Chapter 04.스프링 부트 이해하기 - 04.커스텀 스프링 부트 스타터 만들기

커스텀 스프링 부트 스타터 만들기

1. 스프링 부트 스타터란?

- `스프링 부트 스타터(spring-boot-starter)`는 스프링 부트 기반의 애플리케이션에 다른 스프링 프로젝트를 쉽게 추가할 수 있도록 만들어진 라이브러리를 말한다
- 각각의 스타터에는 주축이 되는 스프링 프로젝트와 프로젝트에서 필요로 하는 필수 라이브러리가 같이 포함돼있다
- 이런 이유로 스프링 부트 스타터를 사용하면 스타터 외에 필요한 라이브러리를 따로 추가하거나 관리할 필요가 없다
- 스프링 부트 공식 스타터는 자체 명명 규칙을 따라 각각의 스타터의 이름을 짓고 있습니다. 자체 명명 규칙은 `spring-boot-starter-*` 와 같은 형태
- 주요 스프링 부트 스타터

이름	설명
spring -boot-starter-web	스프링 MVC기반의 웹 애플리케이션 스타터. 임베디드 톰캣을 포함
spring-boot-starter-security	스프링 시큐리티 관련 설정과 라이브러리를 포함
spring-boot-starter-data-jpa	스프링 트랜잭션, 하이버네이트, 히카리 CP 등을 포함
spring-boot-starter-test	테스팅 프레임워크인 JUnit, Mokito, AssertJ 등을 포함
spring-boot-starter-webflux	리액티브 프레임워크인 프로젝트 리액터와 리액터 네티를 포함

---

2. 서드 파티 스프링 부트 스타터

- 공식 스프링 부트 스타터가 아닌 `서드 파티 스프링 부트 스타터`가 존재
- 서드 파티 스프링 부트 스타터란 외부 프로젝트에서 스프링 부트와 연동 하기 위해 직접 만든 커스텀 스타터
- 서드 파티 스프링 부트 스타터의 경우는 spring-boot-starter-*라는 이름으로 지을 수 없다
- 해당 명명 규칙은 `공식 스타터에만 적용하기로 약속`되어 있기 때문에 서드 파티 스타터는 `*-spring-boot-starter`와 같은 방식으로 명명
- 국내에서 유명한 마이바티스도 `mybatis-spring-boot-starter`라는 이름의 스타터를 직접 만들어서 제공하고 있다

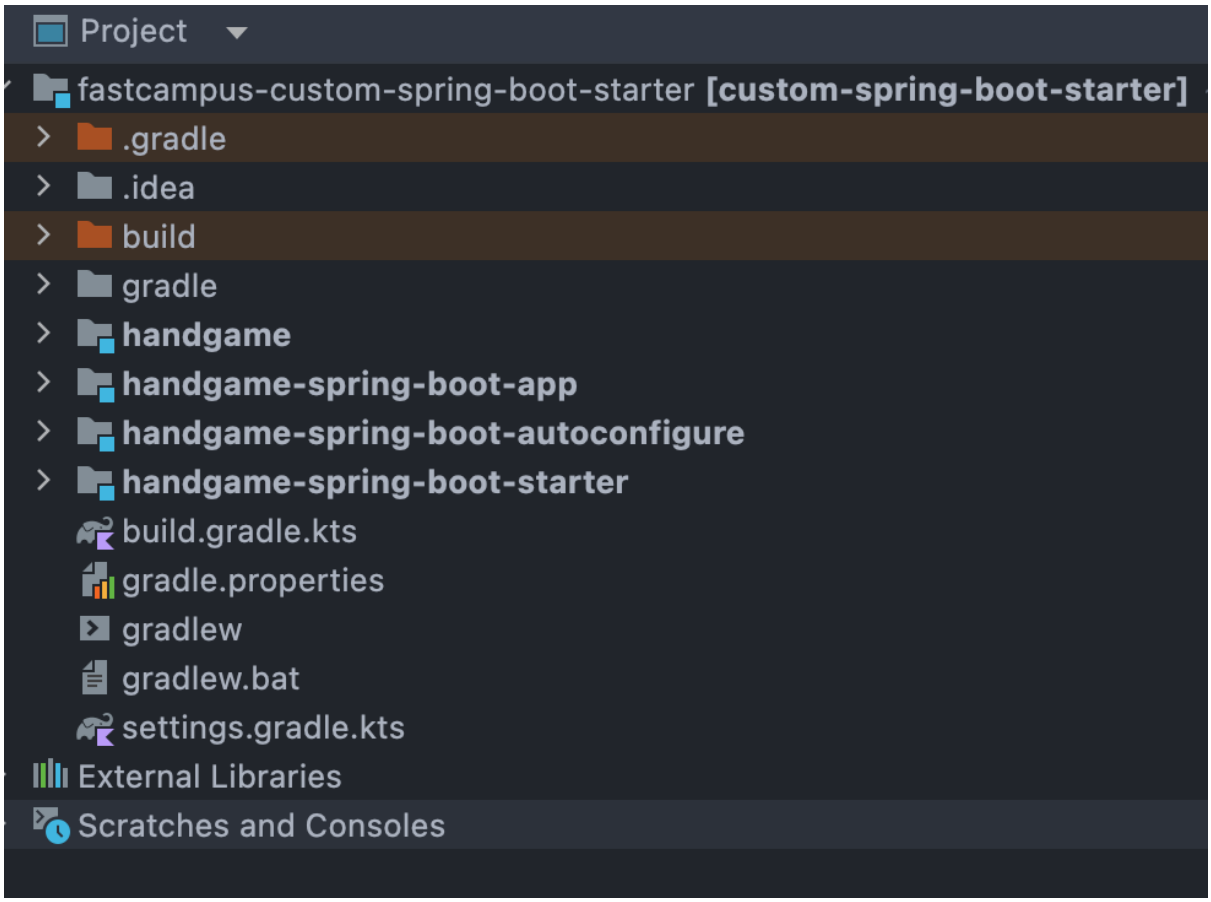
---

3. 커스텀 스프링 부트 스타터 만들기

- 직접 만든 라이브러리와 스프링 부트의 기능을 사용해서 `커스텀 스프링 부트 스타터`를 만들 수 있다
- 이번 예제에선 간단한 가위바위보 게임을 구현한 커스텀 스프링 부트 스타터를 만들어본다

3.1. 프로젝트 구조 설명

- 그레이들의 `멀티 프로젝트`를 사용해서 하나의 프로젝트에 멀티 모듈을 구성함



- 생성된 멀티 모듈의 구조는 다음과 같음
  - handgame
    - 가위바위보 게임의 상세 구현이 포함된 라이브러리
  - handgame-spring-boot-autoconfigure
    - handgame 라이브러리의 자동 설정이 포함
  - handgame-spring-boot-starter
    - handgame, handgame-spring-boot-autoconfigure을 합친 커스텀 spring-boot-starter
  - handgame-spring-boot-app
    - handgame-spring-boot-starter를 사용하는 스프링 부트 애플리케이션

최상위 build.gradle.kts

```
import org.springframework.boot.gradle.plugin.SpringBootPlugin

plugins {
```

```
    id("org.springframework.boot") version "2.7.0" apply false
    id("io.spring.dependency-management") version "1.0.11.RELEASE"
    id("maven")
    id("maven-publish")
    kotlin("jvm") version "1.6.21"
    kotlin("plugin.spring") version "1.6.21"
    kotlin("kapt") version "1.6.21"
}

allprojects {
    group = "com.fastcampus.springboot"
    version = "1.0-SNAPSHOT"

    repositories {
        mavenLocal()
        mavenCentral()
    }
}

subprojects {
    apply(plugin = "kotlin")
    apply(plugin = "kotlin-kapt")
    apply(plugin = "kotlin-spring")
    apply(plugin = "maven")
    apply(plugin = "maven-publish")
    apply(plugin = "io.spring.dependency-management")

    dependencies {
        implementation("org.jetbrains.kotlin:kotlin-reflect")
        implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
    }

    dependencyManagement {
        imports {
            mavenBom(SpringBootPlugin.BOM_COORDINATES)
        }
    }
}
```

1. maven-publish는 메이븐 리파지토리에 배포할 수 있게 하는 기능을 하는 플러그인이다. 이번 예제에선 제작한 스타터 프로젝트를 로컬 메이븐 리파지토리에 배포해서 handgame-spring-boot-app에서 사용할 예정입니다
2. allprojects는 최상위 프로젝트를 포함한 전체 프로젝트에서 사용하는 빌드를 구성한다
3. subprojects는 settings.gradle.kts에서 include에 감싸진 프로젝트의 빌드를 구성한다. 예를 들면 `include(“handgame”,“handgame-spring-boot-starter)` 와 같은 프로젝트가 서브 프로젝트입니다.

handgame/Handgame.kt

```
package com.fastcampus.springboot.handgame

import java.util.*

class Handgame {

    fun play(player: GameCommand): Pair<GameResult, GameCommand> {
        val opponent = GameCommand.values()[Random().nextInt(3)]
        return if (player == opponent) {
            return Pair(GameResult.동점, player)
        } else if (player == GameCommand.바위 && opponent == GameCommand.가위) {
            return Pair(GameResult.승리, GameCommand.가위)
        } else if (player == GameCommand.가위 && opponent == GameCommand.보) {
            return Pair(GameResult.승리, GameCommand.보)
        } else if (player == GameCommand.보 && opponent == GameCommand.바위) {
            return Pair(GameResult.승리, GameCommand.바위)
        } else Pair(GameResult.패배, opponent)
    }
}

enum class GameCommand(num: Int) {
    바위(0), 가위(1), 보(2);
}

enum class GameResult {
    동점, 승리, 패배
}
```

- Handgame 클래스의 play 함수는 사용자의 커맨드를 전달받아 대전자(컴퓨터)와의 가위바위보 게임 결과를 리턴한다

handgame-spring-boot-autoconfigure/build.gradle.kts

```
dependencies {
    kapt("org.springframework.boot:spring-boot-autoconfigure-processor")
    kapt("org.springframework.boot:spring-boot-configuration-processor")

    api(project(":handgame"))

    implementation("org.springframework.boot:spring-boot")
    implementation("org.springframework.boot:spring-boot-autoconfigure")
}
```

1. kapt는 코틀린의 애노테이션 프로세서이다. `애노테이션 프로세서(Annotation Processor)`는 컴파일 타임에 애노테이션을 읽어서 동적으로 코드를 생성하거나 변경하는 등의 기능을 말하는데 kapt는 코틀린 언어에서 이러한 애노테이션 프로세서가 동작하도록 하는 플러그인이다
2. 자동 설정 클래스에서 Handgame.kt 클래스를 불러올 수 있어야 하기 때문에 handgame 프로젝트에 대한 의존성을 추가
3. spring-boot, spring-boot-autoconfigure는 우리가 만들 자동 설정이 스프링 부트 자동 설정 기능을 사용해야 하므로 필수적인 의존성임

handgame-spring-boot-autoconfigure/HandgameAutoconfigure.kt

```
package com.fastcampus.springboot.autoconfigure.handgame

import com.fastcampus.springboot.handgame.Handgame
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration

@Configuration
@ConditionalOnClass(Handgame::class)
@ConditionalOnProperty(prefix = "my.handgame", name = ["enabled"], havingValue = "true")
class HandgameAutoconfiguration {

    @Bean
    @ConditionalOnMissingBean
    fun handgame() = Handgame()
}
```

1. HandgameAutoconfiguration은 Handgame 클래스의 인스턴스를 스프링 빈에 등록하는 자동 설정 클래스이다. 자동 설정 클래스는 설정 클래스임을 나타내기 위해 `@Configuration` 애노테이션을 선언함
2. `@ConditionalOnProperty` 를 사용해서 my.handgame 으로 시작하고 enabled 라는 프로퍼티가 true인 경우에만 해당 설정 클래스가 동작하도록 로드 시점을 조정한다.
3. 만약 사용자가 handgame 빈을 재정의한 경우에는 충돌이 발생할 수 있으므로 `@ConditionalOnMissingBean` 을 사용해서 handgame 빈이 존재하지 않는 경우에만 빈을 로드한다

handgame-spring-boot-autoconfigure/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
com.fastcampus.springboot.autoconfigure.handgame.HandgameAutoconfiguration
```

1. 우리가 작성한 자동 설정 클래스 `HandgameAutoconfiguration` 를 스프링에 알려주기 위해 해당 파일에 명시한다

2. Spring Boot 2.7 이하 버전에서는 META-INF/spring.factories 파일에 자동 설정 클래스를 명시한다

handgame-spring-boot-starter/build.gradle.kts

스타터 모듈에는 코드 구현은 들어가지 않고 빌드 스크립트만 추가한다

```
dependencies {
    api(project(":handgame"))
    api(project(":handgame-spring-boot-autoconfigure"))
}
```

1. 앞서 구현한 handgame 라이브러리와 자동 설정 구현체인 handgame-spring-boot-autoconfigure에 대한 의존성을 추가

3.2. 스프링 부트 스타터 앱 실행하기

- 이렇게 만든 스타터 프로젝트를 maven local publish 기능을 사용해 순서대로 배포한다
- handgame > handgame-spring-boot-autoconfigure > handgame-spring-boot-starter 순서
- 순서대로 배포되면 아래 경로에 jar를 확인할 수 있다

```
# 사용자 홈디렉토리 하위 .m2
/Users/sanghoon/.m2/repository/com/fastcampus/springboot


# 로컬 배포된 모듈들
(base) ➜  springboot ll
drwxr-xr-x  4 sanghoon  staff   128B Jun  1 00:46 handgame
drwxr-xr-x  4 sanghoon  staff   128B Jun  1 00:46 handgame-spring-boot-autoconfigure
drwxr-xr-x  4 sanghoon  staff   128B Jun  1 00:46 handgame-spring-boot-starter
```

handgame-spring-boot-app/build.gradle.kts

```
dependencies {
    implementation("org.springframework.boot:spring-boot-starter")

    // 우리가 만든 커스텀 스프링 부트 스타터
    implementation("com.fastcampus.springboot:handgame-spring-boot-starter:1.0-SNAPSHOT")
}
```

- 배포가 완료됐다면 스타터를 사용할 프로젝트에서 의존성을 추가한다

- 이 상태에서 실행하면 오류가 발생한다
- IntelliJ 실행 옵션의 `program arguments`로 `--debug` 옵션을 추가해서 실행하면 현재 설정된 자동 설정(Positive matches)과 적용되지 않은 자동 설정(Negative matches)을 확인할 수 있다

```
HandgameAutoconfiguration:
      Did not match:
         - @ConditionalOnProperty (my.handgame.enabled=true) did not find property 'enabled' (OnPropertyCondition)
      Matched:
         - @ConditionalOnClass found required class 'org.example.springboot.handgame.Handgame' (OnClassCondition)
```

- 자동 설정 클래스를 제작할때 `my.handgame.enabled` 프로퍼티가 `true` 인 경우에 동작하도록 했기 때문
- application.properties에 아래 설정을 추가한다

```
my.handgame.enabled = true
```

- 애플리케이션이 정상 동작하며 Positive matches에 아래 설정이 동작한 것을 확인할 수 있다

```
HandgameAutoconfiguration matched:
      - @ConditionalOnClass found required class 'org.example.springboot.handgame.Handgame' (OnClassCondition)
      - @ConditionalOnProperty (my.handgame.enabled=true) matched (OnPropertyCondition)

   HandgameAutoconfiguration#handgame matched:
      - @ConditionalOnMissingBean (types: org.example.springboot.handgame.Handgame; SearchStrategy: all) did not find any beans (OnBea
```