소프트웨어의 유지보수성과 소프트웨어 설계 원리

유지보수성이 좋은 소프트웨어를 만들기 위해서는 다양한 설계 원리들을 이해해야 한다. 간단한 월세 관리 프로그램을 통해 소프트웨어 설계 원칙들을 점검해보자.

1. 유지보수가 어려운 기본 프로그램

1) 요구사항 정의

집을 여러 채 소유하고 있는 사람이 남는 방을 월세로 임대하려고 한다. 방의 임대료는 성인은 한 달에 10만원, 어린이는 2만원이다. 지금까지는 임대 관리자가 장부에 각 방의 거주자들의 정보를 적고 임대료를 전자 계산기로 계산했다. 그런데 집주인이 새로 집들을 다량으로 구입할 예정이므로 이런 구식 방법으로 한계가 있다. 따라서 임대 관리자는 자바로 임대료를 자동으로 계산하는 프로그램을 만들기로 결정했다. 이 프로그램은 각 방의 임대료와 특정 집 전체의 임대료를 자동으로 계산하여 화면에 출력하도록 한다.

현재는 2채의 집이 있다. 각 집의 이름은 둘리 하우스, 뽀로로 하우스이며, 다음과 같다.

집 이름	방 이름	거주자 이름	거주자 타입
둘리 하우스	**100**	둘리	성인
		또치	성인
		고길동	성인
		희동이	어린이
	101	마이콜	성인
	102	도우너	성인
뽀로로 하우스	**101**	뽀로로	성인
		포비	성인
		루피	성인

실행 결과는 다음과 같아야 한다.

```
************************************************************
이 집의 이름 : 둘리 하우스
이 집에 살고 있는 거주자의 수 : 6
이 집의 총 임대비용 : 52.0
************************************************************
```

```
**********************************************************
이 집의 이름 : 둘리 하우스
이 방의 이름 : 100
이 방에 살고 있는 거주자의 수 : 4
이 방의 총 임대비용 : 32.0
 거주자 이름 : 둘리, 임대비 : 10.0
 거주자 이름 : 또치, 임대비 : 10.0
 거주자 이름 : 고길동, 임대비 : 10.0
 거주자 이름 : 희동이, 임대비 : 2.0
**********************************************************
**********************************************************
이 집의 이름 : 둘리 하우스
이 방의 이름 : 101
이 방에 살고 있는 거주자의 수 : 1
이 방의 총 임대비용 : 10.0
 거주자 이름 : 마이콜, 임대비 : 10.0
**********************************************************
**********************************************************
이 집의 이름 : 둘리 하우스
이 방의 이름 : 102
이 방에 살고 있는 거주자의 수 : 1
이 방의 총 임대비용 : 10.0
 거주자 이름 : 도우너, 임대비 : 10.0
**********************************************************
**********************************************************
이 집의 이름 : 뽀로로 하우스
이 집에 살고 있는 거주자의 수 : 3
이 집의 총 임대비용 : 30.0
**********************************************************
**********************************************************
이 집의 이름 : 뽀로로
이 방의 이름 : 101
이 방에 살고 있는 거주자의 수 : 0
이 방의 총 임대비용 : 0.0
**********************************************************
```

2) 프로그램 기능 구현

먼저 프로그램이 없는 현재 상황을 떠올려보며 클래스들을 도출해보도록 하자. 임대 관리자는 장부에 거주자의 정보를 적어놓고 계산을 했을 것이다. 그러면 임대 관리자에 해당하는 RentalManager 클래스를 떠올릴 수 있을 것이다. 그리고 이 사람에게 장부가 있을 테니 이를 RentalBook이라는 클래스로 표현한다. 마지막으로 이 장부 안에는 방을 입대한 사람들의 정보가 저장되어 있을 것이다. 이들을 거주자인 Resident라는 클래스로 표현한다.

집주인은 임대 관리자에게 특정 집과 방의 임대료를 조사해서 보고하라고 할 테니 임대 관리자 클래스에 특정 집의 총 임대료, 특정 방의 임대료를 알아내는 메소드가 필요하다. 임대 관리자는 장부를 보고 그런 것들을 알아낼 것이므로 임대 관리자 클래스의 메소드도 장부 객체들에게 장부의 내용을 알려달라고 해서 임대료를 계산한다. 장부 안에는 거주자 정보가 저장되어 있을 테니 장부 객체는 거주자 객체들을 가지고 있어야 한다. 이런 상황을 소스로 표현하면 다음과 같다.

가장 먼저 거주자에 해당하는 Resident 클래스를 작성한다.

```
class Resident {
	private String houseName; // 거주자의 주택 이름
	private String roomName; // 거주자의 방 이름

	private String name; // 거주자의 이름
	private String type; // adult, children 등

	public Resident(String houseName, String roomName, String type, String name) {
		this.houseName = houseName;
		this.roomName = roomName;
		this.type = type;
		this.name = name;
	}

	public double calculateRentalValue() {
		double RentalValue = 0;

		if (type.equals("Adult"))
			RentalValue = 10;
		else if (type.equals("Children"))
			RentalValue = 2;

		return RentalValue;
	}

	public String getName() {
		return name;
	}

	public String getRoomName() {
		return roomName;
	}

	public String getHouseName() {
		return houseName;
	}

}
```

거주자 정보를 저장하는 장부에 해당하는 RentalBook 클래스를 작성한다.

```
class RentalBook {
	private ArrayList<Resident> residents;

	public RentalBook() {
		residents = new ArrayList<Resident>();
	}

	public Resident[] getAllResidents() {
		return (Resident[]) residents.toArray(new Resident[0]);
	}

	public void addResident(Resident res) {
		residents.add(res);
	}

	public void removeResident(Resident res) {
		residents.remove(res);
	}
}
```

임대 관리인에 해당하는 RentalManager를 작성한다.

```
class RentalManager {
	private RentalBook rentalBook;

	public RentalManager(RentalBook rentalBook) {
		this.rentalBook = rentalBook;
	}

	public RentalBook getRentalBook() {
		return rentalBook;
	}

	/* 특정 집의 총 임대비용을 계산 */
	public double calculateRentalValue(String houseName) {
		double sum = 0;

		Resident[] residents = rentalBook.getAllResidents();

		for (int i = 0; i < residents.length; i++)
			if (residents[i].getHouseName().equals(houseName))
				sum += residents[i].calculateRentalValue();

		return sum;
	}

	/* 특정 집안에 있는 특정 방의 임대비용을 계산 */
	public double calculateRentalValue(String houseName, String roomName) {
		double sum = 0;

		Resident[] residents = rentalBook.getAllResidents();

		for (int i = 0; i < residents.length; i++)
			if (residents[i].getHouseName().equals(houseName) &&
residents[i].getRoomName().equals(roomName))
				sum += residents[i].calculateRentalValue();

		return sum;
	}
```

```
        /* 특정 집의 거주자 수를 계산 */
        public int countResidents(String houseName) {
                int count = 0;

                Resident[] residents = rentalBook.getAllResidents();

                for (int i = 0; i < residents.length; i++)
                        if (residents[i].getHouseName().equals(houseName))
                                count++;

                return count;
        }

        /* 특정 집안에 있는 특정 방의 거주자 수를 계산 */
        public int countResidents(String houseName, String roomName) {
                int count = 0;

                Resident[] residents = rentalBook.getAllResidents();

                for (int i = 0; i < residents.length; i++)
                        if (residents[i].getHouseName().equals(houseName) &&
residents[i].getRoomName().equals(roomName))
                                count++;

                return count;
        }

        /* 특정 집안에 있는 특정 방의 거주자들을 반환 */
        public Resident[] getRoomResidents(String houseName, String roomName) {
                ArrayList<Resident> roomResidents = new ArrayList<Resident>();

                Resident[] allResidents = rentalBook.getAllResidents();

                for (int i = 0; i < allResidents.length; i++)
                        if (allResidents[i].getHouseName().equals(houseName) &&
allResidents[i].getRoomName().equals(roomName))
                                roomResidents.add(allResidents[i]);

                return (Resident[]) roomResidents.toArray(new Resident[0]);
        }
}
```

이제 Resident, RentalBook, RentalManager객체를 사용하는 클라이언트 프로그램을 다음과 같이 작성한다.

```
public class RentalValueCalculator {

    public static void main(String[] args) {
        RentalBook rentalBook = new RentalBook();

        rentalBook.addResident(new Resident("둘리 하우스", "100", "Adult", "둘리"));
        rentalBook.addResident(new Resident("둘리 하우스", "100", "Adult", "또치"));
        rentalBook.addResident(new Resident("둘리 하우스", "100", "Adult", "고길동"));
        rentalBook.addResident(new Resident("둘리 하우스", "100", "Children", "희동이"));
```

```
        rentalBook.addResident(new Resident("둘리 하우스", "101", "Adult", "마이콜"));
        rentalBook.addResident(new Resident("둘리 하우스", "102", "Adult", "도우너"));

        rentalBook.addResident(new Resident("뽀로로 하우스", "101", "Adult", "뽀로로"));
        rentalBook.addResident(new Resident("뽀로로 하우스", "101", "Adult", "포비"));
        rentalBook.addResident(new Resident("뽀로로 하우스", "101", "Adult", "루피"));

        RentalManager manager = new RentalManager(rentalBook);

        // 특정 집의 사람수, 총 임대 비용
        // 특정 집의 각 방마다 사람 수, 사람 이름, 임대비용

        printRentalStatus(manager, "둘리 하우스");
        printRentalStatus(manager, "둘리 하우스", "100");
        printRentalStatus(manager, "둘리 하우스", "101");
        printRentalStatus(manager, "둘리 하우스", "102");
        printRentalStatus(manager, "뽀로로 하우스");
        printRentalStatus(manager, "뽀로로", "101");
    }

    static public void printRentalStatus(RentalManager rentalManager, String houseName) {
        System.out.println("************************************************");

        System.out.println("이 집의 이름 : " + houseName);
        System.out.println("이 집에 살고 있는 거주자의 수 : " +
rentalManager.countResidents(houseName));
        System.out.println("이 집의 총 임대비용 : " +
rentalManager.calculateRentalValue(houseName));

        System.out.println("************************************************");
    }

    static public void printRentalStatus(RentalManager rentalManager, String houseName,
String roomName) {
        System.out.println("************************************************");

        System.out.println("이 집의 이름 : " + houseName);
        System.out.println("이 방의 이름 : " + roomName);
        System.out.println("이 방에 살고 있는 거주자의 수 : " +
rentalManager.countResidents(houseName, roomName));
        System.out.println("이 방의 총 임대비용 : " +
rentalManager.calculateRentalValue(houseName, roomName));

        Resident[] roomResidents = rentalManager.getRoomResidents(houseName, roomName);
        for (int i = 0; i < roomResidents.length; i++)
            System.out.println(" 거주자 이름 : " + roomResidents[i].getName() + "," + "
임대비 : " + roomResidents[i].calculateRentalValue());

        System.out.println("************************************************");

    }
}
```

이렇게 작성된 소스는 다음과 같은 문제를 가지고 있다.

1. 관계가 복잡하여 구조 파악이 어려우며, 유지보수가 어려워진다.
2. 객체의 역할이 모호하다.
3. 특정 객체(RentalManager)의 역할이 크고 집중되어 있다. RentalManager가 거주자 수, 임대료 계산 등의 많은 기능을 처리하고 있다.

이런 문제를 해결하기 위해 적절하게 소스를 수정하세요.