Chapter 8 - Handling Failure

Ett program är inte komplett om det inte kan hantera alla möjliga fel. Resultatet av en system-operation ska aldrig vara odefinierad. Exceptions löser ofta problemet.

8.1 UML

[Fig 8.2a] Definierar en Constraint, dvs ett villkor eller en begränsning till elementet. Innehållet är en text kan vara vad som helst.

[Fig 8.2b] Annat sätt att visa exceptions i ett klassdiagram, ser dock ej vilken metod som kastar exceptions.

[Fig 8.3a] Visar exceptions i ett SSD

Observera "enkel pilen" ← Asynkront meddelande

Och det extra strecket vid blå pil.

[Fig 8.36] Kommunikationsdiagram, kan ej visa asynkrona meddelanden i Astah. Därför används en "vanlig pil" ←

I båda fallen är det () efter, ska eg. ej vara det.

8.2 – Exception handling best practice

Always use exceptions for error handling

Rapportera aldrig ett fel med ett tal som returvärde. Oklart vad talet representerar, grötigt med många if-statements.

Use exceptions only for errorhandling.
Kasta ej undantag om inget gått fel.
Förbättra metoder istället.

Checked or Unchecked exception?

Checked exceptions – endast för business logic.
Dessa indikerar ej att programmet har kraschat.
Ex) Försöker ta ut mera pengar än man har.

Unchecked exceptions – Programmerings fel
Ex) NullPointerException.

Name the exceptions after the error condition

Ska beskriva vad som gick fel, Enligt konvensionen ska den sluta med Exception

Har man många exceptions → Stort publikt gränssnitt → dåligt.

Därför bör man ha något mellan ting och förklar i felmeddelandet vad som gick fel.

Include information about the error condition

Användbart för både debuss och generca felmeddelanden

(Exception är själv ett objekt)

Det finns två typer av info som ska vara inkluderad i exception.

- Datan som utlöste exception
- String med felmeddelandets ej menat för end user, endast för utvecklare/admin

Use functionality provided in java.lang.Exception

Alla exceptions måste ärva från java.lang.Exception

Den har en konstruktor som tar imot ett meddelande samt en getMessage metod.

Använd dessa, skriv ej egna.

Use the correct abstraction level for exceptions.

Ett low-level-layer (typ från view) kastar ett undantag, då ska ej view se det-

tex) SQL error.

Bättre att kasta generiska felmeddelande ju närmare view man kommer.

Write Javadoc comments for all exceptions.

Lägg alltid till en Javadoc kommentar när du adderar ett exceptions. @throws

An object shall not change state if it throws an exception.

Ett objekt ska kunna användas efter ett exception har kastats.

- Tex immutable objekt, den kan aldrig ändra tillstånd, alla fält måste vara final

Får ej vara möjligt att ärva.

- Check parameter values before changing any state

Vanlig anledning till att ett exception uppstår är att ett illegalt parameter värde.

Kolla därför värdet först!!!

- Place operations that might fail before operations that alter the state.

Om man inte kan validera parametrarna utan att utföra metoder.

- Use a temporary copy of the state.

Om inga andra strategier fungerar, skapa kopia, återställ innan man kastar exception.

Never write an empty catch block.

Ett tomt catch block, då ignoreras exception helt.

How to notify the user?

Användarens gränssnitt skall alltid reflektera tillståndet av systemet. Om användaren gör en System-operation som inte framgångsrikt genomförs, måste användar gränssnittet visa det. Felmeddelande ska visa info som är relevant för användaren.

How to notify developers and/or administrators?

Admins måste förstå vad som händer, utvecklare måste kunna hitta buggar. Detta kräver mera detaljerade rapporter, som skrivs till the log tex en text fil.

API java.util.logging

Write unit tests for the exception handling.

Exception måste också testas, testa så framåtriktad exekvering ej löser ut exceptions.

Testa att exceptions löser ut på fel och att catch-blocke hanterar dom rätt.

8.2 OK