DAY 2

💡 **JVM**
Os로부터 자원을 할당받아서 그 자원을 기반으로 실행시켜주는 프로그램
JVM + code API + utility
_____JRE_____
_____JDK_____

Class의 구성요소

- **fileds**
- **constructor**
- **methods**

💡 **Program이 실행되면**
1. JVM이 기동
2. Class loader가 해당 Class를 Loading
3. Class file 에 대한 검사
4. main ( ) 호출 → Class Loader와 우리 코드는 서로 다른Package이다.
→ 호출이 가능하려면 Public으로 지정해야한다.
5. Instance를 생성하지 않고 main을 호출해야 하기때문에 Static지정
6. Main이 시작되면 Program이 시작되고 Main이 끝나면 Program이 끝난다
→ 정확한 표현은 아니지만 일단 이렇게 생각 그래서 return값이 없다.

JVM의 Memory구조

💡 **1. Register (실행 pointer 관리)**
2. Runtime Constant Pool (상수값 사용)
→ 여기는 신경 쓸 필요가 없다.

---

3. Method Area
4. Call back
5. Heap
⭐중요⭐

---

static block → main ( ) 을 사용하기 직전에 특수한 처리(library Loading)을 하기 위해 사용.

Inheritance(상속)

부모 Class가 가지는 내용을 확장해서 자식 class를 만드는 방식

1. **extends keyword 상속 구현**
2. **Java는 단일 상속만 지원 (다중 상속을 지원하지 않는다.)**

Q. 상속은 항상 좋은가???

💡 **A. 항상 좋은건 아니다.**
→ class가 tightly coupled 되고 재사용에 문제가 발생한다.
→ 물론 해결가능!!

Q. Constructor가 상속에서는 어떻게 동작하나요??

💡 **A. Constructor는 상속되지 않는다.**
→ Private access modifier로 지정된 field,method도 상속되지 않아요!

Java의 모든 Class는 Object class를 상속하고 있다.

Java의 모든 class는 특정 package안에 포함

→ Object class는 Java.lang.Object라는 package에 묶여서 제공

```
package Inheritance;

class Person extends Object { // 최상위 클래스의 constructor를 호출해서 instance를 만든다.
                // Object를 감싸는 Person이라는 객체를 만든다.
  String name;
```

```
  String mobile;
}

class Student extends Person {
  String dept;
}

public class Main {
  public static void main(String[] args) {
//    Person p = new Person();
//    Student s = new Student();
//    Person s = new Student();
    Object s = new Student();
    // is-a relationship
    // subclass is a superclass
  }
}
```

constructor 가 호출되지 않으면 instance가 만들어지지 않는다.

```
package Inheritance;

class Person extends Object { // 최상위 클래스의 constructor를 호출해서 instance를 만든다.
                // Object를 감싸는 Person이라는 instance를 만든다.
  public Person() {
    super();
    System.out.println("AA"); // 상위 클래스가 먼저 호출된다.
  }

  String name;
  String mobile;
}

class Student extends Person {
  public Student() {
    super();
    System.out.println("BB");
  }

  String dept;
}

public class Main {
  public static void main(String[] args) {
//    Person p = new Person();
    Student s = new Student();
//    Person s = new Student();
//    Object s = new Student();
    // is-a relationship
    // subclass is a superclass
  }
}
```

```
AA
BB
```

Method overiding (메소드 재정의)

this → 현재 사용하는 객체의 reference

super → 현재 사용하는 객체의 reference (type이 상위 type)

super ( ) → 상위 클래스의 생성자 호출

this ( ) → 자신의 클래스가 가진 다른 생성자를 호출

```
class Person {
  String name;
  String mobile;

  public void printAll() {
    System.out.println("모두 출력!");
  }
}

class Student extends Person {

  String dept;

  public void printAll() {
    System.out.println("오버라이딩!");
  }
}

public class Main {
  public static void main(String[] args) {
    Student s = new Student();

  }
}
```

Q. Inheritance 연습문제

```
package Inheritance;

class Superclass {

  // fields
  int a = staticCall("1번입니다.");
  static int b = staticCall("2번입니다.");

  // static method
  static int staticCall(String msg) {
    System.out.println(msg);
    return 100;
  }

  // constructor
  public Superclass() {
    staticCall("3번입니다.");
  }

  public Superclass(int i) {
    this();
    staticCall("4번입니다.");
  }
```

```
  // method

  public void myFunc() {
    System.out.println("5번입니다.");
  }
}

public class InheritanceTest extends Superclass {
  // fields
  int c = staticCall("6번입니다.");
  static int d = staticCall("7번입니다.");

  // constructor
  public InheritanceTest() {
    super(100);
    staticCall("8번입니다.");
    super.myFunc();
  }

  @Override
  public void myFunc() {
    System.out.println("9번입니다.");
  }

  public static void main(String[] args) {
    System.out.println("10번입니다.");
    Superclass obj = new InheritanceTest();
    obj.myFunc();
  }
}
```

A. 정답.

```
2번입니다.
7번입니다.
10번입니다.
1번입니다.
3번입니다.
4번입니다.
6번입니다.
8번입니다.
5번입니다.
9번입니다.
```

풀이

Inheritance 클래스를 올릴려면 자신의 상위 클래스인 super클래스 부터 메소드 영역에 올라간다

메소드 영역에 static b의 공간이 만들어진다. 바로 스테틱 메소드가 호출이 되면서 2번입니다가 출력

나머지는 아직 수행이 될 수가 없다. 메소드 영역에 클래스에대한 정보만 올라가는 단계이므로,

이제 Inheritance에 대한 정보를 올린다. 그래서 7번입니다가 출력된다.

main메소드가 호출 그래서 10번입니다가 출력

superclass obj = new InheritanceTest(); 인스턴스 만들어서 생성자 호출하러 간다

public InheritanceTest()로 간다

super(100) = 상위 클래스의 컨스트럭터를 호출 숫자를 받는놈 인자를 안받는놈

숫자를 받는 컨스트럭터를 호출 this(); → 내가 가지고 있는 다른 컨스트럭터를 호출하래

즉 인자를 안받는 컨스트럭터 호출 즉 3번이 찍힐것같은데? 생성자가 만들어지기 전에 필드에 대한 공간이 먼저 만들어져야 초기화가 가능하기때문에 int a = staticcall (1번입니다)가 출력된다.

그다음 3번 4번 여기까지는 오케이 instance영역 호출이 되어서 c라는 공간이 만들어지고 6번호출

그다음 8번호출 그다음 super.myfunc기 때문에 상위 클래스에 myfunc 5번이 출력

상위 클래스 타입 obj???? 이건 뭐지??

다시한번 5번이 찍힐것같은데 객체에 대한 타입이 상위 타입이라 할지라도 만약 오버라이딩된 메서드가 하위에 존재한다면 메서드는 무조건 오버라이딩된 메소드를 사용한다.

→ 이것을 동적 바인딩이라고 얘기한다(Daynamic binding)

하지만 필드에 대해서는 그렇지 않다. 메서드에 대해서만 적용

---

final

- **변수 앞에 붙으면 ⇒ 상수처리 (final int k = 100;)**
- **class 앞에 붙으면 ⇒ final class A { } → 상속이 안된다.**
- **method 앞에 붙으면 ⇒ final void kk() { } → overriding 금지**

추상 Class (abstract class)

💡 **추상 메소드라고 불리는 abstract method가 단 1개라도 Class 내부에 존재하면 추상 클래스가 된다.**
→ method의 선언만 존재 정의가 없다. method가 하는일이 정해지지 않았어요!!
→ instance를 생성할 수 없다!!

```
package Abstract;

// 추상클래스 (abstract class)
public abstract class UpperClass {
  String name;
  int age;

  // method
  public abstract void printAll(); // abstract method
                                   // 정의만 존재한다.
}

class subClass extends UpperClass {
  @Override
  public void printAll() {
```

```
    // TODO Auto-generated method stub

  }
}
```

추상Class의 아주 특별한 형태

- **모든 method가 추상 method**
- **모든 field는 public static final**
  - **이런 class를 interface라고 부른다.**

```
interface myInterface }
// field
// public static final

int kk = 0;
String aa = "Hello";

// methods
public abstract void printAll();
public abstract void myPrint();
}

public class MyClass implements myInterface {

  @Override
  public void printAll() {
    // TODO Auto-generated method stub

  }

  @Override
  public void myPrint() {
    // TODO Auto-generated method stub

  }

}
```