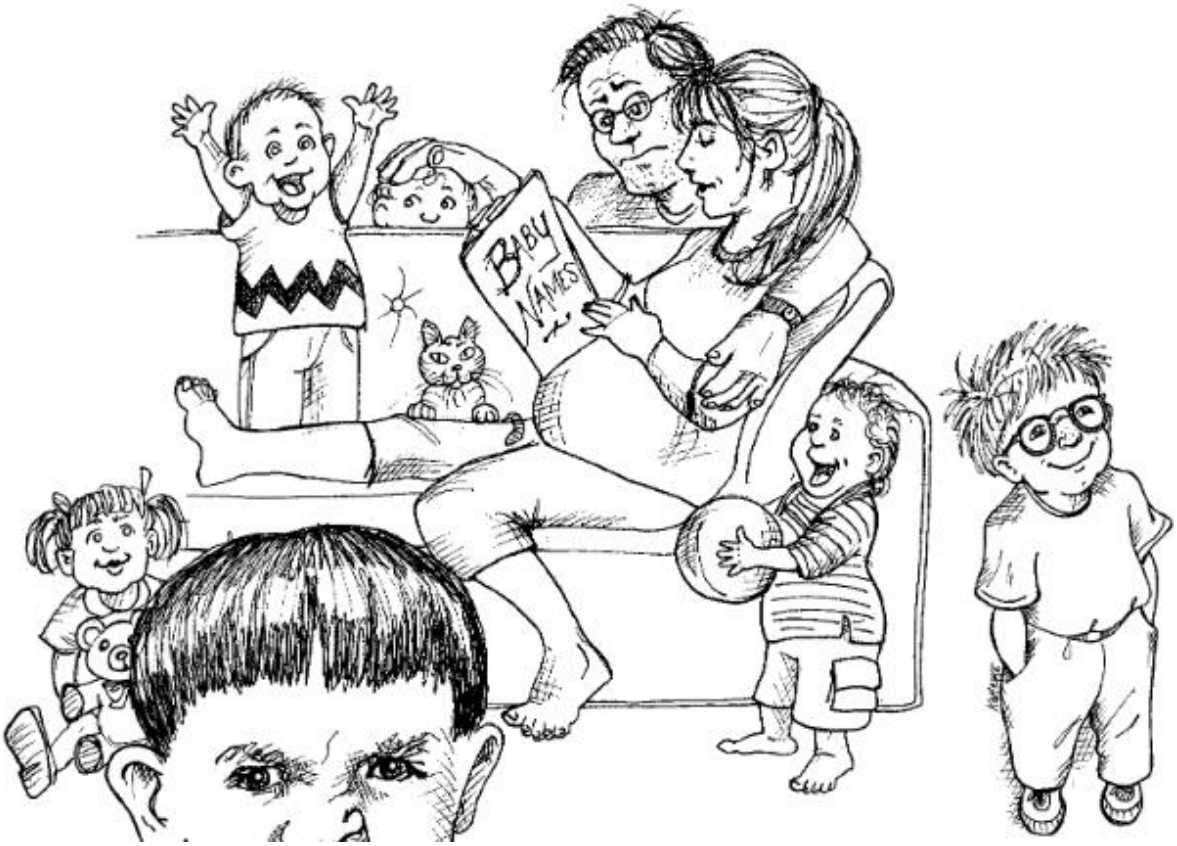
2

اسامی بامعنی



مقدمه

اسم ها همه جای نرم افزار وجود دارند. ما متغیر ها، تابع ها، آرگومان ها، کلاس ها و پکیج هایمان را نام گذاری میکنیم. ما فایل های سورس و دایرکتوری هایی که آنها را شامل میشوند را نام گذاری میکنیم. ما حتی فایل های jar و war و ear را نامگذاری میکنیم. ما نامگذاری میکنیم، نامگذاری میکنیم و نامگذاری میکنیم. از آنجایی که این کار را

خیلی زیاد انجام میدهیم ، بهتر است که آن را با شیوه درست دهیم. آنچه که در ادامه میخوانید،قوانینی ساده برای خلق اسم های خوب هستند.

استفاده از اسم های بیان کننده منظور (Intention-Revealing Names)

کاملا واضح است که اسم ها باید منظور شما را بازتاب دهند. چیزی که ما میخواهیم به شما بگوییم این است که ما در این قضیه کاملا **جدی** هستیم. انتخاب کردن اسم های خوب زمانبر است ولی زمان بیشتری از آنچه میگیرد را ذخیره میکند. پس به اسم هایتان توجه کنید و هر وقت اسم های بهتری یافتید، آنها را عوض کنید. هر کسی که کد شما را بخواند (شامل خودتان) از انجام این کار خوشحال میشود.

اسم یک متغیر، تابع یا کلاس، باید به تمام سوال های بزرگ پاسخ دهد. اسم باید بگوید که چرا وجود دارد، چه میکند و چگونه استفاده میشود.اگر اسمی به کامنت نیاز داشته باشد،پس منظور خود را نمیرساند.

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

اسم d در کد بالا هیچ منظوری را نمیرساند. این اسم احساس گذشت روز ها را ایجاد نمیکند. ما باید اسمی انتخاب کنیم که مشخص کند چه چیزی در حال اندازه گیری شدن است و واحد و مقیاس آن اندازه گیری چیست.

```
int elapsedTimeInDays;

int daysSinceCreation;

int daysSinceModification;

int fileAgeInDays;
```

انتخاب اسم هایی که منظور ما را القا میکند ، فهمیدن و تغییر دادن کد را راحت تر میکند. میتوانید بگویید کد زیر چه میکند؟

```
Public List<int[]> getThem() {

  List<int[]> list1 = new ArrayList<int[]>;

  For (int[] x : theList)

    If (x[0] == 4)

      List1.add(x);

  Return list1;

)
```

چرا گفتن کاری که این کد انجام میدهد سخت است؟ اینجا هیچ کد پیچیده ای وجود ندارد. فاصله ها و تورفتگی ها کاملا معقول هستند. در کل سه متغیر و دو ثابت استفاده شده است. اینجا هیچ کلاس عجیب یا متد پیچیده ای وجود ندارد، فقط یک لیست از آرایه ها وجود دارد.

مشکل سادگی کد نیست ، صراحت کد است؛ میزان صریح بودن کد در بیان منظور خودش. صریح بودن کد نیازمند این است که ما بتوانیم به سوال های نظیر این ها پاسخ دهیم:

1. چه چیز هایی در `theList` وجود دارد؟
2. اهمیت عضو صفرم در `theList` چیست؟
3. اهمیت مقدار `4` چیست؟
4. چگونه از لیستی که برگشت داده میشود استفاده کنم؟

جواب سوال ها در مثال قبل تشخیص نیستند، ولی باید مشخص شوند. فرض کنید ما داریم روی یک بازی مین روبی کار میکنیم. ما میدانیم که صفحه بازی یک لیست از سلول هاست که با عنوان theList نمایش داده میشود. بیایید نام آن را به gameBoard تغییر دهیم.

هر سلول در صفحه توسط یک آرایه ساده نشان داده میشود. همچنین این را میدانیم که مقدار صفرم ، وضعیت سلول است و اینکه وضعیت 4 یعنی "پرچم گذاری شده". بیایید با دادن اسم های این کانسپت ها کد را به شکل قابل توجهی بهبود دهیم:

```
Public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new ArrayList<int[]>();
  For (int[] cell : gameBoard)
    If (cell[STATUS_VALUE] == FLAGGED)
      flaggedCells.add(cell);
  return flaggedCells;
}
```

دقت کنید که سادگی کد تغییری نکرده است و هنوز هم همان تعداد مقادیر ثابت و متغیر وجود دارد، دقیقا با همان تعداد تورفتگی و بیرون زدگی.

ما میتوانیم پا را فراتر بگذاریم و به جای یک آرایه از اعداد ، یک کلاس ساده برای سلول ها بنویسیم.این کلاس میتواند یک تابع را شامل شود (که منظور خود را به درستی به نمایش میگذارد و آن را isFlagged می نامیم) که باعث میشود اعداد عجیب از کد حذف شوند:

```
Public List<Cell> getFlaggedCells() {
  List<Cell> flaggedCells = new ArrayList<Cell>();
  For (Cell cell : gameBoard)
    If (cell.ifFlagged())
      flaggedCells.add(cell);
  return flaggedCells;
)
```

با این تغییرات ساده، دیگر فهمیدن اینکه چه کاری در حال انجام است سخت نیست. این قدرت انتخاب کردن اسم های خوب است.

خودداری از دادن اطلاعات اشتباه

برنامه نویس ها باید از به جا گذاشتن اطلاعات اشتباه که معنی کد را خراب میکنند خودداری کنند. ما باید از کلماتی که مفهوم آنها با منظور ما فاصله زیادی دارد دوری کنیم. برای مثال، کلمات hp ، aix و sco نام های ضعیفی برای متغیر ها هستند زیرا از اسامی یونیکس پلتفرم هستند. حتی اگر شما در حال نوشتن یک Hypotenuse هستید و hp مخفف خوبی برای آن به نظر میرسد، ممکن است اطلاعات ناسازگار در کد به جا بگذارید.

هرگز به یک لیست از اکانت ها اسم accountList را ندهید زیرا آن واقعا یک لیست است ولی کلمه List م به معنی چیزی است که به برنامه نویس ها اختصاص دارد. اگر یک کانتینر اکانت ها را نگهداری میکند، به این معنی نیست که یک List است، این ممکن است به اطلاعات غلط منجر شود. پس accountGroup یا bunchOfAccounts یا فقط accounts اسم های بهتری هستند.

از استفاده از اسم هایی که تفاوت های کوچکی با هم دارند خودداری کنید. چه تضمینی وجود دارد که بعدا دو شی با نام های XYZControllerForEfficientHandlingOfStringsin و XYZControllerForEfficientStorageOfStrings با هم اشتباه گرفته نشوند؟ هر دو اسم شکل یکسانی دارند.

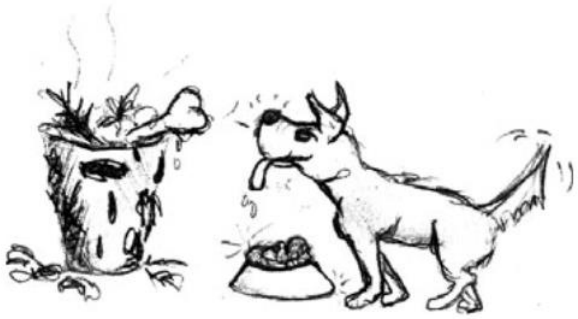
در انتخاب اسم به تلفظ آن دقت کنید. استفاده از اسم با تلفظ اشتباه نوعی اطلاعات غلط است. به کمک محیط های مدرن جاوا ما میتوانیم از تکمیل شدن خودکار کد ها لذت ببریم. ما چند حرف تایپ میکنیم و دکمه ای را فشار میدهیم که لیستی از اسامی قابل استفاده برای تکمیل کلمه را به ما نشان میدهد. مرتب شدن نام های چیزهای مشابه،بسیار مفید است و تفاوت ها را آشکار میکند.

یک مثال بد از نام هایی که اطلاعات غلط میدهند، نام هایی هستند که از حروفی استفاده میکنند که شبیه حروف دیگر هستند. مثلا حرف I و L ، اگر به صورت I و l نوشته شوند،مطمئنا اشتباه گرفته میشوند. همچنین حرف l و 1 نیز ممکن است اشتباه گرفته شوند.

```
int a = l;
if(0==1)
  a=01;
else
  l = 01;
```

شاید فکر کنید این اتفاق محال است ؛ اما ما کد هایی را بررسی کرده ایم که چنین مواردی در آنها به وفور وجود داشت. در وهله اول نویسنده پیشنهاد میکند که فونت متن را تغییر دهید که باعث آشکار شدن بهتر تفاوت ها میشود، اما این راه حل باید به توسعه دهندگان آینده به صورت شفاهی یا کتبی منتقل شود. بهترین راه حل یک تغییر نام ساده است.

تفاوت های بامعنا ایجاد کنید



برنامه نویس ها با نوشتن کد های کامپایلر پسند برای خودشان مشکل ایجاد میکنند. برای مثال شما نمیتوانید از یک اسم در دو متغیر مختلف استفاده کنید،شما مجبورید یکی از اسم ها را کمی تغییر دهید. گاهی اوقات اینکار را با تغییر دادن املای کلمات انجام می دهید، در اینصورت ممکن است تصحیح خطاهای املایی باعث مشکل در کامپایل نرم افزار شود.

افزودن اعداد یا حروف اضافه هرگز کافی نیست. اگر نام ها باید متفاوت باشند، پس معنی آنها نیز باید فرق کند.

اسامی شامل سری اعداد (a1,a2,…,aN) با نام گذاری اصولی در تضاد هستند. این اسم ها اطلاعات غلط نمیدهند،چون اصلا اطلاعاتی نمیدهند و کاملا بی معنی هستند. اینها هیچ سرنخی از منظور شما به دیگران نمیدهند. به مثال نگاه کنید:

```
Public static void copyChars(char a1[], char a2[] {
  For (int i=0; i<a1.length; i++) {
    A2[[i] = a1[i];
  }
}
```

این تابع خواناتر میشود، اگر از اسم های source و destination در آرگومان ها استفاده کنیم.

حروف اضافه(noise words) موارد بی معنی دیگری هستند. تصور کنید شما کلاسی به اسم Product دارید. اگر شما کلاس های دیگری با اسم های ProductInfo یا ProductData بسازید، شاید اسم های مختلفی انتخاب کرده باشید اما در معنای آنها تفاوتی وجود ندارد. کلمات Info و Data کلمات اضافه هستند، مثل a, an و the .

دقت کنید که استفاده از این پیشوند ها مشکلی ندارد، مثلا شما میتوانید از a برای همه متغیر های محلی و از the برای همه آرگومان های توابعتان استفاده کنید. در حقیقت مشکل از جایی شروع میشود که شما یک متغیر را theZork بنامید، فقط به این دلیل که متغیر دیگری به نام Zork دارید.

حروف اضافه ،زائد هستند. کلمه Variabale هیچوقت نباید در اسم یک متغیر نمایان شود. واژه table نباید در اسم یک Table نشان داده شود. چرا فکر میکنید NameString بهتر از Name است؟ آیا ممکن است Name یک عدد اعشاری باشد؟اگر جواب بله است،پس شما کل قوانین را زیر سوال برده اید! فرض کنیدکلاسی به اسم Customer و کلاسی دیگر به اسم CustomerObject دارید.از اختلاف این دو اسم چه چیزی دستگیرتان میشود؟ کدام یک از این دو اسم ، بهتر میتواند تاریخچه پرداخت های یک مشتری را نشان دهد؟

در مثال های زیر استفاده مناسب از حروف اضافه را میبینید.

```
getActiveAccount();
getActiveAccounts();
getActiveAccountInfo();
```

برنامه نویس به راحتی میفهمد که کدام تابع را نیاز دارد.

دقت کنید که حروف اضافه چه تغییری در اسم متغیر شما میدهند. تفاوت متغیر moneyAmount از money غیرقابل تشخیص است. تفاوت customerInfo از customer نیز همینطور.accountData از account و theMessage از message. اسامی قابل تشخیص باید تفاوت خود را به خواننده نشان دهند.

از اسم های قابل تلفظ استفاده کنید

انسان ها در استفاده از کلمات ماهرند. قسمت های مشخصی از مغز ما به درک مفهوم کلمات اختصاص داده شده اند.مفهوم کلمات ارتباط مستقیمی با تلفظ آنها دارد. مغز ما کلمات را با توجه به تلفظ آنها درک میکند، نه نوشتار آنها. بنابراین، بهتر است از اسم های قابل تلفظ استفاده کنید.

اگر نمیتوانید یک اسم را تلفظ کنید، نمیتوانید درباره آن بحث کنید،بدون اینکه مثل یک احمق صدا در بیاورید!

"Well,over here on the bee cee arr three cee enn tee we have a pee ess zee kyew int, see?"

میتوانید این متن را بخوانید؟

فهمیدید که این موضوع مهم است، چون برنامه نویسی یک فعالیت اجتماعی است.

یک کمپانی در برنامه خود ، مفهومی به نام genymdhms دارد (generation date, year, month, day, hour, minute and second). آنها برای اینکه این اسم را به خاطر بسپارند، قدم زنان میگویند " gen why emm dee aich emm ess". من عادت مسخره ای دارم که در آن هر چیزی را به همان شکلی که نوشته شده میخوانم،پس من شروع کردم به گفتن "gen-yah-mudda-hims." .بعدها این مفهوم توسط جمعی از طراحان و آنالیزور ها به همین اسم نامیده شد، و ما هنوز هم به درآوردن این صدای احمقانه ادامه میدهیم. از آنجایی که این برای ما مثل یک شوخی به حساب می آمد، برای ما بانمک بود. بانمک باشد یا نباشد، ما داریم اسم گذاری ضعیف را تحمل میکنیم. برنامه نویس های جدید ما نیازمند این هستند که این مفهوم را برایشان توضیح دهیم و آنها در مورد دلیل در آوردن این صداهای احمقانه به جای استفاده از قوانین صریح و کلمات بامعنای انگلیسی صحبت میکنند.مقایسه کنید:

```
class DtaRcrd102 {
   private Date genymdhms;
   private Date modymdhms;
   private final String pszqint = “102”;
};
```

با

```
class Customer {
   private Date generationTimestamp;
   private Date modificationTimestamp;
   private final String recordId = “102”;
   /*…*/
};
```

مکالمه عاقلانه اکنون امکان پذیر است:

“Hey, Mikey, take a look at this record! The generation timestamp is set to tommorrow’s date! How can that be?”

از اسامی قابل جستجو استفاده کنید

اسامی تک حرفی و ثابت های عددی این مشکل را دارند که در متن قابل پیداکردن نیستند.احتمالا پیدا کردن MAX_CLASSES_PER_STUDENT در یک متن راحت است، اما مثلا پیدا کردن عدد 7 در یک متن طولانی، سخت و زمان بر است. جستجو ها ممکن است اعداد دیگری را نیز برای شما پیدا کنند، مثل قسمت های عددی اسم فایل ها،ثابت های توابع دیگر و در عبارات گوناگونی که از اعداد با اهداف متفاوتی استفاده میکنند. وقتی یک عدد طولانی استفاده میکنید، ممکن است شخصی رقم هایی را سهوا جابجا کند و همزمان این عدد از جستجوی شما فرار میکند.

برای مثال، حرف e ضعیف ترین حرف ممکن برای نامگذاری یک متغیر برای برنامه نویسی است که نیازمند جستجو کردن است. چون این حرف پرکاربردترین حرف در زبان انگلیسی است و در هر عبارتی یافت میشود و باعث میشود حین جستجو هر

متنی را که در هر برنامه ای وجود دارد ببینید. در این راستا در هر کدی، نام های طولانی تر، بر نام های کوتاه تر برتری دارند و نام های قابل جستجو بر ثابت ها.

به نظر من نام های تک حرفی تنها میتوانند به عنوان متغیر های محلی در متد های کوتاه استفاده شوند. طول یک نام باید با دامنه استفاده آن مطابقت داشته باشد. اگر ممکن است از یک متغیر در چندین محل از یک کد استفاده کنید، ضروری است که یک اسم جستجو-دوست (search-friendly) داشته باشید. یکبار دیگر مقایسه کنید

```
For (int j=0; j<34; j++) {
  S+= (t[j]*4)/5;
}
```

با

```
int realDaysPerIdealDay = 3;
cons tint WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j=0; j<NUMBER_OF_TASKS; j++) {
  int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
  int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
  sum += realTaskWeeks;
}
```

به sum دقت کنید، این اسم کاملا مناسب نیست، اما حداقل قابل جستجو کردن است. نوشتن کد با این نامگذاری ها شاید کمی طولانی تر باشد ، اما این را هم در نظر داشته باشید که پیدا کردن WORK_DAYS_PER_WEEK راحت است زیرا فقط 5 بار از آن استفاده شده.

از رمزنگاری خودداری کنید

ما به اندازه کافی از رمزنگاری ها برای زیاد کردن دردسرمان استفاده میکنیم، لطفا آن را از چیزی که هست بیشتر نکنید.نام های رمزگذاری شده به سادگی قابل تلفظ نیستند و برای توسعه دهندگان جدید دردسر ایجاد میکنند،زیرا هر کارمند جدید باید زبان رمزنگاری شما را بیاموزد که خود باعث فشار روحی زیادی است.

نمادگذاری مجارستانی

در روزهای قبل، وقتی ما روی زبان های name-length-challenged کار میکردیم،این امر را با پشیمانی و بخاطر ضرورت نقض کردیم. Fortran رمزگذاری را اجبار میکرد. نسخه های اولیه BASIC فقط یک حرف به اضافه یک رقم را مجاز میکرد. HungarianNotation (HN) این مرحله را به سطح کاملا جدیدی رساند.

HN در Windows C API بسیار مهم به حساب می آمد،هنگامی که همه چیز یک دسته صحیح یا یک نشانگر طولانی یا یک Voidpointer یا یکی از چندین نوع "رشته" (با کاربرد ها و ویژگی های مختلف) بود. کامپایلر نوع متغیر های ورودی را بررسی نمی کرد، بنابراین برنامه نویسان برای کمک به آن در به خاطر سپردن نوع داده ها نیز به عصا داشتند.

در زبان های مدرن سیستم های بسیار غنی تری داریم و کامپایلر ها می توانند به خاطر بیاورند که هر داده ای از چه نوع است.گذشته از این موضوع،گرایش به کلاسهای کوچکتر و عملکرد های کوتاه تر نیز وجود دارد تا افراد معمولا بتوانند نقطه ی اعلام هر متغیری که میخواهند را ببینند.

برنامه نویسان جاوا نیازی به رمزگذاری ندارند.نوع اشیا مشخص است و محیط های ویرایش به گونه ای پیشرفت کرده اند که یک خطای نوع را قبل از اینکه بتوانید برنامه را کامپایل کنید،تشخیص میدهند.

پیشوندهای عضو

شما دیگر نیازی به پیشوند کردن _m در متغیر های عضو ندارید. کلاس ها و توابع شما باید به قدری کوچک باشند که نیازی به آنها نباشد و شما باید از یک محیط ویرایش استفاده کنید که اعضا را برجسته و یا رنگی و آنها را متمایز کند.

```
public class Part {
  private String m_dsc; // The textual description
  void setName(String name) {
    M_dsc = name;
  }
}


public class Part {
  String description;
  void setDescription(String description) {
    this.description = description;
  }
}
```

علاوه بر این،افراد به سرعت یاد میگیرند که پیشوند (یا پسوند) را نادیده بگیرند تا بخش بامعنای نام را ببینند. هرچه آن را بیشتر بخوانند، کمتر the را میبینند.در نهایت پیشوندها به صورت تصادفی دیده می شوند و نشانگر کد قدیمی تر هستند.