[12-14] Java

1) public class 클래스명 { }

- public static void main(String [ ] args){ }
  - 메소드 제어자중 한개는 반드시 필요
  - static : 객체 생성없이 사용 가능
  - void : 리턴하지 않고 현재 값에서 끝내겠다.
  - main : 현재의 오브젝트 이름
  - main( ) : ( ) 는 파라미터
  - String [ ] : 문자를 배열로 받겠다.
  - args : 첫줄에서는 무의미한 편, 일종의 변수 명
  - { } : Method block, 실행할 명령어를 적을 공간

- 메소드 접근자, 제어자
  - public : 내가 만든 클래스 이외에 다른 곳에서 호출할 수 있도록
  - private : 현재의 클래스 안에서만 사용
  - protected :

- Static
  - 메모리에 고정적으로 할당
    - Static이 붙지 않은 메서드나 변수의 경우 객체가 생성될 때마다 호출되어 서로 다른 값을 가지고 있을 수 있습니다. 그렇기 때문에 각 객체들에서 공통적으로 하나의 값이 유지되어야 할 경우 static을 유용하게 사용
  - 객체 생성없이 사용 가능
    - static 키워드를 붙이면 객체 생성 없이도 메서드나 변수를 사용 가능
  - 프로그램이 시작되면 메모리의 static 영역에 적재되고, 프로그램이 종료될 때 해제
    - 프로그램이 시작되어 클래스가 메모리에 올라가게 되면 static이 붙은 변수나 메서드는 클래스와 함께 자동으로 메모리의 static 영역에 생성
    - 일반적인 메서드는 객체를 생성하면 메모리의 Heap 영역에 올라가게 됩니다. 메모리의 이 Heap 영역은 Garbage Collector에 의해 자동으로 관리되게 됩니다. 즉 사용하지 않는 객체의 경우 알아서 삭제시킴으로써 메모리를 관리해주나 static 메서드는 static 영역에 존재하기 때문에 이러한 관리를 받지 못함
    - 프로그램이 종료될 때 메모리를 해제하게 되는데, 이 때문에 과도하게 많은 static을 선언할 경우 메모리에 과부하가 올 수 있음

- String[] args 존재 이유 / public static void main(String[] args)는 무슨 뜻인가요?
  - String [ ] args 는 현재 main( ) 안에서의 파라미터
  - 자바의 String [ ] args 는 자바스크립트의 let args = [ ] 와 동일한 형태
  - 첫 클래스가 시행되고 첫 오브젝트(main)는 무리 없이 실행됨

- 퍼블릭으로 시작하는 것이 객체지향
  - 객체지향 : 내가 만든 곳에서만 사용되는 것이 아니라 다른 곳에서 동일하게 적용

- import java.util.*; : 외부 파일 호출

- 구글 v8 엔진 , 2012년도부터 자바스크립트가 활성, 익스플로러보다 3~4배 증가, 자바기반

2) Java의 장점 : C 언어의 개선된 기능으로 등장한 것이 JAVA

- Garbage Collection : 불필요해진 메모리가 발생시 자동으로 정리
  - 프로그램을 개발 하다 보면 유효하지 않은 메모리인 가비지(Garbage)가 발생
  - JVM의 가비지 컬렉터가 불필요한 메모리를 알아서 정리, 반응속도가 다소 느려짐
  - 대신 Java에서 명시적으로 불필요한 데이터를 표현하기 위해서 일반적으로 null 선언

- JIT(just in time) : 해당하는 파일들을 미리 상주하다가 호출시 즉시 사용
  - Java 코드는 JavaCompiler를 통해 ByteCode로 변환이 된다. 그리고 ByteCode는 다시 기계어로 번역이 된다. 이 과정이 있기 때문에 Java는 속도면에서 느린 편
  - JIT 방식은 실행 시점에 자주 쓸만한 코드들을 기계어로 변환 시켜놓고 저장해 뒀다가, 재사용 할 때 이미 변환된 기계어 코드를 재 사용 하는 방식을 말한다.
  - 물론 이 과정을 하기 위해 초반에 메모리를 잡아두거나 하는 선행 작업들이 있어서 초기 실행 속도는 다소 느릴 수도 있다.
  - 하지만 그 이후로는 ByteCode를 사용 할 때마다 네이티브 코드로 변환하는 작업이 들어 실행속도가 많이 향상된다. 코드가 재사용될 일이 없거나 규모가 작은 프로그램에서는 배 보다 배꼽이 더 클 수도 있지만 일반적으로 빠른 속도를 자랑한다.

- WORA(Write Once, Read Anywhere) :
  - Java 는 JVM 으로 인해 OS에 종속적이지 않다는 특징, 컴파일러가 운영체제마다 의존적이었던 문제를 해결하고자 JAVA(JVM)가 등장한 것이며, JAVA 언어로 작성한 소스파일은 직접 운영체제로 가서 실행하는 것이 아닌, JVM을 거쳐서 운영체제와 상호작용을 하게 된다.
  - 이러한 장점 때문에 자바가 아닌 다른 언어도 클래스 파일만 있다면 JVM을 사용할 수 있게 개발되고 있다. 실제로 Java 외 다른 언어(클로저, 그루비, 코틀린 등)에서도 이 JVM 사용

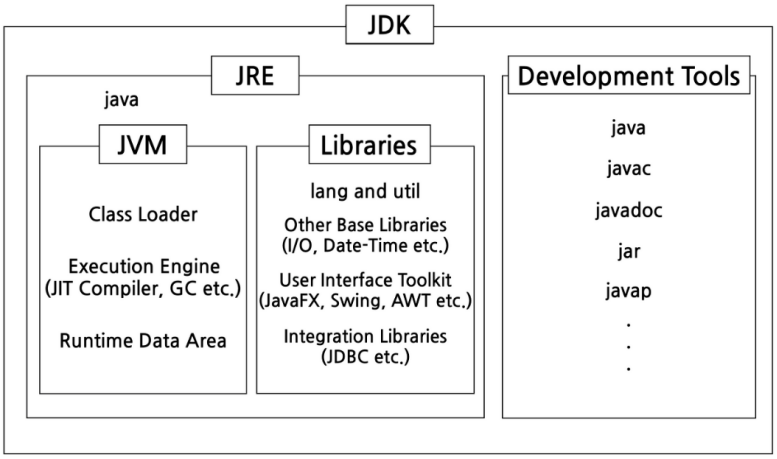
- 그외 내장함수(변수로 사용 불가)

abstract	**continue**	**for**	new	switch
assert	default	goto	package	synchronized
boolean	do	**if**	private	this
break	double	implements	protected	throw
byte	**else**	**import**	public	throws
case	enum	instanceof	**return**	transient
catch	extends	**int**	short	**try**
char	final	interface	static	void
class	**finally**	**long**	strictfp	volatile
const	**float**	native	**super**	**while**

3) Java의 (구조적) 단점

- 자바 프로그램은 일반 프로그램보다 자바 가상 머신이라는 한 단계를 더 거쳐야 하므로, 결국은 상대적으로 실행 속도가 느리다는 단점을 가지고 내포하고 있다.
- 즉, JAVA는 C언어와 달리 두번의 컴파일로 인한 속도의 문제가 발생하는데, 이를 보완하기 위해 JIT 컴파일러 라는내부 프로그램을 사용해서 필요한 부분만을 기계어로 바꾸어 줌으로써 성능 향상을 가져오도록 했지만 그럼에도 C언어의 실행 속도를 따라잡지는 못했다
- JVM을 사용함으로써 얻는 가장 큰 이점은 **JVM**을 사용하면 자바 프로그램을 모든 플랫폼에서 제약 없이 동작하도록 할 수 있다는 장점이 있으나, 단, 간과하지 말아야 할 점은 자바 프로그램과는 달리 자바 가상 머신(JVM)은 운영체제에 종속적이므로, 각 운영체제에 맞는 자바 가상 머신을 설치해야 한다는 점

4) [JDK / JRE / JVM 개념 & 구성 원리]

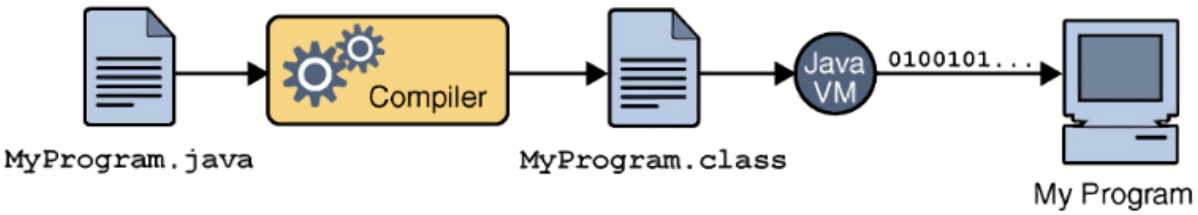


- JDK(Java Development Kit) : JDK안에는 자바를 개발 시 필요한 라이브러리들과 javac, javadoc 등의 개발 도구들을 포함되어 있고, 개발을 하려면 자바 프로그램을 실행도 시켜줘야 하기 때문에 뒤에서 배울 **JRE(Java Runtime Environment)**도 함께 포함되어 있다.

- JRE(Java Runtime Environment) : 컴파일된 Java 프로그램을 실행
    - JVM과 자바 프로그램을 실행(동작)시킬 때 필요한 라이브러리 API를 함께 묶어서 배포되는 패키지

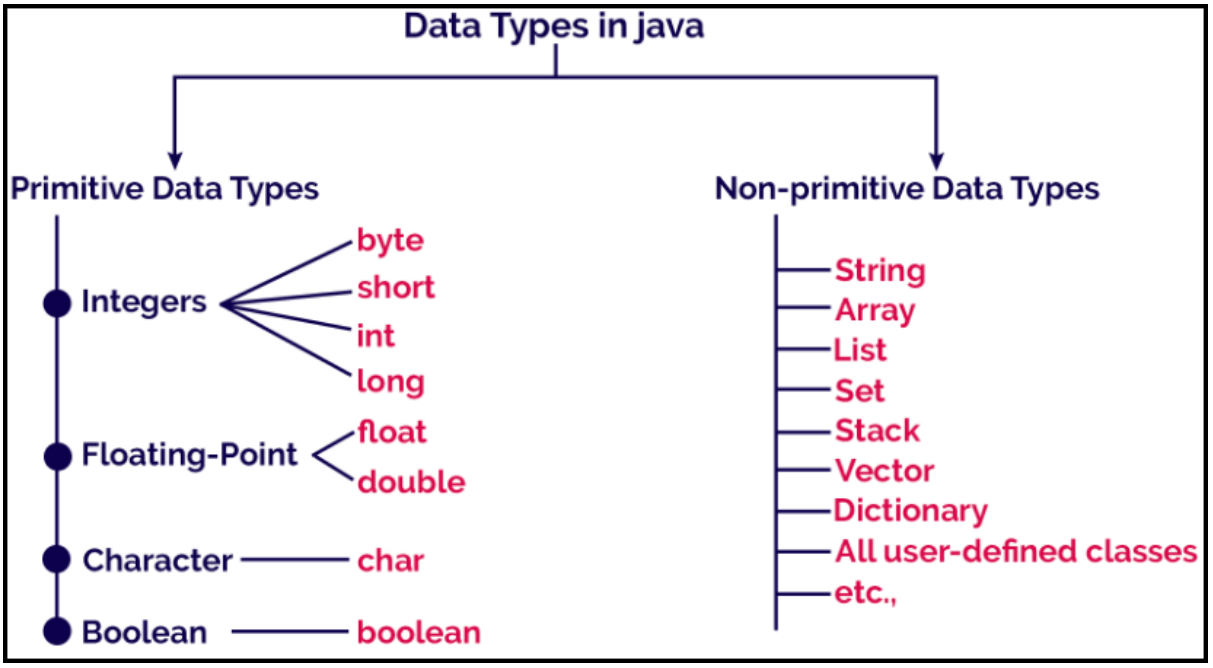
- JVM (Java Virtual Machine) : 자바를 구동하는 프로그램
    - 자바로 작성된 모든 프로그램은 JVM(자바 가상 머신)에서만 실행될 수 있으므로, 자바 프로그램을 실행하기 위해서는 반드시 자바 가상 머신이 설치되어 있어야 한다.
    - 이러한 장점 때문에 자바가 아닌 다른 언어도 클래스 파일만 있다면 JVM을 사용할 수 있게 개발되고 있다. 실제로 Java 외 다른 언어(클로저, 그루비, 코틀린 등)에서도 이 JVM 사용

5) [자바 프로그램의 실행 과정]



- JVM이라는 특성 때문에 자바로 작성한 파일을 실행하기 위해 두 번의 단계가 필요
    - MyProgram.java 파일을 MyProgram.class 파일로 바꾸어 생성해주는 컴파일 단계
    - MyProgram.class 파일을 바이너리 코드(0100101...)로 변환하고 실행하는 단계

6) DataType

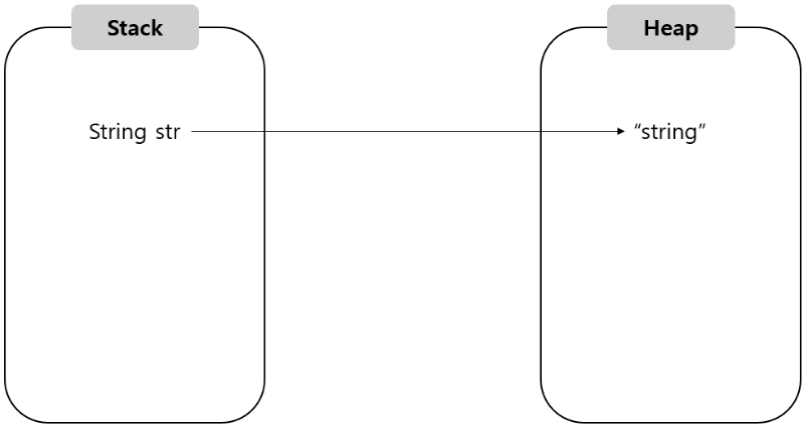


Data type	Meaning	Memory size	Range	Default Value
byte	Whole numbers	1 byte	-128 to +127	0
short	Whole numbers	2 bytes	-32768 to +32767	0
int	Whole numbers	4 bytes	-2,147,483,648 to +2,147,483,647	0
long	Whole numbers	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	0L
float	Fractional numbers	4 bytes	-	0.0f
double	Fractional numbers	8 bytes	-	0.0d
char	Single character	2 bytes	0 to 65535	\u0000
boolean	unsigned char	1 bit	0 or 1	0 (false)

- int(Integer) : 정수, 16bit (2byte), 2^31(약 21억)
- long : 정수, 32bit (4byte), 2^, 최종산출값용도(21억을 넘는 경우)
- float : 실수, 32bit (4byte), 3.4*10^38
- double : 실수, 64bit (8byte), 사용하지 않는 편
- char(character) : 문자, 한글자
- String : 문자열
- boolean : T or F
- List : 배열
- Map : 배열

7) 리터럴(literal)
8) new 연산자

- [Java] String 생성 방식! 리터럴방식(상수풀) vs new 연산자 방식 / [1]
  - new 연산자는 객체를 Heap이라는 메모리 영역에 메모리 공간을 할당해주고 메모리주소를 반환한 후 생성자를 실행시켜준다.
  - 리터럴과는 달리 new 연산자로 생성된 객체는 똑같은 값을 가진(?) 객체가 있어도 서로 다른 메모리를 할당하기 때문에 서로 다른 객체로 분류된다.



Java 의 메모리 상황

  - 일단 변수는 Stack 영역에 할당 될것이고, new String으로 생성된 문자열값이 Heap영역의 메모리 공간을 할당받아 str이라는 변수가 그 메모리주소를 가르키고 있을것이다. 이를 다른 말로 참조라고도 한다.

  - 리터럴 방식은 주소가 동일

```
public class Main {

	public static void main(String[] args) {
		String str = "string";
		String str2 = "string";

		System.out.println(System.identityHashCode(str));	//result:2008362258
		System.out.println(System.identityHashCode(str2));	//result:2008362258
	}

}
```

  - new 연산자 방식은 주소가 다르게 배정

```
public class Main {

	public static void main(String[] args) {
		String str = new String("string");
		String str2 = new String("string");

		System.out.println(System.identityHashCode(str));	//result:2008362258
		System.out.println(System.identityHashCode(str2));	//result:760563749
	//메모리주소는 컴퓨터마다 다르기때문에 실행결과가 다르게 나올수도 있다.
	}

}
```

  - new 연산자의 사용처 : 필요한 객체에 주소를 할당하면서 어떻게 쓸 것인지

[1] https://tcpschool.com/

- int, long, boolean 등은 원시자료는 리터널 방식만 가능, 객체화 불가(New 연산자 불가)

원시 자료형	Wrapper 클래스
int	Integer
long	Long
double	Double
float	Float
boolean	Boolean
char	Character

  - Wrapper 클래스 : 멀티스레드 환경, 다중처리에서 사용
  - 추후 보강

9) 연산 및 실행, 명령어 정리

- / , %

```java
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        System.out.println(7/3);  //2
        System.out.println(3/7);  //0
    }
}
```

```java
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        System.out.println(7%3);  //1
        System.out.println(3%7);  //3
    }
}
```

  - / : (별다른 지정을 하지 않으면) int 로 몫을 반환
  - % : (별다른 지정을 하지 않으면) 는 나머지를 반환, 상품 주문에서 주로 사용

- 증감연산 : 전위법과 후위법의 차이가 있음

```java
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int a = 0;
        System.out.println(a++); //0
        System.out.println(a); //1
    }
}
```

```java
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int a = 0;
        int b = 10;
        System.out.println(a++);  //0
        System.out.println(b++);  //10
        System.out.println(++a);  //2
        System.out.println(++b);  //12
    }
}
```

```java
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int a = 0;
        int b = 10;
        System.out.println(++a);  //1
        System.out.println(++b);  //11
        System.out.println(a++);  //1
        System.out.println(b++);  //11
    }
}
```

  - 연산전에 더할 것인가 연산후에 더할 것인가
  - a++ : a 출력 먼저, 이후에 호출시 ++ 만큼 증감

- 내장된 문자표 출력

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        char a1 = 'a';
        char a2 = 97;
        char a3 = '\u0061';
        System.out.println(a1); //a
        System.out.println(a2); //a
        System.out.println(a3); //a
    }
}
```

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        String a = "hello";
        String b = "java";
        System.out.println(a.equals(b)); //false
    }
}
```

  - equals : if 문을 통해 비밀번호, 상품코드등을 비교해 T/F 로 출력
  - indexof : 주민번호 뒷자리 첫번째 찾을 때 유용
  - contains : 포함된 여부를 T/F로 출력, 이메일주소에 @ 포함여부 등

- format

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        System.out.println(String.format("커피 %d잔째 마시쩡",3));
        //커피 3잔째 마시쩡
    }
}
```

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        int number = 10;
        String set = "잔째";
        System.out.println(String.format("커피 %d%s 마시쩡",number, set));
        //커피 10잔째 마시쩡
    }
}
```

코드	설명
%s	문자열 (String)
%c	문자 1 개 (character)
%d	정수 (Integer)
%f	부동소수 (floating-point)
%o	8 진수
%x	16 진수
%%	Literal % (문자 % 자체)

  - Literal : 당신의 등급은 ~ 입니다.

- %10s : 뒤에 들어오는 String 포함 10개의 공간을 가져라

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        String data = "papa";
        System.out.println(String.format("%10s", data)); //      papa
        System.out.println(data); //papa
    }
}
```

  - 6개의 빈칸 + papa(4칸)

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        String data = "papa";
        System.out.println(String.format("%-10sjin", data)); //papa      jin
        System.out.println(data); //papa
    }
}
```

  -

```java
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        String data = "papa";
        System.out.println(String.format("%-10s%s", data, "jin")); //papa      jin
        System.out.println(data); //papa
    }
}
```

  - %-10s%s
    - %s : 뒤의 파라미터를 적용(jin)

- “%.4f” : 소수점 4자리까지는 출력하되, 반올림으로

```
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        System.out.println(String.format("%.4f", 3.141592)); //3.1416
    }
}
```

- StringBuffer : 문자열을 추가하거나 변경할 때 주로 사용하는 자료형
    - append 를 통한 추가
    - toString : 전체를 모아서 문자열로 변경

```
public class JavaExm12_14_01 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        sb.append("hello");
        sb.append(" ");
        sb.append("go to sky");
        String result = sb.toString();
        System.out.println(result); //hello go to sky
    }
}
```

    - StringBuffer() : () 빈배열을 생성, 대기하므로 메모리 소모가 존재
    - format 은 기존에 있던것에 입력하기에 차이가 있는 편

- boolean

```
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int base = 180;
        int high = 185;
        boolean isTall = high > base;
        System.out.println(isTall); //true
    }
}
```

```
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int base = 180;
        int high = 185;
        boolean isTall = high > base;
        if(isTall){
            System.out.println(isTall); //true
        }
    }
}
```

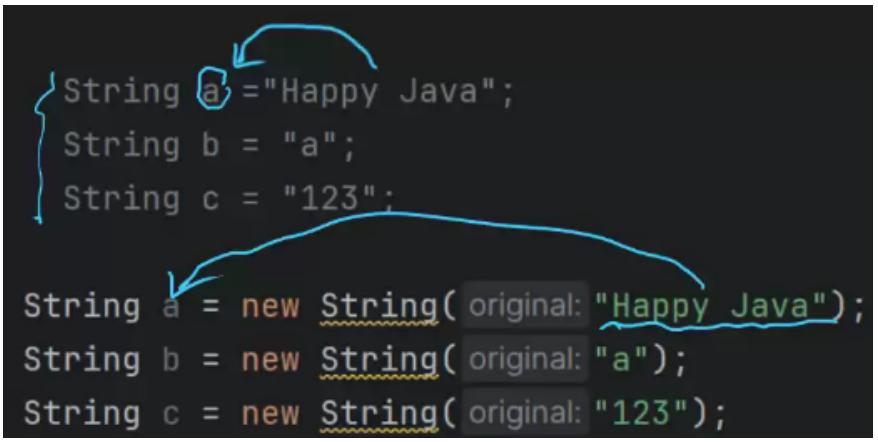
    - true 출력

```
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int base = 180;
        int high = 185;
        boolean isTall = high > base;
        if(isTall){
            System.out.println("크네"); //true
        }else(){
            System.out.println("작네");
        }
    }
}
```

```
public class JavaExm12_14_01 {
    public static void main (String[] args ){
        int base = 180;
        int high = 185;
        boolean isTall = high > base;
        if(isTall){
            System.out.println("크네"); //크네
        }else{
            System.out.println("작네");
        }
    }
}
```

    - () 파라미터는 에러, 파라미터 없으면 ‘크네’ 출력

- new : String a 를 객체 형태로 집어넣겠다



  - 리터럴 방식 : String a = “Happy Java”
    - 단순한 변수, Happy Java 를 문자열로 a 로 담음
    - 다시말해 a는 문자열로 담는 것
  - New 연산자 방식 : String a = New String(“Happy Java”)
    - Happy Java” 는 String 으로 담기고, 해당 String은 a 로 담기지만 Java 는 String 을 찾는 것
    - New 를 쓰면서 String 이라는 하나의 객체로써 저장
    - 앞서 넣은 것은 변수지만, 클래스등으로도 가능
    - 이렇게 넣음으로써 더 많은 것을 저장 가능

- 배열

```java
public class JavaExm12_14_02 {
    public static void main(String[] args){
        int[] odds = {1,3,5,7,9};
        String[] weeks = {"월","화","수","목","금","토","일"};
        String[] week = new String[7];
        week[0] = "월";
        week[1] = "화";
        week[2] = "수";
        week[3] = "목";
        week[4] = "금";
        week[5] = "토";
        week[6] = "일";
        System.out.println(Arrays.toString(weeks)); //[월, 화, 수, 목, 금, 토, 일]
        System.out.println(Arrays.toString(week));  //[월, 화, 수, 목, 금, 토, 일]
    }
}
```

  - int[] odds : odds 변수는 정수형의 배열
  - String[] weeks : weeks 변수는 문자열의 배열
    - 배열의 형식 : dataType[ ] 변수명 = { }
  - weeks : 단순 배열의 문자열
  - week : 배열의 오브젝트화
    - String[7] 과 추가되는 week[N] 이 안맞으면 에러뜸