14. API III

1. Scanner

Scanner클래스는 파일 및 입력, 문자열과 사용자 입력을 가져오는데 사용되는 클래스이며

정수, 실수, 문자열을 읽어올 수 있다.

Scanner 클래스의 메서드	기능
next()	문자열 입력 (띄어쓰기 사용불가)
nextLine()	문장 입력(띄어쓰기 가능)
nextByte()	byte타입의 자료형 값 입력
nextInt()	int 타입의 자료형 값 입력
nextBoolean()	boolean 타입의 자료형 값 입력
nextShort()	short 타입의 자료형 값 입력
nextLong()	long 타입의 자료형 값 입력
nextFloat()	float 타입의 자료형 값 입력
nextDouble()	double 타입의 자료형 값 입력

1) Scanner 주의사항 nextLine();

위 메소드 중 nextLine()은 사용시 주의해야할 특징이 있다. 다른 next(), nextInt() 등의 메소드는 띄어쓰기를 기준으로 입력 값들을 읽고, nextLine() 경우 한 라인을 기준으로 입력 값들을 읽는 특징이 있는데

이때 다른 next()같은 메소드를 입력한 후 Enter를 입력하는 경우, next()계열 메소드는 Enter의

줄바꿈 문자('₩n)를 처리하지 않고 버퍼(buffer)에 남겨 둔다.

이 줄바꿈 문자를 다음 라인에서 nextLine()이 인식하는 순간, 문자('₩n)만 처리되고 바로 종료된다.

Ex)

```java
import java.util.Scanner;

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
        // scanner.() 매소드 실행시 키보드의 입력값을 저장하는데
        // 여기서 사용자의 'Enter'값은 '\n'로 저장된다.
    System.out.print("나이를 입력하세요");
    int age = scanner.nextInt();        //buffer에 저장된 '\n(엔터)'앞의 숫자만 가져옴
    System.out.println("입력하신 나이는 " + age);

    System.out.print("이름을 입력하세요");
    String name = scanner.next();       //whitespace 앞까지만 인식함 // 공란 앞의 문자만 가져옴
    System.out.println("입력하신 이름은 : "+ name);    //앞에 있는 '\n'는 무시하고 이름만 가져옴

    System.out.print("주소를 입력하세요");
    scanner.nextLine();
  // nextLine함수는 '\n'앞에 있는 값을 return하기 때문에 로직 앞에 다른 nextInt나 next등 next함수가
  // 실행되었었다면 위처럼 버퍼에 남아 있는'\n' 제거하기위해 nextLine함수를 한번 실행해 준다.

    String address = scanner.nextLine();
  // '\n' 뒤 부터는 버퍼를 지운다 whitespace상관없이 '\n(엔터)'전까지 리턴해줌
    System.out.println("입력하신 주소는 : "+ address);

    scanner.close();
```

* 버퍼(Buffer)

버퍼란 임시 저장 공간을 의미하는데 정확히 말하면 A와 B가 서로 입출력을 수행하는데 있어서 속도차이를 극복하기 위해 사용하는 임시 저장 공간을 의미한다. 프로그래밍이나 운영체제에서 사용하는 버퍼는 거의 대부분 CPU와 보조기억장치 사이에서 사용되는 임시 저장 공간을 의미한다

2) Scanner 주의사항 close();

사용한 스캐너 함수는 close()함수를 이용, 닫아주는 습관을 들어야 하는데 이때 메소드에서 close()를 실행하면 안된다. 스캐너 함수 사용시에는 같은 버퍼를 공유하고 있기 때문에 특정 매소드에서 close를 하면 해당 메소드를 사용했을시 같은 로직에서의 정의한 스캐너 함수를 사용할 수 없다.

때문에 close()는 메인 메소드에서 실행 하는걸 지향한다.

Ex)

```java
import java.util.Scanner;

public class Ex03_scannerClose {
    public static void main(String[] args) {
        Scanner scMain = new Scanner(System.in);    //아래 메소드와 같은 버퍼를 가르킴
        System.out.println("이름을 입력하세요");
        String name = scMain.nextLine();
        System.out.println("입력하신 이름은 "+name);

        nickName();

        System.out.println("나이 ?");
        int age = scMain.nextInt();     //스캐너를 사용한 매소드에서 버퍼를 close했을시  에러남
        System.out.println("입력하신 나이는 "+ age);
        scMain.close();             //버퍼를 닫는건 메인에서만 하는것을 지향한다.
    }

    private static void nickName(){     //사용자로부터 별명을 출력하는 메소드 정의
        Scanner scNickName = new Scanner(System.in);    //위의 스캐너와 같은 버퍼를 가르킴
        System.out.print("별명은");
        System.out.println("입력하신 별명은 "+scNickName.nextLine());
//      scNickName.close();//scNickName 객체를 null로 만들고 버퍼를 봉쇄한다.

    }
```

2. Wapper

특정 프로그램에 따라서 기본 타입의 데이터를 객체로 취급해야 하는 경우가 있다

예를 들어, 메소드의 매개변수로 객체 타입만이 요구되면, 기본 타입의 데이터를 그대로 사용할 수는 없고 . 기본 타입의 데이터를 먼저 객체로 변환한 후 작업을 수행해야 해줘야한다.

이렇게 8개의 기본 데이터 타입에 해당하는 자료형 타입을 객체 데이터 타입으로 포장(wapper)해 주는 클래스를 래퍼(wapper) 클래스라고 한다. 래퍼 클래스는 각각의 타입에 해당하는 데이터를 인수로 전달받아, 해당 값을 가지는 객체로 변환해 준다.

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

Ex)

```java
public static void main(String[] args) {
    double i = 10.1;        // 기초 자료형으로 선언
    double j = 10.1;
    double sum = i + j;
    System.out.println("합은 " + sum);   // 20.2
    System.out.println(i == j ? "두 int는 같다 " : "두 int는 다르다");  //두 int는 같다.

    Double iObj = new Double(10.1); //기초 타입에서 wapper 클래스의 Double 객체 데이터 타입으로 선언
    Double jObj = 10.1;             // new명령어를 통해 생성하지 않아도 객체를 스스로 생생해서 대입함
//  sum = iObj.intValue() + jObj.intValue(); // 객체의 값을 가져오는 .intValue() 함수
    sum = iObj + jObj;          // 함수를 안써도 안에 있는 값을 알아서 가져옴

    System.out.println("합은 " + sum);    //20.2
    System.out.println(iObj.equals(j) ? "두 wrapper객체 값은 같다." : "두 wrapper 객체 값은 다르다.");
                                        //두 wrapper객채 값은 같다.
    System.out.println(iObj == jObj ? "주소가 같다." : "주소가 다르다.");
                                        //주소가 다르다.   객체화 됨

    System.out.println("String을 정수로 바꾸는 함수 : Integer.parseInt(문자열)");
    int i = Integer.parseInt("10"); //.parseInt() 매소드를 통해 인자값을 int 정수로 바꿔준다.
    System.out.println(i);      // 10

    System.out.println("정수를 String로 바꾸는 함수 : String.valueOf(숫자)");
    // String monthStr =""+ 12;    //빈String + 숫자 =>빈스트링으로 인해 숫자가 String으로 변환되어 대입
    String monthStr = String.valueOf(12);   // .valueOf()매소드의 매개변수 값을 String으로 변환
    System.out.println(monthStr);   //"12"

    }
}
```

3. Timer

Timer클래스는 로직의 실행을 일정시간을 주기로, 반복적으로 수행할 수 있게 해주는 클래스이다.

이때 사용자는 Timer클래스와 TimerTask클래스를 사용하여야 하는데 여기서

TimerTask 클래스는 Timer클래스가 수행해야 할 내용을 작성하는 클래스이고

(TimerTask 클래스내의 run()메소드를 오버라이딩 하여 작성 한다.)

Timer클래스는 TimerTask에서 정의한 기능을 수행하는 클래스이다. (run()에서 정의한 내용이 실행된다.)

TimerTask 클래스는 상위 클래스로부터 강제 구현할 메소드가 있는 abstract 클래스이기 때문에

abstract run()메소드를 반드시 오버라이드 하여 사용해야 한다.

Ex)TimerTask 정의

```java
import java.util.TimerTask;

public class TimerTaskEx1 extends TimerTask {
    @Override   //TimerTask를 상속받아 run()을 구현함
    public void run() {     //호출되면 실행되는 로직
        System.out.println("작업1 : 2초후에 한번 수행될 예정인 TimerTask @@");
    }
}
```

```java
import java.util.TimerTask;

public class TimerTaskEx2 extends TimerTask {

    @Override
    public void run() {
        System.out.println("작업 2: 1초후에 0.5초 마다 계속 수행될 예정인 TimerTask ★★★");
    }
}
```

Ex)실행 메인 클래스

```java
import java.util.Timer;
import java.util.TimerTask;
public class TimerTaskMain {
    public static void main(String[] args) throws InterruptedException {

        System.out.println("시작");
        Timer timer = new Timer(true);
            //Timer객체의 매개변수로 true를 넣어준다. //true로 넣어야 프로그램이 실행된후 Timer 종료
        TimerTask task1 = new TimerTaskEx1();   //정의해놓은 TimerTaskEx1 객체생성
        TimerTask task2 = new TimerTaskEx2();   //정의해놓은 TimerTaskEx2 객체생성

        // 객체명.schedule(실행할 로직의 객체, 딜레이 타임(밀리세컨형식),  필요에 따라 반복 시간)
        timer.schedule(task1, 2000);//2초 후에 task1의 run()매소드 수행
        timer.schedule(task2, 1000, 500);//1초후 0.5 초마다 task2.run 실행

        //3초동안 잠깐 멈춤 로직을 넣어야한다 // timer보다 프로그램이 먼저 종료되기때문
        Thread.sleep(7000);//스레드 하나 생성해서 3초 멈춤

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