Област на видимост и съществуване на променливите в JAVA

Наименования на променливи, методи, класове

- Наименованията на променливите използват стила camelCase.
- Първата буква на всяка дума от името на класа е главна – ComputeArea
- Константите са с главни букви - PI, MAX_VALUE

- Важно е да се спазва правилото за наименуване, тъй като програмите ви стават по-четливи.

Променливи в КЛАС

```
class VariableScope {
    int i;
    int j = i;
    void foo () {
        j = k;
    }
    j = k; // invalid
    int k;
}
```

Област на локални променливи

```
void foo(int p) {
    …
    int x = 0, z = x;
    if (x == 0) {
        int y = x;
        int x = 0; // invalid
    }
    y++; // invalid
}
```

*scope of **x, z***

*scope of **p***

Област на използване на локалните променливи

```
void foo() {
    …
    int x = 0;
    for (int x = 0; x < 1; x++) { // invalid
        …
    }
}
```

Shadowing Class-level Variables

```
int x = 10;
void foo() {
    …
    int x = 0; // shadows class variable x
    x++;
    this.x++;
}
```

class-level

local

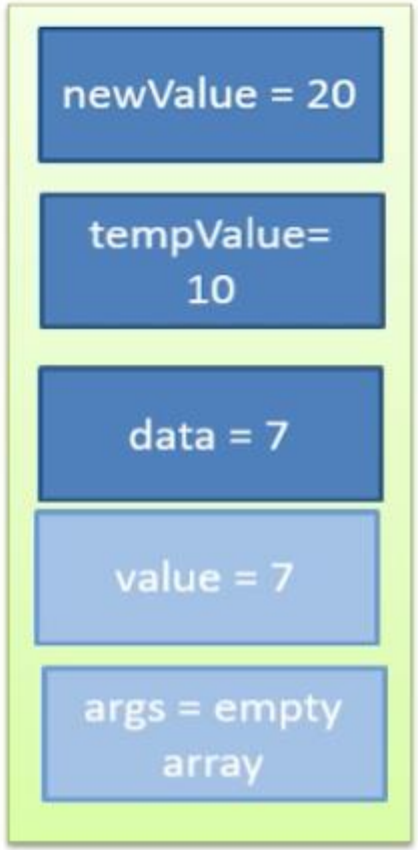
Scope of Local Variables

Methods invoked from current block will have *new* scope

```
void foo() {
    …
    int x = 0;
    bar();  ──────────→  void bar() {
    …                        x++; // invalid
}                        }
```

JAVA Memory

```
public class Main
{
    public static void main(String[] args)
    {
        int value = 7;
        value = calculate(value);
    }

    public static int calculate(int data)
    {
        int tempValue = data + 3;
        int newValue = tempValue * 2;
        return newValue;
    }
}
```



JAVA Memory - 1

```
public class Main
{
    public static void main(String[] args)
    {
        int value = 7;
        value = calculate(value);
    }

    public static int calculate(int data)
    {
        int tempValue = data + 3;
        int newValue = tempValue * 2;
        return newValue;
    }
}
```

value = 7

args = empty array

JAVA Memory - 2

```
public class Main
{
    public static void main(String[] args)
    {
        int value = 7;
        value = calculate(value);
    }

    public static int calculate(int data)
    {
        int tempValue = data + 3;
        int newValue = tempValue * 2;
        return newValue;
    }
}
```

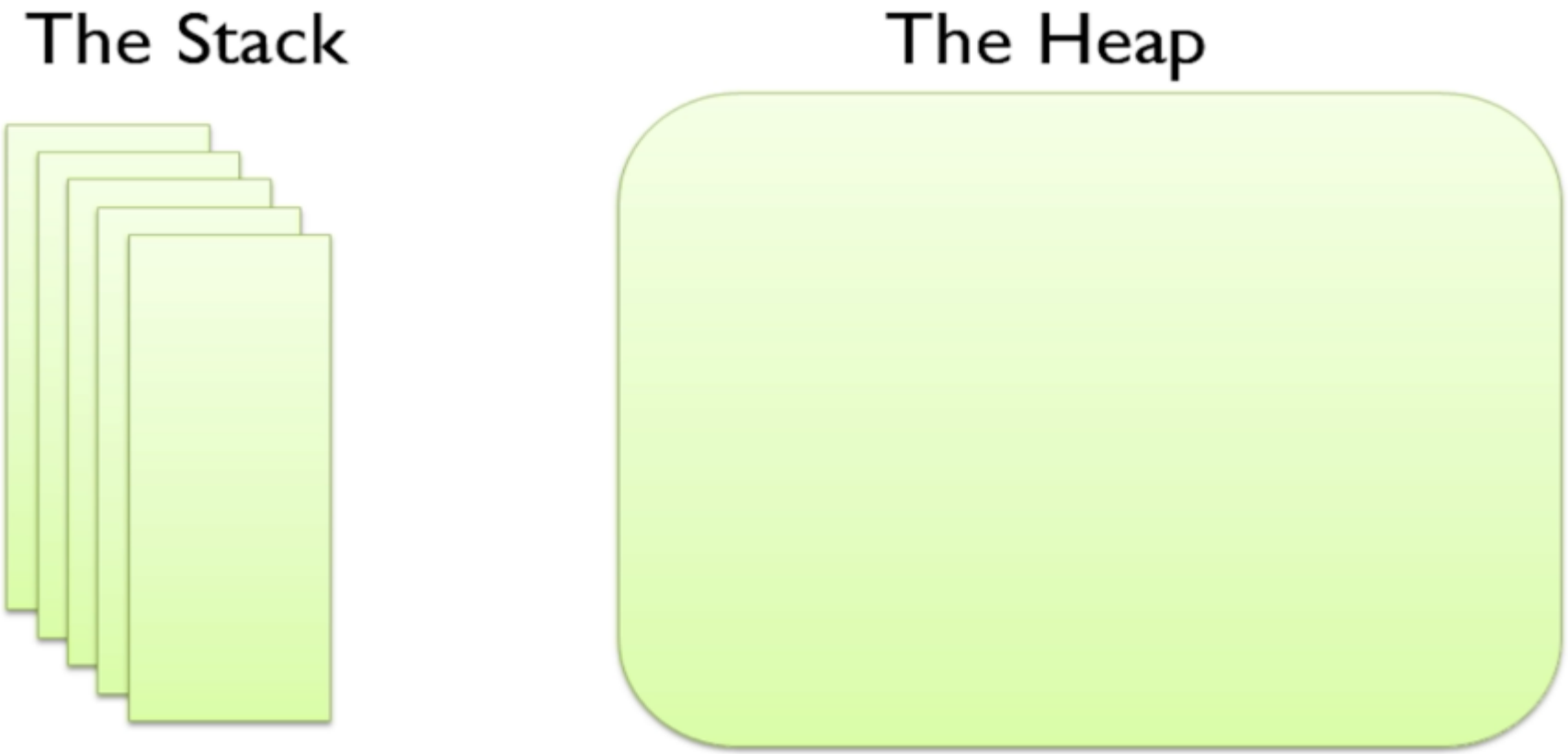
value = 20

args = empty array

JAVA Memory - 3

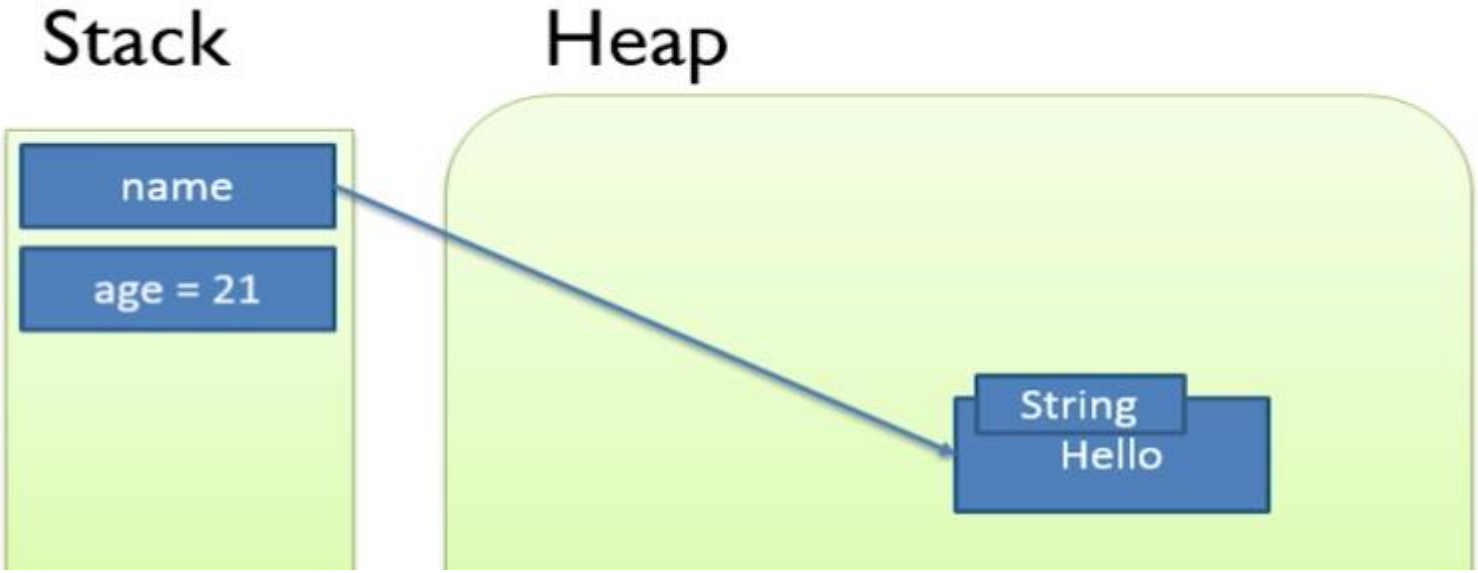
```java
public class Main
{
    public static void main(String[] args)
    {
        int value = 7;
        value = calculate(value);
    }

    public static int calculate(int data)
    {
        int tempValue = data + 3;
        int newValue = tempValue * 2;
        return newValue;
    }
}
```

Java Memory



Java Memory

```
int age = 21;
String name = "Hello";
```



Правила на Java паметта

- Обектите се записват в HEAP
- Променливите са референции към обектите
- Локалните променливи се записват в стека

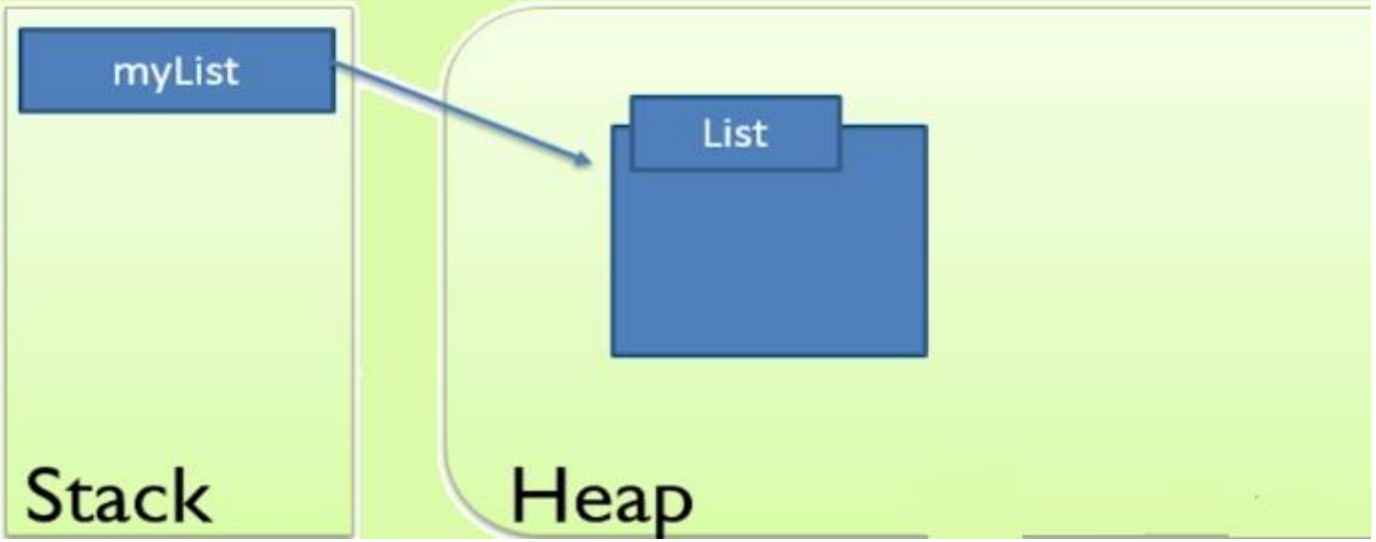
Общи променливи

```
public static void main(String[] args) {
      List<String> myList = new ArrayList<String>();
      myList.add("One");
      myList.add("Two");
      myList.add("Three");
      printList(myList);
   }

   public static void printList(List<String> data) {
      System.out.println(data);
   }
```

```
public static void main(String[] args) {
    List<String> myList = new ArrayList<String>();
    myList.add("One");
    myList.add("Two");
    myList.add("Three");
```

```
List<String> myList = new
  ArrayList<String>();
```

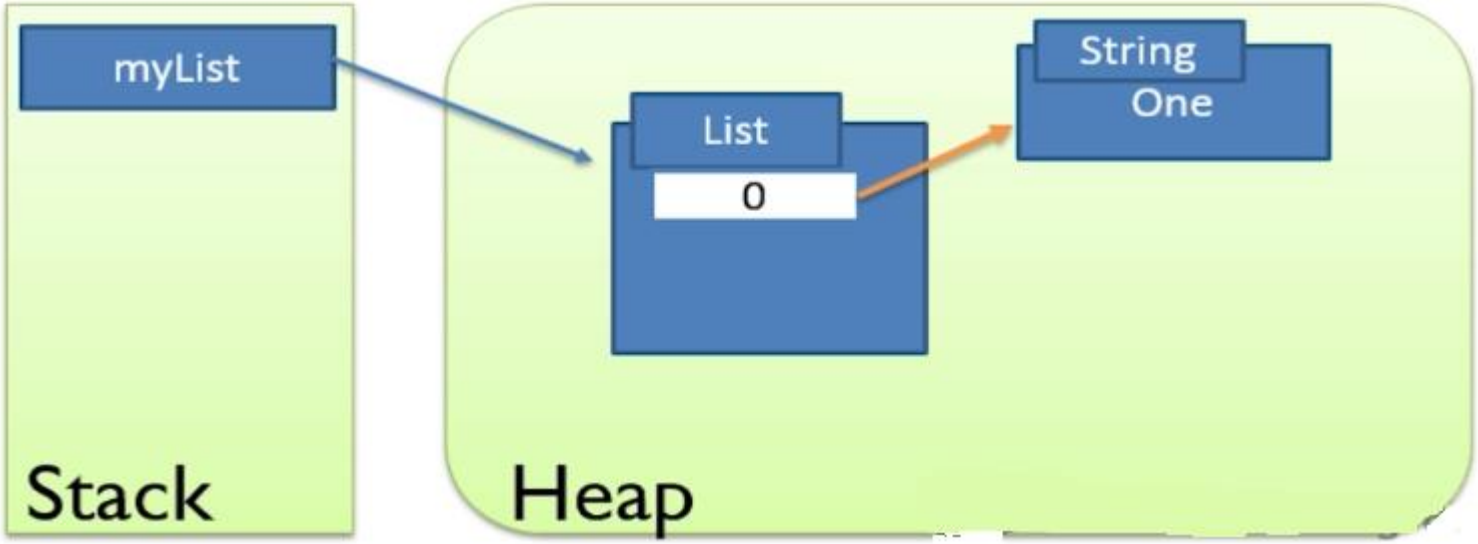
```
    System.out.println(data);
}
```



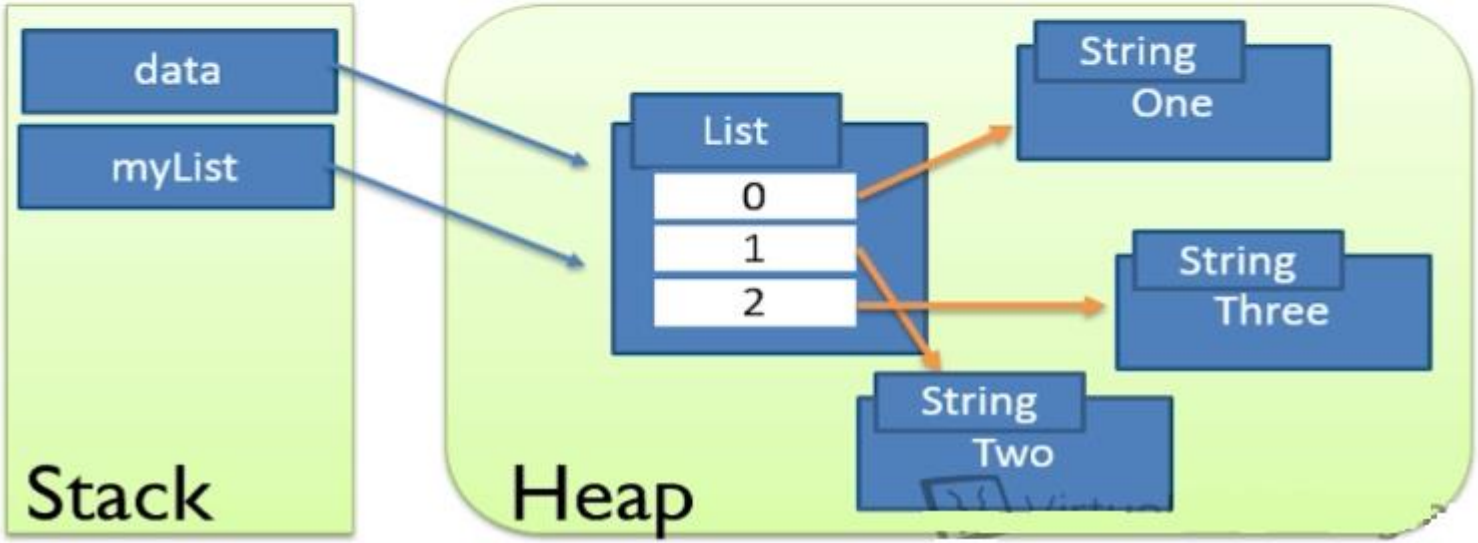
```
public static void main(String[] args) {
    List<String> myList = new ArrayList<String>();
    myList.add("One");
    myList.add("Two");
    myList.add("Three");
    printList(myList);
```

```
myList.add(new String("One"));
```

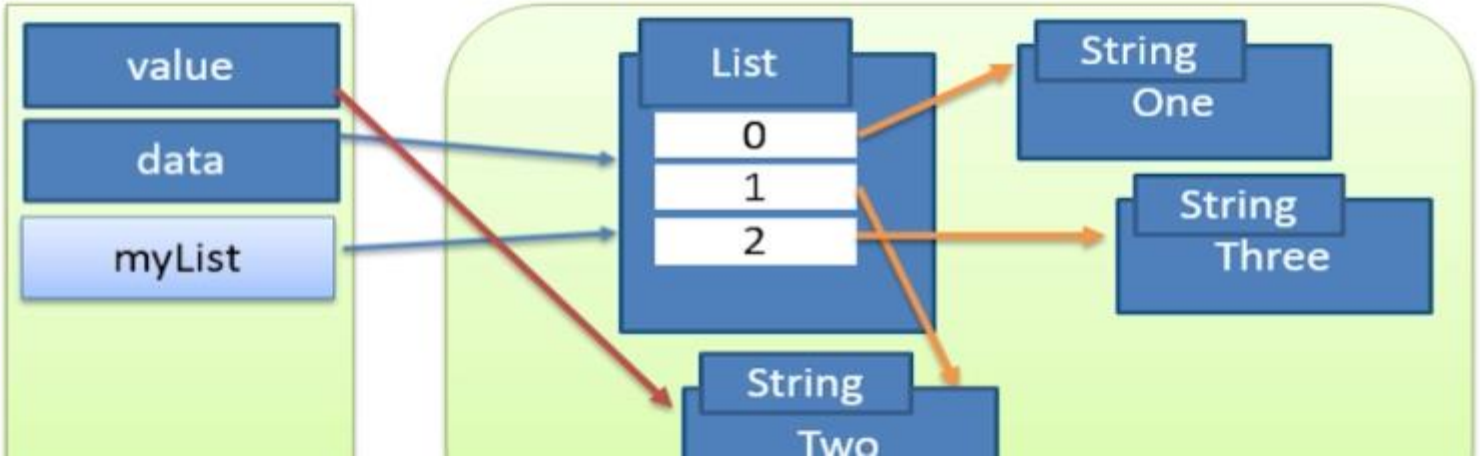
```
public static void printList(List<String> data) {
    System.out.println(data);
}
```



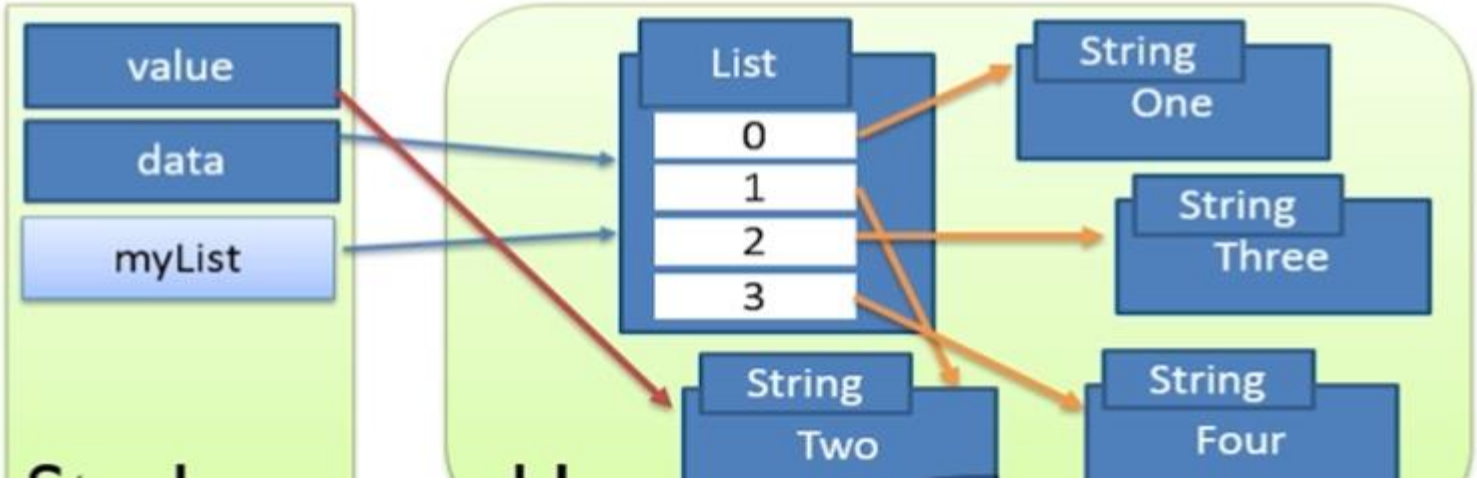
```
public static void main(String[] args) {
      List<String> myList = new ArrayList<String>();
      public void printList(List<String> data)
      printList(myList);
   }

   public static void printList(List<String> data) {
      System.out.println(data);
   }
```



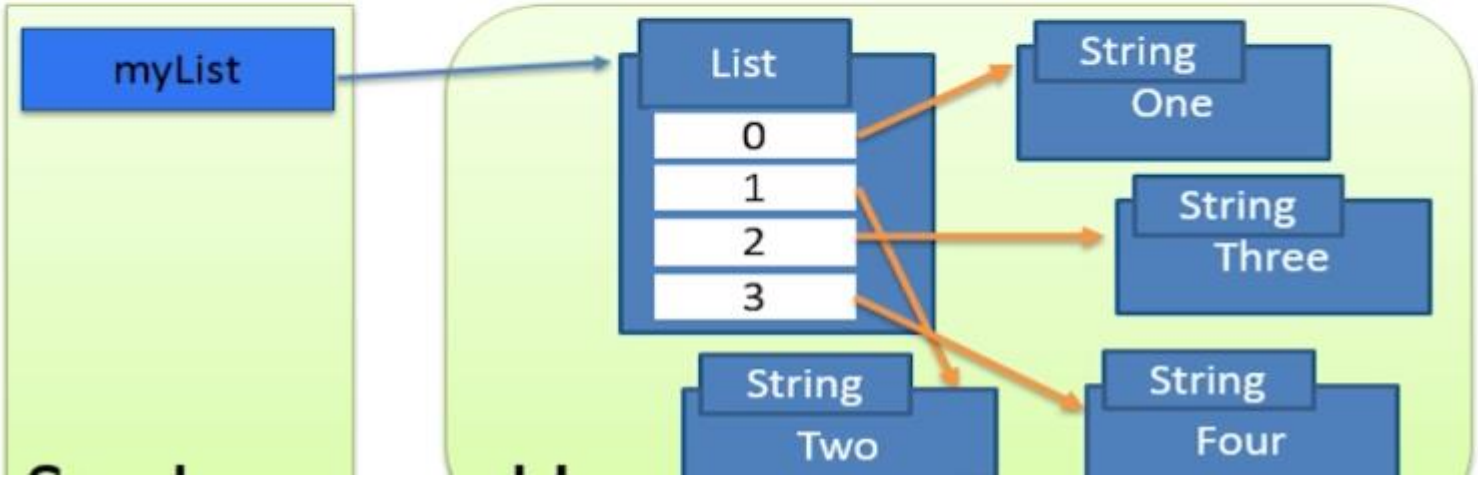
```java
public static void printList(List<String> data) {
    String value = data.get(1);
    data.add("Four");
    System.out.println(value);
}
```



```java
public static void printList(List<String> data) {
    String value = data.get(1);
    data.add("Four");
    System.out.println(value);
}
```



```java
public static void printList(List<String> data) {
    String value = data.get(1);
    data.add("Four");
    System.out.println(value);
}
```



Какъв ще е резултата от изпълнението?

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Passing by value");
        int localValue = 5;
        calculate(localValue);
        System.out.println(localValue);
    }
    public static void calculate(int calcValue) {
        calcValue = calcValue * 100;

    }
}
```

Passing by reference (not possible!)

```
public static void main(String[] args)
    {
        int localValue = 5;
        calculate(localValue);

        System.out.println(localValue);
    }

    public static void calculate
                        (byref int calcValue)
    {
        calcValue = calcValue * 100;

    }
```

calcValue =
localValue =
500

Passing Values

```
public class Main
{
    public static void main(String[] args)
    {
        Customer c = new Customer("Sally");
        renameCustomer(c);
        System.out.println(c.getName());
    }

    public static void renameCustomer(Customer cust)
    {
        cust.setName("Diane");
    }
}
```

Passing Values

```
public static void main(String[] args) {
        Customer c = new Customer("Sally");
        renameCustomer(c);
        System.out.println(c.getName());
    }

public static void renameCustomer(Customer cust) {
        cust.setName("Diane");
    }
```

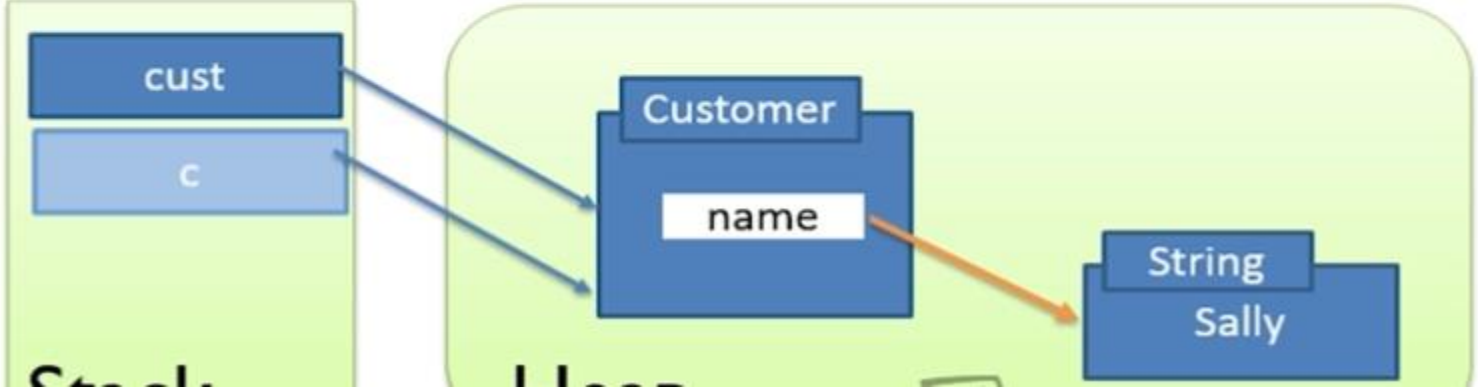
MUTABLE IN JAVA
VERSUS
IMMUTABLE IN JAVA

MUTABLE IN JAVA	IMMUTABLE IN JAVA
Ability of changing a string	Impossibility of changing a string
StingBuffer and StringBuilder are mutable	String is immutable

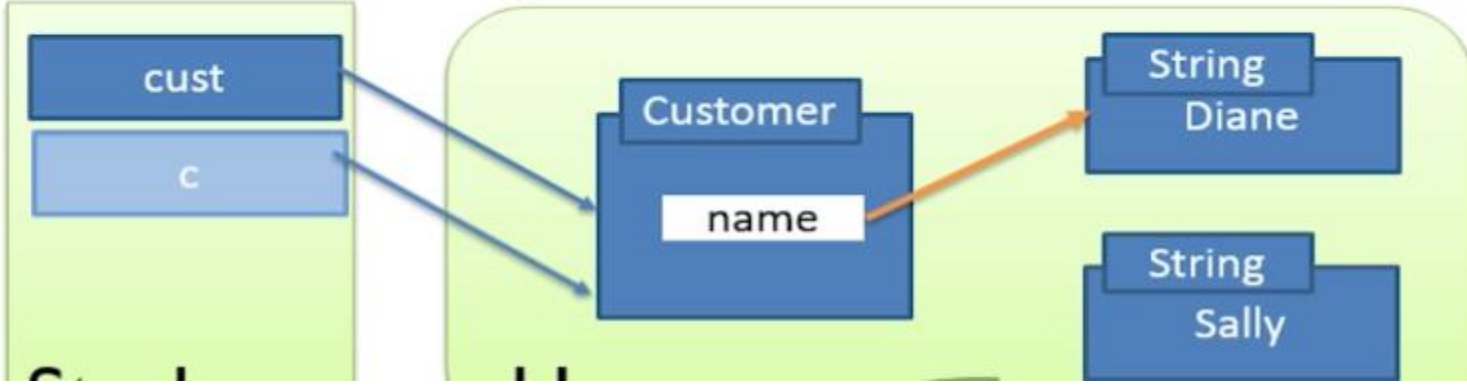
Visit www.PEDIAA.com

Mutable	Immutable
Fields can be changed after the object creation	Fields cannot be changed after object creation
Generally provides a method to modify the field value	Does not have any method to modify the field value
Has Getter and Setter methods	Has only Getter method
Example: StringBuilder, java.util.Date	Example: String, Boxed primitive objects like Integer, Long and etc

```java
public static void main(String[] args) {
        Customer c = new Customer("Sally");
        renameCustomer(c);
        System.out.println(c.getName());
    }

public static void renameCustomer(Customer cust) {
        cust.setName("Diane");
    }
```



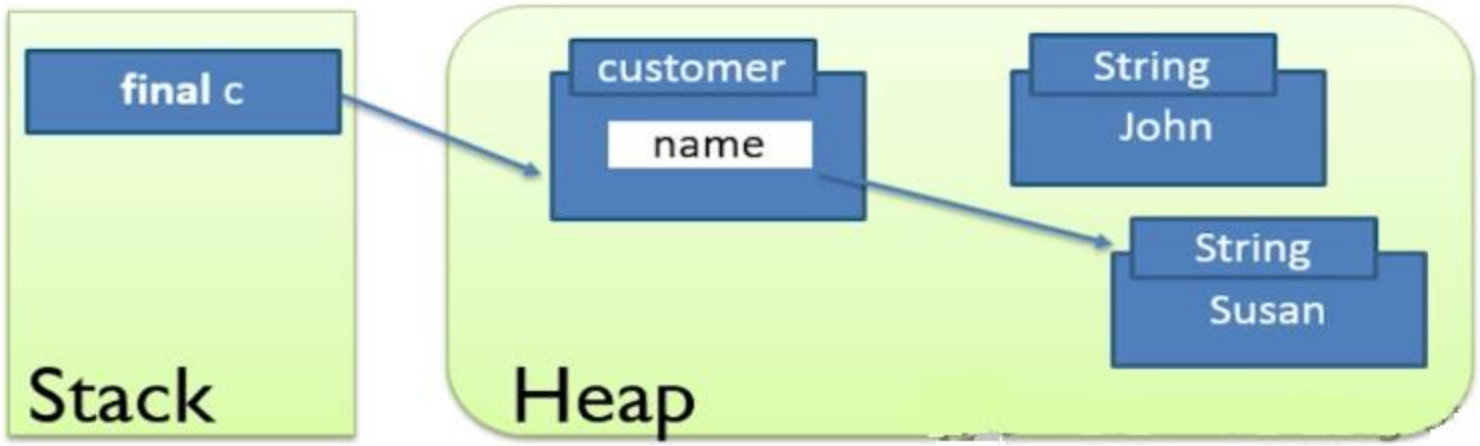
```java
public static void main(String[] args) {
        Customer c = new Customer("Sally");
        renameCustomer(c);
        System.out.println(c.getName());
    }

public static void renameCustomer(Customer cust) {
        cust.setName("Diane");
    }
```

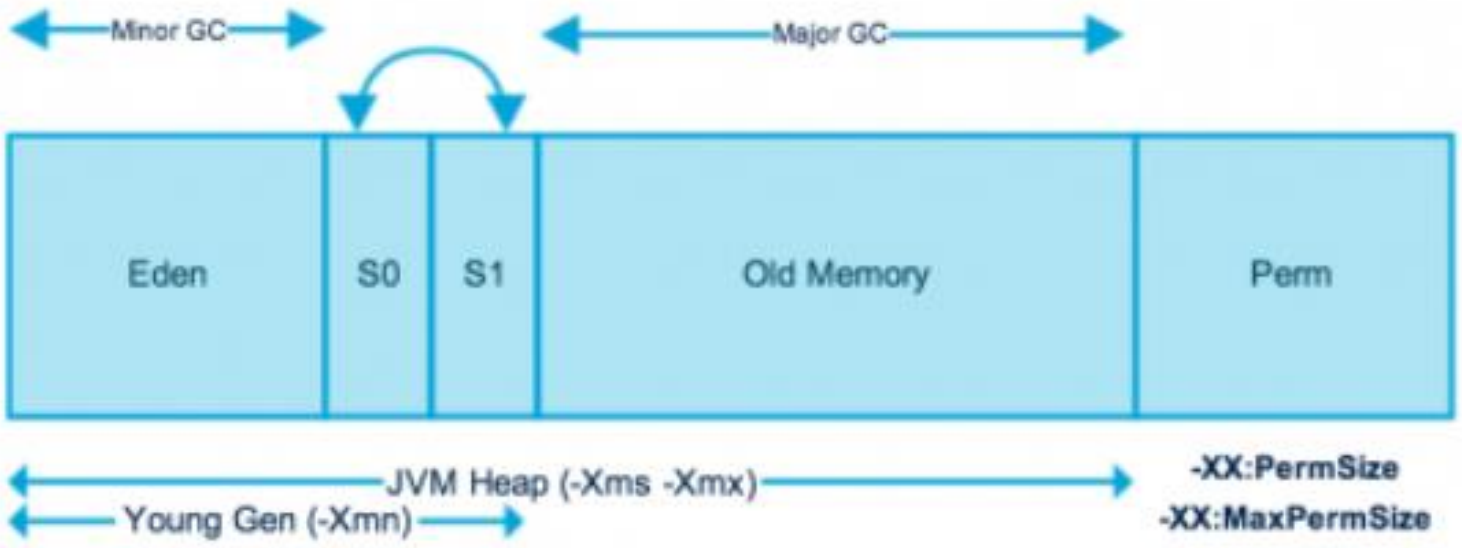


Константите в паметта.

```
final Customer c = new Customer("John");
```

```
final Customer c;
c = new Customer("John");
c.setName("Susan");
```




Minor GC
Major GC
Eden
S0
S1
Old Memory
Perm
JVM Heap (-Xms -Xmx)
Young Gen (-Xmn)
-XX:PermSize
-XX:MaxPermSize

JVM Model

VM Switch	VM Switch Description
-Xms	For setting the initial heap size when JVM starts
-Xmx	For setting the maximum heap size.
-Xmn	For setting the size of the Young Generation, rest of the space goes for Old Generation.
-XX:PermGen	For setting the initial size of the Permanent Generation memory
-XX:MaxPermGen	For setting the maximum size of Perm Gen
-XX:SurvivorRatio	For providing ratio of Eden space and Survivor Space, for example if Young Generation size is 10m and VM switch is -XX:SurvivorRatio=2 then 5m will be reserved for Eden Space and 2.5m each for both the Survivor spaces. The default value is 8.
-XX:NewRatio	For providing ratio of old/new generation sizes. The default value is 2.

Class Scope

```
public class ClassScopeExample {
    private Integer amount = 0;
    public void exampleMethod() {
        amount++;
    }
    public void anotherExampleMethod() {
        Integer anotherAmount = amount + 4;
    }
}
```

Method Scope

```
public class MethodScopeExample {
    public void methodA() {
        Integer area = 2;
    }
    public void methodB() {
        // compiler error, area cannot be resolved to a variable
        area = area + 2;
    }
}
```

Loop Scope

```
public class LoopScopeExample {
    List<String> listOfNames = Arrays.asList("Joe", "Susan", "Pattrick");
    public void iterationOfNames() {
        String allNames = "";
        for (String name : listOfNames) {
            allNames = allNames + " " + name;
        }
        // compiler error, name cannot be resolved to a variable
        String lastNameUsed = name;
    }
}
```

Bracket Scope

```
public class BracketScopeExample {
    public void mathOperationExample() {
        Integer sum = 0;
        {
            Integer number = 2;
            sum = sum + number;
        }
        // compiler error, number cannot be solved as a variable
        number++;
    }
}
```

Scopes and Variable Shadowing

```
1  public class NestedScopesExample {
2      String title = "Baeldung";
3      public void printTitle() {
4          System.out.println(title);
5          String title = "John Doe";
6          System.out.println(title);
7      }
8  }
```