Objektorienterad Programmering (TDDC77)

Föreläsning XIV: Undantag, Design

Ahmed Rezine

IDA, Linköpings Universitet

Höstermin 2016

Outline

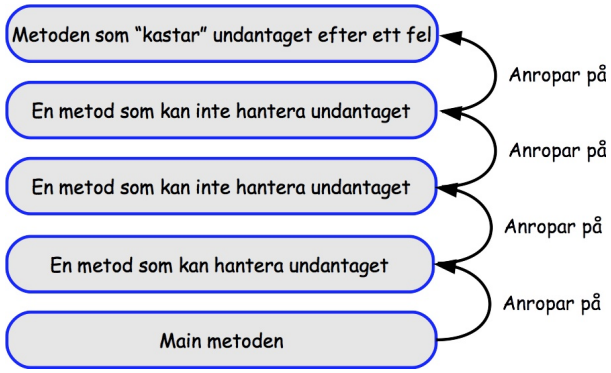
Undantag

Design

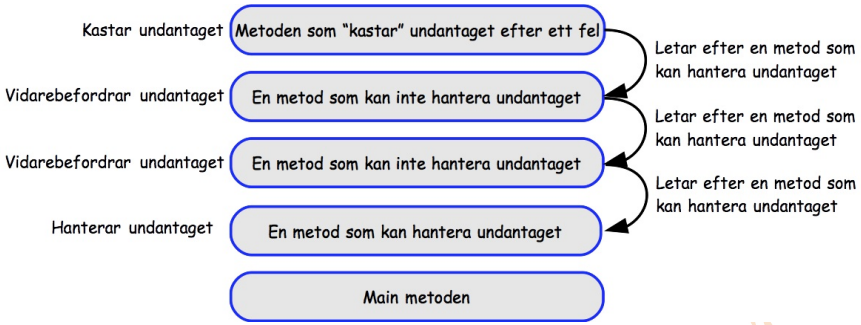
Undantag - Exceptions

- Fel som uppstår under körning kallas undantag (exceptions)
- Metoder kan kasta undantag (throw)
- Metoder kan fånga undantag (catch)

Undantag - Exceptions



Undantag - Exceptions

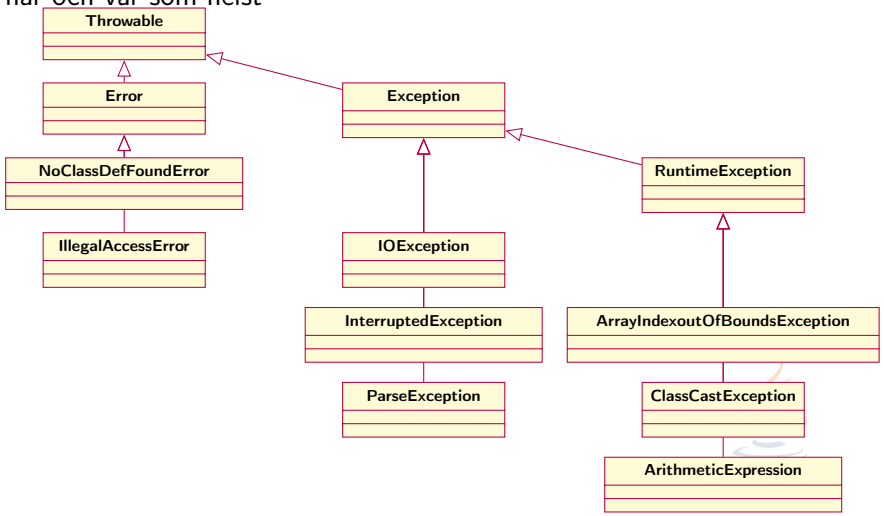


Kasta undantag

```
class DateParsing{

    private static int getMonth(String input) throws DateFormatException {
     if(input.length()!=10)
         throw new DateFormatException("Incorrect date format, should be dd/mm/yyyy");
     return Integer.parseInt(input.substring(3,5));
    }

    private static String printMonth(String date) throws DateFormatException
    {
     int month = 0;
     //try{
     month = getMonth(date);
     /*   } catch(DateFormatException e){
         System.out.println("ett datum!!!");
         }*/
     switch(month){
     case 1: return "January";
     case 2: return "February";
     case 3: return "March";
     default: return "December";
     }
    }

    public static void main(String[] args) {
     //try{
         System.out.println("The month in: " + args[0] + " is " + printMonth(args[0]));
     // } catch(DateFormatException e){
     //     System.out.println("Vnligen skriv ett datum!");
     // }
    }
}
```

Typer av undantag

Vissa undantag måste man förvarna för, medan andra kan uppstå när och var som helst



Typer av undantag

- **Throwable** - Allt som kan kastas och fångas
- **Errors** - Jättealvarliga fel i själva java. Normalt sett bör dessa inte hanteras av programmeraren.
- **Exceptions** (utom **RuntimeException**) - Förutsedda undantag, dessa är del av normal funktion och måste hanteras
- **RuntimeExceptions** - Undantag som kan inträffa under normal körning, men normal sett beror på att programmeraren gjort fel. Behöver inte fångas

Typer av undantag

- Checked exception:
  - e.g. **Exception, ParseException, IOException,** ...
  - Måste hanteras (dvs. antigen med **try-catch** eller genom att skriva **throws** i metodens deklaration)
  - När undantaget förväntas uppstå
  - När det går att rädda systemet
- Unchecked exception:
  - e.g. **RuntimeException, ArrayIndexOutOfBoundsException,** ...
  - Måste inte hanteras ((dvs. behöver inte ha en **try-catch** eller skriva **throws** i metodens deklaration))
  - När undantaget inte förväntas uppstå
  - Men ifall det ske, "Something went horribly wrong!"

Egna undantag

```
class DateFormatException extends Exception{
    public DateFormatException (String message){
     super(message);
    }
}
```

Kasta vidare undantag

- En kedja av metoder kan ha throws i sin signatur
- Lämnar över ansvaret att fånga högre upp i anroskedjan

Stack trace

```
Exception in thread "main" java.lang.NumberFormatException: For input st
     at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.ja
     at java.lang.Double.parseDouble(Double.java:510)
     at lincalc.LinCalcJohn.calc(LinCalcJohn.java:139)
     at lincalc.LinCalc.calc(LinCalc.java:35)
     at lincalc.LinCalc.evaluate(LinCalc.java:45)
     at lincalc.LinCalc.main(LinCalc.java:28)
```

```java
class Undantag {

  public static void main(String[] args) {
    System.out.println("Prog börjar");
    foo1();
    System.out.println("Prog slutar");
  }

  public static void foo1(){
    System.out.println("foo1 börjar");
    try{
     foo2();
    }
    catch (ArithmeticException e){
      System.out.println("foo1 catch1");
    }
    catch(NumberFormatException e){
      System.out.println("foo1 catch2");
     }
    finally{
      System.out.println("foo1 finally");
    }
    System.out.println("foo1 slutar");
  }


  public static void foo2(){
    System.out.println("foo2 börjar");
    try{
      System.out.println(Integer.parseInt("a"));
    }
    catch(ArithmeticException e){
      System.out.println("foo2 catch1");
    }
    /*catch(NumberFormatException e){
      System.out.println("foo2 catch2");
      }*/
    finally{
      System.out.println("foo2 finally");
    }
    System.out.println("foo2 slutar");
```

Outline

Undantag

Design

Design

- Klasser: substantiv, t.ex. Item, DatabaseInitializer
- Gränssnitt: brukar beskriver en egenskap, adjektiv, t.ex. Serializable, Cloneable
- Metoder: verb, t.ex. calculateSum(), initialize()
- Metoder som returnera ett booleskt värde: fråga, t.ex. isBiggerThan(), hasElement()
- Metoder som hämtar eller sätter en attribut, börja med get, respektive set.

Isberg

- Man brukar liknar en väldesignad klass med ett isberg: 80% av massan ligger dold under vattenytan,
- På samma sätt bör klasser bara ha några få synliga metoder

En uppgift

"En klass ska göra en sak, och göra den bra"

- Man ska alltså undvika att ha med funktionalitet som inte logiskt hör ihop med klassen

Undvika överraskningar

"Principle of least surprise"

- Om någonting kan lösas på flera olika sätt, välj den lösning som skulle överraska användaren minst

Dokumentera alltid

- Dokumentera alltid alla publika metoder och variabler
- Normal sett bör man även dokumentera privata metoder och variabler

Bottom up

- Börja bygga från grunden
- Man bygger flera smådelar som successivt sammanfogas till ett större system
- Risk för att helheten blir virrig och "ogenomtänkt"
- Men ofta kan man få körbar, (delvis) fungerande kod snabbt

Top down

- Börja från ett tänkt färdigt system
- Bryt ned systemet i fler mer detaljerade delar
- Risk för att man gör uppdelningar som inte blir bra när man skriver koden
- men ofta enklare att genomföra systematiskt

Båda

- Vilken man väljer är helt upp till en själv
- Vissa gillar att blanda

Hierarkier

- En hierarki är en ordnad samling av saker där vissa har högre "position" än andra

Klasshierarkier

- Se till att klasserna till så stor utsträckning som möjligt motsvarar verkliga saker i världen, även abstrakta klasser
- För varje subklass, fråga dig själv: "är *subklass* verkligen en *basklass*" ?

Typer

- Deklarera alltid variabler till den mest generella typ som kan användas
- Föredra alltså List framför ArrayList