

Java Study Skill Up Team Review

7.1 Inheritance

- 상속(Inheritance[인헤리턴스])은 프로그램에서도 부모 클래스의 필드와 메서드를 자식 클래스에 물려줄 수 있는 것이다.
- 하나의 java 파일에서 상속 이해를 위한 소스를 작성해 보았다.

```
package com.chapter.ch07.example.ex01;

public class Inheritance {
    public class ParentClass {
        String parentField;

        void parentMethod() {
            System.out.println("parent method");
        }
    }

    public class ChildrenClass extends ParentClass {
        String childrenField;

        void childrenMethod() {
            System.out.println("childrenMethod");
        }
    }

    public static void main(String[] args) {
        ParentClass parentClass = new ParentClass(); // ✕ 오류 발생
    }
}
```

[오류 발생]

- new ParentClass(); 소스에서 **cannot be referenced from a static context** 오류가 발생했다.
 - 해석: 정적 컨텍스트(static context)에서는 참조할 수 없다.
 - 즉 main 메서드가 static 으로 정의되어 있는데 ParentClass 와 ChildrenClass 는 Inheritance 클래스의 내부 클래스로 정의되어 있기 때문이다.
 - 그래서 내부 클래스는 외부 클래스의 인스턴스를 필요로 하며 정적 메서드인 main 에서 직접 내부 클래스를 인스턴스화 할 수 없다.

[오류 분석]

- static 관련 참조 오류는 상속과 관계있는 것은 아니다.
- 자바에서 정적(static) 메서드는 클래스 수준에서 실행되며 클래스의 인스턴스 생성 없이 호출할 수 있다.
- 그러나 내부 클래스(ChildrenClass)의 인스턴스를 생성하려면 먼저 외부 클래스의 인스턴스가 생성되어야 한다.
- 내부 클래스는 외부 클래스의 인스턴스에 종속적이기 때문이다.

- 따라서 외부 클래스의 인스턴스 생성 없이 내부 클래스의 인스턴스를 생성하려고 시도하면 오류가 발생한다.

☞ 결론적으로 내부 클래스의 인스턴스를 생성하려면 먼저 외부 클래스의 인스턴스를 생성해야 한다.

[해결 방법]

- 방법 1
 - ParentClass 와 ChildrenClass 를 static 으로 변경하거나 정적 내부 클래스로 정의한다.

```
package com.chapter.ch07.example.ex01;

public class Inheritance {
    public static class ParentClass {
        String parentField;

        void parentMethod() {
            System.out.println("parent method");
        }
    }

    public static class ChildrenClass extends ParentClass {
        String childrenField;

        void childrenMethod() {
            System.out.println("childrenMethod");
        }
    }

    public static void main(String[] args) {
        ChildrenClass childrenClass = new ChildrenClass(); // ○ 방법 1
        childrenClass.parentMethod();
    }
}
```

- 방법 2
 - main 메서드 내에서 Inheritance 클래스의 인스턴스를 생성한 후 그 인스턴스를 사용해서 내부 클래스를 인스턴스화 한다.

```
package com.chapter.ch07.example.ex01;

public class Inheritance {
    public class ParentClass {
        String parentField;

        void parentMethod() {
            System.out.println("parent method");
        }
    }
}
```

```

public class ChildrenClass extends ParentClass {
    String childrenField;

    void childrenMethod() {
        System.out.println("childrenMethod");
    }
}

public static void main(String[] args) {
    // ParentClass parentClass = new ParentClass(); // ✕ 오류 발생
    Inheritance inheritance = new Inheritance(); // ○ 방법 2
    Inheritance.ChildrenClass childrenClass = inheritance.new ChildrenClass();
    childrenClass.parentMethod();
}
}

```

- 방법 3
- static void 에서 static 을 제거한다.

```

package com.chapter.ch07.example.ex01;

public class Inheritance {
    public class ParentClass {
        String parentField;

        void parentMethod() {
            System.out.println("parent method");
        }
    }

    public class ChildrenClass extends ParentClass {
        String childrenField;

        void childrenMethod() {
            System.out.println("childrenMethod");
        }
    }

    public void main(String[] args) { // ○ 방법 3
        ChildrenClass childrenClass = new ChildrenClass();
        childrenClass.childrenMethod();
    }
}

```

[결론]

- static void 에 대해서 알게되었고 예시가 잘못된 거 같아서 정석 느낌으로 상속에 대한 예제를 다시 작성해 보았다.

다시 돌아가 상속이란?

- 상속(Inheritance[인헤러턴스])은 프로그램에서도 부모 클래스의 필드와 메서드를 자식 클래스에 물려줄 수 있는 것이다.

7.2 Inheritance Class


- 예제로 부모 클래스, 자식 클래스, 실행 클래스를 생성하였다.

 ParentClass.java

```
package com.chapter.ch07.example.ex01;

public class ParentClass {
    String parentClassField;

    public void parentClassMethod() {
        System.out.println("parent class method");
    }
}
```

 ChildrenClass.java

```
package com.chapter.ch07.example.ex01;

public class ChildrenClass extends ParentClass {
    String childrenClassField;

    void childrenClassMethod () {
        System.out.println("children class method");
    }
}
```

 Main.java

```
package com.chapter.ch07.example.ex01;

public class Main {
    public static void main(String[] args) {
        System.out.println("ChildrenClass 클래스를 인스턴스화 하여 childrenClass 인스턴스를 생성한다.");
        ChildrenClass childrenClass = new ChildrenClass();
        System.out.println("childrenClass 객체를 통해 부모 클래스에 정의된 parentClassMethod를 호출한다.");
        childrenClass.parentClassMethod();
    }
}
```

- 이처럼 상속은 간단하게 말하자면 이미 잘 개발된 클래스를 재사용 할 수 있다.

- 그렇기에 중복 코드를 줄여 개발 시간을 단축시켜 준다.
- 또 추가 작성해야 하는 필드와 메서드가 상속받은 클래스에 존재한다면 추가 작성할 필요가 없어 효율적이다.
- 부모 클래스만 수정하면 상속 받은 자식 클래스에 내용이 반영되기 때문에 클래스의 수정을 최소화 할 수 있다.

☞ 여러 개의 부모 클래스를 상속 받을 수 있나요?

✕ 정답은 '없다'이다.

- 자바에서는 다중 상속을 허용하지 않는다. (다중 상속을 허용하는 언어는 존재)
- 그 이유는 다중 상속을 받은 부모 클래스에서 동일한 명칭의 메서드가 존재할 수 있기 때문이다.
- 이 경우 어떤 부모 클래스에 접근할지 모호해지기 때문에 다중 상속을 허용하지 않는다.

[상속에 대한 호출]

☹ 상속 받아 ChildrenClass 에서 ParentClass 메서드를 호출은 하였는데 어떻게 호출되는 걸까?

💡해답은 클래스의 생성자에 있었다. 우선 생성자에 대해서 먼저 알아보자.

[생성자]

- Main 소스 코드에서 ChildrenClass childrenClass = new ChildrenClass(); 문법을 작성했다.
 - 이는 'ChildrenClass(클래스)를 인스턴스화 하여 childrenClass(객체[인스턴스])를 생성한다.
 - 생성된 변수(객체[인스턴스])는 new ChildrenClass(); 코드를 통해 생성된 클래스의 인스턴스, 즉 ChildrenClass 를 참조한다.'는 말이다.
 - 그럼 이 객체(인스턴스 || 변수) 는 클래스에서 정의한 필드와 메서드를 사용할 수 있게 되는 것이다.
- 이처럼 생성되는 객체는 생성자를 호출해야 생성이 된다.
- 이 생성자는 ChildrenClass 를 예를 들면 ChildrenClass 클래스 내에 public ChildrenClass() {} 코드로 정의되어 있다.
- 하지만 나는 생성자를 정의해 준 적이 없는데 어떻게 동작이 되는 걸까?
- 그 이유는 생성자가 매개변수를 필요로 하지 않은 상태에서는 직접 추가해 주지 않더라도 숨겨진 상태로 컴파일러가 기본 생성자를 자동으로 추가하기 때문이다.
- 여기까지는 이해가 되었다.

☹ 그런데.. childrenClass 객체(인스턴스)는 컴파일러로 기본 생성자가 자동 추가되어 생성할 수 있었다.

☹ 그럼.. parentClassMethod 메서드가 아닌 childrenClassMethod 만 사용할 수 있는 거 아닌가? 부모 생성자 또한 자동으로 생성되었다 하더라도 자식 클래스에서는 호출한 적이 없는데?

💡 이 해답은 '자식 생성자'에 있었다.

7.3 Parent Constructor Call

[부모 생성자 호출]

- 부모 생성자는 자식 생성자의 제일 첫 줄에 숨겨져 있는 `super()`에 의해 호출된 것이다. 그런데 `super()` 또한 내가 추가해 준 적이 없다.
 - 역시나 `super()` 또한 매개변수를 요구하지 않는 부모 클래스의 경우 자식 생성자의 첫줄에 컴파일러가 `super()` 를 자동으로 추가하고 있었다.
 - 그렇게 `super()` 에 의해 부모 생성자가 호출됨으로써 부모 클래스의 필드와 메서드 또한 초기화 하고 사용할 수 있게 된 것이다.

[숨겨져 있는 생성자와 `super` 모습]

 ParentClass.java

```
package com.chapter.ch07.example.ex01;

public class ParentClass {
    String parentClassField;

    // 부모 기본 생성자 선언 (생략할 경우 컴파일 시 컴파일러가 자동 추가)
    public ParentClass() {
        System.out.println("ParentClass 기본 생성자");
    }

    public void parentClassMethod() {
        System.out.println("parent class method");
    }
}
```

 ChildrenClass.java

```
package com.chapter.ch07.example.ex01;

public class ChildrenClass extends ParentClass {
    String childrenClassField;

    // 자식 기본 생성자 선언 (생략할 경우 컴파일 시 컴파일러가 자동 추가)
    public ChildrenClass() {
        super(); // 생략 가능 (생략할 경우 컴파일 시 컴파일러가 자동 추가)
        System.out.println("childrenClass 생성자 선언");
    }

    void childrenClassMethod () {
        System.out.println("children class method");
    }
}
```

- 이처럼 생성자를 호출해 주어야 객체가 생성이 가능하다.
- 결론적으로 코드 추가해 주지 않았음에도 잘 동작했던 이유는 생략 시 컴파일 할 때 컴파일러가 알아서 기본 생성자를 생성해 주었기 때문이다.

❏ 하지만 (부모, 자식) 기본 생성자에 매개 변수가 존재할 경우에는 super(매개변수...)와 같이 일치하도록 맞추어 직접 추가해 주어야 한다.

[매개 변수가 존재하는 기본 생성자]

📄 ParentClass.java

```
package com.chapter.ch07.example.ex01;

public class ParentClass {
    String parentClassField;

    public ParentClass(String parentClassField) {
        this.parentClassField = parentClassField;
    }

    public void parentClassMethod() {
        System.out.println("parent class method");
    }
}
```

📄 ChildrenClass.java

```
package com.chapter.ch07.example.ex01;

public class ChildrenClass {
    String childrenClassField;

    public ChildrenClass(String childrenClassField) {
        super(childrenClassField);    // 반드시 작성해야 한다.
        this.childrenClassField = childrenClassField;
    }

    public void childrenClassMethod() {
        System.out.println("children class method");
    }
}
```

- 즉 이처럼 매개변수를 가지는 경우에는 생략해도 되는 super(); 가 아닌 인자 개수를 맞춰 주어야 한다는 것이다.

📄 Main.java

```
package com.chapter.ch07.example.ex01;

public class Main {
    public static void main(String[] args) {
        ChildrenClass childrenClass = new ChildrenClass("자식 필드입니다만.");
    }
}
```

```

        childrenClass.parentClassMethod();
    }
}

```

- 출력값 : "자식 필드입니다만."

7.4 Method Overriding

[메서드 재정의]

- 부모 클래스의 모든 메서드가 자식 클래스에 맞도록 설계되었다면 베스트겠지만 모든 자식 클래스에 적합하지 않을 수 있다.
- 이러한 메서드는 자식 클래스에서 정의할 수 있는데 이것을 '오버라이딩'이라고 한다.

[오버라이딩]

 OverridingParent.java

```

package com.chapter.ch07.example.ex02;

public class OverridingParent {
    void hello() {
        System.out.println("안녕");
    }
}

```

- 부모 클래스를 상속 받아 hello 메서드를 호출하는 경우 "안녕"이 출력될 것이다.
- 하지만 인사를 해야 하는 사이트가 미국 사이트라면 영어로 인사해야 한다.
- 이때 메서드 오버라이딩을 통해 부모 클래스의 메서드를 재정의해서 사용할 수 있다.

 OverridingChildren.java

```

package com.chapter.ch07.example.ex02;

public class OverridingChildren extends OverridingParent{
    @Override
    void hello() {
        System.out.println("hello");
    }
}

```

- 이처럼 @Override 어노테이션을 통해 오버라이딩 한 후 출력해 보면

 Main.java


```
package com.chapter.ch07.example.ex02;

public class Main {
    public static void main(String[] args) {
        OverridingChildren children = new OverridingChildren();
        children.hello();
    }
}
```

- "hello"가 출력되는 것을 볼 수 있다.
- 메서드 오버라이딩의 규칙은 다음과 같다.
 - 부모 메서드의 리턴 타입, 메서드 이름, 매개변수는 동일해야 한다.
 - 접근 제한을 높이는 것이 불가능하다. (우측으로 높이는 것 불가능: public < protected < default < private)
 - 새로운 예외 처리 throws 가 불가능하다.

💬 만약 부모 메서드와 오버라이딩 된 자식 메서드를 특정 조건에 따라 사용하고 싶으면 어떻게 할 수 있을까?

📄 OverridingChildren.java

```
package com.chapter.ch07.example.ex02;

public class OverridingChildren extends OverridingParent{
    boolean IsAmericaCheck = false;

    @Override
    void hello() {
        if(IsAmericaCheck == true) {
            System.out.println("hello");
        } else {
            super.hello();
        }
    }
}
```

- 이처럼 만약 조건 로직이 들어가고 IsAmericaCheck 값이 true, false 에 따라 부모 메서드를 혹은 오버라이딩 된 메서드를 사용할 수도 있다.