Linnéuniversitetet
Kalmar Växjö

Jesper Hellquist 05/01-2024
19990218-2212
Webbteknik 5 1ME325

# 1ME325: Webbteknik 5; kursuppgift

## Inledning

I kursuppgiften har en tärningsapplikation skapats med hjälp av att skriva objektorienterad Javascript-kod. Tärningsapplikationen är uppbyggd på 16 Javascript-filer där varje fil representerar en "klass" eller byggsten för applikationen. När dessa "klasser" används i programmet kallas de för objekt, inom objektorienterad Javascript är ett objekt är en representation av en sak eller en person (Stefanov, Sharma 2013, s 13). Ett objekt är alltså ett substantiv och till varje objekt kan det finnas metoder (verb) och egenskaper (adjektiv). Metoder är funktioner som objektet kan använda sig av för att utföra något och egenskaper tilldelas olika värden (Stefanov, Sharma 2013, s 13).

I webbapplikationen finns det två ikoner, en ikon med en klocka och en ikon med en tärning. För att starta en tärningsapplikationen måste användaren klicka på tärningsikonen och då presenteras ett fönster upp på webbsidan som är själva tärningsapplikationen. Programmet tillåter flera antal tärningsapplikationer eller fönster igång samtidigt och varje instans fungerar helt av sig själv. Tärningsfönstret erbjuder "drag and drop" funktionalitet vilket betyder att användaren kan dra och flytta runt fönstret på skärmen. Övrig funktionalitet som gör applikationen användbar är att det finns tre knappar som lägger till, tar bort och kastar om tärningarna samt att det finns en räknare som uppdaterar den totala summan av alla tärningar som ligger ute på fönstret.

Webbapplikationen består av en index.html-sida som dynamiskt uppdateras med hjälp av Javascript och hur detta är implementerat kommer att diskuteras nedan.

## Start

För att starta upp programmet med all funktionalitet skapas en Main.js-fil som innehåller ett objekt bundet till en variabel med variabelnamnet Main. Main definieras mellan två {} och detta är vad som kallas en objekt literal (Stefanov, Sharma 2013, s 99). I Main finns det två metoder, den första "init"-funktionen körs när webbsidan har laddats in och lägger på händelselyssnare på ikonen med en tärning. Om en användare klickar på ikonen med tärning anropas den andra funktionen i Main som skapar en ny instans av DiceWindow som är en "klass" som representerar tärningsfönstret.

Linnéuniversitetet
Kalmar Växjö

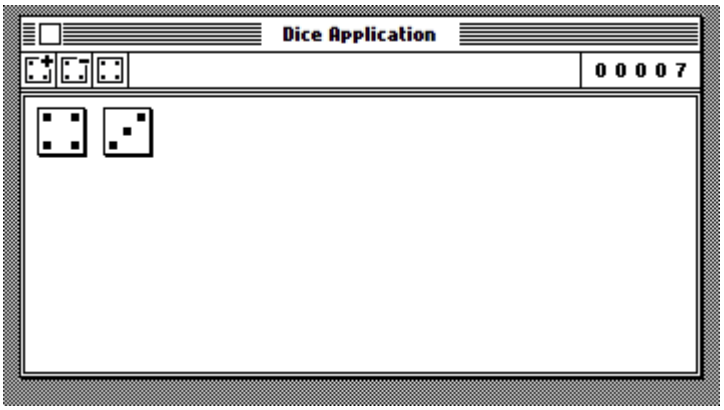
Jesper Hellquist 05/01-2024
19990218-2212
Webbteknik 5 1ME325

# DiceWindow

DiceWindow är en constructor funktion som skapar nya instanser av underliggande objekt som tillsammans bygger ett tärningsfönster med tillhörande funktionalitet. Ett tärningsfönster är uppdelat i flera olika grafiska element som tillsammans bygger hela fönstret (se Figur 1). DiceWindow är själva behållaren och högst upp i behållaren finns en menyrad med en tillhörande stängknapp. Under menyraden finns en verktygsrad med tre knappar och en räknare. Under verktygsraden finns behållaren där tärningarna kan placeras ut, gränssnittet tillåter max 40 tärningar. DiceWindow har en metod som ligger element till html-strukturen för att tärningsfönstret ska kunna presenteras på webbsidan.

Varje element i det grafiska gränssnittet som är en sak representeras av en “klass” eller en constructor-funktion, till exempel MenuBar för menyraden, ToolBar för verktygsraden och Dice för en specifik tärning. Fördelen med constructor funktioner är att de kan ta emot parametrar vilket ger möjligheten till att skicka med egenskaper till andra objekt (Stefanov, Sharma 2013, s 104).

Funktionaliteten för “drag and drop” instansieras som en egenskap i DiceWindow där två argument skickas med, första argumentet är en referens till div-elementet som representerar tärningsfönstret och det andra är en referens som hänvisar till div-elementet som representerar menyraden. Vad som får “drag and drop” att fungera finns inte i DiceWindow utan i en egen “klass” DragnDrop.

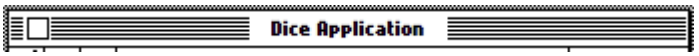



Figur 1 - visar hur ett tärningsfönster ser ut med alla olika grafiska element.

# MenuBar

MenuBar är en “klass” som är en constructor-funktion som skapar den grafiska representationen av menyraden (se Figur 2). MenuBar instansieras av DiceWindow där ett argument skickas med i form av det div-element som fungerar som behållare för hela tärningsfönstret. Detta görs för att MenuBar ska kunna läggas till på rätt plats i HTML-strukturen.

MenuBar består av ett div-element och ett underliggande barn-element som är en instans av CloseBtn, som är en stängknapp. Vid instansiering av CloseBtn skickas två argument med, första är en referens till div-elementet knappen ska läggas till i och den andra är en referens till hela tärningsfönstret. CloseBtn har en primär funktion och det är att ta bort tärningsfönstret från HTML-strukturen och resultatet av det blir att tärningsfönsret stängs ner. Detta görs möjligt eftersom referensen till hela tärningsfönstret finns tillgängligt via en av de två parametrarna i "klassen" CloseBtn.




Figur 2 - visar den grafiska representationen av "klassen" MenuBar med tillhörande stängningsknapp (CloseBtn) i vänster hörn.

## ToolBar

ToolBar representerar en "klass" som skapar den grafiska representationen för verktygsraden och här finns tre knappar som är, lägg till-knapp som är en instans av AddBtn och lägger till en ny tärning i behållaren. En ta bort-knapp som är en instans av RemoveBtn och tar bort den sista tärningen i behållaren om det finns några. En kasta-knapp som är en instans av RollBtn och kastar om de tärningar som finns i behållaren. Till sist finns en räknare i ToolBar som är en instans av "klassen" Counter som uppdaterar räknaren beroende vad summan av alla tärningar är. I Figur 3 visas den grafiska representationen av hela ToolBar.


Figur 3 - visar den grafiska representationen av ToolBar.

## AddBtn

AddBtn är en "klass" som representerar lägg till-knappen i ToolBar som är placerad längst till vänster (se Figur 3). AddBtn är en constructor funktion som skapar en knapp som lägger till en tärning i behållaren. AddBtn består av flera olika egenskaper men en viktig egenskap är "dices" som definieras först som en tom Array. Denna egenskapen kommer användas som en lista för alla tärningar som läggs till i den grafiska behållaren. I metoden "click" som ägs av AddBtn lägger på en händelselyssnare på knappen som "lyssnar" om en användare klickar på knappen. Sker det att en användare klickar körs en anonym funktion som kontrollerar längden på egenskapen "dices" och om längden är mindre än 40 (får max vara 40 tärningar i behållaren och då gäller samma för "dices"-arrayen) skapas en ny instans av "klassen" Dice som representerar en tärning. Därefter läggs tärningen till i behållaren (den grafiska representationen) och in i arrayen "dices". När tärningen är tillagd i "dices" skickas "dices" vidare som argument till en annan metod som ägs av AddBtn och heter "checkDices".

Metoden “checkDices” tar emot en parameter och är en array innehållande en eller flera tärningar. Metoden “checkDices” har en uppgift och det är att loopa igenom arrayen och hämta ut varje tärnings värde och addera det till en lokal variabel "sum" (förkortning på summa). Summan av alla tärningar skickas sedan vidare till en metod som finns i “klassen” Counter.

När en tärning skapas, skapas först ett HTML-element som sedan tilldelas ett slumpmässigt classnamn som är ett “strängvärde” mellan 1-6 och beroende på vilket classnamn som är tilldelat, tilldelas nu ett numeriskt värde som en egenskap av Dice som matchar den tärningens classnamn. Till exempel om classnamnet på tärningen blir “five” får tärningen värdet 5.

## RemoveBtn

RemoveBtn är en “klass” som representerar ta bort-knappen och har en uppgift och det är att ta bort de tärningar som finns i den grafiska behållaren och ta bort tärningarna från egenskapen “dices” som finns i AddBtn. Men för att komma åt egenskapen “dices” som finns i AddBtn måste en referens till AddBtn skickas med som argument när RemoveBtn instansieras, vilket sker i ToolBar “klassen”. Nu när det går att komma åt egenskapen “dices” i RemoveBtn kan den sista tärningen i arrayen plockas bort när användaren klickar på knappen. Därefter skickas den nya arrayen av tärningar till “checkDices” och detta är också möjligt eftersom det redan finns en referens till AddBtn som äger metoden “checkDices”.

## RollBtn

RollBtn är en “klass” som representerar kasta-knappen och har en uppgift som är att när användaren klickar på knappen kastas alla tärningar som finns i den grafiska behållaren om och får nya värden.

Som i RemoveBtn måste också RollBtn komma åt egenskapen “dices” som finns i AddBtn. På samma sätt som när RemoveBtn instansieras skickas en referens till AddBtn-objektet som argument till RollBtn “klassen”.

Metoden “click” lägger på händelselyssnare på knappen och “lyssnar” efter om användaren klickar på knappen. När användaren klickar körs en anonym funktion som först hämtar ut alla tärningar från HTML-strukturen och sparar dem i en variabel. Därefter loopas längden av listan av tärningar igenom och genererar ett nytt slumpat classnamn. Classnamnet tilldelas den aktuella tärningen i iterationen samtidigt som tärningen tilldelas det nya värdet som matchar det nya classnamnet. Efter loopen finns nu en uppdaterad lista med tärningar som skickas med till metoden “checkDices”.

Linnéuniversitetet
Kalmar Växjö

Jesper Hellquist 05/01-2024
19990218-2212
Webbteknik 5 1ME325

# Counter & Digit

Counter är en "klass" som fungerar som behållare för 5 instanser av "klassen" Digit (se Figur 4). Instanserna av Digit läggs till i en egenskap av Counter i form av en Array, detta för att senare kunna bestämma vilken av de 5 instanserna av Digit som ska uppdateras.

Counter har en metod som heter "setValue" och tar emot en parameter i form av ett numeriskt värde som är det tal som ska presenteras i räknaren. Räknaren ska presentera talet från höger till vänster vilket exemplet i Figur 4 visar.

För att uppnå att siffrorna uppdateras från höger till vänster måste ett startindex räknas ut för vilken av de 5 instanserna av Digit som ska uppdateras. Detta sker genom att ta värdet som används som parameter i metoden "setValue" och konverterar det till en text-sträng för att komma åt *length* som är en egenskap av det inbyggda sträng-objektet i Javascript (Stefanov, Sharma 2013, s 330)**.** Till exempel om värdet som kommer in i metoden via parametern är 3, konverteras det nu till text-strängen "3" och har då längden 1. I arrayen som de 5 instanserna av Digit ligger i har 5 insättningar men platserna är inte indexerade med 1-5 utan till 0-4. Till exempel den första digiten i arrayen har index 0 och den sista index 4. Om nu text-strängen "3" har en längd på 1 betyder det att endast Digit på index 4 behöver uppdateras. Det är nu startindexet kommer till nytta och det räknas ut genom att ta 5 minus längden på text-strängen som i tidigare exempel visar vara 1. Då blir startindexet 5 minus 1 som blir 4 och detta betyder att det är Digit med index 4 i arrayen som ska uppdateras. Digiten med index 4 anropar sin metod "setDigit" som kan ta emot ett värde mellan 0-9 och byter därefter classnamn på digiten för att matcha samma värde.


Figur 4 - visar hur Counter och Digit "klassen" presenteras.

# DragnDrop

DragnDrop är en "klass" som skapar funktionalitet för att kunna flytta hela tärningsfönstret via menyraden. DragnDrop bygger på 4 metoder som fungerar som en serie av händelser som användaren gör med muspekaren. Först läggs händelselyssnare på menyraden som "lyssnar" om användaren trycker ner muspekaren på menyraden. När det sker anropas metoden "mouseDown" som sparar positionen för tärningsfönstret och positionen för muspekaren. Därefter lägger metoden till händelselyssnare för mousemove och mouseup på hela dokumentet. Om användaren håller ner musknappen på menybaren och flyttar musen anropas metoden "mouseMove" som ändrar tärningsfönstrets position beroende på muspekarens position. Här sätts även ett z-index på tärningsfönstret i form av en tidsstämpel i sekunder, detta för lägga det fönser man drar i, högst upp i z-leden.

Om användaren släpper musen anropas metoden “mouseUp” som tar bort händelselyssnare från dokumentet. Detta görs för att musen ska släppa fönstret på den positionen “mouseUp” aktiveras.

## Reflektion

Under utvecklingen av tärningsapplikationen möttes flera motgångar och problem som till exempel hur strukturen skulle se ut och vilka “klasser” som behövdes. En bra början var att dela upp tärningsfönstert i de visuella objekt och dela upp dem efter det. Därefter kom det mer naturligt om vad som bör vara en “klass”. Att se objekten som saker gjorde det lättare att greppa och se helheten, till exempel en knapp är en sak och denna knapp har olika egenskaper och ska kunna utföra en eller flera saker.

Brister i programmet är att exempelvis metoden “checkDices” som ägs av AddBtn inte har någon direkt relation till vad AddBtn ska göra. Metoden skulle eventuellt passa bättre i “klassen” Counter som är en räknare vilket är vad metoden “checkDices” gör, räknar ihop värdet av alla tärningar.

Annars är tärningsapplikationen relativt väl organiserad med “klasser” som representerar olika saker och vad dessa saker kan göra. Flera av “klasserna” som har fler än en metod eller metoder som inte behöver vara unika för en viss instans av ett objekt har lagts till i objektets prototyp för att spara minnesallokering (MDN contributors 2023). Till exempel när det är två tärningsfönster igång samtidigt behöver det bara finnas en metod för “click” i AddBtn och inte två likadana (en för varje fönster) som gör samma sak.

## Länk till MeLabb

https://melab.lnu.se/~jh225ci/Jesper_Hellqvist_1ME325/

Linnéuniversitetet
Kalmar Växjö

Jesper Hellquist 05/01-2024
19990218-2212
Webbteknik 5 1ME325

# Källförteckning

MDN contributors. (2023). *Memory management.* https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_management [05/01-2024].

Stefanov, Stoyan. & Sharma, K. Chetan. (2013) *Object-oriented JavaScript*. 2nd ed. Birmingham, UK: Packt Publishing.