239)부터 (120,239)까지 검정색 선을 그려준다.

```
36          //문을 그린다.
37          g.setColor(Color.GRAY);
38          g.fillRect(85, 250, 30, 30);
39          g.setColor(Color.BLACK);
40          g.drawRect(85, 250, 30, 30);
41          g.drawLine(100, 250, 100, 280);
42          g.fillOval(96, 260, 8, 8);
```

setColor메소드를 통해 색을 회색으로 바꾼다. 창문 아래에 문을 표현하기 위해서 fillRect메소드를 통해 (85,250)부터 높이,너비가 30인 회색의 사각형을 그려주게 된다. 그리고 setColor메소드를 통해 검정색을 바꾸어주고 drawRect메소드를 통해 (85,250)부터 시작해 너비,높이가 30인 문의 테두리를 표현해준다. 그리고 drawLine메소드를 통해 (100,250)부터 (100,280)까지 문 정중앙의 세로방향으로 검정색 선을 표현해준다. 마지막으로 fillOval메소드를 통해 (96,260) 위치에 너비, 높이가 8인 검정색인 원을 그려주어 문 손잡이를 표현해준다.

메인함수에서 JFrame 객체를 만들고 프레임의 크기를 너비:250, 높이:350으로 설정하고 프레임의 제목을 "House"로 설정하고 디폴트 닫힘 연산을 지정한다. 사용자가 프레임을 닫을 때, 프로그램은 자동적으로 종료된다.
그리고 HouseDrawingComponent객체를 만들고 그 객체를 frame.add메소드를 통해 프레임에 추가시키고 마지막으로 프레임을 보이도록 만든다.

<문제>8.5

아래 조건에 따라 다음과 같은 얼굴을 그리시오.

얼굴색 (R: 255, G: 220, B: 130)

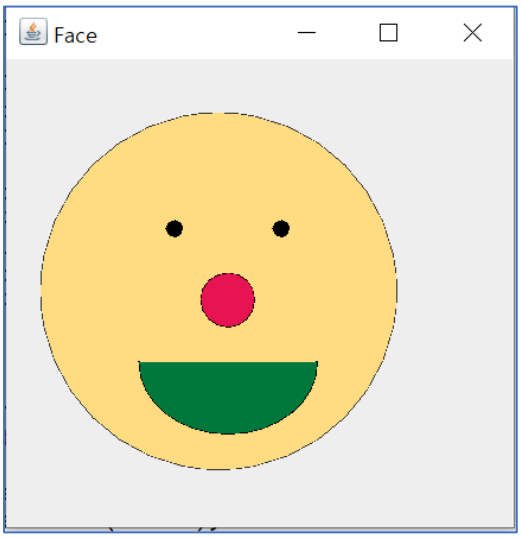
코색 (R: 230, G: 20, B: 80)

입색 (R: 0, G: 120, B: 60)

<코드>

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

class FaceDrawingComponent extends JComponent{
    public void paintComponent(Graphics g) {
        g.drawOval(20, 30, 200, 200);
        g.setColor(new Color(255,220,130));
        g.fillOval(20, 30, 200, 200);

        g.setColor(Color.BLACK);
        g.fillOval(90, 90, 10, 10);
        g.fillOval(150, 90, 10, 10);

        g.drawOval(110, 120, 30, 30);
        g.setColor(new Color(230,20,80));
        g.fillOval(110, 120, 30, 30);

        g.setColor(Color.BLACK);
        g.drawArc(75, 130, 100, 80, 180, 180);
        g.drawLine(75, 170, 175, 170);
        g.setColor(new Color(0,120,60));
        g.fillArc(75, 130, 100, 80, 180, 180);
    }
}

public class ch8_5 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(300,300);
        frame.setTitle("Face");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        FaceDrawingComponent component = new FaceDrawingComponent();
        frame.add(component);
        frame.setVisible(true);
    }
}
```

<실행결과>



<설명>

JComponent를 상속받는 FaceDrawingComponent클래스를 만든다. 도형을 그리기 위해서는 반드시 Graphics클래스 객체가 필요하며 이것을 그래픽 객체라고 부른다. 그래픽 객체는 JComponent 클래스를 확장하여 얻을 수 있다. 여기서 paintComponent 메소드는 화면을 다시 그려야할 때 자바에 의해서 자동으로 호출되는 메소드로 매개변수에 그래픽 객체가 전달된다. 그래서 paintComponent메소드에 그리기 작업을 하면 화면에 나타난다. 그래서 paintComponent메소드를 재정의한다.

```
6           g.drawOval(20, 30, 200, 200);
7           g.setColor(new Color(255,220,130));
8           g.fillOval(20, 30, 200, 200);
9
```

전체적인 얼굴을 그리는 코드이다. drawOval메소드를 통해서 원을 그려준다. X:20, y:30 위치부터 시작해서 너비, 높이 길이가 200인 원을 그려주게 된다. 그리고 setColor메소드를 통해 RGB(255,220,130) 색으로 color를 설정한다. 그리고 fillOval메소드를 통해 원을 채워서 그리게 된다. 이역시 아까 drawOval메소드와 같이 (20,30)부터 시작해서 너비, 높이 길이가 200인 원을 바로 이전에 지정한 색으로 채워진 원을 그리게 된다.

```
10          g.setColor(Color.BLACK);
11          g.fillOval(90, 90, 10, 10);
12          g.fillOval(150, 90, 10, 10);
13
```

얼굴에 눈 두개를 그리는 코드이다. setColor를 통해 색을 BLACK으로 설정한다. 그리고 fillOval메소드를 통해 (90,90)부터 시작해서 너비, 높이가 10인 검정색으로 채워진 원을 그리게 되고, 마찬가지로 (150, 90)부터 시작해서 너비, 높이가 10인 검정색으로 채워진 원을 그려 오른쪽 눈을 그려주게 된다.

```
14          g.drawOval(110, 120, 30, 30);
15          g.setColor(new Color(230,20,80));
16          g.fillOval(110, 120, 30, 30);
17
```

얼굴에 코를 그려주는 코드이다. drawOval를 통해 검정색으로 (110,120) 위치에 너비, 높이 30인 원을 그려준다. 그 후 setColor메소드를 통해 RGB(230,20,80)으로 설정하고, fillOval메소드를 통해 (110,120)위치에 너비, 높이 30인 빨간색에 가까운 색으로 채워진 원을 그려준다.

```
18          g.setColor(Color.BLACK);
19          g.drawArc(75, 130, 100, 80, 180, 180);
20          g.drawLine(75, 170, 175, 170);
21          g.setColor(new Color(0,120,60));
22          g.fillArc(75, 130, 100, 80, 180, 180);
```

얼굴에 입을 그리는 코드이다. setColor메소드를 통해 검정색으로 설정하고, drawArc메소드를 통해 (75,130)위치에 너비:100, 높이: 80 인 호를 그려주는데 이때 호의 시작각은 180도부터 시작해서 호의 중심각이 180도가 되게 그려준다. 그리고 drawLine메소드를 통해 (75, 170)부터 (175, 170)까지 검정색 선을 그려준다.
setColor메소드를 통해 RGB(0,120,60)으로 설정하고 fillArc메소드를 통해 (75,130)위치에 너비:100, 높이:80인 호를 그리는데 이때 180도부터 시작해서 호의 중심각이 180도인 초록색에 가까운 색으로 채워진 호를 그려준다.

메인함수에서 JFrame 객체를 만들고 프레임의 크기를 너비:300, 높이:300으로 설정하고 프레임의 제목을 "Face"로 설정하고 디폴트 닫힘 연산을 지정한다. 사용자가 프레임을 닫을 때, 프로그램은 자동적으로 종료된다.
그리고 FaceDrawingComponent객체를 만들고 그 객체를 frame.add메소드를 통해 프레임에 추가시키고 마지막으로 프레임을 보이도록 만든다.

<문제>8.6.(3)

다음과 같은 그림을 그리시오.

(3) 오륜기

<코드>

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  class OlympicflagDrawingComponent extends JComponent{
5      public void paintComponent(Graphics g) {
6          g.setColor(Color.BLUE);
7          g.drawOval(50, 20, 60, 60);
8          g.setColor(Color.BLACK);
9          g.drawOval(110, 20, 60, 60);
10         g.setColor(Color.RED);
11         g.drawOval(170, 20, 60, 60);
12         g.setColor(Color.YELLOW);
13         g.drawOval(75, 45, 60, 60);
14         g.setColor(Color.GREEN);
15         g.drawOval(145, 45, 60, 60);
16     }
17 }
18
19 public class ch8_6_3 {
20     public static void main(String[] args) {
21         JFrame frame = new JFrame();
22         frame.setSize(300,300);
23         frame.setTitle("Olympicflag");
24         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25
26         OlympicflagDrawingComponent component = new OlympicflagDrawingComponent();
27         frame.add(component);
28         frame.setVisible(true);
29     }
30 }
```

<실행결과>

<설명>

JComponent를 상속받는 OlympicflagDrawingComponent클래스를 만든다. 도형을 그리기 위해서는 반드시 Graphics클래스 객체가 필요하며 이것을 그래픽 객체라고 부른다. 그래픽 객체는 JComponent 클래스를 확장하여 얻을 수 있다. 여기서 paintComponent 메소드는 화면을 다시 그려야할 때 자바에 의해서 자동으로 호출되는 메소드로 매개변수에 그래픽 객체가 전달된다. 그래서 paintComponent메소드에 그리기 작업을 하면 화면에 나타난다. 그래서 paintComponent메소드를 재정의한다.

```
6           g.setColor(Color.BLUE);
7           g.drawOval(50, 20, 60, 60);
```

오륜기를 그려주는 코드이다. setColor 메소드를 통해 색을 파란색으로 설정한다. 그후 drawOval메소드를 통해 (50,20)에 너비, 높이가 60인 파란색 원을 그려준다.

```
8           g.setColor(Color.BLACK);
9           g.drawOval(110, 20, 60, 60);
```

setColor 메소드를 통해 색을 검정색으로 설정한다. 그후 drawOval메소드를 통해 (110,20)에 너비, 높이가 60인 검정색 원을 그려준다.

```
10          g.setColor(Color.RED);
11          g.drawOval(170, 20, 60, 60);
```

setColor 메소드를 통해 색을 빨간색으로 설정한다. 그후 drawOval메소드를 통해 (170,20)에 너비, 높이가 60인 빨간색 원을 그려준다.

```
12          g.setColor(Color.YELLOW);
13          g.drawOval(75, 45, 60, 60);
```

setColor 메소드를 통해 색을 노란색으로 설정한다. 그후 drawOval메소드를 통해 (75,45)에 너비, 높이가 60인 노란색 원을 그려준다.

```
14          g.setColor(Color.GREEN);
15          g.drawOval(145, 45, 60, 60);
```

setColor 메소드를 통해 색을 초록색으로 설정한다. 그후 drawOval메소드를 통해 (145,45)에 너비, 높이가 60인 초록색 원을 그려준다.

메인함수에서 JFrame 객체를 만들고 프레임의 크기를 너비:300, 높이:300으로 설정하고 프레임의 제목을 "Olympicflag"로 설정하고 디폴트 닫힘 연산을 지정한다. 사용자가 프레임을 닫을 때, 프로그램은 자동적으로 종료된다.

그리고 OlympicflagDrawingComponent객체를 만들고 그 객체를 frame.add메소드를 통해 프레임에 추가시키고 마지막으로 프레임을 보이도록 만든다.