객체지향프로그래밍 – 보조 자료

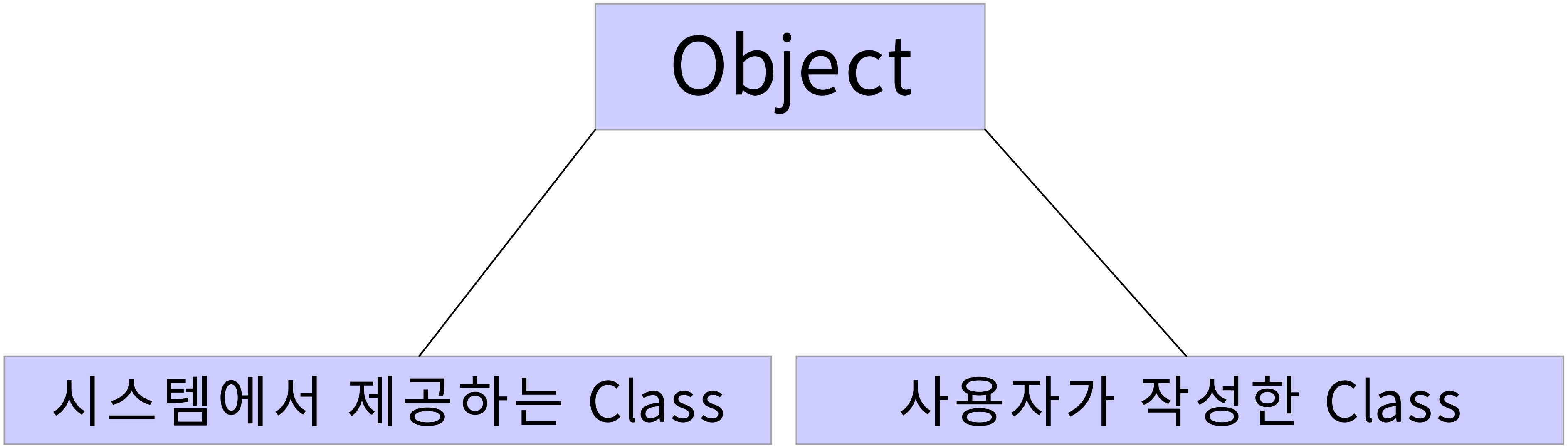
Object 클래스의 이해

동서대학교 인터넷공학부

김 태 용

오브젝트(Object), 리플랙션(Reflection)

- java.lang.Object
  - 모든 클래스의 부모 클래스
  - 모든 클래스는 오브젝트 클래스를 자동으로 상속



Object 클래스

- 모든 클래스는 Object의 하위 클래스
- Object 형 참조 변수는 모든 클래스의 객체를 참조 가능

메서드	목적	비고
Object clone()	복제 대상 객체와 동일한 객체를 새로 생성	오버라이드
boolean equals(Object obj)	한 객체가 다른 객체와 동일한지 판단	오버라이드
void finalize()	사용되지 않는 객체가 소멸되기 직전에 호출	오버라이드
Class getClass()	런타임 단계에 객체의 클래스를 구한다	Final
int hashCode()	호출한 객체의 해시 코드 반환	오버라이드
void notify()	호출한 객체에서 대기하고 있던 스레드 실행 재개	Final
void notifyAll()	호출한 객체에서 대기중인 모든 스레드 실행 재개	Final
String toString()	객체를 설명하는 문자열 반환	오버라이드
void wait()	또 하나의 스레드를 대기 상태로	final
void wait(long millisec)		
void wait(int millisec, intnanosec)		

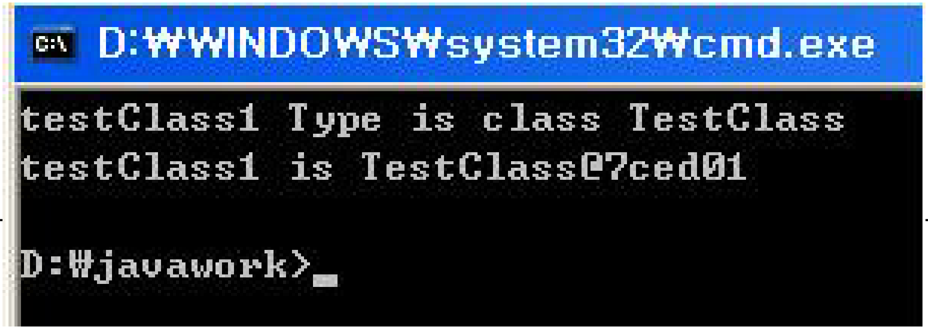
오브젝트(Object), 리플랙션(Reflection)

```
Import java.lang.System;
/* Object 클래스로부터 상속받은 메소드를 사용하기 위해 TestClass
    는 아무 데이터도 메소드도 가지지 않는다. */
class TestClass {}

class MainClass {
    public static void main(String args[]) {
        // Object로부터 상속받은 equals() 메소드를 사용하는 예
        // 객체를 하나 생성하고 두 개의 레퍼런스가 가리키게 한다.
        // 레퍼런스들이 같은지 (equals) 체크를 한다. 답은 같다!
        TestClass testClass1 = new TestClass();
        TestClass testClass2 = new TestClass();
        if(testClass1.equals(testClass2))
            System.out.println("두개의 클래스는 같습니다.");
```

countinue…

오브젝트(Object), 리플랙션(Reflection)

```
        // Object로부터 상속받은 getClass() 메소드를 이용하는 예
        // getClass()메소드가 리턴하는 값은 TestClass
        System.out.println(“testClass1 Type is “ +
                                    testClass1.getClass());

        // Object로부터 상속받은 toString() 메소드를 이용하는 예
        // ToString() 메소드가 리턴하는 값은 TestClass!
        System.out.println(“testClass1 is “ +
                                    testClass1.toString());
    }
}
```


D:₩WINDOWS₩system32₩cmd.exe

```
testClass1 Type is class TestClass
testClass1 is TestClass@7ced01

D:₩javawork>_
```

오브젝트(Object), 리플렉션(Reflection)

- 클래스에서 제공하는 메서드 알아오는 방법
  - 메서드를 알고자하는 클래스를 이용하여 Class 클래스 작성
  - 작성한 클래스(Class타입)의 getMethods메서드를 이용하여 메서드들을 가져온다.
  - 가져온 메서드들을 java.lang.reflect.Method[](배열) 객체에 대입
  - 배열 속에 저장된 메서드들을 하나씩 화면에 출력

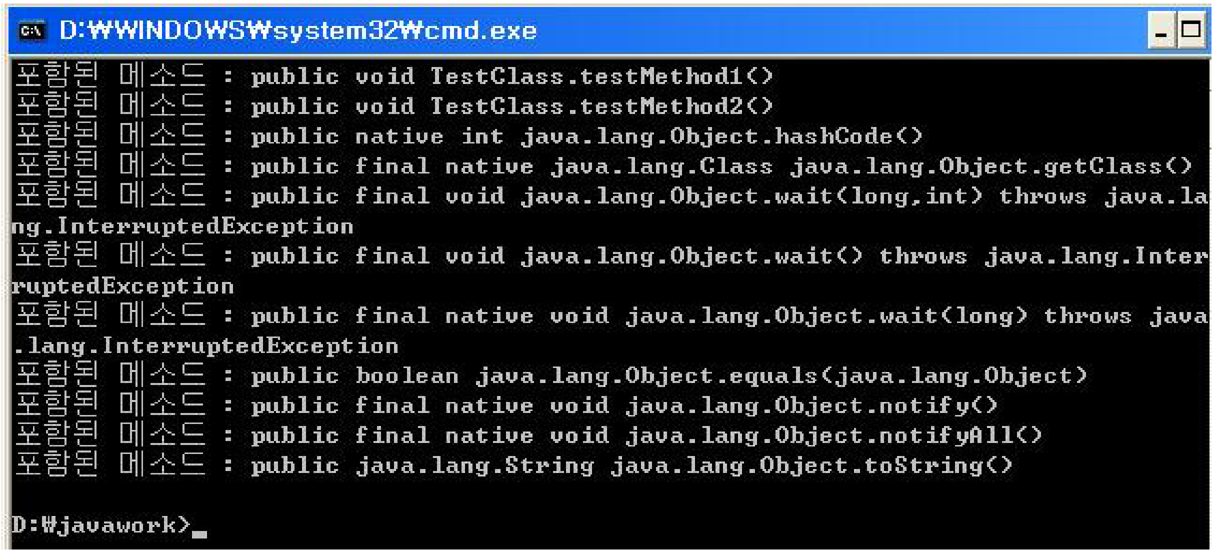
오브젝트(Object), 리플랙션(Reflection)

```
import java.lang.System;
import java.lang.reflect.Method;

// testMethod1()와 testMethod()2 메소드를 가진 테스트 클래스
class TestClass {
    public void testMethod1() {}
    public void testMethod2() {}
}
class MainClass {
    public static void main(String args[]) {
        // 예외 발생을 처리하기 위한  try ~ catch 문 사용
        try {
            // Class의 forName 메소드를 이용하여 Class 생성
            Class c = Class.forName("TestClass");
            /* Class에서 제공하는 getMethods()메소드를 이용하여
                메소드들의 정보를 가져온 후 Method 배열에 입력 */
            Method methods[] = c.getMethods();
```

countinue…

오브젝트(Object), 리플랙션(Reflection)

```
            for(int i = 0; I < methods.length; i++) {
              System.out.println(“포함된 메서드 : “ + methods[i]);
            }
        }
        catch(Exception e) {
            System.out.println(“예외 발생 : “ + e);
        }
    }
}
```



```
포함된 메소드 : public void TestClass.testMethod1()
포함된 메소드 : public void TestClass.testMethod2()
포함된 메소드 : public native int java.lang.Object.hashCode()
포함된 메소드 : public final native java.lang.Class java.lang.Object.getClass()
포함된 메소드 : public final void java.lang.Object.wait(long,int) throws java.la
ng.InterruptedException
포함된 메소드 : public final void java.lang.Object.wait() throws java.lang.Inter
ruptedException
포함된 메소드 : public final native void java.lang.Object.wait(long) throws java
.lang.InterruptedException
포함된 메소드 : public boolean java.lang.Object.equals(java.lang.Object)
포함된 메소드 : public final native void java.lang.Object.notify()
포함된 메소드 : public final native void java.lang.Object.notifyAll()
포함된 메소드 : public java.lang.String java.lang.Object.toString()

D:₩javawork>_
```

- TestClass에서 제공하는 메서드들(testMethod1과 2)외에 Object 클래스로부터 상속받은 메서드들도 화면에 출력 됨.

오브젝트(Object), 리플랙션(Reflection)

- Object 형을 이용한 데이터형 변환

  <예제> 여러 데이터형들을 하나의 배열에 넣어두었다가, 다시 꺼내와야 할 데이터 형들이 다를 경우, 오브젝트형의 배열을 이용하면 간단히 해결.

  1. introduceA()메소드를 가진 KindAClass클래스와 introduceB()메소드를 가진 KindBClass클래스 생성
  2. Object를 두 개 가진 배열을 만들고 각각 KindAClass형과 KindBClass형의 데이터를 참조하도록 한다.
  3. Object 값을 대입받기 위한 kindAClass, kindBClass객체 생성
  4. for문을 이용하여 배열속의 Object를 하나씩 가져온다.
  5. 가져온 Object를 toString() 메서드를 이용하여 적합한 객체에 데이터형을 넘겨준다.
  6. 넘겨받은 데이터형을 이용하여 적절한 introduce메서드(introduceA또는 introduceB)를 실행

오브젝트(Object), 리플랙션(Reflection)

```
import java.lang.System;

// introduceA() 메소드를 가진 KindAClass 클래스
class KindAClass {
    public void introduceA() {
        System.out.println(“KindAClass형입니다.”);
    }
}

// introduceB() 메소드를 가진 KindBClass 클래스
class KindBClass {
    public void introduceB() {
        System.out.println(“KindBClass형입니다.”);
    }
}
```

countinue…

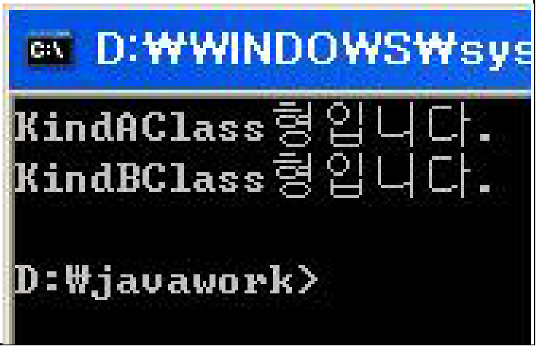
오브젝트(Object), 리플랙션(Reflection)

```
class MainClass {
    public static void main(String args[]) {
        // 배열 object[2]을 만들고 KindAClass와 KindBClass형 참조
        Object[] obj = new Object[2];
        obj[0] = new KindAClass();
        obj[1] = new KindBClass();

        KindAClass kindA;
        KindBClass kindB;

        // object[]배열 속의 값을 참조할 객체 선언
        for(int step = 0 ; step<obj.length; step++) {
            // toString 메소드르르 이용하여 어떤 데이터형인지 체크
            if(obj[step].toString().startsWith(“KindAClass)) {
                // 형 변환 연산자를 이용한 형 변환
                kindA = (KindAClass)obj[step];
```

countinue…

오브젝트(Object), 리플랙션(Reflection)

```
            // introduceA() 메소드 실행
            kindA.introduceA();
        }

        else {
            // 형변환 연산자를 이용한 형 변환
            kindB = (KindBClass)obj[step];
            // introduceB() 메소드 실행
            kindB.introduceB();
        }
      }
    }
}
```



```
D:WWINDOWSWsys
KindAClass형입니다.
KindBClass형입니다.

D:₩javawork>
```