클래스와 객체

객체지향언어의 역사

- 과학, 군사적 모의실험(simulation)을 위해 컴퓨터를 이용한 가상세계를 구현하려는 노력으로부터 객체지향이론이 시작됨
- 1980년대 절차방식의 프로그래밍의 한계를 객체지향방식으로 극복하려고 노력함.(C++, Smalltalk과 같은 보다 발전된 객체지향언어가 탄생)
- 1995년 말 Java탄생. 객체지향언어가 프로그래밍 언어의 주류가 됨

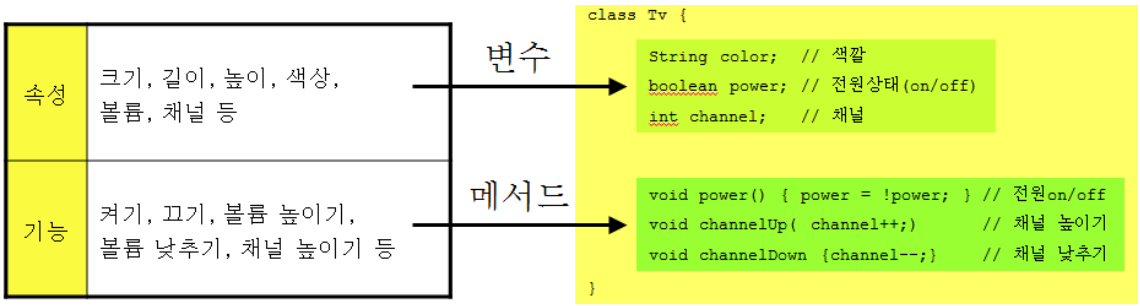
클래스의 정의 – 클래스란 객체를 정의해 놓은 것이다.
클래스의 용도 – 클래스는 객체를 생성하는데 사용된다.

객체의 정의 – 실제로 존재하는 것. 사물 또는 개념

객체 또는 인스턴스

- 객체(object)는 인스턴스(instance)를 포함하는 일반적인 의미

객체의 구성요소 – 속성과 기능



인스턴스의 생성방법

클래스명 참조변수명 = new 클래스명();

예) Tv t = new Tv();

예제실습1

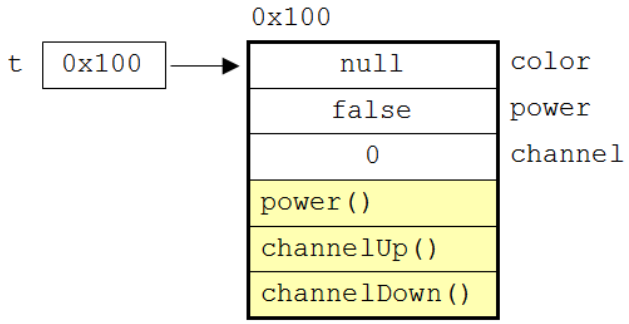
Tv 클래스를 만들고 인스턴스(객체)를 구현하시오.

```
class Tv {
    // Tv의 속성(멤버변수)
    String color;            // 색상
```

```
        boolean power;              // 전원상태(on/off)
        int channel;                // 채널

        // Tv의 기능(메서드)
        void power() {    power = !power; }    /* TV를 켜거나 끄는 기능을 하는 메서드 */
        void channelUp() {      ++channel; }    /* TV의 채널을 높이는 기능을 하는 메서드 */
        void channelDown() {      --channel; } /* TV의 채널을 낮추는 기능을 하는 메서드 */
}

class TvTest {
        public static void main(String args[]) {
                Tv t;                       // Tv인스턴스를 참조하기 위한 변수 t를 선언
                t = new Tv();               // Tv인스턴스를 생성한다.
                t.channel = 7;              // Tv인스턴스의 멤버변수 channel의 값을 7로 한다.
                t.channelDown();            // Tv인스턴스의 메서드 channelDown()을 호출한다.
                System.out.println("현재 채널은 " + t.channel + " 입니다.");
        }
}
```



예제실습2

Car 클래스를 만들고 자동차 1대의 객체를 만들어 출력하시오

```
class Car {
    String name;
    int speed;

    void show() {
        System.out.println("이름: " + name);
        System.out.println("속도: " + speed);
    }
}

class TvTest {
        public static void main(String args[]) {
            Car t = new Car();
            t.name = "BMW";
            t.speed = 200;

            t.show();
        }
}
```

오버로딩

한 클래스 내에 이름을 같지만 매개변수의 타임이나 개수가 다르면 여러 개의 메서드를 중복 작성할 수 있다. 이것을 메서드 오버로딩 이라고 한다. 알겠지만, C언어는 함수 이름이 동일하면 에러로 처리한다.

예를 들어 보면 아래 3개의 getSum(…) 메서드는 이름은 같지만 매개변수가 서로 다르다.

실습해 보세요.

```
class MethodSample {
	int getSum(int i, int j) {
		return i+j;
	}

	int getSum(int i, int j, int k) {
		return i+j+k;
	}

	double getSum(double i, double j) {
		return i+j;
	}
}

class Test {
	public static void main(String[] args) {
		MethodSample a = new MethodSample();

		int i = a.getSum(1,   2);
		int j = a.getSum(1,   2,   3);
		double k = a.getSum(1.1, 2.3);

		System.out.println("i: " + i);
		System.out.println("j: " + j);
		System.out.println("k: " + k);
	}
}
```

```
i: 3
j: 6
k: 3.4
```

this

자바의 중요한 키워드로서 객체 자신을 가리킨다. 다시말해, 자신이 속한 멤버변수나 메서드를 지칭한다.

생성자

객체가 생성될 때 멤버변수의 초기화를 위해 실행되는 메서드이다. 반드시 클래스 이름과 동일해야 한다. 만약 프로그래머가 생성자를 만들지 않으면 컴파일러가 자동으로 기본 생성자를 만든다.

생성자의 조건

생성자의 이름은 클래스의 이름과 같아야 한다. 생성자는 리턴값이 없다. (하지만 void를 쓰지 않는다.)

모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.

실습예제3

클래스 Book을 만들고 생성자를 이용하여 객체를 생성하시오.

```
public class Book {
        String title;
        String author;
        int ISBN;

        public Book(String title, String author, int ISBN) { // 생성자
                this.title = title;
                this.author = author;
                this.ISBN = ISBN;
        }

        Public Book() {}      // 기본 생성자

        void show() {
                System.out.println("책이름: " + title);
                System.out.println("저자: " + author);
                System.out.println("ISBN: " + ISBN);
        }

        public static void main(String [] args) {
                Book javaBook = new Book("Java JDK", "황기태", 3333); // 생성자 호출
                javaBook.show();
        }
}
```

```
책이름: Java JDK
저자: 황기태
ISBN: 3333
```

생성자(constructor)도 method 이기 때문에 오버로딩이 가능하다

default 생성자

아래 코드에는 생성자가 없다. 프로그래머가 생성자를 만들지 않으면 컴파일러가 자동으로 기본 생성자를 만든다. 따라서, 컴파일 오류가 발생하지 않는다.

```
public class DefaultConstructor {
        int x;
        public void setX(int x) {this.x = x;}
        public int getX() {return x;}
        public static void main(String [] args) {
                DefaultConstructor p = new DefaultConstructor();
                p.setX(3);
                System.out.println(p.getX());
        }
}
```

```
3
```

아래 코드와 같이 default 생성자를 프로그래머가 만들어도 된다.

```
public class DefaultConstructor {
		int x;
		public void setX(int x) {this.x = x;}
		public int getX() {return x;}
		public DefaultConstructor() {}
		public static void main(String [] args) {
			DefaultConstructor p = new DefaultConstructor();
			p.setX(3);
			System.out.println(p.getX());
		}
}
```

```
3
```

그런데 만약, 매개변수가 있는 생성자를 만들면 컴파일러는 default 생성자를 만들지 않는다.

왜냐하면 생성자가 있기 때문에 굳이 만들 필요가 없다고 판단하기 때문이다.

그래서 아래 코드에서 default 생성자를 호출하는 객체는 error가 발생한다.

< 문제 > 아래코드를 바르게 수정해 보시오. (hint, default 생성자를 추가하시오)

```
public class DefaultConstructor {
		int x;
		public void setX(int x) {this.x = x;}
		public int getX() {return x;}
		public DefaultConstructor(int x) {
			this.x = x;
		}
		public static void main(String [] args) {
			DefaultConstructor p1 = new DefaultConstructor(3);
			int n = p1.getX();
			DefaultConstructor p2 = new DefaultConstructor(); // 컴파일 오류
			p2.setX(5);
		}
}
```

< 정답 >

```
public class DefaultConstructor {
		int x;
		public void setX(int x) {this.x = x;}
		public int getX() {return x;}
		public DefaultConstructor(int x) {
			this.x = x;
		}
		public DefaultConstructor() {
		}

		public static void main(String [] args) {
			DefaultConstructor p1 = new DefaultConstructor(3);
			int n = p1.getX();
			System.out.println(p1.getX());
			DefaultConstructor p2 = new DefaultConstructor(); // 컴파일 오류
			p2.setX(5);
			System.out.println(p2.getX());
		}
}
```

```
3
5
```

static 변수

인스턴스의 생성과 상관없이 초기화 되는 변수
하나만 선언되는 변수
public로 선언되면 누구나 어디서든 접근 가능

실습예제4

```
class StaticTest {
	static int num = 0;

	public StaticTest() {
		num++;
		System.out.println("Num : " + num);
	}
}

class Test {
	public static void main(String[] args) {
		StaticTest t1 = new StaticTest();
		StaticTest t2 = new StaticTest();
		StaticTest t3 = new StaticTest();
	}
}
```

```
Num : 1
Num : 2
Num : 3
```

static 메서드의 정의와 호출

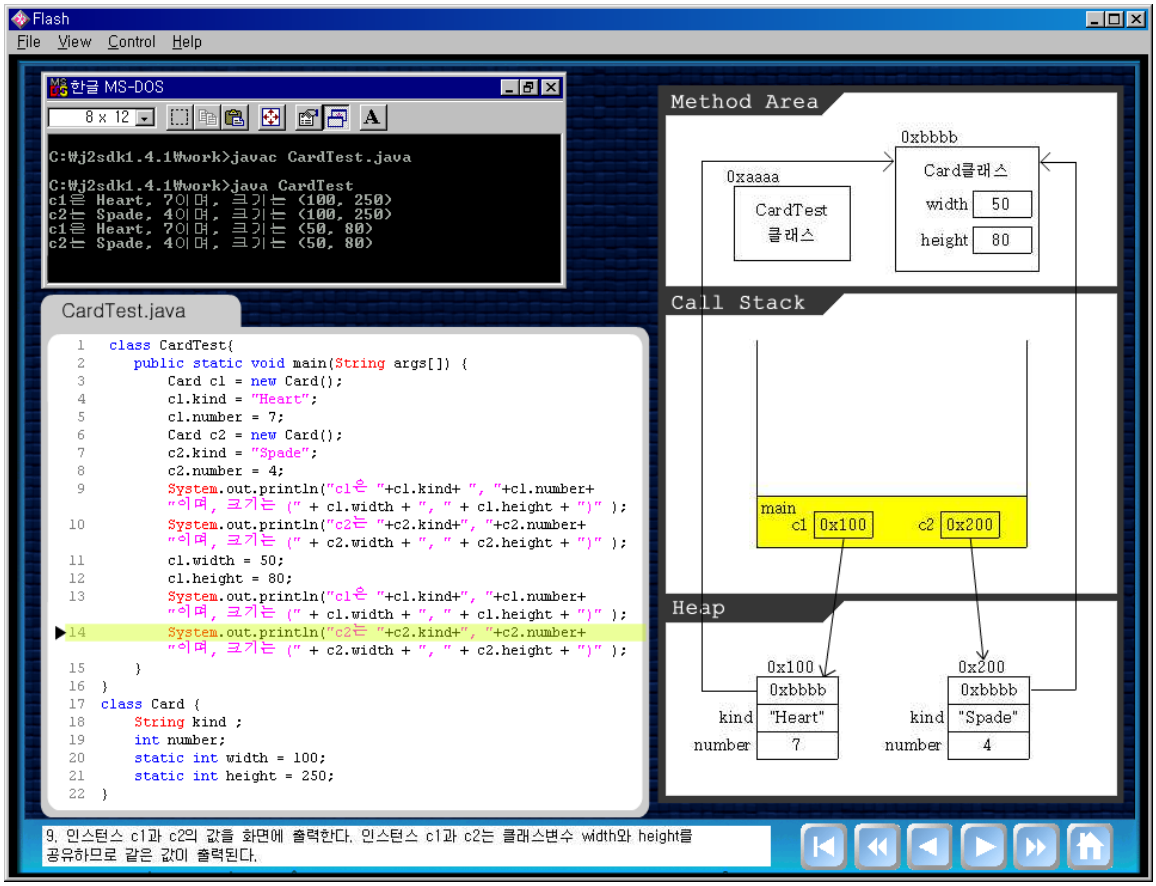
실습예제5

```
class StaticTest {
	public static void show(int n) {
		System.out.println(n);
	}
	public static void show2(double n) {
		System.out.println(n);
	}
}

class Test {
	public static void main(String[] args) {
		StaticTest.show(20); // 클래스 이름을 이용한 호출
		StaticTest t = new StaticTest();
		t.show2(3.15);		// 인스턴스의 이름을 통한 호출
	}
}
```

```
20
3.15
```

위의 예제에서 보이듯이 인스턴스를 생성하지 않아도 static 메서드는 호출 가능하다
아래 그림은 자바코드가 위치하는 JVM 메모리 위치를 나타내는 것이다.



접근 지정자

```
class Sample {
        public int a;
        private int b;
        int c;

        public int getB() {
                return b;
        }
        public void setB(int value) {
                b = value;
        }

        void show() {
                System.out.println("a: " + a);
                System.out.println("b: " + b);
                System.out.println("c: " + c);
        }
}

public class AccessEx {
        public static void main(String[] args) {
                Sample aClass = new Sample();
                aClass.a = 10;
                // aClass.b = 10; // private 이므로 접근 못함
                aClass.setB(10);
                aClass.c = 10;

                aClass.show();
```

```
        }
}
```

```
a: 10
b: 10
c: 10
```

접근 제어자가 사용될 수 있는 곳 - 클래스, 멤버변수, 메서드, 생성자

private - **같은 클래스** 내에서만 접근이 가능하다.
default - **같은 패키지** 내에서만 접근이 가능하다.
protected - 같은 패키지 내에서, 그리고 다른 패키지의 자손클래스에서 접근이 가능하다.
public - 접근 제한이 전혀 없다.

제어자	같은 클래스	같은 패키지	자손클래스	전 체
public				
protected				
default				
private				

접근 제어자를 사용하는 이유
- 외부로부터 데이터를 보호하기 위해서
- 외부에는 불필요한, 내부적으로만 사용되는, 부분을 감추기 위해서

이를 **캡슐화**라고 부른다

실습예제6

```
class Person {
        int age;                    // Student 클래스에서 접근 가능
        public String name;         // Student 클래스에서 접근 가능
        protected int height;       // Student 클래스에서 접근 가능
        private int weight;  // Student 클래스에서 접근 불가
        public void setWeight(int weight) {
                this.weight = weight;
        }
        public int getWeight() {
                return weight;
        }
}

public class Student extends Person {
        void set() {
                age = 30;
```

```
                name = "홍길동";
                height = 175;
                // weight = 50; //  접근 불가
                setWeight(99);

        }
        public static void main(String[] args) {
                Student s = new Student();
                s.set();
                System.out.println(s.getWeight());
        }
}
```

99

패키지 (Package)

- 클래스 들을 하나로 묶어놓은 것이다.
- 클래스 간의 이름 중복으로 발생하는 충돌을 막아준다.
- 클래스를 기능 별로 분류할수 있어 필요한 클래스의 식별이 용이하다.

기본 패키지	설명
java.lang	기본적인 클래스 제공 (자동으로 import)
java.awt	GUI에 관한 클래스 제공
java.io	데이터 입출력에 관한 클래스 제공
java.util	유용한 유틸리티 클래스 제공
java.net	네트워크 관련 클래스 제공
java.text	텍스트 관련 클래스 제공
java.sql	데이터베이스 관련 클래스 제공
java.applet	애플릿 구현에 필요한 클래스 제공

아래 예제는 Data 클래스를 import하여 현재 날짜와 시간을 출력하는 예제입니다.

```
import java.util.Date;

class DateClass {
        public static void main(String args[]) {
                Date date = new Date();

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

이번에는 Random 클래스를 import하여 난수를 생성하는 예제입니다.

```java
import java.util.Random;

class RandomClass {
	public static void main(String args[]) {
		Random random = new Random();

		System.out.println(random.nextInt());
	}
}
```

수고했습니다. 클래스와 객체는 자바에서 제일 중요한 부분 중에 하나이므로 완벽히 이해하고 넘어가야 합니다.

이제 아래 문제를 서비스클래스와 main 메서드를 포함하는 클래스를 만들어 코딩 하시오. 즉, 클래스가 2개 생성이 되어야 합니다.

공통: 1046, 1063, 1064, 1266, 1267, 1403, 1610, 1172, 1451, 1709, 4501

실습문제 1번, 5번