

# 깊은 복사와 얇은 복사 (Deep Copy & Shallow Copy)

---

백엔드 데브코스 4기

---

황준호

---

GitHub : [juno-junho](#)

---

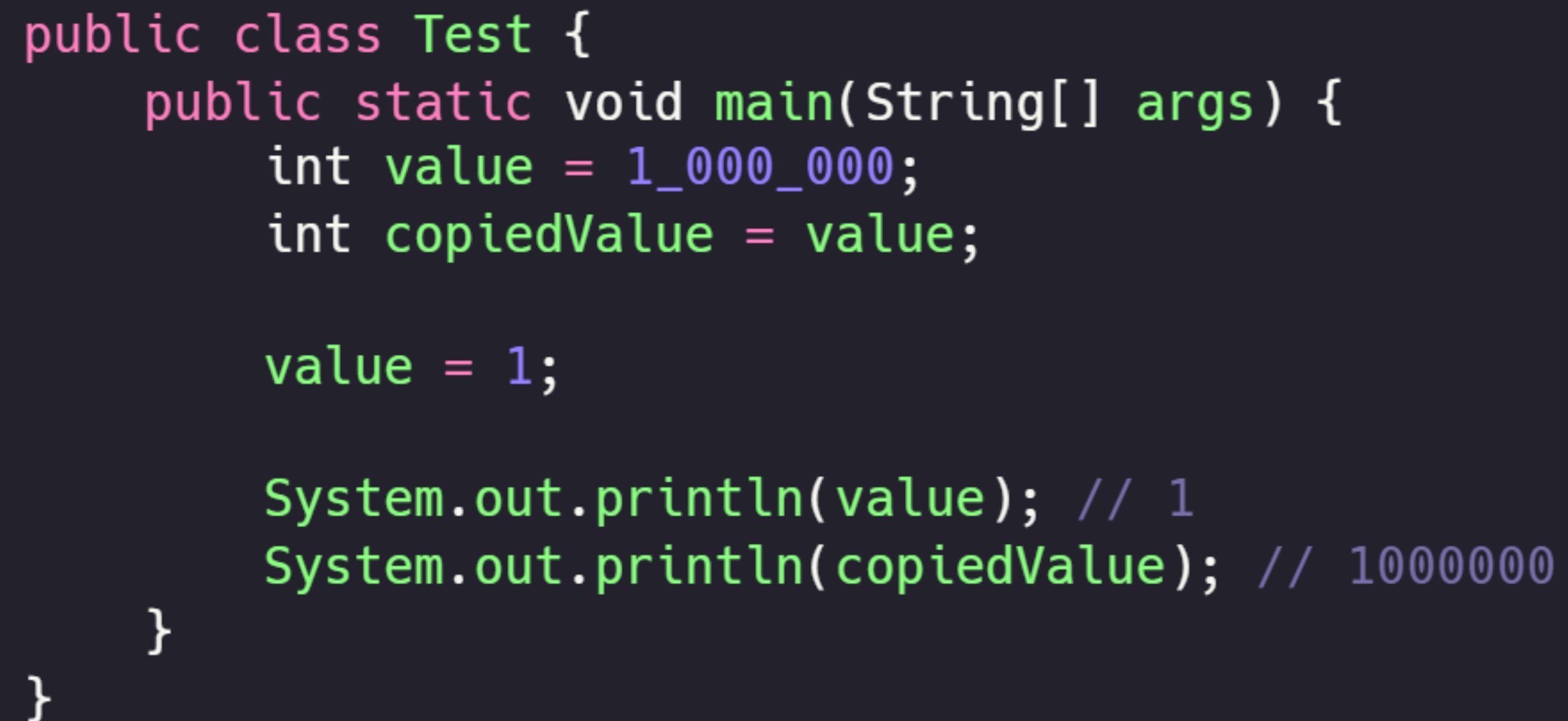
# Contents

1. Primitive Type과 Reference Type에서의 복사
2. 깊은 복사와 얕은 복사
3. Primitive Type 배열에서의 깊은 복사
4. 방어적 복사

# 1. Primitive Type과 Reference Type에서의 복사

## 1. Primitive Type에서의 복사

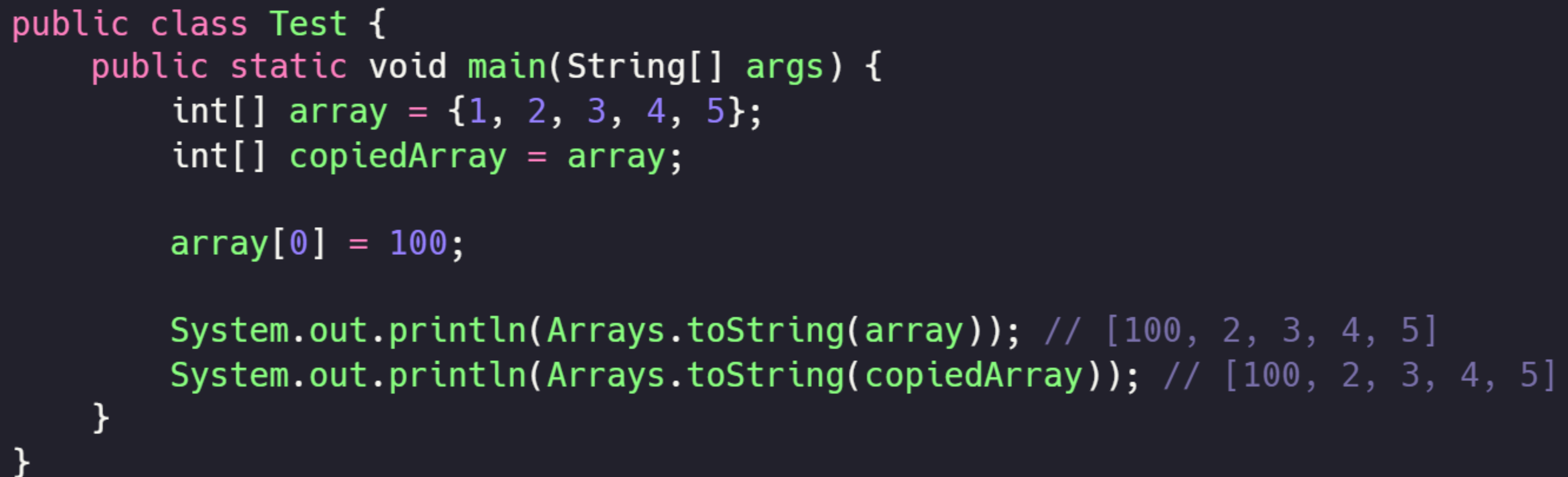
자바에서는 Primitive Type을 어떻게 복사할 수 있을까



```
public class Test {  
    public static void main(String[] args) {  
        int value = 1_000_000;  
        int copiedValue = value;  
  
        value = 1;  
  
        System.out.println(value); // 1  
        System.out.println(copiedValue); // 1000000  
    }  
}
```

## 2. Reference Type에서의 복사

같은 방식으로 Reference Type을 복사 할 수 있을까?



```
public class Test {  
    public static void main(String[] args) {  
        int[] array = {1, 2, 3, 4, 5};  
        int[] copiedArray = array;  
  
        array[0] = 100;  
  
        System.out.println(Arrays.toString(array)); // [100, 2, 3, 4, 5]  
        System.out.println(Arrays.toString(copiedArray)); // [100, 2, 3, 4, 5]  
    }  
}
```

왜 이런 결과값이 나올까요?

## 1. Primitive Type (원시타입)

- int, long, double, float, Boolean, byte, short, char (**총 8가지**)
- 자료형의 길이는 운영체제에 독립적이며 변하지 않는다.
- **Stack 메모리**에 저장된다.

= 을 통해 복사 가능!

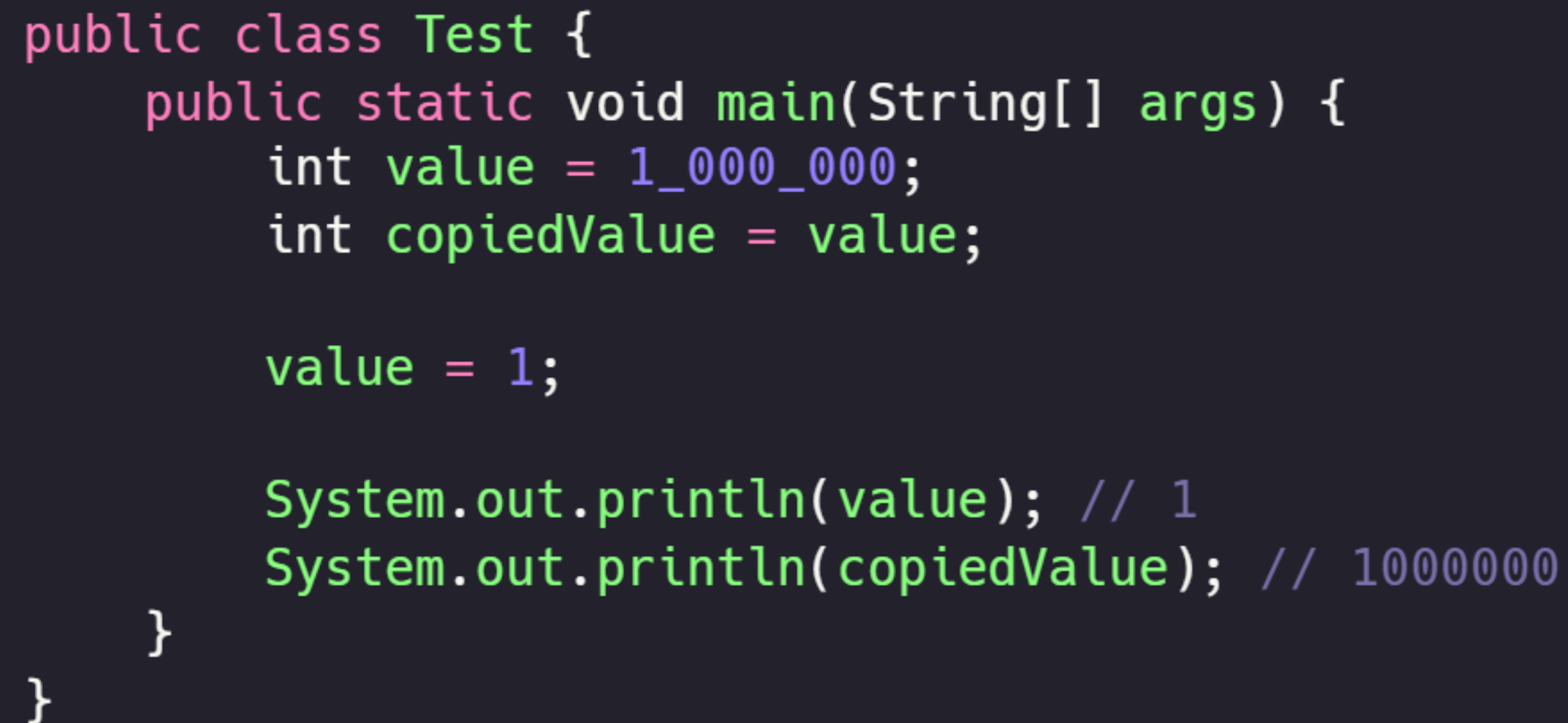
## 2. Reference Type (참조 타입)

- java.lang.Object 클래스를 상속하는 모든 클래스
- 위 Primitive Type을 제외한 타입 (String, 배열, Enum, 객체 ...)
- Garbage Collector의 대상
- 실제 객체는 **Heap 메모리에 저장**되며 **Stack 메모리의 참조 타입 변수**가 실제 객체들의 **주소를 저장**하여 객체를 사용할 때마다 참조 변수에 저장된 객체의 주소를 불러와 사용한다

= 을 통해 복사 불가능!

## 1. Primitive Type에서의 복사

자바에서는 Primitive Type을 어떻게 복사할 수 있을까

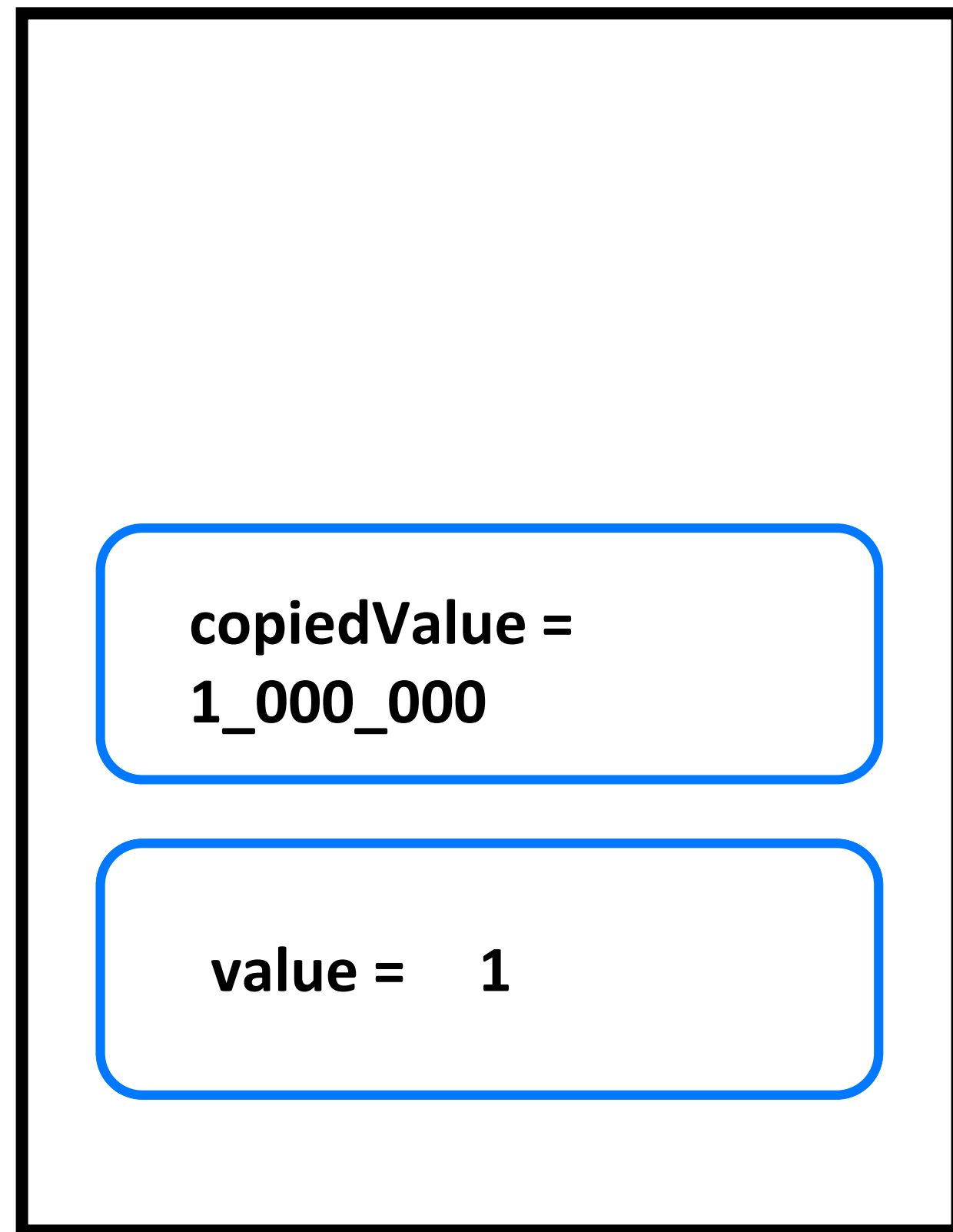


```
public class Test {  
    public static void main(String[] args) {  
        int value = 1_000_000;  
        int copiedValue = value;  
  
        value = 1;  
  
        System.out.println(value); // 1  
        System.out.println(copiedValue); // 1000000  
    }  
}
```

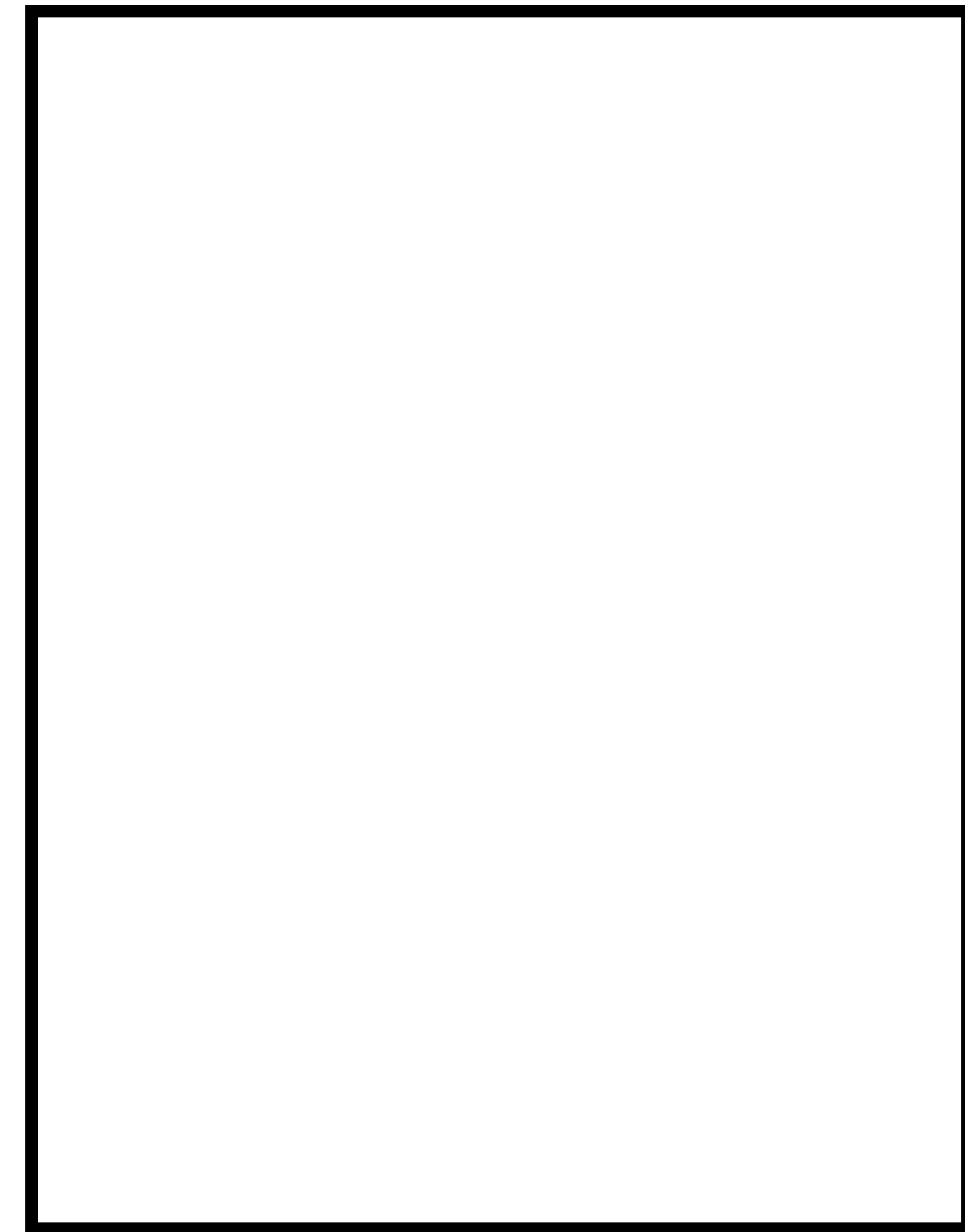


## 1. Primitive Type에서의 복사

Primitive Type에서 = 를 통한 복사 시 메모리 구조



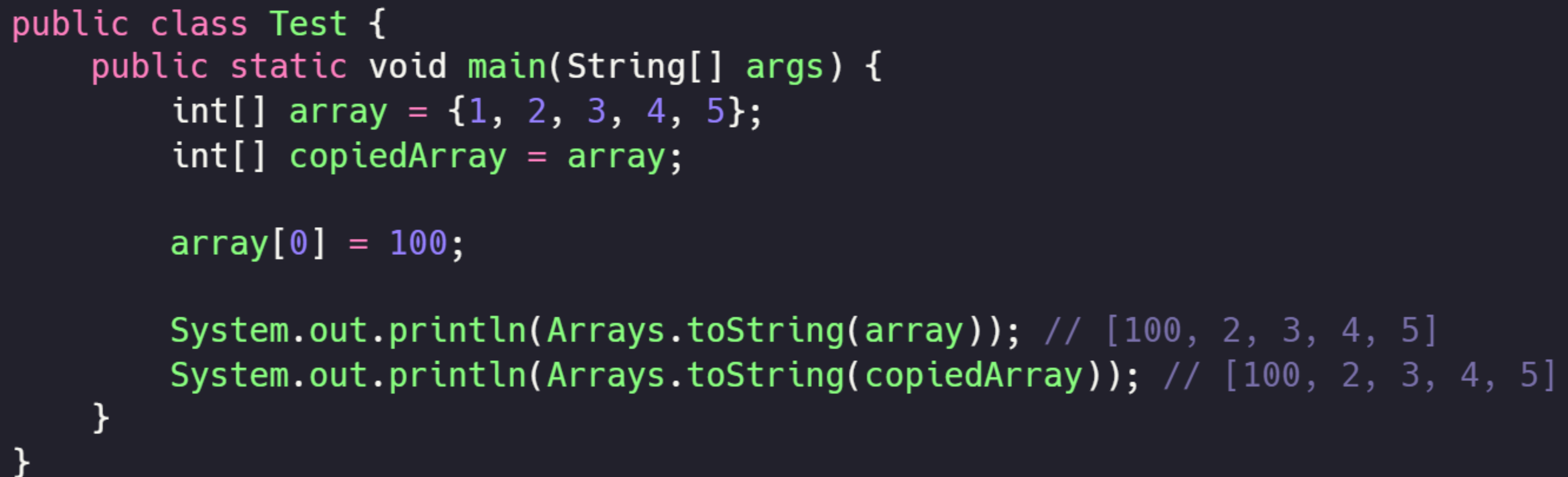
Stack



Heap

## 2. Reference Type에서의 복사

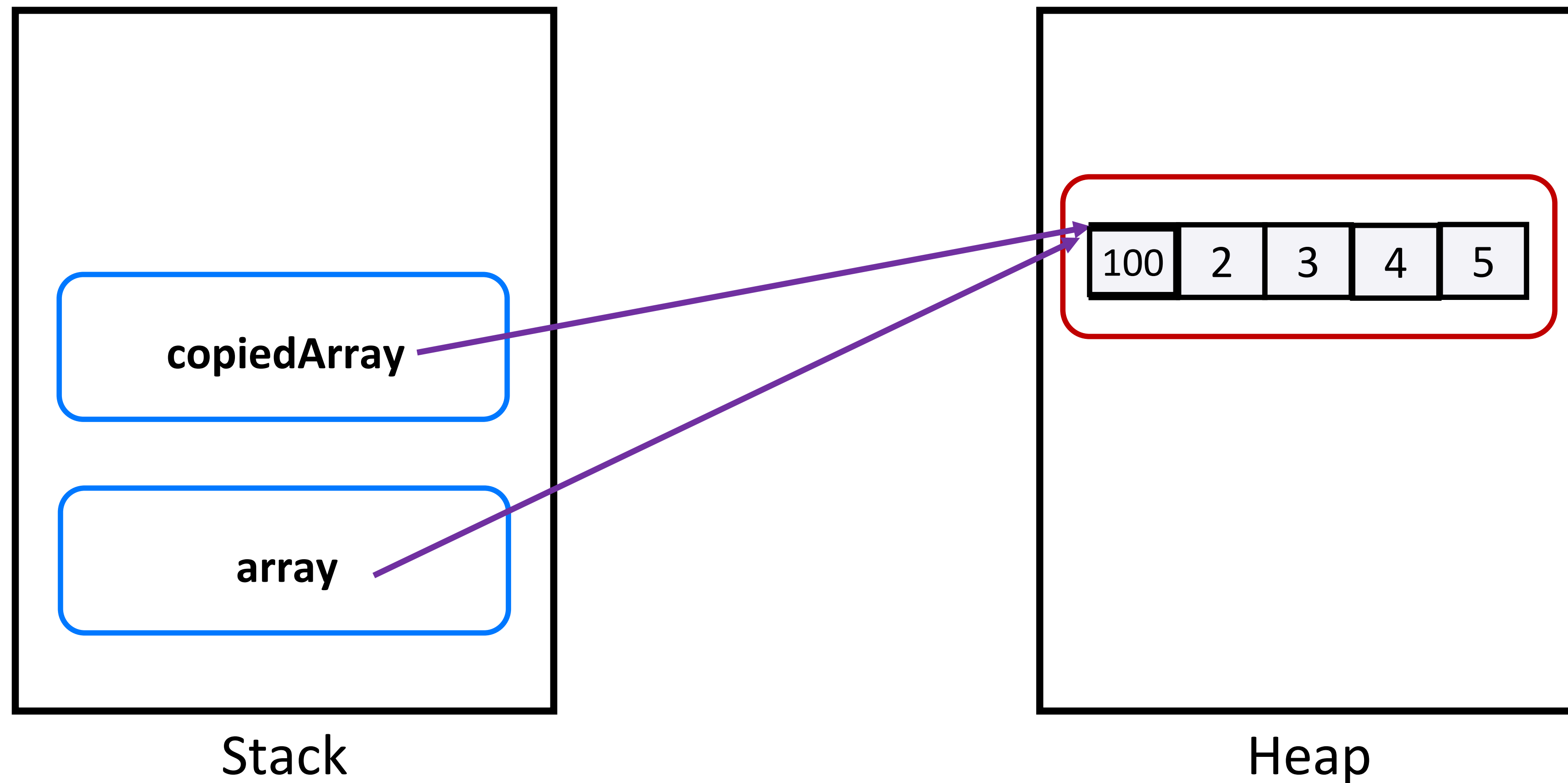
같은 방식으로 Reference Type을 복사 할 수 있을까?



```
public class Test {  
    public static void main(String[] args) {  
        int[] array = {1, 2, 3, 4, 5};  
        int[] copiedArray = array;  
  
        array[0] = 100;  
  
        System.out.println(Arrays.toString(array)); // [100, 2, 3, 4, 5]  
        System.out.println(Arrays.toString(copiedArray)); // [100, 2, 3, 4, 5]  
    }  
}
```

## 2. Reference Type에서의 복사

Primitive Type에서 = 를 통한 복사 시 메모리 구조



## 2. 깊은 복사와 얇은 복사

## 깊은복사와 얇은복사란?

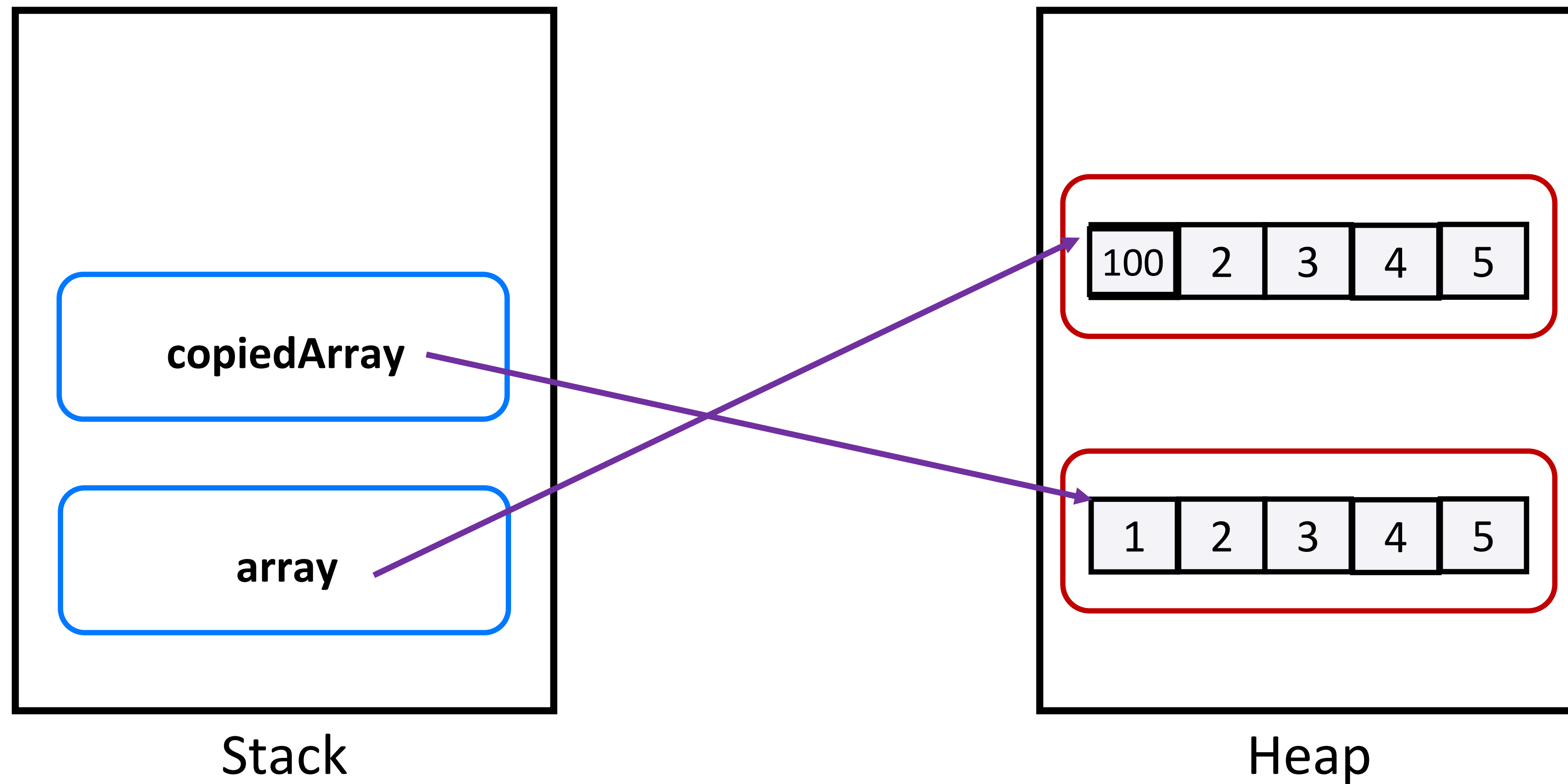
### 1. 얇은 복사

- = 를 통한 복사
- Stack 메모리의 참조 타입 변수를 만들고 원본 객체가 저장되어 있는 Heap 메모리의 주소 값을 참조하는 것.

### 2. 깊은 복사

- 전부를 복사하여 새 주소에 담기 때문에 참조를 공유하지 않는다.
- 깊은 복사를 하기 위해서는 Cloneable 인터페이스를 implement 해야하고 clone 메서드를 오버라이드 해야한다.

## 깊은 복사시 메모리 구조



### 3. Primitive Type 배열에서의 깊은 복사

## 1. For문 사용

## 2. System.arraycopy()

- Native call이라 빠르다
- 복사 할 배열을 생성해서 argument로 넘겨야 한다.
- 개인적으로 파라미터 개수가 너무 많아 사용하기 힘들다.

```
@Contract(mutates = "param3")
@IntrinsicCandidate
public static native void arraycopy( @NotNull @Flow(...) Object src, int srcPos,
                                     @NotNull Object dest, int destPos,
                                     int length);
```

## 3. Arrays.copyOf(), Arrays.copyOfRange()

- 내부적으로 **System.arraycopy()**를 사용한다.
- copyOf(int[] original, int newLength) → 특정 값 까지 복사할 수 있다.
- copyOfRange(int[] original, int from, int to) → 복사하는 배열의 시작과 끝 index를 설정할 수 있다.

## 4. Object 클래스의 clone()

- Clone() 을 통해 배열을 통째로 깊게 복사할 수 있다.



## Q1. 배열의 타입이 Primitive Type이 아닌 Reference Type일 경우 어떻게 해야 하는가?

- Arrays 클래스의 메서드와 clone()은 shallow copy
- 각 클래스 마다 clone() 을 구현하거나 SerializableUtils 클래스의 clone()을 통해 할 수 있다.

## Q2. 이차원 배열의 경우 어떻게 깊은 복사를 할 수 있는가?

- 이중 for문 사용
- For문 + clone 사용

## 4. 방어적 복사

## 방어적 복사란?

### 1. 방어적 복사

- 내부 객체를 반환할 때, 객체의 복사본을 만들어 반환하는 방법
- ⇒ 복사한 외부의 객체를 변경해도 원본 내부 객체가 변경되지 않는다
- ⇒ 불변을 유지할 수 있다.

## 계산기 미션의 MemoryCalculatorRepository

**일급 컬렉션을 예로 들어 봅시다!**

```
public class MemoryCalculatorRepository implements CalculatorRepository {  
  
    private final List<String> calculatedData;  
  
    public MemoryCalculatorRepository(List<String> calculatedData) {  
        this.calculatedData = calculatedData;  
    }  
  
    @Override  
    public void save(String calculatedResult) {  
        calculatedData.add(calculatedResult);  
    }  
  
    @Override  
    public List<String> findAll() {  
        return calculatedData;  
    }  
}
```

## 계산기 미션의 MemoryCalculatorRepository의 문제점

### MemoryCalculatorRepository의 문제점

```
public class MemoryCalculatorRepository implements CalculatorRepository {  
    private final List<String> calculatedData;  
  
    public MemoryCalculatorRepository(List<String> calculatedData) {  
        this.calculatedData = calculatedData;  
    }  
  
    @Override  
    public void save(String calculatedResult) {  
        calculatedData.add(calculatedResult);  
    }  
  
    @Override  
    public List<String> findAll() {  
        return calculatedData;  
    }  
}
```

1. 외부에서 주입하는

calculatedData가 바뀔 경우

2. findAll() 을 통해 반환한 List를

외부에서 변경하는 경우

## 계산기 미션의 MemoryCalculatorRepository의 문제점

### 문제점 해결 1. 외부 주입에 대한 참조 끊기

```
public class MemoryCalculatorRepository implements CalculatorRepository {  
    private final List<String> calculatedData;  
  
    public MemoryCalculatorRepository(List<String> calculatedData) {  
        this.calculatedData = new ArrayList<>(calculatedData);  
    }  
  
    @Override  
    public void save(String calculatedResult) {  
        calculatedData.add(calculatedResult);  
    }  
  
    @Override  
    public List<String> findAll() {  
        return calculatedData;  
    }  
}
```

1. new ArrayList<>(calculatedData)

2. List.copyOf(calculatedData)

⇒ 내부 ImmutableList.listCopy 반환

## 계산기 미션의 MemoryCalculatorRepository의 문제점

### 문제점 해결 2. 새로운 객체로 감싸서 복사해 반환하기

```
public class MemoryCalculatorRepository implements CalculatorRepository {  
    private final List<String> calculatedData;  
  
    public MemoryCalculatorRepository(List<String> calculatedData) {  
        this.calculatedData = new ArrayList<>(calculatedData);  
    }  
  
    @Override  
    public void save(String calculatedResult) {  
        calculatedData.add(calculatedResult);  
    }  
  
    @Override  
    public List<String> findAll() {  
        return Collections.unmodifiableList(calculatedData);  
    }  
}
```

1. new ArrayList<>(calculatedData)

⇒ 복사한 리스트 수정 가능한 문제점

2. Collections.unmodifiableList

⇒ 원본 리스트대로 변경되지만

외부에서 변경할 수는 없음.

감사합니다.



프로그래머스 백엔드 데브코스 4기  
황준호

GitHub: juno-junho