Chapter11. 기본 API 클래스

11.1 자바 API 도큐먼트

- **API(라이브러리)** : 프로그램 개발에 자주 사용되는 클래스 및 인터페이스의 모음을 말합니다.
- **API 도큐먼트** : 쉽게 API를 찾아 이용할 수 있도록 문서화한 것.

  <오라클에서 제공하는 API 도큐먼트 URL>

  ```
  http://docs.oracle.com/javase/8/docs/api/
  ```

  - **좌측 상단 프래임** : 패키지 전체 목록
  - **좌측 하단 프레임** : 패키지에 속하는 클래스와 인터페이스 목록
  - **중앙 프레임** : 선택된 클래스나 인터페이스에 대한 상세 설명
    - **상단 부분** : 클래스가 포함된 패키지 정보, 상속 정보, 인터페이스 구현 정보
    - **중앙 부분** : 클래스의 설명과 사용 방법 설명
    - **하단 부분** : 필드, 생성자, 메소드의 목록

  Field Summary : 필드에 대한 내용

  Field Summary

  Fields

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

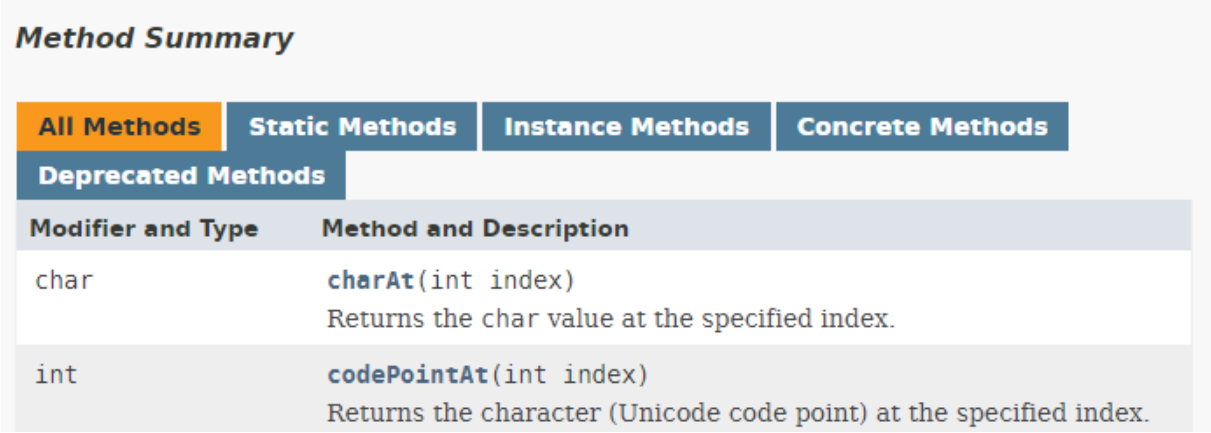
  Constructor Summary : 생성자에 대한 설명

  Constructor Summary

  Constructors

Constructor and Description
String() Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platform's

Method Summary : 메소드에 대한 정보



Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.

11.2 java.lang과 java.util 패키지

11.2.1 java.lang 패키지

: 자바 프로그램의 기본적인 클래스를 담고 있는 패키지이다. 그러므로 import 없이 사용이 가능하다.

클래스	용도
Object	- 자바 클래스의 최상위 클래스로 사용
System	- 키보드로부터 데이터를 입력받을 때 사용 - 모니터로 출력하기 위해 사용 - 자바 가상 기계를 종료시킬 때 사용 - 쓰레기 수집기를 실행 요청할 때 사용
Class	- 클래스를 메모리로 로딩할 때 사용
String	- 문자열 저장
StringBuffer StringBuilder	- 문자열을 저장하고 내부 문자열을 조작할 때 사용
Math	- 수학 함수 사용
Wrapper	- 기본 타입의 데이터를 갖는 객체를 만들 때 사용 - 문자열을 기본 타입으로 변환할 때 사용 - 입력값 검사에 사용

11.2.2 java.util 패키지

: 컬렉션 클래스들이 대부분 차지하고 있다. 컬렉션 클래스란 여러 유용한 알고리즘을 구현한 메소드들을 제공한다.

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 비교, 널(null) 여부 등을 조사할 때 사용
String Tokenizer	- 특정 문자로 구분된 문자열을 뽑아낼 때 사용
Random	- 난수를 얻을 때 사용

11.3 Object 클래스

: 클래스를 선언할 때 상속을 하지 않으면 암시적으로 java.lang.Object 클래스를 상속하게 된다. Object는 자바의 최상위 부모 클래스에 해당한다.

11.3.1 객체 비교(equals())

: 두 객체가 동일한 객체라면 true를 리턴하고 그렇지 않으면 false를 리턴한다.

- **예제(equals 메소드 오버라이딩)**

  Member.java

```
package object_class.equals_method;

public class Member {
    public String id;

    public Member(String id) {
        this.id = id;
    }

    // equals 오버라이딩
    @Override
    public boolean equals(Object obj) {
        // 매개값이 Member 타입인지 확인
        if(obj instanceof Member) {
            // 매개값을 Member타입으로 강제 타입 변환하고
            // id 필드값이 동일한지 비교
            Member member = (Member) obj;
            if(id.equals(member.id)) {
                return true;
            }
        }
        return false;
    }
}
```

MemberExample.java

```
package object_class.equals_method;

public class MemberExample {
    public static void main(String[] args) {
        Member obj1 = new Member("blue");
        Member obj2 = new Member("blue");
        Member obj3 = new Member("red");

        // obj1 과 obj2 둘다 Member 타입이고
        // id 값이 일치
        if(obj1.equals(obj2)) {
            System.out.println("obj1과 obj2는 동듭합니다.");
        } else {
            System.out.println("obj1과 obj2는 동등하지 않습니다.");
        }

        // obj1 과 obj2 둘다 Member 타입이지만
        // id 값이 불일치
        if(obj1.equals(obj3)) {
            System.out.println("obj1과 obj3은 동등합니다.");
        } else {
            System.out.println("obj1과 obj3은 동등하지 않습니다.");
        }
    }
}
```

실행 결과

```
obj1과 obj2는 동듭합니다.
obj1과 obj3은 동등하지 않습니다.
```

11.3.2 객체 해시코드(hashCode())

- 객체 해시코드란 객체를 식별할 하나의 정수값이다.
- **hashCode() 메소드**

  : 객체의 메모리 번지를 이용해서 해시코드를 만들어 리턴시킨다.

- **예제**

  Key.java(**hashCode() 메소드 재정의**)

```
package object_class.hashCode_method;

public class Key {
    public int number;

    public Key(int number) {
```

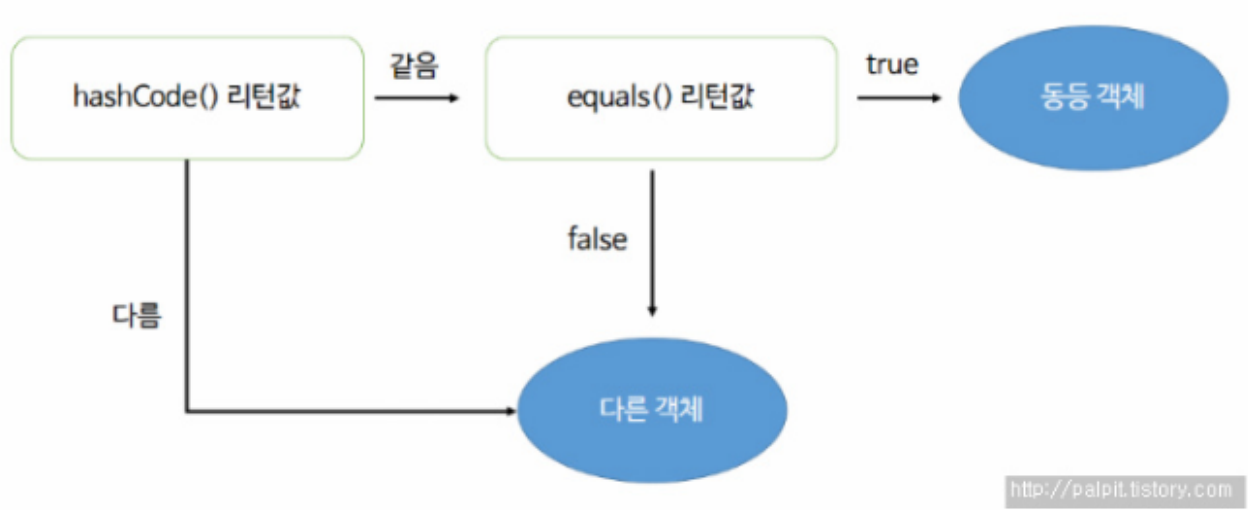
```
        this.number = number;
    }

    // equals 오버라이딩
    @Override
    public boolean equals(Object obj) {
        // 매개값이 Key 타입인지 확인
        if(obj instanceof Key) {
            // Key 타입으로 강제 타입 변환
            Key compareKey = (Key) obj;

            // number 필드 값이 동일한지 비교
            if(this.number == compareKey.number) {
                return true;
            }
        }
        return false;
    }

    // hashCode 오버라이딩
    // 논리적 동등 객체일 경우 동일한 해시코드가
    // 리턴되도록 하기 위해서
    @Override
    public int hashCode() {
        return number;
    }
}
```

KeyExample.java(**HashMap<>을 이용해 키 값을 식별키로 사용**)

```
package object_class.hashCode_method;

import java.util.HashMap;

public class KeyExample {
    public static void main(String[] args) {
        // Key 객체를 식별키로 사용해서 String 값을 저장하는 HashMap 객체 생성
        HashMap<Key, String> hashMap = new HashMap<Key, String>();

        // 식별키 값이 1 로 "홍길동" 저장
        hashMap.put(new Key(1), "홍길동");

        // 식별키 값이 1인 객체의 "홍길동"을 읽어온다
        String value = hashMap.get(new Key(1));
        System.out.println(value);
    }
}
```

실행 결과

```
홍길동
```



> **Key 값으로 Value 값 가져올 때**
>
> hashMap.get()을 실행시키면 hashCode()를 실행시켜 Key 값을 가져오고 equals()를 실행시켜 동일한 객체의 동일한 키 값을 갖는 객체를 찾아서 Value 값을 리턴해준다.

11.3.3 객체 문자 정보(toString())

: 객체의 문자 정보를 리턴한다.

- **예제(객체의 문자 정보 및 현재 날자와 시간 정보)**

  ToStringExample.java

```
package object_class.toString_method;

import java.util.Date;

public class ToStringExample {
    public static void main(String[] args) {
        Object obj1 = new Object();     // Object 객체 생성
        Date obj2 = new Date();         // Date 객체 생성

        System.out.println(obj1.toString());    // 객체 정보 출력
        System.out.println(obj2.toString());    // 날짜 정보 출력
    }
}
```

  실행 결과

```
java.lang.Object@21a06946
Thu Jan 17 13:32:55 KST 2019
```

- **예제(제작회사와 운영체제 리턴)**

  SmartPhone.java

```
package object_class.toString_method;

public class SmartPhone {
    private String company;
    private String os;

    public SmartPhone(String company, String os) {
        this.company = company;
        this.os = os;
    }

    // 객체 문자 정보를 전달해주는 toString 을 오버라이딩해서
    // 회사와 운영체제를 반환한다.
    @Override
    public String toString() {
        return company + ", " + os;
    }
}
```

SmartPhoneExample.java

```
package object_class.toString_method;

public class SmartPhoneExample {
    public static void main(String[] args) {
        SmartPhone myPhone = new SmartPhone("구글", "안드로이드");

        String strObj = myPhone.toString();

        System.out.println(strObj);     // toString 으로 가져온 정보를 출력

        // myPhone 을 출력시켜 myPhone.toString()을 자동 호출 시킨다.
        System.out.println(myPhone);
    }
}
```

실행 결과

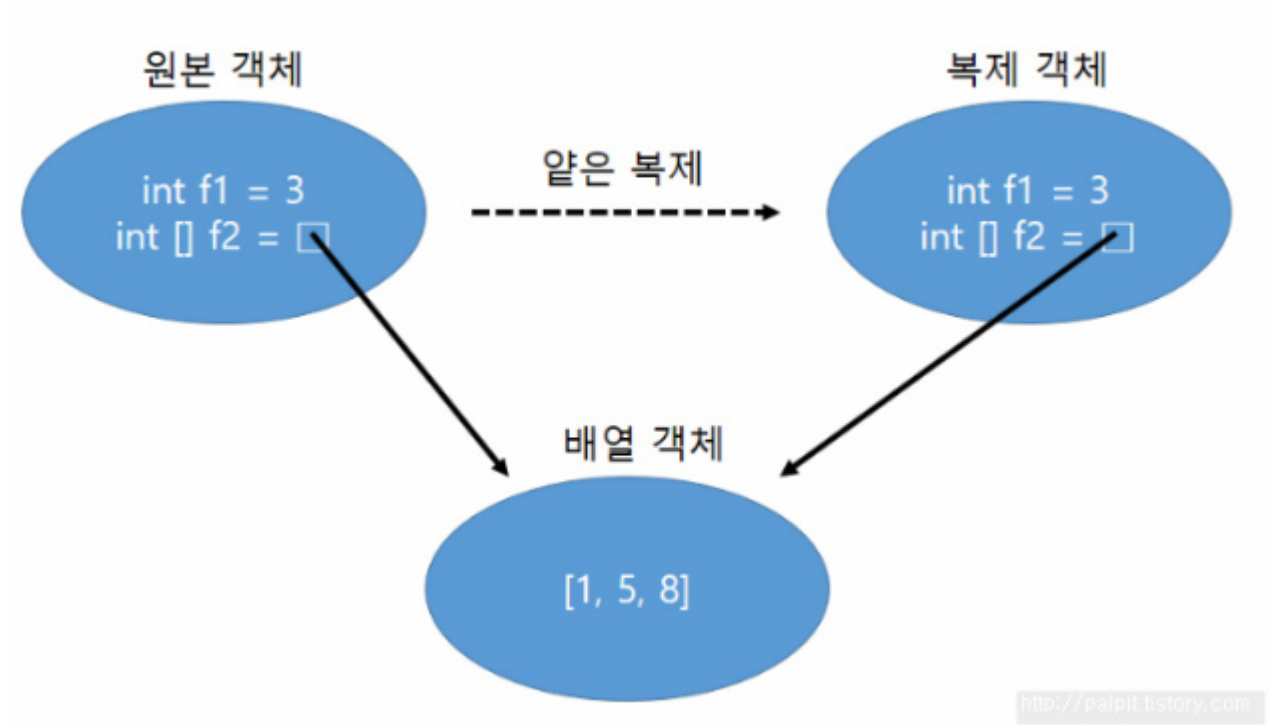
```
구글, 안드로이드
구글, 안드로이드
```

11.3.4 객체 복제(clone())

: 원본 객체의 필드값과 동일한 값을 가지는 새로운 객체를 생성하는 것이다. 객체를 복제하는 이유는 원본 객체를 안전하게 보호하기 위해서이다.

얕은 복제(thin clone)

: 단순히 필드값을 복사해서 객체를 복제하는 것이다.



> 필드가 기본 타입일 경우 값 복사가 일어나고, 필드가 참조 타입일 경우에는 객체의 번지가 복사된다.

- clone() 메소드로 객체를 복제하려면 **원본 객체는 반드시 java.lang.Cloneable 인터페이스를 구현**하고 있어야한다. 이유는 복제를 허용한다는 의도적인 표시를 하기 위해서이다.

- **예제**

  Member.java(**복제할 수 있는 클래스 선언**)

```
package object_class.clone_method;

// Member라는 cloneable 구현 클래스를 선언한다.
public class Member implements Cloneable{
    // Member의 멤버 선언
    public String id;
    public String name;
    public String password;
    public int age;
    public boolean adult;

    // Member 생산자 선언
    public Member(String id, String name, String password,
                  int age, boolean adult) {
        this.id = id;
        this.name = name;
        this.password = password;
        this.age = age;
        this.adult = adult;
```

```
    }

    // 얕은 복제를 위한 메소드 선언
    public Member getMember() {
        // Member 타입 변수 선언
        Member cloned = null;

        // 예외 처리를 위한 try-catch 선언
        try {
            // clone() 메소드의 리턴 타입은 Object 이므로
            // Member 타입으로 강제 타입 변환을 한다.
            cloned = (Member) clone();
        } catch (CloneNotSupportedException e) {
            // 예외가 발생
        }
        return cloned;
    }
}
```

MemberExample.java

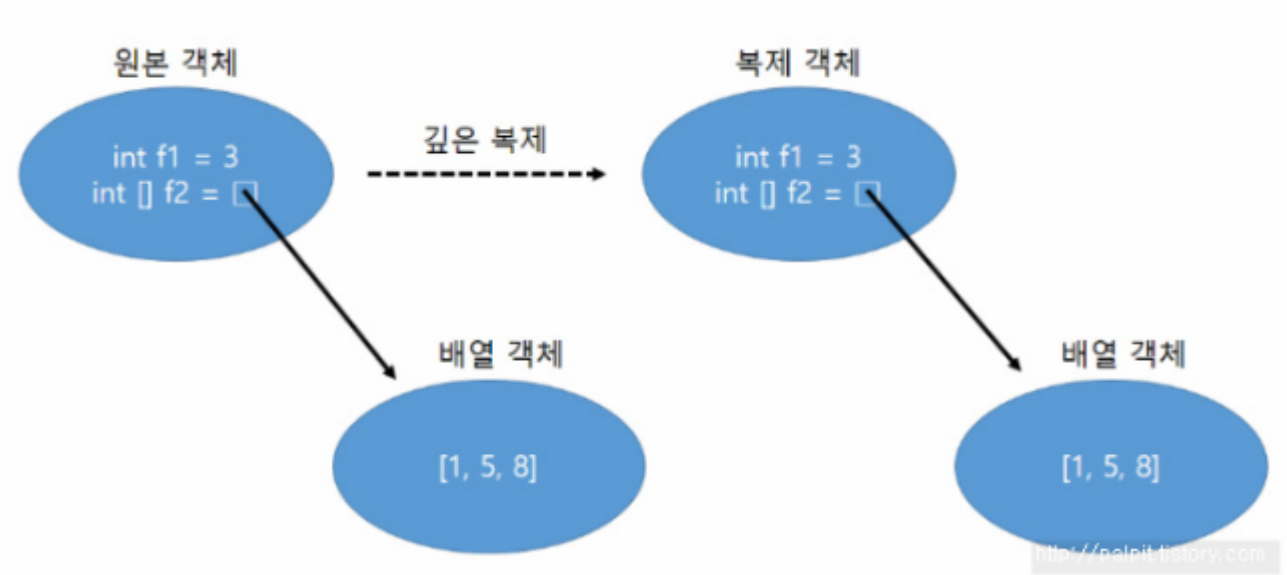
```
package object_class.clone_method;

public class MemberExample {
    public static void main(String[] args) {
        // 원본 Member 객체 생성
        Member original = new Member("blue", "홍길동",
                "12345", 25, true);

        // 복제 객체 생성 후 패스워드 변경
        Member cloned = original.getMember();
        cloned.password = "67890";

        System.out.println("[복제 객체의 필드값]");
        System.out.println("id: " + cloned.id);
        System.out.println("name: " + cloned.name);
        System.out.println("password: " + cloned.password);
        System.out.println("age: " + cloned.age);
        System.out.println("adult: " + cloned.adult);

        System.out.println();

        System.out.println("[원본 객체의 필드값]");
        System.out.println("id: " + original.id);
        System.out.println("name: " + original.name);
        System.out.println("password: " + original.password);
        System.out.println("age: " + original.age);
        System.out.println("adult: " + original.adult);
    }
}
```

실행 결과

```
[복제 객체의 필드값]
id: blue
name: 홍길동
password: 67890
age: 25
adult: true

[원본 객체의 필드값]
id: blue
name: 홍길동
password: 12345
age: 25
adult: true
```

깊은 복제(deep clone)

: 기본 타입의 필드 값은 물론이고 참조하고 있는 객체도 복제하는 것이다.



> 얕은 복제와 다르게 깊은 복제는 참조 객체의 번지 값을 복사하는것이 아니라 참조 객체 자체를 복제해준다.

- 예제

  Car.java

```
package object_class.clone_method.deep_clone;

public class Car {
    public String model;

    public Car(String model) {
        this.model = model;
    }
}
```

Member.java(**clone()을 재정의해서 깊은 복제로 변경**)

```
package object_class.clone_method.deep_clone;

import java.util.Arrays;

public class Member implements Cloneable{
    // 기본 타입 필드
    public String name;
    public int age;

    // 참조 타입 필드(깊은 복제 대상)
    public int[] scores;
    public Car car;

    // 생산자
    public Member(String name, int age, int[] scores, Car car) {
        this.name = name;
        this.age = age;
        this.scores = scores;
        this.car = car;
    }

    // 깊은 복제를 위한 clone 메소드 오버라이딩
    @Override
    protected Object clone() throws CloneNotSupportedException {
        // 먼저 얕은 복사를 해서 name, age 를 복제한다.
        Member cloned = (Member) super.clone();

        // scores 를 깊은 복제한다. 복제할 배열과 길이를 파라미터로 입력
        cloned.scores = Arrays.copyOf(this.scores, this.scores.length);

        // car 도 깊은 복제한다. 자신의 Car 객체의 변수를 파라미터로 해서
        // Car 객체를 생성한뒤 cloned 에 저장한다.
        cloned.car = new Car(this.car.model);
        return cloned;
    }

    public Member getMember() {
        Member cloned = null;
        try {
            cloned = (Member) clone();
```

```
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return cloned;
    }
}
```

MemberExample.java(**깊은 복제 후 복제본 변경은 원본에 영향을 미치지 않는다.**)

```
package object_class.clone_method.deep_clone;

public class MemberExample {
    public static void main(String[] args) {
        // Member 객체를 하나 생성한다.
        Member original = new Member("홍길동", 25, new int[] {90,90},
                new Car("소나타"));

        // 원본 객체를 복사 후 값을 변경한다.
        Member cloned = original.getMember();
        cloned.scores[0] = 100;
        cloned.car.model = "그랜저";

                // 복제 객체를 출력시킨다.
        System.out.println("[복제 객체의 필드값]");
        System.out.println("name: " + cloned.name);
        System.out.println("age: " + cloned.age);
        System.out.print("scores: {");
        for(int i=0; i<cloned.scores.length; i++) {
            System.out.print(cloned.scores[i]);
            System.out.print((i==cloned.scores.length - 1) ? "" : ",");
        }
        System.out.println("}");
        System.out.println("car: " + cloned.car.model);

        System.out.println();

        // 원본 객체를 출력시킨다.
        System.out.println("[원본 객체의 필드값]");
        System.out.println("name: " + original.name);
        System.out.println("age: " + original.age);
        System.out.print("scores: {");
        for(int i=0; i<original.scores.length; i++) {
            System.out.print(original.scores[i]);
            System.out.print((i==original.scores.length - 1) ? "" : ",");
        }
        System.out.println("}");
        System.out.println("car: " + original.car.model);
    }
}
```

실행 결과

```
[복제 객체의 필드값]
name: 홍길동
age: 25
scores: {100,90}
car: 그랜저

[원본 객체의 필드값]       // 원본 객체는 값이 변하지 않는 것을 확인할 수 있다.
name: 홍길동
age: 25
scores: {90,90}
car: 소나타
```

11.4 Objects 클래스

- **Objects의 정적 메소드**

리턴 타입	메소드(매개 변수)	설명
int	compare(T a, T b, Comparator<T> c)	두 객체 a와 b를 비교
boolean	deepEquals(Object a, Object b)	두 객체의 깊은 비교(참조 타입의 값 까지)
boolean	equals(Object a, Object b)	두 객체의 얕은 비교(참조 타입은 번지만 비교)
int	hash(Object... values)	객체가 null인지 조사
int	hashCode(Object o)	객체의 해시코드 생성
boolean	isNull(Object obj)	객체가 null인지 조사
boolean	nonNull(Object obj)	객체가 null이 아닌지 조사
T	requireNonNull(T obj)	객체가 null인 경우 예외 발생
T	requireNonNull(T obj, String message)	객체가 null인 경우 예외 발생(주어진 예외 메시지 포함)
T	requireNonNull(T obj, Supplier<String> messageSupplier)	객체가 null인 경우 예외 발생(람다식이 만든 예외 메시지 포함)
String	toString(Object o)	객체의 toString() 리턴값 리턴
String	toString(Object o, String nullDefault)	객체의 toString() 리턴값 리턴, 첫 번째 매개값이 null일 경우 두 번째 매개값 리턴

11.4.1 객체 비교(compare(T a, T b, Comparator<T>c))

- **T** : 비교할 객체 타입
- **리턴 타입(int)** : compare() 메소드를 실행시켜 a와 b를 비교했을 때, a가 b보다 작으면 음수, 같으면 0, 크면 양수를 리턴하도록 구현 클래스를 만들어야 한다.

- 예제

  CompareExample.java(**비교자 사용**)

```
package objects_class.compare_method;

import java.util.Comparator;
import java.util.Objects;

public class CompareExample {
    public static void main(String[] args) {
        // Student 객체 생성
        Student s1 = new Student(1);
        Student s2 = new Student(1);
        Student s3 = new Student(3);

        // result 에 객체를 비교한 결과 저장
        int result = Objects.compare(s1, s2, new StudentComparator());
        System.out.println(result);
        result = Objects.compare(s1, s3, new StudentComparator());
        System.out.println(result);
    }

    // Student 클래스
    static class Student {
        int sno;
        Student(int sno) {
            this.sno = sno;
        }
    }

    // Student 객체를 비교하는 클래스
    static class StudentComparator implements Comparator<Student> {
        public int compare(Student o1, Student o2) {
            /*if (o1.sno < o2.sno) return -1;
            else if (o1.sno == o2.sno) return 0;
            else return 1;*/
            return Integer.compare(o1.sno, o2.sno);
        }
    }
}
```

  실행 결과

```
0
-1
```

11.4.2 동등 비교(equals()와 deepEquals())

- **Objects.equals(a, b) 결과값**

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

- **Objects.deepEquals(Object a, Object b) 결과값**

a	b	Obejcts.deepEquals(a, b)
not null(not array)	not null(not array)	a.equals(b)의 리턴값
not null(array)	not null(array)	Arrays.deepEquals(a,b)의 리턴값
not null	null	false
null	not null	false
null	null	true

- 예제

```
package objects_class.equals_and_deep_equals_method;

import java.util.Arrays;
import java.util.Objects;

public class EqualsAndDeepEqualsExample {
    public static void main(String[] args) {
        Integer o1 = 1000;
        Integer o2 = 1000;

        // 둘 다 값이 1000 이므로  true
        System.out.println(Objects.equals(o1, o2));

        // 한 쪽이 null 이므로 false
        System.out.println(Objects.equals(o1, null));
        System.out.println(Objects.equals(null, o2));
```

```
        // 둘 다 같은 null 이므로 true
        System.out.println(Objects.equals(null, null));

        // 둘 다 값이 1000 이므로 true
        System.out.println(Objects.deepEquals(o1, o2) + "\n");

        Integer[] arr1 = {1, 2};
        Integer[] arr2 = {1, 2};
        System.out.println("arr1 address : " + arr1.hashCode());
        System.out.println("arr2 address : " + arr2.hashCode());

        // 두 배열 객체의 주소가 다르기 때문에 false
        System.out.println(Objects.equals(arr1, arr2));

        // 두 배열 객체의 값이 같기 때문에 true
        System.out.println(Objects.deepEquals(arr1, arr2));

        // 두 배열 객체의 값이 같기 때문에 true
        System.out.println(Arrays.deepEquals(arr1, arr2));

        // 한 쪽이 null 이므로 false
        System.out.println(Objects.deepEquals(null, arr2));
        System.out.println(Objects.deepEquals(arr1, null));

        // 둘 다 null 이므로 true
        System.out.println(Objects.deepEquals(null, null));
    }
}
```

실행 결과

```
true
false
false
true
true

arr1 address : 2083562754
arr2 address : 2054798982
false
true
true
false
false
true
```

11.4.3 해시코드 생성(hash(), hashCode())

: 매개값으로 주어진 값들을 이용해서 해시 코드를 생성하는 역할을 한다.

- 예제

```
package objects_class.hash_and_hashcode_method;

import java.util.Objects;

public class HashCodeExample {
    public static void main(String[] args) {
        // 필드 값을 동일하게 객체를 생성해준다.
        Student s1 = new Student(1, "홍길동");
        Student s2 = new Student(1, "홍길동");

        // s1과 s2의 해시 코드를 출력한다.
        System.out.println(s1.hashCode());
        System.out.println(Objects.hashCode(s2));
    }

    // Student 클래스
    static class Student {
        // 필드
        int sno;
        String name;

        // 생성자
        Student(int sno, String name) {
            this.sno = sno;
            this.name = name;
        }

        // hashCode 메소드를 재정의해서
        // 필드 값이 동일하면 같은 해시 코드를 가지도록 한다.
        @Override
        public int hashCode() {
            return Objects.hash(sno, name);
        }
    }
}
```

실행 결과

```
54151054
54151054
```

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

- **메소드 설명**

리턴 타입	메소드(매개 변수)	설명
T	requireNonNull(T obj)	not null -> obj null -> NullPointerException
T	requireNonNull(T obj, String message)	not null -> obj null -> NullPointerException(message)
T	requireNonNull(T obj, Supplier<String> msgSupplier)	not null -> obj null -> NullPointerExeption(msgSupplier.get())

- **예제(null 여부 조사)**

```
package objects_class.null_method;

import java.util.Objects;

public class NullExample {
    public static void main(String[] args) {
        // String 객체 선언
        String str1 = "홍길동";
        String str2 = null;

        // str1 null 조사
        System.out.println(Objects.requireNonNull(str1));

        // str2 null 조사 후 null 리턴
        try {
            String name = Objects.requireNonNull(str2);
        } catch(Exception e) {
            System.out.println(e.getMessage());
        }

        // str2 null 조사 후 메시지 출력
        try {
            String name = Objects.requireNonNull(str2, "이름이 없습니다.");
        } catch(Exception e) {
            System.out.println(e.getMessage());
        }

        // str2 null 조사 후 메시지 출력
        try {
            // 람다식 이용
            String name = Objects.requireNonNull(str2, ()->"이름이 없다니깐요");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

실행 결과

```
홍길동
이름이 없습니다.
이름이 없다니깐요
```

11.4.5 객체 문자 정보(toString())

: 객체의 문자 정보를 리턴한다.

- **toString()의 두 가지 오버로딩**

리턴 타입	메소드(매개 변수)	설명
String	toString(Object o)	not null -> o.toString() null -> "null"
String	toString(Object o, String nullDefault)	not null -> o.toString() null -> nullDefault

- **예제(객체 문자 정보)**

```
package objects_class.to_string_method;

import java.util.Objects;

public class ToStringExample {
    public static void main(String[] args) {
        // String 객체 선언
        String str1 = "홍길동";
        String str2 = null;

        // str1의 문자열 리턴
        System.out.println(Objects.toString(str1));

        // str2는 문자열이 없으므로 null 리턴
        System.out.println(Objects.toString(str2));

        // str2는 문자열이 없으므로 nullDefault 메시지 리턴
        System.out.println(Objects.toString(str2, "이름이 없습니다."));
    }
}
```

실행 결과

```
홍길동
null
이름이 없습니다.
```

11.5 System 클래스

: System 클래스를 이용하면 운영체제의 일부 기능을 사용 가능. 즉, 프로그램 종료, 키보드로부터 입력, 모니터 출력, 메모리 정리, 현재 시간 읽기, 시스템 프로퍼티(데이터) 읽기, 환경 변수 읽기 등 이 가능하다.

11.5.1 프로그램 종료(exit())

: 현재 실행하고 있는 프로세스를 강제 종료시키는 역할을 한다.

- **종료 상태값을 조사하는 방법**

```
System.setSecurityManager(new SecurityManager(){
    @Override
    public void checkExit(int status) {
        if(status != 5) {
            throw new SecurityException();
        }
    }
}
```

> 종료 상태값을 조사해서 특정 값이 입력되지 않으면 SecurityException을 발생시켜 System.exit()를 호출한 곳에서 예외 처리를 할 수 있도록 한다.

- **예제(exit 메소드)**

```
package system_class.exit_method;

public class ExitExample {
    public static void main(String[] args) {
        // 보안 관리자 설정
        System.setSecurityManager(new SecurityManager(){
            // 종료 상태 값이 5가 아닐땐 예외 발생
            @Override
            public void checkExit(int status) {
                if(status != 5) {
                    throw new SecurityException();
                }
            }
        });

        for(int i=0; i<10; i++) {
            // i 값 출력
            System.out.println(i);

            try {
                // exit 메소드 실행
                System.exit(i);
            } catch(SecurityException e){ } // 예외 처리
        }
```

```
    }
}
```

실행 결과

```
0
1
2
3
4
5       // 프로그램 종료
```

11.5.2 쓰레기 수집기 실행(gc())

: JVM 에게 가능한한 빨리 쓰레기 수집기를 실행시키라는 요청을 보내는 메소드이다.

- **예제(gc 메소드)**

```
package system_class.gc_method;

public class GcExample {
    public static void main(String[] args) {
        // Employee 타입 변수 선언
        Employee emp;

        // employee 객체 생성
        emp = new Employee(1);

        // Employee(1)은 쓰레기 객체가 된다.
        emp = null;

        emp = new Employee(2);

        // Employee(2)는 쓰레기 객체가 된다.
        emp = new Employee(3);

        System.out.print("emp가 최종적으로 참조하는 사원번호: ");
        System.out.println(emp.eno);

        // 쓰레기 수집기를 통해 쓰레기 객체를 메모리에서 제거시킴
        System.gc();
    }
}

class Employee {
    public int eno;

    public Employee(int eno) {
        this.eno = eno;
        System.out.println("Employee(" + eno + ") 가 메모리에 생성됨");
```

```
    }

    public void finalize() {
        System.out.println("Employee(" + eno + ") 이 메모리에서 제거됨");
    }
}
```

실행 결과

```
Employee(1) 가 메모리에 생성됨
Employee(2) 가 메모리에 생성됨
Employee(3) 가 메모리에 생성됨
emp가 최종적으로 참조하는 사원번호: 3
Employee(1) 이 메모리에서 제거됨
Employee(2) 이 메모리에서 제거됨
```

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

- **예제(프로그램 실행 소요 시간 구하기)**

```
package system_class.time_method;

public class SystemTimeExample {
    public static void main(String[] args) {
        // 시작 시간 읽기
        long time1 = System.nanoTime();

        int sum = 0;
        for(int i=1; i<=1000000; i++) {
            sum += i;
        }

        // 끝 시간 읽기
        long time2 = System.nanoTime();

        System.out.println("1~1000000 까지의 합: " + sum);
        System.out.println("계산에 " + (time2 - time1) + " 나노초가 소요되었습니다.");
    }
}
```

실행 결과

```
1~1000000 까지의 합: 1784293664
계산에 3472501 나노초가 소요되었습니다.
```

11.5.4 시스템 프로퍼티 읽기(getProperty())

: JVM이 시작할 때 자동 설정되는 시스템의 속성값을 말한다. 예를 들어 운영체제의 종류 및 자바 프로그램을 실행시킨 사용자 아이디, JVM의 버전, 운영체제에서 사용되는 파일 경로 구분자 등이 있다. 시스템 프로퍼티는 키(key)와 값(value)으로 구성된다.

키(key)	설명	값(value)
java.version	자바의 버전	1.8.0_20
java.home	사용하는 JRE의 파일 경로	<jdk 설치경로>\jre
os.name	Operating system name	Windows 7
file.separator	File separator("/" on Unix)	\
user.name	사용자의 이름	사용자계정
user.home	사용자의 홈 디렉토리	C:\Users\사용자계정
user.dir	사용자가 현재 작업 중인 디렉토리 경로	다양

- **예제(시스템 프로퍼티 읽기)**

```
package system_class;

import java.util.Objects;
import java.util.Properties;
import java.util.Set;

public class get_property_method {
    public static void main(String[] args) {
        // String 객체에 특정 시스템 프로퍼티 저장
        String osName = System.getProperty("os.name");
        String userName = System.getProperty("user.name");
        String userHome = System.getProperty("user.home");

        System.out.println("운영체제 이름: " + osName);
        System.out.println("사용자 이름: " + userName);
        System.out.println("사용자 홈디렉토리: " + userHome);

        System.out.println("----------------------------------");
        System.out.println(" [ key ]  value");
        System.out.println("----------------------------------");

        // 모든 프로퍼티 속성의 키와 값을 호출
        Properties props = System.getProperties();

        // 키만으로 구성된 Set 객체 저장
        Set keys = props.keySet();

        for(Object objKey : keys) {
            // 키를 하나씩 얻어내어 문자열로 변환한 뒤
            String key = (String) objKey;
```

```
            // 키로 값을 불러온다.
            String value = System.getProperty(key);


            System.out.println("[ " + key + " ]  " + value);
        }
    }
}
```

실행 결과

```
운영체제 이름: Windows 10
사용자 이름: lenovo
사용자 홈디렉토리: C:\Users\lenovo
-----------------------------------
 [ key ]  value
-----------------------------------
[ sun.desktop ]  windows
[ awt.toolkit ]  sun.awt.windows.WToolkit
...
```

11.5.5 환경 변수 읽기(getenv())

- **환경 변수** : 운영체제에서 이름과 값으로 관리되는 문자열 정보. 운영체제가 설치될 때 기본적인 내용이 설정되고, 사용자가 직접 설정하거나 응용 프로그램이 설치될 때 자동적으로 추가 설정되기도 한다.
- **예제(JAVA_HOME 환경 변수 값 얻기)**

```
package system_class.getenv_method;

public class SystemEnvExample {
    public static void main(String[] args) {
        // JAVA_HOME 이라는 환경 변수의 값을 불러옴
        String javaHome = System.getenv("JAVA_HOME");
        System.out.println("JAVA_HOME: " + javaHome);
    }
}
```

실행 결과

```
JAVA_HOME: C:\Program Files\Java\jdk-11.0.1
```

11.6 Class 클래스

: java.lang 패키지에 소속되어 있으며 클래스와 인터페이스의 메타 데이터를 관리한다.

- **메타 데이터**
  - 클래스의 이름, 생산자 정보, 필드 정보, 메소드 정보

11.6.1 Class 객체 얻기(getClass(), forName())

- **예제**

```
package class_class.get_class_method;

import object_class.clone_method.deep_clone.Car;

public class ClassExample {
    public static void main(String[] args) {
        // Car 객체 생성
        Car car = new Car("1");

        // Car 객체의 정보를 clazz1에 저장
        Class clazz1 = car.getClass();

        // Car 클래스 전체 이름(패키지가 포함된 이름)을 저장
        String clazzName = clazz1.getName();

        // 클래스의 전체이름과 간단한 이름 그리고 패키지 이름 출력
        System.out.println(clazz1.getName());
        System.out.println(clazz1.getSimpleName());
        System.out.println(clazz1.getPackage().getName());
        System.out.println();

        try {
            // 클래스의 전체이름과 간단한 이름 그리고 패키지 이름 출력
            Class clazz2 = Class.forName(clazzName);
            System.out.println(clazz2.getName());
            System.out.println(clazz2.getSimpleName());
            System.out.println(clazz2.getPackage().getName());
        } catch (ClassNotFoundException e) {    // 예외 처리
            e.printStackTrace();
        }
    }
}
```

실행 결과

```
object_class.clone_method.deep_clone.Car
Car
object_class.clone_method.deep_clone

object_class.clone_method.deep_clone.Car
Car
object_class.clone_method.deep_clone
```

11.6.2 리플렉션(getDeclaredConstructors(), getDeclaredFields(), getDeclaredMethods())

: Class 객체를 이용하면 클래스의 생성자, 필드, 메소드 정보를 알아낼 수 있는데 이것을 리플렉션이라 한다.

- **예제(동적으로 클래스 멤버 정보 얻기)**

```
package class_class.reflection_method;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class ReflectionExample {
    public static void main(String[] args) throws Exception {
        // Car 객체의 정보를 불러와서 Class 타입의 clazz 에 저장한다.
        Class clazz = Class.forName("object_class.clone_method.deep_clone.Car");

        // 클래스 이름을 호출한다.
        System.out.println("[클래스 이름]");
        System.out.println(clazz.getName());
        System.out.println();

        // 클래스의 생성자 정보를 호출한다.
        System.out.println("[생성자 정보]");
        Constructor[] constructors = clazz.getDeclaredConstructors();
        for(Constructor constructor : constructors) {
            System.out.print(constructor.getName() + "(");
            Class[] parameters = constructor.getParameterTypes();
            printParameters(parameters);
            System.out.println(")");
        }
        System.out.println();

        // 클래스의 필드 정보를 호출한다.
        System.out.println("[필드 정보]");
        Field[] fields = clazz.getDeclaredFields();
        for(Field field : fields) {
            System.out.println(field.getType().getSimpleName() + " " +
                    field.getName());
        }
        System.out.println();

        // 클래스의 메소드 정보를 호출한다.
        System.out.println("[메소드 정보]");
        Method[] methods = clazz.getDeclaredMethods();
        for(Method method : methods) {
            System.out.print(method.getName() + "(");
            Class[] parameters = method.getParameterTypes();
            printParameters(parameters);
```

```
            System.out.println(")");
        }

    }

    // 메소드의 파라미터들을 호출해주는 메소드
    private static void printParameters(Class[] parameters) {
        for(int i=0; i<parameters.length; i++) {
            System.out.print(parameters[i].getName());
            if(i < (parameters.length -1)) {
                System.out.println(",");
            }
        }
    }
}
```

실행 결과

```
[클래스 이름]
object_class.clone_method.deep_clone.Car

[생성자 정보]
object_class.clone_method.deep_clone.Car(java.lang.String)

[필드 정보]
String model

[메소드 정보]
```

11.6.3 동적 객체 생성(newInstance())(질문)

: Class 객체를 이용하면 new 연산자를 사용하지 않아도 동적으로 객체를 생성할 수 있다.

- **newInstance() 메소드** : 기본 생성자를 호출해서 객체를 생성하기 때문에 반드시 클래스에 기본 생성자가 존재해야 한다.
- **예제(동적 객체 생성 및 실행)**

  Action.java(인터페이스)

```
package class_class.new_instance_method;

public class NewInstanceExample {
    public static void main(String[] args) {
        try{
            Class clazz = Class.forName("class_class.new_instance_method.Action");
            Action action = (Action) clazz.newInstance();
            action.execute();
            Class clazz2 =
Class.forName("class_class.new_instance_method.ReceiveAction");
```

```
            ReceiveAction receiveAction = (ReceiveAction) clazz2.newInstance();
            receiveAction.execute();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        }
    }
}
```

실행 결과

```
java.lang.InstantiationException: class_class.new_instance_method.Action
        at java.base/java.lang.Class.newInstance(Class.java:547)
        at
class_class.new_instance_method.NewInstanceExample.main(NewInstanceExample.java:7)
Caused by: java.lang.NoSuchMethodException: class_class.new_instance_method.Action.
<init>()
        at java.base/java.lang.Class.getConstructor0(Class.java:3302)
        at java.base/java.lang.Class.newInstance(Class.java:532)
        ... 1 more
```

11.7 String 클래스

: 문자열을 생허나는 방법과 추출, 비교, 찾기, 분리, 변환 등의 메소드를 제공한다.

11.7.1 String 생성자

: 자바의 문자열은 java.lang 패키지의 String 클래스의 인스턴스로 관리된다.

- **String 클래스의 다양한 매개 값**

```
// 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)
```

- **예제(바이트 배열을 문자열로 변환)**

```
package string_class.string_constructor;

public class ByteToStringExample {
    public static void main(String[] args) {
        // byte 타입 배열 객체 생성
        byte[] bytes = { 72, 101, 108, 108, 111, 32, 74, 97, 118, 97 };

        // String 생산자로 byte 배열을 문자열로 변환
        String str1 = new String(bytes);
        System.out.println(str1);

        // String 생산자로 byte 배열을 6 번째 위치부터 4개를 문자열로 변환
        String str2 = new String(bytes, 6, 4);
        System.out.println(str2);
    }
}
```

실행 결과

```
Hello Java
Java
```

- **예제(바이트 배열을 문자열로 변환)**

```
package string_class.string_constructor;

import java.io.IOException;

public class KeyboardToStringExample {
    public static void main(String[] args) throws IOException {
        // byte 배열 객체 생성
        byte[] bytes = new byte[100];

        // 입력을 받아 bytes 배열에 저장 및
        // bytes 배열의 길이 저장
        System.out.print("입력: ");
        int readByteNo = System.in.read(bytes);

        // bytes 배열을 String 으로 변환하여 String 객체를 생성한다.
        String str = new String(bytes, 0, readByteNo-1);
        System.out.println(str);
    }
}
```

실행 결과

```
입력: hello
hello
```

11.7.2 String 메소드

- **String 메소드**

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

문자 추출(charAt())

: 매개값으로 주어진 인덱스의 문자를 리턴한다.

- **예제(주민등록번호에서 남자와 여자를 구분하는 방법)**

```
package string_class.string_method;

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

```
        // 문자열 타입 ssn 변수 초기화
        String ssn = "010624-1230123";
        // ssn 문자열의 7번 째 값 저장
        char s = ssn.charAt(7);

        // 7번 째의 값이 1이거나 3이면 남자이고
        // 2이거나 4이면 여자이다.
        switch (s) {
            case '1':
            case '3':
                System.out.println("남자 입니다.");
                break;
            case '2':
            case '4':
                System.out.println("여자 입니다.");
                break;
        }
    }
}
```

문자열 비교(equals())

: 문자열을 비교하기 위해 쓰이는 메소드이다.

- **예제(문자열 비교)**

```
package string_class.string_method.equals_method;

public class StringEqualsExample {
    public static void main(String[] args) {
        String strVar1 = new String("신민철");
        String strVar2 = "신민철";

        // String 객체를 비교한다.
        if(strVar1 == strVar2) {
            System.out.println("같은 String 객체를 참조");
        } else {
            System.out.println("다른 String 객체를 참조");
        }

        // 객체의 문자열을 비교한다.
        if(strVar1.equals(strVar2)) {
            System.out.println("같은 문자열을 가짐");
        } else {
            System.out.println("다른 문자열을 가짐");
        }
    }
}
```

실행 결과

```
다른 String 객체를 참조
같은 문자열을 가짐
```

바이트 배열로 변환(getBytes())

: 문자열을 바이트로 배열로 변환하는 메소드. 이 메소드는 시스템의 기본 문자셋으로 인코딩된 바이트 배열로 리턴한다.

- **예제(바이트 배열로 변환)**

```
package string_class.string_method.get_bytes_method;

import java.io.UnsupportedEncodingException;

public class StringGetBytesExample {
    public static void main(String[] args) {
        String str = "안녕하세요";

        // 기본 문자셋으로 인코딩과 디코딩
        byte[] bytes1 = str.getBytes();
        System.out.println("bytes1.length: " + bytes1.length);
        String str1 = new String(bytes1);
        System.out.println("bytes1->String: " + str1);

        try {
            // EUC-KR을 이용해서 인코딩 및 디코딩
            byte[] bytes2 = str.getBytes("EUC-KR");
            System.out.println("bytes2.length: " + bytes2.length);
            String str2 = new String(bytes2, "EUC-KR");
            System.out.println("bytes2->String: " + str2);

            // UTF-8을 이용해서 인코딩 및 디코딩
            byte[] bytes3 = str.getBytes("UTF-8");
            System.out.println("bytes3.length: " + bytes3.length);
            String str3 = new String(bytes3, "UTF-8");
            System.out.println("bytes3->String: " + str3);

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

실행 결과

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

문자열 찾기(indexOf())

: 매개값으로 주어진 문자열이 시작되는 인덱스를 리턴한다. 만약 주어진 문자열이 포함되어 있지 않으면 -1을 리턴한다.

- **예제(문자열 포함 여부 조사)**

```
package string_class.string_method.index_of_method;

public class StringIndexOfExample {
    public static void main(String[] args) {
        // String 타입 변수 초기화
        String subject = "자바 프로그래밍";

        // 프로그래밍이 시작하는 인덱스를 찾아서 리턴
        int location = subject.indexOf("프로그래밍");
        System.out.println(location);

        // "자바" 라는 문자열이 있는지 조사
        if(subject.indexOf("자바") != -1) {
            System.out.println("자바와 관련된 책이군요.");
        } else {
            System.out.println("자바와 관련없는 책이군요.");
        }
    }
}
```

문자열 길이(length())

: 문자열의 길이(문자의 수)를 리턴한다.

- **예제(문자열의 문자 수 얻기)**

```
package string_class.string_method.length_method;

public class StringLengthExample {
    public static void main(String[] args) {
        // String 타입의 변수에 값 초기화
        String ssn = "111111-2222222";
        // String 변수의 문자열 길이를 호출
        int length = ssn.length();
```

```
        // 길이 비교
        if(length == 14) {
            System.out.println("주민번호 자리수가 맞습니다.");
        } else {
            System.out.println("주민번호 자리수가 틀립니다.");
        }
    }
}
```

실행 결과

```
주민번호 자리수가 맞습니다.
```

문자열 대치(replace())

: 첫 번째 매개값인 문자열을 찾아 두 번째 매개값인 문자열로 대치한 새로운 문자열을 생성하고 리턴한다.

- **예제**

```
package string_class.string_method.replace_method;

public class StringReplaceExample {
    public static void main(String[] args) {
        // String 타입 변수 초기화
        String oldStr = "자바는 객체지향언어 입니다.";

        // oldStr 의 자바를 JAVA 로 바꾸고 객체를 생성한다.
        String newStr = oldStr.replace("자바", "JAVA");

        // newStr 과 같은 문자열을 참조하게 한다.
        String newStr2 = "JAVA는 객체지향언어 입니다.";

        System.out.println(oldStr);
        System.out.println(newStr);
        System.out.println("oldStr's address: " + oldStr.hashCode());
        System.out.println("newStr's address: " + newStr.hashCode());
        System.out.println("newStr2's address: " + newStr2.hashCode());
    }
}
```

실행 결과

```
자바는 객체지향언어 입니다.
JAVA는 객체지향언어 입니다.
oldStr's address: -1883772379
newStr's address: 259555143
newStr2's address: 259555143
```

> 주소 값이 바뀐 것을 보고 객체가 새로 생긴 것을 알 수 있다.

문자열 잘라내기(substring())

: 주어진 인덱스에서 문자열을 추출한다.

- **substring(int beginIndex, int endIndex)**
  - int beginIndex : 추출할 문자열의 시작 위치
  - int endIndex : 추출할 문자열의 끝 위치
- **substring(int beginIndex)**

  : beginIndex 부터 끝까지 문자열을 추출한다.

- **예시**

```
package string_class.string_method.substring_method;

public class StringSubstringExample {
    public static void main(String[] args) {
        String ssn = "111111-2222222";

        // 0부터 6까지 문자열 추출
        String firstNum = ssn.substring(0, 6);
        System.out.println(firstNum);

        // 7부터 끝까지 문자열 추출
        String secondNum = ssn.substring(7);
        System.out.println(secondNum);
    }
}
```

실행 결과

```
111111
2222222
```

알파벳 소,대 문자 변경(toLowerCase(), toUpperCase())

: 문자열을 모두 소,대문자로 바꾼 새로운 문자열을 생성한 후 리턴한다.

- **예제**

```
package string_class.string_method.lower_upper_method;

public class StringToLowerUpperCaseExample {
    public static void main(String[] args) {
        String str1 = "Java Programming";
        String str2 = "JAVA PROGRAMMING";

        // 문자열이 동일한지 확인(false)
```

```
        System.out.println(str1.equals(str2));

        // 모든 문자열을 소문자로 바꾸고 새로운 객체 리턴
        String lowerStr1 = str1.toLowerCase();
        String lowerStr2 = str2.toLowerCase();

        // 문자열이 동일한지 확인(true)
        System.out.println(lowerStr1.equals(lowerStr2));

        // 대소문자를 무시하고 문자열을 비교한다.
        System.out.println(str1.equalsIgnoreCase(str2));
    }
}
```

실행 결과

```
false
true
true
```

문자열 앞뒤 공백 잘라내기(trim())

: 문자열의 앞뒤 공백을 제거한 새로운 문자열을 생성하고 새로운 객체를 리턴한다.

- **예제**

```
package string_class.string_method.trim_method;

public class StringTrimExample {
    public static void main(String[] args) {
        String tel1 = "  02";
        String tel2 = "123   ";
        String tel3 = "   1234   ";

        // trim() 메소드로 문자열의 양 옆 공백을 제거한 뒤 저장
        String tel = tel1.trim() + tel2.trim() + tel3.trim();
        System.out.println(tel);
    }
}
```

실행 결과

```
021231234
```

문자열 반환(valueOf())

: 기본 타입의 값을 문자열로 변환하는 기능

- **예제**

```
package string_class.string_method.value_of_method;

public class StringValueOfExample {
    public static void main(String[] args) {
        // 정수, 소수, 부울을 모두 String 타입으로 변환한다.
        String str1 = String.valueOf(10);
        String str2 = String.valueOf(10.5);
        String str3 = String.valueOf(true);

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);

        // 모두 같은 String 객체를 참조하고 있음을 알 수 있다.
        System.out.println(str1.getClass());
        System.out.println(str2.getClass());
        System.out.println(str3.getClass());
    }
}
```

실행 결과

```
10
10.5
true
class java.lang.String
class java.lang.String
class java.lang.String
```

11.8 String Tokenizer 클래스

: 문자열이 특정 구분자(delimiter)로 연결되어 있을 경우, 구분자를 기준으로 부분 문자열을 분리한다.

11.8.1 split() 메소드

: 정규 표현식을 구분자로 해서 문자열을 분리한 후, 배열에 저장하고 리턴한다.

```
String[] result = "문자열".split("정규표현식");
```

- **예제(문자열 분리)**

```
package string_tokenizer_class;

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

```
        String text = "홍길동&이수홍,박연수,김자바-최명호";

        // 파이프를 제외한 기호들을 구분자로 해서 문자열을 추출한다.
        String[] names = text.split("&|,|-");

        for(String name : names) {
            System.out.println(name);
        }
    }
}
```

실행 결과

```
홍길동
이수홍
박연수
김자바
최명호
```

11.8.2 String Tokenizer 클래스

: 문자열이 한 종류의 구분자로 연결되어 있을 경우, 이 메소드를 사용하면 손쉽게 문자열을 분리해 낼 수 있다.

```
StringTokenizer st = new StringTokenizer("문자열", "구분자");
```

- **StringTokenizer의 메소드**

반환형	메소드	설명
int	countTokens()	꺼내지 않고 남아 있는 토큰의 수
boolean	hasMoreTokens()	남아 있는 토큰이 있는지 여부
String	nextToken()	토큰을 하나씩 꺼내옴

- **예제**

```
package string_tokenizer_class;

import java.util.StringTokenizer;

public class StringTokenizerExample {
    public static void main(String[] args) {
        String text = "홍길동/이수홍/박연수";

        // 전체 토큰 수를 얻어 for 문으로 루핑해서 문자열 분리
        StringTokenizer st = new StringTokenizer(text, "/");
        int countTokens = st.countTokens();
```

```
        for(int i=0; i<countTokens; i++) {
            String token = st.nextToken();
            System.out.println(token);
        }

        System.out.println();

        // 남아 있는 토큰을 확인하고 while 문으로 루핑해서 문자열 분리
        st = new StringTokenizer(text, "/");
        while( st.hasMoreElements()) {
            String token = st.nextToken();
            System.out.println(token);
        }
    }
}
```

실행 결과

```
홍길동
이수홍
박연수

홍길동
이수홍
박연수
```

11.9 StringBuffer, StringBuilder 클래스

: 문자열을 변경하는 작업이 많을 경우에는 String 클래스를 사용하는 것보다는 java.lang 패키지의 **StringBuffer 또는 StringBuilder 클래스**를 사용하는 것이 좋다. 이 두 클래스는 내부 버퍼(buffer: 데이터를 임시로 저장하는 메모리)에 문자열을 저장해 두고, 그 안에서 추가, 수정, 삭제 작업을 할 수 있도록 설계되어 있다.

- **StringBuffer** : 멀티 스레드 환경에서 사용
- **StringBuilder** : 단일 스레드 환경에서 사용
  - **StringBuilder 생성자**

    ```
    // StringBuilder(String str) 생성자는 str로 주어진 매개값을 버퍼의
    // 초기값으로 저장한다.
    StringBuilder sb = new StringBuilder();
    StringBuilder sb = new StringBuilder(16);
    StringBuilder sb = new StringBuilder("Java");
    ```

  - **메소드들**

메소드	설명
append(...)	문자열 끝에 주어진 매개값을 추가
insert(int offset, ...)	문자열 중간에 주어진 매개값을 추가
delete(int start, int end)	문자열의 일부분을 삭제
deleteCharAt(int index)	문자열에서 주어진 index의 문자를 삭제
replace(int start, String str)	문자열의 일부분을 다른 문자열로 대치
reverse()	문자열의 순서를 뒤바꿈
setCharAt(int index, char ch)	문자열에서 주어진 index의 문자를 다른 문자로 대치

- **예제**

```
package stringbuffer_and_stringbuilder;

public class StringBuilderExample {
    public static void main(String[] args) {
        // StringBuilder 객체 생성
        StringBuilder sb = new StringBuilder();

        // 문자열을 끝에 추가
        sb.append("Java ");
        sb.append("Program Study");
        System.out.println(sb.toString());

        // 4번째 문자 뒤에 2를 삽입
        sb.insert(4, "2");
        System.out.println(sb.toString());

        // 4번째 문자 뒤의 문자를 6으로 변경
        sb.setCharAt(4, '6');
        System.out.println(sb.toString());

        // 5번째 문자 뒤부터 13번째 문자까지를
        // "Book" 문자열로 대치
        sb.replace(6, 13, "Book");
        System.out.println(sb.toString());

        // 5번째 문자 삭제
        sb.delete(4, 5);
        System.out.println(sb.toString());

        // 총 문자수 얻기
        int length = sb.length();
        System.out.println("총문자수: " + length);
```

```
        // 버퍼에 있는 것을 String 타입으로 리턴
        String result = sb.toString();
        System.out.println(result);
    }
}
```

실행 결과

```
Java Program Study
Java2 Program Study
Java6 Program Study
Java6 Book Study
Java Book Study
총문자수: 15
Java Book Study
```

11.10 정규 표현식과 Pattern 클래스

: 정규 표현식은 문자 또는 숫자 기호와 반복 기호가 결합된 문자열이다.

11.10.1 정규 표현식 작성 방법

- **정규 표현식의 기본적인 기호들**

기호	설명
[]	한 개의 문자 **예) [abc] : a, b, c 중 하나의 문자** **[^abc] : a, b, c 이외의 하나의 문자** **[a-zA-Z] : a~z, A~Z 중 하나의 문자**
\d	한 개의 숫자, [0-9]와 동일
\s	공백
\w	한 개의 알파벳 또는 한 개의 숫자, [a-zA-Z0-9]와 동일
?	없음 또는 한 개
*	없음 또는 한 개 이상
+	한 개 이상
{n}	정확히 n개
{n,}	최소한 n개
{n, m}	n개에서부터 m개까지
()	그룹핑

- **02-123-1234 또는 010-1234-5678과 같은 전화번호를 위한 정규 표현식**

```
(02|010)-\d{3,4}-\d{4}
```

기호	설명
(02|010)	02 또는 010
-	- 포함
\d{3,4}	3자리 또는 4자리 숫자
-	- 포함
\d{4}	4자리 숫자

- **white@naver.com 과 같은 이메일을 위한 정규 표현식**

```
\w+@\w+\.\w+(\.\w+)?
```

기호	설명
\w+	한 개 이상의 알파벳 또는 숫자
@	@
\w+	한 개 이상의 알파벳 또는 숫자
\.	.
\w+	한 개 이상의 알파벳 또는 숫자
(\.\w+)?	\.\w+ 이 없거나 한개

11.10.2 Pattern 클래스

: 문자열을 정규 표현식으로 검증하는 기능은 java.util.regex.Pattern 클래스의 정적 메소드인 matches() 메소드가 제공한다.

```
boolean result = Pattern.matches("정규식", "검증할 문자열");
```

- **예제(문자열 검증하기)**

```
package pattern_class;

import java.util.regex.Pattern;

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

```
        String regExp = "(02|010)-\\d{3,4}-\\d{4}";
        String data = "010-123-4567";
        boolean result = Pattern.matches(regExp, data);

        if(result) {
            System.out.println("정규식과 일치합니다.");
        } else {
            System.out.println("정규식과 일치하지 않습니다.");
        }

        regExp = "\\w+@\\w+\\.\\w+(\\.\\w+)?";
        data = "angel@navercom";
        result = Pattern.matches(regExp, data);
        if(result) {
            System.out.println("정규식과 일치합니다.");
        } else {
            System.out.println("정규식과 일치하지 않습니다.");
        }
    }
}
```

실행 결과

```
정규식과 일치합니다.
정규식과 일치하지 않습니다.
```

11.11 Arrays 클래스

: Array 클래스는 **배열 조작 기능**을 가지고 있다. 배열 조작이란 배열의 **복사, 항목 정렬, 항목 검색**과 같은 기능을 말한다.

- **System.arraycopy() 메소드** : 단순한 배열 복사

Array 클래스의 모든 메소드는 **정적(static)이므로** Arrays 클래스로 바로 사용이 가능하다.

- **메소드들**

리턴 타입	메소드 이름	설명
int	binarySearch(배열, 찾는값)	전체 배열 항목에서 찾는 값이 있는 인덱스 리턴
타겟 배열	copyOf(원본배열, 복사할길이)	원본 배열의 0번 인덱스에서 복사할 길이만큼 복사한 배열 리턴, 복사할 길이는 원본 배열의 길이보다 커도 되며, 타겟 배열의 길이가 된다.
타켓 배열	copyOfRange(원본배열, 시작인덱스, 끝인덱스)	원본 배열의 0번 인덱스에서 복사할 길이만큼 복사한 배열 리턴, 복사 할 길이는 원본 배열의 길이보다 커도 되며, 타겟 배열의 길이가 된다.
boolean	deepEquals(배열, 배열)	두 배열의 깊은 비교(중첩 배열의 항목까지 비교)
boolean	equals(배열, 배열)	두 배열의 얕은 비교(중첩 배열의 항목은 비교하지 않음)
void	fill(배열, 값)	전체 배열 항목에 동일한 값을 저장
void	fill(배열, 시작인덱스, 끝인덱스, 값)	시작 인덱스부터 끝 인덱스까지 항목에만 동일한 값을 저장
void	sort(배열)	배열의 전체 항목을 오름차순으로 정렬
String	toString(배열)	"[값1,값2,...]"와 같은 문자열 리턴

11.11.1 배열 복사

- **copyOf(원본배열, 복사할길이) 메소드** : 원본 배열의 0번 인덱스에서 복사할 길이만큼 복사한 타겟 배열을 리턴.

```
char[] arr1 = {'J', 'A', 'V', 'A'};
char[] arr2 = Arrays.copyOf(arr1, arr1.length);
// arr2 = {'J', 'A', 'V', 'A'}
```

- **copyOfRange(원본배열, 시작인덱스, 끝인덱스)** : 원본 배열의 시작 인덱스에서 끝 인덱스까지 복사한 배열을 리턴한다. 시작 인덱스는 포함되지만, 끝 인덱스는 포함되지 않는다.

```
char[] arr1 = {'J', 'A', 'V', 'A'};
char[] arr2 = Arrays.copyOfRange(arr1, 1, 3);
// arr2 = {'A', 'V'}
```

- **System.arraycopy() 메소드**

```
// System.arraycopy(원본배열, 원본시작인덱스, 타겟배열, 타겟시작인덱스, 복사개수)
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
```

- **예제**

```
package arrays_class;

import java.util.Arrays;

public class ArrayCopyExample {
    public static void main(String[] args) {
        char[] arr1 = {'J', 'A', 'V', 'A'};

        // 방법1
        // arr1 전체를 arr2로 복사
        char[] arr2 = Arrays.copyOf(arr1, arr1.length);
        System.out.println(Arrays.toString(arr2));

        // 방법2
        // arr1[1] ~ arr1[2]를
        // arr3[0] ~ arr3[1]로 복사
        char[] arr3 = Arrays.copyOfRange(arr1, 1,3);
        System.out.println(Arrays.toString(arr3));

        // 방법3
        // arr1 전체를 arr4로 복사
        char[] arr4 = new char[arr1.length];
        System.arraycopy(arr1, 0, arr4, 0, arr1.length);
        for(int i=0; i<arr4.length; i++) {
            System.out.println("arr4[" + i + "]=" + arr4[i]);
        }
    }
}
```

11.11.2 배열 항목 비교

- **equals()** : 1차 항목의 값만 비교
- **deepEquals()** : 1차 항목이 서로 다른 배열을 참조할 경우 중첩된 배열의 항목까지 비교한다.
- **예제**

```
package arrays_class;

import java.lang.reflect.Array;
import java.util.Arrays;

public class EqualsExample {
    public static void main(String[] args) {
        int[][] original = { {1,2}, {3,4} };
```

```
        // 얕은 복사후 비교
        System.out.println("[얕은 복제후 비교]");
        int[][] cloned1 = Arrays.copyOf(original, original.length);
        System.out.println("배열 번지 비교: " + original.equals(cloned1));
        System.out.println("1차 배열 항목값 비교: " +
                Arrays.equals(original, cloned1));
        System.out.println("중첩 배열 항목값 비교: " +
                Arrays.deepEquals(original, cloned1));

        // 깊은 복사후 비교
        System.out.println("\n[깊은 복제후 비교]");
        int[][] cloned2 = Arrays.copyOf(original, original.length);
        cloned2[0] = Arrays.copyOf(original[0], original[0].length);
        cloned2[1] = Arrays.copyOf(original[1], original[1].length);
        System.out.println("배열 번지 비교: " +
                original.equals(cloned2));
        System.out.println("1차 배열 항목값 비교: " +
                Arrays.equals(original, cloned2));
        System.out.println("중첩 배열 항목값 비교: " +
                Arrays.deepEquals(original, cloned2));
    }
}
```

실행 결과

```
[얕은 복제후 비교]
배열 번지 비교: false
1차 배열 항목값 비교: true
중첩 배열 항목값 비교: true

[깊은 복제후 비교]
배열 번지 비교: false
1차 배열 항목값 비교: false
중첩 배열 항목값 비교: true
```

11.11.3 배열 항목 정렬

- **Arrays.sort() 메소드** : 배열이 자동으로 오름차순 정렬이된다.
- **예제**

```
package arrays_class;

import java.lang.reflect.Array;
import java.util.Arrays;

public class SortExample {
    public static void main(String[] args) {
        int[] scores = { 99, 97, 98 };
        Arrays.sort(scores);
        for(int i=0; i<scores.length; i++) {
```

```
            System.out.println("scores[" + i + "]=" + scores[i]);
        }
        System.out.println();

        String[] names = { "홍길동", "박동수", "김민수" };
        Arrays.sort(names);
        for(int i=0; i<names.length; i++) {
            System.out.println("names[" + i + "]=" +
                    names[i]);
        }
        System.out.println();

        Member m1 = new Member("홍길동");
        Member m2 = new Member("박동수");
        Member m3 = new Member("김민수");
        Member[] members = { m1, m2, m3 };
        Arrays.sort(members);
        for(int i=0; i<members.length; i++) {
            System.out.println("members[" + i + "].name=" +
                    members[i].name);
        }
    }
}
```

실행 결과

```
scores[0]=97
scores[1]=98
scores[2]=99

names[0]=김민수
names[1]=박동수
names[2]=홍길동

members[0].name=김민수
members[1].name=박동수
members[2].name=홍길동
```

11.11.4 배열 항목 검색

- **배열 검색** : 배열 항목에서 특정 값이 위치한 인덱스를 얻는 것을 배열 검색이라고 한다.
- 배열 항목을 검색하려면 먼저 **Arrays.sort() 메소드**로 항목들을 오름차순으로 정렬한 후, **Arrays.binarySearch() 메소드**로 항목을 찾아야 한다.
- **예제**

```
package arrays_class;

import java.util.Arrays;
```

```
public class SearchExample {
    public static void main(String[] args) {
        // 기본 타입값 탐색
        int[] scores = { 99, 97, 98 };
        Arrays.sort(scores);
        int index = Arrays.binarySearch(scores, 99);
        System.out.println("찾은 인덱스: " + index);

        // 문자열 검색
        String[] names = { "홍길동", "박동수", "김민수" };
        Arrays.sort(names);
        index = Arrays.binarySearch(names, "홍길동");
        System.out.println("찾은 인덱스: " + index);

        // 객체 검색
        Member m1 = new Member("홍길동");
        Member m2 = new Member("박동수");
        Member m3 = new Member("김민수");
        Member[] members = { m1, m2, m3 };
        Arrays.sort(members);
        index = Arrays.binarySearch(members, m1);
        System.out.println("찾은 인덱스: " + index);
    }
}
```

실행 결과

```
찾은 인덱스: 2
찾은 인덱스: 2
찾은 인덱스: 2
```

11.12 Wrapper(포장) 클래스

: 자바는 기본 타입(byte, char, short, int, long, float, double, boolean)의 값을 갖는 객체를 생성할 수 있다. 이런 객체를 **포장(Wrapper) 객체**라고 한다.

- **포장 클래스들**

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

11.12.1 박싱(Boxing)과 언박싱(Unboxing)

- **박싱(Boxing)** : 기본 타입의 값을 포장 객체로 만드는 과정
- **언박싱(Unboxing)** : 포장 객체에서 기본 타입의 값을 얻어내는 과정
- **박싱하는 방법**

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");

- **생성자를 이용하지 않고 박싱하는 방법**

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

- **박싱된 포장 객체의 값을 얻어내는 방법**

타입	메소드
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();

- **예제(기본 타입의 값을 박싱하고 언박싱하기)**

```
package wrapper_class;

public class BoxingUnboxingExample {
    public static void main(String[] args) {
        // Boxing
        Integer obj1 = new Integer(100);
        Integer obj2 = new Integer("200");
        Integer obj3 = Integer.valueOf("300");

        // Unboxing
        int value1 = obj1.intValue();
        int value2 = obj2.intValue();
        int value3 = obj3.intValue();

        System.out.println(value1);
        System.out.println(value2);
        System.out.println(value3);
    }
}
```

실행 결과

```
100
200
300
```

11.12.2 자동 박싱과 언박싱

- **자동 박싱** : 은 포장 클래스 타입에 기본값이 대입될 경우에 발생한다.

```
Integer obj = 100;      // 자동 박싱
```

- **자동 언박싱** : 기본 타입에 포장 객체가 대입될 경우에 발생.

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

- **예제**

```
package wrapper_class;

public class AutoBoxingUnboxingExample {
    public static void main(String[] args) {
        // 자동 Boxing
        Integer obj = 100;
        System.out.println("value: " + obj.intValue());

        // 대입 시 자동 Unboxing
        int value = obj;
        System.out.println("value: " + value);

        // 연산 시 작동 Unboxing
        int result = obj + 100;
        System.out.println("result: " + result);
    }
}
```

11.12.3 문자열을 기본 타입 값으로 변환

: 대부분의 포장 클래스에는 "parse+기본타입" 명으로 되어 있는 정적(static) 메소드가 있다. 이 메소드는 문자열을 매개값으로 받아 기본 타입 값으로 변환한다.

- **parse 메소드**

타입	메소드
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");

- **예제(문자열을 기본 타입 값으로 변환)**

```
package wrapper_class;

public class StringToPrimitiveValueExample {
    public static void main(String[] args) {
        int value1 = Integer.parseInt("10");
        double value2 = Double.parseDouble("3.14");
        boolean value3 = Boolean.parseBoolean("true");

        System.out.println("value1: " + value1);
        System.out.println("value2: " + value2);
        System.out.println("value3: " + value3);
    }
}
```

실행 결과

```
value1: 10
value2: 3.14
value3: true
```

11.12.4 포장 값 비교

: 포장 객체는 내부의 값을 비교하기 위해 **== 와 != 연산자**를 사용할 수 없다.

- **예외**

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

> 포장 객체에 정확히 어떤 값이 저장될지 모르는 상황이라면 ==와 != 연산자는 사용하지 않는 것이 좋다.

- **equals() 메소드**로 내부 값을 비교하는 것이 좋다.

```
package wrapper_class;

public class ValueCompareExample {
    public static void main(String[] args) {
        System.out.println("[-128~127 초과값일 경우]");
        Integer obj1 = 300;
        Integer obj2 = 300;
        System.out.println("==결과: " + (obj1 == obj2));
        System.out.println("언박싱후 == 결과: " +
```

```
                (obj1.intValue() == obj2.intValue()));
        System.out.println("equals() 결과: " +
                obj1.equals(obj2));
        System.out.println();

        System.out.println("[-128~127 범위값일 경우]");
        Integer obj3 = 10;
        Integer obj4 = 10;
        System.out.println("==결과: " +
                (obj3 == obj4));
        System.out.println("언박싱후 == 결과: " +
                (obj3.intValue() == obj4.intValue()));
        System.out.println("equals() 결과: " +
                obj3.equals(obj4));
    }
}
```

실행 결과

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

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

11.13 Math.Random 클래스

11.13.1 Math 클래스

: java.lang.Math 클래스는 수학 계산에 사용할 수 있는 메소드를 제공한다.

- **Math 클래스의 메소드들**

메소드	설명	예제코드	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)	올림 값	double v3 = Math.ceil(5.3); double v4 = Math.ceil(-5.3);	v3 = 6.0 v4 = -5.0
double floor(double a)	버림 값	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 v10 = Math.max(5.3, 2.5)	v7 = 9 v8 = 5.3
double random()	랜덤 값	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

- **예제**

```
package math_class;

public class MathExample {
    public static void main(String[] args) {
        // 절대값
        int v1 = Math.abs(-5);
        double v2 = Math.abs(-3.14);
        System.out.println("v1= " + v1);
        System.out.println("v2= " + v2);
        System.out.println();

        // 올림 값
        double v3 = Math.ceil(5.3);
        double v4 = Math.ceil(-5.3);
        System.out.println("v3= " + v3);
```

```
        System.out.println("v4= " + v4);
        System.out.println();

        // 버림 값
        double v5 = Math.floor(5.3);
        double v6 = Math.floor(-5.3);
        System.out.println("v5= " + v5);
        System.out.println("v6= " + v6);
        System.out.println();

        // 최대값
        int v7 = Math.max(5, 9);
        double v8 = Math.max(5.3, 2.5);
        System.out.println("v7= " + v7);
        System.out.println("v8= " + v8);
        System.out.println();

        // 최소값
        int v9 = Math.min(5, 9);
        double v10 = Math.min(5.3, 2.5);
        System.out.println("v9= " + v9  );
        System.out.println("v10= " + v10);
        System.out.println();

        // 랜덤값
        double v11 = Math.random();
        System.out.println("v11= " + v11);
        System.out.println();

        // 가까운 정수의 실수값
        double v12 = Math.rint(5.3);
        double v17 = Math.rint(5.5);
        double v13 = Math.rint(5.7);
        System.out.println("v12= " + v12);
        System.out.println("v13= " + v13);
        System.out.println("v17= " + v17);
        System.out.println();

        // 반올림값
        long v14 = Math.round(5.3);
        long v15 = Math.round(5.7);
        System.out.println("v14= " + v14);
        System.out.println("v15= " + v15);
        System.out.println();

        double value = 12.3456;
        double temp1 = value * 100;
        long temp2 = Math.round(temp1);
        double v16 = temp2 / 100.0;
        System.out.println("v16= " + v16);
        System.out.println();
    }
}
```

실행 결과

```
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= 0.7004030061497114

v12= 5.0
v13= 6.0
v17= 6.0

v14= 5
v15= 6

v16= 12.35
```

- **Math.random()** : 0.0과 1.0 사이의 범위에 속하는 하나의 double 타입의 값을 리턴한다.
  - **주사위 번호 뽑기 예시**

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

  - **로또 번호 뽑기 예시**

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

11.13.2 Random 클래스

: boolean, int, long, float, double 난수를 얻을 수 있다. 또 Random 클래스는 종자값(seed)을 설정할 수 있다. 종자값은 난수를 만드는 알고리즘에 사용되는 값으로 종자값이 같으면 같은 난수를 얻는다.

- **Random 객체 생성자**

생성자	설명
Random()	호출 시마다 다른 종자값(현재시간 이용)이 자동 설정된다.
Random(long seed)	매개값으로 주어진 종자값이 설정된다.

- **Random 클래스의 메소드들**

리턴값	메소드(매개 변수)	설명
boolean	nextBoolean()	boolean 타입의 난수를 리턴
double	nextDouble()	double 타입의 난수를 리턴(0.0 <= ~ < 1.0)
int	nextInt()	int 타입의 난수를 리턴(-2^31 <= ~ <= 2^31-1);
int	nextInt(int n)	int 타입의 난수를 리턴(0 <= ~ < n)

- **예제(로또 번호 얻기)**

```
package random_class;

import java.util.Arrays;
import java.util.Random;

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

        // 선택번호
        // 선택 번호 6개가 저장될 배열 생성
        int[] selectNumber = new int[6];
        // 선택 번호를 얻기 위한 Random 객체 생성
        Random random = new Random(3);
        System.out.print("선택 번호: ");
        for(int i=0; i<6; i++) {
            // 선택 번호를 얻어 배열에 저장
            selectNumber[i] = random.nextInt(45) + 1;
            System.out.print(selectNumber[i] + " ");
        }
        System.out.println();

        // 당첨번호
        // 당첨 번호 6개가 저장된 배열 생성
        int[] winningNumber = new int[6];
        // 당첨 번호를 얻기 위한 Random 객체 생성
        random = new Random(5);
        System.out.print("당첨 번호: ");
        for(int i=0; i<6; i++) {
            // 당첨 번호를 얻어 배열에 저장
            winningNumber[i] = random.nextInt(45) + 1;
            System.out.print(winningNumber[i] + " ");
        }
        System.out.println();
```

```
        // 당첨여부
        // 비교하기 전에 정렬시킴
        Arrays.sort(selectNumber);
        Arrays.sort(winningNumber);

        // 배열 항목 값 비교
        boolean result = Arrays.equals(selectNumber, winningNumber);
        System.out.print("당첨 여부: ");

        if(result) {
            System.out.println("1등에 당첨");
        } else {
            System.out.println("당첨되지 않았습니다.");
        }
    }
}
```

실행 결과

```
선택 번호: 15 21 16 17 34 28
당첨 번호: 18 38 45 15 22 36
당첨 여부: 당첨되지 않았습니다.
```

11.14 Date, Calendar 클래스

: 날짜 및 시각을 읽을 수 있도록 하는 클래스들. 이 두 클래스는 모두 java.util 패키지에 포함되어 있다.

11.14.1 Date 클래스

: 날짜를 표현하는 클래스이다. Date 클래스는 객체 간에 날짜 정보를 주고 받을 때 주로 사용된다.

- **Date() 생성자**

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

> 현재 날짜를 문자열로 얻고 싶다면 toString() 메소드를 사용하면 된다. 그리고 특정 문자열 포맷으로 얻고 싶다면 SimpleDateFormat 클래스를 이용하면 된다.

- **예제(현재 날짜를 출력하기)**

```
package date_class;

import java.text.SimpleDateFormat;
import java.util.Date;

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

```
        Date now = new Date();
        String strNow1 = now.toString();
        System.out.println(strNow1);

        SimpleDateFormat sdf =
                new SimpleDateFormat(
                        "yyyy년 MM월 dd일 hh시 mm분 ss초"
                );
        String strNow2 = sdf.format(now);
        System.out.println(strNow2);
    }
}
```

실행 결과

```
Sat Feb 16 18:44:38 KST 2019
2019년 02월 16일 06시 44분 38초
```

11.14.2 Calendar 클래스

: 달력을 표현한 클래스이다. Calendar 클래스는 추상(abstract) 클래스이므로 new 연산자를 사용해서 인스턴스를 생성할 수 없다. Calendar 클래스의 정적 메소드인 getInstance() 메소드를 이용하면 현재 운영체제에 설정되어 있는 시간대를 기준으로 한 Calendar 하위 객체를 얻을 수 있다.

- **예시**

```
// Calendar 하위 객체
Calendar now = Calendar.getInstance();

// Calendar 객체를 얻은 후 get() 메소드를 이용해서 날짜와 시간 정보 읽기
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
int amPm = now.get(Calendar.AM_PM);                     // 오전/오후
int hour = now.get(Calendar.HOUR);                      // 시간
int minute = now.get(Calendar.MINUTE);          // 분
int second = now.get(Calendar.SECOND);          // 초
```

- **예제**

```
package calendar_class;

import java.util.Calendar;

public class CalendarExample {
    public static void main(String[] args) {
        Calendar now = Calendar.getInstance();
```

```
        int year = now.get(Calendar.YEAR);
        int month = now.get(Calendar.MONTH) + 1;
        int day = now.get(Calendar.DAY_OF_MONTH);

        int week = now.get(Calendar.DAY_OF_WEEK);
        String strWeek = null;
        switch (week) {
            case Calendar.MONDAY:
                strWeek = "월";
                break;
            case Calendar.TUESDAY:
                strWeek = "화";
                break;
            case Calendar.WEDNESDAY:
                strWeek = "수";
                break;
            case Calendar.THURSDAY:
                strWeek = "목";
                break;
            case Calendar.FRIDAY:
                strWeek = "금";
                break;
            case Calendar.SATURDAY:
                strWeek = "토";
                break;
            default:
                strWeek = "일";
        }

        int amPm = now.get(Calendar.AM_PM);
        String strAmPm = null;
        if(amPm == Calendar.AM) {
            strAmPm = "오전";
        } else {
            strAmPm = "오후";
        }

        int hour = now.get(Calendar.HOUR);
        int minute = now.get(Calendar.MINUTE);
        int second = now.get(Calendar.SECOND);

        System.out.print(year + "년 ");
        System.out.print(month + "월 ");
        System.out.print(day + "일 ");
        System.out.print(strWeek + "요일 ");
        System.out.print(strAmPm + " ");
        System.out.print(hour + "시 ");
        System.out.print(minute + "분 ");
        System.out.print(second + "초 ");
    }
}
```

실행 결과

```
2019년 2월 16일 토요일 오후 7시 16분 16초
```

11.15 Format 클래스

: 원하는 문자열로 조합하는 형식 클래스이다.

11.15.1 숫자 형식 클래스(DecimalFormat)

: 숫자 데이터를 원하는 형식으로 표현하기 위해서 패턴을 사용.

- **DecimalFormat 생성자**

```
DecimalFormat df = new DecimalFormat("#,###.0");
String result = df.format(1234567.89)
```

- **예제**

```
package format_class;

import java.text.DecimalFormat;

public class DecimalFormatExample {
    public static void main(String[] args) {
        double num = 1234567.89;
        int i = 1;

        DecimalFormat df = new DecimalFormat("0");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("0.0");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("0000000000.00000");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("#");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("#.#");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("##########.#####");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("#.0");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("+#.0");
        System.out.println(i++ + ". " + df.format(num));
```

```
        df = new DecimalFormat("-#.0");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("#,###.0");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("0.0E0");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("+#,### ; -#,###");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("#.# %");
        System.out.println(i++ + ". " + df.format(num));

        df = new DecimalFormat("\u00A4 #,###");
        System.out.println(i++ + ". " + df.format(num));
    }
}
```

실행 결과

```
1. 1234568
2. 1234567.9
3. 0001234567.89000
4. 1234568
5. 1234567.9
6. 1234567.89
7. 1234567.9
8. +1234567.9
9. -1234567.9
10. 1,234,567.9
11. 1.2E6
12. +1,234,568
13. 123456789 %
14. ₩ 1,234,568
```

11.15.2 날짜 형식 클래스(SimpleDateFormat)

: Date 클래스의 toString() 메소드는 영문으로된 날짜를 리턴하는데 만약 특정 문자열 포맷으로 얻고 싶다면 이 형식 클래스를 사용한다.

- **SimpleDateFormat 생성자**

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

- **예제**

```
package format_class;

import java.text.SimpleDateFormat;
import java.util.Date;

public class SimpleDateFormatExample {
    public static void main(String[] args) {
        Date now = new Date();

        SimpleDateFormat sdf = new SimpleDateFormat(
                "yyyy-MM-dd"
        );
        System.out.println(sdf.format(now));

        sdf = new SimpleDateFormat(
                "yyyy년 MM월 dd일"
        );
        System.out.println(sdf.format(now));

        sdf = new SimpleDateFormat(
                "yyyy.MM.dd a HH:mm:ss"
        );
        System.out.println(sdf.format(now));

        sdf = new SimpleDateFormat(
                "오늘은 E요일"
        );
        System.out.println(sdf.format(now));

        sdf = new SimpleDateFormat(
                "올해의 D번째 날"
        );
        System.out.println(sdf.format(now));

        sdf = new SimpleDateFormat(
                "이달의 d번째 날"
        );
        System.out.println(sdf.format(now));
    }
}
```

11.15.3 문자열 형식 클래스(MessageFormat)

: 이 클래스를 사용하면 문자열에 데이터가 들어갈 자리를 표시해 두고, 프로그램이 실행하면서 동적으로 데이터를 삽입해 문자열을 완성시킨다.

- **예시**

  다음과 같이 회원 정보를 출력한다고 가정.

```
회원 ID: blue
회원 이름: 신용권
회원 전화: 010-123-1234
```

MessageFormat 클래스를 사용하면 좀 더 깔끔하게 데이터를 삽입시켜주고 전체 문자열을 쉽게 예측할 수 있다.

```
String message = "회원 ID: {0} \n회원이름 : {1} \n회원 전화: {2}";
String result = MessageFormat.format(message, id, name, tel);
```

> format() 메소드를 호출해서 완성된 문자열을 리턴시킨다.

- **예제**

```
package format_class;

import java.text.MessageFormat;

public class MessageFormatExample {
    public static void main(String[] args) {
        String id = "java";
        String name = "신용권";
        String tel = "010-123-5678";

        String text = "회원 ID: {0} \n회원 이름: {1} \n회원 전화: {2}";
        String result1 = MessageFormat.format(text, id, name, tel);
        System.out.println(result1);
        System.out.println();

        String sql = "insert into member values( {0}, {1}, {2} )";
        Object[] arguments = {"'java'", "'신용권'", "'010-123-5678'"};
        String result2 = MessageFormat.format(sql, arguments);
        System.out.println(result2);
    }
}
```

실행 결과

```
회원 ID: java
회원 이름: 신용권
회원 전화: 010-123-5678

insert into member values( 'java', '신용권', '010-123-5678' )
```