

Øvelse 21 - Exceptions i C sharp

Information

Formålet med denne øvelse er at introducere **exceptions i C#**.

Det er vigtigt, fordi I i de kommende øvelser blandt andet skal bruge exceptions til at opretholde objekters **invarianter gennem constructors**. Senere i forløbet arbejder vi mere nuanceret med exceptions i forbindelse med emnet **fejlhåndtering**.

En exception i C# afbryder programmets normale eksekveringsflow. Det gør exceptions til en meget tydelig eller "**højlydt**" **måde at signalere, at noget er gået galt**.

Hvis en metode eller constructor kaster en exception, kan den kaldende kode ikke blot fortsætte, som om alt var normalt. Exceptionen skal enten håndteres, eller også bevæger den sig videre op gennem programmets kaldestak.

Det kan være en fordel.

Hvis vi eksempelvis forsøger at oprette et objekt med data, der bryder objektets invariant, ønsker vi normalt ikke, at programmet stille og roligt opretter et ugyldigt objekt og fortsætter. I stedet kan vi **fejle hurtigt og tydeligt** (*fail fast*).

Exceptions i constructors

Exceptions giver særlig god mening i en **konstruktør**.

En konstruktør returnerer ikke en almindelig værdi, som kan bruges til at fortælle den kaldende kode, at oprettelsen mislykkedes. En almindelig metode kan eksempelvis returnere `false`, `null` eller et resultatobjekt, men en konstruktør har ikke et almindeligt returresultat.

Hvis de data, der sendes ind i constructoren, betyder, at objektet ikke kan oprettes i en gyldig tilstand, er en exception derfor en naturlig måde at afbryde oprettelsen på.

Princippet bliver dermed:

Enten bliver objektet oprettet i en gyldig tilstand, eller også bliver oprettelsen afbrudt tydeligt.

I næste øvelse skal I anvende denne tilgang til at håndhæve et objekts invariant gennem dets konstruktør.

I denne øvelse skal I først undersøge, hvad der sker, når en exception bliver kastet, håndteret og ikke håndteret.

Den [tutorial](#), du skal følge, kan findes [her](#).

Yderligere information om exceptions i C# kan du finde [her](#).

Instruktioner

1. Opret en ny .NET-konsolapplikation.
2. Følg og implementér tutorialen.
3. Prøv at oprette et `Circle`-objekt med en negativ radius.
4. Observer, hvad der sker, når exceptionen håndteres med `try` og `catch`.
5. Fjern derefter `try` og `catch`, og kørs programmet igen.

Når en exception ikke bliver håndteret, kaldes det en **uncaught exception**.

Læg især mærke til, hvor tydeligt programmet reagerer, når exceptionen ikke håndteres.

□ Refleksion: Hvorfor bruge exceptions?

Exceptions er ikke kun en mekanisme til at vise en fejlmeddelelse. En vigtig egenskab er, at en exception **afbryder det normale programflow**.

Det betyder, at en fejl ikke så let kan blive overset.

Forestil jer eksempelvis, at en constructor modtager data, som bryder objektets invariant. Hvis constructoren blot fortsætter, kan resten af programmet ende med at arbejde med et objekt i en ugyldig tilstand.

Ved i stedet at kaste en exception siger programmet i praksis:

Denne operation kan ikke fortsætte på en gyldig måde.

Det kaldes ofte **fail fast**.

Fejlen bliver synlig tæt på det sted, hvor den opstår, frem for at ugyldige data eller en ugyldig tilstand bevæger sig længere gennem systemet og måske først giver problemer langt senere.

Dette gør exceptions særligt velegnede til håndhævelse af **invarianter i constructors**. Constructoren har ikke et almindeligt returresultat, hvor den kan signalere, at oprettelsen mislykkedes. En exception kan derfor bruges til tydeligt at forhindre, at et ugyldigt objekt overhovedet bliver skabt.

Exceptions bør ikke bruges til alt

Det betyder dog ikke, at exceptions bør anvendes til alle situationer, hvor noget ikke går som forventet.

Der er forskel på:

- en situation, hvor programmets normale eksekvering **ikke bør kunne fortsætte**
- et **forventeligt udfald**, som er en normal del af programmets eller domænets funktion

Hvis en værdi eksempelvis aldrig må være negativ, for at et objekt kan eksistere i en gyldig tilstand, kan en exception være en tydelig måde at håndhæve denne regel på.

Hvis en operation derimod helt normalt kan lykkes eller mislykkes som en del af domænet, kan det i nogle designs være mere hensigtsmæssigt at modellere begge udfald eksplicit frem for at bruge exceptions som almindeligt kontrolflow.

Performance

At kaste og håndtere en exception er typisk dyrere end almindeligt programflow.

Det betyder ikke, at man bør undgå exceptions, når de er den rigtige måde at signalere en fejl på. Pointen er snarere, at exceptions normalt ikke bør bruges som erstatning for almindelige `if`-betingelser eller som en del af et hyppigt forventet kontrolflow.

Det vigtigste spørgsmål er derfor ikke blot, om en exception koster lidt mere performance, men:

■ **Er dette en situation, hvor den normale eksekvering bør kunne fortsætte?**

Brugervenlighed og sikker fejlhåndtering

Den "højlydte" egenskab betyder samtidig, at exceptions skal håndteres bevidst.

Hvis en exception ikke håndteres, kan programmet stoppe. Hvis den håndteres uhensigtsmæssigt, kan brugeren få en teknisk eller misvisende fejlmeddelelse.

En exception kan desuden indeholde information, som er nyttig for udvikleren, eksempelvis en stack trace, men som ikke nødvendigvis bør vises direkte til brugeren.

Senere i forløbet arbejder vi derfor mere med **sikker fejlhåndtering** og med, hvordan fejl kan håndteres uden at kompromittere systemets sikkerhed eller brugeroplevelse.

■ **Den vigtigste pointe i denne øvelse:**

En exception gør en fejl svær at ignorere. Det er særligt nyttigt, når vi vil forhindre, at et objekt bliver oprettet eller fortsætter i en ugyldig tilstand.

Refleksionsspørgsmål

1. Hvad sker der med programmets normale eksekveringsflow, når en exception bliver kastet?
2. Hvad er forskellen på en handled og en uncaught exception?
3. Hvorfor kan det være en fordel, at en exception "larmer"?
4. Hvorfor giver exceptions særlig god mening i en constructor?
5. Hvad kunne konsekvensen være, hvis en constructor accepterede data, der brød objektets invariant?
6. Hvornår kunne et forventeligt resultat være mere hensigtsmæssigt end at kaste en exception?
7. Hvorfor bør en teknisk exception ikke nødvendigvis vises direkte til brugeren?

Kobling til relevante standarder og principper

Exceptions og fejlhåndtering i softwareudvikling relaterer sig til flere centrale standarder og principper, som understøtter robust og sikker software:

Element i øvelsen	Relateret standard / princip	Kommentar
Exception handling	ISO/IEC 27002:2022, afsnit 8.26 (Applikations sikkerhed)	Håndtering af fejlmeddelelser.
	OWASP ASVS 5.0.0, Kontrol 16.5 (Error Handling)	Krav om korrekt håndtering af fejl og undtagelser.
	Secure by Design (Johnsson et al.)	Understreger betydningen af at håndtere fejl på en måde, der beskytter systemets integritet og understøtter <i>fail fast</i> .
Performance og brugeroplevelse	Secure by Design	Exceptions skal anvendes bevidst, og fejlhåndtering bør tage hensyn til både systemets tilstand og brugeroplevelsen.

▮ **Korrekt brug af exceptions kan hjælpe med at forhindre ugyldige tilstande og gøre fejl tydelige tidligt. Senere skal exceptions samtidig håndteres på en måde, der**

understøtter sikker, robust og forståelig fejlhåndtering.

Last update: 2026-09-25 11:22:22