به نام خداوند بخشنده‌ی مهربان

شماره‌ی ۲ | ۱۱ اسفند ۹۹

کد نامه

ویژه‌ی دانش‌جویان برنامه‌سازی پیشرفته نیم‌سال دوم ۱۴۰۰-۱۳۹۹ دانشکده‌ی مهندسی و علم کامپیوتر دانشگاه صنعتی شریف

در این شماره از کدنامه، می‌خوانید:

هنر کد تمیز در جاوا (قسمت دوم)

آیا می‌دانستید؟

یکی از مشکلاتی که بسیاری از مواقع برای برنامه‌نویسان در شرکت‌ها پیش می‌آید، **عدم کدنویسی تمیز** سایر برنامه‌نویسان شرکت است؟

هنر کد تمیز در جاوا (قسمت دوم)

تلاش کنید همواره مرتب، منظم و تمیز کد بنویسید

لازم به یادآوری نیست که هنر نوشتن «کد تمیز» – که با بخش اعظم آن در قسمت نخست کدنامه روبه‌رو شده‌اید – از مهارت‌های بسیار مهمی است که لازم است در درس برنامه‌سازی پیشرفته با آن آشنا شوید و عدم رعایت اصول آن، منجر به کسر نمره خواهد شد. در این شماره از کدنامه، با روش‌های دیگری برای تمیزسازی کدهای «مقدماتی» جاوا، آشنا خواهید شد. با کدنامه همراه باشید.

تقویم برنامه‌سازی پیشرفته

هفته‌ی آینده (در صورت عدم تغییر)

17 اسفند

کلاس شی‌گرایی

آشنایی با مفاهیم مهم «واسط، رده، بسته و سطوح دسترسی»

19 اسفند

کلاس شی‌گرایی

شی‌گرایی در جاوا؛ تعریف کلاس و جزئیات آن

21 اسفند

حل تمرین شی‌گرایی

قسمت اول حل تمرین شی‌گرایی، شروع به انجام یک مینی‌پروژه

هنر کد تمیز در جاوا (قسمت دوم)

متین داغیانی

فاصله‌گذاری اجتماعی!

فاصله‌گذاری صحیح، مفهومی بیش‌تر از رعایت چند قانون سخت‌گیرانه است. فاصله ها به کمک ما می‌آیند تا بتوانیم برنامه‌ای که نوشته‌ایم را بهتر درک کنیم و با کم‌ترین زحمت و در سریع‌ترین زمان، اطلاعات قابل قبولی از چیدمان اجزای مختلف برنامه به دست آوریم. فاصله‌گذاری مناسب و اصولی در یک کتاب، می‌تواند یک متن خسته‌کننده را به یک نوشته‌ی خوانا تبدیل کند، به طوری که بتوان با یک نگاه سریع از کارکردش سر در آورد. به طور مشابه، برنامه‌ها به ما نیاز دارند تا با کمی «تمیزکاری»، آن‌ها را سازمان‌مند کرده و از آشفتگی سابق نجات دهیم. در ادامه خواهیم دید که به چه روش‌هایی، کدمان را مرتب کنیم تا ساختار منظم‌تر و پایاتری پیدا کند.

خطوط نه چندان بلند

طولانی بودن بیش از اندازه‌ی خطوط باعث می‌شود خوانایی کد به شدت کاهش یابد. طول هر خط از کد اجرایی برنامه، لازم است حداکثر به اندازه‌ی **۱۲۰ تا ۱۵۰** کاراکتر در نظر گرفته شود.

کاراکتر Tab

در اکثر برنامه‌های ویرایش متن، امکان تغییر طول کاراکتر Tab بر حسب تعداد فاصله (space) وجود دارد. این مقدار معمولا ۴ یا ۸ فاصله در نظر گرفته می شود. توجه کنید که در تمام کدتان از یک فاصله‌ی مشخص استفاده شده باشد تا تراز آن در تمامی خطوط، یکسان و مشخص باشد.

تورفتگی‌ها (Indentation)

اکثر برنامه‌هایی که با آن ها سر و کار داریم، دارای یک ساختار سلسله‌مراتبی هستند: فایل‌ها، کلاس‌ها، متدهای درون کلاس‌ها، بلاک‌های درون متدها و در حالت کلی، بلاک‌های درون بلاک‌های دیگر. برای اینکه این محدوده‌ها را به بهترین شکل از یک‌دیگر تمیز دهیم، باید متناسب با هر کدام، از فاصله‌گذاری مناسب استفاده کنیم. عبارات موجود در سطح فایل، مانند اکثر تعاریف کلاس، به هیچ وجه فرورفته نیستند. متدهای درون یک کلاس در یک سطح فرورفتگی به سمت راست کلاس قرار دارند. بلاک‌های به کار رفته در پیاده‌سازی متد نیز در یک سطح فرورفتگی به سمت راست متد قرار می‌گیرند؛ و در حالت کلی، هر بلاک داخلی باید نسبت به بلاک بیرونی، یک سطح تورفتگی به سمت راست داشته باشد. به عنوان مثال:

```
public class Test {

    private static int number;

    public static void main() {

        innerBlock() {

            operations;

            innerInnerBlock() {

                operations;

            }

        }

    }

    public static void anotherMethod(){

        operations;

    }

}
```

کاراکترهای متشخص

هر برنامه‌ای، پر از دستورات و عملیات ریاضی و کاراکترهای غیر حرفی است که موارد زیر به افزایش خوانایی آن‌ها کمک می‌کند:

- همواره **قبل از باز کردن** پرانتز یک فاصله قرار دهید (میان پرانتزها و یا عبارت داخلی آن، بهتر است هیچ فاصله‌ای وجود نداشته باشد):

```
if(a==2);      // Improper

if (a == 2);   // Correct
```

- بهتر است همواره قبل و بعد از عملگرهای ریاضی یا دودویی از یک فاصله استفاده کنیم (البته به جز عملگرهای ++ یا ==):

```
a = b + c; // Correct

c = a > b; // Correct

b = a && c; // Correct

a=b+c; // Improper

a ++; // Improper
```

- بعد از کاراکتر نقطه‌ویرگول (سمی‌کالن) در حلقه‌ها، یک فاصله می‌گذاریم، اما قبل از آن معمولا هیچ فاصله‌ای نیست:

```
for(int i = 0 ;i < 10;i++);   // Improper

for (int i = 0; i < 10; i++);  // Correct
```

کامنت‌گذاری

چرا کامنت؟

کامنت‌ها، راهنماهایی هستند که به کمک می‌کنند تا وقتی با یک برنامه‌ی بیگانه روبرو می شویم، بدون درگیر شدن بیش از اندازه با جزئیات و نحوه پیاده سازی، با کلیتی از عملکرد برنامه، نحوه استفاده از توابع، چیدمان اصلی فایل‌ها و مواردی از این دست، آشنا شویم. همچنین کامنت‌ها نقشی حیاتی در کتاب‌خانه‌های مختلف بازی می کنند، چرا که غالبا می‌توانند توضیحات جامعی از نحوه‌ی استفاده از امکانات و توابع آن کتاب‌خانه را در اختیار کاربر قرار دهند.

یک کامنت خوب

کامنت‌ها، باید به ما بگویند که «چه اتفاقی دارد می افتد»، «چگونه انجام می‌شود»، «هر کدام از پارامترها یا آرگومان‌های مورد استفاده، چه معنایی دارند» و احیانا «چه محدودیت‌ها یا ایراداتی ممکن است وجود داشته باشند». این موارد، به ویژه در خصوص متدها، بسیار مهم و کاربردی هستند. توجه داشته باشید که کامنت‌های طولانی (بیش از ۸۰ الی ۱۲۰ کاراکتر) را در چند خط بنویسید و از نوشتن همه‌ی آن در یک خط، خودداری کنید.

زیاده‌روی نکنیم

کامنت‌ها می‌توانند در بسیاری از اوقات به ما کمک کنند، اما استفاده بیش از اندازه از آن‌ها، نتیجه‌ی عکس دارد. نباید این نکته را فراموش کنیم که این کامنت‌ها هستند که در کنار کد ما قرار می‌گیرند و نه برعکس! هم‌چنین، در نظر داشته باشید که استفاده از کامنت برای شفاف‌سازی و ارائه‌ی توضیحات بیشتر، همواره آخرین راه حل می‌باشد؛ به عبارت دیگر، **نباید برای یک کد کثیف کامنت گذاشت. بهتر است فکر دیگری به حال آن بکنید!** در بسیاری از موقعیت‌ها، توجه به نام‌گذاری ها، فاصله‌ها، طراحی و پیاده‌سازی اجزا و مواردی از این دست، می‌تواند بسیار بیش‌تر از یک کامنت طولانی و مبهم، به تمیزی کد ما کمک کند.

کامنت‌های TODO

گاهی اوقات، منطقی است که یادداشت های "TODO" را در قالب کامنت‌های TODO // انجام دهید. اصطلاح TODO به کارهایی گفته می‌شود که به نظر برنامه‌نویس، باید انجام شود، اما به دلایلی فعلاً نمی تواند انجام دهد. این TODO، ممکن است یک یادآوری برای حذف یک ویژگی منسوخ شده یا یک درخواست برای شخص دیگری جهت بررسی یک مشکل باشد، یا حتی ممکن است درخواستی از نویسندگان دیگر کد باشد که به نام بهتری فکر کنند یا یک یادآوری برای ایجاد تغییری وابسته به قسمتی دیگر از برنامه باشد که هنوز پیاده‌سازی نشده است. TODO هرچه باشد، بهانه‌ای برای قراردادن کد بد در برنامه نیست. امروزه، اکثر IDE های خوب (از جمله IntelliJ)، ویژگی‌های خاصی را برای یافتن همه‌ی کامنت‌های TODO ارائه می‌دهند، بنابراین احتمالاً این کامنت‌ها به آسانی گم نمی شوند. اگر هم نمی‌خواهید کد شما با TODO پر شود، مرتباً از طریق آنها اسکن کرده و مواردی را که می‌توانید، حذف کنید. توجه شود که در کد نهایی تحویل داده شده، TODO و یا تکه‌کد کامنت شده، موجود نباشد.

```
public void verifyUser (User user) {

    // TODO: Add code to verify the user

}
```

ده محک کدنویسی تمیز!

هرگاه خواستید کد خود را از منظر «تمیزی»، ارزیابی کنید، می‌توانید فاکتورهای اصلی ذکر شده در زیر را در کد خود بررسی کنید:

- از نام‌های تک‌حرفی، مخفف یا نامفهوم و بی‌معنی برای متغیرها و یا متدها استفاده نشده باشد
- اسامی متغیرها و توابع، camelCase باشند
- طول هر تابع، حداکثر ۲۰ تا ۳۰ خط باشد
- هر خط برنامه، حداکثر به طول ۱۲۰ تا ۱۵۰ کاراکتر باشد؛ شرطهای طولانی به چند خط شکسته شده باشند (سرآیند متد، مشکلی ندارد)
- برای نام متدها (به جز main) از افعال امری و یا پرسشی استفاده شده باشد
- از Ctrl+Alt+L در IntelliJ استفاده شده باشد، به گونه‌ای که کد پس از اعمال این میان‌بر، دیگر تغییری نکند
- فاصله‌گذاری طرفین اپراتورها رعایت شده باشد
- تورفتگی‌های کد، مناسب و درست باشند
- هیچ کامنت TODO ای در کد نباشد و همگی قبل از ارسال پاک شده باشند
- هیچ تکه‌کدی در برنامه، کامنت نشده باشد