Konvencie pre zdrojové kódy v predmetoch Informatika 1 a Informatika 2

Obsah

1. Názvové konvencie

V jazyku Java sa využívajú v rôznych situáciách 3 rôzne názvové konvencie:

- Malá ťavia notácia – názov neobsahuje žiadne oddeľovače slov. Miesto toho každé slovo okrem prvého začína veľkým písmenom a pokračuje malými. Prvé slovo sa píše len malými písmenami.
  Príklady:

```
nazovAtributu
nazov
velmiDlhyNazovMetody
hlavaHrbHrb
```

- Veľká ťavia notácia – názov neobsahuje žiadne oddeľovače slov. Miesto toho každé slovo začína veľkým písmenom a pokračuje malými.
  Príklady:

```
NazovTriedy
SialeneSmutnaPrincezna
HlavaHrbHrb
```

- Veľká podčiarkovníková notácia – názov je písaný výlučne veľkými písmenami. Ako oddeľovač sa využíva znak podčiarkovník.
  Príklady:

```
KONSTANTA
KONSTANTA_S_DLHYM_NAZVOM
```

1.1. Triedy

Checkstyle modul

- TypeName (CLASS_DEF)

Chybové hlásenie

- Name 'XYZ' must match pattern '^[A-Z][a-zA-Z0-9]*$'.

Kde XYZ predstavuje názov triedy.

Popis chyby

Názov triedy XYZ nezodpovedá konvenciám. Názov pre triedu musí zodpovedať veľkej ťavej notácii. Znamená to, že vždy začína veľkým písmenom a pokračuje po koniec slova malými písmenami. Slová nie sú oddelené žiadnym oddeľovačom. Miesto toho vždy začínajú veľkým písmenom, aby bolo možné slová v názve ľahko rozoznať.

Príklad chyby

```
public class pouzivatelSystemu { // Názov musí začínať veľkým písmenom
public class pouzivatel_systemu {// Názov nesmie obsahovať žiadne
                                 // oddeľovače
```

Správny príklad

```
public class PouzivatelSystemu {
```

1.2. Metódy

Checkstyle modul

- MethodName

Chybové hlásenie

- Name 'XYZ' must match pattern '^[a-z][a-zA-Z0-9]*$'.

Kde XYZ predstavuje názov metódy.

Popis chyby

Názov metódy XYZ nezodpovedá malej ťavej notácii. Prvé písmeno názvu musí byť vždy malé a každé ďalšie slovo začína veľkým písmenom. Slová nie sú oddelené žiadnym oddeľovačom.

Príklad chyby

```
public void VykonajUlohu() {  // Názov metódy musí začínať malým
                              // písmenom
public void vykonaj_ulohu() { // V názve sú povolené len písmená a
                              // čísla
```

Správny príklad

```
public void vykonajUlohu() {
```

1.3. Atribúty

Checkstyle modul

- MemberName
- StaticVariableName

Chybové hlásenie

- Name 'XYZ' must match pattern '^[a-z][a-zA-Z0-9]*$'.

Kde XYZ predstavuje názov atribútu.

Popis chyby

Názvy atribútov inštancie a triedy musia zodpovedať malej ťavej notácii. Prvé písmeno názvu musí byť malé a každé ďalšie slovo začína veľkým písmenom. Názov môže obsahovať len písmená a číslice.

Príklad chyby

```
private static int POCETOBJEKTOV; // názov sa píše malými písmenami
private String Nazov;             // prvé slovo musí začínať malým
                                  // písmenom
```

Správny príklad

```
private static int pocetObjektov;
private String nazov;
```

1.4. Konštanty

Checkstyle modul

- ConstantName

Chybové hlásenie

- Name 'XYZ' must match pattern ' ^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$'.

Kde XYZ predstavuje názov konštanty.

Popis chyby

Názov konštanty nevyužíva ťaviu notáciu. Miesto toho sa konštanty píšu veľkými písmenami. Ako oddeľovač slov sa využíva znak podčiarkovník (_).

Príklad chyby

```
private static final int pocetZiakovVTriede = 5;
```

Správny príklad

```
private static final int POCET_ZIAKOV_V_TRIEDE = 5;
```

1.5. Parametre konštruktorov a metód

Checkstyle modul

- ParameterName

Chybové hlásenie

- Name 'XYZ' must match pattern '^[a-z][a-zA-Z0-9]*$'.

Kde XYZ predstavuje názov parametra.

Popis chyby

Názvy parametrov musia zodpovedať malej ťavej notácii. Názov začína malým písmenkom. Každé ďalšie slovo začína veľkým písmenkom. Názov neobsahuje žiadne znaky ako oddeľovače slov.

Príklad chyby

```
public Student(String Meno) { // názov parametra musí začínať malým
                              // písmenom
public int vypocitajUrok(double vyska_pozicky) { // názov parametra
                              // môže obsahovať len písmená a čísla
```

Správny príklad

```
public Student(String meno) {
public int vypocitajUrok(double vyskaPozicky) {
```

1.6. Lokálne premenné

Checkstyle modul

- LocalFinalVariableName
- LocalVariableName

Chybové hlásenie

- Name 'XYZ' must match pattern '^[a-z][a-zA-Z0-9]*$'.

Kde XYZ predstavuje názov lokálnej premennej.

Popis chyby

Názov lokálnej premennej musí dodržiavať malú ťaviu notáciu. Prvé slovo názvu musí začínať malým písmenkom a každé ďalšie slovo musí začínať veľkým písmenkom. Názov môže obsahovať len veľké/malé písmenká a čísla.

Príklad chyby

```
int vypocitany_rok;        // slová v názve lokálnej premennej sa
                           // neoddeľujú žiadnym oddeľovačom
```

Správny príklad

```
int vypocitanyRok;
```

1.7. Interface

Checkstyle modul

- TypeName (INTERFACE_DEF)

Chybové hlásenie

- Name 'XYZ' must match pattern '^I[A-Z][a-zA-Z0-9]*$'.

Kde XYZ predstavuje názov interface.

Popis chyby

Názov interface musí podľa štandardných konvencií zodpovedať veľkej ťavej notácii. Znamená to, že každé slovo začína veľkým písmenom a pokračuje malými. Pri práci v rámci predmetu používame rozšírenú konvenciu: prvým slovom názvu musí byť písmeno i, aby bolo zrejmé, že sa nejedná o triedu.

Príklad chyby

```
public interface Vykonatelny { // prvé slovo musí byť i
public interface iPredmet {    // názov musí dodržiavať veľkú ťaviu
                               // notáciu
```

Správny príklad

```
public interface IVykonatelny {
public interface IPredmet {
```

1.8. Balíčky

Checkstyle modul

- PackageName

Chybové hlásenie

- Name 'XYZ' must match pattern '^[a-z][a-zA-Z0-9]*([.][a-z][a-zA-Z0-9]*)*$'.

Kde XYZ je názov balíčka.

Popis chyby

Pri vymýšľaní názvu balíčka XYZ ste nedodržali názvové konvencie. Názov balíčka dodržiava malú ťaviu notáciu. Prvým znakom názvu balíčka je preto malé písmeno. Názov ďalej pokračuje malými písmenami alebo číslami. Každé ďalšie slovo začína veľkým písmenom. Vnorené balíčky sú oddeľované znakom bodka.

Unikátnosť názvu balíčka je zabezpečená dodržiavaním nasledovných konvencií:

- V nepomenovanom balíčku len najvyšší balíček.
- Najvyšší balíček – vlastné meno, prezývka, názov firmy, webová doména (viacstupňové vnáranie).
- Vnorený balíček – názov programu.
- Do neho vnorené balíčky – vlastné balíčky programu.

Príklad chyby

```
package WorldOfFRI;    // názov balíčka musí začínať malým písmenom
package world_of_fri; // názov balíčka môže pozostávať len z písmen
                      // a číslic
```

Správny príklad

```
package sk.uniza.fri.worldOfFRI.miestnosti;
```

1.9. Typové parametre

Checkstyle modul

- ClassTypeParameterName
- MethodTypeParameterName

Chybové hlásenie

- Name 'XYZ' must match pattern '^[A-Z]$'.

Kde XYZ predstavuje názov typového parametra.

Popis chyby

Podľa štandardných konvencií jazyka Java, musí byť typový parameter reprezentovaný jednopísmenkovým názvom, pričom sa musí jednať o veľké písmenko anglickej abecedy.

Odporúčané je využívať nasledujúce písmenká:

- E – typ položky kontajnera
- K – typ kľúča v Map
- V – typ hodnoty v Map
- N – číselný typ
- T – všeobecný typ
- S, U, V, X, Y, Z – ďalšie typové parametre

Príklad chyby

```
public class Zoznam<Polozka> { // Názov typového parametra môže byť
                               // len jednopísmenkový
```

Správny príklad

```
public class Zoznam<E> {
```

2. Prístup ku členom

2.1. Prístup k vlastným atribútom

Checkstyle modul

- RequireThis

Chybové hlásenie

- Reference to instance variable 'XYZ' needs 'this.'.

Kde 'XYZ' je atribút definovaný v rovnakej triede.

Popis chyby

Pri prístupe k atribútom je nutné písať explicitne kľúčové slovo this. Zjednodušuje to prehľad v zdrojových kódoch.

Príklad chyby

```
menoPriezvisko = "Ferko Mrkvička"; // menoPriezvisko je atribút.
```

Správny príklad

```
this.menoPriezvisko = "Ferko Mrkvička";
```

2.2. Posielanie správ samému sebe

Checkstyle modul

- RequireThis

Chybové hlásenie

- Method call to 'XYZ' needs 'this.'.

Kde 'XYZ' je metóda definovaná v rovnakej triede.

Popis chyby

Pri posielaní správy je nutné vždy písať adresáta správy. V prípade, že posielame správu rovnakému objektu, ako adresáta treba uvádzať this.

Príklad chyby

```
vysledok = vypocitaj(); // vypocitaj je metóda definovaná v rovnakej
                        // triede
```

Správny príklad

```
vysledok = this.vypocitaj();
```

2.3. Prístup k atribútom nadradenej inštancie

Checkstyle modul

- RequireThis

Chybové hlásenie

- Reference to instance variable 'XYZ' needs 'this.'.

Kde 'XYZ' je atribút definovaný v nadradenej triede.

Popis chyby

Pri prístupe k atribútu nadradenej triedy z metódy/konštruktora vnorenej triedy je tiež nutné uvádzať this. Keďže sa však nejedná o vlastný atribút, využíva jazyk Java špeciálnu syntax – NazovNadradenejTriedy.this.nazovAtributu.

Príklad chyby

```
public class Nadradena {
    private int pocet;

    private class Vnorena {
        Vnorena() {
            pocet++; // tiež je nutné uviesť this pri prístupe k
                     // atribútu
        }
    }
}
```

Správny príklad

```
public class Nadradena {
    private int pocet;

    private class Vnorena {
        Vnorena() {
            Nadradena.this.pocet++;
        }
    }
}
```

2.4. Posielanie správ nadradenej inštancie

Checkstyle modul

- RequireThis

Chybové hlásenie

- Method call to 'XYZ' needs 'this.'.

Kde 'XYZ' je metóda definovaná v nadradenej triede.

Popis chyby

Pri posielaní správy treba podľa konvencií vždy uvádzať adresáta. Pri posielaní správy inštancii nadradenej triedy sa v jazyku Java podľa štandardu používa špeciálna syntax NazovNadradenejTriedy.this.selektor(parametre).

Príklad chyby

```
public class Nadradena {
    private class Vnorena {
        private int hodnota;

        public Vnorena() {
            hodnota = vypocitajPredvolenu();
        }
    }

    public int vypocitajPredvolenu() {
        return 5;
    }
}
```

Správny príklad

```
public class Nadradena {
    private class Vnorena {
        private int hodnota;

        public Vnorena() {
            hodnota = Nadradena.this.vypocitajPredvolenu();
        }
```

```
    }

    public int vypocitajPredvolenu() {
        return 5;
    }
}
```

3. Používanie balíčkov

3.1. Import balíčka s hviezdičkou

Checkstyle modul

- AvoidStarImport

Chybové hlásenie

- Using the '.*' form of import should be avoided - XYZ.*.

Kde XYZ je importovaný balíček, resp. importovaná trieda.

Popis chyby

Štandard jazyka Java umožňuje importovať všetky triedy z jedného konkrétneho balíčka, alebo importovať všetky metódy triedy do globálneho menného priestoru. Aj keď táto možnosť vyzerá ako príjemná skratka a zjednodušenie písania, znižuje to čitateľnosť kódu a zvyšuje pravdepodobnosť výskytu chyby a je preto konvenciami zakázaná. Použite miesto toho samostatné príkazy import pre jednotlivé triedy /metódy.

Príklad chyby

```
import java.io.*;                 // importovanie všetkých tried
                                  // z balíčka je konvenciou zakázané
import static java.lang.Math.*; // importovanie všetkých metód triedy
                                  // je konvenciou zakázané
```

Správny príklad

```
import java.io.File;
import java.io.PrintWriter;

import static java.lang.Math.abs;
```

3.2. Zakázané balíčky

Checkstyle modul

- IllegalImport

Chybové hlásenie

- Import from illegal package - XYZ.

Kde XYZ je importovaný balíček.

Popis chyby

Knižnica jazyka Java šírená firmou Oracle obsahuje okrem balíčkov štandardnej knižnice aj mnohé balíčky, ktoré štandardné nie sú. Program, ktorý ich využíva nemusí fungovať správne na iných implementáciách jazyka Java (napr. vo veľkom využívaný balíček icedtea). Preto je ich využívanie zakázané konvenciou. Povolené sú len balíčky štandardnej knižnice, s názvom začínajúcim na java a javax.

Príklad chyby

```
import sun.io.Win32ErrorMode;
```

3.3. Nepoužívané importované triedy

Checkstyle modul

- UnusedImports

Chybové hlásenie

- Unused import - XYZ.

Kde XYZ je importovaný balíček.

Popis chyby

Niekedy sa počas programovania stane, že triedu importovanú z nejakého balíčka, ktorú sme doteraz využívali, už nepotrebujeme. Treba si na taký prípad dať pozor a nezabudnúť odstrániť príkaz import, ktorý je už tým pádom zbytočný.

4. Formátovanie zdrojového kódu

4.1. Odsadenie zdrojového kódu

Checkstyle modul

- Indentation

Chybové hlásenie

- 'XYZ' have incorrect indentation level A, expected level should be B.
- 'XYZ' child have incorrect indentation level A, expected level should be B.

Kde XYZ je štruktúrovaný alebo jednoduchý príkaz, A a B sú čísla.

Popis chyby

Odsadenie na danom riadku má nesprávnu veľkosť. Vo vašom kóde má A medzier, podľa logiky štruktúrovaného kódu má mať B medzier. Podľa konvencie má byť každé vnorenie odsadené o štyri medzery viac, ako odsadenie nadradeného bloku.

Správne odsadenie je veľmi dôležité, nakoľko zvyšuje čitateľnosť kódu.

Príklad chyby

```
   public class Trieda {           // odsadenie malo byť o 0 medzier
public void metoda() {             // odsadenie malo byť o 4 medzery
            System.out.println(5); // odsadenie malo byť o 8 medzier
}
}
```

Správny príklad

```
public class Trieda {
    public void metoda() {
        // prikaz
    }
}
```

4.2. Medzery

Checkstyle modul

- NoWhitespaceAfter
- NoWhitespaceBefore
- WhitespaceAfter
- WhitespaceAround

Chybové hlásenie

- 'A' is not preceded with whitespace.
- 'A' is not followed with whitespace.

Kde A je operátor.

Popis chyby

Môže sa to zdať ako zbytočné, ale súčasťou konvencií je aj správne umiestňovanie medzier v kóde. Samotný jazyk Java je v tomto veľmi benevolentný. Problém však je, že nesprávnym využívaním medzier sa znižuje čitateľnosť kódu.

Je preto nutné dodržiavať nasledovné pravidlá:

- Za unárnymi operátormi sa nesmie dávať medzera,
- za operátorom bodka sa nesmie dávať medzera,
- pred operátormi inkrementácie a dekrementácie sa nesmie dávať medzera,
- pred operátorom bodka sa nesmie dávať medzera – môže sa však v prípade potreby použiť zalomenie riadku,
- za čiarkou sa vždy dáva medzera,
- okolo binárnych operátor sa vždy dáva medzera a
- okolo operátora priradenia sa vždy dáva medzera.

Príklad chyby

```
a=5*a;              // okolo operátora = a operátora * musia byť
                    // medzery
b ++;               // pred operátorom ++ nesmú byť medzery
student . vypis(); // okolo operátora . nesmú byť medzery
```

Správny príklad

```
a = 5 * a;
b++;
student.vypis();
skola                                  // v prípade potreby je možné
    .dajStudenta("Ferko Mrkvička") // zalomiť riadok pred operátorom
    .dajIndex()                        // bodka
    .dajPredmet("Informatika 1")
    .nastavZnamku('A');
```

4.3. Definície premenných

Checkstyle modul

- MultipleVariableDeclarations

Chybové hlásenie

- Each variable declaration must be in its own statement.

Popis chyby

Štandard jazyka Java umožňuje definíciu viac premenných jedným príkazom. Takýto prístup znižuje čitateľnosť kódu a sťažuje proces ladenia. Je preto konvenciou zakázaný.

Príklad chyby

```
int pocetPoloziek, sucet = 3, cisloDomu; // v jednom príkaze má byť
                                         // len jedna definícia
                                         // premennej
```

Správny príklad

```
int pocetPoloziek;
int sucet = 3;
int cisloDomu;
```

4.4. Jeden príkaz na riadok

Checkstyle modul

- OneStatementPerLine

Chybové hlásenie

- Only one statement per line allowed.

Popis chyby

Pri písaní programu v jazyku Java máme pomerne veľkú voľnosť. Môžeme sa napr. rozhodnúť písať na jeden riadok viac príkazov oddelených bodkočiarkou. Je to však zlý nápad z pohľadu čitateľnosti a udržiavateľnosti kódu. Konvenciou preto je, písať len jeden príkaz na jeden riadok.

Príklad chyby

```
this.inicializuj(); this.spracuj(); // na jednom riadku má byť len
                                    // jeden príkaz
```

Správny príklad

```
this.inicializuj();
this.spracuj();
```

4.5. Vetva default

Checkstyle modul

- DefaultComesLast

Chybové hlásenie

- Default should be last label in the switch.

Popis chyby

Vetvy v príkaze switch môžu byť ľubovoľne poprehadzované. Konvenciou však je, aby vetva default bola uvedená ako posledná.

Príklad chyby

```
switch (cisloPozdravu) {
    case 1:
        System.out.println("Ahoj");
        break;
    default:                        // vetva default má byť posledná
        System.out.println("???");
        break;
    case 2:
        System.out.println("čau");
}
```

Správny príklad

```
switch (cisloPozdravu) {
    case 1:
        System.out.println("Ahoj");
        break;
    case 2:
        System.out.println("čau");
        break;
    default:
        System.out.println("???");
}
```

4.6. Nesprávny zápis poľa

Checkstyle modul

- ArrayTypeStyle

Chybové hlásenie

- Array brackets at illegal position.

Popis chyby

Z historických dôvodov sú v jazyku Java povolené dva zápisy definície premennej typu pole: s hranatými zátvorkami pred názvom premennej a s hranatými zátvorkami po názve premennej. Konvenciou je povolená len prvá možnosť.

Príklad chyby

```
int pole[]; // hranaté zátvorky na nesprávnej pozícii
```

Správny príklad

```
int[] pole;
```

5. Modifikátory prístupu

5.1. Poradie

Checkstyle modul

- ModifierOrder

Chybové hlásenie

- 'XYZ' modifier out of order with the JLS suggestions.

Kde XYZ je modifikátor.

Popis chyby

Jazyk Java je veľmi voľný v tom, v akom poradí je možné uvádzať modifikátory (public, private, static, final, ...) pri definícii triedy, enum, interface, atribútov, metód a konštruktorov. Nesprávne uvedené poradie však znižuje čitateľnosť kódu. Bola preto v štandarde jazyka Java zavedená konvencia, ktorá určuje všeobecné poradie modifikátorov, ktoré sa má dodržiavať:

1. public, protected, alebo private,
2. abstract,
3. static,
4. final,
5. transient,
6. volatile,
7. synchronized,

8. native a
9. strictfp.

Príklad chyby

```
static public void nastav(String meno) { // modifikátor public má byť
                                         // pred modifikátorom static
public final static int PI = 3.14;       // modifikátor static má byť
                                         // pred modifikátorom final
```

Správny príklad

```
public static void nastav(String meno) {
public static final int PI = 3.14;
```

5.2. Zbytočné modifikátory

Checkstyle modul

- RedundantModifier

Chybové hlásenie

- Redundant 'XYZ' modifier.

kde XYZ je modifikátor.

Popis chyby

V niektorých prípadoch je uvádzanie modifikátorov (public, static, final, ...) v jazyku Java nepovinné. Je to vtedy, keď existuje jediný modifikátor, ktorý je možné na danom mieste použiť. Jedná sa o nasledujúce prípady:

- uvádzanie modifikátora public v definícii správy v interface,
- uvádzanie modifikátora final v definícii metódy vo final triede a
- uvádzanie modifikátora static v definícii vnoreného interface.

Na predmetoch Informatika 1/2 sa stretneme len s prvou možnosťou.

Keďže uvádzanie týchto modifikátorov je nepovinné a program sa bude správať rovnako aj v prípade ich neuvedenia, konvenciou je zakázané ich uvádzanie.

Príklad chyby

```
public interface INazvany {
    public String dajNazov(); // pri definícii správy v interface sa
                              // public neuvádza
}
```

Správny príklad

```
public interface INazvany {
    String dajNazov();
}
```

6. Bloky

6.1. Zbytočná definícia blokov

Checkstyle modul

- AvoidNestedBlocks

Chybové hlásenie

- Avoid nested blocks.

Popis chyby

Príkaz bloku (sekvencia príkazov v zložených zátvorkách) je možné použiť v jazyku Java na ľubovoľnom mieste v tele konštruktora, alebo metódy. Toto pravidlo však často vedie k chybám a znižuje prehľadnosť kódu. Využívanie príkazu bloku mimo štruktúrovaných príkazov je preto konvenciou zakázané.

Príklad chyby

```
if (hodnota > 5); {                // pozor na bodkočiarku
    System.out.println("správne!");
}
{                                  // blok mimo štruktúrovaného príkazu
    this.doIt();
}
```

Správny príklad

```
if (hodnota > 5) {
    System.out.println("správne!");
}
this.doIt();
```

6.2. Prázdny blokový príkaz

Checkstyle modul

- EmptyBlock

Chybové hlásenie

- Must have at least one statement.

Popis chyby

Každý príkaz bloku (sekvencia príkazov v zložených zátvorkách) musí obsahovať aspoň jeden príkaz. Inak je zbytočný a treba ho zo zdrojových kódov odstrániť (vrátane celého prípadného štruktúrovaného príkazu).

Príklad chyby

```
if (this.skusVykonatOperaciu()) { // ak je výsledok operácie true, nič
}                                 // sa nevykoná.

if (podariloSa) {                 // ak je true, nič sa nestane
```

```
} else {
    System.out.println("Sorry, ale takto nie!");
}
```

Správny príklad

```
this.skusVykonatOperaciu();

if (!podariloSa) {
    System.out.println("Sorry, ale takto nie!");
}
```

6.3. Chýbajúci blok

Checkstyle modul

- NeedBraces

Chybové hlásenie

- 'XYZ' construct must use '{}'s.

kde 'XYZ' je štruktúrovaný príkaz.

Popis chyby

Podľa definície jazyka Java môže štruktúrovaný príkaz obsahovať ľubovoľný príkaz ako svoje telo. Konvenciou však je, ako telo vždy použiť príkaz bloku, tj. uviesť celé telo v zložených zátvorkách. Predídeme tým rôznym chybám, ktoré môžu nastať z nepozornosti pri budúcej úprave kódu. Zvýši sa tým aj prehľadnosť a čitateľnosť kódu.

Príklad chyby

```
if (hodnota > 5); {                 // pozor na bodkočiarku
    System.out.println("správne!");
}

if (hodnota > 5)                    // telo musí byť v bloku
    System.out.println("správne!");
```

Správny príklad

```
if (hodnota > 5) {
    System.out.println("správne!");
}
```

6.4. Pozície zložených zátvoriek

Checkstyle modul

- LeftCurly
- RightCurly

Chybové hlásenie

- '{' at column A should be at previous line.

- '}' at column A should have line break before.
- '{' at column A should have line break after.
- '}' at column A should be alone on a line.
- '}' at column A should be on the same line.

Popis chyby

Umiestnenie zložených zátvoriek príkazu bloku je dôležitou súčasťou konvencií. Správne umiestňovanie otváracej zátvorky je vždy na koniec riadku nadradeného štruktúrovaného príkazu. Uzatváracia zátvorka sa zase píše vždy na samostatný riadok. Jedinou výnimkou je príkaz else, ktorý sa píše na rovnaký riadok, ako zatváracia zložená zátvorka príkazu if.

Príklad chyby

```
if (chyba)
{                                   // Otváracia zložená zátvorka musí
                                    // byť na konci predchádzajúceho
                                    // riadku.
    vysledok = 1; }                 // Zatváracia zožená zátvorka musí
                                    // byť na samostatnom riadku.

if (this.pocet > 5) { vysledok = 3;// Otváracia zložená zátvorka musí
                                    // byť na konci riadku
}
else {                              // Príkaz else musí byť na rovnakom
                                    // riadku ako zatváracia zložená
                                    // zátvorka príkazu if.
    vysledok = 8;
} return vysledok;                  // Zatváracia zložená zátvorka musí
                                    // byť jediný príkaz na riadku.
```

Správny príklad

```
if (chyba) {
    vysledok = 1;
}

if (this.pocet > 5) {
    vysledok = 3;
} else {
    vysledok = 8;
}

return vysledok;
```

7. Kvalita kódu

7.1. Prázdny príkaz

Checkstyle modul

- EmptyStatement

Chybové hlásenie

- Empty statement.

Popis chyby

Prázdny príkaz predstavuje perličku programovacieho jazyka Java, ktorá nemá rozumné využitie, nakoľko znižuje čitateľnosť kódu. Začiatočníkom však niekedy spôsobuje značné problémy, keď ho nevedomky použijú. Prázdny príkaz je predstavovaný jednoduchou bodkočiarkou bez jej priradenia ľubovoľnému neštruktúrovanému príkazu. Konvenciou je prázdne príkazy nepoužívať.

Príklad chyby

```
if (hodnota > 5); {                    // pozor na bodkočiarku
    System.out.println("správne!");; // prázdny príkaz na konci riadku
}
```

Správny príklad

```
if (hodnota > 5) {
    System.out.println("správne!");
}
```

7.2. Zjednodušiteľný výraz

Checkstyle modul

- SimplifyBooleanExpression

Chybové hlásenie

- Expression can be simplified.

Popis chyby

Častým zdrojom chýb začiatočníkov v programovaní býva nadužívanie literálov typu boolean. Príkladom môže byť porovnávanie neznámej hodnoty a boolean literálu na rovnosť. Výrazy „x == true“ a „x“ sú totožné, nakoľko ich hodnota bude true v rovnakých prípadoch. Použitie zložitejšej verzie však môže spôsobiť chybu, ak programátor omylom zabudne jeden znak rovná sa a napíše „x = true“, čo už je príkaz priradenia. Preferovaná a konvenciami vyžadovaná verzia je preto tá jednoduchšia.

Príklad chyby

```
if (jeJedna == true) {  // výraz sa dá napísať jednoduchšie
    return 1;
}
if (jeNieco == false) { // výraz sa dá napísať jednoduchšie
```

```
    return 0;
}
```

Správny príklad

```
if (jeJedna) {
    return 1;
}
if (!jeNieco) {
    return 0;
}
```

7.3. Zjednodušiteľný príkaz návratu

Checkstyle modul

- SimplifyBooleanReturn

Chybové hlásenie

- Conditional logic can be removed.

Popis chyby

Príkaz návratu môže obsahovať ľubovoľný, aj komplikovanejší, výraz. Programátori na to často zabúdajú, hlavne v metódach s typom návratovej hodnoty boolean.

Príklad chyby

```
public boolean jeIchDost() {
    if (this.pocet > this.pozadovanyPocet) { // dá sa zjednodušiť
        return true;
    } else {
        return false;
    }
}
```

Správny príklad

```
public boolean jeIchDost() {
    return this.pocet > this.pozadovanyPocet;
}
```

8. Časté chyby

8.1. Premenná skrývajúca atribút

Checkstyle modul

- HiddenField

Chybové hlásenie

- 'XYZ' hides a field.

kde XYZ je lokálna premenná.

Popis chyby

Prekladač jazyka Java nevyhodnotí ako chybu, ak nazvete rovnako lokálnu premennú, ako atribút. Takýto prístup znižuje čitateľnosť zdrojových kódov a zvyšuje pravdepodobnosť chyby. Je preto konvenciou zakázaný. Je však povolené mať rovnako nazvaný atribút a parameter metódy/konštruktora.

Príklad chyby

```
private int pocet;

public void spocitaj() {
    int pocet = 5;          // lokalna premenna nesmie byt nazvana
                            // rovnako ako atribut
    ...
}
```

Správny príklad

```
private int pocetPrvkov;

public void spocitaj() {
    int pocetSpravnych = 5;
    ...
}

public void nastavPocetPrvkov(int pocetPrvkov) { // parameter moze byt
                                                 // nazvany rovnako
                                                 // ako atribut
    this.pocetPrvkov = pocetPrvkov;
}
```

8.2. Priraďovací výraz

Checkstyle modul

- InnerAssignment

Chybové hlásenie

- Inner assignments should be avoided.

Popis chyby

Jazyk Java umožňuje využiť priraďovací príkaz aj vo forme výrazu. Dôsledok je, že priradenie hodnoty premennej môže byť „skryté“ vo vnútri iného príkazu, ako napr. príkazu na poslanie správy. Toto znižuje čitateľnosť kódu a sťažuje proces ladenia. Konvenciou je preto takáto možnosť zakázaná.

Príklad chyby

```
if (null != (vysledok = this.vyhladaj())) { // priradenie v príkaze if
    return vysledok;
}
return vysledok = this.vytvor();            // priradenie v príkaze
```

```
                                                    // návratu
```

Správny príklad

```
vysledok = this.vyhladaj();
if (vysledok == null) {
    vysledok = this.vytvor();
}
return vysledok;
```

8.3. Porovnávanie reťazcov

Checkstyle modul

- StringLiteralEquality

Chybové hlásenie

- Literal Strings should be compared using equals(), not 'A'.

Kde 'A' je buď operátor ==, alebo operátor !=.

Popis chyby

Častou chybou začínajúcich programátorov v jazyku Java je porovnávanie reťazcov pomocou operátorov == a !=. Toto porovnanie porovnáva iba referencie. Vzhľadom na niektoré ďalšie vlastnosti jazyka sa môže použitie týchto operátorov zdať ako funkčné. Takmer vždy sa však dá nájsť príklad, kedy porovnanie nebude fungovať správne. Treba preto používať výhradne porovnanie reťazcov pomocou správy equals.

Príklad chyby

```
if (odpoved != "yes") {
    return;
}
```

Správny príklad

```
if (odpoved.equals("yes")) {
    return;
}
```

8.4. Neukončený príkaz case

Checkstyle modul

- FallThrough

Chybové hlásenie

- Fall through from previous branch of the switch statement.

Popis chyby

Jazyk Java umožňuje vykonávať postupne viac vetiev v príkaze switch. Ak vetvu neuzavrie programátor príkazom break, alebo príkazom return, bude vykonávanie pokračovať nasledujúcou vetvou. V drvivej

väčšine prípadov sa jedná o nežiaduce chovanie a je zdrojom chýb. Konvencia preto vyžaduje, aby boli miesta, kde toto správanie programátor očakáva, označené komentárom „fallthrough“.

Príklad chyby

```
switch (hodnota) {
    case 1:
        System.out.println("prvý");
    case 2:                             // vetva 2 sa vykoná aj
                                        // v prípade, že hodnota == 1
        System.out.println("druhý");
}
```

Správny príklad

```
switch (hodnota) {
    case 1:
        System.out.println("prvý");
        break;
    case 2:
        System.out.println("druhý");
        break;
}

switch (hodnota) {
    case 1:
        System.out.println("prvý");
        return;
    case 2:
        System.out.println("druhý");
        return;
}

switch (hodnota) {
    case 1:
        System.out.println("prvý");
        /* fallthrough */
    case 2:
        System.out.println("druhý");
}
```

8.5. Zneužitie konštrukcie Interface

Checkstyle modul

- InterfaceIsType

Chybové hlásenie

- Interfaces should describe a type and hence have methods.

Popis chyby

Špecifikom konštrukcie interface v jazyku Java oproti iným jazykom je, že umožňuje vo svojom tele definovať konštanty. Vedie to k častému zneužívaniu interface na definíciu konštánt, ktoré je potrebné mať k dispozícii vo veľkej časti programu.

Správnym spôsobom definície konštánt je ich zavedenie v rámci tela triedy, ktorej sa konštanty týkajú, alebo ešte lepšie je využiť enum.

Príklad chyby

```
public interface IKonstanty {
    int OBDLZNIK_VERTIKALNY = 1;
    int OBDLZNIK_HORIZONTALNY = 2;
}
```

Správny príklad

```
public enum TypObdlznika {
    VERTIKALNY, HORIZONTALNY
}

public class Obdlznik {
    public static final int TYP_VERTIKALNY = 1;
    public static final int TYP_HORIZONTALNY = 2;

    ...
}
```

8.6. Porušenie ukrývania informácií

Checkstyle modul

- VisibilityModifier

Chybové hlásenie

- Variable 'XYZ' must be private and have accessor methods.

Kde 'XYZ' je atribút.

Popis chyby

Vzhľadom na komplexnosť problematiky, nedokáže systém checkstyle automaticky kontrolovať dodržiavanie zapuzdrenia. Pomáha však aspoň pri kontrole dodržiavania princípu ukrývania informácií. Každý atribút musí byť označený ako private. V prípade potreby je možné k nemu dorobiť verejnú prístupovú/zmenovú metódu. Konštanty môžu byť public.

Príklad chyby

```
public int pocet; // atribút musí byť private
```

Správny príklad

```
private int pocet;
```

```
public static final int MAXIMALNY_POCET = 20; // konštanty môžu byť
                                              // public
```

8.7. Zmena riadiacej premennej cyklu

Checkstyle modul

- ModifiedControlVariable

Chybové hlásenie

- Control variable 'XYZ' is modified.

Kde 'XYZ' je riadiaca premenná cyklu for, alebo iteračná premenná cyklu for-each.

Popis chyby

Riadiaca premenná cyklu for a iteračná premenná cyklu for-each sú štandardnými lokálnymi premennými. Iba sú využité v špecifických situáciách. Znamená to, že v celom tele cyklu je možné túto premennú modifikovať. Tým sa však program stáva neprehľadným. Jedná sa preto o konvenciou zakázanú operáciu.

Príklad chyby

```
for (int index = 0; index < 5; index++) {
    while (pole[index].jeNaPreskocenie()) {
        index++;                            // modifikácia riadiacej
                                            // premennej cyklu for
    }
    pole[index].spracuj();
}
for (Clovek clovek : this.ludia) {
    if (clovek == null) {
        clovek = new Clovek("«neznámy»");   // modifikácia iteračnej
                                            // premennej cyklu for-each
    }
    clovek.vypis();
}
```

Správny príklad

```
for (int index = 0; index < 5; index++) {
    if (!pole[index].jeNaPreskocenie()) {
        pole[index].spracuj();
    }
}
for (Clovek clovek : this.ludia) {
    if (clovek == null) {
        System.out.println("«neznámy»");
    } else {
        clovek.vypis();
    }
}
```

8.8. Výnimka ako meniteľný objekt

Checkstyle modul

- MutableException

Chybové hlásenie

- The field 'XYZ' must be declared final.

Kde 'XYZ' je atribút výnimky.

Popis chyby

Objekt výnimky predstavuje informáciu o chybovom stave. Čo sa nepodarilo, prečo nastala chyba a prečo nastala. Tieto informácie sú k dispozícii v čase nastania chyby a nemá zmysel ich dodatočne upravovať. Objekt výnimky by preto mal byť implementovaný ako nemeniteľný objekt. Všetky atribúty by teda mali byť označené final.

Príklad chyby

```
public class NeznamyClovekException {
    private String meno;                              // atribút musí byť
                                                      // final
    public NeznamyClovekException(String meno) {
        this.meno = meno;
    }

    public String getMeno() {
        return this.meno;
    }

    public void setMeno(String meno) {                // výnimka musí byť
                                                      // nemeniteľná
                                                      // - nesmie obsahovať
                                                      // zmenové metódy
        this.meno = meno;
    }
}
```

Správny príklad

```
public class NeznamyClovekException {
    private final String meno;

    public NeznamyClovekException(String meno) {
        this.meno = meno;
    }

    public String getMeno() {
        return this.meno;
    }
}
```