

11-1. java.lang 패키지

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 자바 API 문서
- API 문서에서 클래스 페이지 읽는 방법
- Class 클래스
- String 클래스
- Wrapper 클래스
- Math 클래스
- 키워드로 끝내는 핵심 포인트
- 확인문제



시작하기 전에

[핵심 키워드] : Object 클래스, System 클래스, Class 클래스, String 클래스, Wrapper 클래스, Math 클래스

[핵심 포인트]

java.lang 패키지는 자바 프로그램의 기본적인 클래스를 담은 패키지이다. 그래서 그 클래스와 인터페이스는 import 없이 사용할 수 없다. java.lang 패키지에 속하는 주요 클래스에 대해 알아본다



❖ java.lang 패키지의 주요 클래스와 용도

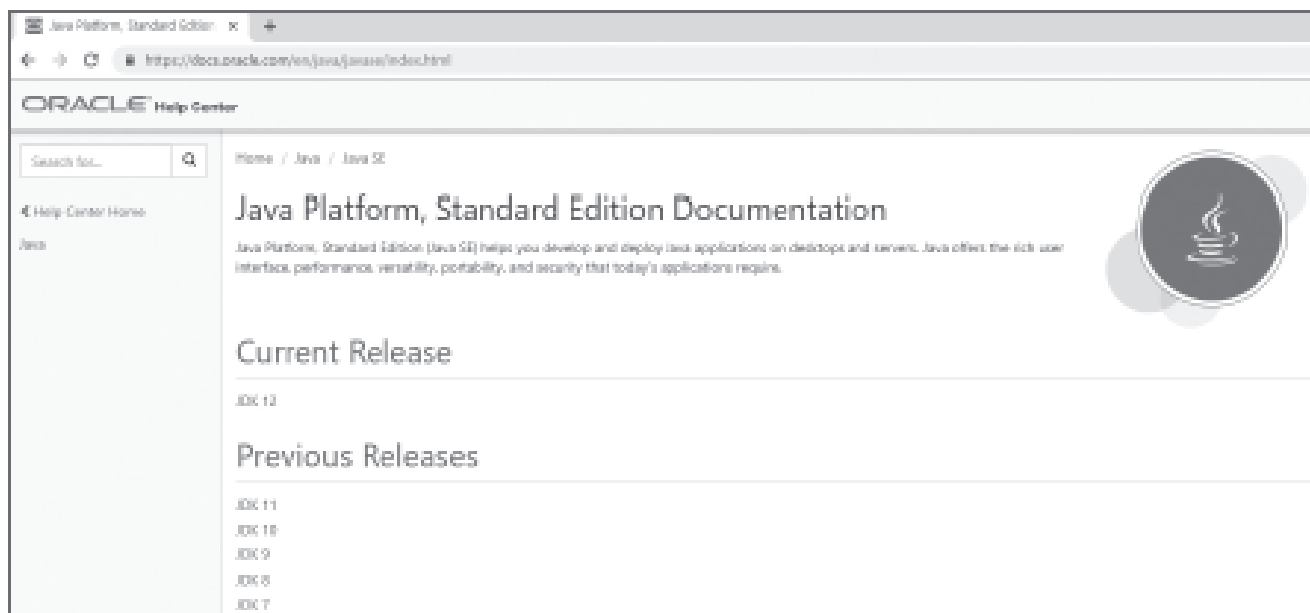
클래스		용도
Object		- 자바 클래스의 최상위 클래스로 사용
System		- 표준 입력 장치(키보드)로부터 데이터를 입력받을 때 사용 - 표준 출력 장치(모니터)로 출력하기 위해 사용 - 자바 가상 기계를 종료할 때 사용 - 쓰레기 수집기를 실행 요청할 때 사용
Class		- 클래스를 메모리로 로딩할 때 사용
String		- 문자열을 저장하고 여러 가지 정보를 얻을 때 사용
Wrapper	Byte, Short, Character Integer, Float, Double Boolean, Long	- 기본 타입의 데이터를 갖는 객체를 만들 때 사용 - 문자열을 기본 타입으로 변환할 때 사용 - 입력값 검사에 사용
Math		- 수학 함수를 이용할 때 사용



❖ API (Application Programming Interface)

- 라이브러리
- 프로그램 개발에 자주 사용되는 클래스 및 인터페이스 모음
- API 도큐먼트로 원하는 API 쉽게 찾아 이용할 수 있음

<https://docs.oracle.com/en/java/javase/index.html>



자바 API 도큐먼트

JDK 8 API 도큐먼트

The screenshot shows the JDK 8 API documentation page. The left sidebar contains a list of packages under 'All Packages' and a list of all classes. The main content area is titled 'Java™ Platform, Standard Edition 8 API Specification'. It includes a description of the document and a list of profiles: 'compliance7', 'compliance8', and 'compliance9'. Below this, there is a table with the following data:

Package	Description
java.applet	Provides the classes to create applets.
java.awt	Contains all of the classes for the Abstract Window Toolkit (AWT).
java.awt.color	Provides classes for color management.
java.awt.dnd	Provides methods and classes for drag and drop.
java.awt.font	Provides methods and classes for text layout.

JDK 11 이후 버전 API 도큐먼트

The screenshot shows the JDK 11 API documentation page. The left sidebar contains a list of packages under 'All Packages' and a list of all classes. The main content area is titled 'Java™ Platform, Standard Edition & Java Development Kit Version 11 API Specification'. It includes a description of the document and a list of profiles: 'compliance11', 'compliance12', and 'compliance13'. Below this, there is a table with the following data:

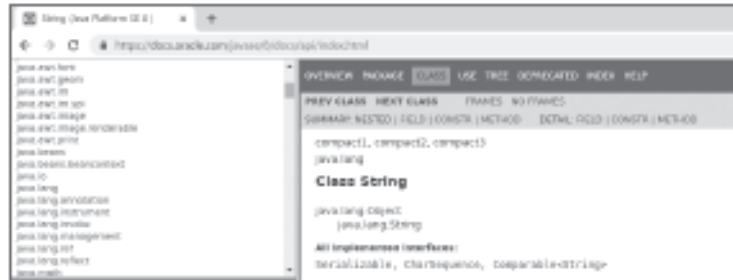
Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for Accessibility, audio, imaging, printing, and text layout.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.

자바 API 도큐먼트

각 버전의 API 도큐먼트에서 java.lang 패키지에 포함된 String 클래스 찾기

JDK 8 API 도큐먼트

1. 왼쪽 상단 Packages 목록에서 java.lang 패키지 링크를 찾아 클릭합니다.
2. 왼쪽 하단에 java.lang 패키지의 내용이 나옵니다.
3. Classes 목록에서 String 클래스 링크를 찾아 클릭합니다.
4. 오른쪽에 Class String 페이지가 나옵니다.



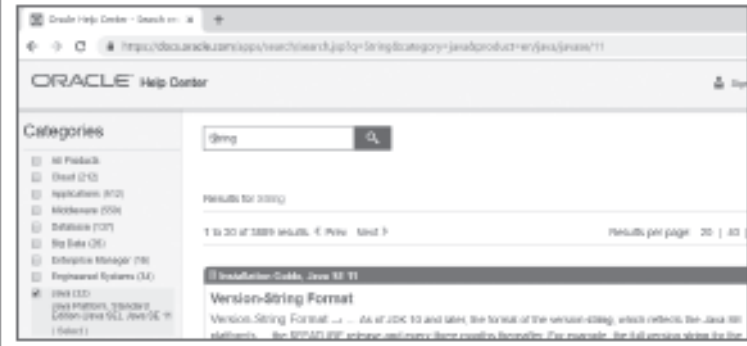
JDK 11 이후 버전 API 도큐먼트

[방법1]

1. All Modules 목록에서 java.base 링크를 찾아 클릭합니다.
2. java.base 모듈 페이지의 Packages 목록에서 java.lang 패키지 링크를 찾아 클릭합니다.
3. java.lang 패키지 페이지의 Class Summary 목록에서 String 클래스 링크를 찾아 클릭합니다.

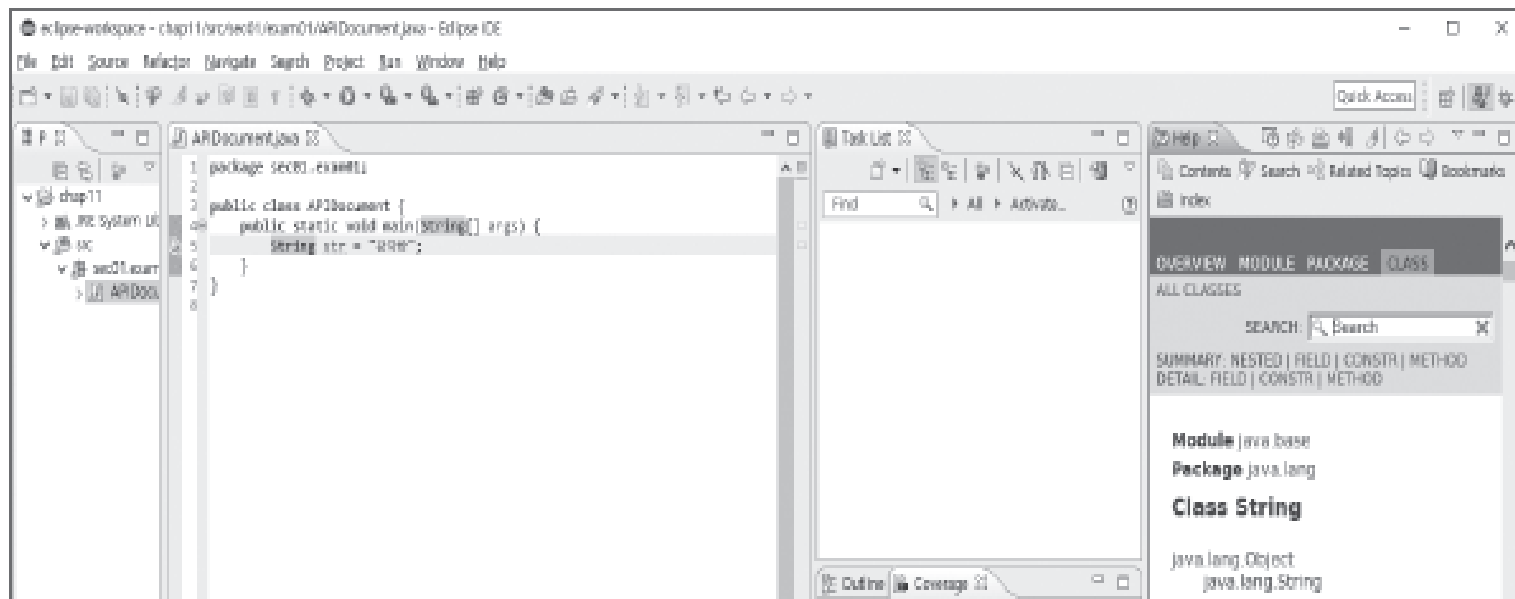
[방법2]

1. 오른쪽 상단의 Search 검색란에 "String"을 입력합니다.
2. 드롭다운 목록에서 java.lang.String 항목을 선택합니다.



자바 API 도큐먼트

- 이클립스에서는 코드 편집 뷰의 String 클래스 선택 후 F1 키 클릭 - Help 뷰로 이동 - java.lang.String 링크 클릭



API 도큐먼트에서 클래스 페이지 읽는 방법

❖ 최상단 SUMMARY: NESTED | FIELD | CONSTR | METHOD

- SUMMARY : 클래스 내 선언된 멤버 어떤 것들 있는지 알려줌
- NESTED : 중첩 클래스 혹은 인터페이스 여부

❖ 그림에서 ① 클래스의 선언부

- final 혹은 abstract 키워드 있는지 확인
- extends 뒤 언급된 부모 클래스 확인
- implements 키워드 뒤의 인터페이스 확인

```
Module java.base
Package java.lang
Class String
  java.lang.Object ②
  java.lang.String
All implemented interfaces:
  Serializable, CharSequence, Comparable<String>
public final class String ①
  extends Object
  implements Serializable, Comparable<String>, CharSequence
```

❖ 클래스에 선언된 필드 목록

- SUMMARY: NESTED | FIELD | CONSTR | METHOD 에서 FIELD 클릭하여 필드 목록으로 이동

Field Summary		
Fields		
Modifier and Type	Field	Description
static Comparator<String>	CASE_INSENSITIVE_ORDER	A Comparator that orders String objects as by compareToIgnoreCase.



API 도큐먼트에서 클래스 페이지 읽는 방법

❖ 클래스에 선언된 생성자 목록

- SUMMARY: NESTED | FIELD | CONSTR | METHOD 에서 CONSTR 클릭하여 생성자 목록으로 이동
- 생성자 이름 클릭하면 상세 페이지로 이동

Constructor Summary	
Constructors	
Constructor	Description
String()	Initializes a newly created String object so that it represents an empty character sequence.

❖ 클래스에 선언된 메소드 목록

- SUMMARY: NESTED | FIELD | CONSTR | METHOD 의 METHOD 클릭하여 메소드 목록으로 이동

Method Summary		
All Methods	Static Methods	Instance Methods
Concrete Methods	Deprecated Methods	
Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
InputStream	chars()	Returns a stream of int zero-extending the char values from this sequence.



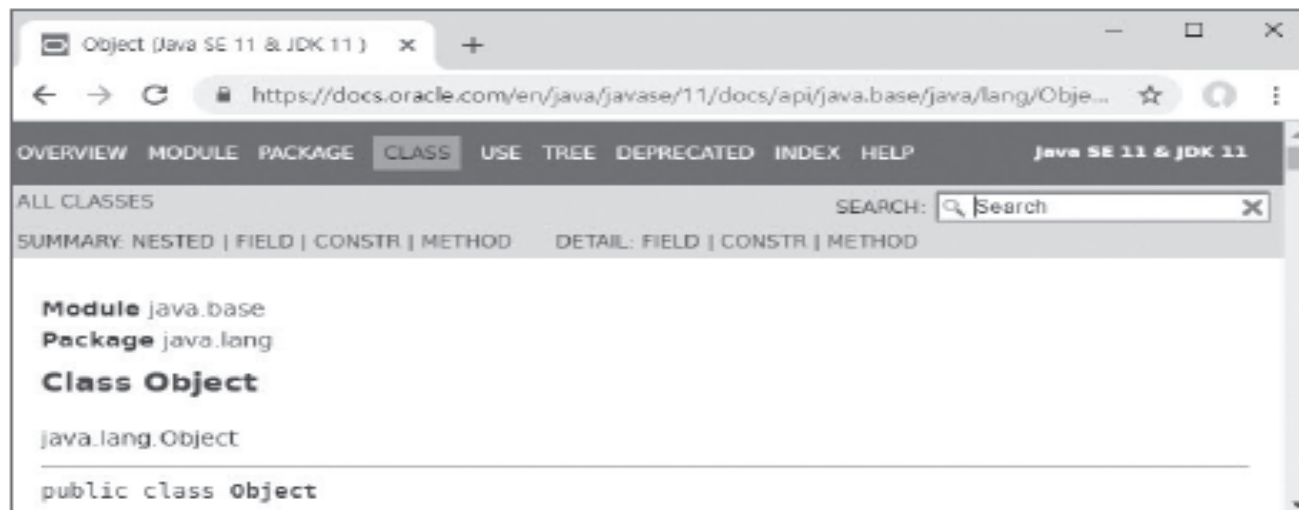
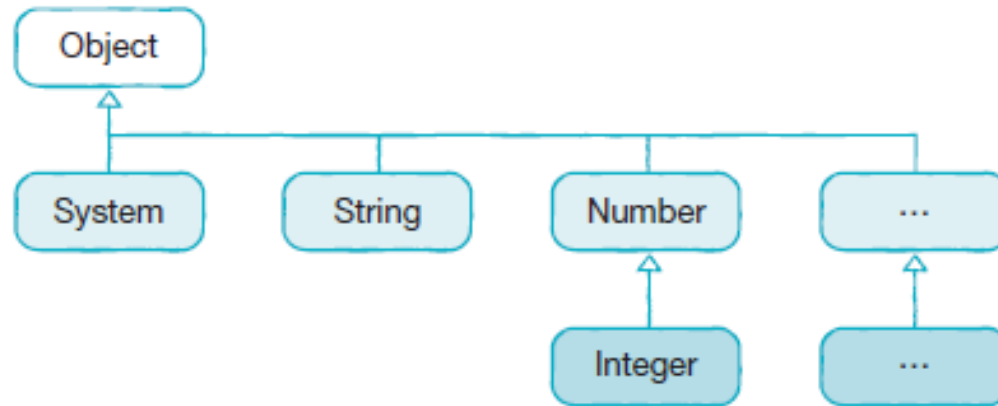
API 도큐먼트에서 클래스 페이지 읽는 방법

- [All Methods] : 전체 메소드 목록
- [Static Methods] : 정적 메소드 목록
- [Instance Methods] : 인스턴스 메소드 목록
- Modifier and Type : static 또는 protected 여부와 리턴 타입 표시
- Method와 Description의 메소드 이름 클릭하면 상세 설명 페이지로 이동



Object 클래스

❖ 모든 클래스는 Object 클래스의 자식이거나 자손 클래스



Object 클래스

❖ 객체 비교 (equals())

- equals() 메소드의 매개 타입은 Object로, 모든 객체가 매개값으로 대입될 수 있음
- Object 클래스의 equals() 메소드는 비교 연산자인 ==와 동일 결과 리턴

```
public boolean equals(Object obj) { ... }
```

```
Object obj1 = new Object();
```

```
Object obj2 = new Object();
```

```
boolean result = obj1.equals(obj2);
```

기준 객체

비교 객체

결과가 동일

```
boolean result = (obj1 == obj2)
```

- equals() 메소드는 두 객체가 논리적으로 동등하면 true를, 그렇지 않으면 false 리턴
- equals() 메소드는 매개값이 기준 객체와 동일 타입 객체인지 먼저 확인 필요



❖ 예시 - equals 메소드

```
01 package sec01.exam01;
02
03 public class Member {
04     public String id;
05
06     public Member(String id) {
07         this.id = id;
08     }
09
10     @Override
11     public boolean equals(Object obj) {
12         if(obj instanceof Member) { ← 매개값이 Member 타입인지 확인
13             Member member = (Member) obj;
14             if(id.equals(member.id)) { ← Member 타입으로 강제 타입 변환하고
15                 return true;           id 필드값이 동일한지 검사한 후,
16             }                         동일하다면 true를 리턴
17         }
18         return false; ← 매개값이 Member 타입이 아니거나
19     }                 id 필드값이 다른 경우 false를 리턴
20 }
```



Object 클래스

```
01 package sec01.exam01;
02
03 public class MemberExample {
04     public static void main(String[] args) {
05         Member obj1 = new Member("blue");
06         Member obj2 = new Member("blue");
07         Member obj3 = new Member("red");
08
09         if(obj1.equals(obj2)) { ← 매개값이 Member 타입이고
10             System.out.println("obj1과 obj2는 동등합니다.");           id 필드값도 동일하므로 true
11         } else {
12             System.out.println("obj1과 obj2는 동등하지 않습니다.");
13         }
14
15         if(obj1.equals(obj3)) { ← 매개값이 Member 타입이지만
16             System.out.println("obj1과 obj3은 동등합니다.");           id 필드값이 다르므로 false
17         } else {
18             System.out.println("obj1과 obj3은 동등하지 않습니다.");
19         }
20     }
21 }
```

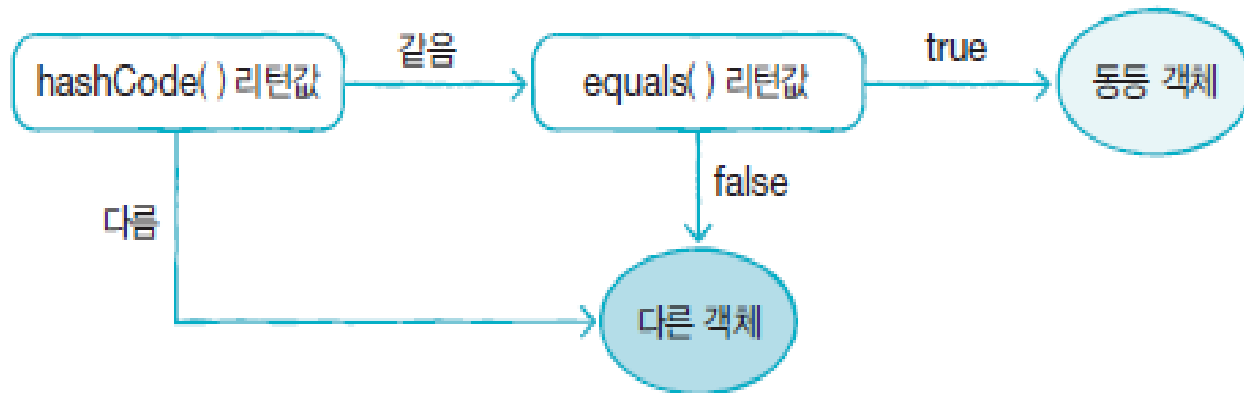
실행결과

obj1과 obj2는 동등합니다.
obj1과 obj3은 동등하지 않습니다.



❖ 객체 해시코드 (hashCode())

- 객체를 식별하는 하나의 정수값
- Object 클래스의 객체 해시코드 메소드는 객체 메모리 번지 이용하여 해시코드 만들어 리턴
객체마다 다른 값 가짐
- 두 객체가 동등한지 비교 필요



❖ 예시 - hashCode() 메소드를 재정의하지 않음

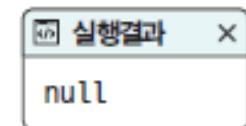
```
01 package sec01.exam02;
02
03 public class Key {
04     public int number;
05
06     public Key(int number) {
07         this.number = number;
08     }
09
10     @Override
11     public boolean equals(Object obj) {
12         if(obj instanceof Key) {
13             Key compareKey = (Key) obj;
14             if(this.number == compareKey.number) {
15                 return true;
16             }
17         }
18         return false;
19     }
20 }
```



❖ 예시 - 다른 키로 인식

- number 필드값 같아도 hashCode() 메소드에서 리턴하는 해시코드 다르면 다른 식별기로 인식

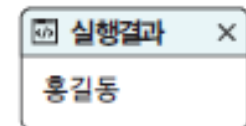
```
01 package sec01.exam02;
02
03 public class KeyExample {
04     public static void main(String[] args) {
05         //Key 객체를 식별기로 사용해서 String 값을 저장하는 HashMap 객체 생성
06         HashMap<Key, String> hashMap = new HashMap<Key, String>();
07
08         //식별키 new Key(1)로 "홍길동"을 저장함
09         hashMap.put(new Key(1), "홍길동");
10
11         //식별키 new Key(1)로 "홍길동"을 읽어옴
12         String value = hashMap.get(new Key(1));
13         System.out.println(value);
14     }
15 }
```



❖ 예시 - hashCode() 메소드 재정의 추가

- 재정의한 hashCode() 메소드 Key 클래스에 추가하여 의도한 대로 읽게 하기

```
01 package sec01.exam02;  
02  
03 public class Key {  
04     ...  
05     @Override  
06     public int hashCode() {  
07         return number;  
08     }  
09 }
```



❖ 예시 - hashCode() 메소드 재정의의 추가

```
01 package sec01.exam03;
02
03 public class Member {
04     public String id;
05
06     public Member(String id) {
07         this.id = id;
08     }
09
10     @Override
11     public boolean equals(Object obj) {
12         if(obj instanceof Member) {
13             Member member = (Member) obj;
14             if(id.equals(member.id)) {
15                 return true;
16             }
17         }
18         return false;
19     }
20
21     @Override
22     public int hashCode() {
23         return id.hashCode();
24     }
25 }
```

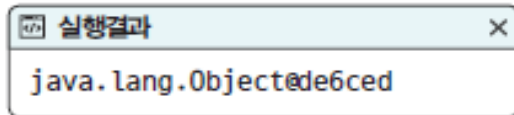
← id가 동일한 문자열인 경우
같은 해시 코드를 리턴

Object 클래스

❖ 객체 문자 정보 (toString())

- Object 클래스의 toString() 메소드는 객체의 문자 정보 리턴
'클래스이름@16진수해시코드'로 구성된 문자 정보

```
Object obj = new Object();  
System.out.println( obj.toString() );
```



- Object의 하위 클래스는 toString() 메소드 재정의하여 간결하고 유익한 정보 리턴



❖ 예시 - 객체의 문자 정보

```
01 package sec01.exam04;
02
03 public class ToStringExample {
04     public static void main(String[] args) {
05         Object obj1 = new Object();
06         Date obj2 = new Date();
07         System.out.println(obj1.toString());
08         System.out.println(obj2.toString());
09     }
10 }
```

실행결과

```
java.lang.Object@16f65612
Mon Apr 29 12:12:36 KST 2019
```



Object 클래스

■ 오버라이딩

```
01 package sec01.exam05;
02
03 public class SmartPhone {
04     private String company;
05     private String os;
06
07     public SmartPhone(String company, String os) {
08         this.company = company;
09         this.os = os;
10     }
11
12     @Override
13     public String toString() {
14         return company + ", " + os;
15     }
16 }
```

← toString() 재정의



Object 클래스

```
01 package sec01.exam05;
02
03 public class SmartPhoneExample {
04     public static void main(String[] args) {
05         SmartPhone myPhone = new SmartPhone("구글", "안드로이드");
06
07         String strObj = myPhone.toString(); ← 재정의된 toString() 호출
08         System.out.println(strObj);
09
10         System.out.println(myPhone); ← 재정의된 toString()을 호출하고 리턴값을 받아 출력
11     }
12 }
```

실행결과

구글, 안드로이드
구글, 안드로이드



❖ System 클래스의 모든 필드와 메소드는 정적 필드 및 메소드로 구성

❖ 프로그램 종료 (exit())

- exit() 메소드 호출하여 JVM을 강제 종료
- exit() 메소드가 지정하는 int 매개값을 종료 상태값이라 함

```
01 package sec01.exam06;
02
03 public class ExitExample {
04     public static void main(String[] args) {
05         for(int i=0; i<10; i++) {
06             if(i == 5) {
07                 System.exit(0);
08                 //break;
09             }
10         }
11         System.out.println("마무리 코드");
12     }
13 }
```



❖ 현재 시각 읽기 (currentTimeMillis(), nanoTime())

- System 클래스의 currentMillis() 및 nanoTime() 메소드로 각기 1/1000초 및 $10/10^9$ 단위 long 값 리턴

```
long time = System.currentTimeMillis();
```

```
long time = System.nanoTime();
```



❖ 예시 - 프로그램 소요 시간 구하기

```
01 package sec01.exam07;
02
03 public class SystemTimeExample {
04     public static void main(String[] args) {
05         long time1 = System.nanoTime(); ← 시작 시간 읽기
06
07         int sum = 0;
08         for(int i=1; i<=1000000; i++) {
09             sum += i;
10         }
11
12         long time2 = System.nanoTime(); ← 끝 시간 읽기
13
14         System.out.println("1~1000000까지의 합: " + sum);
15         System.out.println("계산에 " + (time2-time1) + " 나노초가 소요되었습니다.");
16     }
17 }
```

실행결과

1~1000000까지의 합: 1784293664
계산에 2407700 나노초가 소요되었습니다.

Class 클래스

❖ 자바는 클래스와 인터페이스의 메타 데이터를 Class 클래스로 관리

❖ Class 객체 얻기 (getClass(), forName())

클래스로부터 얻는 방법

- ① `Class clazz = 클래스이름.class`
- ② `Class clazz = Class.forName("패키지...클래스이름")`

객체로부터 얻는 방법

- ③ `Class clazz = 참조변수.getClass();`

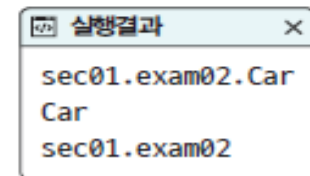
■ ex) String 클래스

- ① `Class clazz = String.class;`
- ② `Class clazz = Class.forName("java.lang.String");`
`String str = "감자바";`
- ③ `Class clazz = str.getClass();`



❖ 예시 - Class 객체 정보 얻기

```
01 package sec01.exam08;
02
03 public class ClassExample {
04     public static void main(String[] args) throws Exception {
05         //첫 번째 방법
06         Class clazz = Car.class;
07
08         //두 번째 방법
09         //Class clazz = Class.forName("sec01.exam08.Car");
10
11         //세 번째 방법
12         //Car car = new Car();
13         //Class clazz = car.getClass();
14
15         System.out.println(clazz.getName());
16         System.out.println(clazz.getSimpleName());
17         System.out.println(clazz.getPackage().getName());
18     }
19 }
```



실행결과

```
sec01.exam02.Car
Car
sec01.exam02
```



❖ 클래스 경로 활용하여 리소스 절대 경로 얻기

- Class 객체는 해당 클래스의 파일 경로 정보 가지고 있어 이 경로 활용해 다른 리소스 파일의 경로 얻을 수 있음

```
C:\SelfJavaStudy\chap11\bin\sec01
    | - exam09
        | - Car.class
        | - photo1.jpg
        | - images
            | - photo2.jpg
```

```
String photo1Path = clazz.getResource("photo1.jpg").getPath();
String photo2Path = clazz.getResource("images/photo2.jpg").getPath();
```



❖ 예시 - 리소스 절대 경로 얻기

```
01 package sec01.exam09;  
02  
03 public class ResourcePathExample {  
04     public static void main(String[] args) {  
05         Class clazz = Car.class;  
06  
07         String photo1Path = clazz.getResource("photo1.jpg").getPath();  
08         String photo2Path = clazz.getResource("images/photo2.jpg").getPath();  
09  
10         System.out.println(photo1Path);  
11         System.out.println(photo2Path);  
12     }  
13 }
```

실행결과

C:/SelfStudyJava/chap11/bin/sec01/exam09/photo1.jpg

C:/SelfStudyJava/chap11/bin/sec01/exam09/images/photo2.jpg

String 클래스

❖ String 생성자

- 직접 String 객체를 생성

```
//배열 전체를 String 객체로 생성
String str = new String(byte[] bytes);

//지정한 문자셋으로 디코딩
String str = new String(byte[] bytes, String charsetName);

//배열의 offset 인덱스 위치부터 length만큼 String 객체로 생성
String str = new String(byte[] bytes, int offset, int length);

//지정한 문자셋으로 디코딩
String str = new String(byte[] bytes, int offset, int length, String charsetName)
```



String 클래스

❖ 예시 - 바이트 배열을 문자열로 변환

```
01 package sec01.exam10;
02
03 public class ByteToStringExample {
04     public static void main(String[] args) {
05         byte[] bytes = { 72, 101, 108, 108, 111, 32, 74, 97, 118, 97 };
06
07         String str1 = new String(bytes);
08         System.out.println(str1);
09
10         String str2 = new String(bytes, 6, 4);
11         System.out.println(str2);
12     }
13 }
```

74의 인덱스 위치



4개

실행결과

```
Hello Java
Java
```

입력 내용:

바이트 배열 내용 :

H	e	l	l	o	\r	\n
72	101	108	108	111	13	10

실제 입력된 내용

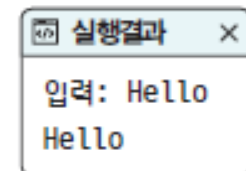
[Enter] 키



String 클래스

```
01 package sec01.exam11;
02
03 public class KeyboardToStringExample {
04     public static void main(String[] args) throws IOException {
05         byte[] bytes = new byte[100]; ← 읽은 바이트를 저장하기 위한 배열 생성
06
07         System.out.print("입력: ");
08         int readByteNo = System.in.read(bytes); ← 배열에 읽은 바이트를 저장하고
09                                                    읽은 바이트 수를 리턴
10         String str = new String(bytes, 0, readByteNo-2); ←
11         System.out.println(str);
12     }
13 }
```

배열을 문자열로 변환



❖ String 메소드

리턴 타입	메소드 이름(매개 변수)	설명
char	charAt(int index)	특정 위치의 문자를 리턴합니다.
boolean	equals(Object anObject)	두 문자열을 비교합니다.
byte[]	getBytes()	byte[]로 리턴합니다.
byte[]	getBytes(Charset charset)	주어진 문자셋으로 인코딩한 byte[]로 리턴합니다.
int	indexOf(String str)	문자열 내에서 주어진 문자열의 위치를 리턴합니다.
int	length()	총 문자의 수를 리턴합니다.
String	replace(CharSequence target, CharSequence replacement)	target 부분을 replacement로 대체한 새로운 문자열을 리턴합니다.
String	substring(int beginIndex)	beginIndex 위치에서 끝까지 잘라낸 새로운 문자열을 리턴합니다.
String	substring(int beginIndex, int endIndex)	beginIndex 위치에서 endIndex 전까지 잘라낸 새로운 문자열을 리턴합니다.
String	toLowerCase()	알파벳 소문자로 변환한 새로운 문자열을 리턴합니다.
String	toUpperCase()	알파벳 대문자로 변환한 새로운 문자열을 리턴합니다.
String	trim()	앞뒤 공백을 제거한 새로운 문자열을 리턴합니다.
String	valueOf(int i) valueOf(double d)	기본 타입 값을 문자열로 리턴합니다.



String 클래스

- 문자 추출 (charAt())

매개값으로 주어진 인덱스의 문자를 리턴

```
String subject = "자바 프로그래밍";  
char charValue = subject.charAt(3);
```

자	바		프	로	그	래	밍
0	1	2	3	4	5	6	7

위 경우 '프' 리턴



String 클래스

- 예시 - 주민등록번호에서 남자와 여자를 구별

```
01 package sec01.exam12;
02
03 public class StringCharAtExample {
04     public static void main(String[] args) {
05         String ssn = "010624-1230123";
06         char sex = ssn.charAt(7);
07         switch (sex) {
08             case '1':
09             case '3':
10                 System.out.println("남자 입니다.");
11                 break;
12             case '2':
13             case '4':
14                 System.out.println("여자 입니다.");
15                 break;
16         }
17     }
18 }
```

실행결과 ×
남자 입니다.

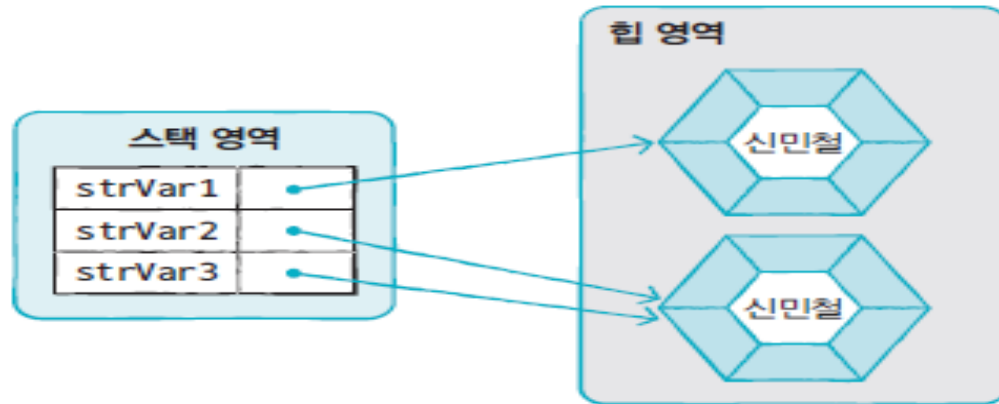


String 클래스

■ 문자열 비교 (equals())

== 연산자 사용할 경우 원하지 않는 결과 나올 수 있음

```
String strVar1 = new String("신민철");  
String strVar2 = "신민철";  
String strVar3 = "신민철";
```



```
strVar1 == strVar2    //false  
strVar2 == strVar3    //true
```

```
strVar1.equals(strVar2) //true  
strVar2.equals(strVar3) //true
```



String 클래스

■ 예시 - 문자열 비교

```
01 package sec01.exam13;
02
03 public class StringEqualsExample {
04     public static void main(String[] args) {
05         String strVar1 = new String("신민철");
06         String strVar2 = "신민철";
07
08         if(strVar1 == strVar2) {
09             System.out.println("같은 String 객체를 참조");
10         } else {
11             System.out.println("다른 String 객체를 참조");
12         }
13
14         if(strVar1.equals(strVar2)) {
15             System.out.println("같은 문자열을 가짐");
16         } else {
17             System.out.println("다른 문자열을 가짐");
18         }
19     }
20 }
```

실행결과

다른 String 객체를 참조
같은 문자열을 가짐



String 클래스

- 바이트 배열로 변환 (getBytes())

```
byte[] bytes = "문자열".getBytes();  
byte[] bytes = "문자열".getBytes(Charset charset);
```

getBytes()의 메소드는 시스템의 기본 문자셋으로 인코딩된 바이트 배열 리턴
getBytes(Charset charset) 메소드는 특정 문자셋으로 인코딩된 바이트 배열 리턴

```
try {  
    byte[] bytes1 = "문자열".getBytes("EUC-KR");  
    byte[] bytes2 = "문자열".getBytes("UTF-8");  
} catch (UnsupportedEncodingException e) {  
}
```

- 바이트 배열을 다시 문자열로 디코딩할 때에는 어떤 문자셋으로 인코딩되었는가에 따라 디코딩 방법 다른
시스템 기본 문자셋과 다른 문자셋으로 인코딩되었을 경우

```
String str = new String(byte[] bytes, String charsetName);
```



String 클래스

■ 예시 - 바이트 배열로 변환

```
01 package sec01.exam14;
02
03 public class StringGetBytesExample {
04     public static void main(String[] args) {
05         String str = "안녕하세요";
06
07         byte[] bytes1 = str.getBytes();
08         System.out.println("bytes1.length: " + bytes1.length);
09         String str1 = new String(bytes1);
10         System.out.println("bytes1->String: " + str1);
11
12     try {
13
```

← 기본 문자셋으로
인코딩과 디코딩



String 클래스

```
14 byte[] bytes2 = str.getBytes("EUC-KR");
15 System.out.println("bytes2.length: " + bytes2.length);
16 String str2 = new String(bytes2, "EUC-KR");
17 System.out.println("bytes2->String: " + str2);
18
19 byte[] bytes3 = str.getBytes("UTF-8");
20 System.out.println("bytes3.length: " + bytes3.length);
21 String str3 = new String(bytes3, "UTF-8");
22 System.out.println("bytes3->String: " + str3);
23
24 } catch (UnsupportedEncodingException e) {
25     e.printStackTrace();
26 }
27 }
28 }
```

← EUC-KR을
이용해서
인코딩 및 디코딩

← UTF-8을
이용해서
인코딩 및 디코딩

실행결과

```
bytes1.length: 10
bytes1->String: 안녕하세요
bytes2.length: 10
bytes2->String: 안녕하세요
bytes3.length: 15
bytes3->String: 안녕하세요
```

String 클래스

■ 문자열 찾기 (indexOf())

매개값으로 주어진 문자열이 시작되는 인덱스 리턴
주어진 문자열 포함되어 있지 않으면 -1 리턴

```
String subject = "자바 프로그래밍";  
int index = subject.indexOf("프로그래밍");
```

자	바		프	로	그	래	밍
0	1	2	3	4	5	6	7

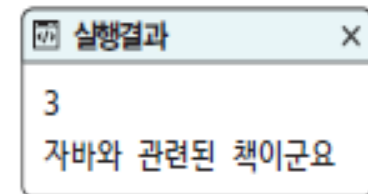
위 경우 index 변수에 3이 저장

```
if( 문자열.indexOf("찾는문자열") != -1 ) {  
    //포함되어 있는 경우  
} else {  
    //포함되어 있지 않은 경우  
}
```

String 클래스

■ 예시 - 문자열 포함 여부 조사

```
01 package sec01.exam15;
02
03 public class StringIndexOfExample {
04     public static void main(String[] args) {
05         String subject = "자바 프로그래밍";
06
07         int location = subject.indexOf("프로그래밍");
08         System.out.println(location);
09
10         if(subject.indexOf("자바") != -1) {
11             System.out.println("자바와 관련된 책이군요");
12         } else {
13             System.out.println("자바와 관련없는 책이군요");
14         }
15     }
16 }
```

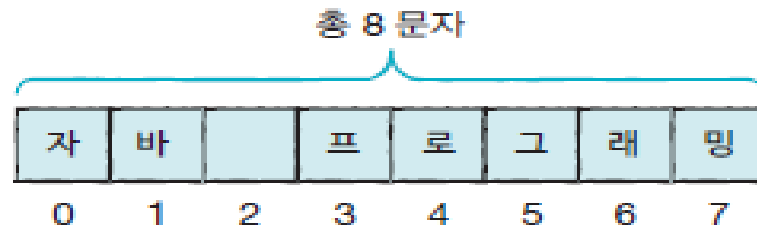


String 클래스

- 문자열 길이 (length())

문자열의 길이를 리턴

```
String subject = "자바 프로그래밍";  
int length = subject.length();
```



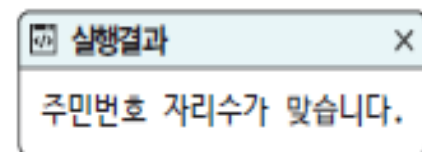
위 경우 8 저장



String 클래스

■ 예시 - 문자열의 문자 수 읽기

```
01 package sec01.exam16;
02
03 public class StringLengthExample {
04     public static void main(String[] args) {
05         String ssn = "7306241230123";
06         int length = ssn.length();
07         if(length == 13) {
08             System.out.println("주민번호 자리수가 맞습니다.");
09         } else {
10             System.out.println("주민번호 자리수가 틀립니다.");
11         }
12     }
13 }
```

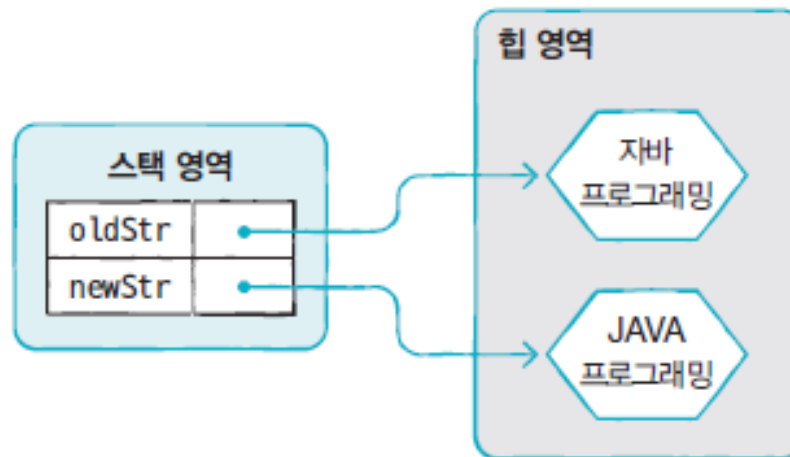


String 클래스

■ 문자열 대치 (replace())

첫 번째 매개값인 문자열을 찾아 두 번째 매개값인 문자열로 대치한 새로운 문자열 생성 및 리턴

```
String oldStr = "자바 프로그래밍";  
String newStr = oldStr.replace("자바", "JAVA");
```



위 경우 newStr 변수는 "JAVA 프로그래밍" 문자열 참조



String 클래스

■ 예시 - 문자열 대치하기

```
01 package sec01.exam17;
02
03 public class StringReplaceExample {
04     public static void main(String[] args) {
05         String oldStr = "자바는 객체 지향 언어입니다. 자바는 풍부한 API를 지원합니다.";
06         String newStr = oldStr.replace("자바", "JAVA");
07         System.out.println(oldStr);
08         System.out.println(newStr);
09     }
10 }
```

실행결과

자바는 객체 지향 언어입니다. 자바는 풍부한 API를 지원합니다.
JAVA는 객체 지향 언어입니다. JAVA는 풍부한 API를 지원합니다.



String 클래스

■ 문자열 잘라내기 (substring())

주어진 인덱스에서 문자열을 추출

substring(int beginIndex, int endIndex) 는 주어진 시작과 끝 인덱스 사이의 문자열 추출

substring(int beginIndex)는 주어진 인덱스부터 끝까지 문자열 추출

```
String ssn = "880815-1234567";  
String firstNum = ssn.substring(0, 6);  
String secondNum = ssn.substring(7);
```

8	8	0	8	1	5	-	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7	8	9	10	11	12	13

위 경우 firstNum 변수값 880815, secondNum 변수값 1234567



String 클래스

■ 예시 - 문자열 추출하기

```
01 package sec01.exam18;
02
03 public class StringSubstringExample {
04     public static void main(String[] args) {
05         String ssn = "880815-1234567 ";
06
07         String firstNum = ssn.substring(0, 6);
08         System.out.println(firstNum);
09
10         String secondNum = ssn.substring(7);
11         System.out.println(secondNum);
12     }
13 }
```

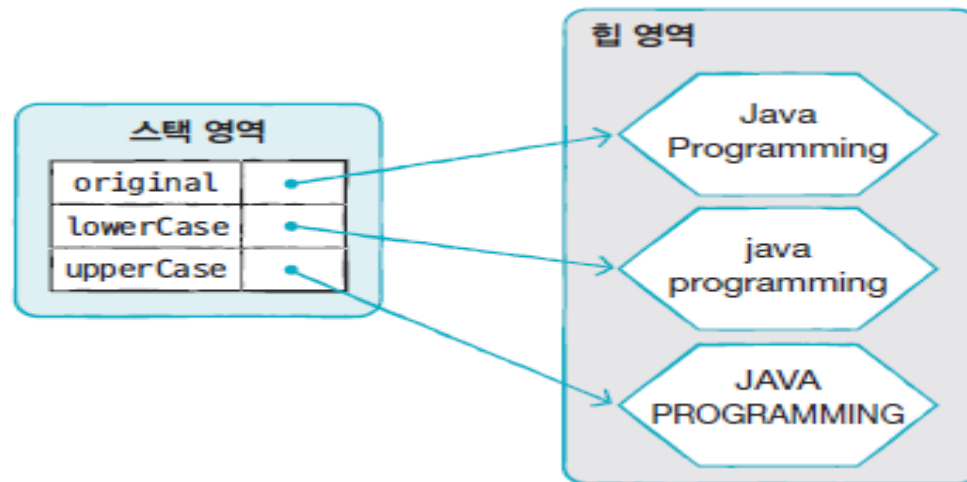
실행결과	
880815	
1234567	



String 클래스

- 알파벳 소, 대문자 변경 (toLowerCase(), toUpperCase())

```
String original = "Java Programming";  
String lowerCase = original.toLowerCase();  
String upperCase = original.toUpperCase();
```



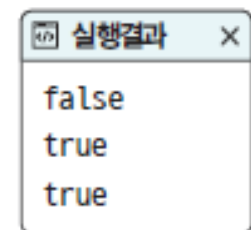
lowerCase 변수는 새로 생성된 "java programming" 문자열 참조
upperCase 변수는 새로 생성된 "JAVA PROGRAMMING" 문자열 참조
원래 original 변수의 "Java Programming" 문자열이 변경된 것 아님



String 클래스

- 예시 - 전부 소문자 또는 대문자로 변경

```
01 package sec01.exam19;
02
03 public class StringToLowerUpperCaseExample {
04     public static void main(String[] args) {
05         String str1 = "Java Programming";
06         String str2 = "JAVA Programming";
07
08         System.out.println(str1.equals(str2));
09
10         String lowerStr1 = str1.toLowerCase();
11         String lowerStr2 = str2.toLowerCase();
12         System.out.println(lowerStr1.equals(lowerStr2));
13
14         System.out.println(str1.equalsIgnoreCase(str2));
15     }
16 }
```

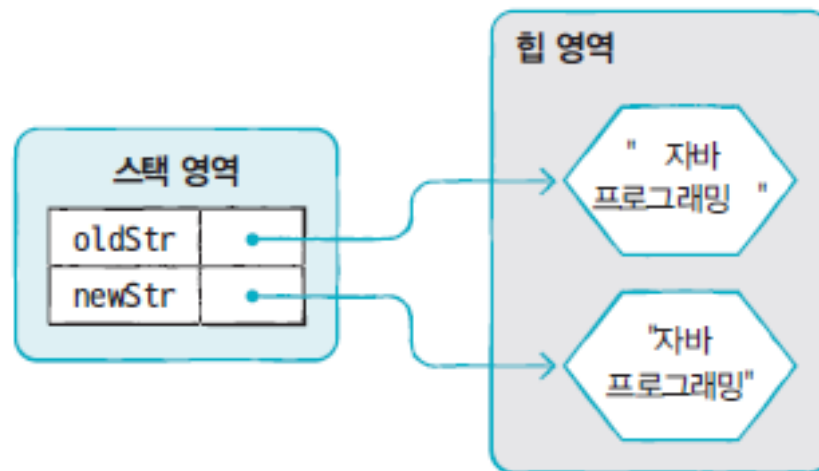


String 클래스

- 문자열 앞뒤 공백 잘라내기 (trim())

문자열의 앞뒤 공백 제거한 새로운 문자열 생성 및 리턴

```
String oldStr = " 자바 프로그래밍 ";  
String newStr = oldStr.trim();
```



위 경우 newStr 변수는 앞뒤 공백 제거되고 새로 생성된 "자바 프로그래밍" 문자열 참조



String 클래스

■ 예시 - 문자열 앞뒤 공백 제거

```
01 package sec01.exam20;  
02  
03 public class StringTrimExample {  
04     public static void main(String[] args) {  
05         String tel1 = " 02";  
06         String tel2 = "123  ";  
07         String tel3 = " 1234  ";  
08  
09         String tel = tel1.trim() + tel2.trim() + tel3.trim();  
10         System.out.println(tel);  
11     }  
12 }
```

실행결과

021231234



String 클래스

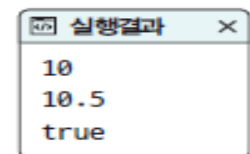
- 문자열 변환 (valueOf())

기본 타입의 값을 문자열로 변환

```
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(double d)
static String valueOf(float f)
```

- 예시 - 기본 타입 값을 문자열로 변환

```
01 package sec01.exam21;
02
03 public class StringValueOfExample {
04     public static void main(String[] args) {
05         String str1 = String.valueOf(10);
06         String str2 = String.valueOf(10.5);
07         String str3 = String.valueOf(true);
08
09         System.out.println(str1);
10         System.out.println(str2);
11         System.out.println(str3);
12     }
13 }
```



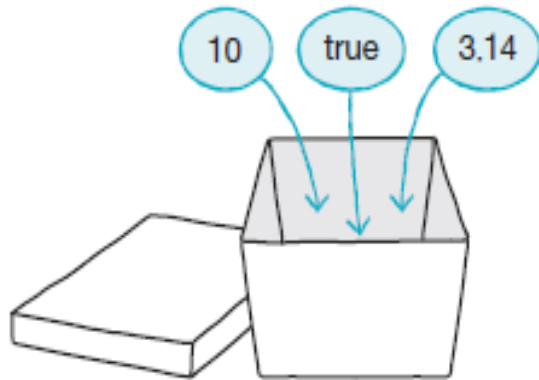
실행결과

```
10
10.5
true
```

Wrapper(포장) 클래스

❖ 포장 객체

- 기본 타입의 값을 내부에 두고 포장
- 포장하고 있는 기본 타입 값은 외부에서 변경할 수 없음
- byte, char, short, int, long, float, double, boolean 기본 타입 값 갖는 객체



기본 타입	포장 클래스
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean



Wrapper(포장) 클래스

❖ Boxing과 Unboxing

- 박싱 : 기본 타입의 값을 포장 객체로 만드는 과정
- 언박싱 : 포장 객체에서 기본 타입의 값을 얻어내는 과정

기본 타입의 값을 줄 경우	문자열을 줄 경우
Byte obj = new Byte(10);	Byte obj = new Byte("10");
Character obj = new Character('가');	없음
Short obj = new Short(100);	Short obj = new Short("100");
Integer obj = new Integer(1000);	Integer obj = new Integer("1000");
Long obj = new Long(10000);	Long obj = new Long("10000");
Float obj = new Float(2.5F);	Float obj = new Float("2.5F");
Double obj = new Double(3.5);	Double obj = new Double("3.5");
Boolean obj = new Boolean(true);	Boolean obj = new Boolean("true");



Wrapper(포장) 클래스

- 생성자 이용하지 않고 각 포장 클래스마다 가진 정적 valueOf() 메소드 활용

```
Integer obj = Integer.valueOf(1000);  
Integer obj = Integer.valueOf("1000");
```

- '기본 타입 이름+Value()' 메소드 호출하여 언박싱

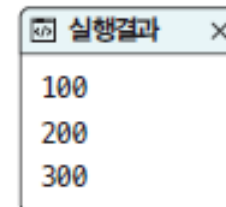
기본 타입의 값을 이용	
byte	num = obj.byteValue();
char	ch = obj.charValue();
short	num = obj.shortValue();
int	num = obj.intValue();
long	num = obj.longValue();
float	num = obj.floatValue();
double	num = obj.doubleValue();
boolean	bool = obj.booleanValue();



Wrapper(포장) 클래스

■ 예시 - 기본 타입의 값 박싱하고 언박싱하기

```
01 package sec01.exam22;
02
03 public class BoxingUnBoxingExample {
04     public static void main(String[] args) {
05         //박싱
06         Integer obj1 = new Integer(100);
07         Integer obj2 = new Integer("200");
08         Integer obj3 = Integer.valueOf("300");
09
10         //언박싱
11         int value1 = obj1.intValue();
12         int value2 = obj2.intValue();
13         int value3 = obj3.intValue();
14
15         System.out.println(value1);
16         System.out.println(value2);
17         System.out.println(value3);
18     }
19 }
```



Wrapper(포장) 클래스

❖ 자동 박싱과 언박싱

- 포장 클래스 타입에 기본값이 대입될 경우 자동 박싱 발생

```
Integer obj = 100; //자동 박싱
```

- 기본 타입에 포장 객체가 대입되는 경우 및 연산에서 자동 언박싱 발생

```
Integer obj = new Integer(200);  
int value1 = obj; //자동 언박싱  
int value2 = obj + 100; //자동 언박싱
```



Wrapper(포장) 클래스

■ 예시 - 자동 박싱과 언박싱

```
01 package sec01.exam23;
02
03 public class AutoBoxingUnBoxingExample {
04     public static void main(String[] args) {
05         //자동 박싱
06         Integer obj = 100;
07         System.out.println("value: " + obj.intValue());
08
09         //대입 시 자동 언박싱
10         int value = obj;
11         System.out.println("value: " + value);
12
13         //연산 시 자동 언박싱
14         int result = obj + 100;
15         System.out.println("result: " + result);
16     }
17 }
```

실행결과

value: 100
value: 100
result: 200



Wrapper(포장) 클래스

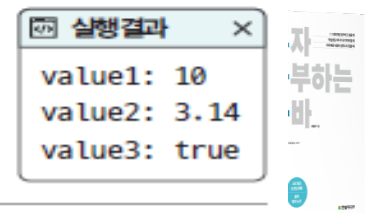
❖ 문자열을 기본 타입 값으로 변환

- 포장 클래스로 문자열을 기본 타입 값으로 변환
- 'parse+기본 타입 이름' 정적 메소드
- 예시 - 문자열을 기본 타입 값으로 변환

기본 타입의 값을 이용

byte	num = Byte.parseByte("10");
short	num = Short.parseShort("100");
int	num = Integer.parseInt("1000");
long	num = Long.parseLong("10000");
float	num = Float.parseFloat("2.5F");
double	num = Double.parseDouble("3.5");
boolean	bool = Boolean.parseBoolean("true");

```
01 package sec01.exam24;
02
03 public class StringToPrimitiveValueExample {
04     public static void main(String[] args) {
05         int value1 = Integer.parseInt("10");
06         double value2 = Double.parseDouble("3.14");
07         boolean value3 = Boolean.parseBoolean("true");
08
09         System.out.println("value1: " + value1);
10         System.out.println("value2: " + value2);
11         System.out.println("value3: " + value3);
12     }
13 }
```



Wrapper(포장) 클래스

❖ 포장 값 비교

- 포장 객체는 내부 값 비교하기 위해 == 및 != 연산자 사용하지 않는 것이 좋음
아래 연산 결과 false

```
Integer obj1 = 300;  
Integer obj2 = 300;  
System.out.println(obj1 == obj2);
```

- 박싱된 값이 아래 표에 나와 있는 범위의 값이 아닌 경우 언박싱한 값 얻어 비교해야 함

타입	값의 범위
boolean	true, false
char	\u0000 ~ \u007f
byte, short, int	-128 ~ 127



Wrapper(포장) 클래스

■ 예시 - 포장 객체 비교

```
01 package sec01.exam25;
02
03 public class ValueCompareExample {
04     public static void main(String[] args) {
05         System.out.println("[ -128~127 초과값일 경우 ]");
06         Integer obj1 = 300;
07         Integer obj2 = 300;
08         System.out.println("==결과: " + (obj1 == obj2));
09         System.out.println("언박싱후 ==결과: " + (obj1.intValue()
10         == obj2.intValue()));
11         System.out.println("equals() 결과: " + obj1.equals(obj2));
12         System.out.println();
13         System.out.println("[ -128~127 범위값일 경우 ]");
14         Integer obj3 = 10;
15         Integer obj4 = 10;
16         System.out.println("==결과: " + (obj3 == obj4));
17         System.out.println("언박싱후 ==결과: " + (obj3.intValue() == obj4.intValue()));
18         System.out.println("equals() 결과: " + obj3.equals(obj4));
19     }
20 }
```

실행결과

[-128~127 초과값일 경우]
==결과: false
언박싱후 ==결과: true
equals() 결과: true

[-128~127 범위값일 경우]
==결과: true
언박싱후 ==결과: true
equals() 결과: true

❖ Math 클래스

- 수학 계산에 사용

메소드	설명	예제 코드	리턴값
int abs(int a) double abs(double a)	절대값	int v1 = Math.abs(-5); double v2 = Math.abs(-3.14);	v1 = 5 v2 = 3.14
double ceil(double a)	올림값	double v3 = Math.ceil(5.3); double v4 = Math.ceil(-5.3);	v3 = 6.0 v4 = -5.0
double floor(double a)	버림값	double v5 = Math.floor(5.3); double v6 = Math.floor(-5.3);	v5 = 5.0 v6 = -6.0
int max(int a, int b) double max(double a, double b)	최대값	int v7 = Math.max(5, 9); double v8 = Math.max(5.3, 2.5);	v7 = 9 v8 = 5.3
int min(int a, int b) double min(double a, double b)	최소값	int v9 = Math.min(5, 9); double v10 = Math.min(5.3, 2.5);	v9 = 5 v10 = 2.5
double random()	랜덤값	double v11 = Math.random();	0.0 ≤ v11 < 1.0
double rint(double a)	가까운 정수의 실수값	double v12 = Math.rint(5.3); double v13 = Math.rint(5.7);	v12 = 5.0 v13 = 6.0
long round(double a)	반올림값	long v14 = Math.round(5.3); long v15 = Math.round(5.7);	v14 = 5 v15 = 6



Math 클래스

■ 예시 - Math의 수학 메소드

```
01 package sec01.exam26;
02
03 public class MathExample {
04     public static void main(String[] args) {
05         int v1 = Math.abs(-5);
06         double v2 = Math.abs(-3.14);
07         System.out.println("v1=" + v1);
08         System.out.println("v2=" + v2);
09
10         double v3 = Math.ceil(5.3);
11         double v4 = Math.ceil(-5.3);
12         System.out.println("v3=" + v3);
13         System.out.println("v4=" + v4);
14
15         double v5 = Math.floor(5.3);
16         double v6 = Math.floor(-5.3);
17         System.out.println("v5=" + v5);
18         System.out.println("v6=" + v6);
19
20         int v7 = Math.max(5, 9);
21         double v8 = Math.max(5.3, 2.5);
22         System.out.println("v7=" + v7);
23         System.out.println("v8=" + v8);
24     }
```



Math 클래스

- math.random() 메소드는 0.0 이상 1.0 미만 범위에 속하는 하나의 double 타입 값 리턴

```
0.0 <= Math.random() < 1.0
```

예시 - 1부터 10까지 정수 난수 얻기

```
0.0 * 10 <= Math.random() * 10 < 1.0 * 10  
  ↑                ↑  
(0.0)            (10.0)
```

```
(int) (0.0 * 10) <= (int) (Math.random() * 10) < (int) (1.0 * 10)  
  ↑                ↑                ↑  
(0)            (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)  (10)
```

```
(int) (0.0 * 10) + 1 <= (int) (Math.random() * 10) + 1 < (int) (1.0 * 10) + 1  
  ↑                ↑                ↑  
(1)            (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  (11)
```

```
int num = (int) (Math.random() * 10) + 1;
```



Math 클래스

start <= ... < (start+n) 범위의 하나의 정수 얻기

[예1] 주사위 번호 뽑기(1, 2, 3, 4, 5, 6)

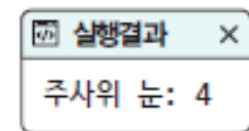
```
int num = (int) (Math.random() * 6) + 1;
```

[예2] 로또 번호 뽑기(1, 2, 3, ... 43, 44, 45)

```
int num = (int) (Math.random() * 45) + 1;
```

■ 예시 - 임의의 주사위 눈 얻기

```
01 package sec01.exam27;  
02  
03 public class MathRandomExample {  
04     public static void main(String[] args) {  
05         int num = (int) (Math.random()*6) + 1;  
06         System.out.println("주사위 눈: " + num);  
07     }  
08 }
```



Math 클래스

```
25     int v9 = Math.min(5, 9);
26     double v10 = Math.min(5.3, 2.5);
27     System.out.println("v9=" + v9);
28     System.out.println("v10=" + v10);
29
30     double v11 = Math.random();
31     System.out.println("v11=" + v11);
32
33     double v12 = Math rint(5.3);
34     double v13 = Math.rint(5.7);
35     System.out.println("v12=" + v12);
36     System.out.println("v13=" + v13);
37
38     long v14 = Math.round(5.3);
39     long v15 = Math.round(5.7);
40     System.out.println("v14=" + v14);
41     System.out.println("v15=" + v15);
42
43     double value = 12.3456;
44     double temp1 = value * 100;
45     long temp2 = Math.round(temp1);
46     double v16 = temp2 / 100.0;
47     System.out.println("v16=" + v16);
48 }
49 }
```

실행결과

```
v1=5
v2=3.14
v3=6.0
v4=-5.0
v5=5.0
v6=-6.0
v7=9
v8=5.3
v9=5
v10=2.5
v11=0.8389347076415933
v12=5.0
v13=6.0
v14=5
v15=6
v16=12.35
```

키워드로 끝내는 핵심 포인트

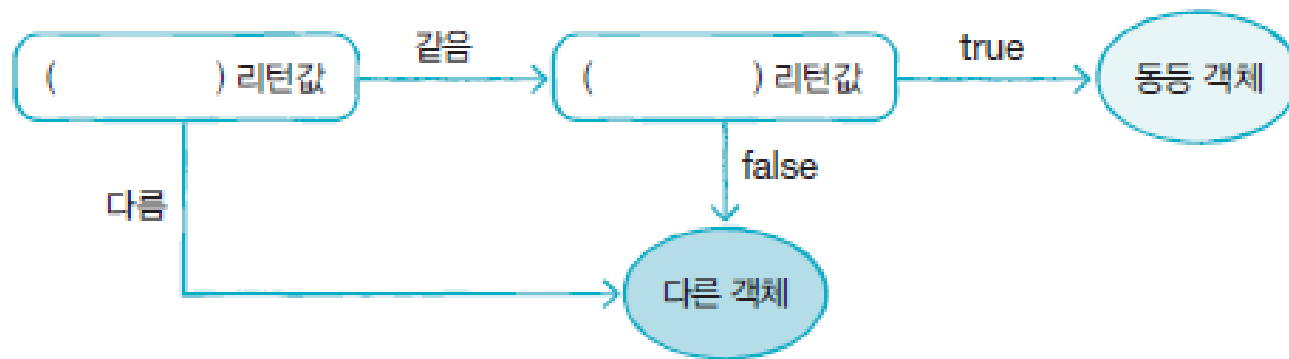
- **Object 클래스** : 자바의 최상위 부모 클래스, Object 클래스의 메소드는 모든 자바 객체에서 사용 가능
- **System 클래스** : 운영체제의 일부 기능 이용할 수 있음. 정적 필드와 정적 메소드로 구성
- **Class 클래스** : 클래스와 인터페이스의 메타 데이터가 Class 클래스로 관리됨.
- **String 클래스** : String 클래스의 다양한 생성자 이용하여 직접 String 객체를 생성 가능.
- **Wrapper 클래스** : 기본 타입의 값 갖는 객체를 포장 객체라 함. 기본 타입의 값을 포장 객체로 만드는 것을 박싱, 반대의 과정을 언박싱이라 함.
- **Math 클래스** : 수학 계산에 사용할 수 있는 메소드 제공하며, Math 클래스가 제공하는 메소드는 모두 정적 메소드이므로 Math 클래스로 바로 사용 가능.



❖ Object 클래스에 대한 설명 중 틀린 것은 무엇입니까?

- 모든 자바 클래스의 최상위 부모 클래스이다
- Object의 equals() 메소드는 == 연산자와 동일하게 번지를 비교한다
- 동등 비교를 위해 equals()와 hashCode() 메소드를 재정의하는 것이 좋다
- Object의 toString() 메소드는 객체의 필드값을 문자열로 리턴한다

❖ 여러분이 작성하는 클래스를 동등 비교하는 컬렉션 객체인 HashSet, HashMap, Hashtable을 사용하려고 합니다. Object의 equals()와 hashCode() 메소드를 재정의했다고 가정할 경우, 메소드 호출 순서를 생각하고 다음 괄호 안을 채워보세요



확인문제

- ❖ Member 클래스를 작성하되, Object의 toString() 메소드를 재정의해서 Member Example 클래스의 실행결과처럼 나오도록 작성해보세요

소스 코드 Member.java

```
01 package sec01.verify.exam04;
02
03 public class Member {
04     private String id;
05     private String name;
06
07     public Member(String id, String name) {
08         this.id = id;
09         this.name = name;
10     }
11
12     //여기서 코드를 작성하세요.
13 }
```



확인문제

소스 코드 MemberExample.java

```
01 package sec01.verify.exam04;
02
03 public class MemberExample {
04     public static void main(String[] args) {
05         Member member = new Member("blue", "이파란");
06         System.out.println(member);
07     }
08 }
```

실행결과

blue: 이파란



확인문제

- ❖ 문자열 "모든 프로그램은 자바 언어로 개발될 수 있다."에서 "자바" 문자열이 포함되어 있는지 확인하고, "자바"를 Java로 대치한 새로운 문자열을 만들어보세요.

소스 코드 FindAndReplaceExample.java

```
01 package sec01.verify.exam07;
02
01 public class FindAndReplaceExample {
02     public static void main(String[] args) {
03         String str = "모든 프로그램은 자바 언어로 개발될 수 있다.";
04         int index = ① ;
05         if(index == -1) {
06             System.out.println("자바 문자열이 포함되어 있지 않습니다.");
07         } else {
08             System.out.println("자바 문자열이 포함되어 있습니다.");
09             str = ② ;
10             System.out.println("-->" + str);
11         }
12     }
13 }
```

실행결과

자바 문자열이 포함되어 있습니다.
--> 모든 프로그램을 Java 언어로 개발될 수 있다.



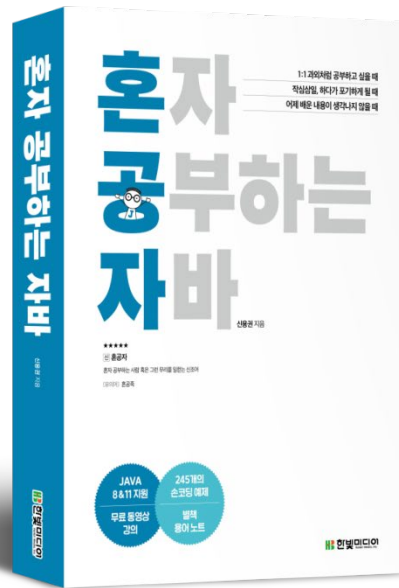
확인문제

- ❖ 문자열 "200"을 정수로 변환하는 코드와 숫자 150을 문자열로 변환하는 코드를 빈칸에 작성해보세요.

소스 코드 StringConvertExample.java

```
01 package sec01.verify.exam09;
02
03 public class StringConvertExample {
04     public static void main(String[] args) {
05         String strData1 = "200";
06         int intData1 =  ① ;
07
08         int intData2 = 150;
09         String strData2 =  ② ;
10     }
11 }
```





11-2. java.util 패키지

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- Date 클래스
- Calendar 클래스
- 키워드로 끝내는 핵심 포인트
- 확인문제



시작하기 전에

[핵심 키워드] : Date 클래스, Calendar 클래스

[핵심 포인트]

java.util 패키지는 프로그램 개발에서 자주 사용되는 자료구조일 뿐만 아니라, 날짜 정보를 제공하는 유용한 API를 포함하고 있다. 날짜 정보를 제공하는 API에 대해 알아본다.



시작하기 전에

클래스	용도
Date	날짜와 시간 정보를 저장하는 클래스
Calendar	운영체제의 날짜와 시간을 얻을 때 사용



Date 클래스

❖ Date 클래스

- 날짜를 표현하는 클래스
- Date는 객체 간 날짜 정보 주고받을 때 매개 변수나 리턴 타입으로 주로 사용

```
Date now = new Date();
```

- 원하는 날짜 형식의 문자열 얻기 위해 java.text 패키지의 SimpleDateFormat 클래스와 함께 사용

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy년 MM월 dd일 hh시 mm분 ss초");
```

- format() 메소드 호출

```
String strNow = sdf.format(now);
```



❖ 예시 - 현재 날짜 출력하기

```
01 package sec02.exam01;
02
03 import java.text.*;
04 import java.util.*;
05
06 public class DateExample {
07     public static void main(String[] args) {
08         Date now = new Date();
09         String strNow1 = now.toString();
10         System.out.println(strNow1);
11
12         SimpleDateFormat sdf =
13             new SimpleDateFormat("yyyy년 MM월 dd일 hh시 mm분 ss초");
14         String strNow2 = sdf.format(now);
15         System.out.println(strNow2);
16     }
17 }
```

실행결과

Mon Mar 11 17:19:08 KST 2019
2019년 03월 11일 05시 19분 08초



Calendar 클래스

❖ Calendar 클래스

- 추상 클래스이므로 new 연산자 사용하여 인스턴스 생성 불가
- getInstance() 메소드 이용하여 현재 운영체제에 설정된 시간대 기준으로 한 Calendar 하위 객체 얻을 수 있음

```
Calendar now = Calendar.getInstance();
```

```
int year    = now.get(Calendar.YEAR);        //연도를 리턴
int month   = now.get(Calendar.MONTH) + 1;    //월을 리턴
int day     = now.get(Calendar.DAY_OF_MONTH); //일을 리턴
int week    = now.get(Calendar.DAY_OF_WEEK);  //요일을 리턴
int amPm    = now.get(Calendar.AM_PM);        //오전/오후를 리턴
int hour    = now.get(Calendar.HOUR);         //시를 리턴
int minute  = now.get(Calendar.MINUTE);       //분을 리턴
int second  = now.get(Calendar.SECOND);       //초를 리턴
```

Calendar 클래스

- 예시 - 운영체제의 시간대 기준으로 Calendar 얻기

```
01 package sec02.exam02;
02
03 import java.util.*;
04
05 public class CalendarExample {
06     public static void main(String[] args) {
07         Calendar now = Calendar.getInstance();
08
09         int year = now.get(Calendar.YEAR);
10         int month = now.get(Calendar.MONTH) + 1;
11         int day = now.get(Calendar.DAY_OF_MONTH);
12
13         int week = now.get(Calendar.DAY_OF_WEEK);
14         String strWeek = null;
15         switch(week) {
16             case Calendar.MONDAY:
17                 strWeek = "월";
18                 break;
19             case Calendar.TUESDAY:
20                 strWeek = "화";
```



Calendar 클래스

```
21         break;
22     case Calendar.WEDNESDAY:
23         strWeek = "수";
24         break;
25     case Calendar.THURSDAY:
26         strWeek = "목";
27         break;
28     case Calendar.FRIDAY:
29         strWeek = "금";
30         break;
31     case Calendar.SATURDAY:
32         strWeek = "토";
33         break;
34     default:
35         strWeek = "일";
36     }
37
38     int amPm = now.get(Calendar.AM_PM);
39     String strAmPm = null;
40     if(amPm == Calendar.AM) {
```



Calendar 클래스

```
41     strAmPm = "오전";
42 } else {
43     strAmPm = "오후";
44 }
45
46 int hour = now.get(Calendar.HOUR);
47 int minute = now.get(Calendar.MINUTE);
48 int second = now.get(Calendar.SECOND);
49
50 System.out.print(year + "년 ");
51 System.out.print(month + "월 ");
52 System.out.println(day + "일 ");
53 System.out.print(strWeek + "요일 ");
54 System.out.println(strAmPm + " ");
55 System.out.print(hour + "시 ");
56 System.out.print(minute + "분 ");
57 System.out.println(second + "초 ");
58 }
59 }
```

실행결과

2019년 4월 29일
월요일 오후
3시 58분 56초



키워드로 끝내는 핵심 포인트

- **Date 클래스** : 날짜를 표현하는 클래스, 객체 간 날짜 정보 주고받을 때 매개 변수나 리턴 타입으로 주로 사용.
- **Calendar 클래스** : 달력을 표현하는 클래스. 추상 클래스이므로 new 연산자 사용한 인스턴스 생성이 불가능. Calendar 클래스의 정적 메소드인 getInstance() 메소드 이용하여 현재 운영체제에 설정된 시간대를 기준으로 한 Calendar 하위 객체 얻을 수 있음



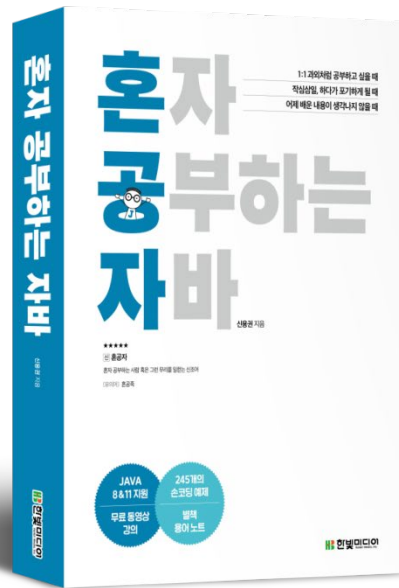
확인문제

- ❖ Date와 SimpleDateFormat 클래스를 이용해서 오늘의 날짜를 다음과 같이 출력하는 프로그램을 작성해보세요

2024년 05월 08일 화요일 10시 30분

- ❖ Calendar 클래스를 이용해서 1번과 동일한 실행결과를 출력하는 프로그램을 작성해보세요.





12-1. 멀티 스레드

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 스레드
- 메인 스레드
- 작업 스레드 생성과 실행
- 동기화 메소드
- 키워드로 끝내는 핵심 포인트
- 확인문제



시작하기 전에

[핵심 키워드] : 프로세스, 멀티 스레드, 메인 스레드, 작업 스레드, 동기화 메소드

[핵심 포인트]

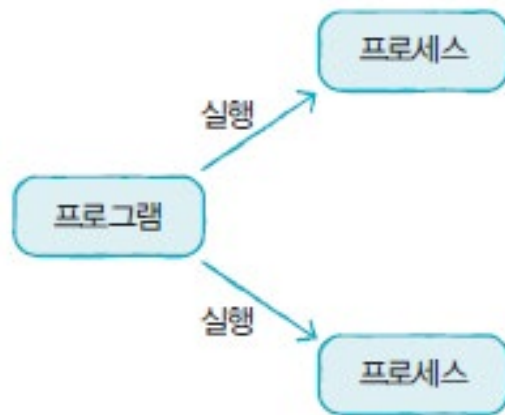
애플리케이션을 실행하면 운영체제로부터 실행에 필요한 메모리를 할당받아 애플리케이션이 실행되는데, 이것을 프로세스라 한다. 그리고 프로세스 내부에서 코드의 실행 흐름을 스레드라 한다. 애플리케이션 개발에 필수 요소인 스레드에 대해 살펴본다.



시작하기 전에

❖ 프로세스 (process)

- 실행 중인 하나의 애플리케이션
- 애플리케이션이 실행되면 운영체제로부터 실행에 필요한 메모리 할당받아 코드를 실행함
- 멀티 프로세스 역시 가능함



작업 관리자 (Task Manager) - 프로세스 탭

이름	상태	9% CPU	24% 메모리
애플리케이션 (3)			
메모장		0.4%	2.2MB
메모장		0.1%	2.2MB
작업 관리자		1.3%	34.2MB
백그라운드 프로세스 (84)			
AcroTray(32비트)		0%	1.3MB
Activation Licensing Service(32...		0%	1.6MB
Adobe Acrobat Update Service(...		0%	0.9MB

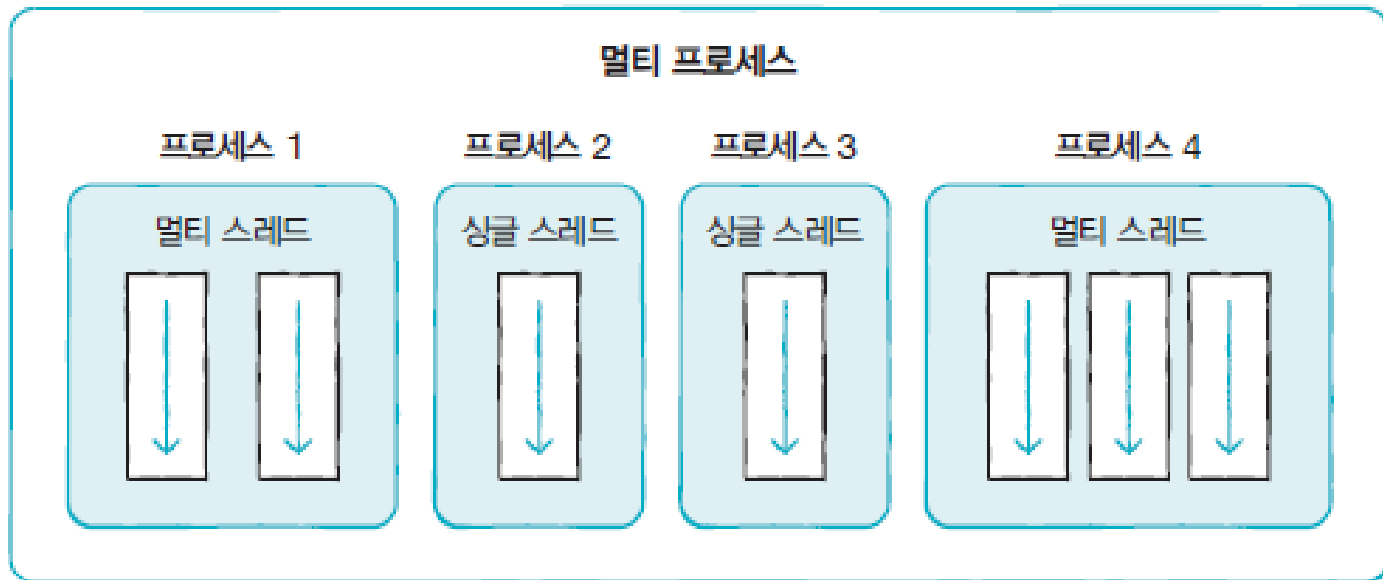


❖ 스레드 (thread)

- 한 가지 작업을 실행하기 위해 순차적으로 실행할 코드를 이어놓은 것
- 하나의 스레드는 하나의 코드 실행 이름

❖ 멀티 스레드 (multi thread)

- 하나의 프로세스로 두 가지 이상의 작업을 처리
- 데이터 분할하여 병렬로 처리하거나 다수 클라이언트 요청 처리하는 서버 개발하는 등의 용도
- 한 스레드가 예외 발생시킬 경우 프로세스 자체가 종료될 수 있음



메인 스레드

❖ 메인 스레드 (main thread)

- 모든 자바 애플리케이션은 메인 스레드가 main() 메소드 실행하면서 시작됨
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행

```
public static void main(String[] args) {  
    String data = null;  
    if(...) {  
    }  
    while(...) {  
    }  
    System.out.println("...");  
}
```

코드의 실행 흐름 → 스레드



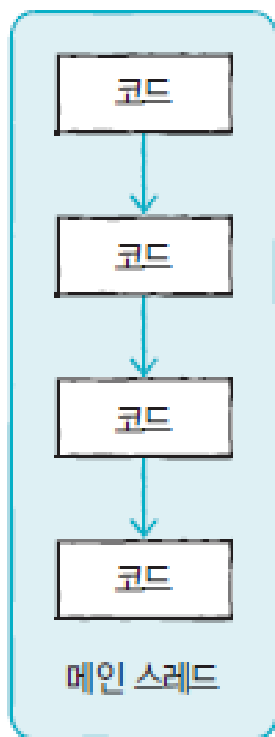
- 필요에 따라 작업 스레드들 만들어 병렬로 코드 실행 가능
- 멀티 스레드 애플리케이션에서는 실행 중인 스레드 하나라도 있으면 프로세스 종료되지 않음



메인 스레드

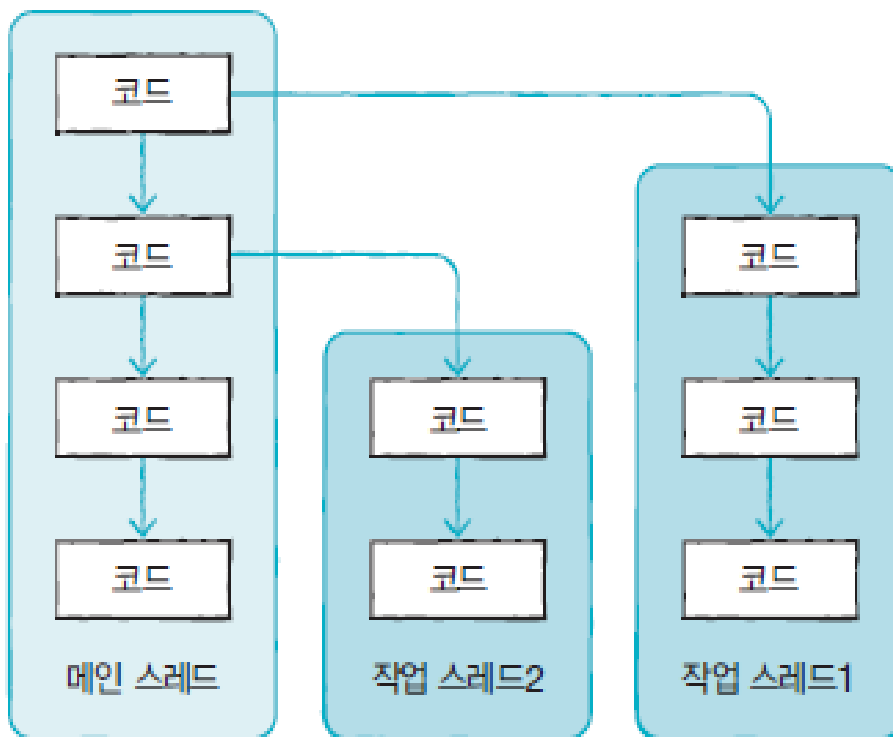
싱글 스레드 애플리케이션

프로세스



멀티 스레드 애플리케이션

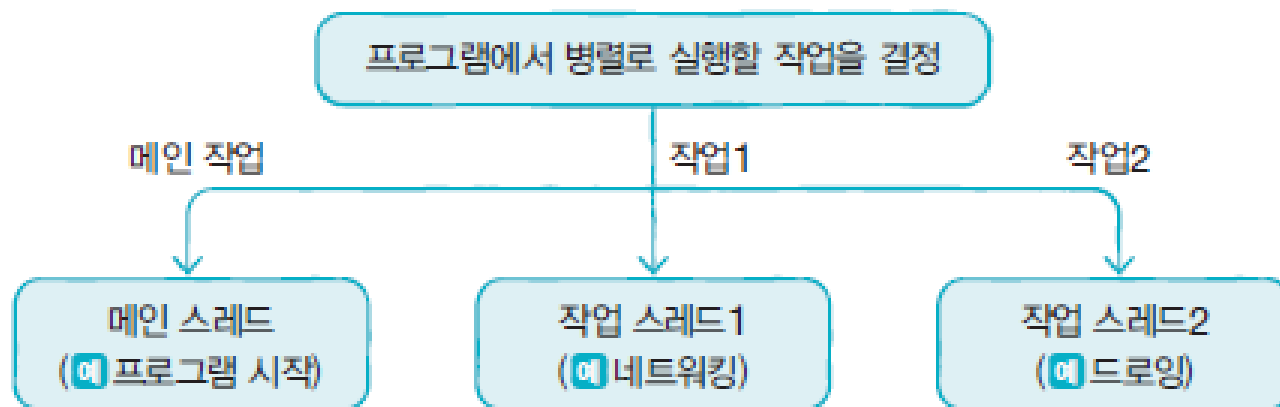
프로세스



작업 스레드 생성과 실행

❖ 작업 스레드

- 멀티 스레드로 실행하는 애플리케이션 개발하려면 몇 개의 작업을 병렬로 실행할지 우선 결정한 뒤 각 작업별로 스레드 생성해야
- 작업 스레드 역시 객체로 생성되므로 클래스 필요
Thread 클래스 상속하여 하위 클래스 만들어 사용할 수 있음



작업 스레드 생성과 실행

❖ Thread 클래스로부터 직접 생성

- Runnable을 매개값으로 갖는 생성자 호출


```
Thread thread = new Thread(Runnable target);
```

구현 객체 만들어 대입 필요

```
class Task implements Runnable {  
    public void run() {  
        스레드가 실행할 코드;  
    }  
}
```

구현 객체 매개값으로 Thread 생성자 호출하면 작업 스레드 생성됨

```
Runnable task= new Task();  
Thread thread = new Thread(task);
```



작업 스레드 생성과 실행

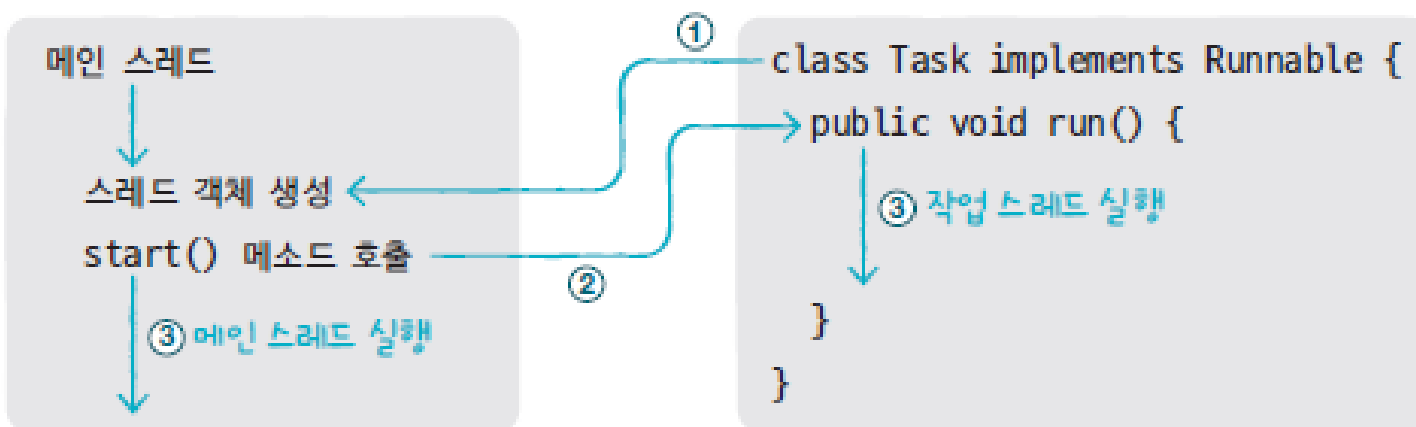
- Runnable 익명 객체를 매개값으로 사용하여 Thread 생성자 호출할 수도 있음

```
Thread thread = new Thread( new Runnable() {  
    public void run() {  
        스레드가 실행할 코드;  
    }  
});
```

← 익명 구현 객체

- start() 메소드 호출하면 작업 스레드 실행

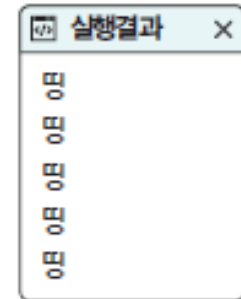
```
thread.start();
```



작업 스레드 생성과 실행

■ 예시 - 메인 스레드만 이용한 경우

```
01 package sec01.exam01;
02
03 import java.awt.Toolkit;
04
05 public class BeepPrintExample1 {
06     public static void main(String[] args) {
07         Toolkit toolkit = Toolkit.getDefaultToolkit(); ← Toolkit 객체 얻기
08         for(int i=0; i<5; i++) {
09             toolkit.beep(); ← 비프음 발생
10             try { Thread.sleep(500); } catch(Exception e) {}
11         } ↑ 0.5초간 일시 정지
12
13         for(int i=0; i<5; i++) {
14             System.out.println("땡");
15             try { Thread.sleep(500); } catch(Exception e) {}
16         } ↑ 0.5초간 일시 정지
17     }
18 }
```



작업 스레드 생성과 실행

- 예시 - 비프음을 들려주는 작업 정의 - Runnable 구현 클래스

```
01 package sec01.exam02;
02
03 import java.awt.Toolkit;
04
05 public class BeepTask implements Runnable {
06     public void run() {
07         Toolkit toolkit = Toolkit.getDefaultToolkit();
08         for(int i=0; i<5; i++) {
09             toolkit.beep();
10             try { Thread.sleep(500); } catch(Exception e) {}
11         }
12     }
13 }
```


← 스레드 실행내용



작업 스레드 생성과 실행

- 예시 - 비프음 - 메인 스레드와 작업 스레드가 동시에 실행

```
01 package sec01.exam02;
02
03 public class BeepPrintExample2 {
04     public static void main(String[] args) {
05         Runnable beepTask = new BeepTask();
06         Thread thread = new Thread(beepTask);
07         thread.start();
08
09         for(int i=0; i<5; i++) {
10             System.out.println("땡");
11             try { Thread.sleep(500); }
12                 catch(Exception e) {}
13         }
14     }
15 }
```



```
public void run() {
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    for(int i=0; i<5; i++) {
        toolkit.beep();
        try { Thread.sleep(500); } catch(Exception e) {}
    }
}
```



작업 스레드 생성과 실행

❖ Thread 하위 클래스로부터 생성

- Thread의 하위 클래스로 작업 스레드를 정의하면서 작업 내용을 포함
- 작업 스레드 클래스 정의하는 법

```
public class WorkerThread extends Thread {  
    @Override  
    public void run() {  
        스레드가 실행할 코드;  
    }  
}  
Thread thread = new WorkerThread();
```

← run() 메소드 재정의

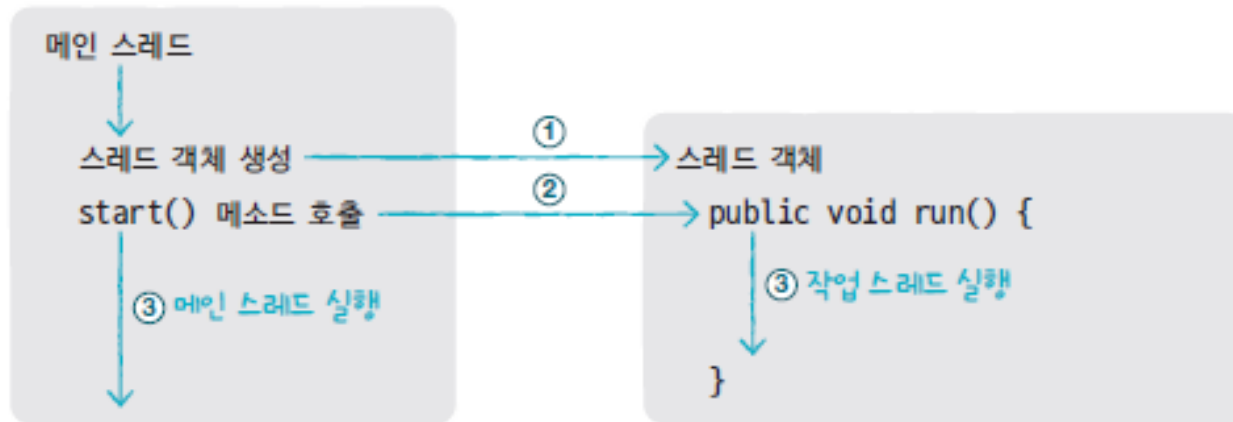
```
Thread thread = new Thread() {  
    public void run() {  
        스레드가 실행할 코드;  
    }  
};
```

← 익명 자식 객체

작업 스레드 생성과 실행

- 작업 스레드 객체 생성 후 start() 메소드 호출하면 run() 메소드가 실행

```
thread.start();
```



작업 스레드 생성과 실행

■ 예시 - 비프음을 들려주는 스레드

```
01 package sec01.exam04;
02
03 import java.awt.Toolkit;
04
05 public class BeepThread extends Thread {
06     @Override
07     public void run() {
08         Toolkit toolkit = Toolkit.getDefaultToolkit();
09         for(int i=0; i<5; i++) {
10             toolkit.beep();
11             try { Thread.sleep(500); } catch(Exception e) {}
12         }
13     }
14 }
```

← 스레드 실행 내용



작업 스레드 생성과 실행

- 예시 - 메인 스레드와 작업 스레드가 동시에 실행

```
01 package sec01.exam04;
02
03 public class BeepPrintExample4 {
04     public static void main(String[] args) {
05         Thread thread = new BeepThread();
06         thread.start();
07
08         for(int i=0; i<5; i++) {
09             System.out.println("띵");
10             try { Thread.sleep(500); }
11                 catch(Exception e) {}
12         }
13     }
14 }
```

메인 스레드

BeepThread

```
public void run() {
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    for(int i=0; i<5; i++) {
        toolkit.beep();
        try { Thread.sleep(500); } catch(Exception e) {}
    }
}
```



작업 스레드 생성과 실행

- 메인 스레드는 'main' 이름 가지며, 우리가 직접 생성한 스레드는 자동적으로 'Thread-n' 이름 설정됨
setName() 메소드로 이름 변경 가능

```
thread.setName("스레드 이름");
```

getName() 메소드로 스레드 이름 알 수 있음

```
thread.getName();
```

currentThread() 메소드로 현재 스레드의 참조 얻을 수 있음

```
Thread thread = Thread.currentThread();
```



작업 스레드 생성과 실행

■ 예시 - 메인 스레드 이름 출력 및 UserThread 생성 및 시작

```
01 package sec01.exam06;
02
03 public class ThreadNameExample {
04     public static void main(String[] args) {
05         Thread mainThread = Thread.currentThread(); ← 이 코드를 실행하는 스레드 객체 얻기
06         System.out.println("프로그램 시작 스레드 이름: " + mainThread.getName());
07
08         ThreadA threadA = new ThreadA(); ← ThreadA 생성
09         System.out.println("작업 스레드 이름: " + threadA.getName()); ← 스레드 이름 얻기
10         threadA.start(); ← ThreadA 시작
11
12         ThreadB threadB = new ThreadB(); ← ThreadB 생성
13         System.out.println("작업 스레드 이름: " + threadB.getName()); ← 스레드 이름 얻기
14         threadB.start(); ← ThreadB 시작
15     }
16 }
```

실행결과

```
프로그램 시작 스레드 이름: main
작업 스레드 이름: ThreadA
ThreadA가 출력한 내용
ThreadA가 출력한 내용
작업 스레드 이름: Thread-1
Thread-1가 출력한 내용
Thread-1가 출력한 내용
```

작업 스레드 생성과 실행

■ 예시 - ThreadA 클래스

```
01 package sec01.exam06;
02
03 public class ThreadA extends Thread {
04     public ThreadA() {
05         setName("ThreadA"); ← 스레드 이름 설정
06     }
07
08     public void run() {
09         for(int i=0; i<2; i++) {
10             System.out.println(getName() + "가 출력한 내용"); ← ThreadA 실행 내용
11         }
12     }
13 }
```

↑ 스레드 이름 얻기



작업 스레드 생성과 실행

■ 예시 - ThreadB 클래스

```
01 package sec01.exam06;  
02  
03 public class ThreadB extends Thread {  
04     public void run() {  
05         for(int i=0; i<2; i++) {  
06             System.out.println(getName() + "가 출력한 내용");  
07         }  
08     }  
09 }
```

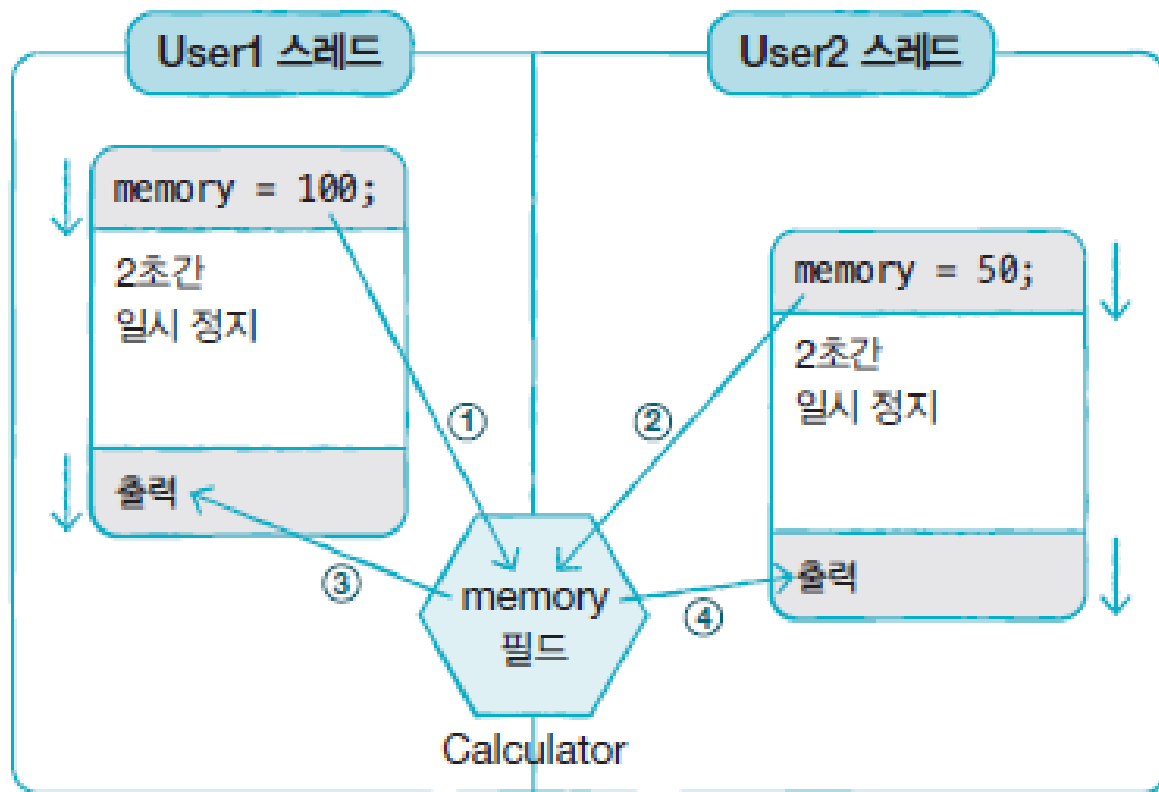
← ThreadB 실행 내용

↑ 스레드 이름 얻기



❖ 공유 객체를 사용할 때 주의할 점

- 멀티 스레드 프로그램에서 스레드들이 객체 공유해서 작업해야 하는 경우 의도했던 것과 다른 결과 나올 수 있음



동기화 메소드

■ 메인 스레드가 실행하는 코드

```
01 package sec01.exam07;
02
03 public class MainThreadExample {
04     public static void main(String[] args) {
05         Calculator calculator = new Calculator();
06
07         User1 user1 = new User1(); ← User1 스레드 생성
08         user1.setCalculator(calculator); ← 공유 객체 설정
09         user1.start(); ← User1 스레드 시작
10
11         User2 user2 = new User2(); ← User2 스레드 생성
12         user2.setCalculator(calculator); ← 공유 객체 설정
13         user2.start(); ← User2 스레드 시작
14     }
15 }
```

실행결과

User1: 50
User2: 50



동기화 메소드

■ 공유 객체

```
01 package sec01.exam07;
02
03 public class Calculator {
04     private int memory;
05
06     public int getMemory() {
07         return memory;
08     }
09
10     public void setMemory(int memory) { ← 계산기 메모리에 값을 저장하는 메소드
11         this.memory = memory; ← 매개값을 memory 필드에 저장
12         try {
13             Thread.sleep(2000); ← 스레드를 2초간 일시 정지시킴
14         } catch (InterruptedException e) {}
15         System.out.println(Thread.currentThread().getName() + ": " + this.memory);
16     }
17 }
```



동기화 메소드

■ User1 스레드

```
01 package sec01.exam07;
02
03 public class User1 extends Thread {
04     private Calculator calculator;
05
06     public void setCalculator(Calculator calculator) {
07         this.setName("User1"); ← 스레드 이름을 User1로 설정
08         this.calculator = calculator; ← 공유 객체인 Calculator를 필드에 저장
09     }
10
11     public void run() {
12         calculator.setMemory(100); ← 공유 객체인 Calculator의
13     }                                     메모리에 100을 저장
14 }
```



동기화 메소드

■ User2 스레드

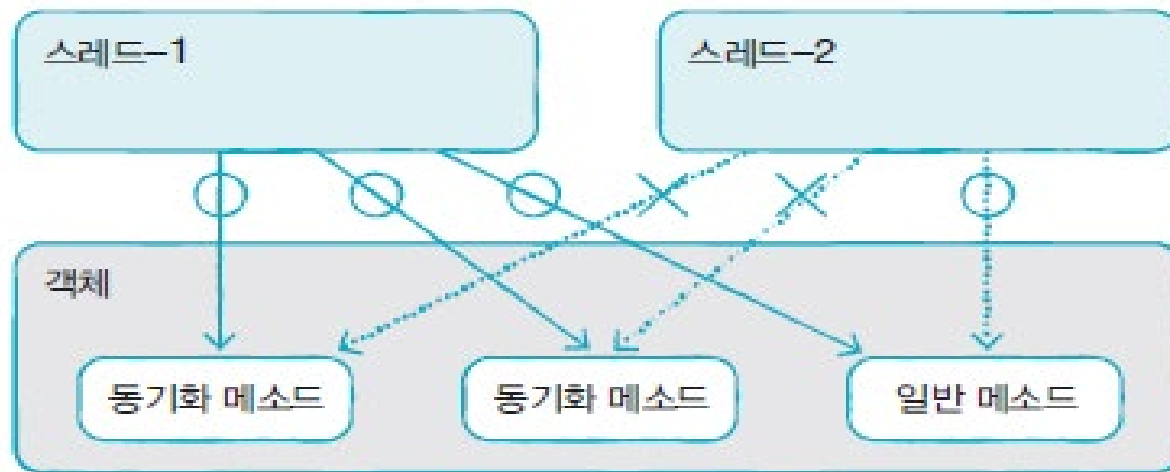
```
01 package sec01.exam07;
02
03 public class User2 extends Thread {
04     private Calculator calculator;
05
06     public void setCalculator(Calculator calculator) {
07         this.setName("User2"); ← 스레드 이름을 User2로 설정
08         this.calculator = calculator; ← 공유 객체인 Calculator를 필드에 저장
09     }
10
11     public void run() {
12         calculator.setMemory(50); ← 공유 객체인 Calculator의
13                                     메모리에 50을 저장
14     }
15 }
```



❖ 동기화 메소드

- 스레드가 사용 중인 객체를 다른 스레드가 변경할 수 없게 하려면 스레드 작업 끝날 때까지 객체에 잠금 걸어야 함
- 임계 영역 (critical section) : 단 하나의 스레드만 실행할 수 있는 코드 영역
- 동기화 (synchronized) 메소드 : 스레드가 객체 내부의 동기화 메소드 실행하면 즉시 객체에 잠금 걸림

```
public synchronized void method() {  
    임계 영역; //단 하나의 스레드만 실행  
}
```



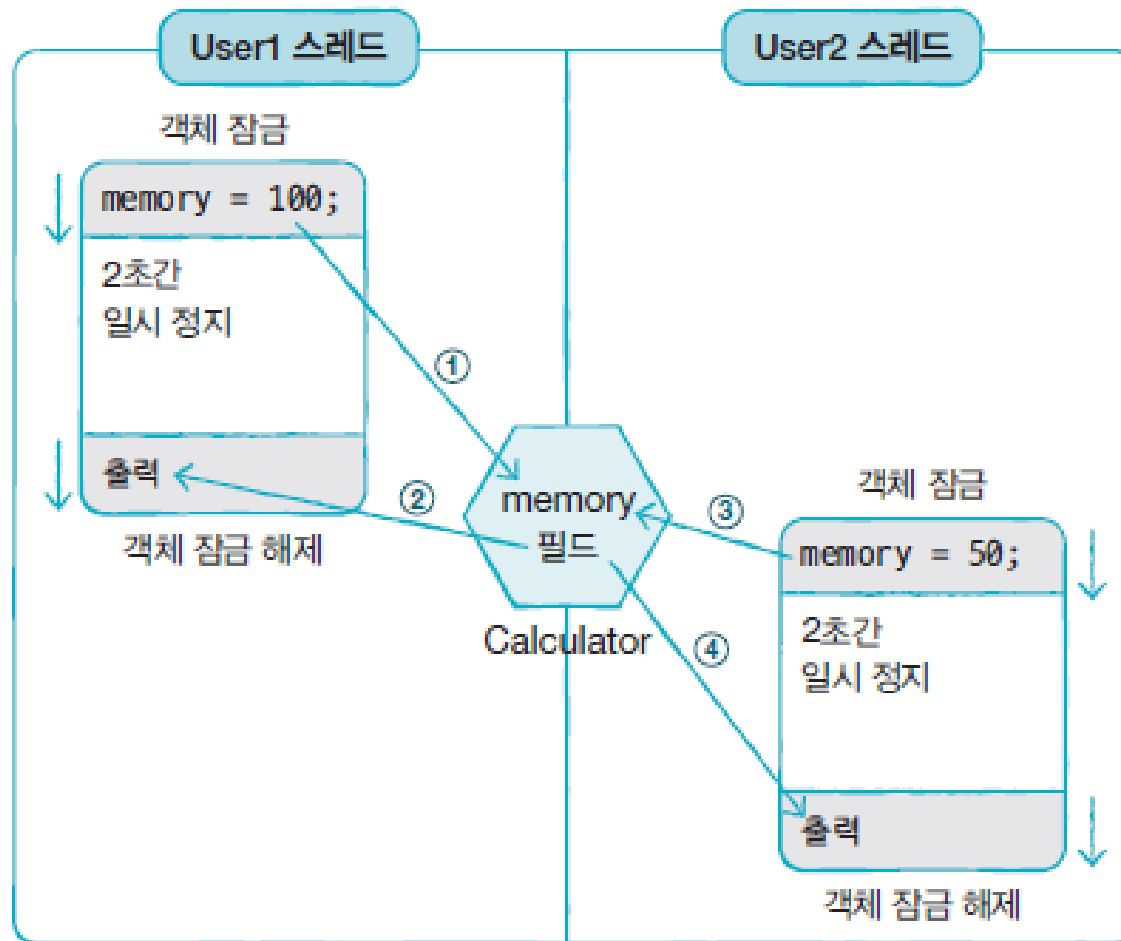
동기화 메소드

- 동기화 메소드로 수정된 공유 객체

```
01 package sec01.exam08;
02
03 public class Calculator {
04     private int memory;
05
06     public int getMemory() {
07         return memory;
08     }
09
10     public synchronized void setMemory(int memory) {
11         this.memory = memory;
12         try {
13             Thread.sleep(2000);
14         } catch (InterruptedException e) {}
15         System.out.println(Thread.currentThread().getName() + ": " + this.memory);
16     }
17 }
```

동기화 메소드

- MainThreadExample.java 실행하면 User1은 100, User2는 50



키워드로 끝내는 핵심 포인트

- **프로세스** : 애플리케이션 실행하면 운영체제로부터 실행에 필요한 메모리 할당받아 애플리케이션 실행됨.
- **멀티 스레드** : 하나의 프로세스 내에 동시 실행하는 스레드가 2개 이상인 경우
- **메인 스레드** : 자바의 모든 어플리케이션은 메인 스레드가 main() 메소드 실행하면서 시작. main() 메소드의 첫 코드부터 아래로 순차 실행하고, main() 메소드의 마지막 코드 실행하거나 return 문 만나면 실행이 종료
- **작업 스레드** : 메인 작업 이외에 병렬 작업의 수만큼 생성하는 스레드. 객체로서 생성되기 때문에 클래스 필요. Thread 클래스를 직접 객체화해서 생성할 수도 있고, Thread 클래스를 상속해서 하위 클래스 만들어 생성할 수도 있음
- **동기화 메소드** : 멀티 스레드 프로그램에서 단 하나의 스레드만 실행할 수 있는 코드 영역을 임계 영역이라 함. 이를 지정하기 위해 동기화 메소드가 제공됨. 스레드가 객체 내부의 동기화 메소드 실행하면 즉시 객체에 잠금 걸어 다른 스레드가 동기화 메소드 실행하지 못하게 함



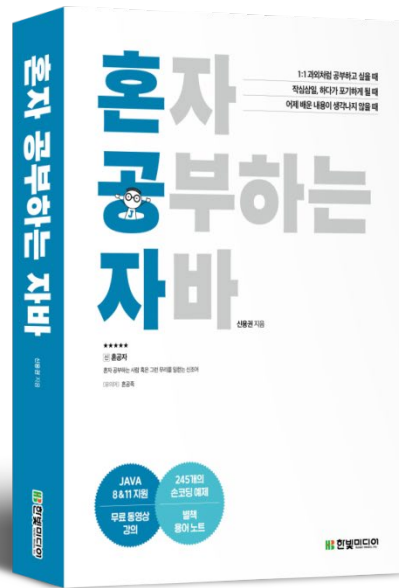
❖ 스레드에 대한 설명 중 틀린 것은 무엇입니까?

- 자바 애플리케이션은 메인 스레드가 main() 메소드를 실행한다
- 작업 스레드 클래스는 Thread 클래스를 상속해서 만들 수 있다
- Runnable 객체는 스레드가 실행해야 할 코드를 가지고 있는 객체라 볼 수 있다
- 스레드 실행을 시작하려면 run() 메소드를 호출해야 한다

❖ 동기화 메소드에 대한 설명 중 틀린 것은 무엇입니까?

- 동기화 메소드는 싱글 스레드 환경에서는 필요하지 않다
- 스레드가 동기화 메소드를 실행할 때 다른 스레드는 일반 메소드를 호출할 수 없다
- 스레드가 동기화 메소드를 실행할 때 다른 스레드는 다른 동기화 메소드를 호출할 수 없다
- 동기화 메소드 선언부에는 synchronized 키워드가 필요하다





12-2. 스레드 제어

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 스레드 상태
- 스레드 상태 제어
- 키워드로 끝내는 핵심 포인트
- 확인문제



시작하기 전에

[핵심 키워드] : 스레드 상태, 일시정지, 안전한 종료, 데몬 스레드

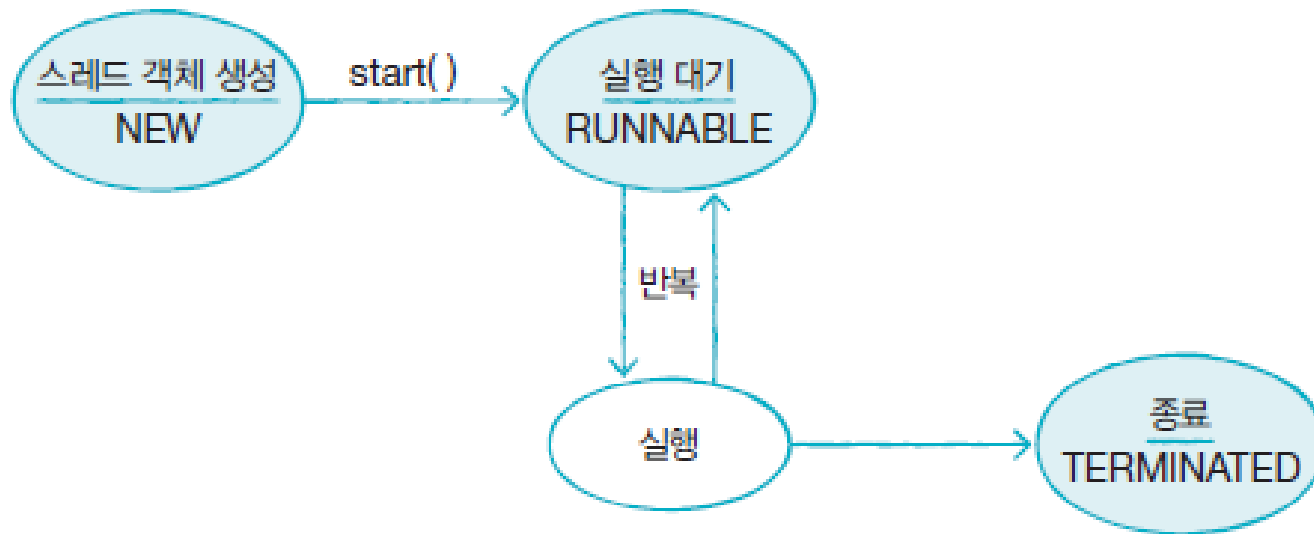
[핵심 포인트]

스레드를 생성하고 시작하면 다양한 상태를 가지게 된다. 이러한 스레드 상태는 자동으로 변경될 수도 있고 코드에 의해 변경될 수도 있다. 스레드의 상태를 변경해 제어하는 방법에 대해 알아본다.



시작하기 전에

- ❖ 스레드 객체 생성하고 start() 메소드를 호출하면 바로 실행되는 것이 아니라 실행 대기 상태가 됨
 - 실행 상태 스레드는 run() 메소드를 모두 실행하기 전 다시 실행 대기 상태로 돌아갈 수 있음
 - 실행 대기 상태에 있는 다른 스레드가 선택되어 실행 상태가 되기도 함
 - 실행 상태에서 run() 메소드의 내용이 모두 실행되면 스레드 실행이 멈추고 종료 상태가 됨



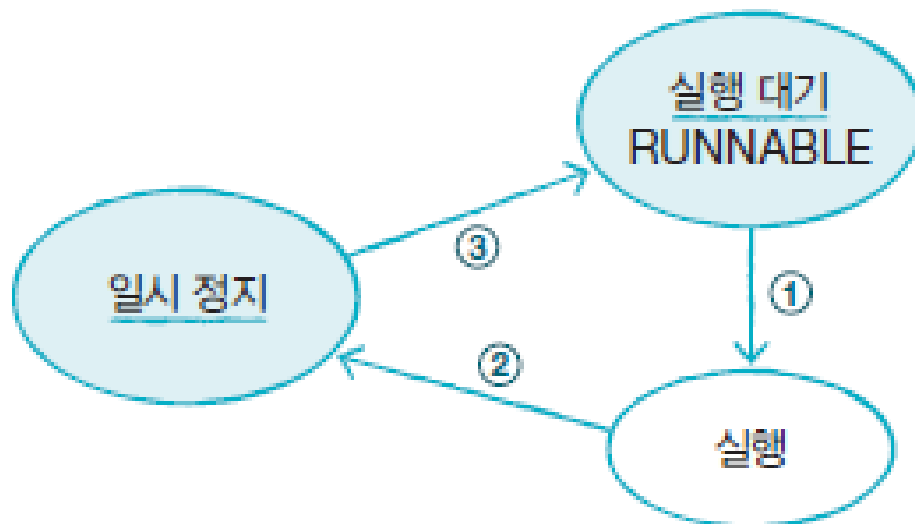
스레드 상태

❖ 실행 (running) 상태

- 실행 대기 상태의 스레드 중에서 운영체제가 하나를 선택하여 CPU가 run() 메소드를 실행하도록 하
- run() 메소드 모두 실행하기 전에 다시 실행 대기 상태로 돌아갈 수 있음

❖ 종료 (terminated) 상태

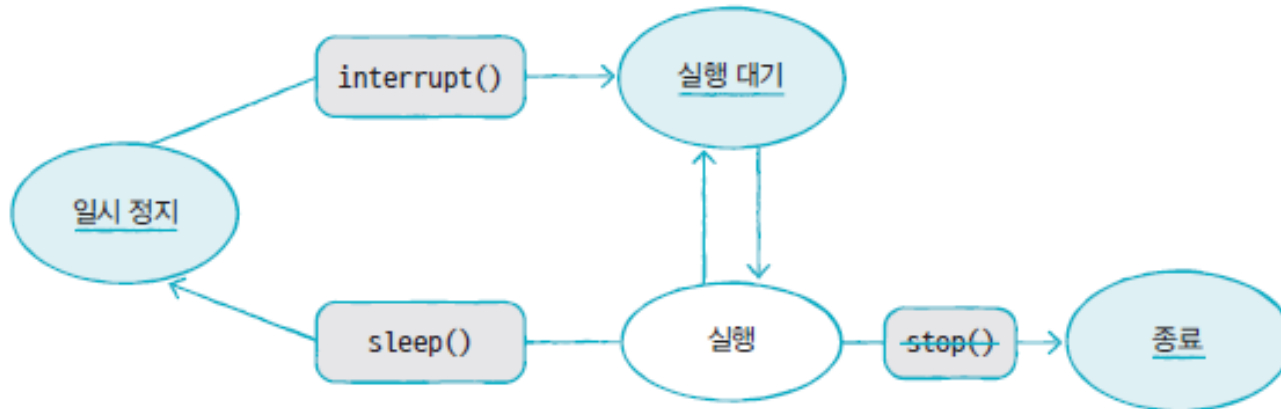
- 실행 상태에서 run() 메소드 종료되면 더 이상 실행할 코드 없기 때문에 스레드 실행이 정지됨



스레드 상태 제어

❖ 스레드 상태 제어

- 실행 중인 스레드의 상태를 변경
- 스레드 상태 변화에 필요한 메소드를 정확히 파악해야



메소드	설명
<code>interrupt()</code>	일시 정지 상태의 스레드에서 <code>InterruptedException</code> 을 발생시켜, 예외 처리 코드(catch)에서 실행 대기 상태로 가거나 종료 상태로 갈 수 있도록 합니다.
<code>sleep(long millis)</code>	주어진 시간 동안 스레드를 일시 정지 상태로 만듭니다. 주어진 시간이 지나면 자동적으로 실행 대기 상태가 됩니다.
<code>stop()</code>	스레드를 즉시 종료합니다. 불안정한 종료를 유발하므로 사용하지 않는 것이 좋습니다.



스레드 상태 제어

❖ 주어진 시간 동안 일시 정지

- Thread 클래스의 정적 메소드 sleep() 사용

```
try {  
    Thread.sleep(1000);  
} catch (InterruptedException e) {  
    //interrupt() 메소드가 호출되면 실행  
}
```



스레드 상태 제어

- 예시 - 3초 주기로 10번 비프음 발생

```
01 package sec02.exam01;
02
03 import java.awt.Toolkit;
04
05 public class SleepExample {
06     public static void main(String[] args) {
07         Toolkit toolkit = Toolkit.getDefaultToolkit();
08         for(int i=0; i<10; i++) {
09             toolkit.beep();
10             try {
11                 Thread.sleep(3000); ← 3초 동안 메인 스레드를
12             } catch (InterruptedException e) { }                  일시 정지 상태로 만들음
13
14         }
15     }
16 }
```



스레드 상태 제어

❖ 스레드의 안전한 종료

- 실행 중인 스레드를 즉시 종료해야 하는 경우
- stop 플래그를 이용하는 방법

```
public class XXXThread extends Thread {  
    private boolean stop; //stop 플래그 필드  
  
    public void run() {  
        while( !stop ) { ← stop이 true가 되면 run()이 종료  
            스레드가 반복 실행하는 코드;  
        }  
        //스레드가 사용한 자원 정리  
    }  
}
```



스레드 상태 제어

■ 예시 - 1초 출력 후 스레드를 중지

```
01 package sec02.exam02;  
02  
03 public class StopFlagExample {  
04     public static void main(String[] args) {  
05         PrintThread1 printThread = new PrintThread1();  
06         printThread.start();  
07  
08         try { Thread.sleep(1000); } catch (InterruptedException e) {}  
09  
10         printThread.setStop(true); ← 스레드를 종료하기 위해  
11     }                                stop 필드를 true로 변경  
12 }
```

실행결과		×
실행	중	
실행	중	
실행	중	
자원	정리	
실행	종료	



스레드 상태 제어

- 예시 - 무한 반복해서 출력하는 스레드

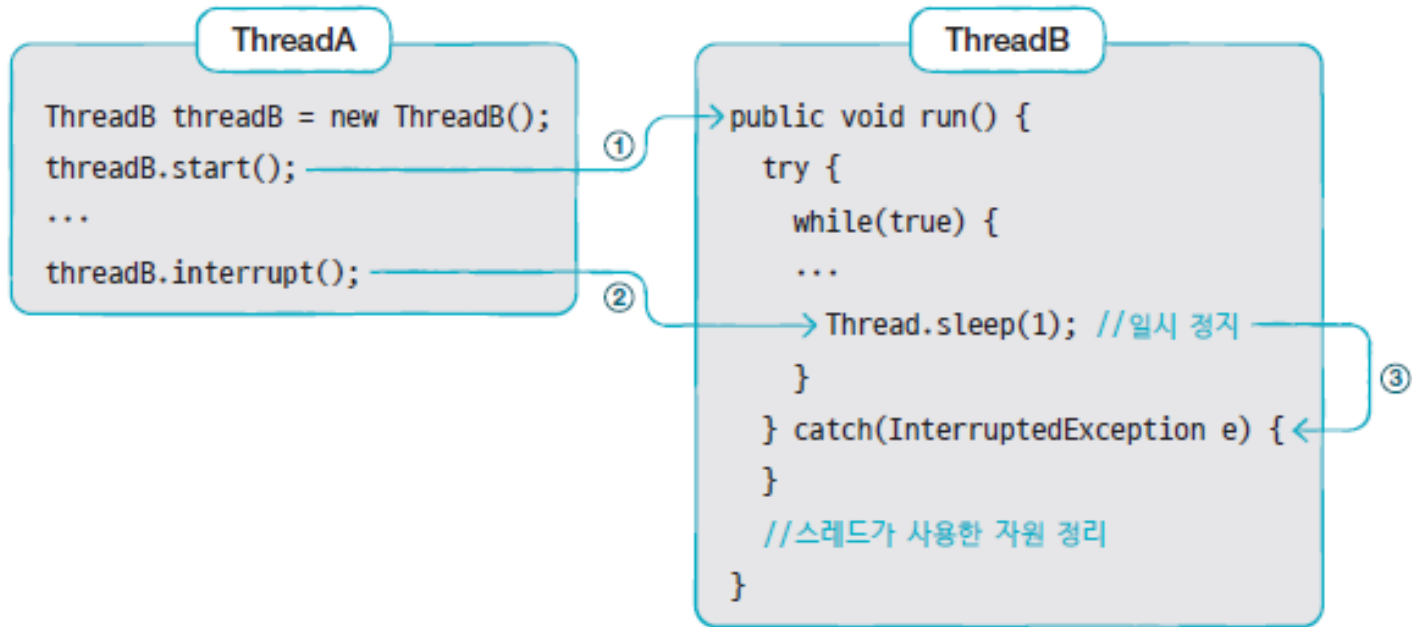
```
01 package sec02.exam02;
02
03 public class PrintThread1 extends Thread {
04     private boolean stop;
05
06     public void setStop(boolean stop) {
07         this.stop = stop;
08     }
09
10     public void run() {
11         while(!stop) {
12             System.out.println("실행 중");
13         }
14         System.out.println("자원 정리");
15         System.out.println("실행 종료");
16     }
17 }
```

stop이 true가 될 때



스레드 상태 제어

■ interrupt() 메소드 이용하는 방법



ThreadA가 ThreadB의 interrupt() 메소드 실행하면 ThreadB가 sleep() 메소드로 일시정지 상태 될 때 ThreadB에서 InterruptedException 발생, 예외 처리 블록으로 이동



스레드 상태 제어

■ 예시 - 1초 후 스레드를 중지

```
01 package sec02.exam03;
02
03 public class InterruptExample {
04     public static void main(String[] args) {
05         Thread thread = new PrintThread2();
06         thread.start();
07
08         try { Thread.sleep(1000); catch (InterruptedException e) {}
09
10         thread.interrupt(); ← 스레드를 종료하기 위해
                               InterruptedException을 발생시킴
11     }
12 }
```

실행결과		×
실행	중	
실행	중	
실행	중	
자원	정리	
실행	종료	



스레드 상태 제어

- 예시 - 무한 반복해서 출력하는 스레드

```
01 package sec02.exam03;
02
03 public class PrintThread2 extends Thread {
04     public void run() {
05         try {
06             while(true) {
07                 System.out.println("실행 중");
08                 Thread.sleep(1);
09             }
10         } catch (InterruptedException e) {}
11
12         System.out.println("자원 정리");
13         System.out.println("실행 종료");
14     }
15 }
```

InterruptedException 발생



스레드 상태 제어

- 스레드가 실행 대기 혹은 실행 상태에 있을 때 interrupt() 메소드 실행되면 즉시 InterruptedException 발생하는 것이 아니라, 스레드가 미래에 일시정지 상태가 되면 InterruptedException 발생하는 것
- 일시정지 만들지 않는 경우
interrupted()와 isInterrupted() 메소드는 interrupt() 메소드가 호출된 경우 true 리턴

```
boolean status = Thread.interrupted();  
boolean status = objThread.isInterrupted();
```



스레드 상태 제어

■ 예시 - 무한 반복해서 출력하는 스레드

```
01 package sec02.exam04;
02
03 public class PrintThread2 extends Thread {
04     public void run() {
05         while(true) {
06             System.out.println("실행 중");
07             if(Thread.interrupted()) {
08                 break;
09             }
10         }
11         ← while문을 빠져나옴
12         System.out.println("자원 정리");
13         System.out.println("실행 종료");
14     }
15 }
```



키워드로 끝내는 핵심 포인트

- **스레드 상태** : 스레드를 생성하고 시작하면 스레드는 다양한 상태를 가지게 되며, 이는 자동으로 혹은 코드에 의해 변경될 수 있다
- **일시 정지** : 실행 중인 스레드를 일정 시간 멈추게 하려는 경우 Thread 클래스의 정적 메소드인 sleep()을 사용. Thread.sleep() 메소드를 호출한 스레드는 주어진 시간 동안 일시정지 상태가 되고 다시 실행 대기 상태로 돌아감
- **안전한 종료** : 스레드를 안전하게 종료하기 위해 stop 플래그나 interrupt() 메소드를 이용할 수 있다.
- **데몬 스레드** : 주 스레드의 작업을 돕는 보조적 역할 하는 스레드. 주 스레드가 종료되면 자동 종료된다.



- ❖ 스레드 상태 제어 메소드에 대한 설명 중 틀린 것은 무엇입니까?
 - start() 메소드는 실행 대기 상태로 만들어준다
 - sleep() 메소드는 일시 정지 상태로 만들어준다
 - interrupt() 메소드는 실행 상태를 간섭해서 일시 정지 상태로 만들어준다
 - 일시 정지 상태에서 실행 상태로 변경하는 메소드는 없다
- ❖ 메인 스레드에서 1초 후 MovieThread의 interrupt() 메소드를 호출해서 MovieThread를 안전하게 종료하고 싶습니다. 빈칸에 알맞은 코드를 작성해보세요

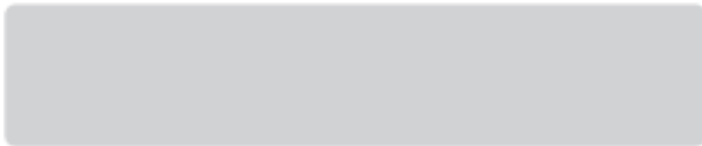
소스 코드 ThreadExample.java

```
01 package sec02.verify.exam03;
02
03 public class ThreadExample {
04     public static void main(String[] args) {
05         Thread thread = new MovieThread();
06         thread.start();
07
08         try { Thread.sleep(1000); } catch (InterruptedException e) {}
09
10         thread.interrupt();
11     }
12 }
```



확인문제

소스 코드 MovieThread.java

```
01 package sec02.verify.exam03;
02
03 public class MovieThread extends Thread {
04     @Override
05     public void run() {
06         while(true) {
07             System.out.println("동영상을 재생합니다.");
08             
09
10
11         }
12     }
13 }
```

