Refactoring

خانه تکانی اساسی

همان طور که پیش از این هم اشاره شد، حیات یک برنامه بزرگ به اشکال زدایی و توسعه آن بستگی دارد. و این کار تنها زمانی ممکن است که سورس کد این برنامه، به ویژه برای کسانی به جز توسعه دهندگان آن، به اندازه کافی قابل درک و فهم باشد.
فرآیند Refactoring، یکی از تکنیک های مهم در دنیای برنامه نویسی که در طی آن، کدی که تا آن لحظه نوشته شده است را بازنگری می کنیم و از ساختارمندی و تمیزی آن اطمینان حاصل می کنیم. در ادامه با برخی مفاهیم اصلی مرتبط با آن آشنا می شویم.

ریفکتورینگ چیست؟

> به طور کلی، refactoring یک تکنیک نظام مند برای پیکربندی مجدد یک قطعه کد است، به طوری که *بدون تغییر در رفتار بیرونی* برنامه، ساختار درونی آن را بهبود ببخشیم.

همان طور که احتمالا حدس زده این، برای ارزیابی صحت عملکرد کدمان پس از فرآیند ریفکتورینگ، لازم است تا پیش از آن کدمان دارای test coverage بالا و مشخصی باشد . بدین ترتیب می توانیم مطمئن باشیم که پس از این فرآیند هم چنان برنامه به درستی کار می کند.

انواع ریفکتورینگ

تغییر نام (Renaming)

هر چقدر هم که به تمیزی کد خود حساس باشید، احتمالا بارها برایتان پیش آمده است که پس از نوشتن یک قطعه کد، احساس کرده باشید که لازم است تا در نام گذاری یک موجودیت ممکن تجدید نظر کنید. این تغییر نام تنها محدود به متغیرها نیست. گاهی لازم است تا نام یک کلاس، متد، فایل یا پکیج را تغییر دهید.
البته این کار آن قدر ها هم که به نظر می رسد آسان نیست، به طوری که اگر با دقت کافی انجام نشود،می تواند بسیار دردسر ساز شود و شما را از تصمیمتان منصرف کند! علت آن این است که اغلب به ویژه در پروژه های بزرگ، تعداد دفعات استفاده از یک موجودیت خاص بسیار قابل توجه است لذا عدم به روز رسانی تمام این قسمت های مرتبط می تواند ما را دچار مشکل کند. در ادامه خواهیم گفت چگونه می توانیم این فرآیند پیچیده را با خیال راحت انجام دهیم.

استخراج (Extracting)

در طی این فرآیند سعی می کنیم تا با تعریف یک متغیر، متد، کلاس یا پارامتر با نام مناسب، خوانایی کدمان را بالا ببریم، یا اینکه از طولانی شدن بیش از حد یک کلاس/متد/خط در برنامه جلوگیری کنیم. در اینجا به ذکر دو مورد کاربردی از انواع استخراج می پردازیم.

۱. استخراج متغیر(Extract Variable)

این تکنیک بیشتر برای کاهش کد تکراری مورد استفاده قرار می گیرد. برای درک بهتر آن به مثال زیر توجه کنید:

```
static String f(String message) {
    return message.substring(message.indexOf("\"screen_name\":\"") + 15
    message.indexOf("\"", message.indexOf("\"screen_name\":\"") + 15));
}
```

آیا متوجه شدید که این قطعه کد چه کاری انجام می دهد؟

حال با تعریف دو متغیر جدید همان کد را به شکل زیر اصلاح می کنیم:

```
static String f(String message) {
    String fieldName = "\"user_name\":\"";
    int indexOfFieldValue = message.indexOf(fieldName) + fieldName.leng
    int indexOfEndOfFieldValue = message.indexOf("\"", indexOfFieldValu
    return message.substring(indexOfFieldValue, indexOfEndOfFieldValue)
}
```

حال با کمی بررسی، متوجه می شویم که قطعه کد بالا نام کاربری را از یک رشته با فرمتی خاص پیدا می کند و آن را بازمی گرداند.

۱. استخراج متد(Extract Method)

برای افزایش خوانایی لازم است تا یک متد طولانی را به قسمت های کوچکتر تقسیم کنیم.هر کدام از این قطعات کوچکتر یک متد را تشکیل می دهند که می توان آن ها را در متد اولیه صدا زد.

هم چنین این روش در بسیاری از مواقع می تواند با تعریف یک متد جدید از تکرار یک قطعه کد خاص جلوگیری کند.

به مثال زیر توجه کنید:

```
static String getTextFromMessage(String message) {
    String fieldName = "\"text\":\"";
    int indexOfFieldValue = message.indexOf(fieldName) + fieldName.leng
    int indexOfEndOfFieldValue = message.indexOf("\"", indexOfFieldValu
    return message.substring(indexOfFieldValue, indexOfEndOfFieldValue)
}

static String getUsernameFromMessage(String message) {
```

```
    String fieldName = "\"screen_name\":\"";
    int indexOfFieldValue = message.indexOf(fieldName) + fieldName.leng
    int indexOfEndOfFieldValue = message.indexOf("\"", indexOfFieldValu
    return message.substring(indexOfFieldValue, indexOfEndOfFieldValue)
}
```

هر کدام از دو تابع بالا کار فرآیند مشابهی را برای یافتن یک زیررشته خاص از یک رشته طی می کنند. برای کاهش تکرار کد، بهترین کار تعریف و پیاده سازی یک متد جدید است:

```
static String getValueForField(String fieldName, String message) {
    int indexOfFieldValue = message.indexOf(fieldName) + fieldName.leng
    int indexOfEndOfFieldValue = message.indexOf("\"", indexOfFieldValu
    return message.substring(indexOfFieldValue, indexOfEndOfFieldValue)
}
```

در ادامه کافیست این متد را در دو متد قبلی صدا بزنیم:

```
static String getTextFromMessage(String message) {
    return getValueForField("\"text\":\"", message);
}

static String getUsernameFromMessage(String message) {
    return getValueForField("\"screen_name\":\"", message);
}
```

حذف (Deleting)

بسیاری از اوقات پس از تکمیل برنامه مان متوجه می شویم که تعداد زیادی موجودیت (متغیر، پارامتر، متد، کلاس و ...) وجود دارند که در هیچ کجا استفاده نشده اند. این شرایط مخصوصا بعد از انجام پروسه refactoring معمولا برای ما ایجاد می شود.
برای ساده سازی سورس کد لازم است تا قسمت های استفاده نشده از کدمان را پیدا کنیم و بعد از اینکه مطمئن شدیم که این قطعه کد استفاده نشده است و *بعدا نیز استفاده نخواهد شد*، آن را حذف کنیم.

Pull Up

گاهی متوجه می شویم که کلاس های فرزند دارای متدها یا رفتارهای مشترک یا مشابهی هستند. در این مواقع بهتر است این رفتار مشترک را به کلاس پدر منتقل کنیم. تکنیک pull up برای جلوگیری از بروز اشکال (در اثر به روز رسانی یا تغییر یک متد مشترک در

یکی از فرزندان و عدم به روزرسانی مابقی)، و هم چنین جلوگیری از کدتکراری بسیار ضروری است.

Push down

در مقابل گاهی متدها یا فیلدهایی را در کلاس پدر تعریف کرده ایم که تنها یک یا تعداد نسبتا محدودی از کلاس های فرزند از آن ها استفاده کرده اند. در این روش سعی می کنیم با انتقال این متد به کلاس(های) فرزند مربوطه، سیر منطقی کد را بهبود ببخشیم.

ابزارهای ریفکتورینگ

همان طور که متوجه شدید، با اینکه refactoring فرآیند بسیار مهمی به حساب می آید، اغلب مواقع انجام صحیح آن به صورت دستی کاری طاقت فرسا و بعضا نشدنی است و می تواند بسیار خطرناک باشد.
به همین خاطر است که اکثر برنامه های IDE مطرح دارای ابزاری برای کمک به انجام این فرآیند هستند.
در اینجا می توانید با ابزار قدرتمند ریفکتورینگ در IntelliJ آشنا شوید و تمام موارد گفته شده و هم چنین بسیاری از موارد کاربردی دیگر را تمرین کنید.