객체 지향 프로그래밍

객체 지향 프로그래밍

- **객체 지향 프로그래밍(Object-Oriented Programming, OOP)**

  프로그래밍에서 필요한 데이터들을 추상화 시켜 객체를 만들고, 객체들간의 상호작용을 통해 프로그램을 개발하는 프로그래밍 패러다임 중의 하나

- **자바 객체 지향 개념에서의 핵심 4가지**

  1. 캡슐화
  2. 상속
  3. 추상화
  4. 다형성

객체지향의 핵심 개념 4가지 - 1

- **캡슐화(Encapsulation)**

변수와 메서드를 클래스로 묶어 독립적으로 동작하지 않도록 하거나 불필요한 정보를 노출시키지 않는 개념

1. 코드의 유지보수성을 향상
2. 객체의 내부 구현을 외부로부터 숨김 (정보 은닉)
3. 객체의 내부 상태를 제어하고, 잘못된 접근으로부터 보호함

- **상속(Inheritance)**

부모 클래스가 가지고 있는 것을 자식 클래스가 물려받아 확장(extends)하는 개념

1. 코드의 재사용성을 높이고, 중복을 최소화
2. 계층적인 구조를 통해 객체 간의 관계를 나타낼 수 있음

객체지향의 핵심 개념 4가지 - 2

- **추상화(Abstraction)**

  구체적인 사실들을 일반화시켜 기술하는 개념으로써 필요한 공통점을 추출하고
  불필요한 공통점을 제거하는 과정

  1. 복잡한 시스템을 단순화하고 모델링이 가능

- **다형성(Polymorphism)**

  여러가지의 형태를 가질 수 있는 개념
  (오버로딩, 오버라이딩, ...)

  1. 코드의 재사용성을 높이고, 중복을 최소화
  2. 계층적인 구조를 통해 객체 간의 관계를 나타낼 수 있음

클래스

클래스

- **클래스(Class)**

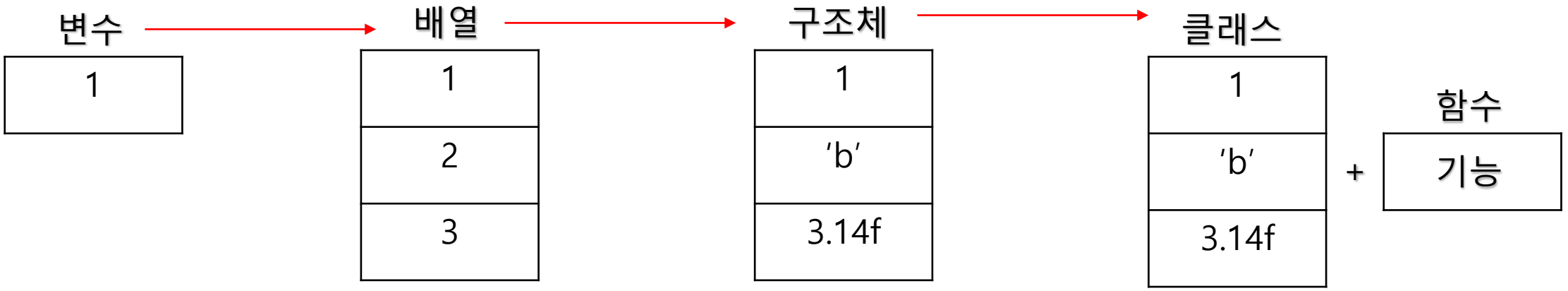
  개발자가 어떠한 프로그램을 제작하기 위한 설계도(틀)

  ex) 자동차 설계도, TV 설계도, 마이크 설계도, ...

클래스의 유래

- **클래스의 유래**

1. 변수 : 한가지의 데이터를 저장하는 공간
2. 배열 : 같은 타입을 가진 데이터를 저장하는 공간
3. 구조체 : 서로 다른 타입을 가진 데이터를 저장하는 공간
4. 클래스 : 서로 다른 타입을 가진 데이터와 특정 기능을 실행하는 함수의 결합



클래스 선언 방식

- 클래스(Class)

```
[접근제한자] [예약어] class 클래스명 {
    [접근제한자] [예약어] 자료형 변수명;

    [접근제한자] 생성자명() { };

    [접근제한자] 반환타입 메소드명(매개변수) {
        실행될 코드
}
```

클래스 접근제한자-1

- 참고

자바에서 public 클래스는 소스파일에서 하나만 가질 수 있음
* 접근 제한자는 캡슐화 개념중의 하나이다.

제어자	같은 패키지에서만 접근 가능	전체 패키지에서 접근 가능
public	O	O
(default)	O	

클래스 접근제한자-2

- 클래스 접근제한자 사용 예시

```
public class ExClass {
    // 실행될 코드
}

class ExClass {
    // 실행될 코드
}
```

여러 개의 클래스 작성

- **하나의 소스파일에서 여러 개의 클래스를 작성할 때 주의점**

1. public class가 있는 경우에는 반드시 소스파일과 클래스의 이름이 일치해야 함
   * 대소문자 구별
2. public class가 없는 경우에는 소스파일의 이름을 클래스 이름 중 하나로 가능
3. 하나의 소스파일에 2개의 public class를 가질 수 없음

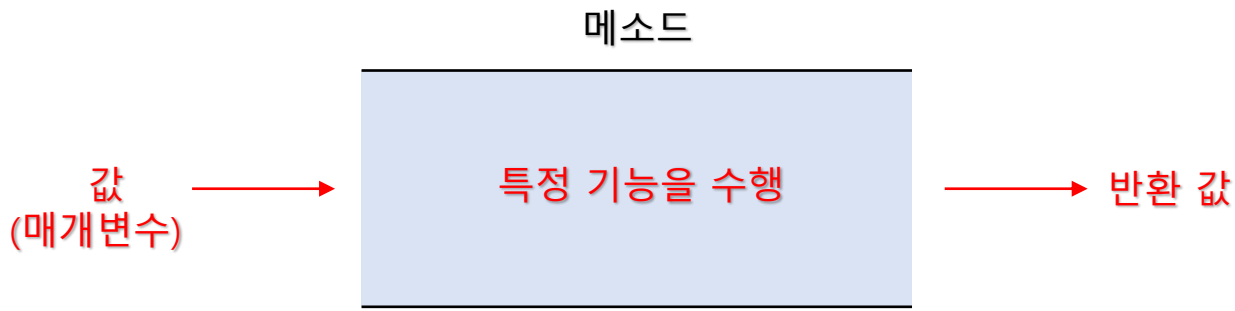
메소드

메소드

- **메소드(Method)**

클래스 내부에서 정의된 동작이나 기능을 수행하는 역할로써 어떠한 값을 전달하여 특정 작업을 수행하고 값을 반환할 수 있음

* 값의 반환은 필수가 아님



메소드 선언 방식

- 메소드(Method)

```
[접근제한자] [예약어] 반환명 메소드명([매개변수타입] [매개변수이름]) {
    실행될 코드
}
```

메소드 접근제한자

- 참고

* 접근 제한자는 캡슐화 개념중의 하나이다.

제어자	같은 클래스	같은 패키지	자식 클래스	전체
public	O	O	O	O
protected	O	O	O	
(default)	O	O		
private	O			

메소드 예약어

- **메소드 예약어**

메소드 예약어는 개발자가 사용하려는 용도에 따라 선택이 가능

즉, 변수의 상수와 같이 한번 만들어진 메소드를 변경할 수 없도록 제한하는 것이다.

예약어	설명
static	정적 메소드 : 객체의 인스턴스를 생성하지 않고 클래스이름으로 직접 호출 가능
final	하위 클래스에서 변경할 수 없도록 제한하며 오버라이딩을 방지함 (상수)
static final	static + final
abstract	추상 메소드 : 미완성된 메소드로써 상속받은곳에서 오버라이딩을 사용하여 완성시켜야함

메소드 반환형

- **메소드 반환형**

메소드에서 코드가 실행되고 어떠한 값을 반환하고 싶을 때 사용되며, 반환은 필수가 아님

반환형	설명	예시
기본 타입	기본형 타입을 반환	int, double, Boolean, …
클래스 타입	객체(인스턴스)를 반환	String, List, …
배열	배열을 반환	int[], String[][], …
void	반환할 값이 없음	

메소드 매개변수 - 1

- **메소드 매개변수**

  메소드에 전달되는 값의 정보를 담는 변수

- **예시**

```
public static void parameterTest1(int a) {
    // 실행될 코드
};
```

메소드 매개변수 - 2

- 주의 사항

1. **기본 자료형**의 경우 메소드 안에서 값을 변경해도 원본 변수는 변경되지 않음
2. **배열과 클래스**는 주소 값을 전달하므로 메소드 안에서 수정할 경우 원본 데이터가 수정됨

* 스택 영역 메모리에 어떠한 값이 들어 있느냐의 차이

- 참고

기본형 매개변수와 참조형 매개변수의 자세한 예시는 호출 스택을 배운 후 다시 언급

메소드 매개변수 - 3

- 주의 사항 예시

```
public static void main(String[] args) {
    int a = 2;
    int[] b = {1, 2};

    F_Product p = new F_Product();
    System.out.println("메소드 호출 전 : "+a);
    p.exVar(a);
    System.out.println("메소드 호출 후 : "+a);
    System.out.println("================");
    for(int i=0; i<b.length; i++) {
        System.out.print("메소드 호출 전 : "+b[i]+" ");
    }
    System.out.println();
    p.exArr(b);
    for(int i=0; i<b.length; i++) {
        System.out.print("메소드 호출 후 : "+b[i]+" ");
    }
}

public int exVar(int a) {
    a = 100;
    return a;
}
public int[] exArr(int[] b) {
    b[0] = 1000;
    b[1] = 2000;
    return b;
}
```

메소드 매개변수 - 3

- **참고**

String 타입은 참조형(주소 값을 가지고 있음) 변수이기 때문에 데이터가 수정될 것 같지만,
불변 타입이기 때문에 원본 데이터가 수정되지 않음.

* 객체를 수정하는 연산을 수행하면 새로운 String 객체가 생성됨(변경이 아님)

메소드 매개변수 - 4

- **가변인자**

매개변수의 개수를 유동적으로 설정하는 방법
가변인자는 배열로 취급되며, 매개변수 가장 마지막에 설정해야함

- **예시**

타입뒤에 ... 을 붙여 사용하며 for문의 경우 '향상된 for문' 또는 'for-each문' 이라고 불림

```
printNumbers(1, 2, 3, 4, 5);
printNumbers(10, 20);
printNumbers(1);
```



```
public static void printNumbers(int a, int... numbers) {
    System.out.println("매개변수 a : "+a);
    for (int number : numbers) {
        System.out.println(number);
    }
}
```

오버로딩

- **오버로딩(Overloading)**

  같은 메소드 이름을 사용하지만 매개 변수의 타입과 위치가 다른 방식
  * 다형성의 개념중 하나

오버로딩 예시

- 예시

메소드의 이름은 모두 같지만, 매개변수의 타입과 위치가 다름
이로써 각자 다른 기능을 수행하는 메소드가 된다.

```
public static void main(String[] args) {
    E_Object obj = new E_Object();

    obj.exOverloading(1);
    obj.exOverloading(2, true);
    obj.exOverloading(false, 3);
}

public void exOverloading(int num) {
    System.out.printf("예시 1번이고, 받은 매개변수는 1.%d 입니다. \n", num);
}

public void exOverloading(int num, boolean isTrue) {
    System.out.printf("예시 1번이고, 받은 매개변수는 1.%d , 2.%b 입니다. \n", num, isTrue);
}

public void exOverloading(boolean isTrue, int num) {
    System.out.printf("예시 1번이고, 받은 매개변수는 첫번째 1.%b , 2.%d 입니다. \n", isTrue, num);
}
```

오버로딩 활용하기

- **Q1. 오버로딩을 활용하여 덧셈을 수행하는 계산기 프로그램을 만드시오.**

1. 스캐너로 사용자로부터 숫자를 입력 받으세요.
2. 오버로딩을 활용하여 덧셈을 수행하고 결과를 출력하세요.
3. 클래스명 : QuizCalc
4. 메소드명 : add
5. 정수와 실수도 같이 계산이 가능해야 합니다. (ex. 3+3.5)
6. 오버로딩은 총 5개 이상 활용 하세요.

오버로딩 활용하기

- **Q1 풀이 코드**

```
public class QuizCalc {
public static void main(String[] args) {
QuizCalc calc = new QuizCalc();  // 객체 생성
        int result1 = calc.add(5, 10); // add(int a, int b) 메소드 호출 후 반환값 result1에 저장
        int result2 = calc.add(5, 6, 7); // add(int a, int b, int c) 메소드 호출 후 반환값 result1에 저장
        double result3 = calc.add(1.5, 2); // add(double a, int b) 메소드 호출 후 반환값 result1에 저장
        double result4 = calc.add(2, 2.5); // add(int a, double b) 메소드 호출 후 반환값 result1에 저장
        double result5 = calc.add(2.7, 2.5); // add(int a, double b) 메소드 호출 후 반환값 result1에 저장

        System.out.println("Result 1: " + result1);
        System.out.println("Result 2: " + result2);
        System.out.println("Result 3: " + result3);
        System.out.println("Result 4: " + result4);
        System.out.println("Result 5: " + result5);
    }

// 오버로딩 : 같은 이름의 메소드가 있을 때, 매개변수의 타입 또는 위치가 서로 다른 것
 public int add(int a, int b) {
       return a + b;
    }

    public int add(int a, int b, int c) {
        return a + b + c;
    }

    public double add(double a, int b) {
        return a + b;
    }

    public double add(int a, double b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }

}
```

객체의 생성과 사용 방법 (메소드)

객체의 생성과 사용 방법 - 1

- **객체 생성**

```
클래스명 변수명;              → 변수 선언
변수명 = new 클래스명();      → 객체 생성

클래스명 변수명 = new 클래스명(); → 변수 선언 및 객체 생성
```

객체의 생성과 사용 방법 - 2

```
public class E_Object {
    public static void main(String[] args) {
        Person p = new Person();

        System.out.println(p.walk());
        System.out.println(p.run());
        System.out.println(p.stop());
        System.out.println("=================");

        String walk = p.walk();
        String run = p.run();
        String stop = p.stop();

        System.out.println(walk);
        System.out.println(run);
        System.out.println(stop);
    }

}

class Person {
    public String walk() {
        return "한걸음 걸었어요.";
    };

    public String run() {
        return "뛰고 있어요.";
    };

    public String stop() {
        return "멈췄어요.";
    };
}
```

객체 생성

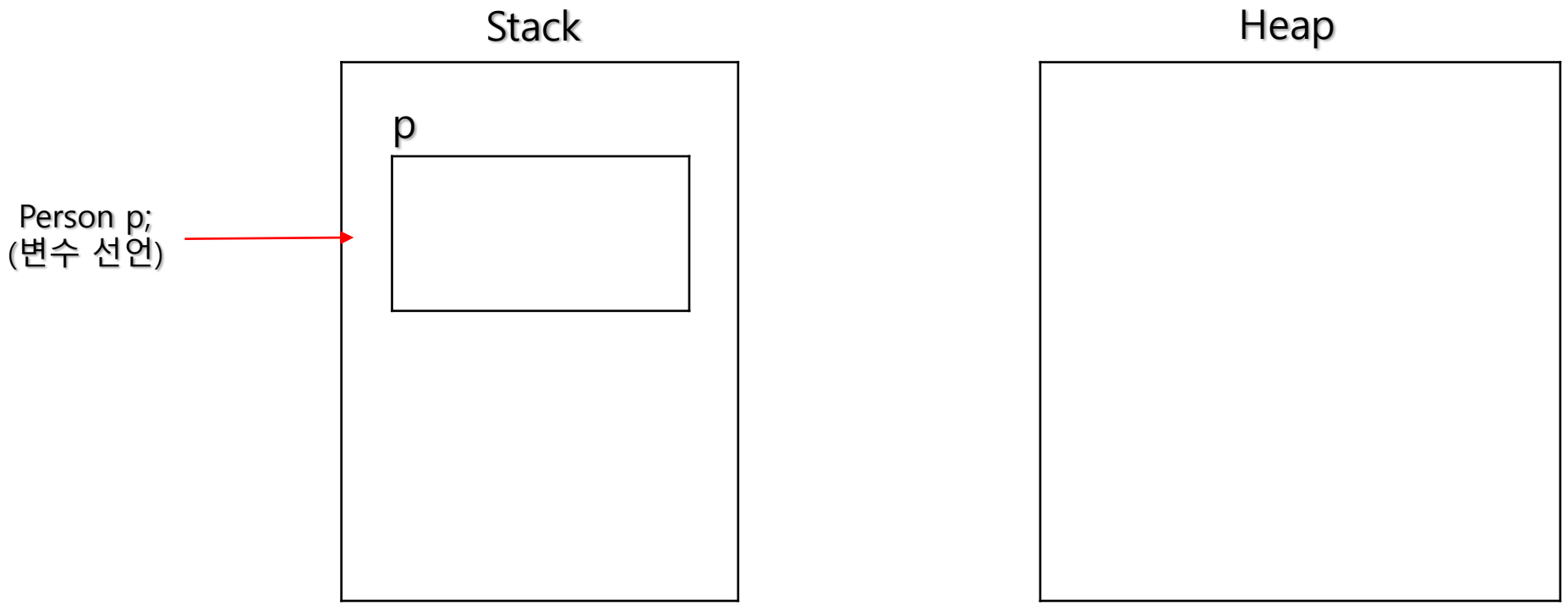
객체 실행(메소드 실행)

실행된 결과를 변수에 저장

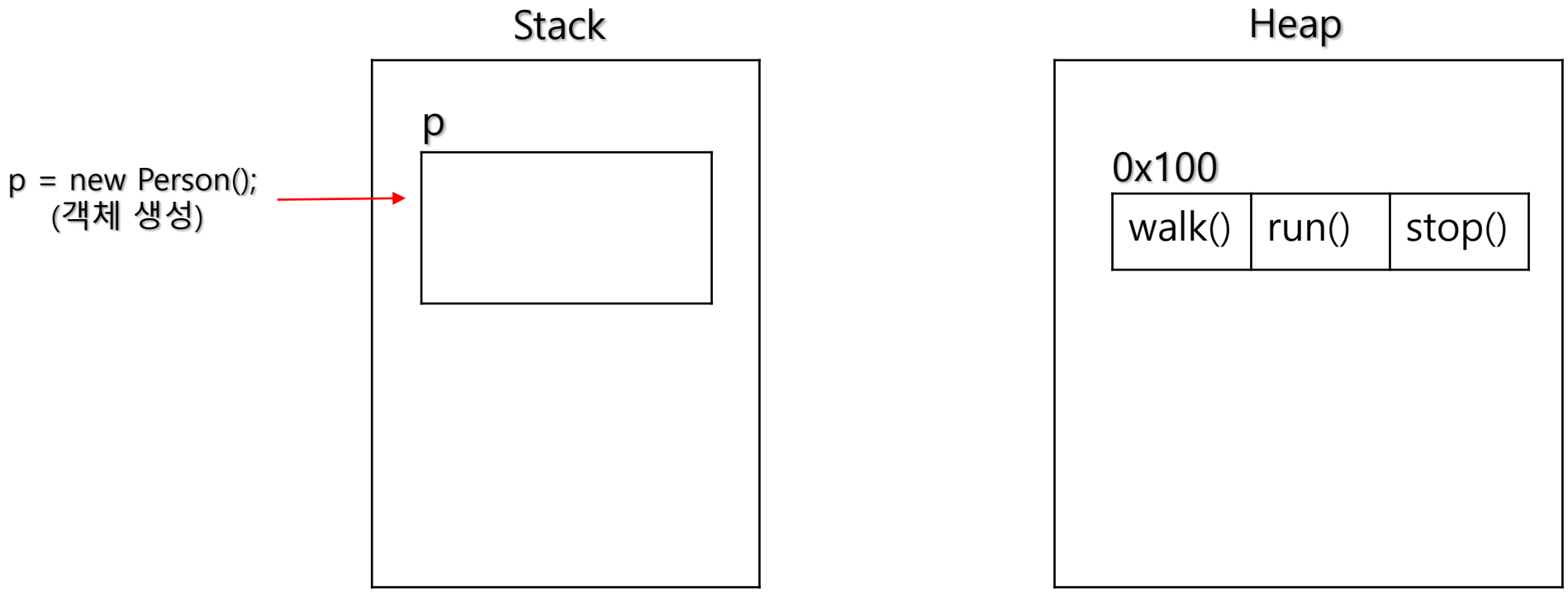
객체의 생성과 사용 방법 - 3

- 참고

참조형의 경우 메모리 주소 저장을 위해 4byte 또는 8byte의 크기를 갖는다.
* 32bit JVM : 4byte / 64bit JBM : 8byte



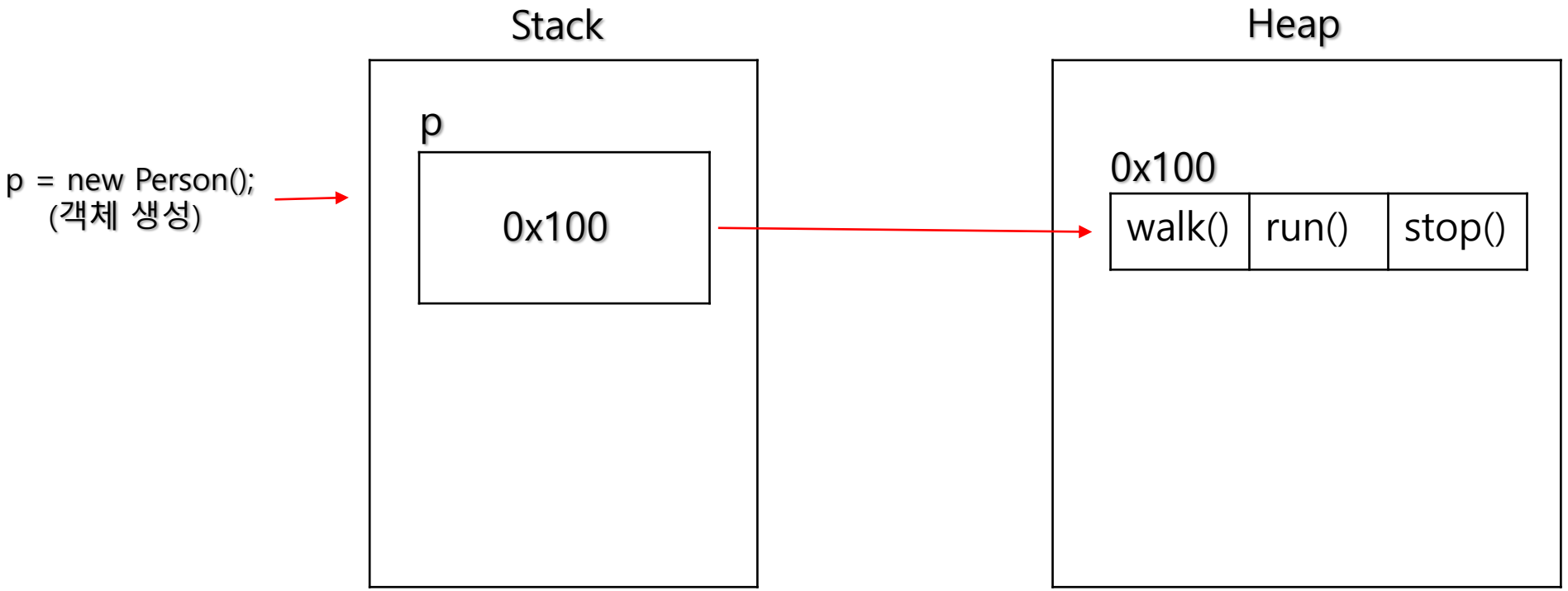
객체의 생성과 사용 방법 - 3



객체의 생성과 사용 방법 - 3

- 참고

new 연산자에 의해 객체가 생성된 후 객체의 메모리 주소를 반환



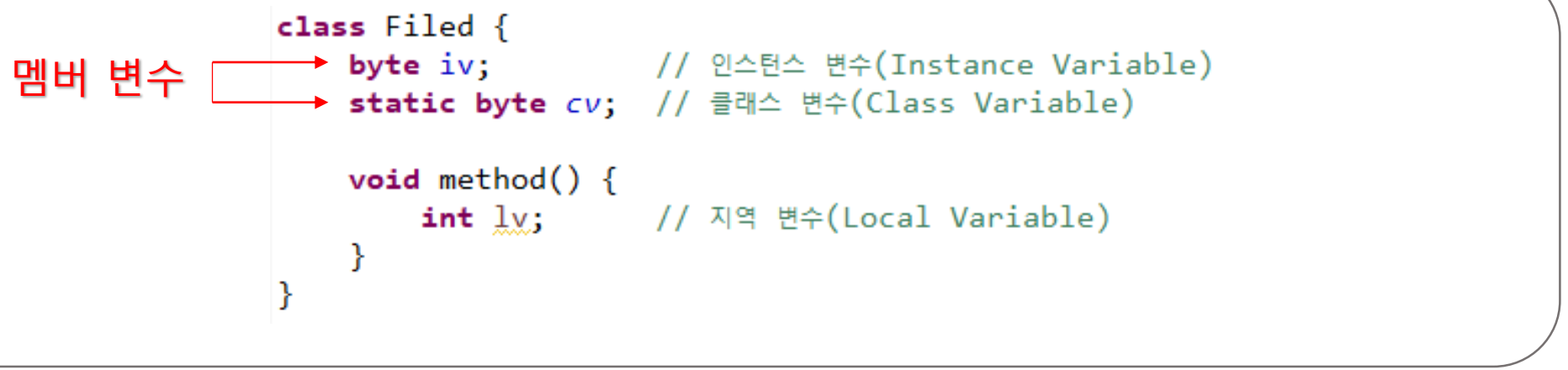
필드

필드

- 필드(Field)

  클래스 또는 메소드의 속성

- 예시



필드

- **멤버 변수(Member Variable)**

  클래스 영역에 있는 변수로써 크게 두가지로 나눠짐
  1. 인스턴스 변수
  2. 클래스 변수

- **지역 변수(Local Variable)**

  메소드 또는 생성자 내부에 위치한 변수

  또는 if, for과 같이 블록 {} 안에 있는 변수

변수의 생명 주기

- **변수 생명 주기(Variable Life Cycle)**

1. 클래스 변수
   - 생성 : 클래스가 메모리에 올라갈 때
   - 해제 : 자바 프로그램이 종료될 때
2. 인스턴스 변수
   - 생성 : 객체가 생성되었을 때
   - 해제 : 객체가 GC에 의해 메모리에서 해제 되었을 때
3. 지역 변수
   - 생성 : 변수 선언문이 수행 되었을 때
   - 해제 : 변수 선언한 블록 {} 이 종료 되었을 때

변수 종류별 메모리 위치

- **변수 종류별 메모리 위치(기본형 기준)**

1. 클래스 변수 : 정적 영역(Static or Method)
2. 인스턴스 변수(Heap)
3. 지역 변수(Stack)

필드 선언 방식

- 필드 선언 방식

```
class Filed {
    [접근제한자] [예약어] 자료형 변수명 [= 값];

    void method() {
        [접근제한자] [예약어] 자료형 변수명 [= 값];
    }
}
```

필드 접근제한자

제어자	같은 클래스	같은 패키지	자식 클래스	전체
public	O	O	O	O
protected	O	O	O	
(default)	O	O		
private	O			

필드 접근제한자 - 2

- 필드 접근제한자 사용 예시1

```
class Filed {
    public int pubInt = 0;
    protected int proInt = 0;
    int defInt = 0;  // default
    private int priInt = 0;
}
```

필드 접근제한자 - 3

- 필드 접근제한자 사용 예시2

Test 클래스

```
private String name = "김재섭"; // private 변수
public int age = 19; // public 변수
```

Person 클래스

```
person.name="홍길동";
person.age=12;

System.out.println(person.age);
```

같은 클래스가 아니므로 접근 불가

필드 활용하기 - 1

```java
class Person {
    public String name;
    public int age;
}
```

```java
Person p1 = new Person();
p1.name = "김재섭";
p1.age = 19;
System.out.printf("p1의 이름은 %s 이고, 나이는 %d 입니다.", p1.name, p1.age);
```



```
p1의 이름은 김재섭 이고, 나이는 19 입니다.
```

필드 활용하기 - 2

- 참고

클래스 변수는 정적 영역(static)에 머물러 있기 때문에, 따로 객체화 하지 않아도 사용이 가능함

```
class Person {
    public static String email;
}
```

```
Person.email = "islandtim@naver.com";
System.out.println(Person.email);
```

객체 생성(인스턴스화) 하지 않은 상태

```
islandtim@naver.com
```

클래스 변수 참고 사항

- 참고

아래의 코드에서 두가지 방법으로 사용이 가능한데 객체변수명.클래스변수명을 사용할 경우 이게 클래스 변수인지, 인스턴스 변수인지 확인 불가

되도록 클래스 변수는 클래스.클래스변수명 을 사용하는것을 권장

```
Person p1 = new Person();
System.out.println(p1.email);
System.out.println(Person.email);
```

멤버 변수 활용하기

- **Q1. 멤버 변수를 활용하여 덧셈을 해보시오.**

1. 인스턴스 변수 instanceNumFirst과 instanceNumSecond를 만드세요.
2. 클래스 변수 classNumThird를 만드세요.
3. 멤버 변수를 활용하여 덧셈을 수행하고 결과를 출력하세요.
   - 인스턴스변수 + 인스턴스 변수
   - 인스턴스변수 + 클래스 변수

멤버 변수 활용하기

- **Q1 풀이 코드**

```
public class QuizMethod {
public static void main(String[] args) {
Calc c = new Calc();  // 객체 생성

c.instanceNumFirst = 10;  // 인스턴스 변수(First)에 10 저장
c.instanceNumSecond = 15; // 인스턴스 변수(Second)에 15 저장
System.out.println(c.instanceNumFirst + c.instanceNumSecond);

c.instanceNumSecond = 25; // 인스턴스 변수(Second)에 25 저장
Calc.instanceNumThird = 37; // 클래스 변수(Third)에 37 저장
System.out.println(c.instanceNumSecond + Calc.instanceNumThird);

// 참고: 클래스 변수는 클래스명.변수명 으로 사용하는것을 권장
    }
}

class Calc {
public int instanceNumFirst;
public int instanceNumSecond;
public static int instanceNumThird;

public int add(int a, int b) {
return a + b;
}
}
```

정리

객체

- **객체 (필드+메소드)**

  클래스 : 설계도 (틀)
  필드 : 클래스 또는 메소드의 속성 (변수, 속성)
  메소드 : 정해진 동작이나 기능을 수행 (행위)

객체

구분	요약	예시
클래스(Class)	설계도, 틀	사람
필드(Field)	속성, 변수	이름, 나이, 핸드폰번호
메소드(Method)	행위	걷다, 뛰다, 웃다

객체

- **추가 설명**

만약, 게임 케릭터를 예시로 들면 아래와 같이도 생각해볼 수 있음

1. 클래스 : 직업 (게임 케릭터의 틀)
2. 필드 : 힘, 민첩, 지력 등등 (속성, 변수)
3. 행위 : 공격하다, 스킬1을 사용하다, 막다 등등 (행위)

접근제한자

- **접근제한자**

이름 그대로 접근을 제한하며, 캡슐화(정보은닉)의 개념 중 하나
아래 표에서 클래스를 제외한 필드와 메소드의 제어자는 같다.

제어자	같은 클래스	같은 패키지	자식 클래스	전체
public	O	O	O	O
protected	O	O	O	
(default)	O	O		
private	O			

오버로딩

- **오버로딩의 조건**

1. 같은 이름의 메소드를 여러 개 가짐
2. 매개변수의 타입 또는 위치가 다름

이는 다형성(여러가지의 형태를 가질 수 있는 개념) 개념중의 하나