Davanje sugestija za popravljanje kvaliteta izvornog koda

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Jakovljević Aleksandar, Karanović Boris,
Popović Olivera, Veljković Marko

Jun 2021

Sažetak

Nekoliko primera kako se u programskom jeziku *Java* određeni sintaksni konstrukti mogu zameniti sintaksnim konstruktima koji su poželjniji, kao i kratak opis razloga i motiva za tu zamenu.

Sadržaj

1 Definisana promenljiva se ne koristi

U slučajevima kada je određena promenljiva u kodu definisana ali se nigde ne koristi moguće je bezbedno ukloniti iz koda. Postoji nekoliko različitih slučajeva javljanja koji su opisani u nastavku.

1.1 Lokalna promenljiva:

Ukoliko u kodu stoji

```
{
    int x;
}

ili

{
    int x = 5;
}
```

x se ne koristi nigde unutar bloka pa treba sugerisati da se x ukloni.

1.2 Parametar funkcije:

Ukoliko u kodu stoji

```
private int miniMethod(int x, int y, int z) {
    int a = x + y;
    return a;
}
```

treba sugerisati da se parametar z ukloni iz deklaracije metoda *miniMethod*.

1.3 Parametar predefinisane funkcije:

Ukoliko u kodu stoji

```
@Override
public int miniMethod(int x, int y, int z) {
 int a = x + y;
 return a;
}
```

ne treba sugerisati ništa, jer je ovo nasleđen metod i možda se neki od parametara metoda koristi u definiciji nad-metoda.

1.4 Main metod:

Ukoliko u kodu stoji

```
public static void main(String[] args) {
    ...
}
```

ne treba sugerisati ništa jer se parametar *args* ne koristi uvek, a ne možemo ga izbaciti iz deklaracije *main* metoda.

2 Definicija i dodeljivanje vrednosti promenljivama

Često se dešava da je definisana promenljivu bez inicijalizacije, a zatim u nastavku dodeljena vrednost definisanoj promenljivi. Ovo se može izbeći približavanjem definicije promenljive mestu dodele vrednosti ili inicijalizovanjem same promenljive.

2.1 Spajanje definicije i dodele vrednosti

Ukoliko unutar koda negde stoji definicija promenljive i dodela vrednosti:

```
int x;
x = 1 + 2;
```

treba sugerisati da se naredba dodele spoji sa definicijom:

```
int x = 1 + 2;
```

2.2 Spajanje definicije i dodele vrednosti - isti doseg

Ukoliko unutar koda negde stoji:

```
int x;
x se koristi u istom ili razlicitom dosegu;
izmedju ove dve linije koda;
x = 1 + 2;
```

ne treba sugerisati ništa.

3 Uklanjanje redudantne inicijalizacije polja

U programskom jeziku *Java* potpuno je nepotrebno inicijalizovati polja klasa narednim vrednostima: `0, false, null` zato što su te vrednosti podrazumevane inicijalne vrednosti polja klasa. Stoga, kako su poznate te podrazumevane vrednosti polja može se izbeći nepotrebna eksplicitna inicijalizacija. Dakle, ukoliko u kodu postoji nešto poput:

```
private static String str = null;
public final boolean flag = false;
```

trebalo bi sugerisati da se taj deo koda zameni narednim kodom:

```
private static String str;
public final boolean flag;
```

4 Vrati Optional, ne null

Primer sa sajta Java practices.

4.1 Osnovni tip:

```
int fieldX = 10;
private int giveMeInt(int parameterX){
    int x = 5;
    return x + parameterX + fieldX;
}
```

Ukoliko u *return* naredbi stoji lokalna promenljiva, parametar funkcije ili globalna promenljiva koja je osnovnog(primitivnog) tipa, ne treba sugerisati ništa, jer su promenljive osnovnog tipa podrazumevano inicijalizovane prilikom definicije.

4.2 Inicijalizovana promenljiva:

```
public List getArticles() {
    List articles = new ArrayList<>(5);
    return articles;
}
```

Lokalna promenljiva koja se koristi u *return* naredbi je inicijalizovana ili joj je dodeljena vrednost kasnije u telu funkcije. Ne treba sugerisati izmene u kodu, jer izraz 'return' naredbe ne može biti *null*.

4.3 Ostali slučajevi

Ukoliko u kodu stoji na primer:

```
public List getArticles() {
    List articles = new ArrayList<>(5);
    articles = null;
    return articles;
}
```

ili

```
public List getArticles() {
    List articles;
    return articles;
}
```

Treba sugerisati da se tip metode promeni u *Optional <T>* (u navedenom primeru *Optional <List>*) i da se izraz *return* naredbe promeni u *Optional.ofNullable(return value)* (na prethodnom primeru *return Optional.ofNullable(articles)*.

5 Razdvajanje Exception-a

Programski jezik *Java* podrzava rad sa izuzecima. Vrlo je česta praksa da programeri sav kod koji može da proizvede izuzetke stave u jedan *try*

blok i da u jednom *catch* bloku obradjuju sve izuzetke koji mogu da se dogode. To nije dobra praksa, i preporucljivo je da se obrade izuzetaka razdvoje.

Izuzeci se mogu desiti na različite načine. Neki od njih zavise od toka izvršavanja programa, i ne možemo ih detektovati ovakvom analizom. Primer: deljenje nulom. Izuzetke koji nastaju kao rezultat neuspelog poziva neke metode ili konstruktora možemo lako da obradimo. U projektu su implementirano razdvajanje nekih od najčešćih izuzetaka koji se javljaju. Sledi primer:

```
try {
    URL url = new URL("www.google.rs");
}
catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}
// Catch blok bi trebao ovako da izgleda:
catch (MalformedURLException e) {
    System.out.println("Error: " + e.getMessage());
}
```

6 Zamena konkatenacije stringova *StringBuilder*-om

U programskom jeziku *Java* stringovi su imutabilni, što znači da jednom kada su napravljeni ne mogu se više menjati. Stoga, kada konkateniramo jedan string sa drugim kreira se novi string i stari se označava za brisanje. Problem se javlja kada treba konkatenirati veliki broj stringova, jer se alocira prostor za veliki broh privremenih stringova. Iz tog razloga uvedena je klasa *StringBuilder*, koja funkcioniše kao mutabilan string objekat.

Java za jednostavnije slučajeve radi konverziju konkatenacije stringova u *StringBuilder*. Međutim za složenije situacije, poput konkatenacije unutar petlji, ta konverzija mora biti urađena ručno. Dakle, ukoliko u kodu postoji nešto poput:

```
String x = "Life";
x += "is wonderful";
```

ne bi trebalo sugerisati nikakvu zamenu. Ukoliko pak u kodu stoji:

```
String[] strings = {"a", "b", "c"};
for (var str: strings) {
    // example is a String defined earlier
    example += str;
}
```

trebalo bi sugerisati da se taj deo koda zameni sa:

```
StringBuilder builder = new StringBuilder(example);
for (var str: string) {
    builder.append(str);
}
example = builder.toString();
```

Na sličan način sugestija bi trebalo da se da i za *For* i *While* petlje.

7 Transformacija *While* u *For* petlju

Idiomi su u ustaljene jezičke konstrukcije koje postoje u svim jezicima, pa tako i u programskim. Tipičan primer idioma je sledeća naredba:

```
for (int i = 0; i < n; i++)
    ...
```

Preporuka je korišćenje ove varijante jer je najčešća i najprepoznatljivija. Prednost ove varijante je i to što je moguće definisati promenljivu po kojoj se iterira samo u nivou petlje, odnosno bez toga da ona nepotrebno "izlazi"u ostatak koda i potencijalno prouzrokuje neočekivano ponašanje. Takođe, još jedna činjenica koja koja ide u prilog *For* petlji je to što se identično ponašanje zamenom *While* petlje dobija u samo jednoj liniji koda. Dakle, ukoliko na nekom mestu u kodu stoji nešto nalik na:

```
int i = 0;
...
while (i < 10) {
    ...
    i++;
}
```

trebalo bi da sugerisati da se taj blok koda zameni sa:

```
for (int i = 0; i < 10; i++) {
    ...
}
```