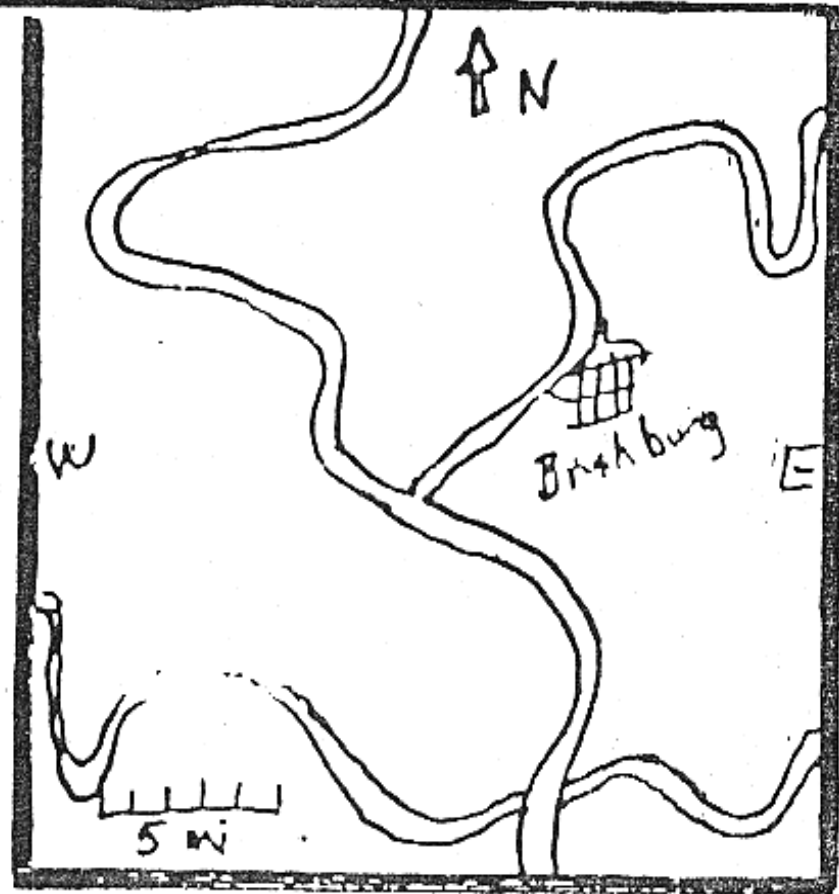


Chap 06. Object–Oriented Programming

1. abstraction
2. 기본 class작성
3. encapsulation
4. constructor
5. package / import



REAL WORLD



IMPERFECT REPRESENTATION

□ Java에서의 Abstraction

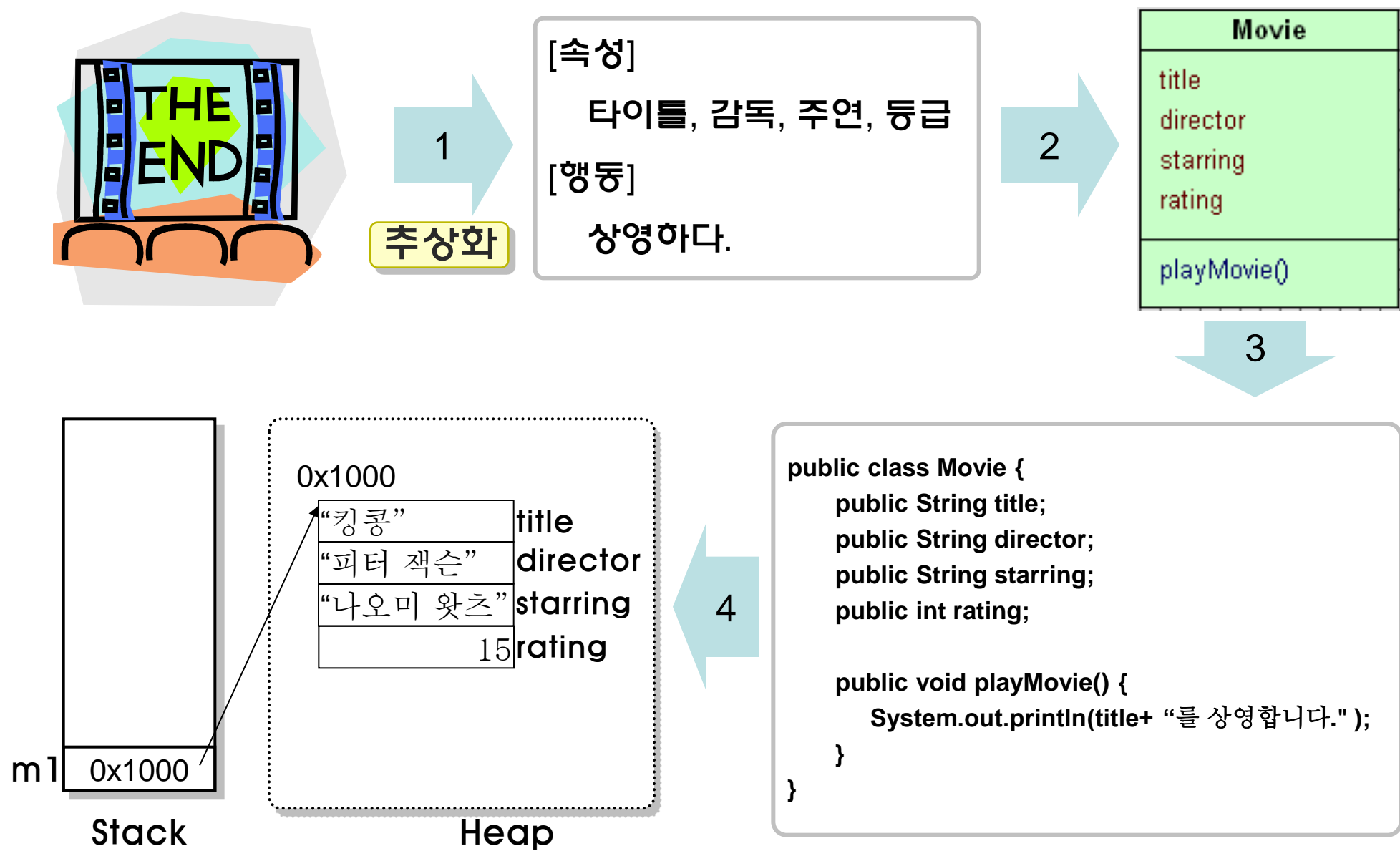
- 실 세계의 객체를 프로그래밍 관점에서 관심의 대상이 되는 속성과 행동을 추출해 내는 것을 말한다.

□ Abstraction의 결과 → Class

- Class : Member Variable + Member Method
- 속성 : Class의 Member Variable
- 행동 : Class의 Member Method

□ Class

- 클래스는 추상화 된 대상이 “이러이러한 속성과 행동을 가진다” 라는 것을 정의한 설계이지, 실제로 값을 가진 객체(object/instance)가 아니다.
- 실제로 값을 갖는 객체(instance)는 new라는 키워드를 이용해서, 클래스 정의에 해당 하는 내용으로 메모리에 할당되어져 이용하게 된다.



□ 클래스 작성하기

패키지 선언

```
{ package chap02.entity;
```

클래스 선언

```
{ public class Movie {
```

멤버변수

```
    public String title;  
    public String director;  
    public String starring;  
    public int rating;
```

멤버메소드

```
    public void setTitle( String title ) {  
        title = newTitle;  
    }  
  
    public void playMovie() {  
        System.out.println( title + “를 상영합니다.”);  
    }  
}
```

□ 클래스 사용하기

패키지 선언

import 선언

main 메소드

```
package chap02.view;
```

```
import chap02.entity.*;
```

```
public class TestMovie {
```

```
    public static void main( String[] args ) {
```

```
        Movie m1 = new Movie();
```

```
        m1.setTitle( “킹콩” );
```

```
        System.out.println( m1.title );
```

```
    }
```

```
}
```

□ package

- 한 자바 소스에서 단 한 번만 쓸 수 있다.
- 관련 있는 클래스를 묶어 주는 역할을 한다.
- .java 파일에서 단 한 번만 사용되어 질 수 있다.

Ex) package shipping.reports;


□ import

- 서로 다른 package에 있는 클래스를 참조하기 위해, 어느 package에 있는 클래스인지 선언해 주는 역할을 한다.
- 한 자바 소스 내에 여러 번 쓸 수 있다.
- 같은 package 클래스를 사용하기 위해서는 필요 없다.

Ex) import shipping.reports.PrintReprot;

import shipping.reports.*;


```
public class Movie {  
    public String title = "킹콩";  
    public String director = "피터 잭슨";  
    public String starring = "나오미 왓츠";  
    public int rating;  
    public void setTitle( String newTitle ) {  
        title = newTitle;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void playMovie() {  
        System.out.println( title + "를 상영합니다.");  
    }  
}
```



① Access Modifier

② Data Type

③ Variable Name

```
public class Movie {  
    public String title = "킹콩";  
    public String director = "피터 잭슨";  
    public String starring = "나오미 왓츠";  
    public int rating = 15;
```

```
    ⑤ {  
        ① ② ③ ④  
        public void setTitle( String newTitle ) {  
            title = newTitle;  
        }  
  
        public String getTitle() {  
            return title;  
        }  
  
        public void playMovie() {  
            System.out.println( title + "를 상영합니다.");  
        }  
    }
```

① Access Modifier

② Return Type

③ Method Name

④ Parameter

⑤ Method Body

```
public class TestMovie {
    public static void main( String[] args ) {

        Movie m1 = new Movie();
        m1.setTitle( "킹콩2" );
        m1.playMovie();
        String title = m1.getTitle();
        System.out.println( "영화 제목은 " +
                           s + "입니다");
    }
}
```

```
public class Movie {
    public String title = "킹콩";
    public String director = "피터 잭슨";
    public String starring = "나오미 왓츠";
    public int rating = 15;

    public void setTitle( String newTitle ) {
        title = newTitle;
    }

    public String getTitle() {
        return title;
    }

    public void playMovie() {
        System.out.println( getTitle() +
                           "를 상영합니다.");
    }
}
```

```
public class TestDog {  
    public static void main( String[] args ) {  
        Dog d = new Dog(); // Dog 객체 생성  
        d.setWeight(42); // setWeight() 호출하며 42 전달  
        System.out.println( "Dog d's weight is" +  
                             d.getWeight() );  
        // getWeight() 통해서 weight 값 얻기  
    }  
}
```

```
public class Dog {  
    private int weight;  
  
    public void setWeight(int newWeight) {  
        weight = newWeight; // 42를 멤버변수인  
                             // weight에 할당  
    }  
  
    public int getWeight() {  
        return weight; // 호출한 쪽으로 weight 리턴  
    }  
}
```

실행 결과 : Dog d's weight is 42

OOP (Object-Oriented Programming) 특징

Encapsulation

Inheritance

Polymorphism

MyDate
+day:int +month:int +year:int

```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```

```
public class MyDateTest {  
    public static void main( String[] args ) {  
        MyDate date = new MyDate();  
        date.day = 32;  
    }  
}
```

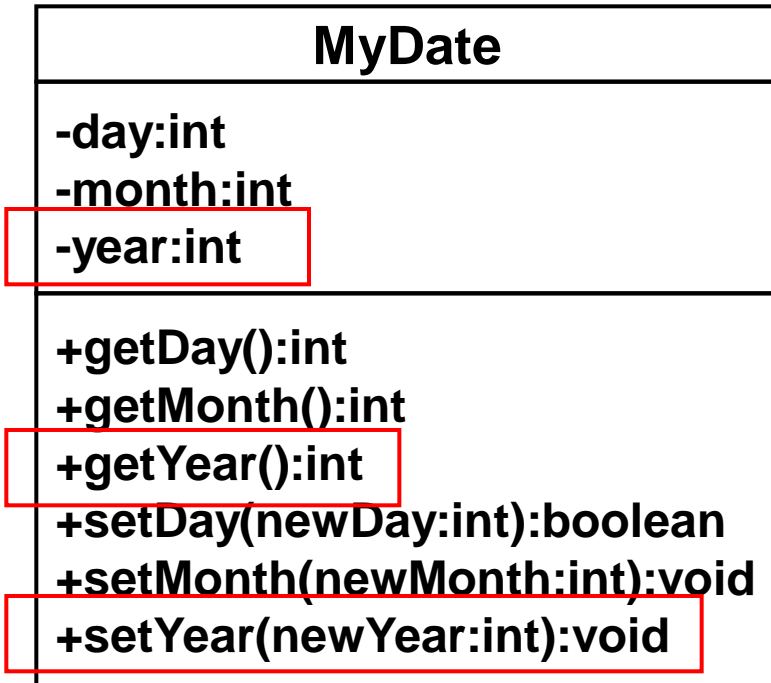
MyDate
-day:int -month:int -year:int
+getDay():int +getMonth():int +getYear():int +setDay(newDay:int):boolean +setMonth(newMonth:int):void +setYear(newYear:int):void

Verify days in month

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public int getDay() {  
        return day;  
    }  
    public boolean setDay( int newDay ) {  
        //월별 해당일이 맞으면, day = newDay;  
        // true return, 아니면 false return  
    }  
    // month, year 관련 메소드 생략  
}
```

```
public class MyDateTest {  
    public static void main( String[] args ) {  
        MyDate date = new MyDate();  
        date.setDay( 31 );  
        date.setDay( date.getDay() + 1 );  
    }  
}
```

- 숨겨진 Data(private 멤버 변수) 에 대한 access는 getter/setter를 이용한다.



□ Member Variable : **xxx**

```
private int year;
```

□ Getter : **getXxx()**

```
public int getYear() {  
    return year;  
}
```

□ Setter : **setXxx()**

```
public void setYear( int newYear ) {  
    year = newYear;  
}
```


MyDate
-day:int -month:int -year:int
+getDay():int +getMonth():int +getYear():int +setDay(newDay:int):void +setMonth(newMonth:int):void +setYear(newYear:int):void -validDay(checkDay:int):void

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public int getDay() {  
        return day;  
    }  
  
    public void setDay( int newDay ) {  
        if ( validDay(newDay) )  
            day = newDay;  
    }  
  
    private boolean validDay( int newDay ) {  
        //각 해당월에 대한 일자가 맞는지 확인  
    }  
  
    // month, year 관련 메소드 생략  
}
```

□ Information hiding

- Class 외부에서 주요 정보에 접근하지 못 하도록 숨기는 것
- 멤버변수는 private로 선언하여, 외부에서 직접 참조하지 못하도록
- 멤버변수는 getter/setter를 이용하여 참조

□ Encapsulation

- Class 구현에 대한 상세내용을 숨기는 것
- Class사용자는 Class의 내부 구현에는 신경 쓸 필요 없이, 제공되는 Interface(method) 를 이용하기만 하면 됨
- 내부 로직은 private로 선언하여, 외부에서 보이지 않게 한다. 즉, 변화에 관계 없이 외부에서는 Interface만을 이용하므로, 사용자 관점에서는 유지보수가 편해 진다.
- Encapsulation은 Data보호 뿐 아니라, Class의 세부 구현도 숨긴다는 점에서 Information Hiding의 확장이라고 볼 수 있다.

- 역할 : 주로 멤버변수를 초기화 하는데 쓰인다.
- 특징
 - 1) 클래스 이름과 같다.
 - 2) Return 타입이 없다.
 - 3) 상속되지 않는다.
 - 4) 객체 생성시 new라는 키워드를 만났을 때 호출된다.

```
public class Dog {  
  
    private int weight;  
    public Dog() {  
        weight = 42;  
    }  
    ....  
}
```

```
public class MyDate {
    private int day;
    private int month;
    private int year;

    public MyDate( int newDay, int newMonth, int newYear) {
        day = newDay;
        month = newMonth;
        year = newYear
    }
    // 메소드 생략
}
```

```
public class MyDateTest {
    public static void main( String[] args ) {
        MyDate d1 = new MyDate( 14, 2, 2006 );
        d1.setDay( 15 );
        System.out.println( m1.getDay() );
    }
}
```

```
public class Movie {  
    public String title;  
    public String director;  
    public String starring;  
    public int rating;  
  
    // 메소드 생략  
}
```



```
public class Movie {  
    public String title;  
    public String director;  
    public String starring;  
    public int rating;  
  
    public Movie() {  
    }  
  
    // 메소드 생략  
}
```


- ❑ 모든 클래스는 최소한 하나의 constructor가 있다.
- ❑ 만약, 아무런 constructor를 정의 하지 않았다면, 자동적으로 default constructor가 생성된다.
- ❑ default constructor는 아무런 argument가 없고, body에 내용이 없는 constructor를 말한다.

- 만약 constructor가 하나라도 있다면, default constructor는 자동 생성이 안 된다.

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public MyDate( int newDay, int newMonth, int newYear) {  
        day = newDay;  
        month = newMonth;  
        year = newYear  
    }  
    // 메소드 생략  
}
```

```
public class MyDateTest {  
    public static void main( String[] args ) {  
        MyDate d1 = new MyDate(); //error  
    }  
}
```

□ 1개의 소스 파일에서



```
package chap02.entity; // 1번만 올 수 있다.  
import chap02.test.Account; // 필요에 따라 여러 개 가능  
import java.util.*;  
  
public class Test { // 단 하나의 public class만 올 수 있으며,  
    // 파일 이름(.java)과 같아야 한다.  
  
}
```