



제 5장. 객체와 클래스



학습 목표

- * 객체지향 프로그램과 객체, 클래스의 개념을 학습합니다.
- * 클래스의 구성 요소, 객체를 생성하는 문장의 구조와 객체를 생성할 때의 메모리 구조에 대하여 학습합니다.
- * 멤버 변수와 modifier의 의미에 대하여 학습합니다.
- * 메소드를 정의하는 방법에 대하여 학습합니다.
- * 생성자의 역할과 정의하는 방법에 대하여 학습합니다.
- * this 키워드를 이용하여 생성자를 호출하는 방법과 멤버 변수를 구분하는 표현 방법에 대하여 학습합니다.
- * 패키지의 개념과 패키지 내의 클래스를 사용하는 방법에 대하여 학습합니다.



객체와 클래스

- * 객체지향 프로그래밍, 객체, 클래스의 개념
- * 클래스의 구조
- * 객체 생성
- * 멤버 변수
- * 메소드
- * 생성자
- * this 키워드
- * 패키지



객체지향 프로그래밍, 객체, 클래스의 개념

- * 객체지향 프로그래밍
- * 객체의 개념
- * 클래스의 개념



객체지향 프로그래밍

- * 자바, C++, C# 이 객체지향 언어
- * 하나의 프로그램을 독립적 단위별로 나누어 설계, 필요시 통합
- * 필요 부분만 수정하므로 개발 편리
- * 절차지향 프로그래밍에 비해 부분으로 분해하여 프로그래밍



객체지향 프로그래밍

* 절차지향

```
학사 관리 {  
    이름  
    학번  
    점수  
    .....  
    수강하다  
    성적 평가하다  
    시험보다  
    .....  
}
```

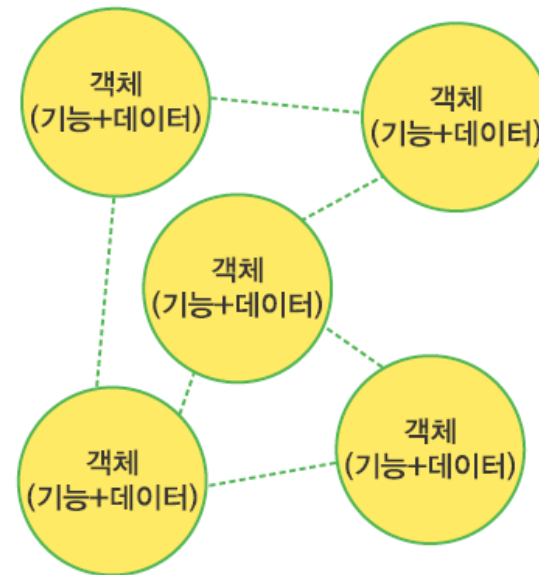
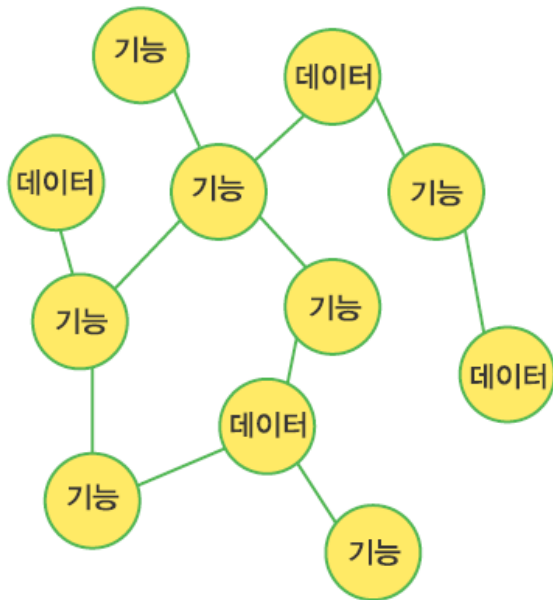
객체지향

```
학생 클래스 {  
    이름  
    학번  
    수강하다  
    시험보다  
}  
  
성적 처리 클래스 {  
    점수  
    성적 평가하다  
}
```

```
학사 관리 클래스{  
    학생의 학사 관리하다{  
        학생이 수강하다  
        학생이 시험보다  
        학생의 성적을 처리하다  
    }  
}
```

객체지향 프로그래밍

- * 객체지향에서의 클래스는 함수와 변수의 묶음으로 클래스 단위로 설계
- * 절차지향 객체지향





객체지향 프로그래밍

- * 캡슐화(encapsulation)는 코드를 보호하는 역할을 합니다.
- * 클래스(class)는 동일한 객체들의 집합체이며 객체의 원형(prototype)입니다.
- * 추상화(abstraction)란 객체가 자신의 정보를 은폐(은닉이라고도 함. information hiding)하고, 외부적으로는 추상적인 내용만 알려주는 것을 말합니다.
- * 상속(inheritance)이란, 기존에 이미 만들어져 있는 객체의 변수와 메소드를 물려받아 다른 새로운 객체를 만드는 것으로 재사용성(reusability)을 얻을 수 있습니다.
- * 다형성(polymorphism)이란 부모 객체로부터 상속받아 전혀 새로운 결과물을 산출합니다.



객체의 개념

- * 현실세계에 존재하는 모든 것
- * 명사로서의 성질을 지닌 것
- * 객체 = 변수 + 메소드
- * 변수 : 객체의 속성 표현
- * 메소드 : 객체의 기능 표현
- * 정의된 객체는 다른 프로그래밍에서
- * 재사용 가능



객체의 개념

* 학생 객체

```
학생 객체 {  
    학번 변수  
    이름 변수  
    수강과목 변수  
    담당교수 변수  
    학년 변수  
    학교명 변수  
    성적 변수  
    수강하다 메소드  
    시험보다 메소드  
    성적받다 메소드  
}
```



클래스의 개념

- * 같은 기능의 객체 여러 개 필요
- * 여러 개의 객체를 표현하기 위해 같은 변수와 메소드도 여러 개 필요
- * 필요한 변수와 메소드를 한번만 정의하고 필요시 복사하여 재사용 필요
- * 여러 개의 동일 구조를 가지는 객체를 생성하기 위해 사용되는 틀

클래스의 개념

```
학생 클래스 {  
    학번, 이름, 수강과목, 담당교수, 학년, 학교명, 성적 변수  
    수강하다 메소드  
    시험보다 메소드  
    성적받다 메소드  
}
```

```
이자바 학생 객체 {  
    학번 = 0700011;  
    이름 = "이자바"  
    수강과목 = "자바프로그래밍";  
    담당교수 = "박이한";  
    학년 = 2;  
    학교명 = "한국대학교"  
    성적 = "A";  
    수강하다 메소드  
    시험보다 메소드  
    성적받다 메소드  
}
```

```
김민국 학생 객체 {  
    학번 = 0800022  
    이름 = "김민국"  
    수강과목 = "C프로그래밍";  
    담당교수 = "박이한";  
    학년 = 1;  
    학교명 = "한국대학교"  
    성적 = "D";  
    수강하다 메소드  
    시험보다 메소드  
    성적받다 메소드  
}
```



클래스의 구조

- * 클래스의 작성
- * 클래스 선언부
- * 클래스 멤버부
- * 자바 modifier



클래스의 작성

- * 모든 객체 표현 이전에 클래스 정의
- * 자바 언어의 프로그램 단위

```
public class 클래스이름 {
```

객체의 정적인 특성을 표현하는 변수들

객체의 행위를 표현하는 메소드들

.....

```
}
```

- * 필요 변수와 메소드 정의하고 클래스 이름과 같은 파일에 저장
- * B 클래스가 A 클래스 객체 생성하여 A 클래스에 정의된 변수와 메소드 사용

클래스 선언부

* 클래스 구조



* 클래스 선언부 형식

```
[modifier] class 클래스이름 [extends 상위클래스이름]
    [implements 인터페이스이름1, 인터페이스이름2, ...] {
    // 클래스 멤버 부분
}
```



클래스 선언부

- * 클래스 선언부 형식
 - * class 키워드와 클래스이름 반드시 기술
 - * modifier는 클래스 활용 방법, 접근 권한 지정
 - * extends 부분은 상속 표현
 - * implements 부분은 인터페이스 구현



클래스 멤버부

- * 클래스의 멤버부는 멤버 변수, 메소드, 생성자로 구성
- * 멤버 변수 : 클래스가 가지는 속성을 정의
- * 생성자(constructor) : 객체의 초기화 담당
- * 메소드 (method) : 클래스가 가지는 데이터를 조작하고 변환
- * 필요시에 멤버 변수와 생성자, 메소드 등에도 modifier를 기술

자바 modifier

* 접근 범위나 활용 방법 지정하는 키워드들

종류	modifier	클래스	멤버 변수	메소드	생성자
접근 권한 modifier	public	O	O	O	O
	protected	X	O	O	O
	private	X	O	O	O
활용 방법 modifier	static	X	O	O	X
	final	O	O	O	X
	abstract	O	X	O	X
	transient	X	O	X	X
	volatile	X	O	X	X
	native	X	X	O	X
	synchronized	X	X	O	X



자바 modifier

* 클래스의 modifier

접근 권한	public	모든 클래스에서 접근이 가능함을 의미합니다.
	final	하위 클래스를 가질 수 없는 클래스를 말합니다.
활용 방법	abstract	추상 클래스를 의미합니다. 추상 클래스는 객체 생성이 불가능한 클래스입니다.



객체 생성

- * 객체 생성 문장의 구성 요소
- * 객체 생성 문장과 자바의 메모리 구조



객체 생성 문장의 구성 요소

- * 클래스 정의 후에는 객체 생성
- * 학생 클래스 정의

```
class Student {  
    String name;        /* 이름 */  
    int number;         /* 학번 */  
    int age;            /* 나이 */  
    int grade;          /* 학년 */  
    String schoolName;  /* 학교명 */  
    String records;     /* 성적 */  
}
```

- * 객체 참조 변수 선언

```
Student stu1 ;
```



객체 생성 문장의 구성 요소

* 객체 생성

```
stu1 = new Student ();
```

* 객체 생성 과정

1. 객체 선언

클래스이름 객체참조변수이름 ;

2. 객체 생성

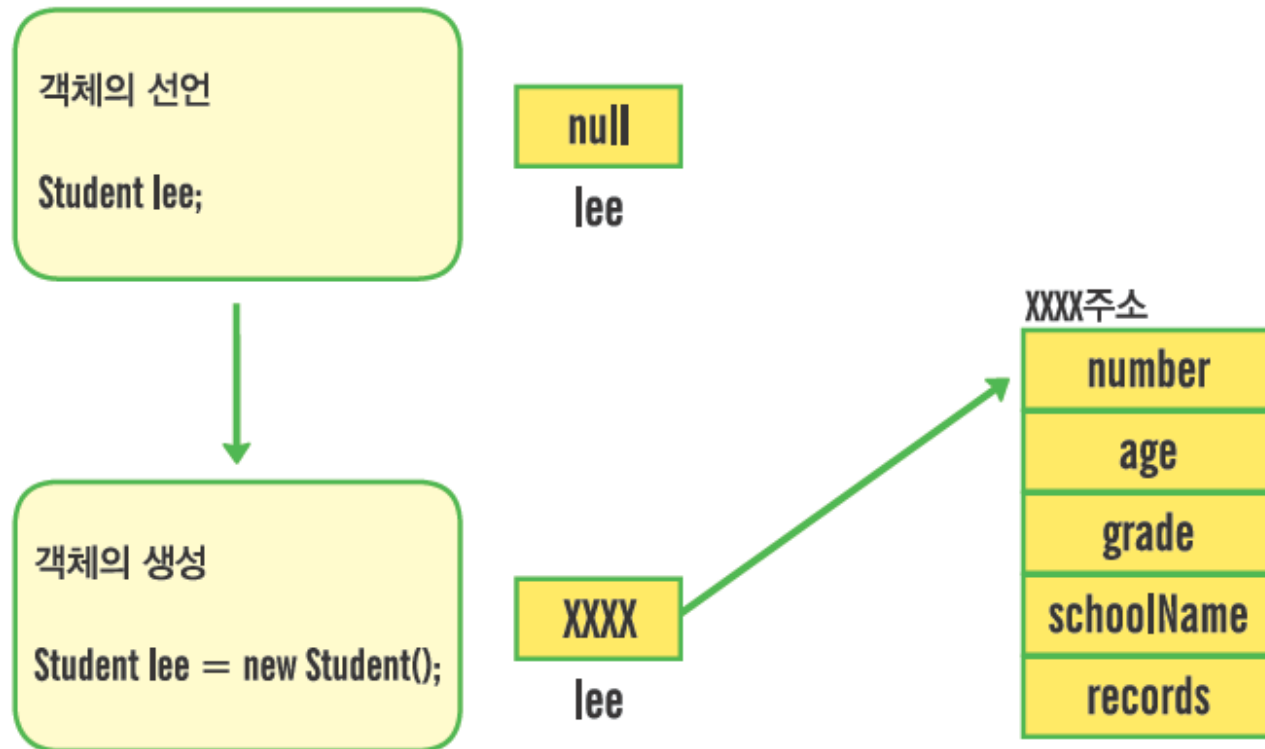
객체참조변수이름 = new 클래스이름 ();

3. 객체 선언과 생성을 한 문장으로 표현

클래스이름 객체참조변수이름 = new 클래스이름 ();

객체 생성 문장과 자바의 메모리 구조

* Student 객체 생성시 메모리 구조





객체 생성 문장과 자바의 메모리 구조

- * 프로그램 5-1 실습
 - * StudentTest.java 작성
 - * Student 객체 생성하여 변수값 초기화

```
C:\WJAVA>java StudentTest
```

```
한 학생의 이름은 이자바이고, 나이는 20입니다.
```




멤버 변수

- * 멤버 변수의 개념
- * 멤버 변수의 구성 요소
- * public, protected, private modifier
- * static modifier
- * final modifier



멤버 변수의 개념

- * 메서드 외부에 선언하는 변수
- * 객체의 속성 표현
- * 지역변수는 특정 블록({}) 내부에서만 사용 가능

```
class A{  
    int i; //멤버 변수. 클래스 A 전체에서 사용 가능.  
    void m1 (){  
        int j = 20; //지역 변수. m1 메소드 내부에서만 사용 가능  
        System.out.println( i + j ); // i, j 변수 모두 사용 가능  
    }  
    void m2 (){  
        System.out.println( i + j ); //오류발생. i 변수 사용 가능하나 j 변수 사용 불가능  
    }  
}
```

멤버 변수의 구성 요소

* 멤버변수 구성 요소

[modifier] 데이터타입 변수이름;

* 멤버변수의 modifier

접근 권한	public	모든 클래스에서 접근이 가능함을 의미합니다.
	protected	동일 패키지에 속하는 클래스와 하위 클래스 관계의 클래스에 의해 접근이 가능하다는 의미입니다.
	private	클래스 내에서만 접근이 가능하다는 의미입니다.
활용 방법	final	변수를 상수로 이용하는 경우 사용합니다.
	static	클래스에 소속된 클래스 변수를 의미합니다. 클래스 정의 시에 만들어집니다.



public, protected, private modifier

- * 캡슐화와 정보 숨김 구현
- * 멤버 변수의 접근 modifier

종류	클래스	하위 클래스	동일 패키지	모든 클래스
private	O	X	X	X
(default)	O	X	O	X
protected	O	O	O	X
public	O	O	O	O



public, protected, private modifier

```
class Student {  
    public String name;          /* 이름 */  
    private int number;         /* 학번 */  
    int age;                    /* 나이 */  
    int grade;                  /* 학년 */  
    protected String records;  /* 성적 */  
    String schoolName;         /* 학교명 */  
}
```

- * number : 외부에서 참조 불가능
- * records : 하위 클래스 참조 가능
- * name : 모든 클래스 참조 가능



public, protected, private modifier

* 프로그램 5-2 실습

- * StudentTest2.java 작성
- * 코멘트 유지하고 실행한 결과

```
C:\WJAVA>javac StudentTest2.java
C:\WJAVA>java StudentTest2
한 학생의 이름은 이자바이고, 나이는 20입니다
```

* 코멘트 해재하고 실행한 결과

```
C:\WJAVA>javac StudentTest2.java
StudentTest2.java:6: number has private access in Student
lee.number = 0500011; // 오류 발생
    ^
1 error
```



static modifier

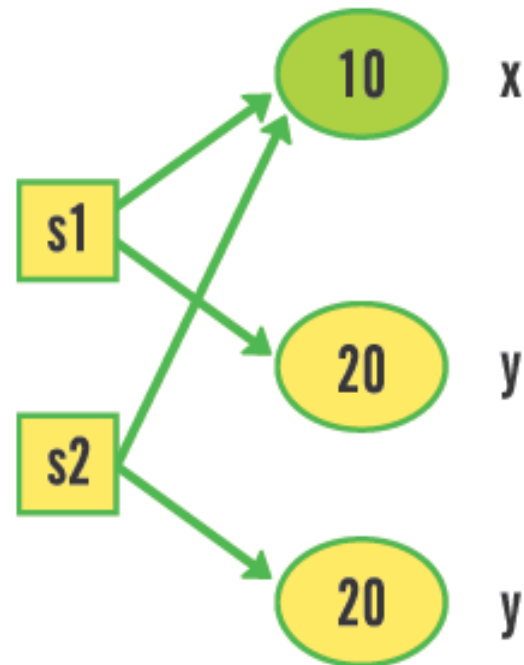
- * static 변수 또는 클래스 변수
- * 해당 클래스의 모든 객체들이 하나의 클래스 변수 공유
- * 클래스 로딩 시기에 메모리에 한번만 할당
- * 객체들 사이의 공통 속성 표현
- * 클래스 이름을 통하여 접근 가능

static modifier

* 클래스 변수의 활용

```
class Sample{  
    static int x = 10;  
    int y = 20;  
    .....  
}
```

```
class Test{  
    .....  
    Sample s1 = new Sample();  
    Sample s2 = new Sample();  
}
```





static modifier

* static 변수와 non-static 변수 비교

	static 변수	non-static 변수
사용 목적	객체 간의 공통된 값 공유	객체의 고유한 값 표현
명칭	클래스 변수	객체 변수
메모리 할당 개수	1개(객체 개수와 무관)	객체 생성 개수만큼 존재
메모리 할당 시기	객체 생성 이전(클래스 로드 타임)	객체 생성 시
메모리 할당 영역	클래스 영역	객체 영역



static modifier

- * 프로그램 5-3 실습
 - * PersonTest.java 작성
 - * static 변수로 각 사람의 국적 표현

```
C:\WJAVA>java PersonTest
```

P1 이 참조하는 사람 객체의 이름은 이자바 이고 나이는 23 이며 국적은 대한민국 입니다.

P2 가 참조하는 사람 객체의 이름은 박이한 이고 나이는 21 이며 국적은 대한민국 입니다.

생성된 사람 객체의 수는 2 명입니다.



final modifier

- * 프로그램 내에서 한번 초기화된 후 값의 수정 불가능한 상수
- * final 변수의 경우
 - * 객체 공유 값 변경 불가능한 static 변수
 - * 절대 불변의 진리 값 표현 변수
 - * 자바 언어의 정해진 규칙에 의해 얻어진 값을 표현하는 변수
 - * final 변수의 예

```
public static final double E;           //자연로그 값 표현 변수  
public static final double PI;         //원주율 값 표현 변수
```



final modifier

- * 프로그램 5-4 실습
 - * PersonTest2.java 작성
 - * final static 변수로 각 사람의 국적 표현

```
C:\WJAVA>javac PersonTest2.java
PersonTest2.java:7: cannot assign a value to final variable nation
p1.nation = "미국"; // 오류 발생
^
1 error
```



메소드

- * 메소드 선언부
- * return 문과 리턴 타입
- * 메소드 구현부
- * public, protected, private modifier
- * static modifier
- * 매개변수의 개념과 매개변수 전달 원리
- * 메소드 overloading

메소드 선언부

* 메소드 선언부의 구성 요소

```
[modifier] 리턴타입 메소드이름 ( [ 매개변수타입 매개변수이름, ... ] )  
[throws 예외클래스, ...] {  
    // 객체의 행위나 기능을 구체적으로 기술하는 문장들;  
}
```

* 메서드의 modifier

접근 권한	public	모든 클래스에서 접근이 가능함을 의미합니다.
	protected	동일 패키지에 속하는 클래스와 하위 클래스 관계의 클래스에 의해 접근이 가능하다는 의미입니다.
	private	클래스 내에서만 접근이 가능하다는 의미입니다.
활용 방법	final	오버라이딩이 불가능한 메소드를 정의할 때 이용합니다.
	static	클래스에 소속된 클래스 메소드를 의미합니다. 클래스 생성 시 만들어집니다.
	abstract	추상 메소드를 의미하며, 하위 클래스에 의해 구현됩니다.
	synchronized	스레드의 동기화를 위한 메소드입니다.



return 문과 리턴 타입

- * 메소드 이름 앞에 오는 필수 요소
- * 메소드 수행 후 복귀하면서 전달하는 값의 데이터 타입
- * int 데이터 리턴하는 메서드 호출 예

```
void m1() {  
    int result1 = m2();    //라인 1  
}  
  
int m2(){ //라인 2  
    int j = 100;  
    return j*j;           //라인 3  
}
```



return 문과 리턴 타입

* String 데이터 리턴하는 메소드 호출 예

```
void m1() {  
    String result1 = m2();           //라인 1  
}  
  
String m2(){ //라인 2  
    return "m2 메소드 리턴 결과";    //라인 3  
}
```

* 리턴값이 없는 메소드 호출 예

```
void m1() {  
    m2();                             //라인 1  
}  
  
void m2(){                             //라인 2  
    int j = 100;  
    System.out.println(j*j);          //라인 3  
}
```


return 문과 리턴 타입

* return의 다른 예

```
void m1(){  
    m2();  
} //라인 1  
  
void m2(){  
    boolean stop= true;  
    System.out.println("m2 메소드를 시작합니다");  
    if(stop) return;  
    System.out.println("m2 메소드를 종료합니다");  
} //라인 2
```

* return 문의 용도

형태	return;	return 데이터;
의미	메소드 중단하고 제어 복귀	메소드 수행 완료 시 데이터를 전달하면서 제어 복귀
사용 위치	메소드 중간	메소드 마지막



메소드 구현부

- * private 변수 접근하는 public 메소드 필요 : 캡슐화 표현

```
class Student {  
    String name;  
  
    public void setName(String n){  
        name = n;  
    }  
    public String getName(){  
        return name;  
    }  
}
```



메소드 구현부

- * 사원 이름 출력 등 기능 반복적 필요하면 하나의 메소드로 정의하고 호출

```
class Student {  
    String name;  
  
    .....  
  
    public void print(){  
        System.out.println("학생의 이름은 "+ name +" 입니다.");  
    }  
}
```



메소드 구현부

- * 프로그램 5-5 실습
 - * StudentTest3.java 작성
 - * Student 객체 생성하여 메소드 호출

```
C:\WJAVA>java StudentTest3  
학생의 이름은 이자바 입니다.  
학생의 이름은 김대한 입니다.  
학생의 이름은 박민국 입니다.
```



public, protected, private modifier

- * 캡슐화와 정보 숨김 구현
- * 메소드의 접근 modifier

종류	클래스	하위 클래스	동일 패키지	모든 클래스
private	O	X	X	X
(default)	O	X	O	X
protected	O	O	O	X
public	O	O	O	O



static modifier

- * static 메소드는 클래스 메소드
- * 클래스 이름으로 접근
- * 객체 생성 이전 단계인 클래스 로드시에 메소드 사용 가능
- * 객체 변수 사용 불가능
- * 클래스 변수만 사용하는 경우 필요



static modifier

- * 프로그램 5-6 실습
 - * StaticTest.java 작성
 - * 클래스 메소드 호출
 - * 코멘트 해제 실행

```
C:\WJAVA>javac StaticTest.java
StaticTest.java:7: non-static variable b cannot be referenced from a static con-
text
    System.out.println(b++);
                       ^
StaticTest.java:19: non-static method m2() cannot be referenced from a static
context
    StaticClass.m2();
                ^
2 error
```

코멘트 유지하고 실행

```
C:\WJAVA>java StaticTest
10
11
22
20
13
14
20
```



매개변수의 개념과 매개변수 전달 원리

- * 메소드나 생성자 정의시 부가 정보는 매개변수로 정의하여 전달
- * 메소드 정의시 필요 정보의 타입과 개수, 순서 정의는 형식매개변수임

[modifier] 리턴타입 메소드이름([매개변수타입 매개변수이름,])

- * 메소드 호출시에 형식매개변수로 전달하는 실제 매개변수는 실매개변수임

메소드이름(매개변수값,...)



매개변수의 개념과 매개변수 전달 원리

- * 자바 언어에서 메소드 호출시에 매개변수 전달 방법은 call by value임
- * call by value 방식은 실매개변수의 값만 복사하여 형식매개변수로 전달
- * 기본형 변수는 실제값 전달
- * 참조형 변수는 객체의 주소값 전달

매개변수의 개념과 매개변수 전달 원리

* 기본형 변수의 매개변수 전달

값의 전달 (기본형)

```
class Expr{  
    void add(int i){  
        i++;  
    }  
    void call(){  
        int j = 10;  
        add(j);  
    }  
}
```

call () 메소드 실행 시의 메모리



매개변수의 개념과 매개변수 전달 원리

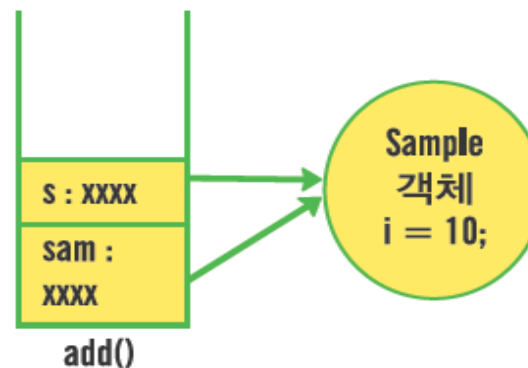
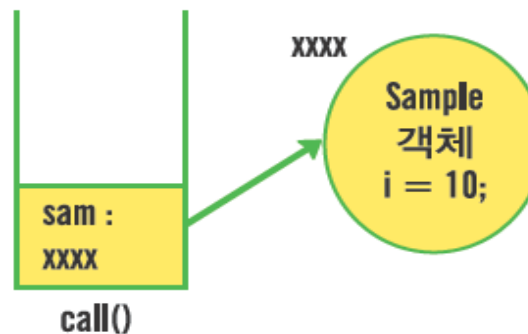
* 참조형 변수의 매개변수 전달

주소 전달 (참조형)

```
class Expr{  
    void add(Sample s){  
        s.i++;  
    }  
    void call(){  
        Sample sam = new Sample();  
        add(sam);  
    }  
}
```

```
class Sample{  
    int i = 10;  
}
```

call () 메소드 실행 시의 메모리





매개변수의 개념과 매개변수 전달 원리

- * 프로그램 5-7 실습
 - * CallByValueTest.java 작성
 - * 기본형 변수와 참조형 변수 값 전달

```
C:\W\JAVA>javac CallByValueTest
10
main에서 add(j) 호출 후 : 10
10
main에서 add(t) 호출 후 : 11
```



메소드 overloading

- * 하나의 클래스 내부에 같은 이름의 메서드 정의 가능
- * 매개변수 리스트 다르게 정의
 - * 매개변수 타입이나 개수, 또는 순서 중 하나가 다르게 정의
- * 객체지향언어의 다형성 지원



메소드 overloading

* java.lang.Integer.parseInt() 메서드 overloading

```
public class Integer {  
    public static int parseInt(String s).....{  
        s 변수에 포함된 값을 10진수 정수로 변경하는 기능 정의  
    }  
  
    public static int parseInt(String s, int radix).....{  
        s 변수에 포함된 값을 두 번째 매개변수로 지정된 radix 값을  
        베이스로 하는 정수로 변경하는 기능 정의. 즉 radix 가 2라면 2진수 정수,  
        16이라면 16진수의 정수로 변경.  
    }  
    .....  
}
```

* parseInt() 메소드 호출

```
int result1 = Integer.parseInt("100");  
int result2 = Integer.parseInt("100", 2);
```



메소드 overloading

- * 프로그램 5-8 실습
 - * OverloadTest.java 작성
 - * + 연산자를 이용한 다양한 연산 수행 메서드 overloading

```
C:\WJAVA>java OverloadTest  
문자열 결합의 결과 : javaprogram  
실수 덧셈의 결과 : 3.66  
정수 덧셈의 결과 : 30
```



생성자

- * 생성자의 개념과 특징
- * 생성자의 구성 요소
- * public, protected, private modifier
- * 생성자와 객체생성
- * 생성자 overloading



생성자의 개념과 특징

- * 객체 생성시 객체 초기화 과정 기술하는 특수한 메소드
- * 객체 생성 문장의 구조

```
Student stu = new Student();
```

```
클래스이름 객체참조변수이름 = 객체생성연산자 생성자명();
```

- * new 연산자 뒤에서 호출



생성자의 개념과 특징

* 생성자의 특징

- * 메소드처럼 리턴 타입을 설정하지 않습니다.
- * 클래스에 대한 객체를 생성할 때 초기화를 담당합니다.
- * 자바 클래스는 자동으로 기본 생성자(default constructor)가 존재하므로 생성자가 필요 없으면 반드시 프로그래머가 정의할 필요는 없습니다.
- * 생성자 overloading을 지원합니다.
- * `super()`, `this()`와 같은 특수한 형태의 생성자가 존재합니다.



생성자의 구성 요소

* 생성자의 구성 요소

```
[modifier] 생성자이름([매개변수타입 매개변수이름, ..]){  
    객체 생성 시에 수행할 문장 정의.  
}
```


* 리턴 타입이 없고 클래스와 이름이 같은 점이 형태상 메소드와 구별

public, protected, private modifier

* 생성자의 modifier

접근 권한	public	모든 클래스에서 접근이 가능하다는 의미입니다.
	protected	동일 패키지에 속하는 클래스와 하위 클래스 관계의 클래스에 의해 접근이 가능하다는 의미입니다.
	private	클래스 내에서만 접근이 가능하다는 의미입니다.

* default : 생성자 생략한 상태로 같은 패키지 내부에서 접근



생성자와 객체 생성

- * 기본 생성자가 모든 클래스에 자동 포함
- * 기본 생성자 형태

```
클래스이름 ( ) { super(); }
```

- * 기본 생성자 이용 예

```
class Student {  
    String name;          /* 이름 */  
    int number;           /* 학번 */  
    int age;              /* 나이 */  
    int grade;            /* 학년 */  
    String schoolName;    /* 학교명 */  
    String records;       /* 성적 */  
    /* Student( ) { super(); } 자동 정의된 기본 생성자 */  
}
```

```
public class StudentTest{  
    public static void main(String args[]){  
        Student lee = new Student();  
        .....  
    }  
}
```



생성자와 객체 생성

- * 명시적 생성자 정의 후에 기본 생성자 자동 상실
- * 명시적 생성자 이용 예

```
class Student {  
    String name;        /* 이름 */  
    int number;         /* 학번 */  
    int age;            /* 나이 */  
    int grade;          /* 학년 */  
    String schoolName;  /* 학교명 */  
    String records;     /* 성적 */  
  
    //이름과 학번을 초기화하는 생성자 정의  
    Student(String n, int num){  
        name = n;  
        number = num;  
    }  
}
```

```
public class StudentTest{  
    public static void main(String args[]){  
        Student lee = new Student();  
        Student kim = new Student("김철수", 9957634);  
        .....  
    }  
}
```



생성자와 객체 생성

- * 프로그램 5-9 실습
 - * StudentTest4.java 작성
 - * 명시적 생성자 호출
 - * 출력 결과

```
C:\WJAVA>java StudentTest4  
한 학생의 이름은 박대한이고, 나이는 22 입니다
```

- * 주석 해제 후 결과

```
C:\WJAVA>javac StudentTest4.java  
StudentTest4.java:4: cannot find symbol  
symbol   : constructor Student()  
location : class Student  
        Student lee = new Student();  
                        ^
```

```
1 error
```



생성자 overloading

- * 하나의 클래스에서 같은 이름의 생성자 여러 개 정의 가능
- * 생성자의 매개변수 타입, 개수, 순서 다르게 정의

```
//이름과 나이를 초기화하는 생성자 정의
Student(String n, int a){
    name = n;
    age = a;
}

//이름만 초기화하는 생성자 정의
Student(String n){
    name = n;
    age = -1;
}
```

```
//기본 생성자 정의
Student(){
    name = "none";
    age = -1;
}
```




생성자 overloading

- * 3가지 형태의 객체 생성 가능

```
Student lee = new Student();  
Student park = new Student("박이한");  
Student kim = new Student("김대한", 22);
```



생성자 overloading

- * 프로그램 5-10 실습
 - * StudentTest5.java 작성
 - * 여러가지 생성자 호출
 - * 출력 결과

```
C:\WJAVA>java StudentTest5
```

```
한 학생의 이름은 none이고, 나이는 -1 입니다
```

```
한 학생의 이름은 박이한이고, 나이는 -1 입니다
```

```
한 학생의 이름은 김대한이고, 나이는 22 입니다
```



this 키워드

- * this 키워드의 개념
- * this 키워드와 멤버 변수
- * this 키워드와 생성자 호출



this 키워드의 개념

- * this 키워드는 메시지 전달받은 객체
- * this 키워드는 형태 실행 중인 클래스 타입의 객체
- * 매개 변수나 지역 변수 이름이 객체 변수 이름과 같을 때 구분

```
this.객체변수이름
```

- * 같은 클래스 내의 다른 생성자 호출

```
this ( [매개변수리스트] )
```



this 키워드와 멤버 변수

* this 키워드를 이용하여 객체 변수 구별

```
class Student {  
    String name;    /* 이름 */  
    int age;        /* 나이 */  
  
    //이름과 나이를 초기화하는 생성자 정의  
    Student(String name, int age){  
        name = name;    // 라인 1  
        age = age;      // 라인 2  
    }  
    .....  
}
```



this 키워드와 생성자 호출

* 생성자 호출

```
class Student {  
    String name; /* 이름 */  
    int age;     /* 나이 */  
  
    //이름과 나이를 초기화하는 생성자 정의  
    Student(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
}
```



this 키워드와 생성자 호출

- * 프로그램 5-11 실습
 - * StudentTest6.java 작성
 - * this 키워드 이용
 - * 출력 결과

```
C:\WJAVA>java StudentTest6
```

```
한 학생의 이름은 none이고, 나이는 -1 입니다
```

```
한 학생의 이름은 박이한이고, 나이는 -1 입니다
```

```
한 학생의 이름은 김대한이고, 나이는 22 입니다
```



패키지

- * 패키지의 개념
- * import 키워드
- * package 키워드



패키지의 개념

- * 관련 클래스와 인터페이스 저장 폴더
- * 패키지 사용 이유
 - * 같은 이름의 클래스 충돌 회피
 - * 관리 용이
 - * 클래스 접근 권한을 패키지 단위로 제어
- * API 는 패키지화 구조로 제공되며 1.5 버전의 경우 150여개 패키지 제공
- * 도트(.)로 표현

```
java.lang.Integer  
java.io.FileInputStream  
java.awt.Frame
```



import 키워드

- * 패키지화된 클래스 사용
- * 자동 import
 - * 같은 패키지의 클래스 사용
 - * java.lang 패키지의 클래스 사용
- * 사용 형태
 - * 패키지 이름만 사용하여 import
 - * 클래스 이름 포함하여 import

```
import java.lang.Integer ;  
import java.io.FileInputStream ;  
import java.awt.* ;
```

```
class A {      }
```



import 키워드

- * 패키지화된 클래스 사용

- * import 사용하지 않음

```
java.util.Date d = new java.util.Date();
```

- * import 사용함

```
import java.util.Date;  
class...{  
    Date d = new Date();  
    ....  
}
```

```
import java.util.*;  
  
class...{  
    Date d = new Date();  
    ArrayList array = new ArrayList();  
}
```



import 키워드

- * 프로그램 5-12 실습
 - * ImportTest.java 작성
 - * import 사용하여 패키지화된 클래스 사용
 - * 출력 결과

```
C:\WJAVA>java ImportTest  
현재 시간은 2009. 3. 21 입니다.
```



package 키워드

- * 클래스의 패키지화시에 사용
- * 자바 소스 파일의 첫번째 문장으로 사용

```
package 패키지이름;  
  
class 클래스이름 {  
  
    }  
  
public class 클래스이름 {  
  
    }
```



정리

- * 객체란 실세계에서 명사로 표현할 수 있는 사물이며 자바 프로그램에서 하나의 소프트웨어적인 단위입니다.
- * 클래스는 클래스의 선언부와 몸체로 나누어지고 몸체에는 멤버 변수 부분과 객체의 초기화를 담당하는 생성자, 데이터를 조작하고 변환하는 메소드로 구성됩니다.
- * 클래스와 클래스 내부의 구성 요소 선언 시에는 modifier를 이용하여 요소별 접근 권한과 사용 방법을 지정할 수 있습니다.
- * static 변수는 특정 클래스 타입의 모든 객체가 공통적인 값을 공유하여 사용하는 경우 선언합니다. 클래스 이름을 통해 접근 가능합니다.



정리

- * final 변수는 클래스 내부에서 변수의 값을 수정하지 못하도록 설정합니다. 주로 절대 불변의 진리 값이나 공통적으로 사용하는 변수 앞에 붙여 사용합니다.
- * 객체의 동적인 특징과 행동을 정의한 부분인 메소드를 메소드의 접근 권한과 특성에 맞도록 선언할 수 있습니다.
- * 메소드의 리턴 타입은 메소드가 수행 종료된 후 리턴하는 값의 데이터 타입을 지정하는 것으로 메소드 선언부의 메소드 이름 앞에 기술해야 합니다.
- * 매개변수 값의 전달에는 call by value 방식을 사용하며 기본형 변수는 실제 값이 전달되고, 참조형 변수는 주소 값이 복사되어 전달됩니다.



정리

- * 하나의 클래스 내부에 같은 이름의 메소드를 여러 개 정의하되 매개변수 리스트가 다르게 정의하여 메소드를 overloading 할 수 있습니다.
- * 생성자는 클래스로부터 객체를 생성할 때 객체의 초기화 과정을 기술하는 특수한 메소드로 일반 메소드와 비슷하면서도 메소드와는 다르게 클래스와 이름이 동일하고 리턴 타입이 없는 특징을 가지고 있습니다.
- * 생성자는 메소드처럼 매개변수 리스트를 다르게 하여 overloading 할 수 있으며 이는 객체 생성 시에 여러 생성자 중 선택적으로 호출 가능하다는 의미입니다.



정리

- * this 키워드는 현재 객체를 의미하는 자바 키워드로, 멤버 변수와 매개변수, 또는 지역 변수 이름이 동일할 때 구분하거나 this() 문장처럼 생성자를 호출할 때 사용합니다.
- * 패키지란 관련된 클래스들을 모아놓은 구조로, 물리적으로는 폴더 또는 디렉터리와 유사합니다. 패키지는 도트 (.)를 이용하여 표현합니다.
- * import 예약어에 패키지 이름을 명시하면, 컴파일러가 이 클래스들의 패키지 이름을 인식하여 해당 클래스를 찾게 됩니다.