DAY 1

JAVA language 특징

OS에 독립적 ⇒ JVM (JAVA virtual machine)

- C언어 → compile (machine code로 변환)
- Javascript → interpret (해석)
- 소스코드 → 중간코드(사람,OS도 모름 효율성을 위해) → Machine Code
- JAVA = 소스코드 → 바이트코드(JAVAC에 의해 변환) →
- . JAVA → . Class → JVM이 한줄한줄 실행 (interpret)
- 플랫폼에 독립적!

객체지향 언어 (JAVA)

1. **Hardware, Memory에 대한 직접적인 제어가 안된다.**
    a. garbage collection에 의해서 메모리가 자동 관리!
2. Network와 분산처리하기에 API가 잘 되어있다(thread)
3. Dynamic loading제공!! → class의 로딩이 동적으로 일어난다.
    a. 모든 것을 다 로딩하는 것이 아니라 클래스가 사용될 때 동적으로 메모리에 로딩한다.
4. 해결해야 하는 문제의 구성요소를 파악, 데이터가 어떻게 전달되는지 파악
5. 분석 설계가 어렵다.
6. 유지보수에 강점을 가진다.

구성요소 파악

- 상태:
  - 이름
  - 학번
  - 학과
  - **변수로 표현 (field)**
- 행위(Process; 절차):
  - 등교한다
  - 출석한다
  - **함수로 표현 (method)**

위와 같이 학생이라는 개념을 **단순화** 시킨다. → **Abstraction** 작업을 한다. → 모델링(**Modeling**)

- 모델링을 하려면 약속된 도구가 있어야 하는데 그 도구가 바로 **클래스(class)**이다.

> 💡 따라서 **class**는 객체모델링의 수단이라고 한다.

구조적,절차적 언어 (C)

- 기능으로 프로그램을 세분화
- 빠른 분석과 설계 → 빠르게 프로그램을 작성
- 유지보수에 취약

Source Code를 생성

- source code의 이름은 class의 이름과 같아야 해요!!

→ **정확히 말하면 public class의 이름으로 file명을 지어요**

- public class가 없는 경우는 일반 class의 이름을 사용해요
- 하나의 Java file안에 class는 여러개 나올 수 있어요

→ **단, public class는 하나의 file에 2개이상 존재할 수 없어요**

- program의 entry point (시작지점)

→ **public static void main(String args[ ]) { }**

Language spec

정적 Type언어라서 값 뿐만 아니라 변수에도 Data type이 붙어요!!

▼ Primitive type

byte (1 byte)

short (2 byte)

int (4 byte) → 가장 많이 사용!!

long (8 byte)

→ 정수형

---

float (4 byte)

double (8 byte)

→ 실수형

---

char (2 byte, Unicode 사용) : 한 글자 표현할 때 무조건 2 byte 사용

boolean (1 byte)

▼ Reference type

우리가 만들거나, 제공되는 모든것이 **class (ADT)**

```
public class Student {
  // fields (변수들)

  // methods (함수들)
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
```

상수

= 값에 재할당이 되지 않는 것

- Java에서는 `final` 이라는 키워드를 쓴다.
  - `final int SCORE = 100;` 과 같이쓴다.
  - `final int MY_SCORE = 100;` 과 같이 두 개의 단어가 합쳐져 있다면 전부 대문자 + `_` (snake case)를 이용한다.
- **선언과 동시에 초기화**가 돼야한다.

▼ 형변환 (type casting)

( )를 이용해서 형변환을 한다

묵시적 형변환 → implicit type casting

명시적 형변환 → explicit type casting

Constructor(생성자)

```
package com.kakao.test;

public class Student {
  // fields (변수들)
  String stuName; // String(문자열) -> Class := reference type
  String stuNum; // 연산에 쓰이지 않을 것이라면 문자열로 선언하는게 좋다.
  // methods (함수들)
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
```

String 이라는 클래스가 있는데 원래는 new를 이용해서 쓰지만, 워낙 많이 쓰기 때문에 primitive type처럼 이용할 수 있게 해놨다.

```
String stuName; // 이름
String stuNum; //학번
```

또한, 학번과 같이 연산에 사용하지 않는 숫자들은 문자열로 처리하는것이 편하다.

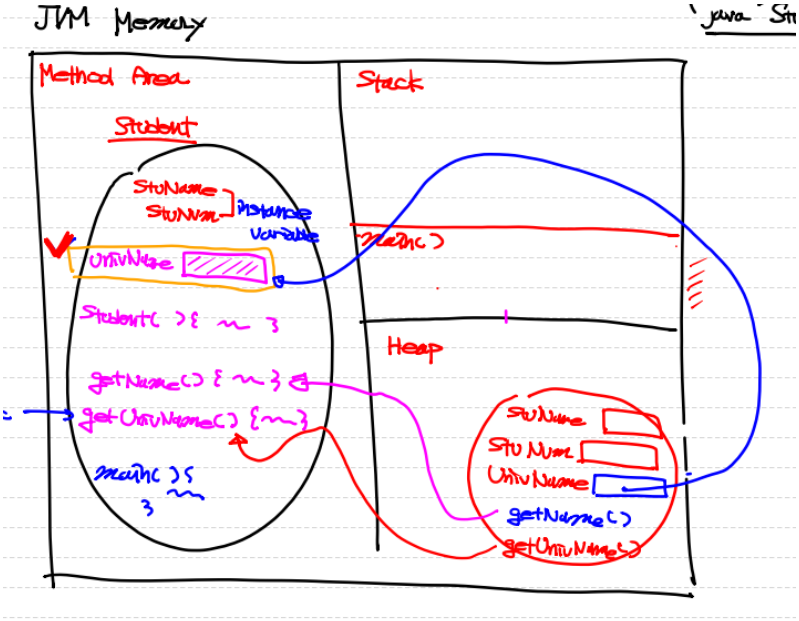
```
package com.kakao.test;

public class Student {
  // fields (변수들)
  String stuName; // String(문자열) -> Class := reference type

  // methods (함수들)
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
```

- **생성자는 클래스 이름과 동일해야 한다.**
- **method는 무조건 리턴 타입이 있어야 하지만, 생성자는 리턴 타입이 존재하지 않는다.**
- **인스턴스의 초기화를 담당한다.**

JVM이 관리하는 Memory구조



- **Register** : 프로그램의 실행 포인터를 관리 (JVM에 의해서만 관리)

- **Runtime Constant Pool** : 정해져있는 만들어지는 상수값들이 여기에서 관리
- **Method Area** : class 자체에 대한 정보, class를 처음 사용하는 시점에 Memory에 올라간다.
- **Heap** : new keyword로 생성되는 모든 instance가 저장
- **Call Stack** : method가 호출되면 method를 위한 공간이 형성된다.

```
package com.kakao.zemni;

public class Student {

  // fields (변수들)
  String stuNmae; // String -> Class -> reference type -> 문자열
  String stuNum; // 학번 -> 연산을 하지 않는 숫자는 문자열 처리가 좋다!!
  int stuAge; // 나이 -> 숫자
  String stuDept; // 학과 -> 문자열

  // 생성자 (클래스 이름과 완전히 동일해야 한다.)
  // 인스턴스를 초기화 해주는 역할
  // default constructor
  Student() {

  }

  // methods (함수들)
  public static void main(String[] args) {

    Student stu = new Student(); // instance를 생성
    // . operator ( . 연산자)
    stu.stuNmae = "홍길동";

  }

}
```

```
package com.kakao.zemni;

public class Student {

  static int a = staticCall("1번");
  int b = staticCall("2번");

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

  public Student() {
    this.b = staticCall("3번");

  }

  public static void main(String[] args) {
    System.out.println("4번");
    int c = staticCall("5번");
    Student s = new Student();
  }
}
```

field는 특별한 이유가 없는한 싹 다 private

class 내부의 field는 보호해야 하는 정보.

method는 행위를 하는 작업이기 때문에 특별한 이유가 없는 한

외부에서 사용할 수 있도록 public으로 설정.

단일 class의 기본적인 형태 (package로 묶여있다)

- private field
- static field
- constructor 여러개 (constructor overloading)
- 일반적인 비즈니스 메소드는 public설정
- priviate field에 대한 getter/setter
- main method (optional)

연습문제

- Class 명 : Account
  - 계좌를 관리하기 위해서 클래스가 필요하며, Account라는 이름의 class를 만든다.
  - 필요로 하는 정보로는 예금주, 계좌번호, 잔액이 있고, 입금(deposit)과 출금(withdraw)할 수 있고, 현 잔액에 대한 조회가 가능하다.

- Account class를 이용하여 다음과 같은 실행 결과가 나오도록 필드에 값을 저장하고, 적절한 method를 이용하여 출력한다.

실행 결과

```
계좌 123-456789 ( 예금주 : 홍길동 )
잔액 : 10000 원
20000 원 입금합니다.
잔액 : 30000 원
45000 원 출금합니다.
잔액 : -15000 원
```

A.

```
package com.kakao.test;

public class Account {
  private String personName;
  private String accountNumber;
  private int balance;

  public void Deposit(int money) {
    this.balance += money;
    System.out.println(money + "원 입금합니다.");
    System.out.println("잔액 : " + balance + "원");
  }

  public void withDraw(int money) {
    this.balance -= money;
    System.out.println(money + "원 출금합니다.");
    System.out.println("잔액 : " + balance + "원");

  }

  public String getPersonName() {
    return personName;
  }

  public void setPersonName(String personName) {
    this.personName = personName;
  }

  public String getAccountNumber() {
    return accountNumber;
  }

  public void setAccountNumber(String accountNumber) {
    this.accountNumber = accountNumber;
  }

  public int getBalance() {
    System.out.println("잔액 : " + this.balance + "원");
    return balance;
  }
```

```
  public void setBalance(int balance) {
    this.balance = balance;

  }
}
```

```
package com.kakao.test;

public class AccountTest {

  public static void main(String[] args) {
    Account account = new Account();
    account.setAccountNumber("123-45678");
    account.setPersonName("홍길동");
    System.out.println("계좌 " + account.getAccountNumber() + "( 예금주 : " + account.getPersonName() + " )");
    account.setBalance(10000);
    account.getBalance();
    account.Deposit(20000);
    account.withDraw(45000);
  }

}
```