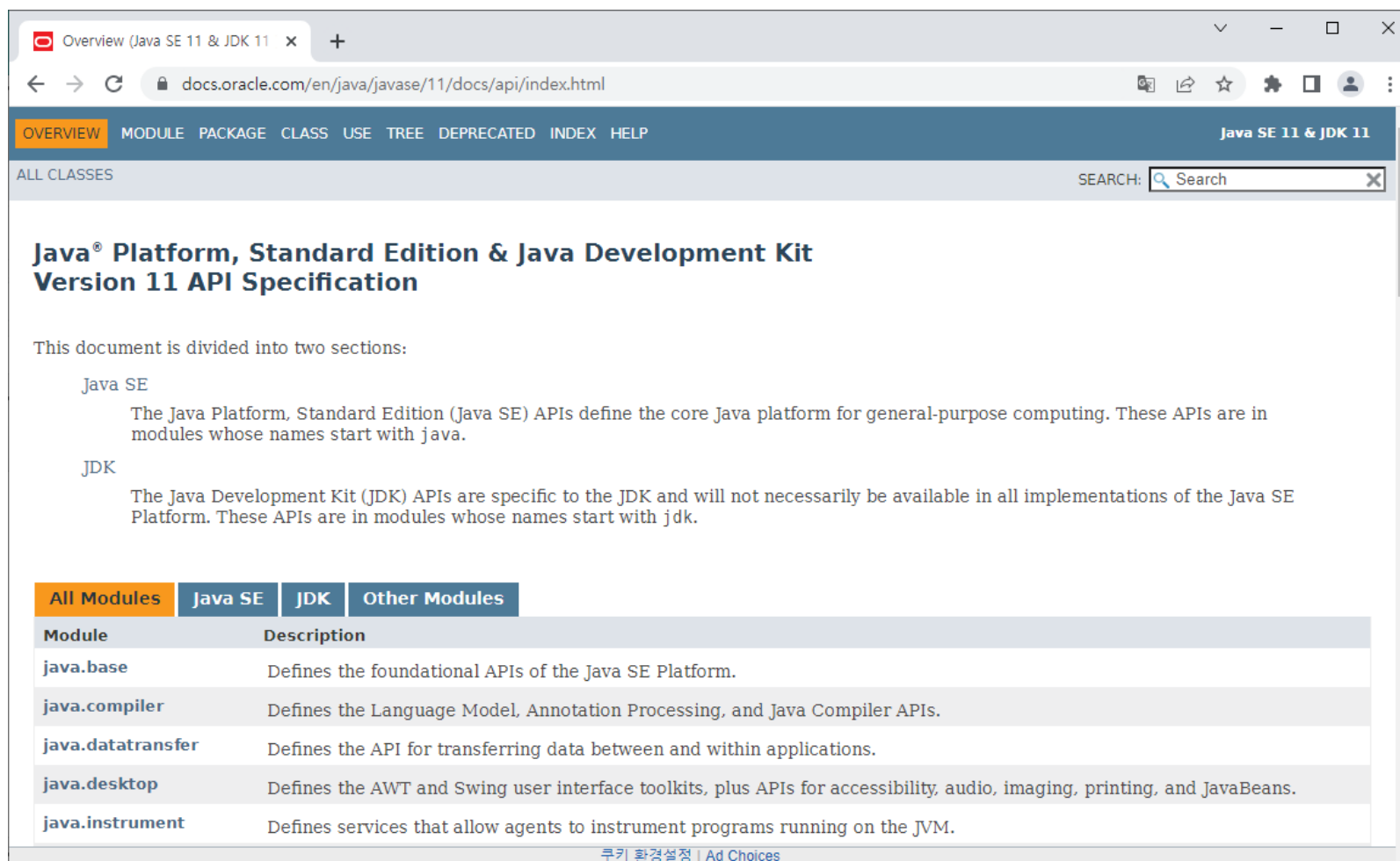




자바 API

# 온라인 자바 API 참조

- 오라클에서 제공하는 자바 클래스의 상세 정보
- <https://docs.oracle.com/en/java/javase/11/docs/api/index.html> (자바 11 기준)



Overview (Java SE 11 & JDK 11) x +

docs.oracle.com/en/java/javase/11/docs/api/index.html

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 11 & JDK 11

ALL CLASSES SEARCH: Search

## Java® Platform, Standard Edition & Java Development Kit Version 11 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules	Java SE	JDK	Other Modules
Module	Description		
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.		
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.		
<code>java.datatransfer</code>	Defines the API for transferring data between and within applications.		
<code>java.desktop</code>	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.		
<code>java.instrument</code>	Defines services that allow agents to instrument programs running on the JVM.		

쿠키 환경설정 | Ad Choices

# Object 클래스

- 패키지 : java.lang
- 모든 클래스의 슈퍼 클래스
  - ✓ 모든 클래스는 명시하지 않아도 강제로 Object 클래스를 상속 받음
  - ✓ 모든 클래스의 타입으로 사용할 수 있음
  - ✓ 모든 클래스가 사용할 수 있는 공통 메소드를 가지고 있음
- 주요 공통 메소드

메소드	역할
boolean equals(Object obj)	매개변수 obj 객체와 현재 객체가 동일하면 true 반환
Class<T> getClass()	현재 객체의 클래스타입을 반환
int hashCode()	현재 객체의 해시코드를 반환
String toString()	현재 객체의 정보를 문자열 형태로 반환
void notify()	현재 객체의 대기(wait)중인 하나의 스레드를 깨움
void notifyAll()	현재 객체의 대기(wait)중인 모든 스레드를 깨움
void wait()	현재 스레드를 일시적으로 대기(wait)시킴

# 객체 정보를 문자열로 변환하는 toString()

- Object 클래스의 toString() 메소드
  - ✓ 클래스의 이름과 객체의 해시코드를 @로 연결한 문자열을 반환
  - ✓ 객체의 해시코드를 통해서 객체의 참조값을 확인 가능

```
public String toString() {  
    return getClass().getName() + "@" + Integer.toHexString(hashCode());  
}
```

- toString() 메소드가 사용되는 경우
  - ✓ System.out.println()과 같은 출력 메소드에 객체가 전달되는 경우
  - ✓ 문자열 연산에 객체가 사용되는 경우

```
User user = new User();  
System.out.println(user);      // User@7637f22  
  
String message = "Hello " + user;  
System.out.println(message);  // Hello User@7637f22
```

System.out.println(user.toString());

String message = "Hello" + user.toString();

# toString() 오버라이드

- Object 클래스의 toString() 메소드는 객체의 필드 정보를 출력하지 못함
- 객체의 필드 정보를 출력할 수 있도록 toString() 메소드를 오버라이드 하는 것이 일반적임

```
public class User {  
  
    private String name;  
    private int age;  
  
    public User(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return "이름:" + name + ",나이:" + age + "살";  
    }  
  
    public static void main(String[] args) {  
        User user = new User("홍길동", 30);  
        System.out.println(user);  
    }  
}
```

System.out.println(user.toString());으로  
처리되기 때문에 아래와 같이 출력됨

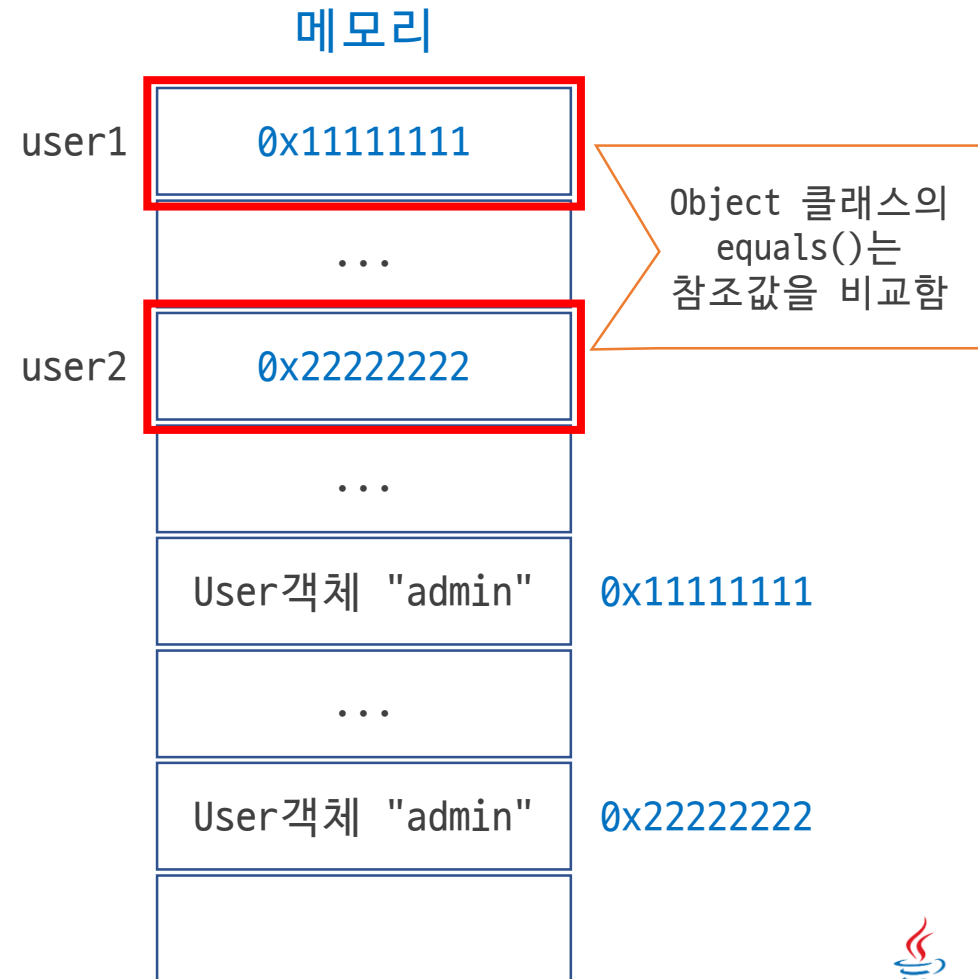
이름:홍길동,나이:30살

# Object 클래스의 equals()

- Object 클래스의 equals() 메소드는 객체의 참조값을 비교해서 같으면 true를 반환

```
public class User {  
    private String id;  
  
    public User(String id) {  
        this.id = id;  
    }  
  
    public static void main(String[] args) {  
        User user1 = new User("admin");  
        User user2 = new User("admin");  
        System.out.println(user1.equals(user2));  
    }  
}
```

객체 user1과 user2는 아이디가 "admin"으로 동일  
But 서로 다른 객체이기 때문에  
equals() 메소드를 호출하면 false가 반환됨



# equals() 오버라이드

- Object 클래스의 equals() 메소드는 객체의 필드값을 비교하지 못하므로 오버라이드해서 필드값을 비교할 수 있도록 처리함

```
public class User {  
  
    private String id;  
  
    public User(String id) {  
        this.id = id;  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        return id.equals(((User)obj).id);  
    }  
  
    public static void main(String[] args) {  
        User user1 = new User("admin");  
        User user2 = new User("admin");  
        System.out.println(user1.equals(user2));  
    }  
}
```

두 객체의 id 문자열이 동일하면  
동일한 객체로 판단할 수 있도록  
equals() 메소드를 오버라이드 함

true로 인식

# Wrapper 클래스

- Wrapper 클래스
  - ✓ 자바의 기본 타입(Primitive Type)을 클래스화한 8개의 클래스
  - ✓ 기본 타입은 값만 저장할 수 있으나 Wrapper 클래스는 기본 타입의 값과 기능을 제공
  - ✓ 기본 타입을 참조 타입(Reference Type)으로 변경할 때 사용
- 종류

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



# Wrapper 클래스의 주요 메소드

- 문자열을 기본 타입으로 변환

```
int a = Integer.parseInt("5");  
long b = Long.parseLong("10");  
double c = Double.parseDouble("1.5");
```

- 기본 타입을 Wrapper 클래스 객체로 변환(생성자를 대체하는 메소드)

```
Integer a1 = Integer.valueOf(5);  
Long b1 = Long.valueOf(10);  
Double c1 = Double.valueOf(1.5);  
Integer a2 = Integer.valueOf("5");  
Long b2 = Long.valueOf("10");  
Double c2 = Double.valueOf("1.5");
```

- Wrapper 클래스 객체를 기본 타입으로 변환

```
int a = a1.intValue();  
long b = b1.longValue();  
double c = c1.doubleValue();
```

# Boxing과 UnBoxing

- 기본 타입과 Wrapper 클래스 타입은 자동(Auto)으로 변환됨
- Boxing : 기본 타입 → Wrapper 클래스 타입
- UnBoxing : Wrapper 클래스 타입 → 기본 타입

```
public class AutoBoxing {  
    public static void main(String[] args) {  
        Integer a = 10; // Auto Boxing  
        int b = a + 5;   // Auto UnBoxing  
        System.out.println(a); // 10  
        System.out.println(b); // 15  
    }  
}
```

# String 클래스

- 패키지 : java.lang
- 하나의 문자열을 표현할 때 사용
- byte[], char[], String, StringBuilder 등 여러 매개변수를 이용해서 생성
- 예시) 문자열 "hello"를 만들기

```
① String str = "hello";

② char[] data = {'h', 'e', 'l', 'l', 'o'};
   String str = new String(data);

③ String str = new String("hello");

④ byte[] bytes = "hello".getBytes();
   String str = new String(bytes);

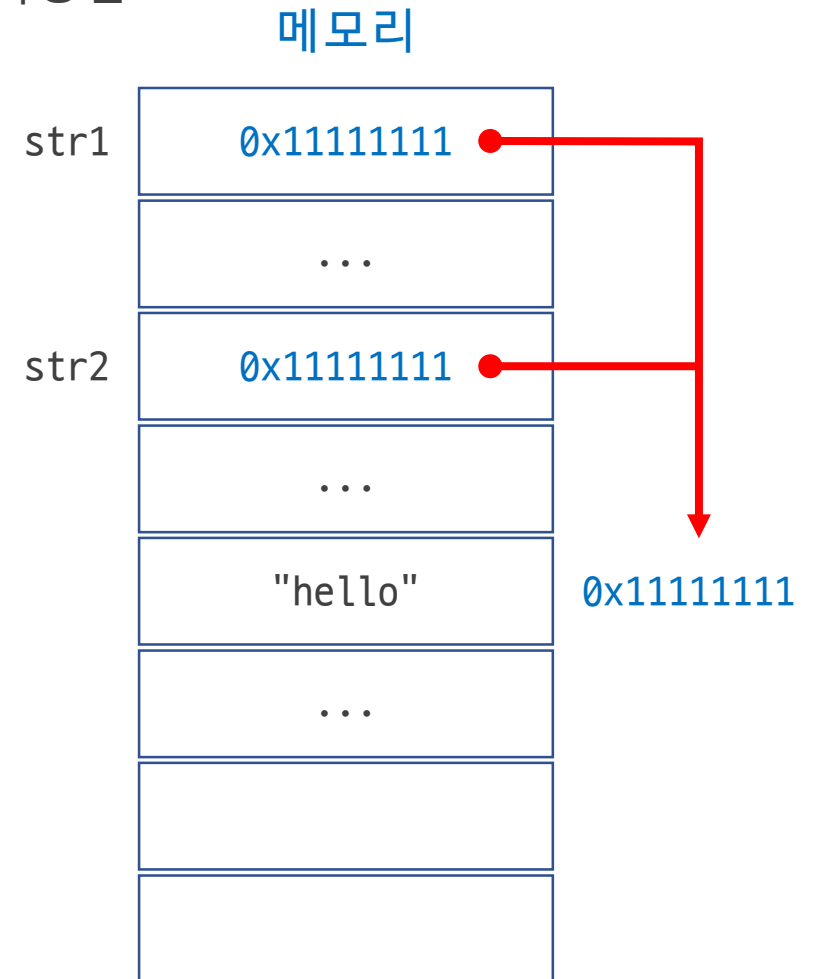
⑤ StringBuilder sb = new StringBuilder();
   sb.append("hello");
   String str = new String(sb);
```

# 문자열 리터럴

- 리터럴로 생성되는 문자열은 JVM이 관리
- 동일한 문자열이 발생되면 JVM이 기존에 생성한 문자열을 재사용함

```
public class StringLiteral {  
    public static void main(String[] args) {  
        String str1 = "hello";  
        String str2 = "hello";  
        System.out.println(str1 == str2);  
    }  
}
```

JVM에 의해서 생성된 문자열  
"hello"를 공유하므로 true 반환

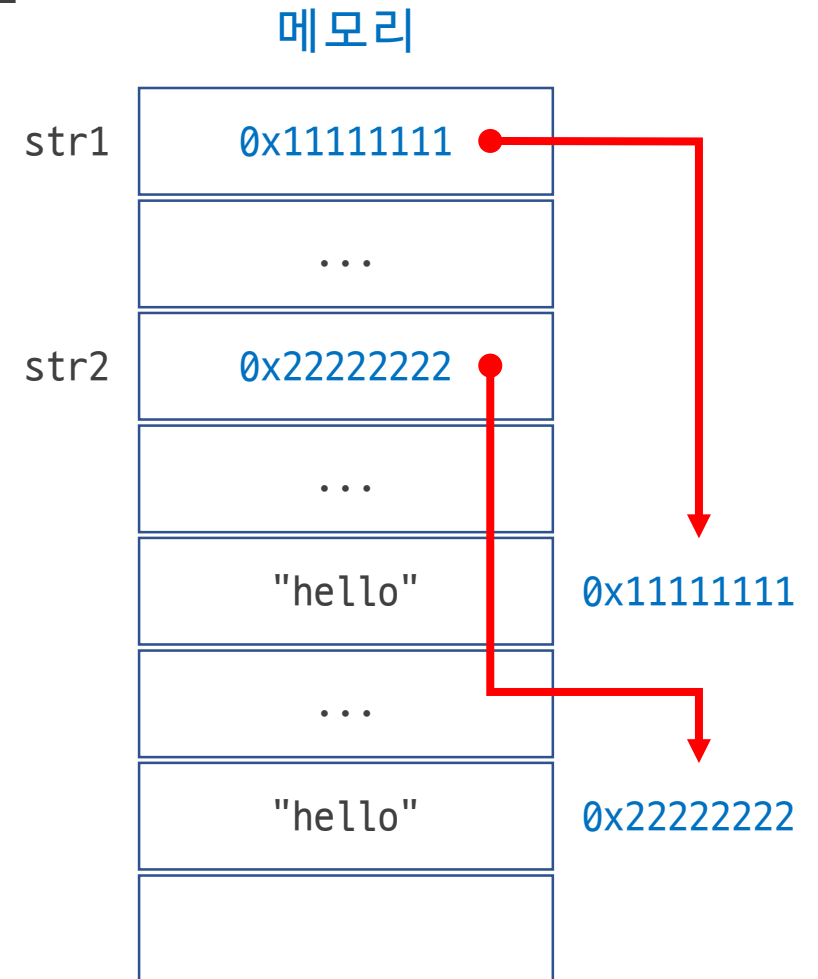


# new String

- 개발자 명령에 의해서 힙(Heap) 영역에 새로운 문자열이 생성됨
- 항상 새로 생성하므로 동일한 문자열도 여러 개 생성될 수 있음

```
public class StringLiteral {  
    public static void main(String[] args) {  
        String str1 = new String("hello");  
        String str2 = new String("hello");  
        System.out.println(str1 == str2);  
    }  
}
```

서로 다른 두 개의 "hello"가  
생성되므로 false 반환



# String 클래스의 주요 메소드

- char charAt(int index)
  - ✓ 현재 문자열의 문자 중 전달된 index에 위치한 문자를 반환

```
String str = "hello";  
char ch = str.charAt(0);  
System.out.println(ch); // h
```

- String substring(int beginIndex)
  - ✓ 전달된 beginIndex의 위치부터 끝까지 문자열을 반환

```
String str = "home made";  
String result = str.substring(5);  
System.out.println(result); // made
```

- String substring(int beginIndex, int endIndex)
  - ✓ 전달된 beginIndex의 위치부터 endIndex 이전까지 문자열을 반환

```
String str = "home made";  
String result = str.substring(0, 4);  
System.out.println(result); // home
```

# String 클래스의 주요 메소드

- boolean equals(Object anObject)
  - ✓ 현재 문자열이 전달된 문자열과 같은 문자열이면 true 반환

```
String str1 = "HELLO";
String st2 = "hello";
if(str1.equals(str2)) {
    System.out.println("같은 문자열입니다.");
} else {
    System.out.println("다른 문자열입니다."); // 다른 문자열입니다.
}
```

- boolean equalsIgnoreCase(String anotherString)
  - ✓ 현재 문자열이 전달된 문자열과 대소문자를 무시하고 비교하여 같은 문자열이면 true 반환

```
String str1 = "HELLO";
String st2 = "hello";
if(str1.equalsIgnoreCase(str2)) {
    System.out.println("같은 문자열입니다."); // 같은 문자열입니다.
} else {
    System.out.println("다른 문자열입니다.");
}
```

# String 클래스의 주요 메소드

- boolean startsWith(String prefix)
  - ✓ 현재 문자열이 전달된 prefix 값으로 시작하면 true 반환

```
String str = "hello";
if(str.startsWith("he")) {
    System.out.println("he로 시작합니다."); // he로 시작합니다.
} else {
    System.out.println("he로 시작하지 않습니다.");
}
```

- boolean endsWith(String suffix)
  - ✓ 현재 문자열이 전달된 suffix 값으로 끝나면 true 반환

```
String str = "apple.jpg";
if(str.endsWith("jpg")) {
    System.out.println("jpg 이미지입니다."); // jpg 이미지입니다.
} else {
    System.out.println("jpg 이미지가 아닙니다.");
}
```



# String 클래스의 주요 메소드

- boolean contains(CharSequence s)
  - ✓ 현재 문자열이 전달된 s 값을 포함하고 있으면 true 반환

```
String str = "hello";  
if(str.contains("e")) {  
    System.out.println("e를 포함합니다."); // e를 포함합니다.  
} else {  
    System.out.println("e를 포함하지 않습니다.");  
}
```

- int length()
  - ✓ 문자열의 길이(글자수)를 반환

```
String str = "hello";  
int length = str.length();  
System.out.println(length); // 5
```

- String concat(String str)
  - ✓ 전달된 문자열 str을 현재 문자열의 마지막에 추가한 문자열을 반환

```
String str = "hello";  
String result = str.concat(" world");  
System.out.println(result); // hello world
```

# String 클래스의 주요 메소드

- `int indexOf(String str)`
  - ✓ 전달된 문자열 `str`이 존재하는 첫 번째 인덱스를 반환

```
String str = "hahaha";  
int index = str.indexOf("ha");  
System.out.println(index); // 0
```

- `int lastIndexOf(String str)`
  - ✓ 전달된 문자열 `str`이 존재하는 마지막 인덱스를 반환

```
String str = "hahaha";  
int index = str.lastIndexOf("ha");  
System.out.println(index); // 4
```

- `String replace(CharSequence target, CharSequence replacement)`
  - ✓ 전달된 문자열 `target`을 모두 찾아서 문자열 `replacement`로 변환한 문자열을 반환

```
String str = "hahaha";  
String result = str.replace("ha", "ho");  
System.out.println(result); // hohoho
```

# String 클래스의 주요 메소드

- String[] split(String regex)
  - ✓ 전달된 정규식 regex을 기준으로 문자열을 분리하여 문자열 배열에 저장한 뒤 배열을 반환

```
String str = "010-1234-5678";  
String[] tokens = str.split("-");  
System.out.println(Arrays.toString(tokens)); // [010, 1234, 5678]
```

- String toLowerCase()
  - ✓ 현재 문자열을 모두 소문자로 변환한 문자열을 반환

```
String str = "HELLO";  
String result = str.toLowerCase();  
System.out.println(result); // hello
```

- String toUpperCase()
  - ✓ 현재 문자열을 모두 대문자로 변환한 문자열을 반환

```
String str = "hello";  
String result = str.toUpperCase();  
System.out.println(result); // HELLO
```

# String 클래스의 주요 메소드

- String trim()
  - ✓ 현재 문자열 앞뒤에 추가된 공백 문자(Space Character)를 모두 제거한 문자열을 반환

```
String str = "  hello world  ";  
String result = str.trim();  
System.out.println(result); // hello world
```

- byte[] getBytes()
  - ✓ 현재 문자열을 시스템의 기본 인코딩 방식을 이용하여 바이트 배열로 변환한 뒤 배열을 반환

```
String str = "hello";  
byte[] bytes = str.getBytes();  
System.out.println(Arrays.toString(bytes)); // [104, 101, 108, 108, 111]
```

- byte[] getBytes(String charsetName)
  - ✓ 현재 문자열을 지정된 charsetName 인코딩 방식을 이용하여 바이트 배열로 변환한 뒤 배열을 반환(예외 처리가 필요함)

```
String str = "hello";  
try {  
    byte[] bytes = str.getBytes("UTF-8");  
    System.out.println(Arrays.toString(bytes)); // [104, 101, 108, 108, 111]  
} catch (Exception e) { }
```

# StringBuilder 클래스

- 패키지 : java.lang
- 가변 길이를 가지는 문자열 저장용 클래스
- 문자열의 길이가 늘어나면 StringBuilder 객체의 크기도 자동으로 늘어남
- String 클래스의 + 연산(연결 연산)은 많이 사용하면 메모리를 비효율적으로 사용해서 성능이 떨어지기 때문에 StringBuilder 클래스의 append() 메소드를 이용해서 성능을 높임

- StringBuilder 객체 생성

```
StringBuilder sb = new StringBuilder();
```

- StringBuilder vs String

```
StringBuilder sb = new StringBuilder();  
sb.append("happy")  
sb.append(" birthday");  
sb.append(" to");  
sb.append(" you");  
String result = sb.toString();
```



```
String result = "";  
result += "happy";  
result += " birthday";  
result += " to";  
result += " you";
```

**StringBuilder 성능이 더 우수함**

# Math 클래스

- 패키지 : java.lang
- 수학 처리용 클래스
- 모든 메소드는 static 타입이므로 클래스이름 Math를 이용해서 호출해야 함
- 주요 메소드

메소드	역할
double abs(double a)	실수 a의 절대값을 double 타입으로 반환
double ceil(double a)	실수 a의 정수로 올린 값을 double 타입으로 반환
double floor(double a)	실수 a의 정수로 내린 값을 double 타입으로 반환
long round(double a)	실수 a의 정수 반올림 값을 long 타입으로 반환
double pow(double a, double b)	실수 a의 b 제곱 값을 double 타입으로 반환
double random()	0.0 이상 1.0 미만의 임의의 난수를 반환

# Math.random()을 이용한 난수 생성

- static double random() 메소드
  - ✓ 0.0 이상 1.0 미만의 임의의 double 값을 반환
  - ✓ 연산을 통해 원하는 범위의 int형 난수를 구할 수 있음

- 주사위(1 ~ 6)

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

6개의 난수 발생

1부터 난수 발생

- 로또(1 ~ 45)

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

45개의 난수 발생

1부터 난수 발생

# Random 클래스

- 패키지 : java.util
- 난수 생성 클래스
- 기본적으로 현재 시간(timestamp)을 seed 값으로 사용하여 난수를 생성함
- 주요 메소드

메소드	역할
double nextDouble()	0.0 이상 1.0 미만의 임의의 실수를 반환
int nextInt()	int 범위 내에서 임의의 정수를 반환
int nextInt(int bound)	0 이상 bound 미만의 임의의 정수를 반환
long nextLong()	long 범위 내에서 임의의 정수를 반환
void setSeed(long seed)	현재 시간 대신 전달된 seed를 사용



# Date 클래스

- 패키지 : java.util
- 날짜 시간 표현 클래스
- java.sql.Date 클래스와 java.sql.Timestamp 클래스의 슈퍼클래스

```
import java.util.Date;

Date now = new Date();
System.out.println(now); // Thu May 05 13:15:30 KST 2022
```

- 패키지 : java.sql
- 날짜 시간 표현 클래스
- JDBC에서 SQL DATE 값을 인식하기 위한 클래스

```
import java.sql.Date;

Date now = new Date(System.currentTimeMillis());
System.out.println(now); // 2022-05-05
```

# SimpleDateFormat 클래스

- 패키지 : java.text
- 날짜 시간에 특정 패턴을 적용시킬 수 있는 클래스
- 주요 패턴

Pattern Letter	의미	Pattern Letter	의미
yy	2자리 년도	a	오전/오후
yyyy	4자리 년도	h	시(1 ~ 12)
M	월(1 ~ 12)	hh	시(01 ~ 12)
MM	월(01 ~ 12)	H	시(0 ~ 23)
MMM	월(1월 ~ 12월)	HH	시(00 ~ 23)
d	일(1 ~ 31)	m	분(0 ~ 59)
dd	일(01 ~ 31)	mm	분(00 ~ 59)
u	요일(1 ~ 7), 1:월 ~ 7:일	s	초(0 ~ 59)
E	요일(월 ~ 일)	ss	초(00 ~ 59)

# SimpleDateFormat 클래스 사용 예시

- 형식 : 2022-05-05 13:15:30

```
import java.util.Date;

Date now = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String result = sdf.format(now);
```

- 형식 : 2022년 05월 05일 목요일

```
import java.util.Date;

Date now = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("yyyy년 MM월 dd일 E요일");
String result = sdf.format(now);
```

- 형식 : 2022/05/05 오후 1:15:30

```
import java.util.Date;

Date now = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd a h:mm:ss");
String result = sdf.format(now);
```

# Calendar 클래스

- 패키지 : java.util
- 날짜 시간 정보 저장 클래스
- 년, 월, 일, 시, 분, 초 등의 정보를 알아낼 수 있음
- 주요 필드

필드	의미
YEAR	년도
MONTH	월(0 ~ 11)
DAY_OF_MONTH	일(1 ~ 31)
DAY_OF_WEEK	요일(1 ~ 7), 1은 일요일, 2는 월요일, ... 7은 토요일
AM_PM	오전/오후(0 ~ 1), 0은 오전, 1은 오후
HOUR	시(0 ~ 11)
HOUR_OF_DAY	시(0 ~ 23)
MINUTE	분(0 ~ 59)
SECOND	초(0 ~ 59)

# Calendar 클래스 사용 예시

- 현재 날짜 표시하기
- 형식 : 2022년 5월 5일 목요일 오전 9시 10분 30초

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

int year = calendar.get(Calendar.YEAR);
int month = calendar.get(Calendar.MONTH) + 1;
int day = calendar.get(Calendar.DAY_OF_MONTH);
String[] week = {"", "일", "월", "화", "수", "목", "금", "토"};
int dayNo = calendar.get(Calendar.DAY_OF_WEEK);

String[] ampm = {"오전 ", "오후 "};
int ap = calendar.get(Calendar.AM_PM);
int hour = calendar.get(Calendar.HOUR);
int minute = calendar.get(Calendar.MINUTE);
int second = calendar.get(Calendar.SECOND);

String now = year + "년 " + month + "월 " + day + "일 " + week[dayNo] + "요일 " +
    ampm[ap] + hour + "시 " + minute + "분 " + second + "초";
System.out.println(now);
```