# primitive type

Value of primitive type variable is stored in memory.

value can't be changed or updated. For example you can't negate a boolean value using logical complement operator ! or int value using minus operator -.

|  |
| --- |
| 2 and -2 are stored in different address |
| int a= 2;  a= -a; |
| true and !true are different |
| boolean b = true;  b = !true; |

## int

char/unicode

byte/short/int/long

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## Integer

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## casting

char/byte/short will be automatically casted to int type.

|  |
| --- |
| sum==90  wrong:  byte sum = (byte) a + (byte) b;  byte sum = (byte) a + b; |
| byte a = 40, b = 50;  **int sum = a + b;**  System.out.println(sum); |
| byte a = 40, b = 50;  **byte sum = (byte) (a + b); // must explicitly announce** |

|  |
| --- |
|  |
|  |

## debugging

### wrong declaration

|  |
| --- |
| Unicode code should be encompassed by single quote '\u004e'. |
| char[] d = {0x4e, \u004e, 78}; |

|  |
| --- |
| string can't be declared as boolean type  the ternary operator has two expressions, one of them a String and the other a boolean value. The ternary operator is permitted to have expressions that don’t have matching types, but the key is here the assignment to the String reference. The compiler knows how to assign the first expression value as a String, but the second boolean expression cannot be set as a String; therefore, this line will not compile. |
| public class CompareValues {  public static void main(String[] args) {  int x = 0;  while(x++ ˂ 10) {}  String message = x ˃ 10 ? "Greater than" : false;  System.out.println(message+","+x);  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

# array

## array

array must be initialized once it is declared.

Elements of an array are stored in consecutive memory block.

Array support random access which access memory directly. Any elements can be accessed by index.

Array is static type. Array doesn't support append, insert or delete. Size of array can't be changed once it was initialized.

**Note: array is mutable because elements of array could be updated.**

### define

|  |
| --- |
| The below are equal. |
| int[] a = {1,2,3}; |
| int c[] = {1,2,3}; |
| int[] a = {1,2,3};  int[] b;  b=a; |
| int[] a = {1,2,3};  int[] b = a; |
| int[] d= new int[10];  d[0]=1; d[1]=2; d[2]=3; |
| int[] d= new int[] {1,2,3}; |

|  |
| --- |
| output:0 |
| int[] e = new int[0];  System.out.println(e.length); |

print array using java.util.Arrays.toString()

|  |
| --- |
| [good] |
| String[] s = {"good"};  System.out.println(**java.util.Arrays.toString(s)**); |

Integer type array

|  |
| --- |
| output:  [null, null, null, null, null, null, null, null, null, null]  [1, 2, 3] |
| import java.util.Arrays;  public class ArrayInteger {  public static void main(String[] args){  // method 1  **Integer[] a1 = new Integer[10];**  System.out.println(Arrays.asList(a1));  //2:  **int[] a = {1,2,3};**  **Integer[] a2 = new Integer[a.length];**  **Arrays.setAll(a, i->a2[i]=a[i]);**  System.out.println(Arrays.asList(a2));  }  } |

|  |
| --- |
| declaration is legal. but numbers[0][2] or numbers[1][1] is out of bounds. |
| int[][] numbers = {{1,2}, {3}, {4,5,6}}; |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### features

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## 2D-array

### define

|  |
| --- |
|  |
|  |

|  |
| --- |
| one-dimension array b, and 2-D array c are created. |
| int[] b, c[]; |

### features

|  |
| --- |
|  |
|  |

# ArrayList

double-linked list. Differing from Array, Arraylist has such features:

* All elements may be stored in memory discontinuously.
* list is dynamic type. support add/insert/update/delete. array is static
* list doesn't support random access by index directly. array supports random access.
* list is wrapper class which define methods. Array only define attributes. for example: "array.length" compared with "list.size()".

## declaration and initialization

### declaration

The approaches below are equal

|  |
| --- |
| works since java5 |
| ArrayList a1 = new ArrayList();  ArrayList a2 = new ArrayList(10);  ArrayList a3 = new ArrayList(a2); |

|  |
| --- |
| works since java8. |
| ArrayList<String> a4 = new ArrayList<String>();  ArrayList<String> a5 = new ArrayList<>(); |
|  |

|  |
| --- |
| interface pattern. "List" is interface. ArrayList is concrete class |
| List<String> a6 = new ArrayList<>();  List<String> a7 = Arrays.asList("one", "two"); |
|  |

|  |
| --- |
| convert array to arraylist |
| String[] lizards = {"iguana", "gecko"};  ArrayList<String>list = **Arrays.asList(lizards);** |
| String[] lizards = {"iguana", "gecko"};  List<String>list = Arrays.asList(lizards); |

|  |
| --- |
| output: Bacd |
| List˂String˃ list = Arrays.asList("a", "B", "d", "c");  Collections.sort(list);  String[] array = list.toArray(new String[4]);  for (String string : array) System.out.print(string); |

|  |
| --- |
|  |
|  |

### types of arraylist

|  |
| --- |
| String type arraylist |
| ArrayList<String> a4 = new ArrayList<String>();  a4.add("a");  a4.add("b"); |

|  |
| --- |
| a8 = [5, 6, 53, 97, 1] |
| ArrayList<Integer> a8 = new ArrayList<>();  a8.add(new Integer(5)); //standard patterns  a8.add(6); //cast int to wrapper class  a8.add(new Integer('5'));  a8.add(new Integer('a'));  a8.add(new Integer("1")); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## CRUD methods

obj = list.get(int) return element

int = list.IndexOf(obj) return index

int = list.size() return number of elements

boolean = list.contains(obj)

boolean = list.add(obj) / list.add(int, obj) insert one element

boolean = list.remove(obj) / list.remove(int) delete one element

boolean = list.set(int, obj). update one element

null = list.clear()

|  |
| --- |
| [false, goldfish, false, 1] |
| List differentTypes = new ArrayList();  differentTypes.add("goldfish");  differentTypes.add(0, false);  differentTypes.add(new Boolean(false));  differentTypes.add(1);  System.out.println(differentTypes); |

|  |
| --- |
| b=[3,0,1]. remove the element index=3 |
| ArrayList<Integer> b = new ArrayList<Integer>(10);  b.add(3);b.add(0);b.add(1);b.add(2);  **b.remove(3);**  System.out.println(b); |
| b=[0,1,2] remove the integer 3. |
| ArrayList<Integer> b = new ArrayList<Integer>(10);  b.add(3);b.add(0);b.add(1);b.add(2);  **b.remove(new Integer(3));** |

|  |
| --- |
| output: 04 |
| ArrayList<Integer> b = new ArrayList<Integer>(10);  System.out.print(**b.size()**);  b.add(3);b.add(0);b.add(1);b.add(2);  System.out.println(**b.size()**); |

|  |
| --- |
| true  false |
| List d = new ArrayList();  d.add(false);  System.out.println(d.contains(new Boolean(false)));  System.out.println(d.get(0) == (new Boolean(false))); |

|  |
| --- |
| output:  567Exception in thread "main" java.lang.NullPointerException: Cannot invoke "java.lang.Integer.intValue()" because the return value of "java.util.Iterator.next()" is null      at ArrayTest.main(ArrayTest.java:13) |
| List<Integer> ages = new ArrayList<>();  ages.add(Integer.parseInt("5"));  ages.add(Integer.valueOf("6"));  ages.add(7); //autoboxing  **ages.add(null); //Integer object could be null**  for (int age : ages) System.out.print(age); //throws NullPointerException |

|  |
| --- |
|  |
|  |

## Reshape

### convert

convert array to ArrayList:

ArrayList = java.util.Arrays.asList(array)

java.util.Collections.addAll(ArrayList, array)

for( Object el: array) ArrayList.add(el);

|  |
| --- |
| output:  [a, b, c]  [a, b, c]  [a, b, c] |
| import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collections;  import java.util.List;  public class ArrayListConvert {  public static void main(String args[]) {  String[] array = {"a", "b", "c"};    //Method 1  **List<String> list = Arrays.asList(array);**  System.out.println(list);    //Method 2  List<String> list1 = new ArrayList<String>();  **Collections.addAll(list1, array);**  System.out.println(list1);    //Method 3  List<String> list2 = new ArrayList<String>();  for(String text:array) { list2.add(text); }  System.out.println(list2);  }  } |

convert ArrayList to array:

array = ArrayList.toArray();

for(Object el: ArrayList) array[i++]=el

|  |
| --- |
|  |
| import java.util.ArrayList;  import java.util.Arrays;  public class ArrayConvert {  public static void main(String[] args) {  ArrayList<String> list = new ArrayList<>(2);  list.add("A");  list.add("B");  list.add("C");  list.add("D");    //method 1:  **Object[] array = list.toArray();**  System.out.println( Arrays.toString(array) );    //method 2:  String[] arr = new String[4];  int i=0;  for(Object o : list)  arr[i++] = (String) o;  System.out.println(Arrays.toString(arr));  }  } |

### copy

Arraylist is object in Java.

shallow copy: create reference variable and refer to the same object.

deep copy: create a new object.

|  |
| --- |
| output:  [a, b]  [a, b, c]  list (class variable), list2(local variable of copy1()), list3(local variable of main()) refer to the same object. |
| import java.util.ArrayList;  public class ArrayListCopy {  ArrayList<String> list = new ArrayList<String>();  ArrayListCopy(){  list.add("a");  }  ArrayList copy1(){  **ArrayList<String> list2 = list; //shallow copy**  list2.add("b");  return list2;  }  void print(){  System.out.println(list);  }  public static void main(String[] args){  ArrayListCopy t = new ArrayListCopy();  ArrayList<String> list3 = t.copy1();  t.print();  list3.add("c");  t.print();  }  } |

deepcopy

ArrayList<String> list1 = new ArrayList<String>(list);

OR

ArrayList<String> list1 = (ArrayList<String>) list.clone();

OR

        ArrayList<String> list1 = new ArrayList<String>();

        list1.addAll(list);

OR

        ArrayList<String> list1 = new ArrayList<String>();

        list.forEach((i)->list1.add(i));

OR

        ArrayList<String> list1 = new ArrayList<String>();

        for(String i: list) list1.add(i);

OR

OR

        ArrayList<String> list1 = (ArrayList<String>) list.stream().collect(Collectors.toList());

|  |
| --- |
| output:  [a, b]  [a]  [a] |
| import java.util.ArrayList;  public class ArrayListCopy2 {  ArrayList<String> list = new ArrayList<String>();  ArrayListCopy2(){  list.add("a");  }  void print(){  System.out.println(list);  }  ArrayList copy(){  **ArrayList<String> list1 = new ArrayList<String>(list);**  list1.add("b");  return list1;  }  public static void main(String[] args){  ArrayListCopy2 t = new ArrayListCopy2();  ArrayList<String> b = t.copy();  System.out.println(b);  t.print();  b.add("1");  t.print();  }  } |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

# String

## define String

All String literals are automatically instantiated into a String object.

create string

|  |
| --- |
| content of the three strings are identical |
| String s1= "abcdef";  String s2 = new String("abcdef");  String s3 = new String(s1); |

concatenate strings using operator +

|  |
| --- |
| s=="\#" |
| String s = "\\"+"#";  System.out.println(s); |

|  |
| --- |
| copy reference |
| String s3, s4="abc";  s3 = s4;  System.out.println(s3); |

|  |
| --- |
| convert stringbuilder type to string. |
| StringBuilder s = new StringBuilder("abcdef");  String s5= s.toString();  System.out.println(s5); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## immutable

**Keep in mind that String is immutable. So String type variable is final variable. Stringbuilder or Array is mutable.**

An immutable object is an object whose internal state remains constant after it has been entirely created. Immutable object can't be modified, and can be garbage collected.

String as well as StringBuffer is thread-safe because they are immutable.

|  |
| --- |
| output: rock |
| String s = "java";  s += "rocks";  s = s.substring(4,8);  **s.toUpperCase(); //don't update s.**  System.out.println(s); |

|  |
| --- |
| output: abd |
| String e = "abc";  **e.concat("d");**  System.out.println(e); |

|  |
| --- |
| output: abc |
| String c = new String("abc");  String d = c; //c,d refer to "abc"  c = "xyz"; // c refers to "xyz", d to "abc"  System.out.println(d); |

|  |
| --- |
| output:  abcde  abcde  reference "f" and "g" point to the same object |
| StringBuilder f = new StringBuilder("abc");  StringBuilder g = f.append("d");  g.append("e");  System.out.println(f);  System.out.println(g); |

|  |
| --- |
| output:  abc  abc  abc  abcde  abcde |
| public class StringImmutable{  public static void main(String[] args){  // string is immutable  // copy string  String a = "abc";  String b = a;  // create a new object "xyz" and assign to reference "a"  a="xyz";  System.out.println(b);  String c = new String("abc");  String d = c;  c = "xyz";  System.out.println(d);  //string is immutable  String e = "abc";  e.concat("d");  System.out.println(e);  // stringBuilder is mutable  StringBuilder f = new StringBuilder("abc");  StringBuilder g = f.append("d");  g.append("e");  // reference "f" and "g" point to the same object  System.out.println(f);  System.out.println(g);  }  } |

|  |
| --- |
|  |
|  |

## methods

### get

**int = str.length()**

**char = str.charAt(index)** get char by index

**index = str.indexOf(char/str2)**

**str = str.toString()** convert object to string type

|  |
| --- |
|  |
| String s1= "abcdef";  System.out.print(s1.length());//==6 |
| String s = "";  System.out.println(s.length()); //==0 |

|  |
| --- |
|  |
| String s1= "abcdef";  String s2 = new String("abcdef");  System.out.print(s1.charAt(0)); //=='a'  System.out.print(s2.charAt(1)); //=='b' |

|  |
| --- |
| // indexOf(val/str2) return index or -1 if not found |
| String s1= "abcdef";  String s2 = new String("abcdef");  System.out.print(s1.indexOf('a'));  System.out.print(s2.indexOf(32));  System.out.println(s2.indexOf("def")); //==3 |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### reshape

**str2 = str.substring(start-index, <end-index>)**

**str2 = str.trim()** Removes whitespace from both ends of a string. Note: In python, str.trim() only remove whitespaces of the end.

**String[] = str.split(str2/<regular expression>)**

**str2 = str.toUpperCase() / str2 = str.toLowerCase()**

|  |
| --- |
|  |
| String s1= "abcdef";  System.out.println(s1.substring(1,3)); // =="bc"  System.out.println(s2.substring(2)); //=="cdef"  System.out.println(s2.substring(2,2)); //"" |
| String s1= "abcdef";  String s2 = s1.substring(0, s1.length()-1);  System.out.println(); //remove the last char |

|  |
| --- |
|  |
| String s = "(345) 456 -1234";  String[] s2= s.split(" ");  System.out.println(s2[0]); //=="(345)"  String[] s3= s.split("");  System.out.println(s3[0]); //=="("  String[] s4= s.split("[ -]");  System.out.println(s4[0]); //=="(345)" |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### find

**boolean = str.contains(char/str2)**

**boolean = str.startswith(str2)**

**boolean = str.endswith(str2)**

|  |
| --- |
| output:  truefalse  truefalse |
| String s1= "abcdef";  String s2 = new String("abcdef");  System.out.print(s1.startsWith("a"));  System.out.println(s2.endsWith("a"));  System.out.print(s1.contains("a"));  System.out.println(s2.contains("1")); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### replace

**newstr =str.replace(char1/str1, char2/str2)**

**newstr =str.replaceFirst(char1/str1, char2/str2)**

**newstr =str.replaceAll(<regular expression>, str2)**

|  |
| --- |
| replace white space with '-' |
| String s = " abc d";  **s=s.replace(' ', '-'); // "**-abc-d**"** |
| remove the first white space |
| String s = " abc d";  **s=s.replace(" ", ""); //"**abcd**"** |

|  |
| --- |
| remove all white space |
| String s = " abc d";  s=s.replaceFirst(" ", ""); //"abc d" |

|  |
| --- |
| remove all whitespace and '-' |
| String s = " ab-c d";  s=s.replaceAll("[ -]", ""); //"abcd" |
|  |
| String s = "(456) 789 -1234";  s=s.replaceAll("[() -]", ""); //4567891234 |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### concatenate

operate +

**str2 = str.concat(str1)**

|  |
| --- |
| output: abcde12.3false  // + works with char, string, int, float, boolean |
| String s= "a";  String s1= "bc";  s+=s1; //string type  s+="d";  s+='e'; //char type  s+=1; // int type  s+=2.3f; // float type  s+=false; // boolean  System.out.println(s); |

|  |
| --- |
| s=="ab" |
| String s="a";  s=s.concat("b");  s.concat("c"); //string is immutable.  System.out.println(s); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

### compare

**Boolean = str.equals(str2)**

**Boolean = str.equalsIgnoreCase(str2)**

**index= str.compareTo(str2)** Compares two strings lexicographically

**index= str.compareToIgnoreCase(str2)** Compares two strings lexicographically

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

# Stringbuilder

## define

StringBuilder replace StringBuffer after SE5. StringBuilder is faster because it doesn’t need to be thread-safe.

|  |
| --- |
|  |
| StringBuilder s1 = new StringBuilder();  StringBuilder s2 = new StringBuilder("");  StringBuilder s3 = new StringBuilder("abc");  StringBuilder s4 = new StringBuilder("123");  StringBuilder s5 = new StringBuilder(4);  System.out.println(s5.toString());  System.out.println(s5.length()); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## methods

Only append(), insert(), reverse() and delete() support chaining because they all return StringBuilder type.

### get

**len = str.length()**

**char = str.charAt(index)**

**index/-1 = str.indexOf(char/str2)**

**str2 = str.substring(start-index, <end-index>)**

**str2 = str.toString()**

Note: those methods can't be chained because they return int or String type.

|  |
| --- |
| 6 |
| StringBuilder s3 = new StringBuilder("abcdef");  System.out.println(s3.length()); |

|  |
| --- |
| d |
| StringBuilder s3 = new StringBuilder("abcdef");  System.out.println(s3.charAt(3)); |

|  |
| --- |
|  |
| StringBuilder s = new StringBuilder("abcdef");  System.out.println(s.substring(0)); //==abcdef  System.out.println(s.substring(2,3)); //==c  System.out.println(s.substring(3)); //=def  System.out.println(s.substring(2,2)); //empty string |

|  |
| --- |
|  |
| StringBuilder s3 = new StringBuilder("abcdef");  System.out.print(s3.indexOf('a')); //==0  System.out.print(s3.indexOf(32)); // ==-1 not found  System.out.println(s3.indexOf("def")); //==3 |

### change/update

strBuilder = str.append(char/str2/obj)

strBuilder = str.insert(index, char/str2/obj)

strBuilder r = str.reverse()

Note: those methods could be chained.

|  |
| --- |
| output:  abcdef45.6Ntrue7.8  abcdef45.6Ntrue7.8 |
| StringBuilder s3 = new StringBuilder("abc");  s3.append('d');  s3.append("ef");  s3.append(4);  s3.append(5.6f);  s3.append('\u004e');  s3.append(true);  System.out.println(s3.append(7.8));  System.out.println(s3); |

|  |
| --- |
| output:  da3becjava.lang.Object@2c7b84de |
| StringBuilder s = new StringBuilder("abc");  s.insert(0, 'd');  s.insert(3, "e");  s.insert(2, 3);  **s.insert(s.length(), new Object()); //append the end**  System.out.println(s); |

|  |
| --- |
|  |
| StringBuilder s = new StringBuilder("0123456");  s.reverse();  System.out.println(s); |

### delete

str = str.delete(start, end)

str = str.deleteCharAt(index)

Note: those methods could be chained.

|  |
| --- |
| output:  012345  01245  012  012 |
| StringBuilder s3 = new StringBuilder("0123456");  // delete value by index  System.out.println(s3.deleteCharAt(6));  // delete substring by start-end  // delete 4th  s3.delete(3,4);  System.out.println(s3);  // delete all from 4th  // note: can't omit end index and out of bounds is ok.  s3.delete(3, 20);  System.out.println(s3);  // no deletion though it is ok.  s3.delete(1, 1);  System.out.println(s3); |

|  |
| --- |
| s=123456 |
| StringBuilder s = new StringBuilder("0123456");  s.deleteCharAt(0);  System.out.println(s); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## debugging

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

# Date and Time

Data of Date, Time and DateTime are declared in different class of the package "java.time", which upgrade from "java.util." in Java5.

|  |  |
| --- | --- |
| class | static methods |
| import java.time.LocalDate;  import java.time.LocalTime;  import java.time.LocalDateTime; | <>.now()  <>.of(…)  <>.parse("…") |
| import java.time.Month; | Month.JANUARY |
| import java.time.Period; | Period.of(year,month,day)  Period.ofDays(int)  Period.ofMonths(int)  Period.ofYears(int) |
| import java.time.format.DateTimeFormatter; | DateTimeFormatter.ofPattern("…"); |
|  |  |

Note: They are static methods. No instance is required. For example: "new LocalDate.of(2014,5,21);" is wrong./

## create data/time

<String> = LocalDate.now()

<String> = LocalDateTime.now()

<String> = LocalTime.now()

<LocalDate> = LocalDate.of(int year, int month/ Month month, int dayofmonth)

<LocalDateTime> = LocalDateTime.of(int year, int month/ Month month, int dayofmonth, int hour, int minute, int second, int nano)

<LocalTime> = LocalTime.of(int hour, int minute, int second, int nano )

|  |
| --- |
| output:  2022-01-11  08:59:32.109968400  2022-01-11T08:59:32.109968400 |
| System.out.println(LocalDate.now());  System.out.println(LocalTime.now());  System.out.println(LocalDateTime.now()); |

Here are the range of date and time. if out of the range, a runtime exception would be thrown.

int year: 0-9999

int month: 1-12

int days: 1-31/28

int hours: 0-23

int minutes/second: 0-59

|  |
| --- |
| output:  2015-01-02  2015-01-02  2015-01-02  0015-01-31 |
| LocalDate d1 = LocalDate.of(2015, 1, 2);  System.out.println(d1);  LocalDate d2 = LocalDate.of(2015, 01, 02);  System.out.println(d2);  LocalDate d3 = LocalDate.of(2015, Month.JANUARY, 2);  System.out.println(d3);  **LocalDate d4 = LocalDate.of(15, 1,31);**  System.out.println(d4); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## manipulate date/time

|  |  |
| --- | --- |
| reference | example |
| LocalDate date | LocalDate date = LocalDate.of(2010,1,3);  date = date.plusDays(2);  date = date.plusWeeks(1);  date = date.plusMonths(1);  date = date.plusYears(5);  date = date.minusDays(2);  date = date.minusWeeks(1);  date = date.minusMonths(1);  date = date.minusYears(5); |
| LocalTime time | LocalTime time = LocalTime.of(4,12,10, 100);  time = time.plusNaos(1);  time = time.plusSeconds(40);  time = time.plusMinutes(10);  time = time.plusHours(1);  time = time.minusNaos(1);  time = time.minusSeconds(40);  time = time.minusMinutes(10);  time = time.minusHours(1); |
|  |  |
|  |  |

Keep in mind that objects of LocalDate, LocalDateTime, or LocalTime are immutable. Any manipulations should be reassigned to the original reference variables. Compare the two examples below

|  |  |
| --- | --- |
| output: |  |
| 2022-01-13  2022-01-13  Object is not updated. | LocalDate date = LocalDate.of(2022,1,13);  System.out.println(date);  **date.plusDays(3);**  **date.plusMonths(1);**  System.out.println(date); |
| 2022-01-13  2022-02-16 | LocalDate date = LocalDate.of(2022,1,13);  System.out.println(date);  **date=date.plusDays(3);**  **date=date.plusMonths(1);**  System.out.println(date); |

period

|  |  |
| --- | --- |
|  |  |
| 2020-01-02T12:00  2021-03-05T12:00 | static void test1(){  LocalDateTime d = LocalDateTime.of(2020,1,2,12,0,0);  System.out.println(d);  **Period p = Period.of(1,2,3);**  d = d.plus(p);  System.out.println(d);  } |
| 2020-01-02T12:00  2020-01-01T12:00 | static void test2(){  LocalDateTime d = LocalDateTime.of(2020,1,2,12,0,0);  System.out.println(d);  **Period p = Period.ofYears(2);**  **p = Period.ofMonths(1);**  **p = Period.ofDays(1);**  d = d.minus(p);  System.out.println(d);  } |
| 2020-01-02T12:00  2020-01-05T12:00  the same as:  Period p = Period.ofDays(1);  because the object Period object is immutable. | static void test3(){  LocalDateTime d = LocalDateTime.of(2020,1,2,12,0,0);  System.out.println(d);  **Period p = Period.ofYears(1).ofMonths(2).ofDays(3);**  d = d.plus(p);  System.out.println(d);  } |
| 2020-01-02T12:00  2021-02-04T12:00  p is 1year, 1month,1day | static void test4(){  LocalDateTime d = LocalDateTime.of(2020,1,2,12,0,0);  System.out.println(d);  Period p = Period.of(1,1,1);  p = p.plusDays(1);  **p.minusMonths(1);//useless**  d = d.plus(p);  System.out.println(d);  } |
| 2038 MARCH 30 | LocalDate date = LocalDate.of(2018, Month.APRIL, 30**).plusMonths(-1).plusYears(20);**  System.out.println(date.getYear() + " " + date.getMonth() + " "  + date.getDayOfMonth()); |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

## format date/time

|  |  |
| --- | --- |
|  |  |
| LocalDate date | date.getDayOfWeek();  date.getMonth();  date.getYear();  date.getDayOfYear(); |
|  |  |
|  |  |
|  |  |
|  |  |

The class **java.time.format.DateTimeFormatter** is used for formatting datetime. The instance of DateTimeFormatter could be passed as arguments of the format methods:

date.format(<formatter>)

datetime.format(<formatter>)

time.format(<formatter>)

Some formatters are predefined in class attributes of DateTimeFormatter.

|  |
| --- |
| LocalDate date = LocalDate.of(2010,4,4);  date.format(**DateTimeFormatter.ISO\_LOCAL\_DATE**); |
| LocalTime time = LocalTime(12,0,23);  time.format(**DateTimeFormatter.ISO\_LOCAL\_TIME**); |
| LocalDateTime datetime = LocalDateTime(2022,01,13,9,12,4);  datetime.format(**DateTimeFormatter.ISO\_LOCAL\_DATE\_TIME**) |

Formatter could be defined by combining **FormatStyle.SHORT** with DateTimeFormatter.ofLocalDate/.ofLocalTime/.ofLocalDateTime.

|  |
| --- |
| DateTimeFormatter **shortDate** = DateTimeFormatter.**ofLocalizedDate**(FormatStyle.SHORT);  LocalDateTime datetime = LocalDateTime.of(2022,01,13,9,12,4);  shortDate.format(datetime);  LocalDate date = LocalDate.of(2022,01,13);  shortDate.format(date); |
| DateTimeFormatter **shortTime** = DateTimeFormatter.**ofLocalizedTime**(FormatStyle.SHORT);  LocalDateTime datetime = LocalDateTime.of(2022,01,13,9,12,4);  shortTime.format(datetime);  LocalDateTime time = LocalTime.of(9,12,4);  shortTime.format(time); |
| DateTimeFormatter **shortDateTime** = DateTimeFormatter.**ofLocalizedDateTime**(FormatStyle.SHORT);  LocalDateTime datetime = LocalDateTime.of(2022,01,13,9,12,4);  shortDateTime.format(datetime); |

Moreover, some formatters could be customarily defined as the below:

customary format

|  |  |
| --- | --- |
| MMMM | month: January, Feburary,…December |
| MMM | month: Jan, Feb, …Dec |
| MM | month: 01,02,…,12 |
| M | month: 1,2,…,12 |
| dd | day |
| yyyy | year |
| mm | minute |

Note: "MMM" and "mm" represent month and minute, respectively. "MMM" or "mmmm" is wrong.

|  |
| --- |
| DateTimeFormatter f = DateTimeFormatter.ofPattern("hh::mm"); |
| DateTimeFormatter f = DateTimeFormatter.ofPattern("MM"); |

|  |
| --- |
| output:  2018-02-12  Feb 12, 2018  Feb 15, 2018 |
| DateTimeFormatter f = DateTimeFormatter.ofPattern("MMM dd, yyyy");  LocalDate d1 = LocalDate.of(2018,2,12);  System.out.println(d1);  LocalDate d2 = d1.plusDays(3);  System.out.println(f.format(d1));  System.out.println(d2.format(f)); |

|  |
| --- |
|  |
|  |

## parse date and time

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

# Others

## size

|  |  |  |
| --- | --- | --- |
| Data type | Methods |  |
| array | arr.length | size of memory used. length is class attribute rather than class method  that can't be changed once the array is declared in initialized. |
| ArrayList | list.size() | arraylist is dynamic. returns how many slots in the list. |
| String | str.length() | number of characters.  Maximum of string in Java is 0-2147483647 characters. That is capacity which is redundant property and can't be changed. Length of an empty string is zero. |
| StringBuilder | str.length() | number of characters. |
|  | str.capacity() | the current capacity of StringBUilder object. The capacity is the amount of storage available to insert new characters. That is set by content of object. default is 2^4=16. |
|  |  |  |
|  |  |  |

array.length(), array.size() is wrong.

|  |
| --- |
|  |
| int[] a1 = {1,2};  System.out.println(a1.length);//==2 |
| int[] a2 = new int[10];  a2[2]=2;  System.out.println(a2.length);//==10 |
|  |

here is the size of 2D array

|  |
| --- |
|  |
| int[][] a3 = {{1,2,3}, {1,2}};  System.out.println(a3.length);//==2  **System.out.println(a3[0].length);//==3**  **System.out.println(a3[1].length);//==2** |

The syntax list.length(), or list.capacity() is wrong.

|  |
| --- |
|  |
| ArrayList b1 = new ArrayList();  System.out.println(b1.size());//==0  ArrayList<String> b2 = new ArrayList<>(10);  System.out.println(b2.size());//==0 |

|  |
| --- |
|  |
|  |

## compare methods

CRUD: create, read(get), update(add, insert), delete

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Array | ArrayList | String | StringBuilder |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |