# Chapter 5 - Core Java APIs

### Create and manipulate Strings

- Strings are immutable - once created it can't change

```
String s1 = "1";
String s2 = s1.concat("2");
s2.concat("3");    // a new String will be created and will contain value "123", while s2 is immutable and equ
System.out.println(s2);
```

- String methods
  - `int length()` - returns the length of string
  - `char charAt(int index)` - returns a char at specific position.
  - ```
    int indexOf(int ch)

    int indexOf(int ch, int fromIndex)
    int indexOf(String str)
    int indexOf(String str, int fromIndex)

    ```

    In takes an int because char can be passed around as int.
    In case the char or string is not found, it will return -1.

  - ```
    String substring(int beginIndex)
    String substring(int beginIndex, int endIndex)
    ```

    in the first overload - it takes all characters till end.
    when endIndex < beginIndex OR endIndex >= string.length() → throw exception
    beginIndex - is included, entIndex - is excluded. Ex: "abcd".substring(0,3) = "abc"

  - ```
    String toLowerCase()
    String toUpperCase()
    ```

  - ```
    boolean equals(Object obj)
    boolean equalsIgnoreCase(String str)
    ```

    first takes an object as it come from Object class

  - ```
    boolean startsWith(String prefix)
    boolean endsWith(String suffix)
    ```

  - ```
    String replace(char oldChar, char newChar)
    String replace(CharSequence target, CharSequence replacement)
    ```

  - `boolean contains(CharSequence charSeq)`
  -

```
String trim()   -> removes white spaces (space, \t, \n, \r) from beggining and end of string
String strip()  -> same as trim() but support unicode e.g '\u2000'
String stripLeading()  -> removes whitespaces from beggining
String stripTrailing() -> removes whitespaces from end
```

- `String intern()` - if values is in the string pool → returns it from there, otherwise adds it to the pool

### Manipulate data using StringBuilder

- StringBuilder is mutable and used in cases many String objects should be created.
- Constructing StringBuilder

```
StringBuilder sb1 = new StringBuilder();            // empty sequence of characters
StringBuilder sb2 = new StringBuilder("animal");    // from a non-empty string
StringBuilder sb3 = new StringBuilder(10);      //creates it with a certaing capacity, number of slots, for ch
```

- StringBuilder methods
  - ```
    charAt(), indexOf(), length(), and substring() - same as in String
    ```
  - `StringBuilder append(String str)` - ads the parameter to string builder
  - `StringBuilder insert(int offset, String str)` - adds characters to the `StringBuilder` at the requested index and returns a reference to the current `StringBuilder`
    *offset - index where we want to insert the requested parameter.
  - ```
    StringBuilder delete(int startIndex, int endIndex)
    StringBuilder deleteCharAt(int index)
    ```
    startIndex - included, endIndex - excluded.
    if endIndex >= stringBuilder.length() → **does not throw exception**
  - `StringBuilder replace(int startIndex, int endIndex, String newString)`
    startIndex - included, endIndex - excluded.
    if endIndex >= stringBuilder.length() → **does not throw exception**
  - `StringBuilder reverse()`
  - `String toString()`

### Understanding equality

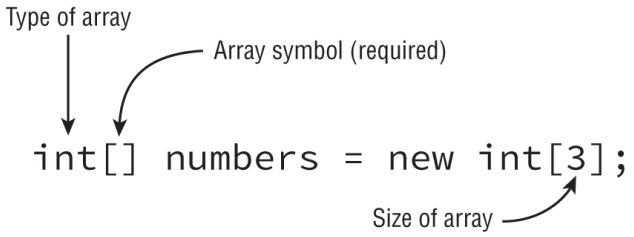
- **=** checks reference
- .equals - in case implemented, check the content for equality
- In case you are wondering, the authors of `StringBuilder` **did not implement** `equals()`. If you call `equals()` on two `StringBuilder` instances, **it will check reference equality**.

### String pool

- The *string pool*, also known as the intern pool, is a location in the Java virtual machine (JVM) that collects all these strings. Java realizes that many strings repeat in the program and solves this issue by reusing common ones.
- The string pool contains literal values and constants that appear in your program. For example, `"name"` is a literal and therefore goes into the string pool. `myObject.toString()` is a string but not a literal, so it does not go into the string pool.
- ```
  String x = "Hello World";
  String z = " Hello World".trim();
  System.out.println(x == z); // false
  ```
- They are not the same at compile-time, a new String object is created
- ```
  String x = "Hello World";
  String y = new String("Hello World");
  System.out.println(x == y); // false
  ```
- It says “No, JVM, I really don’t want you to use the string pool. Please create a new object for me even though it is less efficient.”
- ```
  String name = "Hello World";
  String name2 = new String("Hello World").intern();
  System.out.println(name == name2);     // true
  ```

  You can also do the opposite and tell Java to use the string pool. The `intern()` method will use an object in the string pool if one is present. If the literal is not yet in the string pool, Java will add it at this time.

### Java arrays




- When you use this form to instantiate an array, all elements are set to the default value for that type.
- `int[] numbers2 = new int[] {42, 55, 99};` = `int[] numbers2 = {42, 55, 99};`
- Arrays can also be casted

```
3: String[] strings = { "stringValue" };
4: Object[] objects = strings;
5: String[] againStrings = (String[]) objects;
6: againStrings[0] = new StringBuilder();   // DOES NOT COMPILE
7: objects[0] = new StringBuilder();        // careful!
```

- Sorting: Arrays.sort(stringArray)
- Searching
  - Only when array sorted
  - If found → returns index
  - If not found → -1 - (negative of were the number should have been) e.g Arrays.binarySearch(new int[] {2,4,6,8),  7) = -4
  - If not sorted → undefined response
- Comparing
  - Negative nr → first array smaller
  - Zero → equal arrays
  - Positive → first array greater
```

- If both arrays are the same length and have the same values in each spot in the same order, return zero.
- If all the elements are the same but the second array has extra elements at the end, return a negative number.
- If all the elements are the same but the first array has extra elements at the end, return a positive number.
- If the first element that differs is smaller in the first array, return a negative number.
- If the first element that differs is larger in the first array, return a positive number.
- `null` is smaller than any other value.
- For numbers, normal numeric order applies.
- For strings, one is smaller if it is a prefix of another.
- For strings/characters, numbers are smaller than letters.
- For strings/characters, uppercase is smaller than lowercase.
- can't compare arrays with different type

- Mismatch
  - If the arrays are equal, `mismatch()` returns `-1`. Otherwise, it returns the first index where they differ.
- Varargs
  - `public static void main(String... args) // varargs`

### Multidimensional array

```
int[][] vars1;          // 2D array
int vars2 [][];         // 2D array
int[] vars3[];          // 2D array
int[] vars4 [], space [][];  // a 2D AND a 3D array
```

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

- You have to know how many elements will be in the array when you create it

ArrayList

- ```
  ArrayList list1 = new ArrayList();
  ArrayList list2 = new ArrayList(10);
  ArrayList list3 = new ArrayList(list2);
  ArrayList<String> list5 = new ArrayList<>();
  var list = new ArrayList<>();   // compiles, var in this case is of type ArrayList<Object>
  List<String> list6 = new ArrayList<>();
  ```

- Methods
  - ```
    boolean add(E element)
    void add(int index, E element)
    ```
  - ```
    boolean remove(Object object)
    E remove(int index)
    ```
  - `E set(int index, E newElement)`
  -
    ```

```
boolean isEmpty()
int size()
```

- `void clear()`
- `boolean contains(Object object)`
- `boolean equals(Object object)`

### Wrapper Classes

| | | |
|---|---|---|
| `Boolean` | `Boolean.parseBoolean("true")` | `Boolean.valueOf("TRUE")` |
| `Byte` | `Byte.parseByte("1")` | `Byte.valueOf("2")` |
| `Short` | `Short.parseShort("1")` | `Short.valueOf("2")` |
| `Integer` | `Integer.parseInt("1")` | `Integer.valueOf("2")` |
| `Long` | `Long.parseLong("1")` | `Long.valueOf("2")` |
| `Float` | `Float.parseFloat("1")` | `Float.valueOf("2.2")` |
| `Double` | `Double.parseDouble("1")` | `Double.valueOf("2.2")` |
| `Character` | None | None |

### Autoboxing and Unboxing

- Unboxing a null returns NullPointerException
- Be careful when dealing with autoboxing e.g IntegerList.remove(1) removes the value at index 1 instead of Integer with value 1.

### Converting between Array and List

- `list.toArray()` - defaults to array of Objects
  `list.toArray(new String[0])` - by specifying the type, list is converted to an array of that type
- By converting from list to array, they are not linked, array is a new object created.
- `Arrays.asList(array)` - creates a lists that is linked to the array. By changing array we also changing the list. It is a fixed-size list (trying to remove objects will throw exception) and is also known as a backed `List` because the array changes with it.

### Using varargs to create a list

- Creates a fixed-size arrays

| | `toArray()` | `Arrays.asList()` | `List.of()` |
|---|---|---|---|
| Type converting from | `List` | Array (or varargs) | Array (or varargs) |
| Type created | Array | `List` | `List` |
| Allowed to remove values from created object | No | No | No |
| Allowed to change values in the created object | Yes | Yes | No |
| Changing values in the created object affects the original or vice versa. | No | Yes | N/A |

### Creating Sets and Maps

- Adding an element that exists in Set, the add method will return false

- Map methods

| Method | Description |
| --- | --- |
| `V get(Object key)` | Returns the value mapped by key or `null` if none is mapped |
| `V getOrDefault(Object key, V other)` | Returns the value mapped by key or `other` if none is mapped |
| `V put(K key, V value)` | Adds or replaces key/value pair. Returns previous value or `null` |
| `V remove(Object key)` | Removes and returns value mapped to key. Returns `null` if none |
| `boolean containsKey(Object key)` | Returns whether key is in map |
| `boolean containsValue(Object value)` | Returns whether value is in map |
| `Set<K> keySet()` | Returns set of all keys |
| `Collection<V> values()` | Returns `Collection` of all values |

### Math APIs

- Min/Max

```
double min(double a, double b)
float min(float a, float b)
int min(int a, int b)
long min(long a, long b)
```

- max is same

```
long round(double num)
int round(float num)
```

- `double pow(double number, double exponent)`

- `double random()`