


Lab 10

<https://padlet.com/dcslabcp/0517-8v3iei1n9ifcvs5n>

오늘 수업 내용

- OOP (Object-Oriented Programming)
 - Inheritance (상속): class, extends, implements
 - Encapsulation (캡슐화): public, protected, (default), private
 - Polymorphism (다형성): overriding, overloading
- 관련 주요 개념들: static, this, super, final, instanceof, toString() 등
- Java 오픈 소스 사례를 통해 살펴보는 OOP 쓰임새  **Jenkins**
 - <https://github.com/jenkinsci/jenkins/tree/master/core/src/main/java/jenkins>

Class 기본 코드

```
class Bicycle {  
    // field of class (State)  
    int gear = 5;  
  
    // method of class (Behavior)  
    void braking() {}  
  
    public static void main(String[] args) {  
        // create object  
        Bicycle sportsBicycle = new Bicycle();  
  
        // access field and method  
        int g = sportsBicycle.gear;  
        sportsBicycle.braking();  
    }  
}
```

- Class는 State와 Behavior로 구성
- 객체를 생성하여 두 영역 접근/활용
- 자바 언어는 설계부터 OOP 고려

Overloading 기본 코드

```
class HelperService {  
    private String formatNumber(int value) {  
        return String.format("%d", value);  
    }  
    private String formatNumber(double value) {  
        return String.format("%.3f", value);  
    }  
    private String formatNumber(String value) {  
        return String.format("%.2f", Double.parseDouble(value));  
    }  
    public static void main(String[] args) {  
        HelperService hs = new HelperService();  
        System.out.println(hs.formatNumber(500));  
        System.out.println(hs.formatNumber(89.9934));  
        System.out.println(hs.formatNumber("550"));  
    }  
}
```

- Overloading: 같은 이름 메소드 여러 개
- Overriding: (Superclass의) 메소드 재정의

```
public void print(long l) {  
  
    Prints a floating-point number. The string produced by  
    bytes according to the platform's default character encoding  
    the manner of the write(int) method.  
  
    Params: f – The float to be printed  
    See Also: Float.toString(float)  
  
    public void print(float f) {  
  
    Prints a double-precision floating-point number. The string  
    is translated into bytes according to the platform's default  
    written in exactly the manner of the write(int) method.  
  
    Params: d – The double to be printed  
    See Also: Double.toString(double)  
  
    public void print(double d) {
```

- 생성자 오버로딩 많이 활용
- Print() 오버로딩 확인 방법!

Overriding 기본 코드

```
class Animal {  
    protected void displayInfo() {  
        System.out.println("I am an animal.");  
    }  
}  
  
class Dog extends Animal {  
    public void displayInfo() {  
        System.out.println("I am a dog.");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Dog d1 = new Dog();  
        d1.displayInfo();  
    }  
}
```

- Overloading: 같은 이름 메소드 여러 개
- Overriding: (Superclass의) 메소드 재정의

Access Modifiers (접근제어자)

* Default: 접근제어자를 명시하지 않는 경우

	Public	Protected	Default*	Private
Class	YES	YES	YES	YES
Package	YES	YES	YES	NO
Subclass	YES	YES	NO	NO
Global	YES	NO	NO	NO

- Class: 동일 클래스 내에 있는 필드/메소드 접근 가능
- Package: 동일 패키지 내에 있는 필드/메소드 접근 가능
- Subclass: Subclass에서 Superclass에 있는 필드/메소드 접근 가능
- Global: 클래스, 패키지, 상속과 무관하게 어디서나 접근 가능

[공통 예제] Rectangle class (Rectangle.java)

```
Rectangle.java x
1  ▶ public class Rectangle{
2  ▶  ▶ public static void main(String[] args) {
3      Rectangle r = new Rectangle();
4      System.out.println(r);
5  }
6  }
```

- 위의 코드를 직접 타이핑해서 작성해 주세요. (선 코드 타이핑, 후 이론 설명)
- 코드를 확장해 가면서 **친절한** Java의 OOP 주요 로직을 연습해봅시다.



2분

[공통 예제] Shape, Rectangle classes (Rectangle.java)

```
Rectangle.java x
1  class Shape {
2      protected int offset_x, offset_y;
3      public Shape() {
4          this.offset_x = 0;
5          this.offset_y = 0;
6      }
7  }
8  public class Rectangle extends Shape {
9      public static void main(String[] args) {
10         Shape s1 = new Shape();
11         Shape s2 = new Rectangle();
12         Rectangle r = new Rectangle();
13         System.out.println(r.offset_x);
14     }
15 }
```

1. Constructor
2. static keyword
3. this keyword
4. super keyword
5. final keyword
6. instanceof keyword
7. abstract keyword
8. implements keyword
9. toString() method



3분

1. Constructor

- 객체가 생성될 때, 자동으로 생성자가 호출됨 (부모 포함)
- 생성자 오버로딩을 활용하여 필드 초기화 등의 작업 수행

연습문제

- Shape class 기존 생성자에 파라미터 2개를 넣어주면 에러가 나는 이유는? 해결 방법은?
- 아래 s2 객체가 생성될 때, 어떤 class의 생성자가 호출될까? (Shape만? 둘 다?)

```
Shape s2 = new Rectangle();
```



1. Constructor

- 객체가 생성될 때, 자동으로 생성자가 호출됨 (부모 포함)
- 생성자 오버로딩을 활용하여 필드 초기화 등의 작업 수행

연습문제

- Shape class 기존 생성자에 파라미터 2개를 넣어주면 에러가 나는 이유는? 해결 방법은?
 - => 에러 발생 이유: 디폴트 생성자를 찾지 못해서
 - => 해결 방법 1: Shape 클래스에 파라미터 없는 생성자 추가
 - => 해결 방법 2: Rectangle 클래스 생성자에 super() 추가
- 아래 s2 객체가 생성될 때, 어떤 class의 생성자가 호출될까? (Shape만? 둘 다?)

```
Shape s2 = new Rectangle();
```

=> Rectangle 객체가 생성되는 과정에서, Shape, Rectangle 생성자 순으로 실행



5분

2. static keyword

- static variable (정적 변수), static method (정적 메소드)
- 객체 생성 없이 활용 가능 (메모리 절약 효과, 꼭 필요한 경우에만 활용)
- 메모리에 고정적으로 할당 (프로그램 종료시까지)

연습문제

- Rectangle class 내에 non-static printRectangle() method를 생성하고 이를 main method에서 호출하는 코드를 작성하라

```
public class Rectangle extends Shape {  
    public void printRectangle() {  
        System.out.println("Rectangle");  
    }  
}
```



2. **static** keyword

```
public class Rectangle extends Shape {  
    public void printRectangle() {  
        System.out.println("Rectangle");  
    }  
    public static void main(String[] args) {  
        Rectangle r = new Rectangle();  
        r.printRectangle();  
        // printRectangle(); doesn't work.  
    }  
}
```

3. **this** keyword

- 생성자 또는 메소드 내에서 현재 객체를 참조하기 위해 활용
- 아래 코드와 같이 동일 변수명을 쓰는 것이 일반적인 방식임을 확인하려면?

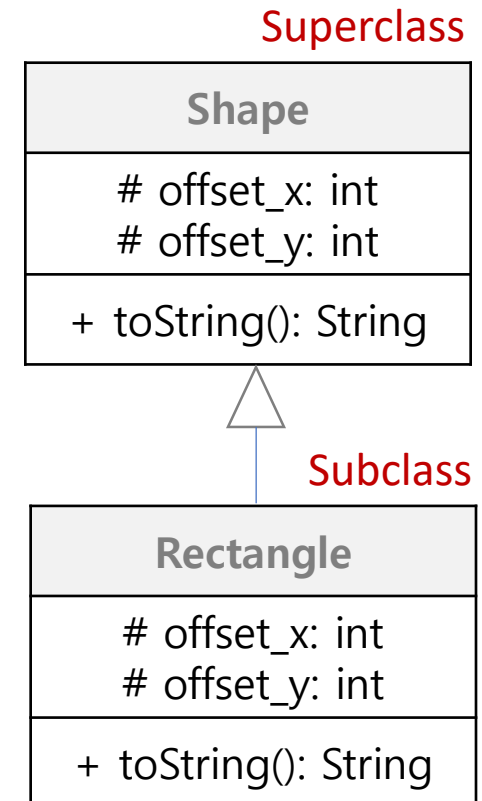
오픈 소스 코드를 살펴보자!

```
class Shape {  
    protected int offset_x, offset_y;  
    public Shape(int offset_x, int offset_y) {  
        this.offset_x = offset_x;  
        this.offset_y = offset_y;  
    }  
}
```

4. **super** keyword

- Subclass 내에서 Superclass 내 멤버들에 접근하기 위해 사용
- Superclass에 있는 overriding한 메소드 호출 등 다양한 상황에서 활용

```
class Shape {  
    protected int offset_x = 0, offset_y = 0;  
}  
  
public class Rectangle extends Shape {  
    protected int offset_x = 10, offset_y = 10;  
    public Rectangle() {  
        // [Print offset_x in Shape class]  
    }  
}
```



4. **super** keyword

- Subclass 생성자에서 Superclass 생성자를 호출할 때도 `super()` 사용
- 이 때, `super()` 호출은 반드시 가장 상단에 명시해야 함 (그렇지 않으면 에러 발생)

```
class Shape {  
    protected int offset_x, offset_y;  
    public Shape(int i, int j) {  
        this.offset_x = 0;  
        this.offset_y = 0;  
    }  
}  
  
public class Rectangle extends Shape {  
    public Rectangle() {  
        super(0, 0);  
    }  
}
```




7분

4. **super** keyword

연습문제

1. Subclass 객체에서 Superclass 멤버 접근은 **super** 키워드로 가능하다.
그럼 아래의 Shape 객체에서 Subclass 필드 값을 출력해보자.

```
Shape s = new Rectangle();
```

2. Subclass 내에서 Superclass 필드 값을 수정한 전후 값을 출력해보자
3. Superclass 내 필드 접근 제어자를 **protected**에서 **private**으로 변경 후
Subclass에서 해당 값에 접근했을 때 발생하는 에러메시지를 확인해보자



4. **super** keyword

연습문제

1. Subclass 내에서 Superclass 멤버 접근은 **super** 키워드로 가능하다.
그럼 반대로 아래 Superclass 객체에서 Subclass 필드를 출력해보자.

```
Shape s = new Rectangle();  
System.out.println(((Rectangle)s).width);
```

- Widening casting
- Narrowing casting

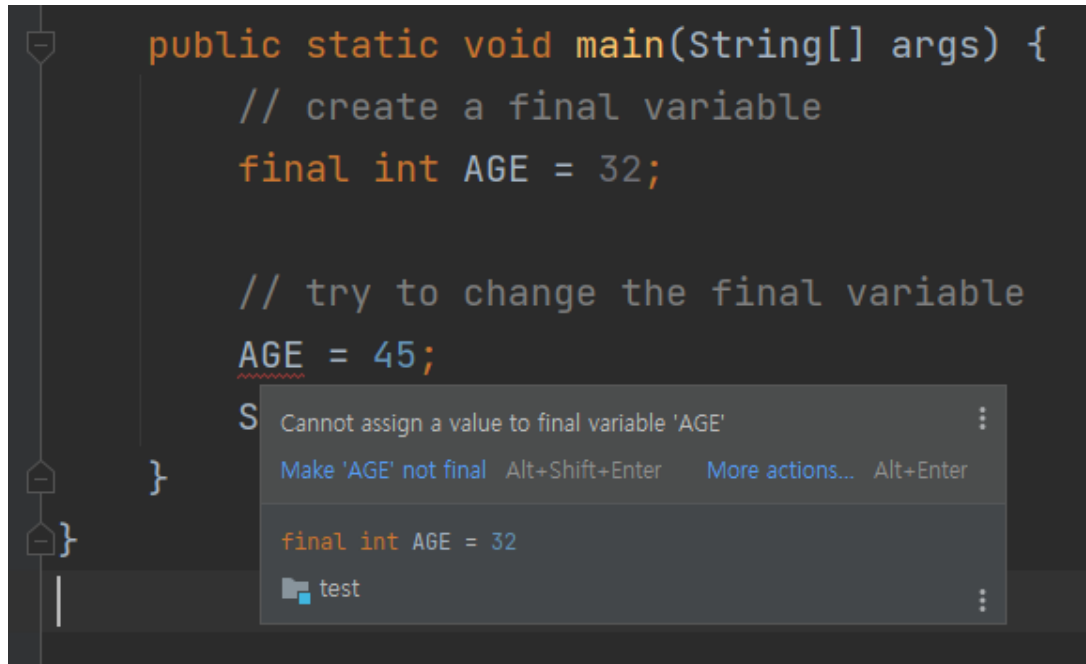
2. Subclass 내에서 Superclass 필드 값을 수정한 전후 값을 출력해보자

```
System.out.println(super.offset_x);  
super.offset_x = 1;  
System.out.println(super.offset_x);
```

```
java: offset_x has private access in Shape
```

5. final keyword

- 변경할 수 없게 하는 속성을 변수/메소드/클래스에 부여할 때 사용
 - final variable은 한 번 초기화 되면 값을 변경할 수 없음
 - final method는 subclass에서 Overriding 할 수 없음
 - final class는 다른 class에서 상속받을 수 없음



```
public static void main(String[] args) {  
    // create a final variable  
    final int AGE = 32;  
  
    // try to change the final variable  
    AGE = 45;  
}
```

The screenshot shows an IDE with a Java code snippet. The code declares a final variable AGE and attempts to reassign it. An error message is displayed: "Cannot assign a value to final variable 'AGE'". The error message includes options: "Make 'AGE' not final" (Alt+Shift+Enter), "More actions..." (Alt+Enter), and a "test" button.

final variable 초기화 방법

1. 선언 시점에 바로 값 정의
2. 생성자 내에서 초기화



5분

5. **final** keyword

- final variable의 변수명은 보통 대문자로 작성하여 상수임을 표기
- final method는 다른 곳에서 재정의하지 못하게 막고자 할 때 사용
- final class는 다른 곳에서 상속받지 못하게 막고자 할 때 사용

연습문제

1. Jenkins 오픈 소스 내 final 키워드 검색 결과에서
공통적으로 나타나는 경향을 찾아보자 (결과 수: 약 1,000개)
<https://github.com/jenkinsci/jenkins/tree/master/core/src/main/java/jenkins>
2. Math.java 내 final 키워드 검색 결과도 살펴보며 각각의 의도를 추측해보자
(Win10 IntelliJ 기준 Shift 버튼 두 번 누르고 파일명 검색)

6. instanceof keyword

- 객체가 특정 클래스의 인스턴스인지 여부를 검사하는 연산자 (operator)

```
class Shape {}  
public class Rectangle extends Shape {  
    public static void main(String[] args) {  
        Rectangle r = new Rectangle();  
        System.out.println(r instanceof Object);  
        System.out.println(r instanceof Shape);  
        System.out.println(r instanceof Rectangle);  
        System.out.println(r.getClass().equals(Shape.class));  
    }  
}
```

```
true  
true  
true  
false
```



3분

6. instanceof keyword

- 객체가 특정 클래스의 인스턴스인지 여부를 검사하는 연산자 (operator)

연습문제

1. 아래 코드의 실행 결과는 어떻게 될까?

직접 타이핑해서 확인해보고 수정해가며 탐구해보자.

```
class Shape {}  
public class Rectangle extends Shape {  
    public static void main(String[] args) {  
        Shape s = new Rectangle();  
        System.out.println(s instanceof Shape);  
        System.out.println(s.getClass().equals(Shape.class));  
    }  
}
```



6. instanceof keyword

- 객체가 특정 클래스의 인스턴스인지 여부를 검사하는 연산자 (operator)

연습문제

- 아래 코드의 실행 결과는 어떻게 될까?
직접 타이핑해서 확인해보고 수정해가며 탐구해보자.

```
class Shape {}  
public class Rectangle extends Shape {  
    public static void main(String[] args) {  
        Shape s = new Rectangle();  
        System.out.println(s instanceof Shape);  
        System.out.println(s.getClass().equals(Shape.class));  
    }  
}
```

true
false

7. **abstract** keyword

- Abstract class는 직접 객체 생성 불가하며 상속 받는 목적에만 활용 가능
- Abstract method는 내부에 구현을 가질 수 없고 선언만 가능 (Overriding!)

```
abstract class Shape {  
    abstract void printShape();  
}  
  
public class Rectangle extends Shape {  
    public void printShape() {  
        System.out.println("Rectangle");  
    }  
  
    public static void main(String[] args) {  
        // Shape s = new Shape();  
        Rectangle r = new Rectangle();  
    }  
}
```




5분

7. **abstract** keyword

- Abstract method가 있는 클래스를 상속 받으면 해당 method 구현 필수
- 대규모의 개발을 쪼개서 작업할 때 등의 상황에서 활용

연습문제

1. Shape을 abstract class로 변경, printShape() abstract method를 선언한 후 이를 각각 Rectangle, Circle 클래스에서 상속받아 서로 다르게 출력해보자 (20줄 이내면 Great!, 단일 파일 내 "public class" 키워드는 Rectangle에만 명시 필요)

```
Circle  
Rectangle
```



7. **abstract** keyword

```
abstract class Shape {  
    abstract void printShape();  
}  
  
class Circle extends Shape {  
    public void printShape() {  
        System.out.println("Circle");  
    }  
}  
  
public class Rectangle extends Shape {  
    public void printShape() {  
        System.out.println("Rectangle");  
    }  
  
    public static void main(String[] args) {  
        Circle c = new Circle();  
        Rectangle r = new Rectangle();  
        c.printShape();  
        r.printShape();  
    }  
}
```

8. implements keyword

- Interface는 “a fully abstract class” 개념으로, 자체 구현을 가지지 않음 (Java 8 이전까지)
- Abstract class와의 차이점: abstract method만 가질 수 있음, 다중 상속 허용

```
interface Line {  
    int length = 0;  
}  
  
interface Polygon {  
    abstract void getArea();  
}  
  
class Rectangle implements Line, Polygon {  
    public void getArea() {}  
}
```

Java 8부터 static/default method를

Java 9부터 private method를

각각 Interface 내에 정의 가능

(Java 8은 14년, Java 9는 17년에 공개)

(Java 17*은 21년, Java 20은 23년 공개)

(LTS 버전은 Java 8, 11, 17 순으로 배포)

9. toString() method

- 모든 Class에서 상속 받는 Object Class에는 toString 메소드가 정의되어 있음
- 리턴값은 "클래스명 + @ + 16진수 해시값" 포맷이며, 보통 Method Overriding를 통해 활용

```
class Shape {  
}  
  
public class Rectangle extends Shape {  
    public static void main(String[] args) {  
        Shape s = new Shape();  
        Rectangle r = new Rectangle();  
        System.out.println(s);  
        System.out.println(r);  
    }  
}
```

```
Shape@3b07d329  
Rectangle@41629346
```



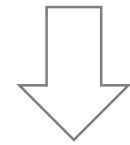
10분

9. toString() method

연습문제

- 아래 코드를 확장해서 Shape 클래스에 toString 메소드를 정의하고 (두 필드 값 출력)
이를 통해 Object, Shape, Rectangle 각각 순서대로 toString()을 출력해보자 (최소한의 코드로!)

```
class Shape {  
    protected int offset_x = 0, offset_y = 0;  
}  
  
public class Rectangle extends Shape {  
    protected int width = 10, height = 10;  
    public String toString() {  
        return "Rect size: (" + width + ", " + height + ")";  
    }  
    public static void main(String[] args) {  
        Rectangle r = new Rectangle();  
        System.out.println(r);  
    }  
}
```



정답 출력

```
Rectangle@3b07d329  
Shape offset: (0, 0)  
Rect size: (10, 10)
```



9. toString() method

```
class Shape {  
    protected int offset_x = 0, offset_y = 0;  
    public String toString() {  
        System.out.println(super.toString());  
        return "Shape offset: (" + offset_x + ", " + offset_y + ")";  
    }  
}  
  
public class Rectangle extends Shape {  
    protected int width = 10, height = 10;  
    public String toString() {  
        System.out.println(super.toString());  
        return "Rect size: (" + width + ", " + height + ")";  
    }  
    public static void main(String[] args) {  
        Rectangle r = new Rectangle();  
        System.out.println(r);  
    }  
}
```