Real-Time Test – Selected Methods Reference

ArrayList (of Java Class Library)

ArrayList()	Creates a ArrayList object with no elements
ArrayList(Collection< ? extends E> c)	Creates a ArrayList object with the elements in the collection c Throws NullPointerException if the specified collection is null
int size()	Returns the number of elements in the list
boolean isEmpty()	Returns true if the size is 0
boolean add (E e)	Adds the element e to the end of the list
void add(int index, E element)	Adds the element e at the specified index. Throws IndexOutOfBoundsException if index is out of range
E remove(int index)	Retrieves and deletes the element at the specified index. Throws IndexOutOfBoundException if index is out of range
E set(int index, E element)	Replaces the element at index with the specified element. Returns the element previously at index. Throws IndexOutOfBoundException if index is out of range
E get(int index)	Retrieves the element at the specified index. Throws IndexOutOfBoundException if index is out of range
boolean contains(Object o)	Determines whether object o is in the list
int indexOf(Object o)	Returns the index where o first occurs in the list. (Returns -1 if object o is not found)
int lastIndexOf(Object o)	Returns the index where o last occurs in the list (Returns -1 if object o is not found)
boolean remove(Object o)	Removes the first occurrence of element o from the list. Returns true if the list has the specified element, false otherwise
boolean addAll (Collection<? extends E> c)	Adds each element from the collection c to the end of the ArrayList
boolean removeAll (Collection<?> c)	Deletes any element that is also in c
boolean retainAll (Collection<?> c)	Retains only the elements that are also in collection c
void clear()	Deletes all of the elements from the list

Scanner

Scanner(InputStream)	Can be used with System.in (System.in is a BufferedInputStream object)
Scanner(File)	Create a Scanner object to read from a text file Throws FileNotFoundException
nextLine()	Can throw various unchecked exceptions
nextInt()	Does not consume delimiter character after the number
nextDouble()	
nextBoolean()	
hasNext()	Returns true if the scanner has another token
hasNextLine()	
hasNextInt()	
hasNextDouble()	
hasNextBoolean()	
close()	

BufferedReader

`BufferedReader(Reader)`	`BufferedReader in = new BufferedReader(`
	`new FileReader("sample.txt"));`
	Can throw `FileNotFoundException`

`readLine()`	Reads and returns the next line of text (as a `String`) Returns null if end of file has been reached Can throw `IOEXception`
`read()`	Reads the next character and returns its numeric value Returns -1 if the end of file has been reached Can throw `IOEXception`
`close()`	Can throw `IOEXception`

StringTokenizer

`StringTokenizer(String s)`	Creates a tokenizer for string s using *whitespace* as delimiters
`StringTokenizer(String s, String delimiters)`	Creates a tokenizer for string s using *characters in the second parameter* as delimiters
`boolean hasMoreTokens()`	Returns true if there are remaining tokens, false otherwise
`int countTokens()`	Returns the number of remaining tokens - so the return value changes as tokens are removed from the `StringTokenizer` object
`String nextToken()`	Returns the next token - throws `NoSuchElementException` if there are no more tokens
`String nextToken( String delimiters)`	Returns the next token using the *characters in the parameter as delimiters* - throws `NoSuchElementException`

Converting String tokens into other data types

`Integer.parseInt(String s)`	Returns the int value that is represented by the string s. Throws a NumberFormatException (unchecked) if the String argument cannot be converted to an int value.
`Double.parseDouble(String s)`	
`Boolean.parseBoolean(String s)`	

The split method of String class

`String [] split(String regex)`	Takes a string argument which is treated as a regular expression `s.split(''12'')` $\Longrightarrow$ delimiter is "12" `s.split(''[12]'')` $\Longrightarrow$ delimiters are "1" or "2" `s.split(''\\s")` $\Longrightarrow$ delimiters are whitespace characters

printf

Format specifier (to specify how a data item is to be displayed):

```
% [flags] [width] [.precision] conversion-character
```

Flags:

`-`	left-justify (default is to right-justify)
`+`	output a plus (+) or minus (-) sign for a numerical value
`0`	forces numerical values to be zero-padded (default is blank padding)
`,`	comma grouping separator (for numbers > 1000)
	space will display a minus sign if the number is negative or a space if it is positive

Conversion-Characters:

d	decimal integer [byte, short, int, long]
f	floating-point number [float, double]
c	character. Capital C will uppercase the character
s	String. Capital S will uppercase all the letters in the string

Creating Formatted Strings

e.g. `String totalString = String.format("Total: %-10.2f", total);`

The a format string and argument list is identical to its use in the `printf` method

■