



Java

TDDC77

Objektorienterad Programmering

Föreläsning 10

Sahand Sadjadee
IDA, Linköpings Universitet

Outline

- Felhantering(Exception Handling)

Felhantering

Felhantering

- Ett program förväntas att fungera utan problem mest av tiden.
- Om något går fel under körtiden(runtime) då kastas ett exception.
- Ett exception är en händelse/ett fel under körningstiden som avbryter exekveringen.
- Ett exception kan också kastas av/från egen kod eller av/från en metod som vi inte har skrivit själva. Till exempel: `nextInt()` kan kasta `InputMismatchException`
- Alla fel kan **fångas** och **hanteras** med `try/catch`-konstruktionen.
- Om ett exception kastas och inte fångas då programmet avslutas okontrollerat/kraschar.

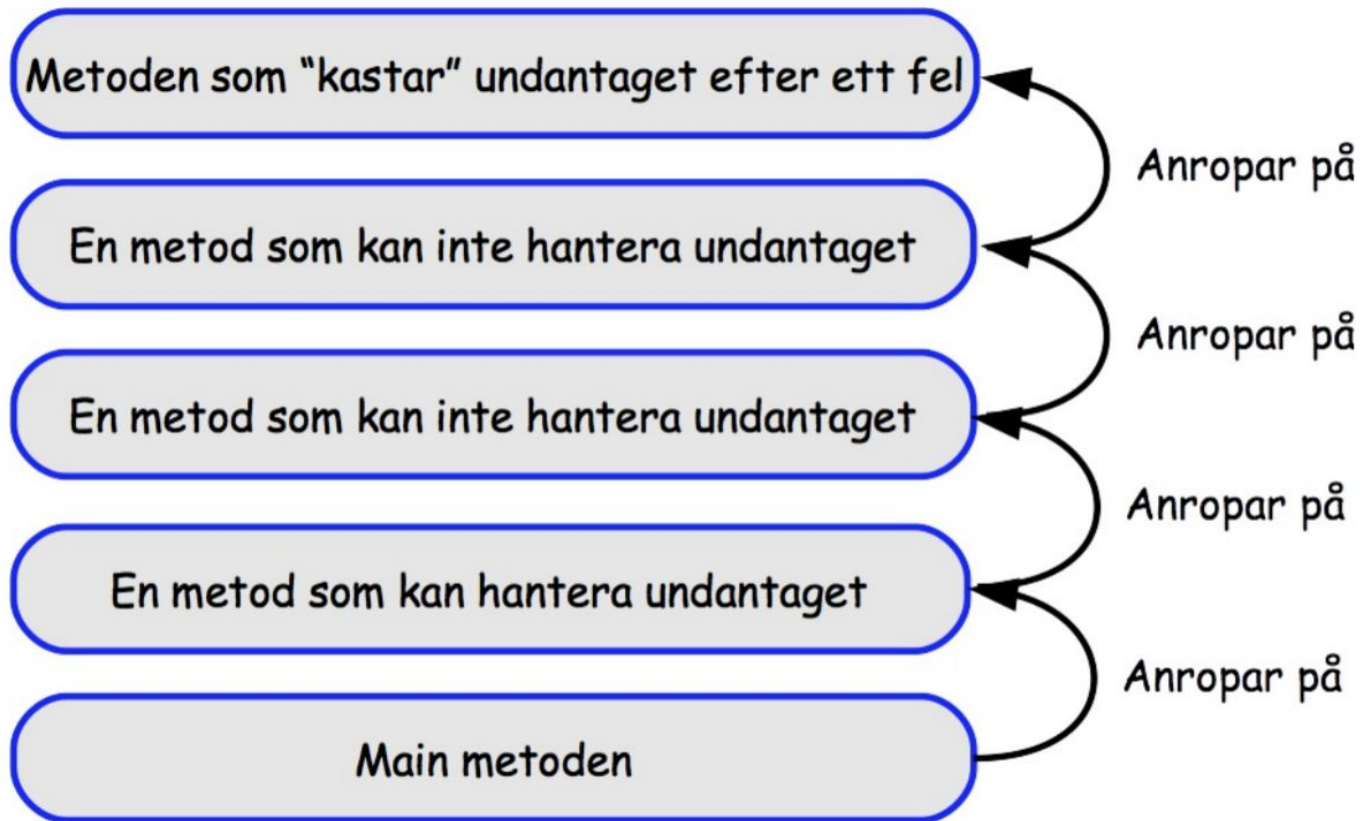
Syftet med felhantering

1. Fånga alla möjliga exception.
2. Lösa/hantera problemet eller avsluta programmet på ett kontrollerat sätt.

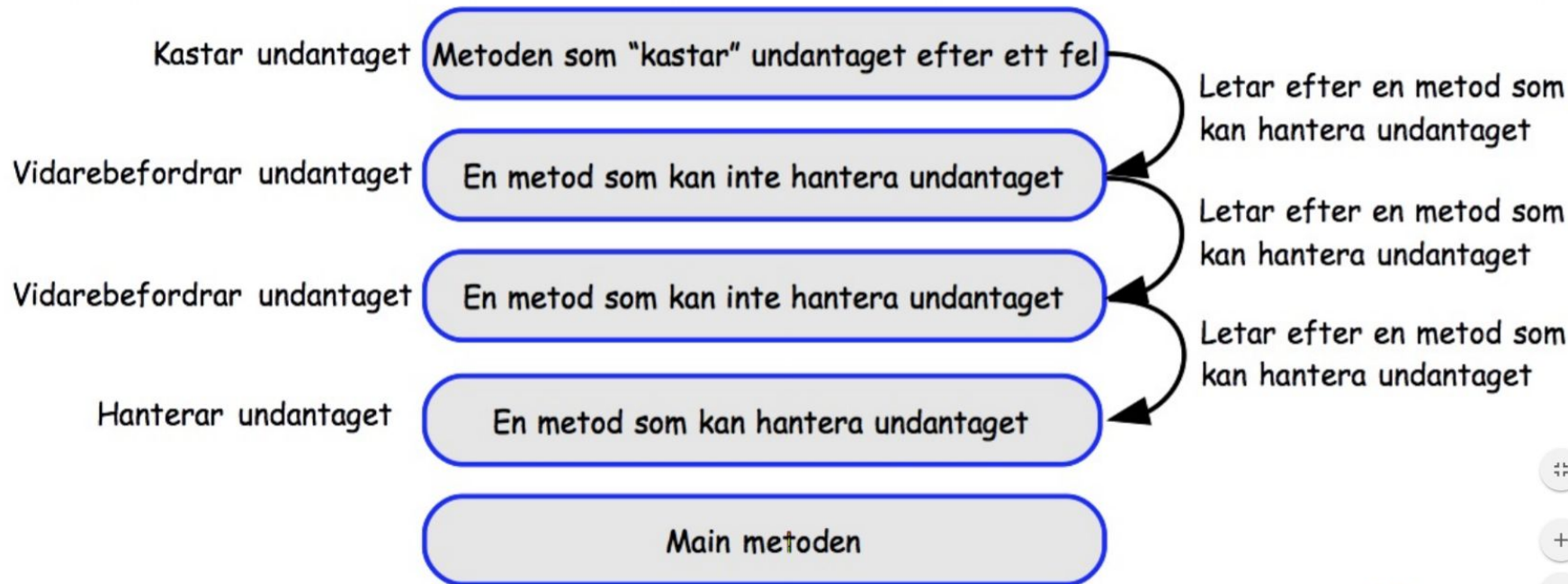
Undantag - Exceptions

- Metoder kan kasta exception (throw)
- Metoder kan fånga exception (catch)

Undantag - Exceptions



Undantag - Exceptions



StackTrace

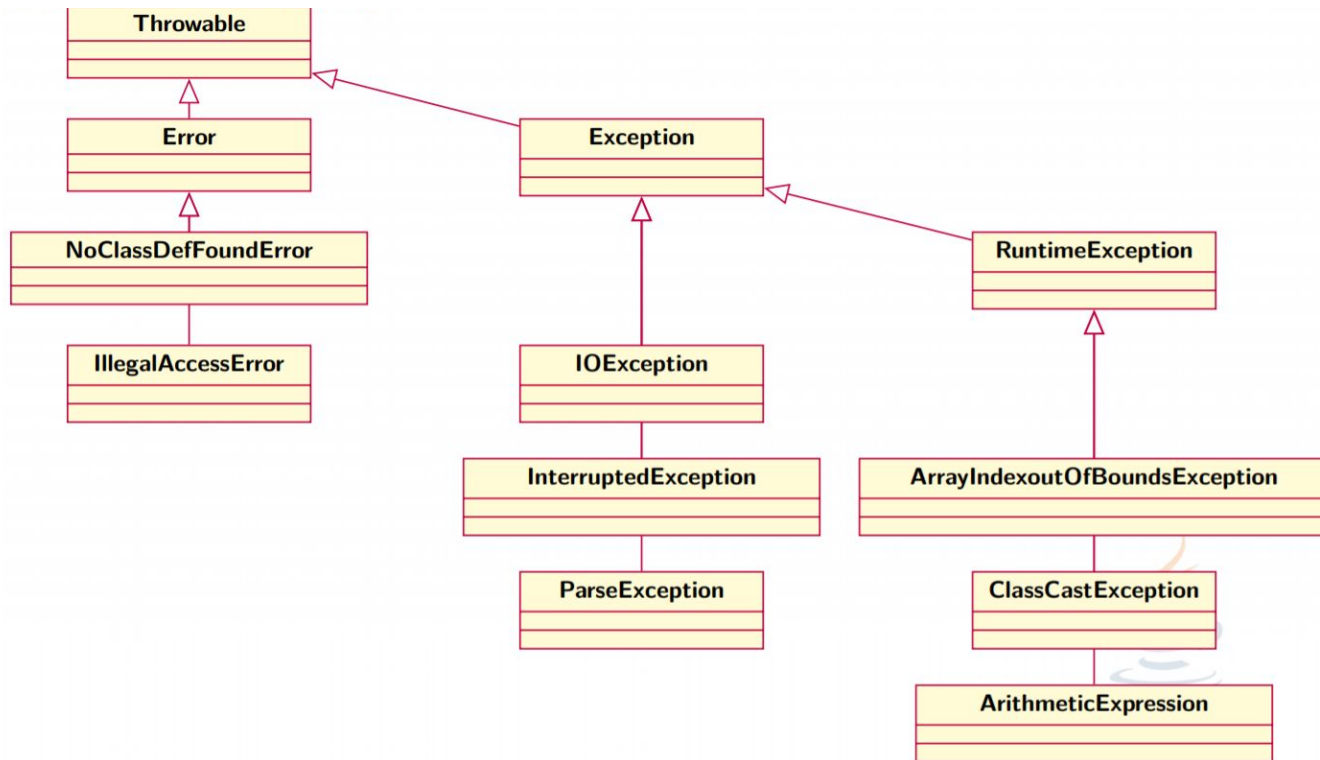
- Stack trace visar alla anrop från `main` fram till metoden där felet uppstått.
- Om ett undantag inte fångas i ingen av metoderna då programmet kraschar och Stack trace visas automatiskt.
- Stack trace skrivs inte ut automatiskt om undantaget fångas någonstans i programmet.
- Man kan dock skriva ut stack trace manuellt genom att anropa metoden `printStackTrace` på `exception:et`.

Stack trace

```
Exception in thread "main" java.lang.NumberFormatException: For ir
    at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDeci
    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)
```

Typ av undantag

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



Typer av undntag

- **Throwable** - Allt som kan kastas och fångas
- **Errors** - Jätte allvarliga fel i själva java. Normalt sett bör dessa inte hanteras av programmeraren.
- **Exceptions** - Undantag som kontrolleras vid kompilering(checked).
- **RuntimeExceptions** - Undantag som inte kontrolleras vid kompilering(unchecked).

java.lang.Exception klassen

String **getMessage()**

Returns the detail message string of this throwable.

StackTraceElement[] **getStackTrace()**

Provides programmatic access to the stack trace information printed by **printStackTrace()**.

Void **printStackTrace()**

Prints this throwable and its backtrace to the standard error stream.

Typ av undantag

- Checked exception:
 - e.g. Exception, ParseException, IOException, ...
 - Måste hanteras (dvs. antingen med try-catch eller genom att skriva throws i metodens deklaration)
- Unchecked exception:
 - e.g. RuntimeException, ArrayIndexOutOfBoundsException, ...
 - Måste inte hanteras ((dvs. behöver inte ha en try-catch eller skriva throws i metodens deklaration))

Vanliga undantag

Unchecked Exception List

- `ArrayIndexOutOfBoundsException`
- `ClassCastException`
- `IllegalArgumentException`
- `IllegalStateException`
- `NullPointerException`
- `NumberFormatException`
- `AssertionError`
- `ExceptionInInitializerError`
- `StackOverflowError`
- `NoClassDefFoundError`

Checked Exception List

- `Exception`
- `IOException`
- `FileNotFoundException`
- `ParseException`
- `ClassNotFoundException`
- `CloneNotSupportedException`
- `InstantiationException`
- `InterruptedException`
- `NoSuchMethodException`
- `NoSuchFieldException`

try/catch

- De kodraderna som är misstänkta att kasta exception ligger i try-blocket.
 - `try {`
 - `}`
- Det ska finnas ett catch-block för varje misstänkt exception.
 - `catch (IndexOutOfBoundsException ex) {`
 - `}`
- Exceptionet `Exception` används för att fånga alla möjliga exception. Catch-blocket för `Exception` skrivs sist efter alla andra catch-block.
 - `catch (Exception ex) {`
 - `}`
- När ett exception kastas slutas exekveringen i try-blocket och fortsätter i catch-blocket som matchar kastade exceptionet.
- Om det är inget catch-block som matchar exceptionet då exekveringen i metoden slutar och anropande metoden blir ansvarig att hantera exceptionet.

Kasta vidare undantag

- Metoder kan ha `throws` i sin signatur.
- Lämnar över ansvaret till metoden som ligger högre upp i anrops kedjan att fånga och hantera `exception:et`.

Egna undantag

Kan man skapa egna exception för fel som är specifika i eget program?

Egna undantag

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

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 = getMonth(date);
        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(e.getMessage());
        }
    }
}
```



Läs in heltal

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter an integer: ");  
int number = input.nextInt();  
System.out.println("You just entered " + number);
```

- Fungerar för det mesta, men vad händer om man matar in "2.5" eller "e" ?

Läs in heltal med felhantering

```
Scanner input = new Scanner(System.in);  
try {  
    System.out.print("Enter an integer: ");  
    int number = input.nextInt();  
    System.out.println("You just entered " + number);  
}  
catch(InputMismatchException ex) {  
    System.out.println("You entered wrong type of data!");  
}
```

Läs in heltal med felhantering

```
Scanner input = new Scanner(System.in);  
boolean tryAgain;  
do {  
    try {  
        System.out.print("Enter an integer: ");  
        int number = input.nextInt();  
        System.out.println("You just entered " + number);  
        tryAgain = false;  
    }  
    catch(InputMismatchException ex) {  
        System.out.println("You entered wrong type of data!");  
        input.nextLine();  
        tryAgain = true;  
    }  
}while(tryAgain);
```

Hantera felet `ArrayIndexOutOfBoundsException`

```
int[] grades = {3, 4, 4, 5, 3, 5, 5, 4, 5, 3};  
Scanner input = new Scanner(System.in);  
System.out.println("Enter an index(0-10)");  
int index = input.nextInt();  
System.out.println("Your grade is " + grades[index]);
```


Hantera felet `ArrayIndexOutOfBoundsException`

```
int[] grades = {3, 4, 4, 5, 3, 5, 5, 4, 5, 3};
Scanner input = new Scanner(System.in);
System.out.print("Enter an index(0-10): ");
int index = input.nextInt();
try {
    System.out.println("Your grade is " + grades[index]);
}
catch (IndexOutOfBoundsException ex) {
    System.out.println("You entered a wrong index!");
}
```

Hantera både `ArrayIndexOutOfBoundsException` och `NumberFormatException`

```
int[] grades = {3, 4, 4, 5, 3, 5, 5, 4, 5, 3};
Scanner input = new Scanner(System.in);
System.out.print("Enter an index(0-10): ");

try {
    int index = input.nextInt();
    System.out.println("Your grade is " + grades[index]);
}
catch (NumberFormatException ex) {
    System.out.println("You entered wrong a type of data!");
}
catch (IndexOutOfBoundsException ex) {
    System.out.println("You entered a wrong index!");
}
```

Hantera felet ArithmeticException

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter the first operand(integer): ");  
int op1 = input.nextInt();  
System.out.print("Enter the second operand(integer): ");  
int op2 = input.nextInt();  
System.out.println(op1 + "+" + op2 + "=" + (op1 + op2));  
System.out.println(op1 + "-" + op2 + "=" + (op1 - op2));  
System.out.println(op1 + "*" + op2 + "=" + op1 * op2);  
System.out.println(op1 + "/" + op2 + "=" + op1 / op2);
```

Hantera felet ArithmeticException

```
Scanner input = new Scanner(System.in);
System.out.print("Enter the first operand(integer): ");
int op1 = input.nextInt();
System.out.print("Enter the second operand(integer): ");
int op2 = input.nextInt();
System.out.println(op1 + "+" + op2 + "=" + (op1 + op2));
System.out.println(op1 + "-" + op2 + "=" + (op1 - op2));
System.out.println(op1 + "*" + op2 + "=" + op1 * op2);
try {
    System.out.println(op1 + "/" + op2 + "=" + op1 / op2);
} catch (ArithmeticException ex) {
    System.out.println("Op2 can't be 0!");
}
```



Thanks for listening!