1. Gradle Project 만들기
  1)Gradle을 사용하여 Java project를 작성한다.
  2)그리고 project가 어떻게 구성되어 있는지 살펴 보고 설명한다.
  3)Gradle project 초기화
    -project를 초기화하기
      --Gradle 개발을 하려면, 먼저 Gradle에 의한 project를 준비한다.
      --이는 다음과 같은 단계를 수행한다.

    1. project folder를 만든다.

      C:\>mkdir GradleHome
      C:\>cd GradleHome

    2. project folder로 이동한다.

      C:\GradleHome>mkdir GradleApp
      C:\GradleHome>cd GradleApp

    3. project를 초기화한다.
      -이것이 실질적으로 Gradle project의 기반을 만드는 작업이다.
      -다음 명령을 실행한다.

      $ gradle init --type java-application

      C:\GradleHome\GradleApp>gradle init --type java-application

      -gradle init라는 것이 Gradle 초기화를 위한 명령이다.
      -이후에 --type java-application는 Java 응용 program project 유형을 지정한다.

      Starting a Gradle Daemon (subsequent builds will be faster)

      BUILD SUCCESSFUL in 9s
      2 actionable tasks: 2 executed

      -잠시 기다리고 있으면 folder에 필요한 file이나 folder가 만들어 진다.

  4)Gradle project 구성
    -생성된 project가 어떻게 구성되어 있는지, folder의 내용을 확인한다.
    -다음과 같은 것들이 준비되어 있어야 한다.

```
├── build.gradle
├── gradle
│   └── wrapper
│           ├── gradle-wrapper.jar
│           └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── App.java
    └── test
        └── java
            └── AppTest.java
```

    -.gradle folder
      --Gradle이 사용하는 folder이다.
      --작업(task)로 생성된 file이 저장된다.
      --이 내용을 편집하는 일은 거의 없다.

-gradle folder
--이것도 Gradle이 필요한 경우 사용할 folder이다.
--기본적으로 Gradle 환경을 정리한 "wrapper file"이라는 file들이 저장되어 있다.

-src folder
--이것이 project에서 만든 program 관련 folder이다.
--project에서 사용하는 file(source code file, 각종 resource file 등)은 모두 이 안에 들어간다.

-build.gradle
--Gradle 기본 build 설정 file이다.
--이 안에 project의 build 처리에 대해서 내용이 작성되어 있다.

-gradlew, gradlew.bat
--이 2개는 Gradle의 명령이다.
--bat가 붙어있는 것이 Windows 용이고, macOS 및 Linux 용이다.

-settings.gradle
--project에 대한 설정 정보를 작성한 file이다.

-가장 중요한 것은 src folder이다.
-이 안에 개발하는 project에서 사용하는 file이 모두 저장된다.
-다음으로 중요한 것은 "build.gradle" file이다.
-이것은 build file이고 그래서 build 처리의 내용을 작성하는 file이다.
-이 file은 Groovy 언어로 작성되어 있다.

-src folder
--그럼, 개발의 본체 부분이 되는 src folder를 열어 본다.
--이 folder에는 이미 여러 folder와 file이 준비되어 있다.
--folder의 구성을 정리하면 다음과 같이 되어 있다.

```
src
├── main
│       └── java
│               └── App.java
└── test
        └── java
                └── AppTest.java
```

-src foler에는 main과 test라는 2개의 folder가 포함된다.
-이들은 각각 다음과 같은 역할을 한다.

--main folder
---이것이 만드는 program 본체의 folder을 모아 두는 folder이다.
---이것을 열면 java folder가 있다.
---이는 Java source code를 넣어두기 위한 folder이다.
---이 안에 sample로 App.java는 source code file이 포함되어 있다.

--test folder
---이것은 unit test file을 모아 두는 folder이다.
---역시 java folder가 있고 그 안에 AppTest.java sample foler이 있다.

--main와 test에도 그 중에 먼저 언어 이름의 folder가 있고 거기에 그 언어로 작성된 source dode file이 배치되는 구조로 되어 있는 것을 알 수 있다.
--이것이 Gradle project의 기본 folder 구조이다.

-src의 내용은 Maven과 같다?
--folder 구성을 보고 어디 선가 본 적이 있는 사람도 있을 수도 있다.
--사실은 src folder의 구성은 Apache Maven에 의한 project와 똑같다.
--Maven project도 project folder에 src folder가 있고 그 안에 main과 test가 있고 등등

똑같은 구성되어 있다.
--이러한 build 도구는 아무래도 Maven에 의해 일반적으로 영향받았다고 해도 될 것이다.
--Maven은 좋든 나쁘든 Java build 도구의 표준이 되고 있으며, Gradle도 Maven의 folder 구조를 그대로 따르고 있는 것이다.

5)gradle init 명령과 type 종류
-Gradle project를 만든 gradle init 명령에 대해 설명한다.
-이것은 "init"라는 작업을 수행하는 것이다.
-Gradle은 수행할 작업은 "task(task)"라고 한다.
-gradle 명령은 이 task을 지정하고 실행하는 것이다.
-init task은 folder에 project file이나 folder들을 생성하고 folder를 초기화한다.
- --type 옵션이 있다.
-이 옵션에 의해 "어떤 종류의 program 작성을 위한 project에 초기화 하는지"를 지정할 수 있다.
-이 유형은 조금씩 증가하고 있어 2017년 10 월 현재는 다음의 것이 준비되어 있다.

--java-application
---Java application project 작성에 대한 type이다.
---기본적으로 App.java가 제공된다.

--java-library
---Java library project 생성을 위한 type이다.
---단순히 샘플로 제공되는 source code file이 응용 program의 메인 class가 되어 있지 않다는 정도의 차이이다.
---(그리고, build.gradle도 조금 다르다)

--groovy-application
---Groovy application 개발을 위한 project이다.
---Groovy 개발의 기본 type이라고 해도 좋을 것이다.

--groovy-library
---Groovy library 개발을 위한 project이다.
---기본적으로 groovy-application과 같고, 샘플 code가 다른 정도이다.

--scala-library
---이것은 Java 가상 machine에서 구동되는 언어 Scala의 개발 type이다.
---Scala에서는 여전히 응용 program의 type은 준비되어 있지 않은 것 같다.

--basic
---기본 type이다.
---이것은 모든 type의 기반이 되는 것으로 src 는 제공되지 않는다.
---또한 build file도 구체적인 처리 등은 기술되지 않고, build.gradle과 settings.gradle만 생성된다.
- --type을 붙이지 않고, 단순히 gradle init만 실행하면 이 basic type이 지정된다.

--pom
---Maven의 pom.xml을 바탕으로 build.gradle 을 생성한다.

--Java 프로그래머는 java-application, java-library만 알고 있으면 충분하다.

6)compile 및 실행
-여기서는 만든 project를 Gradle 명령으로 처리를 설명한다.
-여기에서 compile, 실행, package 같은 기본 조작 처리를 설명한다.
-program compile

$ gradle compileJava

C:\GradleHome\GradleApp>gradle compileJava
Download https://jcenter.bintray.com/com/google/guava/guava/23.0/guava-23.0.pom
Download

https://jcenter.bintray.com/com/google/guava/guava-parent/23.0/guava-parent-23.0.pom
Download https://jcenter.bintray.com/org/sonatype/oss/oss-parent/7/oss-parent-7.pom
Download https://jcenter.bintray.com/com/google/j2objc/j2objc-annotations/1.1/j2objc-annotations-1.1.pom
Download https://jcenter.bintray.com/org/codehaus/mojo/animal-sniffer-annotations/1.14/animal-sniffer-annotations-1.14.pom
Download https://jcenter.bintray.com/com/google/code/findbugs/jsr305/1.3.9/jsr305-1.3.9.pom
Download https://jcenter.bintray.com/org/codehaus/mojo/animal-sniffer-parent/1.14/animal-sniffer-parent-1.14.pom
Download https://jcenter.bintray.com/com/google/errorprone/error_prone_annotations/2.0.18/error_prone_annotations-2.0.18.pom
Download https://jcenter.bintray.com/org/codehaus/mojo/mojo-parent/34/mojo-parent-34.pom
Download https://jcenter.bintray.com/com/google/errorprone/error_prone_parent/2.0.18/error_prone_parent-2.0.18.pom
Download https://jcenter.bintray.com/org/codehaus/codehaus-parent/4/codehaus-parent-4.pom
Download https://jcenter.bintray.com/org/codehaus/mojo/animal-sniffer-annotations/1.14/animal-sniffer-annotations-1.14.jar
Download https://jcenter.bintray.com/com/google/guava/guava/23.0/guava-23.0.jar
Download https://jcenter.bintray.com/com/google/j2objc/j2objc-annotations/1.1/j2objc-annotations-1.1.jar
Download https://jcenter.bintray.com/com/google/code/findbugs/jsr305/1.3.9/jsr305-1.3.9.jar
Download https://jcenter.bintray.com/com/google/errorprone/error_prone_annotations/2.0.18/error_prone_annotations-2.0.18.jar

BUILD SUCCESSFUL in 5s
1 actionable task: 1 executed

--compile은 compileJava 라는 task로 제공한다.
--이것은 Java source code를 compile하기 위한 것이다.

-program 실행

$ gradle run

> Task :run
Hello world.

BUILD SUCCESSFUL in 0s
2 actionable tasks: 1 executed, 1 up-to-date

--java-application type의 project에는 run task라는 것이 제공되고, 이를 실행하여 메인 class를 실행할 수 있다.
--디폴트로 App.java가 실행된다.

-program package

$ gradle jar

BUILD SUCCESSFUL in 0s
2 actionable tasks: 1 executed, 1 up-to-date

--jar task은 그 이름대로 program을 Jar file에 모와서 저장한다.
--이는 project에 생성되는 build folder의 libs folder에 저장된다.

-project 클린

$ gradle clean

BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed

--project를 build할 때에 build folder에 여러 file이 저장된다.
--clean 작업은 이 file들을 제거하고 build 이전 상태로 되돌린다.

-build 및 실행
--project 실행 시에 "Hello World."라는 메시지가 출력된다.
--또한 build folder의 libs folder에는 GradleApp.jar이라는 Jar file이 생성된 것을 확인할 수 있다.
--그러나 java -jar GradleApp.jar으로 이 Jar file을 실행할 수 없다.
--왜냐하면 매니페스트 file이 포함되어 있지 않아 실행 가능한 Jar file로 아니기 때문이다.
--jara -classpath GradleApp.jar App와 같은 방식으로 -classpath를 지정하여, 명시적으로 App class를 실행하면 제대로 실행할 수 있다.
--조금은 귀찮지만, 일단은 동작하는 것만 확인한다.

2. Lab
1)개발 환경
-Java : 1.8.x
-Gradle : 4.8

2)Gradle project를 생성한다.
-directory gradle-hello-world를 생성하고, Gradle project를 초기화한다.

$ mkdir gradle-hello-world
$ cd gradle-hello-world/
$ gradle init

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed

3)source code를 작성한다.
-source code directory를 생성한다.

$ mkdir src\main\java\hello

-source code src/main/java/hello/HelloWorld.java를 작성한다.

```
package hello;

public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
```

```
266         }
267
268     -Gradle로 build한다.
269       --build를 위해 build.gradle에 아래 내용을 추가한다.
270
271           apply plugin: 'java'
272
273
274           /*
275            * This file was generated by the Gradle 'init' task.
276            *
277            * This is a general purpose Gradle build.
278            * Learn how to create Gradle builds at
                https://guides.gradle.org/creating-new-gradle-builds/
279            */
280
281           apply plugin: 'java'          //새로 code 추가
282
283
284       --NOTE : 모든 build에 대한 내용은 build.gradle에 기입된다.
285       --build 명령어 gradle build로 build을 하면 build directory를 생성하고, source code
          build를 진행하게 된다.
286
287           $ gradle build
288
289           BUILD SUCCESSFUL in 2s
290           2 actionable tasks: 2 executed
291
292     -Gradle project를 gradlew으로 실행하기
293       --build.gradle file에 아래 내용을 추가하면 application을 직접 구동할 수 있다.
294
295           apply plugin: 'application'
296           mainClassName = 'hello.HelloWorld'
297
298       --구동 명령어 gradlew run로 application을 직접 구동해 본다.
299
300           $ gradlew run
301           Downloading https://services.gradle.org/distributions/gradle-4.8-bin.zip
302           ..........................................................................
303
304           > Task :run
305           Hello World!
306
307           BUILD SUCCESSFUL in 28s
308           2 actionable tasks: 1 executed, 1 up-to-date
309
310     -directory 구조
311       --생성된 directory를 확인하기 위해 tree를 입력한다.
312
313           $ tree
314           folder PATH의 목록입니다.
315           볼륨 일련 번호는 F8A9-F182입니다.
316           C:.
317           ├─.gradle
318           │  ├─4.8
319           │  │  ├─fileChanges
320           │  │  ├─fileContent
321           │  │  ├─fileHashes
322           │  │  └─taskHistory
323           │  ├─buildOutputCleanup
```

```
|   └─vcsWorkingDirs
├─build
|   ├─classes
|   |   └─java
|   |       └─main
|   |           └─hello
|   ├─libs
|   └─tmp
|       ├─compileJava
|       └─jar
├─gradle
|   └─wrapper
└─src
    └─main
        └─java
            └─hello
```

3. Lab

1)C:\GradleHome에서 command-line 명령을 실행하도록 한다.

```
git clone https://github.com/spring-guides/gs-gradle.git
```

2)실행결과 gs-gradle이라는 folder가 생길 것이다.

-gs-gradle folder 밑에 있는 "complete" folder로 이동한다.

-아래와 같은 file 및 folder들이 보일 것이다.

```
gradle
src
build.gradle
gradlew
gradlew.bat
```

3)우선, initial\src\main\java\hello\HelloWorld.java 코드를 추가하자.

```
package hello;

public class HelloWorld {
  public static void main(String[] args) {
    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
  }
}
```

위의 코드를 아래와 같이 수정한다.

```
package hello;

import org.joda.time.LocalTime;                    //추가

public class HelloWorld {
  public static void main(String[] args) {
    LocalTime currentTime = new LocalTime();    //추가
    System.out.println("The current local time is: " + currentTime);  //추가

    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
  }
}
```

-org.joda.time.LocalTime 이라는 외부 모듈을 사용하고 있다.
-maven이라면, 이것을 pom.xml의 dependenies에 이렇게 추가했을 것이다.

```
<dependency>
   <groupId>joda-time</groupId>
   <artifactId>joda-time</artifactId>
   <version>2.2</version>
</dependency>
```

-이것을 build.gradle file에 reposigory를 설정해 두어야 한다.

```
repositories {
   mavenLocal()
   mavenCentral()
}
```

-이제 앞서 언급한 maven 형식의 dependency를 아래와 같이 추가한다.

```
dependencies {
   compile "joda-time:joda-time:2.2"
}
```

-compile할 때 사용한다는 의미로 "compile"을 맨 앞에 써주었고, 한 칸 이상 띄운다음

```
"{groupId}:{artifactId}:{version}"
```

-위와 같이 groupId, artifactId, version을 순서대로 해서, 사이 사이에 ":"을 넣으면 된다.
-build.gradle file에 아래의 코드를 추가한다.

```
apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'application'

mainClassName = 'hello.HelloWorld'
```

-apply plugin: 'java'는 java 코드를 build할 수 있게 해준다.
-apply plugin: 'application'은 java 코드를 실행할 수 있게 해 준다.
-java code를 실행하기 위해서는 mainClassName을 위와 같이 설정해 주어야 한다.
-소스코드 build와 java code를 실행하는 명령은 각각 다음과 같다.

```
gradle build
gradle run
```

-"gradle build"를 실행하면, "build" folder가 생성되는 데, 이것이 바로 "target" folder 즉, 컴파일된 결과물들이 생성되는 folder이다.
-gradlew을 사용해서 run을 해보자.

```
gradlew run

Downloading https://services.gradle.org/distributions/gradle-4.6-bin.zip
.....
.....
.....
.....
.....
.....
Unzipping
C:\Users\Instructor\.gradle\wrapper\dists\gradle-4.6-bin\4jp4stjndanmxuerzfseyb6wo\gradle-4.6-bin.zip to
```

```
C:\Users\Instructor\.gradle\wrapper\dists\gradle-4.6-bin\4jp4stjndanmxuerzfseyb6wo
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :run
The current local time is: 11:28:46.173
Hello world!


BUILD SUCCESSFUL in 20s
2 actionable tasks: 2 executed
```

4. build.gradle 내용 및 plugin

1)Gradle에는 build.gradle라는 file에 build에 대한 처리를 작성한다.

2)이 build file의 기본에 대해 설명한다.

-이러한 Gradle 명령으로 수행하는 처리는 "build file"라는 file에 작성된 내용을 바탕으로 실행된다.

```
/*
 * This file was generated by the Gradle 'init' task.
 *
 * This generated file contains a sample Java project to get you started.
 * For more details take a look at the Java Quickstart chapter in the Gradle
 * user guide available at
 https://docs.gradle.org/4.8/userguide/tutorial_java_projects.html
 */

plugins {
    // Apply the java plugin to add support for Java
    id 'java'

    // Apply the application plugin to add support for building an application
    id 'application'
}

// Define the main class for the application
mainClassName = 'App'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

-이 file 내용을 보면서 build file의 내용을 확인한다.

-build.gradle는 Groovy로 작성되어 있다.

-Groovy는 Java와 마찬가지로, //와 / * */으로 주석을 입력할 수 있다.

3)java plugin 추가

```
    apply plugin: 'java'

    -처음에 apply plugin:라는 것은 Gradle plugin을 사용하기 위한 것이다.
    -java는 Java program을 위한 기능을 제공하는 plugin이다.
    -compileJava이라는 tast를 사용하는 것은 사실 java plugin에서 제공하는 것이다.

4)application plugin 추가

    apply plugin: 'application'

    -또 다른 plugin이 추가되어 있다.
    -이 application은 응용 program에 대한 기능을 제공하는 plugin이다.
    -run 응용 program을 실행한다는 것은 이것이 application plugin에 의해 제공되는 task이다.

5)main class 이름

    mainClassName = 'App'

    -이것은 application plugin으로 사용되는 것으로, main class를 지정한다.
    -run으로 응용 program을 실행할 수 있었던 것도 이 mainClassName main class가 지정되어
    있었기 때문이다.

6)java plugin
    -project에서 java plugin을 사용하려면 build.gradle file에 다음과 같이 설정하면 된다.

    apply plugin: 'java'

7)기본 project layout

    directory(Directory)          설명
    src/main/java                 Java source code를 관리하는 directory.
    src/main/resources            resource을 관리하는 directory.
    src/test/java                 test Java source를 관리하기 위한 directory.
    src/test/resources            test resource를 관리하기 위한 directory.
    src/sourceSet/java            Java source를 위한 특정한 source set
    src/sourceSet/resources       Java resource를 위한 특정한 source set

    -여기서 배포 시 test source code가 같이 배포되지 않는다.

8)저장소(Repositories)
    -build.gradle에 기술된 내용에는 "dependency library"에 대한 기술이 있었다.
    -Gradle에는 program으로 필요한 library를 자동으로 download하고 통합하는 기능이 있다.
    -따라서 중요해지는 것은 "저장소(repository)"이다.
    -저장소라는 것은 각종 program들이 저장되는 위치이다.
    -이 저장소는 "어떤 저장소를 사용하는지"를 build file에 작성하여 설정할 수 있다.

        repositories {
            ..... 저장소 설정 .....
        }

    -이것이 저장소를 지정하기 위한 기술이다.
    -이 {} 안에 저장소를 설정하는 문장을 작성한다.
    -online으로 access하여 사용할 수 있는 저장소로는 Gradle은 대체로 다음 두 개의 저장소 service를
    이용한다.

    -Maven 중앙 저장소
        --mavenCentral()
        --이것은 Apache Maven 중앙 저장소를 이용하기 위한 것이다.
        --Gradle은 중앙 저장소를 그대로 사용할 수 있다.
```

-JCenter 저장소
--jcenter()
--이 밖에 JCenter라는 저장소도 사용할 수 있다.
--이것은 Maven과 Gradle 등 각종 build 도구에서 사용할 수 있는 공개 저장소이다.
--이를 이용하려면 jcenter()을 repositories에 기술해 둔다.
--mavenCentral()와 jcenter()는 Gradle method이다.
--이러한 repositories 안에서 호출하여 지정된 저장소를 사용할 수 있다.

9)의존 library (dependencies)
-저장소에서 필요한 library를 사용하는데 사용할 수 있는 것이 dependencies라는 문이다.
-이것은 다음과 같이 기술된다.

```
dependencies {
    ..... library 지정 .....
}
```

-이 {} 안에 종속 library에 대한 기술을 한다.
-여기에서는 2개의 문장이 기술되어 있다.

-compile시 의존 library

```
compile 'com.google.guava:guava:23.0'
```

--이것은 compile시에 사용하는 library를 지정하고 있다.
--compile ~ 이라고 기술하는 것으로 그 library가 compile 시에 참조되는 것을 나타낸다.

-test compile시 의존 library

```
testCompile 'junit:junit:4.12'
```

--이것은 test compile (unit test의 program을 compile)에 사용하는 library를 지정한다.
--testCompile ~라고 기술하는 것으로 그 library가 test compile 시에 참조되는 것을 나타낸다.

-이 외에도 다양한 처리를 수행할 때 참조하는 종속 library를 지정할 수 있다.
-하나 기억해야 할 것은 classpath의 지정이다.

```
classpath '... library ...'
```

-이렇게 하면 지정된 library를 class 경로에 추가할 수 있다.
-compile에서 실행시까지 의존하는 library 지정에 사용한다.

10)Library 지정
-여기에서는 2개의 library를 사용하고 있지만, 이것들은 각각 다음과 같이 값을 지정한다.

```
'com.google.guava:guava:32.0'
'junit:junit:4.12'
```

-이러한 작성법을 정리하면 대략 다음과 같이 된다.

```
'group : 이름 : version'
```

-group은 그 library가 속해 있는 기업 및 단체 등을 나타낸다.
-예를 들어 기업에서 만드는 것은 그 기업 group을 정해지고, 그것이 지정된다.
-이름은 library의 이름이다.
-이상을 바탕으로 하여 여기에서 사용하는 library가 무엇인지 살펴 보도록 한다.

```
'com.google.guava:guava:22.0'
group : com.google.guava
이름 : guava
version : 23.0
```

'junit:junit:4.12'
group : junit
이름 : junit
version : 4.12

-개별적으로 지정하는 방법
--이와 같이 하나의 text에 library 정보를 정리한 작성법은 간단하지만, 보기 어려운 것 같은 느낌도 든다.
--Gradle에는 이 밖에 group, 이름, version을 각각 분리하여 작성하는 방법도 가능하다.

group:'그룹', name:'이름', version:'버전'
--이런 식으로 작성한다.
--예를 들어, 샘플로 준비되어 있는 library의 지정하려면 다음과 같다.

compile 'com.google.guava:guava:23.0' ==> compile group:'com.google.guava', name:'guava', version:'23.0'

testCompile 'junit:junit:4.12' ==> testCompile group:'junit', name:'junit', version:'4.12'

-이렇게 작성하는 것이 하나 하나의 값이 명확하게 알아 볼 수 있다.
-하나의 텍스트로 정리하는 작성법은 쓰고 틀렸을 때 실수가 찾기 어려운 것이다.

5. Gradle Task 생성
1)Gradle는 task을 준비하고 실행하는 것이 기본이다.
2)task는 Groovy를 사용하여 작성할 수 있다.
3)task의 기본적인 생성 방법에 대해 설명한다.
4)task 정의
-Gradle은 명령에 의해 "task(task)"을 수행하는 program이다.
-지금까지 gradle compileJava라든지 gradle run와 같은 명령을 사용하였는데, 이들도 모두 "compileJava task 수행", "run task 수행"이다.

5)task 정의 기본
-이 task는 사용자가 정의할 수 있다.
-build file(build.gradle)에서 task의 처리를 기술해두면, 그것을 gradle 명령으로 호출 실행 시킬 수 있다.
-task는 다음과 같은 형태로 정의한다.

```
task task명 {
    ..... 수행할 처리 .....
}
```

-task는 "task"라는 키워드를 사용하여 정의한다.
-이 후에 task명을 작성하고, 그 다음에 {} 내에 task의 내용을 작성한다.
-task 선언는 다른 작성법도 있는데, 다음과 같은 작성도 가능하다.

```
task (task명) {...}
task ('task명') {...}
```

-이것으로 {} 안에 기술된 처리를 실행하는 작업을 정의할 수 있다.
-그럼 실제로 해 보도록 하자.
-build.gradle 아래 부분에, 아래와 같이 code를 추가한다.

```
task hello {
    println('Doing hello task')
}
```

-그리고 file을 저장하고, 명령 prompt 또는 terminal에서 다음과 같이 실행한다.

```
$ gradle hello
```

-이것으로 hello task가 실행된다.
-실행해 보면, println으로 출력하는 문장 이외에도 다양한 문장이 출력된다.

```
> Configure project :
Doing hello task

BUILD SUCCESSFUL in 0s
```

-이는 "quiet 모드"로 task를 수행하면 많은 부분이 사라진다.
-q 옵션을 지정하고 아래 같이 실행한다.

```
$ gradle -q hello
```

-이로 표시되는 출력은 상당히 심플하게 될 것이다.

```
Doing hello task
```

6)doFirst와 doLast
-task는 이렇게 task 후에 {} 부분에 처리를 쓰는 것만으로 만들 수 있다.
-사실 보통은 이런 작성법은 많이 쓰지 않는다.
-일반적인 task의 형태를 정리하면, 대체로 다음과 같은 형태가 된다.

```
task task명 {

    doFirst {
        ..... 수행할 처리 .....
    }

    doLast {
        ..... 수행할 처리 .....
    }
}
```

-task {} 에는 doFirst, doLast 라는 것이 준비된다.
-이것은 일종의 closure이다.
-이들은 각각 다음과 같은 기능을 한다.

```
doFirst : 최초에 수행하는 액션이다.
doLast : 최후에 수행 하는 액션이다.
```

-task는 준비된 "action"을 순서대로 실행해 나가는 역할을 한다.
-action이라는 것은 구체적으로 수행하는 처리의 "실행 단위" 같은 것이다.
-task 중에는 여러 가지 action이 준비되어 있고, 그것이 순차적으로 실행된다.
-doFirst과 doLast는 그 action의 최초, 최후에 실행한다.
-즉, "task의 기본적인 처리 등이 있을 때는 그 전에 실행하는 것과 후에 실행하는 것"을 이렇게 준비한다.
-이는 2개를 세트로 준비할 필요는 없다.
-어느 한쪽만으로도 괜찮다.
-그러면 실제로 간단한 예를 움직여 보자.

```
task hello {
    doLast {
        println('hello task\'s doLast.')
    }
    doFirst {
        println('hello task\'s doFirst.')
    }
}
```

-아래 목록 아래처럼 hello 작업을 다시 시도한다.
-그리고 gradle hello를 실행한다.

```
$ gradle -q hello
```

-그러면 다음과 같이 출력된다.

```
> Task :hello
hello task's doFirst.
hello task's doLast.

BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
```

-샘플은 doLast 먼저, doFirst가 후에 쓰여져 있지만, 실행 결과를 보면, 우선 doFirst이 실행된 후에 doLast가 실행되고 있는 것을 알 수 있다.

7)매개 변수 전달
-task는 수행할 때 필요한 값을 매개 변수로 전달할 수 있다.
-단순히 작업 처리 중 변수를 사용하면 된다.
-예를 들어, 다음과 같다.

```
task msg {
    println("you typed: " + x)
}
```

-여기에서는 println으로 변수 x의 값을 표시하고 있다.
-이 변수 x에 값을 설정하려면 gradle 명령을 실행시에 다음과 같이 입력한다.

```
$ gradle msg -Px=123

> Configure project :
you typed: 123

BUILD SUCCESSFUL in 0s
```

-이렇게 -P 후에 변수명을 지정하고 그 뒤에 등호로 값을 지정한다.
-변수 hoge에 123 값을 전달 싶다면 -Phoge=123 식으로 기술하면 된다.
-다음 예제를 보자.
-이는 숫자를 전달하여 그 숫자까지를 더하는 예제이다.

```
task hello {
    doLast {
        def n = max.toInteger()
        for(def i in 1..n){
            println("No," + i + " count.")
        }
        println("--end.")
    }
}
```

-task는 "max"라는 변수를 사용하여 최대 값을 지정한다.
-예를 들어,

```
$ gradle hello -Pmax=5
```

-이렇게 실행하면, 다음과 같이 메시지가 출력된다.

```
> Task :hello
```

```
No,1 count.
No,2 count.
No,3 count.
No,4 count.
No,5 count.
--end.
```

-여기에서는 def n = max.toInteger()와 같이 하여 변수 max를 정수 값으로 변환한 것을 변수 n에 대입하고 있다.
-그리고 이 n 값을 이용하여 for으로 반복 계산을 실시하고 있다.
-이런 상태로 매개 변수를 사용하여 쉽게 값을 변수로 전달할 수 있다.

8)다른 Task 호출 및 종속
-다른 task 호출
--task에서 다른 task를 호출해야 하는 경우도 있다.
--예를 들어 아래와 같은 task가 있다고 해보자.

```
task a {……}
task b {……}
```

--a와 b라는 task가 있을 때, task a에서 task b를 호출하려면 어떻게 해야 하는가?
--Java적으로 생각한다면 아래와 같이 호출하면 될거라 생각할 것이다.

```
b()
```

--하지만 이렇게는 작동을 하지 않는다.
--그럼 어떻게 해야 하는가?
--그것은 "tasks"에 있는 작업 객체 안의 method를 호출하여 수행한다.
--작업하는 것은 모든 tasks라고 객체에 정리하고 있다.
--이것은 예를 들어 a, b라는 task가 있다면 tasks.a과 tasks.b로 지정할 수 있다.
--이 task 객체 안에 있는 "execute"라는 method를 호출하여 task를 수행할 수 있다.

```
tasks.a.execute()
tasks.b.execute()
```

--이런 식으로 실행하여 task a, b를 호출한다.
--다음은 간단한 예이다.

```
task hello  {
    doFirst {
        println("*** start:hello task ***")
        tasks.aaa.execute()
     }
    doLast {
        tasks.bbb.execute()
        println("*** end:hello task ***")
    }
}

task aaa {
    doLast {
        println("<< This is A task! >>")
    }
}

task bbb {
    doLast {
        println("<< I'm task B!! >>")
    }
}
```

--gradle hello와 같이 hello task를 실행해 보면, 아래와 같이 출력이 된다.

```
> Task :aaa
<< This is A task! >>

> Task :bbb
<< I'm task B!! >>

> Task :hello
*** start:hello task ***
*** end:hello task ***

Deprecated Gradle features were used in this build, making it incompatible with Gradle 5.0.
See https://docs.gradle.org/4.8/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 0s
3 actionable tasks: 3 executed
```

--여기에서는 hello의 doFirst 안에서 aaa, doLast에서 bbb를 호출하고 있다.
--출력되는 text가 호출되는 순서를 잘 확인한다.

-종속 task 지정
--어떤 task를 수행할 때, 다른 작업 수행이 필수적인 경우도 있다.
--이러한 경우에는 "dependsOn '라는 기능을 사용할 수 있다.
--이는 다음과 같이 작성한다.

```
task task명 (dependsOn : 'task') {
    ..... 중략 .....
}
```

--또는 다음과 같은 작성도 가능한다.

```
task task명 {
    dependsOn : 'task'
    ..... 중략 .....
}
```

--이와 같이 기술해 두면 작업이 호출될 때, 먼저 dependsOn에 지정된 작업을 수행하고 그것이 끝난 후에 task의 본 처리를 수행한다.
--여러 task를 지정해야 하는 경우는 task명을 배열로 지정한다.
--['a', 'b', 'c']와 같은 식이다.
--이 경우 최초에 작성한 task부터 실행된다.
--다음 예제를 보자.

```
task hello(dependsOn:['aaa', 'bbb'])  {
    doFirst {
        println("*** start:hello task ***")
    }
    doLast {
        println("*** end:hello task ***")
    }
}

task aaa {
    doLast {
        println("<< This is A task! >>")
```

```
    }
}

task bbb {
    doLast {
        println("<< I'm task B!! >>")
    }
}
```

--이를 "gradle hello"로 실행한다.
--다음과 같이 출력이 된다.

```
> Task :aaa
<< This is A task! >>

> Task :bbb
<< I'm task B!! >>

> Task :hello
*** start:hello task ***
*** end:hello task ***

BUILD SUCCESSFUL in 0s
3 actionable tasks: 3 executed
```

--최초에 aaa task, bbb task이 실행되면, 이후에 hello task가 호출되었는지 알 수 있다.
--dependsOn에 의해, aaa, bbb가 종속 task가 되는 test가 실행된 후가 아니면 hello가 실행되지 않게 된다.

6. DefaultTask class 사용
1)Gradle은 표준으로 DefaultTask class가 준비되어 있으며,이를 상속한 Task class가 준비되어 있다.
2)이러한 목적을 위해 task 생성에 대해 설명한다.
3)DefaultTask 상속 class
-Gradle은 표준으로 다양한 task가 포함되어 있는데, 이것들은 "DefaultTask"라는 class를 상속한 class로 준비되어 있다.
-이 DefaultTask 상속 class는 자신의 task을 만들어 커스텀 마이징을 할 수 있다.
-우선은 "DefaultTask 상속 class"가 어떤 것인지 직접 만들어 본다.
-이 class는 다음과 같은 형태로 정의된다.

```
class class extends DefaultTask {
    ..... 필드 .....

    void method(인수) {
        ..... 처리 .....
    }

    @TaskAction
    void method() {
        ..... 처리 .....
    }
}
```

-class는 DefaultTask라는 class를 상속하여 만든다.
-이 class 내에 task로 수행할 처리를 method로 제공한다.
-이 method에는 @TaskAction annotation을 붙여 둔다.
-그러면 task로 실행되었을 때, 이 method가 호출된다.
-task로 사용하는 각종의 값은 field로 사용할 수 있어야 한다.
-이것은 그대로 이용해도 되지만, 외부에서 사용하는 경우는 private field로 설정하여 접근을 위한 method를 따로 제공하는 것이 스마트하다.

-다음의 예제를 보자.
-아래의 코드를 build.gradle 안에 작성한다.

```
class Calc extends DefaultTask {
    private int num
    private String op

    void num(p1){
        num = p1
    }

    void op(p1){
        op = p1
    }

    @TaskAction
    void calc() {
        switch(op) {
        case 'total':
            int total = 0
            for(def i in 1..num) {
                total += i
            }
            println("total: ${total}")
        break

        case 'count':
            for(def i in 1..num) {
                println("NO, ${i}")
            }
        break

        default:
            println('not found operator...')
        }
    }
}
```

-Calc class에는 calc 라는 task 액션을 준비하고 있다.
-여기에서 num와 op의 값에 따라 총의 계산과 수치 계산을 하고 있다.

4)Calc class를 지정한 task
-그럼, DefaultTask 상속 class를 이용하는 task는 어떻게 작성할 수 있을까?
-그 작성법은 아래와 같다.

```
task task(type : class) {
    ..... 수행할 처리 .....
}
```

-task의 ()에는 인수로 'type' 값을 준비하고, 이 type에서 사용하는 class를 지정하다.
-실제로 수행하는 처리에는 사용하는 class에 필드로 준비되어 있는 변수에 값을 할당하는 처리를 준비해 둔다.
-이렇게 하면, class의 각 필드의 값을 변경하여 task method를 실행할 수 있다.
-다음의 간단한 예를 보자.

```
task total(type:Calc) {
    group 'javaexpert'
    description 'Task for calculating total.'
    num 100
    op 'total'
```

```
}

task count(type:Calc) {
    group 'javaexpert'
    description 'Task for count number.'
    num 10
    op 'count'
}
```

-여기에서는 앞 전에 Calc class를 type에 지정한 total, count라는 두 가지 task를 만들었다.
-gradle total라고 실행하면 100까지의 합계가 계산된다.

```
$ gradle total

> Task :total
total: 5050


BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
```

-"gradle count"라고 실행하면 1 ~ 10까지의 숫자를 순서대로 출력한다.

```
$ gradle count

> Task :count
NO, 1
NO, 2
NO, 3
NO, 4
NO, 5
NO, 6
NO, 7
NO, 8
NO, 9
NO, 10


BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
```

-여기에서는 task의 인수로 (type:Calc)을 지정하고 있다.
-그러면 Calc class의 task를 수행하는 task으로 정의될 수 있다.
-여기에는 다음과 같은 문장이 작성되어 있다.

```
group 그룹명
description 설명
num 정수
op 조작의 유형
```

-이들은 모두 상속의 Calc class에 있는 method를 호출하는 것이다.
-group과 description은 DefaultTask class에 있는 것으로, 각 그룹명과 설명 텍스트를 설정한다.
-그리고 Calc class에 준비되어 있는 num와 op으로 계산의 정수 값와 작업의 유형을 지정하고 있다.
-이런 식으로 task로 정의된 가운데, type 지정한 class의 메서드를 호출하여 필요한 설정을 한다.
-그러면, 그 설정이 된 후에 태스크 액션이 수행된다.

5)JavaExec class 이용

-DefaultTask 상속 class를 사용한 task 작성의 기본이 알았으니, Gradle에 제공되는 주요 DefaultTask 상속 class에 대한 사용법을 살펴보기로 한다.
-JavaExec class

--JavaExec는 Java program의 실행을 위한 task를 구현하는 class이다.
--이 class에는 실행에 필요한 각종 method가 준비되어 있다.

```
main "class"
```

--실행하는 class를 지정하는 method이다.
--class명을 텍스트로 인수로 지정하여 호출한다.

```
classpath "텍스트"
```

--여기에서는 실행 시에 classpath로 지정하는 텍스트를 설정한다.
--디폴트 classpath 로 좋다면, sourceSets.main.runtimeClasspath라는 값을 지정해 둔다.

```
args "Iterator"
args "값1, 값2, ..."
```

--인수로 전달할 정보를 지정하는 것이다.
--이것은 Iterator로 정리하여 준비할 수 있으며, 부정 인수로 필요한 값을 개별적으로 인수에 지정할 수도 있다.

```
jvmArgs "Iterator"
jvmArgs "값1, 값2, ..."
```

--여기에서는 Java 가상 machine에 전달되는 인수를 지정한다.
--이것도 역시 Iterator와 같은 이상한 인수가 준비되어 있다.

```
workingDir "텍스트"
```

--작업 directory를 지정하는 것이다.
--인수에는 설정하고자 하는 directory의 경로를 지정한다.
--이것은 project folder에서의 상대 경로로 지정한다.
--그럼, 이 JavaExec을 이용한 task의 예를 들어 둔다.

```
task appRun(type: JavaExec) {
    group 'devkuma'
    description 'exec App class.'
    main 'App'
    classpath sourceSets.main.runtimeClasspath

    doFirst {
        println()
        println('---------- Start ----------')
        println()
    }
    doLast {
        println()
        println('----------- end -----------')
        println()
    }
}
```

--여기에서는 App class를 실행하는 appRun 작업을 만들었다.
--gradle appRun라고 실행하면 콘솔에 다음과 같이 출력된다.

```
$ gradle appRun

    > Task :appRun

    ---------- Start ----------
```

```
        Hello world.

        ---------- end ----------
```

--Gradle에서의 실행은 다양한 출력이 있어 실행 결과를 알아보기 어렵기에, doFirst과 doLast으로 텍스트를 출력해서 한눈에 "이것이 실행 내용"이라고 알 수 있도록 해본다.
--여기에서는 다음과 같이 ExecJava 설정을 하고 있다.

```
    main 'App'
    classpath sourceSets.main.runtimeClasspath
```

--일단 main과 classpath 만 준비한다.
--이 2개는 디폴트로 값이 설정되어 있지 않아서 생략할 수도 없다.
--그 외의 것은 디폴트인 상태로도 문제가 없을 것이다.

6)커멘드를 실행하는 Exec 이용
-project에서 작성하고 있는 Java program이 아닌, 다른 program을 실행하려는 경우도 있다.
-이러한 경우에 사용되는 것이 Exec class이다.
-Exec class는 커멘드 라인에서 명령을 실행하는 기능을 한다.
-이에 몇 가지 method가 준비되어 있으며, 명령 실행에 관한 설정을 할 수 있게 되어 있다.

```
    commandLine "실행 명령", "인수"...
```

-실행할 명령의 내용을 지정한다.
-첫번째 인수에 커멘드를 작성하고, 그 이후에 옵션 등을 인수로 지정한다.

```
    workingDir "텍스트"
```

-이것은 앞에서 이미 설명 했었다.
-작업 directory를 지정하는 것이다.

```
    args "Iterator"
    args "값1, 값2, ..."
```

-이것도 앞에서 이미 등장 했었다.
-인수로 전달할 정보를 지정하는 것이다.
-이것은 Iterator로 정리하여 준비할 수 있으며, 부정 인수로 필요한 값을 개별적으로 인수 지정할 수도 있다.

8)Windows에서 실행
-일단 이것만 알고 있으면 명령의 실행은 충분히 있을 것이다.
-그러면 실제로 간단한 예를 들어 보겠다.

```
    task javaVer(type:Exec) {
        group 'javaexpert'
        description 'print java version.'
        workingDir '.'
        commandLine 'cmd'
        args '/c', 'java.bat'
        doFirst {
            println()
            println('***** Java Version *****')
        }
    }
```

-먼저 실행하는 명령으로 간단한 배치 file을 만들어 둔다.
-여기에서는 Windows에서 실행하기 위한 전제로 설명을 한다.
-project folder에 "java.bat"라는 이름으로 file을 준비한다.
-그리고 다음과 같이 작성해 둔다.

```
java.exe -version
```

-보면 알 수 있듯이 Java 버전을 출력하는 명령을 실행하고 있다.
-이 배치 file을 실행하는 작업을 만들 수 있다.
-작성한 후에 gradle javaVer라고 실행해 본다.

```
$ gradle javaVer

> Task :javaVer

***** Java Version *****

C:\GradleHome\GradleApp>java.exe -version
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

-위의 결과와 같이 버전이 출력된다.
-설치된 JDK를 사용하여 표시는 달라질 것이지만, 대체로 이런 출력이 된다.
-여기에서는 다음과 같이 실행 명령 설정을 한다.

```
workingDir '.'
commandLine 'cmd'
args '/c', 'javaver.bat'
```

-이것으로 cmd /c java.bat라는 명령이 실행된다.
-그러면 java.bat에 작성된 java.exe -version가 실행되어 Java 버전 정보가 출력된다.

9)Mac 및 리눅스 계열에서 실행
-Window가 아닌 Mac 및 리눅스 계열에서 실행되는 예제는 아래와 같다.

```
task javaVer(type:Exec) {
    group 'javaexpert'
    description 'print java version.'
    workingDir '.'
    commandLine './javaver'
    doFirst {
        println()
        println('***** Java Version *****')
    }
}
```

-javaver file을 생성하여 아래와 같이 file을 저장하고, chmod로 실행 권한을 부여한다.

```
java -version
```

-출력은 Window와 동일하다.

10)file을 복사하는 Copy 이용
-program 실행 관계 외에도, 비교적 기억해 두면 도움이 되는 것으로 file 관련 class를 몇 가지 살펴보고 설명한다.
-Copy class
--Copy class는 그 이름과 같이 file을 복사할 수 있는 기능을 제공한다.
--여기에는 다음과 같은 method가 준비되어 있다.

```
from "원본 경로"
```

--복사할 원본 file와 folder의 경로를 텍스트로 지정한다.

into "대상 경로"

--복사할 file이나 folder의 경로를 텍스트로 지정한다.

include "패턴", ...

--복사 대상에 포함 할 file을 ANT 스타일 패턴이라는 형식으로 지정한다. 이것은 와일드 카드로 지정되는 패턴이다.

exclude "패턴", ...

--복사 대상에서 제외 file을 ANT 스타일 패턴에서 지정하는 것이다.
--그럼, Copy 간단한 사용 예를 아래에 들어 둔었다.

```
task copyJava(type: Copy) {
    group 'javaexpert'
    description 'backup java files.'
    from 'src/main/java'
    into '../java_backup'
}
```

--이것은 main 안에 java folder를 project folder 외부에 복사한다.
--gradle copyJava라고 실행해 본다.

```
$ gradle copyJava

BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
```

--java folder가 project folder와 같은 위치에 "java_backup"라는 이름으로 복사된다.

-Delete class
--Delete class는 file과 folder를 삭제하는 것이다. 여기에는 다음의 method가 준비되어 있다.

delete "file", ...

--삭제 대상 file을 지정한다.
--이것은 file 경로 텍스트이다.

7. Web Application Project

1)일반적인 Java application은 gradle init으로 생성하였다.
2)그러면 Web application은 어떻게 생성하고 구성되는지 설명한다.
3)Web application 생성
-Web application은 program의 구성도 또한 실행 방법도 다르다.
-Java class뿐만 아니라 HTML 및 JSP도 사용하므로, 그 file들을 하나로 모아서 war file을 작성해야 한다.
-또한 실행은 서블릿 컨테이너 (이른바 Java 서버)도 필요하다.
-이렇게 생각한다면, 일반 Java application 생성 방법으로는 되지 않는다고 생각할 것이다.
-그러면 일단 실제 project를 만들면서 Web application 개발 단계를 설명한다.
-우선 project를 만든다.
-명령 prompt 또는 terminal을 시작하고 적당한 위치에 project directory를 준비한다.

```
$ mkdir GradleWebApp
$ cd GradleWebApp
```

-여기에서는 "GradleWebApp"라는 directory를 만들고, 그 안으로 이동하고 Gradle의 초기화를 실시한다.

```
$ gradle init --type java-application

BUILD SUCCESSFUL in 0s
2 actionable tasks: 2 executed
```

-이것으로 생성이 된다.
-그런데 보면 알 수 있듯이, 극히 일반 Java application과 같은 방식이다.

```
$ tree
폴더 PATH의 목록입니다.
볼륨 일련 번호는 1EB0-A263입니다.
C:.
├─.gradle
│  ├─4.8
│  │  ├─fileChanges
│  │  ├─fileHashes
│  │  └─taskHistory
│  └─buildOutputCleanup
├─gradle
│  └─wrapper
└─src
   ├─main
   │  └─java
   └─test
      └─java
```

4)Web 응용 program folder 구성
-그럼, Web application의 folder 구성을 준비한다.
-Web application의 folder는 다음과 같은 형태로 구성된다.
-GradleWebApp folder에서 아래와 같이 수동으로 각각의 folder를 생성한다.

```
└── src
      └── main
            ├── java
            ├── resources
            └── webapp
                  └── WEB-INF
```

-우선 application program 관련에 대해 정리한다.
-Web application 및 일반 Java application의 가장 큰 차이점은 main folder에 있는 folder의 구성이다.
-여기에는 아래와 같이 3개의 folder가 있다.

-java folder
--익숙한 Java source code file을 배치하기 위한 folder이다.
--Servlet 등은 여기에 source code를 배치한다.

-webapp folder
--이는 Web application의 공개되는 folder로 static web 자원을 두는 곳이다.
--JSP file이나 HTML file 등은 여기에 배치한다.
--또한 여기에는 WEB-INF folder를 준비하고 그 안에 web.xml을 배치한다.

-resources folder
--Web application의 program에 필요한 자원을 제공한다.
--이는 Web page에 표시할 image file 등은 아니다(그것들은 webapp 이다).
--예를 들어, JPA 등을 이용할 때 준비하는 persistence.xml 같은 program이 필요로 하는 resource file이다.

-그럼, 만든 GradleWebApp 안에 이러한 folder들을 준비한다.
-main 안에 새롭게 webapp와 resources를 만들고 webapp에 추가로 WEB-INF를 생성한다.

5)web.xml
-Web application의 각종 정보를 작성하는 web.xml를 준비한다.
-여기에서는 최신 버전으로 Servlet 3.1를 사용한다.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        version="3.1">

</web-app>
```

6)서블릿 준비
-Web application에서 표시하는 Web 페이지로 간단한 서블릿을 작성한다.
-main folder에 있는 java folder > com folder > javasoft folder > web folder를 만든다.
-그리고 web 에 SampleServlet.java라는 이름으로 Java source code file을 작성한다.
-source code의 내용은 아래와 같다.

```
package com.javasoft.web;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class SampleServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;

   @Override protected void doGet(HttpServletRequest request,
   HttpServletResponse response)
      throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Servlet</title></head>");
      out.println("<body><h1>Sample Servlet</h1>");
      out.println("<p>Welcome to Sample Servlet page!</p>");
      out.println("</body></html>");
   }
}
```

-여기에서는 @WebServlet 어노테이션을 사용하여 /hello 서블릿을 주소를 설정하고 있다.
-그외에 webapp folder에 JSP file이나 HTML file 등을 배치 해두면 된다.

7)최종 folder 및 file 구조
-최종으로 file을 모두 만들면 같이 된다.

```
├── build.gradle, 그외 생략
└── src
        ├── main
        │       ├── java
        │       │       ├── App.java
```

```
                |       |       └── com
                |       |           └── javasoft
                |       |               └── web
                |       |                   └── SampleServlet.java
                |       ├── resources
                |       └── webapp
                |           └── WEB-INF
                |               └── web.xml
                └── test
                    └── java
                        └── AppTest.java
```

8)build.gradle 작성

-Web application을 위한 build.gradle를 작성한다.

-이번에는 일반적인 Java 응용 program과 여러가지가 다른 부분이 있다.

-build.gradle

--우선 아래에 build.gradle의 전체 source code 이다.

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'gretty'

buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        classpath group:'org.akhikhl.gretty', name:'gretty-plugin', version:'+'
    }
}

repositories {
    jcenter()
}

dependencies {
    testCompile group:'junit', name:'junit', version:'4.12'
}
```

--우선, 전체 code를 작성하면 실제로 동작시켜 본다. 다음과 같이 명령을 실행한다.

```
$ gradle war

Download https://jcenter.bintray.com/org/akhikhl/gretty/gretty-plugin/maven-metadata.xml
Download https://jcenter.bintray.com/org/akhikhl/gretty/gretty-plugin/0.0.24/gretty-plugin-0.0.24.pom
Download https://jcenter.bintray.com/org/codehaus/groovy/groovy-all/1.8.6/groovy-all-1.8.6.pom
Download https://jcenter.bintray.com/org/akhikhl/gretty/gretty9-plugin/0.0.24/gretty9-plugin-0.0.24.pom
Download https://jcenter.bintray.com/org/eclipse/jetty/jetty-util/9.1.0.v20131115/jetty-util-9.1.0.v20131115.pom
Download
```

https://jcenter.bintray.com/org/akhikhl/gretty/gretty-plugin-commons/0.0.24/gretty-plugin-commons-0.0.24.pom
Download https://jcenter.bintray.com/org/eclipse/jetty/jetty-project/9.1.0.v20131115/jetty-project-9.1.0.v20131115.pom
Download https://jcenter.bintray.com/org/eclipse/jetty/jetty-parent/20/jetty-parent-20.pom
Download https://jcenter.bintray.com/org/bouncycastle/bcprov-jdk16/1.46/bcprov-jdk16-1.46.pom
Download https://jcenter.bintray.com/ch/qos/logback/logback-classic/1.1.2/logback-classic-1.1.2.pom
Download https://jcenter.bintray.com/ch/qos/logback/logback-parent/1.1.2/logback-parent-1.1.2.pom
Download https://jcenter.bintray.com/commons-io/commons-io/2.4/commons-io-2.4.pom
Download https://jcenter.bintray.com/org/apache/commons/commons-lang3/3.3.2/commons-lang3-3.3.2.pom
Download https://jcenter.bintray.com/org/apache/commons/commons-parent/25/commons-parent-25.pom
Download https://jcenter.bintray.com/org/apache/commons/commons-parent/33/commons-parent-33.pom
Download https://jcenter.bintray.com/ch/qos/logback/logback-core/1.1.2/logback-core-1.1.2.pom
Download https://jcenter.bintray.com/org/slf4j/slf4j-api/1.7.6/slf4j-api-1.7.6.pom
Download https://jcenter.bintray.com/org/slf4j/slf4j-parent/1.7.6/slf4j-parent-1.7.6.pom
Download https://jcenter.bintray.com/commons-io/commons-io/2.4/commons-io-2.4.jar
Download https://jcenter.bintray.com/org/akhikhl/gretty/gretty-plugin/0.0.24/gretty-plugin-0.0.24.jar
Download https://jcenter.bintray.com/org/apache/commons/commons-lang3/3.3.2/commons-lang3-3.3.2.jar
Download https://jcenter.bintray.com/ch/qos/logback/logback-core/1.1.2/logback-core-1.1.2.jar
Download https://jcenter.bintray.com/org/slf4j/slf4j-api/1.7.6/slf4j-api-1.7.6.jar
Download https://jcenter.bintray.com/org/eclipse/jetty/jetty-util/9.1.0.v20131115/jetty-util-9.1.0.v20131115.jar
Download https://jcenter.bintray.com/org/bouncycastle/bcprov-jdk16/1.46/bcprov-jdk16-1.46.jar
Download https://jcenter.bintray.com/ch/qos/logback/logback-classic/1.1.2/logback-classic-1.1.2.jar
Download https://jcenter.bintray.com/org/codehaus/groovy/groovy-all/1.8.6/groovy-all-1.8.6.jar
Download https://jcenter.bintray.com/org/akhikhl/gretty/gretty9-plugin/0.0.24/gretty9-plugin-0.0.24.jar

```
Download
https://jcenter.bintray.com/org/akhikhl/gretty/gretty-plugin-commons/0.0.24/gretty-plugin-commons-0.0.24.jar

BUILD SUCCESSFUL in 12s
2 actionable tasks: 2 executed
```

--이는 Web 응용 program을 war file로 생성하기 위한 것이다.
--실행하면 project의 build folder에 있는 libs folder에 "GradleWebApp.war"라는 War file이 생성된다.
--이어 응용 program의 동작 체크를 한다. 다음과 같이 실행한다.

```
$ gradle run
```

--실행되면 Jetty가 download되고, 바로 기동된다.
--그러면 아래 주소에 액세스해 본다.
--서블릿에 액세스되고 화면이 나타난다.

```
http://localhost:8080/GradleWebApp/hello


Sample Servlet
Welcome to Sample Servlet page!
```

-사용 플러그인
--그럼, build.gradle에 플러그인 내용을 확인해 보자.
--여기에서는 3개의 플러그인이 포함되어 있다.

```
apply plugin : 'java'
apply plugin : 'war'
apply plugin : 'gretty'
```

--최초의 "java"는 이미 익숙할 것이다.
--Java program에 대한 compileJava 등의 task를 제공하는 플러그이다.
--"war" 플러그인은 이름 그대로 war file로 패키징하기 위한 플러그인이다.
--이는 Web 응용 program에서 필수 플러그인이라고 할 수 있다.
--앞에서 gradle war는 이 플러그인에서 제공하는 것이다.
--"gretty"는 Jetty 서블릿 컨테이너를 이용하기 위한 플러그인이다.
--이 Gretty는 Groovy에서 Jetty를 이용하기 위한 것으로, 앞에서 실행했던 gradle run으로 Jetty를 시작하고 이 Web application이 기동된 것이다.

-buildscript와 dependencies
--build.gradle에는 낯선 문장이 추가되어 있다.
--그것은 "buildscript"라는 것이다.
--이것은 다음과 같은 형태로 되어 있다.

```
buildscript {
    repositories {
        ..... 저장소 .....
    }

    dependencies {
        ..... package 지정 .....
    }
}
```

--이 buildscript라는 것은 build script를 위한 것이다.
--build script라는 것은 그 이름대로 build를 위해 실행되는 script이다.
--Gradle에 있는 기능 그대로 build를 할 경우에는 이 buildscript이 필요없다.
--이것은 build할 때 외부 library 등을 사용하는 경우에 필요하다.

--이번에는 Gretty 플러그인을 사용하고 있지만, 이것은 Gradle 표준이 아니라 외부 library이다.
--따라서 build시 어떤 저장소에서 어떤 library를 사용할 것인지를 지정해야 한다.
--그 부분이 buildsript에 있는 repositories과 dependencies이다.
--이 buildscript 이 후에 repositories과 dependencies이 있지만, 이것은 build 처리 이외의 곳에서 사용되는 것이다.
--여기에서는 JUnit이 dependencies에 준비되어 있다.

8. Lab

1)Building Java Applications
-https://guides.gradle.org/building-java-applications/

2)Building Java Web Applications
-https://guides.gradle.org/building-java-web-applications/

3)Refer to
-http://www.vogella.com/tutorials/EclipseGradle/article.html

4)Gradle Project 생성
-In J2SE
-Package Explorer > right-click > New > Other > Gradle > Gradle Project > Next
-Project name : MyGraldeDemo
--Press the Finish button to create the project.
--This triggers the gradle init --type java-library command and imports the project.
--Press the Next > button to get a preview of the configuration before the projects created.

5)build.gradle
-아래 코드로 파일을 수정한다.

```
apply plugin: 'java'
apply plugin: 'application'

repositories {
    jcenter()
}

dependencies {
  compile 'com.google.guava:guava:20.0'  // Google Guava dependency
  testCompile 'junit:junit:4.12'         // JUnit dependency for testing
}

mainClassName = 'com.javasoft.MainApp'  // Main class with main method
```

-파일을 저장 후 Gradle Tasks view 에서 [Refresh Tasks for All Projects]를 클릭하여 project를 refresh한다.

6)Create Java Programs
-Remove the java files, generated by the gradle build tool, from project and create new files as follows.
-src/main하위에 com.javasoft package 생성
-Create MainApp.java file.

```
package com.javasoft;

public class MainApp {
    public String sayHello() {
        return "Hello Gradle";
    }

    public static void main(String[] args) {
        MainApp app = new MainApp();
```

```
            System.out.println(app.sayHello());
        }
    }
```

-src/test/java 하위에 com.javasoft package 생성
-Create MainAppTest.java file under src/test/java folder and write the following JUnit testcase in it.

```
    package com.javasoft;

    import org.junit.Test;
    import static org.junit.Assert.*;

    public class MainAppTest {
        @Test
        public void testSayHello() {
            MainApp app = new MainApp();
            assertNotNull("Success", app.sayHello());
        }
    }
```

7)Build project
-In Gradle Tasks view/tab, right-click on the build folder/build task and select [Run Gradle Tasks] to build the java project.
-In Console view

```
    Working Directory: C:\WebHome\MyGradleDemo
    Gradle User Home: C:\Users\Instructor\.gradle
    Gradle Distribution: Gradle wrapper from target build
    Gradle Version: 4.3
    Java Home: C:\Program Files\Java\jdk1.8.0_162
    JVM Arguments: None
    Program Arguments: None
    Build Scans Enabled: false
    Offline Mode Enabled: false
    Gradle Tasks: build

    :compileJava UP-TO-DATE
    :processResources NO-SOURCE
    :classes UP-TO-DATE
    :jar UP-TO-DATE
    :startScripts UP-TO-DATE
    :distTar UP-TO-DATE
    :distZip UP-TO-DATE
    :assemble UP-TO-DATE
    :compileTestJava
    :processTestResources NO-SOURCE
    :testClasses
    :test
    :check
    :build

    BUILD SUCCESSFUL in 1s
    7 actionable tasks: 2 executed, 5 up-to-date
```

8)Run project
-In Gradle Tasks view, right-click on the application folder/run task and select [Run Gradle Tasks] to run the java project.
-In Console view

```
Working Directory: C:\WebHome\MyGradleDemo
Gradle User Home: C:\Users\Instructor\.gradle
Gradle Distribution: Gradle wrapper from target build
Gradle Version: 4.3
Java Home: C:\Program Files\Java\jdk1.8.0_162
JVM Arguments: None
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: run

:compileJava UP-TO-DATE
:processResources NO-SOURCE
:classes UP-TO-DATE
:run
Hello Gradle

BUILD SUCCESSFUL in 0s
2 actionable tasks: 1 executed, 1 up-to-date
```

9)Run JUnit testcase

-In Gradle Tasks view, right-click on the verification/test task and select [Run Gradle Tasks] to run the JUnit testcase.

-In Console view

```
Working Directory: C:\WebHome\MyGradleDemo
Gradle User Home: C:\Users\Instructor\.gradle
Gradle Distribution: Gradle wrapper from target build
Gradle Version: 4.3
Java Home: C:\Program Files\Java\jdk1.8.0_162
JVM Arguments: None
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: test

:compileJava UP-TO-DATE
:processResources NO-SOURCE
:classes UP-TO-DATE
:compileTestJava UP-TO-DATE
:processTestResources NO-SOURCE
:testClasses UP-TO-DATE
:test UP-TO-DATE

BUILD SUCCESSFUL in 0s
3 actionable tasks: 3 up-to-date
```

9. J2EE Lab

1)Create Gradle Project

-Project Explorer > right-click > New > Other > Gradle > Gradle Project > Next > Next

-Project name : HelloGradleWebApp > Next > Next > Finish

2)Configuring Gradle

-build.gradle

-This is the default content of build.gradle file created by Eclipse, and remove all the comments:

```
/*
 * This build file was generated by the Gradle 'init' task.
```

```
1728          *
1729          * This generated file contains a sample Java Library project to get you started.
1730          * For more details take a look at the Java Libraries chapter in the Gradle
1731          * user guide available at
              https://docs.gradle.org/4.3/userguide/java_library_plugin.html
1732          */
1733
1734         // Apply the java-library plugin to add support for Java Library
1735         apply plugin: 'java-library'
1736
1737         // In this section you declare where to find the dependencies of your project
1738         repositories {
1739             // Use jcenter for resolving your dependencies.
1740             // You can declare any Maven/Ivy/file repository here.
1741             jcenter()
1742         }
1743
1744         dependencies {
1745             // This dependency is exported to consumers, that is to say found on their
                compile classpath.
1746             api 'org.apache.commons:commons-math3:3.6.1'
1747
1748             // This dependency is used internally, and not exposed to consumers on their
                own compile classpath.
1749             implementation 'com.google.guava:guava:23.0'
1750
1751             // Use JUnit test framework
1752             testImplementation 'junit:junit:4.12'
1753         }
1754
1755        -You need to add the configuration for your application to become "WEB Application".
1756        -And can be run directly on Eclipse + Tomcat Plugin.
1757        -Refer to https://github.com/bmuschko/gradle-tomcat-plugin
1758
1759         apply plugin: 'java-library'
1760         apply plugin: 'war'
1761         apply plugin: 'com.bmuschko.tomcat'
1762
1763         repositories {
1764             mavenCentral()
1765         }
1766
1767         dependencies {
1768             def tomcatVersion = '8.5.16'
1769             tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
1770                    "org.apache.tomcat.embed:tomcat-embed-logging-juli:8.5.2",
1771                    "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
1772         }
1773
1774         tomcat {
1775             httpProtocol = 'org.apache.coyote.http11.Http11Nio2Protocol'
1776             ajpProtocol  = 'org.apache.coyote.ajp.AjpNio2Protocol'
1777         }
1778
1779         buildscript {
1780             repositories {
1781                 jcenter()
1782             }
1783
1784             dependencies {
```

```
            classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'
        }
    }
```

-Note that each time there is a change in build.gradle you need to update the project, using the tool of Gradle.
-build.gradle > right-click > Gradle > Refresh Gradle Project

3)Edit application structure
-In "src/main" folder, you need to create 2 sub folders are "resources" and "webapp".
--src/main/java: This folder has java sources.
--src/main/resources: This folder can hold property files and other resources
--src/main/webapp: This folder holds jsp and other web application content.

4)Code Project
-src/main/java folder > right-click > New > Other > Java > Package
-Name : com.javasoft.bean > Finish
-src/main/java/com/javasoft/bean > right-click > Class > Next
-Name : Greeting
-Greeting.java

```
package com.javasoft.bean;

public class Greeting {

    public String getHello()  {
        return "Hello Gradle Web Application";
    }
}
```

-webapp > right-click > New > Other > Web > JSP File > Next
-File name : hello.jsp > Finish

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hello Gradle Web App</title>
    </head>
    <body>
    <jsp:useBean id="greeting" class="com.javasoft.bean.Greeting"/>
    <h3>${greeting.hello}</h3>
    </body>
</html>
```

5)Gradle Build
-Open "Gradle Task" view.
-Right-click on build folder > "build" and select "Run Gradle Tasks".
-Check Gradle Executions tab, you should see a list of tasks executed.

6)Configure to run application
-In Project Explorer > right-click > Run As > Run Configurations
-Gradle Project > right-click > New
-Name : Run HelloGradleWebApp
-Gradle Tasks : tomcatRun
-Working Directory : ${workspace_loc:/HelloGradleWebApp}
-Click [Apply] button > Run
-In Console

```
Working Directory: C:\JspHome\HelloGradleWebApp
```

```
Gradle User Home: C:\Users\Instructor\.gradle
Gradle Distribution: Gradle wrapper from target build
Gradle Version: 4.3
Java Home: C:\Program Files\Java\jdk1.8.0_162
JVM Arguments: None
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: tomcatRun

:compileJava UP-TO-DATE
:processResources NO-SOURCE
:classes UP-TO-DATE
:tomcatRun
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-jasper/8.5.16/tomcat-embed-jasper-8.5.16.pom
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-core/8.5.16/tomcat-embed-core-8.5.16.pom
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-logging-juli/8.5.2/tomcat-embed-logging-juli-8.5.2.pom
Download https://repo1.maven.org/maven2/org/eclipse/jdt/ecj/3.12.3/ecj-3.12.3.pom
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-el/8.5.16/tomcat-embed-el-8.5.16.pom
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-el/8.5.16/tomcat-embed-el-8.5.16.jar
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-logging-juli/8.5.2/tomcat-embed-logging-juli-8.5.2.jar
Download https://repo1.maven.org/maven2/org/eclipse/jdt/ecj/3.12.3/ecj-3.12.3.jar
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-core/8.5.16/tomcat-embed-core-8.5.16.jar
Download https://repo1.maven.org/maven2/org/apache/tomcat/embed/tomcat-embed-jasper/8.5.16/tomcat-embed-jasper-8.5.16.jar
Started Tomcat Server
The Server is running at http://localhost:8080/HelloGradleWebApp
```

7)In Browser
http://localhost:8080/HelloGradleWebApp/hello.jsp