Standard programiranja: C#

1 Pravila in napotki

1.1 Poimenovanje

Ustrezno poimenovanje spremenljivk, funkcij, metod, … zelo poveča razumljivost logike programa. Imena morajo razlagati KAJ in ne KAKO. Če ne moremo najti ustreznega imena za npr. spremenljivko, ki bi opisovalo čemu je spremenljivka namenjena, lahko to pomeni, da je potrebna dodatna analiza problema, ki ga rešujemo.

Za prevajalnik in nato procesor je čisto vseeno kako imamo poimenovane spremenljivke, objekte, metode, ... Je pa to zelo pomembno za tiste, ki vzdržujejo tako kodo. Zato je najbolj pomembno, da je programska koda razumljiva ljudem – programerjem.

1.1.1 PRA.2.01: Vsa imena morajo biti jasna in opisna.

Mišljena so imena spremenljivk, datotek, funkcij, razredov, … Ne omejuj dolžine imen razen, če je dolžina omejena s tehnologijo.

1.1.2 PRA.2.02: Imena naj bodo sestavljena le iz angleških besed.

1.1.3 PRA.2.03: Imena lahko vsebujejo le znake iz osnovnega nabora ASCII.

Torej nobenih čššjev, posebnih znakov, ...

1.1.4 PRA.2.04: Ime spremenljivke naj odraža (opisuje) entiteto, ki jo ta spremenljivka predstavlja.

Iz imena spremenljivke mora biti razviden namen spremenljivke. Zaradi tega se ne uporablja eno-črkovnih imen spremenljivk (razen izjemoma v posebnih primerih – npr. *i, j* ali *k* za števce v kratkih in enostavnih zankah).

1.1.5 PRA.2.05: V imenih naj bodo samo alfanumerični znaki.

1.1.6 NAP.2.01: Imena naj ne bodo predolga in ne prekratka.

Idealne dolžine so nekako med 10 in 18 znaki. Kratka imena nosijo premalo informacij, predolga pa so zamudna za tipkanje in nepregledna.

1.1.7 PRA.2.06: Prvi znak v imenu spremenljivke naj bo mala črka.

Primer: `idJobSeeker`

1.1.8 PRA.2.07: Prvi znak v imenu funkcije (metode) naj bo velika črka.

Primer: `GetJobSeekers`

1.1.9 PRA.2.08: Vsaka beseda v imenu naj se začne z veliko črko.

Primer: `deleteAllRelatedData`

1.1.10 PRA.2.09: Imena razredov naj se začnejo z veliko začetnico (Pascalski način)

Primer: `MyMembershipProvider`

1.1.11 NAP.2.02: Ne uporabljaj Madžarske notacije za poimenovanje spremenljivk.

Pri Madžarski notaciji je ime spremenljivke vezano na njen tip – namesto, da bi bilo iz imena razvidno čemu služi spremenljivka. Poleg tega moramo ob vsaki spremembi tipa spremenljivke spremeniti tudi njeno ime, kar otežuje vzdrževanje.

1.1.12 NAP.2.03: Globalnih spremenljivk praviloma ne uporabljamo.

1.1.13 PRA.2.10: Imena globalnih spremenljivk naj se začnejo z g_.

Praviloma se globalnim spremenljivkam izogibamo. Če pa jih že uporabljamo, jih s predpono »g_« jasno označimo. V ostalem so pravila za poimenovanje globalnih spremenljivk enaka kot za ostale spremenljivke.

1.1.14 PRA.2.11: Imena konstant naj bodo z velikimi črkami. Posamezne besede naj bodo ločene s podčrtajem.

Primer: `THIS_IS_CONST`

1.1.15 NAP.2.04: Funkcijo (metodo), katere glavni namen je izvedba neke akcije, poimenuj v obliki GlagolObjekt.

Primer: `GetNextEmployeeID ()`

1.1.16 NAP.2.05: Funkcijo (metodo), ki samo vrne neko vrednost, poimenuj tako, da je razvidna vrednost, ki jo vrača.

Primer: `GetNextEmployeeID ()`

1.1.17 NAP.2.07: Okrajšav v imenih se je potrebno izogibat.

1.1.18 NAP.2.08: V imenih ne uporabljamo podčrtajev.

Razen pri poimenovanju konstant.

1.1.19 NAP.2.09: V imenih (predvsem kratkih) se izogibajmo črk O, o, I in l ter številk 1in 0.

1.1.20 NAP.2.10: Ime DLLjev naj bo enako imenu imenskega prostora, ki ga vsebuje.

1.1.21 PRA.2.13: Ime datotek z izvorno kodo naj bo enako imenu razreda, ki ga vsebujejo.

Posledično to pomeni tudi, da naj bo samo en razred v eni datoteki. Pa tudi, da veljajo za uporabo malih/velikih črk v imenih datotek enaka pravila kot za poimenovanje razredov.

1.2 Komentarji

1.2.1 NAP.3.01: Komentarje se vedno piše sproti, ko se programira.

Takrat najbolj vemo kaj delamo oz. želimo narediti. Kasneje lahko določene (pomembne) podrobnosti pozabimo.

1.2.2 PRA.3.01: Komentarji naj opisujejo kaj naj bi koda delala, ne pa kaj dela.

Primer: komentar »izračuna se povprečna plača« je boljši kot »sešteje se plača vseh delavcev in deli s številom vseh delavcev«.

1.2.3 NAP.3.02: Komentar, ki razlaga kaj delajo posamezne vrstice kode, je neuporaben za vzdrževalce (prav pride le komu, ki se želi naučiti programskega jezika).

Edina izjema bi bila lahko koda v zbirniku oz. podobnem nižje-nivojskem jeziku. V ostalih jezikih je običajno dovolj, če so s komentarji opremljeni segmenti kode veliki med 3 in 10 vrsticami.

1.2.4 NAP.3.03: Vsak pomembnejši blok kode mora biti komentiran.

1.2.5 PRA.3.03: Komentarji so enako zamaknjeni kot koda, ki jo opisujejo.

1.2.6 PRA.3.04: Komentarji naj bodo v svojih vrsticah, ne v istih kot koda.

1.2.7 NAP.3.04: Pred komentarjem naj bo prazna vrstica.

Če je že del kode tako pomemben, da ga komentiraš, bo z dodatno prazno vrstico še posebej označen.

1.2.8 PRA.3.05: Vsaka datoteka z izvorno kodo naj ima standardno glavo s podatki o vsebini in namenu.

Priporočena oblika:

```
/// <summary>
/// Opis vsebine</summary>
/// <remarks>
/// dodatne opombe (po potrebi)</remarks>
/// <copyright></copyright>
```

1.2.9 PRA.3.06: Vsaka funkcija naj se začne z opisom kaj naj bi funkcija delala, opisom parametrov in kaj vrača.

Priporočena oblika:

```
/// <summary>
/// opis
/// </summary>
```

```
/// <param name="param1">opis parametra param1 (opcijsko)</param>
/// <param name="param2">opis parametra param2 (opcijsko)</param>
/// <returns>kaj vrača (opcijsko)</returns>
```

1.2.10 PRA.3.07: Nedokončani deli kode morajo biti označeni s komentarjem v obliki TODO <kdo>, <opis>

<kdo> je oznaka programerja, ki je označil kodo kot nedokončano, v <opis> pa je vsaj razlog zakaj ni bila dokončana.

1.2.11 PRA.3.08: Za komentarje uporabljamo le //.

Se pravi vsaka vrstica komentarja mora vsebovati // - bločnih komentarjev (/*...*/) se ne uporablja.

1.2.12 PRA.3.09: Komentiranje funkcij, metod, procedur,… - v glavi morajo biti naslednji podatki:

- ***Čemu je funkcija (metoda) namenjena in kaj naj bi delala. (/// <summary>)***
- ***Kaj metoda vrača. (/// <returns>)***
- ***Znane hrošče. Navesti je potrebno primere, v katerih metoda ne deluje pravilno. (/// <bug>)***
- ***Katere napake sporoča. Dokumentiramo vse napake, ki jih metoda sporoča in v katerih primerih. (/// <exception>)***
- ***Razloge za njeno vidnost. Mogoče se bo kdo kdaj spraševal, zakaj je metoda npr.*** *private* ***(/// <remarks>)***
- ***Zgodovina sprememb. Kadar spreminjamo metodo moramo navesti, kdo je spremenil metodo, kaj je bilo spremenjeno ter kdaj in zakaj. (Če se uporablja sistem za nadzor nad verzijami (npr. SourceSafe), to ni nujno.)***
- ***Primer klica metode. Najlažji način da nekdo ugotovi, kako koda dela, je primer. (/// <example>)***
- ***Potrebni pogoji. V katerih pogojih metoda deluje pravilno. Če so od klica metode odvisni drugi deli kode (njihova pravilnost), je potrebno navesti tudi to. (/// <presumption>)***

1.2.13 PRA.3.10: Komentiranje razredov: v glavi morajo biti naslednji podatki:

- ***Namen razreda (/// <summary>*** **in** ***<remarks>)***
- ***Zgodovino razreda. Povemo kdo je spreminjal razred, kdaj, zakaj in kako. (Če se uporablja sistem za nadzor nad verzijami (npr. SourceSafe), to ni nujno.)***

1.3 Oblika kode

1.3.1 NAP.4.01: Vrstica naj bo dolga največ 79 znakov.

1.3.2 NAP.4.02: Vrstice daljše od 79 se razlomi v več vrstic tako, da posamezna vrstica ne presega 79 znakov.

1.3.3 NAP.4.03: Za razmejitev kode, uporabljaj prazne vrstice.

Po eno vrstico znotraj funkcij, po dve za razmejitev posameznih funkcij, 3-4 vrstice za razmejitev posameznih sekcij v datotekah z izvorno kodo (npr. razredov).

1.3.4 NAP.4.04: Naj bo presledek pred in za vsakim operatorjem ali ukazom.

1.3.5 NAP.4.05: Za vsako vejico v seznamu parametrov naj bo presledek.

1.3.6 NAP.4.06: Za oklepajem in pred zaklepajem naj bo presledek.

1.3.7 NAP.4.07: Naj bo presledek pred in za indeksi tabel (arrays).

1.3.8 NAP.4.08: Označbe začetka in konca bloka (begin-end, {},…) naj bodo v svojih vrsticah.

1.3.9 NAP.4.09: Za kodo se vedno uporablja "monotype" pisava.

1.3.10 NAP.4.10: Vsak del SQL stavka naj bo v svoji vrstici.

Primer:

```
SELECT Surname, Name
  FROM JobSeekers
  ORDER BY Surname, Name
```

1.3.11 NAP.4.11: Zgradba razredov naj bo naslednja:

- glava (opis,...)
- podpora življenjskemu ciklu (»constructors«, »destructors«, »initialization«)
- prekrite metode (»overrides«)
- javni
  - tipi
  - metode
- zaščiteni (»protected«)
  - tipi
  - metode
  - spremenljivke (atributi)
- privatni
  - tipi
  - metode
  - spremenljivke (atributi)
- »friend« razredi

1.3.12 NAP.4.12: Spremenljivke naj ne bodo deklarirane kot javne, ampak se raje uporabi metode za dostop (Set/Get).

1.3.13 PRA.4.02: V vrstici naj bo le po en stavek oz. ukaz.

1.4 Tehnika programiranja

1.4.1 NAP.5.01: Tudi za števce v zankah je priporočljivo uporabljati opisna imena.

Nujno pa pri več vgnezdenih zankah in pri zankah katerih telo je daljše od nekaj vrstic.

Primer: `vrstica` namesto `i`

1.4.2 NAP.5.02: Potrebno se je izogibati izhodom iz zank z break, return, goto in podobnimi stavki.

1.4.3 NAP.5.03: Uporabljaj "for" zanko, kjer je mogoče.

1.4.4 PRA.5.01: Znotraj "for" zank ne spreminjaj vrednosti števca.

A nasprotnem se zmanjša razumljivost kode in s tem poveča verjetnost napak. Če moramo početi kaj takega s števcem, je potrebno razmisliti o uporabi kakšnega drugega tipa zanke npr. *»while«*.

1.4.5 PRA.5.02: Ne uporabljaj goto stavka.

1.4.6 PRA.5.03: Vsi "flow control" ukazi (if, while, for, switch,..) morajo biti opremljeni z blokom ({ .. }), četudi je le-ta prazen.

Primer:

```
if (testResult == null)
{
}
else
{
 Calculate(testResult);
}
```

1.4.7 PRA.5.04: V funkcijah naj bo samo po en return stavek.

1.4.8 NAP.5.04: Deklariraj spremenljivke čim bliže njihovi prvi uporabi.

1.4.9 NAP.5.05: Ob deklaraciji tudi inicializiraj spremenljivko (če je možno).

Primer: `private string firstWeekDay := »Monday«;`

1.4.10 NAP.5.06: Deklaracije spremenljivk naj bodo komentirane.

Ne glede na to kako opisno je njeno ime.

1.4.11 NAP.5.07: Velike, zapletene dele kode se razbije v več manjših modulov.

Priporočljiva maksimalna velikost posamezne procedure, funkcije,... je do okrog 80 vrstic oz. do 2 ekrana (skupaj z vsemi komentarji in praznimi vrsticami).

1.4.12 NAP.5.08: Spremenljivke naj imajo čim manjšo vidnost.

Začnemo tako, da so vse spremenljivke lokalne bloku kode ali proceduri. Kasneje jim po potrebi postopoma povečujemo vidnost (lažje je iz lokalne spremenljivke narediti globalno kot pa obratno). Globalnih spremenljivk naj bo čim manj!

1.4.13 PRA.5.05: Namesto "golih" števil in tekstovnih nizov uporabljaj konstante.

Sama številka (»magic number«) oz. niz običajno ne pove nič o namenu oz. pomenu – posledično je tako kodo težje vzdrževati.

Primer:

namesto

```
for i = 1 to 7
```

je bolje

```
for i = 1 to DAYS_IN_WEEK
```

1.4.14 PRA.5.06: Pri matematično-logičnih izrazih se ne zanašaj na prioriteto operatorjev, temveč uporabljal oklepaje.

1.4.15 NAP.5.09: Izjeme javljal le v izjemnih primerih.

Namesto javljanja izjem v pričakovanih primerih (npr. konec datoteke) se uporabi recimo različne statuse…

1.4.16 PRA.5.07: Ne javljaj izjem iz destruktorjev.

1.4.17 NAP.5.10: Izjeme naj se ustrezno beležijo.

1.4.18 NAP.5.11: Če je možno, naj se zagotovijo ustrezne metode s katerimi lahko uporabnik objekta preveri pred klicem metode ali bo prišlo do izjeme.

1.4.19 NAP.5.12: Po možnosti naj se uporabljajo v .NET Frameworku pre-definirane izjeme

Npr. `ArgumentException`, `ArgumentNullException`, `ApplicationException`,…

1.4.20 NAP.5.13: Javi vedno najbolj specifično izjemo.

S tem dobimo največ informacij o vzroku za izjemo.

1.4.21 NAP5.14-: Priporoča se uporaba enum za parametere, lastnosti,...

1.4.22 PRA.5.09: Ne mešaj različnih stilov programiranja v isti izvorni datoteki.