Min Kursrapport

DV1610
Objektorienterade webbteknologier
(mvc)

Johan Kristoffersson
joki20@student.bth.se

v1.0.0
April 03, 2021

Läsperiod 4
Våren 2021

Institutionen för datavetenskap (DIDA)
Blekinge Tekniska Högskola (BTH)
Sverige

Innehåll

Kapitel 1

Objektorientering

1.1 Bakgrund och fördelar

Jag är bekant med objektorientad programmering genom kursen i objektorienterad Python tidigare under våren. Det har underlättat förståelsen för klasser i detta kursmoment, trots annat programmeringsspråk. Alexander Petkovs artikel på hemsidan freeCodeCamp tar upp viktiga principer kopplat till objektorienterad programmering [4]:

1. Inkapsling - möjlighet att skapa unika objekt som användaren kan interagera med via publika metoder. 2. Abstraktion - det underliggande arbetet som objektet utför via de publika metoderna hålls dolda för användaren 3. Arv - möjlighet att skapa subklasser som ärver bas- och föräldraklasser.

Säg du jobbar med ett stort projekt med olika moduler. För att undvika att variabler krockar i globala scopet kan inkapsling göra att var och en kan skapa metoder genom inkapsling där publika metoder kan kommunicera med de andra klasserna. Gällande abstraktion så kan vi ta användare som går in på SMHI eller gör en beställning från en webbshop. Då är det viktigaste med ett tydligt gränssnitt där fokus ligger på att användaren kan slutföra en uppgift (beställa, boka, se lokala vädret etc). Arv i sin tur är viktigt för att kunna skapa exempelvis nya användare på en sida där basklassen innehåller grundläggande info, och där ärvda subklasser arv gör att användarens attribut/egenskaper kan specificeras.

1.2 Klasser och objekt i PHP

Vid skapandet av klasser är det viktigt att utgå från en grund. I den officiella manualen för PHP [5] tas viktiga kodstandarder och grunder upp, likaså på Tutorials Points artikel om objektorienterad PHP [3]

Klasser innehåller variabler (kallas medlemsvariabler) och funktioner. Klassen fungerar som en mall som kan instansiera/skapa objekt utifrån sin mall genom nyckelordet **new** och klassens namn. Objekten tilldelas klassens variabler (kallas då medlemsvariabler/attribut/properties) medans objektens metoder (medlemsfunktioner) delas mellan alla objekt. Om du har definierat en konstruktorfunktion kan du skicka med argument när objekt skapas som då tilldelas attributvärden som du valt ut. Objekt ska komma åt sina egna metoder och attribut används inom objektet syntaxen **this**, medan utanför objektet används objektnamnet som referens. Om du har definierat en klasskonstant nås den inom klassen med **self::konstant** och utanför med klass::konstant Attribut och metoder kan vara antingen **privata** (åtkomliga endast inom klassen), **publika** (åtkomliga även utanför klassen) eller **protected** (för ärvda klasser, åtkomliga inom basklassen och den ärvda klassen). De metoder som du vill att användare ska kunna komma sätter du som publika, medan medlemsvariabler bör vara privata.

1.3 Tankar om kodbasen

Fördelen som jag ser med **Namespace** är att jag kan skapa ett projekt under ett visst namespace, och sedan ett annat projekt under ett annat namespace och inte oroa mig för att skriva över variabler. Användningen av **use** för att definiera om referensen sina klasser kan vara fördelaktig om man vill komprimera sin kod, så länge det inte blir på bekostnad av tydligheten. Gällande autoloader (som inkluderar filer) och composer känns det som områden man inte helt förstår hur de fungerar i bakgrunden men som har ett värde för att exempelvis kunna hantera objekt i sessioner. Strukturen i övrigt med view och Router påminner en del om tidigare kurser som exempelvis databas-kursen där vi använde express och .ejs-filer. Däremot känns Router-filen lite rörig med alla elseif-satser och det blir svårt att få en bra överblick över alla sina views.

1.4 Tankar om game 21

Game21-klassen anropar publika metoder från övriga klasser. Jag använde funktionen **isset** som kontrollerade vilket formulär som postats. I början kollas om knappens name är startgenom **isset och POST[start"]** som nollställer al-

la resultat och visar formulärknappar för 1 eller 2 tärningar. Genom multipla elseif-satser som kollade vilken knapp som trycktes i en bestämd ordning från start till slut, och genom att spara spelarens poäng och antal rundor i sessionsvariabler, så kunde poängen föras vidare när användaren trycker på POST. Koden ligger i klassen Game21 och anropas genom publika metoden game21() i vyn.

Såhär i efterhand borde jag kanske använt mig av switch case-satser för att skapa en tydligare kod-struktur. Jag kunde då också använt mig mer av en sessionsvariabel som definierar vilket läge sessionen befinner sig i. Har försökt kommentera koden noggrannt för att jag och andra ska kunna gå tillbaka till den och förstå vad som händer stegvis. Det underlättade att börja skriva koden i vyn, för att sedan flytta över den till en klass. Då kunde jag fokusera mer på kodens struktur i början när jag löste uppgiften. För denna uppgift använde jag inte graphic-metoden vilket blivit mer tilltalande visuellt. Anledningen till detta var att jag inte på ett lätt sätt kunde få ut och använda siffrorna för tärningsslagen.

1.5 Tankar om PHP The Right Way

Dokumentet PHP The Right Way"[2] är en communitybaserad online-handbok inom PHP. De kanske viktigaste och mest intressanta områdena i dokumentationen är de om sessioner, databaser och inloggning med säkra lösenord. Hur man gör för att bygga upp säkradatabassystem med användare och lösen känns högst relevant att behärska som programmerare. Väl ute på arbetsmarknaden kommer det att ställas krav på att vi ska behörska dessa delar och PHP är fortfarande ett vanligt språk som hängt med länge och som många arbetsplatser fortfarande sitter med. Att kunna skapa nya användare utifrån klasser och lägga in dem i databasen känns som en bra fortsättning.

1.6 Today I learned

Jag tar med mig vikten av att låta logisk kod tillhöra klasserna för att skapa renare vyer. Det är också viktigt att endast göra metoder som användaren ska komma åt publika. Det är viktigt att då inte ändra metodernas namn om koden ska ändras, utan att istället ändra på klassens kod.

Kapitel 2

Controller

2.1 Modellering av flödesdiagram och pseudokod

I skapandet av flödesdiagram och pseudokod känns det bäst att utgå från ett bottom-up perspektiv. Jag brukar oftast definiera ett steg i taget i detalj. När jag är klar med första steget går jag vidare till nästa, och definierar det. På detta sätt byggs flödesschemat upp, steg för steg, och sätts ihop till en helhetsbild. Därefter kan jag skapa pseudokoden för att gå in mer på detaljer i vad som ska hända under stegen. Sedan bygger jag upp min grund utifrån flödesschemat, och tittar sedan närmare på pseudokoden för att lägga till ytterligare detaljer.

2.2 Arv, komposition, interface och trait

Subklasser kan ärva metoder och attribut från en annan klass (som då kallas föräldraklass eller basklass) genom nyckelordet "extends". Subklassen har då ett is-a förhållande till föräldraklassen. Arv fungerar för klasser som bara behöver ärva från en enda klass. Komposition innebär ett ägandeskap (has-a), exempelvis att inuti en klass "hus"eller dicehand"konstruktor har objekt rum"eller dicesskapats. Om det ägande objektet försvinner så försvinner även objektet som ägdes från den andra klassen. Interface använder nyckelordet "implementsöch är ett slags kontrakt där man definierar vilka metoder klassen ska innehålla. Exempelvis att klassen Dice ska implementera vissa metoder från detta interface (rulla tärning och få ut senaste tärningsslag exempelvis). Trait är ett alternativ till arv som tillåter en klass att använda metoder och attribut från andra klasser

(jämför med arv som erbjuder metoder från en enda föräldraklass). [6]

2.3 Utvecklingen av Yatzi-spelet

Jag utgick som grund från mitt flödesschema och försökte bygga upp ett grundläggande flöde. Därefter kollade jag i min pseudokod för ytterligare detaljer. Jag är nöjd utifrån förutsättningarna. Valde att tillämpa så att användaren kan välja vilken av 1-6 som denne vill sätta poäng för. Inser att eftersom jag har en enda funktion i min Yatzi-klass som startar igång och kör igenom spelet (samt ärver funktioner DiceHand-klassen, som i sin tur ärver från Dice-klassen) så blir koden ganska lång, det hade jag kunnat splitta upp i fler funktioner som anropar varandra i sekvens för att reducera den cyklomatiska komplexiteten. Det största problemet uppstod när jag skulle skapa koden för att spara vissa tärningars indexposition (för att de inte skulle slås om). Jag sparade tärningarnas indexposition i en array, och vid nytt slag kollade jag om alla tärningars index fanns i arrayen vilket då skulle göra att tärningen inte slogs om. Problemet var att när jag letade efter matchning i index för varje tärning, så tog den för nästa sparade indexposition inte hänsyn till den förra och då slogs den tärningen om fast den inte skulle. Detta löstes till slut genom inbyggda funktionen in_array där tärningsindex som inte förekom i arrayen slogs om. Koden passerade make test som ok, men det kan förbättras ytterligare enligt ovan. Givetvis hade jag även kunnat skapa ett komplett spel med två spelare.

2.4 Kodstrukturen

Jag ser en förbättring med kodstrukturen. I och med att vi använder trait för att använda attribut och metoder från en baskontroller-klass blir koden renare i de övriga controllerklasserna. Routerkoden känns mer komprimerad, men att ha en mapp view/layout som importerar header, sida och footer, undrar jag om det inte hade räckt med motsvarande filer i view. Ju fler mappar att hålla reda på, desto svårare blir överblicken. Annars är den största fördelen med att vi använder externa ramverksmoduler för request och response att systemet sköter dessa bitar automatiskt för oss. Det gör att vi kan fokusera mer på innehållet i koden för våra spel än att få ihop en fungerande sida. Dock känns koden rätt maffig att förstå sig på i detalj, så det får räcka i nuläget med att förstå sig på hur request-Controller-response-flödet fungerar i grunden. Men det blir lite Separation of Content"över det hela, att var kod har sin plats i vårat system.

2.5 Today I learned

Från kursmomentet tar jag med mig att en god planering i förväg kan hjälpa dig att följa en röd tråd när du väl börjar programmera. Istället för att behöva göra om och göra rätt, kan du bygga upp en grundstruktur och sedan börja gå ner på detaljnivå. Att lägga upp arbetet på detta sätt är användbart när större projekt utvecklas och du ansvarar antingen för en del av projektet eller för hela. Att då planera i förväg hur du skall ta dig an problemet är minst lika viktigt oavsett vilken nivå du ansvarar för.

Kapitel 3

Enhetstestning

3.1 Testning med PHPUnit

För oss som har läst objektorienterad python och testning med unittest är detta område väldigt bekant, inte minst assertEquals. Jag använde mig av manualen [1] och stackoverflow för att hitta de assertions jag behövde, exempelvis ässertStringContainsStringför att hitta substrängar. I princip går det att utläsa enbart från en assertion vad den testar, men samtidigt är exemplen i manualen inte de mest konkreta, bland annat gällande att hitta värden i arrayer. För kvalitetssäkring av sin kod så är PHPUnit bra under förutsättning att programmeraren skriver tester som tar hänsyn till olika scenarier, exempelvis vad som ska hända ifall det inte skickar med argument till en funktion som behöver det.

3.2 Kodtäckning

Min totala kodtäckning var 93,3% för rader och 80,43% för funktioner. Alla mina egengjorda klasser hade 100% kodtäckning. Controllerklassen hade en kodtäckning på 70 respektive 61,11 %. Jag tolkade det som att vi skulle fokusera på den kod vi själva skapat, därav inga tester på denna klass. Hade jag arbetat på ett mindre företag hade jag förmodligen sett till att även denna kod varit 100% eftersom ett fel i Controllern kan bli förödande för företagets ekonomi. För detta kursmoment räckte inte tiden till för att hinna med detta. Som webbprogrammerare har du ett stort ansvar för att din kod håller en god kvalitet och att kunden blir nöjd, därför är en god (och relevant) kodtäckning mycket viktigt.

3.3 Testbar kod

När jag gjort mina tester så har jag använt mig klart mest av assertEquals för att kolla att jag fått den respons jag ville ha vid anrop av klassers metoder. Det jag kan utveckla är inte minst testning när metoder eller instansiering av klasser tar emot argument som inte är tänkta att skickas med och att då lyfta exceptions. I övningen använde jag mig av expectException för detta ändamål. Vid enstaka fall, som i början på testningen av Yatzy-klassen, satte jag assertEquals för att kolla att ett sessionsvärde jag satt verkligen var det satta. Just denna aspekt ser jag som mindre relevant att testa, men PHPUnit var nöjd med detta för att godkänna några rader kod i början. För metoder som inte ger någon return men kanske modifierar exempelvis en array eller variabel har jag exempelvis kollat att arrayerna innehåller det som förväntas av dem.

3.4 Anpassa kod utifrån testning

I förra kursmomentet, när jag skapade klasserna Game21 och Yatzy, så hade jag en enda metod för vardera klassen som hanterade hela spelet (samtidigt som vynernas uppgift endast var att starta igång metoden. Då fokuserade jag mycket på att kommentera koden väl. I detta kursmoment skapade jag en switch-controller som analyserar vilken typ av POST som görs, och därefter aktiverar rättmetod i klassen. Vissa metoder blev fortfarande rätt omfattande, exempelvis rollDice-metoden i Yatzy-klassen som hanterar såväl omslag av alla eller enskilda tärningar. Hur som är koden mer strukturerad än innan och min switch-controller gör det lättare att kontrollera och se vad som ska hänta i koden än tidigare. Dessutom gav det mig möjlighet att skapa separerade tester som fokuserade på varje funktion.

3.5 Testbar kod = snygg kod?

Genom att bryta upp och isolera sin kod i mindre beståndsdelar, exempelvis i metoder som inte inkluderar andra metoder, blir det lättare att testa det som avses utföras. Dessutom underlättar det möjligheten att använda en metod även i andra program om den inte är beroende av övriga metoder. En metod som utför en uppgift istället för flera kan också få mer precis funktionsnamn. Jämför exempelvis rollDice() och score() som vardera sköter en sak, vilket ger tydligare och mer avgränsad kod än om man slog ihop dessa funktioner till rollDiceAndScore(). Om du lämnar över ett projekt till någon annan och har skrivit testbar och ren kod så sparar det tid för dig och företaget (och tid är som bekant pengar).

3.6 Today I learned

Jag har lärt mig vikten av att skriva testbar kod. Det har gjort att jag tvingats tänka om mycket gällande min kodstruktur och hur jag kan isolera min kod till mer testbara beståndsdelar. Jag inser vikten av att en vältestad kod redan från början kan förhindra buggar som längre fram kan leda till kostsamt debuggingsarbete.

Kapitel 4

Ramverk

Här skriver du din redovisningstext för detta kursmoment.

Kapitel 5

Autentisering

Här skriver du din redovisningstext för detta kursmoment.

Kapitel 6

To be defined

Här skriver du din redovisningstext för detta kursmoment.

Kapitel 7

Projekt & Examination

Här skriver du din redovisningstext för detta avslutande kursmoment.

7.1 Projektet

Här skriver du din redovisningstext rörande projektet.

7.2 Avslutningsvis

Här skriver du de avslutande orden om kursen.

Litteratur

[1] Sebastian Bergmann. "Writing Tests for PHPUnit)". I: (2020). URL: https://phpunit.readthedocs.io/en/9.5/writing-tests-for-phpunit.html. (besökt: 04.18.2021).

[2] Phil Sturgeon et al Josh Lockhart. "PHP The Right Way". I: (2021). URL: https://phptherightway.com/. (besökt: 04.03.2021).

[3] "Object Oriented Programming in PHP". I: (2021). Dnr 2019:00860, Promemoria, Beskrivande statistik. URL: https://www.tutorialspoint.com/php/php_object_oriented.htm%22. (besökt: 04.03.2021).

[4] Alexander Petkov. "How to explain object-oriented programming concepts to a 6-year-old". I: (2018). URL: https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260/. (besökt: 04.03.2021).

[5] "PHP: The Basics - Manual". I: (2021). URL: https://www.php.net/manual/en/language.oop5.basic.php. (besökt: 04.03.2021).

[6] Mikael Roos. "Arv, komposition, interface, trait i PHP (med Mikael)". I: (2021). URL: https://www.youtube.com/watch?v=n3dC2E4zACM. (besökt: 04.13.2021).