Tratarea excepțiilor în Java

Exceptions handling in Java

Obiective:
- înțelegerea mecanismelor de tratare a excepțiilor; excepții simple; excepții multiple;
- throw vs. throws;
- metode care semnalizează excepții;
- clase proprii de tip excepție;

Objectives:
- understanding the exceptions handling mechanisms; simple exceptions; multiple exceptions;
- throw vs. throws;
- metods that launch exceptions;
- programmer defined exception classes;

Excepții Java

O excepţie nu reprezintă o eroare de compilare şi este dată de o situaţie anormală ce poate să apară în timpul execuţiei unei aplicaţii Java (program stand-alone, applet, etc.) cerând aplicaţiei:

- să îşi încheie imediat execuţia (în caz de eroare şi în lipsa unui mecanism de gestionare a situaţiei apărute)
- să execute o acţiune specială cu acea excepţie (evitând blocarea sistemului, dacă s-a prevăzut un mod de a gestiona situaţia apăruta)

Cele mai multe excepţii care se întâlnesc apar ca răspuns la unele condiţii de eroare cum ar fi :

- împărţirea cu zero a unui număr întreg
- depăşirea domeniului unei variabile
- acces nepermis la elementele unui tablou
- etc.

Când interpretorul întâlneşte o asemenea situaţie, se creează un obiect excepţie care memorează starea aplicaţiei din acel moment şi care va cauza un salt la o zonă dată de un indicator de excepţie (handler), dependent de natura excepţiei. Dacă nu e definit nici un indicator pentru o anumită excepţie, executarea programului sare la sfârşitul metodei în care a apărut excepţia. Acest mecanism poate continua, urcând astfel prin întreaga ierarhie de metode a aplicaţiei, până în momentul în care se ajunge în vârful acesteia (de exemplu funcţia *main*). Rezultatul este stoparea procesului software respectiv, de multe ori această inseamnand oprirea totală a aplicaţiei.

Excepţii de tip eroare

Prima parte conţine erorile, care sunt subclase ale *java.lang.Error*, cum ar fi *AnnotationFormatError, AssertionError, AWTError, CoderMalfunctionError, FactoryConfigurationError, LinkageError, ThreadDeath, TransformerFactoryConfigurationError, VirtualMachineError*. Acestea nu sunt de obicei în sarcina programatorilor şi constituie de multe ori erori de sistem (depăşiri de memorie, etc.)

Excepţii specifice

A doua parte conţine excepţiile specifice. În această categorie intră problemele care pot să apară într-un program corect care conţine date incorecte în contextul dat. Ca reprezentare, acestea sunt subclase ale *java.lang.Exception* (mai puţin

java.lang.RuntimeException) cum ar fi *AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, ClassNotFoundException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalAccessException, IllegalClassFormatException, InstanţiationException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvocationTargetException, IOException, JMException, LastOwnerException, LineUnavailableException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NoSuchFieldException, NoSuchMethodException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ServerNotActiveException, SQLException, TimeoutException, TooManyListenersException, TransformerException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URISyntaxException, UserException, XAException, XMLParseException, XPathException*.

Excepţii la rulare

Excepţiile la rulare constituie a treia categorie de excepţii iar aici intră de obicei bug-urile de programare (indecşi de şiruri în afara limitelor, variabile neiniţializate, etc.). Subclasele *java.lang.RuntimeException* sunt *AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, EnumConstantNotPresentException, EventException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IncompleteAnnotationException, IndexOutOfBoundsException, JMRuntimeException, LSException, MalformedParameterizedTypeException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, RejectedExecutionException, SecurityException, SystemException, TypeNotPresentException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationException*.

Clasa *java.lang.Throwable*

Metodele din clasa *java.lang.Throwable* se propagă prin ierarhia de mai sus şi sunt esenţiale pentru funcţionarea oricărei clase de tratare a excepţiilor.

Constructori:

- *Throwable()*

=> construieşte un nou obiect, având *null* drept mesaj de eroare.

- *Throwable(String message)*

=> construieşte un obiect care are mesajul de eroare specificat că parametru

- *Throwable(String message, Throwable cause)*

=> construieşte un obiect care are mesajul de eroare şi cauza specificate ca parametri

- *Throwable(Throwable cause)*

=> construieşte un obiect care are cauza specificată ca parametru

Metode membre:

- *Throwable fillInStackTrace()*

=> completează stiva de execuţie a programului

- *Throwable getCause()*

=> returnează cauza excepţiei în cauză sau *null* dacă cauza este inexistenta sau necunoscută

- *String getLocalizedMessage()*

=> crează o descriere localizată a obiectului în cauză

- *String getMessage()*

=> returnează un mesaj detaliat care descrie excepţia curentă

- *StackTraceElement[] getStackTrace()*

=> pune la dispoziţie accesul la elementele conţinute de stiva de execuţie a programului

- *Throwable initCause(Throwable cause)*

=> iniţializează cauza obiectului curent cu valoarea parametrului primit

- *void printStackTrace()*

=> tipăreşte (în fluxul standard de eroare) informaţia stocată în stiva de execuţie a programului

- *void printStackTrace(PrintStream s)*

=> tipăreşte informaţia stocată în stiva de execuţie a programului în fluxul de date de ieşire primit ca parametru

- *void printStackTrace(PrintWriter s)*

=> tipăreşte informaţia stocată în stiva de execuţie a programului în fluxul de date de ieşire de tip *writer* specificat

- *void setStackTrace(StackTraceElement[] stackTrace)*

=> setează elementele care vor fi returnate sau tipărite ulterior la apelul *getStackTrace()* şi *printStackTrace()*

- *String toString()*

=> returnează o descriere textuala a obiectului curent

Mecanismele de tratare a excepţiilor în Java

Pentru a intercepta şi trata o excepţie, se foloseşte instrucţiunea *try/catch/finally*.

Sintaxa:

```
try{
        //instrucţiuni care pot genera excepţii
}
catch(Tip_1_Excepţie nume_exceptie1){
        //bloc de tratare a tipului 1 de excepţie
}
catch(Tip_2_Excepţie nume_exceptie2){
        //bloc de tratare a tipului 2 de excepţie
}
//...
catch(Tip_N_Excepţie nume_exceptieN){
        //bloc de tratare a tipului N de excepţie
}
finally{
        //bloc de instrucţiuni care se execută fie că au apărut excepţii, fie că nu
  }
```

Există 2 reguli care guvernează tratarea excepţiilor multiple:

- excepţiile de tipuri mai specifice trebuie plasate înaintea celor mai generale
- un singur bloc *catch* şi numai unul (şi anume primul întâlnit care se potriveşte ca tip cu excepţia curentă) va fi executat la apariţia unei excepţii

Mecanismul de delegare a excepţiilor în Java

Sunt însă cazuri în care se permite delegarea excepţiilor, adică ”aruncarea” mai sus în ierarhia software a aplicaţiei respective. În acest sens se folosesc cuvintele cheie *throw* şi *throws*.

Pentru a delega (”arunca”) o excepţie trebuie parcurse două etape:

- crearea unei instanțe a unei clase de excepţie
- lansarea excepţiei create

Suprascrierea metodelor şi tratarea excepţiilor

În cazul în care o clasă derivată suprascrie o metodă a clasei de bază, pe lângă restricţiile impuse specificatorilor de vizibilitate, Java prevede faptul că metodele care suprascriu funcţiile originale nu pot semnaliza alte excepţii decât cele ale metodei originale. Este permisă însă declararea unei metode care nu semnalizează nici o excepţie, deşi metoda originală făcea acest lucru.

Definirea propriilor clase de tip excepţie

Programatorul este liber să definească propriile clase de tip excepţie, pentru că apoi să le folosească după necesităţi.

Mecanismul cel mai des întâlnit este alegerea unei clase de tip excepţie deja existente şi derivarea ei. Se poate alege clasa cea mai apropiată de necesităţi, sau se poate porni chiar de la *java.lang.Throwable*.

Lucru individual

1. Scrieți o aplicație Java care, în cadrul metodei *main()*, conține o secvență de cod care poate să arunce excepții de tip *ArithmeticException*, *ArrayIndexOutOfBoundsException*, *NullPointerException*, *NumberFormatException* și altele pe care considerați că e util să le testați. Afișați în cadrul blocurilor *catch* mesajul generat de aceste excepții. În blocul *finally*, afișați mesajul ”Am prins sau nu excepții”.

==========

2. Creați o clasă derivată din *Exception*, numită *SuperExceptie*. O a doua clasă *ExceptieMaiMica* este derivată din *SuperExceptie*. În constructorii claselor afisați un mesaj ce indică excepția generată. În cadrul unei alte clase creați o metodă a() ce semnalizeaza (aruncă) o excepție de tip *ExceptieMaiMica* și o metodă b() ce aruncă o exxcepție de tip *SuperExceptie*.
În cadrul metodei *main()* apelați aceste metode și încercați să determinați tipul de excepție aruncată, precum și dacă blocul *catch* aferent tipului de exceptie *ExceptieMaiMica* poate prinde o *SuperExcepție*.

==========

3. Creati o aplicație ce verifică numerele de înmatriculare ale mașinilor din România. Formatul acestora este următorul: [L{L}][NN{N}][LLL], unde L reprezintă o literă, N o cifră, iar acoladele semnifică faptul că pentru București numărul e format dintr-o singură literă în primul grup, iar grupul de cifre poate să aibă 3 cifre. Implementați metoda de verificare a numerelor de mașină și aruncați excepții (instante ale unei clase proprii de tip exceptie) specifice fiecărei erori ce poate să apară la verificare (mesaje particularizate). De exemplu, dacă județul e compus din 2 litere, setul de cifre nu poate să aibă dimensiunea 3. Ultimele litere nu pot să conțină ”I” și ”O” pe prima și ultima poziție.

==========

4. Scrieți o aplicație Java ce implementează o excepție numită *Saturat*. Această excepție este generată atunci când saturația unei culori este peste valoarea 200. Creați o metodă ce generează aleator culori în spațiul RGB și face conversia acestora la spațiul HSB/HSV. (https://www.cs.rit.edu/~ncs/color/t_convert.html) . Dacă după conversie, saturația culorii depășește valoarea 200, regenerați culoarea. După 10 regenerări consecutive, dacă nu se obține o culoare nesaturată, se aruncă o excepție.

==========

5. Scrieți o aplicație ce verifică dacă 3 puncte aleatoare formează un triunghi obtuzunghic. Dacă nu este îndeplinită condiția, se aruncă o excepție specifică: *TriunghiAscutitunghic*, *TriunghiDreptunghic*. Dacă cele 3 puncte se află pe aceeași dreaptă sau dacă segmentele pe care le determină nu pot forma un triunghi, se aruncă o excepție *TriunghiImposibil*, iar în blocul *catch* aferent, se afișează un mesaj corespunzător și se aruncă o excepție de tip *RuntimeException*.

==========

6. Definiți o clasă numită *Pozitie* care controlează poziția unui punct in spațiu (denumire punct (String), coordonatele pe axele X, Y, Z (valori intregi), constructori, accesori, mutatori). Se declară un tablou de maxim 3 obiecte de tip *Pozitie* după care utilizatorul este solicitat să introducă numărul dorit de obiecte. În secvența de initializare a datelor prin citire de la tastatură se tratează excepția de tip *ArrayIndexOutOfBoundsException* dată de introducerea unui număr incorect de obiecte.

Se testează coordonatele tuturor punctelor introduse iar dacă oricare dintre acestea sunt mai aproape de un alt punct cu coordonatele predefinite în program, se lansează o excepție de tip *PunctPreaAproape*. În blocul *catch* aferent utilizatorul este solicitat să reintroducă repetitiv coordonatele obiectului până când datele furnizate corespund criteriului.

Individual work

1. Write a Java application which, within the *main()* method, contains a sequence of code that may throw various exceptions, such as *ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, NumberFormatException,* as well as others which you consider to be useful for testing. In the *catch* block show the corresponding message generated by these exceptions. The *finally* block just prints the message, "I may or may not have caught an exception".

==========

2. Write a Java class derived from the *Exception* class, called *SuperException.* Another class, called *SmallerException* is derived from *SuperException.* Within the classes' constructors print a message which indicates which exception was generated. In a third class create a method a*()* which signals (throws) an exception of type *SmallerException*, and a method *b()* which throws a *SuperException.*

In the *main()* method call these two methods and try to determine the type of exception which occurs, as well as if the corresponding *catch* block for the *SmallerException* can catch a *SuperException.*

==========

3. Write an application which checks the Romanian vehicle registration numbers. Their format is the following: [L{L}][NN{N}][LLL], where L represents a letter, N a digit, and the curly braces represent the fact that for Bucharest the number is composed of a single letter in the first group, and the digit group can be composed of 3 digits. Implement a method which checks the registration numbers and throw exceptions (instances of specialized exception classes) specific to each error which may occur upon check-up (specialized messages). For example, if the county letters group is composed of 2 letters, the digit group cannot be of size 3. The last letters group cannot contain "I" and "O" on the first and last position.

==========

4. Write a Java application which implements an exception called OverSaturated. This exception is generated when the saturation of a colour has a value over 200. Write a method which randomly generated colors in the RGB space and then converts them to the HSB/HSV space (https://www.cs.rit.edu/~ncs/color/t_convert.html) . If after the conversion, the color's saturation is over 200 throw, regenerate the color. After 10 consecutive tries to regenerate, throw an exception.

==========

5. Write an application which checks if 3 random points draw an obtuse triangle. If the condition is not met, a specific exception is thrown: *AcuteTriangle, RightTriangle.* If the 3 points are on the same line or if the segments determined by the 3 points cannot make up a triangle, throw an *ImpossibleTriangle* exception, and within the corresponding *catch* block print a warning and throw a *RuntimeException*.

==========

6. Define a class called *Position* which controls the position of a point in space (point_name (String), X,Y,Z coordinates (Integer), constructors, setters, getters). Declare an array of maximum 3 objects of type *Position*. The user is then asked to enter from the keyboard the number of desired objects to initialise. In the initialisation sequence handle the *ArrayIndexOutOfBoundsException* generated by an incorrect number of points asked for by the user.

Test all the coordinates of the entered points, and if any of these are closer to another point hard-coded in the program, an exception of type *PointTooClose* is thrown. In the corresponding *catch* block the user is asked to re-enter the point's coordinates until the data is cosidered to be correct.