Laboration 1 – Java Code Conventions

Författad av: *Gabriel Afram, Thomas Astner, Alex Darborg, Andreas Edin, Adam Hjernquist, Joakim Sundqvist* och *Mattias Vängman*

Inledning

Kod konventioner är bra att kunna eftersom det underlättar en programmerare att förstå andras kod då det blir mer lättläst. Java använder två filtyper, java source samt java bytecode. Suffixen för dessa filtyper är .java och .class. Rekommenderade namn för att bygga makefiler är GNU make för att bygga vårt program. En textfil med README är också bra att ha med för att ge instruktioner till nya projekt arbetare. Filer som är större än 2000 rader anses vara besvärliga och skall undvikas.

Java Source Files

Varje java source fil innehåller antingen en singel publik klass eller ett interface. När privata klasser och interfaces är associerade med en publik klass så kan man lägga dem i samma source fil som den publika klassen ligger i.

Java source file har följande ordning:

- Kommentarer
- package java.awt;

Paketen kan se ut såhär:

```
import java.applet.Applet;
import java.awt.*;
import java.net.*;
```

- Klass och interface deklarationer

Klass & Interface struktur:

- Klass eller interface dokumentation kommentarer.
- Klass eller interface påstående.
- Klass eller interface implementations kommentarer.
- Klass (static) variabler, publika först sedan skyddade och sist privata variabler.
- Instans variabler, publika först sedan skyddade och sist privata variabler.
- Konstruktorer.
- Metoder.

När klassvariabler deklareras, börjar man med dem publika variablerna och sedan dem skyddade och sist dem privata.

Indentering/Indrag

Fyra mellanslag borde användas som enhets mått för ett indrag. Undvik rader som är längre än 80 karaktärer då det kan ställa till problem för terminaler och verktyg. Om du får längre än 80 karaktärer bör du bryta upp den i mindre bitar.

Grundprinciper för att bryta upp en rad:

- Bryt efter ett kommatecken
- Bryt före en operator
- Föredrar higher-level breaks före lower-level breaks
- Justera den nya raden precis under uttrycket från den föregående raden.
- Om ovanstående grundprinciper leder till förvirrande kod, då underlättas det med 8 mellanslag.

Ett Java program kan ha två olika typer av kommentarer: implementationskommentarer och dokumentationskommentarer.

/*....*/ eller // använder man för implementations kommentarer medans /**...*/ används för dokumentations kommentarer.

Dokumentationskommentarer som kallas "doc comments" finns bara i Java. Implementationskommentarer är menade till att kommentera koden eller för att kommentera en specifik implementation. Dokumentskommentarer är menade att beskriva specifikationen av koden från ett implementations-fritt perspektiv, det ska kunna läsas av utvecklare som inte nödvändigtvis har koden framför sig. Kommentarer bör användas till att ge en översikt av koden och bara innehålla information som är relevant till att läsa och förstå programmet. Undvik att kommentera för mycket, saker som är uppenbara i koden behöver inte kommenteras, det gör bara att koden blir svårläst, försök hellre att göra koden mer lättläst än att lägga in kommentarer till allt. Det finns fyra olika stilar av implementations kommentarer som ett program kan ha: block, single-line, trailing och end-of-line.

Block kommentarer: Används för att ge en beskrivning av filer, metoder, data strukturer och algoritmer. De borde användas i början av varje fil och förre varje metod, de kan även användas inne i en metod men den ska då vara indenterad på samma nivå som koden den beskriver. Exempel på en block kommentar:

```
/*
 * Här är en block kommentar
 */
```

Single-line kommentarer: Kan förekomma på en enkel rad som är indenterad till samma nivå som koden som följer. Om inte kommentaren kan skrivas på samma rad borde den följa block kommentar formatet. Exempel på en single-line kommentar:

```
If (condition) {
        /* Handle the condition. */
        ....
}
```

Trailing kommentarer: Väldigt korta kommentarer som förekommer på samma rad som koden den beskriver ska hålla ett tillräckligt långt avstånd till koden. Är det fler korta kommentarer i samma stycke av kod så ska de vara indenterade till samma nivå. Exempel på en trailing kommentar:

```
if(a == 2) {
        return true;                    /* special case */
} else {
        return isprime(a); /* works only for odd a */
}
```

End-OF-Line kommentarer: När man använder sig av // så kommer kommentaren att fortsätta enda tills det blir en nyrad. Exempel på End-Of-Line kommentarer:

```
if(foo >1) {
        // Do a double-flip
        ...
} else {
        return false;        //Explain why here
}
```

Dokumentations kommentarer: beskriver Java klasser, gränssnitt, konstruktorer, metoder och fält. Varje dokumentations kommentar borde förekomma precis innan deklarationen. Exempel:

```
/**
* The Example class provides ..
*/
class Example { ...
```

Notera att klasser och gränssnitt inte är indenterade medan deras medlemmar är det. Första raden (/**) för klasser och gränssnitt är inte indenterad, raderna nedanför är indenterade så det hamnar i linje med *-tecknet. Medlemmar, inkluderat konstruktorer har fyra blanksteg på första raden och fem på de andra. En dokumentationskommentar borde inte vara positionerad inne i en metod eller konstruktor, för Java associerar dokumentations kommentarer med den första deklarationen efter kommentaren.

En deklaration per rad rekommenderas då det fungerar bättre om man vill göra en kommentar för en specifik deklaration. Aldrig ska både en variabel och en metod deklareras på samma rad. Inte heller bör olika typer av variabler deklareras på samma rad. Det går bra att använda indentering mellan variabeltyp, variabelnamn och kommentar. Se exempel nedan:

```
int           level;        // indentation level
int           size;         // size of table
Object        currentEntry;              // currently selected table entry
```

Lägg alla deklarationer i början av ett block. Vänta inte med deklarationen tills den ska användas. Detta kan försvåra läsbarheten hos en annan användare. Det enda undantaget är i “for”-iterationer, där räknaren kan deklareras i metoden. Undvik att deklarera en ny variabel med samma namn i ett inre block. Om det är möjligt bör variabler initieras direkt vid deklarationen. Det vill säga ge den ett värde när den skapas. Vid kodning av klasser och interface bör det inte vara något mellanrum mellan metodnamn och första parentesen “(”, första måsvingen förekommer på samma rad och sista måsvingen som avslutar metoden på en egen rad. Metoder separeras med ett blankt radbyte.

Varje rad bör innehålla max ett påstående. Varje returvärde för en metod bör vara parantesfri.

Alla ”if”, ”else if” och ”else” -satser bör ha denna form:

```
if (condition) {
        statements;
} else if (condition) {
        statements;
} else {
        statements;
}
```

Alla ”if”-satser ska ha måsvingar, även om den bara innehåller ett påstående.

En for-loop skall innehålla initiering, tillstånd, uppdatering och påstående/påståenden. En tom for-loop bör innehålla initiering, tillstånd och uppdatering. Vid användning av kommaoperatorn i initiering eller uppdatering av en klausul undvik att använda mer än tre variabler. om det behövs använd separata påståenden innan for-loopen påbörjats. En while-loop skall innehålla tillstånd och påstående en while-loop deklareras på följande sätt:

```
while (condition)  {
        statements;
}
```

Vid användning av while-loop utan statement deklareras den på detta sätt.

```
while (condition);
```

Do-while loop är lik en while loop enbart att den är omvänd och statementet kommer innan deklarationen av while-loopen. Do-while loopen deklareras på följade sätt.

```
do {
        statements;
} while (condition);
```

Implementation av switch statements ser ut på följande sätt

```
switch (condition) {
case ABC:
        statements;
        /* falls through */
case DEF:
        statements;
        break;
case XYZ:
        statements;
        break;
default:
        statements;
        break;
}
```

Varje switch statement bör innehålla ett default case. Varje case som inte innehåller en break skall innehålla en kommentar (/*Falls through*/) för att visa att den faller igenom. Se exempel ovan.

try-catch statement deklareras på följande sätt.

```
try {
        statements;
} catch (ExceptionClass e) {
        statements;
}
```

Blankrader förbättrar läsbarheten av koden, det blir som styckesindelning av koden. Man delar upp koden i stycken för den kod som är relaterad till varandra. Två blankrader skall alltid användas under följande omständigheter, mellan delar av en källfil och mellan klass och gränssnittsdefinitioner. En blankrad bör alltid användas under följande omständigheter, mellan metoder, mellan lokala variabler i en metod och metodens första statement. Samt för en kommentar innan ett block eller på ensam rad. En blankrad bör implementeras mellan logiska sektioner inuti en metod för att förbättra läsbarheten. Ett nyckelord följt av en parentes bör separeras med ett mellanslag. Ett mellanslag bör ej användas mellan en metods namn och dess öppningsparetens. Detta bidrar till att särskilja sökord från metodanrop. Ett mellanslag bör förekomma efter ett komma i argumentlistor. Alla binära operatorer utom operatorn (.) bör skiljas från sina operander med ett mellanslag. Unära aktörer som increment och decrement bör ej skiljas från dess operand.

Konventioner för namngivning

För att göra programkod lätt att läsa och förstå finns det konventioner (överenskommelser) som bör följas. Om detta görs blir det lätt för läsaren av koden att avgöra om en identifierare är en variabel, klass eller en metod t.ex. Nedan följer en kortfattad beskrivning av hur konventionen för respektive identifierare ser ut.

Klass: Namnet utgörs av ett substantiv. Namnet börjar alltid med stor bokstav. Om namnet är sammansatt av flera ord börjar varje delord med stor bokstav. Inga blanksteg används. Några exempel:

```
class Hus

class SommarStuga
```

Interface: Namnges på samma sätt som klasser.

Metod: Namnet utgörs av ett verb. Om namnet är sammansatt av flera ord är första ordet ett verb. Namnet inleds med små bokstäver, sedan inleds varje delord i namnet med stor bokstav. Några exempel:

```
rulla();

getNumberOfRooms();
```

Variabel: Namnet inleds med liten bokstav. Om namnet består av flera ord inleds varje delord med stor bokstav. Variabelnamn bör vara korta och meningsfulla. Namnet bör vara utformat så att det är lätt att komma ihåg. I allmänhet undviks variabelnamn som består av endast en bokstav, förutom för kortlivade variabler, t.ex. i en for-loop. Bokstäverna i, j, k, m och n betecknar vanligtvis heltal och d, d, e betecknar vanligtvis tecken (characters). Några exempel:

```
int i;

string firstName;
```

Konstant: Namnet skrivs med stora bokstäver. Om namnet består av flera ord separeras de av understreck. Ett exempel:

```
Int MAX_ANTAL_RUM = 8;
```

Tillgänglighet och referens till variabler

Variabler och instanser bör inte göras publika om det inte särskilt krävs.

När man refererar till ett objekt är det viktigt att skilja på klassvariabler (statiska) och på instansierade objekt. Ex:

```
klassMetod();                          //OK

EnKlass.klassMetod();                  //OK

ettObjekt.klassMetod();                //EJ OK!
```

Tilldelning av variabler

Undvik att tilldela flera variabler samma värde i ett kommando, det blir svårläsligt. Ex:

```
firstName = secondName = ”no name”;  // FEL
```

Även inbäddade tilldelningar bör undvikas, ex:

```
a = ((b = c + d)-e);                   // FEL
```

bör skrivas:

```
b = c + d;

a = b – e;                             // RÄTT
```