1. Wrapper 클래스

■ int나 double같은 기초자료형을 객체로 사용할 때 Wraaper클래스 이용

■ 주방에서 사용하는 랩(Wrap)처럼 기초자료형을 객체로 담아 객체 생성

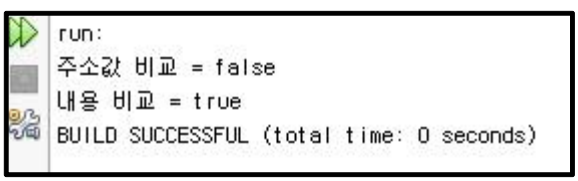
■ 각 자료형마다 Wrapper 클래스가 있음

■ 기본 자료형과 Wrapper 클래스 생성자

기본 자료형	Wrapper 클래스의 생성자
boolean	Boolean(boolean value), Boolean(String s)
byte	Byte(byte value), Byte(String s)
char	Character(char value)
short	short(short value), short(String s)
int	Integer(int value), Integer(String s)
long	Long(long value), Long(String s)
float	Float(double value), Float(float value), Float(String s)
double	Double(double value), Double(String s)

```
package wrapper;

public class WrapperEx01 {
    public static void main(String[] args) {
        Boolean wrap_b = new Boolean(true);
        Boolean wrap_b2 = new Boolean("TRUE");
        Boolean wrap_b3 = new Boolean("true");

        System.out.println(wrap_b == wrap_b2);
        System.out.println(wrap_b.equals(wrap_b2));

    }
}
```

결과화면



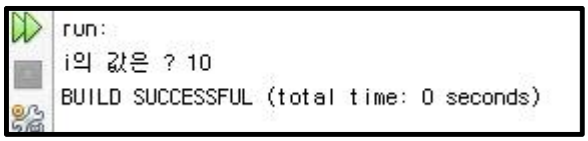
2. 오토박싱과 언박싱

■ int형과 같은 기본자료형들과 Wrapper클래스로 서로 변환해줄시 불편함을 해소할 목적으로 JDK 5.0부터 생김

■ 이전 버전(JDK 1.4 이하)의 Wrapper 클래스와 기초자료형 변환

```
package wrapper;

public class WrapperEx02 {
    public static void main(String[] args) {
        int a = 10;
        Integer i = new Integer(a);
        WrapperEx02.test(i);

    }
    public static void test(Integer i) {
        System.out.println("i의 값은 ? " + i.toString());
    }
}
```

결과화면



■ 오토박싱 : Integer i = 3; / 언박싱 : int a = i;

```
package wrapper;

public class WrapperEx03 {
    public static void main(String[] args) {
        int a = 10;
        Integer i = a;//autoBoxing
        a = i;//unBoxing
        WrapperEx03.test(a);
    }

    public static void test(Integer i) {
        System.out.println("i의 값은 ? " + i.toString());
    }
}
```

결과화면 생략

■ Wrapper 클래스와 String 클래스의 공통점

1) 암시적 객체 생성(오토박싱으로 가능)과 명시적 객체 생성이 있음

2) '==' 주소값 비교, equals() 메서드는 내용 비교

```
package wrapper;

public class WrapperEx04 {
    public static void main(String[] args) {
        Boolean b1 = true;
        Boolean b2 = new Boolean(true);
        Boolean b3 = true;
        Boolean b4 = new Boolean(true);

        System.out.println(b1==b2);
        System.out.println(b1==b3);
        System.out.println(b1.equals(b2));
        System.out.println(b1.equals(b4));
    }
}
```

결과화면

상 속

has a = 특정 객체 내에서 다른 객체를 가지고 있는 것을 말함

is a = 특정 객체가 다른 객체에게 자신의 능력을 포함 시켜주는

상속 관계를 의미

1. 상속의 개념과 중요성

■ 자바 클래스의 상속은 단일 상속

■ 상속관계의 용어

용 어	설 명
Base Class(기본 클래스) Super Class(슈퍼 클래스) Parent Class(부모 클래스)	상속을 주기 위해 준비된 특정 클래스
Derivation Class(유도 클래스) Sub Class(하위 클래스) Child Class(자식 클래스)	특정 클래스로부터 상속을 받아 새롭게 정의 되는 클래스

■ 상속에서의 접근 제한자.

private : 클래스 내부에서만 접근 가능

public : 모든 클래스에서 접근 가능

default : 같은 패키지 내의 클래스에서 접근 가능

protected: 같은 패키지내의 클래스 뿐만 아니라 자식 클래스 접근 가능

```
package ex3;
public class Ex4 {
    private int a=10;
    private int b=20;
    private int c=30;

   void setA(int a) {
        this.a = a;
    }

   protected  void setB(int b) {
        this.b = b;
    }

    private void setC(int c) {
        this.c = c;
    }



    public int getA() {
        return a;
    }
```

```
    public int getB() {
        return b;
    }

    public int getC() {
        return c;
    }
}
```

```
package ex3;

public class Ex4Test extends Ex4{

    String testM;

    public String getTestM() {
        return testM;
    }

    public void setTestM(String testM) {
        this.testM = testM;
    }

}
```

```
package ex3;
public class Ex4Main {
    public static void main(String[] args) {

        Ex4Test ref = new Ex4Test();

        ref.setA(1120);
        ref.setB(1130);

        System.out.println(""+ref.getA()+"  "+ref.getB());

    }
}
```

```
run:
1120  1130
BUILD SUCCESSFUL (total time: 0 seconds)
```

오버라이딩(Overriding)

■ 상속받은 메서드의 내용을 변경하는 것

CellPhone.java

```
class CellPhone {
    public void call() {
        System.out.println("통화를 합니다.");
    }
}
```

CellPhone3G.java

```
class CellPhone3G extends CellPhone {
    public void call() {
        System.out.println("영상통화를 합니다.");
    }
}
```

■ 오버라이딩의 조건

자손클래스에서 오버라이딩하는 메서드는 조상클래스의 메서드와
- 이름이 같아야 함
- 매개변수가 같아야 함
- 리턴타입이 같아야 함

접근제한자는 더 넓게 지정해야 함

오버라이딩 vs 오버로딩

차이점	오버라이딩	오버로딩
영역	상속간의 클래스들(최소 2개 이상)	하나의 클래스
메서드명	똑같아야 함	똑같아야 함
인자	똑같아야 함	반드시 달라야 함
리턴값	똑같아야 함	달라도 됨(같아도 상관없음)
접근제한자	달라도 되나, 자식 클래스가 부모보다 넓게	달라도 됨(같아도 상관없음)

■ 오버로딩

```
public class OverloadEx01 {
①    public String test(int a) {
       return "test";
     }

②    void test() {

     }

③    public void test() {

     }

④    protected boolean test(int b) {
       return false;
     }

⑤    private String test(String s) {
       return "test";
     }

 ⑥   public void test2() {

     }
}
```

■ 오버라이딩

```
package overriding;

public class OverridingEx01 {
    public String test() {
        return "test";
    }
}
```

```
package overriding;

public class OverridingEx01_Sub extends OverridingEx01 {

①    public String test() {
       return "test";
    }

②    private String test2() {
       return "test";
    }

③    void test(int a) {

    }

④    public int test(String s) {
       return 1;
    }

⑤    protected String test() {
       return "test";
    }
}
```

2. super 키워드

■ 부모 객체의 레퍼런스

■ 부모클래스의 멤버와 자손클래스의 멤버가 중복 정의되어 서로 구별해야 하는 경우에만 super를 사용하는 것이 좋음

```
package supertest;

public class Super {
    int a;
}
```

```
package supertest;

public class Sub extends Super {
    public void test() {
        System.out.println(this.a);
        System.out.println(super.a);
    }
}
```

```
package supertest;

public class Test {
    public static void main(String[] args) {
        Sub sub = new Sub();
        sub.test();

    }
}
```

결과화면

```
run:
10
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. 상속에서의 생성자

■ 자식의 기초생성자에는 super()가 생략되어 있음

■ 인스턴스 생성시 무조건 부모 클래스부터 생성됨

■ 즉, 부모의 생성자를 무조건 먼저 호출함

```
package constructor;

public class Super {

    public Super(int a) {
        System.out.println("Super 클래스의 생성자입니다.");
    }
}
```

```
package constructor;

public class Sub extends Super {

    public Sub(int a) {
        System.out.println("Sub 클래스의 생성자입니다.");
    }

}
```

```
package constructor;

public class Test {
    public static void main(String[] args) {
        Sub sub = new Sub();
    }
}
```

■ this(), super()는 생성자의 가장 위에 명시해야 함

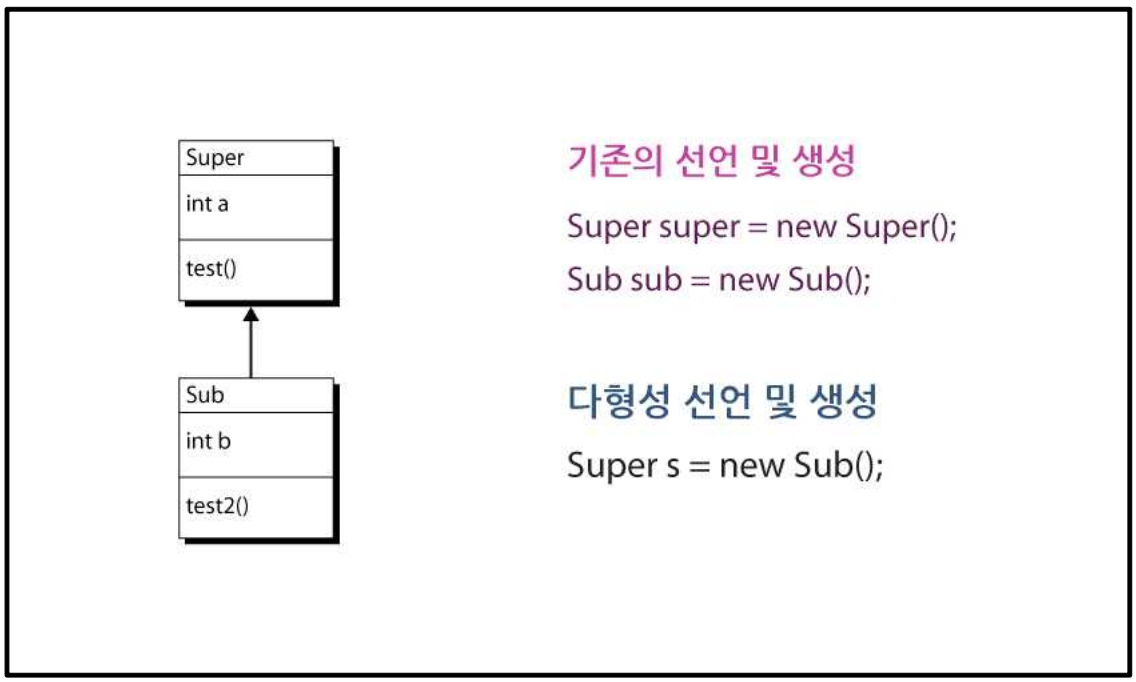
```
public class ArrayTest {

    public void test(String[] sArr) {
        for(String s : sArr) {
            System.out.println(s);
        }
    }
}
```

```
package array;

import java.util.Scanner;

public class ArrayTestMain {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("숫자를 입력");

        int length = sc.nextInt();
        sc.nextLine();

        String[] sArry = new String[length];

        for(int i = 0 ; i < sArry.length ; i++) {
            sArry[i] = sc.nextLine();
        }

        ArrayTest t = new ArrayTest();

        t.test(sArry);

    }
}
```

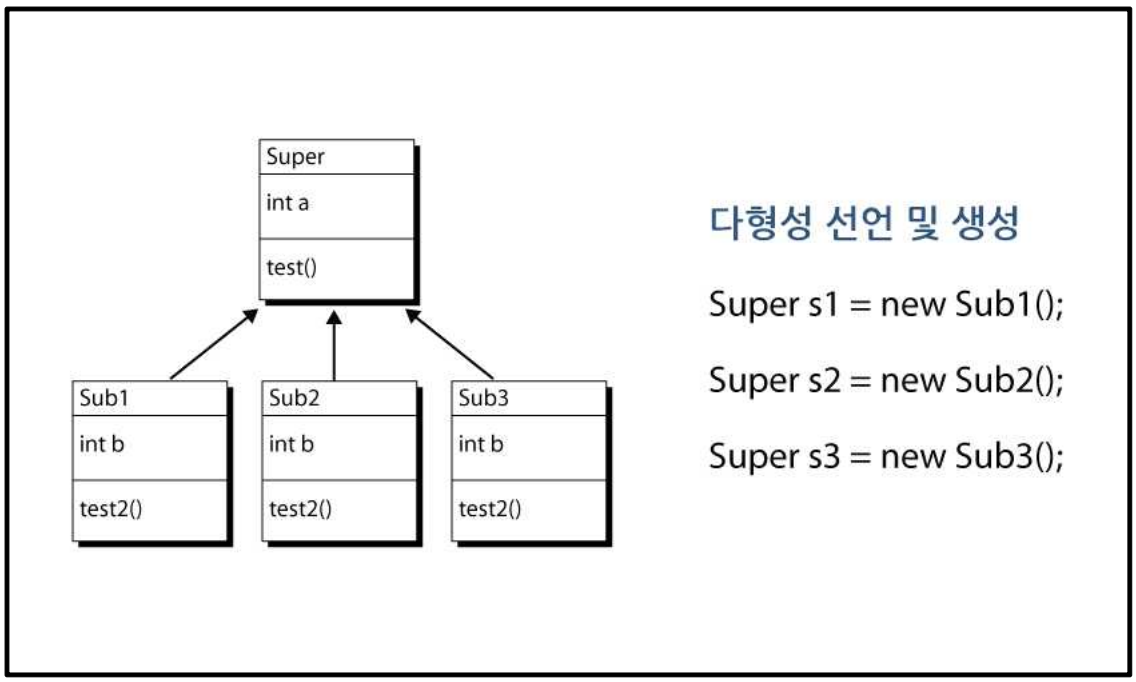
4. 다형성

■ 한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있도록 하는 것

■ 부모클래스 타입의 참조변수로 자식클래스의 인스턴스를 참조할 수 있도록 함



■ 다형성의 실제 모습



■ 동물병원

예제 6 : C:\Java_Study\AnimalHospital\src\animal\Animal.java, Dog.java, Tiger.java

```
package animal;

public class Animal {

    public String scream() {
        return "동물 울음소리";
    }

    public String getName() {
        return null;
    }
}
```

```
package animal;

public class Dog extends Animal{
    private String name;

    public Dog() {
        name = getClass().getSimpleName();
    }

    public String scream() {
        return "왈왈왈";
    }

    public String getName() {
        return name;
    }
}
```

```
package animal;

public class Cat extends Animal {
    private String name;

    public Cat() {
        name = getClass().getSimpleName();
    }

    public String scream() {
        return "야옹야옹";
    }

    public String getName() {
```

```
        return name;
    }
}
```

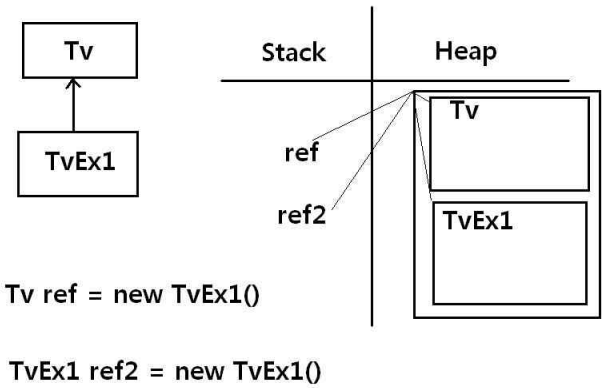
```
package animal;

public class Tiger extends Animal{
    private String name;

    public Tiger() {
        name = getClass().getSimpleName();
    }

    public String scream() {
        return "으르렁으헝헝";
    }

    public String getName() {
        return name;
    }
}
```

C:\Java_Study\AnimalHospital\src\hospital\Hospital.java, HospitalMain.java

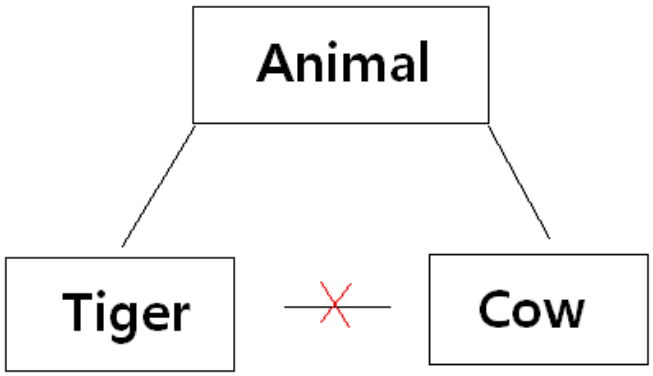
```
package hospital;

import animal.Animal;

public class Hospital {
    public void inject(Animal animal) {
        System.out.println(animal.getName() + "을(를) 치료하기 위해 주사를 놓았습니다.");
        System.out.println(animal.scream());
    }
}
```

```
package hospital;

import animal.Animal;
import animal.Cat;
import animal.Dog;
import animal.Tiger;

public class HospitalMain {
    public static void main(String[] args) {
        Hospital hospital = new Hospital();
        Animal dog = new Dog();
        Animal cat = new Cat();
        Animal tiger = new Tiger();
```

```
        hospital.inject(dog);
        hospital.inject(cat);
        hospital.inject(tiger);
    }
}
```

설 명	그 림
참조형 변수 ref2일 경우 : 상속되는 멤버의 제한과 은폐에서 오버라이딩(Overriding)되는 것과 부모에 정의된 멤버만 사용 가능	

그 림	내 용

	■ 자바에서는 부모 자식관계만 정의
	■ Tiger ref2 = new Cow(); //컴파일에러
	■ Tiger ref2 = (Tiger)new Cow();//에러
	■ 자식이 부모를 참조 하는경우..
	Cow ref3 = (Cow) new Animal2();
	ref3.eat();
	ref3.barkStart();

예제 7 : Product.java

```
package ex2;

public class Product {
      int price;   // 제품의 가격
      int bonusPoint;  // 제품구매 시 제공하는 보너스점수
      Product(int price) {
            this.price = price;
```

```
            bonusPoint =(int)(price/10.0);  // 보너스점수는 제품가격의 10%
      }
}
```

```
package ex2;
public class NetBook extends Product{

    public NetBook(int price) {
        super(price);
    }
    @Override
    public String toString() {
        return "Netbook";
    }
}
```

```
package ex2;
public class Computer extends Product{

    public Computer(int price) {
        super(price);
    }

    @Override
    public String toString() {
        return "Computer";
    }

}
```

```
package ex2;

public class Buyer {
int money = 1000;       // 소유금액
int bonusPoint = 0;       // 보너스점수

      void buy(Product p) {
            if(money < p.price) {
                  System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
                  return;
            }
            money -= p.price;             // 가진 돈에서 구입한 제품의 가격을 뺀다.
            bonusPoint += p.bonusPoint;       // 제품의 보너스 점수를 추가한다.
            System.out.println(p + "을/를 구입하셨습니다.");
      }
```

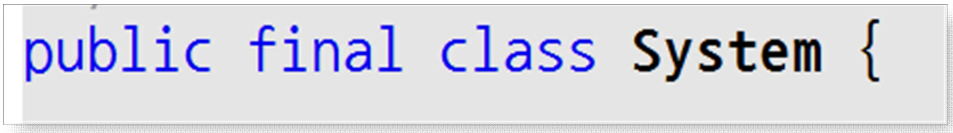
```
}
```

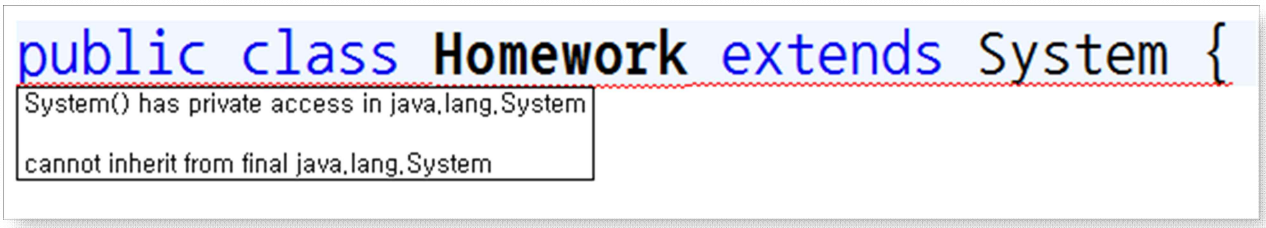
```
package ex2;

public class UseBuy {
 public static void main(String args[]) {
            Buyer b = new Buyer();
            NetBook netb = new NetBook(70);
            Computer com = new Computer(100);

            b.buy(netb);
            b.buy(com);

            System.out.println("현재 남은 돈은 " + b.money + "만원입니다.");
            System.out.println("현재 보너스점수는 " + b.bonusPoint + "점입니다.");
      }

}
```

```
run:
Netbook을/를 구입하셨습니다.
Computer을/를 구입하셨습니다.
현재 남은 돈은 830만원입니다.
현재 보너스점수는 17점입니다.
BUILD SUCCESSFUL (total time: 0 seconds
```

1. final 키워드

① 클래스에 선언시 : 상속불가 클래스





② 멤버필드에 선언시 : 상수(초기화를 꼭 해야 함)

```
public class A {
    public static final int PRICE = 1000;
}
```

③ 메서드에 선언시 : 오버라이딩이 불가능한 메서드로 선언

```
public class A {
    public final void test() {
    }
}
```

2. 추상클래스

■ abstract 키워드로 선언된 클래스

■ 일반클래스와 동일하게 멤버필드, 메서드, 생성자 모두 가질 수 있음

■ 추상클래스는 해당 클래스로 객체 생성이 불가능

■ 추상메서드를 가질 수 있음

■ 추상메서드를 가진 클래스는 무조건 추상클래스로 선언해야 함

예제 1 :

```
package abs;
// 추상 클래스 선언.
abstract public class Ex1 {
// 추상 클래스 내부에 일반 메서드를 사용.
    public void exTest(){
        System.out.println("Test");
    }
}
```

```
package abs;

public class Ex1Main {
    public static void main(String[] args) {
      //  Ex1 ref = new Ex1(); // Ex1은 추상 클래스 이기 때문에 new로 생성 시킬 수 없다.
      //  ref.exTest();

    }
}
```

3. 클래스 / 멤버필드 / 메서드 / 생성자의 선언

① 클래스의 선언

구 분	modifier	설 명
접근권한	public	모든 클래스에서 접근이 가능함
	(default)	같은 패키지에서 접근이 가능함
활용방법	final	상속이 불가능한 클래스
	abstract	추상클래스

② 멤버필드

구 분	modifier	설 명
접근권한	public	모든 클래스에서 접근이 가능함
	protected	같은 패키지와 상속관계에서 접근이 가능함
	(default)	같은 패키지에서 접근이 가능함
	private	같은 클래스 내부에서만 접근이 가능함
활용방법	final	상속이 불가능한 클래스
	static	static영역

③ 생성자

구 분	modifier	설 명
접근권한	public	모든 클래스에서 접근이 가능함
	protected	같은 패키지와 상속관계에서 접근이 가능함
	(default)	같은 패키지에서 접근이 가능함
	private	같은 클래스 내부에서만 접근이 가능함

④ 메서드

구 분	modifier	설 명
접근권한	public	모든 클래스에서 접근이 가능함
	protected	같은 패키지와 상속관계에서 접근이 가능함
	(default)	같은 패키지에서 접근이 가능함
	private	같은 클래스 내부에서만 접근이 가능함
활용방법	final	상속이 불가능한 클래스
	static	static영역
	abstract	오버라이딩이 불가능한 메서드를 정의
	synchronized	스레드의 동기화를 위한 메서드

4. 인터페이스

■ 상수와 추상메서드만 가지고 있음

■ 인터페이스 구현시 implements 키워드 사용

■ 인터페이스의 경우 다중구현이 가능함

■ 필드의 경우 static final을 생략해도 됨

■ 메서드의 경우도 abstract 생략해도 됨

■ 인터페이스를 인터페이스가 상속

예제 1 :

```
package inter;

public interface SuperInter {

    public void testMethod();

}
```

```
package inter;

public interface SubInter extends SuperInter{

    public abstract void testMethod2();

}
```

```
package inter;

public class SubEx1 implements SubInter{

    public void testMethod2() {
      System.out.println("SubInter 인터페이스의 메서드를 구현");
    }

    public void testMethod() {
        System.out.println("SuperInter 인터페이스의 메서드를 구현");
```

```
    }

}
```

```
package inter;

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

        SubEx1 ref = new SubEx1();
        ref.testMethod();
        ref.testMethod2();
    }

}
```

```
run:
SuperInter 인터페이스의 메서드를 구현
SubInter 인터페이스의 메서드를 구현
BUILD SUCCESSFUL (total time: 1 second)
```

5. Object

■ 한국사람으로 보면 '단군'과 같은 존재

■ 모든 클래스의 super클래스(배열을 제외하고 상속 받음)

■ 'extends' 라는 키워드를 쓰지 않음

ex) import java.lang.*과 같이 내부적으로 선언되어 있음

■ 주요 메서드

반환형	메서드명	설 명
boolean	equals(Object obj)	obj와 같은지 검사하여 같으면 true (String,Wrapper 제외시 주소값 비교(==)와 같음)
int	hashCode()	현 객체의 해시코드를 반환함
String	toString()	현 객체의 파생 클래스명과 @에 이어 해쉬코드를 출력

```
package ex3;

public class ObjectTest {
    private String name;
    private int price;

    public ObjectTest(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public static void main(String[] args) {
        ObjectTest test1 = new ObjectTest("1", 1);
        ObjectTest test2 = new ObjectTest("1", 1);

        System.out.println("test1은? " + test1);
        System.out.println("test2은? " + test2);
        System.out.println("test1의 해시코드는? " + test1.hashCode());
        System.out.println("test2의 해시코드는? " + test2.hashCode());

        System.out.println("test1과 test2는 같은가? " + test1.equals(test2));
    }
}
```

결과화면

```
run:
test1은? ex3.ObjectTest@de6ced
test2은? ex3.ObjectTest@c17164
```

```
test1의 해시코드는? 14576877
test2의 해시코드는? 12677476
test1과 test2는 같은가? false
BUILD SUCCESSFUL (total time: 2 seconds)
```

■ hashcode는 주소값과 다름

```
package ex3;

public class HashcodeTest {
    public static void main(String[] args) {
        String str = new String("TEST");
        String str2 = new String("TEST");

        System.out.println("str과 str2의 주소값은 같나요 ? : " + (str==str2));

        System.out.println("str과 str2의 해시코드는 같나요? : " +
                (str.hashCode()==str.hashCode()));

        System.out.println("str의 해시코드? : " + str.hashCode());
        System.out.println("str2의 해시코드? : " + str2.hashCode());

        //객체의 해시코드는?
        HashcodeTest test1 = new HashcodeTest();
        HashcodeTest test2 = new HashcodeTest();

        System.out.println("test1과 test2의 주소값은 같나요 ? : " + (test1==test2));
        System.out.println("test1과 test2의 해시코드값은 같나요? : "
                +(test1.hashCode()==test2.hashCode()));

        System.out.println("test1의 해시코드는 ? : " + test1.hashCode());
        System.out.println("test2의 해시코드는 ? : " + test2.hashCode());
    }
}
```

결과화면

```
run:
str과 str2의 주소값은 같나요 ? : false
str과 str2의 해시코드는 같나요? : true
str의 해시코드? : 2571410
str2의 해시코드? : 2571410
test1과 test2의 주소값은 같나요 ? : false
test1과 test2의 해시코드값은 같나요? : false
test1의 해시코드는 ? : 14576877
test2의 해시코드는 ? : 12677476
BUILD SUCCESSFUL (total time: 2 seconds)
```