[12/18]

1) 클래스(Class)

```java
class Animal {
    1 usage
    String name; //객체 변수(instance variable)
}
public class Sample {
    public static void main(String[] args) {
        Animal any = new Animal();
        System.out.println(any.name); //null
    }
}
```

- Class Animal : 클래스
  - 이 클래스에 의해 생성되는 것은 객체
  - 그리고 그 클래스에 선언된 변수는 객체 변수

- String name; : 객체 변수
  - 객체 변수 또한 변수이므로 값을 대입할 수 있다.

- Animal any = new Animal( ); : 객체 생성
  - 앞서 Class Animal 로 클래스가 선언된 상황
  - Class Animal 의 인스턴스 : Class Animal 에 의해 Animal any 라는 객체 생성 가능
    - Animal any = new Animal( );
    - Animal bny = new Animal( );
    - Animal cny = new Animal( );
      - 이런식으로 1개이상의 인스턴스를 생성 가능

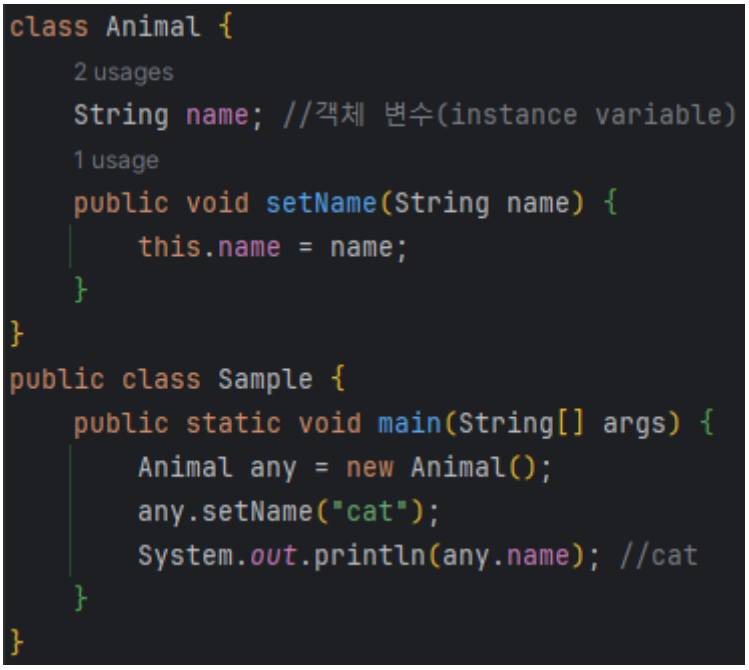
- any.name : 객체 변수에 접근
  - any 라는 객체의 .name 을 찾는 것

2) 메소드(Method) : 클래스 내에 생성될 수 있는 함수의 선언

다른 프로그래밍 언어에는 함수가 별도로 존재한다. 하지만 자바는 클래스를 떠나 존재하는 것은 있을 수 없기 때문에 자바의 함수는 따로 존재하지 않고 클래스 내에 존재한다. 자바는 이러한 클래스 내의 함수를 메서드(method)라고 부른다.

// 파이썬에서는 편하게 어디서나 함수를 선언했지만, 자바에서는 클래스내에서만 함수를 생성할 수 있고, 함수라는 단어는 사용하지 않고 메서드라고 칭한다.

2-1) 메서드를 이용하여 Animal 클래스의 객체 변수인 name에 값을 대입하기 위해 setName 메서드를 추가



```
class Animal {
    2 usages
    String name; //객체 변수(instance variable)
    1 usage
    public void setName(String name) {
        this.name = name;
    }
}
public class Sample {
    public static void main(String[] args) {
        Animal any = new Animal();
        any.setName("cat");
        System.out.println(any.name); //cat
    }
}
```

- public void setName(String name)
    - Animal 클래스에 추가된 setName 메서드는 다음과 같은 형태의 메서드이다.
        - 입력: String name
        - 출력: void (‘리턴값 없음’을 의미)
    - 즉, 입력으로 name이라는 문자열을 받고 출력은 없는 형태의 메서드이다.
    - 해당 메서드를 통해 name을 지정시 name 저장한다.

- any.setName(“cat”) : 객체의 name 에 (“ “)의 내용을 저장
    - 객체.setName 에 (“ “) 을 저장

3) 매개 변수(parameter)와 인수(arguments)

- 매개 변수는 메서드에 전달된 입력값을 저장하는 변수를 의미
- 인수는 메서드를 호출할 때 전달하는 입력값을 의미

```
public class Sample {
    int sum(int a, int b) {  // a, b 는 매개변수
        return a+b;
    }

    public static void main(String[] args) {
        Sample sample = new Sample();
        int c = sample.sum(3, 4);  // 3, 4는 인수

        System.out.println(c);
    }
}
```

4) 메서드 더 살펴보기

4-1) 메서드는 입출력 유무에 따라 다음과 같이 4가지로 분류할 수 있다.

- 입력과 출력이 모두 있는 메서드 : 일반적인 메서드는 입력값과 리턴값이 모두 있다.

```
int sum(int a, int b) {
    return a+b;
}
```

  - 입력값 : int 자료형 a, int 자료형 b
  - 리턴값 : int 자료형

```
Sample sample = new Sample();
int result = sample.sum(3, 4);
```

  - sum 메소드를 사용
  - return 이란 변수는 sample의 sum 메서드에 3,4를 전달, return 값을 받음

- 입력은 없고 출력은 있는 메서드 :입력 인수가 없을 경우에는 괄호 안을 비워 놓으면 된다.

```
String say() {
    return "Hi";
}
```

  - 입력값 : 없음
  - 리턴값 : String 자료형
  - System.out.println(sample.say); //”Hi” 출력

- 입력은 있고 출력은 없는 메서드 : 리턴값이 없는 메서드는 명시적으로 리턴 자료형 부분에 void라고 표기, 리턴값은 오직 return 명령어로만 반환이 가능

```
public class Sample {
    void sum(int a, int b) {
        System.out.println(a+"과 "+b+"의 합은 "+(a+b)+"입니다.");
    }

    public static void main(String[] args) {
        Sample sample = new Sample();
        sample.sum(3, 4);
    }
}
```

  - 입력값 : int 자료형 a, int 자료형 b
  - 리턴값 : void (리턴값 없음)

- 입력과 출력이 모두 없는 메서드

```
public class Sample {
    void say() {
        System.out.println("Hi");
    }

    public static void main(String[] args) {
        Sample sample = new Sample();
        sample.say();
    }
}
```

  - 입력값 : 없음
  - 리턴값 : void

5) 값에 의한 호출과 객체에 의한 호출

- 메서드에 값(원시 자료형)을 전달하는 것과 객체를 전달하는 것에는 큰 차이가 있다. 메서드에 객체를 전달할 경우 메서드에서 객체 변수의 값을 변경할 수 있다.

5-1) 값에 의한 호출

```
class Updater {
    void update(int count) {
        count++;
    }
}

class Counter {
    int count = 0;  // 객체변수
}

public class Sample {
    public static void main(String[] args) {
        Counter myCounter = new Counter();
        System.out.println("before update:"+myCounter.count);
        Updater myUpdater = new Updater();
        myUpdater.update(myCounter.count);
        System.out.println("after update:"+myCounter.count);
    }
}
```

- Updater 클래스는 전달받은 숫자를 1만큼 증가시키는 update라는 메서드를 가지고 있다.
- Counter 클래스는 count라는 객체 변수를 가지고 있다.
- Sample 클래스의 main 메서드는 Counter 클래스에 의해 생성된 myCounter 객체의 객체 변수인 count값을 Updater 클래스를 이용하여 증가시키고자 한다.
- 실행시 : before update:0 와 after update:0 로 나온다

5-2) 객체에 의한 호출

```
class Updater {
    void update(Counter counter) {
        counter.count++;
    }
}

class Counter {
    int count = 0;  // 객체변수
}

public class Sample {
    public static void main(String[] args) {
        Counter myCounter = new Counter();
        System.out.println("before update:"+myCounter.count);
        Updater myUpdater = new Updater();
        myUpdater.update(myCounter);
        System.out.println("after update:"+myCounter.count);
    }
}
```

- 실행시 : before update:0 와 after update:1 로 나온다
- 메서드의 입력으로 객체를 전달하면 메서드가 입력받은 객체를 그대로 사용한다.
- 메서드가 객체의 속성값을 변경하면 메서드 수행 후에도 객체의 변경된 속성값이 유지된다.

6) 상속(inheritance) // CSS 의 상속과 유사, 자식노드에서 선언이 없다면 부모 노드의 속성을 승계 받고, 자식노드에서 선언 한 것이 있다면 자식 노드의 속성을 우선,

- extends : 해당 클래스를 상속하며, 코드를 중복 작성하지 않고 해당 속성을 승계받는다.

```
1 usage  1 inheritor
class Animal {
    2 usages
    String name;
    1 usage
    void setName(String name) {
        this.name = name;
    }
}
2 usages
class Dog extends Animal {  // Animal 클래스를 상속한다.
}

public class Sample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.setName("poppy");
        System.out.println(dog.name);
    }
}
```

6-1) 상속 및 기능 추가

- sleep 메서드를 추가하여 이제 Dog 클래스는 Animal 클래스보다 좀 더 많은 기능을 가지게 되었다.
- 보통 부모 클래스를 상속받은 자식 클래스는 부모 클래스의 기능에 더하여 좀 더 많은 기능을 갖도록 작성할 수 있다.

```
class Dog extends Animal {
    no usages
    void sleep() {
        System.out.println(this.name+" zzz");
    }
}
```

6-2) IS-A

- IS : 하위, 자식 클래스
- A : 상위, 부모 클래스를 의미

7) Method Overriding : 입력 항목이 다른 경우 동일한 이름의 메소드 생성 가능

- HouseDog 클래스 생성

```
1 usage  2 inheritors
class Animal {
    2 usages
    String name;
    1 usage
    void setName(String name) {
        this.name = name;
    }
}

1 usage  1 inheritor
class Dog extends Animal {
    1 usage
    void sleep() {
        System.out.println(this.name+" zzz");
    }
}

2 usages
class HouseDog extends Dog {
}

public class Sample {
    public static void main(String[] args) {
        HouseDog houseDog = new HouseDog();
        houseDog.setName("happy");
        houseDog.sleep();  // happy zzz 출력
    }
}
```

HouseDog 는 Dog 의 자식 클래스이며, Dog 는 Animal 의 자식 클래스이다.

- sleep이라는 메서드가 있지만 동일한 이름의 sleep 메서드를 또 생성할 수 있다.

```
2 usages
class HouseDog extends Dog {
    1 usage
    void sleep() {
        System.out.println(this.name + " zzz in house");
    }
    no usages
    void sleep(int hour) {
        System.out.println(this.name + " zzz in house for " + hour + " hours");
    }
}
```

새로 만든 sleep 메서드는 입력 항목으로 hour라는 int 자료형이 추가

8) 다중상속 : 하나 이상이 클래스를 상속 받는 것, 자바는 다중 상속을 미지원

- 자바에서 아래 코드는 실행되지 않는다.
- 파이썬과 같은 다중 상속을 지원하며, 상속받는 우선순위를 정하는 규칙이 있다.

```
class A {
    public void msg() {
        System.out.println("A message");
    }
}

class B {
    public void msg() {
        System.out.println("B message");
    }
}

class C extends A, B {
    public void static main(String[] args) {
        C test = new C();
        test.msg();
    }
}
```

9) 생성자(constructor)

- this 생성자 : 메서드명이 클래스명과 동일하고 리턴 자료형을 정의하지 않는 메서드를 생성자라고 한다.

```
void setName(String name) {
    this.name = name;
}
```

  - 클래스명과 메서드명이 같다.
  - 리턴 타입을 정의하지 않는다(void도 사용하지 않는다.)

- new 생성자 : 생성자는 객체가 생성될 때 호출된다. 즉, 생성자는 다음과 같이 new 키워드가 사용될 때 호출된다.

```
new 클래스명(입력인수, ...)
```

9-1) 생성자 규칙 준수

- this 생성자를 setName 메소드를 통해 지정했다면, sample은 아래와 같다.

```
class Animal {
    3 usages
    String name; //String name="";
    2 usages
    void setName(String name) {
        this.name = name;
    }
}
```

```
public class Sample {
    public static void main(String[] args) {
        Animal cat = new Animal();
        cat.setName("baby");
        System.out.println(cat.name);
    }
}
```

- 만약 this 생성자를 다음과 같이 작성했다면, sample 클래스도 변경해야 한다.

```
class Animal {
    2 usages
    String name;
    1 usage
    Animal(String name) {
        this.name = name;
    }
}
```

```
public class Sample {
    public static void main(String[] args) {
        Animal dog = new Animal( name: "happy");
        System.out.println(dog.name);
    }
}
```

  - 이 경우 Animal 은 return 할게 생기므로 메소드 앞에 void 가 빠져야 한다.

9-2)디폴트 생성자(default constructor) : name(){ } 의 형태로 입력항목과 내부에 아무 내용이 없는 상태

- 만약 클래스에 생성자가 하나도 없다면 컴파일러는 자동으로 이와 같은 디폴트 생성자를 추가한다. 하지만 사용자가 작성한 생성자가 하나라도 구현되어 있다면 컴파일러는 디폴트 생성자를 추가하지 않는다.

9-3) 생성자 오버로딩(constructor overloading) : 하나이상의 다른 타입으로 생성자가 생성될 수 있다.

- 메소드의 오버로딩과 동일한 개념

10) 인터페이스(interface) : 클래스의 인스턴스와 유사하며, 별도의 기능을 추가로 수행

- interface 선언 : class 선언과 유사
- implements 인터페이스명 : 상속형태의 끝에 작성
    - **class Tiger extends Animal implements Predator {**
- 인터페이스는 USB 포트와 유사하다. USB 포트의 규격만 알면 어떤 기기도 연결할 수 있다. 또, 컴퓨터는 USB 포트만 제공하고 어떤 기기가 연결되는지 신경 쓸 필요가 없다. 바로 이 점이 자바의 인터페이스와 매우 비슷하다.

10-1) 인터페이스의 메서드

- 규칙 : 인터페이스의 메서드는 메서드의 이름과 입출력에 대한 정의만 있고 그 내용은 없다.

```
interface Predator {
    String getFood();
}
```

```
class Tiger extends Animal implements Predator {
    public String getFood() {
        return "apple";
    }
}
```

메서드를 바꿨으면 리턴호출도 그에 맞게 작성해야 한다

10-2) 상속과 인터페이스

- 상속은 자식 클래스가 부모 클래스의 메서드를 오버라이딩하지 않고 사용할 수 있기 때문에 해당 메서드를 반드시 구현해야 한다는 '강제성'을 갖지 못한다. 그래서 상황에 맞게 상속을 사용할 것인지, 인터페이스를 사용해야 할지를 결정해야 한다.
- 인터페이스는 인터페이스의 메서드를 반드시 구현해야 하는 강제성을 갖는다는 점을 반드시 기억하자.

10-3) 디폴트 메소드(자바 8 버전 이후부터)

- 디폴트 메서드를 사용하면 실제 구현된 형태의 메서드를 가질 수 있다.
- 디폴트 메서드는 메서드명 가장 앞에 default라고 표기해야 한다.
- Predator 인터페이스에 printFood 디폴트 메서드를 구현하면 Predator 인터페이스를 구현한 Tiger, Lion 등의 실제 클래스는 printFood 메서드를 구현하지 않아도 사용할 수 있다.
- 디폴트 메서드는 오버라이딩이 가능하다. 즉, printFood 메서드를 실제 클래스에서 다르게 구현하여 사용할 수 있다.