23.03.21 12강

어제 피드백

UID 넣을때 숫자 다르게 해야함

람다식과 함수, 스트림

1급 객체

변수에 대입 가능한 객체를 1급 객체 (first class object) 라고 한다.

대표적인 1급 객체 : 값, 인스턴스, 배열

메소드 자체를 변수에 넣는 것 ⇒ 안됨

java 8에서 추가됨 ⇒ 함수도 1급 객체로 취급 됨

입력을 처리하여 출력하는 것

입출력 타입만 같다면 메서드를 변수에 대입하는 것이 가능

ex)

```
package Lesson.day12.rambdaandfunc;

import java.util.function.IntBinaryOperator;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator func = RambdaAndFunction::add;

        int result = func.applyAsInt(5, 3);
        System.out.println("5 + 3 = " + result);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

:: ⇒ 메소드 레퍼런스

해당 클래스의 메소드를 불러옴

`ClassName::MethodName`

메서드와 함수의 차이

메서드는 클래스에 속하고 클래스를 조작하기 위한 일종의 함수
메서드는 이름이 있지만 함수에게 이름은 중요치 않다.

내용이 동일하면 같은 함수

함수 저장용 인터페이스 선언

```
package Lesson.day12.rambdaandfunc;

import java.util.function.IntBinaryOperator;
```

```
public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;

        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static int duhagi(int x, int y) {
        return x + y;
    }

    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

1) 인터페이스로 저장

2) 함수선언 및 사용

추상 메서드가 1개인(SAM : Single Abstract Method) 인터페이스는 함수 저장용으로 사용 가능

함수 저장용 범용 API

함수를 저장하기 위해 매번 SAM 인터페이스를 만드는 것은 귀찮음

따라서 Java에서는 자주 사용할 것 같은 함수 저장용 범용 API를 준비 해 놨음

대표적인 SAM 인터페이스 (함수형 인터페이스라고도 함)

함수형 인터페이스	Descriptor	Method명
Predicate<T>	T -> boolean	test()
Consumer<T>	T -> void	accept()
Supplier<T>	() -> T	accept()
Function<T,R>	T -> R	apply()
UnaryOperator<T>	T -> T	identity()

함수형 인터페이스 써도 되고 안써도 되고.

IntBinaryOperator와 마찬가지

람다식

함수는 이름이 중요하지 않음 → 없어도 그만

함수 내용을 바로바로 정의해서 사용하고 싶다~!!

```
package Lesson.day12.rambdaandfunc;

import java.util.function.IntBinaryOperator;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;
        MyFunction rambdafunc = (int a, int b) -> {
            return a + b;
        };
        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        int result3 = rambdafunc.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
        System.out.println("5 + 3 = " + result3);
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static int duhagi(int x, int y) {
        return x + y;
    }

    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

익명함수(람다식) → 익명 클래스와 유사함

익명클래스를 줄일수 있는것 (단 인터페이스에 함수가 하나일때만)

```
package Lesson.day12.rambdaandfunc;

import java.util.function.IntBinaryOperator;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;
        MyFunction rambdaFunc = (int a, int b) -> {
            return a + b;
        };
        MyFunction anomimousCall = new MyFunction() {

            @Override
            public int call(int x, int y) {
                return x + y;
            }
        };
        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        int result3 = rambdaFunc.call(5, 3);
        int result4 = anomimousCall.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
        System.out.println("5 + 3 = " + result3);
        System.out.println("5 + 3 = " + result4);
    }

    public static int add(int x, int y) {
        return x + y;
    }
```

```
    public static int duhagi(int x, int y) {
        return x + y;
    }

    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

회사마다 람다식을 금지하는 경우가 있다 → 재활용성과 가독성 특성에서 좋지 않다.

람다식의 표기법

입력타입을 생략 가능하다.

입력값이 1개일때 소괄호 생략이 가능하다.

return문 한줄로 구성 된 경우 return도 생략 가능하다.

스트림(stream)

함수를 값으로 취급할 때의 이점은?? ⇒ 함수형 프로그래밍

Java8에 추가된 함수를 다루기 위한 범용 API인 Stream을 사용하면 좀 더 함수적인 사고를 가지고 개발할 수 있다.

```
package Lesson.day12.rambdaandfunc;

import java.util.*;
import java.util.function.IntBinaryOperator;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;
        MyFunction rambdaFunc = (int a, int b) -> {
            return a + b;
        };
        MyFunction anomimousCall = new MyFunction() {

            @Override
            public int call(int x, int y) {
                return x + y;
            }
        };

        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        int result3 = rambdaFunc.call(5, 3);
        int result4 = anomimousCall.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
        System.out.println("5 + 3 = " + result3);
        System.out.println("5 + 3 = " + result4);

        List<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6));
        nums.stream().forEach(num -> System.out.print(num));
        System.out.println();
        nums.stream().forEach(System.out::print);
    }

    public static int add(int x, int y) {
        return x + y;
    }
```

```
    public static int duhagi(int x, int y) {
        return x + y;
    }

    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

리스트의 요소들이 일렬로 서서 각각 프린트 나오는 것 ⇒ 반복문을 줄여준다.

input output이 완전이 같다면 아래처럼 :: 만으로 할 수 있다.

foreach의 범용 API : consumer<T>

리턴이 없는 void함수만 가능

filter

filter 범용 API : **Predicate**<T> ⇒ boolean

주어진 조건에 맞는 것만 골라냄

for if를 줄일 수 있음

```
package Lesson.day12.rambdaandfunc;

import java.util.*;
import java.util.function.IntBinaryOperator;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;
        MyFunction rambdaFunc = (int a, int b) -> {
            return a + b;
        };
        MyFunction anomimousCall = new MyFunction() {

            @Override
            public int call(int x, int y) {
                return x + y;
            }
        };

        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        int result3 = rambdaFunc.call(5, 3);
        int result4 = anomimousCall.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
        System.out.println("5 + 3 = " + result3);
        System.out.println("5 + 3 = " + result4);

        List<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6));
        nums.stream().forEach(num -> System.out.print(num));
        System.out.println();
        nums.stream().forEach(System.out::print);
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).forEach(System.out::print);
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static int duhagi(int x, int y) {
        return x + y;
    }
```

```
    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

Map

filter 범용 API : Function<T, R> ⇒ 입력 T, 출력 R

key에 맞는 value 나옴 → 다른 타입

```
package Lesson.day12.rambdaandfunc;

import java.util.*;
import java.util.function.IntBinaryOperator;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;
        MyFunction rambdaFunc = (int a, int b) -> {
            return a + b;
        };
        MyFunction anomimousCall = new MyFunction() {

            @Override
            public int call(int x, int y) {
                return x + y;
            }
        };

        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        int result3 = rambdaFunc.call(5, 3);
        int result4 = anomimousCall.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
        System.out.println("5 + 3 = " + result3);
        System.out.println("5 + 3 = " + result4);

        List<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6));
        nums.stream().forEach(num -> System.out.print(num));
        System.out.println();
        nums.stream().forEach(System.out::print);
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).forEach(System.out::print);
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).forEach(num -> System.out.print(num / 2));
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).map(num -> num + "의 몫 : " + num / 2 + "\n")
                .forEach(num -> System.out.print(num));
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static int duhagi(int x, int y) {
        return x + y;
    }

    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

```
package Lesson.day12.rambdaandfunc;

import java.util.*;
import java.util.function.IntBinaryOperator;
import java.util.stream.Collectors;

public class RambdaAndFunction {
    public static void main(String[] args) {
        IntBinaryOperator funcIBO = RambdaAndFunction::add;
        MyFunction funcMy = RambdaAndFunction::duhagi;
        MyFunction rambdaFunc = (int a, int b) -> {
            return a + b;
        };
        MyFunction anomimousCall = new MyFunction() {

            @Override
            public int call(int x, int y) {
                return x + y;
            }
        };

        int result1 = funcIBO.applyAsInt(5, 3);
        int result2 = funcMy.call(5, 3);
        int result3 = rambdaFunc.call(5, 3);
        int result4 = anomimousCall.call(5, 3);
        System.out.println("5 + 3 = " + result1);
        System.out.println("5 + 3 = " + result2);
        System.out.println("5 + 3 = " + result3);
        System.out.println("5 + 3 = " + result4);

        List<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5, 6));
        nums.stream().forEach(num -> System.out.print(num));
        System.out.println();
        nums.stream().forEach(System.out::print);
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).forEach(System.out::print);
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).forEach(num -> System.out.print(num / 2));
        System.out.println();
        nums.stream().filter(num -> num % 2 == 0).map(num -> num + "의 몫 : " + num / 2 + "\n")
                .forEach(num -> System.out.print(num));

        List<String> names = new ArrayList<>();
        names.add("박경덕");
        names.add("박준하");
        names.add("박태현");
        names.add("이동학");

        List<String> parks = getParkList(names);
        System.out.println(parks);

        System.out.println(
                names.stream().filter(name -> name.startsWith("박")).collect(Collectors.toList()));
    }

    public static List<String> getParkList(List<String> names) {
        return names.stream().filter(name -> name.startsWith("박")).collect(Collectors.toList());
        // List<String> results = new ArrayList<>();
        //
        // for (String name : names) {
        // if (name.startsWith("박")) {
        // result.add(name);
        // }
        // }
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static int duhagi(int x, int y) {
```

```
        return x + y;
    }

    interface MyFunction {
        public abstract int call(int x, int y);
    }
}
```

연습문제

연습문제 5-1

survivalcoding

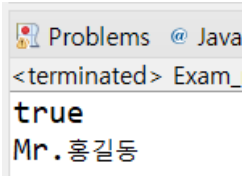
다음 코드에 작성되어 있는 두 개의 **static** 메서드를 함수로서 변수에 담고, 그것을 호출하는 프로그램을 작성하시오.
이 때 함수를 대입하기 위해 필요한 **Func1**, **Func2** 인터페이스를 정의하시오.
메서드 이름이나 인수 이름은 자유롭게 정해도 됩니다.

```
1   public class Utils {
2
3     public static boolean isOdd(int n) {
4       return n % 2 == 1;
5     }
6
7     public static addNamePrefix(boolean male, String name) {
8       if (male == true) {
9         return "Mr." + name;
10      }
11      return "Ms." + name;
12    }
13  }
```

```
package Lesson.day12.rambdaandfunc;

interface Func1 {
    public abstract boolean func1(int n);
}

interface Func2 {
    public abstract String func2(boolean male, String name);
}
```

```
package Lesson.day12.rambdaandfunc;

public class Exam_rambda_1 {
    public static void main(String[] args) {
        Func1 func1 = Exam_rambda_1::isOdd;
        Func2 func2 = Exam_rambda_1::addNamePrefix;

        System.out.println(func1.func1(15));
        System.out.println(func2.func2(true, "홍길동"));
    }

    public static boolean isOdd(int n) {
        return n % 2 == 1;
    }

    public static String addNamePrefix(boolean male, String name) {
        if (male == true) {
            return "Mr." + name;
```

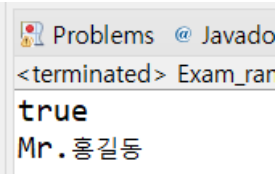
```
        }
        return "Ms." + name;
    }
}
```



연습문제 5-2

연습문제 5-1의 Utils 클래스의 2가지 메서드와 동일한 내용을 람다식으로 표현하여 각각 Func1, Func2 타입 변수에 대입하는 문장을 작성하시오.

```
package Lesson.day12.rambdaandfunc;

public class Exam_rambda_2 {
    public static void main(String[] args) {
        Func1 func1 = n -> n % 2 == 1;
        Func2 func2 = (male, name) -> male ? ("Mr." + name) : ("Ms." + name);

        System.out.println(func1.func1(15));
        System.out.println(func2.func2(true, "홍길동"));
    }
}
```



네트워크 통신

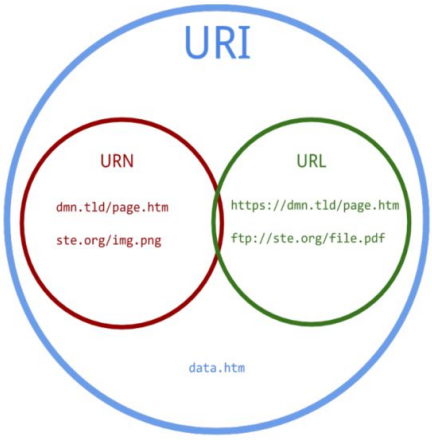
URL을 사용한 고수준 액세스

고수준 : 사람이 이해하기 쉽게 작성된 프로그램 또는 API

저수준 : 기계어에 가까운거겠지?

저수준 : 컴퓨터가 이해하기 쉽게 작성된 프로그램 또는 API

url

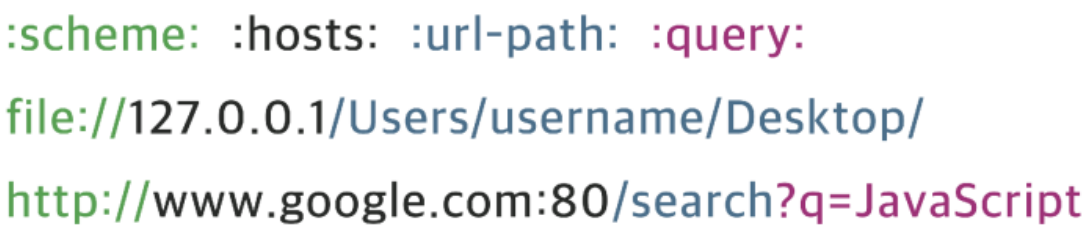


결론부터 말하자면 URI는 URL의 상위 개념이다.

URI란?

- URI는 Uniform Resource Identifier, 통합 자원 식별자의 줄임말이다.
- 즉, URI는 인터넷의 자원을 식별할 수 있는 문자열을 의미한다.
- URI의 하위 개념으로 URL과 URN이 있다.
- URI 중 URL이라는, URN이라는 하위 개념을 만들어서 특별히 어떤 표준을 지켜서 자원을 식별하는 것이다.
- 결론은 URI라는 개념은 어떤 형식이 있다기 보다는 특정 자원을 식별하는 문자열을 의미한다. 그래서 URL이 아니고 URN도 아니면 그냥 URI가 되는 것이다.

URL구조




```
scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```

1. scheme : 사용할 프로토콜을 뜻하며 웹에서는 http 또는 https를 사용
2. user와 password : (서버에 있는) 데이터에 접근하기 위한 사용자의 이름과 비밀번호
3. host와 port : 접근할 대상(서버)의 호스트명과 포트번호
4. path : 접근할 대상(서버)의 경로에 대한 상세 정보
5. query : 접근할 대상에 전달하는 추가적인 정보 (파라미터)
6. fragment : 메인 리소스 내에 존재하는 서브 리소스에 접근할 때 이를 식별하기 위한 정보

URL이란?

- URL은 Uniform Resource Locator의 줄임말이다.
- URL은 네트워크 상에서 리소스(웹 페이지, 이미지, 동영상 등의 파일) 위치한 정보를 나타낸다.
- URL은 HTTP 프로토콜 뿐만아니라 FTP, SMTP 등 다른 프로토콜에서도 사용할 수 있다.
- URL은 웹 상의 주소를 나타내는 문자열이기 때문에 더 효율적으로 리소스에 접근하기 위해 클린한 URL 작성을 위한 방법론이다.

URN이란?

- URN은 Uniform Resource Name의 줄임말이다.
- URN은 URI의 표준 포맷 중 하나로, 이름으로 리소스를 특정하는 URI이다.
- http와 같은 프로토콜을 제외하고 리소스의 name을 가리키는데 사용된다.
- URN에는 리소스 접근방법과, 웹 상의 위치가 표기되지 않는다.
- 실제 자원을 찾기 위해서는 URN을 URL로 변환하여 이용한다.

URL과 URN의 차이점

```
URL: ftp://ftp.is.co.za/rfc/rfc1808.txt
URL: http://www.ietf.org/rfc/rfc2396.txt
URL: ldap://[2001:db8::7]/c=GB?objectClass?one
URL: mailto:John.Doe@example.com
URL: news:comp.infosystems.www.servers.unix
URL: telnet://192.0.2.16:80/
URN (not URL): urn:oasis:names:specification:docbook:dtd:xml:4.1.2
URN (not URL): tel:+1-816-555-1212 (?)
```

- **URL**은 **어떻게 리소스를 얻을 것**이고 **어디에서 가져와야하는지 명시**하는 URI이다.
- **URN**은 리소스를 어떻게 접근할 것인지 명시하지 않고 **경로와 리소스 자체를 특정**하는 것을 목표로하는 URI이다.

Web 페이지에 접속

1. 브라우저에서 http://www.google.com 을 입력
2. 브라우저가 http://www.google.com 으로 접속
3. 페이지 내용이 HTML 형식으로 송신됨
4. 브라우저가 HTML을 파싱하여 화면에 출력

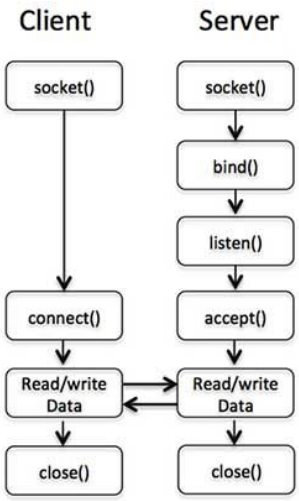
```
package Lesson.day12.newwork;

import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;

public class Network {
    public static void main(String[] args) throws IOException {
        try {
            URL url = new URL("https://www.google.com");
            InputStream is = url.openStream();
            InputStreamReader isr = new InputStreamReader(is);
            int i = isr.read();
            while (i != -1) {
                System.out.println((char) i);
                i = isr.read();
            }
            isr.close();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

TCP/IP

웹페이지 접속, 메일 전송, 게임 등은 모두 TCP/IP 를 통한 통신에 의해 이루어 진다.

1. java.net.Socket 클래스를 사용하면 TCP/IP 통신을 할 수 있음
2. 접속하기 위해 IP 주소와 포트 번호가 필요
3. 프로토콜(통신시 사용되는 데이터 형식이나 순서 등)은 RFC 문서에 정해둔 것을 따른다



연습문제

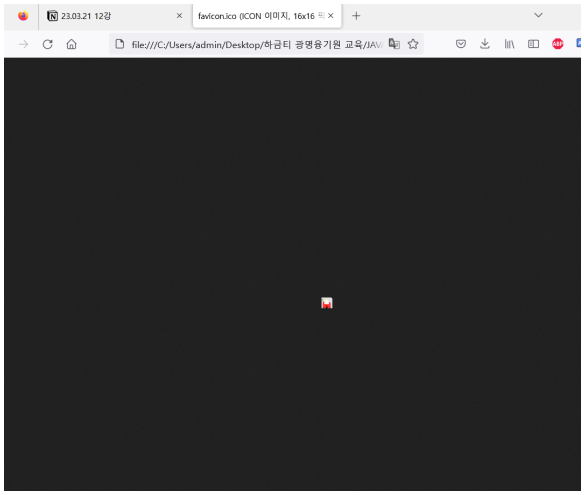
연습문제 8-1

java.net.URL 클래스를 사용하여 다음 주소의 그림 파일을 읽어서 PC에 저장하는 프로그램을 작성하시오.

- 그림 파일 주소 : https://alimipro.com/favicon.ico
- 저장 위치 : 아무데나
- 파일명 : icon.ico

https://alimipro.com/favicon.ico

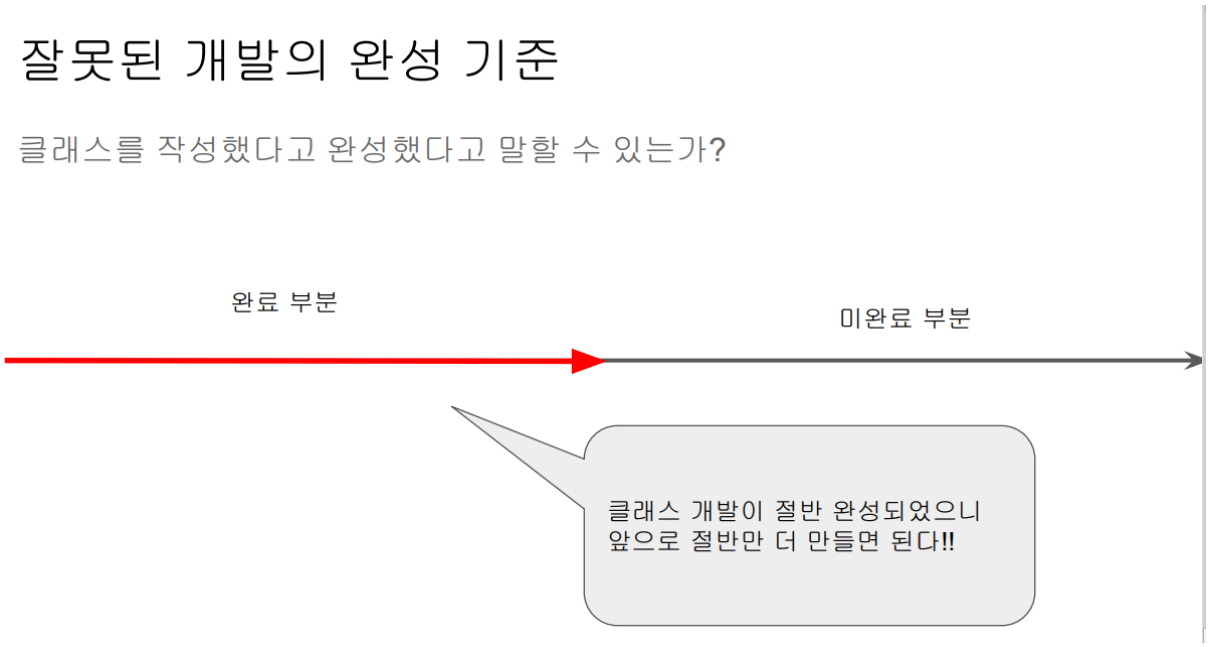
```
package Lesson.day12.newwork;

import java.io.*;
import java.net.*;

public class Exam_networks_1 {
    public static void main(String[] args) throws IOException {
        String fileLink = "https://alimipro.com/favicon.ico";
        String fileName = fileLink.split("/")[fileLink.split("/").length - 1];
        String fileDir =
                "C:\\\\Users\\\\admin\\\\Desktop\\\\하금티 광명융기원 교육\\\\JAVA_KOPOXHANA\\\\수업자료\\\\JAVA\\\\";
        String DownLoadLink = fileDir + fileName;
        DownLoadFile(fileLink, DownLoadLink);
    }

    public static void DownLoadFile(String fileLink, String DownLoadLink) throws IOException {
        try {
            URL url = new URL(fileLink);
            InputStream is = url.openStream();
            InputStreamReader isr = new InputStreamReader(is);
            try (FileOutputStream fos = new FileOutputStream(DownLoadLink, true);) {
                int i = isr.read();
                while (i != -1) {
                    fos.write((char) i);
                    i = isr.read();
                }
                isr.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```



Unit Test

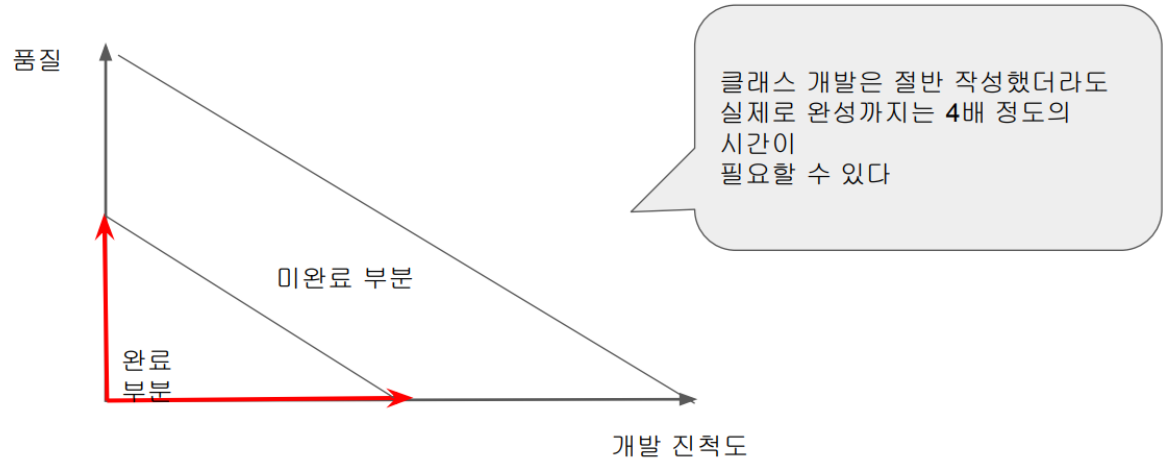
중요하지만 배울 기회가 없으므로 잘 배워 보자구!

만들었다고 다가 아니다



올바른 완성의 기준

품질을 생각한 개발



평가의 기준이 개발 진척도만 있는게 아니라 품질도 있다 볼 수 있다.

테스트를 통한 품질 향상

테스트를 하는 방법들

=> 수동 테스트 : 인간이 하는 테스트

=> 단위 테스트 : 1개 클래스를 테스트

=> 통합 테스트 : 여러개 연관된 클래스를 함께 테스트

단위(Unit) 테스트

단위 테스트란?

특정 모듈이 의도한 대로 잘 작동하는가를 테스트하는 것

테스트 케이스

가능한 모든 가능성의 범위를 테스트하는 것이 좋은 테스트 케이스이다

- 장애에 관한 신속한 피드백
- 개발 주기에서 조기 장애 감지
- 회귀에 신경 쓸 필요 없이 코드를 최적화할 수 있도록 하는 더 안전한 코드 리팩터링
- 기술적 문제를 최소화하는 안정적인 개발 속도

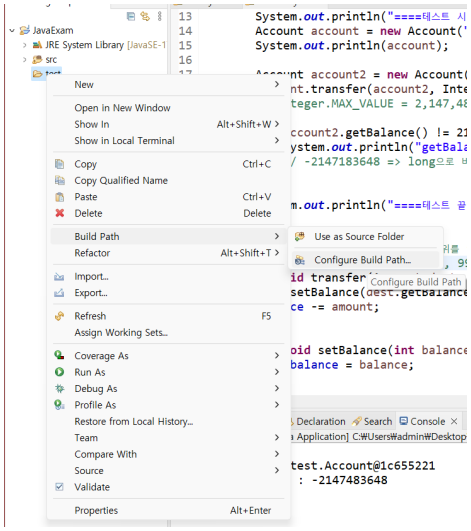
단위 테스트가 꼭 필요한 경우

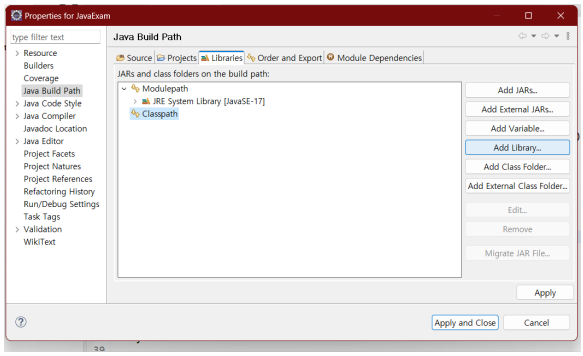
- DB
  - 스키마가 변경되는 경우
  - 모델 클래스가 변경되는 경우
- Network

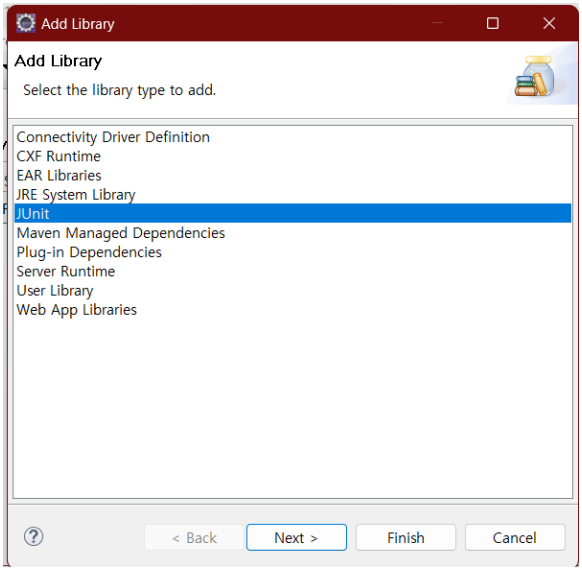
○ 예측한 데이터가 제대로 들어오는지
● 데이터 검증
○ 예측한 데이터를 제대로 처리하고 있는지

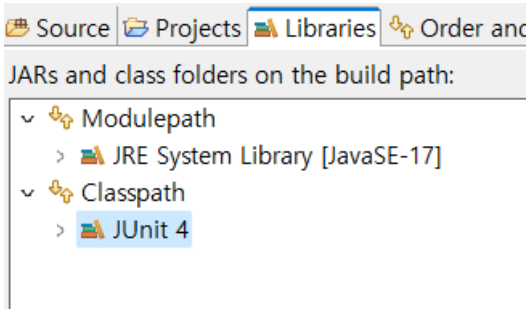
JUnit

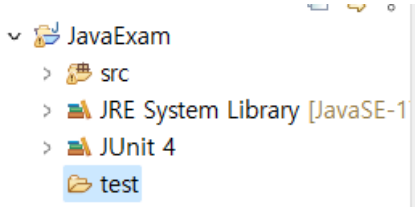
Java 용 Unit Test 용 라이브러리

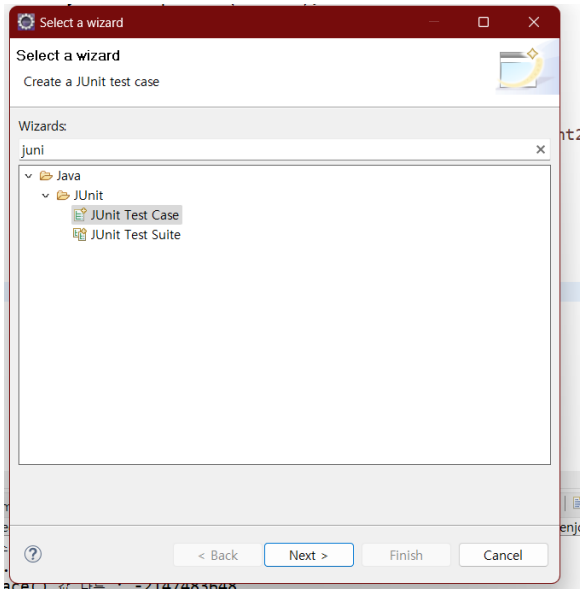










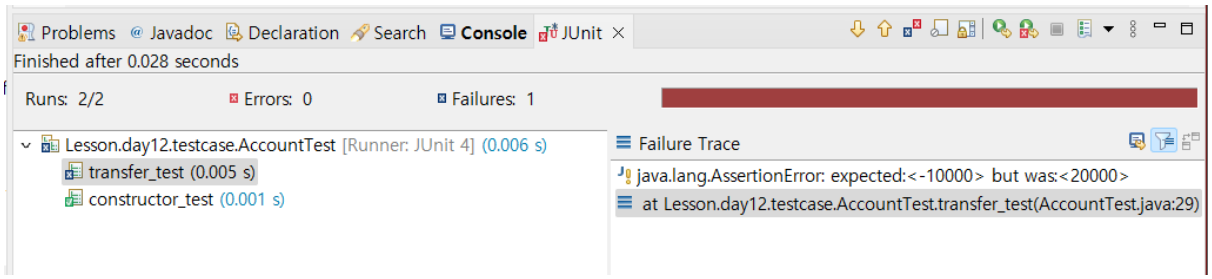


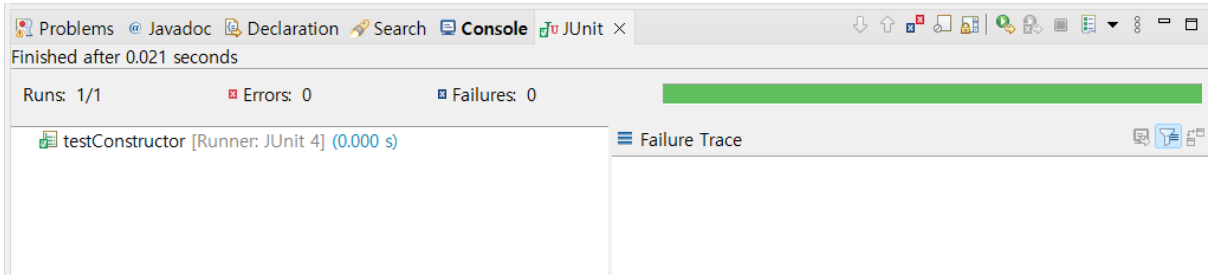
Main.java | Account.java | AccountTest.java

```java
package Lesson.day12.testcase;

import static org.junit.Assert.*;

public class AccountTest {

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

본 코드와 섞이지 않게 다른 패키지에 선언한다.

```java
package Lesson.day12.testcase;

import static org.junit.Assert.*;
import org.junit.Test;
import Lesson.day12.unittest.Account;

public class AccountTest {

    @Test
    public void testConstructor() {
        Account account = new Account("홍길동", 30000);
        assertEquals("홍길동", account.getOwner());
        assertEquals(30000, account.getBalance());
    }

}
```





특정 테스트가 특정 예외가 발생되어야 하는 것을 테스트

```java
@Test(expected = IllegalArgumentException.class)
public void throwsExceptionWithTwoCharName() {
    ...
}
```

연습문제

연습문제 1

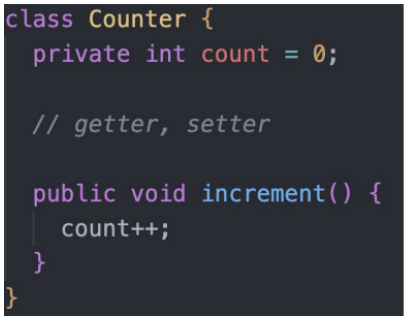
다음 코드를 검사하는 JUnit 테스트 클래스 BankTest 클래스를 작성하시오.

```java
public class Bank {
  private String name;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    if (name.length() <= 3) {
      throw new IllegalArgumentException("이름이 잘못 되었음");
    }
    this.name = name;
  }
}
```

연습문제 2

다음과 같은 Counter 클래스가 있습니다. 이 클래스는 1씩 증가하는 기능이 필요할 때 쉽게 활용할 수 있습니다.

1씩 감소하는 카운터가 필요할 때를 위해, 1씩 감소하는 기능을 추가하시오.



```
class Counter {
  private int count = 0;

  // getter, setter

  public void increment() {
    count++;
  }
}
```

```
package Lesson.day12.unittest;

public class Counter {
    private int count = 0;

    public void setCount(int count) {
        this.count = count;
    }

    public int getCount() {
        return this.count;
    }

    public void increment() {
        count++;
    }
}
```

```
package Lesson.day12.unittest;

public class DownCounter {
    private int count = 0;

    public void setCount(int count) {
        this.count = count;
    }

    public int getCount() {
        return this.count;
    }

    public void increment() {
        count--;
    }
}
```

연습문제 3

Counter 클래스의 Unit 테스트 코드를 작성합니다.

테스트는 public 메소드 전부를 테스트 합니다.

```
package Lesson.day12.testcase;

import static org.junit.Assert.*;
import org.junit.Test;
import Lesson.day12.unittest.Account;
import Lesson.day12.unittest.Counter;

public class CounterTest {
    @Test
    public void constructor_test() {
        Counter counter = new Counter(30000);
        assertEquals(30000, counter.getCount());
    }

    @Test
    public void setCount_test() {
        Counter counter = new Counter(30000);
        assertEquals(30000, counter.getCount());
        counter.setCount(1000);
        assertEquals(1000, counter.getCount());
        counter.setCount(100);
        assertEquals(100, counter.getCount());
    }

    @Test
    public void getCount_test() {
        Counter counter = new Counter(30000);
        assertEquals(30000, counter.getCount());
        counter.setCount(1000);
        assertEquals(1000, counter.getCount());
        counter.setCount(100);
        assertEquals(100, counter.getCount());
    }

    @Test
    public void increment_test() {
        Counter counter = new Counter(30000);
        counter.increment();
        assertEquals(30001, counter.getCount());
        counter.setCount(1000);
        counter.increment();
        assertEquals(1001, counter.getCount());
        counter.setCount(100);
        counter.increment();
        assertEquals(101, counter.getCount());
    }

}
```

```
package Lesson.day12.testcase;

import static org.junit.Assert.*;
import org.junit.Test;
import Lesson.day12.unittest.DownCounter;

public class DownCounterTest {
    @Test
    public void constructor_test() {
        DownCounter downcounter = new DownCounter(30000);
        assertEquals(30000, downcounter.getCount());
    }

    @Test
    public void setCount_test() {
        DownCounter downcounter = new DownCounter(30000);
        assertEquals(30000, downcounter.getCount());
        downcounter.setCount(1000);
        assertEquals(1000, downcounter.getCount());
        downcounter.setCount(100);
        assertEquals(100, downcounter.getCount());
    }

    @Test
    public void getCount_test() {
        DownCounter downcounter = new DownCounter(30000);
        assertEquals(30000, downcounter.getCount());
        downcounter.setCount(1000);
        assertEquals(1000, downcounter.getCount());
        downcounter.setCount(100);
        assertEquals(100, downcounter.getCount());
    }

    @Test
    public void increment_test() {
        DownCounter downcounter = new DownCounter(30000);
        downcounter.increment();
        assertEquals(29999, downcounter.getCount());
        downcounter.setCount(1000);
        downcounter.increment();
        assertEquals(999, downcounter.getCount());
        downcounter.setCount(100);
        downcounter.increment();
        assertEquals(99, downcounter.getCount());
    }
}
```

연습문제 4

카운터는 항상 카운트를 하는 동작 하나를 제공합니다.

Counter와 DownCounter 의 동작 메소드 increment() 는 적절한 이름이 아닌 것 같습니다. count() 와 같이 증가나 감소에 상관없는 기능의 이름으로 수정하는 것이 좋아 보입니다. 수정 하시오.

```
package Lesson.day12.unittest;

public class Counter {
    private int count = 0;

    public Counter(int count) {
        this.count = count;
    }
```

```
    public void setCount(int count) {
        this.count = count;
    }

    public int getCount() {
        return this.count;
    }

    public void count() {
        count++;
    }
}
```

```
package Lesson.day12.unittest;

public class DownCounter {
    private int count = 0;

    public DownCounter(int count) {
        this.count = count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public int getCount() {
        return this.count;
    }

    public void count() {
        count--;
    }
}
```

연습문제 5

공통의 메소드를 가지는 2개의 클래스는 하나의 공통 인터페이스를 구현할 수 있습니다.

Counter 인터페이스를 구현하면 int count() 메소드를 가지고 있으며 카운트 동작을 수행한 후에 현재의 카운트 값을 리턴합니다.

Counter 클래스는 적절한 이름으로 수정하고 대신 Counter 인터페이스를 구현하도록 수정하시오.

DownDounter 또한 Counter 인터페이스를 구현하도록 수정하시오.

```
package Lesson.day12.unittest;

public interface Countable {
    int count();
}
```

```
package Lesson.day12.unittest;

public class UpCounter implements Countable {
    private int count = 0;
```

```
    public UpCounter(int count) {
        this.count = count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public int getCount() {
        return this.count;
    }

    public int count() {
        return this.count++;
    }
}
```

```
package Lesson.day12.unittest;

public class DownCounter implements Countable {
    private int count = 0;

    public DownCounter(int count) {
        this.count = count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public int getCount() {
        return this.count;
    }

    public int count() {
        return this.count--;

    }
}
```

```
package Lesson.day12.testcase;

import static org.junit.Assert.*;
import org.junit.Test;
import Lesson.day12.unittest.UpCounter;

public class UpCounterTest {
    @Test
    public void constructor_test() {
        UpCounter upCounter = new UpCounter(30000);
        assertEquals(30000, upCounter.getCount());
    }

    @Test
    public void setCount_test() {
        UpCounter upCounter = new UpCounter(30000);
        assertEquals(30000, upCounter.getCount());
        upCounter.setCount(1000);
        assertEquals(1000, upCounter.getCount());
        upCounter.setCount(100);
        assertEquals(100, upCounter.getCount());
    }

    @Test
    public void getCount_test() {
        UpCounter upCounter = new UpCounter(30000);
        assertEquals(30000, upCounter.getCount());
```

```
        upCounter.setCount(1000);
        assertEquals(1000, upCounter.getCount());
        upCounter.setCount(100);
        assertEquals(100, upCounter.getCount());
    }

    @Test
    public void count_test() {
        UpCounter upCounter = new UpCounter(30000);
        upCounter.count();
        assertEquals(30001, upCounter.getCount());
        upCounter.count();
        assertEquals(30002, upCounter.getCount());
        upCounter.setCount(1000);
        upCounter.count();
        assertEquals(1001, upCounter.getCount());
        upCounter.setCount(100);
        upCounter.count();
        assertEquals(101, upCounter.getCount());
    }

}
```

```
package Lesson.day12.testcase;

import static org.junit.Assert.*;
import org.junit.Test;
import Lesson.day12.unittest.DownCounter;

public class DownCounterTest {
    @Test
    public void constructor_test() {
        DownCounter downCounter = new DownCounter(30000);
        assertEquals(30000, downCounter.getCount());
    }

    @Test
    public void setCount_test() {
        DownCounter downCounter = new DownCounter(30000);
        assertEquals(30000, downCounter.getCount());
        downCounter.setCount(1000);
        assertEquals(1000, downCounter.getCount());
        downCounter.setCount(100);
        assertEquals(100, downCounter.getCount());
    }

    @Test
    public void getCount_test() {
        DownCounter downCounter = new DownCounter(30000);
        assertEquals(30000, downCounter.getCount());
        downCounter.setCount(1000);
        assertEquals(1000, downCounter.getCount());
        downCounter.setCount(100);
        assertEquals(100, downCounter.getCount());
    }

    @Test
    public void count_test() {
        DownCounter downCounter = new DownCounter(30000);
        downCounter.count();
        assertEquals(29999, downCounter.getCount());
        downCounter.count();
        assertEquals(29998, downCounter.getCount());
        downCounter.setCount(1000);
        downCounter.count();
        assertEquals(999, downCounter.getCount());
        downCounter.setCount(100);
        downCounter.count();
        assertEquals(99, downCounter.getCount());
```

```
    }
}
```

오늘의 교훈

스트림 코테 풀 때 쓸만할 것 같다