Exceptions

Què és una excepció?

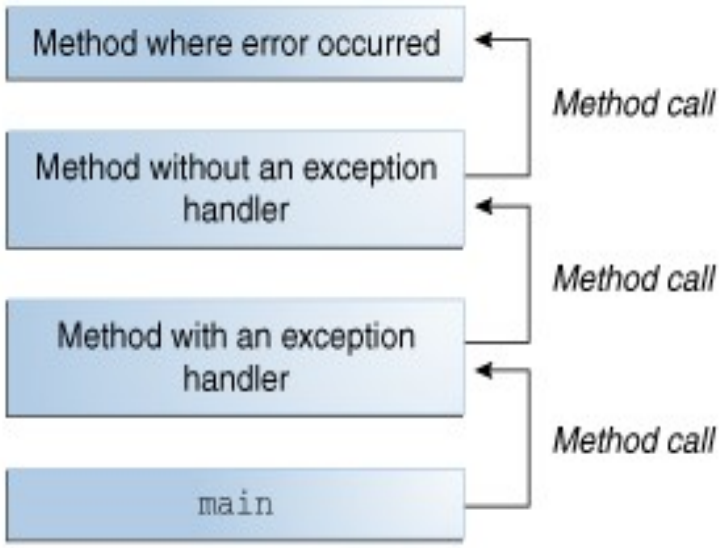
És un esdeveniment que té lloc durant l'execució d'una aplicació o programa i que provoca el trencament del fluxe normal d'instruccions.

Exceptions

- Quan un error té lloc dins d'un mètode, el mètode crea un objecte (excepció) i el llença cap amunt cap el sistema d'execució (runtime system).
- Aquest objecte llençat, l'excepció, inclou informació sobre l'error, com ara el tipus d'error i l'estat en què ha quedat el sistema.
- Crear l'objecte d'excepció i llençar-lo cap el runtime system s'anomena "throwing an exception".

Exceptions

- Quan una excepció és llençada (thrown) el runtime system intenta trobar quelcom que la manegi (handle).
- Aquesta cerca del "quelcom" que pot ser un handle de l'excepció segueix tota la pila de crides (call stack) en ordre invers fins arribar a la jvm en última instància.

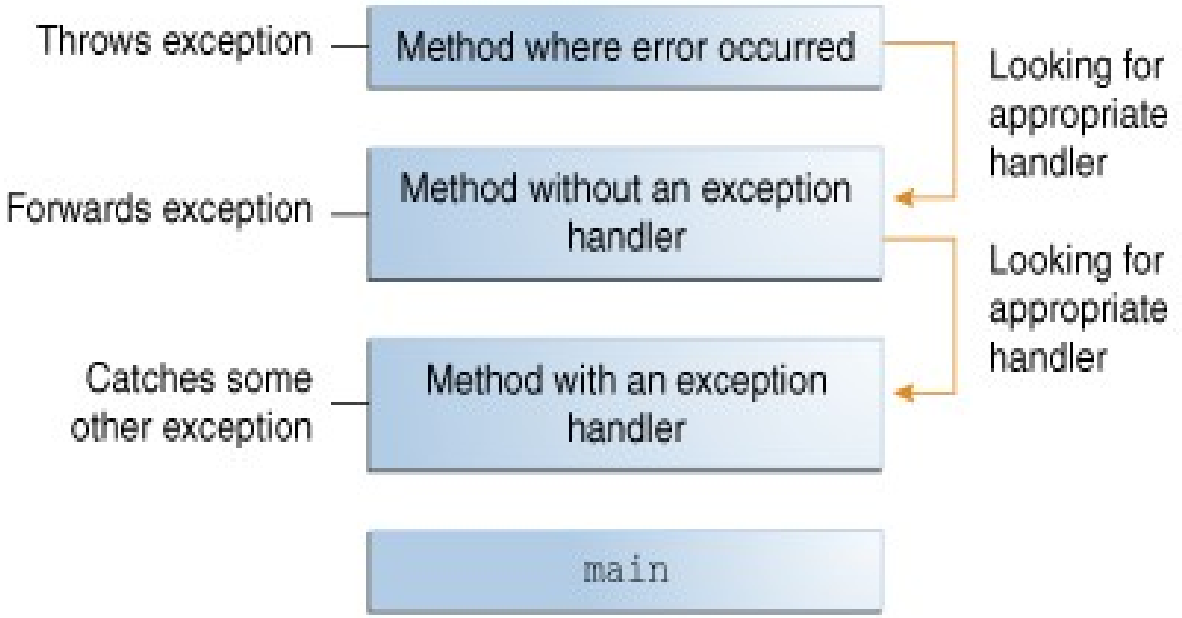


Exceptions

- El mètode que manejarà l'excepció haurà de tenir un bloc de codi anomenat *exception handler*.
- Quan el runtime system troba un exception handler, li passa l'objecte de tipus exception perquè la manegi. L'exception handler pot estar, en primer lloc, en el mateix mètode on s'ha generat l'excepció, o bé en qualsevol mètode de la pila de crides.

Exceptions

- Un exception handler diem que captura l'excepció: catch the exception.
- Si després de recorrer tota la pila de crides no es troba cap exception handler, el programa finalitza



The Catch or Specify Requirement

- Un tractament d'excepcions vàlid ha d'incloure la seva captura o bé una especificació (de llençament):
  - La captura (catch) es fa amb amb un bloc `try-catch`, que proveeix un handler per a l'excepció, si té lloc.
  - L'Specify es dóna proveint una clàusula `throws` que llisti l'excepció, llençant-la cap amunt en la pila de crides.

  Si no es compleix el requeriment de Catch or Specify, el programa no compila.
- Però no totes les exceptions estan subjectes a aquest requeriment...

Els tres tipus d'exceptions

1 - Checked exceptions:

Un programa ben escrit ha d'anticipar la possibilitat que hi hagi condicions excepcionals que puguin provocar un error.

Per exemple, una aplicació que pregunti a l'usuari el nom d'un fitxer, i que després l'obre amb un FileReader. Quan l'usuari entra un nom de fitxer no existent, el constructor de FileReader llença una `java.io.FileNotFoundException`. Un programa ben escrit capturarà aquesta excepció, i notificarà l'usuari que entri un nom de fitxer correcte.

- Les checked exceptions estan sotmeses al requeriment de *catch or specify*.

Els tres tipus d'exceptions

- Totes les excepcions són del tipus *checked exceptions*, menys les de tipus `Error` i `RuntimeException` i les seves subclasses.

  2 - Excepcions de tipus Error:

  Es tracta de situacions excepcionals, i que provoquen un error, que són externes a l'aplicació i que el programador no pot anticipar.

  Per exemple, una aplicació obre correctament un fitxer, però no en pot llegir el seu contingut degut a un mal funcionament físic del disc.

  En aquests casos es llença `java.io.IOError`

Els tres tipus d'exceptions

- Les excepcion de tipus Error **no estan subjectes al requeriment Cath or Specify**, tot i que opcionalment es poden capturar

  3 – The runtime exception:

- S'originen per condicions excepcionals que són internes a l'aplicació, i que la mateixa aplicació no pot anticipar i recuperar-se'n.

  Exemples en són els **bugs**, errors de programació, de lògica o d'ús indegut de l'API.

  En aquests casos té més sentit corregir el bug que no pas capturar l'excepció.

Els tres tipus d'exceptions

- Un exemple més concret de runtime exception podria ser un bug que fes passar com a paràmetre del constructor de FileReader un valor NULL. En aquest cas, el constructor llençaria una excepció `NullPointerException`.

  Runtime exceptions no estan subjectes al requeriment de Cath or Specify.

- Les excepcions de tipus Error i runtime exceptions també s'anomenen, en conjunt, com a *unchecked exceptions*.

Catching and Handling Exceptions

- Partim de l'exemple `ListOfNumbers:`

```
public class ListOfNumbers {
    private List<Integer> list;
    private static final int SIZE = 10;

    public ListOfNumbers () {
        list = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++) {
            list.add(new Integer(i));
        }
    }
```

Catching and Handling Exceptions

```
public void writeList() {
        PrintWriter out = new PrintWriter(new
FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = "
+ list.get(i));
        }
        out.close();
    }
}
```

Catching and Handling Exceptions

Aquest exemple no compila, ja que la instanciació de FileWriter pot tenir problemes d'I/O i per tant, en tractar-se d'una *checked exception*, el NetBeans ens retorna el següent avís:

`unreported exception IOException; must be caught or declared to be thrown`

És a dir, ens manca fer el tractament de l'excepció amb catch o un llençament (throws)

Catching and Handling Exceptions

- Aquest exemple també té una altra excepció, però és una unchecked exception, dins el mètode writeList, `list.get(i)`

  pot llençar una `IndexOutOfBoundsException`

  i opcionalment es pot capturar (catch) o llençar cap amunt a la pila de crides (throws). Si no es tracta, sí compila.

Catching and Handling Exceptions

- El primer pas en el tractament de les excepcions i crear un exception handler és tancar el codi que podria llençar una excepció dins d'un bloc `try`.

```
try {
    code
}
catch and finally blocks . . .
```

Catching and Handling Exceptions

- Hi ha dues maneres de fer-ho: crear un bloc `try` per a cada troç de codi que pugui llençar una excepció (en el nostre exemple, crearíem un `try` per al `new FileWriter`, i un `try` per a `list.get(i)`); o bé crear un únic bloc `try` que contingui tots els codis que puguin llençar excepcions.

Catching and Handling Exceptions

```
private List<Integer> list;
private static final int SIZE = 10;
PrintWriter out = null;
try {
    System.out.println("Entered try statement");
    out = new PrintWriter(new
FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++) {
        out.println("Value at: " + i + " = " +
list.get(i));
    }
} catch and finally statements . . .
```

Catching and Handling Exceptions

- Si una excepció té lloc dins del try, serà capturada per un bloc catch. Darrere d'un bloc try es poden posar 1 o més blocs catch, un per a cada tipus d'excepció (subclasses d'`Exception`). Si volem que totes les excepcions rebin el mateix tractament, posarem només un únic bloc catch usant la classe més genèrica Exception.

Catching and Handling Exceptions

```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

Catching and Handling Exceptions

Cada bloc catch és un exception handler diferent, que maneja una excepció del tipus que se li ha indicat.

ExceptionType indica el tipus d'excepció, i ha de ser una sublasse de la classe Throwable. La classe Exception és la classe més genèrica que deriva (o hereda) de Throwable.

El codi dins el bloc catch és el codi que s'executarà si té lloc l'excepció de tipus ExceptionType.

- Continuant amb el nostre exemple, creem dos blocs catch, un per a cada tipus d'excepció:

Catching and Handling Exceptions

```
try {


} catch (FileNotFoundException e) {
    System.err.println("FileNotFoundException: "
+ e.getMessage());
    throw new SampleException(e);


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

Catching and Handling Exceptions

- Els dos handlers imprimeixen un missatge d'error.
- El primer handler, a més, llença una excepció definida per l'usuari (programador). Efectivament, com a part del tractament d'excepcions, també en podem llençar una.

Capturant més d'un tipus d'excepció amb només un únic catch (Java 7)

- Des de la versió 7 de la JDK podem capturar més d'un tipus d'excepció amb només un bloc catch:

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

The finally block

- The finally block **sempre** s'executa quan el bloc try existeix. (Nota: si la JVM surt -el fil o procés s'acaba- mentre un bloc try o catch s'està executant, el bloc finally podria no executar-se)
- D'aquesta manera, encara que una excepció inesperada o no capturada per un catch tingui lloc, el bloc finally sempre s'executarà.
- Però amb el bloc finally no només podem tractar excepcions no esperades, sinó també tancar recursos o netejar o reinicialitzar variables, entre d'altres.

The finally block

- En el nostre exemple, en el bloc try obrim un FileWriter, que és un recurs que s'haurà de tancar abans de sortir del mètode writeList.
- Es pot sortir de 3 maneres diferents del bloc try:
  - Hi ha una excepció de tipus IOException
  - Hi ha una excepció de tipus IndexOfBoundsException
  - No hi ha cap excepció i finalitza correctament

En qualsevol dels casos caldrà tancar el recurs de l'stream, i el lloc on fer-ho és dins del finally block, perquè com hem dit, sigui quin sigui el cas, sempre s'executarà.

The finally block

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

Try-with-resources (JSE 7)

- Des de la versió 7 de java, existeix la possibilitat de tractar les excepcions amb l'ús de recursos amb una nova sintaxi del bloc try.
- Aquesta nova sintaxi només es pot aplicar a tots aquells recursos que implementen la interfície `java.lang.AutoCloseable`, que inclou tots els objectes que implementen `java.io.Closeable` (I/O streams).

Try-with-resources (JSE 7)

```
static String
readFirstLineFromFile(String path)
throws IOException {
    try (BufferedReader br =
                   new
BufferedReader(new FileReader(path)))
{
        return br.readLine();
    }
}
```

```
public static void writeToFileZipFileContents(String
zipFileName,String outputFileName) throws
java.io.IOException {

    java.nio.charset.Charset charset =
java.nio.charset.StandardCharsets.US_ASCII;

    java.nio.file.Path outputFilePath =
java.nio.file.Paths.get(outputFileName);

    // Open zip file and create output file with

    // try-with-resources statement

    try (java.util.zip.ZipFile zf =

new java.util.zip.ZipFile(zipFileName);

        java.io.BufferedWriter writer =
java.nio.file.Files.newBufferedWriter(outputFilePath,
charset)

    ) {
```

```
// Enumerate each entry
        for (java.util.Enumeration entries =
                                    zf.entries();
entries.hasMoreElements();) {
            // Get the entry name and write it to the
output file
            String newLine =
System.getProperty("line.separator");
            String zipEntryName =

((java.util.zip.ZipEntry)entries.nextElement()).getName() +
                  newLine;
            writer.write(zipEntryName, 0,
zipEntryName.length());
        }


    }
}
```

Try-with-resources (JSE 7)

- Quan s'ha de treballar amb més d'1 recurs, com en l'exemple anterior, se separen per un ;
- Cal tenir en compte que, a l'hora de tancar els recursos, es tancaran en ordre invers a com apareixen en la seva creació al bloc try-with-resources
- Un bloc try-with-resources pot tenir blocs catch i un bloc finally igual que els blocs try normals. La diferència és que abans que s'executi els continguts dels catch o del finally, els recursos ja estaran tancats.

Specifying the Exceptions Thrown by a Method

- En ocasions no voldrem capturar les excepcions i preferirem llençar-les cap a munt a la pila de crides.

```
public void writeList() throws
IOException,
ArrayIndexOutOfBoundsException {
```

- Com que la segona excepció és unchecked, i per tant no és oblogatori el seu tractament, podríem fer:

```
public void writeList() throws
IOException {
```

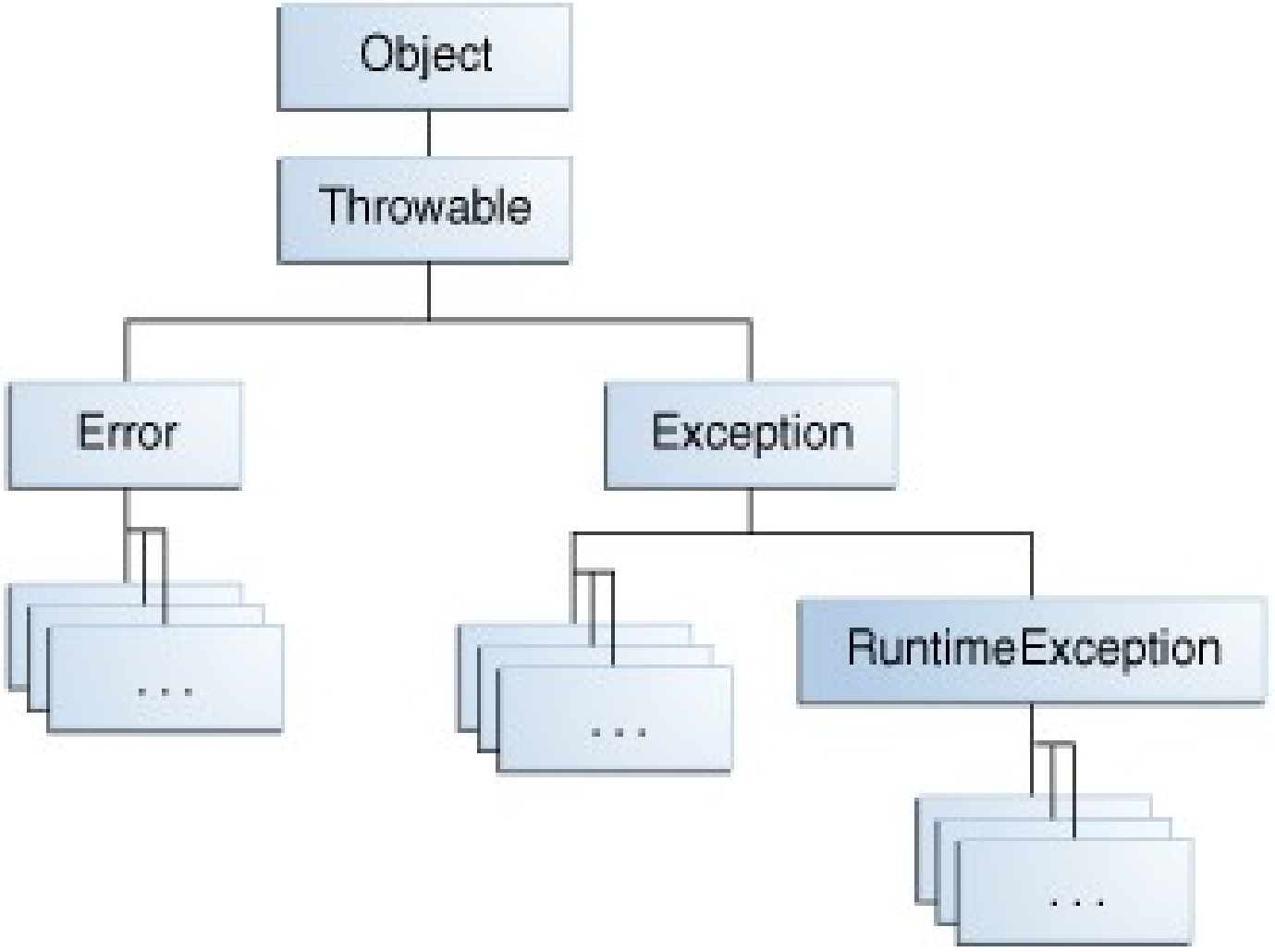
How to Throw Exceptions

- Com llençar excepcions? El programador pot crear excepcions personalitzades i llençar-les.
- Per a llençar-les, veiem-ne un exemple:

```
public Object pop() {
    Object obj;
    if (size == 0) {
        throw new EmptyStackException();
    }
    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

How to Throw Exceptions

- Els objectes que llencem amb throw han de ser objectes derivats de la classe Throwable:



How to Throw Exceptions

- El programador normalment usarà classes derivades de la classe Exception.
- La JVM llença errors (classe Error) normalment per causes externes al programa, com una fallada de maquinari.