# Brugervenlighed og robusthed af GUI

Det kan være svært at forestille sig hvad en bruger kan finde på at gøre med et program, men et godt program tager højde for (næsten) alt hvad en bruger kunne finde på. Systemet må aldrig ”gå ned”, og hvis brugeren gør noget programmet ikke kan håndtere, skal brugeren have at vide hvad der er galt på en måde så det er nemt for brugeren at reagere fornuftigt og gøre det på den rigtige måde.

## Eksempler på behov for robusthed

Her er nogle eksempler på ting en bruger kunne finde på at gøre, enten ved en tastaturfejl eller fordi brugeren ikke har gennemskuet hvad han/hun skal gøre:

* Brugeren kan skrive noget der ikke er tilladt i et felt, f.eks. skrive bogstaver i et felt beregnet til tal, benytte et forkert dato-format mv.
* Brugeren kan forsøge at slette fra en tom liste eller han kan trykke slet uden at have valgt noget der kan slettes.
* Brugeren kan være kommet til at trykke på slet ved en fejl. Så vil det være rart hvis han lige får en advarsel inden sletningen gennemføres.
* Brugeren kan undlade at udfylde felter han forventes at udfylde, eller han kan overse at han skal udfylde dem hvis der allerede står nogle værdier i felterne. Programmet skal sørge for at tømme felter når informationen fra felterne er brugt og systemet er gået videre.

## Tre muligheder for at skabe robusthed:

## 1. Løbende kontrol i koden, giv advarsler ved hjælp af GUIen:

Hvis man har en forventning om hvad brugeren kunne finde på at gøre, kan man hjælpe ham allerede i forbindelse med indtastning, f.eks. farve bogstaver røde hvis man forventer tal i et felt eller at en knap er dimmed indtil man har udfyldt de nødvendige felter. Udfordringen her kan være at gætte på forhånd hvad brugeren kan finde på at gøre med programmet.

## 2. Forbygge med betingelser:

Man kan også tjekke med en betingelse når man aktiverer en knap om den data man skal bruge er tilgængelig og tilladt.

## 3. try-catch

Instruktionerne try, catch og finally kan bruges sammen til at sikre at programmet ikke går ned. Udfordringen i catch-delen kan være at finde ud af hvad der egentlig gik galt. Til gengæld kan man med denne metode fange alle situationer der ellers ville kunne få programmet til at gå ned.

try-catch, som man ofte benævner disse instruktioner under et, er et meget stærkt værktøj til at sikre robusthed af sit program, det er meget fleksibelt.

Eksempel:

try

{

Bestilling bestilling = new Bestilling(cbxIs.SelectedValue as Vare, Antal());

BestillingsListe.Add(bestilling);

RefreshGui();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Fejl i bestilling");

}

Når koden ovenfor udføres, forsøger computeren (det er derfor det hedder try) at udføre instruktionerne i try-blokken. Hvis programmet forsøger at gøre noget der ikke giver mening *smider* systemet en såkaldt exception.

Exceptions fanges i catch-blokken. I koden ovenfor, udskrives en fejlmeddelelse hvis et eller andet går galt i forsøget på at oprette en ny bestilling. Man kan også vælge at afbryde programmet i catch blokken ([Eksempel](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/how-to-explicitly-throw-exceptions)).

Som nævnt findes også nøgleordet finally. finally bruges til at sikre at noget kode bliver afviklet både hvis operationen er gået godt eller der er opstået en fejl. Et eksempel kan være at lukke en fil som er blevet åbnet i try-blokken. Finally bruges ikke så meget som catch.

For mere baggrundsviden kig på [Microsofts dokumentation om exceptions](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/). Den indeholder blandt andet links til mere eksempler.