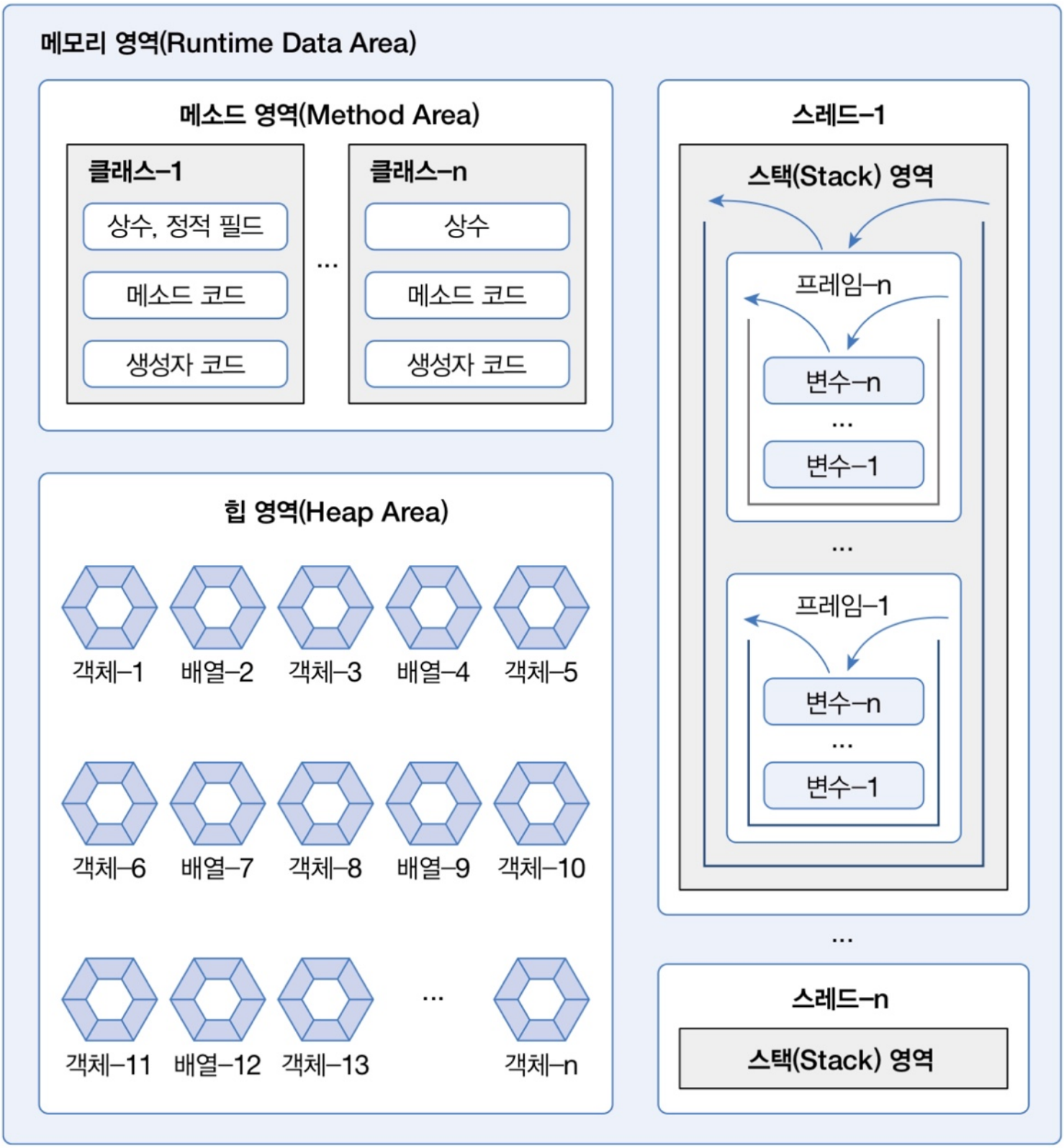
4th-week
메모리 사용 영역

> JVM이 구동되면 JVM은 운영체제에서 할당받은 메모리 영역(Runtime Data Area)을 다음과 같이 구분해서 사용한다.



- `메소드 영역` : 바이트코드 파일을 읽은 내용이 저장되는 영역. 클래스 별로 상수, 정적 필드, 메소드 코드, 생성자 코드 등이 저장된다.
- `힙(heap) 영역` : 객체가 생성되는 영역. 객체의 번지는 메소드 영역과 스택 영역의 상수와 변수에서 참조할 수 있다.
- `스택(stack) 영역` : 메소드를 호출할 때마다 생성되는 프레임(Frame)이 저장되는 영역. 메소드 호출이 끝나면 프레임은 자동 제거 된다. 프레임 내부에는 로컬 변수 스택이 있다. 여기에서 기본 타입 변수와 참조 타입 변수가 생성되고 제거된다.

기본 타입 vs 참조 타입

기본 타입(primitive type)

- 정수 타입(byte, char, short, int, long), 실수 타입(float, double), 논리 타입(boolean)
- 값 자체를 변수에 저장.
- 변수 선언시 기본 타입은 소문자로 시작
- 자바가 기본으로 제공하는 타입 - 개발자가 수정 불가
- 산술 연산을 수행할 수 있다.
- null을 할당할 수 없다.

참조 타입(reference type)

- 배열 타입, 열거 타입, 클래스, 인터페이스(기본 타입 외에 모든 타입)
- 객체가 저장된 메모리의 위치를 가르키는 참조값(주소값)을 저장
- 클래스는 대문자로 시작한다. (`String`타입도 참조 타입)
- 개발자가 직접 정의 가능
- 산술 연산을 수행할 수 없다.
- null을 할당할 수 있다.

참조 타입 변수의 ==, != 연산

> 참조 타입 변수의 값은 객체의 번지(주소)이므로 `==`, `!=` 연산자는 번지(주소)를 비교하는 것이다.

```
Ref ref1 = new Ref();
Ref ref2 = ref1;
Ref ref3 = new Ref();
System.out.println("ref1의 참조값 : " + ref1);
System.out.println("ref2의 참조값 : " + ref2);
System.out.println("ref3의 참조값 : " + ref3);
>>ref1의 참조값 : x001
>>ref2의 참조값 : x001
>>ref3의 참조값 : x003
>>ref1 == ref2  // true
>>ref1 != ref2  // false
```

```
>>ref2 == ref3  // false - ref3 : x002
>>ref2 != ref3  // true
```

```
int[] arr1 = new int[]{1,2,3};
int[] arr2 = new int[]{1,2,3};
int[] arr3 = arr2;
System.out.println(arr1 == arr2);
System.out.println(arr2 == arr3);
>> false
>> true
```

변수와 초기화

변수 대입

> 자바는 항상 변수의 값을 복사해서 대입한다.(기본형, 참조형 둘다 해당됨)

기본 타입의 변수 대입은 값을 복사해서 전달한다. → 기본 타입 변수를 대입해서 각각 변경해도 서로 영향을 주지 않음
참조 타입의 변수 대입은 변수가 가르키는 참조값만 복사해서 전달한다. → 참조 타입 변수의 변수 대입은 번지를 전달하기 때문에 각각 변경하면 번지 안에 값이 변경됨.

```
int a = 10;
int b = a;
System.out.println("a = " + a);    >> 10
System.out.println("b = " + b);    >> 10

// a 변경
a = 20;
System.out.println("변경 a = 20");
System.out.println("a = " + a);    >> 20
System.out.println("b = " + b);    >> 10

// b 변경
b = 30;
System.out.println("변경 b = 30");
System.out.println("a = " + a);    >> 20
System.out.println("b = " + b);    >> 30
```

```
Data dataA = new Data();
dataA.value = 10;
Data dataB = dataA;

System.out.println("dataA 참조값= " + dataA);  >> x001
System.out.println("dataB 참조값= " + dataB);  >> x001
System.out.println("dataA.value = " + dataA.value);  >> 10
System.out.println("dataB.value = " + dataB.value);  >> 10
```

```
// dataA 변경
dataA.value = 20;
System.out.println("변경 dataA.value = 20");
System.out.println("dataA.value = " + dataA.value);  >> 20
System.out.println("dataB.value = " + dataB.value);  >> 20

// dataB 변경
dataB.value = 30;
System.out.println("변경 dataB.value = 30");
System.out.println("dataA.value = " + dataA.value);  >> 30
System.out.println("dataB.value = " + dataB.value);  >> 30
```

메서드 호출

메서드를 호출할 때 사용하는 매개변수(파라미터)도 결국 변수일 뿐이다. 이 변수의 값이 실제 값인지 참조(메모리 주소)값인지에 따라 동작하는게 달라진다.

기본형

- 메서드로 기본형 데이터를 전달하면, 해당 값이 복사되어 전달된다.
- 이 경우, 메서드 내부에서 매개변수(파라미터)의 값을 변경해도, 호출자의 변수 값에는 영향이 없다.

```
public static void main(String[] args){
        int a = 10;
        System.out.println("메서드 호출 전: a = " + a);  >> 10
        changePrimitive(a);
        System.out.println("메서드 호출 후: a = " + a);  >> 10
}
static void changPrimitive(int x){
        x = 20;
}
```

참조형

- 메서드로 참조형 데이터를 전달하면, 참조값이 복사되어 전달된다.
- 이 경우, 메서드 내부에서 매개변수(파라미터)로 전달된 객체의 멤버 변수를 변경하면, 호출자의 객체도 변경된다.
- 참조형은 메서드를 호출할 때 참조값을 전달한다. 따라서 메서드 내부에서 전달된 참조값을 통해 객체의 값을 변경하거나, 값을 읽어서 사용할 수 있다.

```
public static void main(String[] args){
        Data dataA = new Data();
        dataA.value = 10;
        System.out.println("메서드 호출 전: dataA.value = " + dataA.value);  >> 10
        changePrimitive(dataA);
        System.out.println("메서드 호출 후: dataA.value = " + dataA.value);  >> 20
}
static void changPrimitive(Data dataX){
        dataX.value = 20;
}
```

변수의 값 초기화

- 멤버 변수 : 자동 초기화
  - 인스턴스의 멤버 변수는 인스턴스를 생성할 때 자동으로 초기화 된다.
  - 숫자(int) = 0, boolean = false, 참조형 = null(null 값은 참조할 대상이 없다는 뜻으로 사용된다.)
  - 개발자가 초기값을 직접 지정할 수 있다.
- 지역 변수 : 수동 초기화
  - 지역 변수는 항상 직접 초기화해야 한다.

```
public class InitDate{
        int value1;  //초기화 하지 않음
        int value2= 10;  // 10으로 초기화
}
```

```
public class InitMain{
        public static void main(String[] args){
                InitData data = new InitData();
                System.out.println("value1 = " + data.value1);
                System.out.println("value2 = " + data.value2);
        }
}
```

null값 할당

기본형 변수는 null을 할당할 수 없지만, 참조형 변수는 null을 할당할 수 있다.

참조형 변수에는 객체가 있는 위치를 가르키는 참조값이 들어간다. 그런데 아직 번지를 저장하고 있지 않다는 뜻으로 null을 넣을 수 있다.

Garbage Collector

> 자바는 어떤 변수에서도 객체를 참조하지 않으면 해당 객체를 자동으로 제거하는 GC기능이 있다.

어떤 변수도 해당 객체를 참조하지 않으면 Garbage Colletcor를 실행 시켜 자동으로 제거한다.
자바에서는 직접 객체를 제거하는 방법을 제공하지 않는다.따라서 자바에서 객체를 제거하는 유일한 방법은 객체의 모든 참조를 없애는 것이다.

```
String hobby= "여행";
hobby = null;  //  "여행"에 해당하는 String 객체가 GC에 의해 제거됨.

String kind1 = "자동차";
String kind2 = kind1;  // kind1 변수에 저장되어 있는 번지를 kind2 변수에 대입
kind1 = null; // "자동차"에 해당하는 String 객체가 쓰레기가 되지는 않음.
```

위에 코드처럼 kind1 변수를 null로 처리해도 "자동차" 객체는 여전히 kind2에서 참조하고 있기 때문에 해당 객체는 제거되지 않는다.

NullPointerException

> 변수가 null인 상태에서 객체의 데이터나 메소드를 사용하려 할 때 발생하는 예외

참조 변수에 멤버 변수나 메소드를 사용할 경우 변수에 `.`을 찍어 사용하는데 이 경우 해당 참조 변수가 null인데 `.`을 찍으면 `NullPointerException` 예외가 발생한다.

```
public class NullTest{}
        public static void main(String[] args) {
                BigData bigData = new BigData();
                System.out.println("bigData.count=" + bigData.count);
                System.out.println("bigData.data=" + bigData.data);

                //NullPointerException
                System.out.println("bigData.data.value=" + bigData.data.value);
        }
}
class Data{
        int value;
}
class BigData {
        Data data;
        int count;
}
```

```
bigData.count=0
bigData.data=null
Exception in thread "main" java.lang.NullPointerException: Cannot read field "value" b
        at LogicTest/Math1.BJ1193.main(BJ1193.java:14)
```

위 예제 에러를 해결하려면 BigData.data 멤버 변수에 참조값을 할당하면 된다.

```
public class NullTest{}
        public static void main(String[] args) {
                BigData bigData = new BigData();
                bigData.data = new Data();
                System.out.println("bigData.count=" + bigData.count);
                System.out.println("bigData.data=" + bigData.data);
                System.out.println("bigData.data.value=" + bigData.data.value);
        }
}
```

```
bigData.count=0
bigData.data=Math1.Data@2d209079
bigData.data.value=0
```