

자바의 메모리 모델과 Object 클래스

19-1.

자바 가상머신의 메모리 모델

운영체제 입장에서 자바 가상머신

운영체제의 관점에서는 가상머신도 그냥 프로그램의 하나.

운영체제가 일반 프로그램에게 4G의 메모리 공간을 할당해준다면,
JVM에게도 4G 메모리 공간을 할당해준다.

자바 프로그램이 두 개 실행되면, 가상머신도 두 개가 실행된다.

이는 메모장을 두 번 띄우면 두 개의 메모장 프로그램이 실행되는 이치와 같다.

자바 가상머신의 메모리 모델

- 메소드 영역 (Method Area)

메소드의 바이트코드, static 변수

- 스택 영역 (Stack Area)

지역변수, 매개변수

- 힙 영역 (Heap Area)

인스턴스

메모리 공간 활용의 효율성을 높이기 위해 메모리 공간을 이렇듯 세 개의 영역으로 구분하였다.



메소드 영역

```
class Boy {  
    static int average = 0;  
    public void Run() {...}  
}
```

```
class MyMain {  
    public static void main(String[] args) {  
        Boy b = new Boy();    // 인스턴스 생성  
        Boy.average += 5;    // 클래스 변수 접근  
        ....  
    }  
}
```

메소드 영역 : 바이트코드와 static 변수가 할당되는 메모리 공간
이 영역에 저장된 내용은 프로그램 종료 시 소멸된다.

스택 영역

```
public static void main(String[ ] args) {  
    int num1 = 10;  
    int num2 = 20;  
    adder(num1, num2);  
    System.out.println("end of program");  
}
```

```
public static void adder(int n1, int n2) {  
    int result = n1 + n2;  
    return result;  
}
```

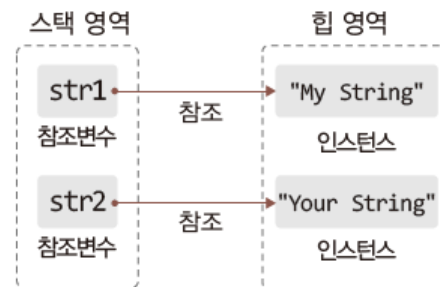
스택 영역 : 지역변수 매개변수 할당되는 영역

이 영역에 저장된 변수는 해당 변수가 선언된 메소드 종료 시 소멸된다.

힙 영역

```
public static void simpleMethod() {  
    String str1 = new String("My String");  
    String str2 = new String("Your String");  
    ....  
}
```

힙 영역 : 인스턴스가 저장되는 영역
가비지 컬렉션의 대상이 되는 영역이다.



자바 가상머신의 인스턴스 소멸 시기

```
public static void simpleMethod() {  
    String str1 = new String("My String");  
    String str2 = new String("Your String");  
    ....  
    str1 = null;    // 참조 관계 소멸  
    str2 = null;    // 참조 관계 소멸  
    ....  
}
```



참조 관계가 끊어진 인스턴스는 접근이 불가!
따라서 가비지 컬렉션의 대상이 된다.

19-2. Object 클래스

Object 클래스의 finalize 메소드

```
protected void finalize() throws Throwable
```

Object 클래스에 정의되어 있는 이 메소드는 인스턴스 소멸 시 자동으로 호출이 된다.

자식 클래스에서 오버라이딩 할 수 있음.

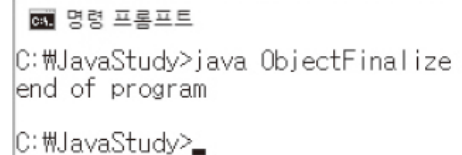
finalize 메소드의 오버라이딩 예

```
class Person {
    String name;
    public Person(String name) {
        this.name = name;
    }

    @Override
    protected void finalize() throws Throwable {
        super.finalize(); // 상위 클래스의 finalize 메소드 호출
        System.out.println("destroyed: " + name);
    }
}
```

```
public static void main(String[] args) {
    Person p1 = new Person("Yoon");
    Person p2 = new Person("Park");
    p1 = null; // 참조대상을 가비지 컬렉션의 대상으로 만들
    p2 = null; // 참조대상을 가비지 컬렉션의 대상으로 만들
    // System.gc();
    // System.runFinalization();

    System.out.println("end of program");
}
```



```
C:\JavaStudy>java ObjectFinalize
end of program
C:\JavaStudy>
```

인스턴스의 비교: equals 메소드

```
class INum {  
    private int num;  
    public INum(int num) {  
        this.num = num;  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        if(this.num == ((INum)obj).num)  
            return true;  
        else  
            return false;  
    }  
}
```

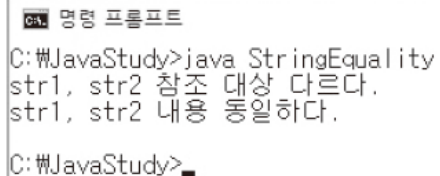
```
public static void main(String[] args) {  
    INum num1 = new INum(10);  
    INum num2 = new INum(12);  
    INum num3 = new INum(10);  
  
    if(num1.equals(num2))  
        System.out.println("num1, num2 내용 동일하다.");  
    else  
        System.out.println("num1, num2 내용 다르다.");  
  
    if(num1.equals(num3))  
        System.out.println("num1, num3 내용 동일하다.");  
    else  
        System.out.println("num1, num3 내용 다르다.");  
}
```

인스턴스의 내용 비교를 위한 기능을 equals 메소드에 담아 정의한다.

equals는 Object 클래스의 메소드이다.

String 클래스의 equals 메소드

```
public static void main(String[] args) {  
    String str1 = new String("So Simple");  
    String str2 = new String("So Simple");  
  
    // 참조 대상을 비교하는 if ~ else문  
    if(str1 == str2)  
        System.out.println("str1, str2 참조 대상 동일하다.");  
    else  
        System.out.println("str1, str2 참조 대상 다르다.");  
  
    // 두 인스턴스 내용 비교하는 if ~ else문  
    if(str1.equals(str2))  
        System.out.println("str1, str2 내용 동일하다.");  
    else  
        System.out.println("str1, str2 내용 다르다.");  
}
```



```
C:\JavaStudy>java StringEquality  
str1, str2 참조 대상 다르다.  
str1, str2 내용 동일하다.  
C:\JavaStudy>
```

String 클래스는 내용 비교를 하는 형태로 equals 메소드를 오버라이딩 하고 있음

인스턴스 복사: clone 메소드

protected Object **clone**() throws CloneNotSupportedException

Object 클래스에 정의되어 있는 clone 메소드가 호출되면 인스턴스의 복사가 이뤄진다.

클래스 정의 시, clone 메소드의 호출을 허용하려면 **Cloneable** 인터페이스를 구현해야 한다.

Cloneable 인터페이스는 구현해야 할 추상 메소드가 없는 **마커 인터페이스**이다.

clone 메소드 호출의 예

```
class Point implements Cloneable {
    private int xPos;
    private int yPos;

    public Point(int x, int y) {
        xPos = x;
        yPos = y;
    }

    public void showPosition() {
        System.out.printf("[%d, %d]", xPos, yPos);
        System.out.println();
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone(); // Object 클래스의 clone 메소드 호출
    }
}

class InstanceCloning {
    public static void main(String[] args) {
        Point org = new Point(3, 5);
        Point cpy;

        try {
            cpy = (Point)org.clone();
            org.showPosition();
            cpy.showPosition();
        }
        catch(CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

접근 수준 지시자를 `protected`에서 `public`으로 바꾸기 위한 메소드 오버라이딩

Shallow Copy

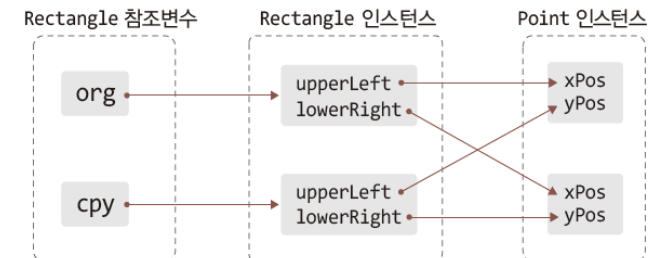
```
class Rectangle implements Cloneable {
    private Point upperLeft; // 좌측 상단 좌표
    private Point lowerRight; // 우측 하단 좌표

    public Rectangle(int x1, int y1, int x2, int y2) {
        upperLeft = new Point(x1, y1);
        lowerRight = new Point(x2, y2);
    }
    . . . .
    @Override
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
    . . . .
}

class Point implements Cloneable
{
    private int xPos;
    private int yPos;
    . . . .
}
```

```
public static void main(String[] args) {
    Rectangle org = new Rectangle(1, 1, 9, 9);
    Rectangle cpy;

    try {
        cpy = (Rectangle)org.clone();
        . . . .
    }
    catch(CloneNotSupportedException e) {
        e.printStackTrace();
    }
}
```



Deep Copy

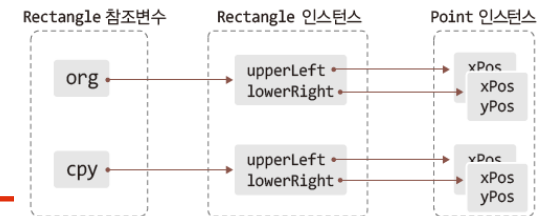
```
class Rectangle implements Cloneable {  
    private Point upperLeft; // 좌측 상단 좌표  
    private Point lowerRight; // 우측 하단 좌표  
  
    public Rectangle(int x1, int y1, int x2, int y2) {  
        upperLeft = new Point(x1, y1);  
        lowerRight = new Point(x2, y2);  
    }  
    . . . .  
    @Override  
    public Object clone() throws CloneNotSupportedException  
    {  
        return super.clone();  
    }  
    . . . .  
}
```

얕은 복사 clone

깊은 복사 clone

```
@Override  
public Object clone() throws CloneNotSupportedException {  
    Rectangle copy = (Rectangle)super.clone();  
  
    copy.upperLeft = (Point)upperLeft.clone();  
    copy.lowerRight = (Point)lowerRight.clone();  
  
    return copy;  
}
```

```
public static void main(String[] args) {  
    Rectangle org = new Rectangle(1, 1, 9, 9);  
    Rectangle cpy;  
    . . .  
    cpy = (Rectangle)org.clone();  
    . . . .  
}
```



String 인스턴스 대상 깊은 복사

```
class Person implements Cloneable {  
    private String name;    // String 클래스는 Cloneable 구현 안함!  
    private int age;  
    ....  
}
```

깊은 복사 clone

```
@Override  
public Object clone() throws CloneNotSupportedException {  
    Person cpy = (Person)super.clone();    // clone 메소드 호출을 통한 복사본 생성  
    cpy.name = new String(name);    // 깊은 복사의 형태로 복사본을 완성  
    return cpy;    // 완성된 복사본의 참조 값 반환  
}
```

clone 메소드의 호출과 형 변환

```
class Point implements Cloneable {  
    . . . .  
    @Override  
    public Object clone() throws CloneNotSupportedException {  
        return super.clone();  
    }  
}
```

```
Point org = new Point(1, 2);  
Point cpy = (Point)org.clone();    // 형 변환해야 함
```

clone 메소드의 반환형 수정

```
class Point implements Cloneable {  
    ....  
    @Override  
    public Point clone() throws CloneNotSupportedException {  
        return (Point)(super.clone());  
    }  
}
```

```
Point org = new Point(1, 2);  
Point cpy = org.clone();    // 형 변환 필요 없음
```