



# Håndtering av unntak og transaksjoner i J2EE miljø

---

**Steinar Overbeck Cook**  
**Balder Programvare AS**



# Håndtering av unntak og transaksjoner i J2EE miljø

- **Agenda**

- Feilen oppstår eller oppdages
- Behandling av feilen
- Hvordan fungerer dette I J2EE sammenheng
- Regler for bruk av unntak



## Hvor oppstod feilen?

- Hvis unntaks-objektet har blitt opprettet annet sted vil det inneholde informasjon om:
  - Java klasse
    - Av pakkenavnet kan du utlede subsystem osv.
  - Metode hvor feilen ble oppdaget
  - Linjenummer hvor feilen ble oppdaget. Dvs. linjen hvor du utførte "throw new Exception" via "stack trace"
    - OBS! kun hvis "debug" flag er satt
- Hvis det er du som oppdager feilen og avviket – vet du jo hvor du er!
  - Eksempel: en variabel inneholder ugyldig verdi



## Hvordan vet vi at det er en feil?

- `java.lang.Throwable` fra subsystem - f.eks. `SQLException`, `NullPointerException` osv.
  - unntak som må "fanges" ("checked")
  - unntak som kommer som lyn fra klar himmel, f.eks. `NullPointerException` ("unchecked")
- Brudd på forretningsregler
  - Ugyldige data
  - Mangler autorisasjon osv.

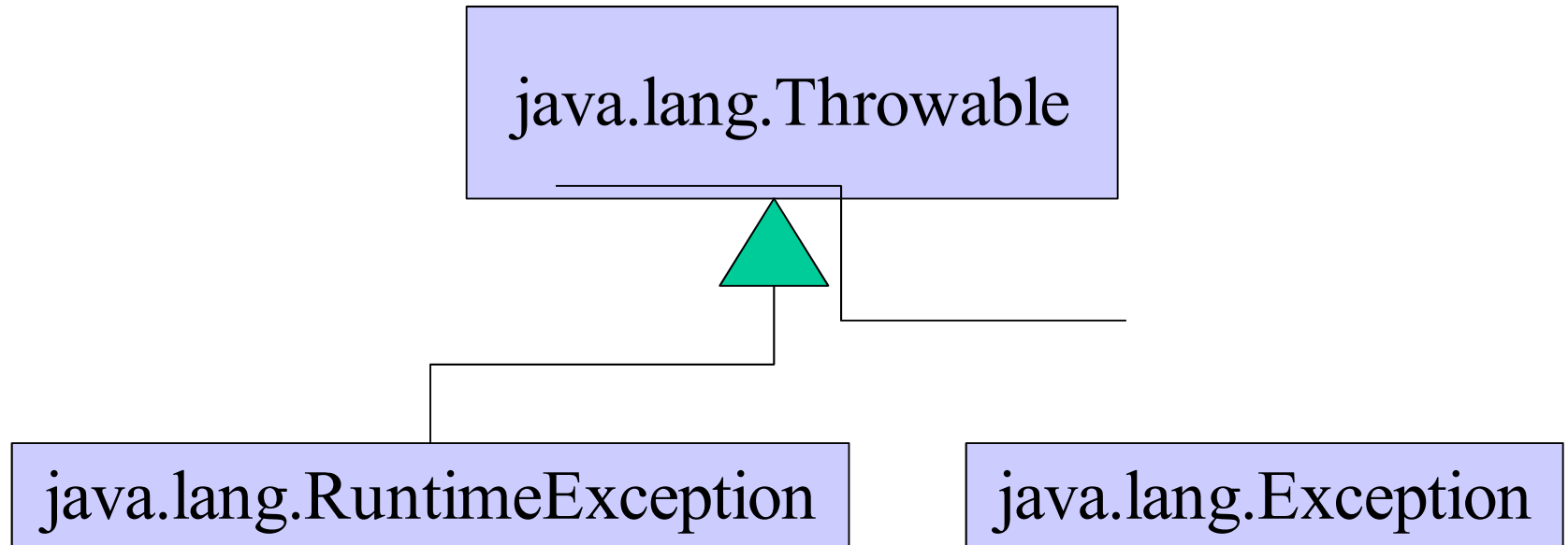


## Sammenheng - kontekst?

- I hvilken sammenheng oppstod feilen?
  - `java.lang.NullPointerException` - hva var det vi gjorde, hvor var i koden?
  - Hvilken database tabell arbeidet vi mot?
  - Hvilken forekomst i tabellen?
  - Hvilket brukstilfelle, hva gjorde vi?



# Unntakstyper



Blir automatisk sendt videre til anropende metode helt til Noen "fanger" unntaket eller til vi er ute

Må deklarerer i metode signaturen:

```
public void metode()  
throws DingException {
```



# Behandling

- Kan feilen rettes her og nå?
- Hvis unntak skal kastes
- Kan vi kaste problemet videre?



## Kan feilen rettes av en eller annen?

- Standardregel for behandling:
  - Hvis Nei det er system- eller programmeringsfeil, vi kaster: unntak som arver fra `java.lang.RuntimeException`
  - Hvis Ja;
    - Ja, men ikke her: kast unntak som arver fra `java.lang.Exception`
    - Ja, rett opp og prøv igjen
- Feil som kan rettes:
  - Brukeren har skrevet inn ugyldige data og kan bare skrive dem inn på nytt.



## Innsamling av informasjon

- For å opprette en meningsfull unntaksmelding trenger vi å vite sammenhengen som unntaket oppstod i:
  - Bakenforliggende exception
    - RemoteException – indikerer et “unchecked Exception” i en annen JVM (for eksempel i J2EE)
  - Kontekst - hva holdt vi på med, kundenummer, etc.
  - Har ikke alltid totaloversikten der hvor feilen oppstår?
  - Årsakssammenheng - `exception.getCause()`



## Kan vi kaste unntaket videre?

- Hvis Nei, den må presenteres sluttbehandles og presenteres for brukeren:
  - Skriv til feillogg
  - Kan brukeren rette det opp?
    - Ja; la brukeren rette opp feilen
    - Nei; vis relevant feilmelding og avslutt
- Hvis Ja, kast den videre



# Sluttbehandling i forskjellige miljøer

- **Swing GUI klient**
  - Vis feilmelding
  - Skriv til logg-fil
- **Struts Action**
  - Deklarer feilhåndtering i struts-config.xml
  - Varsle bruker vha. feil-visnings-side
    - Skriv til logg-fil
  - Skriv til logg-fil
- **JSP**
  - Deklarer feilhåndtering i web.xml
  - Varsle bruker vha. feil-visnings-side
- **Message Bean o.l.**
  - skriv til logg-fil



## Vi kaster ”apekatten” videre

- **Kast ny Throwable som er:**
  - Arvet fra `java.lang.RuntimeException` hvis feilen ikke kan rettes opp av oss eller brukeren
  - Arvet fra `java.lang.Exception` hvis feilen kan rettes opp
  - Hvis vi er en EJB i J2EE:
    - Feilen kan rettes:
      - Anrop `setRollbackOnly()` hvis vi er del av en transaksjon
      - Evt. håndter ”rollback” vha. `afterCompletion(false)`
      - Deretter skal du enten:
        - Pakk inn unntak på et høyere abstraksjonsnivå;
        - Eller la den slippe videre fordi du har ”Throws i signatur”
    - Feilen kan ikke rettes:
      - Kast videre evt. med endringer i abstraksjonsnivå



# Når du kaster ny "Throwable"

- Inkluder evt. forårsakende Throwable
  - Standard i JDK 1.4.x
  - Må implementeres på egenhånd i utgaver før JDK 1.4.x
- Kontekst
  - I18N – unngå bruk av språkavhengig tekst i unntakene. Det skal oversettes der hvor feilen behandles.
  - Unntaket sier noe om kontekst:

```
KundeIkkeFunnetException (KundeNummer kundeNummer)
```



## Abstraksjonsnivå

- Høyere lag bør fange unntak fra lavere lag og kaste disse på nytt med en forklaring som passer i høyere lag.
- Pass på å inkludere opprinnelig unntak når nytt unntak konstrueres.
- Organiser kontrollerte unntak ("checked exception") slik:
  - Hver java pakke har sin egen Exception-klasse som andre i samme pakke kan arve fra.
    - Arver igjen fra global exception for hele applikasjonen
  - IKKE legg kontekstinformasjon som fri-tekst, bruk egne felter.
    - Kundenummer, ordrenummer osv.



# Håndtering av unntak i J2EE?

- Entitetsbønner
  - RuntimeException
    - TX rulles automatisk tilbake
  - Exception
    - setRollbackOnly() hvis TX skal rulles tilbake
- Sesjonsbønner
  - Game over: RuntimeException
    - Fremkommer som RemoteException i klienten
    - EJB kan ikke anropes på nytt
  - Kan anropes på nytt: Exception
    - EJB kan anropes på nytt



## Typer av TX håndtering i J2EE

- **Container Managed Transaction - *CMT***
  - Brukes alltid av entitets bønner
  - Kan brukes av sesjon og meldings bønner
  - Anbefales!
- **Bean Managed Transaction - *BMT***
  - Message beans
  - Session beans
    - Statefull
    - Stateless
  - Entity beans - kan ikke bruke BMT, kun CMT
  - Kun for spesielt interesserte!



## Transaksjonsattributter i ejb-jar.xml

- *NotSupported* -eventuell TX suspenderes inntil metoden med denne attributten har kjørt ferdig
- *Supports* – kan kjøre i en TX som andre har startet. Starter ikke egen TX
- *Required* – krever å kjøre som del av en TX, starter evt. egen TX
- *RequiresNew* – starter egen TX uansett
- *Mandatory* – må være med i TX som andre har startet
- *Never* – EJB'en må ikke være med i TX



# Håndtering av forskjellige unntak i J2EE

- System unntak - krever ikke deklarasjon.
  - Alle unntak som arver fra `java.lang.RuntimeException`
    - `javax.ejb.EJBException` arver fra `java.lang.RuntimeException`
  - Oppstår som `RemoteException` på klient-siden
- Application Exception - krever deklarasjon i metode signatur
  - Alle unntak som arver fra `java.lang.Exception`



## System unntak - krever ikke deklarasjon

- Alle arver fra `java.lang.RuntimeException`
- Transaksjonen rulles tilbake automatisk
- Exception pakkes inn i en `TransactionRolledBackException`, som igjen ....
- ... oversendes til anropende metode innpakket i `java.rmi.RemoteException`
  - OBS! → `RemoteException` krever deklarasjon ←
- `javax.ejb.EJBException` - kan du bruke for å signalisere systemfeil, f.eks. `SQLException`
- Alle EJB'er som deltok i transaksjonen invalidiseres
- Loggføring automatisk av J2EE server iht. spec'en



# Applikasjonsunntak - krever deklarasjon i metode signatur

- Arver fra `java.lang.Exception`
- Transaksjonene rulles ikke tilbake automatisk
- Exception oversendes til anroper uten å bli pakket inn
- Bruk `ctx.setRollbackOnly()` når du vil tilbakeføre transaksjonen samtidig som applikasjons unntak skal kastes
- Hvis du vil ha detaljert styring i "session" EJB'er:
  - Implementer `javax.ejb.SessionSynchronization` grensesnittet



## Noen grunnregler

- Ikke bruk unntak for å styre logikk i programmet
- Kontrollerte unntak for gjenopprettlige feil, ellers ukontrollerte unntak
- Unngå unødig bruk av kontrollerte unntak
- Bruk standard unntak der hvor det er mulig (ikke lag egne hele tiden)
- Kast unntak iht. abstraksjonsnivå
- Dokumenter alle unntak, også de udeklarete.
- Inkluder verdien til alle parametere som forårsaket unntaket
- Etterstreb atomitet – etterlat objekt i stabil tilstand
- Ikke ignorerer unntak



# Ikke bruk unntak for å styre logikk i programmet

**// Fryktelig bruk av unntak. Ikke gjør dette!**

```
try {  
    int i = 0;  
    while (true)  
        a[i++].f();  
} catch (ArrayIndexOutOfBoundsException e) {  
    //  
}
```



## Kan feilen rettes?

- Bruk unntak arvet fra `java.lang.Exception` for tilfeller hvor man med rimelighet antar at anroper kan gjenopprette feilen.
  - Brukeren skriver inn data på nytt
  - `SQLException` som egentlig angir at entiteten allerede finnes
    - DBMS avhengig
- Ellers; bruk unntak arvet fra `java.lang.RuntimeException`
  - For eksempel ved `SQLException` som er forårsaket av programmeringsfeil



## Unngå unødig bruk av kontrollerte unntak

```
} catch (TheCheckedException e) {  
    e.printStackTrace(); // Du sliter!  
    System.exit(1);  
}
```

Hvis dette er den beste håndteringen du klarer,  
bruk `avvik` som arver fra  
`java.lang.RuntimeException`



## Bruk standard unntak, eks:

Unntak	Bruksområde
<code>IllegalArgumentException</code>	Verdien av overførte parametere var ugyldige.
<code>IllegalStateException</code>	Objektet er i en ugyldig tilstand for dette metodekallet
<code>NullPointerException</code>	Parameterverdi er null
<code>UnsupportedOperationException</code>	Objektet støtter ikke denne metoden



## Kast unntak iht. abstraksjonsnivå

```
// Oversetting av unntak:  
try {  
    // bla bla bla resten står i ukebla'  
    ...  
} catch (LowerLevelException e) {  
    throw new HigerLevelException(..., cause);  
}
```



## Gode råd på veien

- Organiser unntak pr. java "package"
- Ikke tolk teksten i unntaket
- Bruk navn på unntak til å oversette til mennesketekst:
  - UkjentKundeException → "Kunde {0} er ukjent"



# Avvikshåndtering i J2EE

Programmering med Exception i Java

## Flytkart for unntak

