

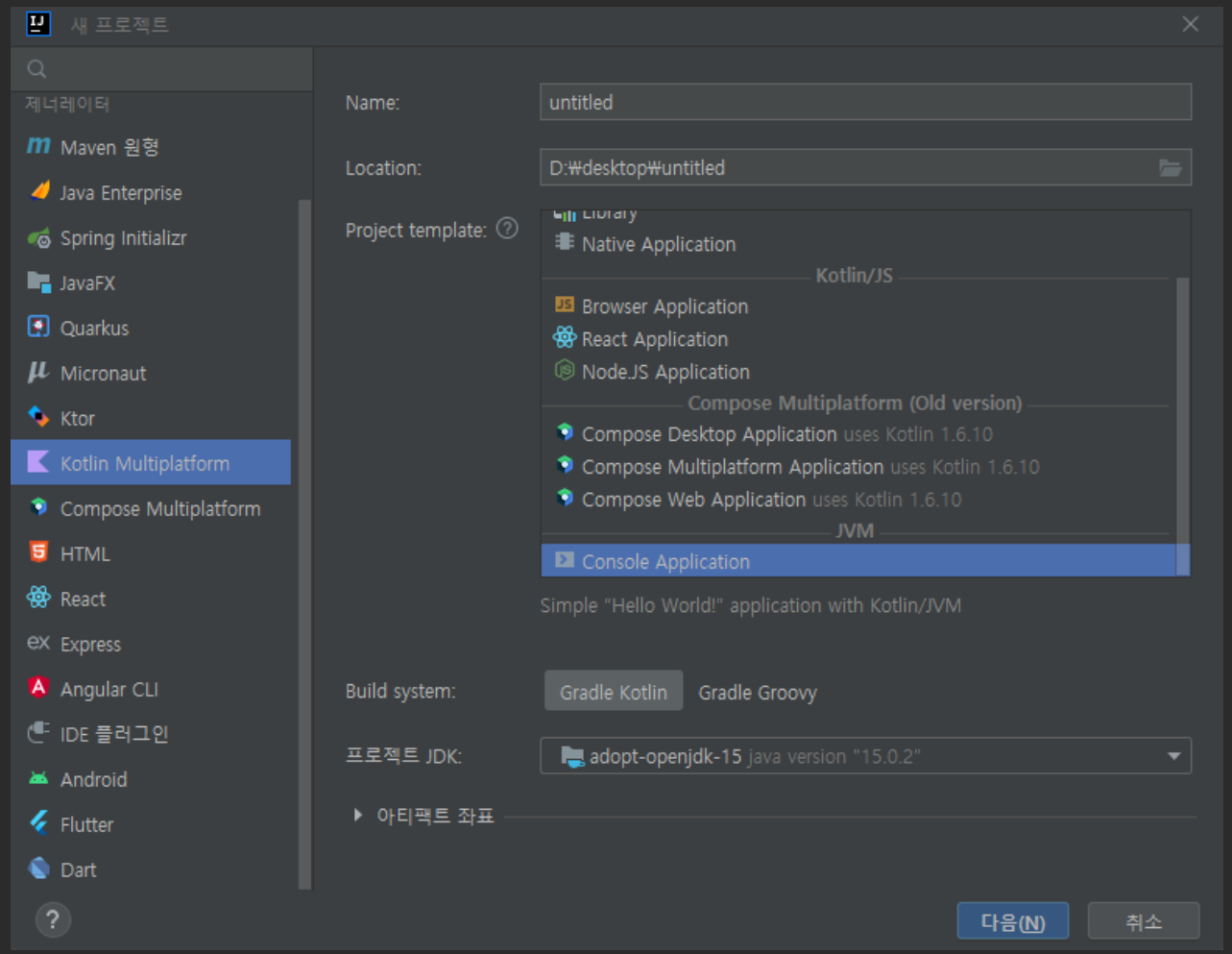
Bsidesoft co.

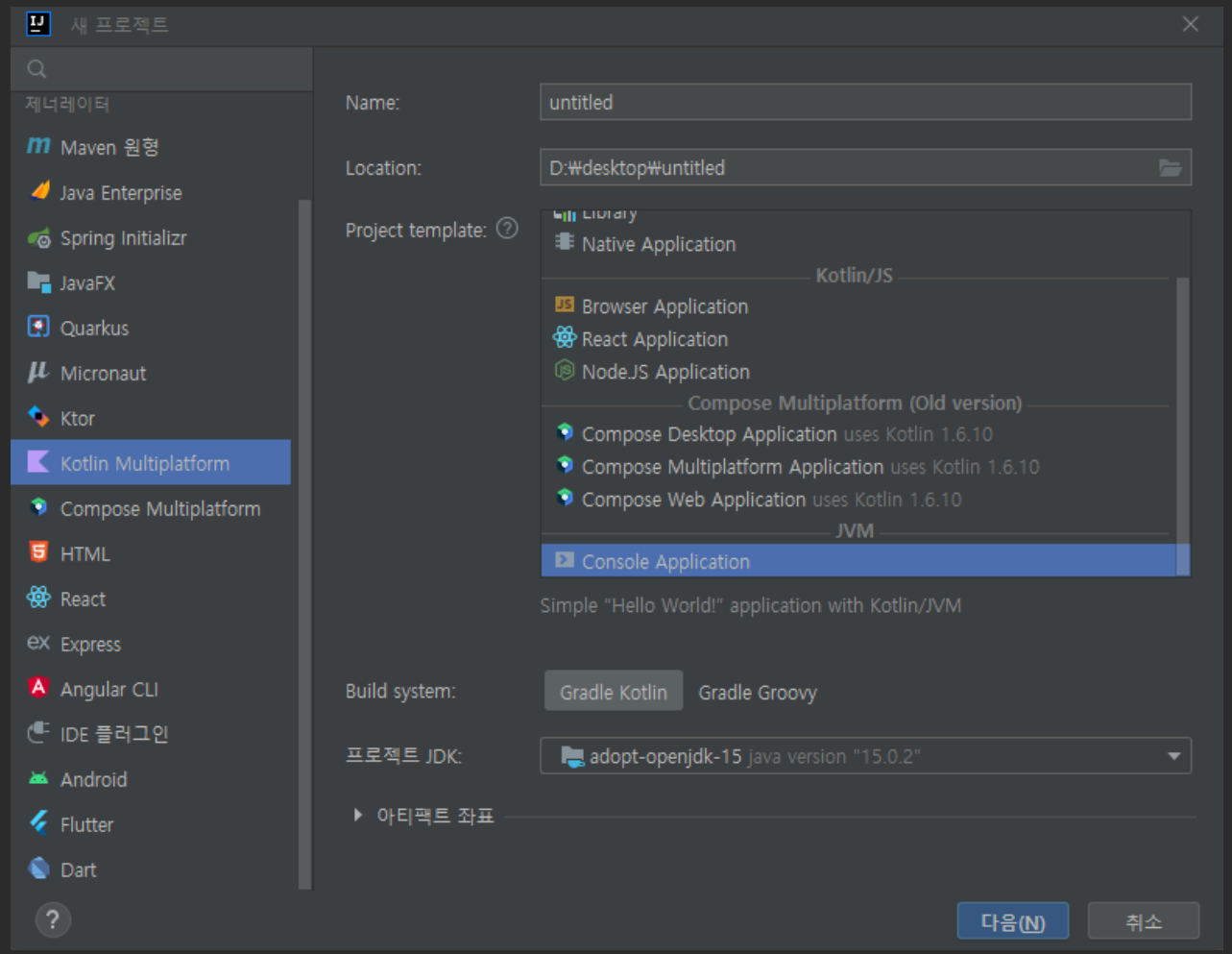
since 2004

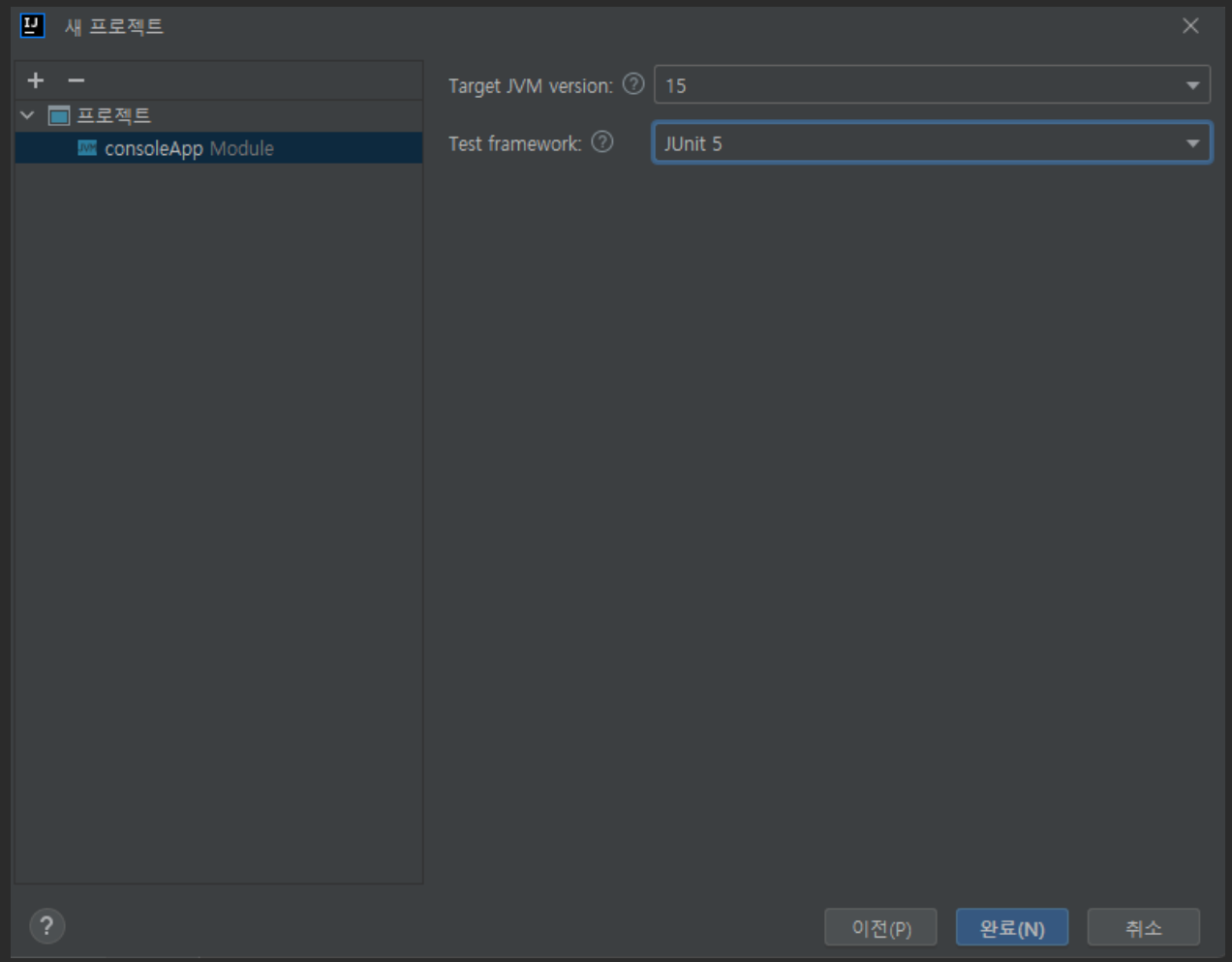
KOTLIN LANGUAGE

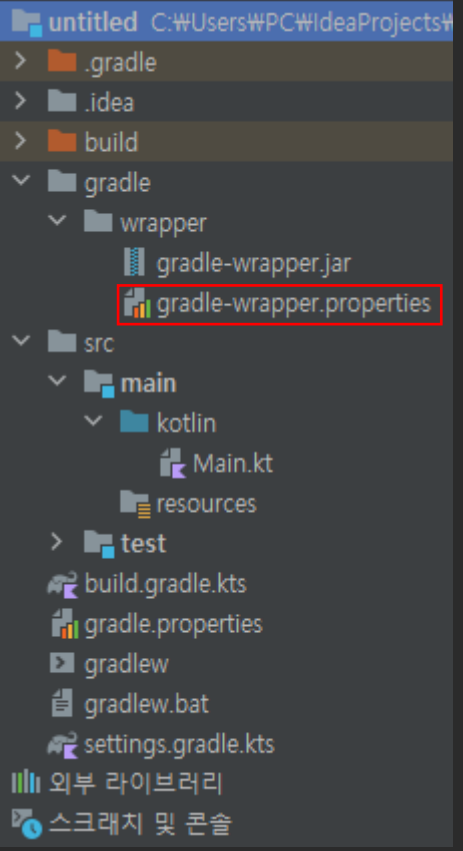
1 2 3 4 5 6

HELLO WORLD


새 프로젝트
제너레이터
Maven 원형
Java Enterprise
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform
Compose Multiplatform
HTML
React
Express
Angular CLI
IDE 플러그인
Android
Flutter
Dart
Name:
untitled
Location:
D:₩desktop₩untitled
Project template:
Native Application
Kotlin/JS
Browser Application
React Application
Node.JS Application
Compose Multiplatform (Old version)
Compose Desktop Application uses Kotlin 1.6.10
Compose Multiplatform Application uses Kotlin 1.6.10
Compose Web Application uses Kotlin 1.6.10
JVM
Console Application
Simple "Hello World!" application with Kotlin/JVM
Build system:
Gradle Kotlin
Gradle Groovy
프로젝트 JDK:
adopt-openjdk-15 java version "15.0.2"
아티팩트 좌표
다음(N)
취소

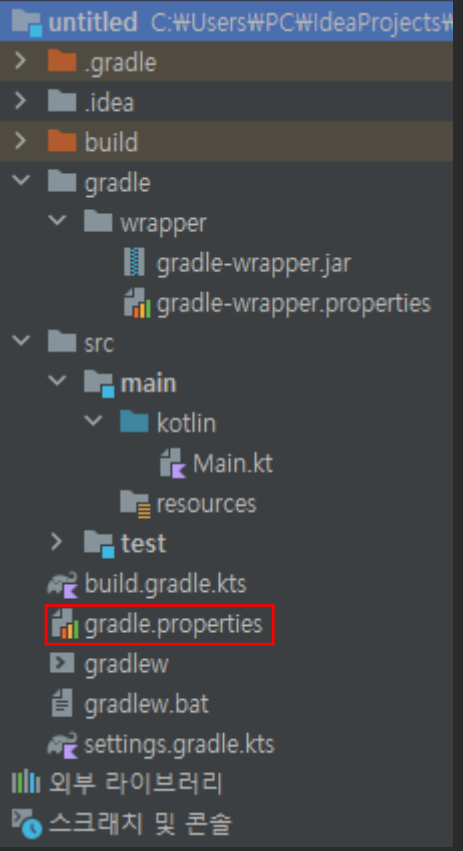

새 프로젝트
제너레이터
Maven 원형
Java Enterprise
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform
Compose Multiplatform
HTML
React
Express
Angular CLI
IDE 플러그인
Android
Flutter
Dart
Name:
untitled
Location:
D:₩desktop₩untitled
Project template:
Native Application
Kotlin/JS
Browser Application
React Application
NodeJS Application
Compose Multiplatform (Old version)
Compose Desktop Application uses Kotlin 1.6.10
Compose Multiplatform Application uses Kotlin 1.6.10
Compose Web Application uses Kotlin 1.6.10
JVM
Console Application
Simple "Hello World!" application with Kotlin/JVM
Build system:
Gradle Kotlin
Gradle Groovy
프로젝트 JDK:
adopt-openjdk-15 java version "15.0.2"
아티팩트 좌표
다음(N)
취소


새 프로젝트
프로젝트
consoleApp Module
Target JVM version:
15
Test framework:
JUnit 5
이전(P)
완료(N)
취소




```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-7.3.3-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

https://docs.gradle.org/



kotlin.code.style=official

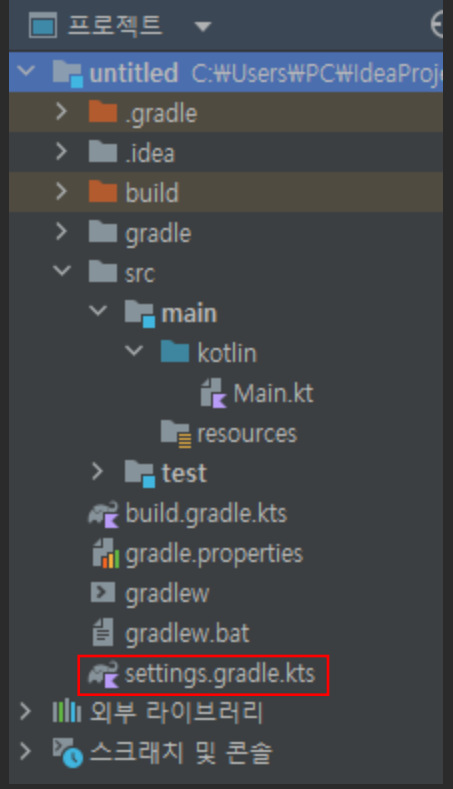
사용자 정의 속성관련
https://docs.gradle.org/current/userguide/organizing_gradle_projects.html
#declare_properties_in_gradle_properties_file

그래이들 설정 관련
https://docs.gradle.org/current/userguide/build_environment.html
#sec:gradle_configuration_properties

코틀린 설정 관련
https://kotlinlang.org/docs/gradle.html

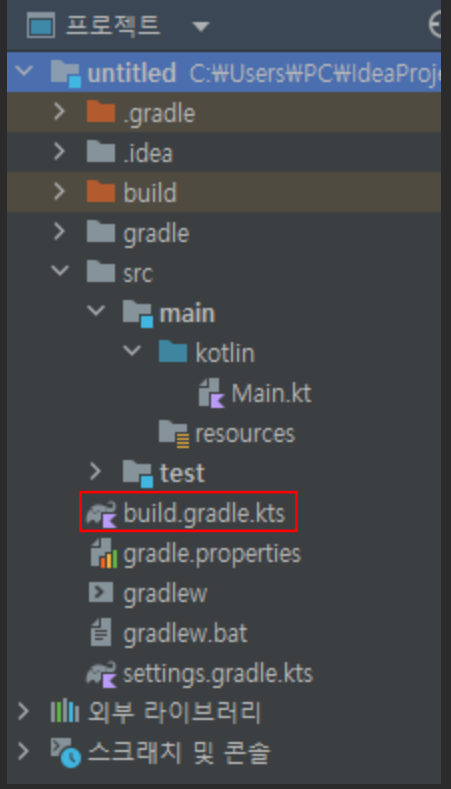
kapt 설정 관련
https://kotlinlang.org/docs/reference/kapt.html

안드로이드 설정 관련
https://developer.android.com/jetpack/androidx
https://developer.android.com/topic/libraries/data-binding/start
https://developer.android.com/studio/build/optimize-your-build
https://android.googlesource.com/platform/tools/base/+/mirror-goog-studio-master-dev/
build-system/gradle-core/src/main/java/com/android/build/gradle/options/BooleanOption.kt

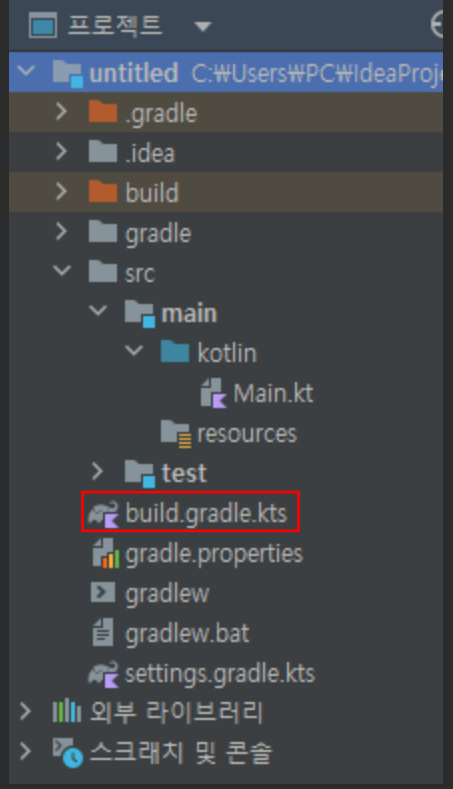


rootProject.*name* = "untitled"

https://docs.gradle.org/current/dsl/org.gradle.api.initialization.Settings.html

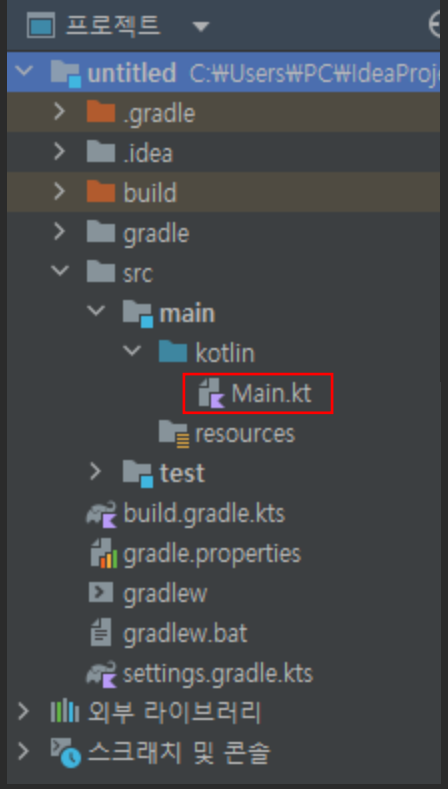



```kotlin
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
    kotlin("jvm") version "1.6.21"
    application
}

group = "me.pc"
version = "1.0-SNAPSHOT"

repositories {
    mavenCentral()
}

dependencies {
    testImplementation(kotlin("test"))
}

tasks.test {
    useJUnitPlatform()
}

tasks.withType<KotlinCompile> {
    kotlinOptions.jvmTarget = "16"
}

application {
    mainClass.set("MainKt")
}
```

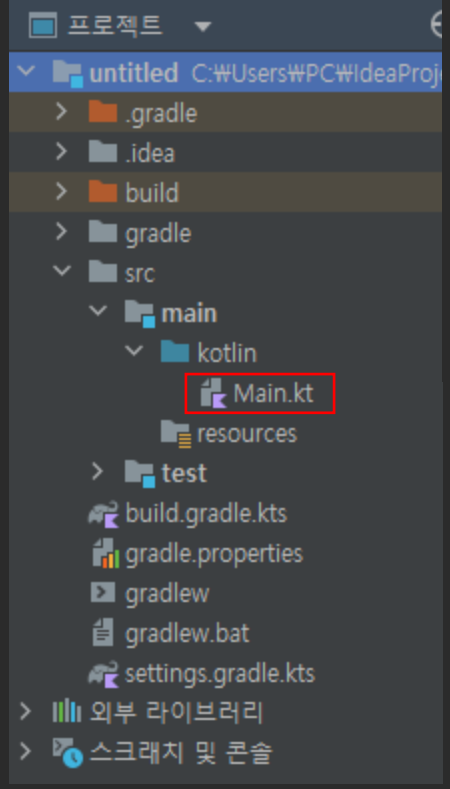



```kotlin
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
    kotlin("jvm") version "1.6.21"
    application
}

group = "me.pc"
version = "1.0-SNAPSHOT"

repositories {
    mavenCentral()
}

dependencies {
    testImplementation(kotlin("test"))
}

tasks.test {
    useJUnitPlatform()
}

tasks.withType<KotlinCompile> {
    kotlinOptions.jvmTarget = "16"
}

application {
    mainClass.set("MainKt")
}
```

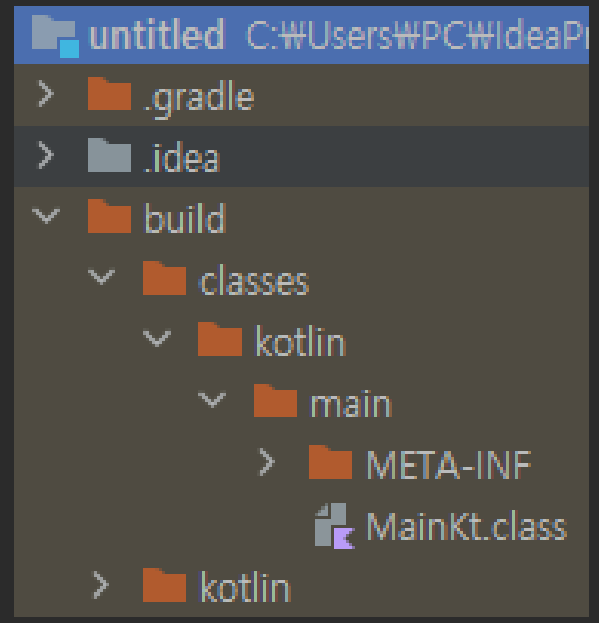
https://docs.gradle.org/current/userguide/application_plugin.html

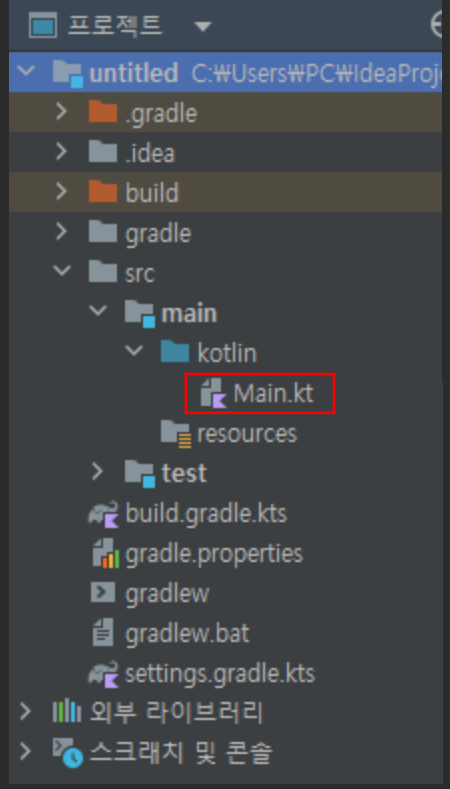



```kotlin
fun main(args: Array<String>) {
    println("Hello World!")

    // Try adding program arguments via Run/Debug configuration.
    // Learn more about running applications: https://www.jetbrains.com/help/idea/running-applications.html.
    println("Program arguments: ${args.joinToString()}")
}
```



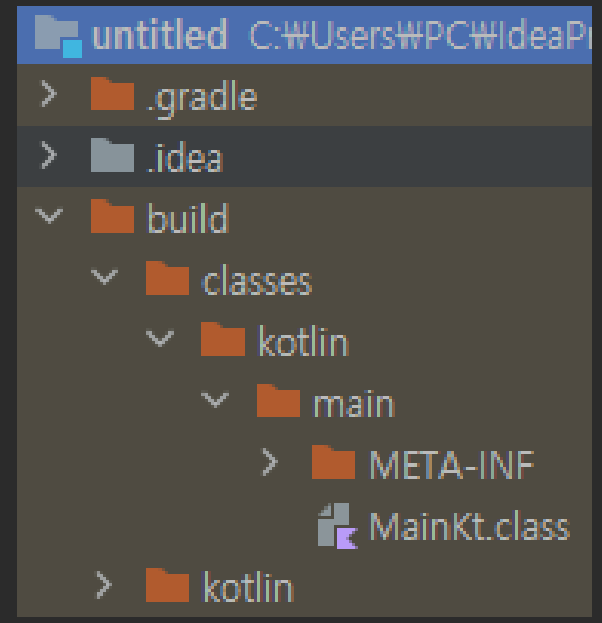

```kotlin
fun main(args: Array<String>) {
    println("Hello World!")

    // Try adding program arguments via Run/Debug configuration.
    // Learn more about running applications: https://www.jetbrains.com/help/idea/running-applications.html.
    println("Program arguments: ${args.joinToString()}")
}
```

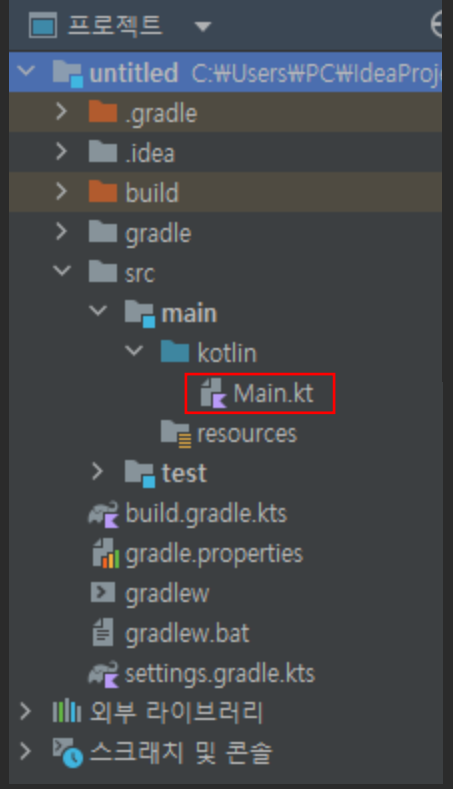





```kotlin
fun main(args: Array<String>) {
    println("Hello World!")

    // Try adding program arguments via Run/Debug configuration.
    // Learn more about running applications: https://www.jetbrains.com/help/idea/running-applications.html.
    println("Program arguments: ${args.joinToString()}")
}
```

코틀린 함수를 JVM에서 사용하기
"파일명Kt" 이름으로 클래스를 생성하여 그 안에 static메소드로 제공

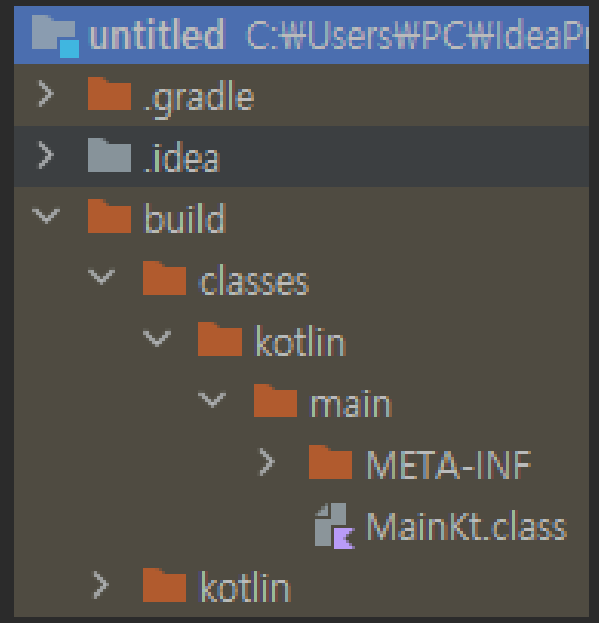


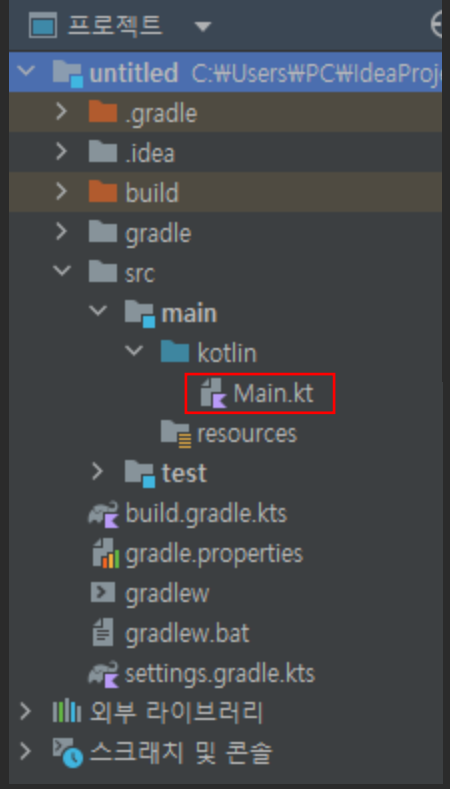



```kotlin
fun main(args: Array<String>) {
    println("Hello World!")

    // Try adding program arguments via Run/Debug configuration.
    // Learn more about running applications: https://www.jetbrains.com/help/idea/running-applications.html.
    println("Program arguments: ${args.joinToString()}")
}
```

코틀린 함수를 JVM에서 사용하기
"파일명Kt" 이름으로 클래스를 생성하여 그 안에 static메소드로 제공

```java
public class MainKt {
    static void main(String[] args){
        ...
    }
}
```

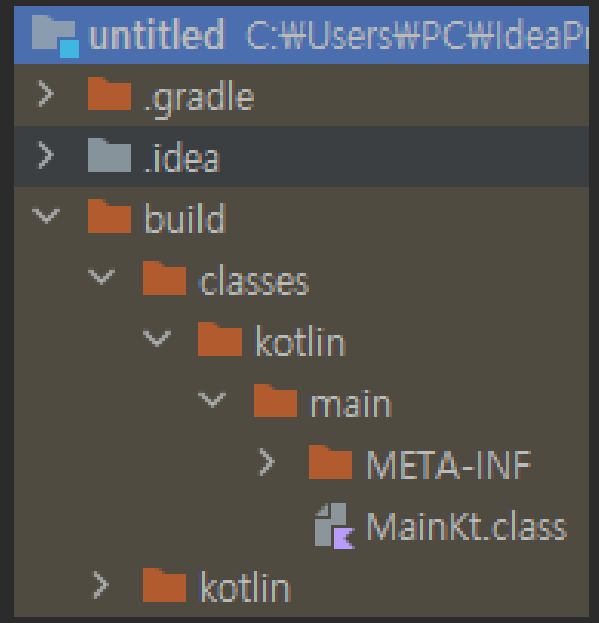





```kotlin
fun main(args: Array<String>) {
    println("Hello World!")

    // Try adding program arguments via Run/Debug configuration.
    // Learn more about running applications: https://www.jetbrains.com/help/idea/running-applications.html.
    println("Program arguments: ${args.joinToString()}")
}
```

코틀린 함수를 JVM에서 사용하기
"파일명Kt" 이름으로 클래스를 생성하여 그 안에 static메소드로 제공

```java
public class MainKt {
    static void main(String[] args){
        ...
    }
}
```



https://kotlinlang.org/docs/java-to-kotlin-interop.html

KOTLIN 101

코틀린에 대한 이해

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지

- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지

- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.
- JS : 컴파일이 느리지만 리플렉션을 제외하면 대부분의 기능 제공

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지

- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.
- JS : 컴파일이 느리지만 리플렉션을 제외하면 대부분의 기능 제공
- android : 구글과의 협력으로 보다 폭 넓은 지원을 통해 달빅의 특수성을 해소

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지

- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.
- JS : 컴파일이 느리지만 리플렉션을 제외하면 대부분의 기능 제공
- android : 구글과의 협력으로 보다 폭 넓은 지원을 통해 달빅의 특수성을 해소
- iOS : 코틀린이 자체적으로 iOS용 GC를 생성하여 메모리관리
- 네이티브 및 웹어셈블리 : 상동 (심지어 웹어셈블리는 표준으로 제안 중)

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지

- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.
- JS : 컴파일이 느리지만 리플렉션을 제외하면 대부분의 기능 제공
- android : 구글과의 협력으로 보다 폭 넓은 지원을 통해 달빅의 특수성을 해소
- iOS : 코틀린이 자체적으로 iOS용 GC를 생성하여 메모리관리
- 네이티브 및 웹어셈블리 : 상동 (심지어 웹어셈블리는 표준으로 제안 중)

기존 언어의 실무적인 사용예를 참고하여 보다 생산성 높고 추상화된 개념사용

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지
- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.
- JS : 컴파일이 느리지만 리플렉션을 제외하면 대부분의 기능 제공
- android : 구글과의 협력으로 보다 폭 넓은 지원을 통해 달빅의 특수성을 해소
- iOS : 코틀린이 자체적으로 iOS용 GC를 생성하여 메모리관리
- 네이티브 및 웹어셈블리 : 상동 (심지어 웹어셈블리는 표준으로 제안 중)

기존 언어의 실무적인 사용예를 참고하여 보다 생산성 높고 추상화된 개념사용

UI영역을 배제하고 순수 메모리와 연산 분야에서의 플랫폼 추상화를 지향함

코틀린에 대한 이해

독립된 언어로 여러 플랫폼으로 번역되는 가운데서도 고유의 철학을 유지
- JVM : 가장 완전한 번역 및 최대한의 관련 기능 제공.
- JS : 컴파일이 느리지만 리플렉션을 제외하면 대부분의 기능 제공
- android : 구글과의 협력으로 보다 폭 넓은 지원을 통해 달빅의 특수성을 해소
- iOS : 코틀린이 자체적으로 iOS용 GC를 생성하여 메모리관리
- 네이티브 및 웹어셈블리 : 상동 (심지어 웹어셈블리는 표준으로 제안 중)

기존 언어의 실무적인 사용예를 참고하여 보다 생산성 높고 추상화된 개념사용

UI영역을 배제하고 순수 메모리와 연산 분야에서의 플랫폼 추상화를 지향함

새로운 언어로서 코틀린만의 언어 구성과 철학을 익혀갈 것

언어 시스템의 형

언어 시스템의 형

자바를 비롯해 포인터를 직접 쓰지 않는 언어 대부분의 형 분류

언어 시스템의 형

자바를 비롯해 포인터를 직접 쓰지 않는 언어 대부분의 형 분류

- 원시형(Primitive)
- 참조형(Reference)

언어 시스템의 형

자바를 비롯해 포인터를 직접 쓰지 않는 언어 대부분의 형 분류

원시형 (Primitive)

- 대입이나 함수 인자 전달 또는 반환 시 복사됨
- 언어에 따라 스택메모리에 잡아서 속도가 빠름

참조형 (Reference)

언어 시스템의 형

자바를 비롯해 포인터를 직접 쓰지 않는 언어 대부분의 형 분류

원시형 (Primitive)

- 대입이나 함수 인자 전달 또는 반환 시 복사됨
- 언어에 따라 스택메모리에 잡아서 속도가 빠름

참조형 (Reference)

- 대입 등에서 참조만 복사됨
- 대부분의 언어에서 힙이라 불리는 영역을 사용

언어 시스템의 형

자바를 비롯해 포인터를 직접 쓰지 않는 언어 대부분의 형 분류

원시형 (Primitive)

- 대입이나 함수 인자 전달 또는 반환 시 복사됨
- 언어에 따라 스택메모리에 잡아서 속도가 빠름

참조형 (Reference)

- 대입 등에서 참조만 복사됨
- 대부분의 언어에서 힙이라 불리는 영역을 사용

원시형과 참조형의 다양한 차이로 인해 boxing, unboxing이 발생하고 래퍼타입이 존재함

코틀린의 기본형(basic type)

코틀린의 기본형(basic type)

코틀린은 원시형의 개념을 사용하지 않고 기본형을 정의함

basic type – number, Boolean, character, string, array

코틀린의 기본형(basic type)

코틀린은 원시형의 개념을 사용하지 않고 기본형을 정의함
basic type – number, Boolean, character, string, array

코드에서는 무조건 래퍼형으로 표현하고 컴파일러가 컨텍스트와 플랫폼에 따라 래퍼형 또는 원시형 중 유리한 것으로 변환함

코틀린의 기본형(basic type)

코틀린은 원시형의 개념을 사용하지 않고 기본형을 정의함
basic type – number, Boolean, character, string, array

코드에서는 무조건 래퍼형으로 표현하고 컴파일러가 컨텍스트와 플랫폼에 따라 래퍼형 또는 원시형 중 유리한 것으로 변환함

코틀린에서는 복사냐 레퍼런스냐의 문제보다 불변성을 갖는 형이냐 아니냐가 중요함
number, Boolean, character, string : 불변형으로 복사든 레퍼런스든 별 상관없음
→ 플랫폼 별 가장 효율적인 번역은 코틀린이 알아서 할 것!

코틀린의 기본형(basic type)

코틀린은 원시형의 개념을 사용하지 않고 기본형을 정의함
basic type – number, Boolean, character, string, array

코드에서는 무조건 래퍼형으로 표현하고 컴파일러가 컨텍스트와 플랫폼에 따라 래퍼형 또는 원시형 중 유리한 것으로 변환함

코틀린에서는 복사냐 레퍼런스냐의 문제보다 불변성을 갖는 형이냐 아니냐가 중요함
number, Boolean, character, string : **불변형으로 복사든 레퍼런스든 별 상관없음**
→ 플랫폼 별 가장 효율적인 번역은 코틀린이 알아서 할 것!

단 제네릭 배열과 원시형 배열을 분리하여 호환시키기 위해 원시형 배열을 별도로 제공
IntArray, LongArray 등

코틀린의 내장형(built-in type)

코틀린의 내장형(built-in type)

Any - 모든 형의 부모

코틀린의 내장형(built-in type)

Any - 모든 형의 부모

Nothing - 모든 형의 자식이자 제어흐름의 종결을 형으로 표현함

코틀린의 내장형(built-in type)

Any - 모든 형의 부모

Nothing - 모든 형의 자식이자 제어흐름의 종결을 형으로 표현함

Unit - 없음을 나타내는 형이자 싱글톤 객체

코틀린의 내장형(built-in type)

Any - 모든 형의 부모

Nothing - 모든 형의 자식이자 제어흐름의 종결을 형으로 표현함

Unit - 없음을 나타내는 형이자 싱글톤 객체

Throwable - 모든 예외의 부모

코틀린의 내장형(built-in type)

Any - 모든 형의 부모

Nothing - 모든 형의 자식이자 제어흐름의 종결을 형으로 표현함

Unit - 없음을 나타내는 형이자 싱글톤 객체

Throwable - 모든 예외의 부모

Function - 함수를 나타내는 형

코틀린의 내장형(built-in type)

Any - 모든 형의 부모

Nothing - 모든 형의 자식이자 제어흐름의 종결을 형으로 표현함

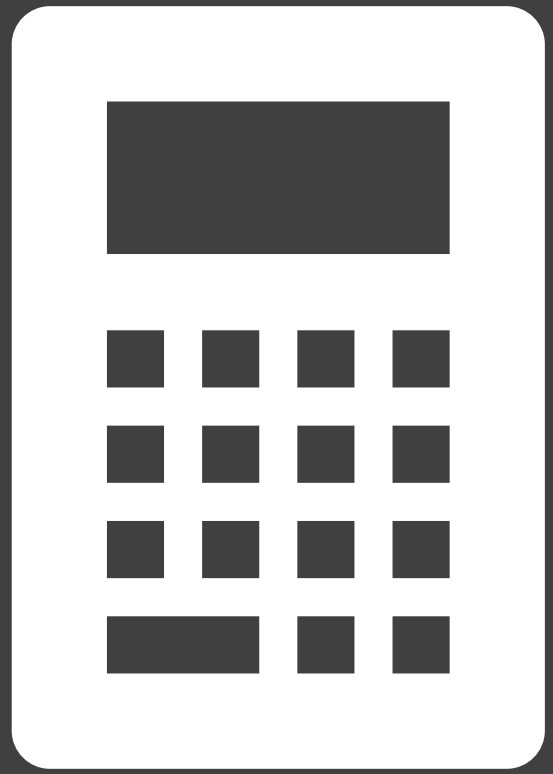
Unit - 없음을 나타내는 형이자 싱글톤 객체

Throwable - 모든 예외의 부모

Function - 함수를 나타내는 형

KClass, KCallable, KProperty, KFunction, KType - 리플렉션 타입

CALCULATOR



calculator logic

-2 * -3 + 0.4 / -0.2

calculator logic

-2 * -3 + 0.4 / -0.2

`-2*-3+0.4/-0.2` trim

calculator logic

-2 * -3 + 0.4 / -0.2

```
-2*-3+0.4/-0.2          trim
+-2*+-3+0.4/+-0.2       Replace - to +-
```

calculator logic

-2 * -3 + 0.4 / -0.2

```
-2*-3+0.4/-0.2                trim
+-2*+-3+0.4/+-0.2             Replace - to +-
(+-2*+-3), (+0.4/+-0.2)       */ group
```

calculator logic

-2 * -3 + 0.4 / -0.2

```
-2*-3+0.4/-0.2                  trim
+-2*+-3+0.4/+-0.2               Replace - to +-
(+-2*+-3), (+0.4/+-0.2)         */ group
+-2, *, +-3                     split
```

calculator logic

`-2 * -3 + 0.4 / -0.2`

`-2*-3+0.4/-0.2`	trim
`+-2*+-3+0.4/+-0.2`	Replace - to +-
`(+-2*+-3), (+0.4/+-0.2)`	*/ group
`+-2, *, +-3`	split
`+-2 → -2, +-3 → -3`	remove +

calculator logic

-2 * -3 + 0.4 / -0.2

-2*-3+0.4/-0.2	trim
+-2*+-3+0.4/+-0.2	Replace - to +-
(+-2*+-3), (+0.4/+-0.2)	*/ group
+-2, *, +-3	split
+-2 → -2, +-3 → -3	remove +
-2 * -3 = 6	calc & replace+-

calculator logic

-2 * -3 + 0.4 / -0.2

-2*-3+0.4/-0.2	trim
+-2*+-3+0.4/+-0.2	Replace - to +-
(+-2*+-3), (+0.4/+-0.2)	*/ group
+-2, *, +-3	split
+-2 → -2, +-3 → -3	remove +
-2 * -3 = 6	calc & replace+-

calculator logic

`-2 * -3 + 0.4 / -0.2`

`-2*-3+0.4/-0.2`	trim
`+-2*+-3+0.4/+-0.2`	Replace - to +-
`(+-2*+-3), (+0.4/+-0.2)`	*/ group
`+-2, *, +-3`	split
`+-2 → -2, +-3 → -3`	remove +
`-2 * -3 = 6`	calc & replace+-
`+0.4, /, +-0.2`	split

calculator logic

-2 * -3 + 0.4 / -0.2

-2*-3+0.4/-0.2	trim
+-2*+-3+0.4/+-0.2	Replace - to +-
(+-2*+-3), (+0.4/+-0.2)	*/ group
+-2, *, +-3	split
+-2 → -2, +-3 → -3	remove +
-2 * -3 = 6	calc & replace+-
+0.4, /, +-0.2	split
+0.4 → 0.4, +-0.2 → -0.2	remove +

calculator logic

`-2 * -3 + 0.4 / -0.2`

`-2*-3+0.4/-0.2`	trim
`+-2*+-3+0.4/+-0.2`	Replace - to +-
`(+-2*+-3), (+0.4/+-0.2)`	*/ group
`+-2, *, +-3`	split
`+-2 → -2, +-3 → -3`	remove +
`-2 * -3 = 6`	calc
`+0.4, /, +-0.2`	split
`+0.4 → 0.4, +-0.2 → -0.2`	remove +
`0.4 / -0.2 = -2`	calc

calculator logic

`-2 * -3 + 0.4 / -0.2`

`-2*-3+0.4/-0.2`	trim
`+-2*+-3+0.4/+-0.2`	Replace - to +-
`(+-2*+-3), (+0.4/+-0.2)`	*/ group
`+-2, *, +-3`	split
`+-2 → -2, +-3 → -3`	remove +
`-2 * -3 = 6`	calc
`+0.4, /, +-0.2`	split
`+0.4 → 0.4, +-0.2 → -0.2`	remove +
`0.4 / -0.2 = -2`	calc

`6, -2` sum

calculator logic

-2 * -3 + 0.4 / -0.2

-2*-3+0.4/-0.2	trim
+-2*+-3+0.4/+-0.2	Replace - to +-
(+-2*+-3), (+0.4/+-0.2)	*/ group
+-2, *, +-3	split
+-2 → -2, +-3 → -3	remove +
-2 * -3 = 6	calc
+0.4, /, +-0.2	split
+0.4 → 0.4, +-0.2 → -0.2	remove +
0.4 / -0.2 = -2	calc

6, -2

sum

4

calculator logic

-2 * -3 + 0.4 / -0.2

trim

```
-2*-3+0.4/-0.2                  trim
+-2*+-3+0.4/+-0.2               Replace - to +-
(+-2*+-3), (+0.4/+-0.2)         */ group
+-2, *, +-3                     split
+-2 → -2, +-3 → -3              remove +
-2 * -3 = 6                     calc
+0.4, /, +-0.2                  split
+0.4 → 0.4, +-0.2 → -0.2        remove +
0.4 / -0.2 = -2                 calc
```

6, -2

sum

4

calculator logic

-2 * -3 + 0.4 / -0.2

trim
repMtoPM

-2*-3+0.4/-0.2	trim
+-2*+-3+0.4/+-0.2	Replace - to +-
(+-2*+-3), (+0.4/+-0.2)	*/ group
+-2, *, +-3	split
+-2 → -2, +-3 → -3	remove +
-2 * -3 = 6	calc
+0.4, /, +-0.2	split
+0.4 → 0.4, +-0.2 → -0.2	remove +
0.4 / -0.2 = -2	calc

6, -2

sum

4

calculator logic

-2 * -3 + 0.4 / -0.2

```
-2*-3+0.4/-0.2                  trim
+-2*+-3+0.4/+-0.2               Replace - to +-
(+-2*+-3), (+0.4/+-0.2)         */ group
+-2, *, +-3                    split
+-2 → -2, +-3 → -3              remove +
-2 * -3 = 6                     calc
+0.4, /, +-0.2                  split
+0.4 → 0.4, +-0.2 → -0.2        remove +
0.4 / -0.2 = -2                 calc
```

6, -2

sum

4

```
trim
repMtoPM
groupMD
```

calculator logic

`-2 * -3 + 0.4 / -0.2`

```
-2*-3+0.4/-0.2                trim
+-2*+-3+0.4/+-0.2             Replace - to +-
(+-2*+-3), (+0.4/+-0.2)       */ group
+-2, *, +-3                   split
+-2 → -2, +-3 → -3            remove +
-2 * -3 = 6                   calc
+0.4, /, +-0.2                split
+0.4 → 0.4, +-0.2 → -0.2      remove +
0.4 / -0.2 = -2               calc
```

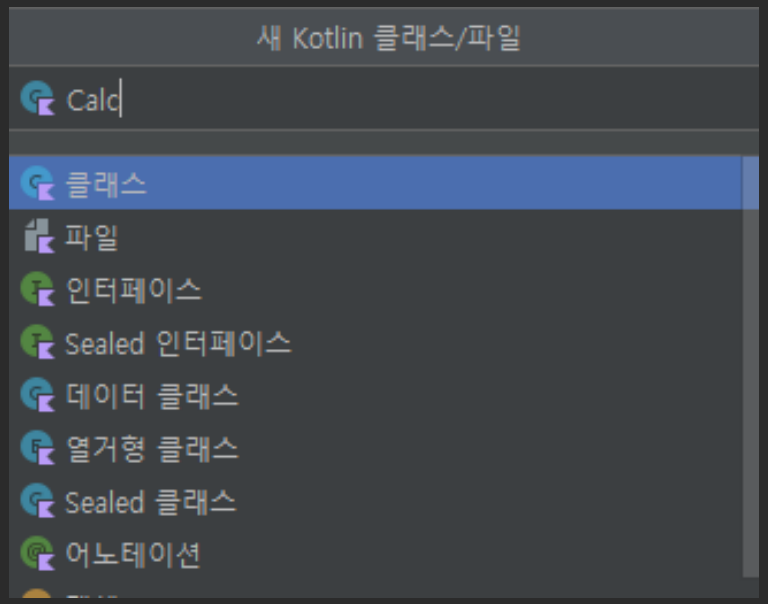
`6, -2`

sum

4

```
trim
repMtoPM
groupMD
splitMD
```

Calc.kt


새 Kotlin 클래스/파일
Calc
클래스
파일
인터페이스
Sealed 인터페이스
데이터 클래스
열거형 클래스
Sealed 클래스
어노테이션

trim

```
class Calc {
}
```

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

```
-2*-3+0.4/-0.2                trim
+-2*+-3+0.4/+-0.2             Replace - to +-
(+-2*+-3), (+0.4/+-0.2)       */ group
+-2, *, +-3                  split
+-2 → -2, +-3 → -3            remove +
-2 * -3 = 6                   calc
+0.4, /, +-0.2                split
+0.4 → 0.4, +-0.2 → -0.2      remove +
0.4 / -0.2 = -2               calc
```

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

```
val : 값(value)
var : 변수(variable)
const val : 상수(constant)
```

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

```
val : 값(value)
var : 변수(variable)
const val : 상수(constant)
```

한 줄에 하나만

```
val a, b, c (x)
val a       (o)
```

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

```
val : 값(value)
var : 변수(variable)
const val : 상수(constant)
```

한 줄에 하나만

```
val a, b, c (x)
val a       (o)
```

이름 : 타입 = 값

```
val a:Int = 3
var b:String = "abc"
```

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

```
'   '
 ...  : Char literal
"   "
 ...  : String literal
"""   """
   ...    : no escaping, newlines
```

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

`[...]` : Character Class

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

`[...]` : Character Class

`[^...]` : Exception Character Class

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

`[...]` : Character Class

`[^...]` : Exception Character Class

`\d` : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

trim

```
class Calc {
}
val trim = """[^.\d-+*/]""".toRegex()
```

`[...]` : Character Class

`[^...]` : Exception Character Class

`\d` : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

except

. 0 1 2 3 4 5 6 7 8 9 - + * /

trim

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
```

trim

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
```

```
fun 함수명(인자):반환형{
    몸체
    return 반환값
}
```

repMtoPM

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2          Replace - to +-
(+-2*+-3), (+0.4/+-0.2)    */ group
+-2, *, +-3                split
+-2 → -2, +-3 → -3         remove +
-2 * -3 = 6                calc
+0.4, /, +-0.2             split
+0.4 → 0.4, +-0.2 → -0.2   remove +
0.4 / -0.2 = -2            calc
```

repMtoPM

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
```

```
fun 함수명(인자):반환형 = 반환식
```

repMtoPM

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
```

```
fun 함수명(인자):반환형 = 반환식
    fun repMtoPM(v:String):String = v.replace("-", "+-")
```

repMtoPM

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
```

```
fun 함수명(인자):반환형 = 반환식
    fun repMtoPM(v:String):String = v.replace("-", "+-")

fun 함수명(인자)(:반환형) = 반환식의 형
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
                    (...) : capture group
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
```

```
(...) : capture group
(?:...) : non-capture group
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
```

```
(...) : capture group
(?:...) : non-capture group
(..|..) : alternative
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
```

```
(...) : capture group
(?:...) : non-capture group
(..|..) : alternative
? : zero or one = {0, 1}
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
```

```
(...) : capture group
(?:...) : non-capture group
(..|..) : alternative
? : zero or one = {0, 1}
+ : one or unlimited = {1,}
```

groupMD

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
```

(...) : capture group

(?:...) : non-capture group

(..|..) : alternative

? : zero or one = {0, 1}

\+ : one or unlimited = {1,}

(+-1, +1, 1) (*, /) (+-1, +1, 1)

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value
}
```

```
val sum = {a:Int, b:Int -> Int
    a + b
}
println( sum(2, 3) ) //5
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value
}
```

```
val sum = {a:Int, b:Int -> Int
    a + b
}
println( sum(2, 3) ) //5
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value(Type)
}
```

```
val sum = {a:Int, b:Int ->
      a + b
}
println( sum(2, 3) ) //5
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value
}
```

```
val sum:(Int, Int)->Int = {a:Int, b:Int -> Int
    a + b
}
println( sum(2, 3) ) //5
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value
}
```

```
val sum:(Int, Int)->Int = {a:Int, b:Int ->
    a + b
}
println( sum(2, 3) ) //5
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type, a2:Type -> Type
    return value
}
```

```
val sum:(Int, Int)->Int = {a, b ->
    a + b
}
println( sum(2, 3) ) //5
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type->
    return value
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type->
    return value
}
```

```
val twice = {a:Int ->
      a + a
}
println( twice(2) ) //4
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type ->  it
    return value
}
```

```
val twice = {a:Int ->
      a + a
}
println( twice(2) ) //4
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type -> it
    return value
}
```

```
val twice = {
        it + it
}
println( twice(2) ) //4
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
{a1:Type -> it
    return value
}
```

```
val twice:(Int)->Int = {
        it + it
}
println( twice(2) ) //4
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
fun delegate(a:Int, b:(Int)->Int):Int{
    return b(a)
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
fun delegate(a:Int, b:(Int)->Int):Int{
    return b(a)
}

delegate(3, { it * 3 })
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
fun delegate(a:Int, b:(Int)->Int):Int{
    return b(a)
}

delegate(3, { it * 3 })
delegate(3){ it * 3 }
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
fun delegate(a:Int, b:(Int)->Int):Int{
    return b(a)
}

delegate(3, { it * 3 })
delegate(3){ it * 3 }
```

Passing trailing lambdas

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

Sequence<MatchResult>

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
      ...
      다음 요소에 넘길 합산값R
    } = R
}
```

Sequence<MatchResult>

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
      ...
      다음 요소에 넘길 합산값R
    } = R
}
```

Sequence<MatchResult>

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "-").toDouble()
    val rightValue = right.replace("+-", "-").toDouble()
    val result = when(op){
        "*" -> leftValue * rightValue
        "/" -> leftValue / rightValue
        else -> throw Throwable("invalid operator $op")
    }
    acc + result
}
```

Sequence<MatchResult>

```
Sequence<T>.fold<R>(초기값R){이전까지 합산한 값R, 현재요소T->
    ...
    다음 요소에 넘길 합산값R
} = R
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    "*"->leftValue * rightValue
    "/"->leftValue / rightValue
    else->throw Throwable("invalid operator $op")
  }
  acc + result
}
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    "*"->leftValue * rightValue
    "/"->leftValue / rightValue
    else->throw Throwable("invalid operator $op")
  }
  acc + result
}
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+-", "").toDouble()
    val rightValue = right.replace("+-", "").toDouble()
    val result = if(op == "*"){
        leftValue * rightValue
    }else if(op == "/"){
        leftValue / rightValue
    }else throw Throwable("Invalid Operator $op")
    acc + result
}
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
[+-2, *, +-3], [+0.4, /, +-0.2]
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    "$op"->leftValue * rightValue
    else -> throw Throwable("Invalid Operator $op")
    }
    acc + result
}
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
[+-2, *, +-3], [+0.4, /, +-0.2]
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    "$op"->leftValue * rightValue
    else -> throw Throwable("Invalid Operator $op")
    }
    acc + result
}
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
[+-2, *, +-3], [+0.4, /, +-0.2]
```

Destructuring

내부에 N번째 반환할 값을 정의
(..)를 통해 얻고 필요없는 값은 _로 처리

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
[+-2, *, +-3], [+0.4, /, +-0.2]
```

Destructuring

내부에 N번째 반환할 값을 정의

(..)를 통해 얻고 필요없는 값은 _로 처리

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
[+-2, *, +-3], [+0.4, /, +-0.2]
```

Destructuring

내부에 N번째 반환할 값을 정의
(..)를 통해 얻고 필요없는 값은 _로 처리

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    acc + result
}
```

```
MatchResult.groupValues =
List<String>(전체, 그룹1, 그룹2, 그룹3..)
(+-2*+-3), (+0.4/+-0.2)
[+-2, *, +-3], [+0.4, /, +-0.2]
 left  op  right      left  op  right
```

Destructuring

내부에 N번째 반환할 값을 정의
(..)를 통해 얻고 필요없는 값은 _로 처리

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2           Replace - to +-
(+-2*+-3), (+0.4/+-0.2)     */ group
+-2, *, +-3                 split
+-2 → -2, +-3 → -3          remove +
-2 * -3 = 6                 calc
+0.4, /, +-0.2              split
+0.4 → 0.4, +-0.2 → -0.2    remove +
0.4 / -0.2 = -2             calc
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
when(대상){
    값1->
    값2->{...}
    ...
    else->
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
when(대상){
    값1->
    값2->{...}
    ...
    else->
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
when(대상){
    값1->
    값2->{...}
    ...
    else->
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
when(대상){
   값1->
   값2->{...}
   ...
   else->
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
v:T = when(대상){
  값1->
  값2->{...}
  ...
  else->
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
v:T = when(대상){
  값1->T
  값2->{
    ...
    T
  }
  ...
  else->T
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue   Double
        "/"->leftValue / rightValue   Double
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

```
v:T = when(대상){
  값1->T
  값2->{
    ...
    T
  }
  ...
  else->T
}
```

foldGroup

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
```

calc

```
val trim = """[^.\d-+*/]""".toRegex()
fun trim(v:String):String{
    return v.replace(trim, "")
}
fun repMtoPM(v:String) = v.replace("-", "+-")
val groupMD = """((?:\+|\+-)?[.\d]+)([*/])((?:\+|\+-)?[.\d]+)""".toRegex()
fun foldGroup(v:String):Double = groupMD.findAll(v).fold(0.0){ acc, curr->
    val (_, left, op, right) = curr.groupValues
    val leftValue = left.replace("+", "").toDouble()
    val rightValue = right.replace("+", "").toDouble()
    val result = when(op){
        "*"->leftValue * rightValue
        "/"->leftValue / rightValue
        else->throw Throwable("invalid operator $op")
    }
    acc + result
}
fun calc(v:String) = foldGroup( repMtoPM( trim(v) ) )
```

```
-2*-3+0.4/-0.2              trim
+-2*+-3+0.4/+-0.2          Replace - to +-
(+-2*+-3), (+0.4/+-0.2)    */ group
+-2, *, +-3                split
+-2 → -2, +-3 → -3         remove +
-2 * -3 = 6                calc
+0.4, /, +-0.2             split
+0.4 → 0.4, +-0.2 → -0.2   remove +
0.4 / -0.2 = -2            calc
```

main

```
fun main(args: Array<String>) {
    println(calc("-2 * -3 + 0.4 / - 0.2"))
}
```

과제 1

```
fun main(args: Array<String>) {
    println(calc("-2 -3 + 0.4"))
}
```

현재 계산 로직은 *, / 가 없으면 제대로 작동하지 않는다.
이를 계선하여 단순 +, -도 잘 작동하게 개선하라

과제2

```
fun main(args: Array<String>) {
    println(calc("-2 * (-3 + 0.4) / - 0.2"))
}
```

수식에 괄호가 작동하도록 개선하라.

과제3

```
fun main(args: Array<String>) {
    println(calc("-2 * (-3 + 0.4) / - 0.2"))
}
```

과제2는 괄호를 처리해도 정상작동 하지 않는다. 이유는 a*b + c/d 처럼 곱셈이나 나눗셈이 반드시 쌍이 맞는 짝수의 숫자로 맞아야하기 때문이다. a*b/c 같이 홀수로 주어진 경우는 fold가 정상적으로 작동하지 않는다. 이를 해결하라.