Java Useful Classes

AP Fall-1404 Dr. Mojtaba Vahidi Asl

by Danial Yaghooti

Introduction – Java Core & Utility Classes

Java provides a rich set of core classes that make programming easier, more efficient, and more reliable.

In this presentation, we will explore some of the most useful classes in Java:

- Math.
- String.
- Wrapper.
- String Builder.
- Arrays.

Math Class

Provides static methods for mathematical operations.

- No need to create an object: All methods are static.
- Examples:

$|x|$
```java
Math.abs(-1); // 1
```

$\sqrt{x}$
```java
Math.sqrt(121); // 11
```

x^a
```java
Math.pow(2,5); // 32
```

max
```java
Math.max(7,5); // 7
```

$\lfloor a \rfloor$
```java
Math.floor(3.7); // 3
```

$\lceil a \rceil$
```java
Math.ceil(2.5); // 3
```

Generating Random Numbers in

In Java, the Math.random() method is a powerful tool for generating random numbers. This method returns a random floating-point number between 0.0 and 1.0. So:

$0 \leq Math.ramdom() < 1$:

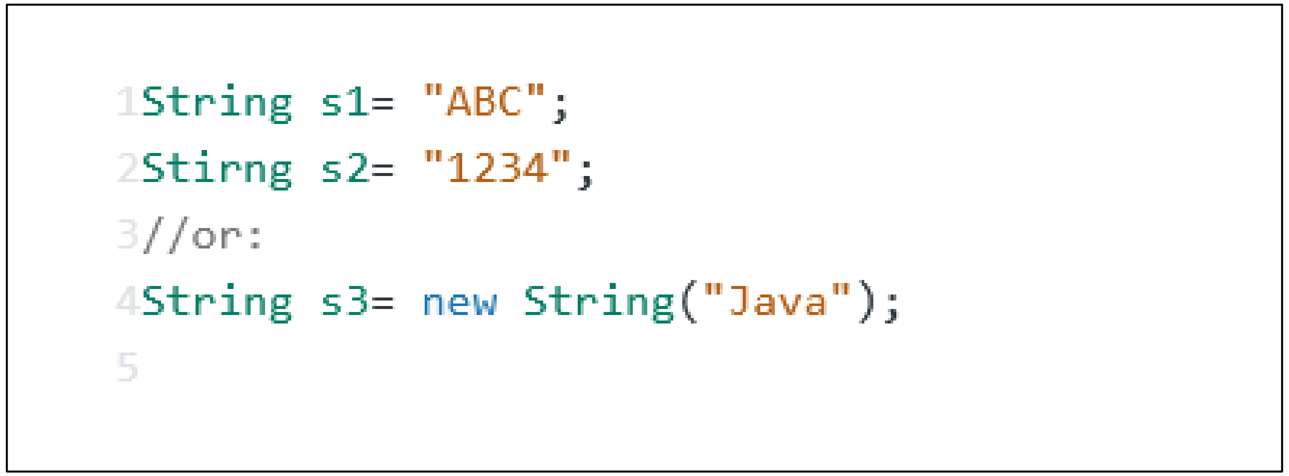
Random $\in [0, A] \rightarrow$ `int R = (int)(Math.random()*(A+1));`

Random $\in [min, max] \rightarrow$ `int R = (int)(Math.random()*(max – min +1) + min)`

String Class(immutable)

The String class has a set of built-in methods that you can use on strings. It provides numerous methods for manipulating strings, including searching, comparing, and modifying string content.

Initialize:



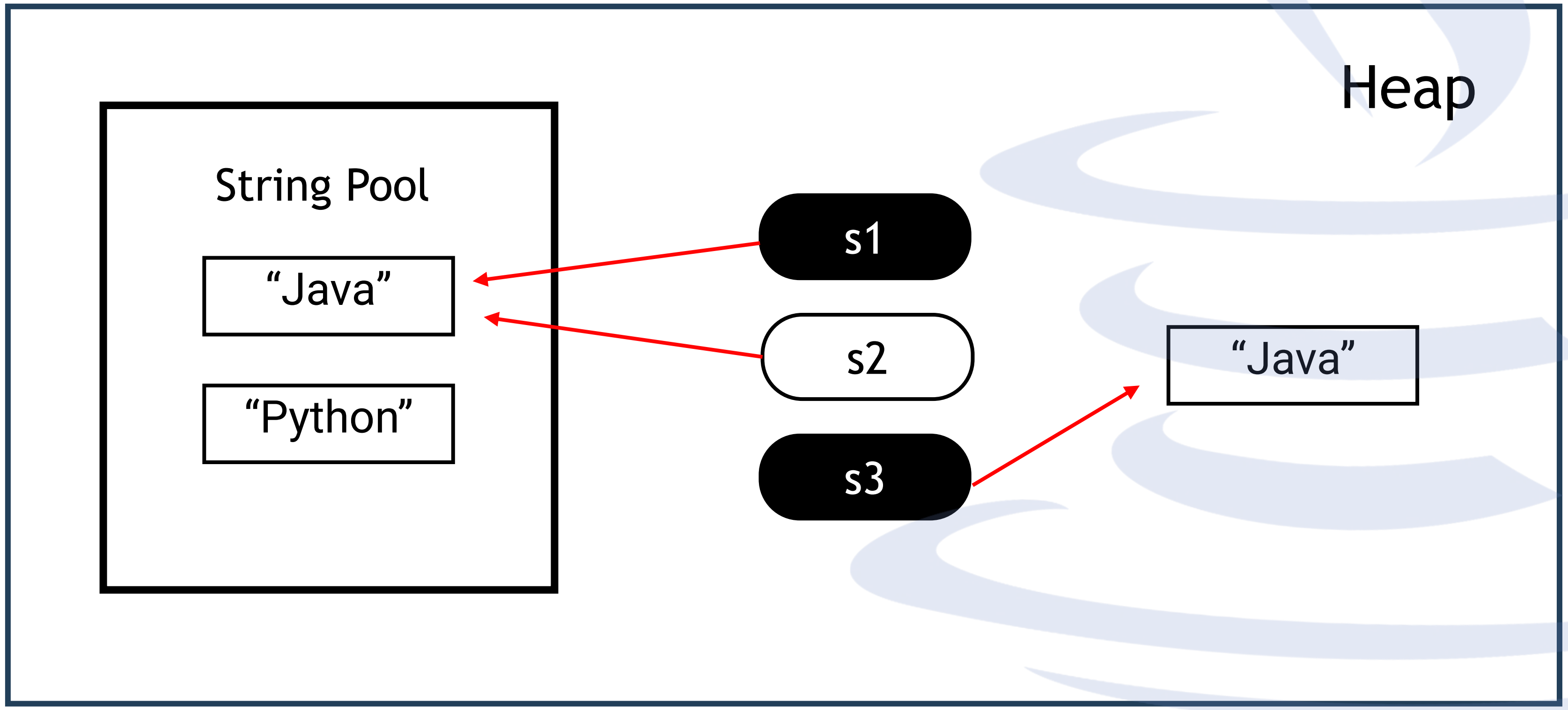
```java
String s1= "ABC";
Stirng s2= "1234";
//or:
String s3= new String("Java");

```

But what is the difference?

String Pool

Suppose we wrote following code:

```
String s1= “Java”;
String s2 = “Java”;
String s3 = new String(“Java”);
```



Common String Methods

Here are some common methods to deal with strings:

1
```
.trim(); // removing space from first and end
```

2
```
s1.equals(String s2); //check if s1==s2
```

3
```
.toLowerCase(); // A -> a .toUpperCase(); // a -> A
```

4
```
.charAt(int i); //return char at index i
```

5
```
.replaceAll(String Re, String replaceTo);
```

6
```
.split(String Re); // split string to string array
```

7
```
String.format("%d is: %s", int a, String s); //works like prtintf
```

Recourses

You can find more details and methods from following links:

[Java String Reference]

https://www.geeksforgeeks.org/java/strings-in-java/

StringBuilder Class(Mutable)

A mutable sequence of characters.

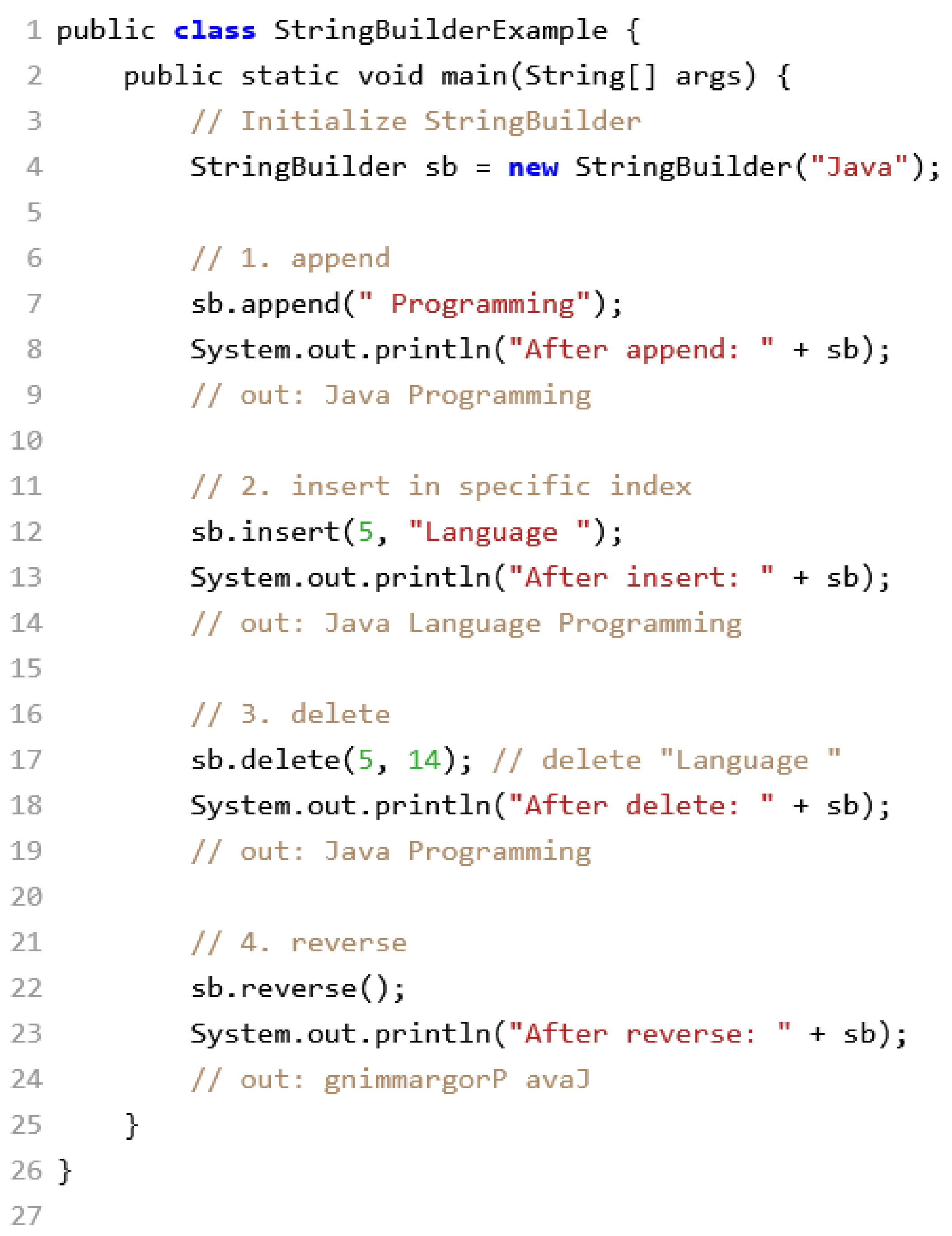
Examples:

1. **`.append(String str);`**
   Add string at the end.

2. **`.Insert(int index, String str);`**
   Inser a string at specific index.

3. **`.delete(int start, int end);`**
   Delete between two specific index.

4. **`.reverse();`**
   Reverses the string.



```java
public class StringBuilderExample {
    public static void main(String[] args) {
        // Initialize StringBuilder
        StringBuilder sb = new StringBuilder("Java");

        // 1. append
        sb.append(" Programming");
        System.out.println("After append: " + sb);
        // out: Java Programming

        // 2. insert in specific index
        sb.insert(5, "Language ");
        System.out.println("After insert: " + sb);
        // out: Java Language Programming

        // 3. delete
        sb.delete(5, 14); // delete "Language "
        System.out.println("After delete: " + sb);
        // out: Java Programming

        // 4. reverse
        sb.reverse();
        System.out.println("After reverse: " + sb);
        // out: gnimmargorP avaJ
    }
}

```

Mutable vs Immutable

In Java, classes can be classified as mutable or immutable. Mutable classes allow their instances to be modified after creation, while immutable classes prevent any changes to their instances once they're created.

Feature	String (Immutable)	StringBuilder (Mutable)
Mutability	Immutable (cannot change)	Mutable (can be modified)
Performance	Slower for repeated modifications	Faster for repeated modifications
Thread-Safety	Safe without synchronization	Not thread-safe (use StringBuffer if needed)
Memory Usage	Creates new objects on modification	Reuses the same object
Use Cases	Constants, keys, security-sensitive data	Large text building, dynamic content

Wrapper Classes

Convert primitive types into objects.
Examples:

```
Integer num = 17;
Integer sum = num + 3; //20

//convert to primitive type value
int intValue = num.intValue(); // 17
double doubleValue = num.doubleValue(); //17.0

//Max and Min Value of an Integer
System.out.println("Integer.MAX_VALUE = " + Integer.MAX_VALUE);
System.out.println("Integer.MIN_VALUE = " + Integer.MIN_VALUE);

//Parsing
int parsed =  Integer.parseInt("123"); // 123

System.out.println("Compare 42 and 100:"  + Integer.compare(42,100)); // -1
System.out.println("Compare 17 and 5:" + num.compareTo(5)); // +1
```

Integer

The Integer class is the wrapper for the primitive type int.
It provides methods to convert between primitive (int) and object (Integer), compare numbers, parse strings into integers (parseInt), and access constants like MAX_VALUE and MIN_VALUE.

Wrapper Classes

```
Character ch = 'A';


// Check Character
System.out.println("Is digit? " + Character.isDigit(ch));       // false
System.out.println("Is letter? " + Character.isLetter(ch));     // true
System.out.println("Is lowercase? " + Character.isLowerCase(ch)); // false
System.out.println("Is uppercase? " + Character.isUpperCase(ch)); // true


// Change Character
System.out.println("To lowercase: " + Character.toLowerCase(ch)); // a
System.out.println("To uppercase: " + Character.toUpperCase('b')); // B
```

Character

The Character class is the wrapper for the primitive type char.
It offers methods to analyze and manipulate characters, such as checking whether a character is a digit or a letter (isDigit, isLetter), or converting between uppercase and lowercase (toLowerCase, toUpperCase).

Arrays Class

The Arrays class in Java is a utility class from the java.util package that provides static methods to manipulate arrays efficiently. These are some common methods:

```
int[] numbers = {5, 2, 9, 1, 3};

// 1. sorting
Arrays.sort(numbers);
System.out.println("Sorted: " + Arrays.toString(numbers)); //out: [1,2,3,5,9]

// 2. binary-Search (Required a sorted array)
int index = Arrays.binarySearch(numbers, 5); // index = 3

// 3. fill array
int[] filled = new int[5];
Arrays.fill(filled, 7);
System.out.println(Arrays.toString(filled)); // out: [7,7,7,7,7]
```

Arrays Class

```
// 4. Comparing
int[] arr1 = {1, 2, 3};
int[] arr2 = {1, 2, 3};
int[] arr3 = {3, 2, 1};
System.out.println("arr1 == arr2? " + Arrays.equals(arr1, arr2)); // true
System.out.println("arr1 == arr3? " + Arrays.equals(arr1, arr3)); // false

// 5. Copy
int[] copy = Arrays.copyOf(numbers, 3);
System.out.println("Copy: " + Arrays.toString(copy));
// out: [1,2,3]
```

( )

Thank you for your time [__])