7. 상속(Inheritance)

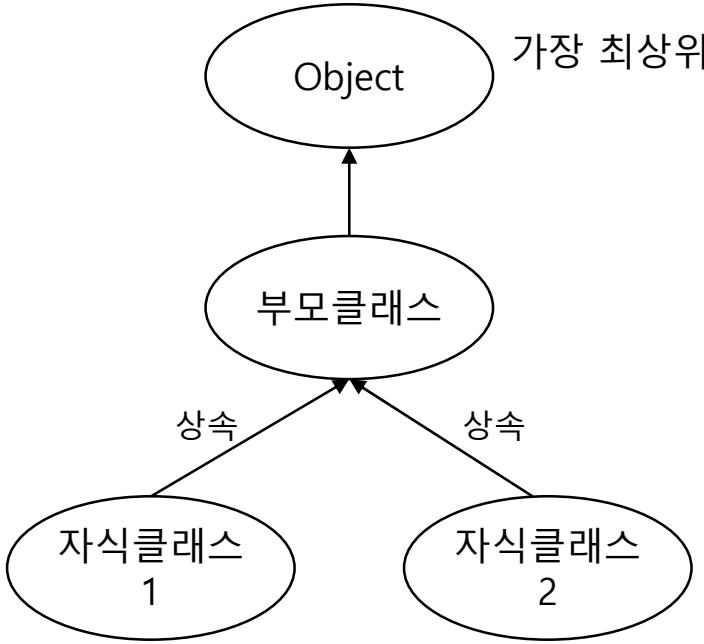
상속(Inheritance)

- 기존의 클래스를 확장하여 새로운 클래스를 만들 수 있게 해주는 기능으로서 객체지향 프로그래밍의 중요한 철학 중 하나인 재사용성을 풍부하게 해주는 개념

  - 기존 코드를 이용하므로 프로그램의 생산성과 유지보수의 편이성을 가능하게 함

  - 상속을 받는 클래스는 extends 키워드를 통해 상속할 부모 클래스를 정의하는데, 이 때 부모 클래스는 이미 구현되어져 있어야 함
    - 자식 클래스는 하나의 부모클래스만 상속 가능

  - 자식 클래스는 부모 클래스의 생성자, 멤버변수 및 메소드를 모두 상속



가장 최상위에는 Object 클래스가 있음

```
class ParentClass{
    int value; // 자식 클래스로 상속

    void method(){ … } // 자식 클래스로 상속
}

class ChildClass extends ParentClass{
    this.value = 10;

    method();
}
```

실습(1)

[Ex1-1] CellPhone.java

```java
package chap07.inheritance;

public class CellPhone {
    String model;
    String color;

    void powerOn() {
        System.out.println("전원을 켭니다.");
    }
    void powerOff() {
        System.out.println("전원을 끕니다.");
    }
    void bell() {
        System.out.println("벨이 울립니다.");
    }
    void sendVoice(String message) {
        System.out.println("자기: " + message);
    }
    void receiveVoice(String message) {
        System.out.println("상대방: " + message);
    }
    void hangUp() {
        System.out.println("전화를 끊습니다.");
    }
}
```

실습(2)

[Ex1-2] DmbCellPhone.java

```
package chap07.inheritance;

public class DmbCellPhone extends CellPhone {
    //필드
    int channel;

    //생성자
    DmbCellPhone(String model, String color, int channel) {
        this.model = model;
        this.color = color;
        this.channel = channel;
    }

    //메소드
    void turnOnDmb() {
        System.out.println("채널 " + channel + "번 DMB 방송 수신을 시작합니다.");
    }
    void changeChannelDmb(int channel) {
        this.channel = channel;
        System.out.println("채널 " + channel + "번으로 바꿉니다.");
    }
    void turnOffDmb() {
        System.out.println("DMB 방송 수신을 멈춥니다.");
    }
    void powerOnDmb() {
        powerOn();
        System.out.println("DMB를 켭니다.");
    }
}
```

실습(3)

[Ex1-3] DmbCellPhoneExample.java

```java
package chap07.inheritance;

public class DmbCellPhoneExample {
    public static void main(String[] args) {
        //DmbCellPhone 객체 생성
        DmbCellPhone dmbCellPhone = new DmbCellPhone("자바폰", "검정", 10);

        //CellPhone으로부터 상속 받은 필드
        System.out.println("모델: " + dmbCellPhone.model);
        System.out.println("색상: " + dmbCellPhone.color);

        //DmbCellPhone의 필드
        System.out.println("채널: " + dmbCellPhone.channel);

        //CellPhone으로부터 상속 받은 메소드 호출
        dmbCellPhone.powerOnDmb();
        dmbCellPhone.bell();
        dmbCellPhone.sendVoice("여보세요");
        dmbCellPhone.receiveVoice("안녕하세요! 저는 홍길동인데요");
        dmbCellPhone.sendVoice("아~ 예 반갑습니다.");
        dmbCellPhone.hangUp();

        //DmbCellPhone의 메소드 호출
        dmbCellPhone.turnOnDmb();
        dmbCellPhone.changeChannelDmb(12);
        dmbCellPhone.turnOffDmb();
    }
}
```

실습(4)

[Ex2-1] Shape.java

```java
package chap07.inheritance;

class Shape {
    String color = "black";
    void draw() {
        System.out.printf("[color=%s]%n", color);
    }
}
```

[Ex2-2] Point.java

```java
package chap07.inheritance;

class Point {
    int x;
    int y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    Point() {
        this(0,0);
    }

    String getXY() {
        return "("+x+","+y+")"; // x와 y의 값을 문자열로 반환
    }
}
```

실습(5)

[Ex2-3] Circle.java

```java
package chap07.inheritance;

class Circle extends Shape {
    Point center;   // 원의 원점좌표
    int r;          // 반지름

    Circle() {
        this(new Point(0, 0), 100); // Circle(Point center, int r)를 호출
    }

    Circle(Point center, int r) {
        this.center = center;
        this.r = r;
    }

    void draw() { // 원을 그리는 대신에 원의 정보를 출력하도록 했다.
        System.out.printf("[center=(%d, %d), r=%d, color=%s]%n", center.x, center.y, r, color);
    }
}
```

[Ex2-4] Triangle.java

```java
package chap07.inheritance;

class Triangle extends Shape {
    Point[] p = new Point[3];

    Triangle(Point[] p) {
        this.p = p;
    }

    void draw() {
        System.out.printf("[p1=%s, p2=%s, p3=%s, color=%s]%n", p[0].getXY(), p[1].getXY(), p[2].getXY(), color);
    }
}
```

실습(6)

[Ex2-5] DrawShape.java

```
package chap07.inheritance;

class DrawShape {
    public static void main(String[] args) {
        Point[] p = {   new Point(100, 100),
                        new Point(140,  50),
                        new Point(200, 100)
                    };

        Triangle t = new Triangle(p);
        Circle   c = new Circle(new Point(150, 150), 50);

        t.draw(); // 삼각형을 그린다.
        c.draw(); // 원을 그린다.
    }
}
```

부모 생성자 호출

- 자식 클래스의 생성자에서 항상 부모 객체의 생성자를 먼저 호출해야 함

  - **super**(매개변수)는 부모 클래스의 생성자를 호출하는 방식

    - 부모 클래스에 기본 생성자(매개변수가 없는 생성자)가 있을 때에는 부모 클래스 생성자의 호출을 생략해도 됨 (생략할 경우, 부모 클래스의 기본 생성자가 호출됨)

```
class ParentClass {
    ParentClass(매개값, ...) {
        ...
    }
}

class ChildClass extends ParentClass {
    ChildClass(매개변수){
        super(매개값, ...);  // 부모 클래스의 생성자를 첫 줄에서 호출해야 함
        .....
    }
}
```

실습

[Ex3-1] People.java

```java
package chap07.inheritance;

public class People {
    public String name;
    public String ssn;

    public People(String name, String ssn) {
        this.name = name;
        this.ssn = ssn;
    }
}
```

[Ex3-2] Student.java

```java
package chap07.inheritance;

public class Student extends People{
    public int studentNo;

    public Student(String name, String ssn, int studentNo) {
        super(name, ssn);
        this.studentNo = studentNo;
    }
}
```

[Ex3-3] StudentExample.java

```java
package chap07.inheritance;

public class StudentExample {
    public static void main(String[] args) {
        Student student = new Student("홍길동", "123456-1234567", 1);
        System.out.println("name : " + student.name);
        System.out.println("ssn : " + student.ssn);
        System.out.println("studentNo : " + student.studentNo);
    }
}
```

오버라이딩(Overriding)

- 부모 클래스로부터 상속받은 메소드를 재정의(변경)하여 사용하는 것

  - 부모의 메소드와 동일한 메소드 정의 형식을 가져야 함
    - 즉, 부모 메소드와 메소드 이름, 리턴 타입, 매개변수 등 모든 정의가 같아야 함
  - 부모 클래스의 메서드보다 좁은 범위로 접근 제한을 변경할 수 없음
  - Exception 선언은 부모 클래스의 Exception 종류 내에서 선언되어야 함
    - 부모 클래스의 Exception 종류 중, 제외는 가능하나, 새로운 Exception을 추가할 수 없음
  - 부모 클래스에 정의된 원본 메소드를 호출할 때는 **super.메소드명( )** 형태로 호출

```
class ParentClass {
    void method1( ){
        .......
    }
    int method2(int a){
        .......
    }
}

class ChildClass extends ParentClass {
    void method1( ){
        .......  // 부모 클래스의 method1의 내용을 재정의
    }
    int method2(int a){
        ....... // 부모 클래스의 method2의 내용을 재정의
        super.method2(a);  // 부모 클래스의 method2 호출
    }
}
```

실습(1)

[Ex4-1] Calculator.java

```java
package chap07.inheritance;

public class Calculator {
    double areaCircle(double r) {
        System.out.println("Calculator 객체의 areaCircle() 실행");
        return 3.14159 * r * r;
    }
}
```

[Ex4-2] Computer.java

```java
package chap07.inheritance;

public class Computer extends Calculator {
    @Override
    double areaCircle(double r) {
    System.out.println("Computer 객체의 areaCircle() 실행");
        return Math.PI * r * r;
    }
}
```

[Ex4-3] ComputerExample.java

```java
package chap07.inheritance;

public class ComputerExample {
    public static void main(String[] args) {
        int r = 10;
        Calculator calculator = new Calculator();
        System.out.println("원면적 : " + calculator.areaCircle(r));
        System.out.println();
        Computer computer = new Computer();
        System.out.println("원면적 : " + computer.areaCircle(r));
    }
}
```

실습(2)

[Ex5-1] Airplane.java

```java
package chap07.inheritance;

public class Airplane {
    public void land() {
        System.out.println("착륙합니다.");
    }
    public void fly() {
        System.out.println("일반비행합니다.");
    }
    public void takeOff() {
        System.out.println("이륙합니다.");
    }
}
```

[Ex5-2] SupersonicAirplane.java

```java
package chap07.inheritance;

public class SupersonicAirplane extends Airplane {
    public static final int NORMAL = 1;
    public static final int SUPERSONIC = 2;

    public int flyMode = NORMAL;

    @Override
    public void fly() {
        if(flyMode == SUPERSONIC) {
            System.out.println("초음속비행합니다.");
        } else {
            //Airplane 객체의 fly() 메소드 호출
            super.fly();
        }
    }
}
```

[Ex5-3] SupersonicAirplaneExample.java

```java
package chap07.inheritance;

public class SupersonicAirplaneExample {
    public static void main(String[] args) {
        SupersonicAirplane sa = new SupersonicAirplane();
        sa.takeOff();
        sa.fly();
        sa.flyMode = SupersonicAirplane.SUPERSONIC;
        sa.fly();
        sa.flyMode = SupersonicAirplane.NORMAL;
        sa.fly();
        sa.land();
    }
}
```

final 클래스와 final 메소드

- final로 정의된 클래스는 상속할 수 없음

```
public final class FinalClass {  // 상속 받을 수 없는 final class
    .......
}

public class MyClass extends FinalClass{ }  // 에러
```

- final 선언된 메소드는 자식 클래스에서 오버라이딩(Overriding) 할 수 없음

[Ex6-1] Car.java

```
package chap07.inheritance;

public class Car {
    //필드
    public int speed;

    //메소드
    public void speedUp() {
        speed += 1;
    }

    //final 메소드
    public final void stop() {
        System.out.println("차를 멈춤");
        speed = 0;
    }
}
```

[Ex6-2] SportsCar.java

```
package chap07.inheritance;

public class SportsCar extends Car {
    @Override
    public void speedUp() {
        speed += 10;
    }

    //오버라이딩을 할 수 없음
    /*
    @Override
    public void stop() {
        System.out.println("스포츠카를 멈춤");
        speed = 0;
    }
    */
}
```

다형성(Polymorphism)

- 런타임 시, 하나의 부모 클래스 타입에 다양한 자식 클래스 객체 타입을 대입할 수 있도록 함으로써 유연성을 제공하는 기능(동적 바인딩)

  - 자식 클래스를 부모 클래스에 대입할 때는 자동 형변환(자식 클래스 → 부모 클래스)

    - 자식 클래스를 부모 클래스로 형변환을 한 경우, 자식 클래스 메소드 중, 오버라이딩 메소드 이외에는 호출할 수 없음

```
class ParentClass {
    void method1( ){ … }
    void method2( ){ … }
}

class ChildClass extends ParentClass {
    void method2( ){ … }  // 오버라이딩
    void method3( ){ … }
}


class MainClass{
    ChildClass c = new ChildClass( );
    ParentClass p = c;

    ParentClass p1 = new ChildClass( );

    p.method1( );
    p.method2( );  // 오버라이딩된 ChildClass의 메소드가 호출됨
    p.method3( );  // 에러
```

* 자식 클래스에 부모 클래스를 대입하는 경우는 강제 형변환 필요

  (부모클래스 → 자식 클래스)

  - 이 때, 부모 클래스의 참조변수는 자식 클래스의 인스턴스로 생성되어야 함

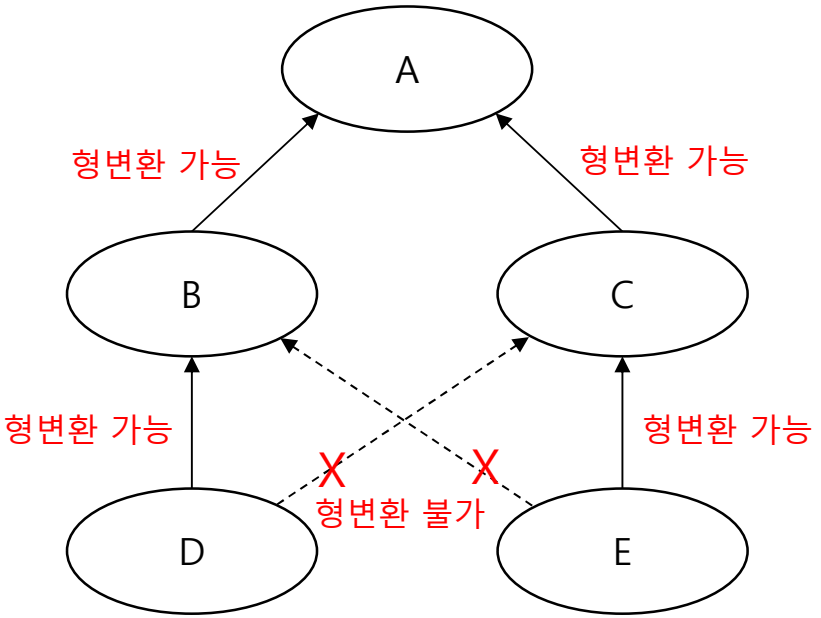
```
class MainClass{
    // 자식클래스의 인스턴스로 생성
    ParentClass p = new ChildClass( );

    ChildClass c = (Child) p;
}
```

ParentClass p = new ParentClass( ); 로 선언하면 안 됨

실습(1)

[Ex7] PolymorphismEX.java

```
package chap07.inheritance;

class A {}

class B extends A{}
class C extends A{}

class D extends B{}
class E extends C{}

public class PolymorphismEX {
    public static void main(String[] args) {
        B b = new B();
        C c = new C();
        D d = new D();
        E e = new E();

        A a1 = b;
        A a2 = c;
        A a3 = d;
        A a4 = e;

        B b1 = d;
        C c1 = e;

        //B b2 = e;  // 에러(상속관계가 아니므로)
        //C c2 = d;  // 에러(상속관계가 아니므로)
    }
}
```



실습(2)

[Ex8-1] Parent.java

```java
package chap07.inheritance;

public class Parent {
    public void method1() {
        System.out.println("Parent-method1()");
    }

    public void method2() {
        System.out.println("Parent-method2()");
    }
}
```

[Ex8-2] Child.java

```java
package chap07.inheritance;

public class Child extends Parent {
    @Override
    public void method2() {
        System.out.println("Child-method2()");
    }

    public void method3() {
        System.out.println("Child-method3()");
    }
}
```

[Ex8-3] MainClass.java

```java
package chap07.inheritance;

public class MainClass {
    public static void main(String[] args) {
        Child child = new Child();

        Parent parent = child;

        parent.method1();
        parent.method2(); // 오버라이딩된 Child의 method2() 호출
        //parent.method3();  (호출 불가능)
    }
}
```

실습(3)

[Ex9] CastingTest1.java

```
package chap07.inheritance2;

class CastingTest1 {
    public static void main(String args[]) {
        Car car = null;
        FireEngine fe = new FireEngine();
        FireEngine fe2 = null;

        fe.water();
        car = fe;    // 부코 클래스 타입으로 자동형변환
//      car.water();    // 에러! 호출 불가
        fe2 = (FireEngine)car; // 명시적 형변환(자손타입으로)
        fe2.water();
        fe = (FireEngine)car;
        fe.water();
    }
}

class Car {
    String color;
    int door;

    void drive() {
        System.out.println("drive, Brrrr~");
    }

    void stop() {
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {
    void water() {
        System.out.println("water!!!");
    }
}
```

실습(4)

[Ex10] CastingTest2.java

```
package chap07.inheritance2;

class CastingTest2 {
    public static void main(String args[]) {
        Car2 car = new Car2();
        Car2 car2 = null;
        FireEngine2 fe = null;

        car.drive();
        fe = (FireEngine2)car;      // 실행 시, 에러 발생
        fe.drive();
        car2 = fe;
        car2.drive();
    }
}

class Car2 {
    String color;
    int door;

    void drive() {
        System.out.println("drive, Brrrr~");
    }

    void stop() {
        System.out.println("stop!!!");
    }
}

class FireEngine2 extends Car2 {
    void water() {
        System.out.println("water!!!");
    }
}
```

실습(5)

[Ex11-1] Parent.java

```java
package chap07.inheritance3;

public class Parent {
    public String field1;

    public void method1() {
        System.out.println("Parent-method1()");
    }

    public void method2() {
        System.out.println("Parent-method2()");
    }
}
```

[Ex11-2] Child.java

```java
package chap07.inheritance3;

public class Child extends Parent {
    public String field2;

    public void method3() {
        System.out.println("Child-method3()");
    }
}
```

[Ex11-3] MainClass.java

```java
package chap07.inheritance3;

public class MainClass {
    public static void main(String[] args) {
        Parent parent = new Child();
        parent.field1 = "data1";
        parent.method1();
        parent.method2();
        /*
        parent.field2 = "data2";  //(불가능)
        parent.method3();         //(불가능)
        */

        Child child = (Child) parent;
        child.field2 = "yyy";  //(가능)
        child.method3();     //(가능)
    }
}
```

실습(6)

[Ex12-1] Parent.java

```java
package chap07.inheritance4;

public class Parent {
}
```

[Ex12-2] Child.java

```java
package chap07.inheritance4;

public class Child extends Parent {
}
```

[Ex12-3] MainClass.java

```java
package chap07.inheritance4;

public class InstanceofExample {
    public static void method1(Parent parent) {
        if(parent instanceof Child) {
            Child child = (Child) parent;
            System.out.println("method1 - Child로 변환 성공");
        } else {
            System.out.println("method1 - Child로 변환되지 않음");
        }
    }

    public static void method2(Parent parent) {
        Child child = (Child) parent;
        System.out.println("method2 - Child로 변환 성공");
    }

    public static void main(String[] args) {
        Parent parentA = new Child();
        method1(parentA);
        method2(parentA);

        Parent parentB = new Parent();
        method1(parentB);
        method2(parentB); //예외 발생
    }
}
```

* instanceof 연산자

- 참조변수가 참조하고 있는 인스턴스의 실제 타입을 알아보기 위해 사용하는 연산자

실습(7)

[Ex13-1] Tire.java

```
package chap07.inheritance;

public class Tire {
    public int maxRotation;                 //최대 회전수(최대 수명)
    public int accumulatedRotation;     //누적 회전수
    public String location;                     //타이어의 위치

    public Tire(String location, int maxRotation) {
        this.location = location;
        this.maxRotation = maxRotation;
    }

    public boolean roll() {
        ++accumulatedRotation;
        if(accumulatedRotation<maxRotation) {
            System.out.println(location + " Tire 수명: " + (maxRotation-accumulatedRotation) + "회");
            return true;
        } else {
            System.out.println("*** " + location + " Tire 펑크 ***");
            return false;
        }
    }
}
```

실습(8)

[Ex13-2] Car.java

```
package chap07.inheritance;

public class Car {
    Tire frontLeftTire = new Tire("앞왼쪽", 6);
    Tire frontRightTire = new Tire("앞오른쪽", 2);
    Tire backLeftTire = new Tire("뒤왼쪽", 3);
    Tire backRightTire = new Tire("뒤오른쪽", 4);

    int run() {
        System.out.println("[자동차가 달립니다.]");
        if(frontLeftTire.roll()==false) { stop(); return 1; };
        if(frontRightTire.roll()==false) { stop(); return 2; };
        if(backLeftTire.roll()==false) { stop(); return 3; };
        if(backRightTire.roll()==false) { stop(); return 4; };
        return 0;
    }

    void stop() {
        System.out.println("[자동차가 멈춥니다.]");
    }
}
```

실습(9)

[Ex13-3] HankookTire.java

```
package chap07.inheritance;

public class HankookTire extends Tire {
    //필드
    //생성자
    public HankookTire(String location, int maxRotation) {
        super(location, maxRotation);
    }
    //메소드
    @Override
    public boolean roll() {
        ++accumulatedRotation;
        if(accumulatedRotation<maxRotation) {
            System.out.println(location + " HankookTire 수명: " + (maxRotation-accumulatedRotation) + "회");
            return true;
        } else {
            System.out.println("*** " + location + " HankookTire 펑크 ***");
            return false;
        }
    }
}
```

실습(10)

[Ex13-4] KumhoTire.java

```
package chap07.inheritance;

public class KumhoTire extends Tire {
    //필드
    //생성자
    public KumhoTire(String location, int maxRotation) {
        super(location, maxRotation);
    }
    //메소드
    @Override
    public boolean roll() {
        ++accumulatedRotation;
        if(accumulatedRotation<maxRotation) {
            System.out.println(location + " KumhoTire 수명: " + (maxRotation-accumulatedRotation) + "회");
            return true;
        } else {
            System.out.println("*** " + location + " KumhoTire 펑크 ***");
            return false;
        }
    }
}
```

실습(11)

[Ex13-5] CarMain.java

```
package chap07.inheritance;

public class CarMain {
    public static void main(String[] args) {
        Car car = new Car();

        for(int i=1; i<=5; i++) {
            int problemLocation = car.run();
            switch(problemLocation) {
                case 1:
                    System.out.println("앞왼쪽 HankookTire로 교체");
                    car.frontLeftTire = new HankookTire("앞왼쪽", 15);
                    break;
                case 2:
                    System.out.println("앞오른쪽 KumhoTire로 교체");
                    car.frontRightTire = new KumhoTire("앞오른쪽", 13);
                    break;
                case 3:
                    System.out.println("뒤왼쪽 HankookTire로 교체");
                    car.backLeftTire = new HankookTire("뒤왼쪽", 14);
                    break;
                case 4:
                    System.out.println("뒤오른쪽 KumhoTire로 교체");
                    car.backRightTire = new KumhoTire("뒤오른쪽", 17);
                    break;
            }
            System.out.println("----------------------------------------");
        }
    }
}
```

매개변수의 다형성

- 메소드 호출 시, 매개변수에 다형성을 적용함으로써 호출되는 시점에 매개변수의 동적 바인딩 지원
  - 매개변수에 부모 클래스 타입으로 지정하고 런타임에 매개변수로 자식 클래스 인스턴스 전달

```
class ParentClass {
    void method( ){
        .....
    }
}

class ChildClass1 extends ParentClass {
    void method( ){
        .....
    }
}

class ChildClass2 extends ParentClass {
 void method( ){
        .....
    }
}

class MainClass{
    void method_main(ParentClass p){
        p.method( );
    }
}
```

```
ChildClass1 c1 = new ChildClass1( );
MainClass m = new MainClass( );
m.method_main(c);
```

ChildClass1의 method()가 실행됨

실습

[Ex14-1] Vehicle.java

```java
package chap07.inheritance3;

public class Vehicle {
    public void run() {
        System.out.println("차량이 달립니다.");
    }
}
```

[Ex14-2] Bus.java

```java
package chap07.inheritance3;

public class Bus extends Vehicle {
    @Override
    public void run() {
        System.out.println("버스가 달립니다.");
    }
}
```

[Ex14-5] DriverExample.java

```java
package chap07.inheritance3;

public class DriverExample {
    public static void main(String[] args) {
        Driver driver = new Driver();

        Bus bus = new Bus();
        Taxi taxi = new Taxi();

        driver.drive(bus);
        driver.drive(taxi);
    }
}
```

[Ex14-3] Taxi.java

```java
package chap07.inheritance3;

public class Taxi extends Vehicle {
    @Override
    public void run() {
        System.out.println("택시가 달립니다.");
    }
}
```

[Ex14-4] Driver.java

```java
package chap07.inheritance3;

public class Driver {
    public void drive(Vehicle vehicle) {
        vehicle.run();
    }
}
```

추상 클래스

- 자식 클래스들이 공통적으로 가져야 할 멤버변수와 메소드들을 정의해 놓은 클래스

  - 추상 클래스 자체로 인스턴스 생성을 할 수 없으며, 단지 상속만 가능함

  - 클래스들의 표준적인 설계 규격을 정의해 놓는 용도 사용하며, abstract를 통해 선언

  - 인스턴스 생성을 할 수 없지만, 멤버변수, 생성자, 메소드 등을 가질 수 있음

  - 자식 클래스에서 꼭 구현해야 하는 메소드를 추상 메소드로 선언

    - 자식 클래스에서 해당 추상 메소드를 무조건 구현해야 함

- 일반 클래스와 마찬가지로 다형성(Polymorphism) 지원

```
abstract class AbstractClass {
    int val;  // 멤버변수

    AbstractClass(매개값, ...) {  // 생성자
        ...
    }
    void method(매개값, ...) {  // 메소드
        ...
    }
    abstract void method( );  // 추상 메소드 선언
}

class ChildClass extends AbstractClass {  // 추상 클래스 상속
    .....
    abstract void method( ){  // 추상 메소드에 대한 구현
        ...
    }
}
```

실습(1)

[Ex15-1] Phone.java

```java
package chap07.inheritance4;

public abstract class Phone {
    //필드
    public String owner;

    //생성자
    public Phone(String owner) {
        this.owner = owner;
    }

    //메소드
    public void turnOn() {
        System.out.println("폰 전원을 켭니다.");
    }
    public void turnOff() {
        System.out.println("폰 전원을 끕니다.");
    }
}
```

[Ex15-2] SmartPhone.java

```java
package chap07.inheritance4;

public class SmartPhone extends Phone {
    //생성자
    public SmartPhone(String owner) {
        super(owner);
    }
    //메소드
    public void internetSearch() {
        System.out.println("인터넷 검색을 합니다.");
    }
}
```

[Ex15-3] PhoneExample.java

```java
package chap07.inheritance4;

public class PhoneExample {
    public static void main(String[] args) {
        //Phone phone = new Phone(); (x)

        SmartPhone smartPhone = new SmartPhone("홍길동");

        smartPhone.turnOn();
        smartPhone.internetSearch();
        smartPhone.turnOff();
    }
}
```

실습(2)

[Ex16-1] Animal.java

```java
package chap07.inheritance4;

public abstract class Animal {
    public String kind;

    public void breathe() {
        System.out.println("숨을 쉽니다.");
    }
    public abstract void sound();
}
```

[Ex16-2] Cat.java

```java
package chap07.inheritance4;

public class Cat extends Animal {
    public Cat() {
        this.kind = "포유류";
    }
    public void sound() {
        System.out.println("야옹");
    }
}
```

[Ex16-3] dog.java

```java
package chap07.inheritance4;

public class Dog extends Animal {
    public Dog() {
        this.kind = "포유류";
    }
    public void sound() {
        System.out.println("멍멍");
    }
}
```

[Ex16-4] AnimalExample.java

```java
package chap07.inheritance4;

public class AnimalExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        Cat cat = new Cat();
        dog.sound();
        cat.sound();
        System.out.println("-----");

        //변수의 자동 타입 변환
        Animal animal = null;
        animal = new Dog();
        animal.sound();
        animal = new Cat();
        animal.sound();
        System.out.println("-----");

        //매개변수의 자동 타입 변환
        animalSound(new Dog());
        animalSound(new Cat());
    }

    public static void animalSound(Animal animal) {
        animal.sound();
    }
}
```

- 추상클래스 타입으로 선언 후, 자식 클래스 인스턴스로 생성 가능(다형성)