2

Smislena imena

Tim Ottinger

Uvod

Imena su svuda u softveru. Imenujemo naše varijable, funkcije, argumente, klase i pakete. Imenujemo izvorne datoteke i direktorijume koji ih sadrže. Mi imenujemo naše jar datoteke i war datoteke i ear datoteke. Imenujemo i imenujemo i imenujemo. Budući da to radimo toliko mnogo, bilo bi bolje da radimo kako treba. Sledi nekoliko jednostavnih pravila za dobijanje dobrih imena.

Koristite imena koja otkrivaju namenu

Lako je reći da imena treba da otkriju nameru. Ono što želimo da naglasimo je da se *ozbiljno* posvetimo tome. Odabir dobrih imena iziskuje vreme, ali i doprinosi uštedi na vremenu. Zato vodite računa o imenovanju i menjajte imena kada god pronađete bolja. Svi koji čitaju vaš kod (uključujući i vas) biće sretniji ako to učinite.

Naziv promenljive, funkcije ili klase trebalo bi da odgovara na sva važna pitanja. Trebalo bi vam kaže zašto postoji, šta radi i kako se koristi. Ako je za ime potreban komentar, ime ne otkriva našu nameru.

```
int d; // proteklo vreme u danima
```

Ime d ne otkriva ništa. Ne evocira osećaj proteklog vremena, niti dana. Treba da odberemo ime koje određuje šta se meri i jedinicu te mere:

```
int minutedTimeInDais;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDais;
```

Odabirom imena koja otkrivaju nameru može se znatno olakšati razumevanje i promena koda. Koja je namena ovog koda?

```
public List<int[]> getThem() {
  List<int[]> list1 = new ArrayList<int[]>();
  for (int[] x : theList)
    if (x[0] == 4) list1.add(x);
  return list1;
}
```

Zašto je teško reći šta ovaj kod radi? Nema složenih izraza. Razmak i uvlačenje su razumni. Spominju se samo tri promenljive i dve konstante. Ne postoje čak ni neke maštovite klase ili polimorfne metode, samo spisak nizova (ili tako nekako izgleda).

Problem nije u jednostavnosti koda, već u značenju koda: stepen do koga smisao nije izričit u samom kodu. Kod implicitno zahteva da znamo odgovore na pitanja kao što su:

1. Kakve su stvari u `theList`?
2. Kakav je značaj stavke sa indeksom nula u `theList`?
3. Kakav je značaj vrednosti `4`?
4. Kako bih koristio listu koja se vraća?

Odgovori na ova pitanja ne postoje u gornjem primeru koda, *ali mogli su da budu*. Recimo da se radi o igri čišćenje minskog polja (engl. *mine sweeper*). Tabla na kojoj se igra ta igrica sastavljena je od ćelija (engl. *cell*) i zove se `theList`. Preimenujmo je u `gameBoard`.

Svaka ćelija na tabli predstavljena je jednostavnim nizom. Dalje, otkrivamo da je nulta lokacija statusne vrednosti i da statusna vrednost 4 znači „označena zastavicom“ (engl. *flagged*). Samo davanjem imena tim pojmovima možemo znatno poboljšati kod:

```
public List <int []> getFlaggedCells() {
  List <int []> flaggedCells = nova ArrayList <int []>();
  for (int [] cell: gameBoard)
    if (cell [STATUS_VALUE] == FLAGGED)
      flaggedCells.add (cell);
  return flaggedCells;
}
```

Obratite pažnju na to da se jednostavnost koda nije promenila. I dalje ima sasvim isti broj operatora i konstanti, sa potpuno istim brojem ugnježđenja. Ali, kod je postao mnogo eksplicitniji.

Možemo ići dalje i napisati jednostavnu klasu za ćelije umesto da koristimo niz `ints`. Može da se doda funkcija otkrivanja namera (nazovimo je `isFlagged`) za sakrivanje magičnih brojeva. To rezultira novom verzijom funkcije:

```
public List<Cell> getFlaggedCells() {
  List<Cell> flaggedCells = nova ArrayList<Cell>();
  for (Cell cell: gameBoard)
    if (cell.isFlagged())
      flaggedCells.add (ćelija);
  return flaggedCells;
}
```

Uz ove jednostavne promene imena, nije teško razumeti o čemu se radi. U tome se ogleda snaga izbora dobrih imena.

Izbegavajte dezinformacije

Programeri moraju da izbegavaju ostavljanje loših tragova koji prikrivaju značenje koda. Treba da izbegavamo reči čija ukorenjena značenja skreću od naših nameravanih značenja. Na primer, `hp`, `aix` i `sco` bi bila loša imena promenljivih jer su to imena Unix platformi ili varijanti. Čak i ako kodirate hipotenuzu, a `hp` izgleda kao dobra skraćenica, to može predstavljati dezinformaciju.

Ne označavajte grupu računa (engl. *account*) kao accountList, osim ako nije u pitanju `List`. Reč List znači nešto specifično za programere. Ako kontejner sadrži račun to zapravo nije `List` i može dovesti do pogrešnih zaključaka.[1] Dakle `accountGroup` ili `bunchOfAccounts` ili samo prosto `account` bi bilo bolje.

Pazite pri korišćenju imena koja se razlikuju u malim detaljima. Koliko vremena treba da se uoči suptilna razlika između `XYZControllerForEfficientHandlingOfStrings` u jednom modulu i, negde malo udaljenije, `XYZControllerForEfficientStorageOfStrings`? Reči imaju strašno slične oblike.

Napisati slične pojmove na sličan način je *informacija*. Nedosledno pisanje imena je *dezinformacija*. Sa modernim Java okruženjima uživamo u automatskom dovršavanju koda. Pišemo nekoliko znakova imena i pritisnemo neku kombinaciju tastera (ako je to

1. Kao što ćemo kasnije videti, čak i ako je kontejner `List`, verovatno je bolje ne kodirati vrstu kontejnera u ime.

tako) i dobijemo spisak mogućih kompletiranja za to ime. Veom je korisno ako su imena za vrlo slične stvari poređana po abecedi i ako su razlike očigledne, jer je verovatno da će programer odabrati objekat po imenu a da ne vidi vaše bogate komentare ili čak listu metoda koje nudi ta klasa.

Zaista grozan primer dezinformativnih imena bila bi upotreba malih slova `L` ili velikih slova `O` kao promenljivih imena, naročito u kombinaciji. Problem je, naravno, što oni izgledaju gotovo u potpunosti kao konstante jedan i nula.

```
int a = l;
if (O == l)
  a = 01;
else
  l = 01;
```

Čitalac može ovo da smatra zgodnim, ali imali smo prilike da vidimo kod u kome je takvih stvari bilo u izobilju. U jednom slučaju autor koda predložio je korišćenje različitog fonta kako bi razlike bile očiglednije; rešenje koje bi trebalo preneti svim budućim programerima usmenim putem ili u pisanom dokumentu. Konačno, problem je u potpunosti i bez novih dodataka premošćen jednostavnim preimenovanjem.

Pravite smislene razlike

Programeri sami sebi stvaraju probleme kada pišu kod isključivo da bi zadovoljili kompajler (sastavljač) ili interpreter (prevodilac). Primera radi, iz razloga što ne možete da koristite isto ime koje se odnosi na dve različita pojma u istom opsegu; možda ćete doći u iskušenje da promenite jedno ime na proizvoljan način. Ponekad se to desi greškom u pisanju imena, što dovodi do iznenađujuće situacije da kada ispravite slovne greške nije moguće uraditi kompajliranje programa.[2]

Nije dovoljno dodati nizove brojeva ili zvučne reči, iako je kompajler zadovoljan. Ako imena moraju biti različita, onda treba i da znače nešto različito.

Imenovanje nizova brojeva (`a1, a2, .. aN`) suprotno je imenovanju sa namerom. Takva imena nisu dezinformativna – ona su neinformativna; ne daju nikakav zaključak o autorovoj nameri. Razmotrite:

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

2. Razmotrite, na primer, zaista groznu praksu imenovanja promenljive imenom `klass` samo zato što je `class` ime upotrebljeno za nešto drugo.

Ova funkcija se mnogo bolje čita kada se pojmovi `source` i `destination` koriste za nazive argumenata.

Prazne reči su još jedna besmislena razlika. Zamislite da imate klasu `Product`. Ako imate drugu sa imenom `ProductInfo` ili `ProductData`, napravili ste različita imena, ali ne čineći ih po značenju različitim. `Info` i `Data` su nejasne prazne reči poput `a`, `ovo`, `to` i `the`.

Imajte na umu da nema ničeg lošeg u korišćenju za prefiks reči kao što su `to` i `ono` sve dok one daju smislenu razliku. Na primer, možete koristiti `a` za sve lokalne varijable i `the` za sve argumente funkcija.[3] Problem nastaje kada odlučite da nazovete promenljivu `theZork`, a već imate drugu promenljivu koja se zove `zork`.

Prazne reči su suvišne. Reč `promenljiva` nikada ne bi trebalo da se pojavljuje u imenu promenljive. Reč `tabela` nikada ne bi trebalo da se pojavljuje u nazivu tabele. Kako je `ImeStringa` bolje od `Ime`? Da li će `Ime` ikada označavati broj sa pokretnim zarezom? Ako je tako, krši se ranije pravilo o dezinformacijama. Zamislite da u kodu nađete jednu klasu koja se zove `Kupac`, a druga da se zove `KupacObjekat`. Kako da shvatite razliku? Koja će predstavljati najbolji put do istorije plaćanja kupca?

Postoji aplikacija za koju znamo gde je to tako. Promenili smo imena da bismo zaštitili krivca, ali evo kako je to tačno izgledalo:

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

Kako programeri koji rade na ovom projektu da znaju koju od ovih funkcija treba da pozovu?

U nedostatku posebnih konvencija, promenljiva `novacIznos` ne razlikuje se od `novac`, `kupacInfo` se ne razlikuje od `kupac`, `racunPodaci` se ne razlikuje od `racun`, a `theMessage` se ne razlikuje od `message`. Razložite imena na takav način da onaj ko čita kod zna šta te razlike nude.

Koristite izgovorljive nazive

Ljudi su dobri sa rečima. Značajan deo našeg mozga posvećen je konceptu reči. A reči su, po definiciji, izgovorljive. Šteta bi bilo ne iskoristiti prednost velikog dela našeg mozga koji je evoluirao da bi se nosio sa govornim jezikom. Zato neka vaša imena budu izgovorljiva.

Ako nešto ne možete da izgovorite, ne možete o tome ni da razgovarate, a da ne zvučite kao idiot. „Pa, ovde, na frgrmn, imamo agazagbag, vidite?“ Ovo je veoma važno jer programiranje predstavlja društvenu aktivnost.

U jednoj kompaniji postoji pojam `genymdhms` (datum generisanja, godinu (`y`), mesec (`m`), dan (`d`), sat (`h`), minut (`m`) i sekunda (`s`)), tako da se izgovara: „gen vaj em di ejč em es“. Imam neprijatnu naviku da sve izgovaram onako kako je napisano, pa sam i ja počeo da govorim „gen vaj em di ejč em es“. Brojni dizajneri i analitičari su nas kritikovali

3. Ujak Bob je to radio u C++ ali je odustao od tog načina zbog modernih IDE-a koji su ga učinili nepotrebnim.