Introduction to Programming

Rania Baghernejad

Java – part 9

Scope & Shadowing & Final

مباحث

Scope یا دامنه متغیر

وقتی شما یک متغیر تعریف می‌کنید، آن متغیر فقط در یک بخش مشخص از برنامه قابل استفاده است. این بخش به آن می‌گوییم Scope(دامنه). یعنی هر متغیر در محدوده‌ای خاص از برنامه دیده می‌شود و معتبر است.

در جاوا، بسته به اینکه متغیر کجا تعریف شده، دامنه آن هم فرق می‌کند.

متغیرهای محلی(Local Variables)

متغیرهایی که داخل یک تابع، شرط یا حلقه تعریف می‌شوند. فقط در همان بخش قابل استفاده هستند.

```
public class Example {
    public static void main(String[] args) {
        int x = 10; // این متغیر فقط در این تابع معتبر است

        if (x > 5) {
            int y = 20; // این متغیر فقط درون اینجا دیده می‌شود
            System.out.println(y);
        }

        // System.out.println(y); // خطا: این متغیر دیگر وجود ندارد
    }
}
```

متغیرهای سراسری درون کلاس (Class-Level Variables)

اگر متغیری خارج از همه توابع ولی داخل کلاس تعریف شود، در تمام بخش‌های آن کلاس قابل استفاده است.

```
public class Example {
    static int count = 0; // متغیر سراسری

    public static void main(String[] args) {
        System.out.println(count); // قابل استفاده
    }

    public static void anotherMethod() {
        count++;
    }
}
```

Block Scope یا دامنه بلاک

هر جفت { } در جاوا یک بلاک است. متغیرهایی که داخل این بلاک‌ها تعریف می‌شوند، فقط داخل همان بلاک معتبر هستند.

```
for (int i = 0; i < 3; i++) {
    System.out.println(i); // i داخل حلقه معتبر است
}

// System.out.println(i); // خطا: i دیگر وجود ندارد
```

Shadowing یا سایه انداختن

Shadowing زمانی اتفاق می‌افتد که شما دوباره یک متغیر را با همان نام قبلی درون یک محدوده داخلی‌تر تعریف کنید. در این حالت، متغیر جدید، موقتاً جای متغیر قبلی را می‌گیرد و باعث می‌شود دیگر به متغیر قبلی دسترسی نداشته باشید (مگر با راه‌های خاص در آینده).

مثال ساده:

```
public class ShadowExample {
    public static void main(String[] args) {
        int number = 10;

        {
            int number = 5; // این متغیر، متغیر بیرونی را پوشش داده
            System.out.println(number); // چاپ: ۵
        }

        System.out.println(number); // چاپ: ۱۰
    }
}
```

توضیح:

در بلاک داخلی (بین { })، دوباره یک متغیر به نام number تعریف شده. این باعث شده که number جدید، بر روی متغیر بیرونی "سایه" بیندازد و جلوی دیده شدن آن را بگیرد.

نکات مهم:

۱. نباید در یک بلاک، دو بار متغیری با نام یکسان تعریف کرد. این باعث خطا می‌شود.

۲. Shadowing ممکن است باعث اشتباه در برنامه‌نویسی شود، به همین دلیل بهتر است نام متغیرها طوری انتخاب شود که با هم اشتباه گرفته نشوند.

۳. این مبحث برای درک دقیق‌تر رفتار متغیرها و جلوگیری از باگ‌های مخفی در برنامه ضروری است.

---

final (سطح پایه)

در جاوا، زمانی که یک متغیر را با کلیدواژه‌ی final تعریف می‌کنیم، به این معنی است که **بعد از مقداردهی اولیه، دیگر نمی‌توان مقدار آن را تغییر داد** .چنین متغیرهایی **فقط یک‌بار قابل مقداردهی** هستند.

---

final برای متغیرهای عددی (یا رشته‌ای و منطقی)

```
public class Main {
    public static void main(String[] args) {
        final int x = 10;
        // x = 20;  // Error: cannot assign a value to final variable 'x'

        final String name = "Ali";
        // name = "Reza";  // Error: cannot assign a value to final variable
'name'

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

توضیح: متغیرهای final فقط یک‌بار می‌توانند مقدار بگیرند. اگر بخواهیم بعداً مقدار جدیدی به آن‌ها اختصاص دهیم، کامپایلر خطا می‌دهد. در اینجا x پس از برابر شدن با ۱۰ دیگر نمی‌تواند مقدار ۲۰ را بپذیرد. به طور مشابه name هم بعد از یک بار مقدار گرفتن "Ali" برای بار بعدی نمی‌تواند مقدار "Reza" را بپذیرد.

استفاده از final برای آرایه‌ها

```
public class Main {
    public static void main(String[] args) {
        final int[] numbers = {1, 2, 3};
```

```
        numbers[0] = 10;  // This is allowed
        // numbers = new int[]{4, 5, 6};  // Error: cannot assign a new array

        System.out.println(numbers[0]);
    }
}
```

توضیح مهم: وقتی یک آرایه را final می‌کنیم:

- نمی‌توانیم **خود آرایه** را به یک آرایه‌ی جدید تغییر دهیم.
- اما می‌توانیم **درایه‌های درون آرایه** را تغییر دهیم.

مقداردهی هم‌زمان(Initialization)

اگر متغیری final باشد، **باید در زمان تعریف یا خیلی سریع بعد از آن مقداردهی شود**؛ وگرنه کامپایلر خطا می‌دهد.

```
public class Main {
    public static void main(String[] args) {
        final int x;
        x = 5;  // OK
        // x = 10;  // Error: cannot assign again
        System.out.println(x);
    }
}
```

نکات تکمیلی:

- از final معمولاً برای تعریف مقادیر **ثابت** که نباید تغییر کنند، استفاده می‌شود. مثل .PI = 3.14
- اگرچه فعلاً وارد کلاس‌ها نشده‌ایم، اما در آینده یاد خواهیم گرفت که final می‌تواند برای جلوگیری از تغییر در ساختار کلاس‌ها و توابع نیز استفاده شود.