دانشکده ریاضی و علوم کامپیوتر

برنامه‌سازی پیشرفته و کارگاه

انوتیشن‌ها

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

امین رضائی مهر و آرمان حسینی

بهار ۱۴۰۳

فهرست

مقدمه

توی داک «ادامهٔ شی‌گرایی»، شما مقداری با Annotationها توی جاوا آشنا شدین.[1] Annotationها، به ما امکان این رو می‌دن که به متدهامون، کلاس‌هامون، یا هر بخش دیگهٔ برنامه‌های جاوا metadata اضافه کنیم. این metadataها، توسط بخش‌های مختلف قابل خوندن هستن.

مثلا، Override@ به کامپایلر می‌گفت که «این متد، داره یکی از متدهای کلاس پدر رو override می‌کنه.»، یا Deprecated@ به برنامه‌نویس‌های دیگه (و IDEهاشون) می‌گفت که «این بخش کد، قدیمیه و دیگه نباید استفاده بشه».

توی این داک، ما می‌خوایم ببینیم این Annotationها چطور تعریف شدن و چطور می‌تونیم با استفاده از Refelection، اطلاعات این Annotationها رو توی کلاس‌مون بخونیم. احتمالا شما حالا حالاها Annotationهای جدید زیادی تعریف نمی‌کنید، ولی فریم‌ورک‌هایی که از اون‌ها استفاده می‌کنید، مثل Hibernate و Spring Boot کلی Annotation جدید استفاده می‌کنن و خوبه که بدونید چطور اون‌ها رو تعریف کردن.

[1] اگر یادتون رفته برین یه نگاه سریع بندازین و برگردین.

تعریف Annotationها

IntelliJ رو باز کنید و هر جایی از صفحه، یه @Override بنویسید. بعدش با کلیک روی Go To و Declaration and Usages به سورس کد انوتیشن Override برین. همچین کدی می‌بینید:

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.SOURCE)
public @interface Override {
}
```

توی جاوا، انوتیشن‌ها با عبارت @interface تعریف می‌شن. بالای Override دو annotation دیگه هم می‌بینید:

- **Target:** این annotation، مشخص می‌کنه که Override به چه عناصری از کد اعمال‌شدنیه. @Override فقط بالای سر متدهامون میاد، پس این‌جا هم فقط ElementType.METHOD توی Target مشخص شده.
- **Retention:** بعضی Annotationها، فقط در زمان کامپایل به درد ما می‌خورن ولی بعضی از اون‌ها، توی زمان اجرای برنامه هم استفاده می‌شن. مثلا، @Override فقط توسط Compiler استفاده می‌شه تا در صورتی که متد پدر رو به درستی override نکردیم بهمون خطا بده. بعد از این که کار Compiler با کد تموم شد، این انوتیشن از خروجی compiler حذف می‌شه.

  این که طول عمر هر annotation چقدره توسط Retention مشخص می‌شه؛ اگر RetentionPolicy.SOURCE انتخاب بشه annotationمون فقط توسط Compiler خونده می‌شه و اگر RetentionPolicy.RUNTIME انتخاب بشه، حین اجرای برنامه هم می‌تونیم از این Annotation استفاده کنیم.[2] در ادامه می‌بینید که چطور می‌تونیم Annotationهای بالای عناصر برنامه‌مون رو بخونیم.

حالا، بیاین نگاهی به @Deprecated هم بندازیم. سورس اون هم توی IntelliJ بالا بیارین:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(value={
        CONSTRUCTOR,
        FIELD,
        LOCAL_VARIABLE,
```

[2] علاوه بر این، گزینهٔ سومی به اسم RetentionPolicy.CLASS هم وجود داره که این‌جا بهش نمی‌پردازیم. اگر دوست داشتین، از این‌جا می‌تونید بیشتر در این مورد بخونید.

```
        METHOD,
        PACKAGE,
        MODULE,
        PARAMETER,
        TYPE})
public @interface Deprecated {
    String since() default "";
    boolean forRemoval() default false;
}
```

همون‌طور که می‌بینید، این انوتیشن هم Target و Retention داره. با توجه به این که هر چیزی می‌تونه قدیمی و بلاکاربرد بشه، Deprecated@ به کانستراکتورها، فیلدها، متدها و خیلی چیزهای دیگه اعمال‌شدنیه.

بر خلاف Override@، Retention این انوتیشن، RetentionPolicy.RUNTIMEئه که یعنی در حین اجرای برنامه هم قابل استفاده‌ست.

علاوه بر این، Deprecated دو پارامتر هم داره. پارامترهای since و forRemoval که به ترتیب، String و boolean هستن. شما می‌تونید موقع استفاده از Deprecated این دو پارامتر رو مقداردهی کنید:

```
@Deprecated(since = "2.2", forRemoval = true)
public static void remove(int id) {

    // CODE CODE CODE

}
```

اگر هر کدوم از پارامترها رو مقداردهی نکنید، مقدار دیفالت اون‌ها (که جلوی عبارت default اومده) بهشون داده می‌شه:

```
// forRemoval is false now
@Deprecated(since = "2.2")
public static void remove(int id) {
    // CODE CODE CODE

}
```

حالا که یه آشنایی حداقلی با شیوهٔ تعریف Annotationها داریم، می‌تونیم ببینیم که چطور می‌شه Annotationهای یک کلاس رو خوند. برای این کار، از قابلیت Reflection استفاده می‌کنیم.

Reflection

جاوا، بهتون این قابلیت رو می‌ده که حین اجرای برنامه، ویژگی‌های کلاس‌هاتون، متدهای اون‌ها، فیلدهاشون و همگی این‌ها رو بررسی کنید. به این قابلیت، Reflection می‌گن و یکی از قدرتمندترین ویژگی‌های زبان جاواست.

کلاس زیر رو توی کدتون تعریف کنید:

```
public class User {
    private String username;

    private String password;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

حالا یه main برین و کد زیر رو بنویسید:

```
public static void main(String[] args) {
    Class<User> userClass = User.class;
}
```

توی این کد، یه آبجکت ساختیم که شامل ویژگی‌های کلاس Userئه. همهٔ عناصر این کلاس، از متدها و فیلدهاش گرفته تا اسم پکیجش و انوتیشن‌هاش، همگی و همگی از طریق userClass قابل دسترسی‌ان. همچنین، اگر توجه کنید خود این آبجکت، از جنس Class<User>ئه.

به عنوان مثال، کد زیر اسم کامل این کلاس رو به همراه اسم پکیجش چاپ می‌کنه:

```
public static void main(String[] args) {
    Class<User> userClass = User.class;
    System.out.println(userClass.getCanonicalName());
}
```

من کلاس User رو توی پکیج aut.ap تعریف کردم، پس خروجی این کد برام این شکلیه:

```
aut.ap.User
```

یا مثلا، می‌تونیم با استفاده از آبجکت userClass، تمام متدهای این کلاس رو ببینیم:

```
public static void main(String[] args) {
    Class<User> userClass = User.class;

    for (Method method : userClass.getMethods()) {
        System.out.println(method.getName());
    }
}
```

خروجی این کد، به شکل زیره:

```
getUsername
setUsername
getPassword
setPassword
equals
toString
hashCode
getClass
notify
notifyAll
wait
wait
wait
```

همون‌طور که می‌بینید، علاوه بر متدهای خود کلاس‌مون، متدهایی که این کلاس از Object به ارث برده هم توی این خروجی دیده می‌شه.

کد زیر هم، تمام فیلدهای این کلاس رو نشونمون می‌ده:

```
public static void main(String[] args) {
    Class<User> userClass = User.class;

    for (Field field : userClass.getDeclaredFields()) {
        System.out.println(field.getName());
    }
}
```

و خروجی‌ش به این شکله:

```
username
password
```

ما با استفاده از reflection، می‌تونیم annotationهای یه کلاس و فیلدهای اون رو هم بخونیم. به کلاس User برگردین و چندتا از متدها و فیلدهای اون رو با Deprecated، annotate کنید:

```
@Deprecated
public class User {

    @Deprecated
    private String username;

    private String password;

    @Deprecated
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    @Deprecated(forRemoval = true, since = "2.2")
    public String getUsername() {
        return username;
    }

    @Deprecated
    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

با استفاده از کد زیر، می‌تونید تمام annotationهای متدهاتون رو ببینید. اگر متدی هیچ annotationای نداشته باشه، توی خروجی این کد چاپ نمی‌شه:

```
Class<User> userClass = User.class;

for (Method method : userClass.getMethods()) {

    if (method.getAnnotations().length == 0) {
        continue;
    }


    System.out.println("Annotations for " + method.getName() + ": ");

    for (Annotation annotation : method.getAnnotations()) {
        System.out.println(annotation);
    }
```

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

خروجی این کد، به شکل زیره:

```
Annotations for getUsername:
@java.lang.Deprecated(forRemoval=true, since="2.2")

Annotations for setUsername:
@java.lang.Deprecated(forRemoval=false, since="")

Annotations for hashCode:
@jdk.internal.vm.annotation.IntrinsicCandidate()

Annotations for getClass:
@jdk.internal.vm.annotation.IntrinsicCandidate()

Annotations for notify:
@jdk.internal.vm.annotation.IntrinsicCandidate()

Annotations for notifyAll:
@jdk.internal.vm.annotation.IntrinsicCandidate()
```

همون‌طور که می‌بینید، علاوه بر Deprecated@هایی که برای getUsername و setUsername و مقادیر اون‌ها مشخص کرده بودیم، annotationهایی که روی متدهای کلاس Object اومده بودن هم این‌جا چاپ شدن.

چیزی که یاد گرفتیم

توی این داک، ما یه نگاه خیلی خیلی مختصر به annotationها و reflection توی جاوا انداختیم. reflection، از قدرتمندترین ابزارهای جاواست که به دولوپرهای جاوا اجازه می‌ده فریم‌ورک‌های خیلی خوبی بنویسن.

ما توی این داک یاد گرفتیم که:

- Annotationها چطور تعریف می‌شن.
- چطور می‌شه با استفاده از Reflection، به annotationهای یک کلاس دسترسی پیدا کرد.

منابع بیشتر

اگر دوست داشتین راجع به reflection بیشتر بخونید، این بلاگ توضیحات خیلی خوبی راجع به اون داده. همچنین اگر دوست دارین راجع به Annotationها بیشتر یاد بگیرین، یه نگاه به داک رسمی Oracle بندازید.