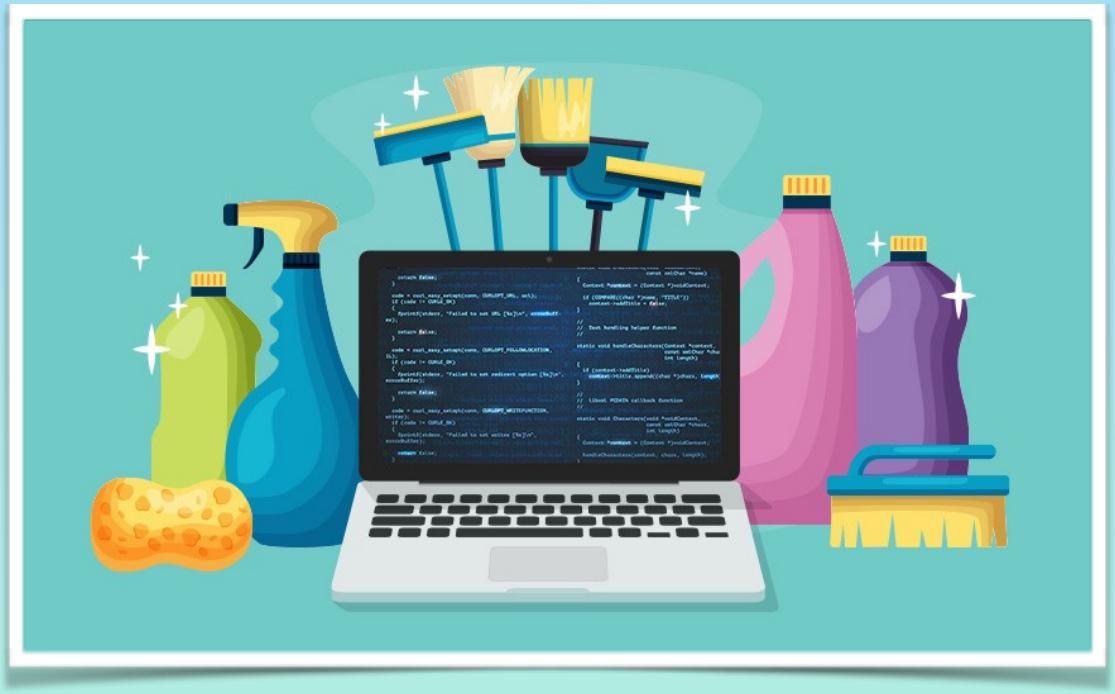
به نام خداوند بخشنده‌ی مهربان

شماره‌ی ۱ | ۷ 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 اول)

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 اول)

تلاش کنید همواره مرتب، منظم و تمیز کد بنویسید

به عنوان دانش‌جویی که درس مبانی برنامه‌سازی را گذرانده‌اید، به خوبی با نحوه حل مسائل گوناگون به کمک برنامه‌نویسی آشنایید، و در درس برنامه‌سازی پیشرفته نیز دانش خود را تقویت خواهید کرد؛ اما یکی از موارد بسیار با اهمیت که گاها به آن بی‌توجهی می‌شود، هنر نوشتن «کد تمیز» است که در درس برنامه‌سازی پیشرفته، عدم رعایت اصول آن، منجر به کسر نمره خواهد شد؛ در نتیجه لازم است به خوبی با اصول آن آشنا شوید. با کدنامه همراه باشید.

تقویم برنامه‌سازی پیشرفته

هفته‌ی آینده

9 اسفند

انتشار تمرین نخست

انتشار تمرین مفاهیم اولیه‌ی جاوا (مرتبط با کلاس‌های حل تمرین ۷ و ۸ اسفند)

10 اسفند

کلاس شی‌گرایی

آشنایی با مقدمات و مفاهیم برنامه‌نویسی شی‌گرا در زبان برنامه‌نویسی جاوا

12 اسفند

کلاس شی‌گرایی

آشنایی با مقدمات و مفاهیم برنامه‌نویسی شی‌گرا در زبان برنامه‌نویسی جاوا

هنر کد تمیز در جاوا (قسمت اول)

متین داغیانی

بیایید از ابتدا شروع کنیم...

هنگامی که می‌خواهیم یک برنامه بنویسیم، احتمالا اولین سوالی که در ذهنمان ایجاد می‌شود، این است که «چه کار کنیم که این برنامه، کار کند؟». بعد از آن که جواب این سوال را در مدت نسبتا کوتاهی پیدا کردیم، ویرایشگر خود را باز می‌کنیم و بدون توقف محسوسی، به کد زدن می‌پردازیم؛ در نهایت، به برنامه‌ی نوشته شده نگاه می‌کنیم و از این که در نگاه اول بدون هیچ مشکلی در حال کار کردن است، به خودمان می‌بالیم!

اما این پایان ماجرا نیست. نوشتن برنامه‌ای که به درستی و بدون نقص کار می‌کند، تنها بخشی از فرآیند نوشتن یک کد خوب است. کمتر پیش می‌آید که یک برنامه‌ی بزرگ، بعد از متولد شدن، به حال خود رها شود؛ چرا که حتی بهترین برنامه‌هایی که ساخته شده‌اند هم به بازنگری، توسعه و رفع اشکالات (دیباگ شدن) نیازمندند. اینجاست که **تمیز کد زدن**، بیش از هر چیز دیگری، اهمیت پیدا می‌کند. تمیز کد زدن، هنری است که به ما کمک می‌کند تا بتوانیم از کدی که نوشته ایم به خوبی مراقبت کنیم و آن را توسعه دهیم، و حتی زمینه‌ی همکاری دیگر برنامه‌نویسان را در پروژه‌مان فراهم کنیم.

چگونه تمیز کد بزنیم؟

یادگیری هر مهارتی، از دو بخش تشکیل شده است: دانش و کار. شما باید اصول، الگوها، تمرین‌ها و شیوه‌های اکتشافی که یک هنرمند می‌داند را یاد بگیرید و آن دانش را با انگشتان، چشم‌ها و تمام وجودتان حس کنید و سخت تمرین کنید! فرض کنید می‌خواهید یک شناگر حرفه‌ای شوید. برای این کار، تعداد زیادی مقاله و مجلات ورزشی مختلف را مطالعه می‌کنید و چندین ویدیوی آموزشی از انواع تکنیک‌های شنای حرفه‌ای را مشاهده می‌کنید؛ با همه‌ی این‌ها، احتمالا پس از اینکه برای اولین بار به داخل آب می‌پرید، شاید به زحمت بتوانید خود را در سطح آب نگه دارید (اگر غرق نشوید)! نوشتن کد تمیز هم چندان به این ماجرا بی‌شباهت نیست. برای آن‌که در این هنر خبره شوید، به خواندن اکتفا نکنید و هر آن چه که فرا می‌گیرید را در برنامه‌نویسی خود به کار گیرید و آن‌ها را تمرین کنید تا رفته رفته به شگرد شما تبدیل شوند.

متغیرهای تمیز

متغیرها، همه جا هستند و ما همیشه در حال تعریف و نام‌گذاری متغیرهای گوناگون هستیم. از آن جایی که این کار را بسیار زیاد انجام می دهیم، بهتر است آن را با شیوه‌ی درست انجام دهیم. در ادامه به برخی نکات ساده و مفید برای خلق متغیرهای خوش‌نام می پردازیم.

اسامی پرمعنا

انتخاب یک اسم خوب برای یک متغیر، کاری زمان‌بر و گاهی حوصله‌سربر است، اما زمان بیشتری را برای شما در آینده‌ای نه چندان دور (که به اشکال‌زدایی و به‌روز کردن کد خود می پردازید) ذخیره خواهد کرد. بنابراین در انتخاب اسامی دقت و وسواس به خرج دهید و اگر بعدا نام بهتری به ذهنتان رسید بلافاصله کد خود را به‌روز کنید (در آینده به طور مفصل در مورد این مبحث صحبت خواهیم کرد).

روی نام‌گذاری متغیرها فکر کنید. سعی کنید از اسامی یا گروه‌های اسمی استفاده کنید، به گونه‌ای که که بتوانند به این سوالات پاسخ دهند:

- چرا تعریف شده است؟
- چه کار می‌کند؟

به مثال زیر دقت کنید:

```
int d; // elapsed time in days
```

در این مثال، اسم d هیچ گونه اطلاعاتی در مورد این‌که این متغیر برای چه تعریف شده است و نشان‌گر چه چیزی است (زمان سپری شده در چند روز) را به ما نمی‌دهد؛ به همین دلیل است که باید نام بهتری برایش انتخاب کنیم، مانند:

```
int elapsedTimeInDays;
```

متغیرها، اسم اند!

سعی کنید در نام‌گذاری متغیرهایتان، از اسم‌ها استفاده کنید. استفاده از فعل یا صفت به تنهایی توصیه نمی‌شود. هم‌چنین، اسامی مخفف یا خاص را در نام‌گذاری به کار نبرید.

```
int best_number;          // Correct

int best;                 // Improper

int the_sample_result;    // Correct

int smplerslt;            // Improper
```

از نام‌های طولانی، نترسید!

طولانی بودن نام متغیر، بهتر از بی‌معنا بودن آن است. در یک برنامه شاید صدها یا هزاران متغیر وجود داشته باشند و اگر تمام آنها را با حروف الفبا یا ترکیب آنها با اعداد تعریف کرده باشید، شاید هرگز نتوانید بفهمید که این متغیر چیست و چه اطلاعاتی را در خود ذخیره می‌کند. البته نباید در این موضوع زیاده‌روی کنید؛ طول نام متغیر به شرطی که معنادار باشد و

اطلاعات کافی درباره متغیر و کاربردش به ما بدهد، باید تا حد امکان کوتاه باشد.

```
int s = a + b; // Improper

int sumOfTwoVariables = a + b; // Correct
```

جدا کردن کلمات

در زبان‌های مختلف برنامه نویسی، از انواع مختلفی از قواعد برای جدا کردن واژه‌های تشکیل‌دهنده‌ی اسامی متغیرها استفاده می‌شود (به عنوان مثال، PascalCase، separated_by_underscore و...). در زبان جاوا برای تعریف متغیرها از روش camelCase (حرف نخست کلمه‌ی اول کوچک، حرف نخست سایر کلمات بزرگ) استفاده می‌کنیم، مثلا:

```
int client_message_code; // Incorrect

int ClientMessageCode;   // Incorrect

int clientMessageCode;   // Correct
```

عبارات شرطی و حلقه‌ها

عبارت‌های شرطی از پراستفاده ترین جملات در مکالمات عادی و روزانه ما هستند. شاید به همین دلیل است که سر و کله آن‌ها در اکثر زبان‌های برنامه نویسی پیدا می‌شود. به علاوه، حلقه ها نیز از پررنگ‌ترین ویژگی‌های هر زبان محسوب می شوند تا امکان مدیریت عملیات پی‌درپی و متوالی را در اختیار توسعه‌دهنده‌ها قرار دهند؛ البته، این ابزارهای مفید می‌توانند بعضا بسیار گیج‌کننده نیز باشند، مانند زمانی که با تعداد بسیاری پرانتز، آکولاد، حلقه های تو در تو و... دست و پنجه نرم می کنید. در ادامه، قصد داریم تا با اصولی آشنا شویم که می توانند ما را از این سردرگمی‌ها نجات دهند.

شرط‌های مرکب

از جمله مواردی که می‌تواند به شدت کدمان را ناخوانا کند، شرط‌های مرکب یا چندخطی است. برای آنکه بتوانیم بهتر آن‌ها را تحلیل و بررسی کنیم، بهتر است هر شرط را در یک خط بنویسیم (تمام عملگرها باید یا در ابتدا و یا در انتهای خطوط آورده شوند):

```
/* valid style */

if (condition1 ||

(condition2 && condition3) ||

condition4 ||

(condition5 && (condition6 || condition7))) {

        // commands

}
```

از شر آکولاد، خلاص شوید!

همان‌طور که می‌دانید، آکولادها بخش جدایی‌ناپذیر بسیاری از زبان‌های برنامه‌نویسی – از جمله جاوا – هستند، به طوری که می‌توانید ردپای آن‌ها را همه جا پیدا کنید، مانند شرط‌ها، حلقه‌ها، توابع و بسیاری از مکان‌های دیگر؛ در نتیجه، گریز از آن‌ها به طور کامل، تقریبا ناممکن است؛ اما راه‌هایی وجود دارند که می‌توانند به ما در استفاده‌ی بهینه‌تر از آن‌ها، کمک کنند. به قطعه کد زیر دقت کنید. فرض کنید سه متغیر زیر تعریف شده‌اند و ما می‌خواهیم از آن‌ها استفاده کنیم تا وضعیت سلامتی یک بازیکن را در یک بازی اکشن، به او گزارش دهیم:

```
boolean isHurt;

String playerStatus;

int health = 100;
```

برای این کار، از عبارات شرطی زیر استفاده می‌کنیم:

```
if (health < 5) {

    isHurt = true;

} else {

    isHurt = false;

}

if (isHurt) {

    playerStatus = "You're hurt!";

} else {

    playerStatus = "You're cool dude :)";

}
```

به نظر همه چیز مرتب است! اما با کمی دقت، می‌توان مشاهده کرد که می‌توان با یک اقدام ساده این قطعه کد را کوتاه‌تر کرد، بی‌آن‌که به خوانایی آن لطمه‌ای وارد شود: «اگر عبارتی شرطی دارید که تنها شامل یک خط دستور است، آکولادها را حذف کرده و دستور را در همان خط شرط و با یک فاصله بعد از اتمام پرانتز شرط بنویسید، مشروط بر آن‌که

خط از ۱۵۰ کاراکتر طولانی‌تر نشود». بدین‌ترتیب، قطعه کد بالا را به‌روز می‌کنیم:

```
if (health < 5) isHurt = true;

else isHurt = false;

if (isHurt) playerStatus = "You're hurt!";

else playerStatus = "You're cool dude";
```

همان‌گونه که می‌بینید، نسخه‌ی به‌روز شده، به زبان انگلیسی نزدیک‌تر است. البته، در این مثال خاص، می‌توان حتی باز هم کد را ساده‌تر کرد، ولی باید دقت کنیم که این ساده‌سازی، سبب کاهش خوانایی کد نشود. در این مثال به خصوص، ساده‌سازی به شکل زیر (جای‌گزینی شرط با عبارت boolean)، توصیه می‌شود:

```
isHurt = health < 5;

if (isHurt) playerStatus = "You're hurt!";

else playerStatus = "You're cool dude";
```

نکته مهم: ساده‌سازی شرط با استفاده از عملگر سه‌گانه شرطی ? و : را تنها زمانی به کار ببرید که صورت عبارت شرطی و مقدار آن، بسیار ساده، سرراست و به سرعت و آسانی، قابل فهم باشد.

توابع (متدها)

تابع راهکار هوشمندانه‌ای است که بیش از آن‌چه به نظر می‌آید می‌تواند در طراحی یک برنامه‌ی خوب به شما کمک کند. با استفاده از توابع متعدد در برنامه‌ی خود، می‌توانید با کدهای طولانی چندصدخطی خداحافظی کرده و به طور چشمگیری برنامه خود را بهینه سازید. در ادامه با اصولی آشنا می‌شویم که به ما کمک می‌کنند تا بتوانیم بهتر از این ابزار استفاده کنیم و کیفیت کدهای خود را بالاتر ببریم.

متدهای خوش‌نام

فرض کنید در یک پروژه‌ی بزرگ (مانند پروژه‌ی AP!)، بیش از هزاران متد مختلف با عملکردهای متنوع در یک کلاس تعریف شده باشند. شکی نیست که یافتن یک متد خاص در میان آن‌ها یا فهمیدن منطق و علت تعریف آن به بزرگترین معمای زندگیتان تبدیل خواهد شد! اینجا همان نقطه‌ای است که انتخاب اسامی مناسب و اصولی می‌تواند کار شما را بسیار آسان‌تر کند. برای نام‌گذاری متدها:

- از افعال امری یا پرسشی استفاده کنید.
- تابع را **توصیف** کنید، طوری که بتوان با خواندن آن متوجه شد که این تابع **چه کار** می کند و **چرا** تعریف شده است.

```
public String name() // wrong

public String getCustomerName() // Correct
```

یا به عنوان مثالی دیگر،

```
public boolean adult() { // Improper

	return age >= 18;

}

public boolean isAdult() { // Correct

	return age >= 18;

}
```

کوتاه و مختصر

یکی از دلایلی که از توابع استفاده می‌کنیم، تقسیم کد اصلی و منطق آن به قسمت‌های کوچک‌تر و در عین حال کارا است تا از نامفهوم شدن آن جلوگیری کنیم، اما توابع بزرگ و طولانی به همان نسبت می‌توانند دردسرساز باشند؛ بنابراین بهتر است تا حد امکان سعی کنیم تا توابع را در تعداد خطوط کمتری پیاده سازی کنیم. این که تعداد خطوط یک تابع استاندارد حداکثر چقدر باید باشد به عوامل متفاوتی بستگی دارد و می‌تواند بسته به پروژه، عملکرد تابع، زبان برنامه نویسی و ... این میزان متغیر باشد. با این وجود، یکی از اصول کدنویسی تمیز در جاوا، بیان می‌دارد که طول یک تابع تمیز در جاوا، بدون حساب سرآیند و آکولاد باز و بسته‌ی تابع، حداکثر **۲۰ تا ۳۰ خط** است.

توابع مسئولیت‌پذیر

«یک تابع، باید تنها **یک** کار انجام دهد و آن را به **بهترین** شکل ممکن به سرانجام برساند». شاید بتوان گفت این جمله مهم‌ترین نکته‌ای است که در پیاده سازی توابع باید به آن توجه کنیم. به این مثال توجه کنید:

فرض کنید می خواهید برنامه‌ای بنویسید که تعداد اعداد اول سه رقمی را چاپ کند. برای این کار لازم است تا مراحل زیر را طی کنیم:

۱. در یک حلقه بر روی اعداد ۱۰۰ تا ۹۹۹ پیمایش کنیم.

۲. به ازای هر عدد، به کمک یک حلقه اول و یا مرکب بودن عدد را تشخیص داده و در صورت نیاز، به شمارنده یک واحد اضافه کنیم.

۳. در نهایت، تعداد را چاپ کنیم.

طبق توضیحات داده شده، به جای آن‌که تمام این مراحل را در یک تابع پیاده‌سازی کنیم، باید برای هر کدام از مراحل ۱ و ۲، یک متد مجزا تعریف کرده و در نهایت جواب را در متد main چاپ کنیم:

```
public class ThreeDigitsPrimeNumbers {

    public static boolean isPrime(int n){

          // Checking...

    }


    public static int getPrimesCount() {

        int counter = 0;

        for (int i = 100; i < 1000; ++i) {

              if (isPrime(i)) counter ++;

         }

         return counter;

    }


    public static void main(String[] args) {

      System.out.println(getPrimesCount());

    }

}
```

چند میان‌بر تمیز در IntelliJ

برای سرعت کار در رعایت قوانین کدنویسی تمیز، می‌توانید از shortcutهای زیر در IntelliJ استفاده کنید:

- میان‌بر Ctrl+Alt+L (در اوبونتو، Ctrl+Alt+Win+L و در مک، Option+Command+L): این میان‌بر، بسیاری از اصول کلین‌کد را مستقیما در کد شما اعمال می‌کند؛ گرچه تمامی آن‌ها را به درستی پوشش نمی‌دهد و لازم است پس از اعمال آن، مجددا کد خود را به طور کامل بررسی کنید.
- میان‌بر Shift+F6 (در مک‌های مجهز به تاچ‌بار، Shift+Fn+F6): با فراخوانی این میان‌بر، می‌توانید کلیدواژه‌ای که انتخاب کرده‌اید (اسم متد و یا متغیر) را با یک نام بهتر، به راحتی در کل کد تعویض کنید، بدون این‌که نگران مشکلات احتمالی Replace All باشید.
- میان‌بر Ctrl+Alt+M (در مک، Option+Command+M): این میان‌بر، می‌تواند قطعه کدی که انتخاب کرده‌اید را به یک متد دیگر انتقال دهید تا متد اول، طولانی، شلوغ و «کثیف» نشود.

قوانین کدنویسی تمیز، به هیچ عنوان به موارد ذکر شده خلاصه نمی‌شود؛ در شماره‌ی آینده‌ی کدنامه نیز قسمت دوم «**هنر کد تمیز در جاوا**» عرضه خواهد شد تا بتوانید از نکات ذکر شده در آن، در انجام تمرین‌های درس، بهره ببرید.

کدنامه چیست؟

دستیاران آموزشی دروس مبانی برنامه‌سازی و برنامه‌سازی پیشرفته، طی نیم‌سال‌های گذشته، مطالب متعدد درسی را در قالب پی‌دی‌اف‌های گوناگون منتشر می‌کردند. از نیم‌سال گذشته، برآن شدیم تا با تشکیل «کدنامه»، تمامی مطالب آموزشی را در این قالب تدوین کنیم تا هم به سرعت بتوان آن‌ها را مطالعه کرد و هم بتوانیم به این بهانه، محتوای کاربردی‌تری را در اختیار دانش‌جویان قرار دهیم. توجه شود که «کدنامه»، به هیچ عنوان، یک نشریه **نیست** و زمان عرضه مشخص نیز ندارد، بلکه تنها قالبی جدید برای عرضه همان مطالب و محتوای آموزشی است و در آن تنها به مباحث درسی مخصوص درس برنامه‌سازی پیشرفته نیم‌سال جاری پرداخته می‌شود.

مسئول تیم محتوای آموزشی و کدنامه: سید پارسا نشایی

نویسندگان گرانقدر تیم محتوای آموزشی و کدنامه:

- سید پارسا نشایی
- محمدمهدی برقی
- علیرضا حسین‌خانی
- متین داغیانی
- علی حاتمی

مطالعه‌ی مطالب «کدنامه»، برای انجام دادن بهتر پروژه و تمرین‌های درس برنامه‌سازی پیشرفته، اکیدا توصیه می‌شود.