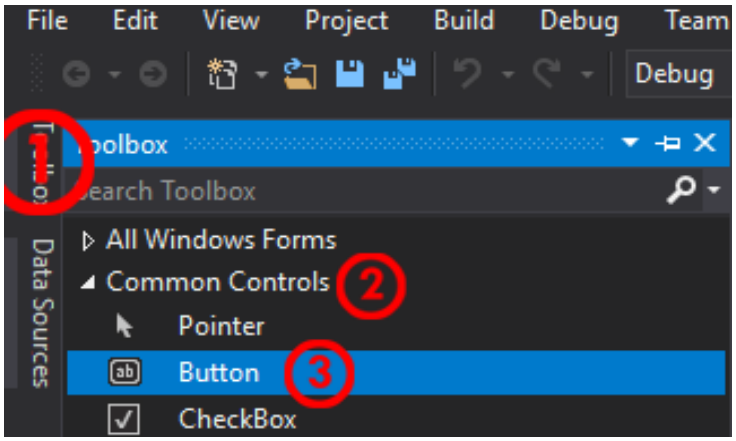
Programmering 2

Övning 3 – Tipsmaskinen

I kursens tredje uppgift ska vi arbeta med ett nytt sorts projekt; Windows Forms. Vi kommer även titta på hur man kan arbeta med några grundläggande externa filtyper, i det här fallet text. Vi kommer gå igenom några av grunderna för Windows Forms, och arbeta oss fram från det.

Windows Forms

Vi börjar med att starta ett Windows Forms-projekt. Det är då istället för en Console App.
Det som skiljer Windows Forms från de konsolapplikationer vi har arbetat med hittills är att vi inte längre kommunicerar med en interface direkt genom text, utan istället klickar vi och skriver oss fram; som vi gör med de flesta vanliga Windows-program. Det här sättet att interagera med ett program via klick kallas i bred bemärkelse för eventbaserad programmering.
När vi har startat projektet kommer vi att bemötas av en grundläggande form. Vi kan forma och vrida och vända på den, och genom att markera fönstret kan vi ställa in sådana saker som bakgrundsfärger, grafik, benämning, etc.



Om vi vill lägga till något, säg en grundläggande knapp, så går vi till *Toolbox* (längst till vänster i programmet) -> *Common Controls* -> *Button*.
Den kan vi dra in i programmet. Windows Forms stödjer Drag-and-Drop interaktion, vilket gör att vi enkelt kan inkludera nya element genom att dra in dem.

För att bestämma vilken kod som ska köras när vi klickar på olika objekt behöver vi bara dubbelklicka på elementet som ska ändras, så får vi upp en färdiggenererad grundläggande kod. Det kan se ut såhär;

```
private void button1_Click(object sender, EventArgs e)
{
    // KOD HÄR
}
```

Här kan vi skriva vår kod; vad som händer om användaren klickar på vår knapp.
Varje Form är ett eget fönster, och ett program kan bestå av flera. För att få kontroll över vad som händer när vardera fönster öppnas kan du använda dig av varje forms konstruktor. De kan se ut såhär:

```
public Form1()
{
    // KOD HÄR
}
```

Importering

Utöver att arbeta med ett grafiskt gränssnitt ska vi börja arbeta med externa filer. I den här uppgiften har vi kravet att vi ska importera text från en textfil och spara det som böcker i en lista. Vi tar det här steg för steg vad vi behöver göra för att det ska fungera;

1. Läs in all text från filen och spara, rad för rad, i en stränglista. `List<string>`
2. Plocka ut varje sträng från listan, separera strängen till strängvektorer. `string[]`
3. Spara strängvektorer i en strängvektorlista. `List<string[]>`
4. Använd sparade värden i vektorerna för att skapa böcker.
5. Spara böcker i boklista.

Hela den här processen ska ske i en Importeringsklass. Så vi skapar en klass kallad FileLoader och börjar gå igenom de olika stegen.
För att läsa in en text, rad för rad, kan vi använda en StreamReader. En StreamReader etablerar kontakt med en textfil och läser in dess värden. Det är ett objekt, likt ett Random-objekt, och har diverse metoder tillgängliga. Så vi skapar en ny StreamReader;

```
StreamReader reader = new StreamReader("texter.txt", Encoding.Default, true);
```

En viktig punkt att ha i åtanke är just sökvägen. I det ovanstående exemplet använder vi endast ”texter.txt” som sökväg, inga undermappar eller liknande. Det betyder att filen kommer att förväntas finnas i samma mapp som vår .exe-fil. När vi kör programmet i Debugläge är den mappen i MittProjektNamn > bin > Debug. När vi kör programmet i Releaseläge ligger grundmappen istället i MittProjektNamn > bin > Release. Se till att du har din textfil med rätt namn, på rätt plats.

Vi skapar sedan en strängvariabel som kommer hålla varje rad som vi importerar från ”texter.txt”. Då vi vet att textfilen inte innehåller några plötsliga avbrott eller tomma rader kan vi göra det enkelt för oss genom att skapa en while-loop som avbryts när det inte längre finns ett värde att läsa in från reader.

```
while (reader.ReadLine() != null)
```

Varje rad kommer då ha värdet av ”`reader.ReadLine();`” inom loopen. Vi kan tilldela dessa värden till vår sträng, som vi sparar i en stränglista.
För det andra steget behöver vi separera ut våra strängar (sparade i en stränglista) till strängvektorer. Vi kan göra det genom Split-metoden. Split-metoden utgår vanligtvis från ett char-värde, men då den här texten innehåller ”###” som separator behöver vi bygga det lite annorlunda. Vi vill då loopa igenom alla värden vi har fått in i vår stränglista, och applicera en Split för varje element.

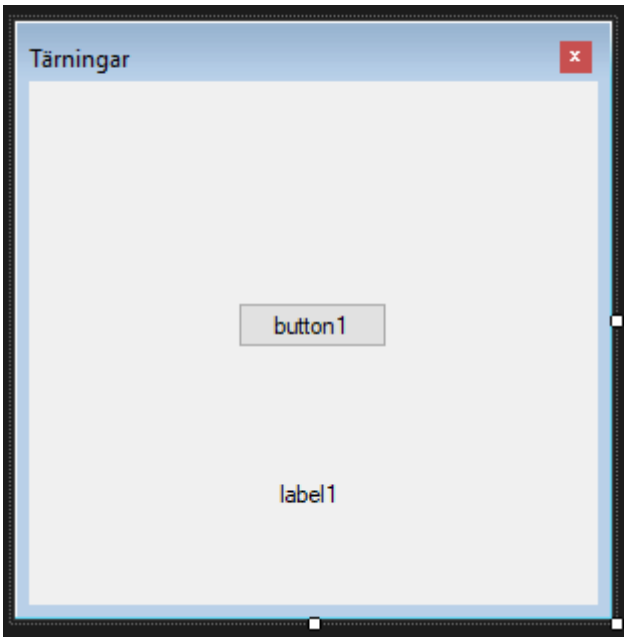
```
string[] vektor = sparadSträng.Split(new string[] { "###" }, StringSplitOptions.None);
```

Vi sparar sedan vår vektor, som innehåller sorterade och separerade strängar, i en vektorlista. Här börjar vi sedan arbeta med att använda listan av vektorer för att bygga upp böcker. Vi kan se i vår textfil att alla värden kommer i en specifik ordning, och kan utifrån det bygga upp en metod som går igenom alla vektorer och skapar böcker utifrån dess värden. Böcker som i sin tur sparas i en boklista.

Den här slutgiltiga boklistan är vad vi utgår ifrån när vi sedan ska slumpa fram värden till vår Tipsmaskin. Då varje bok ska ha en egen ToString-override kan vi referera direkt till olika element i vår boklista för att få ett textvärde.

Exempel; Windows Forms

Låt oss säga att vi vill göra ett enkelt program som har en stor knapp. När man trycker på knappen ska ett värde mellan 1-6 slumpas fram.
Vi börjar med att skapa ett nytt Windows Forms-projekt. Vi går via Toolbox -> Common Controls -> Button och drar in en ny knapp, samt en Label.



Vi dubbelklickar på knappen så att vi får fram kod för vad som händer vid ett click-event.

Vi börjar med att sätta upp två statiska variabler. En int som kan nås av alla metoder i klassen, samt ett Random-objekt som vi kan använda för att få fram tärningsvärden.

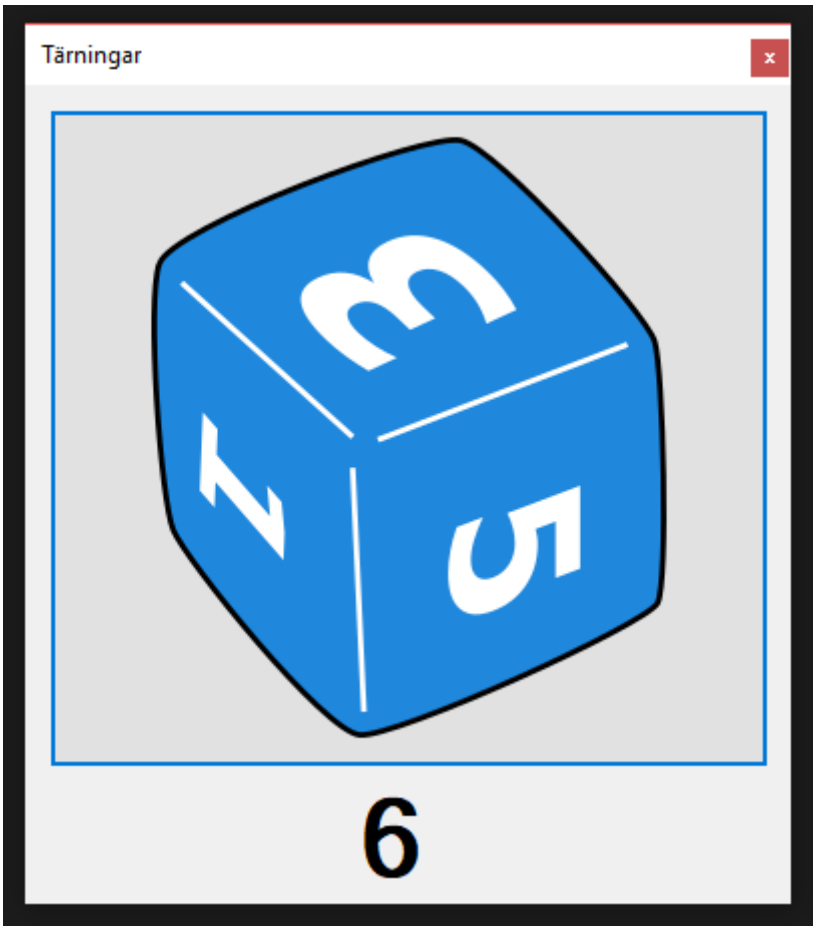
Vi går sedan in i click-eventet och gör två saker; ger vår statiska int ett nytt randomvärde mellan 1-6 samt uppdaterar texten till label1. Den grundläggande koden som vi behöver ser då ut såhär, inuti Form1.cs ;

```
public partial class Form1 : Form
    {
        public static Random rng;
        public static int siffra;

        public Form1()
        {
            InitializeComponent();
            siffra = 1; // Ger ”siffra” startvärdet 1
            label1.Text = Convert.ToString(siffra); // Ger label1 ett startvärde
            rng = new Random(); // Deklarerar ”rng” som en ny Random.
        }

        private void button1_Click(object sender, EventArgs e)
        {
            siffra = rng.Next(1,7); // Slumpar fram en ny siffra från 1-6
            label1.Text = Convert.ToString(siffra); // Uppdaterar label1
        }
    }
```

Så nu har vi grundfunktionen för programmet. Vi markerar label1 och ändrar dess font (under Properties) till att vara storlekt 40 och Bold. Vi kan även ändra dess starttext till ”1” för att lättare se i vårt Designfönster hur det kommer att se ut.
Vi markerar sedan button1 och tar bort dess text. Vi använder ”image” och vidare ”import”, där vi väljer en bild på en tärning i lämplig storlek. Vi flyttar och ändrar storlek på fönster och knapp, och får till sist fram ett slutresultat;



Det här har då varit ett exempel på hur vi på ett enkelt vis kan arbeta med Windows Forms. Vi använder vår Toolbox och skiftar mellan Designfönstret och Form1-koden. När du arbetar med Windows Forms är det bra att ha i åtanke åtkomstnivåer, och hur olika fönster interagerar med varandra. Testa dig fram och prova gärna att göra ditt eget tärningsprogram.