200422 Задание: Числа фибоначчи нахождение по номеру:

int first = 0;  
int second = 1;  
int fib;  
if (index==1||index==2){  
 return index-1;  
}

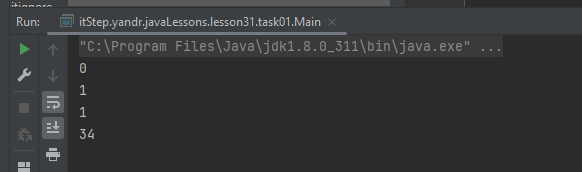
//1 2 3 4 5 6 7 8 9 10 11 12 index  
//0 1 1 2 3 5 8 13 21 34 55 89 number Fib

// 3 : fib = second+first; 0 + 1 = 1  
// first = second;0 = 1  
// second=fib; 1 =

//1 2 3 4 5 6 7 8 9 10 11 12 index  
//0 1 1 2 3 5 8 13 21 34 55 89 number Fib  
// 4 : fib = second + first; 1 + 1 = 2  
// first = second; 1 = 1  
// second = fib ; 1 =

//1 2 3 4 5 6 7 8 9 10 11 12 index  
//0 1 1 2 3 5 8 13 21 34 55 89 number Fib  
// 5 : fib = second + first; 2 + 1 = 3  
// first = second; 1 = 2  
// second = fib ; 2 =

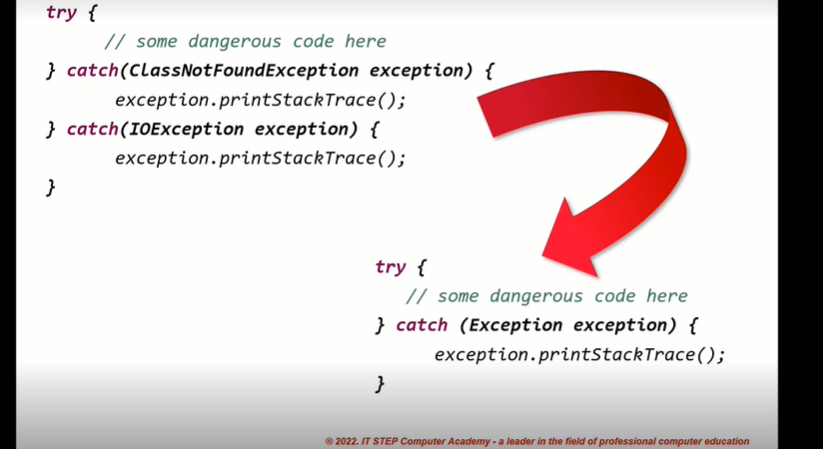
//1 2 3 4 5 6 7 8 9 10 11 12 index  
//0 1 1 2 3 5 8 13 21 34 55 89 number Fib

package itStep.yandr.javaLessons.lesson31.task01;  
  
public class FibonacciLogic {  
  
 public static int getFibonacciNumber(int index) {  
 if(index<=0){  
 return -1;  
 }  
 int first = 0;  
 int second = 1;  
 if (index == 1 || index == 2) {  
 return index - 1;  
 }  
 int fib = second + first;  
   
 for (int i = 3; i < index; i++) {  
 first = second;  
 second = fib;  
 fib = second + first;  
 }  
 return fib;  
 }  
}

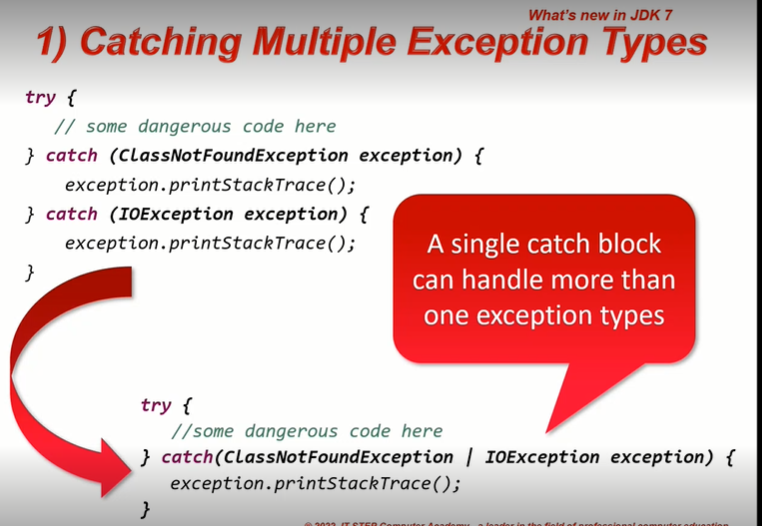
package itStep.yandr.javaLessons.lesson31.task01;

public class Main {  
 public static void main(String[] args) {  
 System.*out*.println(FibonacciLogic.*getFibonacciNumber*(1));  
 System.*out*.println(FibonacciLogic.*getFibonacciNumber*(2));   
 System.*out*.println(FibonacciLogic.*getFibonacciNumber*(3));  
 System.*out*.println(FibonacciLogic.*getFibonacciNumber*(10));  
 }  
}

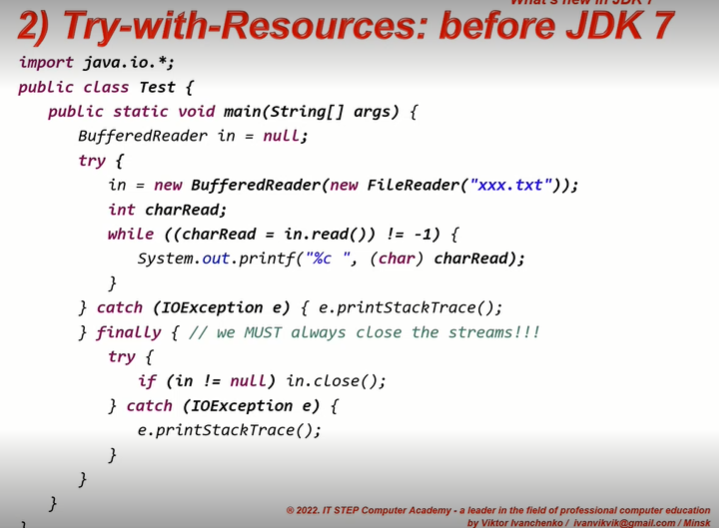
Что нового в JDK 7 1)Обработка множественных исключений: Если обработчики содержат одинаковый код, то для избежания copypast, используется побитовое or. Ранее в таких случаях использовали исключение базового класса, которое отлавливало все производные объекты:



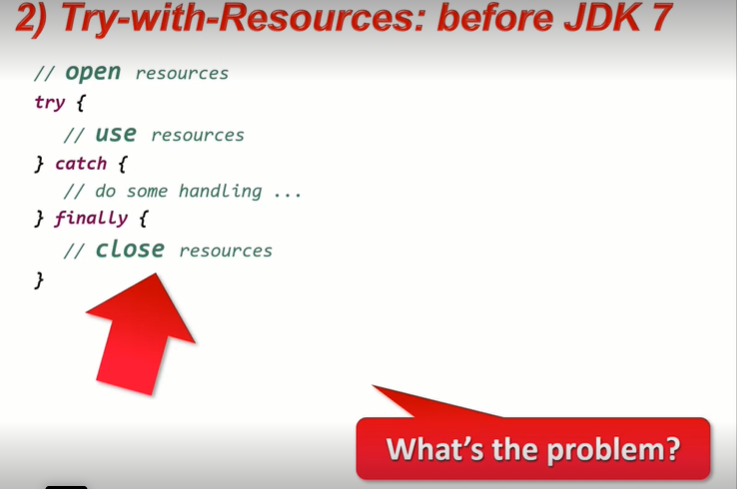
Но такой подход имеет значимые недостатки, так как отлавливаться будут и те производные объекты, которые находятся вне ответственности данного уровня, по –этому с 7 версии JDK , был перегружен побитовый оператор или для использования в блоке catch для совмещения нескольких исключений в одном обработчике



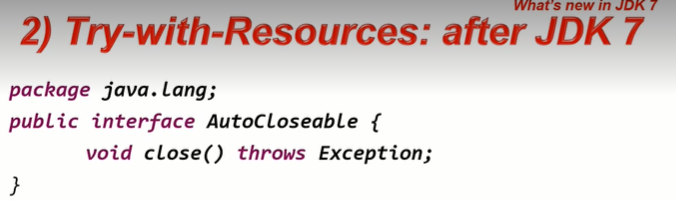
2)Добавлена модификация структуры try: Try-with-Resources



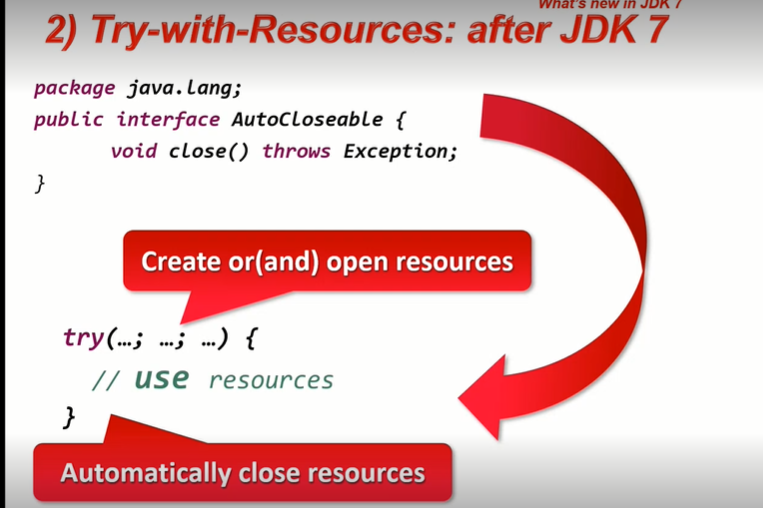
Ранее для закрытия ресурсов блоке finally необходимо было обработать исключения



С появлением новой модификации : Try-with-Resources,благодаря интерфейсу AutoCloseable



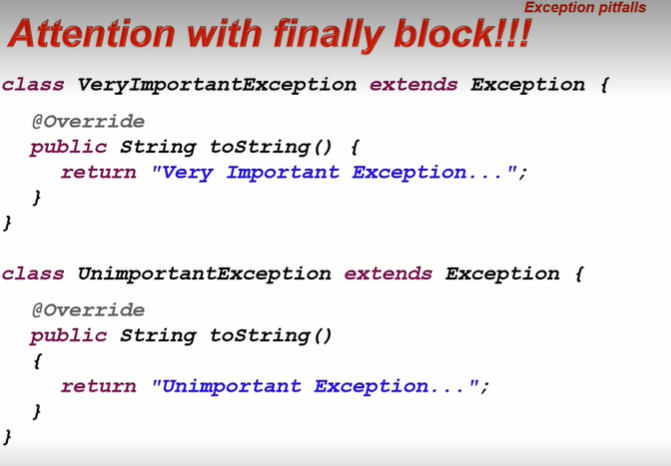
Появился новый блок try(..;..;..;…)в котором можно через ; указать cсоздаваемые ресурсы, требующие закрытия и не описывать их в блоке finally



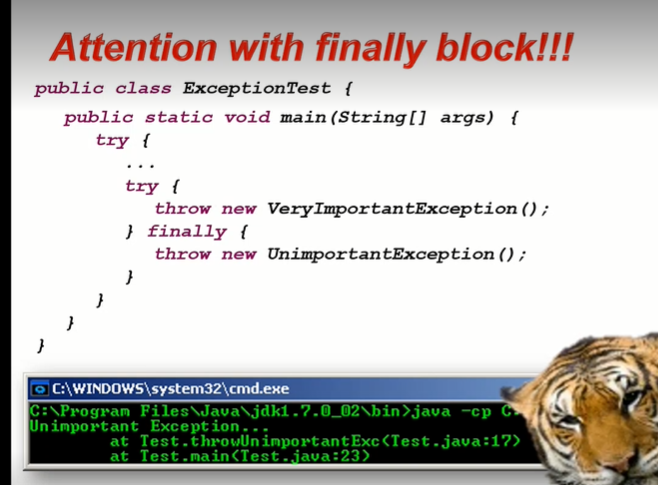


Теперь те ресурсы, которые создаютсяв try-W-R, за его пределами будут освобождаться, путем автоматического вызова . close(), тк теперь они имплементят Autocloseable

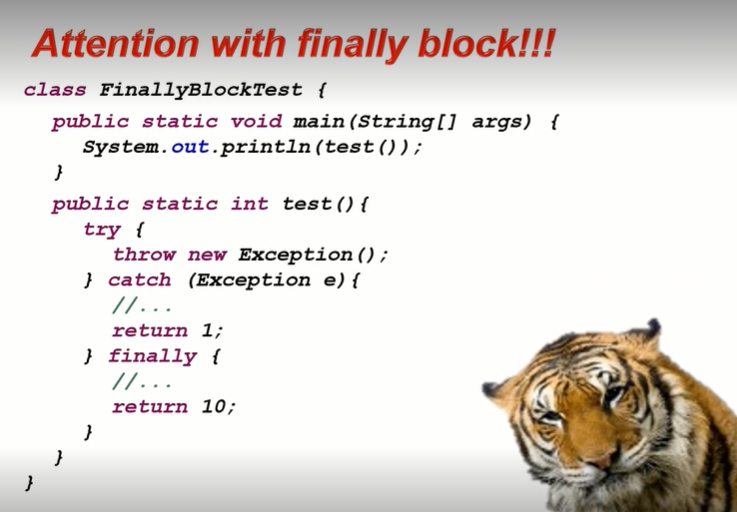
Подводные камни:



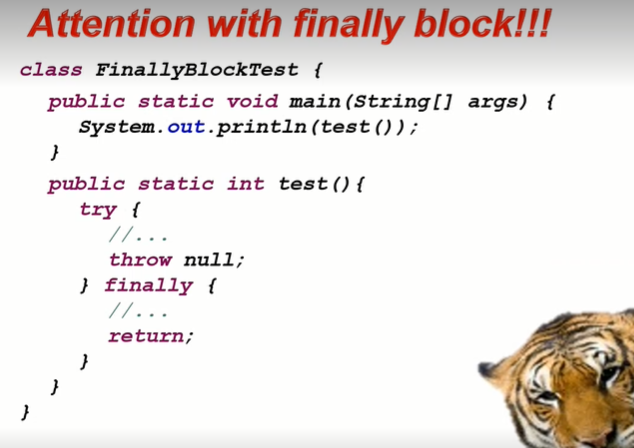
Исключение созданное в блоке finally, затемняет исключение из блока try



Блок finally выполняется даже , если ему предшествует return, тк finally освобождает ресурсы, а блок catch обрабатывает exception, ресурсы конечно важнее



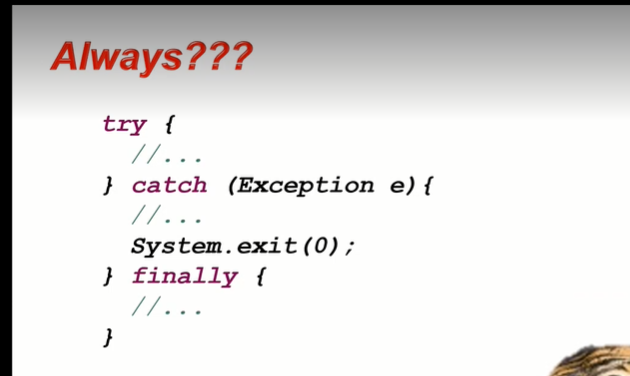
Когда входит в противоречие со throw, то генерация исключения не успевает выполнится, тк finally приоритетнее, те это исключение так и не сгенерируется, только кэшируется



Блок finally не выполняется в 2х ситуациях:

- в JVM возникает критическая ошибка, при которой JVM не может работать

- при ручном отключении JVM System.exit



Рекурсия: вызов методом самого себя

Все задачи, решаемые с помощью рекурсии, относятся к категории “разделяй и властвуй”

package itStep.yandr.javaLessons.lesson31.task01;  
  
public class Main {  
 public static void main(String[] args) {  
 *test*(1);  
  
 }  
 public static void test(int count){  
 System.*out*.println(count);  
 *test*(++count);  
 }  
}

рассмотрим задачу с числами фибонначи, где n индекс искомого ЧФ

n = n-1+n-2 n 1 2 3 4 5

n-1 = n-2 + n-3

getFib (index 5)

getFib (index -2)->*3*

getFib (index -1)->4

+

+ + +

getFib (index -3)->*2*

getFib (index -2)->*3*

getFib (index -3)->*2*

getFib (index -4)->*1*

base-case

getFib (index -4)->*1*

getFib (index -3)->*2*

+ base-case

getFib(1)

getFib(2)

getFib(3)

getFib(2)

getFib(1)

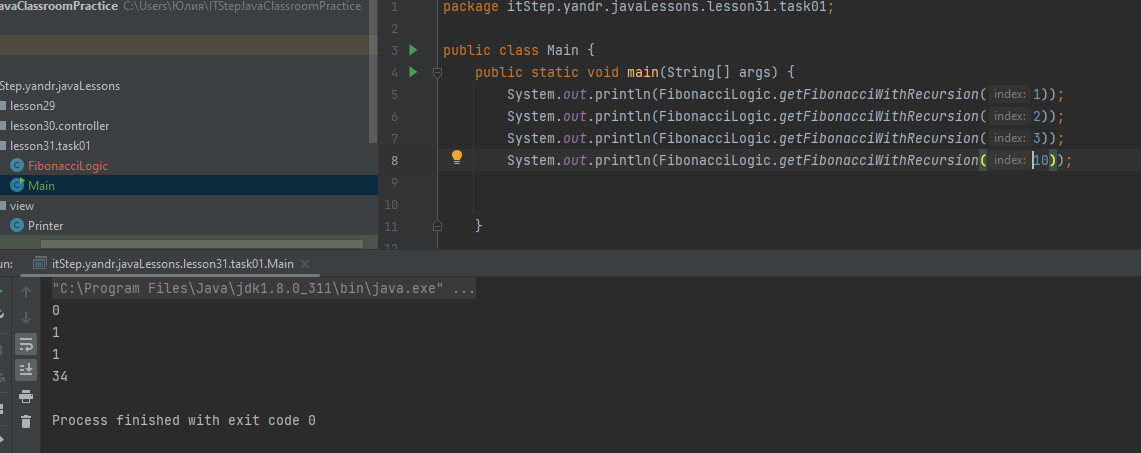
getFib(2)

getFib(4)

getFib(3)

getFib(5)

public static int getFibonacciWithRecursion(int index) {  
 if (index <=2) {  
 return index - 1;  
 }  
 return *getFibonacciWithRecursion*(index -1)+*getFibonacciWithRecursion*(index -2);  
}



Если можно разбить алгоритм на одинаковые в реализации задачи, то это рекурсия



Однако рекурсия ресурсозатратна, тк речь идет о множественных вызовах метода, алгоритмическая сложность, такая же как у цикла. Рекурсивный метод всегда состоит из 3х частей:

Развертывание стека, базовый случай и свертывание стека