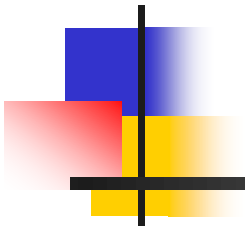
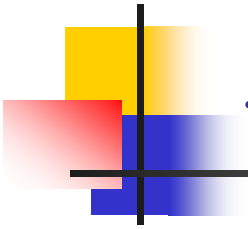


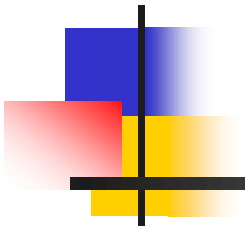
java 11강 - 기타 예제

양 명 숙
[now4ever7@gmail.com]

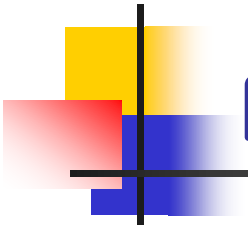


목차

- 클래스 예제-과일 장사
- 매개변수의 다형성
- 포함관계 이용



클래스 예제-과일 장사



FruitSeller 클래스 정의

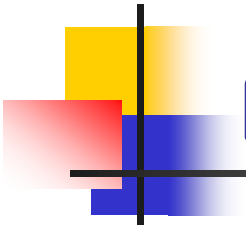
- 예) 나는 과일장사에게 2000원을 주고 두 개의 사과를 구매했다.
  - '과일 장사' 클래스 – 과일 판매자 표현
    - 멤버변수
      - 사과의 개수
      - 판매수익 (남은 돈)
      - 사과 하나의 가격 – 상수(final)
    - 메서드
      - 과일 판매 메서드
      - 현재 상태를 나타내는 메서드
        - 오늘 2000원 벌었고, 남은 사과는 18개다.

FruitSeller 클래스 정의

- 과일장사의 상태정보를 변수로 표현
  - 보유하고 있는 사과의 수 => int numOfApple
  - 판매수익 => int myMoney

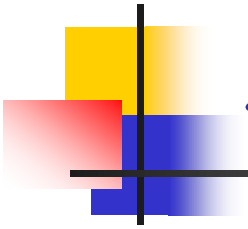
- 과일의 판매를 메소드로 표현

```
int saleApple(int money){ //과일 구매액
    int num=money/1000; //사과가 개당 1000원
    numOfApple -= num;   //사과의 수가 줄어들고,
    myMoney += money;  //판매수익이 발생
    return num; //실제 구매가 발생한 사과의 수
}
```



FruitBuyer 클래스 정의

- '나' 클래스 – 과일 구매자 표현
  - 데이터(멤버 변수) – 돈, 사과
    - 소유하고 있는 현금 => int myMoney
    - 소유하고 있는 사과의 수 => int numOfApple
  - 기능(메서드)
    - 과일 구매 메서드
    - 현재 상태를 나타내는 메서드



생성자

- 두 명의 과일 장사가 있고, 이들의 판매내용
  - 과일장사1 : 보유하고 있는 사과의 개수는 30개, 개당 가격은 1500원
  - 과일장사2 : 보유하고 있는 사과의 개수는 20개, 개당 가격은 1000원
- 나는 과일장사1 에게 4500원어치 사과를 구매했고, 과일장사2 에게 2000원어치 사과를 구매했다.
  - 두 개의 과일장사 객체 생성
  - 과일장사의 사과 보유수와 개당 가격이 다르므로, 변수의 초기 값도 달라져야 함
  - 클래스를 정의하면서 변수 값을 초기화할 수 없음
  - 객체 생성 후, 멤버변수를 각각 초기화하자

FruitSeller 클래스

```
//과일 판매자 클래스 (과일 장사)
class FruitSeller{
	private int numOfApple;  //사과의 개수
	private int myMoney;  //판매수익 (남은 돈)
	private final int APPLE_PRICE;  //사과 하나의 가격

	FruitSeller(int money, int appleNum, int price){
		myMoney=money;
		numOfApple=appleNum;
		APPLE_PRICE=price;  //final 상수 - 생성자에서 단 한번 초기화:인스턴스별로 다른 값 할당
	}

	public int saleApple(int money){  //사과를 판매하는 메서드 - 2000원어치 사과 주세요
		int num=money/APPLE_PRICE;
		numOfApple-=num;
		myMoney+=money;
		return num;  //구매자에게 줄 사과개수
	}

	public void showSaleResult()  //추가된 메서드{
		System.out.println("남은 사과: " + numOfApple);
		System.out.println("판매 수익: " + myMoney);
	}
}
```

FruitBuyer 클래스

```
class FruitSeller{
        ...
    public int saleApple(int money){
            int num=money/APPLE_PRICE;
            numOfApple -= num;
            myMoney += money;
            return num;
    }
```

```
class FruitBuyer{  //과일 구매자 클래스(나)
    private int myMoney;  //남은 돈
    private int numOfApple;  //구매한 사과 개수

    public FruitBuyer(int money){
            myMoney=money;
            //numOfApple=0;
    }

    public void buyApple(FruitSeller seller, int money){  //사과를 구매하는 메서드
            //사과를 구매하는데 있어서 필요한 것 - 구매대상, 구매금액 => 매개변수로 전달되어야 함
            //seller 아저씨, 사과 2000원어치 주세요
            numOfApple+=seller.saleApple(money);
            myMoney-=money;

            //=> seller 가 참조하는 객체는 사과를 판매하고 수익이 생긴다
            //=> buyer 가 참조하는 객체는 돈을 지불하고 사과를 얻게 된다
    }
    public void showBuyResult() {
            System.out.println("현재 잔액: " + myMoney);
            System.out.println("사과 개수: " + numOfApple);
    }
}
```

예제 완성하기

```
과일 판매자1의 현재 상황
남은 사과: 27
판매 수익: 4500

과일 판매자2의 현재 상황
남은 사과: 18
판매 수익: 2000

과일 구매자의 현재 상황
현재 잔액: 3500
사과 개수: 5
```

```
class FruitSalesMain
{
    public static void main(String[] args)
    {
        FruitSeller seller1 = new FruitSeller(0, 30, 1500);  //money, 사과개수, 사과가격
        FruitSeller seller2 = new FruitSeller(0, 20, 1000);
        FruitBuyer buyer = new FruitBuyer(10000); //money

        buyer.buyApple(seller1, 4500);  // 유일한 과일 구매 방법
        buyer.buyApple(seller2, 2000);

        System.out.println("과일 판매자1의 현재 상황");
        seller1.showSaleResult();

        System.out.println("₩n과일 판매자2의 현재 상황");
        seller2.showSaleResult();

        System.out.println("₩n과일 구매자의 현재 상황");
        buyer.showBuyResult();
    }
}
```

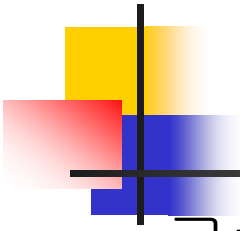
```
buyer.buyApple(seller1, 2000);

buyer.buyApple(seller2, 5000);
```

예- 정보 은닉

```
===과일 판매자의 현재 상황===
남은 사과: 10
판매수익: 500

===과일 구매자의 현재 상황===
현재 잔액: 9500
사과 개수: 20
```

외부에서 멤버변수에 직접 접근이 가능했기 때문에 문제 발생

```
class FruitSalesMain
{
	public static void main(String[] args)
	{
		FruitSeller seller = new FruitSeller(0,30,1500);
		FruitBuyer buyer = new FruitBuyer(10000);

		seller.myMoney += 500;  //돈 500원 내고,
		buyer.myMoney -= 500;

		seller.numOfApple -= 20;  //사과 20개 가져가기
		buyer.numOfApple += 20;


		System.out.println("===과일 판매자의 현재 상황===");
		seller.showSaleResult();

		System.out.println("₩n===과일 구매자의 현재 상황===");
		buyer.showBuyResult();
	}
}
```



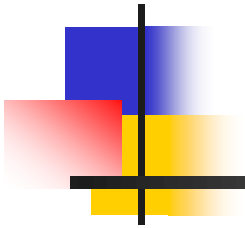
과제

- 구슬치기 놀이 클래스 정의하기
  - 어린아이가 소유하고 있는 구슬의 개수 정보를 담을 수 있다.
  - 놀이를 통한 구슬의 주고받음을 표현하는 메소드가 존재한다.
    - 두 아이가 구슬치기를 하는 과정에서 구슬의 잃고 얻음을 의미함
  - 어린이의 현재 보유자산(구슬의 수)을 출력하는 메소드가 존재한다.
- 다음 조건을 만족하는 객체를 각각 생성하기
  - 어린이1의 보유자산 => 구슬 15개
  - 어린이2의 보유자산 => 구슬 9개
- 객체의 생성이 완료되면 다음의 상황을 main 메소드 내에서 시뮬레이션 하기
  - 1차 게임에서 어린이1은 어린이2의 구슬 2개를 획득한다.
  - 2차 게임에서 어린이2는 어린이1의 구슬 7개를 획득한다.
  - 각각의 어린이가 보유하고 있는 구슬의 수를 출력한다.

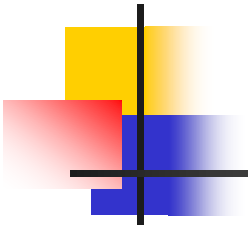
과제

```
게임 전 구슬의 보유 개수
===어린이1====
보유 구슬의 개수: 15
===어린이2====
보유 구슬의 개수: 9

게임 후 구슬의 보유 개수
===어린이1====
보유 구슬의 개수: 10
===어린이2====
보유 구슬의 개수: 14
```

- Child 클래스
  - 멤버변수 – 구슬의 개수(numOfBead)
  - 생성자에서 초기화
  - 메서드
    - loseBead() – 구슬을 잃는 메소드
      - public int loseBead(int loseCount)
      - 보유한 구슬보다 더 많이 잃은 경우, 보유한 구슬만큼만 잃는다
      - 구슬을 잃으면 보유구슬의 개수 감소
    - obtainBead()– 다른 어린이에게 구슬을 획득하는 메소드
      - public void obtainBead(Child child, int obtainCount)
      - 상대 어린이의 구슬은 감소
      - 구슬을 획득하면 보유구슬 증가
    - 현재 보유하고 있는 구슬의 개수를 출력하는 메소드
      - public void showProperty()



매개변수의 다형성



예제1-매개 변수의 다형성

- 고객(Buyer)이 buy(Product p) 메서드를 이용해서 Tv와 Computer를 구입하고, 고객의 잔고와 보너스 점수를 출력하는 예제
- Product, Tv, Computer, Buyer 클래스 정의
  - Product 클래스는 Tv, Computer 클래스의 부모
  - Buyer 클래스는 제품(Product)를 구입하는 사람을 클래스로 표현한 것
  - Buyer 클래스에 물건을 구입하는 기능의 메서드 추가
    - 구입할 대상이 필요하므로 구입할 제품을 매개변수로 넘겨받아야 함
    - buy(Tv t) – 제품을 구입하면 제품을 구입한 사람이 가진 돈에서 제품의 가격을 빼고, 보너스 점수를 추가하는 작업을 하는 메서드
    - 다른 제품들도 구입할 수 있도록 하려면 메서드의 매개변수에 다형성 적용
    - => buy(Product p)

Product, Tv, Computer, Buyer 클래스 정의

```
class Product {
    private int price;                  // 제품의 가격
    private int bonusPoint;             // 제품구매 시 제공하는 보너스점수
}

class Tv extends Product {
}

class Computer extends Product {
}

class Buyer {                           // 고객, 물건을 사는 사람
    private int money = 1000;                   // 소유금액-1000만원
    private int bonusPoint = 0;                 // 보너스점수
}
```

```
abstract class Product{
	private int price;  //상품의 가격
	private int bonusPoint;  //포인트 점수
	private final double POINT_RATE=0.02;  //가격의 2%가 포인트점수

	Product(int price){
		this.price=price;
		this.bonusPoint = (int)(price*POINT_RATE);
	}

	public int getPrice(){
		return price;
	}
	public int getBonusPoint(){
		return bonusPoint;
	}

	public abstract String findInfo();
}//class

class TV extends Product{
	TV(int price){
		super(price);
	}
	public String findInfo(){
		return "TV";
	}
}
```

```
TV을(를) 구매하였습니다
현재 잔고:900
보너스 포인트:2

Computer을(를) 구매하였습니다
현재 잔고:700
보너스 포인트:6
```

```
class Computer extends Product{
		Computer(int price){
			super(price);
		}
		public String findInfo(){
			return "Computer";
		}
}
```

```
class Buyer{   //고객 - 물건을 사는 사람
    private int myMoney;
    private int point;

    Buyer(int myMoney){
        this.myMoney=myMoney;
    }

    //물건을 구매하는 메서드
    public void buy(Product p){
        if (myMoney< p.getPrice()){
            System.out.println("잔액이 부족하여 물건을 구매할 수 없습니다");
            return;
        }

        //잔고는 줄고, 보너스포인트는 늘어난다
        this.myMoney -= p.getPrice();
        this.point +=p.getBonusPoint();

        System.out.println(p.findInfo() + "을(를) 구매하였습니다");
    }

    public void showInfo(){
        System.out.println("현재 잔고:"+myMoney);
        System.out.println("보너스 포인트:"+point+"₩n");
    }

}
```

```
class PdBuyerTest {
    public static void main(String[] args) {
        Buyer b = new Buyer(1000);
        //1. TV 구매 - 100
        //buy(Product p)   ← new Product(), new TV()
        //Product p = new Product();
        //Product p = new TV();

        Product tv= new TV(100);
        b.buy(tv);  //
        b.showInfo();

        //2. Computer 구매 - 200
        Computer com = new Computer(200);
        b.buy(com);  //
        b.showInfo();

    }
}
```

여러 종류의 객체를 하나의 배열로 다루기

- Buyer 클래스에 구입한 제품을 저장하기 위한 Product 배열을 추가
- buy() 메서드에 item[i++]=p; 문장을 추가해서 물건을 구입하면, 배열 item에 저장되도록 한다

```
1. 상품 구매  2. 구매내역 조회 3. 종료
1

구매할 상품을 선택하세요 (1. TV 2. Computer)
1
상품가격을 입력하세요
100
TV을(를) 구매하였습니다

현재 잔고:900
보너스 포인트:2

1. 상품 구매  2. 구매내역 조회 3. 종료
1

구매할 상품을 선택하세요 (1. TV 2. Computer)
2
상품가격을 입력하세요
300
Computer을(를) 구매하였습니다

현재 잔고:600
보너스 포인트:8
```

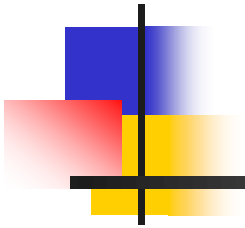
```
1. 상품 구매  2. 구매내역 조회 3. 종료
1

구매할 상품을 선택하세요 (1. TV 2. Computer)
1
상품가격을 입력하세요
700
잔액이 부족하여 물건을 구매할 수 없습니다

현재 잔고:600
보너스 포인트:8

1. 상품 구매  2. 구매내역 조회 3. 종료
2

구입하신 물건의 총금액은 400입니다
구입하신 제품은 TV, Computer입니다
```

```
class Buyer{
    private int myMoney; //900
    private int point; //2
    //구입한 상품을 저장할 배열 추가
    private Product[] pdArr=new Product[10];
    private int count;  //배열의 첨자로 사용될 변수

    Buyer(int myMoney){
        this.myMoney=myMoney;
    }

    //물건을 구매하는 메서드
    public void buy(Product p){
        if (myMoney< p.getPrice()){ //1000<100
            System.out.println("잔액이 부족하여 물건을 구매할 수 없습니다₩n");
            return;
        }
        //잔고는 줄고, 보너스포인트는 늘어난다
        this.myMoney -= p.getPrice();  //100
        this.point +=p.getBonusPoint();

        //구매한 상품을 배열에 저장한다
        pdArr[count++]=p;
        System.out.println(p.findInfo() + "을(를) 구매하였습니다₩n");
    }
```

```
    public void summary(){
        //구입한 상품명들과 상품가격의 합계 구하기
        int sum=0;
        String itemList="";

        for (int i=0;i<count ;i++ ){
                Product p = pdArr[i];  //tv(100), com(200)
                sum+=      p.getPrice(); //100+200
                itemList += p.findInfo(); //TV, Computer
                if (i<count-1)          {
                        itemList+=", ";  //TV, Computer
                }
        }//for

        System.out.println("₩n구입하신 물건의 총금액은 " + sum+"입니다");
        System.out.println("구입하신 제품은 " + itemList+"입니다₩n");
    }

    public void showInfo(){
        System.out.println("현재 잔고:"+myMoney);
        System.out.println("보너스 포인트:"+point+"₩n");
    }

}
```

```
class PdBuyerTest2 {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        Buyer b = new Buyer(1000);
        while (true){
            System.out.println("1. 상품 구매  2. 구매내역 조회 3. 종료");
            int type = sc.nextInt();
            switch (type)
            {
                case 1:
                    System.out.println("₩n구매할 상품을 선택하세요 (1. TV 2. Computer)");

                    int kind = sc.nextInt();
                    if (kind!=1 && kind!=2){
                        System.out.println("구매할 상품을 다시 선택하세요₩n");
                        continue;
                    }

                    System.out.println("상품가격을 입력하세요");
                    int price = sc.nextInt();
```

```
                                        Product p =null;
                                        if (kind==1){
                                                p = new TV(price);
                                        }else if (kind==2)      {
                                                p = new Computer(price);
                                        }

                                        b.buy(p);
                                        b.showInfo();
                                        break;
                                case 2:
                                        b.summary();
                                        break;
                                case 3:
                                        System.out.println("프로그램을 종료합니다₩n");
                                        return;
                                default:
                                        System.out.println("다시 선택하세요₩n");
                                        continue;
                        }//switch

                }//while
        }
}
```



포함 관계 이용

포함관계 이용

• Deck 클래스를 작성하는데 Card 클래스를 재사용하여 포함관계로 작성
• 카드 한 벌(Deck)은 모두 52장의 카드로 이루어져 있으므로 Card 클래스를 크기가 52인 배열로 처리

```
class Card{
    private final int KIND;  // 카드 무늬의 수 1~4
    private final int NUMBER;  // 무늬별 카드 수(카드의 숫자) 1~10,  J, Q, K

    private static final int SPADE=1;
    private static final int DIAMOND=2;
    private static final int HEART=3;
    private static final int CLOVER=4;

    public static final int MAX_KIND=4;
    public static final int MAX_NUMBER=13;

    Card(){
        this(SPADE, 1);
    }
    Card(int kind, int number){
        this.KIND=kind;
        this.NUMBER=number;
    }
```

```
    public String findInfo(){
        String kind="", number="";
        switch (KIND){
            case SPADE:
                kind="SPADE";break;
            case DIAMOND:
                kind="DIAMOND";break;
            case HEART:
                kind="HEART";break;
            case CLOVER:
                kind="CLOVER";break;
        }
        switch (NUMBER){
            case 11:
                number="J";break;
            case 12:
                number="Q";break;
            case 13:
                number="K";break;
            default:
                number=NUMBER+"";
        }
        return "card[kind="+ kind +", number="+ number+"]";     //SPADE, K
    }
}
```

```
//카드 한벌 - 52장의 카드를 갖는 클래스
class Deck{
    private static final int CARD_NUM=52;  //카드의 개수
    private Card[] cardList = new Card[CARD_NUM]; //카드 52장을 담는 배열
    private int count;
    Deck(){
        //카드 52장 초기화 4*13
        for (int i=1;i<= Card.MAX_KIND;i++ ){  //4
            for (int j=1;j<=Card.MAX_NUMBER ; j++){ //13
                Card c = new Card(i, j);
                cardList[count++] = c;
            }//안쪽 for
        }//바깥 for
    }
    //getter
    public Card[] getCardList(){
        return cardList;
    }
    //카드 뽑는 메서드- 지정된 위치(idx)에 있는 카드 하나를 선택한다.
    public Card pick(int idx){
        Card c=null;
        if (idx<0 || idx>=CARD_NUM){
            c=pick();
        }else{
            c = cardList[idx];
        }
        return c;
    }
```

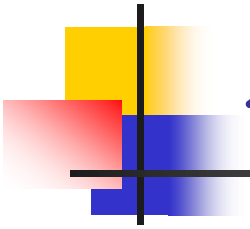
```
    //임의의 카드 한 장 뽑기
    public Card pick(){
        //0~51 사이의 idx에 해당하는 랜덤값 뽑기
        int rnd = (int)(Math.random()*52);  //0~51
        Card c = cardList[rnd];  //return pick(rnd);
        return c;
    }

    // 카드의 순서를 섞는다.
    public void shuffle(){
        for (int i=0;i<1000 ;i++ ){
            //랜덤한 index 값 읽어오기
            int rnd = (int)(Math.random()*52);  //0~51

            //0번째 배열의 값과 서로 맞바꾸기
            Card temp = cardList[0];
            cardList[0] = cardList[rnd];
            cardList[rnd] = temp;
        }//for
    }
}//class
```

카드 한 벌의 첫 번째 장과 임의로 선택한 위치에 있는 카드의 위치를 서로 바꾸는 방식으로 카드를 섞는다

```
class CardTest {
    public static void main(String[] args) {
        Deck d = new Deck();  //카드 한 벌 만들기
        Card c = d.pick(0);  //0번째 위치의 카드 뽑기
        System.out.println(c.findInfo());

        //카드 한 벌 출력하기
        System.out.println("₩n₩n----카드 한벌----");
        Card[] cardArr = d.getCardList();
        for (Card card : cardArr){
                System.out.println(card.findInfo());
        }

        //카드 섞기
        d.shuffle();

        //카드 섞은 후 0번째 위치의 카드 뽑기
        c = d.pick(0);
        System.out.println("₩n카드 섞은 후 결과 : "+c.findInfo()+"₩n");
    }
}
```

실습

- 소스 코드는 도형을 정의한 Shape클래스이다. 이 클래스를 부모로 하는 Circle클래스와 Rectangle클래스를 작성하시오. 이 때, 생성자도 각 클래스에 맞게 적절히 추가해야 한다.
- (1) 클래스명 : Circle
  - 부모클래스 : Shape
  - 멤버변수 : double r – 반지름
- (2) 클래스명 : Rect
  - 부모클래스 : Shape
  - 멤버변수 : int width – 폭, int height – 높이
  - 메서드 :
    - 1. 메서드명 : isSquare
    - 기 능 : 정사각형인지 아닌지를 알려준다.
    - 반환타입 : boolean
    - 매개변수 : 없음

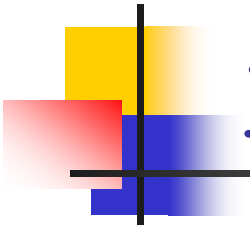


실습

- 앞에서 정의한 클래스들의 면적을 구하는 메서드를 작성하시오.
- 1. 메서드명 : sumArea
  - 기 능 : 주어진 배열에 담긴 도형들의 넓이를 모두 더해서 반환한다.
  - 반환타입 : double
  - 매개변수 : Shape[] arr

```
abstract class Shape {
    private Point p;
    Shape() {
        this(new Point(0,0));
    }
    Shape(Point p) {
        this.p = p;
    }
    Point getP() {
        return p;
    }
    void setP(Point p) {
        this.p = p;
    }

    abstract double calcArea();  // 도형의 면적을 계산해서 반환하는 메서드
}
class Point {
    private int x;
    private int y;
    Point() {
        this(0,0);
    }
    Point(int x, int y) {
        this.x=x;
        this.y=y;
    }
```

```
    public String findInfo() {
        return "["+x+","+y+"]";
    }
}
class Rect { //클래스를 작성하시오}

class Circle { //클래스를 작성하시오}

class Exercise{

    // sumArea 메서드를 작성하시오

    public static void main(String[] args){
        Shape[] arr = {new Circle(5.2), new Rect(3, 4), new Circle(1)};
        System.out.println("면적의 합:"+sumArea(arr));
    }
}
```



과제-야구 게임

- 야구 게임 - 3개의 숫자 중 자리수까지 맞히면 strike, 숫자만 맞추면 ball

- 컴퓨터가 만든 0~9 사이의 임의의 세 수를 맞추는 게임
- [1] 0~9 사이의 임의의 세 수를 만드는 투수
- [2] 이를 맞추려고 0~9 사이의 서로 다른 세 수를 입력하는 타자
- [3] 둘을 비교하는 심판이 필요

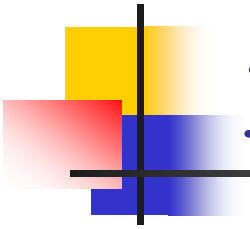
- 스트라이크 - 투수가 만든 수와 타자에서 입력 받은 수의 자리와 값이 같으면 스트라이크
- 볼 - 자리가 다르고 값이 같으면 볼

```
다른 세 수를 입력하세요(0~9)
1
7
8

반복회수:1, 1 Strike!! 0 Ball!!

다른 세 수를 입력하세요(0~9)
5
7
8

반복회수:2, 2 Strike!! 0 Ball!!

다른 세 수를 입력하세요(0~9)
5
7
4

You Win in 3

계속하시겠습니까?(Y/N) :
```

```
다른 세 수를 입력하세요(0~9)
1
5
8

반복회수:7, 0 Strike!! 1 Ball!!

다른 세 수를 입력하세요(0~9)
4
5
2

반복회수:8, 0 Strike!! 2 Ball!!

다른 세 수를 입력하세요(0~9)
2
4
5

반복회수:9, 2 Strike!! 0 Ball!!

다른 세 수를 입력하세요(0~9)
2
4
9

반복회수:10, 1 Strike!! 0 Ball!!


You Lose, Pitcher is
3       4       5

계속하시겠습니까?(Y/N) :
```

과제- 야구게임

- main() 에서
  - 10번의 기회를 준다
- main() 메서드가 속한 클래스
  - 메서드 - 사용자가 입력한 값을 배열에 담아 리턴하는 메서드
    - 중복 제거
- Pitcher - 투수 클래스
  - 멤버변수 - 요소 3개를 갖는 배열 (정답)
  - 메서드 - 0~9 중 숫자 3개를 배열에 저장
    - 중복 제거
- Hitter - 타자 클래스
  - 사용자가 입력한 값 3개를 배열에 담는 용도
  - 멤버변수 - 요소 3개를 갖는 배열 (사용자가 입력한 값)



과제- 야구게임

- Umpire – 심판 클래스
  - 메서드
  - [1] 스트라이크
    - Pitcher 클래스와 Hitter 클래스의 멤버변수인 두 배열을 비교
    - 투수가 만든 수와 타자에서 입력 받은 수의 자리와 값이 같으면 스트라이크
      - 스트라이크 갯수를 리턴한다
  - [2] 볼
    - 자리가 다르고 값이 같으면 볼
      - 볼의 갯수를 리턴한다