Java VI. – Kivételek

# Hibaészlelés, hibajelzés

## Kivételdobás

public class Time {

private int hour, minute; // 0 <= hour < 24, 0 <= minute < 60

public Time( int hour, int minute ){ ... }

public int getHour(){ return hour; }

public int getMinute(){ return minute; }

public void setHour( int hour ){

if( 0 <= hour && hour <= 23 ){

this.hour = hour;

} else {

throw new IllegalArgumentException("Invalid hour!");

}

}

public void setMinute( int minute ){ ... }

public void aMinutePassed(){ ... }

}

## Az assert utasítás

// may throw AssertionError

public void setHour( int hour ){

assert 0 <= hour && hour <= 23;

this.hour = hour;

}

Végrehajtás során kiértékelésre kerül egy feltétel, ami, ha nem teljesül egy AssertionError kivétel kerül eldobásra. A JVM alapesetben nem veszi figyelembe az assert-eket, ezeket nekünk kell bekapcsolni az alábbi módon:

$ java -enableassertions TestTime

Exception in thread "main" java.lang.AssertionError

at Time.setHour(Time.java:7)

at TestTime.main(TestTime.java:5)

Tehát az assert-ek opcionálisan bekapcsolható „biztosítékok”.

### Dokumentációs megjegyzés:

/\*\* May throw AssertionError. \*/

public void setHour( int hour ){

assert 0 <= hour && hour <= 23 ;

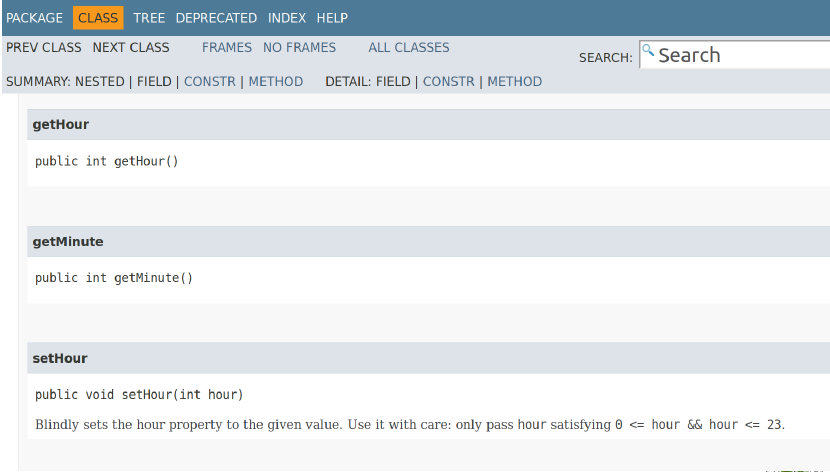
this.hour = hour;

}

Dokumentációkészítő program automatikusan legenerálja a dokumentációt a forráskód és a dokumentációs megjegyzések alapján.

A dokumentáció a következő paranccsal hozható létre:

$ javadoc Time.java



Kifejezetten bőbeszédű sok információt hordozó kommenteket lehet írni:

/\*\*

\* Sets the hour property. Only pass an {@code hour}

\* satisfying {@code 0 <= hour && hour <= 23}.

\* @param hour The value to be set.

\* @throws IllegalArgumentException

\* If the supplied value is not between 0 and 23,

\* inclusively.

\*/

Dokumentálhatjuk a metódus paraméterlistáját és azt is, hogy milyen kivételeket dobhat.

# Kivételek:

Ellenőrzött kivétel: olyan kivétel, melynek fellépési helyét a fordítóprogram ellenőrzi

Amikor egy metódus valamilyen ellenőrzött kivételt dob kötelesek vagyunk azt a szignatúrában feltüntetni.

public Time readTime( String fname ) throws java.io.IOException {...}

A fordítóprogram megvizsgálja, hogy merre terjedhet a kivétel és ellenőrzi, hogy mindenhol jelölve van, hogy az adott kivétel felléphet.

Nem ellenőrzött kivételeket használunk, ha a hiba oka kifejezetten a programban található. Nem ellenőrzött kivételek bárhol keletkezhetnek. Ilyen például:

* IllegalArgumentException
* IndexOutOfBoundsException
* NullPointerException

## Kivételkezelés:

import java.io.IOException;

class TestTime {

public Time readTime( String fname ) throws IOException {

... new java.io.FileReader(fname) ...

}

public static void main( String[] args ){

TestTime tt = new TestTime();

try {

Time wakeUp = tt.readTime("wakeup.txt");

wakeUp.aMinutePassed();

} catch( IOException e ){

System.err.println("Could not read wake-up time.");

}

}

}

A kivételkezelést a try, catch és a finally utasításokkal végezzük. A fenti kódrészlet megpróbálja beolvasni az időt a megadott fájlból, azonban, ha a fájlbeolvasás hibába ütközik IOException kivétel lép fel. Ennek hatására a program belép a hibára elsőként passzoló catch vezérlési ágba.

Így, ha tevékenységet nem sikerül végrehajtani, akkor a programunk tovább tud futni.

Egy try/catch blokk több catch ágat is tartalmazhat:

public static Time parse( String str ){

String errorMessage;

try { String[] parts = str.split(":");

int hour = Integer.parseInt(parts[0]);

int minute = Integer.parseInt(parts[1]);

return new Time(hour,minute);

} catch( NullPointerException e ){

errorMessage = "Null parameter is not allowed!";

} catch( ArrayIndexOutOfBoundsException e ){

errorMessage = "String must contain \":\"!";

} catch( NumberFormatException e ){

errorMessage = "String must contain two numbers!";

}

throw new IllegalArgumentException(errorMessage);

}

Ezeket akár egy ágba is tömöríthetjük:

...} catch(NullPointerException | ArrayIndexOutOfBoundsException|

NumberFormatException e){...

Több catch ág esetén mindig a legelső az eldobott kivételre illő ágba lép be a program ezért fontos, hogy először a specifikusabb kivételeket kezeljük le az általánosabbak felé haladva.

A finally blokk minden esetben lefut. Ez fájlkezelésnél különösen hasznos, mivel így minden esetben gondoskodhatunk a fájl bezárásáról. Ez egy nagyon fontos és gyakori idióma ezért Java-ban egy speciális utasítás a try-with-resources is helyet kapott.

try(

BufferedReader in = ...

){

String line = in.readLine();

return parse(line);

}

Ekkor a try speciális zárójeles részében megadott erőforrások automatikusan bezárásra kerülnek.

A try-with-resources-nek tetszőleges számú erőforrást megadhatunk.

static void copy( String in, String out ) throws IOException {

try (

FileInputStream infile = new FileInputStream(in);

FileOutputStream outfile = new FileOutputStream(out)

){

int b;

while( (b = infile.read()) != -1 ){ // idióma!

outfile.write(b);

}

}

}

(Olvasási idióma, ha az olvasás sikeres akkor az int változóba valami előjel nélküli szám kerül, ha sikertelen a beolvasás b értéke -1 (fájl vége) lesz és leáll a ciklus.)