java.lang 패키지

기본 패키지, import 없이 사용 가능

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

- equals() : 객체 비교

  ex) obj1.equals(obj2)

- **toString()**

: 클래스명@16진수 해시코드 문자 정보 반환

  ex) obj.toString() → java.lang.Object@de6ced

- **clone()**

: 객체 복제, 복제할 클래스에 Cloneable 인터페이스를 구현해야 함

```
public class Member implements Cloneable {
    ...
    public Member getMember() {
        Member cloned = null;
        try {
            cloned = (Member) clone();  // s 타입과 동일하게 캐스팅 必, 얕은 복제
            // cloned = (Member) super.clone();  // 깊은 복제
        } catch (CloneNotSupportedException e) { }
        return cloned;
    }
}
```

얕은 복제	깊은 복제

번지만 복제되므로 원본 객체의 필드와 복제 객체의 필드는 같은 객체 참조	번지와 함께 참조하고 있는 객체도 복제
복사된 객체를 수정하면 원본 객체도 수정됨	복사된 객체를 수정해도 원본 객체는 그대로임
clone()	super.clone()

- **finalize()**

: 객체 소멸자

참조하지 않는 객체나 배열이 소멸될 때 실행됨, 오버라이드하여 사용 가능

실행/생성 순서대로 소멸시키지 않음

전부 소멸시키지 않고 메모리 상태에 따라 일부만 소멸시킴

```
// 소멸자 예시
class Employee {
   public int eno;
   public Employee(int eno) {
      this.eno = eno;
      sout("Employee 등록");
   }
   public void finalize() {  // 객체 소멸자 오버라이드
      sout("Employee가 메모리에서 제거됨");
   }
}
```

운영체제의 일부 기능(프로그램 종료, 키보드 입력, 모니터 출력, 메모리 정리, 현재 시간·시스템 프로퍼티·환경변수 읽기) 이용 가능

System 클래스의 모든 필드와 메소드는 정적(static)으로 구성

- **exit()** : 프로세스 강제 종료

ex) System.exit(0); // 정상 종료

- **gc()**

: 쓰레기 수집기를 실행하여 사용하지 않는 객체를 자동 제거

직접 실행 X → JVM에게 실행시켜달라고 요청하는 메소드

ex) System.gc();

- **currentTimeMills()** : 현재 시간과 1970-01-01 0시 0분의 차이를 long형으로 반환

- **nanoTime()** : 프로그램의 실행 시간을 측정할 때 사용

```
// 시간 재기 예시
main() {
   long start = System.nanoTime();
   ... // 프로그램 실행
   long end = System.nanoTime();
   sout("실행 시간: " + (end - start) + "나노초");
}
```

- **getProperty()** : 시스템 프로퍼티 읽기

ex) System.getProperty("os.name");

→ java.version, java.home, os.name, file.separator, user.name, user.home, user.dir 등

- **getProperties()** : 시스템 프로퍼티의 키, 값 쌍을 맵 객체로 읽기

```
Properties props = System.getProperties();
Set keys = props.keySet();
for (Object objkey : keys) {
   String key = (String) objKey;
   String value = System.getProperty(key);
   sout(key + "," + value);
}
```

- **getenv()** : 환경변수 읽기

ex) System.getenv("JAVA_HOME"); → 자바 환경변수 문자열로 반환

- **getClass()** : 해당 객체의 클래스 리턴

ex) student.getClass()

- **getDeclaredConstructors(), getDeclaredFields(), getDeclaredMethods()**

: 클래스의 생성자, 필드, 메소드 정보를 배열로 리턴(상속된 멤버 제외)

- **getFields(), getMethods()**

: 상속된 항목을 포함한 멤버 리턴(public 멤버만 리턴)

- **newInstance()**

: new 연산자를 사용하지 않고 동적 객체 생성

기본 생성자가 반드시 존재해야 함

InstantiationException 혹은 IllegalAccessException 예외가 발생할 수 있으므로 예외처리 필요

→ InstantiationException : 해당 클래스가 추상 클래스일 때 발생

→ IllegalAccessException : 클래스나 생성자가 접근 제한자로 인해 접근할 수 없는 경우 발생

```
// 동적 객체 생성 예시
try {
    Class cls = Class.forName("클래스 이름");
    Action act = (Action) cls.newInstance();  // newInstance()의 결과가 Object이므로 메소드를 사용하려면 타입 변환 필요
} carch (Exception e) { ... }
```

- String 생성자

: 지정한 문자셋으로 디코딩하거나 매개변수 배열의 일부만 Stirng 객체로 생성 가능

```
main() {
    byte[] bytes = { 72, 101, 108, 108, 111, 32, 74, 97, 118, 97 };
    String str1 = new String(bytes);  // 바이트 배열을 문자열로 변환
    Srting str2 = new String(bytes, 6, 4);  // 바이트 배열의 인덱스 6부터 4개만 문자열로 변환
}
```

- **charAt()** : 주어진 인덱스의 문자 리턴

  ex) str.charAt(3);

- equals() : 문자열 값 비교

- **getBytes()** : 바이트 배열로 변환

  ex) str.getBytes();

- **indexOf()** : 주어진 문자열의 위치(인덱스) 리턴

  ex) str.indexOf("자바");

- length() : 문자열 길이 리턴

- replace() : 문자열 교체

- **substring()**

: 문자열 자르기

매개변수가 1개일 때는 해당 인덱스부터 끝까지 리턴

- **toLowerCase(), toUpperCase()** : 소문자/대문자 변환

- trim() : 앞 뒤 공백 제거

- **valueOf()** : 숫자나 논리 변수를 문자열로 변환

ex) String.valueOf(true);

- split()

: 문자열을 분리하여 문자열 배열로 리턴

두 번째 매개변수 지정시 해당 개수만큼만 배열로 리턴

- StringTokenizer 클래스

: 문자열이 한 종류의 구분자료 연결된 경우 사용하여 분리

countTokens(), hasMoreTokens(), nextToken()

```
// 토큰 분리 예제
StringTokenizer st = new StringTokenizer("홍길동/이수홍/박연수", "/");
int count = st.countTokens();  // 토큰 개수
while(st.hasMoreTokens()) {  // 남은 토큰 존재 여부
   sout(st.nextToken());  // 토큰 pop
}
```

- **문자열 연결 연산자(+)를 많이 쓰면 새로운 String 객체가 계속 생성돼서 프로그램 부하 발생**

- StringBuffer 클래스

: 문자열로 변환할 때는 뒤에 toString()

```
// 스트링 버퍼 선언 예제
StringBuffer sb = new StringBuffer();
StringBuffer sb = new StringBuffer(16);
StringBuffer sb = new StringBuffer("안녕요");
```

메소드	설명
append(문자열)	문자열 끝에 주어진 매개값을 추가
insert(숫자, 문자열)	문자열 중간에 주어진 매개값을 추가
delete(시작 인덱스, 종료 인덱스)	문자열의 일부분을 삭제

deleteChatAt(인덱스)	문자열에서 주어진 인덱스의 문자를 삭제
replace(시작 인덱스, 종료 인덱스, 문자열)	문자열의 일부분을 다른 문자열로 대치
StringBuilder reverse()	문자열의 순서를 뒤바꿈
setCharAt(인덱스, 문자)	문자열에서 주어진 인덱스의 문자를 다른 문자로 대치

- Pattern 클래스

: 정규 표현식으로 문자열을 검증하기 위해 matches() 메소드 사용

ex) **boolean result = Pattern.matches(정규식, 검증할 문자열);**

- 정규 표현식 표(나중에 따로 정리)

기호	설명
[]	한 개의 문자
[abc]	a, b, c 중에 한 개의 문자
[^abc]	a, b, c 이외의 한 개의 문자
[a-zA-Z]	a~z, A~Z 중 한 개의 문자
₩d	한 개의 숫자, [0-9]와 동일
₩s	공백
₩w	한 개의 알파벳 똔느 한 개의 숫자 [a-zA-Z_0-9]와 동일
?	없거나 한 개
*	없거나 한 개 이상
+	한 개 이상
{n}	n개
{n,}	최소한 n개
{n, m}	n에서 m까지
()	그룹핑

- **copyOf(배열, 복사할 길이), copyOfRange(배열, 시작 인덱스, 끝 인덱스)**

: 배열 복사(얕은 복제)

- equals() : 1차원 배열 항목 비교

- deepEquals() : 2차원 배열의 항목 비교

```
char[][] arr2 = Arrays.copyOf(arr1, arr1.length);  // 얕은 복사 예제
// 깊은 복사 예제
char[][] arr2= Arrays.copyOf(arr1, arr1.length);
```

```
arr2[0] = Arrays.copyOf(arr1[0], arr1[0].length);
arr2[1] = Arrays.copyOf(arr1[1], arr1[1].length);
```

- sort() : 오름차순 정렬

ex) Arrays.sort(arr);

- **binarySearch(배열, 찾을 값)** : 이진탐색

포장되어 있는 기본 타입 값은 외부에서 변경할 수 없음

char → Character / int → Integer / 그 외 기본타입의 첫 문자 대문자로 사용

```
// 박싱(포장) 예시
Byte obj = new Byte(10);
Character obj2 = new Character('A');
Integer obj3 = new Integer(1000);
```

- 자동 박싱/언박싱

```
// 자동 박싱 예제
Integer obj = 100;

// 자동 언박싱 예제
int value1 = obj;
int value2 = obj + 100;
```

- 언박싱

: **"기본타입명 + Value()"** 메소드 호출

```
Integer obj = new Integer(100);
int value = obj.intValue();
```

- 문자열을 기본 타입 값으로 변환

: **"박싱할 때 기본타입명.parse + 기본타입명()"** 메소드 사용

```
// 문자열 → 기본타입 값 예시
byte num = Byte.parseByte("10");
int num2 = Integer.parseInt("20");
```

- 포장 값 비교

: ==, != 연산자 사용 불가, equals() 권장

- abs() : 절대값

- ceil() / floor() : 올림, 내림

- round() : 반올림

- max() / min() : 최대, 최소

- random() : 랜덤, 형변환 주의

```
(int) Math.random() * 10 + 1; // 1 이상 10 이하의 정수
```

- **rint()** : 반올림 실수값

java.util 패키지

클래스	용도
Arrays	배열을 조작(비교, 복사, 정렬, 찾기)할 때 사용
Calendar	운영체제의 날짜와 시간을 얻을 때 사용
Date	날짜와 시간 정보를 저장하는 클래스
Objects	객체 배겨, 널 여부 등을 조사할 때 사용
StringTokenizer	특정 문자로 구분된 문자열을 뽑아낼 때 사용
Random	난수를 얻을 때 사용

- compare()

: 비교 메소드, 두 객체를 비교자로 비교하여 -1(>), 0(=), 1(<) 리턴하도록 메소드를 오버라이드 한 후 사용해야 함

```
// 변수 비교
Integer.compare(1, 2);
// 객체 비교
static class StudentComparator implements Comparator<Student> {  // 1. 비교자 선언
```

```
    @Override
    public int compare(Student o1, Student o2) {
        return Integer.compare(o1.sno, o2.sno);
    }
}
// 2. 비교자를 사용하여 객체 비교
main() {
    Student s1 = new Student(1);
    Student s2 = new Student(1);
    sout(Objects.compare(s1, s2, new StudentComparator()));
}
```

- **equals()**

: 비교하는 대상이 둘 다 null일 경우엔 true 리턴

- **deepEquals()**

: Arrays.deepEquals(a, b)와 동일, 객체의 항목 값이 모두 같으면 true 리턴

- **hash()**

: 매개변수로 주어진 값들을 이용하여 해시코드 생성

동일한 필드값을 갖는 경우엔 동일한 해시코드 리턴

ex) Objects.hash(sno, name);

- **hashCode()**

: 매개값으로 주어진 객체의 해시코드 리턴 but 매개값이 null이면 0 리턴

ex) Objects.hashCode(s2);

- isNull() : 매개값이 null일 경우 true

- nonNull() : 매개값이 null이 아닐 경우 true

- **requireNonNull()**

: 매개값이 null일 경우 첫 번째 매개변수 리턴, null이면 NullPointerException 발생

두번째 매개변수는 NullPointerException의 예외 메시지

- **toString()**

: null이 아니면 첫 번째 매개변수 리턴

두번째 매개변수는 null일 때의 default값

ex) Objects.toString(str1, "이름이 없습니다");

랜덤 객체 생성 후 그 객체에서 원하는 타입의 값을 받음

ex) **nextBoolean(), nextDouble, nextInt(), nextInt(n)**

```
Random random = new Random(3);
for(int i = 0; i < 6; i++) {
   selectNumber[i] = random.nextInt(45) + 1;
   System.out.println(selectNumber[i] + " ");
}
```

- Math 클래스의 Random 메소드와의 차이점

1. int 외에 long, double, float, boolean 난수도 얻을 수 있음

2. 시드값 설정 가능

   : 서로 같은 시드값을 주면 같은 결과가 나옴

1) Date 클래스

```
Date now = new Date();  // date 객체 선언
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
sout(sdf.format(now));
```

2) Calendar 클래스

: 추상 클래스, new 연산자 사용 불가

```
Calendar now = Calendar.getInstance();  // calendar 객체 선언

int year = now.get(Calendar.YEAR);  // 년도
int month = now.get(Calendar.MONTH) + 1;  // 월
int day = now.get(Calendar.DAY_OF_MONTH);  // 일
int week = now.get(Calendar.DAY_OF_WEEK);  // 요일(1~7, 일~토)
int ampm = now.get(Calendar.AM_PM);  // 오전(0) or 오후(1)
int hour = now.get(Calendar.HOUR);  // 시
int minute = now.get(Calendar.MINUTE);  // 분
int second = now.get(Calendar.SECOND);  // 초
```

- 숫자 형식 클래스(DecimalFormat)

```
// 숫자 포맷 예시
DecimalFormat df = new DecimalFormat("#,###.0");
String res = df.format(1234567.89);  // 출력: 1,234,567.9
```

기호	의미
0	10진수(빈자리는 0으로 채움)
#	10진수(빈자리는 채우지 않음)
.	소수점
-	음수 기호
,	단위 구분
;	패턴 구분자
%	100을 곱한 후에 % 문자 붙임
₩u00A4	통화 기호

- 날짜 형식 클래스(SimpleDateFormat)

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
String date = sdf.format(new Date());  // 출력: 2021-06-25 20:19:44
```

- 문자열 형식 클래스(MessageFormat)

```
// String message = "ID: " + id + "₩n이름: " + name + "₩n전화번호: " + tel;
// 위와 같음
String message = "ID: {0}₩n이름: {1}₩n전화번호: {2}";
String res = MessageFormat.format(message, id, name, tel);  // 포맷 스트링, 매개변수1, 매개변수2, ... 순
```

java.time 패키지

```
// LocalDate : 날짜 정보만 저장
LocalDate curDate = LocalDate.now();  // 현재 날짜 정보를 저장한 LocalDate 객체 리턴
LocalDate targetDate = LocalDate.of(2021, 6, 26);  // 입력한 날짜 정보를 저장한 LocalDate 객체 리턴
```

```
// LocalTime : 시간 정보만 저장
LocalTime curTime = LocalTime.now();  // 현재 시간 정보를 저장한 LocalTime 객체 리턴
LocalTime targetTime = LocalTime.of(20, 57, 26);  // 입력한 시간 정보를 저장한 LocalTime객체 리턴

// LocalDateTime : LocalDate와 LocalTime을 결합한 클래스
LocalDateTime curDateTime = LocalDateTime.now();
LocalDateTime targetDateTime = LocalDateTime.of(2021, 6, 26, 20, 57, 26);

// ZonedDateTime : 타임존의 날짜와 시간을 저장
ZonedDateTime utcDateTime = ZonedDateTime.now(ZoneId.of("UTC"));
ZonedDateTime seoulDateTime = ZonedDateTime.now(ZoneId.of("Asia/Seoul"));

// Instant : 특정 시점의 타임스탬프로 사용
Instant instant1 = Instant.now();
Instant instant2 = Instant.now();
if (instant1.isBefore(instant2)) {
    System.out.println("instant1 > instant2");
} else if (instant1.isAfter(instant2)) {
    System.out.println("instant1 < instant2");
} else System.out.println("instant1 == instant2");
System.out.println("차이: " + instant1.until(instant2, ChronoUnit.SECONDS));
```

- getYear() : 년도

- getMonth() : 월(문자)

- getMonthValue() : 월(숫자)

- getDayOfMonth() : 일(숫자)

- getDayOfWeek() : 요일(문자)

- isLeapYear() : 윤년 여부

- getHour(), getMinute(), getSecond(), getNano()

- withYear() : 년 변경

- withMonth() : 월 변경

- withDayOfMonth() : 일 변경

- withDayOfYear() : 년 기준 일 변경

- with() : 상대적인 변경, TemporalAdjusters 사용

- withHour() : 시 변경

- withMinute() : 분 변경

- withSecond() : 초 변경

- withNano() : 나노초 변경

- isBefore(), isAfter() : 이전/이후 날짜/시간인지 비교

- isEqual() : 동일 날짜인지 비교

- until() : 시간 차이 비교

- between() : 두 날짜/시간의 차이 리턴

  → ChronoUnit 사용

```
// ChronoUnit 예시
long remainYear = startDateTime.until(endDateTime, ChronoUnit.YEARS);
long remainMonth = startDateTime.until(endDateTime, ChronoUnit.MONTHS);
long remainDay = startDateTime.until(endDateTime, ChronoUnit.DAYS);

remainYear = ChronoUnit.YEARS.between(startDateTime, endDateTime);
remainMonth = ChronoUnit.MONTHS.between(startDateTime, endDateTime);
remainDay = ChronoUnit.DAYS.between(startDateTime, endDateTime);
```

: 문자열을 날짜와 시간으로 변환, parse()

```
// 파싱 예시
LocalDate localDate = LocalDate.parse("2024-01-28");  // 문자열 파싱

DateTimeFormatter formatter = DateTimeFormatter.ofParrern("yyyy/MM/dd");
LocalDate localDate2 = LocalDate.parse("2024/01/28", formatter);
```

: 날짜와 시간을 원하는 문자열 포맷으로 변환, format()

```
// 포맷팅 예시
LocalDateTime now = LocalDateTime.now();
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy년 M월 d일 a h시 m분");
String nowStr = now.format(formatter);
```