Mustafa Hakimi 14/02 2017

Static Test Techniques Exercise

1. Static Code Analysis of Triangle Program

- Jeg har haft problemer med at anvende Metric Software/Plugins til Netbeans IDE. Kenai SQE's Installations guide var "non-existent". Fik dog installereret JaCoCoverage, men jeg ved ikke hvordan dette plugin benyttes. Jeg har også prøvet oversætte min kode til C# for at kunne benytte Visual Studio's Analytiske værktøjer. Visual Studios's Analytiske værktøjer er ikke tilgængelig på Visual Studio Preview til Mac. Hvis de analytiske værktøjer virkede, vil jeg mene at min Cyclomatic Complexity for programmet ville ligge mellem 7-9, da de fleste værktøjer tager +1 for if-else statements.
  "*Cyclomatic Complexity = Number of decision points + 1 The decision points may be your conditional statements like if, if… else, for loop, while loop etc.*"

2. Peer Review of your Triangle program

Review af Kevin Mert Turan's Triangle Program - https://github.com/KevinMertTuran/TestAflevering

- Du har deklareret tre static integer attributter hver for sig. Nu hvor de tre attributter er af den samme data type (integer), så kunne du forkorte din kode, ved at deklarere dem sammen, eksempelvis - *static int a, b, c.* Derudover har du sat dine tre attributter lig med 0, hvilket er unødvendigt i denne sammenhæng. Du har hellere ikke benyttet "private" til dine attributter, derved følger du ikke "Encapsulation", hvilket er et Objekt Orienteret programmerings koncept. Ved at bruge "private" indkapsler du adgang til dine objekt variabler og tillader ikke eksterne objekter at ændre status på dine objekter.

- Du beder brugeren om at indtaste en Integer, og bruger derefter Scanner til at læse inputtet. Hvad sker der hvis brugeren indtaster et bogstav? Du har ikke inkluderet Exception handling, så dit program vil højst sandsynligt give et "InputMismatchException". Du bør derfor benytte en "try catch" eller en "do while" for at kunne håndtere Exceptions. Du håndtere hellere ikke negative og 0 input fra brugeren, eksempelvis 0, -1,-2 osv. Din else if condition på linje 41 er unødvendig, da trekanten i dette tilfælde ville blive en Isosceles (ligebenet) lige meget hvad.

3. Coding standard document

- DRY principle (Don't Repeat Yourself). De fleste applikationer har til formål at automatisere gentagne opgaver. Denne princip bør opretholdes i al kode. Det samme stykke kode bør ikke gentages igen og igen.

- Organisering af mapper, klasser, filer osv. Teknisk set kunne man skrive programmet i en enkelt fil, men det er ikke optimalt at mange grunde, bl.a. vedligeholdelse, læsbarhed osv. Det er derfor vigtigt at man strukturere applikationen i passende mapper. Nogle frameworks har en struktureret mappestruktur, eksempelvis MVC.
- Naming Conventions.
    - Klassenavn, stort startbogstav og resten med små, dog stort startbogstav ved sammensat ord. Eksempelvis mainMenu.java, animalDog.java.
    - Konstantnavn, store bogstaver
    - Metodenavne, små bogstaver, dog stort startbogstav ved sammensat ord. Eksempelvis getInfo(), calculateFibonacci().
    - Betydningsfulde navne. Hvis man bruger andre navngivnings konventioner, er det vigtigt at man er konsistent igennem hele applikationen.
- Korrekte Datatyper. Det er vigtigt at man benytter de rigtige datatyper til variabler. Skal man bruge Float eller Double?
- Undgå for meget nesting. Alt for mange niveauer kan gøre koden svær at læse og vedligeholde.
- Korrekt anvendelse af OOP koncepter, eksempelvis Inheritance, Polymorphism, Abstraction og Encapsulation.
- Refactoring, man kan altid forbedre sin kode. Det er en form for rengøring, for at forbedre kvalitet, vedligeholdelse og læsbarhed.
- Kommentere programmets komplekse dele. Evt. Dokumentere komplekse komponenter. Det er ligeledes vigtigt at man undgår åbenlyse- og redundante kommentar.
- Håndtering af Exceptions. Anvende ”try catch”, ”Throw” eller andre former for Exception handling.
- Kosmetiske standarder, såsom at benytte ”Indentation” hvor det er nødvendigt. Der er flere forskellige måder man kan bruge Indentation på, det er derfor vigtigt at man i et team bliver enige om hvilken type man benytter og er konsistent igennem hele applikationen. Det er også vigtigt at man gruppere sin kode, ved at separere med passende afstand. Derudover bør man også tage højde for linje længde. Det er en god ide at undgå lange horisontale linjer kode.