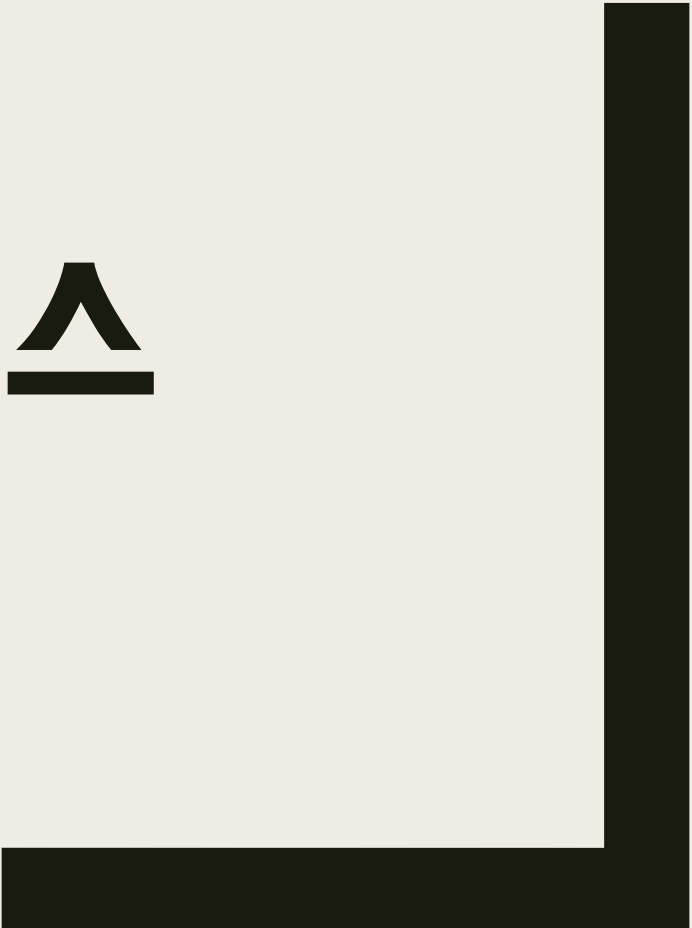




# 기본 API 클래스

SUMMARY 1  
JAVA STUDY  
YJ



# java.lang (기본적)

## java.util (조미료)

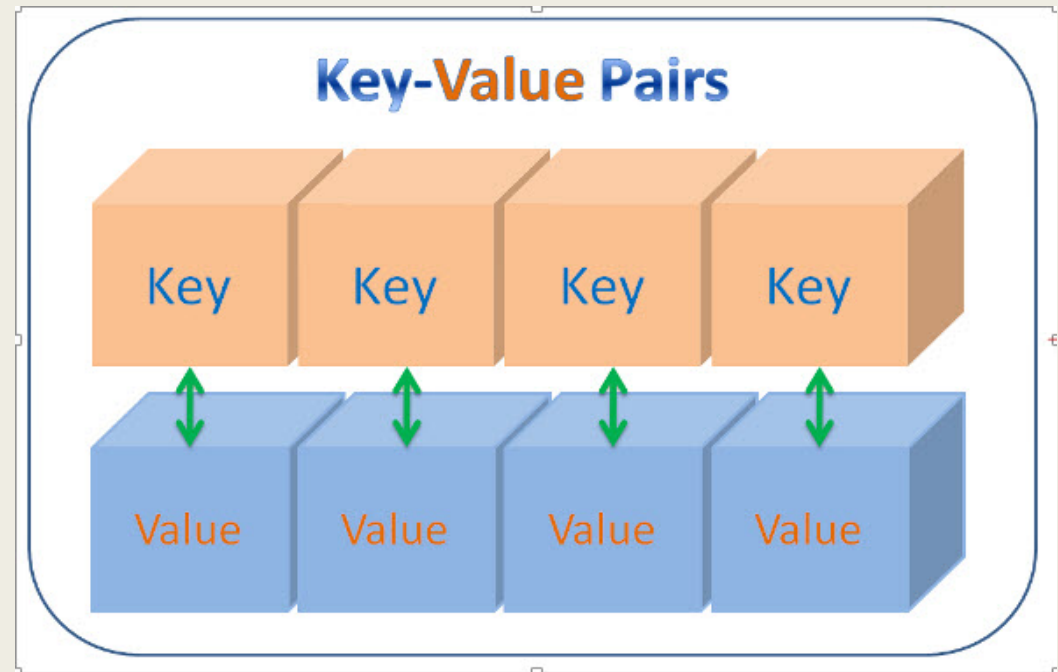
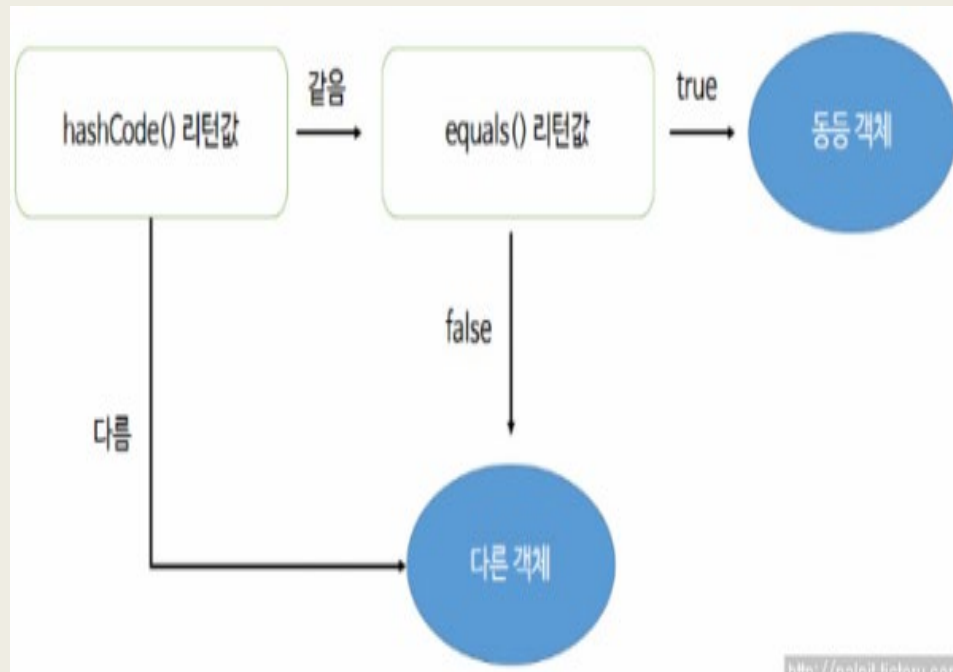
클래스	용도
Object	- 자바 클래스의 최상위 클래스로 사용
System	- 표준 입력 장치로부터 데이터를 입력받을 때 사용 - 표준 출력 장치로부터 출력하기 위해 사용 - 자바 가상 기계를 종료시킬 때 사용 - 쓰레기 수집기를 실행 요청할 때 사용
Class	- 클래스를 메모리로 로딩할 때 사용
String	- 문자열을 저장하고 여러 가지 정보를 얻을 때 사용
StringBuffer, StringBuilder	- 문자열을 저장하고 내부 문자열을 조작할 때 사용
Math	- 수학 함수를 이용할 때 사용
Wrapper Byte, Short,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

<http://palpit.tistory.com>

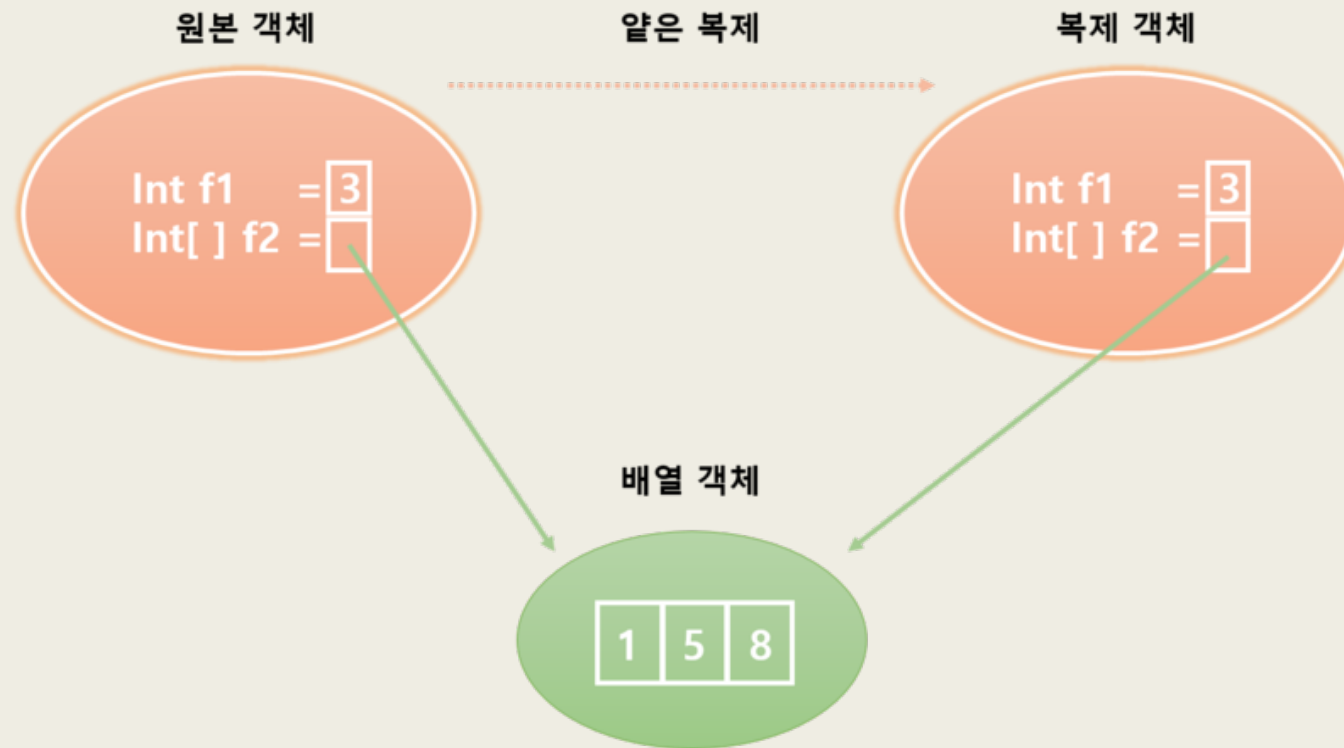
클래스	용도
Arrays	- 배열을 조작(비교, 복사, 정렬, 찾기)할 때 사용
Calendar	- 운영체제의 날짜와 시간을 얻을 때 사용
Date	- 날짜와 시간 정보를 저장하는 클래스
Objects	- 객체 비교, 널(null) 여부 등을 조사할 때 사용
StringTokenizer	- 특정 문자로 구분된 문자열을 뽑아낼 때 사용
Random	- 난수를 얻을 때 사용

<http://palpit.tistory.com>

# HashCode(HashMap)



# Clone (얕은 복제)



```

class Point implements Cloneable //Cloneable 구현
{
    private int xPos;
    private int yPos;

    public Point(int x, int y)
    {
        xPos=x;
        yPos=y;
    }
    public void showPosition()
    {
        System.out.printf("[%d, %d]", xPos, yPos);
    }
    public void changePos(int x, int y)
    {
        xPos=x;
        yPos=y;
    }
    //clone이 protected로 선언되었는것을 외부에서 호출하기 위해 public으로 선언
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}

```

```

class Rectangle implements Cloneable //Cloneable 구현
{
    Point upperLeft, lowerRight;

    public Rectangle(int x1, int y1, int x2, int y2)
    {
        upperLeft=new Point(x1, y1);
        lowerRight=new Point(x2, y2);
    }

    public void changePos(int x1, int y1, int x2, int y2)
    {
        upperLeft.changePos(x1, y1);
        lowerRight.changePos(x2, y2);
    }
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }

    public void showPosition()
    {
        System.out.println("좌상좌하 좌상좌하...");
        System.out.print("좌 상좌: ");
        upperLeft.showPosition();
        System.out.println("");
        System.out.print("우 하좌: ");
        lowerRight.showPosition();
        System.out.println("\n");
    }
}

```

```

class study
{
    public static void main(String[] args)
    {
        Rectangle org=new Rectangle(1, 1, 9, 9);
        Rectangle cpy;

        try
        {
            cpy=(Rectangle)org.clone();
            org.changePos(2, 2, 7, 7);
            org.showPosition();
            cpy.showPosition();
        }
        catch(CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
    }
}

```

직사각형 위치정보...

좌 상단: [2, 2]

우 하단: [7, 7]

직사각형 위치정보...

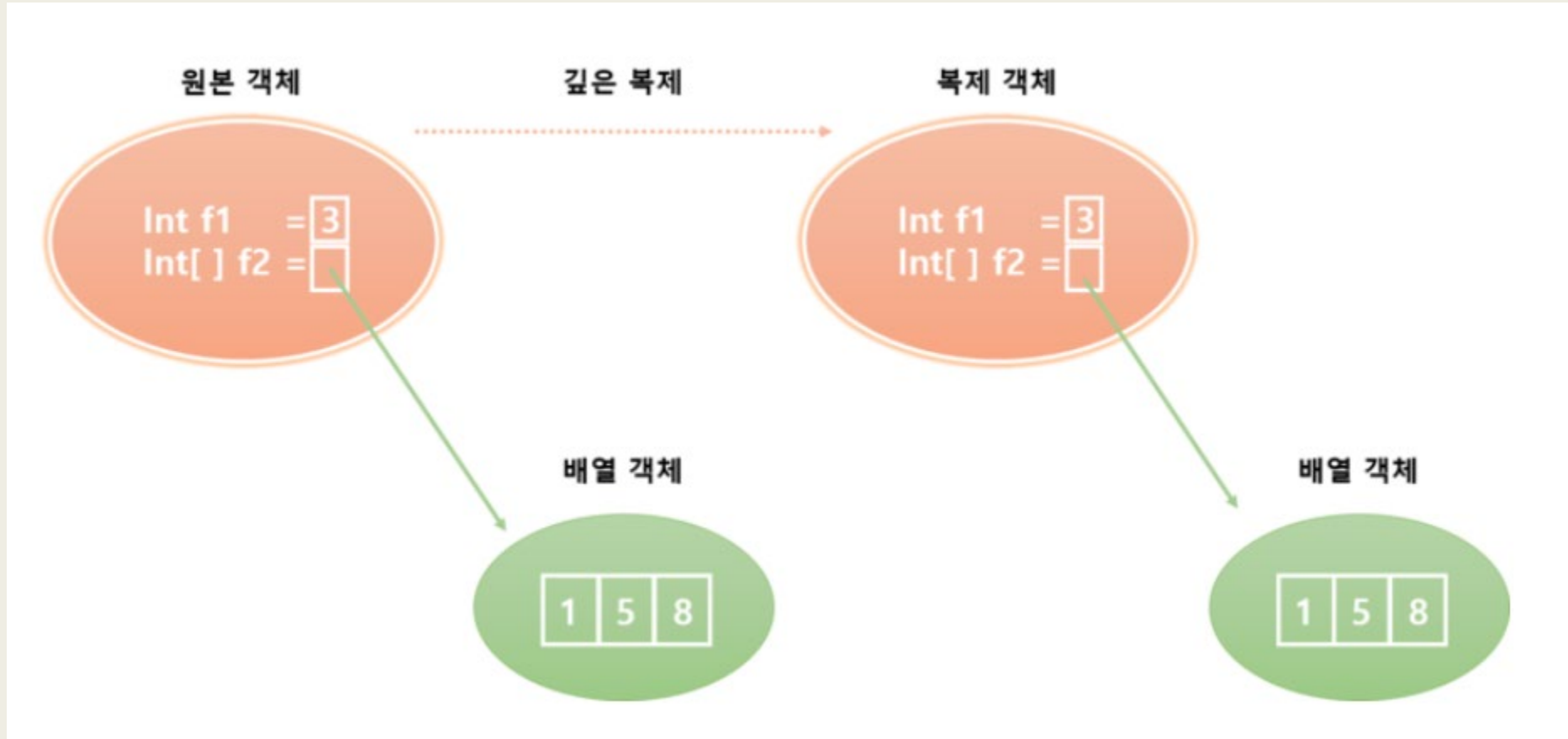
좌 상단: [2, 2]

우 하단: [7, 7]

# Clone (얕은 복제)

- `java.lang.cloneable` 인터페이스 구현
- 참조 타입 필드는 번지만 복제되기 때문에 원본 객체의 필드와 복제 객체의 필드는 같은 객체를 참조
- 만약 복제 객체에서 참조 객체를 변경하면 원본 객체도 변경된 객체를 가지게 된다.
- 이것이 얕은 복제의 단점

# Clone 깊은 복제





```

class Rectangle implements Cloneable //Cloneable 구현
{
    Point upperLeft, lowerRight;

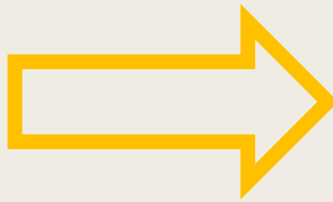
    public Rectangle(int x1, int y1, int x2, int y2)
    {
        upperLeft=new Point(x1, y1);
        lowerRight=new Point(x2, y2);
    }

    public void changePos(int x1, int y1, int x2, int y2)
    {
        upperLeft.changePos(x1, y1);
        lowerRight.changePos(x2, y2);
    }

    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }

    public void showPosition()
    {
        System.out.println("직사각형 위치정보...");
        System.out.print("좌 상단: ");
        upperLeft.showPosition();
        System.out.println("");
        System.out.print("우 하단: ");
        lowerRight.showPosition();
        System.out.println("\n");
    }
}

```



```

public Object clone() throws CloneNotSupportedException
{
    Rectangle copy=(Rectangle)super.clone();
    copy.upperLeft=(Point)upperLeft.clone();
    copy.lowerRight=(Point)lowerRight.clone();
    return copy;
}

```

#### - 실행결과

```

직사각형 위치정보...
좌 상단: [2, 2]
우 하단: [7, 7]

```

```

직사각형 위치정보...
좌 상단: [1, 1]
우 하단: [9, 9]

```

# Clone 깊은 복제

- 참조하고 있는 객체도 복제하는 것을 말한다.
- 다음 그림은 원본 객체를 깊은 복제 했을 경우 참조하는 배열 객체도 복제된다는 것을 볼 수 있다. (새로운 번지 수가 만들어짐)
- 참고
- <https://m.blog.naver.com/PostView.nhn?blogId=yds9211&logNo=220088069668&proxyReferer=https%3A%2F%2Fwww.google.com%2F>
- <https://m.blog.naver.com/PostView.nhn?blogId=mals93&logNo=220722358438&proxyReferer=https%3A%2F%2Fwww.google.com%2F>

# 객체 비교

```
1  import java.util.*;
2  public class GardenTool implements Comparable<GardenTool> {
3      String sku;
4      String description;
5      public GardenTool() {}
6      public GardenTool (String sku, String description) {
7          this.sku=sku;
8          this.description=description;
9      }
10     public String toString() {
11         return "SKU " + sku + " description: " + description;
12     }
13     public int compareTo (GardenTool gardenTool) {
14         return this.sku.compareTo(gardenTool.sku);
15     }
16     public boolean equals(Object obj) {
17         if (obj instanceof GardenTool ) {
18             GardenTool gardenTool=(GardenTool) obj;
19             return this.sku==gardenTool.sku;
20         }
21         else {
22             throw new IllegalArgumentException("Expected GardenTool");
23         }
24     }
25 }
```

## 참고

<https://www.webucator.com/how-to/how-use-comparable-comparator-java.cfm>

## \*제네릭(Generic)

클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법을 의미한다.

## 참고

<https://opentutorials.org/module/516/6237>

# 동등비교

- `equal()`
- `deepEquals()`
- `Objects.deepEquals(Object a, Object b)`는 두 객체의 동등을 비교하는데, `a`와 `b`가 서로 다른 배열일 경우(주소), 항목 값까지도 모두 같다면 `true`를 리턴

a	b	Objects.deepEquals(a , b)
not null (not array)	not null (not array)	<code>a.equals(b)</code>
not null (array)	not null (array)	<code>Arrays.deepEquals(a, b)</code>
not null	null	false
null	not null	false
null	null	true

<http://palpit.tistory.com>

# copyOf()

```
int[] arr1 = {1, 2, 3, 4, 5};  
① int[] arr2 = Arrays.copyOf(arr1, 3);  
  
for (int i = 0; i < arr2.length; i++) {  
    System.out.print(arr2[i] + " ");  
}  
  
② int[] arr3 = Arrays.copyOf(arr1, 10);  
for (int i = 0; i < arr3.length; i++) {  
    System.out.print(arr3[i] + " ");  
}
```

## 실행 결과

1 2 3

1 2 3 4 5 0 0 0 0 0

## 참고

[http://tcpschool.com/java/java\\_api\\_arrays](http://tcpschool.com/java/java_api_arrays)

# 람다식

(Long val1, String val2) -> val1 + val2.length()

↑  
파라미터

↑  
결과 값

참고

<https://www.slideshare.net/madvirus/8-35205661>

# 동적 객체 생성

- 참고
- <http://blog.naver.com/PostView.nhn?blogId=ngomteng&logNo=120126001931>

# String build

## ■ Delete 기능

```
package com.tutorialspoint;

import java.lang.*;

public class StringBuilderDemo {

    public static void main(String[] args) {

        StringBuilder str = new StringBuilder("Java lang package");
        System.out.println("string = " + str);

        // deleting characters from index 4 to index 9
        str.delete(4, 9);
        System.out.println("After deletion = " + str);
    }
}
```

[Live Demo](#)

## 참고

[https://www.tutorialspoint.com/java/lang/stringbuilder\\_delete.htm](https://www.tutorialspoint.com/java/lang/stringbuilder_delete.htm)

string = Java lang package

After deletion = Java package



# 정규 표현식

- <https://regexr.com/>

# Wrapper(포장) 클래스

- 자바는 기본 타입의 값을 갖는 객체를 생성할 수 있다.
- 이런 객체를 포장 객체라고 하는데, 그 이유는 기본타입의 값을 내부에 두고 포장하기 때문
- 포장 객체의 특징은 포장하고 있는 기본 타입 값은 외부에서 변경할 수 없다.
- 만약 내부의 값을 변경하고 싶다면 새로운 포장 객체를 만들어야 한다.

# Java.text

- **Format 클래스**
- **숫자형식 클래스(DecimalFormat)**
- **날짜 형식 클래스(SimpleDateFormat)**
- **문자열 형식 클래스(MessageFormat)**

# Java.time

- 날짜와 시각 객체
- 시스템의 날짜를 표현하는 클래스 -> Date \*생성자 Date()로만 권장
- 시스템의 달력을 표현한 클래스 -> Calendar \*abstract class!!! (new 연산자 불가)
- 로컬날짜클래스 -> LocalDate
- 로컬시간클래스 -> LocalTime
- 로컬날짜 및 시간클래스(LocalDate + LocalTime) -> LocalDateTime
- 특정타임존(TimeZone)의 날짜 시간 클래스 -> ZoneDateTime
- 특정 시점의 Time-Stamp클래스 -> Instant