Chapter 3. Utility Class

STRING

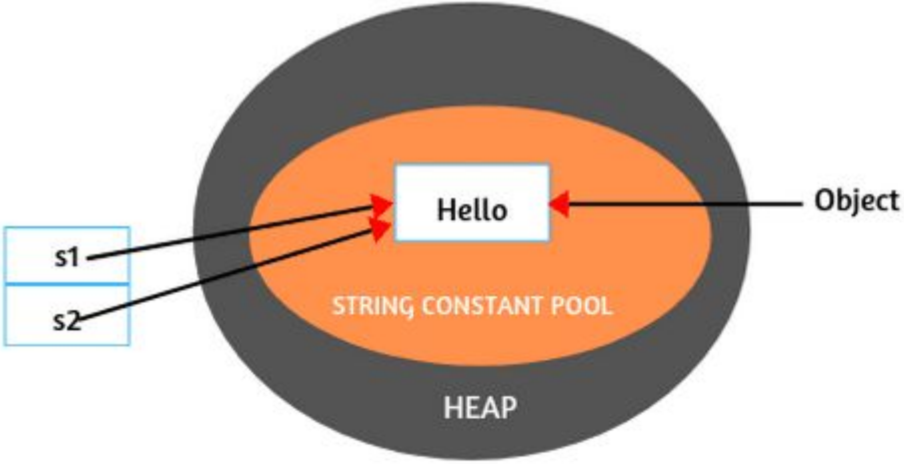
String - What is String?

- String class represents character strings.
- Strings are **constant**; **their value can not be changed** after they are created. **If you try to modify the contents of string object, a new string object is created** with modified content.

String - What is String?

Whenever you create a string literal, JVM checks string constant pool first. If the string already exists in string constant pool, no new string object will be created in the string pool by JVM.

```
String s1 = "Hello";
String s2 = "Hello";
```

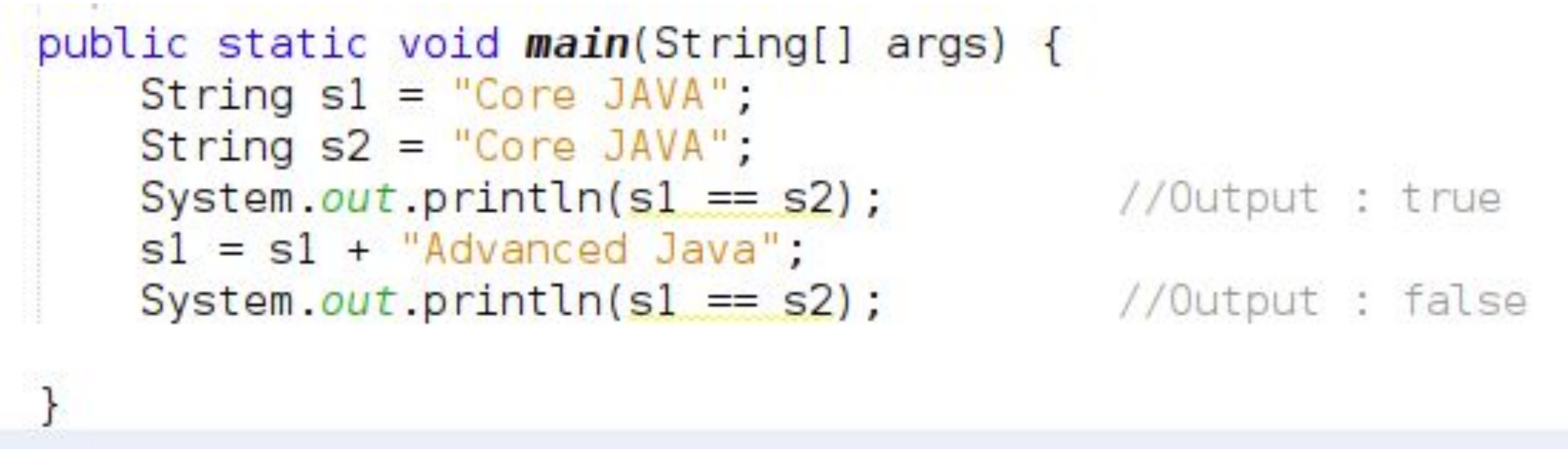


The reference variables s1 and s2 are pointing to the same object.

Fig: Allotting memory for storing object.

Source: https://www.scientecheasy.com/2020/05/string-in-java.html/

String - Example



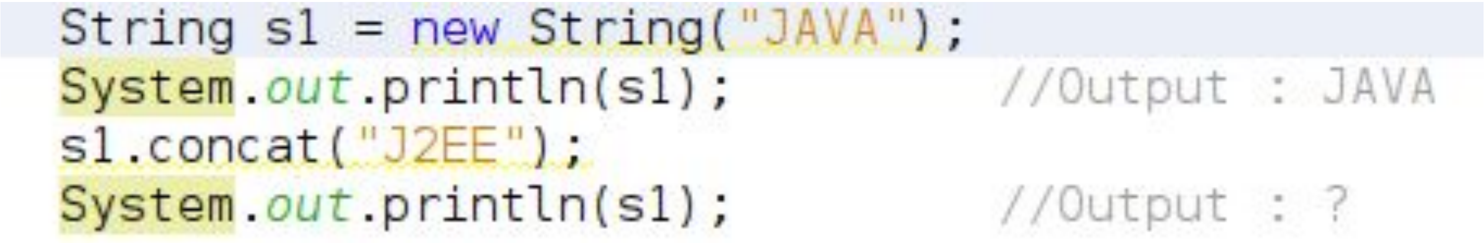
```
public static void main(String[] args) {
    String s1 = "Core JAVA";
    String s2 = "Core JAVA";
    System.out.println(s1 == s2);         //Output : true
    s1 = s1 + "Advanced Java";
    System.out.println(s1 == s2);         //Output : false

}
```

String - String is immutable?

- **Security**: parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. **If it were mutable, these parameters could be easily changed**.

- **Efficiency:** when compiler optimizes your String objects, it sees that if two objects have same value **(a="test", and b="test") and thus you need only one string object (for both a and b, these two will point to the same object)**.

String - Example



```
String s1 = new String("JAVA");
System.out.println(s1);               //Output : JAVA
s1.concat("J2EE");
System.out.println(s1);               //Output : ?
```

String - Example

```
String s1 = “uit.", s2 = " edu.vn ", s3, s4;
int i = s1.length();           // ?
boolean b = s1.isEmpty();      // ?
char c = s1.charAt(i - 1);     // ?
s3 = s1.concat(s2);            // ?
s4 = s1 + s2;                  // ?
b = (s3 == s4);                // ?
b = s3.equalsIgnoreCase(s4);   // ?
s3 = s4.substring(3);          // ?
s3 = s4.substring(5, 8);       // ?
s3 = s1 + s2;                  // ?
b = s3.contains(s1);           // ?
b = s3.endsWith(s2);           // ?
b = s3.startsWith(s1);         // ?
b = s3.startsWith(“edu", 5);   // ?
```

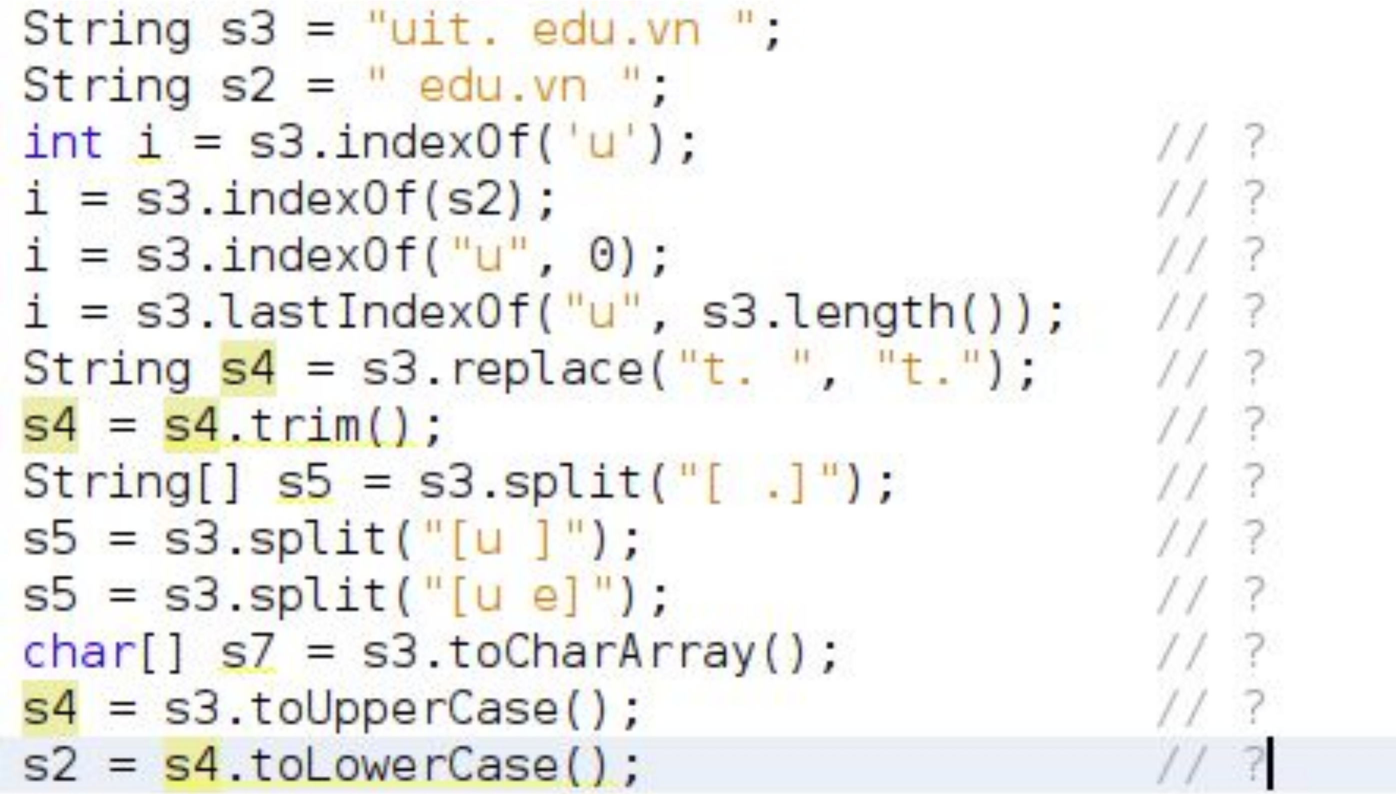
String - Example

```
String s1 = “uit.", s2 = " edu.vn ", s3, s4;
int i = s1.length();           // i = 4
boolean b = s1.isEmpty();      // false
char c = s1.charAt(i - 1);     // c = ‘.'
s3 = s1.concat(s2);            // “uit. edu.vn "
s4 = s1 + s2;                  // "uit. edu.vn "
b = (s3 == s4);                // false – String is an object
b = s3.equalsIgnoreCase(s4);   // true
s3 = s4.substring(3);          // “. edu.vn "
s3 = s4.substring(5, 8);       // “edu"
s3 = s1 + s2;                  // “uit. edu.vn "
b = s3.contains(s1);           // true
b = s3.endsWith(s2);           // true
b = s3.startsWith(s1);         // true
b = s3.startsWith(“edu", 5);   // true
```

String - Example

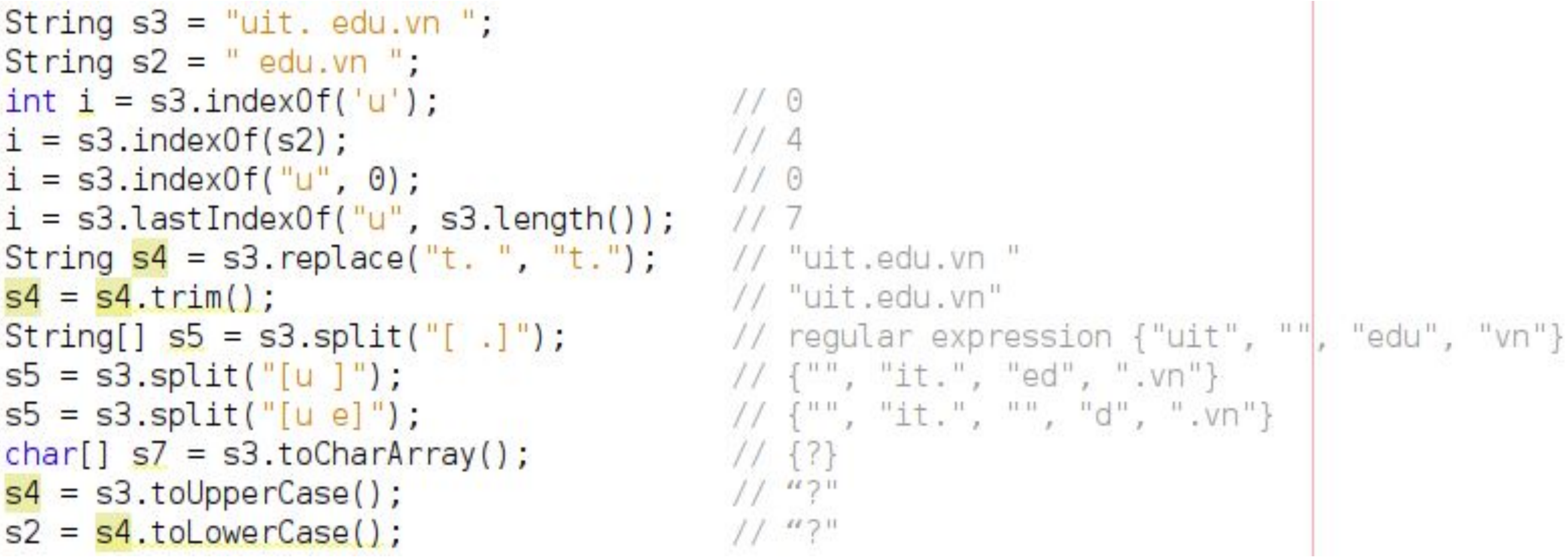
```
// s3 = “uit. edu.vn “; s2 = " edu.vn "
i = s3.indexOf(‘u');             // 0
i = s3.indexOf(s2);              // 4
i = s3.indexOf(“u”, 0);          // 0
i = s3.lastIndexOf(“u”, 2);      // 7

s4 = s3.replace("t. ", "t.");    // “?"
s4 = s3.trim();                  // “?"
String[] s5 = s3.split("[ .]");// regular expression {?}
s5 = s3.split("[u ]");            // {?}
s5 = s3.split("[u e]");           // {?}
char[] s7 = s3.toCharArray();    // {?}
s4 = s3.toUpperCase();           // “?"
s2 = s4.toLowerCase();           // “?"
```

String - Example



```
String s3 = "uit. edu.vn ";
String s2 = " edu.vn ";
int i = s3.indexOf('u');                               // ?
i = s3.indexOf(s2);                                    // ?
i = s3.indexOf("u", 0);                                // ?
i = s3.lastIndexOf("u", s3.length());                  // ?
String s4 = s3.replace("t. ", "t.");                   // ?
s4 = s4.trim();                                        // ?
String[] s5 = s3.split("[ .]");                        // ?
s5 = s3.split("[u ]");                                 // ?
s5 = s3.split("[u e]");                                // ?
char[] s7 = s3.toCharArray();                          // ?
s4 = s3.toUpperCase();                                 // ?
s2 = s4.toLowerCase();                                 // ?
```

String - Example



```
String s3 = "uit. edu.vn ";
String s2 = " edu.vn ";
int i = s3.indexOf('u');                // 0
i = s3.indexOf(s2);                     // 4
i = s3.indexOf("u", 0);                 // 0
i = s3.lastIndexOf("u", s3.length());   // 7
String s4 = s3.replace("t. ", "t.");    // "uit.edu.vn "
s4 = s4.trim();                         // "uit.edu.vn"
String[] s5 = s3.split("[ .]");         // regular expression {"uit", "", "edu", "vn"}
s5 = s3.split("[u ]");                  // {"", "it.", "ed", ".vn"}
s5 = s3.split("[u e]");                 // {"", "it.", "", "d", ".vn"}
char[] s7 = s3.toCharArray();           // {?}
s4 = s3.toUpperCase();                  // “?"
s2 = s4.toLowerCase();                  // “?"
```

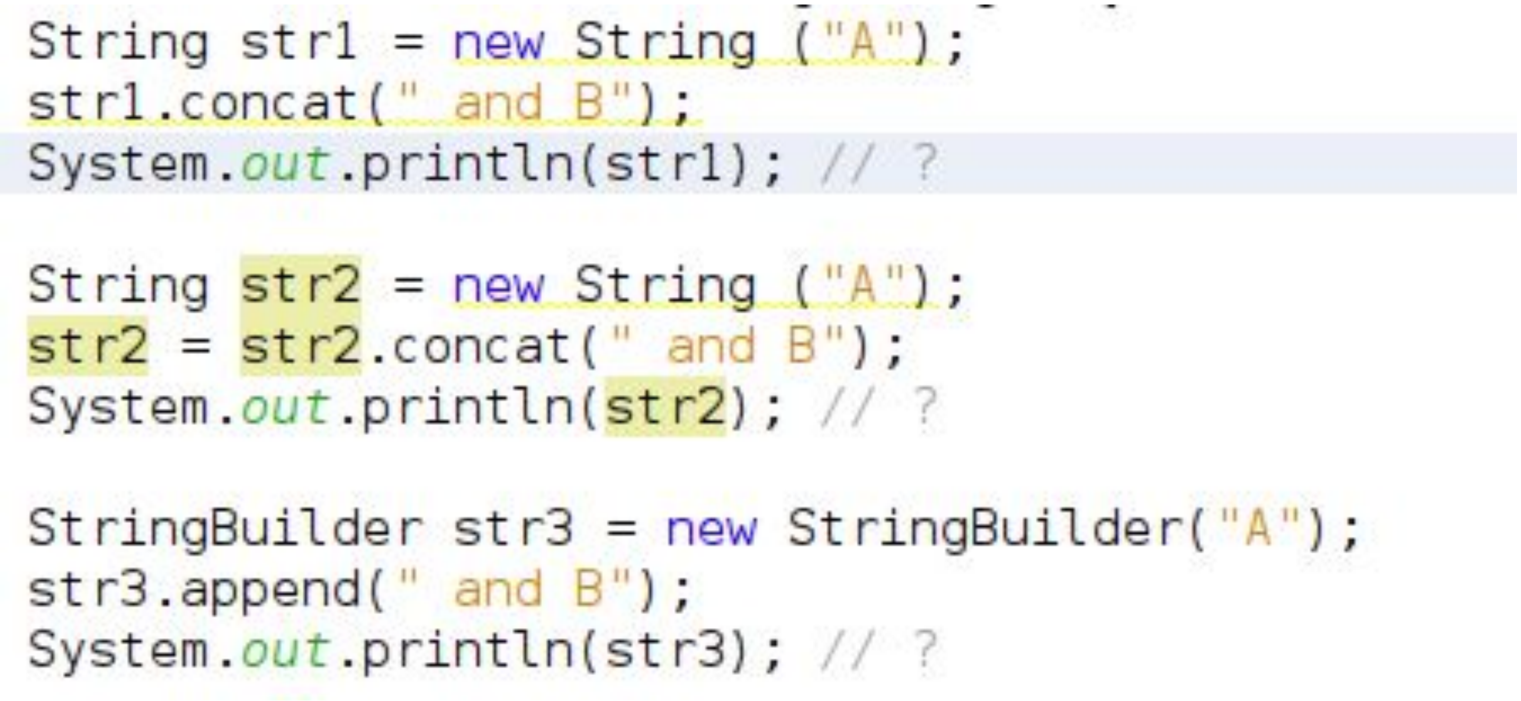
STRING/NUMBER CASTING

String/Number casting

```
// Each class in right hand side is called wrapper
// class of the corresponding primitive type
byte  b = Byte.parseByte("128");     // NumberFormatException
short s = Short.parseShort("32767");
int   x = Integer.parseInt("2");
int   y = Integer.parseInt("2.5");  // NumberFormatException
int   z = Integer.parseInt("a");    // NumberFormatException
long  l = Long.parseLong("15");
float f = Float.parseFloat("1.1");
double d = Double.parseDouble("2.5");
```

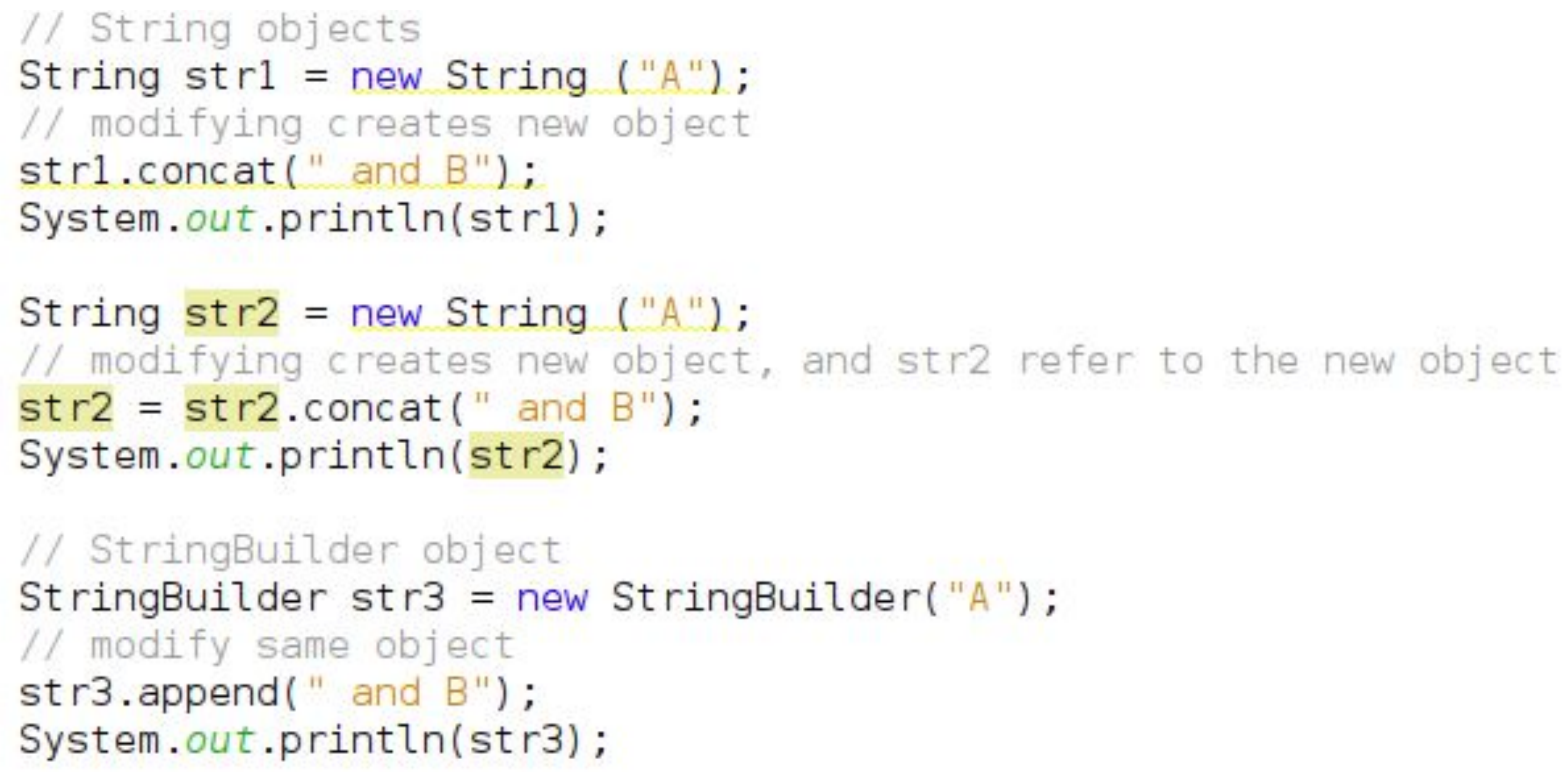
StringBuilder, StringBuffer

StringBuilder - Example



```
String str1 = new String ("A");
str1.concat(" and B");
System.out.println(str1); // ?

String str2 = new String ("A");
str2 = str2.concat(" and B");
System.out.println(str2); // ?

StringBuilder str3 = new StringBuilder("A");
str3.append(" and B");
System.out.println(str3); // ?
```

StringBuilder - Example



```
// String objects
String str1 = new String ("A");
// modifying creates new object
str1.concat(" and B");
System.out.println(str1);

String str2 = new String ("A");
// modifying creates new object, and str2 refer to the new object
str2 = str2.concat(" and B");
System.out.println(str2);

// StringBuilder object
StringBuilder str3 = new StringBuilder("A");
// modify same object
str3.append(" and B");
System.out.println(str3);
```

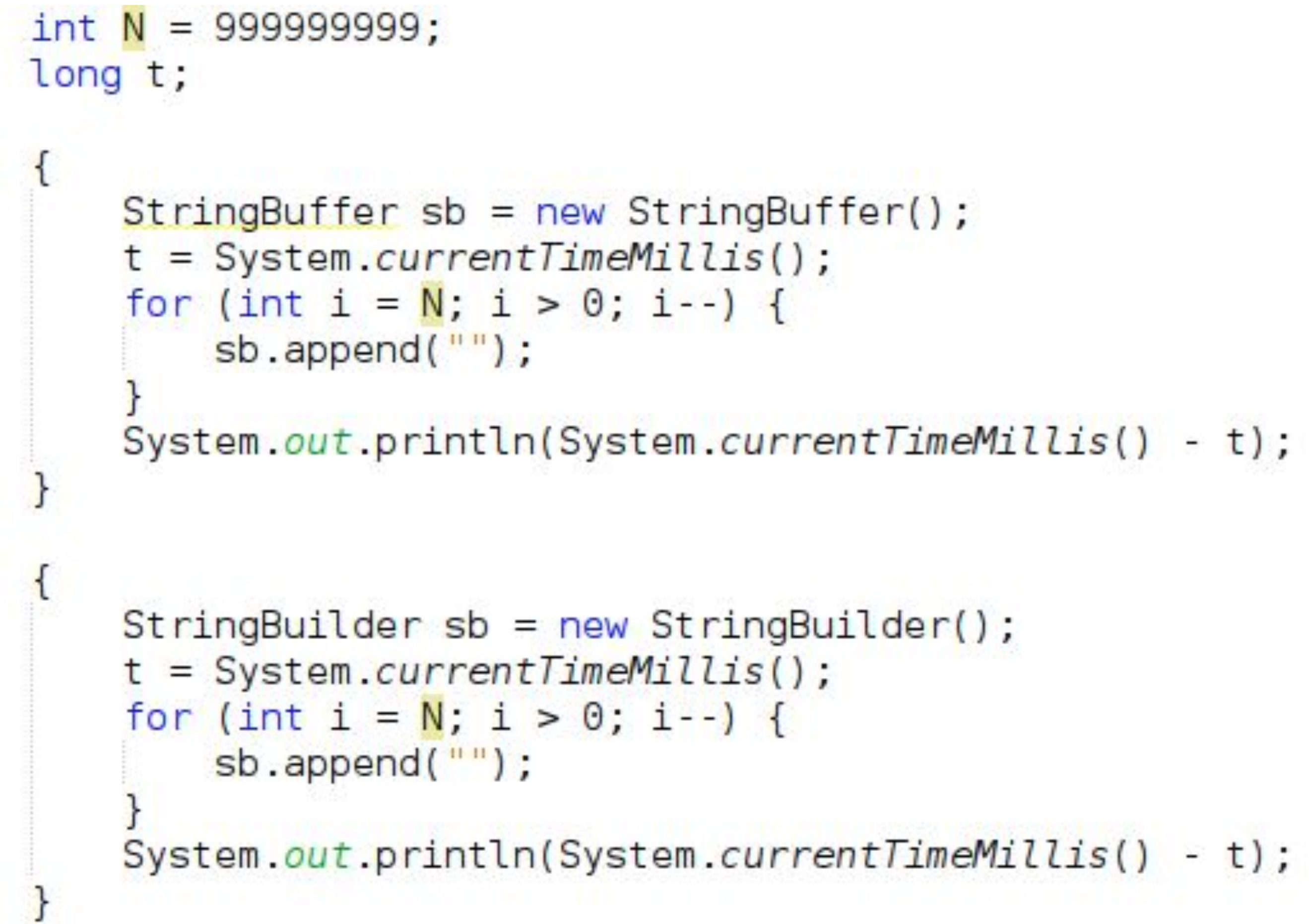
StringBuilder, StringBuffer?

- The **StringBuffer** and **StringBuilder** classes are used when there is a **necessity to make a lot of modifications to Strings of characters**.
- Unlike Strings, objects of type **StringBuffer and StringBuilder can be modified over and over again** without leaving behind a lot of new unused objects.
- It is recommended to use StringBuilder whenever possible because it is faster than StringBuffer. However, **if the thread safety is necessary, the best option is StringBuffer objects**.

StringBuilder/StringBuffer

```
StringBuilder sb = new StringBuilder("abc");
sb.append(" def");            // "abc def"
sb.delete(3, 5);              // "abcef"
sb.deleteCharAt(4);           // "abce"
sb.insert(3, " d");           // "abc de"
sb.replace(2, 4, " ghi");     // "ab ghide"
sb.reverse();                 // "edihg ba"
sb.setCharAt(5, 'j');         // "edihgjba"
// StringBuffer: thread safe version
// of StringBuilder
   => StringBuilder is faster?
```

StringBuilder is faster than StringBuffer?



```java
int N = 999999999;
long t;

{
    StringBuffer sb = new StringBuffer();
    t = System.currentTimeMillis();
    for (int i = N; i > 0; i--) {
        sb.append("");
    }
    System.out.println(System.currentTimeMillis() - t);
}

{
    StringBuilder sb = new StringBuilder();
    t = System.currentTimeMillis();
    for (int i = N; i > 0; i--) {
        sb.append("");
    }
    System.out.println(System.currentTimeMillis() - t);
}
```

Random Class

Random (java.util.Random)

```
Random rdm = new Random();
int i = rdm.nextInt(10); // a number from 0 to 9
i = rdm.nextInt();
 // equivalent to rdm.nextInt(Integer.MAX_VALUE)
long l = rdm.nextLong();
 // long number can be returned
byte[] bar = new byte[10];
rdm.nextBytes(bar);
 // bar now contains 10 byte random numbers
float f = rdm.nextFloat();    // from 0.0 to 1.0
double f = rdm.nextDouble();  // from 0.0 to 1.0
```

Math (java.lang.Math)

The Java Math class has many methods that allows you to perform mathematical tasks on numbers.

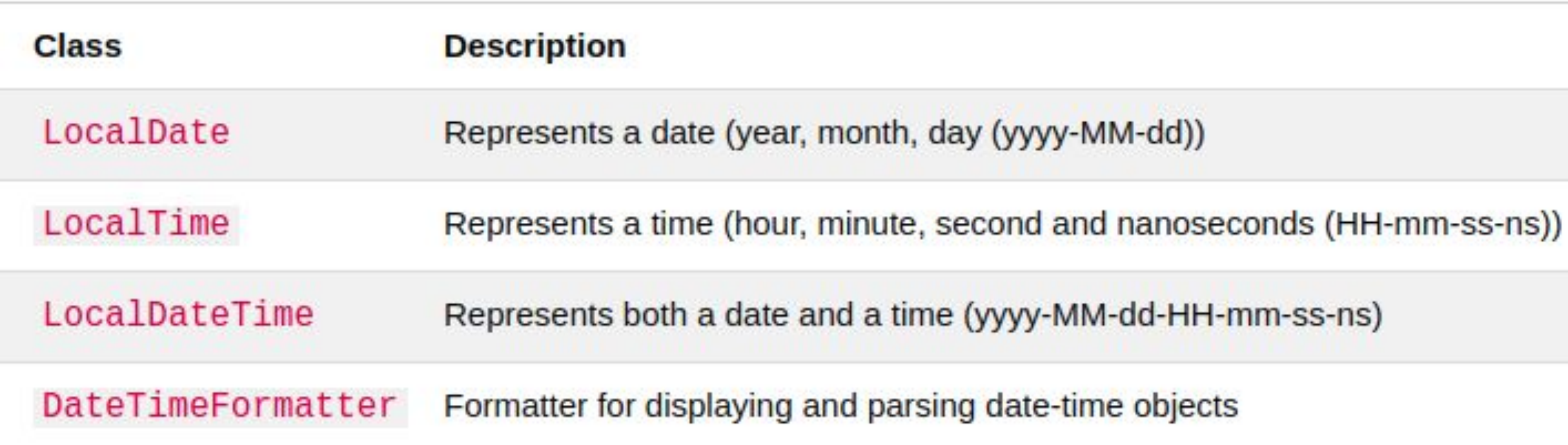
Demo

https://www.w3schools.com/java/java_math.asp

Java Data Time (java.time.*)

Some popular classes

(https://www.w3schools.com/java/java_date.asp)



Class	Description
`LocalDate`	Represents a date (year, month, day (yyyy-MM-dd))
`LocalTime`	Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))
`LocalDateTime`	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)
`DateTimeFormatter`	Formatter for displaying and parsing date-time objects

Java Data Time (java.time.*)

- Display Current Date
- Display Current Time
- Display Current Date and Time
- Formatting Date and Time

Demo

(https://www.w3schools.com/java/java_date.asp)

Java Data Time (java.util.*)

java.util.GregorianCalendar;

java.util.Date;

java.text.SimpleDateFormat;

java.util.Calendar;

Java Data Time (java.util.*)

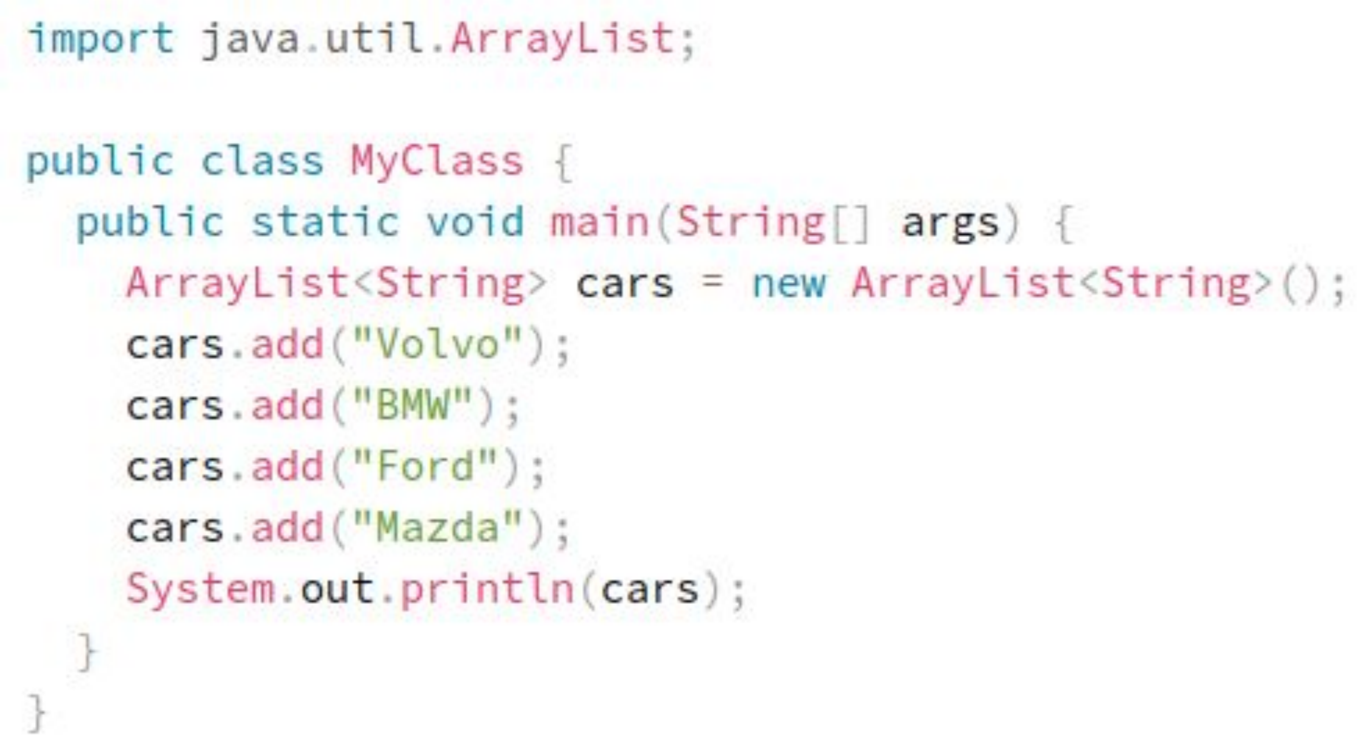
```
import java.util.*;
import java.text.SimpleDateFormat;
SimpleDateFormat df = new SimpleDateFormat(
                                     "yyyy-MM-dd hh:mm:ss.SSS");
GregorianCalendar cld1 = new GregorianCalendar();
// current date time
try {
    Date d = df.parse("2014-13-36 36:65:82.976");
    String s = df.format(d); // "2015-02-06 13:06:22.976"
    cld1.setTime(d);
} catch (ParseException e) {}
int year = cld1.get(Calendar.YEAR);          // 2015
int month = cld1.get(Calendar.MONTH);        // 02
boolean b = month == Calendar.JANUARY;       // false
int day = cld1.get(Calendar.DAY_OF_MONTH);   // 02
int dayw = cld1.get(Calendar.DAY_OF_WEEK);   // 06
b = dayw == Calendar.FRIDAY;                 // true
```

Java Data Time (java.util.*)

```
int hour = cld1.get(Calendar.HOUR);          // 04
int minute = cld1.get(Calendar.MINUTE);      // 06
int second = cld1.get(Calendar.SECOND);      // 22
int milisec = cld1.get(Calendar.MILLISECOND);// 976

GregorianCalendar cld2 = (GregorianCalendar)cld1.clone();
cld2.add(Calendar.YEAR, -1);
// same operator for other fields too
year = cld2.get(Calendar.YEAR);              // 2014
b = cld1.after(cld2);                        // true
b = cld1.before(cld2);                       // false
```

Java ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.



```
import java.util.ArrayList;

public class MyClass {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

https://www.w3schools.com/java/java_arraylist.asp

Java ArrayList

Loop through the elements of an `ArrayList` with a `for` loop

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
for (int i = 0; i < cars.size(); i++) {
    System.out.println(cars.get(i));
}
```

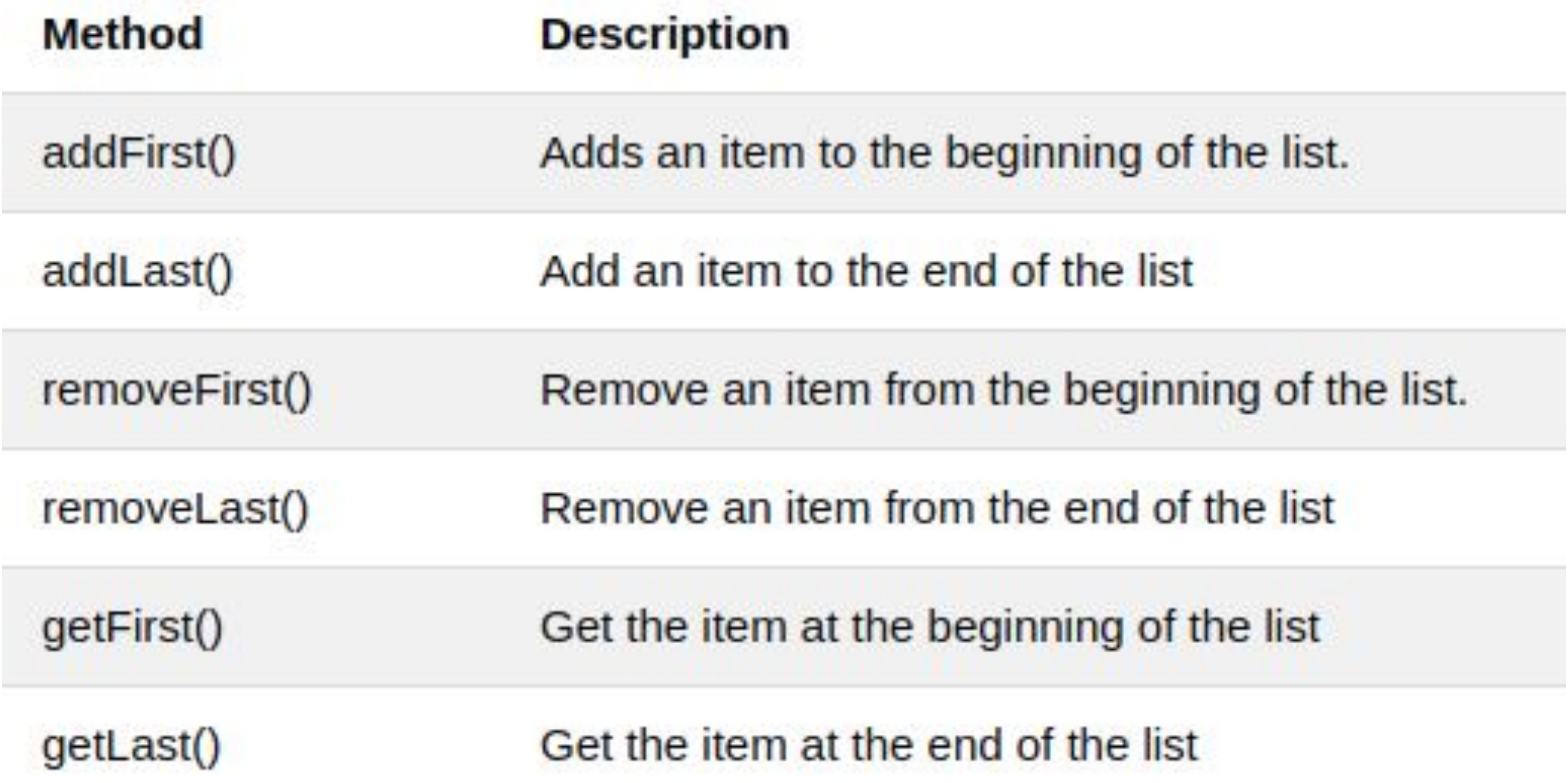
Java ArrayList

loop through an ArrayList with the for-each loop

```
ArrayList<String> cars = new
ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");
for (String i : cars) {
    System.out.println(i);
}
```

Java LinkedList

- The LinkedList class is a collection which **can contain many objects of the same type, just like the ArrayList**.
- The LinkedList class **has all of the same methods as the ArrayList class because they both implement the List interface**. This means that you can add items, change items, remove items and clear the list in the same way.



Method	Description
addFirst()	Adds an item to the beginning of the list.
addLast()	Add an item to the end of the list
removeFirst()	Remove an item from the beginning of the list.
removeLast()	Remove an item from the end of the list
getFirst()	Get the item at the beginning of the list
getLast()	Get the item at the end of the list

Java LinkedList - When to use?

It is best to use an ArrayList when:

- You want to **access random items frequently**
- You only need to add or remove elements at the end of the list

It is best to use a LinkedList when:

- You only use the list by looping through it instead of accessing random items
- You frequently need to add and remove items from the beginning or middle of the list

Java HashMap (java.util.HashMap)

- With the ArrayList, you learned that Arrays store items as an ordered collection, and you have to access them with an index number (int type). A HashMap however, store items in "key/value" pairs, and you can access them by an index of another type (e.g. a String).
- One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values:

```java
// Create a HashMap object called people
HashMap<String, Integer> people = new
HashMap<String, Integer>();
// Add keys and values (Name, Age)
people.put("John", 32);
people.put("Steve", 30);
people.put("Angie", 33);
for (String i : people.keySet()) {
    System.out.println("Name: " + i + " Age: "
    + people.get(i));
}
```

Java HashSet (java.util.HashSet)

A HashSet is **a collection of items** where every item is unique

- add()
- contains()
- remove()
- clear()

Demo: https://www.w3schools.com/java/java_hashset.asp

```java
// Create a HashSet object called numbers
HashSet<Integer> numbers = new HashSet<Integer>();

// Add values to the set
numbers.add(4);
numbers.add(7);
numbers.add(8);

// Show which numbers between 1 and 10 are in the set
for(int i = 1; i <= 10; i++) {
    if(numbers.contains(i)) {
          System.out.println(i + " was found in the set.");
     } else {
        System.out.println(i + " was not found in the set.");
    }
}
```

Enumerate

Simple Enum

```
// Declaration
enum WorkingDays {MONDAY, TUESDAY,
  WEDNESDAY, THURSDAY, FRIDAY}

// Using
WorkingDays wd = WorkingDays.TUESDAY;
switch (wd){…}
```

Complex enum

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6);
    // two members, correspond to two constants in enum elements
    private final double mass;   // in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) { // call automatically
        this.mass = mass;
        this.radius = radius;
    }
    public double mass() { return mass; }
    public double radius() { return radius; }
}
…
float mass = EARTH.mass()
…
for (Planet p: Planet.values()){…p.mass() … p.radius()… }
```

Generic Type

One type generic

```
class GenericType<T>{
   // T is a type representation, not a specific type
   private T aT;
   public T getMember(){return aT;}
   public void setMember(T newT){aT = newT;}
}
class A{}

// use generic class with specific type int
GenericType<int> gInt = new GenericType<int>();
gInt.setMember(5);
int i = gInt.getMember();
```

Bounded generic type

```java
class GenericType<T extends A>{
   // T is a type representation, not a specific type
   // A is a specific type
   private T aT;
   public T getMember(){return aT;}
   public void setMember(T newT){aT = newT;}
}
class A{}
class B extends A{}
class C{}

GenericType<A> gA = new GenericType<A>(); // OK
GenericType<B> gB = new GenericType<B>(); // OK too
GenericType<C> gA = new GenericType<C>(); // Error, C is not A
```

Generic Collection

ArrayList: Input

```
class A{int i;}
A[] arA = new A[10];              // Predefined capacity required
…
List<A> alA = new ArrayList<A>(); // No predefined capacity
boolean b = alA.isEmpty();        // true

A aA = new A(); aA.i = 1;
alA.add(aA);                           // add new
b = alA.isEmpty();                     // false
alA.add(aA);                 // add new again, duplicate accepted

A aoA = new A(); aoA.i = 2;
alA.add(1, aoA);          // insert to the 2nd position, (1, 2, 1)
```

ArrayList: Output

```
int s = alA.size();  // 3

A outA = alA.get(2);
b = outA == aoA;     // true

outA = alA.get(3);   // error, out of range

alA.set(2, aoA);     // replace the 3rd position, (1, 2, 2)

int i = alA.indexOf (aoA); // 1
i = alA.lastIndexOf (aoA); // 2

for (A a: alA){System.out.println(a.i);}         // 1, 2, 2

alA.remove(1);       // remove the 2nd position, (1, 2)
```

ArrayList: Sort by Arrays

```
class A implements Comparable<A>{   // implement Comparable<T>
   int i;
   public int compareTo(A another){ // implement compareTo(T t)
       if (i == another.i) return 0;
       if (i < another.i) return -1;
       return 1;
   }
}

Object[] arA = alA.toArray();       // convert to array
Arrays.sort(arA);                   // using Arrays.sort
for (Object a: arA){
   A a1 = (A)a;                     // revert to original type
   System.out.println(a1.i);
}
```

HashMap: Input

```
class A{int i;}
HashMap<int, A> aMap = new HashMap<int, A>();
// Error, key must be an object type
HashMap<Integer, A> aMap = new HashMap<Integer, A>();
// use the hash code of key then no order is warranted
boolean b = aMap.isEmpty();             // true

A aA = new A(); aA.i = 1;
aMap.put(1, aA);   // add new
b = aMap.isEmpty();    // false
int i = aMap.size();   // 1

aMap.put(1, aA);   // replace the older one
i = aMap.size();   // no new adding with the same key
```

HashMap: Output

```
b = aMap.containsKey(1);      // true
b = aMap.containsValue(aA);   // true

A oA = aMap.get(1);           // access by key
B = oA == aA;          // true

oA = aMap.get(2);          // oA = null

b = aMap.remove(1);           // access by key
```

Annotation

Annotation

```
class A{
   public int doSmt();

   @Deprecated()       // Do not use the next method
   public int oldMethod(){}

   @SuppressWarnings("deprecation")
               // Do not display the warning on
               // the use of a deprecated method
   public int aMethod(){oldMethod();}
}
class B{
   @Override   // The next method overrides a base method
   public int doSmt();
}
```

PlanText File I/O

Plan text file I/O

```
// Type
import java.io;
…
try{
   // File exist
   if (File.exists("a.txt")){
      // Open
      BufferedReader input =  new BufferedReader(new FileReader("a.txt"));
      BufferedWriter output =  new BufferedWriter(new FileWriter("b.txt"));
      String line;
      // Repeat access until end of input
      while ((line = input.readLine()) ==  null){
         output.write(line); output.newLine();
      }
      // close
      input.close(); output.close();
   }
} catch (IOException e){
   String msg = e.getMessage();
}
```