

java.lang

- ❖ 자바 프로그래밍에서 가장 기본이 되는 클래스들을 포함하고 있는 패키지
- ❖ 이 패키지에 속한 클래스들은 import 구문 없이 바로 사용 가능
- ❖ 이 패키지에 속한 대표적인 클래스가 String 과 System

1. java.lang.Object

❖ Object 클래스

- ✓ 자바 클래스 중 최상위 클래스
- ✓ 다른 클래스로부터 상속받지 않으면(extends 예약어를 사용하지 않으면) 이 클래스로부터 상속된 것

❖ Object 클래스의 메소드

- ✓ Object () 디폴트 생성자
- ✓ Object clone() - 인스턴스를 복제하기 위해 사용하는 메소드로 재정의해서 사용
이 메소드를 재정의하는 클래스는 Cloneable 인터페이스를 implements 하는 것을 권장
- ✓ boolean equals(Object object) - 두 개 인스턴스의 내용이 같은 지를 비교하여 같으면 true, 그렇지 않으면 false를 반환하는 메소드로 재정의 해서 사용
- ✓ void finalize() - 인스턴스가 소멸될 때 호출되는 메소드로 재정의 가능한 메소드인데 이 메소드의 호출 시점은 인스턴스가 garbage collection의 대상이 될 때가 아니고 garbage collectio이 호출되는 시점이며 garbage collection을 강제로 호출하고자 하는 경우에는 System.gc()와 System.runFinalization()을 연속적으로 호출
- ✓ Class getClass() - 인스턴스의 자료형을 Class 형의 인스턴스로 반환
- ✓ int hashCode() - 인스턴스와 연관된 hash 코드를 반환
- ✓ String toString() - 현재 인스턴스의 문자열 표현을 반환
출력하는 메소드에 인스턴스 이름만 사용하면 자동으로 호출되며 재정의 해서 사용하는 것이 일반적 - 재정의 하지 않으면 자료형과 해시코드를 문자열로 리턴
- ✓ void notify() - 대기 중인 스레드 중 하나의 스레드를 시작
- ✓ void notifyAll() - 대기 중인 모든 스레드를 시작
- ✓ void wait() - 실행을 중지시키고 대기 상태로 이동

실습(ObjectClass.java)

```
public class ObjectClass{
    private int num;

    public int getNum() {
        return num;
    }
    public void setNum(int num) {
        this.num = num;
    }
    @Override
    //equals 메소드 재정의
    //num의 값이 같으면 동일한 인스턴스로 간주하기 위한 메소드
    public boolean equals(Object aObj){
        if(this.num == ((ObjectClass)aObj).num)
            return true;
        else
            return false;
    }
}
```

실습(ObjectMain.java)

```
public class ObjectMain {  
  
    public static void main(String[] args) {  
        ObjectClass obj1 = new ObjectClass();  
        obj1.setNum(10);  
  
        ObjectClass obj2 = new ObjectClass();  
        obj2.setNum(10);  
  
        if(obj1 == obj2){  
            System.out.println(  
                "2개의 변수가 동일한 인스턴스를 가리키고 있습니다.");  
        }  
        else{  
            System.out.println(  
                "2개의 변수가 동일한 인스턴스를 가리키고 있지 않습니다.");  
        }  
    }  
}
```

실습(ObjectMain.java)

```
if(obj1.equals(obj2)){  
    System.out.println(  
        "2개의 변수가 가리키는 인스턴스의 내용이 같습니다.");  
}  
else{  
    System.out.println(  
        "2개의 변수가 가리키는 인스턴스의 내용이 같지 않습니다.");  
}  
}  
}
```

1. java.lang.Object

- ❖value 타입의 데이터는 다른 기억장소에 대입될 때 저장된 값이 복사가 되기 때문에 다른 기억 장소에 대입을 했을 때 복사본의 변화가 원본에 영향을 주지 않지만 참조형은 대입을 하면 주소가 복사되서 복사본의 변화가 원본에 영향을 주게 됨
- ❖이러한 경우를 방지하고자 참조형을 다른 참조형 변수에 바로 대입하지 않고 내부 내용을 복제해서 대입하는 것을 권장
- ❖얕은 복제: 인스턴스 내부의 내용을 단순히 복사해서 대입하는 것으로 내부 데이터 중에서 참조형이 존재하게 되면 복사본의 변화가 원본에 영향을 주게 됨
- ❖깊은 복제: 인스턴스 내부의 데이터 중에서 참조형이 존재하는 경우 참조형을 다시 복제해서 대입해주는 것으로 복사본의 변화가 원본에 영향을 주지 않게 됨

실습(CloneObject.java)

```
import java.util.Arrays;

public class CloneObject implements Cloneable {
    private int num;
    private int[] ar;

    public int getNum() {
        return num;
    }
    public void setNum(int num) {
        this.num = num;
    }
    public int[] getAr() {
        return ar;
    }
    public void setAr(int[] ar) {
        this.ar = ar;
    }
}
```

실습(CloneObject.java)

```
@Override
```

```
// clone 메소드 재정의
```

```
public CloneObject clone() throws CloneNotSupportedException {  
    CloneObject temp = new CloneObject();  
    temp.num = num;  
    // 얕은 복제  
    temp.ar = ar;  
    return temp;  
}
```

```
@Override
```

```
public String toString() {  
    return "CloneObject [num=" + num + ", ar=" + Arrays.toString(ar) + "];"  
}
```

```
}
```


실습(CloneObjectMain.java)

```
public class CloneObjectMain {  
    public static void main(String[] args) throws Exception {  
        CloneObject obj1 = new CloneObject();  
        obj1.setNum(10);  
        int [] ar = {10,30,50};  
        obj1.setAr(ar);  
        //주소를 복사  
        CloneObject obj2 = obj1;  
        //clone을 이용한 복제  
        CloneObject obj3 = obj1.clone();  
        obj1.setNum(99);  
        System.out.println(obj1);  
        //주소를 대입한 것이므로 원본의 변화가 영향을 주게 됩니다.  
        System.out.println(obj2);  
        //복제를 한 것이므로 원본의 변화에 영향이 없습니다.  
        System.out.println(obj3);  
        System.out.println("=====");  
    }  
}
```

ObjectClass [num=99, ar=[10, 30, 50]]
ObjectClass [num=99, ar=[10, 30, 50]]
ObjectClass [num=10, ar=[10, 30, 50]]
=====

실습(CloneObjectMain.java)

```
obj1.getAr()[0] = 123;  
System.out.println(obj1);  
//얕은 복제를 한 것이므로 내부 데이터 중에서 참조형의 데이터를 변경하면  
//영향을 미치게 됩니다.  
System.out.println(obj3);  
}  
}
```

```
ObjectClass [num=99, ar=[10, 30, 50]]  
ObjectClass [num=99, ar=[10, 30, 50]]  
ObjectClass [num=10, ar=[10, 30, 50]]  
=====  
ObjectClass [num=99, ar=[123, 30, 50]]  
ObjectClass [num=10, ar=[123, 30, 50]]
```

실습(clone 메소드 수정)

```
@Override
//clone 메소드 재정의
public ObjectClass clone() throws CloneNotSupportedException{
    ObjectClass temp = new ObjectClass();
    temp.num = num;
    //얕은 복제
    //temp.ar = ar;

    //깊은 복제
    temp.ar = new int[ar.length];
    for(int i=0; i<ar.length; i++){
        temp.ar[i] = ar[i];
    }
    return temp;
}
```

CloneObject [num=99, ar=[10, 30, 50]]

CloneObject [num=99, ar=[10, 30, 50]]

CloneObject [num=10, ar=[10, 30, 50]]

=====

CloneObject [num=99, ar=[123, 30, 50]]

CloneObject [num=10, ar=[10, 30, 50]]

1. java.lang.Objects

❖ Object의 유틸리티 클래스

리턴타입	메소드(매개변수)	설명
int	compare(T a, T b, Comparator<T> c)	두 객체 a 와 b 를 Comparator 를 사용해서 비교
boolean	deepEquals(Object a, Object b)	두 객체의 깊은 비교(필드도 비교)
boolean	equals(Object a, Object b)	두 객체의 얕은 비교(번지만 비교)
int	hash(Object... values)	매개값이 저장된 배열의 해시코드 생성
int	hashCode(Object o)	객체의 해시코드 생성
boolean	isNull(Object obj)	객체가 널 인지 조사
boolean	nonNull(Object obj)	객체가 널이 아닌지 조사
T	requireNonNull(T obj)	객체가 널인 경우 예외 발생
T	requireNonNull(T obj, String message)	객체가 널인 경우 예외 발생(주어진 예외 메시지 포함)
T	requireNonNull(T obj, Supplier<String> messageSupplier)	객체가 널인 경우 예외 발생(람다식이 만든 예외 메시지 포함)
String	toString(Object o)	객체의 toString() 리턴값 리턴
String	toString(Object o, String nullDefault)	객체의 toString() 리턴값 리턴, 첫번째 매개값이 null 일 경우 두번째 매개값 리턴

1. java.lang.Objects

❖ 객체 비교 (Objects.compare(T a, T b, Comparator<T> c))

- ✓ a, b 두 객체를 비교해 int값 리턴
- ✓ Comparator<T> 인터페이스
 - 제너릭 인터페이스 타입
 - T 타입의 객체를 비교하는 compare(T a, T b) 메소드를 소유

```
public interface Comparator<T> {  
    int compare(T a, T b)  
}
```

1. java.lang.Objects

❖ 동등 비교(equals()와 deepEquals())

✓ Objects.equals(Object a, Object b)

a	b	Objects.equals(a, b)
not null	not null	a.equals(b)
null	not null	false
not null	null	false
null	null	true

✓ Objects.deepEquals(Object a, Object b)

○ 비교할 객체가 배열일 경우 항목 값까지도 비교

a	b	Objects.deepEquals(a, b)
not null (not array)	not null (not array)	a.equals(b)
not null (array)	not null (array)	Arrays.deepEquals(a, b)
not null	null	false
null	not null	false
null	null	true

1. java.lang.Objects

❖ 해시코드 생성(hash(), hashCode())

✓ Objects.hash(Object... values)

- 매개값으로 주어진 값들 이용해 해시 코드 생성하는 역할
- Arrays.hashCode(Object[]) 호출해 해시코드 얻어 리턴
- 클래스의 hashCode()의 리턴값 생성할 때 유용하게 사용

```
@Override  
public int hashCode() {  
    return Objects.hash(field1, field2, field3);  
}
```

✓ Objects.hashCode(Object o)

- o.hashCode() 호출하고 받은 값 리턴
- 매개값이 null 이면 0 리턴

1. java.lang.Objects

❖ 널 여부 조사(isNull(), nonNull(), requireNonNull())

- ✓ Objects.isNull(Object obj)
 - obj가 null일 경우 true
- ✓ Objects.nonNull(Object obj)
 - obj가 not null일 경우 true
- ✓ requireNonNull()

리턴타입	메소드(매개변수)	설명
T	requireNonNull(T obj)	not null → obj null → NullPointerException
T	requireNonNull(T obj, String message)	not null → obj null → NullPointerException(message)
T	requireNonNull(T obj, Supplier<String> msgSupplier)	not null → obj null → NullPointerException(msgSupplier.get())

1. java.lang.Objects

❖ 객체 문자정보(toString())

- ✓ 객체의 문자정보 리턴
- ✓ 첫 번째 매개값이 not null - toString ()으로 얻은 값을 리턴
- ✓ null이면 “null” 또는 두 번째 매개값인 nullDefault 리턴

2. Wrapper 클래스

❖ Wrapper Class

- ✓ 자바에서는 실행의 효율성을 위해 기본 자료형(8가지)은 값을 직접 저장해서 사용할 수 있도록 하고 8가지의 기본 자료형을 참조형으로 만들 수 있는 클래스를 제공
- ✓ 8개의 기본 자료형과 관련된 클래스가 포장(Wrapper) 클래스
- ✓ 기본 자료형을 인스턴스로 변환해서 사용하면 해당 클래스에서 제공하는 메소드를 사용할 수 있고 참조형 만을 지원하는 다른 클래스(Collection)들에서도 사용할 수 있음
- ✓ Wrapper 클래스: Boolean, Character, Byte, Short, Integer, Long, Float, Double
- ✓ Wrapper 클래스로부터 생성된 인스턴스의 내용은 한번 생성된 다음에는 변경 할 수 없음

❖ Integer Class

- ✓ 정수 값을 포장하는 클래스
- ✓ 생성자
 - Integer(int n) n: 정수형 값
 - Integer(String str) str: 문자열(문자열과 동등한 정수 값을 생성)

❖ Boolean 클래스

- ✓ True 또는 false 값을 포장하는 클래스
- ✓ 생성자
 - Boolean(boolean value)
 - Boolean(String s)

2. Wrapper 클래스

❖ Character 클래스

- ✓ 문자 값을 포장하는 클래스
- ✓ 생성자
- ✓ Character (char value) value: char 형의 값

❖ Long 클래스

- ✓ long 값을 포장하는 클래스
- ✓ 생성자
 - Long(long l) l: long 형의 값
 - Long(String str) throws NumberFormatException str: long 형의 값과 같은 의미의 문자열

❖ Float 클래스

- ✓ float 형의 값을 포장하는 클래스
- ✓ 생성자
 - Float(float f)
 - Float(double d) f, d : float, double 형의 값
 - Float(String str) throws NumberFormatException str : float 형의 값과 같은 의미의 문자열

2. Wrapper 클래스

❖ Double 클래스

- ✓ double 형의 값을 포장하는 클래스
- ✓ 생성자
 - Double(double d): d: double 형의 값
 - Double(String str) throws NumberFormatException str: double 형의 값과 같은 의미의 문자열

❖ Float 클래스와 Double 클래스는 두 개의 특수한 메소드를 소유

- ✓ isInfinite(): 인스턴스의 값이 무한대의 작은 값이나 큰 값을 가질 때 true를 반환하는 메소드
- ✓ isNaN(): 인스턴스가 숫자가 아니면(Not a Number) true를 반환하는 메소드

2. Wrapper 클래스

❖ Character와 Boolean을 제외하고 모두 Number라는 추상클래스로부터 상속을 받았으며 대부분의 클래스는 String(문자열)을 매개변수로 받아서 변환해주는 parse로 시작하는 메소드를 소유하고 있고 기본형으로 값을 리턴하는 자료형 Value 메소드를 소유

❖ Auto Boxing

- ✓ Java에서 기본 자료형과 Wrapper 클래스의 인스턴스와의 형 변환을 자동으로 수행해주는 기능
- ✓ Wrapper class의 참조형 변수에 value 타입의 값을 직접 대입하는 것
- ✓ 반대로 value 타입의 변수에 Wrapper Class의 인스턴스 주소를 직접 대입하는 것도 가능한데 Auto UnBoxing이라고 함
- ✓ JDK 1.5 이전 버전에서는 프로그래머가 직접 형 변환을 수행해야 했지만 지금은 자동으로 자바 컴파일러가 수행

실습(Parse.java)

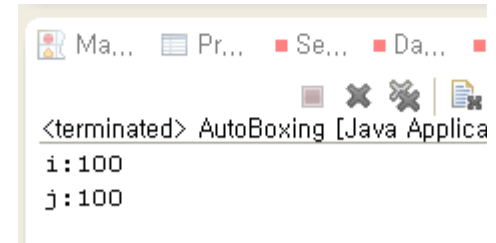
```
public class Parse {  
    public static void main(String[] args)  
    {  
        String str1 = "100100";  
        String str2 = "011";  
        Integer int1 = Integer.parseInt(str1);  
        Integer int2 = Integer.parseInt(str2);  
  
        System.out.println("int1:" + int1);  
        System.out.println("int2:" + int2);  
    }  
}
```

int1:100100
int2:11

실습(AutoBoxing.java)

```
class AutoBoxing
{
    public static void main(String[] args)
    {
        //100은 int 값인데 참조형 변수에 직접 대입
        Integer i = 100;
        //위의 문장은 Integer i = new Integer(100)으로 변환되어 수행됩니다.

        System.out.println("i:" + i.intValue());
        //인스턴스를 직접 value 형 변수인 j에 대입
        int j = i;
        //위의 문장은 int j = i.intValue()로 변환되어 수행됩니다.
        System.out.println("j:" + j);
    }
}
```



2. Wrapper 클래스

❖ BigInteger 클래스

- ✓ Short와 Integer 클래스는 아주 큰 수나 작은 수의 표현이 불가능한데 이 문제를 해결하기 위해서 등장한 클래스가 BigInteger 클래스
- ✓ 내부적으로 int 배열로 저장
- ✓ 생성은 생성자에 숫자로 구성된 문자열 대입해서 생성
- ✓ Number 타입으로 변환을 할 때는 intValue() 와 같은 메소드 이용

❖ BigDecimal 클래스

- ✓ Float과 Double 클래스는 정밀한 값의 표현이 불가능한데 문제를 해결하기 위해서 등장한 클래스가 BigDecimal 클래스
- ✓ 실수 사이의 연산을 수행하면 정확하게 이루어지지 않는 경우가 있는데 이를 보완하기 위한 목적으로도 사용
- ✓ MyBatis(iBatis)를 이용해서 Oracle 데이터베이스의 Number 타입의 데이터를 리턴받으면 BigDecimal 타입으로 리턴
- ✓ 생성은 생성자에 숫자로 구성된 문자열을 대입해서 생성
- ✓ Number 타입으로 변환을 할 때는 intValue() 와 같은 메소드 이용

❖ 2개의 클래스는 java.math 패키지에 존재

❖ 2개의 클래스도 인스턴스의 내용을 변경할 수 없음

❖ 산술 연산을 수행해주는 메소드가 별도로 존재

실습(BigData.java)

```
import java.math.*;
```

```
public class BigData {
```

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

```
        BigInteger bint = new BigInteger("10000000000000000000000000000000");
```

```
        BigDecimal bdec = new BigDecimal("0.000195312512345");
```

```
        int i = bint.intValue();
```

```
        float f = bdec.floatValue();
```

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

```
        System.out.println("bint:" + bint);
```

```
        System.out.println("f:" + f);
```

```
        System.out.println("bdec:" + bdec);
```

```
    }
```

```
}
```

```
i:-469762048  
bint:10000000000000000000000000000000  
f:1.9531252E-4  
bdec:0.000195312512345
```

실습(NoErrorBigDecimal.java)

```
import java.math.*;
```

```
class NoErrorBigDecimal
```

$$\{$$

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

$$\{$$

```
double d1 = 1.60000000000000000000000000;
```

```
double d2 = 0.10000000000000000000000000;
```

```
System.out.println("두 실수의 덧셈결과: "+ (d1+d2));
```

```
BigDecimal e1=new BigDecimal("1.6");
```

```
BigDecimal e2=new BigDecimal("0.1");
```

```
System.out.println("두 실수의 덧셈결과: " + e1.add(e2));
```

}

}

두 실수의 덧셈결과: 1.7000000000000002

두 실수의 덧셈결과: 1.7

3.System 클래스

- ❖System 클래스는 실행 환경과 관련된 속성과 메소드를 제공하는 클래스
- ❖모든 멤버가 static이므로 인스턴스 생성없이 사용
- ❖System 클래스의 클래스 변수 in과 out은 입출력 패키지의 InputStream 클래스와 PrintStream 클래스의 인스턴스를 리턴

3.System 클래스

❖ 메소드

<code>static void arraycopy(Object source, int sourceStart, Object target, int targetStart, int size)</code>	배열을 복사 source와 target 은 복사될 배열의 이름이고 sourceStart와 targetStart는 복사가 시작될 위치이고 size는 복사될 배열의 크기
<code>static long currentTimeMillis()</code>	1970년 1월 1일 자정부터 현재까지의 시간을 밀리초 단위로 반환
<code>static void exit(int exitcode)</code>	현재 수행중인 응용 프로그램을 종료 exitcode에 따라 메시지를 나타낼 수 있음 exitcode를 0으로 지정하면 정상적인 종료
<code>static void gc()</code>	쓰레기 수집(garbage collection)을 시작
<code>static void runFinalization()</code>	garbage collection 실제 수행
<code>static String getProperty(String key)</code>	Key에 해당하는 시스템의 속성값을 문자열로 리턴
<code>static long nanoTime()</code>	현재 시간을 나노 초 단위로 반환
<code>static Map<String, String>getenv()</code>	현재 시스템 환경을 스트링 형태의 맵으로 리턴
<code>static String getenv(String name)</code>	name에 해당하는 환경 변수의 값을 리턴
<code>public static int identityHashCode(Object x)</code>	x의 해시코드를 리턴 Object 클래스의 hashCode()는 재정의 되어 있는 경우가 많음

실습(StopWatch.java)

```
class StopWatch
{
    public static void main(String args[])
    {
        long start, end;
        int ar [] = new int [10000];
        //1970년 1월 1일부터 현재까지의 시간을 나노 초로 반환
        start = System.nanoTime();
        System.out.println("10000회의 Loop를 반복 시작" );
        int cnt = ar.length;
        for(int i=0; i < cnt ; i++)
        {
            System.out.println(ar[i]);
        }
        end = System.nanoTime();
        System.out.println("10000회 반복에 소요된 시간 : " + ( (((double)end -
start)/1000000000)+ "초" ));
    }
}
```

실습(SystemProperty.java)

운영체제 이름: Mac OS X

사용자 이름: munseokpark

사용자 홈디렉토리: /Users/munseokpark

자바 디렉토리: /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/jre

자바 버전: 1.8.0_181

[path] null



실습(SystemProperty.java)

```
public class SystemProperty {  
    public static void main(String[] args) throws Exception {  
        String osName = System.getProperty("os.name");  
        String userName = System.getProperty("user.name");  
        String userHome = System.getProperty("user.home");  
        String javaHome = System.getProperty("java.home");  
        String javaVersion = System.getProperty("java.version");  
  
        System.out.println("운영체제 이름: " + osName);  
        System.out.println("사용자 이름: " + userName);  
        System.out.println("사용자 홈디렉토리: " + userHome);  
        System.out.println("자바 디렉토리: " + javaHome);  
        System.out.println("자바 버전: " + javaVersion);  
  
        String path = System.getenv("path");  
        System.out.println("[ path ] " + path);  
    }  
}
```

4.Class 클래스

❖클래스 로더 : '.class' 바이트 코드를 읽어 들여 class 인스턴스를 생성하는 역할로 클래스가 요청될 때 파일로부터 읽어 메모리로 로딩하는 역할

❖클래스 로딩 과정

✓로딩(loading) : 클래스 파일을 바이트 코드로 읽어 메모리로 가져오는 과정

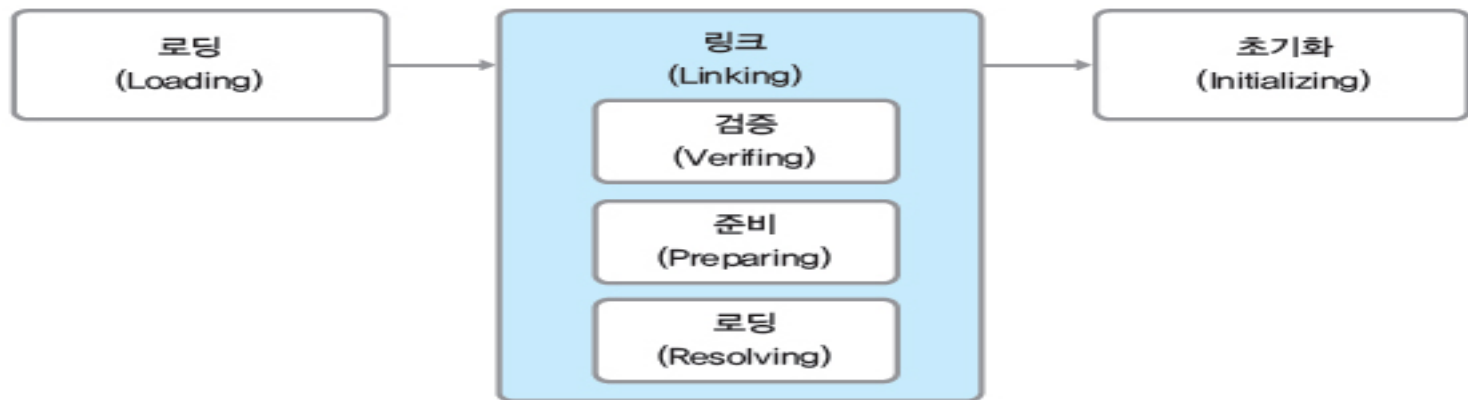
✓링크(linking) :

◆ 읽어온 바이트 코드가 자바 규칙을 따르는지 검증(Verifying)

◆ 클래스에 정의된 필드, 메소드, 인터페이스들을 나타내는 데이터 구조를 준비(Preparing)

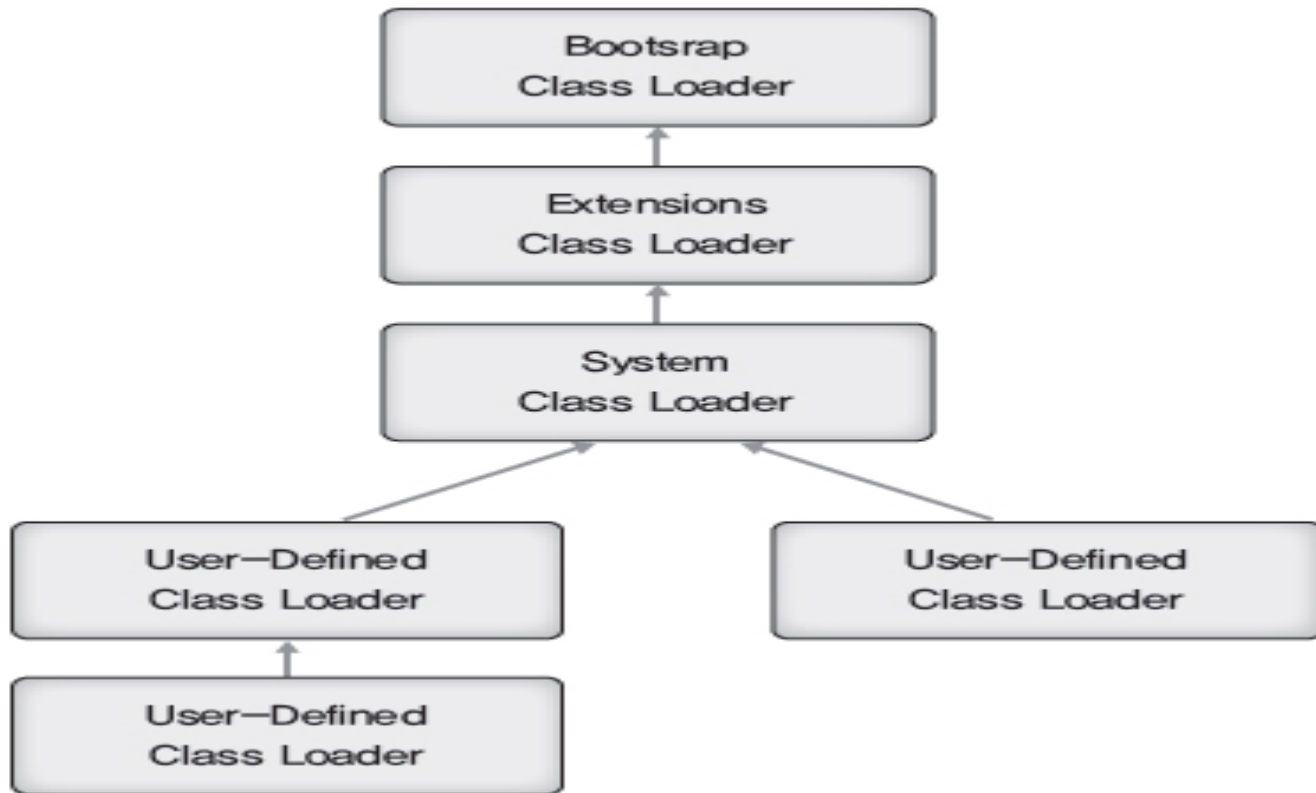
◆ 클래스가 참조하는 다른 클래스를 로딩(Loding)

✓초기화(Initializing) : 슈퍼 클래스 및 정적 필드들을 초기화



4.Class 클래스

❖Java의 클래스 로더



4.Class 클래스

❖Java의 클래스 로더

- ✓ 부트스트랩 클래스 로더(Bootstrap class loader)
 - JVM이 실행될 때 맨 처음 실행되는 클래스 로더
- ✓ 익스텐션 클래스 로더(Extensions class loader)
 - BootStrap Loading 후 기본적으로 로딩되는 클래스
- ✓ 시스템 클래스 로더(System class loader)
 - 다음으로 CLASS PATH에 정의되거나 JVM option에서 -cp, -classpath에 지정된 클래스들이 로딩
- ✓ 사용자가 정의하는 사용자 정의 클래스 로더(User-Defined class loader)
 - 사용자가 직접 생성해서 사용하는 클래스 로더

4.Class 클래스

❖인스턴스나 인터페이스의 실행 상태를 저장할 수 있는 클래스로 직접 인스턴스를 생성할 수는 없고 인스턴스의 getClass 메소드를 호출해서 리턴받아 사용하는데 개발자가 사용하는 경우보다는 프레임워크나 IDE에서 주로 사용

❖인스턴스 생성을 할 때 생성자를 이용하지 않고 forName 메소드와 newInstance 메소드를 이용해서 동적으로 인스턴스를 생성할 수 있음

❖메소드

String getName()	클래스의 이름을 반환
Class getClass()	클래스의 타입을 반환
Class getSuperclass()	상위 클래스의 이름을 반환
static Class forName(String name) throws ClassNotFoundException	name으로 지정된 문자열에 해당하는 Class 인스턴스를 반환하고 메모리에 로드
Object newInstance()	동적 인스턴스 생성

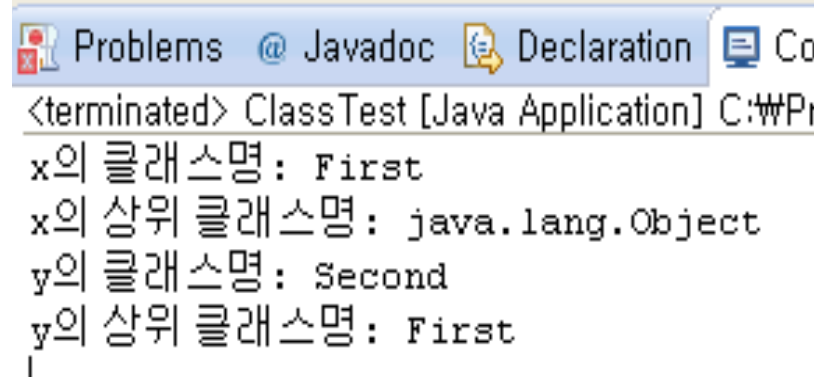
실습(etc-ClassTest.java)

```
class First
{}

class Second extends First
{}

public class ClassTest
{
    public static void main(String args[])
    {
        First x = new First();
        Second y = new Second();
        Class<?> cyberclass = x.getClass();

        System.out.println("x의 클래스명: " + cyberclass.getName());
        cyberclass = cyberclass.getSuperclass();
        System.out.println("x의 상위 클래스명: " + cyberclass.getName());
        cyberclass = y.getClass();
        System.out.println("y의 클래스명: " + cyberclass.getName());
        cyberclass = cyberclass.getSuperclass();
        System.out.println("y의 상위 클래스명: " + cyberclass.getName());
    }
}
```



<terminated> ClassTest [Java Application] C:\WPi
x의 클래스명: First
x의 상위 클래스명: java.lang.Object
y의 클래스명: Second
y의 상위 클래스명: First

실습(동적 인스턴스 생성)

```
import java.util.Date;
```

```
public class DynamicInstance {
```

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

```
        Class<?> clazz = Class.forName("java.util.Date");
```

```
        Date obj = (Date)clazz.newInstance();
```

```
        System.out.println(obj);
```

```
    }
```

```
}
```

5.Math 클래스

❖수학과 관련된 클래스로 모든 멤버가 static으로 구성되어 있으며 플랫폼에 따라서 메소드의 호출 결과가 달라질 수 있음

❖이 클래스 대용으로 StrictMath 클래스가 존재하는데 이 클래스는 모든 플랫폼에서 동일한 연산 결과를 리턴하기 때문에 Math 보다 유용하게 사용될 수 있지만 플랫폼의 특정 기능을 사용하지 않아서 연산 속도는 떨어짐

```
class MathTest
{
    public static void main(String args[])
    {
        System.out.println("파이 : " + Math.PI);
        System.out.println("10 절대값 : " + Math.abs(10));
        System.out.println("큰 값을 반환합니다. : " + Math.max(30, 60));
        System.out.println("작은 값을 반환합니다. : " + Math.min(10.0, 20.0));
    }
}
```

파이 : 3.141592653589793

10 절대값 : 10

큰 값을 반환합니다. : 60

작은 값을 반환합니다. : 10.0

6. Runtime

- ❖ Process 실행과 관련된 클래스
- ❖ 생성자를 이용하지 않고 `getRuntime()`이라는 메소드를 이용해서 인스턴스 생성
- ❖ `exec` 메소드를 이용해서 프로세스를 실행



6. Runtime

```
public class RuntimeTest {  
    public static void main(String[] args) {  
        try {  
            Runtime r1 = Runtime.getRuntime();  
            Runtime r2 = Runtime.getRuntime();  
            System.out.println("r1의 해시코드:" + r1.hashCode());  
            System.out.println("r2의 해시코드:" + r2.hashCode());  
  
            //Windows에서 실행  
            //Runtime.getRuntime().exec("notepad.exe " +  
            "c:\\Weula.1028.txt");  
  
            //Mac에서 실행 - Mac은 사용 권한 문제로 애플리케이션을 직  
            접 열지 못할 수도 있음  
            //Runtime.getRuntime().exec("open  
            /Users/munseokpark/Documents");  
            //Runtime.getRuntime().exec("open  
            /System/Applications/TextEdit.app");  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```


7.String 클래스

- ❖ 자바에서는 문자열을 인스턴스로 취급
- ❖ java.lang 패키지에 포함
- ❖ java.lang.String 클래스와 java.lang.StringBuffer와 java.lang.StringBuilder 클래스 제공
- ❖ String 클래스: 한번 생성되면 내부 문자열을 변경할 수 없는 클래스
- ❖ StringBuffer 클래스: 멀티 스레드에 안전한 내부 문자열을 변경할 수 있는 클래스
- ❖ StringBuilder 클래스: 멀티 스레드에 안전하지 않은 내부 문자열을 변경할 수 있는 클래스
- ❖ 문자열을 여러 개의 클래스로 제공하는 이유: 효율성

❖ 생성자

- ✓ String()
- ✓ String(byte[] bytes)
- ✓ String(byte[] bytes, Charset charset)
- ✓ String(char chars[])
- ✓ String(char chars[], int startindex, int numChars)
- ✓ String(String strObj)
- ✓ String 클래스는 문자열 상수를 가지고 생성할 수 있고 문자열 상수를 직접 대입해서 생성할 수 있는데 이 경우 동일한 문자열을 대입하면 동일한 해시코드를 갖게 됨

실습(StringCreate.java)

```
class StringCreate
{
    public static void main(String args[])
    {
        char str[] = {'C','o','m','p','u','t','e','r'};
        String s1 = new String(str);
        // 문자배열로부터 문자열 인스턴스 s1 생성
        String s2 = new String(str,3,2);
        // 문자배열로부터 부분 문자열을 추출하여 s2 생성
        String s3 = new String("Java 문자열");
        // 문자열로부터 직접 인스턴스 s3 생성
        String s4 = "Java 문자열";
        String s5 = "Java 문자열";

        System.out.println("문자열 s1 : " + s1);
        System.out.println("문자열 s2 : " + s2);
        System.out.println("문자열 s3 : " + s3);
        System.out.println("문자열 s4 : " + s4);

        System.out.println("s3의 해시코드:" + System.identityHashCode(s3));
        System.out.println("s4의 해시코드:" + System.identityHashCode(s4));
        System.out.println("s4의 해시코드:" + System.identityHashCode(s5));
    }
}
```

문자열 s1 : Computer
문자열 s2 : pu
문자열 s3 : Java 문자열
문자열 s4 : Java 문자열
s3의 해시코드:2018699554
s4의 해시코드:1311053135
s4의 해시코드:1311053135

7.String 클래스

❖메소드

- ✓ `int length()`: 문자열의 길이를 반환
- ✓ `char charAt(int i)`: 문자열중에서 i번째 문자를 반환
- ✓ `void getChars(int sourceStart, int sourceEnd, char target[], int targetStart)`: 문자열의 일부를 문자 배열로(target[])제공.
- ✓ `String format(String format, Object args)`: 포맷에 맞추어 문자열을 생성
- ✓ `int indexOf(int ch)`: ch가 문자열에 있으면 첫 번째 위치 반환하고 없으면 -1을 반환
- ✓ `int indexOf(int ch, int fromIndex)`: ch가 문자열에 있으면 첫 번째 위치 반환하고 없으면 -1을 반환하는데 fromindex로부 시작
- ✓ `int indexOf(String str)`: str이 있으면 str의 위치를 반환하고 없으면 -1을 반환
- ✓ `int indexOf(String str, int fromIndex)`
- ✓ `int lastIndexOf(int ch)`: 마지막 위치 반환
- ✓ `int lastindexOf(int ch, int startIndex)`
- ✓ `int lastindexOf(String str, int startIndex)`

7.String 클래스

❖메소드

- ✓ String replace(CharSequence target, CharSequence replacement): target의 문자열을 replacement 문자열로 치환
- ✓ String toLowerCase(): 모두 소문자로 변환해서 리턴
- ✓ String toUpperCase(): 모두 대문자로 변환해서 리턴
- ✓ String trim(): 좌우 공백 제거
- ✓ boolean equals(Object str) : str로 지정된 문자열과 현재의 문자열이 같은지를 비교
- ✓ boolean equalsIgnoreCase(String str): 문자열 비교 시 대소문자 무시
- ✓ boolean startsWith(String str): 현재의 문자열이 str로 지정된 문자열로 시작하면 true, 아니면 false를 반환
- ✓ boolean endsWith(String str): 문자열이 str로 끝나면 true, 아니면 false
- ✓ String[] split(String regex): regex 단위로 분할
- ✓ String substring(int beginIndex) : beginIndex부터 끝까지 문자열을 분할
- ✓ String substring(int beginIndex, int endIndex) : beginIndex 번째 부터 endIndex-1 번째 까지 문자열을 분할

실습(StringMethod.java)

```
public class StringMethod {  
    public static void main(String[] args) {  
        String str1 = "Test" ;  
        String str2 = "test";  
        System.out.println("str1과 str2 비교 : " + str1.equals(str2));  
        System.out.println("str1와 str2 비교(대소문자 무시) : " + str1.equalsIgnoreCase(str2));  
        String str = " Park Moon Seok ";  
        System.out.println(str.trim());  
        System.out.println("indexOf(o) = " + str.indexOf('o'));  
        System.out.println("lastIndexOf(o) = " + str.lastIndexOf('o'));  
  
        String [] ar = str.split(" ");  
        int cnt = ar.length;  
        for(int i=0; i<cnt; i++){  
            System.out.println(ar[i]);  
        }  
    }  
}
```

str1과 str2 비교 : false
str1와 str2 비교(대소문자 무시) : true
Park Moon Seok
indexOf(o) = 7
lastIndexOf(o) = 13

Park
Moon
Seok

7.String 클래스

❖format(“서식”, 대상)

- ✓ 5.0에서 새로 추가된 메소드로 포맷에 맞춰서 문자열을 생성한 후 리턴
- ✓ 이 형식은 콘솔에 문자열을 출력할 때 System.out.printf()에 그대로 적용 가능
- ✓ 서식
 - %d: 10진 정수 출력
 - %5d: 자리 출력으로 빈 자리는 공백
 - %-5d: 왼쪽 맞춤을 수행하고 5자리로 출력하고 빈 자리는 공백
 - %0d: 5자리로 출력하고 빈 자리는 0
 - %o: 8진수 출력
 - %x: 16진수
 - %c: 문자 출력
 - %e: 지수 형태로 실수 출력
 - %f: 고정 소수점 형태로 실수 출력
 - %3.1f: 최소 3자리 소수는 1자리(2번째 자리에서 반올림)
 - %03.1f: 최소 3자리 소수는 1자리(2번째 자리에서 반올림), 빈자리는 0
 - %-3.1f: 최소 3자리 소수는 1자리(2번째 자리에서 반올림), 왼쪽 맞춤
 - %s: 문자열
 - %5s: 문자열을 5자리로 출력하고 빈 자리는 공백
 - %-5s: 문자열을 5자리로 출력하고 빈자리는 공백으로 출력하고 왼쪽 맞춤

7.String 클래스

- ✓ 서식(Date)
 - %tR : 시분(24시간)
 - %tH:%M : 시분(24시간)

 - %tT : 시분초(24시간)
 - %tH:%H:%S : 시분초(24시간)

 - %tD : 월일년
 - %tm:%d:%y : 월일년

 - %tF : 년월일
 - %ty:%m:%d : 년월일

실습(StringFormat.java)

```
class StringFormat
{
    public static void main(String args[])
    {
        String str ;
        float x = 10.7654f;
        float y = 11.74754f;
        str = String.format("%d", 20);
        System.out.println("str = " + str);

        str = String.format("%10d", 20);
        System.out.println("str = " + str);

        str = String.format("x:%.1f y:%.1f", x, y);
        System.out.println("str = " + str);
    }
}
```

```
str = 20
str =          20
str = x:10.8 y:11.7
```


7.String 클래스

- ❖getBytes(“인코딩방식”) 메소드를 이용하면 문자열을 매개변수의 인코딩 방식으로 변환해서 바이트 배열로 리턴
- ❖바이트 배열로 String 인스턴스를 생성할 때는 new String(바이트 배열, “인코딩방식”)을 이용해서 생성하면 매개변수의 인코딩 방식으로 String 인스턴스를 생성
- ❖한글에서 사용되는 인코딩 방식으로는 MS949, EUC-KR, UTF-8, 8859_1 등이 있음

실습(EncodingConvert.java)

```
import java.io.*;

public class EncodingConvert {
    public static void main(String args[]) {
        String str = "대한민국";
        try {
            byte[] by = str.getBytes("MS949");
            System.out.println("str:" + str);
            String msg = new String(by, "UTF-8");
            System.out.println("msg:" + msg);
        } catch (UnsupportedEncodingException e) {
            System.out.println("인코딩을 지원하지 않습니다.");
        }
    }
}
```

Problems @ Javad
<terminated> Encoding
str:대한민국
msg:?????α?

8.StringBuilder 클래스

❖변할 수 있는 문자열을 저장하며 스레드에 안전하지 않은 클래스

❖생성자

- ✓ `StringBuilder()`: 묵시적으로 16개의 문자를 저장할 수 있는 인스턴스를 생성
- ✓ `StringBuilder(int capacity)`: capacity 크기의 인스턴스를 생성
- ✓ `StringBuilder(CharSequence seq)`: str로 지정된 문자열과 추가로 16개의 문자를 더 저장할 수 있는 인스턴스를 생성
- ✓ `StringBuilder` 인스턴스는 인스턴스의 크기가 변할 때 마다 메모리를 재 할당(16개의 문자를 저장할 수 있는 버퍼 단위로)

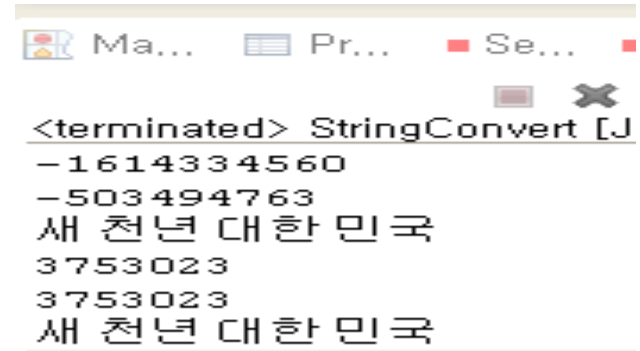
❖메소드

- ✓ `StringBuilder append(문자열)`: 문자열을 추가
- ✓ `int capacity()`: 현재의 용량
- ✓ `char charAt(int index)`
- ✓ `StringBuffer delete (int start, int end)`
- ✓ `StringBuffer deleteCharAt (int index)`
- ✓ `int indexOf (String str)`
- ✓ `int indexOf (String str, int fromIndex)`
- ✓ `StringBuffer insert (int offset, String str)`
- ✓ `int length()`
- ✓ `String substring (int start)`

실습(StringConvert.java)

```
class StringConvert
{
    public static void main(String args[])
    {
        String st1 = "새 천년 ";
        System.out.println(st1.hashCode());
        String st2 = "대한 민국";
        st1 = st1 + st2;
        System.out.println(st1.hashCode());
        System.out.println(st1);

        StringBuilder sb1 = new StringBuilder("새 천년 ");
        System.out.println(sb1.hashCode());
        sb1.append("대한 민국");
        System.out.println(sb1.hashCode());
        System.out.println(sb1);
    }
}
```



```
<terminated> StringConvert [J
-1614334560
-503494763
새 천년 대한 민국
3753023
3753023
새 천년 대한 민국
```

9. 주요 인터페이스

- ❖ Cloneable: Object.clone() 메소드 재정의를 명시적으로 표현하기 위한 인터페이스
- ❖ Comparable: int compareTo(T o) 메소드 재정의를 위해서 사용하는 인터페이스로 이 메소드가 재정의되어 있는 클래스의 인스턴스 컬렉션은 sort() 메소드에 의해 정렬
- ❖ Iterable: Iterator iterator () 메소드를 재정의해서 반복자를 이용한 모든 데이터 접근이 가능
- ❖ Runnable: run()메소드를 재정의해서 스레드를 생성할 수 있도록 해주는 인터페이스
- ❖ Serializable: 파일 저장과 네트워크 전송이 가능하다는 것을 명시적으로 표현하기 위한 인터페이스
- ❖ Listener: Java나 Android의 GUI 이벤트 처리 인터페이스

10. 빠른 열거

❖ 1.5에서 추가된 문법으로 for(임시변수:컬렉션이름)의 형태를 사용하게 되면 컬렉션에 있는 모든 요소들을 순서대로 임시변수에 대입

```
public class FastEnumeration {  
    public static void main(String[] args) {  
        int[] ar = { 200, 230, 150, 100, 190 };  
        long start, end;  
        start = System.nanoTime();  
        for (int i = 0; i < ar.length; i++)  
            System.out.print(ar[i] + "Wt");  
        System.out.println("");  
        end = System.nanoTime();  
        System.out.println(end-start);  
  
        System.out.println("=====");  
        start = System.nanoTime();  
        for (int i : ar)  
            System.out.print(i + "Wt");  
        System.out.println("");  
        end = System.nanoTime();  
        System.out.println(end-start);  
  
    }  
}
```

11. Generics

- ❖ 데이터 타입을 컴파일 할 때 결정하지 않고 실행 할 때 결정하는 것
- ❖ 클래스 내부에서 사용할 데이터 타입을 인스턴스를 생성할 때 결정하는 것
- ❖ 다른 언어의 템플릿 기능과 유사하며 스크립트 언어에서 자료형을 선언하지 않고 변수를 만들고 데이터가 대입될 때 타입이 결정되는 것과 유사한 기능
- ❖ Generic의 장점
 - ✓ 인스턴스의 타입을 실행 시에 결정해서 인스턴스의 타입 안정성을 높이고 형 변환의 번거움을 줄여줌
 - ✓ 의도하지 않은 타입의 인스턴스가 삽입되는 것을 방지
 - ✓ 인스턴스를 꺼내서 사용할 때 잘못된 형 변환으로 인한 ClassCastException 방지
- ❖ 형식
 - ✓ 선언 : class 클래스이름<미지정 자료형 이름 ...>
 - ✓ 인스턴스 생성시 : 클래스이름 <확정자료형> 인스턴스명 = new 생성자이름 <확정자료형>
 - ✓ 확정 자료형은 일반 value 타입은 안되고 reference 타입 만 가능
 - ✓ 미지정 자료형 이름을 만들 때 일반적으로 대문자 한 글자를 이름으로 사용하는데 가급적 의미가 있는 대문자를 사용하는 것이 좋는데 E는 element, T는 type, K는 key, V는 value와 같이 지정
 - ✓ 1.7 버전 이상에서는 인스턴스를 생성할 때 확정 자료형을 생략하고 <>으로만 기재해도 가능

실습(Generic.java)

```
public class Generic <T>{
    private T data[];
    public Generic(T...n){
        data = n;
    }
    public void disp(){
        for(T i : data){
            System.out.print(i + " ");
        }
        System.out.println();
    }
    public static void main(String[] args){
        Generic <String> Obj1 = new Generic <String>("Hello", "Java");
        Generic <String> Obj2 = new Generic <String>("Hello", "Java", "Test");
        Generic <Integer> Obj3 = new Generic <Integer>(1,2,3);
        Obj1.disp();
        Obj2.disp();
        Obj3.disp();
    }
}
```

Hello Java
Hello Java Test
1 2 3

11. Generics

- ❖ 메소드의 리턴 타입이나 매개변수로 Generics 사용 가능
- ❖ static 멤버에는 Generics를 적용할 수 없음
- ❖ Generics를 이용해서 배열을 생성하는 것은 안됨
- ❖ 배열 변수를 만드는 것은 허용하며 참조하는 것도 허용안됨
- ❖ <타입 extends 인터페이스나 클래스>를 설정하게 되면 인터페이스나 클래스 또는 하위 클래스 타입만 가능
- ❖ <? extends T>: T 또는 하위 타입만 가능
- ❖ <? super T>: T 또는 상위 타입만 가능
- ❖ <?>: 모든 데이터 타입 가능
- ❖ Generics가 적용된 데이터와 Generics가 적용되지 않은 데이터 간의 형 변환은 허용하지만 경고가 발생
- ❖ 컴파일이 되서 .class 파일로 만들어질 때는 Generic은 모두 사라지고 형 변환으로 모두 대체

12. enum

❖ 자바에서 열거형 상수를 표현할 때 이전에는 클래스 안에 final static 상수를 선언해서 사용했는데 1.5 버전에서 enum을 지원하고 enum(나열형 상수)도 하나의 클래스로 간주

❖ 선언 방법

```
enum 이름{  
    멤버 이름 나열;  
}
```

❖ 각 멤버는 enum 이름으로 접근이 가능(public static final 필드로 간주)

❖ 다른 클래스와 별도로 선언하며 값을 제한하는 역할을 수행하는데 일반 자료형과 동일한 값을 지정했다 하더라도 다른 enum의 값과는 비교할 수 없는데 이는 컴파일 시 타입을 비교해서 동일한 타입이 아니면 비교가 되지 않도록 하기 때문

실습(Enum.java)

```
enum Telecom {  
    SKT, KT, LGT, Other;  
}
```

t:SKT

```
public class Enum{  
    public static void main(String args[]){  
        Telecom t = Telecom.SKT;  
        System.out.println("t:" + t);  
    }  
}
```

12. enum

- ❖ 생성자 및 일반 메소드도 선언이 가능
- ❖ 각 멤버는 enum으로 만들어진 인스턴스
- ❖ 멤버 이름을 나열할 때 생성자의 호출이 가능
- ❖ 열거형 상수끼리는 == 으로 비교가 가능하며 compareTo()로도 비교가 가능하지만 >, <를 이용한 비교는 불가능
- ❖ 메소드
 - public final int ordinal(): enum의 인덱스 리턴 - 0부터 시작
 - public String name(): 상수의 이름을 문자열로 반환
 - public Class<E> getDeclaringClass(): 열거형의 Class 인스턴스를 반환
 - public static <E>[] values(): 모든 멤버를 배열로 리턴
 - public static <E> valueOf(String name): 모든 멤버를 배열로 리턴

실습(Enum1.java)

```
enum TelecomCompany {  
    SKT("011"), KT("016"), LGT("019"), Other("MVNO");  
    String number;  
    TelecomCompany(String num){  
        number = num;  
    }  
  
    public String getNumber(){  
        return number;  
    }  
}
```

t의 번호:011
SKT:0
KT:1
LGT:2
Other:3

```
public class Enum1{  
    public static void main(String args[]){  
        TelecomCompany t = TelecomCompany.SKT;  
        System.out.println("t의 번호:" + t.getNumber());  
  
        TelecomCompany [] mobile = TelecomCompany.values();  
        for(TelecomCompany m : mobile){  
            System.out.println(m + ":" + m.ordinal());  
        }  
    }  
}
```

13.Annotation

❖ 1.5에서 부터 지원되는 문법으로 @를 이용해서 주석을 만들거나 자바코드에 특별한 의미를 부여하는 것을 Annotation

❖ 용도

❖ 컴파일러가 특정 오류를 체크하도록 지시

❖ IDE가 Build 나 Batch를 할 때 코드를 자동 생성 - 설정에 이용한 XML 파일을 읽어서 자바 코드로 변환

❖ Runtime 시 특정 코드 실행

❖ JDK 제공 기본 Annotation

- ✓ @Override : 메소드를 재정의(Overriding)할 것임을 컴파일러에게 알려주는 역할
- ✓ @Deprecated : 클래스, 메소드, 필드 등에 선언하며 지정한 요소를 더 이상 사용하지 않는 것을 권장(가급적 사용을 자제해달라는 의미)
- ✓ @SuppressWarnings : 클래스, 메소드, 필드의 선언, 컴파일러의 경고를 제거(이 부분에 대해서 경고문을 출력하지 말라는 의미)
- ✓ @SafeVarargs: Varargs(매개변수의 개수를 가변으로 하는 메소드)에 Generic을 사용한 다는 것을 컴파일러에게 알림
- ✓ @FunctionalInterface: 함수형 인터페이스임을 컴파일러에게 알림 - 메소드가 1개인 인터페이스

13.Annotation

❖ Annotation 정의와 사용

- ✓ 어노테이션 타입 정의

```
public @interface AnnotationName {  
}
```

- ✓ 어노테이션 타입 적용

```
@AnnotationName
```

13.Annotation

❖ Element

✓ 생성

```
public @interface AnnotationName {  
    String value();  
    int elementName() default 5;  
}
```

----- 기본 엘리먼트 선언

✓ 사용 - 1개 일 때는 이름을 생략할 수 있지만 2개 이상일 때는 생략 불가

```
@AnnotationName("값");
```

```
@AnnotationName(value="값", elementName=3);
```


13.Annotation

❖ 적용 대상

- ✓ 코드 상에서 어노테이션을 적용할 수 있는 대상
- ✓ `java.lang.annotation.ElementType` 열거형 상수로 정의

ElementType 열거 상수	적용 대상
TYPE	클래스, 인터페이스, 열거 타입
ANNOTATION_TYPE	어노테이션
FIELD	필드
CONSTRUCTOR	생성자
METHOD	메소드
LOCAL_VARIABLE	로컬 변수
PACKAGE	패키지

13.Annotation

❖ 적용 대상

```
@Target(ElementType.TYPE, Element.FIELD, ElementType.METHOD)
public @interface AnnotationName{
}
```

```
@AnnotationName
public class ClassName{
    @AnnotationName
    private int fieldname;
```

```
    @AnnotationName
    public void methodName(){
    }
```

```
    @AnnotationName{
    public ClassName(){
    }
```

```
}
```

에러 - Target에 Constructor가
포함되지 않음

13.Annotation

❖ 유지 정책

- ✓ 어노테이션 적용 코드가 유지되는 시점을 지정하는 것
- ✓ `java.lang.annotation.RetentionPolicy` 열거형 상수로 정의

RetentionPolicy 열거 상수	설명
SOURCE	소스상에서만 어노테이션 정보를 유지한다. 소스 코드를 분석할 때만 의미가 있으며, 바이트 코드 파일에는 정보가 남지 않는다.
CLASS	바이트 코드 파일까지 어노테이션 정보를 유지한다. 하지만 리플렉션을 이용해서 어노테이션 정보를 얻을 수는 없다.
RUNTIME	바이트 코드 파일까지 어노테이션 정보를 유지하면서 리플렉션을 이용해서 런타임에 어노테이션 정보를 얻을 수 있다.

❖ 리플렉션(Reflection): 런타임에 클래스의 메타 정보를 얻는 기능

- ✓ 클래스가 가지고 있는 필드, 생성자, 메소드, 어노테이션의 정보를 얻을 수 있음
- ✓ 런타임 시 어노테이션 정보를 얻으려면 유지 정책을 RUNTIME으로 설정

13.Annotation

❖ 런타임시 어노테이션 정보 사용하기

- ✓ 클래스에 적용된 어노테이션 정보 얻을 때는 클래스.class 의 어노테이션 정보를 얻는 메소드 이용
- ✓ 전체 필드, 생성자, 메소드에 적용된 어노테이션 정보 얻기

리턴타입	메소드명(매개변수)	설명
Field[]	getFields()	필드 정보를 Field 배열로 리턴
Constructor[]	getConstructors()	생성자 정보를 Contructor 배열로 리턴
Method[]	getDeclaredMethods()	메소드 정보를 Method 배열로 리턴

13.Annotation

- ❖ 런타임시 어노테이션 정보 사용하기
 - ✓ 각각의 필드, 생성자, 메소드에 적용된 어노테이션 정보 얻기

리턴타입	메소드명(매개변수)
boolean	isAnnotationPresent(Class<? extends Annotation> annotationClass) 지정한 어노테이션이 적용되었는지 여부. Class 에서 호출했을 경우 상위 클래스에 적용된 경우에도 true 를 리턴한다.
Annotation	getAnnotation(Class<T> annotationClass) 지정한 어노테이션이 적용되어 있으면 어노테이션을 리턴하고 그렇지 않다면 null 을 리턴한다. Class 에서 호출했을 경우 상위 클래스에 적용된 경우에도 어노테이션을 리턴한다.
Annotation[]	getAnnotations() 적용된 모든 어노테이션을 리턴한다. Class 에서 호출했을 경우 상위 클래스에 적용된 어노테이션도 모두 포함한다. 적용된 어노테이션이 없을 경우 길이가 0 인 배열을 리턴한다.
Annotation[]	getDeclaredAnnotations() 직접 적용된 모든 어노테이션을 리턴한다. Class 에서 호출했을 경우 상위 클래스에 적용된 어노테이션은 포함되지 않는다.

실습(SampleAnnotation)

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
```

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface SampleAnnotation {
    String value() default "-";
    int number() default 15;
}
```

[method3]

실행 내용3

[method2]

실행 내용2

[method1]

실행 내용1

실습(SampleService)

```
public class SampleService {  
    @SampleAnnotation  
    public void method1() {  
        System.out.println("실행 내용1");  
    }  
  
    @SampleAnnotation("*")  
    public void method2() {  
        System.out.println("실행 내용2");  
    }  
  
    @SampleAnnotation(value="#", number=20)  
    public void method3() {  
        System.out.println("실행 내용3");  
    }  
}
```

실습(SampleAnnotationTest)

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

public class SampleAnnotationTest {
    public static void main(String[] args) {
        //Service 클래스로부터 메소드 정보를 얻음
        Method[] declaredMethods = SampleService.class.getDeclaredMethods();

        //Method 객체를 하나씩 처리
        for(Method method : declaredMethods) {
            //PrintAnnotation이 적용되었는지 확인
            if(method.isAnnotationPresent(SampleAnnotation.class)) {
                //PrintAnnotation 객체 얻기
                SampleAnnotation sampleAnnotation =
method.getAnnotation(SampleAnnotation.class);

                //메소드 이름 출력
                System.out.println "[" + method.getName() + " ] ";
                //구분선 출력
                for(int i=0; i<sampleAnnotation.number(); i++) {
                    System.out.print(sampleAnnotation.value());
                }
                System.out.println();
            }
        }
    }
}
```


실습(SampleAnnotationTest)

```
try {  
    //메소드 호출  
    method.invoke(new SampleService());  
} catch (Exception e) {}  
System.out.println();  
}  
  
}  
  
}
```



연습문제

- ❖ 두번째 문자열에서 첫번째 문자열을 제외한 부분을 가져오는 코드를 작성

```
String urlString = https://localhost:8080/insert.jsp”;
```

```
String contextPath = https://localhost:8080”;
```

결과 : </insert.jsp>

- ❖ 문자열에서 숫자, 소문자, 대문자 그리고 기타 문자의 의 개수를 세는 프로그램을 작성

```
String password = “HelloJava12!!!”;
```

결과

대문자:2개

소문자:7개

숫자:2개

기타문자:3개

연습문제

❖ anagram 인지 판별하는 예제

- ✓ 문자의 구성은 같고 순서가 다른 문자열을 anagram이라고 합니다.
- ✓ eros 와 rose는 anagram

입력: rose

출력: eros 와 rose는 anagram

입력: orse

출력: eros 와 orse는 anagram이 아님