Läsanvisningar

Börja med att läsa kapitlen i kursboken, därefter resurserna på Internet och till sist själva lektionsmaterialet.

Kursboken (5-8:e upplagan): Kapitel 11

Internet: The Java Tutorial, Trail: Essential Classes: - Exceptions (http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html)

Exceptionella händelser

Det kommer att komma tillfällen då ditt program råkar ut för olika typer av fel. Allt ifrån enkla fel som editorn påpekar till att ditt program kraschar helt och hållet under körningen. De fel som kan uppstå kan grovt delas in i en av följande typer:

- Kompileringsfel
  Detta är fel som uppstår när den källkod du skriver inte följer Javas språkregler. Editorerna brukar påpeka dessa fel och även ibland föreslå lösningar.

- Logiska fel
  Detta är fel som uppstår då programmet inte gör som det är tänkt. Det kan vara att du på ett ställe i koden utför subtraktion i stället för addition. Oftast är detta den svåraste typen av fel att hitta och att förebygga.

- Exekveringsfel
  Detta är fel som uppstår när programmet körs. Det kan vara allt från väldigt små fel som tillåter att programmet fortsätter till stora fel som gör att programmet kraschar. Det finns som tur sätt att hantera dessa fel och det är dessa typer av fel som denna lektion kommer att behandla.

När en situation uppstår som skapar fel utöver det normala kallar man det en exceptionell händelse eller att ett undantag uppstått (jag kommer växelvis använda begreppen exceptionell händelse och undantag, men menar samma sak). Dessa händelser är något man i Java kan fånga upp och hantera.

Alla typer av undantag är egna klasser i Java och alla ärver i ett eller flera steg från en klass som heter java.lang.Throwable. Här finns det två direkta subklasser, nämligen Error och Exception (vilka i sin tur är superklasser till ännu fler subklasser).

java.lang.Error är en klass som den virtuella maskinen själv använder och är därför inget du som programmerare normalt använder. Denna typ av händelse leder också oftast till att programmet stannar. Exempel på felsituationer som kan uppstå här är slut på minnen eller att hårddisken är full.

java.lang.Exception däremot är en klass som du ofta kommer att komma i kontakt med. Det här är fel som uppstår i våra program och som vi kan fånga och hantera. Det kan vara olika IO-fel (försöker använda en fil som inte finns) eller att man försöker använda en position i en array som inte finns (går utanför högsta index).

I kursboken finns en bra figur i kursboken (11.1 "Exception-klasser") som visar de flesta underklasser till Exception. Du behöver inte kunna känna igen alla dessa olika typer av exceptions

eller i vilka sammanhang de kan uppstå. En del av dessa klasser kommer du att komma i kontakt med under kursens gång, de mest vanliga, och som du bör lägga på minnet.

Att kasta egna undantag

Att själv kasta undantag i koden är ett bra sätt att signalera att något inte är som det ska. Det kan vara att förväntad indata inte är korrekt. För att kasta ett undantag används en throw-sats följt av någon undantagsklass (antingen någon befintlig eller någon vi själv skapat):

```
throw exception-klass;
```

Alla undantagsklasser (exception-klasser) följer nedanstående mall.

```
public class Klassnamn extends SuperKlassnamn {
  public Klassnamn();
  public Klassnamn(String s);
}
```

På sida 418 (412 i upplaga 6 och 394 i upplaga 5) finner du ett exempel på hur en egen helt ny undantagsklass kan se ut och även kod för hur man använder denna klass för att generera en händelse.

Nedan finns ett exempel som visar hur man kan definiera en egen klass för händelser och hur en tänkt klass skulle kunna använda sig av den klassen för att kasta en händelse.

```
public class EmailFormatException extends Exception {
  public EmailFormatException() {
    super();
  }

  public EmailFormatException(String msg) {
    super(msg);
  }
}

public class Student {
  private String email;

  public void setEmail(String email) throws EmailFormatException {
    if (email.indexOf('@') == -1) {
      throw new EmailFormatException("Email is missing @: " + email);
    }

    this.email = email;
  }

  public String toString() {
   return email;
  }
}
```

Vi har skapat en egen undantagsklass och använder klassen genom att kasta (throw) den om e-postadressen inte innehåller tecknet @.

```
throw new EmailFormatException("Email is missing @: " + email);
```

I metoddeklarationen måste vi specificera alla undantag som metoden på något sätt kan kasta. Det kan dels vara undantag som metoden själv kastar (med throw) eller undantag som kastas av andra metoder den aktuella metoden själv anropar.

```
public void method(parameters) throws E1, E2, E3 {
  …
}
```

Där E1, E2 och E3 är någon typ av undantagsklass och dessa specificeras i en kommaseparerad så kallad händelselista.

Fånga undantag

Nu kommer vi till den del som kanske är den viktigare och den vi oftast är ute efter, nämligen att fånga upp och hantera de undantag som kan uppstå. Det finns några saker man önskar göra genom att fånga en händelse

- Ta hand om händelsen och hantera situationen själv
- Fånga händelsen och skicka den vidare till någon annan som får hantera problemet
- Inte alls fånga händelsen (vilket normalt leder till att programmet avslutas)

Något boken kan vara dålig på att visa är hur man vet vilka händelser en metod kan generera. Om du tidigare inte använt dig av Javas API på nätet är det på tiden att du tar en snabb titt.

Öppna http://docs.oracle.com/javase/7/docs/api/ i din webbläsare. Om länken inte fungerar kan du prova att googla efter sökorden: java api <JDK#> (i detta fall valdes 7).

Som exempel kan vi titta på möjligheterna att konvertera strängar till heltal. Leta i den nedre listan i vänstra hörnet efter klassen Integer. Klicka på länken för att ”öppna” dokumentationen för Integer. Leta nu efter metoden parseInt. Det första vi ser är följande:

```
public static int parseInt(String s) throws NumberFormatException
```

I händelselistan (efter throws) finns undantaget NumberFormatException specificerat. Läser du vidare under stycket Throws får du reda på vad detta betyder.

```
NumberFormatException - if the string does not contain a parsable integer.
```

Det betyder att om vi skulle försöka omvandla en sträng som inte går att omvandla skulle undantaget NumberFormatException kastas. Vad det innebär för oss är att om vi vill kunna hantera detta och kanske begära att användaren matar in ett giltigt heltal måste vi fånga undantaget.

Men hur fångar man då ett undantag? Jo, det görs genom att omsluta de kodrader som kan kasta undantaget med en try-sats:

```
try {
  // de kodrader eller metodanrop som kan generera felet
}
catch  (E1 e1) {
  // kod för att hantera första felsituationen som kan uppstå
}
catch (E2 e2) {
```

```
    // kod för att hantera andra felsituationen
  }
```

En try-sats består av ett try-block samt en eller flera catch-block (samt eventuellt ett finally-block). I try-blocket lägger vi de kodrader, eller snarare metodanrop, som kan leda till att ett undantag kastas. För varje undantag som kan kastas i try-blocket lägger vi till ett motsvarande catch-block i vilken kod placeras för att hantera undantaget.

Om vi återgår till exemplet med att omvandla en sträng till ett heltal skulle det kunna se ut så här för att hantera felet som kan uppstå.

```
Scanner scan = new Scanner(System.in);
String inputData;
int heltal;

do {
  inputData  = scan.nextLine();

  try {
    heltal = Integer.valueOf(inputData);
    break; // if input is correct
  }
  catch (NumberFormatException e) {
    System.out.println("Felaktig inmatning!");
  }
}
while (true);
```

Det finns ett liknande exempel i boken sida 424 (417 i upplaga 6 och 399 i upplaga 5).

Det kan finnas tillfällen man vill att viss kod ska köras oavsett om det i en try-sats uppstått ett fel eller inte. Detta görs genom att lägga till ett finally-block i slutet av try-satsen.

```
try {
}
catch (…) {

}
finally {
}
```

Ett exempel skulle kunna vara att man väljer att stänga alla öppna filer oavsett om man lyckas läsa från filen eller inte. Om det uppstått något fel så stänger den filerna ändå.

Nedan har du ett exempel som visar hur man fångar undantaget EmailFormatException som vi tittat på i tidigare exempel.

```
public class ExceptionTest {
  public static void main(String[] args) {
    Student s = new Student();

    try {
      s.setEmail("student");
    } catch (EmailFormatException e) {
      System.out.println("Invalid email!");
      System.out.println(e.getMessage());
    } finally {
```

```
      System.out.println("Email is: " + s);
    }
  }
}
```

Vad tror du utskriften i finally-blocket blir?