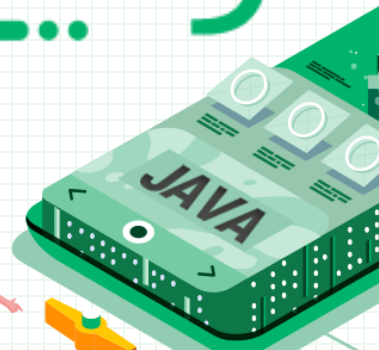




# 안드로이드 프로그래밍을 위한 자바기초 \_...



## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기



## 학습목표

- 상속에 대하여 이해하고 관련 기본 문법을 프로그래밍에 적용할 수 있다.
- 일관성과 다형성에 관련된 기본 문법을 프로그래밍에 적용할 수 있다.



## 학습내용

- 상속 기본 문법 이해하기
- 일관성 및 다형성 이해하기

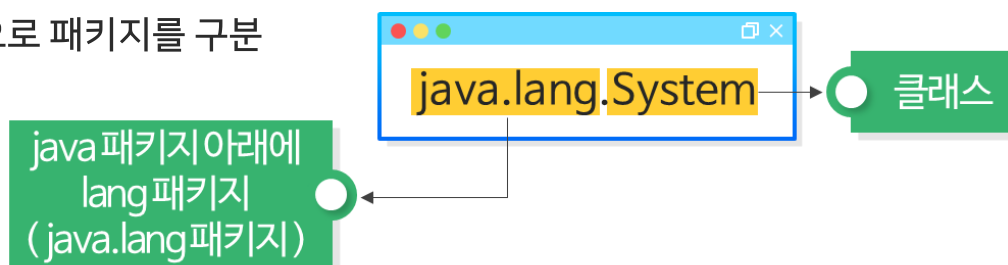
# ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 상속 기본 문법 이해하기

### ▶ 패키지(package) 이해하기

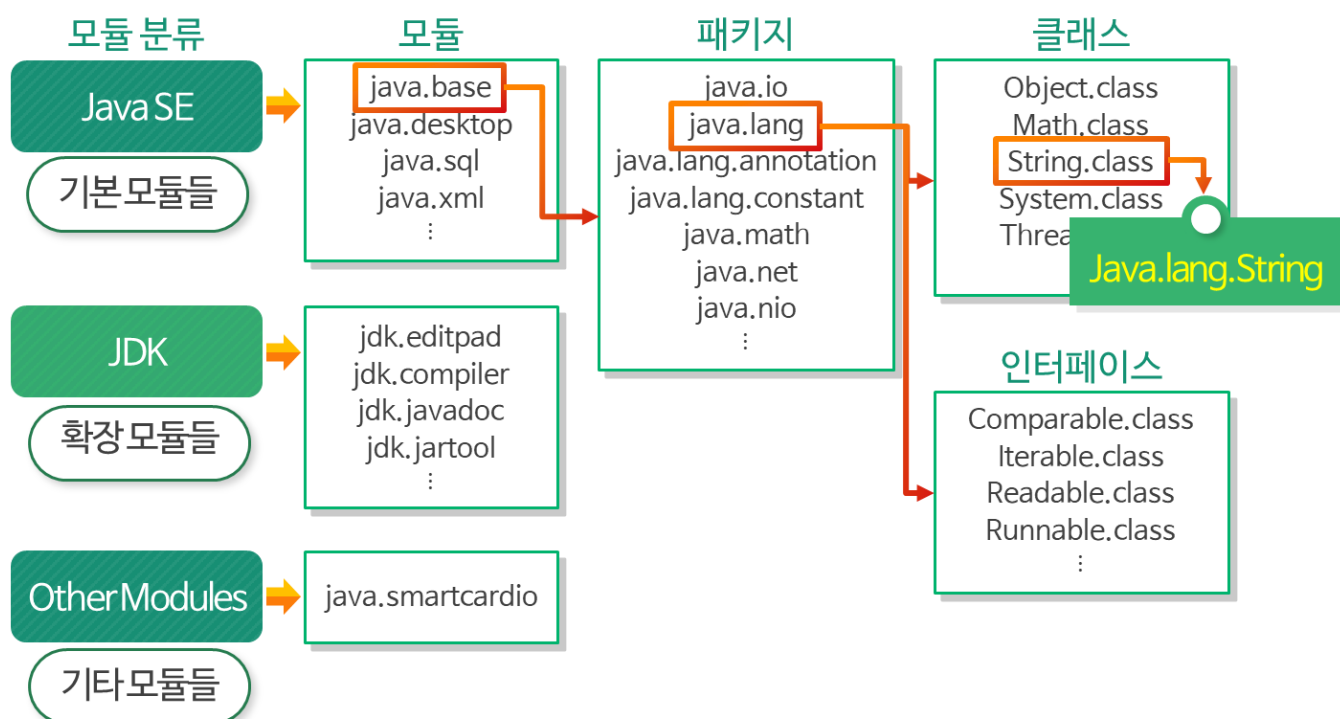
#### 1) 패키지란?

- 관련된 클래스 및 인터페이스 등을 모아 놓은 폴더
- 패키지에 클래스들이 존재함
- 패키지 이름은 소문자로 시작
- 점(.)으로 패키지를 구분



#### 2) 모듈이란?

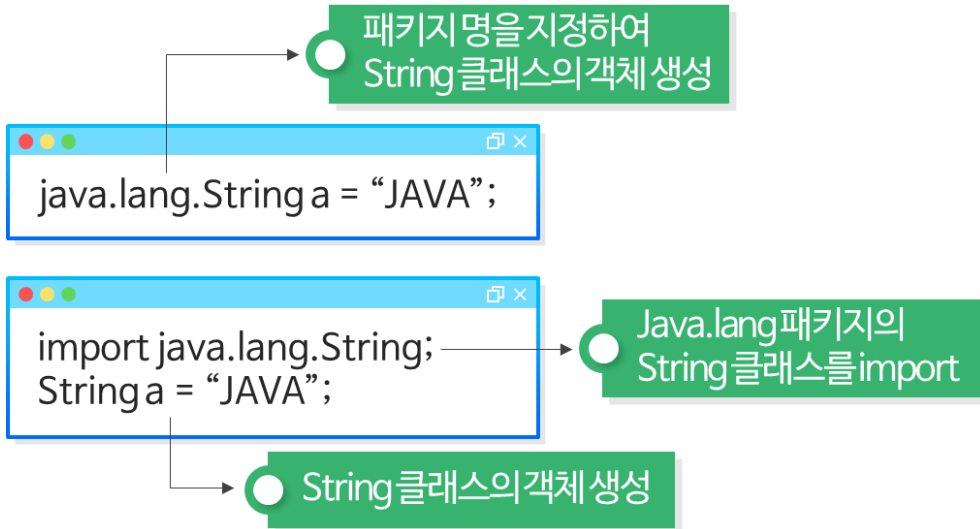
- 관련 패키지들을 모아 놓은 것
- 프로그램 실행 시 필요한 모듈만 사용하여 메모리를 효율적으로 사용
- 모듈이름은 소문자로 시작하고, 점(.)으로 구분



## 상속 기본 문법 이해하기

### 3) import 문

- 패키지에 있는 클래스들을 참조하기 위해 사용
- 소스의 제일 위에 정의
- import 문을 사용하지 않을 경우 패키지 이름을 지정해야 함



#### ■ 사용 방법

- ✓ 참조하고자 하는 클래스만 지정

```
import 패키지명.클래스명;
```

- ✓ 참조하고자 하는 패키지의 전체 클래스를 지정

```
import 패키지명.*;
```

- ▶ 해당 패키지 안에 또 다른 패키지가 존재하면 또 다른 패키지는 참조되지 않음

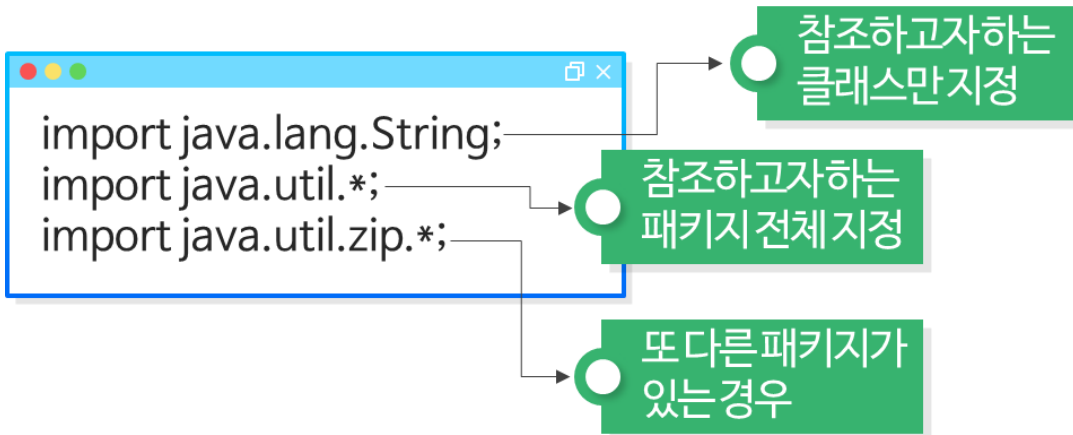
- ✓ 자바의 기본 패키지는 import 하지 않아도 됨 (자동 import 됨)

```
import java.lang.*;
```

- ▶ 생략 가능

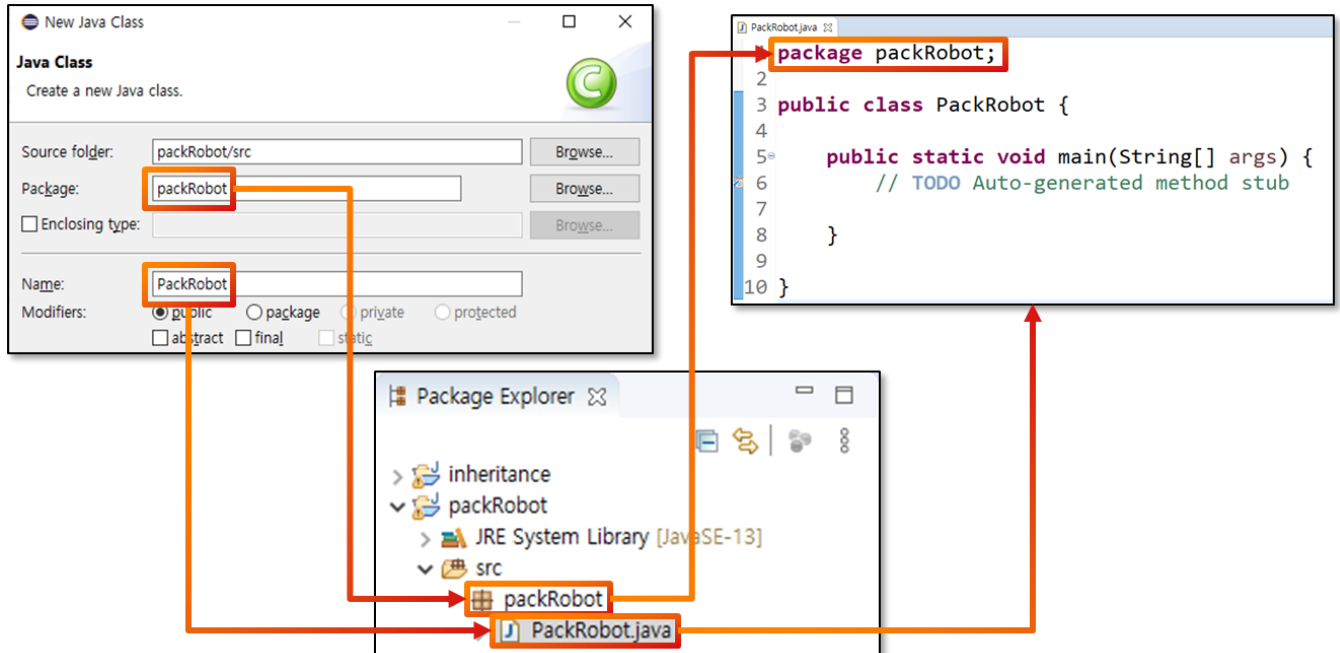
## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

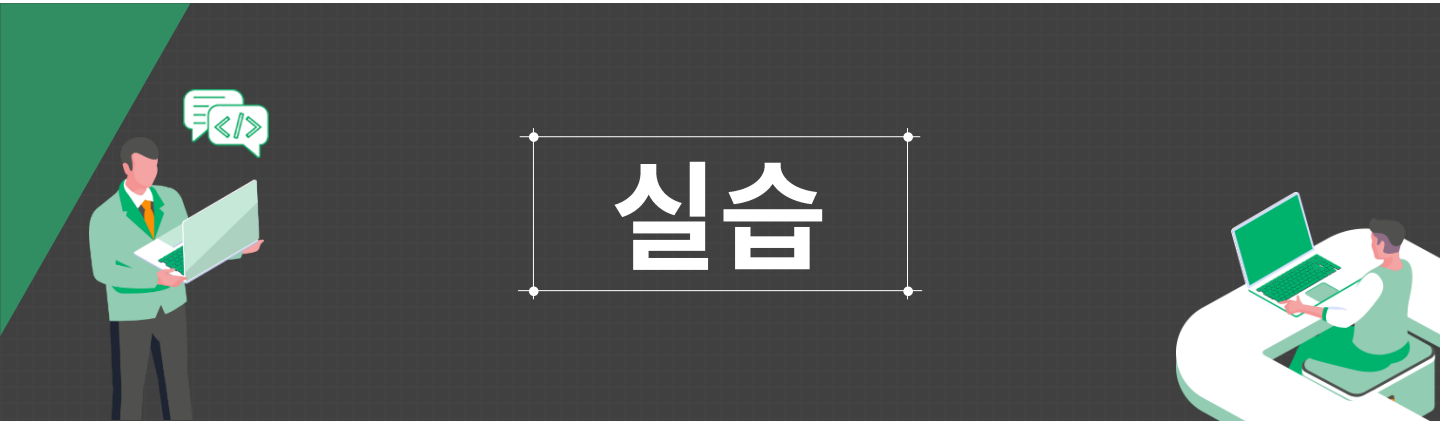
## 상속 기본 문법 이해하기



## 4) package문

- 프로그래머가 작성한 프로그램의 패키지를 지정
- 정의 방법
  - ✓ 클래스 생성 시 지정하면, package문이 자동으로 생성됨





## import와 package문 실습



### 실행 화면

자바 : 프로그래밍

Sat Oct 03 11:49:23 GMT+09:00 2020

Sat Oct 03 11:49:23 GMT+09:00 2020

- 소스 파일명 : [PackRobot.java]
- 자세한 내용은 실습 영상을 확인해보세요.

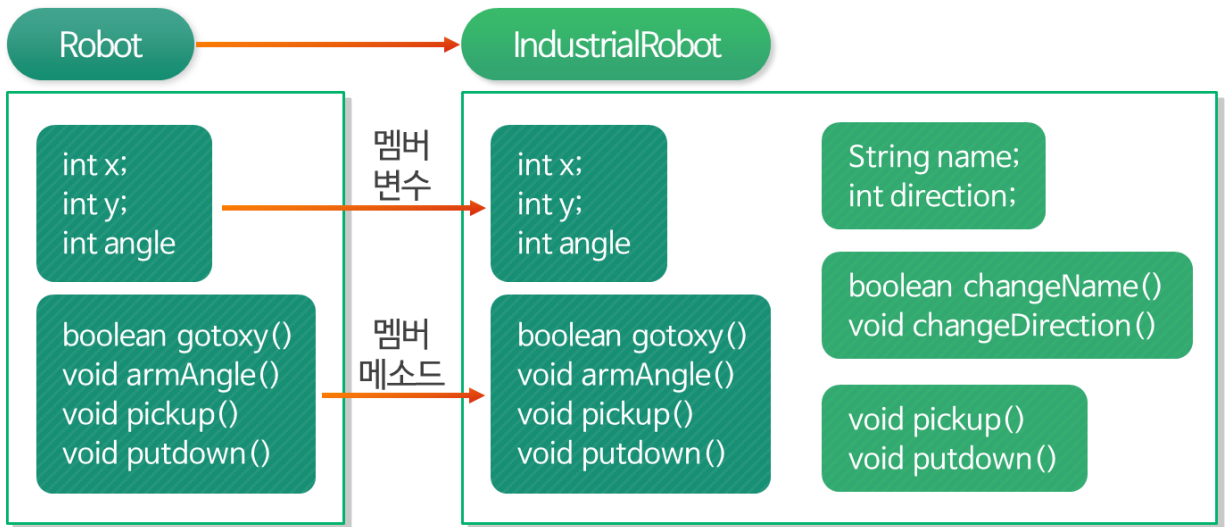
## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 상속 기본 문법 이해하기

## ➤ 상속(inheritance) 이해하기

## 1) 상속이란?

- 기존 클래스의 멤버 변수(필드)와 멤버 메소드 (메소드)를 모두 가짐
- 추가적인 멤버 변수와 멤버 메소드를 정의
- 기존의 멤버 변수와 멤버 메소드를 재정의

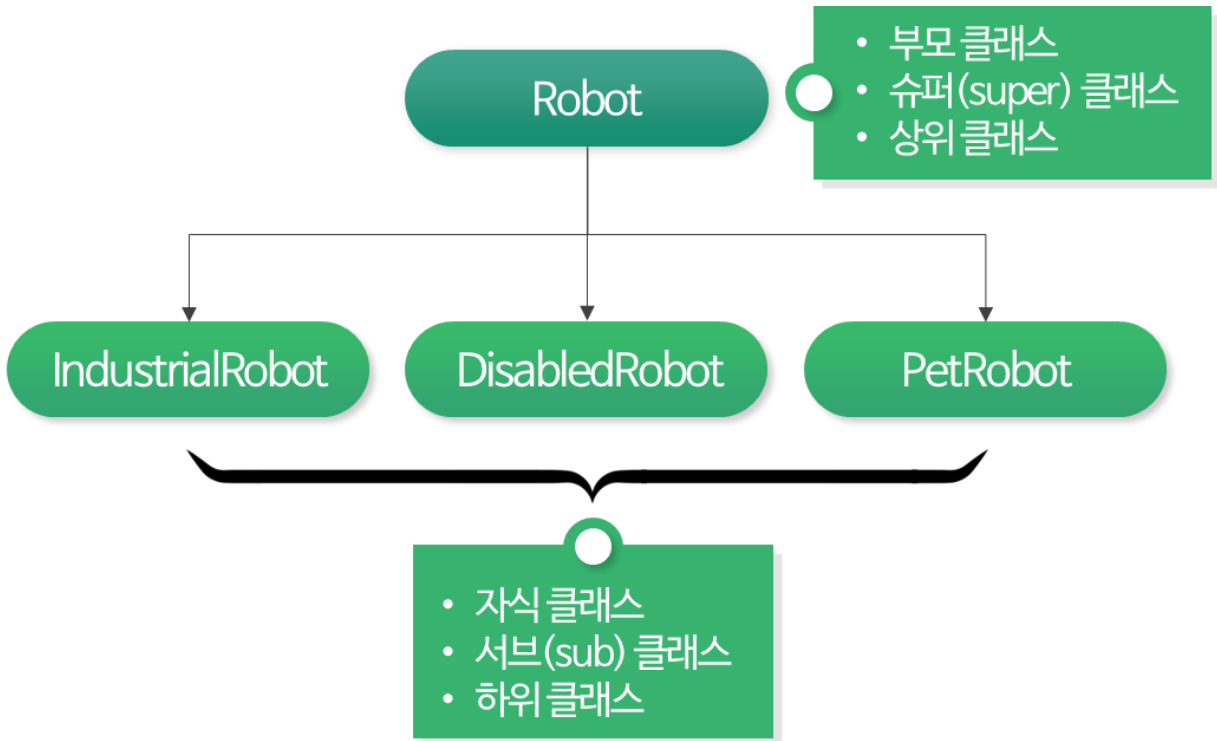


## 2) 상속의 구현

- 상속해 주는 클래스
  - ✓ 부모 클래스, 슈퍼(super) 클래스, 상위 클래스
- 상속받는 클래스
  - ✓ 자식 클래스, 서브(sub) 클래스, 하위 클래스
- 단일 상속
  - ✓ 부모 클래스는 반드시 하나이고, 자식클래스는 여러 개일 수 있음

## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 상속 기본 문법 이해하기

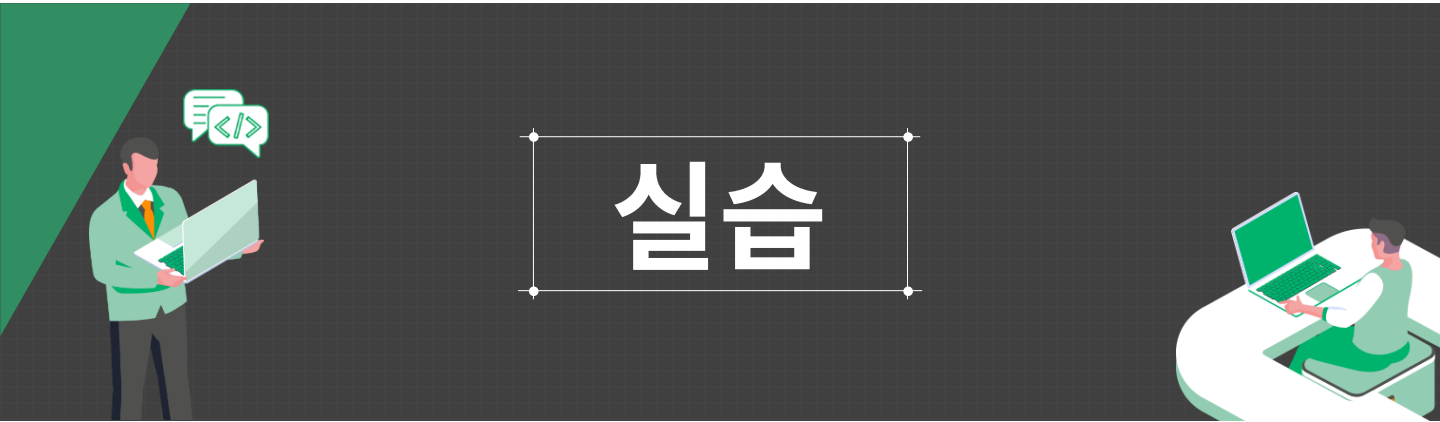


## ■ 상속을 정의하는 법

- ✓ extends 키워드를 사용

```
class Robot { ... }
class IndustrialRobot extends Robot { ... }
class DisabledRobot extends Robot { ... }
class PetRobot extends Robot { ... }
```





## 상속 기본 실습




### 실행 화면

```
Computer  
xy[25,25], angle[160] Pick Up !!  
xy[25,25], angle[160] Put Down !!  
SmartPhone  
xy[115,100], angle[365] Pick Up !!  
xy[115,100], angle[365] Put Down !!  
Job Count = 102
```

- 소스 파일명 : [inheritance]
- 자세한 내용은 실습 영상을 확인해보세요.

### 3) 접근 제한자(access modifier)

- (1) 클래스 또는 클래스의 멤버에 대한 접근을 제한하는 키워드
- (2) 클래스, 멤버 변수(필드), 멤버 메소드(메소드), 생성자 메소드에 사용
- (3) 종류

| 접근 제한자                | 설명   | 접근 가능 범위  |
|-----------------------|--|---|
| private               | 클래스 내부에서만 접근 가능  |  좁음<br><br><br><br><br><br><br><br><br><br>넓음 |
| default<br>(friendly) | <ul style="list-style-type: none"> <li>클래스 내부에서 접근 가능</li> <li>같은 패키지안의 클래스에서 접근 가능</li> <li>기본 접근 제한자 (접근 제한자를 지정하지 않으면 됨)</li> </ul> |   |
| protected             | <ul style="list-style-type: none"> <li>클래스 내부에서 접근 가능</li> <li>같은 패키지 안의 클래스에서는 접근 가능</li> <li>상속관계에 있는 클래스에서는 접근 가능</li> </ul>        |   |
| public                | <ul style="list-style-type: none"> <li>모든 클래스에서 접근 가능</li> <li>생성자 메소스는 public을 지정해야 다른 클래스에서 사용할 수 있음</li> </ul>                      |   |

# 실습

## 접근 제한자 실습



### 실행 화면

[Robot Class]

X=10

Y=20

getX()=10

getY()=20

JOB=100

[IndustrialRobot Class]

getX()=10

getY()=20

JOB=100

- 소스 파일명 : [modifierA], [modifierB]
- 자세한 내용은 실습 영상을 확인해보세요.

## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

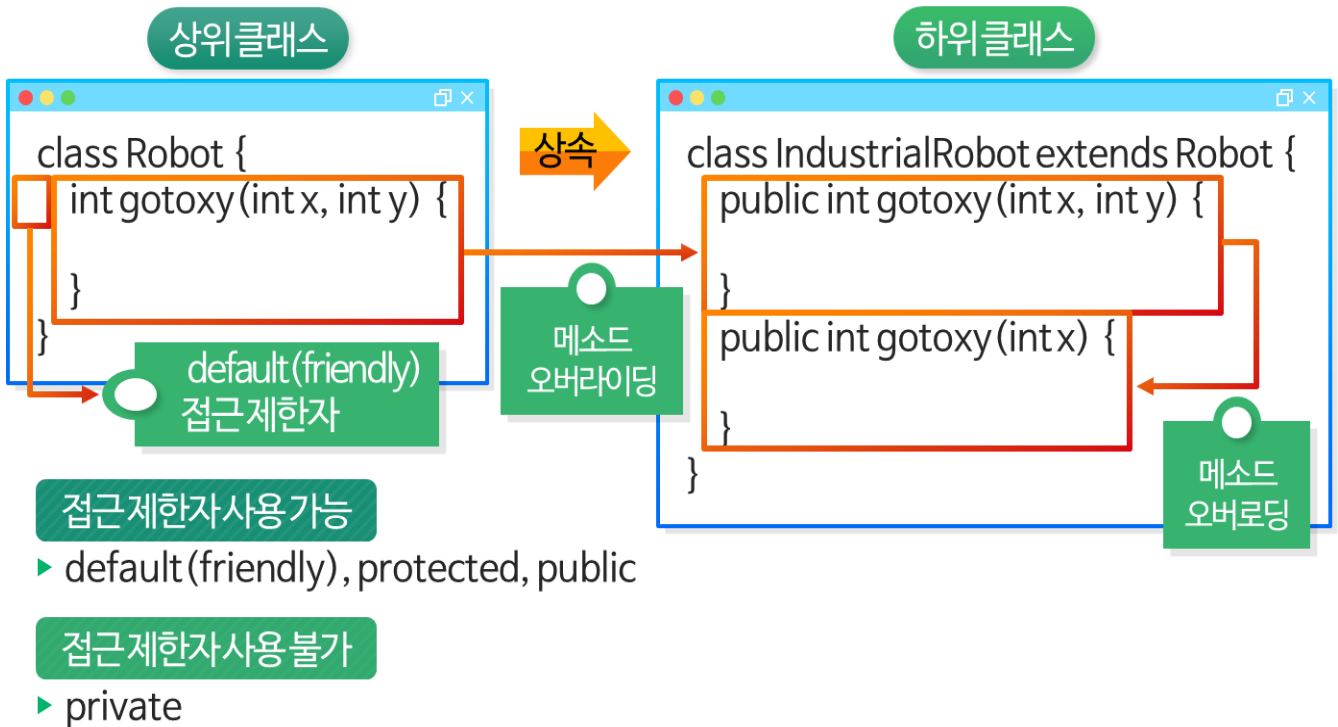
### 상속 기본 문법 이해하기

#### 4) 메소드 오버라이딩 (method overriding)

- (1) 하위 클래스에서 상위 클래스의 멤버 메소드를 동일하게 재정의 하는 것
- (2) 하위 클래스에서 메소드 이름, 매개 변수, 반환 자료형은 동일하게 재정의
- (3) 접근 제한자의 범위

✓ 하위 클래스  $\geq$  상위 클래스

- (4) 메소드의 내용은 다르게 프로그래밍함
- (5) 오버라이딩 된 메소드도 오버로딩 될 수 있음



## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 상속 기본 문법 이해하기

### 5) super 키워드

- 상위 클래스의 객체를 참조하는 참조변수
- 하위 클래스에서 상위 클래스의 멤버 변수 또는 멤버 메소드를 참조할 때 사용

상위클래스

```
class Robot {
    int x, y;
    int gotoxy(int x, int y) {
    }
}
```

하위클래스

```
class IndustrialRobot extends Robot {
    int x, y;
    public int setxy(int a, int b) {
        gotoxy(10,20);
        super.gotoxy(x, y);
        super.x = 10;
        super.y = 10;
        this.x = 20;
        this.y = 20;
    }
    int gotoxy(int x, int y) { ... }
}
```

### 6) super() 메소드

- (1) 상위 클래스의 생성자 메소드를 명시적으로 호출
- (2) 하위 클래스의 생성자 메소드에서 첫 줄에 한번만 사용 가능
- (3) 소스 코드의 중복을 줄이기 위함
- (4) 하위 클래스 객체 생성 시

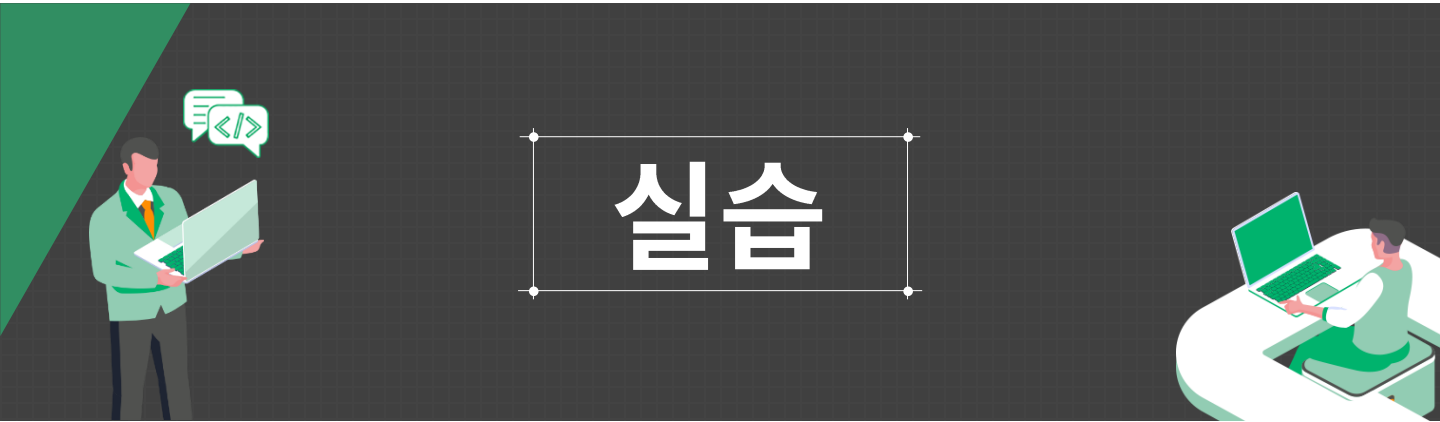
- ✓ 상위 클래스의 디폴트 생성자(매개 변수 없는 생성자)가 자동으로 호출됨
- ✓ 명시적으로 상위 클래스의 생성자를 호출할 경우 super() 메소드 사용함

## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 상속 기본 문법 이해하기

(5) 생성자 메소드 정의 유무에 따른 실행 결과

| 상위 클래스   | 하위 클래스   | 실행 결과                           |
|--|--|---------------------------------|
| 생성자메소드를<br>정의하지않거나,<br>디폴트 생성자메소드를<br>정의한 경우           | <ul style="list-style-type: none"> <li>• 생성자메소드를 정의한 경우</li> <li>• 생성자메소드를 정의하지 않은 경우</li> </ul> | 오류 없음                           |
| 디폴트 생성자메소드를<br>정의하지 않고,<br>매개 변수가 있는<br>생성자메소드만 정의된 경우 | • 생성자메소드를 정의하지 않은 경우   | 오류 발생                           |
|  | • 디폴트 생성자메소드만 정의한 경우   | super() 메소드를 이용하여<br>명시적 호출해야 함 |
|  | • 매개 변수 있는 생성자 메소드만 정의한 경우   |                                 |



## 오버라이딩 실습



### 실행 화면

```
[Robot Class]
[X=45]
[Y=45]
[ANGLE=100]
[JOB=0]
[IndustrialRobot Class] computer
[X=25]
[Y=25]
```

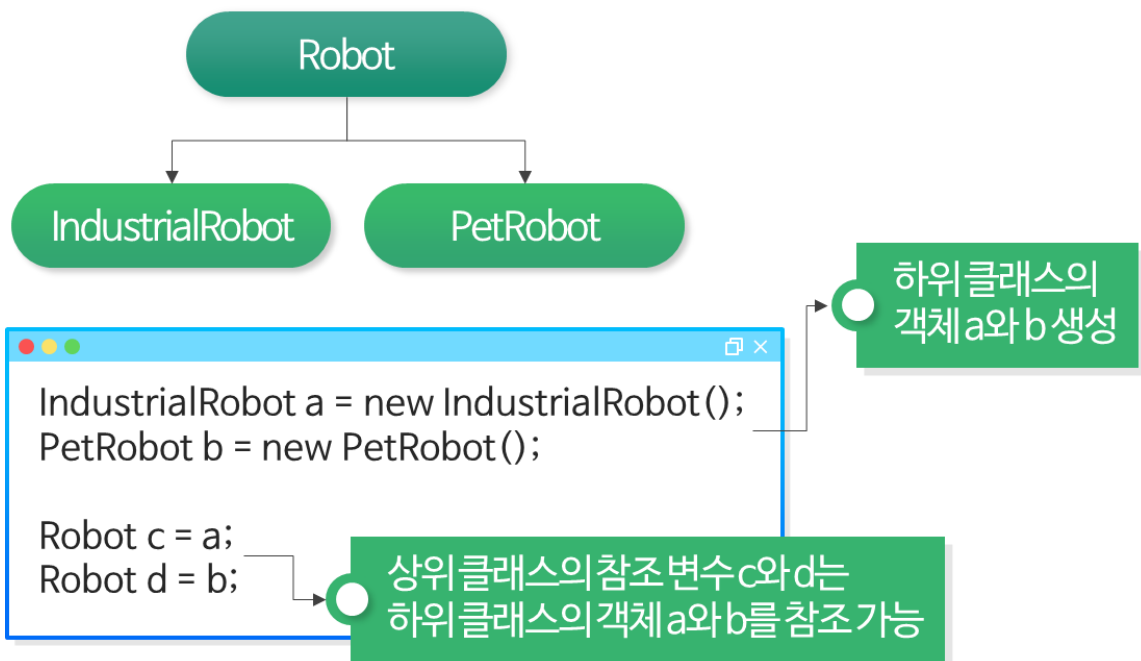
- 소스 파일명 : [overriding]
- 자세한 내용은 실습 영상을 확인해보세요.

## 일관성 및 다형성 이해하기

### ▶ 객체 형변환

#### 1) 업 캐스팅 (up casting)

- 상위 클래스의 참조 변수가 하위 클래스의 객체를 참조하는 것
- 묵시적 형변환이라고도 함
- 하위 클래스에서 추가된 멤버는 참조하지 못 함

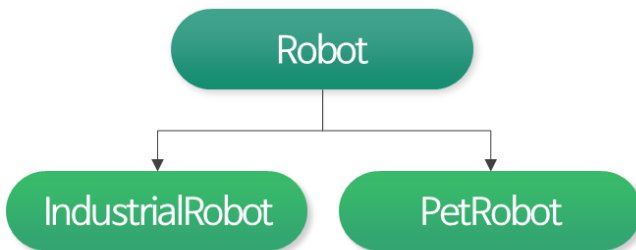




## 일관성 및 다형성 이해하기

### 2) 다운 캐스팅 (down casting)

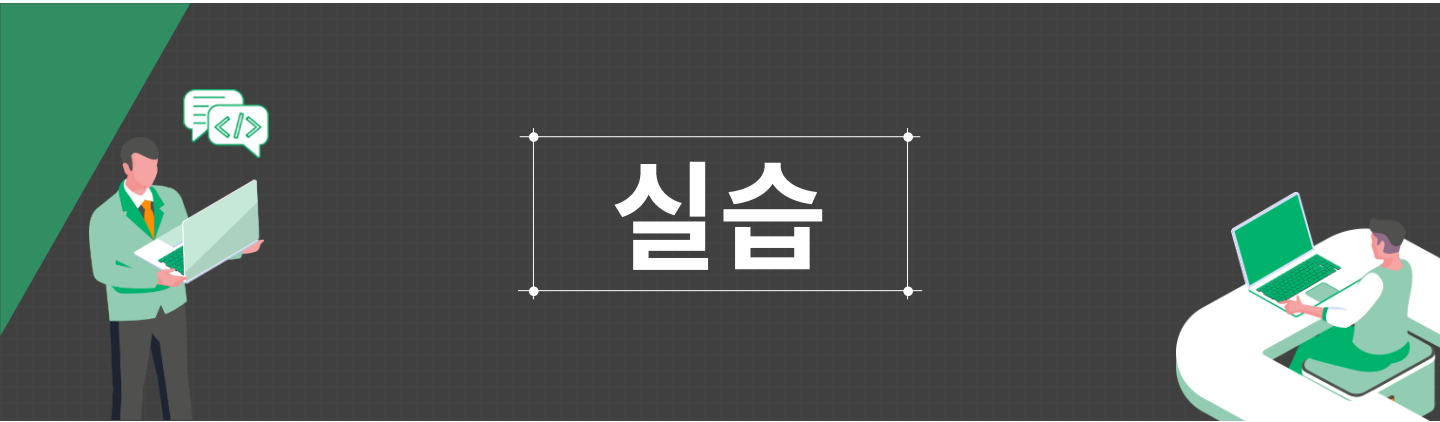
- 하위 클래스의 참조 변수가 상위 클래스의 객체를 참조하는 것
- 명시적 형변환이라고도 함
- 형변환 연산자 사용하여야 함
- 상위 클래스의 참조 변수가 실제로 하위 클래스의 객체를 참조하는 경우



```
Robot a = new IndustrialRobot();  
Robot b = new PetRobot();  
  
IndustrialRobot c = (IndustrialRobot)a;  
PetRobot d = (PetRobot)b;
```

상위 클래스의 참조 변수가  
하위 클래스의 객체를 참조 (업캐스팅)

상위 클래스의 참조 변수 c와 d는  
하위 클래스의 객체 a와 b를 참조 가능



## 객체 형변환 실습



### 실행 화면

```
[Robot]
[X=30]
[Y=30]
[ANGLE=10]
[JOB=0]
[IndustrialRobot]
computer
[Robot]
[X=40]
[Y=40]
[ANGLE=10]
[JOB=0]
[PetRobot] ...
```

- 소스 파일명 : [casting]
- 자세한 내용은 실습 영상을 확인해보세요.

## 일관성 및 다형성 이해하기

### ➡ 추상 클래스(abstract class)

#### 1) 추상 클래스란?

##### (1) 추상 메소드를 가지는 클래스

- ✓ 추상 메소드 : 메소드의 이름만 정의하고 내용은 구현되어 있지 않는 메소드

##### (2) abstract 키워드를 지정하여 클래스와 메소드를 정의

##### (3) 추상 메소드 뿐만 아니라 정상적인 메소드도 정의할 수 있음

##### (4) 추상 클래스를 상속 받은 하위 클래스들은 반드시 추상 메소드를 오버라이딩 해야함

- ✓ 같은 상위 클래스를 가지는 하위 클래스들은 일관성을 가짐

##### (5) 오버라이딩 한 메소드는 프로그램의 내용에 따라 다양하게 실행될 수 있음

- ✓ 동일한 이름의 메소드들이 다양하게 실행됨(다형성)

##### (6) 추상 클래스를 상속하여 또다른 추상 클래스를 만들 수 있음

- ✓ 추상 클래스들이 가지고 있는 모든 추상 메소드들을 오버라이딩 해야 함

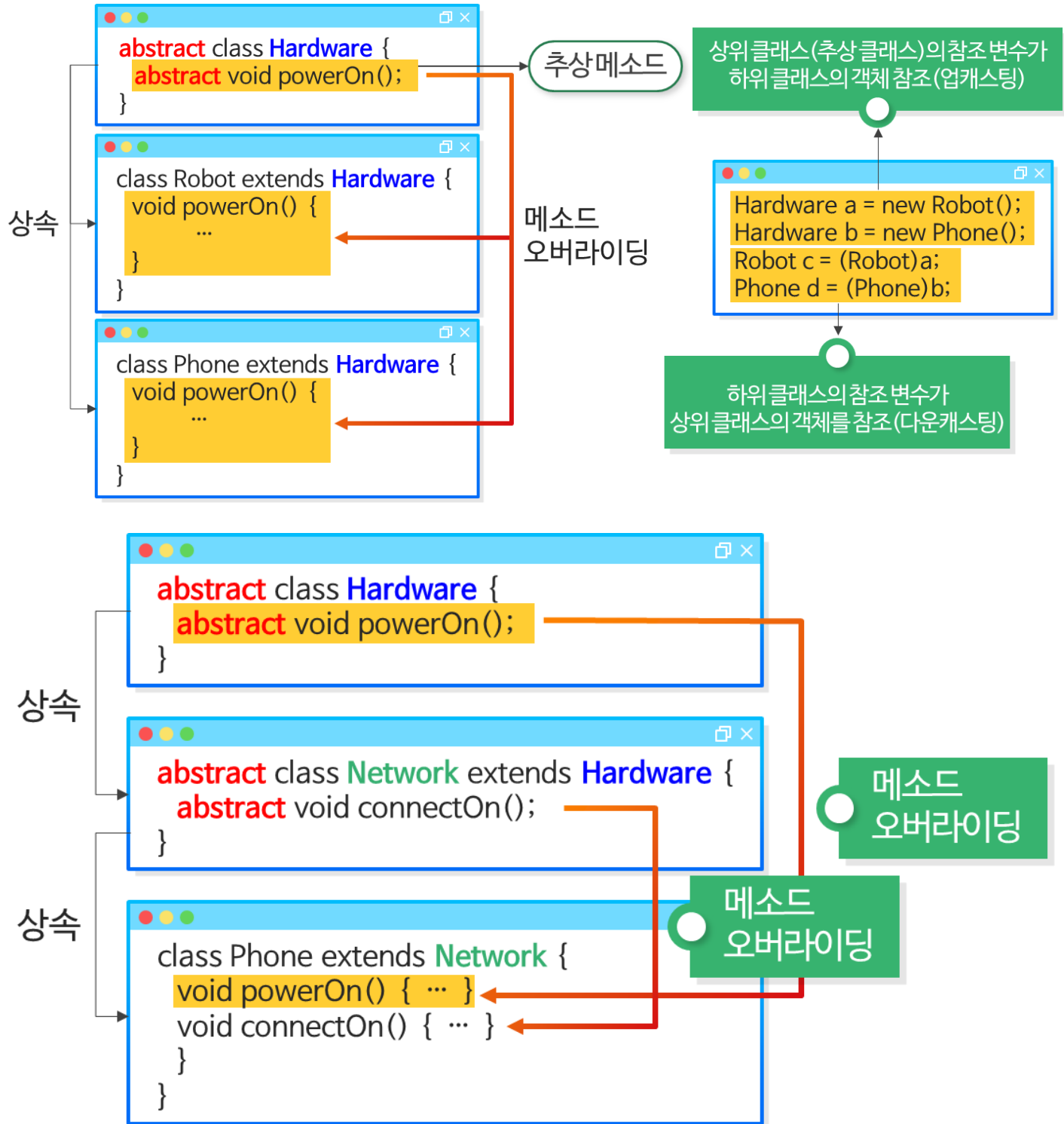
##### (7) 추상 클래스의 참조 변수는 하위 클래스의 객체를 참조할 수 있음

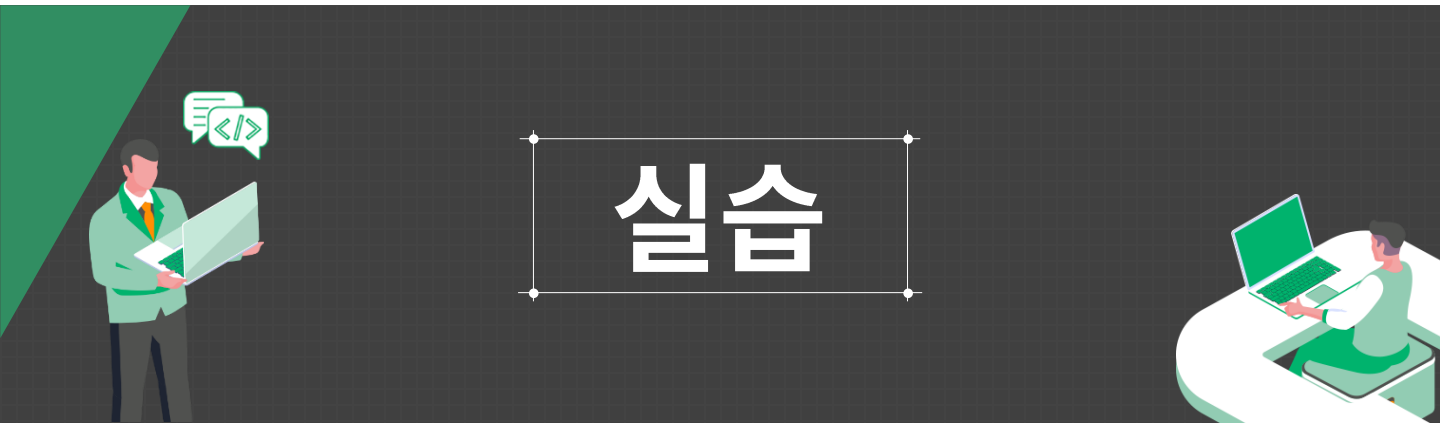
- ✓ 객체 형변환 가능

# ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 일관성 및 다형성 이해하기

### 2) 추상 클래스 구현





## 추상 클래스 실습



### 실행 화면

```
[Robot Class]
Hardware[0].power = true
Hardware[1].power = true
[Phone Class]
power = true
connect = true
```

- 소스 파일명 : [abstractsrc], [abstractclass]
- 자세한 내용은 실습 영상을 확인해보세요.

## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 일관성 및 다형성 이해하기

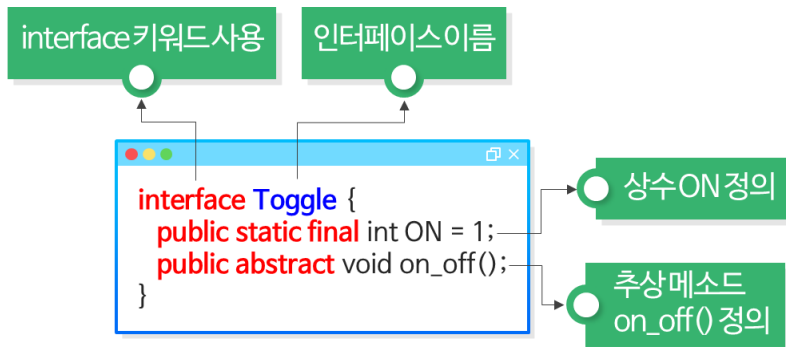
## ▶ 인터페이스(interface)

## 1) 인터페이스란?

- 자바는 단일 상속만 허용, 다중 상속의 대안으로 사용

## 2) 인터페이스 정의

- (1) interface 키워드를 사용하여 정의
- (2) 상수와 메소드만 정의 가능
- (3) 접근제어는 일반적으로 public



## 3) 인터페이스 구현

- (1) 정의된 인터페이스를 사용하는 것을 "구현한다"라고 함
- (2) implement 키워드를 사용
- (3) 여러 개의 인터페이스를 구현할 수 있음(콤마로 구분)
- (4) 인터페이스를 구현한 클래스에서 인터페이스의 추상 메소드들을 모두 재정의해야함

```
interface Toggle {
    public static final int ON = 1;
    public abstract void on_off();
}
```

```
interface PushButton {
    public abstract void press();
}
```

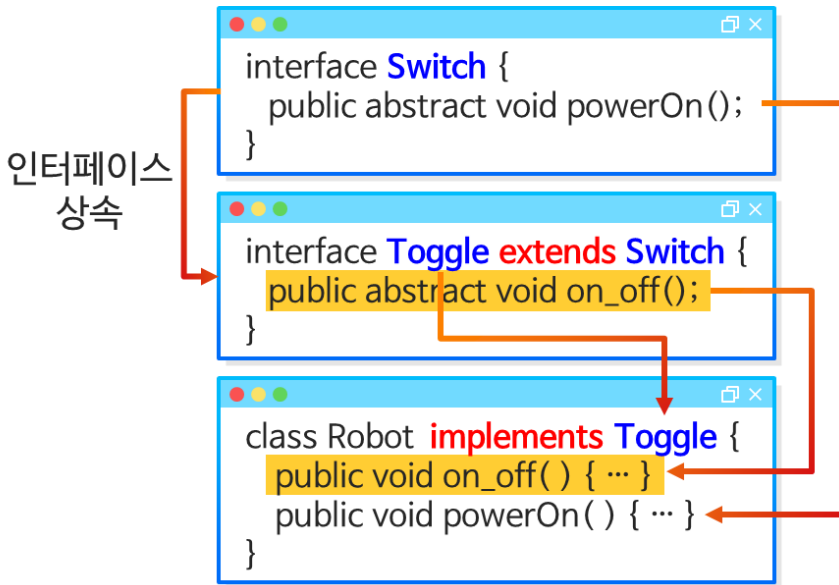
```
class Robot implements Toggle, PushButton {
    public void on_off() { ... }
    public void press() { ... }
}
```

## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 일관성 및 다형성 이해하기

## 4) 인터페이스 상속

- 인터페이스를 상속하여 또 다른 인터페이스를 정의할 수 있음
- ✓ 인터페이스 들이 가지고 있는 모든 추상 메소드를 재정의하여야 함



## 5) 인터페이스 참조 변수

- 인터페이스를 구현한 클래스의 객체는 인터페이스

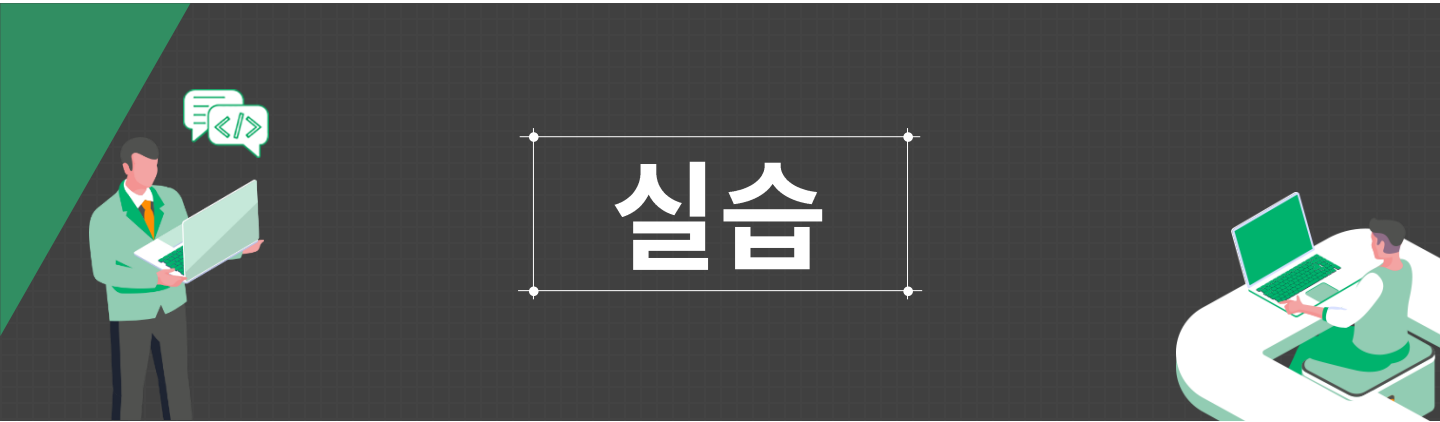
```

interface Switch { ... }
class Robot implements Switch { ... }
class Phone implements Switch { ... }

Switch a = new Robot();
Switch b = new Phone();

```

참조 변수 a와 b는  
Robot과 Phone 클래스의 객체 참조 가능



## 인터 페이스 실습



### 실행 화면

```
[Robot Class]
[Toggle]
[PushButton]
Toggle[0] : [Toggle]
Toggle[1] : [Toggle]
[Phone Class]
[Toggle]
[Switch.RED] = 1
```

- 소스 파일명 : [interfacesrc], [interfaces]
- 자세한 내용은 실습 영상을 확인해보세요.



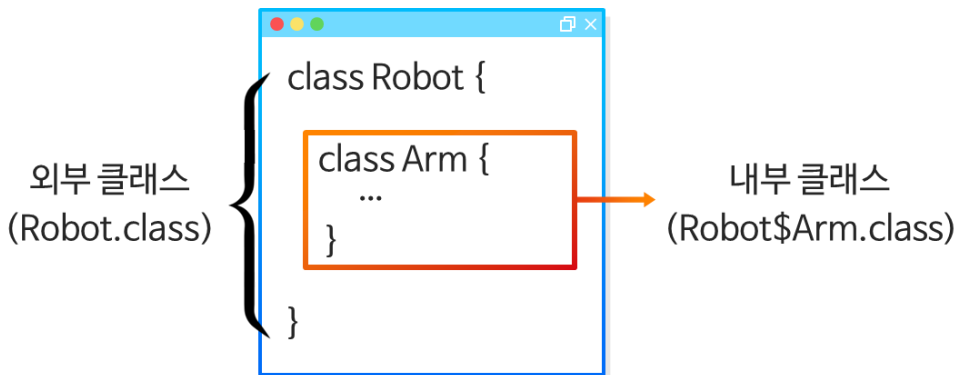
## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 일관성 및 다형성 이해하기

## ▶ 내부 클래스(inner class)

## 1) 내부 클래스란?

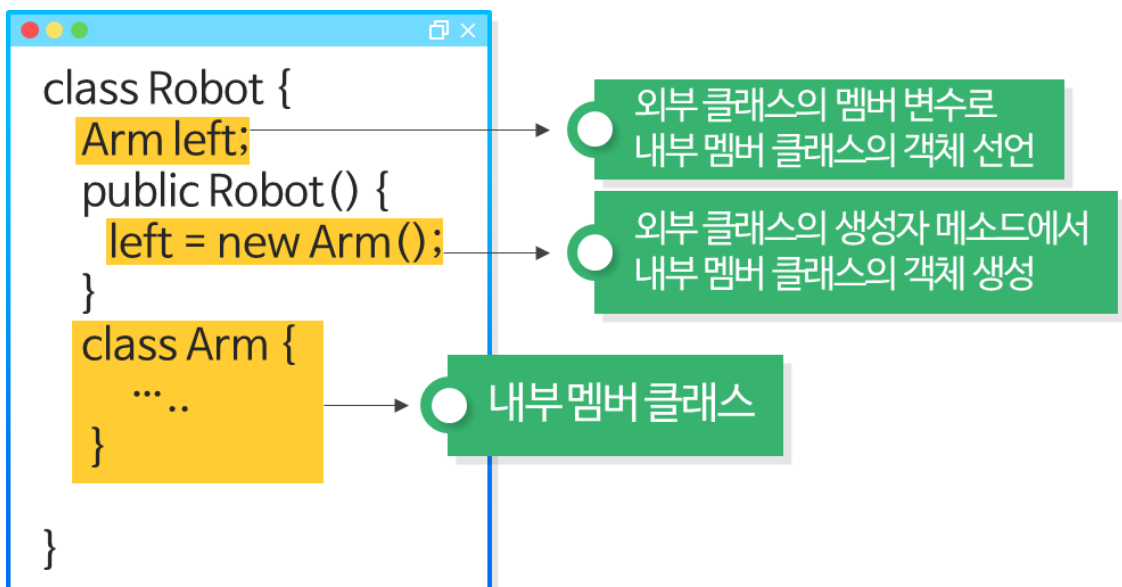
- 중첩된 클래스(nested class)로 클래스 내부에서 또다시 선언된 클래스
- 외부 클래스에서 내부클래스의 객체를 생성해서 사용
- 내부 클래스에서는 외부 클래스의 멤버에 public 접근



## 2) 내부 클래스 종류

## (1) 내부 멤버 클래스

- ✓ 클래스의 멤버와 동일한 레벨로 내부 클래스 정의
- ✓ 외부 클래스의 생성자 메소드에서 내부 멤버 클래스의 객체 생성

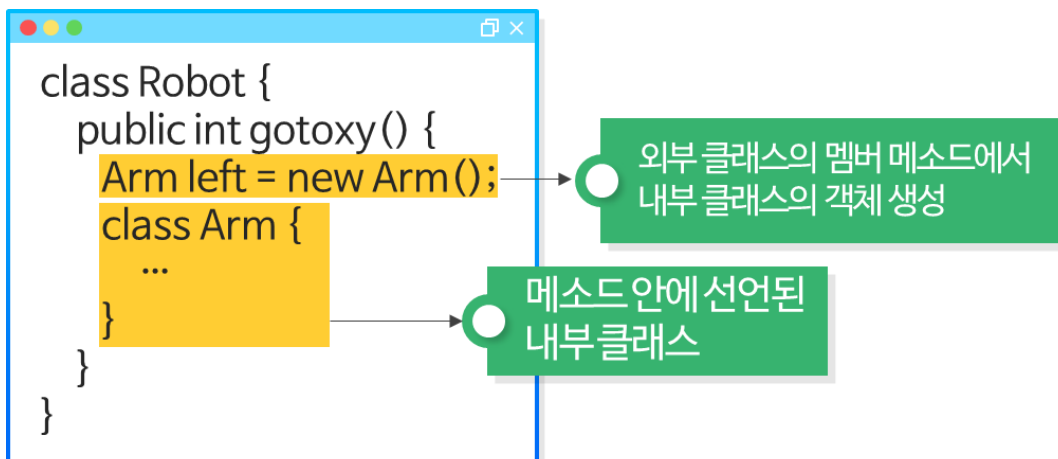


## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

### 일관성 및 다형성 이해하기

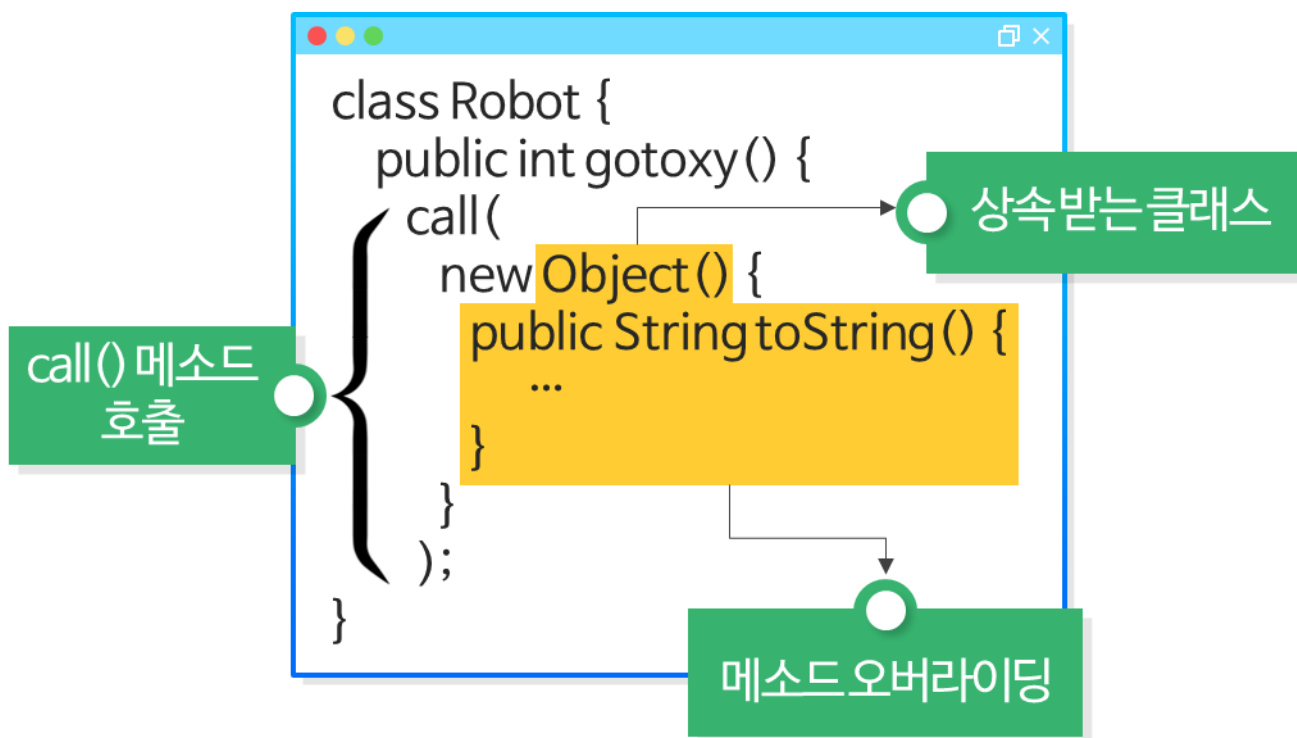
#### (2) 메소드 안에서 선언된 내부 클래스

- ✓ 외부 클래스의 멤버 메소드 안에 선언된 내부 클래스
- ✓ 외부 클래스의 멤버 메소드 내부에서만 사용 가능



#### (3) 내부 무명 클래스

- ✓ 클래스의 이름없는 내부 클래스 : 내부 익명 (Anonymous) 클래스
- ✓ 상속 받는 클래스의 선언과 동시에 객체를 생성하여 사용
- ✓ 주로 메소드의 인수로 객체를 지정할 때 사용



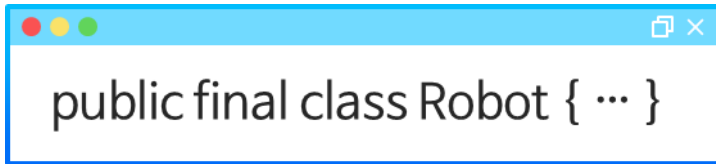
## ⑦ 상속을 활용한 객체지향 프로그래밍 기본 문법 이해하기

## 일관성 및 다형성 이해하기

### ➡ final

#### 1) 클래스 선언 시 지정

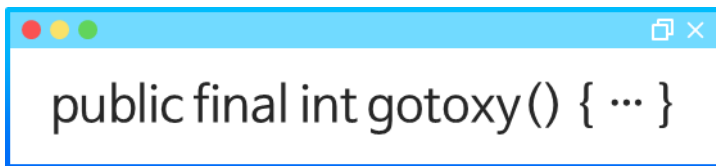
- 해당 클래스는 더 이상 상속을 허락하지 않음



```
public final class Robot { ... }
```

#### 2) 메소드 선언 시 지정

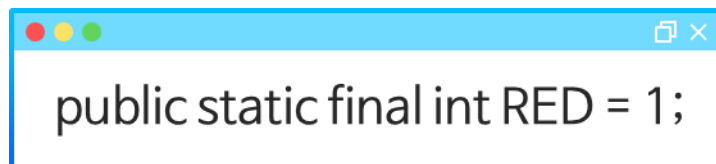
- 해당 메소드는 더 이상 오버라이딩을 허락하지 않음



```
public final int gotoxy() { ... }
```

#### 3) 변수 선언 시 지정

- 해당 변수는 더 이상 값의 변경을 허락하지 않음
- static 키워드와 함께 정의하여 상수를 정의함



```
public static final int RED = 1;
```

# 실습

## 중첩 클래스와 final 실습



### 실행 화면

```
[내부 멤버 클래스]
[Robot.Arm.getPower()] = 5
[메소드안에 정의된 중첩 클래스]
[MOVE][X] = 20
[MOVE][Y] = 20
[내부 익명 클래스]
{내부 익명 클래스}
```

- 소스 파일명 : [inner]
- 자세한 내용은 실습 영상을 확인해보세요.



## 정리하기

### ■ 상속 기본 문법 이해하기

- 패키지(package)란?
  - 관련된 클래스 및 인터페이스 등을 모아 놓은 폴더
- 모듈(module)이란?
  - 관련 패키지들을 모아 놓은 것
  - 프로그램 실행 시 필요한 모듈만 사용하여 메모리를 효율적으로 사용
- import 문
  - 패키지에 있는 클래스들을 참조하기 위해 사용
- package문
  - 프로그래머가 작성한 프로그램의 패키지를 지정
- 상속(inheritance)
  - 기존 클래스의 멤버 변수와 멤버 메소드를 모두 가짐
  - 추가적인 멤버 변수와 멤버 메소드를 정의
  - 기존의 멤버 변수와 멤버 메소드를 재정의
  - 상속해 주는 클래스 : 부모 클래스, 슈퍼(super) 클래스, 상위 클래스
  - 상속받는 클래스 : 자식 클래스, 서브(sub) 클래스, 하위 클래스
  - 단일 상속 : 부모 클래스는 반드시 하나이고, 자식 클래스는 여러 개일 수 있음
  - 상속을 정의하는 법 : extends 키워드를 사용
- 접근 제한자(access modifier)
  - 클래스 또는 클래스의 멤버에 대한 접근을 제한하는 키워드
  - 클래스, 멤버 변수(필드), 멤버 메소드(메소드), 생성자 메소드에 사용



## 정리하기

### ■ 일관성 및 다형성 이해하기

- 객체 형변환
  - 업 캐스팅(up casting): 상위 클래스의 참조 변수가 하위 클래스의 객체를 참조하는 것
  - 다운 캐스팅(down casting): 하위 클래스의 참조 변수가 상위 클래스의 객체를 참조하는 것
- 추상 클래스(abstract class)
  - 추상 메소드를 가지는 클래스
  - abstract 키워드를 지정하여 클래스와 메소드를 정의
  - 추상 클래스를 상속 받은 하위 클래스들은 반드시 추상 메소드를 오버라이딩 해야 함
  - 추상 클래스를 상속하여 또 다른 추상 클래스를 만들 수 있음
  - 추상 클래스의 참조 변수는 하위 클래스의 객체를 참조할 수 있음 (객체 형변환 가능)
- 인터페이스(interface)
  - 자바는 단일 상속만 허용, 다중 상속의 대안으로 사용
  - 인터페이스 정의: interface 키워드를 사용
  - 인터페이스 구현: implements 키워드를 사용
- 내부 클래스(inner class)
  - 클래스 내부에 또다시 선언된 클래스
  - 종류: 내부 클래스, 메소드 안에 선언된 내부 클래스, 내부 무명(익명) 클래스
- final
  - 클래스 선언 시 지정
  - 메소드 선언 시 지정
  - 변수 선언 시 지정