String: immutable sequence of characters and index starts from 0 the end index is mostly exclusive.  
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#startsWith-java.lang.String-

**String()**: Initializes a newly created String object so that it represents an empty character sequence.

**String(byte[] bytes)**: Constructs a new String by decoding the specified array of bytes using the platform's default charset.

**String(char[] value)**: Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

**String(char[] value, int offset, int count)**: Allocates a new String that contains characters from a subarray of the character array argument.

**String(int[] codePoints, int offset, int count)** : Allocates a new String that contains characters from a subarray of the Unicode code point array argument.

**String(String original)**: Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

**String(StringBuffer buffer)**: Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

**String(StringBuilder builder)**: Allocates a new string that contains the sequence of characters currently contained in the string builder argument.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Parameters | Return Type | Description | Examples |
| Concat | String | String | Concatenates the specified string to the end of this string | "cares".concat((1+2+3)+"s") = “cares6s” |
| Length | None | Int | Returns the length of this string | “animals”.length() == 7 |
| CharAt | Int | Char | Returns the char value at the specified index. An index ranges from 0 to length() - 1 | Throws StringIndexOutOfBound Exception when the index is not valid.  Ex :  String str = "animals"; str.charAt(str.length()); |
| indexOf | Char, but can also be an Int as Char as Int are interchangeable | Int | Returns the index within this string of the first occurrence of the specified character. If no such character occurs in this string, then -1 is returned | indexOf is case sensitive.  Ex :  String str = "bananas";  str.indexOf('A') == -1  str.indexOf(**""**) == 0  str.indexOf(**""**,4) == 4 |
| indexOf | (Char char, int fromIndex) | Int | Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index | indexOf is case sensitive  String str = **"bananas"**;  *assertTrue*(str.indexOf(**""**) == 0); *assertTrue*(str.indexOf(**""**,4) == 4); |
| lastIndexOf | Char or Int | Int | Returns the index within this string of the last occurrence of the specified character. In either case, if no such character occurs in this string, then -1 is returned | indexOf is case sensitive |
| lastIndexOf | (Char char, Int fromIndex) | Int | Returns the index within this string of the last occurrence of the specified character | indexOf is case sensitive  String str = "bananas";  str.lastIndexOf('a',2) ==1 |
| indexOf | String | Int | Same as above | Same as above |
| indexOf | String str, int fromIndex | Int | Same as above | Same as above |
| lastIndexOf | String | Int | Same as above | Same as above |
| lastIndexOf | String str, int fromIndex | Int | Same as above | Same as Above |

Item Found

Ex: **“bananas”.lastIndexOf(‘a’,2)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Index** |  |  |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| S | A | N | A | N | **A** | B |

**Start here**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Substring | Int beginIndex | String | The substring begins with the character at the specified index and extends to the end of this string | "unhappy".substring(2) returns "happy"  [IndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/IndexOutOfBoundsException.html) - if beginIndex is negative or larger than the length of this String object |
| Substring | Int begin, Int end | String | The substring begins at the specified beginIndex and extends to the character at index endIndex - 1 | "hamburger".substring(4, 8) returns "urge"  [IndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/IndexOutOfBoundsException.html) - if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex |

Ex: **“bananas”.substring(2,6);**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Start index |  |  |  | End Index |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| B | A | **N** | **A** | **N** | **A** | S |

**start here Stop Here**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| toLowerCase | None | String | Converts all of the characters in this String to lower case using the rules of the default locale |  |
| toUpperCase | None | String | Converts all of the characters in this String to upper case using the rules of the default locale |  |
| startsWith | String prefix | boolean | Tests if this string starts with the specified prefix . true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise. Note also that true will be returned if the argument is an empty string or is equal to this String object as determined by the [equals(Object)](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#equals-java.lang.Object-)method | *1) Is case sensitive 2) if string to compare is empty, returns true 3) if string to compare contains the entire string compared  then returns true.* |
| startsWith | String prefix, int toffset | boolean | true if the character sequence represented by the argument is a prefix of the substring of this object starting at index toffset; falseotherwise. The result is false if toffset is negative or greater than the length of this String object; otherwise the result is the same as the result of the expression: this.substring(toffset).startsWith(prefix) | String str = “bananas”  str.startsWith(**"as"**,5) == true  str.startsWith(**"as"**,5) == true |
| endsWith | String suffix | boolean | true if the character sequence represented by the argument is a suffix of the character sequence represented by this object; false otherwise. Note that the result will be true if the argument is the empty string or is equal to this String object as determined by the [equals(Object)](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#equals-java.lang.Object-)method | *1) Is case sensitive 2) if string to compare is empty, returns true 3) if string to compare contains the entire string compared  then returns true.*  String str = “bananas”;  str.endsWith(**"bananas"**) == true  String str = “bananas”;  str.endsWith(**""**) == true |
| Contains | Character sequence | Boolean | Returns true if and only if this string contains the specified sequence of char values | Is case Sensetive. Convenience method so you don’t have to write str.indexOf(otherString) != -1  String str = **"animals"**;  str.contains(**""**) == true; |
| Replace | CharacterSequence target, Character Sequence replacement | String | Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The replacement proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaa" will result in "ba" rather than "ab" | String str = **"bananas"**;  str.replace(**"an"**,**"su"**) = “**bsusuas**”  str.replace(**"ana"**,**"su"**) = “**bsunas**”  replaceFirst and replaceAll are used with regular expression, they only work on String parameters  str = **"bananas"**; str = str.replace(**"na"**,**"86"**); str = str.replace((85+1)+**""**,**"81"**);  = “ba8181s” |
| Trim | None | String | Returns a string whose value is this string, with any leading and trailing whitespace removed |  |

StringBuilder: A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to StringBuffer as it will be faster under most implementations

Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger.

Unless otherwise noted, passing a null argument to a constructor or method in this class will cause a [NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) to be thrown.

<https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html>

**StringBuilder()** : Constructs a string builder with no characters in it and an initial capacity of **16** characters.

**StringBuilder(CharSequence seq)** : Constructs a string builder that contains the same characters as the specified CharSequence.

**StringBuilder(int capacity):** Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

**StringBuilder(String str):** Constructs a string builder initialized to the contents of the specified string.

**Methods:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Append | append(boolean b) , append(char c), append(char[] str), append(char[] str, int offset, int len). append(CharSequence s), append(CharSequence s, int start, int end), append(double d), append(float f), append(int i), append(long lng), append(Object obj), append(String str), append(StringBuffer sb) | StringBuilder | Appends the specified string to this character sequence.  The characters of the String argument are appended, in order, increasing the length of this sequence by the length of the argument. If str is null, then the four characters "null" are appended. | *// sb.append(null); Compile error*  Notice you cannot append null to a StringBuilder but if you append a null object or a string, then the characters “null” are appended. |
| charAt | Int | Char | Returns the char value in this sequence at the specified index. The first char value is at index 0, the next at index 1, and so on, as in array indexing.  The index argument must be greater than or equal to 0, and less than the length of this sequence | StringBuilder strb = **new** StringBuilder(**"animals"**); *assertEquals*(strb.charAt(1),**'n'**); |
| indexOf | String | Int | Returns the index within this string of the first occurrence of the specified substring. | Note that IndexOf, last IndexOf in String Buffer only works with String and not Char. |
| indexOf | String str, int fromIndex | Int | Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.  If no such value of k exists, then -1 is returned |  |
| lastIndexOf | String str | Int | if the string argument occurs one or more times as a substring within this object, then the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned. | StringBuilder strb = **new** StringBuilder(**"animals"**);  *assertEquals*(strb.lastIndexOf(**""**,4),4); |
| lastIndexOf | String str, int fromIndex | Int | Same as above |  |
| Length | None | Int | Returns the length (character count) |  |
| Substring | Int | String | Returns a new String that contains a subsequence of characters currently contained in this character sequence. [StringIndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/StringIndexOutOfBoundsException.html) - if start is less than zero, or greater than the length of this object | Notice the String API throws IndexOutOfBound for substring while the Substring API for StringBuilder throws StringIndexOutOfBound exception. |
| Substring | Int start, int end | String | Returns a new String that contains a subsequence of characters currently contained in this sequence. The substring begins at the specified start and extends to the character at index end – 1. [StringIndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/StringIndexOutOfBoundsException.html) - if start or end are negative or greater than length(), or start is greater than end |  |
| Insert | insert(int offset, boolean b)  insert(int offset, char c)  insert(int offset, char[] str)  insert(int index, char[] str, int offset, int len)  insert(int dstOffset, CharSequence s)  insert(int dstOffset, CharSequence s, int start, int end)  insert(int offset, double d)  insert(int offset, float f)  insert(int offset, int i)  insert(int offset, long l)  insert(int offset, Object obj)  insert(int offset, String str) | StringBuilder | Inserts the string representation of the argument into this sequence.The overall effect is exactly as if the second argument were converted to a string by the method String.valueOf(argument), and the characters of that string were then inserted into this character sequence at the indicated offset. The offset argument must be greater than or equal to 0, and less than or equal to the length of this sequence.StringIndexOutOfBoundsException - if the offset is invalid | *// sb.insert(1,null); Compile error*  StringBuilder sb = **new** StringBuilder(**"animals"**); sb.insert(0,**"str"**,0,1); sb.insert(0,**new char**[]{**'B'**,**'U'**},0,1); *assertEquals*(sb.toString(),**"Bsanimals"**);  Notice for Char array and CharSequence you can specify additional two parameters which indicate,which part of the character sequence to start from and the length of the character sequence to insert into the StringBuilder. |
| Delete | Int start, int end | StringBuilder | Removes the characters in a substring of this sequence. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. If start is equal to end, no changes are made.StringIndexOutOfBoundsException - if start is negative, greater than length(), or greater than end | It is ok to have the end index greater than the length of the character sequence. |
| DeleteCharAt | Int index | StringBuilder | Removes the char at the specified position in this sequence. This sequence is shortened by one char. [StringIndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/StringIndexOutOfBoundsException.html) - if the index is negative or greater than or equal to length() |  |
| Replace | Int start, int end , String str | StringBuilder | Replaces the characters in a substring of this sequence with characters in the specified String. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. First the characters in the substring are removed and then the specified Stringis inserted at start | Remember Replace in String takes two char sequences or two characters while the replace in String builder uses a start and end index and the string to replace with.  StringBuilder strb = **new** StringBuilder(); strb.append(**"ABCDEFG"**); strb.replace(2,3,**"boomer"**); *assertTrue*(strb.equals(**"ABboomerEFG"**)); |
| Capacity | None | Int | Returns the current capacity. The capacity is the amount of storage available for newly inserted characters | StringBuilder strBuilder = **new** StringBuilder(**"animals"**); *assertTrue*(23 == strBuilder.capacity()); |
| ensureCapacity | Int minimumCapacity | Void | Ensures that the capacity is at least equal to the specified minimum. If the current capacity is less than the argument, then a new internal array is allocated with greater capacity. The new capacity is the larger of:   * The minimumCapacity argument. * Twice the old capacity, plus 2. * If the minimumCapacity argument is nonpositive, this method takes no action and simply returns The minimumCapacity argument. | StringBuilder sb = **new** StringBuilder(); sb.ensureCapacity(35);  Sb.capacity() == 35;   * Twice the old capacity (16), plus 2.   StringBuilder sb = **new** StringBuilder(); sb.ensureCapacity(24);  Sb.capacity() == 34; |
| trimToSize | None | Void | Attempts to reduce storage used for the character sequence. If the buffer is larger than necessary to hold its current sequence of characters, then it may be resized to become more space efficient | If the capacity is larger then what it needs, then it reduces the capacity to the length of the string builder |
| setLength | Void | Int | Sets the length of the character sequence. The sequence is changed to a new character sequence whose length is specified by the argument. For every nonnegative index *k* less than newLength, the character at index *k*in the new character sequence is the same as the character at index *k* in the old sequence if *k* is less than the length of the old character sequence |  |
| getChars | Int srcBegin, int srcEnd,char[] dest,int dstBegin | Void | Characters are copied from this sequence into the destination character array dst. The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1. The total number of characters to be copied is srcEnd-srcBegin. The characters are copied into the subarray of dst starting at index dstBegin and ending at index:  dstbegin + (srcEnd-srcBegin) - 1 |  |

**ArrayList:** Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

**ArrayList():**Constructs an empty list with an initial capacity of ten.

**ArrayList(Collection<? extends E> c)**

**ArrayList(int initialCapacity):** Constructs an empty list with the specified initial capacity.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Add | Element E | Boolean | Appends the specified element to the end of this list. Boolean always returns true | List strList = **new** ArrayList<>();  *assertTrue*(strList.add(**new** StringBuilder(**"boom"**))); *assertTrue*(strList.add(1.0f)); |
| Add | Int index, E Element | Void | Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices) | Throws IndexOutOfBound Exception if the index is invalid. |
| Remove | Int index | Element e | Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices) |  |
| Remove | Element e | Boolean | Removes the first occurrence of the specified element from this list, if it is present. If the list does not contain the element, it is unchanged |  |
| Clear | None | Void | Removes all of the elements from this list. The list will be empty after this call returns |  |
| addAll | Collection | Boolean | Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator. The behavior of this operation is undefined if the specified collection is modified while the operation is in progress | Returns true if this list changed as a result of the call |
| addAll | Int index, Collection c, | Boolean | Inserts all the elements in the specified collection into this list, starting at the specified position. Shifts the element currently at that position (if any) and any subsequent elements to the right (increases their indices) | Returns true if this list changed as a result of the call |
| removeAll | Collection c | Boolean | Removes from this list all of its elements that are contained in the specified collection |  |
| indexOf | Object o | Int | Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element |  |
| lastIndexOf | Object o | Int | Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element |  |
| Size | None | Int | Returns the number of elements in this list. |  |
| Contains | Object o | Boolean | Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that(o==null ? e==null : o.equals(e)) |  |
| Get | Int index | Element e | Returns the element at the specified position in this list. [IndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/IndexOutOfBoundsException.html) - if the index is out of range |  |
| subList | Int fromIndex, int toIndex | List<E> | Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive. (If fromIndex and toIndex are equal, the returned list is empty.) The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list, and vice-versa. The returned list supports all of the optional list operations | Remember the end index does not get included i.e the collection will only contain item from (from index to end index -1)  Any modifications to the backed list will in turn modify the new sublist  charArray.toString().equals(**"[A, B, C, D, E, F, G, H, I]"**)  List<Character> subListChar = charArray.subList(4,7);  charArray.set(5,**'T'**); *assertTrue*(subListChar.get(1).equals(**'T'**));  subListChar.clear(); *assertTrue*(charArray.toString().equals(**"[A, B, C, D, H, I]"**)); |
| Set | Int index, E newElement | E | The set method changes one of the elements of the ArrayList without changing the size. The E return type is the element that got replaced. |  |
| containsAll | Collection c | Boolean | Returns true if this list contains all of the elements of the specified collection. | List<Character> charList = returnCharArray(**"ABCDEFGH"**); List<Character> anotherCharList = returnCharArray(**"HGF"**); System.***out***.println(charList.**containsAll**(anotherCharList));  **Returns True**, no matter what is the order  Note that the contains method will also accept the collection of List, but will return false for below scenario.  i.e  List<Character> charList = returnCharArray(**"ABCDEFGH"**); List<Character> anotherCharList = returnCharArray(**"HGF"**); System.***out***.println(charList.**contains**(anotherCharList));  Will **return false** |
| toArray | None | Object[] | Returns an array containing all of the elements in this list in proper sequence (from first to last element)  The returned array will be "safe" in that no references to it are maintained by this list. (In other words, this method must allocate a new array even if this list is backed by an array). The caller is thus free to modify the returned array.  This method acts as bridge between array-based and collection-based APIs |  |
| Size | None | Int | Returns the number of elements in this list |  |

**Local Date:**

A date without a time-zone in the ISO-8601 calendar system, such as 2007-12-03  
LocalDate is an **immutable date-time object** that represents a date, often viewed as year-month-day. Other date fields, such as day-of-year, day-of-week and week-of-year, can also be accessed. For example, the value "2nd October 2007" can be stored in a LocalDate.

This class does not store or represent a time or time-zone. Instead, it is a description of the date, as used for birthdays. It cannot represent an instant on the time-line without additional information such as an offset or time-zone

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Static** now | None | LocalDate | Obtains the current date from the system clock in the default time-zone.  This will query the [system clock](https://docs.oracle.com/javase/8/docs/api/java/time/Clock.html#systemDefaultZone--) in the default time-zone to obtain the current date | Format of the LocalDate: YYYY-MM-DD.  Just keep in mind that LocalDate now method can also include a Clock or ZoneId as a parameter |
| **Static**  Of | of(int year,Month month,int day)  of(int year, int month, int day) | LocalDate | The day must be valid for the year and month | LocalDate localDate = LocalDate.*of*(2017,02,31);  Throws : DateTimeException.**class** |
| **Static**  ofYearDay | Int year, int dayOfYear | LocalDate | Obtains an instance of LocalDate from a year and day-of-year.  This returns a LocalDate with the specified year and day-of-year. The day-of-year must be valid for the year, otherwise an exception will be thrown | dayOfYear - the day-of-year to represent, from 1 to 366 |
| **Static**  From | TemporalAccessor temporalAccessor (LocalDate, LocalDateTime, LocalTime) | LocalDate |  | LocalDate fromLocalDateTime = LocalDate.*from*(LocalDateTime.*now*());   * 2018-03-31 |
| **Static**  Parse | Parse(CharSequence charsequence)  Parse(CharSequence charSeq,DateTimeFormatter dtf) | LocalDate | Obtains an instance of LocalDate from a text string such as 2007-12-03.  The string must represent a valid date and is parsed using [DateTimeFormatter.ISO\_LOCAL\_DATE](https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html#ISO_LOCAL_DATE) | LocalDate parsedLocalDate = localDate.*parse*(**"2007-12-03"**)  LocalDate parsedLocalDate = LocalDate.*parse*(**"2017-01-31T03:31"**)  Throws DateTimeParseException.class  LocalDate ld = LocalDate.*parse*(**"2018-03-31T00:00:00"**,DateTimeFormatter.***ISO\_DATE\_TIME***);  Prints : 2018-03-31  LocalDate date = LocalDate.now();  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");  String text = date.format(formatter);  LocalDate parsedDate = LocalDate.parse(text, formatter); |
| Get | Get(TemporalField temporalFiled) | Int | Gets the value of the specified field from this date as an int | LocalDate fromLocalDateTime = LocalDate.*parse*(**"2018-03-31"**); **int** generalLong = fromLocalDateTime.get(ChronoField.***MONTH\_OF\_YEAR***); *assertTrue*(generalLong == 3); **int** generalDay = fromLocalDateTime.get(ChronoField.***DAY\_OF\_MONTH***); *assertTrue*(generalDay == 31);  There is a long version of this exact same method which returns a long  long getLong([TemporalField](https://docs.oracle.com/javase/8/docs/api/java/time/temporal/TemporalField.html) field) |
| getYear | None | Int | This method returns the primitive int value for the year |  |
| getMonthValue | None | Int | This method returns the month as an int from 1 to 12. |  |
| getMonth | None | Month | Gets the month-of-year field using the Month enum | LocalDate fromLocalDateTime = LocalDate.*parse*(**"2018-03-31"**); Month month = fromLocalDateTime.getMonth(); *assertTrue*(month == Month.***MARCH***); |
| getDayOfMonth | None | Int | gets the day-of-month field.  This method returns the primitive int value for the day-of-month |  |
| getDayOfWeek | None | DayOfWeek | Gets the day-of-week field, which is an enum DayOfWeek.  This method returns the enum [DayOfWeek](https://docs.oracle.com/javase/8/docs/api/java/time/DayOfWeek.html) for the day-of-week | LocalDate fromLocalDateTime = LocalDate.*parse*(**"2018-03-31"**); DayOfWeek dayOfWeek = fromLocalDateTime.getDayOfWeek(); *assertTrue*(dayOfWeek == DayOfWeek.***SATURDAY***); |
| Plus | Long amountToAdd, TemporalUnit unit | LocalDate | Returns a copy of this date with the specified amount added | LocalDate fromLocalDateTime = LocalDate.*parse*(**"2018-01-31"**); plusLocalDate = fromLocalDateTime.plus(1,ChronoUnit.***MONTHS***); *assertEquals*(plusLocalDate.toString(),**"2018-02-28"**); |
| Plus | TemporalAmount amountToAdd | LocalDate | This returns a LocalDate, based on this one, with the specified amount added. The amount is typically [Period](https://docs.oracle.com/javase/8/docs/api/java/time/Period.html) but may be any other type implementing the [TemporalAmount](https://docs.oracle.com/javase/8/docs/api/java/time/temporal/TemporalAmount.html) interface | LocalDate fromLocalDateTime = LocalDate.*parse*(**"2018-03-31"**); LocalDate plusLocalDate = fromLocalDateTime.plus(Period.*of*(1,1,1)); *assertEquals*(plusLocalDate,**"2019-05-01"**); |
| plusYears | Long years | LocalDate | This method adds the specified amount to the years field in three steps:   1. Add the input years to the year field 2. Check if the resulting date would be invalid 3. Adjust the day-of-month to the last valid day if necessary | For example, 2008-02-29 (leap year) plus one year would result in the invalid date 2009-02-29 (standard year). Instead of returning an invalid result, the last valid day of the month, 2009-02-28, is selected instead |
| plusMonths | Long months | LocalDate | Returns a copy of this LocalDate with the specified number of months added |  |
| plusWeeks | Long weeksToAdd | LocalDate | Returns a copy of this LocalDate with the specified number of weeks added. |  |
| plusDays | Long days | LocalDate | Returns a copy of this LocalDate with the specified number of days added |  |
| Minus | Minus(TemporalAmount temporalAmount)  Minus(long amountToAdd, TemporalUnit temporalUnit)  MinusYears(long years),MinusMonths(long months),MinusWeeks(long weeks), minusDays(long days) | LocalDate | Same as above |  |
| Format | DateTimeFormatter dtf | String | Formats this date using the specified formatter.  This date will be passed to the formatter to produce a string | *assertEquals*((fromLocalDateTime.format(DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***FULL***))).toString(),**"Saturday, March 31, 2018"**); *//LONG assertEquals*((fromLocalDateTime.format(DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***LONG***))).toString(),**"March 31, 2018"**);  *//MEDIUM assertEquals*((fromLocalDateTime.format(DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***MEDIUM***))).toString(),**"Mar 31, 2018"**);  *//SHORT assertEquals*((fromLocalDateTime.format(DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***SHORT***))).toString(),**"3/31/18"**); |
| atTime | LocalTime localTime | LocalDateTime | Combines this date with a time to create a LocalDateTime. |  |
| AtTime | Int hour, int minute  Int hour, int minute, int seconds  Int hour , int minute, int seconds, int nanoseconds | LocalDateTIme | Combines this date with a time to create a LocalDateTime. | LocalDateTime localDateTime = localDate.atTime(03,31); *assertTrue*(localDateTime.toString().equals(**"2017-01-31T03:31"**)); |
| atStartOfDay | None | LocalDateTime | Combines this date with the time of midnight to create a LocalDateTime at the start of this date.  This returns a LocalDateTime formed from this date at the time of midnight, 00:00, at the start of this date |  |
| isAfter | ChronoLocalDate localDate | Boolean | Checks if this date is after the specified date.  This checks to see if this date represents a point on the local time-line after the other date |  |
| isBefore | ChronoLocalDate localDate | Boolean | Checks if this date is before the specified date. |  |

**Local Time:**

A time without a time-zone in the ISO-8601 calendar system, such as 10:15:30.

LocalTime is an immutable date-time object that represents a time, often viewed as hour-minute-second. Time is represented to nanosecond precision. For example, the value "13:45.30.123456789" can be stored in a LocalTime

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| atDate | LocalDate localDate | LocalDateTime | Combines this time with a date to create a LocalDateTime. |  |
| **Static** Now | None | LocalTime | Obtains the current time from the system clock in the default time-zon | Outputs with format as follows: HH:MM:SS:NNN |
| **Static**  Of | Int hour, int minute | LocalTime | Obtains an instance of LocalTime from an hour and minute. |  |
| **Static**  Of | Int hour, int minute, int second  Int hour, int minute, int second, int nanosecond | LocalTime |  |  |
| **Format** | DateTimeFormatter | String |  | LocalTime toFormatLocalTime = LocalTime.*of*(11,25); String formattedTime = toFormatLocalTime.format(DateTimeFormatter.***ISO\_TIME***); *assertTrue*(formattedTime.equals(**"11:25:00"**)); |
| **Static**  Parse | CharSequence | LocalTIme |  | LocalTime parsedLocalTime = LocalTime.*parse*(**new** StringBuilder(**"12:46"**),DateTimeFormatter.*ofPattern*(**"HH:mm"**)); *assertTrue*(parsedLocalTime.equals(**"12:46"**)); |
| **Static**  Parse | CharSequence,DateTimeFormatter | LocalTime |  |  |
| Plus/minus | Long, TemporalUnit(ChronoUnit) | LocalTIme | Variations : plus(TemporalAmount), plusHours(long), plusMinutes(long), plusNanos(long),plusSeconds(long) |  |
| atDate | LocalDate | LocalDateTime |  |  |
| Get | TemporalField | Return int | Variations: getHour(), getLong(TemporalField) : returns long, getMinute(), getSecond() |  |

**LocalDateTime:** A date-time without a time-zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30 **.** It is an immutable date-time object that represents a date-time, often viewed as year-month-day-hour-minute-second

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| From | TemporalAccessor | LocalDateTime | Variations: getHour(), getLong(TemporalField) : returns long, getMinute(), getSecond() |  |
| **Static**  Now | None | LocalDateTime | Variations : now(ZoneId) , now(Clock) | Prints until the nano seconds  System.***out***.println(LocalDateTime.*now*());  2018-04-20T08:28:48.881 |
| **Static**  Of | Int year , int month/Month,int day, int hour,int min | LocalDateTime | Variations : of(int year,int/Month month,int day, int hour, int min, int sec) ,  Of(int year, int/Month month, int day, int hour, int min, int sec,int nanosec)  Of(LocalDate, LocalTime) |  |
| Format | DateTimeFormatter | String | Notice that the format method can take DateTimeFormatter.LocalDate or DateTimeFormatter.LocalTime and this method would still work without any exceptions |  |
| Get | TemporalField | Int | Variations:  int getDayOfYear()  int getHour()  long getLong(TemporalField field)  int getMinute()  Month getMonth()  int getMonthValue()  int getNano()  int getSecond()  int getYear()  int getDayOfMonth()  DayOfWeek getDayOfWeek() | **long** nanoSeconds = ldt.getLong(ChronoField.***NANO\_OF\_DAY***);  **long** milis = ldt.getLong(ChronoField.***MILLI\_OF\_SECOND***); System.***out***.println(milis);  Prints : 511 |
| Plus / Minus | Temporal Amount | LocalDateTime | Variations : plus(long amountToAdd, TemporalUnit unit)  LocalDateTime plus(TemporalAmount amountToAdd)  LocalDateTime plusDays(long days)  LocalDateTime plusHours(long hours)  LocalDateTime plusMinutes(long minutes)  LocalDateTime plusMonths(long months)  LocalDateTime plusNanos(long nanos)  LocalDateTime plusSeconds(long seconds)  LocalDateTime plusWeeks(long weeks)  LocalDateTime plusYears(long years) | TemporalUnit is similar to TemporalField . |
| **Static** parse | CharSequence | LocalDateTime | Variation: CharSequence, DateTimeFormatter |  |
|  |  |  |  |  |

**Period:** A date-based amount of time in the ISO-8601 calendar system, such as '2 years, 3 months and 4 days'

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| addTo | Temporal | Temporal |  | Period period = Period.*of*(2018,03,06); LocalDate localDate = LocalDate.*of*(2018, Month.***JANUARY***, 01); localDate = (LocalDate) period.addTo(localDate); *assertEquals*(**"4036-04-07"** ,localDate.toString()); |
| **Static**  Between | LocalDate start, LocalDate end | Period |  | LocalDate ld0 = LocalDate.*of*(2018,Month.***MAY***,01); LocalDate ld01 = LocalDate.*of*(2018,Month.***MAY***,02); *assertEquals*(**"P1D"**,Period.*between*(ld0,ld01).toString()); |
| **Static**  From | TemporalAmount | Period |  |  |
| Get | TemporalUnit | Long | Variations : int getDays(), int getMonths(), List<TemporalUnit> getUnits(),int getYears() | Period period = Period.*of*(2018,03,06);  *assertEquals*(period.get(ChronoUnit.***DAYS***),6) |
| Plus/MInus | TemporalAmount | Period | Variations : plusDays(long) , plusMonths(long), plusYears(long),minusDays(long), minusMonths(long),minusYears(long) |  |
| **Static**  Of | Int year, int month, int day | Period | Variations: ofDays(int days), ofMonths(int months), ofWeeks(int weeks) , ofYears(int years) |  |
| **Static**  Parse | CharSequence | Period |  | Obtains a Period from a text string such as PnYnMnD. |
| Normalized | No param | Period |  | Period unNormalizedPeriod = Period.*of*(2018,25,49); *assertEquals*(**"P2020Y1M49D"**,unNormalizedPeriod.normalized().toString());  Period unNormalizedPeriod1 = Period.*of*(1,-25,49); *assertEquals*(**"P-1Y-1M49D"**,unNormalizedPeriod1.normalized().toString());  *// Testing the ofDays, ofMonths, ofWeeks, ofYears method in Period* Period ofDaysPeriod = Period.*ofMonths*(51); *assertEquals*(**"P4Y3M"**,ofDaysPeriod.normalized().toString()); |

**DateTimeFormatter:** Formatter for printing and parsing date-time objects

**BASIC\_ISO\_DATE** - '20111203'.

**ISO\_DATE** - parses a date with the offset if available, such as '2011-12-03' or '2011-12-03+01:00'.

**ISO\_DATE\_TIME** - parses a date-time with the offset and zone if available, such as '2011-12-03T10:15:30', '2011-12-03T10:15:30+01:00' or '2011-12-03T10:15:30+01:00[Europe/Paris]'.

**ISO\_INSTANT** - parses an instant in UTC, such as '2011-12-03T10:15:30Z'.

**ISO\_LOCAL\_DATE** - parses a date without an offset, such as '2011-12-03'.

**ISO\_LOCAL\_DATE\_TIME** - parses a date-time without an offset, such as '2011-12-03T10:15:30'.

**ISO\_LOCAL\_TIME** - parses a time without an offset, such as '10:15' or '10:15:30'.

**ISO\_OFFSET\_DATE** - parses a date with an offset, such as '2011-12-03+01:00'.

**ISO\_OFFSET\_DATE\_TIME** - parses a date-time with an offset, such as '2011-12-03T10:15:30+01:00'.

**ISO\_OFFSET\_TIME** - parses a time with an offset, such as '10:15+01:00' or '10:15:30+01:00'.

**ISO\_ORDINAL\_DATE** - parses the ordinal date without an offset, such as '2012-337'.

**ISO\_TIME** - parses a time, with the offset if available, such as '10:15', '10:15:30' or '10:15:30+01:00'.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format | TemporalAccessor | String | Variations: void (TemporalAccessor, Appendable appendable) | Format() method works with LocalDateTime as a parameter when Date or Time only are passed :  LocalDateTime ldt = LocalDateTime.*of*(2018,Month.***MAY***,01,01,01); String formattedDate = DateTimeFormatter.***BASIC\_ISO\_DATE***.format(ldt); *assertEquals*(**"20180501"**,formattedDate);  formattedDate = DateTimeFormatter.***ISO\_TIME***.format(ldt); *assertEquals*(**"01:01:00"**,formattedDate); |
| **Static**  Of..() | FormatStyle : full , long , medium , short | DateTImeFormatter | Variations : ofLocalizedDate(FormatStyle), ofLocalizedTime(FormatStyle), ofLocalizedDateTime(FormatStyle) , ofLocalizedDateTime(FormatStyle,FormatStyle) |  |
| **Static**  ofPattern() | String pattern | DateTimeFormatter |  |  |
| Parse | CharSequence | TemporalAccessor | Variations : parse(charSequence, ParsePosition) |  |

DateTimeFormatter behavior with various TemporalAccessors

|  |  |  |  |
| --- | --- | --- | --- |
| DTF f = DTF.Formatter.\_\_\_\_(FormatterShort); | Calling f.format(localDate) | Calling f.format(localDateTime) | Calling f.format(localTime) |
| ofLocalizedDate() | Legal shows whole object | Legal shows the Date object | Throws runtimeExceptino |
| ofLocalizedDateTime() | Throws Runtime Exception | Legal shows whole object | Throws Runtime Exception |
| ofLocalizedTime() | Throws Runtime Exception | Legal shows just the time part | Legal shows the whole object |

TemporalAccessors behavior with various DateTimeFormatter

|  |  |  |  |
| --- | --- | --- | --- |
| Type of Temporal Accessor  TemporalAccessor.format(DTF) | Calling DateTimeFormatter.ofLocalizedDate() | Calling DateTimeFormatter.ofLocalizedDateTime() | Calling DateTimeFormatter.ofLocalizedTime() |
| LocalDate.format(..) | Legal shows the whole object | Throws Runtime Exception | Throws Runtime Exception |
| LocalDateTime.format(..) | Legal shows the Date object | Legal shows whole object | Legal shows the time object |
| LocalTime.format(..) | Throws Runtime Exception | Throws Runtime Exception | Legal shows the whole object |