PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Programació

UT 7.1: Control d'errades. Excepcions.

Joan Pons Tugores

Índex de continguts

Tractament d'errades

El tractament d'errades pot complicar molt el codi de l'aplicació. Acaben apareixent if's anidats, els mètodes tornen valors que d'altra manera no serien necessaris,...

```
errorCodeType readFile {
   initialize errorCode = 0;

   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
      if (theFileDidntClose && errorCode == 0) {
         errorCode = -4;
      } else {
         errorCode = errorCode and -4;
      }
```

```
    } else {
        errorCode = -5;
    }
    return errorCode;
}
```

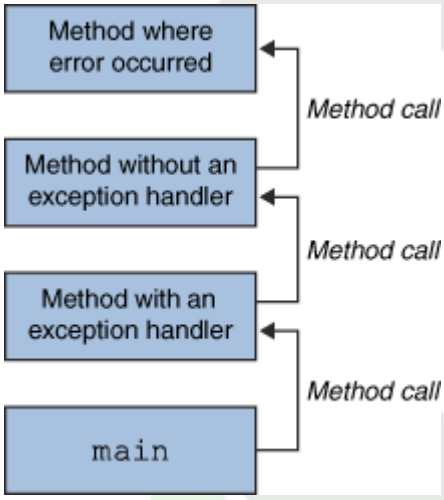
Java proporciona el mecanisme de les excepcions per el tractament d'errades. Els mètodes que poden provocar una errada llencen una excepció i on sigui possible tractar-la es captura.

Permet separar el tractament d'errades de la resta del codi i no implica modificar la signatura dels mètodes.

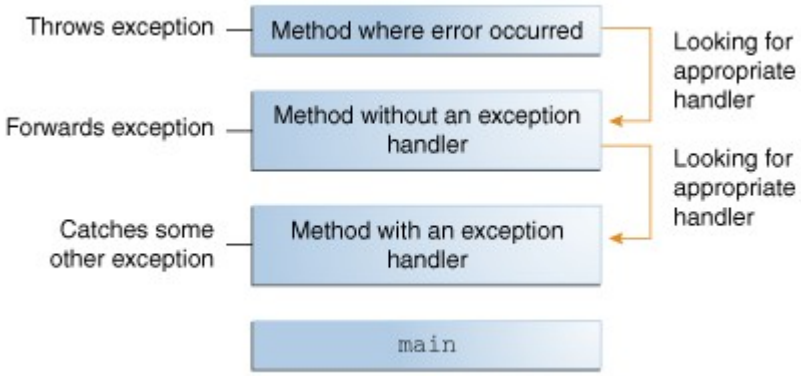
```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

Excepcions

El terme excepció en realitat hauria de ser esdeveniment excepcional. Una excepció és un esdeveniment que té lloc durant l'execució d'un programa i altera el flux normal de les seves instruccions. Un exemple d'excepció podria ser un programa que realitzi una divisió com a part dels seus càlculs i en un moment donat es trobi fent una divisió per zero.



En una situació com aquesta el mètode on ha tengut lloc l'error crea un objecte que inclou informació sobre l'error i el *llença* al sistema d'execució. El runtime system intenta trobar qualque manera de resoldre aquest error. Primer mira si el mètode on s'ha causat el poc tractar, si disposa d'un *exception handler*, un bloc de codi que tracti l'error i permeti seguir amb l'execució del programa, i si no és així recorre la pila d'execució intentant trobar-ne un que ho pugui fer. **Si no en troba cap, l'execució de l'aplicació acaba**.



Tipus d'excepcions

Podem distingir tres tipus d'excepcions:

- **Unchecked exceptions**
  - **Runtime exception**: condicions excepcionals internes a l'aplicació, i que normalment l'aplicació no podrà resoldre. Normalment indiquen errors de programació. Té més sentit intentar arreglar-los que recollir l'excepció. Tampoc estan subjectes al principi d'especificar o recollir.
  - **Error**: condicions excepcionals externes a l'aplicació que normalment no es poden anticipar o solventar, per exemple una errada del maquinari. Es poden recollir per exemple per notificar l'usuari, però no és obligatori. No segueixen el principi d'especificar o recollir.
- **Checked exceptions**: Situacions excepcionals que el programa ha de preveure i tractar. Per exemple, demanam a l'usuari el nom d'un fitxer i resulta que no existeix. Segueixen el principi de capturar o recollir. Seran les que noltros programarem.

Capturar o propagar

Qualsevol programa java ha de seguir el principi de capturar o propagar, és a dir, qualsevol codi que pugui llençar una excepció del tipus *checked exception* ha d'estar inclòs dins:

- Una sentència try – catch que capturi l'excepció o
- Un mètode que propagui l'excepció amb una clàusula throws.

Si aquest principi no es segueix, el programa no compila.

Creació i llançament d'excepcions

Per poder capturar una excepció abans qualque ú l'ha d'haver llençada.

Quan llençam una excepció? Bàsicament quan detectam una errada a un mètode que no té la possibilitat de tractar-la, de manera que l'errada arribi fins a un altre mètode que si que ho pugui fer.

En detectar una errada, per llançar una excepció hem d'utilitzar una comanda *throw*:

```
if (stack.size == 0) {
        throw new EmptyStackException();
}
```

Això ens obligarà a capturar o propagar aquesta excepció. Normalment no es capturen al mateix mètode on es llencen, si es pot tractar l'errada al mateix mètode es tracta on llençam l'excepció.

L'únic cas en que es sol capturar l'excepció en el mateix mètode que la llença és quan volem simplificar el codi d'un mètode molt complexe.

Propagació d'excepcions

Pot donar-se el cas que no ens interessi o no poguem tractar l'excepció dins el mètode que executa el codi que la produeix.

```
public void utilitzaStack() throws EmptyStackException{
                if (stack.size()==0){
                        throw new EmptyStackException();
                }
                ...
}
```

Per exemple, imaginem que ha botat una excepció per que el fitxer que vol obrir l'usuari no existeix. L'únic que podem fer es mostrar un missatge a l'usuari però pot ser que el mètode on ha botat no pugui interactuar amb la interfície d'usuari.

Podem propagar les excepcions afegint la clàusula *throws* a la declaració del mètode seguida de la llista amb totes les excepcions que volguem propagar separades per comes.

Captura d'excepcions

Per a capturar excepcions tancarem el codi que les pot provocar dins un bloc *try{}*.

Seguidament posarem un *catch* per a cada tipus d'excepció que volem capturar. Cada *catch* especifica un objecte d'un tipus d'excepció en concret. Dins el bloc executarem el codi necessari per a tractar l'excepció.

El bloc *finally* ens permet especificar codi que sempre s'executarà, tant si s'ha llençat cap excepció com si no.

A l'hora d'ordenar els distints blocs *catch* s'ha de tenir en compte que les excepcions formen una jerarquia d'herència i que només s'executarà el primer *catch* el tipus del qual coincideixi amb el de l'excepció que s'ha generat.

L'arrel de tota aquesta jerarquia és *Exception*. Si el primer catch és d'aquest tipus, mai s'executaran els altres blocs *catch*.

```
try{
    codi;
}catch(ExceptionType1 nom1){
    codi;
}catch(ExceptionType2 nom2){
    codi;
}finally{
    codi;
}
```

En aquest exemple intentam escriure el contingut d'una llista dins un fitxer.

En executar-lo poden passar dues coses:

- Que tot vagi bé. En aquest cas s'executarà el codi dels blocs try i finally.
- Que hi hagi qualque problema quan s'executi el bloc try. En aquest cas s'executarà el bloc catch corresponent a l'errada, i després el bloc finally.

Si es produeix alguna errada no contemplada del tipus unchecked exception l'excepció es propagarà fins que trobem un handler per a l'excepció o, en el pitjor dels casos, s'acabarà l'execució del programa.

```
public void writeList() {
    PrintWriter out = null;
    try {
        System.out.println("Entering try statement");
        out = new PrintWriter(
                    new FileWriter("OutFile.txt"));
          for (int i = 0; i < SIZE; i++)
              out.println("Value at: "
                          + i + " = "
                          + vector.elementAt(i));

    } catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("Caught "
            + "ArrayIndexOutOfBoundsException: "
            +   e.getMessage());

    } catch (IOException e) {
        System.err.println("Caught IOException: "
                          +  e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        }
        else {
            System.out.println("PrintWriter not open");
        }
    }
```

Avantatges de la utilització d'excepcions

Separació del codi de tractament d'errors de la resta de codi.

Mesclar el tractament d'errors amb la resta del codi sovint allarga el codi i dificulta la seva interpretació.

```
errorCodeType readFile {
   initialize errorCode = 0;

   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
   } else {
      errorCode = -5;
   }
   return errorCode;
}
```

```
readFile {
   try {
      open the file;
      determine its size;
      allocate that much memory;
      read the file into memory;
      close the file;
   } catch (fileOpenFailed) {
      doSomething;
   } catch (sizeDeterminationFailed) {
      doSomething;
   } catch (memoryAllocationFailed) {
      doSomething;
   } catch (readFailed) {
      doSomething;
   } catch (fileCloseFailed) {
      doSomething;
   }
}
```

Propagació d'errors per la pila d'execució

Únicament els mètodes que han de tractar els errors s'han de preocupar d'ells. Els altres només s'han de preocupar del codi que realment necessiten.

```
void method1() {
    errorCodeType error;
    error = method2();
    if (error)
        doErrorProcessing();
    else
       instruccions....
}

errorCodeType method2() {
    errorCodeType error=null;
    error = method3();
    if (!error){
       instruccions ....
    }
    return error;
}

errorCodeType method3() {
    errorCodeType error=null;
    error = readFile();
    if (!error){
        instruccions...
    }
   return error;
}
```

```
void method1() {
    try {
        method2();
        instruccions...
    } catch (exception e) {
        doErrorProcessing();
    }
}

void method2() throws exception {
    method3();
    instruccions...
}

void method3() throws exception {
    readFile();
    instruccins...
}
```

Agrupació i diferenciació de tipus d'error

La jerarquia que formen les distintes classes d'excepcions ens permet decidir amb quin nivell de detall tractam els errors.

És aconsellable tractar les excepcions al major nivell de detall possible ja que això ens permetrà respondre de manera més adequada.

A l'opció 1 tractam individualment l'excepció FileNotFoundException. Podrà rebre un tractament diferent d'altres excepcions, per exemple, demanar a l'usuari un altre nom de fitxer

A l'opció2 tractam de forma global totes les excepcions d'entrada sortida. Tant si el problema és que no troba el fitxer com qualsevol altre, ho haurem de tractar igual.

A l'opció 3 tractam per igual qualsevol tipus d'excepció. No és gens aconsellable.

```
Opció 1:
catch (FileNotFoundException e) {
    ...
}

Opció 2:

catch (IOException e) {
    ...
}

Opció 3:
catch (Exception e) {
    ...
}
```

Mala pràctica

Les excepcions s'utilitzen per tractar errades, es a dir, per detectar errades i intentar solucionar-les o com a mínim informar a l'usuari que s'han produït. Si no es tracten el programa acaba de forma inesperada amb un codi d'error que probablement l'usuari no podrà veure.

```
try{
    instruccions...
}catch (Exception e) {
    //No fer res
}
```

L'anterior bloc de codi és una mostra del pitjor que es pot fer: Es captura l'excepció però ni es tracta de solventar l'errada ni s'informa a l'usuari d'aquesta errada, ni el programa acaba de forma inesperada, senzillament continua (si pot) sense fer el que havia de fer de forma inexplicable tant per l'usuari com per el desenvolupador.

Un codi d'aquest tipus invalida l'exercici (o pràctica) on es trobi.

Classe Exception

Totes les excepcions són subclasses d'Exception. Per tant, podem crear les nostres pròpies excepcions d'aquesta manera:

```
public class DemoException extends Exception{
```

En principi la podem deixar buida, ja té tot el necessari per el fet de ser una subclasse d'Exception. Només hem de crear els constructors que necessitem, normalment el que no té paràmetres i el que té un paràmetre String. Aquest darrer ens permetrà enviar un missatge descriptiu de l'error dins l'excepció.

Mètodes útils de la classe Exception:

- .getMessage(): ens torna el missatge inclòs a l'excepció.
- .printStackTrace(): Mostra per consola (o al fitxer que li passem per paràmetre) la pila de l'errada, el text que estam acostumats a veure quan tenim una errada, amb la classe i la línia on ha tengut lloc.