

## Лекция 4



# ЛЕКЦИЯ 4

- BREAK label
- CONTINUE label

# BREAK LABEL

## ■ Without labeled statement

```
while (Some condition)
{
    if ( a specific condition )
        break;          //Default usage
    else
        normal business goes here..
}
```

## ■ With labeled statement

```
hackit:
while (Some condition)
{
    if ( a specific condition )
        break hackit;    //Usage with label
    else
        normal business goes here..
}
```

# EXAMPLES

```
outer: for (int i = 0; i < 10; i++) {  
    inner: for (int j = 10; j > 0; j--) {  
        if (i != j) {  
            System.out.println(i);  
            break outer;  
        }else{  
            System.out.println("-->" + i);  
            continue inner;  
        }  
    }  
}
```

```
int a = 10;  
int b = 12;  
  
block1: {  
    if (a < 0) {  
        break block1;  
    }  
    if (b < 0) {  
        break block1;  
    }  
    System.out.println( a + b );  
}  
}
```

```
class BreakDemo {  
    public static void main(String[] args) {  
  
        int[] arrayOfInts =  
            { 32, 87, 3, 589,  
              12, 1076, 2000,  
              8, 622, 127 };  
        int searchfor = 12;  
  
        int i;  
        boolean foundIt = false;  
  
        for (i = 0; i < arrayOfInts.length; i++) {  
            if (arrayOfInts[i] == searchfor) {  
                foundIt = true;  
                break;  
            }  
        }  
  
        if (foundIt) {  
            System.out.println("Found " + searchfor + " at index " + i);  
        } else {  
            System.out.println(searchfor + " not in the array");  
        }  
    }  
}
```

```
class BreakWithLabelDemo {
    public static void main(String[] args) {

        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
        };
        int searchfor = 12;

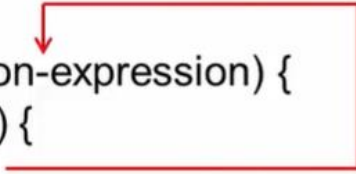
        int i;
        int j = 0;
        boolean foundIt = false;

    search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length;
                j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

# CONTINUE


- Използва се само в цикли
- Продължава(*continue*) със следващата итерация от най-външния(*innermost*) ЦИКЪЛ

```
while (condition-expression) {  
    if (condition) {  
        continue;   
    }  
    ...  
}
```

```
while (getNext(line)) {  
    if (line.isEmpty() || line.isComment())  
        continue;  
    // More code here  
}
```

# CONTINUE В ЦИКЪЛ FOR

```
for (initialization, condition-expression, expression-list) {  
    if (condition) {  
        continue;  
    }  
}
```





Ще се компилира ли този код?

```
if () {  
    continue;  
}
```

# Labeled break vs Labeled continue

```
label1: if () {  
    for () {  
        break label1;  
    }  
}
```

```
label1: if () {  
    for () {  
        continue label1;  
    }  
}
```

```
class ContinueDemo {  
    public static void main(String[] args) {  
  
        String searchMe = "peter piper picked a " + "peck of pickled peppers";  
        int max = searchMe.length();  
        int numPs = 0;  
  
        for (int i = 0; i < max; i++) {  
            // interested only in p's  
            if (searchMe.charAt(i) != 'p')  
                continue;  
  
            // process p's  
            numPs++;  
        }  
        System.out.println("Found " + numPs + " p's in the string.");  
    }  
}
```

```
class ContinueWithLabelDemo {
    public static void main(String[] args) {

        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;

        int max = searchMe.length() -
            substring.length();

        test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                    continue test;
                }
            }
            foundIt = true;
            break test;
        }
        System.out.println(foundIt ? "Found it" : "Didn't find it");
    }
}
```

# Рекурсии в JAVA



Recursion

```
public static void main(String[] args) {
```

```
    ... ..
```

```
    recurse()
```

```
    ... ..
```

```
}
```

```
static void recurse() {
```

```
    ... ..
```

```
    recurse()
```

```
    ... ..
```

```
}
```

Normal  
Method Call

Recursive  
Call

*method* → addresses a problem

*recursive call* → *addresses a sub-problem*

**base case**

# Изпълнение на метода factorialNR

```
static int factorialNR(int n) {  
    if (n == 0 || n == 1) {  
        return 1;  
    }  
  
    int factorial = n;  
  
    while (n >= 2) {  
        factorial = factorial * (n - 1); // 4 * 3 = 12 * 2 = 24 * 1 = 24  
        n--; // 3, 2, 1  
    }  
  
    return factorial;  
}  
  
static int factorial(int n) {  
    return n * factorial(n - 1);  
}  
  
public static void main(String[] args) {  
    System.out.println(factorialNR(4));  
    System.out.println(factorial(4));    // Clearer Code (slightly slow)}
```



```
5  @ static int factorialNR(int n) {  
6      if (n == 0 || n == 1) {  
7          return 1;  
8      }  
9  
10     int factorial = n;  
11  
12     while (n >= 2) {  
13         factorial = factorial * (n - 1); // 4 * 3 = 12 * 2 = 24 * 1 = 24  
14         n--; // 3, 2, 1  
15     }  
16  
17     return factorial;  
18 }  
19  
20 static int factorial(int n) {  
21     return n * factorial(n - 1);  
22 }  
23  
24 public static void main(String[] args) {  
25     System.out.println(factorialNR(n: 4));  
26     System.out.println(factorial(n: 4)); // Clearer Code (slightly slow)
```

"C:\Program Files\Java\jdk1.8.0\_221\bin\java" ...

24

Exception in thread "main" java.lang.StackOverflowError

at Main.factorial(Main.java:22)

at Main.factorial(Main.java:22)

at Main.factorial(Main.java:22)

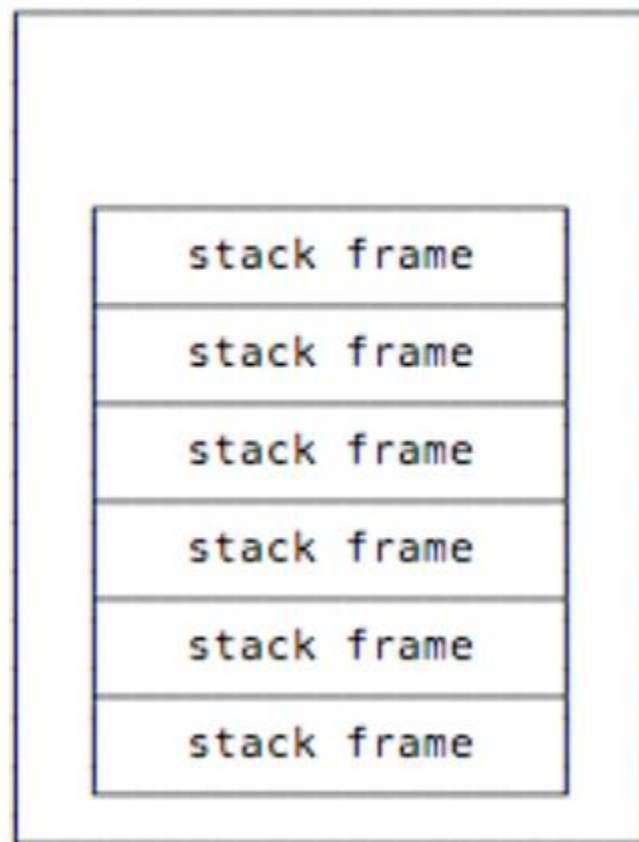
at Main.factorial(Main.java:22)

at Main.factorial(Main.java:22)

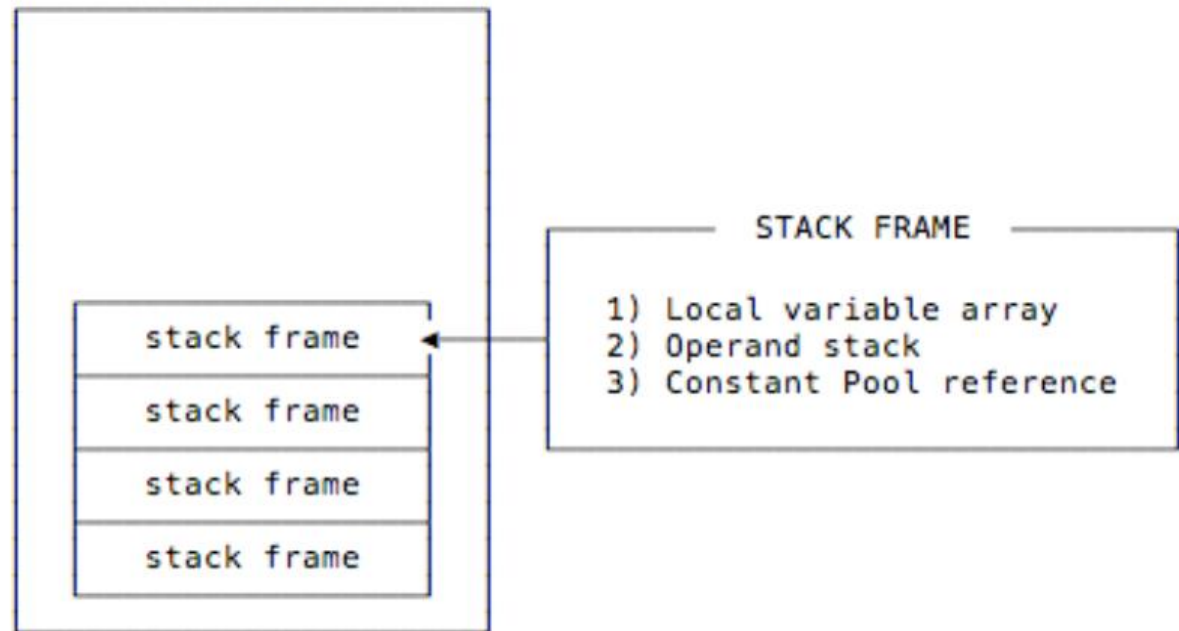
# Официална дефиниция на ORACLE за „stack“ и „stack frame“

*“Each JVM thread has a private Java virtual machine stack, created at the same time as the thread. A JVM stack stores frames, also called “stack frames”. A JVM stack is analogous to the stack of a conventional language such as C — it holds local variables and partial results, and plays a part in method invocation and return.”*

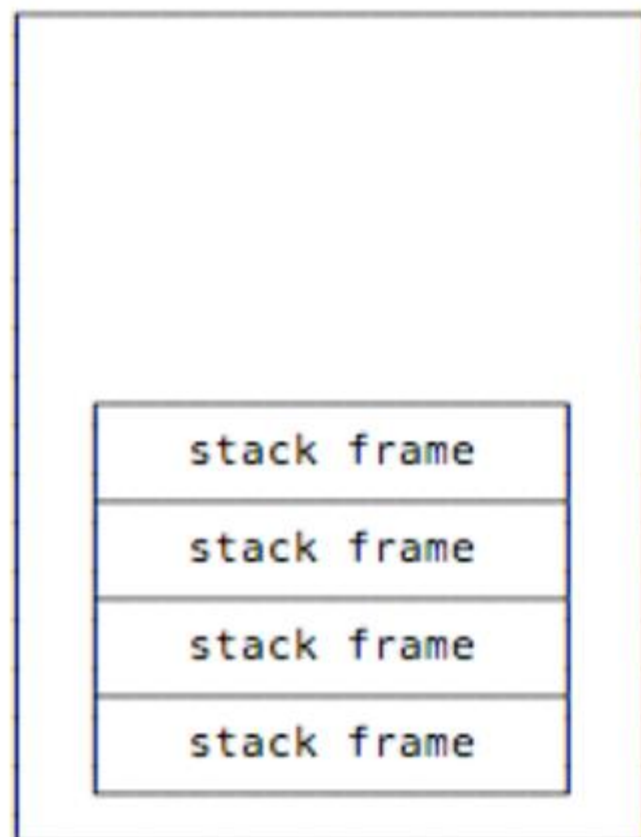
## THE STACK



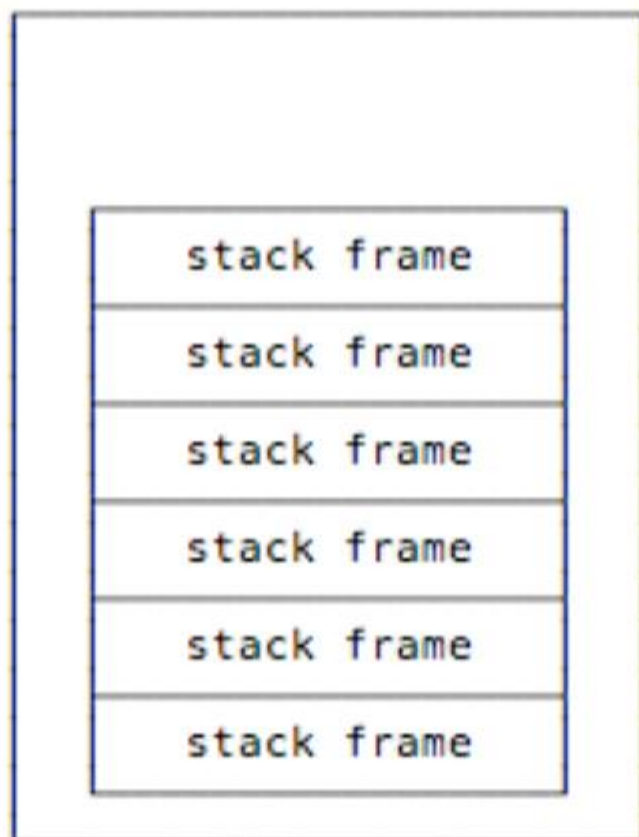
## THE STACK



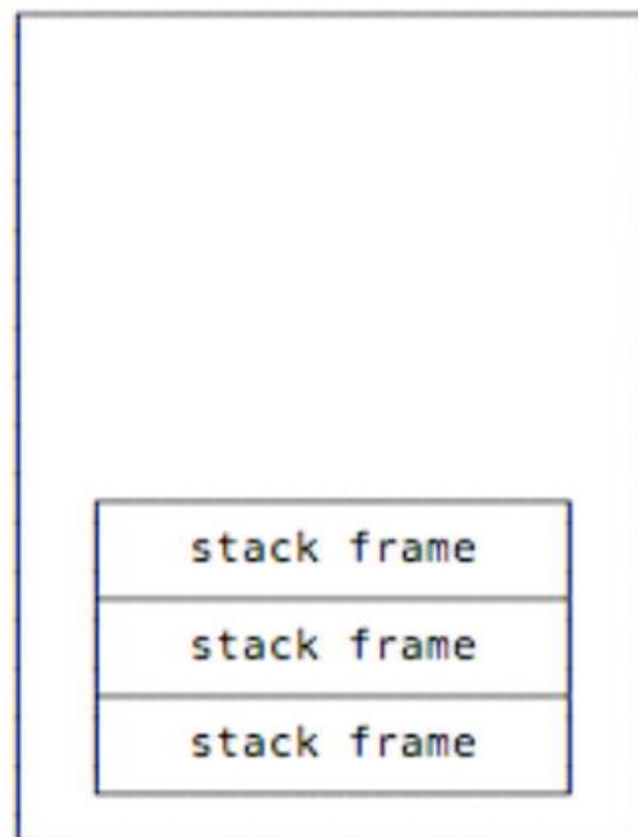
THREAD 1



THREAD 2



THREAD 3



# Използване на база за да отстраним StackOverflowError

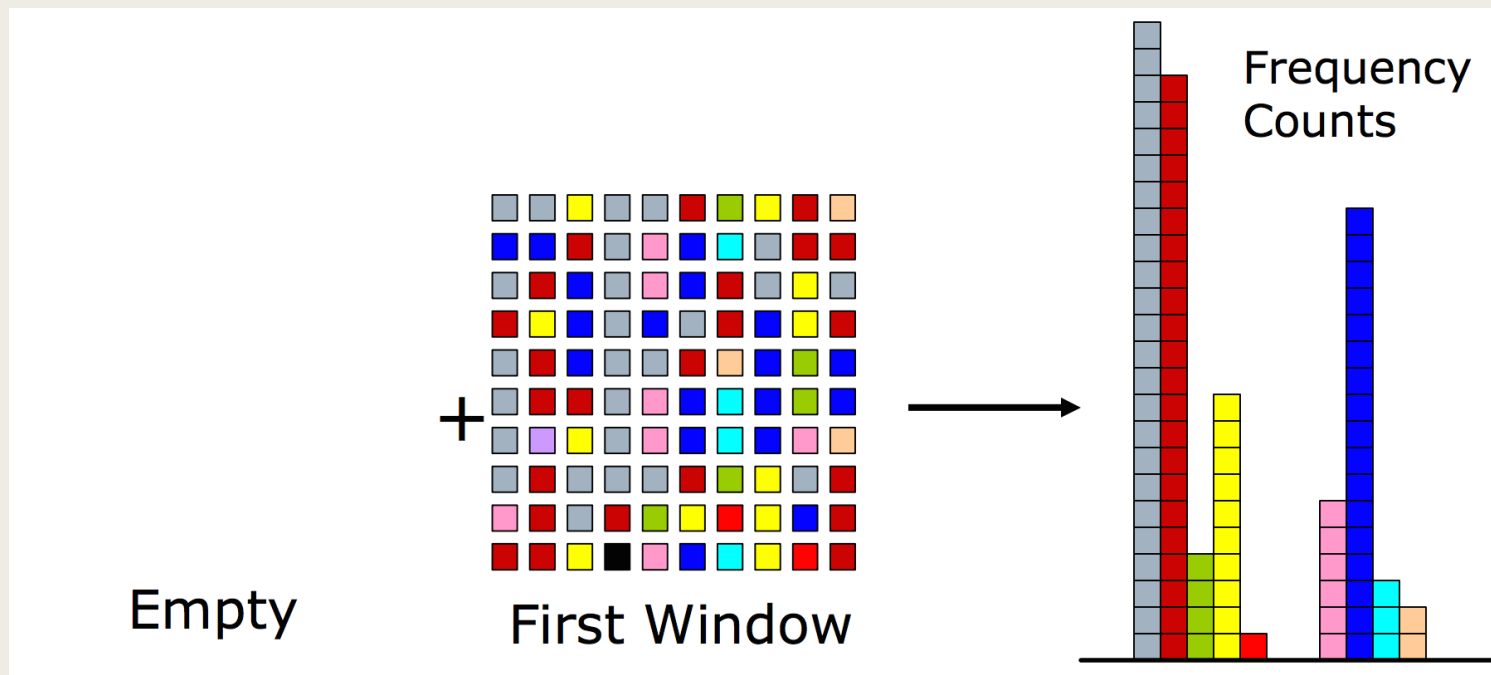
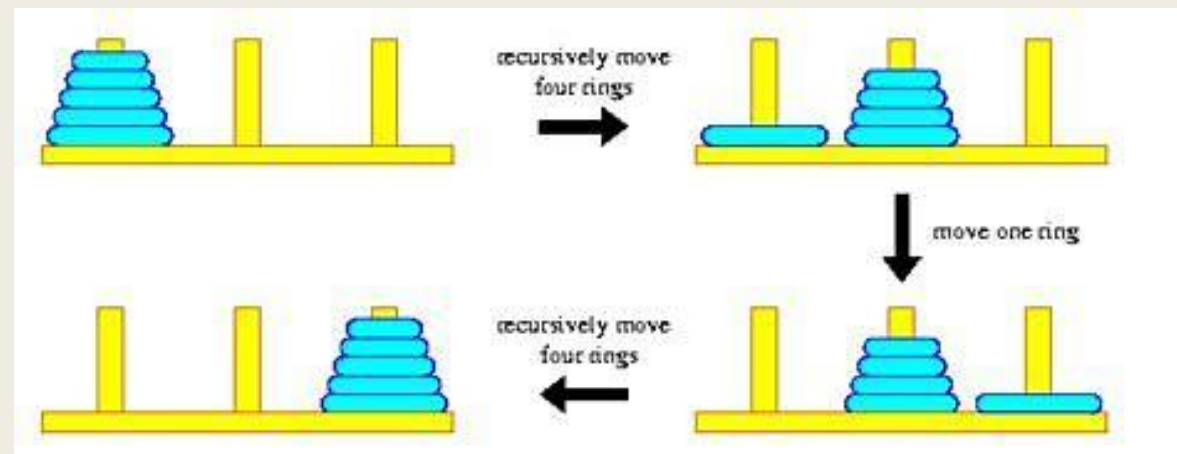
```
public int main(String args[]) {  
    System.out.println(factorial(2));  
}
```

```
static int factorial(int n) {  
    if (n == 0) {  
        return 1;  
    }  
    return n * factorial(n-1); // 4 * factorial(3):  
                                // 3 * factorial(2):  
                                // 2 * factorial(1)  
                                // 1 * factorial(0)
```

4

5

```
public int factorial(int 0) {  
    if (0 == 0)  
        return 1;  
    return 0 * factorial(0-1);  
}
```



# I Вариант

```
/*  
  When recursion?  
    Problem addressed via similar sub-problems  
  
    e.g., Binary Search, Towers of Hanoi, Word Frequency Count  
  
    int[] a = {11, 19, 24, 34, 55, 65, 71, 83, 91};  
    binarySearch(a, 0, 8, 65):  
        mid = 0 + 8/2 = 4,
```

# II Вариант

```
/*  
  When recursion?  
    Problem addressed via similar sub-problems  
  
    e.g., Binary Search, Towers of Hanoi, Word Frequency Count  
  
    int[] a = {11, 19, 24, 34, 55, 65, 71, 83, 91};  
    binarySearch(a, 0, 8, 65):  
        mid = 0 + 8/2 = 4, binarySearch(a, 5, 8, 65):  
            mid = 13/2 = 6, binarySearch(a, 5, 5, 65)|
```

# Програмна реализация

```
// O(log(n)):  
public static int binarySearch(int a[], int l, int h, int key) {  
    if (l == h) {  
        if (key == a[l]) {  
            return l;  
        } else {  
            return -1;  
        }  
    }  
    int mid = (l + h)/2;  
  
    if (key == a[mid]) {  
        return mid;  
    } else if (key > a[mid]) {  
        return binarySearch(a, l: mid+1, h, key);  
    } else {  
        return binarySearch(a, l, h: mid-1, key);  
    }  
}
```

```
public static void main(String[] args) {  
  
    int[] a = {11, 19, 24, 34, 55, 65, 71, 83, 91};  
    int index = binarySearch(a, l: 0, h: 8, key: 65);  
    System.out.println("index: " + index);  
}
```



# Заклучение

## Оператори за аритметични операции

- Операнди – *primitive numeric* типове
- ПРАВИЛА
  - Изпълнение на операциите
  - Промотъри на операциите
  - Операнди с еднакъв тип
  - Миксирани операнди

- Оператори за сравнение

- `==` и `!=` използват се за сравнение на референции на обекти

- Логически оператори

- Тестване на множество условия
  - `&&` и `||` са оператори ***short-circuted***

# Операции за побитово сравнение

- Използват се в приложение с ограничени ресурси
- Побитовите операнди – **integer** & **boolean** примитиви
- Побитови операции с отместване – **integer** примитиви
  - *Изместване на ляво* – умножение с число, степен на 2
  - *Безнаково отместване на дясно* – делене на число, което е степен на 2
- Приложения : компилатори, хеш таблици, програмиране на всградени системи и игри

- Switch statements
- Кога се предпочита switch вместо if?
  - *Ясност и четливост на кода*
  - *Бързодействие*
  - *Използва се при много голям брой избори*

# Преминахме към ефективно програмиране на JAVA

- Разгледахме циклите и обяснихме предимствата на използването на **for-each** пред традиционните цикли
- Разгледахме for циклите и посочихме предимствата им при използването вместо while цикли

# ЛЕКЦИЯ: Packages & Information Hiding



- JAVA API



- Own packages

**Information  
Hiding**



- String manipulation



- Modifiers