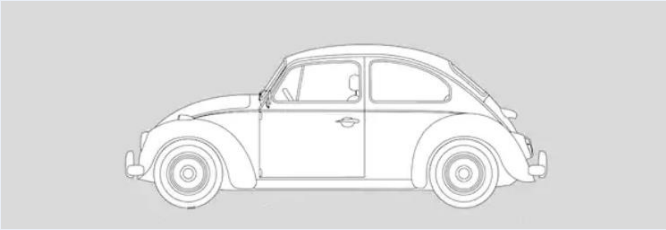





Class

클래스 : 서로 다른 자료형의 변수, 함수들을 묶어서 하나의 객체를 표현하기 위한 자료형

객체 : 클래스를 이용하여 생성한 실제 데이터

클래스	객체
자료형	변수
구조체	구조체 변수
자동차 설계도	자동차
	



Class

클래스의 구성 요소

Member Field (변수) : 생성된 객체가 가지게 될 속성을 저장하는 **변수**

Member Method (함수) : 생성된 객체가 가지게 될 기능을 정의하는 **함수**

Constructor : 객체 생성 시 각 필드의 **초기값**을 지정하거나, **초기작동**을 정의하는 **함수**

Access Modifier : 멤버 필드나 메서드의 **외부 접근 허용 범위**를 지정하는 키워드

Garbage Collector : 참조되지 않는 객체의 **메모리 할당**을 자동으로 **해제**

멤버 필드는 속성을 정의하고, 멤버 메서드는 기능을 정의한다.

예)

TV의 속성 : 전원, 채널, 볼륨

TV의 기능 : 전원 On/Off, 채널 Up, 채널 Down, 볼륨 Up, 볼륨 Down, 음소거 등



Class

예시) 이름과 점수를 저장할 수 있고, 출력 기능이 있는 Student 클래스

```
class Student{                                // 사용자 정의 자료형 Student
    String name;                               // Member Field 선언
    int score;

    void Show() {                             // Member Method 정의
        System.out.println(name + "의 점수는 " + score + "점 입니다");
    }
}

public class Ex_Making_Class {
    public static void main(String[] args) {
        Student st1 = new Student();          // 객체 생성

        st1.name = "이지은";                  // Member Field에 값 대입
        st1.score = 97;

        st1.Show();                           // Member Method 호출
    }
}
```



Class

예시) 이름과 점수를 저장할 수 있고, 출력 기능이 있는 Student 클래스

Stack (main)

Heap

Student st1



Class

예시) 이름과 점수를 저장할 수 있고, 출력 기능이 있는 Student 클래스

Stack (main)

Student st1

Heap

new Student();

```
class Student{  
    String name;  
    int score;  
  
    void Show() { ... }  
}
```



Class

예시) 이름과 점수를 저장할 수 있고, 출력 기능이 있는 Student 클래스

Stack (main)

Student st1

```
st1.name = "이지은";  
st1.score = 97;
```

Heap

new Student();

```
class Student{  
    String name = "이지은";  
    int score = 97;  
  
    void Show() { ... }  
}
```



Class

예시) 이름과 점수를 저장할 수 있고, 출력 기능이 있는 Student 클래스

Stack (main)

Student st1

```
st1.name = "이지은";  
st1.score = 97;
```

```
st1.Show();
```

Heap

```
new Student();
```

```
class Student{  
    String name = "이지은";  
    int score = 97;  
  
    void Show() { ... }  
}
```



Class

예시) 이름과 점수를 저장할 수 있고, 출력 기능이 있는 Student 클래스

Stack (main)

Student st1

```
st1.name = "이지은";  
st1.score = 97;
```

```
st1.Show();
```

Heap

```
new Student();
```

```
class Student{  
    String name = "이지은";  
    int score = 97;  
  
    void Show() { ... }  
}
```

```
System.out.println(name + "의 점수는 " + score + "점 입니다");
```

이지은의 점수는 97점 입니다



Class

연습문제) Quiz01.java

TV클래스를 만들면서, 클래스와 객체의 특징을 익혀보자

```
class TV{
    // TV의 속성값을 저장하는 Member Field 선언 (전원, 채널, 볼륨 등...)

    // TV의 기능을 정의하는 Member Method 정의
    // (전원버튼, 채널/볼륨 up/down 기능, 화면 표시 등...)

}

public class Quiz01_TV {
    public static void main(String[] args) {
        // TV 클래스를 활용하여 tv 객체를 생성

        // tv 객체의 필드에 값을 대입

        // tv 객체의 메서드를 호출하여 기능이 잘 작동하는지 확인
    }
}
```



Class – Constructor

생성자 : 객체를 생성할 때 호출되는 메서드

생성자의 특징

1. 클래스의 이름과 동일한 이름을 가지는 메서드이다.
2. 반환형이 존재하지 않는다. (void도 설정하지 않음)
3. 생성자는 메서드이므로, 중복선언 (Overloading)이 가능하다.
4. 생성자를 따로 작성하지 않으면, JVM이 기본 생성자를 만든다.
5. 어떠한 형식이든 생성자를 작성하면, JVM이 기본 생성자를 만들지 않는다.
6. 생성자 간의 호출이 가능하다. (생성자 코드의 첫번째 줄에서만)

```
String str = new String();
```



Class – Constructor

예시) 2차원의 좌표를 저장하고, 표현할 수 있는 클래스

```
class Pos {  
    // Pos() {  
    //     System.out.println("기본 생성자 호출 !!");  
    // }
```

생성자가 없으면
JVM이 자동으로 작성한다
(기본 생성자)

```
    int x, y;        // JVM에서 정수 필드의 초기값을 0으로 설정한다
```

```
    void ShowPos() {  
        System.out.println("x : " + x);  
        System.out.println("y : " + y);  
        System.out.println();  
    }  
}
```

```
public class Ex01_Constructor {  
    public static void main(String[] args) {  
  
        Pos ob1 = new Pos();  
  
        ob1.ShowPos();  
    }  
}
```

x : 0
y : 0



Class – Constructor

예시) 2차원의 좌표를 저장하고, 표현할 수 있는 클래스

```
class Pos {  
    Pos() {  
        System.out.println("기본 생성자 호출 !!");  
    }  
  
    int x, y;        // JVM에서 정수 필드의 초기값을 0으로 설정한다  
  
    void ShowPos() {  
        System.out.println("x : " + x);  
        System.out.println("y : " + y);  
        System.out.println();  
    }  
}  
  
public class Ex01_Constructor {  
    public static void main(String[] args) {  
  
        Pos ob1 = new Pos();  
  
        ob1.ShowPos();  
    }  
}
```

기본 생성자 호출 !!
x : 0
y : 0



Class – Constructor

예시) 객체를 생성하면서, **멤버 필드의 초기값**을 직접 지정하고 싶다면

```
class Pos {
    Pos() {
        System.out.println("기본 생성자 호출 !!");
    }
    Pos(int a, int b) {           // 기본 생성자와 매개변수 개수가 다르다 (Method Overloading)
        x = a;                   // 전달받은 매개변수를 멤버 필드에 대입한다
        y = b;
    }

    int x, y;                    // JVM에서 정수 필드의 초기값을 0으로 설정한다
    void ShowPos() { ... }
}

public class Ex01_Constructor {
    public static void main(String[] args) {

        Pos ob1 = new Pos();
        Pos ob2 = new Pos(2, 3);

        ob1.ShowPos();
        ob2.ShowPos();
    }
}
```

기본 생성자 호출 !!

x : 0

y : 0

x : 2

y : 3



Class – Constructor

예시) 객체를 생성하면서, **멤버 필드의 초기값**을 직접 지정하고 싶다면

```
class Pos {  
    Pos() {  
        System.out.println("기본 생성자 호출 !!");  
    }  
    Pos(int x, int y) {           // 기본 생성자와 매개변수 개수가 다르다 (Method Overloading)  
        this.x = x;              // 전달받은 매개변수를 멤버 필드에 대입한다  
        this.y = y;              // 매개변수와 멤버필드의 이름이 같으면 this. 을 붙여서  
                                  // 멤버필드임을 명시한다  
    }  
    int x, y;  
    void ShowPos() { ... }  
}  
  
public class Ex01_Constructor {  
    public static void main(String[] args) {  
  
        Pos ob1 = new Pos();  
        Pos ob2 = new Pos(2, 3);  
  
        ob1.ShowPos();  
        ob2.ShowPos();  
    }  
}
```

기본 생성자 호출 !!

x : 0

y : 0

x : 2

y : 3



Class – Constructor

연습문제) Quiz01_Constructor.java

```
class Circle{
// 1. 기본 생성자 : Scanner를 이용해서 사용자에게 반지름을 입력받도록 설정하기
// 2. 정수를 전달받아서, radius 필드에 저장하는 생성자
// 3. 실수를 전달받아서, radius 필드에 저장하는 생성자

    double radius;
    double pi;

    void Show() {
        // 원의 반지름을 출력
        // 원의 넓이 소수점 둘째자리까지 출력하고 한줄 비우기
    }
}

public class Quiz01_Constructor {
    public static void main(String[] args) {
        // Circle 클래스의 객체 3개를 생성하면서, 서로 다른 생성자를 호출하세요
        // 각 객체의 Show() 메서드를 호출하세요
    }
}
```



Class – this

this : 객체 내부에서, 객체 자기 자신을 가리키는 키워드

1. this.[멤버요소]

메서드 내에서 매개변수와 멤버필드의 이름이 중복될 경우 구분하기 위해서 활용한다.

2. this()

클래스의 이름과 같은 메서드가 생성자이므로,
자신을 나타내는 this뒤에 ()를 붙여서 자신의 생성자를 가리킨다.
(생성자내에서 다른 생성자를 호출하려면 첫 줄에서만 가능 !!)

3. this

객체 자신의 참조값을 메서드의 매개변수로 전달하거나
자신의 메서드에서 반환값으로 사용할 수 있다.

```
this.x = x;
```




Class – this

예시 1) 객체 자신의 멤버 요소를 가리키고 싶을 때

예시 2) 생성자 내에서 자신의 또다른 생성자를 호출할 때

```
class Circle{

    Circle(double radius){
        this.radius = radius;
    }
    // 메서드 내에서는 지역변수, 매개변수를 우선적으로 참조하므로,
    // this.을 붙여 멤버 필드를 가리킨다
    // 멤버 필드 = 매개 변수;

    Circle(int radius){
        // this.radius = radius;
        this((double)radius);
    }
    // this()는 자신의 또다른 생성자를 호출한다
    // 매개변수를 int로 전달받아서, 그 값을 double로 캐스팅하면
    // double형 매개변수를 가지는 위쪽의 생성자에게 값을 전달한다
}
```



Class – Reference

예시) 서로 다른 객체 간의 참조를 구현하기

```
class Kid {  
    Kid(String name) {        this.name = name;        }  
  
    String name;  
    Ball b;        // Ball 타입의 객체를 참조할 수 있는 b  
  
    void getBall(Ball b) {  
        this.b = b;  
        System.out.println(name + "가 공을 가지고 있다 !!\n");  
    }  
    void throwBall(Kid other) { // Kid 타입을 참조할 수 있는 매개변수 other  
        System.out.println(name + "가 " + other.name + "에게 공을 던졌다");  
        other.getBall(b);  
        this.b = null;  
    }  
}  
  
class Ball {    String name = "공";    }
```



Class – Reference

예시) 서로 다른 객체 간의 참조를 구현하기

```
public class Ex02_Reference {  
    public static void main(String[] args) {  
  
        Kid minho = new Kid("민호");  
        Kid chanhyuk = new Kid("찬혁이");  
        Ball b = new Ball();  
  
        minho.getBall(b);  
        minho.throwBall(chanhyuk);  
        chanhyuk.throwBall(minho);  
    }  
}
```

민호가 공을 가지고 있다 !!

민호가 찬혁이에게 공을 던졌다
찬혁이가 공을 가지고 있다 !!

찬혁이가 민호에게 공을 던졌다
민호가 공을 가지고 있다 !!



Class – Reference

연습문제) Quiz02_Remote.java

```
class TV {  
    // 자신이 작성했던 TV 클래스 붙여넣기  
}  
  
class RC {  
    // 자신이 작성했던 TV 클래스를 참조할 수 있는 멤버 필드 작성  
  
    // 리모컨의 버튼을 누르면(메서드 호출), TV의 기능을 호출  
    // 리모컨은 상세한 내용을 가지지 않고, 대상 TV의 기능을 호출하기만 함  
    // 숫자를 직접 입력받아서 채널을 변경하는 기능 추가  
}  
  
public class Quiz01_Constructor {  
    public static void main(String[] args) {  
        // TV와 리모컨을 생성하고, 리모컨으로 TV를 조작해보세요  
    }  
}
```



Class – Access Modifier

접근 제한자 : 클래스 외부에서 클래스 내부로의 접근을 제한하는 예약어

1. 객체 내부의 요소들이 모두 개방적인 경우, 의도치 않은 값의 변경을 막기 위함.
2. 클래스, 생성자, 필드, 메서드에 모두 적용된다.
3. 메서드 내의 지역변수는 적용하지 않는다.
4. 클래스나 생성자는 일반적으로, 외부에서의 접근을 막지 않는다.
5. 멤버 필드는 private, 멤버 메서드는 public 으로 설정한다

접근 제한자	접근 허용 범위
private	단일 클래스 내부에서의 접근만 허용
package (default)	단일 클래스, 파일, 폴더에서의 접근 허용
protected	단일 클래스, 파일, 폴더, 상속관계 접근 허용
public	모든 범위에서의 접근을 허용



Class – Access Modifier

예시) 접근 제한자 없는 일반 클래스

```
class Test{  
    // 필드  
    int pri = 1;  
    int pac = 2;    // 접근 제한자를 작성하지 않으면 default (package) 적용  
    int pro = 3;  
    int pub = 4;  
  
    // 메서드  
    void ShowField() {  
        System.out.println("pri : " + pri);  
        System.out.println("pac : " + pac);  
        System.out.println("pro : " + pro);  
        System.out.println("pub : " + pub);  
    }  
}
```



Class – Access Modifier

예시) 접근 제한자 없는 일반 클래스 - 메인함수

```
public class Ex01_Access_Modifier {  
    public static void main(String[] args) {  
        Test ob1 = new Test();  
  
        ob1.pri = 5;  
        ob1.pac = 5;  
        ob1.pro = 5;  
        ob1.pub = 5;  
  
        ob1.ShowField();  
    }  
}
```



Class – Access Modifier

예시) 접근 제한자를 추가한 클래스

```
class Test{  
    // 필드  
    private int pri = 1;  
    int pac = 2;    // 접근 제한자를 작성하지 않으면 default (package) 적용  
    protected int pro = 3;  
    public int pub = 4;  
  
    // 메서드  
    public void ShowField() {  
        System.out.println("pri : " + pri);  
        System.out.println("pac : " + pac);  
        System.out.println("pro : " + pro);  
        System.out.println("pub : " + pub);  
    }  
}
```

작성된 코드를 실행했을 때
문제가 발생하는 곳이 어디인지

어떠한 이유 때문에
문제가 발생하는지
생각해보세요

에러 메시지를 잘 읽어보세요



Class – Encapsulation

캡슐화 (Encapsulation)

1. 필드와 메서드 (속성 정보와 기능)을 하나의 객체로 묶어서 처리할 수 있다.
2. 구현 내용 중의 일부를 객체 내부, 혹은 다른 클래스에 은닉한다.
(외부에서는 내부에서 처리되는 흐름을 알 수 없다)
3. 내부 값을 직접 변경하거나 삭제할 수 없도록 **Field는 private**으로 처리한다.
4. 내부 값을 조정할 수 있는 메서드는 클래스 내부에서 정의한다.
5. 클래스의 **Method**는 외부에서 접근 가능해야하므로, **public**으로 처리한다.
6. 사용자는 공개된 **Method에 의해서만** Field를 참조하거나 변경할 수 있다.
7. private 필드의 값을 설정(set)하거나, 값을 구하는(get) 메서드가 있어야 한다



Class – getter / setter

예시) 접근 제한자를 적용한 Student Class

```
class Student {  
  
    private String name;  
    private int score;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getScore() {  
        return score;  
    }  
    public void setScore(int score) {  
        this.score = score;  
    }  
}
```



Class – getter / setter

예시) 접근 제한자를 적용한 Student Class - 메인 함수

```
public class Ex02_getset {  
    public static void main(String[] args) {  
  
        Student st1 = new Student();  
  
        // st1.name = "이수현";      // The field Student.name is not visible  
        // st1.score = 94;           // 객체는 있으나, 내부의 name이라는  
                                    // 필드에 값을 넣을 수 없다 (접근 제한 private)  
  
        st1.setName("이수현");      // 메서드를 이용하면 가능  
        st1.setScore(94);  
  
        // System.out.printf("%s의 점수 : %d점\n", st1.name, st1.score);  
        // 값을 출력하려면 값을 확인해야 하지만 확인 불가  
  
        System.out.printf("%s의 점수 : %d점\n", st1.getName(), st1.getScore());  
        // 메서드는 클래스에 정의된 멤버이므로,  
        // 멤버 메서드에서는 private이라도 접근이 가능하다  
    }  
}
```



Class – Access Modifier

연습문제) Quiz01_School.java – 새로운 패키지를 생성하여 만들기

```
class Student{
    // 학생의 이름과 3개의 과목(국,영,수), 합계, 평균을 저장하는 필드

    // 이름과 3개의 과목을 매개변수로 전달받는 생성자 작성

    // 각 필드에 대한 getter / setter 작성
}

class Handler{
    // Student를 저장할 수 있는 길이 10 짜리의 배열을 멤버 필드로 선언

    // 학생 데이터를 입력/출력 받을 수 있는 제어 부분 메서드 start() 작성
}

public class Quiz01_School {
    public static void main(String[] args) {
        // Handler 클래스의 인스턴스를 생성하고 start() 메서드 호출
    }
}
```



Class – Access Modifier

연습문제) Quiz02_Vending_machine.java

```
class VM {  
    // 자판기를 구성하는 속성값들은 필드  
    // 자판기의 필드를 조절하는 기능은 메서드  
    // 자판기의 객체 생성에 필요한 값을 전달받거나, 초기화하는 생성자  
    // 각각의 멤버들은 적절한 접근제한자를 가질것  
    // 기능을 세분화하고, 작은 기능들을 제어하는 메서드는 별도로 만들기  
    // 큰 구상부터 하고, 상세 기능은 이후에 구현할 수 있음  
  
    // 예) 금액을 입력받아서 음료의 가격과 넣은 돈과 거스름돈을  
    // 모두 판별하여 처리한다  
    // 예) 음료의 수량을 객체 생성 단계에서 지정하고, 수량이 0이되면 판매할 수  
    // 없다  
}  
public class Quiz02_Vending_machine {  
  
}
```



Class – Static / Final

Static : 멤버 구성 요소를 객체가 아닌 클래스(자료형)에 소속시킨다

1. static 속성을 가지는 멤버 구성요소는 객체를 생성하지 않아도 접근할 수 있다
2. static 속성을 가지는 멤버 구성요소는 일반 멤버 구성요소보다 생성 시점이 빠르다
3. static 속성을 가지는 멤버 구성요소는 모든 객체들이 공유하는 값이 된다

Final : 멤버 구성 요소를 변경 불가능한 형태로 설정한다

1. final 속성을 가지는 멤버 필드는 초기화 이후 값의 변경이 불가능하다
(상수화)
2. final 속성을 가지는 멤버 메서드는 메서드 재정의가 불가능하다
(상속 시 오버라이딩 불가)



Class – Static / Final

예시) static 필드와 static 메서드

```
class Member {  
    String name;  
    int age;  
    static int all;  
  
    Member(String name, int age){  
        this.name = name;  
        this.age = age;  
        System.out.println(name + " 객체 생성 !!");  
        all += 1;  
    }  
    public void ShowInfo() {  
        System.out.println("이름 : " + name);  
        System.out.println("나이 : " + age + "살\n");  
    }  
    public static void PrintAll() {  
        System.out.println("생성된 총 객체 수 : " + all + "\n");  
    }  
}
```



Class – Static / Final

예시) static 필드와 static 메서드

```
public class Ex01_Static {  
    public static void main(String[] args) {  
        Member.PrintAll();  
  
        Member ob1 = new Member("이지은", 27);  
        Member ob2 = new Member("민경훈", 34);  
        System.out.println();  
  
        ob1.ShowInfo();  
        ob2.ShowInfo();  
  
        Member.PrintAll();  
        ob1.PrintAll();  
        ob2.PrintAll();  
    }  
}
```




Class – Static / Final

static 멤버와 non-static 멤버의 생성 시점

	Static	Non-Static
Member Field/Method	클래스 로딩	객체 생성

1. Code를 확인하고 필요한 클래스를 메모리에 로드한다
2. 클래스를 로드하는 시점에서 Static 구성 요소를 생성한다
3. 이후 객체 생성 시점에서 객체의 필드를 생성한다

-> Non-Static 은 Static 요소를 참조할 수 있으나, Static은 Non-Static을 참조하지 못할 수도 있다



Class – Inheritance

상속 : 기존의 클래스의 특징을 물려받아, 개념을 확장하여 사용하는 작업

1. 객체 지향에서 중요한 개념
2. 상속이라고 표현하지만, **확장**의 개념으로 생각하자
3. 코드의 **재사용성**을 높인다
4. 코드의 유지보수를 용이하게 한다
5. 기존 클래스를 부모 클래스, 혹은 **슈퍼 클래스** 라고 한다
6. 새로운 클래스를 자식 클래스, 혹은 **서브 클래스** 라고 한다
7. 자바에서는 특수한 경우(interface)를 제외하고, **단일 상속**만 가능하다



Class – Inheritance

예시) 클래스 상속 예시

```
class Pos2D {    // 2차원의 좌표를 표현할 수 있는 객체의 자료형을 정의
    int x;
    int y;

    void setPos(int x, int y) {
        this.x = x;
        this.y = y;
    }
    void ShowPos() {
        System.out.println("x : " + x + ", y : " + y);
    }
}
```



Class – Inheritance

예시) 클래스 상속 예시

```
class Pos2D {    // 2차원의 좌표를 표현할 수 있는 객체의 자료형을 정의
    int x;
    int y;

    void setPos(int x, int y) { ... }
    void ShowPos() { ... }
}

class Pos3D {    // Pos2D와 유사한 내용의 Pos3D를 새롭게 작성
                // 중복되는 내용이 많고, 비효율적이다
    int x, y, z;
    void setPos(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
    void ShowPos() {
        System.out.println("x : " + x + ", y : " + y + ", z : " + z);
    }
}
```



Class – Inheritance

예시) 클래스 상속 예시

```
class Pos2D {    // 2차원의 좌표를 표현할 수 있는 객체의 자료형을 정의
    int x;
    int y;

    void setPos(int x, int y) { ... }
    void ShowPos() { ... }
}

class Pos3D extends Pos2D { // Pos2D의 특징을 물려받는 클래스 Pos3D (상속)
    int z;                // 상위클래스 내용에서 추가되는 필드만 작성한다
    void setPos3D(int x, int y, int z) { // 필요한 메서드는 새로 작성한다
        this.x = x;
        this.y = y;
        this.z = z;
    }                      // 정의하지 않아도 기존 메서드를 사용할 수 있다
    void ShowPos() {       // 같은 이름이면 기존 메서드를 재정의(Override)한다
        System.out.println("x : " + x + ", y : " + y + ", z : " + z);
    }
}
```



Class – Override

Override : 부모 클래스로부터 상속받은 메서드를 자식 클래스에서 재정의
- 오버라이딩의 조건

1. 메서드의 선언부가 같아야 한다.

반환형, 메서드 이름, 매개변수의 개수 및 자료형과 순서

2. 부모클래스에서 지정한 접근제한자보다 더 작아질 수 없다

public > protected > package(default) > private

오버로딩 : 중복 정의
오버라이딩 : 재 정의



Class – Override

예시) Override 예시

```
class A{
    public void func1() {};
    protected void func2() {};
    void func3() {};
    private void func4() {};
}
class B extends A {
    protected void func1() { System.out.println("pub->pro"); }
    // Cannot reduce the visibility of the inherited method from A

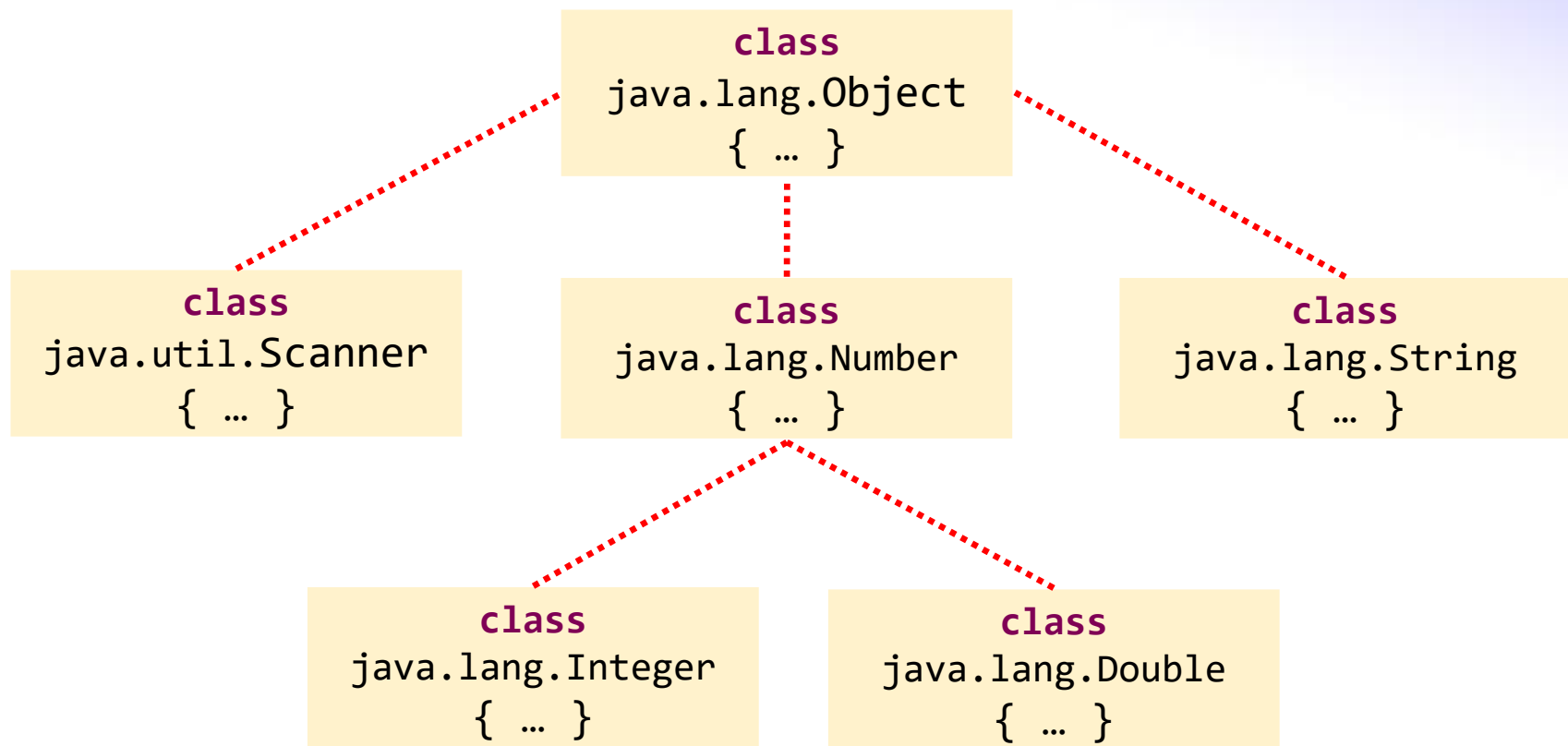
    void func2() {System.out.println("pro->pac");}
    // A의 접근 제한자 보다 더 적은 범위로 설정할 수 없다

    // Override 어노테이션 : 오버라이딩임을 컴파일러에게 명시한다
    @Override
    protected void func3() {
        System.out.println("pac->pro");
    }
}
```



Class – Object Class

Object : 자바에 정의된 모든 클래스의 최상위 클래스





Class – Object Class

Object : 자바에 정의된 모든 클래스의 최상위 클래스

- class가 다른 class를 상속받지 않는다면, 자동으로 extends Object 가 된다.
- `java.lang.Object` 에 정의되어 있다
- 자주 사용되는 메서드 중 일부는 Object 클래스에 정의되어 있다
- 이미 정의된 메서드를 오버라이딩 해서 새롭게 구성해서 쓸 수 있다
- 다형성에 의해서, 모든 객체는 Object 자료형으로 참조할 수 있다
- interface는 Object를 상속받지 않는다



Class – Object Class

Object 클래스의 주요 메서드

Object Method	설명
protected Object clone()	객체의 복사본을 반환
public boolean equals(Object obj)	객체를 비교하여 논리값을 반환
public final Class getClass()	객체의 클래스 정보를 반환
public int hashCode()	객체의 해시 코드를 반환
public String toString()	객체 자신의 정보를 문자열로 반환 객체 참조 변수를 출력할 때의 값