Laboration 2 Module 1dv610

Syfte

Målet med laboration 2 är att ni skall börja skifta fokus från att bara “lösa ett problem” till att också “kommunicera er lösning som kod”. Att effektivt kommunicera sin lösning är viktigt då det underlättar både lösandet av problemet och när ni skall arbeta tillsammans med andra.

Ni kommer i laborationen att skriva kod som ska återanvändas av dig själv och andra programmerare. Koden skall fungera som övning för att arbeta med sin kodkvalitet och som ett reflektionsunderlag för att arbeta med innehållet i kursboken.

Under laborationen kommer ni arbeta med kapitel 2 och 3 i boken Clean Code.

En “modul”

Ni har tidigare skrivit appar som driftsatts för att användas av slutanvändare. Då är kvalitet i funktionalitet viktigt, att appen fungerar som den skall.

I den här laborationen skall ni istället skriva en kod som skall användas av andra programmerare, en “modul”, då är kvalitet i själva koden viktigt.

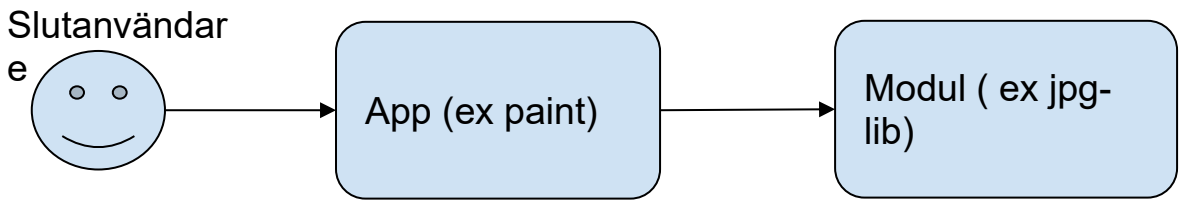
Moduler kan ibland kallas bibliotek (*Eng. libraries*), alltså kod som används för att skriva appar men i sig inte är en egen app. Ni har tidigare importerat bibliotek för att få tillgång till funktioner och klasser som löst olika problem åt er.

Exempel: I laborationen skulle vi kunna skapa ett bibliotek som kan läsa in jpg-bilder. Vi kallar det ‘jpeg-lib’. ‘jpeg-lib’ avkodar informationen i den inlästa bilden så att den som använder biblioteket får tag i RGB-värden för varje pixel samt kan få fram metainformation om bilden, som storlek eller när den är tagen. Se bilden nedan för hur användandet av biblioteket ‘jpeg-lib’ skulle kunna se ut.

```
var jpeg = require('jpeg-lib')

//Loads image into a Picture object
var picture = jpeg.load('funnycat.jpg')

var height = picture.getHeight()
var width = picture.getWidth()

//Extracts the center pixel as a Pixel object
var centerPixel = picture.getPixelAt(width/2, height/2)

//Print RGB to console
console.log(centerPixel.r, centerPixel.g, centerPixel.b)
```

Figur 1. Exempelkod för hur jpeg-lib’s gränssnitt skulle kunna se ut. Modulen ger möjlighet att läsa bild-filer och få ut värden ur dessa.

Som ni ser i Figur 1 så består jpeg-lib av ett antal klasser (jpeg, picture, pixel) och metoder på dessa som användaren av biblioteket kan använda. Olika programmerare kan använda jpeg-lib för att få funktionalitet för att skriva t.ex. bildbehandlingsappar eller spel. Skillnaden mellan ett bibliotek och en app är alltså att målgruppen för en modul är andra programmerare och inte slutanvändaren av en app.



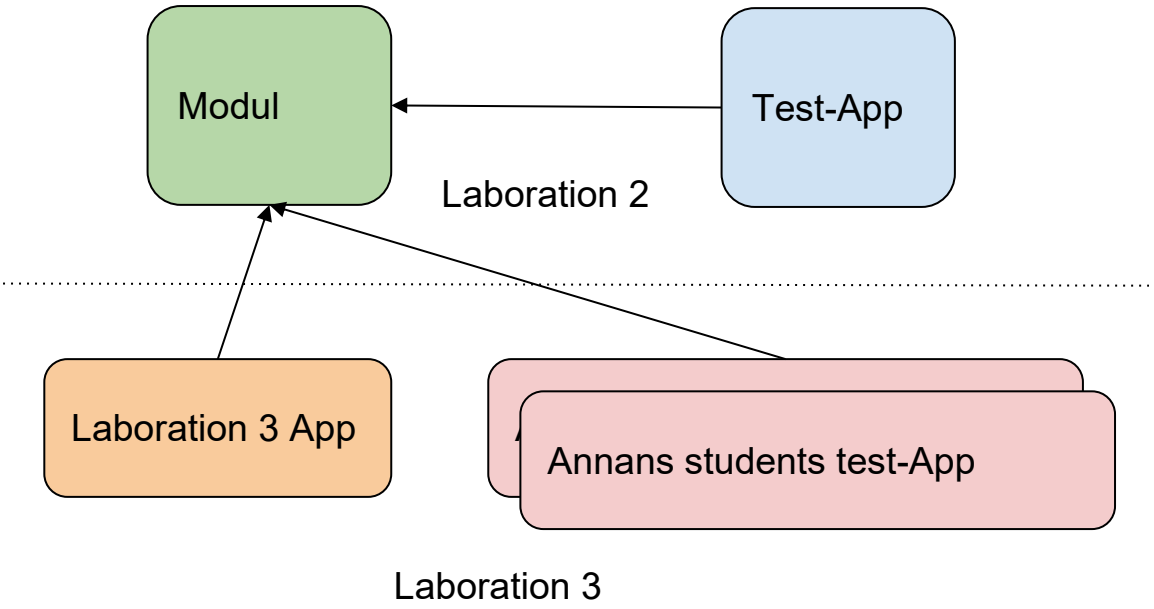
Fler exempel finns att finna på internet. Tänk på att flera av dessa är för omfattande och komplicerade för den här laborationen. Ni behöver hitta en lagom nivå på er modul utan att återanvända någons kod. Mer om storleken på ert bibliotek kommer senare.

För exempel på existerande javascript-libraries se här:
https://kinsta.com/blog/javascript-libraries/
För exempel på Java-libraries se här: https://towardsdatascience.com/top-10-libraries-every-java-developer-should-know-37dd136dff54

Laborationer som beror på varandra

Laborationen och nästföljande laboration hör ihop, se Figur 2, nedan. Er modul från den här laborationen (grön i Figur 2) skall vara till nytta för er i nästa laboration. Där kommer ni skriva en enkel app som riktar sig till slutanvändare som då använder sig av er modul.

Eftersom en modul inte är en hel applikation behöver ni demonstrera att den fungerar redan i den här laborationen med hjälp utav en Test-App (blå i Figur 2). Mer om det senare.



Figur 2. *Grön ruta är modulen ni skriver i den här laborationen. Den testar ni den här iterationen med en Test-App. Under laboration 2 kommer ni använda er modul för att bygga en app som använder er modul men andra studenter också kommer använda er modul för att bygga test-appar.*

Exempelvis skulle ni kunna skriva en modul i den här laborationen som olika typer av diagram utifrån data (ex stapeldiagram, boxplots etc.). För att veta att modulen fungerar skriver ni en liten test-app som ritar ut diagram med test-data.

I nästa laboration skapar ni en app som hämtar temperaturdata från ett api och använder modulen för att visa diagram för olika orter som användaren väljer.
Ni kommer även prova att använda andra studenters moduler.

Mål med era moduler

Eftersom det är så stor frihet att välja vilken typ av funktionalitet er modul skall innefatta så kommer nu några olika mål för att ni skall hitta rätt nivå och fokusera på rätt saker. Sammanfattningsvis är det följande mål ni skall uppfylla

- Skriva en modul och inte en app.
- Lagom storlek och komplexitet.
- Vältestad
- Kodkvalitetskrav
- Reflektion

Skriva en modul och inte en app

Målgruppen för modulen är andra programmerare. **Se till att hitta något som är till nytta för någon.** Precis som i 1dv613 fundera vilken din USP är och beskriv med några meningar vad ditt bibliotek gör och inte gör. Notera att detta kan behöva justeras något under tiden ni arbetar för att bli tillräckligt stort så att ni uppfyller laborationens krav, men tillräckligt litet så att hinner både att lösa uppgiften men också att kommunicera lösningen och reflektera.

Driftsättning skall ske så att andra programmerare kan använda ert lib. Driftsättning skall därför ske publikt på https://github.com/.

Fundera över hur er kommunikationen med de programmerare som skall använda er modul skall ske. Titta gärna på andras libbar för att få bra exempel och diskutera i Slack.

Några saker som jag tänker på är bra om det finns är:

- Dokumentation (README.md)
- Kodexempel
- Installationsbeskrivning
- Kommunikation om beroenden, språk och versioner
- Testrapporter
- Buggrapporter/issues
- Information om licens för öppen källkod
- Versionsnumrering och releaser
- Kommunikation om hur användare kan bidra till projektet.

Fundera över vilka funktioner/metoder och klasser som det är tänkt att användaren skall använda. Försök förenkla för denne så att modulen blir lätt att använda och även att det är lätt att lära sig hur din modul fungerar. Fundera också på vilka klasser/metoder/funktioner/filer som användaren (programmeraren) inte skall röra, titta på och ändra i. Fundera på hur du döljer det som användaren helst inte skall behöva se.

Lagom storlek och komplexitet

Undvik för enkla eller för komplexa problem. Vad som är “lagom” är dock individuellt. Ni måste tro att ni själva kan klara av hela uppgiften utan att generera, skriva av eller kopiera koden från någon annan.

Jag beräknar ca 40 h arbete med koden, detta kan såklart skilja sig mellan olika studenter. Undvik något där ni lockas att kopiera kod från en existerande lösning (även era egna tidigare lösningar). Förlita er inte på andra moduler (dvs andra libbar) utan försök hitta ett problem som främst hanteras av er egen kod. Exempelvis om er kod bygger på ett API så bör ni inte bara göra ett nytt interface till API’et utan tillföra någon typ av funktionalitet.

En ungefärlig storleksangivelse för er modul (Test-App ingår ej i detta):

- Minst 5 meningsfulla klasser/metoder/funktioner i det publika interfacet
- Minst 10 kontrollsatser(if, for, while) som används för att lösa huvudproblemet
- Minst 200 rader egen kod som används för att lösa huvudproblemet
- Undvik att mappa eller översätta existerande bibliotek
- Tillräckligt för att uppfylla kodkvalitetskrav, se egna rubriker.

Använd er av språk och en programmeringsmiljö som ni har erfarenhet av så att ni kan fokusera på kodkvalitet och inte andra saker.

Verifiering och validering av din modul.

För att nå samförstånd om kodens funktion behöver den testas. Hur detta sker är mindre viktigt än att det sker. Som jag ser det finns det tre val (men fler varianter kan finnas). Oavsett val kommer ni redovisa för mig med hjälp utav en testrapport i Markdown där det tydligt framgår vad som är testat, hur det är testat och vilket utfall det fick. Se även betygskraven.

- Ni testar er kod genom att skapa ett användargränssnitt (webb, console, ui) i en separat TestApp. Ni skapar manuella testfall (se 1dv613) och går igenom varje testfall och matar in indata och observerar själva manuellt utdata och jämför med förväntat utdata. Ni dokumenterar er testning i en testrapport (se 1dv613) som markdown.
- Ni skapar en testapplikation som automatiskt kör varje test var för sig och kör den koden och observerar testernas utfall antingen med kod eller manuellt. Testapplikationen eller ni skapar en testrapport som markdown.
- Ni skapar automatiska enhetstester för er modul med hjälp utav ett testramverk. Ni kör dessa och redovisar resultat med en testrapport i markdown. ni kan länka in eventuella testrapporter ifrån testramverk eller ta screenshots och inkludera i er testrapport.

Skriv en kort summering om hur modulen testats. Var så tydlig så att någon annan skulle kunna utföra testet. För testresultaten.
Skapa en tabell i markdown med tre kolumner. Varje rad blir ett test.

1. Vad som testats / Ex metodnamn, eller krav
2. Hur det testats
3. Testresultat

Kodkvalitetskrav

Eftersom laborationen påbörjas innan många av föreläsningarna har skett så begränsar vi mängden krav på kodkvalitet i första inlämningen till de som nämns i kapitel 2 och 3. Det är viktigare att reflektera och återkoppla till boken än att er kod är perfekt. Jag kräver alltså inte att ni justerar er kod men däremot att er reflektion visar på förståelse för kursmaterialet. Det är här tillåtet att antingen ändra er kod eller leva vidare med den, men bristen skall då finnas angiven i era reflektioner.

Namngivning (kapitel 2)

Läs kapitel 2 i Clean Code. Skapa en tabell över **fem** namn på identifierare (Ex. namn på klasser, metoder/funktioner och variabler) som finns i ditt **publika interface** hos modulen. Det publika interfacet är alltså den kod som andra programmerare ska använda. Utgå ifrån kapitel-2s titlar och ange de viktigaste “reglerna” som applicerats **eller skulle kunna appliceras** på just ditt namn. Försök variera regler mellan namnen så att **inte alla har samma titlar applicerade**. Visa upp att ni förstår flera regler och inte bara ett par.

Ange även en kort reflektion kring innehållet i kapitel 2. Ni kanske upptäcker en brist hos er tidigare namngivning, ni kanske inte håller med någon av “reglerna” från kursboken. Jag ser hellre att ni hittar och reflekterar över era brister än att ni döljer dem.

Exempel.

Namn och förklaring	Reflektion och regler från Clean Code
Tokenizer2000 Klassnamn på huvudklassen i modulen	**Avoid Disinformation** 2000 betyder inget speciellt och tillför därför inget till namnet. **Don't Be Cute** Namnet kan verka sött men är vilseledande. De som inte är millennium-romantiker kan missa det roliga. Bara "Tokenizer" är ett tydligare namn. Jag väljer dock att behålla Tokenizer2000 eftersom användare redan använder min modul
boolean **TokenMatch.isBetter**(other) Metodnamn på metod som avgör om en tokenmatchning är bättre än en annan baserat på maximal munch.	**Method Names** Is hintar att returvärdet är boolskt. Argumentet och metodnamnet är tänkt att läsas som "is this better (than) other". **Use Problem Domain names** Better är oklart i kontexten och borde bytas ut mot namnet "Maximal munch" som problemdomänen använder MEN detta är inte säkert att programmeraren som skall läsa är insatt i detta därför bör vi istället använda "Solution Domain". **Use Solution Domain names** hasMoreMatchedCharacters är ett tydligare namn och kräver inte att läsaren vet om Maximal munch.

Funktioner (kapitel 3)

Läs kapitel 3. Skapa en tabell över dina fem längsta metoder/funktioner. Utgå ifrån kapitel-3s titlar och ange de viktigaste reglerna. Föreslå förändringar.

Metodnamn och länk eller kod	antal rader (ej ws)	Reflektion
boolean TokenMatch.isBetter (other)	101	**Do one thing** Metoden gör bara en sak (jämför längden på matchad text med längden på other). **Function Argument** Metoden har bara ett argument (**monadic**) Eftersom jag har skrivit i javascript vore det bra att typen på argumentet **other** framgår via exempelvis en metodkommentar. **Common Monadic Form** Vi ställer en fråga om argumentet, är this bättre än other och metoden gör endast då en “query” (**Comman Query Separation)** och ändrar inte värdet på objektet eller argumentet.

Ange även en kort reflektion kring innehållet i kapitel 3. Ni kanske upptäcker en brist hos er tidigare skrivning av funktionerna, ni kanske inte håller med någon av “reglerna” från kursboken. Jag ser hellre att ni hittar och reflekterar över era brister än att ni döljer dem.

Reflektion

Skriv en kortare reflektion (halv sida 12pt) där du beskriver dina erfarenheter från din egen kodkvalitet. Använd begrepp från boken.

Inlämning och examination

Koden lämnas in genom formulär Daniel kommer tillhandahålla i Slack några dagar innan deadline (se schema). Formuläret kommer innehålla ert namn samt länk till git-repositorium samt några frågor.

Testrapport och kodkvalitetskrav lämnas in som ett markdowndokument i ert repositorium där även reflektionen skrivs. Namnge dessa filer som testrapport.md och reflektion.md

Regler

- Skriv **all** kod själv, skriv inte tillsammans med någon (sida vid sida-programmering). Ni får inte heller generera kod med hjälp utav LLM eller på andra sätt.
- Kopiera inte kod från någon annan. Skriv inte av kod.
- Använd inte bibliotek eller färdiga metoder för att lösa huvudproblemet, dvs er modul. Om ni tänker if-satser och for-loopar så tänker ni rätt. Ni får använda inbyggda typer och metoder som finns i språkets standardbibliotek. Men om ni behöver “installera beroenden” så skall ni helst undvika det. Diskutera med Daniel. Det är dock **helt okej att använda ramverk i Test-App för att testa er modul.**
- Koden skall kunna delas med klasskamrater under senare laborationer och workshop. Ange licens för öppen källkod och publicera publikt på GitHub.
- Koden är skriven i ett programmeringsspråk som förekommit tidigare än den här kursen i utbildningen. En annan student på samma program skall kunna förstå din modul utan att läsa in sig på ett ramverk.
- Fokusera på Objektorienterad kod med klasser och metoder i klasser. Du kan ha kod utanför klasser men bara om den behövs för att starta upp koden (ex. nodejs “server.listen(port…”). Inga metoder i dina klasser får vara statiska mer än om det behövs för att starta upp koden (Ex. java “public static void main(...”-

För godkänd nivå “E”

- Funktionell kod. All kod skall vara testad. De flesta testfall skall fungera, **något enstaka testfall får misslyckas eller känd bugg får finnas**. Modulen måste dock fungera i stort. Det finns en testrapport som visar vad som fungerar och hur det är testat.
- Koden är tillräckligt stor/komplex för att räknas
- Koden är förberedd för att återanvändas.
- Det finns en läsbar dokumentation på github.
- All kod finns med historik på github (tänk minst daglig commit).
- Tabeller med Kodkvalitetskrav är ifyllda för det publika interfacet som baseras på kursboken
- En reflektion är skriven baserat på vad studenten har lärt dig
- Studenten uppfyller Reglerna.

För högre betyg “A-D”

- Samtliga punkter från “För godkänd nivå ”
- Testningen är övertygande och noggrann. Viktigaste kraven fungerar.
- Koden överträffar storlek och komplexitet
- Det är mycket tydligt utifrån git-repositoriet hur modulen skall återanvändas. Exempel och dokumentation, installationsanvisningar.
- En tydlig separation finns mellan modulen som skall återanvändas och de sätt som modulen har använts för att testa den. Exempelvis kan test-appen ligga i en separat katalog.
- Kodkvalitetskraven är varierade i vilka regler de använder. Studenten är tydlig i hur regler och kodnamn samt metoder hänger ihop.
- En välskriven reflektion är skriven som baseras på erfarenheter och lärdomar utifrån laborationen samt bokens kapitel 2 och 3.