

# Програмування-1

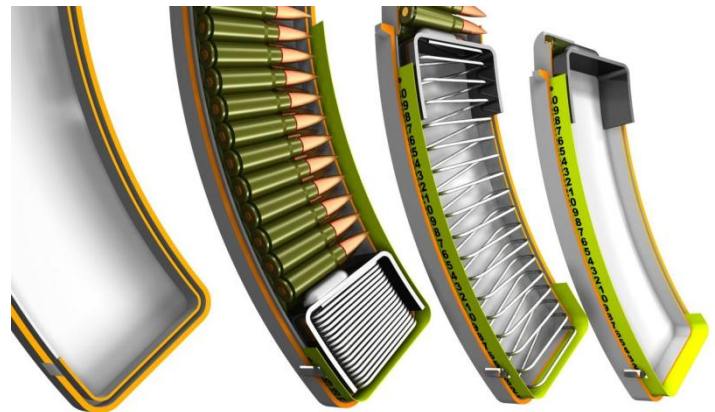
## Лекція 4

# Agenda

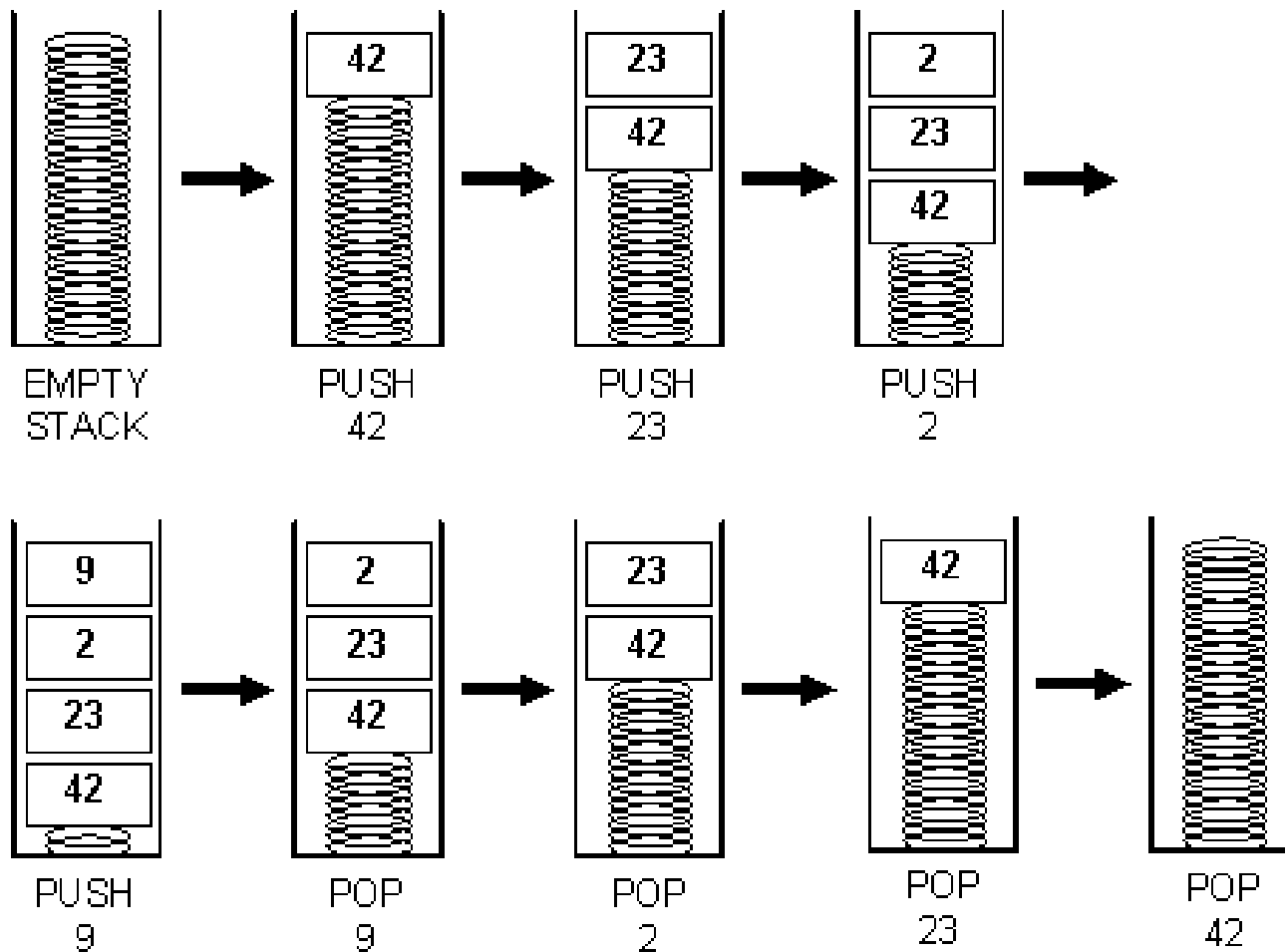
- Стек
- Передача параметрів та повернення значень
- Куча
- Посилання на об'єкти
- Масиви

# Стек

- а.к.а **Stack**
- Структура даних, що працює за принципом **LIFO** (**L**ast **I**n **F**irst **O**ut)
  - а.к.а. **FILO** (**F**irst **I**n **L**ast **O**ut)
- Дві базові операції:
  - **PUSH**
  - **POP**



# Як працює стек



# Як насправді працює стек

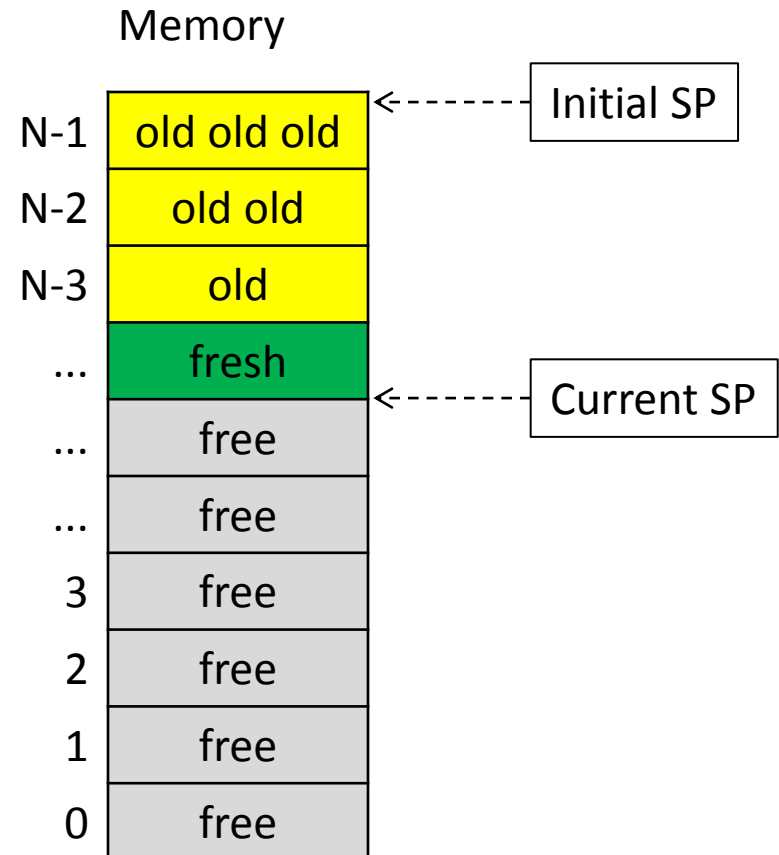
## PUSH value

$SP = SP - 1$   
 $Mem[SP] = \text{value}$

## POP

$\text{value} = Mem[SP]$   
 $SP = SP + 1$   
return value

\* SP – Stack Pointer



# Для чого потрібен стек?

- Передача параметрів у метод
- Запис адреси повернення з метода
- Пам'ять під локальні змінні метода
  - В стеку можуть зберігатися лише примітиви та посилання на об'єкти
  - Об'єкти в стеку зберігатися не можуть
  - Об'єкти зберігаються у купі

# Виклики методів

```
public static void main(String[] args) {  
    int firstParam = 2;  
    int dummyValue = 42;  
    printSum(firstParam, 3);  
    System.out.println("Done!");  
}  
  
public static void printSum(int a, int b) {  
    int result = a + b;  
    System.out.println(result);  
}
```

# Виклики методів

```
public static void main(String[] args) {  
    int firstParam = 2;  
    int dummyValue = 42;  
    printSum(firstParam, 3);  
    System.out.println("Done!");  
}  
  
public static void printSum(int a, int b) {  
    int result = a + b;  
    System.out.println(result);  
}
```

args
Адреса повернення з main()
firstParam = 2
dummyValue = 42
2
3
Адреса повернення з printSum()
result = 5
5
Адреса повернення з println()
...
...
...

?



Параметри  
Локальні змінні  
Адреси повернення

# Питання: що побачимо на екрані?

```
public static void main(String[] args) {  
    int a = 42;  
    System.out.println(a);  
    tryToChange(a);  
    System.out.println(a);  
}
```

```
public static void tryToChange(int a) {  
    System.out.println(a);  
    a = 13;  
    System.out.println(a);  
}
```

# Питання: що побачимо на екрані?

```
public static void main(String[] args) {  
    int a = 42;  
    System.out.println(a);  
    tryToChange(a);  
    System.out.println(a);  
}
```

```
public static void tryToChange(int a) {  
    System.out.println(a);  
    a = 13;  
    System.out.println(a);  
}
```

42  
42  
13  
42

```
graph LR; subgraph tryToChange; TC1[System.out.println(a)] --> O1[42]; TC2[System.out.println(a)] --> O2[42]; TC3[System.out.println(a)] --> O3[13]; end; subgraph main; M1[System.out.println(a)] --> O1; M2[tryToChange(a)] --> TC1; M3[System.out.println(a)] --> O4[42]; end;
```

# Hello, Stack Overflow ☺

```
17 public static void main(String[] args) {
18     stackOverflow(0);
19 }
20
21 public static void stackOverflow(int counter) {
22     counter++;
23     System.err.println("Hello, Stack " + counter);
24     stackOverflow(counter);
25 }
```

javaapplication2.JavaApplication2 >>

Вывод - JavaApplication2 (run) %

```

Hello, Stack 9048
Hello, Stack 9049
Hello, Stack 9050
Hello, Stack 9051Exception in thread "main" java.lang.StackOverflowError
    at sun.nio.cs.UTF_8$Encoder.encodeLoop(UTF_8.java:691)
    at java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:579)
    at sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:271)
    at sun.nio.cs.StreamEncoder.write(StreamEncoder.java:125)
    at java.io.OutputStreamWriter.write(OutputStreamWriter.java:207)
    at java.io.BufferedWriter.flushBuffer(BufferedWriter.java:129)
    at java.io.PrintStream.write(PrintStream.java:526)
    at java.io.PrintStream.print(PrintStream.java:669)
    at java.io.PrintStream.println(PrintStream.java:806)
    at javaapplication2.JavaApplication2.stackOverflow(JavaApplication2.java:23)
    at javaapplication2.JavaApplication2.stackOverflow(JavaApplication2.java:24)
    at javaapplication2.JavaApplication2.stackOverflow(JavaApplication2.java:24)
```

# Купа



- а.к.а. **Heap, Куча**
- динамічно розподілювана пам'ять
- у купі можна зберігати **лише об'єкти**
  - примітиви неможна зберігати у купі
  - але примітиви можна зберігати в середині об'єктів у купі 😊
- Виділяємо пам'ять та створюємо об'єкти за допомогою оператора **new**
- «В ручну» очищати пам'ять в Java не потрібно
  - цим автоматично займається **Garbage Collector**
  - якщо не тримати посилання на непотрібні об'єкти, то вони скоріше стануть доступними для знищення

# Memory

- Stack + Heap



# Масиви

- Масиви призначені для зберігання великої кількості однотипних даних
- В Java **масиви є об'єктами**
  - як усі об'єкти зберігаються у купі (**heap**)
  - ім'я масиву = посилання на масив
  - масиви мають поле **length** (**read-only**) та методи (від **Object**)
- Поле **length** - **розмір** масиву (**у елементах**)
- Нумерація комірок **zero-based !!!**
  - **перший** елемент знаходиться у комірці № **0**
  - **останній** елемент знаходиться у комірці № (**length-1**)
- В Java **неможливо вийти за границі масиву**
  - якщо спробувати, буде викинуто **IndexOutOfBoundsException**
- **Розмір масиву змінити неможна**
  - але можна створити новий масив іншого розміру, скопіювати в нього дані і використовувати посилання на нього замість посилання на старий масив

# Масиви – Декларування масивів

- Кращій варіант:

```
тип [] назва;
```

- Компілюється, але так робити не треба:

```
тип []назва; // те ж саме
```

```
тип назва[]; // майже те ж саме
```

- Приклад:

```
int []a, b; // 2 масиви: a та b
```

```
int c[], d; // масив c та змінна d
```

```
int[] e, f; // 2 масиви (пробіли впливають лише на читабельність)
```

```
String[] s; // для посилань на об'єкти масиви декларуються так само
```

# Масиви – Створення

- Створення пустого масиву:

```
int[] a;  
a = new int[10];
```



Обов'язково вказується розмір масиву

```
int[] b = new int[10];
```



Заповнюється значеннями за замовчуванням:

**0** – для чисел

**false** – для **boolean**

**null** – для посилань на об'єкти

- Створення + ініціалізація

```
int[] a = {1, 2, 3, 4, 5};
```

```
int[] b = new int[] {2, 4, 6};
```



Розмір масиву не вказується  
(підраховується компілятором)

# Багатомірні масиви

- В Java багатомірний масив = масив посилань на інші масиви

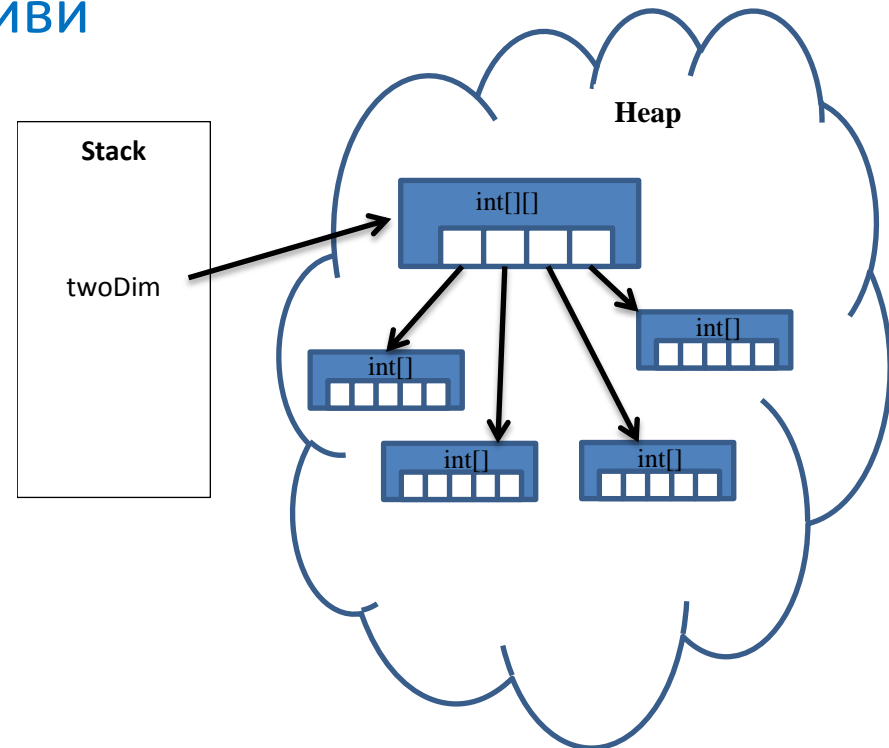
```
int[][] twoDim = new int [4][];  
twoDim[0] = new int[5];  
twoDim[1] = new int[5];  
twoDim[2] = new int[5];  
twoDim[3] = new int[5];
```



Можна створювати непрямокутні матриці

- Для прямокутних масивів можна простіше:

```
int[][] twoDim = new int [4][5];
```



# Масиви – Зміна розміру

- Розмір масиву змінити **неможна**
- **Можна** створити новий масив, скопіювати в нього елементи та почати використовувати посилання на нього замість старого посилання

```
int[] elements = { 1, 2, 3, 4, 5 }; // масив на 5 елементів
```

```
int[] tmp = new int[10];  
System.arraycopy(elements, 0, tmp, 0, elements.length);  
elements = tmp;
```

```
for (int e : elements) {  
    System.out.println(e);  
}
```



for-each цикл

Питання?