Chapter 07. 상속

7.1 상속 개념

- 상속(Inheritance) : 부모 클래스의 필드와 메소드를 자식 클래스에게 물려주는 행위
- 이미 잘 개발된 클래스를 재사용해 새로운 클래스를 만드므로 중복되는 코드를 줄여 개발 시간 단축
- 수정을 최소화

7.2 클래스 상속

Phone.java

```
package ch07.sec02;

public class Phone {
    //필드 선언
    public String model;
    public String color;

    //메소드 선언
    public void bell() {
        System.out.println("벨이 울립니다.");
    }

    public void sendVoice(String message) {
        System.out.println("자기: " + message);
    }

    public void receiveVoice(String message) {
        System.out.println("상대방: " + message);
    }

    public void hangUp() {
        System.out.println("전화를 끊습니다.");
    }
}
```

SmartPhone.java

```
package ch07.sec02;

public class SmartPhone extends Phone {
    //필드 선언
    public boolean wifi;

    //생성자 선언
    public SmartPhone(String model, String color) {
        this.model = model;
```

```
        this.color = color;
    }

    //메소드 선언
    public void setWifi(boolean wifi) {
        this.wifi = wifi;
        System.out.println("와이파이 상태를 변경했습니다.");
    }

    public void internet() {
        System.out.println("인터넷에 연결합니다.");
    }
}
```

SmartPhoneExample.java

```
package ch07.sec02;

public class SmartPhoneExample {

    public static void main(String[] args) {
        //SmartPhone 객체 생성
        SmartPhone myPhone = new SmartPhone("갤럭시", "은색");

        //Phone으로부터 상속받은 필드 읽기
        System.out.println("모델: " + myPhone.model);
        System.out.println("색상: " + myPhone.color);

        //SmartPhone의 필드 읽기
        System.out.println("와이파이 상태: " + myPhone.wifi);

        //Phone으로부터 상속받은 메소드 호출
        myPhone.bell();
        myPhone.sendVoice("여보세요.");
        myPhone.receiveVoice("안녕하세요! 저는 홍길동인데요.");
        myPhone.sendVoice("아~ 네, 반갑습니다.");
        myPhone.hangUp();

        //SmartPhone의 메소드 호출
        myPhone.setWifi(true);
        myPhone.internet();
    }
}
```

7.3 부모 생성자 호출

- 부모 클래스가 기본 생성자를 가지고 있는 경우

Phone.java

```java
package ch07.sec03.exam01;

public class Phone {
    //필드 선언
    public String model;
    public String color;

    //기본 생성자 선언
    public Phone() {
        System.out.println("Phone() 생성자 실행");
    }
}
```

SmartPhone.java

```java
package ch07.sec03.exam01;

public class SmartPhone extends Phone {
    //자식 생성자 선언
    public SmartPhone(String model, String color) {
        super();
        this.model = model;
        this.color = color;
        System.out.println("SmartPhone(String model, String color) 생성자 실행됨");
    }
}
```

SmartPhoneExample.java

```java
package ch07.sec03.exam01;

public class SmartPhoneExample {

    public static void main(String[] args) {
        //SmartPhone 객체 생성
        SmartPhone myPhone = new SmartPhone("갤럭시", "은색");

        //Phone으로부터 상속 받은 필드 읽기
        System.out.println("모델: " + myPhone.model);
        System.out.println("색상: " + myPhone.color);
    }
}
```

결과

```
Phone() 생성자 실행
SmartPhone(String model, String color) 생성자 실행됨
모델: 갤럭시
색상: 은색
```

- 부모 클래스가 매개변수를 갖는 생성자가 있는 경우

Phone.java

```java
package ch07.sec03.exam02;

public class Phone {
    //필드 선언
    public String model;
    public String color;

    //매개변수를 갖는 생성자 선언
    public Phone(String model, String color) {
        this.model = model;
        this.color = color;
        System.out.println("Phone(String model, String color) 생성자 실행");
    }
}
```

SmartPhone.java

```java
package ch07.sec03.exam02;

public class SmartPhone extends Phone {
    //자식 생성자 선언
    public SmartPhone(String model, String color) {
        super(model, color);        // 반드시 작성
        System.out.println("SmartPhone(String model, String color) 생성자 실행됨");
    }
}
```

SmartPhoneExample.java

```java
package ch07.sec03.exam02;

public class SmartPhoneExample {

    public static void main(String[] args) {
        //SmartPhone 객체 생성
        SmartPhone myPhone = new SmartPhone("갤럭시", "은색");
```

```
        //Phone으로부터 상속 받은 필드 읽기
        System.out.println("모델: " + myPhone.model);
        System.out.println("색상: " + myPhone.color);
    }
}
```

결과

```
Phone(String model, String color) 생성자 실행
SmartPhone(String model, String color) 생성자 실행됨
모델: 갤럭시
색상: 은색
```

7.4 메소드 재정의

- 메소드 오버라이딩(Overriding)
    - 상속된 메소드를 자식 클래스에서 재정의
    - 해당 부모 메소드는 숨겨지고, 자식 메소드가 우선적으로 사용
- 규칙
    - 부모 메소드의 선언부와 동일해야 한다.
    - 접근 제한을 더 강하게 오버라이딩할 수 없다.
    - 새로운 예외를 throws할 수 없다.
- `super.method()`
    - 부모 메소드를 재사용
    - 자식 메소드의 중복 작업 내용을 없애는 효과를 가져온다

Airplane.java

```
package ch07.sec04.exam02;

public class Airplane {
    //메소드 선언
    public void land() {
        System.out.println("착륙합니다.");
}

    public void fly() {
        System.out.println("일반 비행합니다.");
    }

    public void takeOff() {
        System.out.println("이륙합니다.");
```

```
    }
}
```

SupersonicAirplane.java

```
package ch07.sec04.exam02;

public class SupersonicAirplane extends Airplane {
    //상수 선언
    public static final int NORMAL = 1;
    public static final int SUPERSONIC = 2;
    //상태 필드 선언
    public int flyMode = NORMAL;

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

SupersonicAirplaneExample.java

```
package ch07.sec04.exam02;

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();
    }
}
```

결과

```
이륙합니다.
일반 비행합니다.
```

```
초음속 비행합니다.
일반 비행합니다.
착륙합니다.
```

7.5 final 클래스와 final 메소드

- `final` 클래스
  - 최종적인 클래스이므로 더 이상 상속할 수 없는 클래스
  - 부모 클래스가 될 수 없어 자식 클래스를 만들 수 없다.
  - 대표적인 예 : `String` 클래스
- `final` 메소드
  - 최종적인 메소드이므로 오버라이딩할 수 없는 메소드
  - 자식 클래스에서 재정의할 수 없다.

7.6 protected 접근 제한자

- 같은 패키지에서는 `default`처럼 접근이 가능하나, 다른 패키지에서는 자식 클래스만 접근을 허용
- 필드, 생성자 그리고 메소드 선언에 사용 가능

A.java

```java
package ch07.sec06.package1;

public class A {
    //필드 선언
    protected String field;

    //생성자 선언
    protected A() {
    }

    //메소드 선언
    protected void method() {
    }
}
```

B.java

```java
package ch07.sec06.package1;
// 같은 패키지 접근 가능

public class B {
    //메소드 선언
    public void method() {
        A a = new A();          //o
        a.field = "value";      //o
```

```
        a.method();             //o
    }
}
```

C.java

```
package ch07.sec06.package2;
// 다른 패키지 접근 불가능

import ch07.sec06.package1.A;

public class C {
    //메소드 선언
    public void method() {
        //A a = new A();            //x
        //a.field = "value";        //x
        //a.method();               //x
    }
}
```

D.java

```
package ch07.sec06.package2;
// 다른 패키지, 상속을 통해서만 사용 가능

import ch07.sec06.package1.A;

public class D extends A {
    //생성자 선언
    public D() {
        //A() 생성자 호출
        super();                //o
    }

    //메소드 선언
    public void method1() {
        //A 필드값 변경
        this.field = "value";   //o
        //A 메소드 호출
        this.method();          //o
    }

    //메소드 선언
    public void method2() {
        //A a = new A();        //x
        //a.field = "value";    //x
        //a.method();           //x

        // 직접 객체 생성해서 사용하는 것은 안됨
```

```
    }
}
```

7.7 타입 변환

자동 타입 변환 (Promotion)

- 부모타입 변수 = 자식타입객체;
- 부모 타입으로 자동 타입 변환된 이후에는 부모 클래스에 선언된 필드와 메소드만 접근 가능
- 그러나 자식 클래스에서 오버라이딩된 메소드가 있다면 부모 메소드 대신 오버라이딩된 메소드가 호출

Parent.java

```
package ch07.sec07.exam02;

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

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

Child.java

```
package ch07.sec07.exam02;

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

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

ChildExample.java

```java
package ch07.sec07.exam02;

public class ChildExample {
    public static void main(String[] args) {
        //자식 객체 생성
        Child child = new Child();

        //자동 타입 변환
        Parent parent = child;

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

결과

```
Parent-method1()
Child-method2()
```

강제 타입 변환 (Casting)

- 자식타입 변수 = (자식타입) 부모타입객체;
- 자식 객체가 부모 타입으로 자동 변환된 후 다시 자식 타입으로 변환할 때 강제 타입 변환 사용 가능

Parent.java

```java
package ch07.sec07.exam03;

public class Parent {
    //필드 선언
    public String field1;

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

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

Child.java

```java
package ch07.sec07.exam03;

public class Child extends Parent {
    //필드 선언
    public String field2;

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

ChildExample.java

```java
package ch07.sec07.exam03;

public class ChildExample {
    public static void main(String[] args) {
        //객체 생성 및 자동 타입 변환
        Parent parent = new Child();

        //Parent 타입으로 필드와 메소드 사용
        parent.field1 = "data1";
        parent.method1();
        parent.method2();
        /*
        parent.field2 = "data2";        //(불가능)
        parent.method3();               //(불가능)
        */

        //강제 타입 변환
        Child child = (Child) parent;

        //Child 타입으로 필드와 메소드 사용
        child.field2 = "data2";         //(가능)
        child.method3();                //(가능)
    }
}
```

결과

```
Parent-method1()
Parent-method2()
Child-method3()
```

7.8 다형성

- 사용 방법은 동일하지만 실행 결과가 다르게 나오는 성질
- **자동 타입 변환**과 **메소드 재정의**가 필요

Tire.java

```
package ch07.sec08.exam01;

public class Tire {
    //메소드 선언
    public void roll() {
        System.out.println("회전합니다.");
    }
}
```

HankookTire.java

```
package ch07.sec08.exam01;

public class HankookTire extends Tire {
    //메소드 재정의(오버라이딩)
    @Override
    public void roll() {
        System.out.println("한국 타이어가 회전합니다.");
    }
}
```

KumhoTire.java

```
package ch07.sec08.exam01;

public class KumhoTire extends Tire {
    //메소드 재정의(오버라이딩)
    @Override
    public void roll() {
        System.out.println("금호 타이어가 회전합니다.");
    }
}
```

Car.java

```
package ch07.sec08.exam01;

public class Car {
    //필드 선언
    public Tire tire;
```

```
    //메소드 선언
    public void run() {
        //tire 필드에 대입된 객체의 roll() 메소드 호출
        tire.roll();
    }
}
```

CarExample.java

```
package ch07.sec08.exam01;

public class CarExample {
    public static void main(String[] args) {
        //Car 객체 생성
        Car myCar = new Car();

        //Tire 객체 장착
        myCar.tire = new Tire();
        myCar.run();

        //HankookTire 객체 장착
        myCar.tire = new HankookTire();
        myCar.run();

        //KumhoTire 객체 장착
        myCar.tire = new KumhoTire();
        myCar.run();
    }
}
```

결과

```
회전합니다.
한국 타이어가 회전합니다.
금호 타이어가 회전합니다.
```

7.9 객체 타입 확인

- 변수가 참조하는 객체의 타입을 확인하고자 할 때 `instanceof` 연산자 사용

InstanceofExample

```
package ch07.sec09;

public class InstanceofExample {
    //main() 메소드에서 바로 호출하기 위해 정적 메소드 선언
```

```
    public static void personInfo(Person person) {
        System.out.println("name: " + person.name);
        person.walk();

        //person이 참조하는 객체가 Student 타입인지 확인
        /*if (person instanceof Student) {
            //Student 객체일 경우 강제 타입 변환
            Student student = (Student) person;
            //Student 객체만 가지고 있는 필드 및 메소드 사용
            System.out.println("studentNo: " + student.studentNo);
            student.study();
        }*/

        //person이 참조하는 객체가 Student 타입일 경우
        //student 변수에 대입(타입 변환 발생)
        //Java 12부터 가능
        if(person instanceof Student student) {
            System.out.println("studentNo: " + student.studentNo);
            student.study();
        }
    }

    public static void main(String[] args) {
        //Person 객체를 매개값으로 제공하고 personInfo() 메소드 호출
        Person p1 = new Person("홍길동");
        personInfo(p1);

        System.out.println();

        //Student 객체를 매개값으로 제공하고 personInfo() 메소드 호출
        Person p2 = new Student("김길동", 10);
        personInfo(p2);
    }
}
```

Person.java

```
package ch07.sec09;

public class Person {
    //필드 선언
    public String name;

    //생성자 선언
    public Person(String name) {
        this.name = name;
    }

    //메소드 선언
    public void walk() {
        System.out.println("걷습니다.");
```

```
    }
}
```

Student.java

```
package ch07.sec09;

public class Student extends Person {
    //필드 선언
    public int studentNo;

    //생성자 선언
    public Student(String name, int studentNo) {
        super(name);
        this.studentNo = studentNo;
    }

    //메소드 선언
    public void study() {
        System.out.println("공부를 합니다.");
    }
}
```

결과

```
name: 홍길동
걷습니다.

name: 김길동
걷습니다.
studentNo: 10
공부를 합니다.
```

7.10 추상 클래스

- 클래스들의 공통적인 필드나 메소드를 추출해서 선언한 클래스
- 실체 클래스의 부모 역할
- `new` 연산자를 사용해서 객체 직접 생성할 수 없음.
- 새로운 실체 클래스를 만들기 위한 부모 클래스로만 사용

추상 메소드와 재정의

- `abstract` 키워드가 붙고 메소드 실행 내용인 중괄호 `{}`가 없음.
- 자식 클래스의 공통 메소드라는 것만 정의할 뿐 실행 내용 가지지 않음.
- 자식 클래스에서 반드시 재정의(오버라이딩)해서 실행 내용 채워야 함.

Animal.java

```
package ch07.sec10.exam02;

public abstract class Animal {
    //메소드 선언
    public void breathe() {
        System.out.println("숨을 쉽니다.");
    }

    //추상 메소드 선언
    public abstract void sound();
}
```

Dog.java

```
package ch07.sec10.exam02;

public class Dog extends Animal {
    //추상 메소드 재정의
    @Override
    public void sound() {
        System.out.println("멍멍");
    }
}
```

Cat.java

```
package ch07.sec10.exam02;

public class Cat extends Animal {
    //추상 메소드 재정의
    @Override
    public void sound() {
        System.out.println("야옹");
    }
}
```

AbstractMethodExample.java

```
package ch07.sec10.exam02;

public class AbstractMethodExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound();
```

```
        Cat cat = new Cat();
        cat.sound();

        //매개변수의 다형성
        animalSound(new Dog());
        animalSound(new Cat());
    }

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

실행결과

```
멍멍
야옹
멍멍
야옹
```

7.11 봉인된 클래스

- 무분별한 자식 클래스 생성 방지 위해 Java 15부터 도입
- `sealed` 키워드를 사용하면 `permits` 키워드 뒤에 지정된 클래스만 상속 가능함.
- 봉인된 클래스를 상속하는 자식 클래스는 `final` 또는 `non-sealed` 키워드 선언 또는 `sealed` 키워드 사용해서 또 다른 봉인 클래스로 선언해야 함.

Person.java

```
package ch07.sec11;

public sealed class Person permits Employee, Manager {
    //필드
    public String name;

    //메소드
    public void work() {
        System.out.println("하는 일이 결정되지 않았습니다.");
    }
}
```

Employee.java

```
package ch07.sec11;
```

```
public final class Employee extends Person {
    @Override
    public void work() {
        System.out.println("제품을 생산합니다.");
    }
}
```

Manager.java

```
package ch07.sec11;

public non-sealed class Manager extends Person {
    @Override
    public void work() {
        System.out.println("생산 관리를 합니다.");
    }
}
```

Director.java

```
package ch07.sec11;

public class Director extends Manager {
    @Override
    public void work() {
        System.out.println("제품을 기획합니다.");
    }
}
```

SealedExample.java

```
package ch07.sec11;

public class SealedExample {
    public static void main(String[] args) {
        Person p = new Person();
        Employee e = new Employee();
        Manager m = new Manager();
        Director d = new Director();

        p.work();
        e.work();
        m.work();
        d.work();
    }
}
```

실행 결과

```
하는 일이 결정되지 않았습니다.
제품을 생산합니다.
생산 관리를 합니다.
제품을 기획합니다.
```