

String

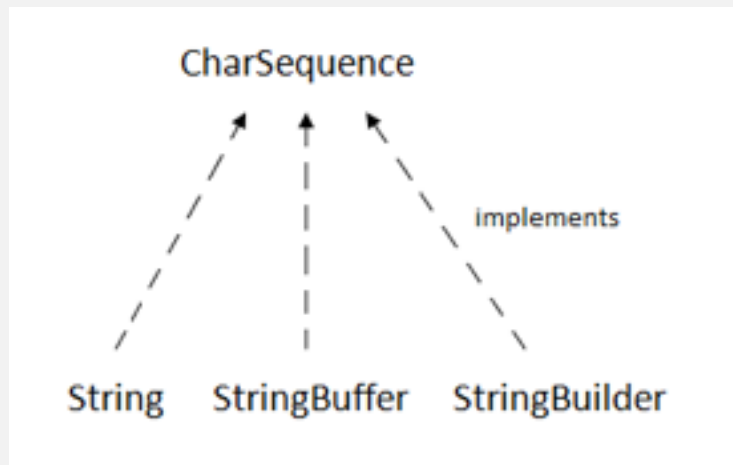
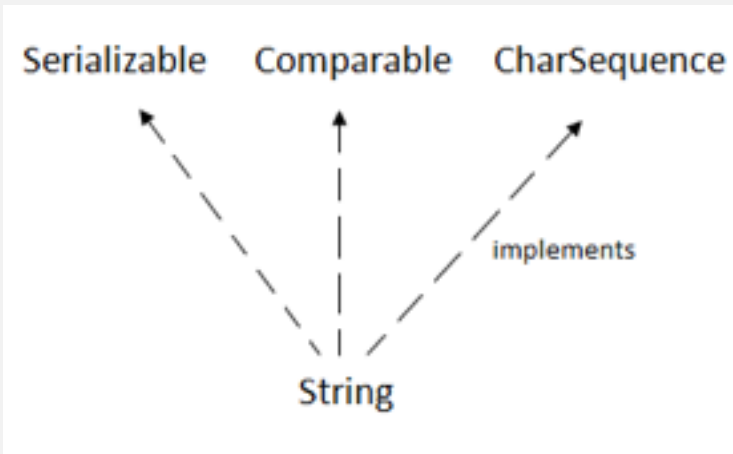


Agenda

- Czym jest String ?
- Do czego służy ?
- W jaki sposób z niego korzystać ?

Klasa String

- Klasa String wykorzystywana jest do operacji na ciągach znaków (napisach) takich jak *compare()*, *concat()*, *equals()*, *split()*, *length()*, *replace()*, *compareTo()*, *intern()*, *substring()* etc
- Implementuje interfejsy *Serializable*, *Comparable* i *CharSequence*.
- Interface *CharSequence* – reprezentuje ciąg znaków char
- Interface *CharSequence* jest implementowany także przez klasy *StringBuffer* i *StringBuilder*. Oznacza to, że możemy tworzyć obiekty String przy ich użyciu



Tworzenie obiektów

```
String s1 = "Hello";           // String literal
String s2 = "Hello";           // String literal
String s3 = s1;                 // same reference
String s4 = new String("Hello"); // String object
String s5 = new String("Hello"); // String object
```

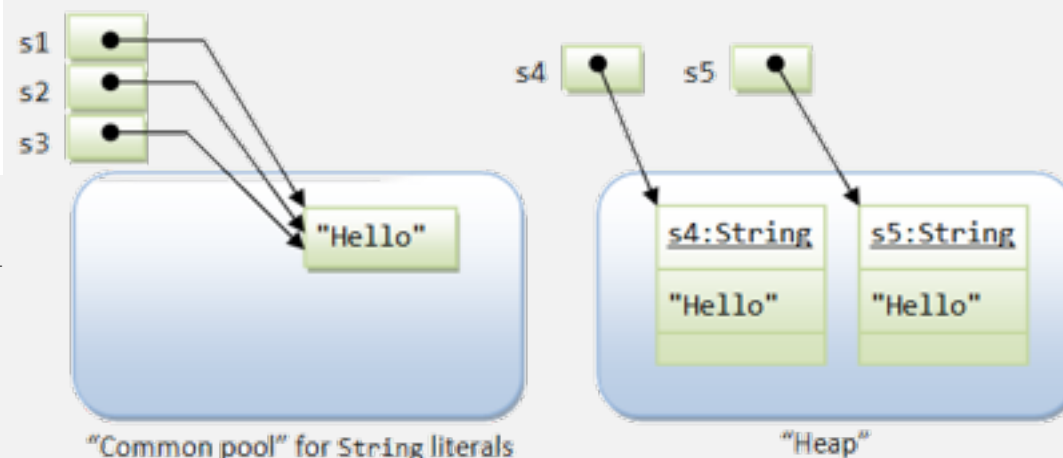
Obiekty typu String można tworzyć na kilka sposobów.

- String literal

String Common Pool – mechanizm Javy (JVM), który zapisuje w skompilowanym programie napisy bez powtórzeń w celu zaoszczędzenia pamięci. Wszystkie zmienne które posiadają taką samą wartość „...” przypisaną wskazują na to samo miejsce w pamięci.

- String object

Obiekty String tworzone przy użyciu `new String(...)` nawet pomimo takiej samej zawartości zapisywane są jako nowe referencje do nowych obiektów.

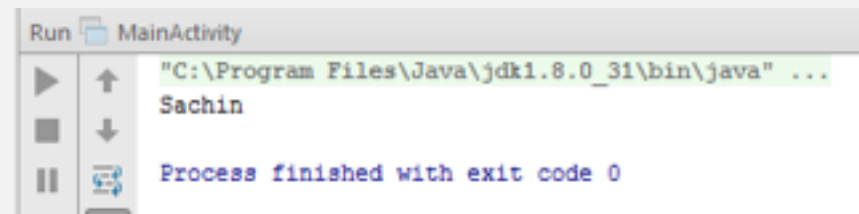


Immutable

- *Immutable String* oznacza że raz utworzone obiekty są niemodyfikowalne i niezmiennie, innymi słowy, każda modyfikacja na obiekcie String powoduje utworzenie nowego obiektu ze zmienioną wartością. Stary obiekt pozostaje bez zmiany.
- Dlaczego?

W Javie, w sytuacji kiedy kilka zmiennych posiada referencję do jednego napisu (np. „Sachin”) zmienienie wartości jednej ze zmiennych powodowałoby zmienienie wartości pozostałych wskazujących na ten sam napis gdyby nie fakt że String jest immutable.

```
String s = "Sachin";  
s.concat(" Tendulkar");  
//concat() dodaje napis na koniec  
System.out.println(s);
```



Porównywanie Stringów

- *equals()*

Porównywanie obiektów po zawartości.

Metoda `equalsIgnoreCase()` wykonuje porównanie napisów ignorując wielkość znaków

- `==`

Porównywanie pod kątem referencji. Oznacza to, że nawet takie same napisy mogą być zapisane w innym miejscu w pamięci – będą wskazywać na inny adres.

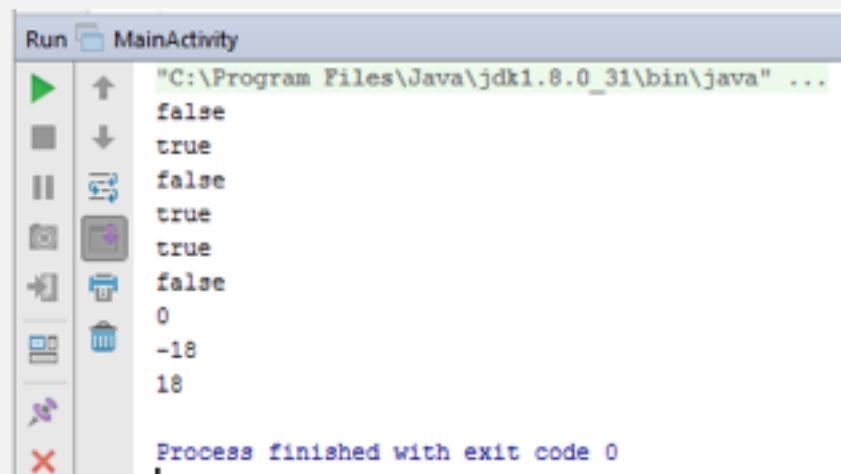
- *compareTo()*

Porównywanie używane przy sortowaniu. Wynik 0 oznacza, że porównywane obiekty są równe.

```
String s1 = "Sachin";
String s2 = "SACHIN";
String s3 = new String("Sachin");
String s4 = "Saurav";
System.out.println(s1.equals(s2)); // false
System.out.println(s1.equals(s3)); // true
System.out.println(s1.equals(s4)); // false
System.out.println(s1.equalsIgnoreCase(s3)); // true
```

```
String t2 = "Sachin";
System.out.println(s1 == t2); // true
System.out.println(s1 == s3); // false
```

```
System.out.println(s1.compareTo(s3)); // 0
System.out.println(s1.compareTo(s4)); // < 0 (because s1 > s3)
System.out.println(s4.compareTo(s1)); // > 0 (because s3 < s1 )
```



```
Run MainActivity
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...
false
true
false
true
true
false
0
-18
18
Process finished with exit code 0
```

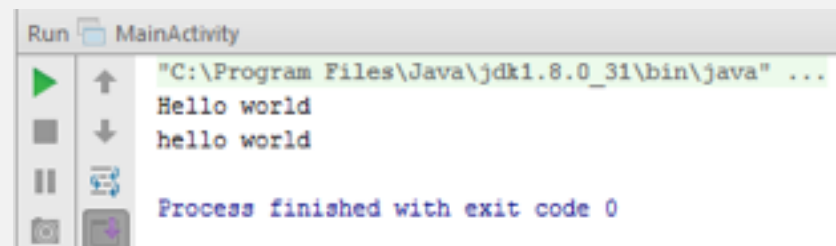
Konkatenacja

- Konkatenacja – łączenie Stringów
- Konkatenacji dokonuje się przy użyciu „+” lub metody concat();

Kompilator Javy na etapie kompilacji zamieni konkatenację przez „+” na:

```
String s = (new StringBuilder()).append("hello").append(" world").toString();
```


```
String s = "Hello" + " world";  
System.out.println(s);  
  
String hello = new String("hello").concat(" world");  
System.out.println(hello);
```



Substring

- Substring jest częścią napisu
- Metoda substring może przyjmować jeden lub dwa parametry. Pierwszy z nich oznacza indeks od którego należy zacząć wydzielać znaki. Drugi – indeks końca - jest opcjonalny. Nie podanie drugiego parametru oznacza, że koniec wydzielania napisu jest równoznaczny z końcem całego napisu bazowego.

```
String s = "Hello world";  
System.out.println(s.substring(3));  
System.out.println(s.substring(4, s.length()));
```



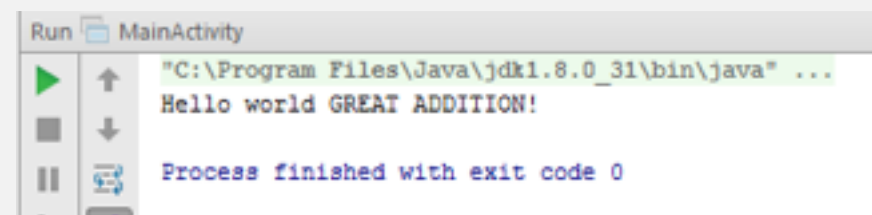
The screenshot shows a Java IDE's run console for a class named MainActivity. The console output displays the results of the substring method calls: "lo world" and "o world". The process finished with exit code 0.

```
Run MainActivity  
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
lo world  
o world  
Process finished with exit code 0
```


StringBuffer

- Klasa StringBuffer jest mutable – pozwala na modyfikacje
- StringBuffer jest thread-safe, czyli bezpieczny do modyfikacji wielowątkowej
- StringBuffer podobnie do String wykonuje operacje na ciągach znaków, wiele metod tych klas jest podobnych
- StringBuffer jest synchronizowaną klasą i działa wolniej od StringBuilder

```
StringBuffer buffer = new StringBuffer("Hello world");  
buffer.append(" GREAT ADDITION!");  
System.out.println(buffer);
```



The screenshot shows the 'Run' console of an IDE. The title bar indicates 'Run MainActivity'. The console output shows the command prompt path 'C:\Program Files\Java\jdk1.8.0_31\bin\java' followed by the output 'Hello world GREAT ADDITION!'. At the bottom, it states 'Process finished with exit code 0'.

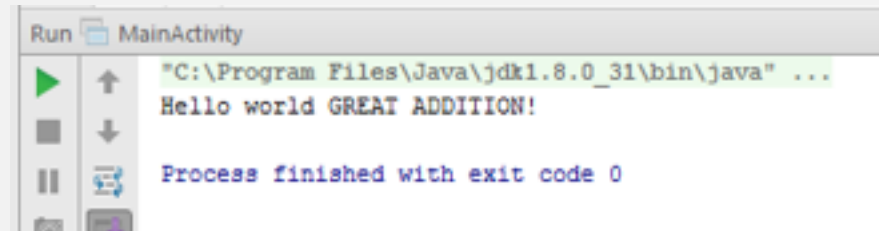
StringBuilder

Klasa `StringBuilder` jest bardziej wydajna niż `StringBuffer`.

Wynika to z faktu, że klasa `StringBuilder` nie jest bezpieczna w środowisku wielowątkowym w porównaniu do `StringBuffer`'a – nie jest thread-safe.

Obie klasy mają identyczne możliwości.

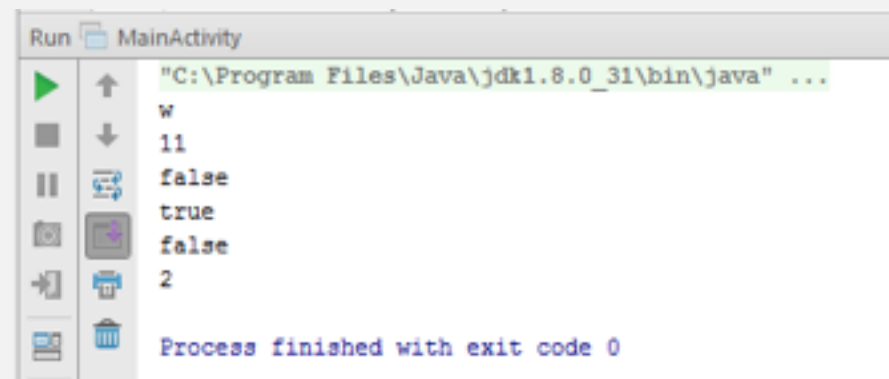
```
StringBuilder buffer = new StringBuilder("Hello world");  
buffer.append(" GREAT ADDITION!");  
System.out.println(buffer);
```



Najczęściej używane metody String

- *charAt()* – zwraca znak na zadanej pozycji w napisie
- *contains()* – zwraca informację, czy napis zawiera określone znaki
- *endsWith()* i *startsWith()* – zwraca informację czy napis kończy / zaczyna się zadany parametrem
- *indexOf()* – zwraca pozycję zadanego napisu
- *isEmpty()* – zwraca informację czy napis jest pusty
- *length()* – długość napisu

```
String s = "Hello world";  
System.out.println(s.charAt(6));  
System.out.println(s.length());  
System.out.println(s.endsWith("H"));  
System.out.println(s.startsWith("H"));  
System.out.println(s.isEmpty());  
System.out.println(s.indexOf("l"));
```



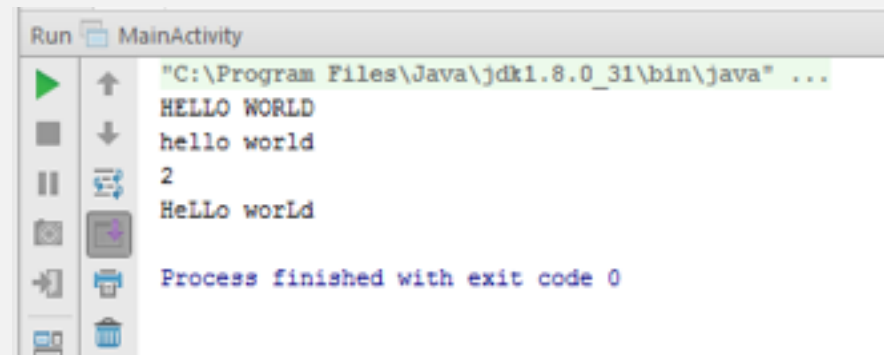
The screenshot shows the 'Run' console of an IDE. The title bar says 'Run MainActivity'. The command line shows the Java command being executed: `"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...`. The output of the program is displayed as follows:

```
w  
11  
false  
true  
false  
2  
  
Process finished with exit code 0
```

Najczęściej używane metody String

- *replace()* – zamienia wszystkie wystąpienia jednego znaku na drugi znak
- *split()* – rozdziela napis na elementy wg. Podanego parametru
- *trim()* – usunięcie białych znaków z końca i początku napisu
- *toLowerCase()* i *toUpperCase* – konwersja napisu na małe/wielkie znaki

```
String s = "Hello world";  
System.out.println(s.toUpperCase());  
System.out.println(s.toLowerCase());  
System.out.println(s.split(" ").length);  
System.out.println(s.replace("l", "L"));
```



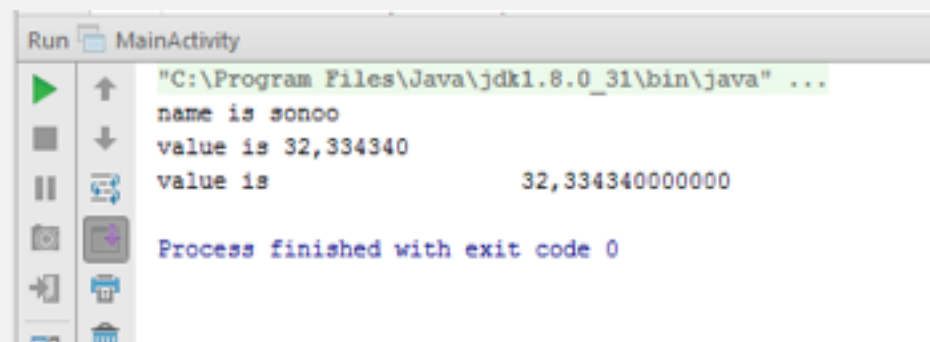
The screenshot shows a 'Run' window for 'MainActivity'. The command line shows the execution of a Java program. The output is as follows:

```
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
HELLO WORLD  
hello world  
2  
HeLlO worLd  
Process finished with exit code 0
```

String format

Metoda format służy do podstawiania wartości pod zadany szablon.

```
String name = "sonoo";  
String sf1 = String.format("name is %s", name);  
String sf2 = String.format("value is %f", 32.33434);  
String sf3 = String.format("value is %32.12f", 32.33434);  
  
System.out.println(sf1);  
System.out.println(sf2);  
System.out.println(sf3);
```

A screenshot of the Java Run console window. The title bar says 'Run MainActivity'. The console shows the execution of the code, with the output of the println statements. The first line is 'name is sonoo'. The second line is 'value is 32,334340'. The third line is 'value is 32,3343400000000'. The final line is 'Process finished with exit code 0'.

```
Run MainActivity  
"C:\Program Files\Java\jdk1.8.0_31\bin\java" ...  
name is sonoo  
value is 32,334340  
value is 32,3343400000000  
Process finished with exit code 0
```