finally

Не користити контролу тока (flow control) у finally!

Иако је то допуштено Јава спецификацијом, не треба га и радити! То лако може постати ноћна мора за програмера који ради на одржавању кода, а то можете бити и Ви сами. finally блокови постоје из једног јединог разлога, а то је да омогуће да се комплетно „почисти“ за собом без обзира шта се десило у коду који претходи (без обзира да ли је у придруженом try блоку дошло до избацивања изузетка или не). Ту се пре свега мисли на затварање фајлова, конекција са базом итд.
Јава упозорава ако постоји контрола тока (у шта спада и return) у finally блоку. Нажалост због компатибилности са претходним верзијама није могуће забранити такву непромишљеност.

Пример:
(узет са адресе: http://weblogs.java.net/blog/staufferjames/archive/2007/06/_dont_return_in.html)

```
public class Test {

  public static void main(String[] args) {
    try {
      doSomething();
      System.out.println("yikes!");
    } catch (RuntimeException e) {
      System.out.println("got it.");
    }
  }

  public static void doSomething() {
    try {
      //Normally you would have code that doesn't explicitly appear
      //to throw exceptions so it would be harder to see the problem.
      throw new RuntimeException();
    } finally {
      return;
    }
  }
}
```

Овај пример штампа „`yikes!`“. Ако се у finally појави return, сви изузеци који нису ухваћени неким од catch блокова придруженог try блока бивају потпуно изгубљени. Оно што је веома лоше је да све то изгледа потпуно „невино“, што се може видети из горњег примера. Осим за return, ово се односи и на све остало што се тиче преноса контроле тока (continue, throw, break):

```
      for(;;) {
           try {
               return 1;
           } finally {
               break;
           }
       }
       return -1;
```

Овај фрагмент кода враћа -1.

```
while (true) {
        try {
            return;
        } finally {
            continue;
        }
    }
    /* NOT REACHED */
```

А овај представља бесконачну петљу.

finally
(Извор: http://pages.cs.wisc.edu/~cs536-1/NOTES/JAVA/Exceptions.html)

Ако finally укључује наредбу за пренос контроле тока (return, break, continue, throw), онда та наредба има предност у односу на пренос контроле из одговарајућег try, односно catch блока. Најпре претпоставимо да finally не садржи пренос контроле тока. Могу наступити следеће ситуације:

1. није се десило избацивање изузетка у try и није извршен пренос контроле тока у try
   → извршава се finally, а затим и код који следи испод
2. није се десило избацивање изузетка у try, али try извршава пренос контроле тока
   → извршава се finally, а тек онда пренос контроле тока из try
3. десило се избацивање изузетка у try и нема catch-блока за тај изузетак
   → извршава се finally, а потом се неухваћени изузетак прослеђује навише до наредног придруженог try-блока, могуће у позивајућем методу
4. десило се избацивање изузетка у try и постоји catch-блок за тај изузетак, који не врши пренос контроле тока
   → извршава се catch, затим finally, па наредбе које следе испод
5. десило се избацивање изузетка у try, постоји catch-блок за тај изузетак, који врши пренос контроле тока
   → извршава се catch, затим finally, па пренос контроле тока.

   Ако finally укључује пренос контроле тока, онда он има предност у односу на пренос контроле из try или catch. Према томе, за све горње случајеве, извршава се finally, а затим и његов пренос контроле тока.

   Нпр.

   ```
   try {
       return 0;
   } finally {
       return 2;
   }
   ```

   резултат извршавања овог кода је да је враћена вредност 2.
   То је прилично збуњујуће! Поука је да вероватно НЕ ЖЕЛИМО да укључимо наредбе за пренос контроле тока и у try и у finally, односно и у catch и у finally.