تمرین دوم: پیاده‌سازی کلاس IPv4 (زمان تحویل: ۱۵ اسفند)

در این تمرین قصد داریم که کلاس IPv4 زیر را تکمیل کنیم.

```
class IPv4 {
    var parts = ShortArray(4)

    constructor(address: String) {

    }

    constructor(vararg va: Short) {

    }

    override fun toString(): String {

    }

    override fun equals(other: Any?): Boolean {

    }
}
```

برای انجام این تمرین، ابتدا یک پروژه جدید در IntelliJ ایجاد کنید و از بیلد سیستم maven استفاده کنید. بعد از ایجاد پروژه، وارد فایل pom.xml بشید و وابستگی‌های زیر را در محل مناسب اضافه کنید. دقت کنید که junit-jupiter موجود در فایل را پاک کنید!

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.9.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.9.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.9.1</version>
    <scope>test</scope>
</dependency>
```

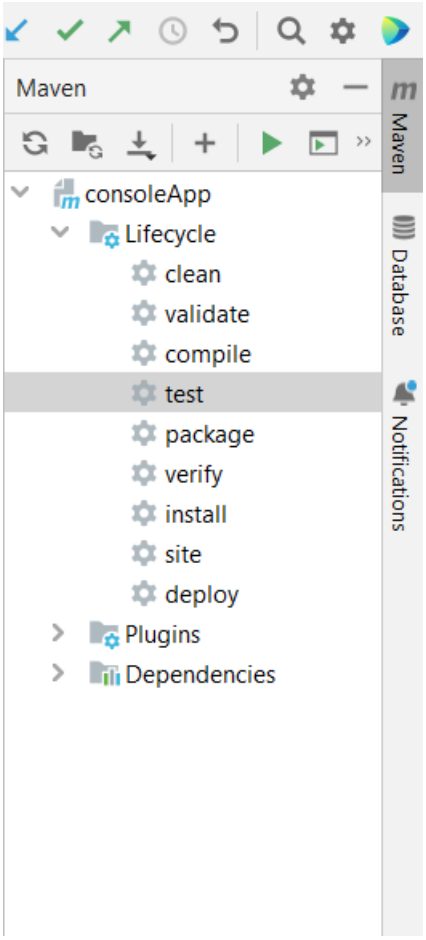
بعد از اضافه کردن وابستگی‌ها از نوار سمت راست Maven را انتخاب کنید و دکمه Reload All Maven Projects را بزنید تا وابستگی‌ها دانلود بشن.

سپس فایل src/main/kotlin/IPv4.kt را ایجاد کنید و کد بالای صفحه را داخلش چاپ کنید. سپس فایل src/test/kotlin/IPv4Test.kt را با محتوای زیر ایجاد کنید.

```
import org.junit.jupiter.api.Assertions.*
import org.junit.jupiter.api.Test

internal class IPv4Test {
    @Test
    fun `constructor throws exception for invalid string address - out of
range`() {
        val invalidAddress = "256.255.255.255"
        assertThrows(Exception::class.java) {
            IPv4(invalidAddress)
        }.let {
            assertEquals("Invalid address: $invalidAddress", it.message)
        }
    }

    @Test
    fun `constructor throws exception for invalid string address - non-
numeric`() {
        val invalidAddress = "a.255.255.255"
        assertThrows(Exception::class.java) {
            IPv4(invalidAddress)
        }
    }

    @Test
    fun `constructor throws exception for invalid string address - missing
segments`() {
        val invalidAddress = "255.255.255"
        assertThrows(Exception::class.java) {
            IPv4(invalidAddress)
        }.let {
            assertTrue(it.message!!.contains("Invalid array size: expected 4
but received 3"))
        }
    }

    @Test
    fun `constructor throws exception for invalid string address - extra
segments`() {
        val invalidAddress = "255.255.255.255.255"
        assertThrows(Exception::class.java) {
            IPv4(invalidAddress)
        }.let {
            assertTrue(it.message!!.contains("Invalid array size: expected 4
but received 5"))
        }
    }

    @Test
    fun `constructor throws exception for invalid string address - negative
value`() {
        val invalidAddress = "10.-20.255.255"
        assertThrows(Exception::class.java) {
            IPv4(invalidAddress)
        }.let {
            assertTrue(it.message!!.contains("Invalid address:
$invalidAddress"))
```

```
        }
    }

    @Test
    fun `constructor throws exception for invalid array size`() {
        val invalidArray = shortArrayOf(1, 2, 3)
        assertThrows(Exception::class.java) {
            IPv4(*invalidArray)
        }.let {
            assertEquals("Invalid array size: expected 4 but received
${invalidArray.size}", it.message)
        }
    }

    @Test
    fun `constructor throws exception for invalid array element - out of
range`() {
        val invalidArray = shortArrayOf(256, 0, 0, 0)
        assertThrows(Exception::class.java) {
            IPv4(*invalidArray)
        }.let {
            assertTrue(it.message!!.contains("Invalid value:
${invalidArray[0]}"))
        }
    }

    @Test
    fun `valid string address constructor creates correct IPv4 object`() {
        val address = "192.168.1.1"
        val ipv4 = IPv4(address)
        assertArrayEquals(shortArrayOf(192, 168, 1, 1), ipv4.parts)
    }

    @Test
    fun `valid string address constructor string method`() {
        val address = "192.168.1.1"
        val ipv4 = IPv4(address)
        val s = "$ipv4"
        assertEquals(s, address)
    }

    @Test
    fun `valid array constructor creates correct IPv4 object`() {
        val parts = shortArrayOf(10, 20, 30, 40)
        val ipv4 = IPv4(*parts)
        assertArrayEquals(parts, ipv4.parts)
    }

    @Test
    fun `valid equal method`() {
        val address = "192.168.1.1"
        val ip1 = IPv4(address)
        val ip2 = IPv4(address)
        assertTrue(ip1 == ip2)
    }
}
```

در کد بالا برای کلاس IPv4 تعدادی تست نوشته شده که میتونید با انتخاب Maven از نوار سمت چپ و انتخاب consoleApp>Lifecycle>test اجراشون کنید. از این تست‌ها استفاده کنید و کلاس IPv4 را بنویسید 😊



شکل ۱: منوی Lifecycleهای Maven

نکته: Test Driven Development

در این تمرین قراره که یک روش توسعه نرم‌افزار به نام Test Driven Development آشنا بشید. در این روش توسعه نرم‌افزار، بعد از توافق روی امضای توابع، یک نفر به پیاده سازی تست میپردازه که ورودی و خروجی مطلوب توابع را بررسی میکنه و یک نفر خود توابع را پیاده‌سازی می‌کنه، به نحوی که تست‌ها پاس بشن.

مزیت مهم این شیوه توسعه اینه که طی فرآیند رسیدن به امضای توابع هم از توسعه کد بی‌کیفیت جلوگیری میشه و هم افراد به صورت همزمان می‌تونند بخش‌های مختلف کد را پیاده‌سازی و بلافاصله تست کنند.