Object Oriented Programming in Java

2: Objects creation
strings, arrays, program memory organization (stack and heap)

Licence

You are free to

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

under the following terms

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial** — You may not use the material for commercial purposes.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

- The samples and slides are inspired by the Object Oriented Programming course at the University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia.
  - Original materials in Croatian were created by (in alphabetical order): Ivica Botički, Marko Čupić, Mario Kušek, Boris Milašinović, and Krešimir Pripužić under CC-BY-NC-SA licence.
  - Adaption for this course is done by Boris Milašinović and shared under the same licence
  - https://creativecommons.org/licenses/by-nc-sa/4.0/

Importing Java files into an IDE

- Introductory examples had used *src* and *bin* folder and no IDE had been used
  - only *src* was included in git repository
- Source code does not contain an IDE's specific project data
  - it differs for Eclipse, Netbeans, etc…
  - new Java project should be created, source code copied, dependencies set up, … → quite inconvienient
- Solution:
  - use one of uniform build systems supported by majority of IDEs
  - *Apache Maven* is one of the possible choices
    - Alternatives: Gradle, …

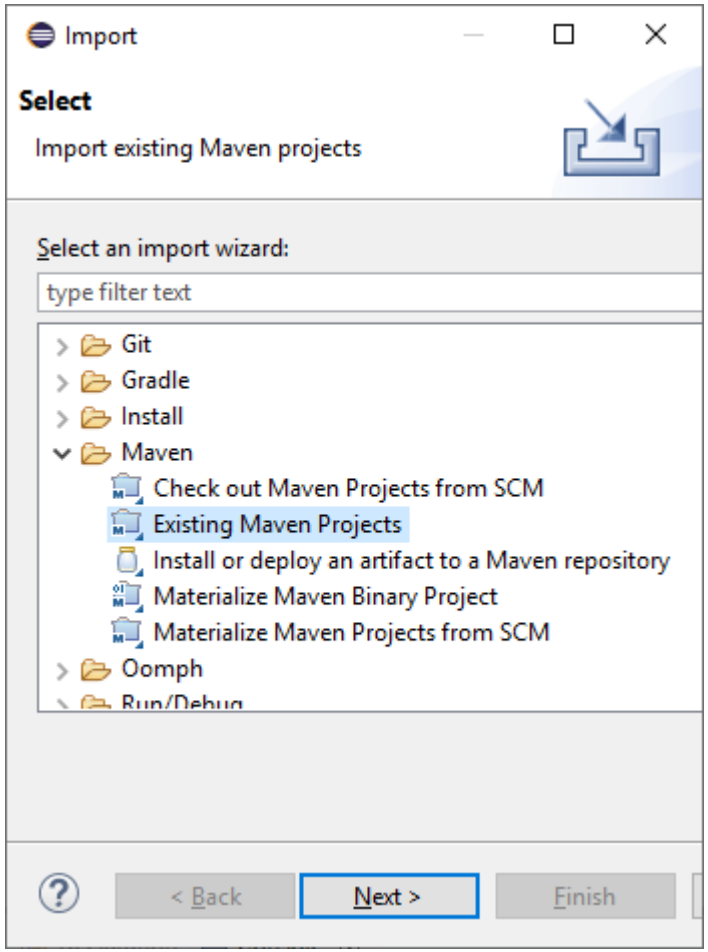
Structure of Maven projects

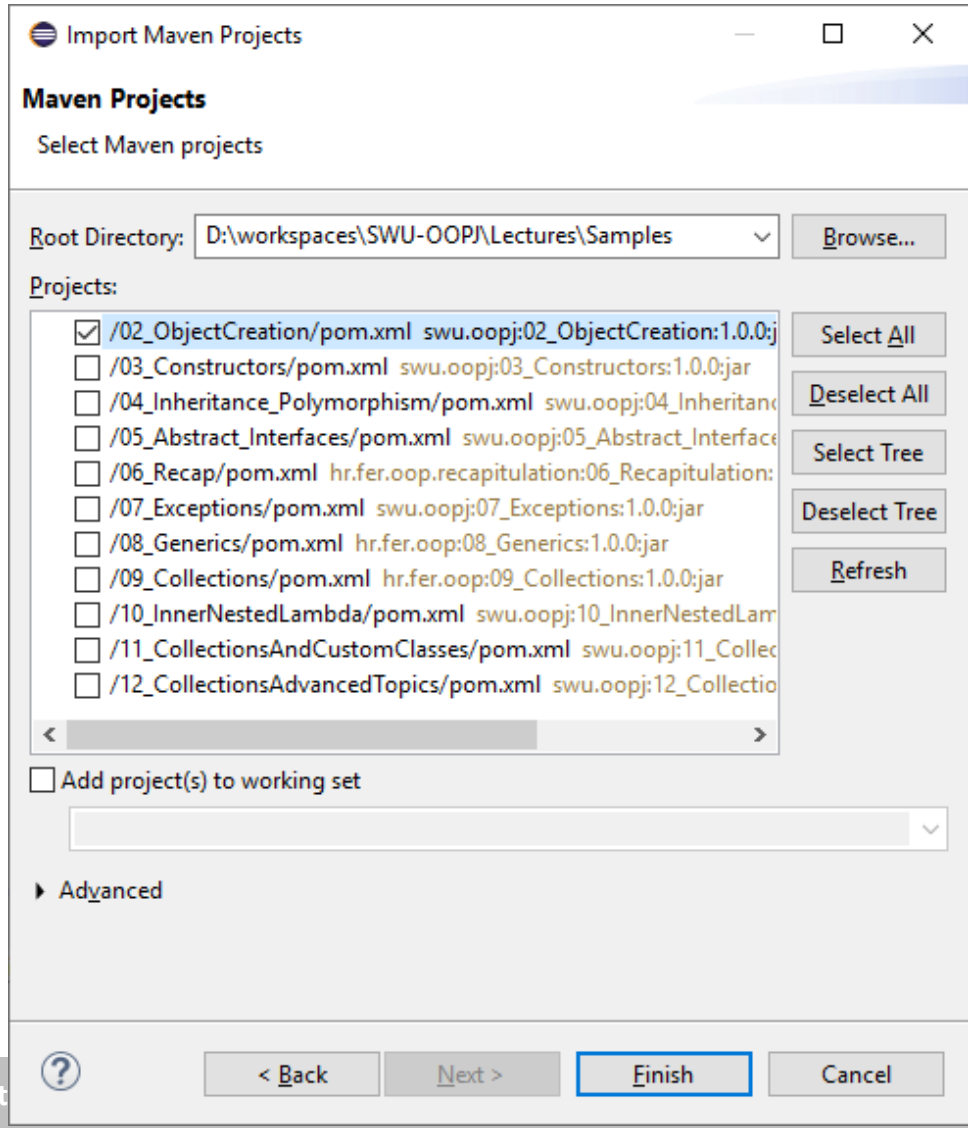
- Project's root folder contains pom.xml that configures
  - project identifiers
    - important for libraries shared at central Maven repository
  - dependencies on third party libraries
    - Maven or IDE downloads them if the do not exists locally
  - Java version that it should be used
  - …

- Maven project by convention use different folder structure
  - *src → src/main/java*
    - src may contains also other source file types (resources, tests, …)
  - *bin → target/classes*
    - *target* can contain other binaries, e.g. target/test-classes, …

Import Maven project to Eclipse

- Import → Existing Maven Projects → Choose projects to import

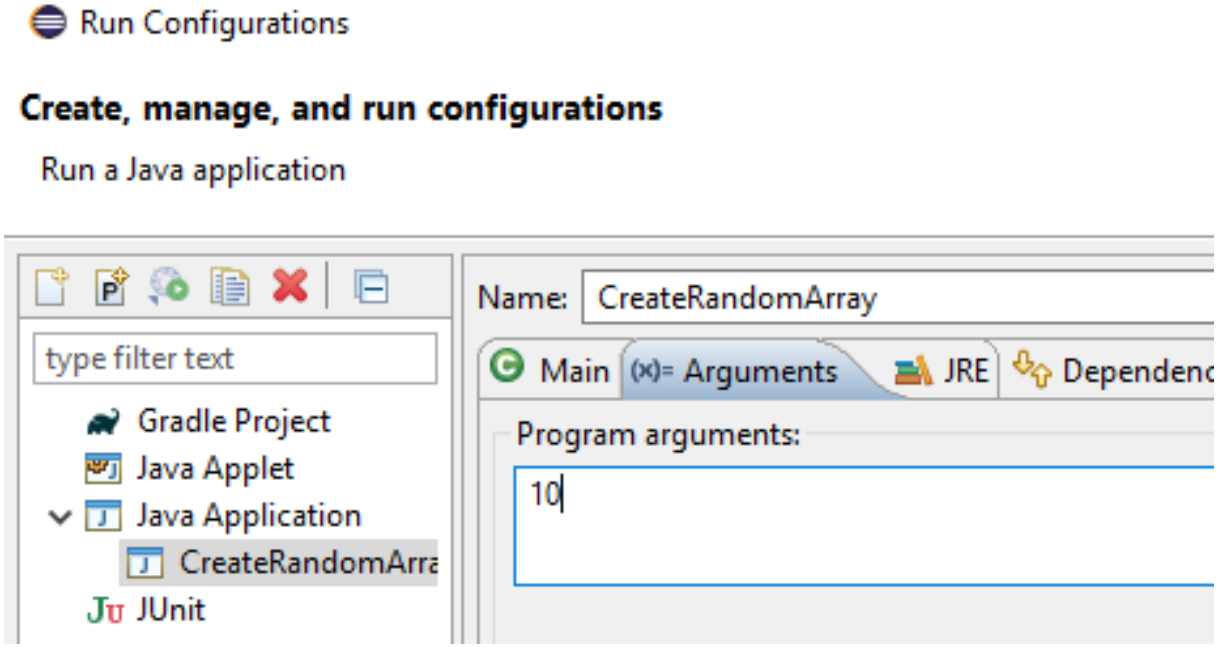




How to enter program arguments in Eclipse

- Right click on the source file containing main program that must be run → Run As → Run Configurations

- Enter desired arguments into text field in tab *Arguments*



Stack and heap

- Stack holds local variables, partial results, enables invocation of functions, recursion, …
  - calling a methods puts method arguments and return address on the program stack
  - method exists “removes” the element from the stack
  - declaring primitive type variable in a method take place on the stacks holding variable value
- Heap keeps other data
  - dynamically allocated data
  - program code
  - classes and methods information
  - constants

Arrays in Java

- Arrays are declared with `type[] variable_name;`
  - Array declaration does not create an array!
  - It create a place (typically 32-bit or 64-bit) on the program stack for declared variable.
    - Variable value would be an address of a continuous block on heap for storing array elements created later with keyword *new*
    - Initially value is *null* meaning that currently that reference does not refer to any object.

Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

- (Very) simplified memory state after step 1 in case method is called with n=5

Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

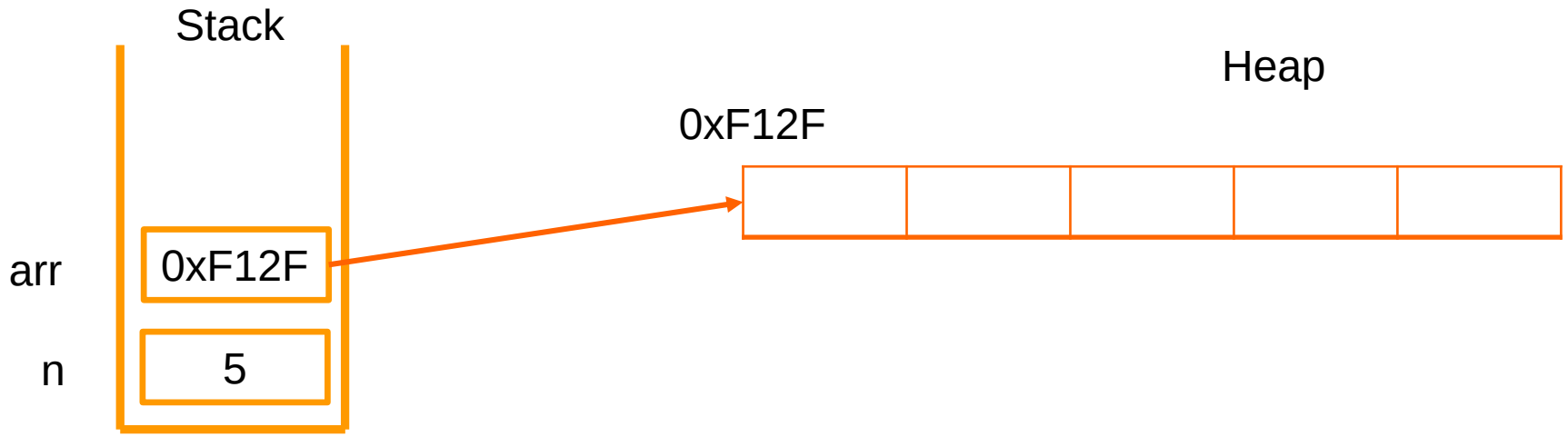
- Simplified memory state after step 2

Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

- step 3 - for illustration, suppose that place for array is created at address 0xF12F

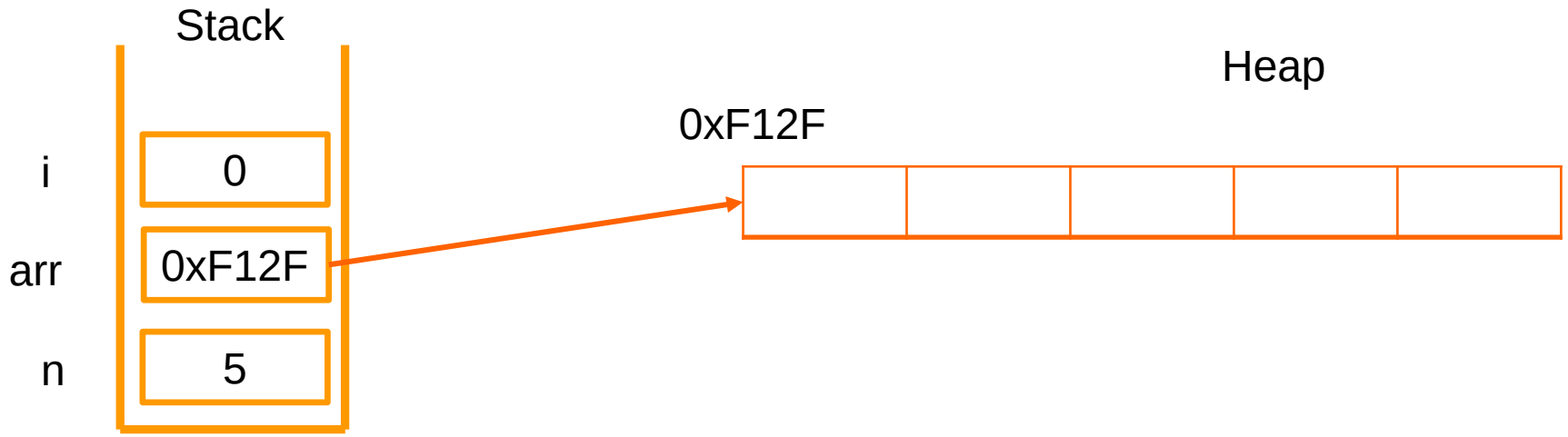


Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

- step 4 – initial step

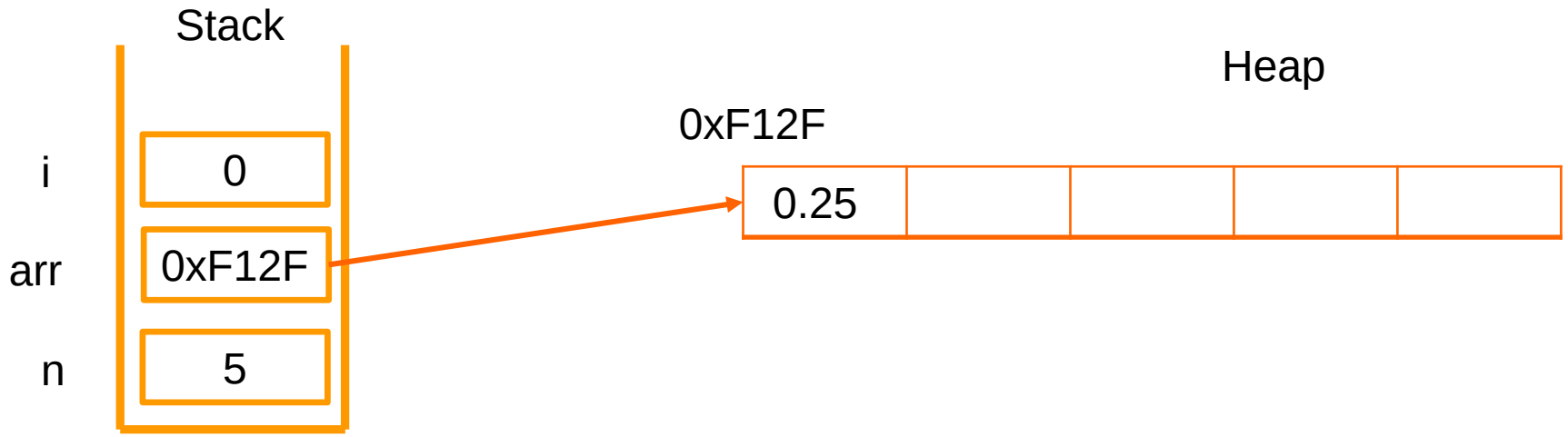


Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

- step 5 – when *i* is 0, and random number is 0.25

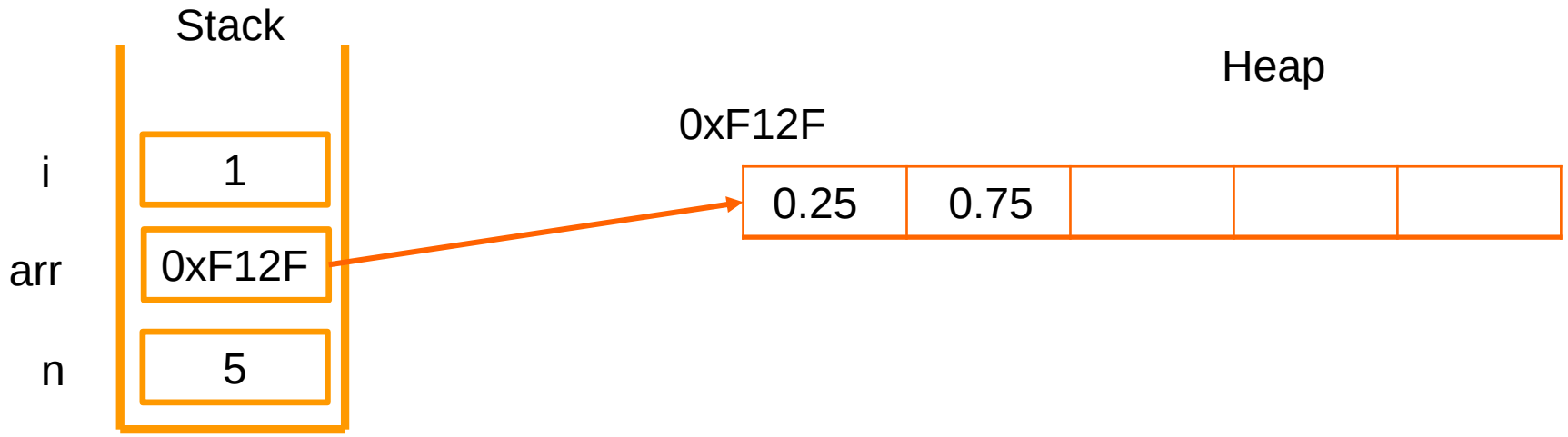


Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

- step 5 – when *i* is 1, and random number is 0.75

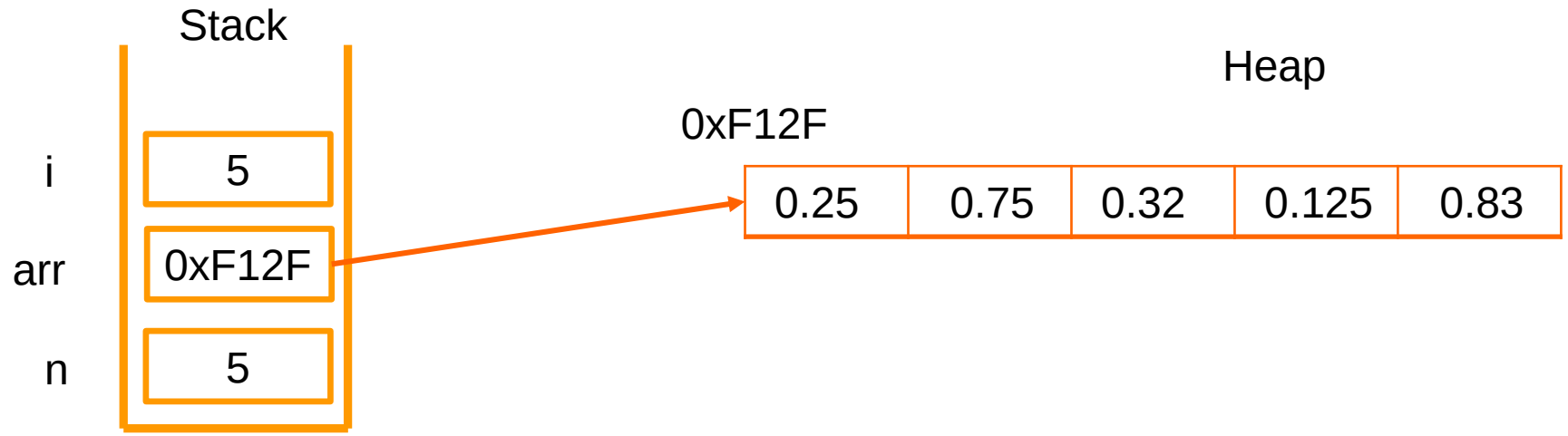


Creating an array in Java

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/swu/oopj/objectcreation/CreateRandomArray.java

- step 6 – what `return arr` means → it returns value 0xF12F

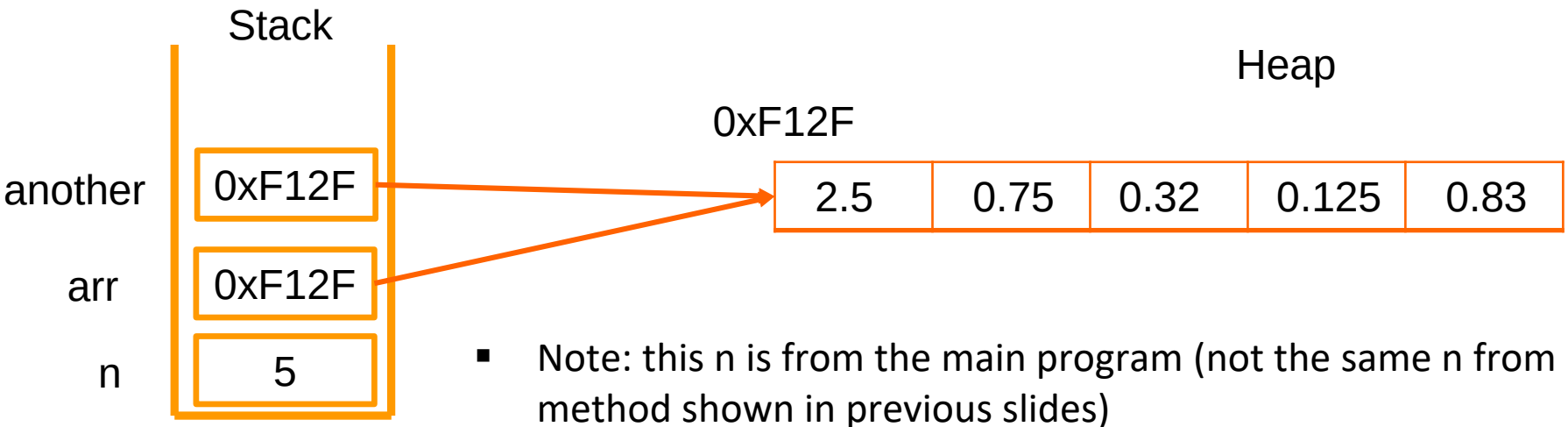


operator = copies values from/to the stack

- operator copy values on the stack and does not clone a data

```
double[] arr = create(n);
double[] another = arr; //what happens here?
another[0] = 2.5;
System.out.printf("#1 = %.4f %.4f %n", arr[0], another[0]);
```

…02_ObjectCreation/…/CreateRandomArray.java

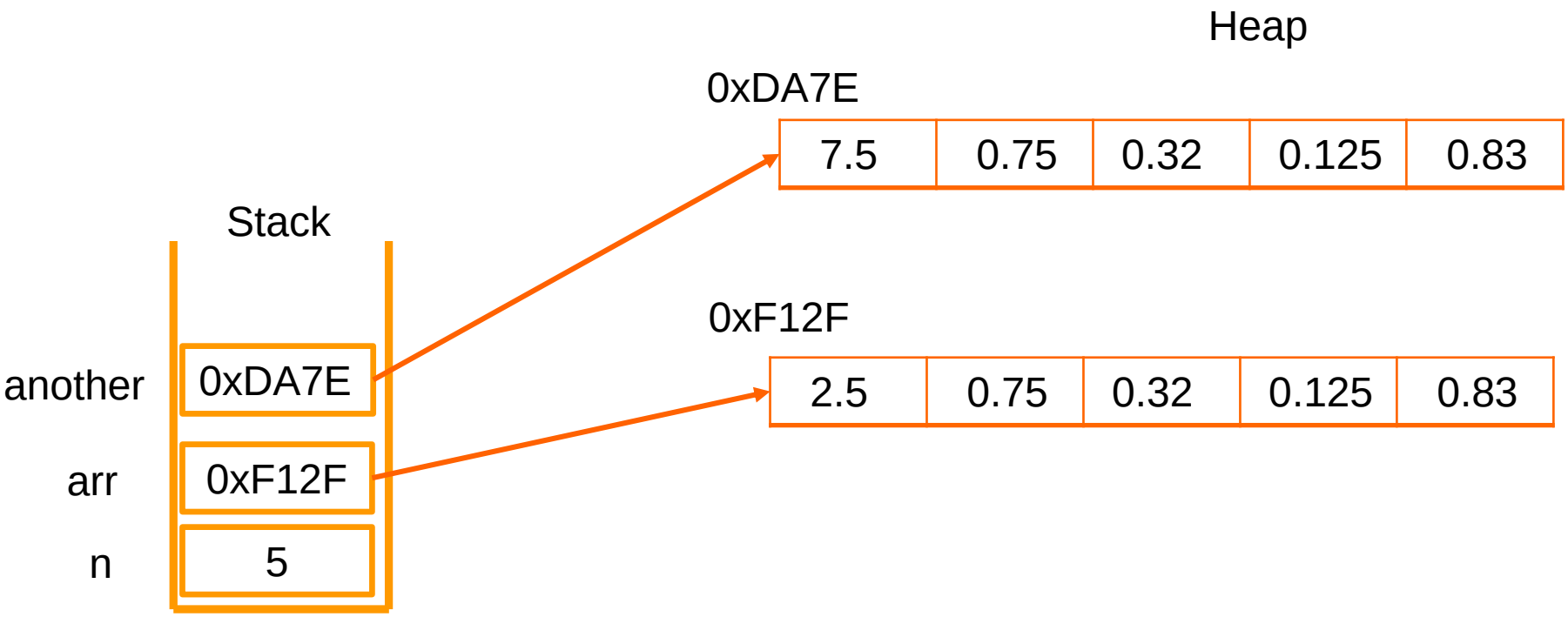


- Note: this n is from the main program (not the same n from method shown in previous slides)

operator = copies values from/to the stack

- operator copy values on the stack and does not clone a data

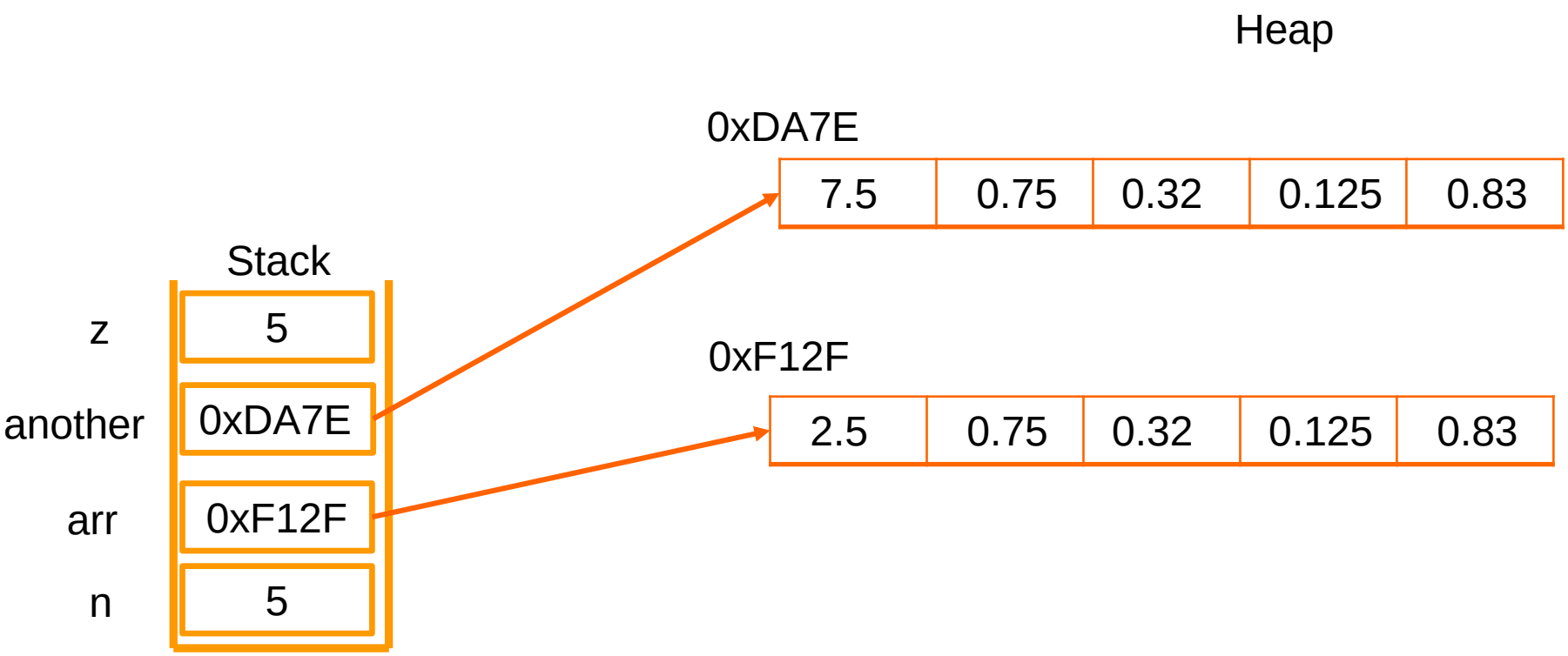
```
another = arr.clone(); //what happens here?
another[0] = 7.5;
System.out.printf("#2 = %.4f %.4f %n", arr[0], another[0]);
```



operator = copies values from/to the stack

- operator copy values on the stack and does not clone a data

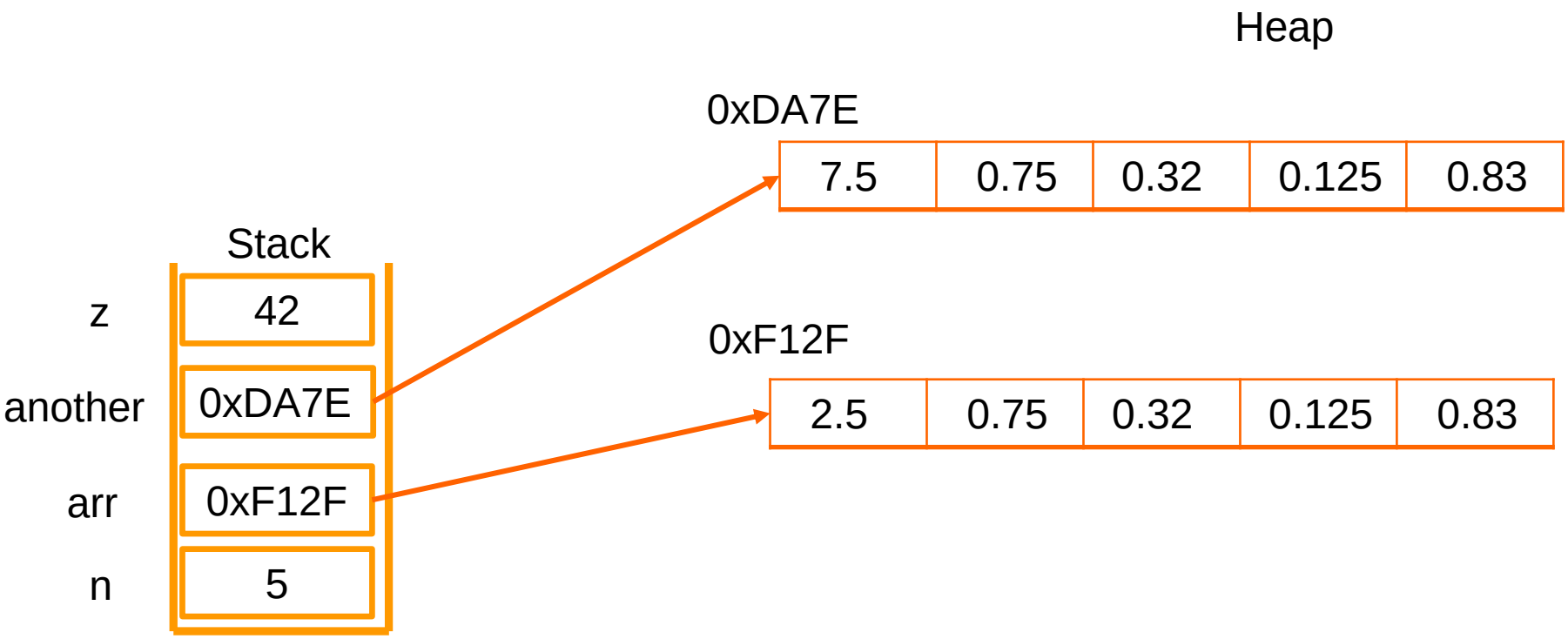
```
int z = n;
```



operator = copies values from/to the stack

- operator copy values on the stack and does not clone a data

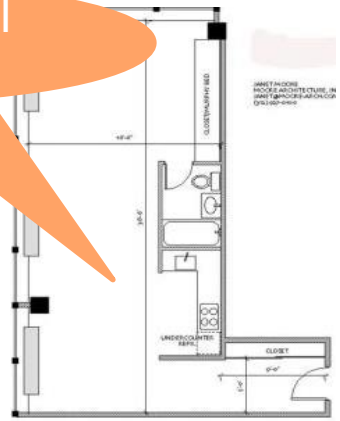
```
z = 42;
System.out.printf("#3 = %d %d %n", n, z);
```



Classes and objects

- A class is a definition from which new objects would be created
  - like a blueprint or template
  - defines common properties (variables and methods) that objects of the group would poses
- An object is one instance of a class.

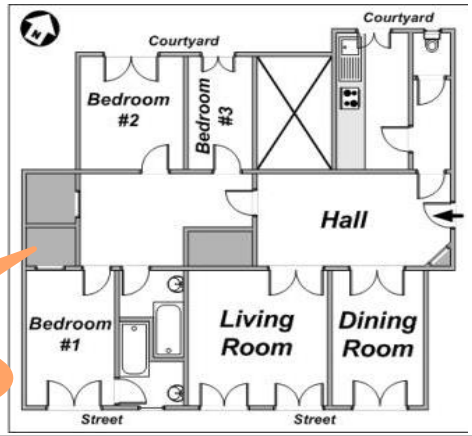
Class: Small Apartment



Objects – instances of *LargeApartment*

Objects – instances of *SmallApartment*

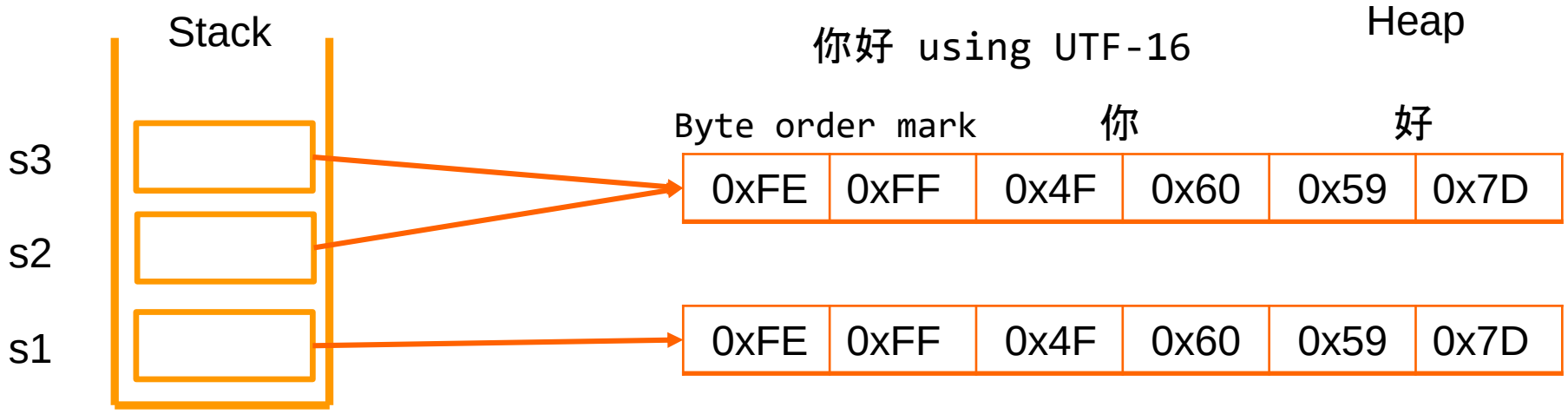
Class: Large Apartment



Strings in Java

- String is sequence of characters
  - *char* is primitive type, *String* is class
  - Instance of string stores characters as bytes (depends of encoding)
- Variables of type *String* are references!
  - new strings (objects) are created using operator *new*

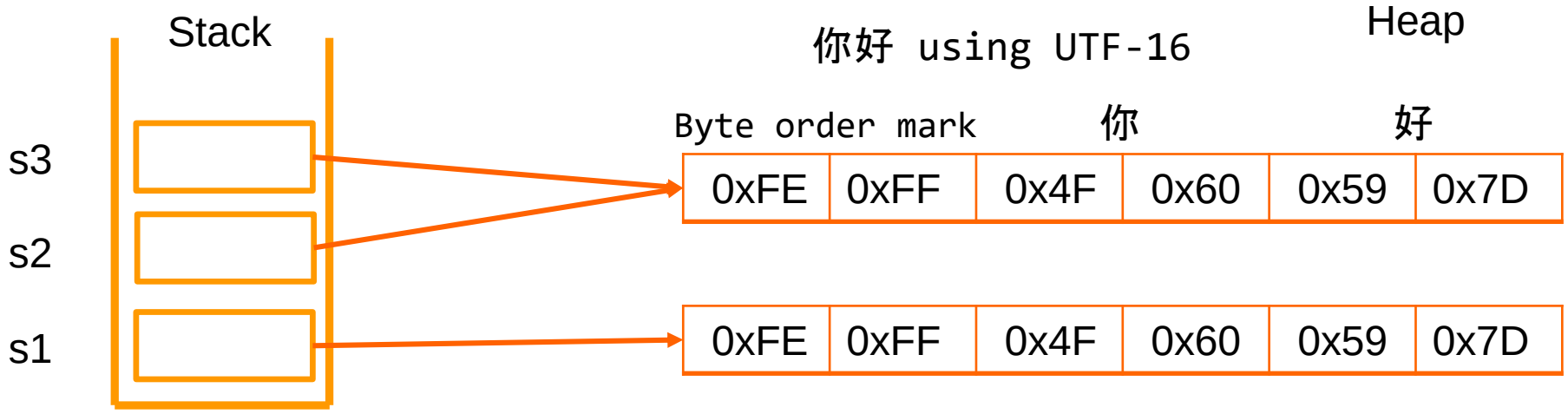
```
String s1 = new String("你好"); //Nǐ hǎo
String s2 = new String("你好"); //Nǐ hǎo
String s3 = s2;
```



Comparing strings (1)

- Operator == compares stack values!
  - *s2 == s3* is *true*
  - s1 == s2 is *false*

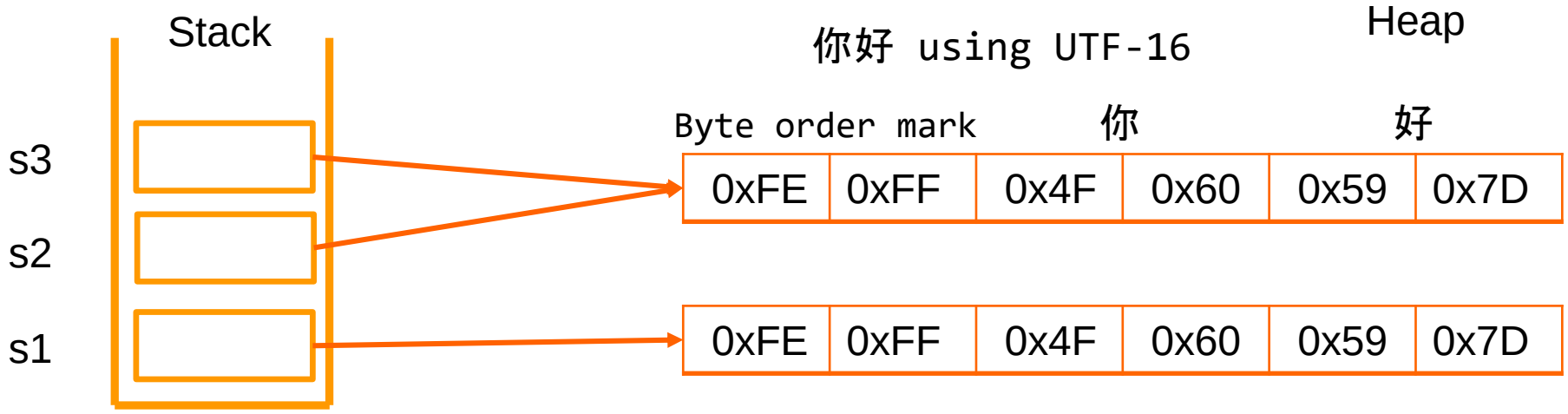
```
String s1 = new String("你好"); //Nǐ hǎo
String s2 = new String("你好"); //Nǐ hǎo
String s3 = s2;
```



Comparing strings (2)

- How to compare content?
  - *s2.equals(s3)* is *true*
  - *s1.equals(s2)* is *true*

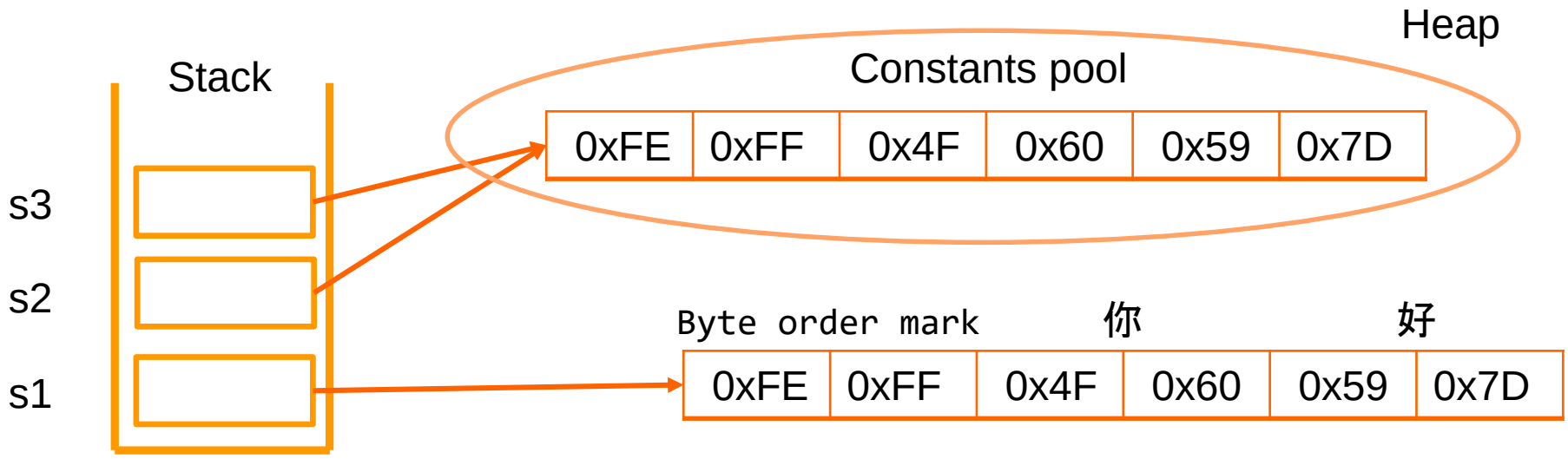
```
String s1 = new String("你好"); //Nǐ hǎo
String s2 = new String("你好"); //Nǐ hǎo
String s3 = s2;
```



Comparing strings (3)

- Note: all constants strings are stored only once in memory
  - only *new* creates a new string (may be copy of an existing one)
  - *s2 == s3* is *true and* s1 == s2 is *false*
  - *s2.equals(s3)* is *true* and *s1.equals(s2)* is *true*

```
String s1 = new String("你好"); //Nǐ hǎo
String s2 = "你好"; //Nǐ hǎo
String s3 = "你好";
```

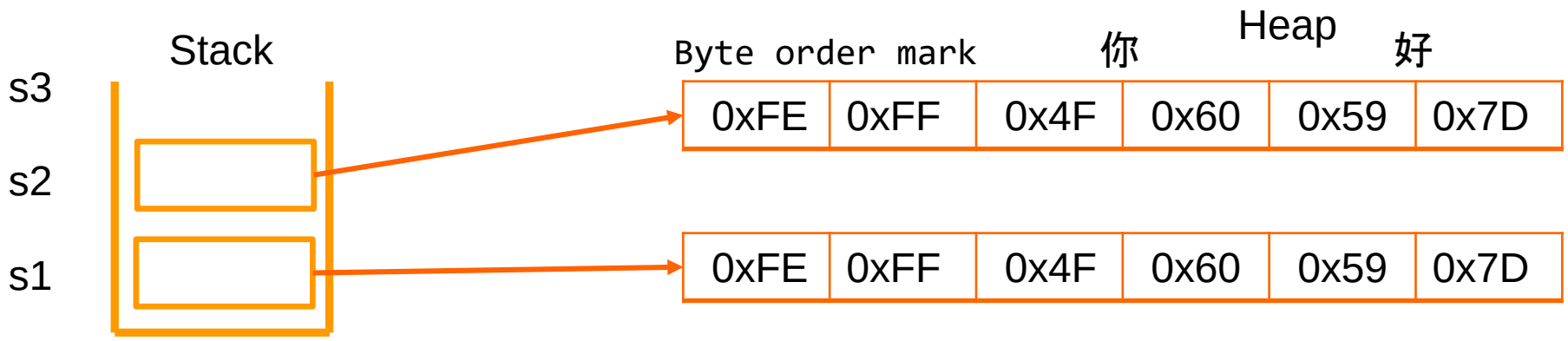


Instance methods

```
String s1 = new String("你好");
char c = s1.charAt(0); // 你
String s2 = new String("你好");
boolean b = s2.equals(s1)
```

- Notice the difference between
  - *s2.equals(s3), s1.charAt(0)*

  and some method calls from previous presentation (T1)
  - *Math.abs, Integer.parseInt, Taylor.ePowerX*
- Methods like *equals* and *charAt* from *String* are instance methods
  - needs an object of its class to be run, and access the object's content
- Static methods can be run even if there is no object of that type
  - more about it in the next presentation (T3)



String methods

- Class *String* has many usable methods from strings manipulation
  - e.g. finding or getting substrings, changing case, replacing parts, …
- String is immutable
  - all methods for string manipulation returns a new string

```
String text = "The quick brown fox jumps over lazy dogs.";
String upperCase = text.toUpperCase();
System.out.println("Upper case text: " + upperCase);
System.out.println("Replacing fox with wolf: "
       + text.replace("fox", "wolf"));
int position = text.indexOf("quick");
System.out.println("quick starts at index: " + position);
System.out.println(text.substring(position, position + 15));
System.out.println("Original: " + text);
```

- Inspect and run the code from the sample and try some other string methods

…02_ObjectCreation/…/WorkingWithStrings.java

Strings concatenation

- Operator + for String could be used for concatenation
  - can be used in combination with numbers or other objects
    - possible due to autoboxing and existence of toString method (explained in some of later presentations)

```
String text = "The quick " + "brown ";
text += "fox jumps over ";
text += 3;
text += " lazy dogs.";
System.out.println(text);
```

…02_ObjectCreation/…/WorkingWithStrings.java

- Note: Do not forget that strings are immutable
  - in case of many concatenations use *StringBuilder* instead

Object lifecycle

- What happens with objects on heap
  - allocated with operator *new*,
  - or e.g. with new strings created during use of various methods from class *String*?
  - memory leaks?
- No need to worry (to much) - *Java* has *Garbage collector (GC)*
  - GC would remove objects that are not referenced any more.
    - Unknown when GC is run (not related to variable scope)
  - *finalize* method is run before cleaning an object
    - in most cases, there is no need for this method
  - To many unnecessary allocated objects causes GC to run more often and may incur performance penalties
    - thus, e.g. StringBuilder is more appropriate if there is lot of concatenations

Standard input/output

…02_ObjectCreation/…/ReadFromStandardInput.java

- System.out.println, System.out.printf for writing to std. output
  https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Formatter.html#syntax
- Class Scanner for reading from system input

```
Scanner sc = new Scanner(System.in);
System.out.println("Please enter an integer number");
int x = sc.nextInt();
System.out.println("Entered: " + x);
System.out.println("Now, enter a floating point number");
double y = sc.nextDouble();
System.out.println("Entered: " + y);
System.out.println("Enter several lines. Use Q or q to quit.");
while(sc.hasNextLine()) {
        String line = sc.nextLine();
        if (line.equalsIgnoreCase("Q")) break;
        System.out.println(line);
}
```