Joanna Klukowska
joannakl@cs.nyu.edu

String Class

`char charAt(int index)`
Returns the char value at the specified index.

`int compareTo(String anotherString)`
Compares two strings lexicographically.

`int compareToIgnoreCase(String str)`
Compares two strings lexicographically, ignoring case differences.

`boolean contains(CharSequence s)`
Returns true if and only if this string contains the specified sequence of char values.

`boolean equals(Object anObject)`
Compares this string to the specified object.

`boolean equalsIgnoreCase(String anotherString)`
Compares this String to another String, ignoring case considerations.

`static String format(Locale l, String format, Object... args)`
Returns a formatted string using the specified locale, format string, and arguments.

`static String format(String format, Object... args)`
Returns a formatted string using the specified format string and arguments.

`boolean isEmpty()`
Returns true if, and only if, length() is 0.

`int length()`
Returns the length of this string.

`String replace(char oldChar, char newChar)`
Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

`String replace(CharSequence target, CharSequence replacement)`
Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.

`String[] split(String regex)`
Splits this string around matches of the given regular expression.

`String[] split(String regex, int limit)`
Splits this string around matches of the given regular expression.

`String substring(int beginIndex)`
Returns a new string that is a substring of this string.

`String substring(int beginIndex, int endIndex)`
Returns a new string that is a substring of this string.

`char[] toCharArray()`
Converts this string to a new character array.

`String toLowerCase()`
Converts all of the characters in this String to lower case using the rules of the default locale.

`String toString()`
This object (which is already a string!) is itself returned.

`String toUpperCase()`
Converts all of the characters in this String to upper case using the rules of the default locale.

`String toUpperCase(Locale locale)`
Converts all of the characters in this String to upper case using the rules of the given Locale.

Math Class

`static double abs(double a)`
Returns the absolute value of a double value.

`static int abs(int a)`
Returns the absolute value of an int value.

`static double ceil(double a)`
Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.

`static double exp(double a)`
Returns Euler's number e raised to the power of a double value.

`static double floor(double a)`
Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.

`static double log(double a)`
Returns the natural logarithm (base e) of a double value.

`static double log10(double a)`
Returns the base 10 logarithm of a double value.

`static double max(double a, double b)`
Returns the greater of two double values.

`static int max(int a, int b)`
Returns the greater of two int values.

`static double min(double a, double b)`
Returns the smaller of two double values.

`static int min(int a, int b)`
Returns the smaller of two int values.

`static double pow(double a, double b)`
Returns the value of the first argument raised to the power of the second argument.

`static double random()`
Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

`static double sqrt(double a)`
Returns the correctly rounded positive square root of a double value.

Integer Class

`static int MAX_VALUE`
A constant holding the maximum value an int can have, $2^{31}-1$.

`static int MIN_VALUE`
A constant holding the minimum value an int can have, -2^{31}.

`static int compare(int x, int y)`
Compares two int values numerically.

`int compareTo(Integer anotherInteger)`
Compares two Integer objects numerically.

`double doubleValue()`
Returns the value of this Integer as a double.

`boolean equals(Object obj)`
Compares this object to the specified object.

`static int parseInt(String s)`
Parses the string argument as a signed decimal integer.

`static int parseInt(String s, int radix)`
Parses the string argument as a signed integer in the radix specified by the second argument.

`String toString()`
Returns a String object representing this Integer's value.

`static String toString(int i)`
Returns a String object representing the specified integer.

`static Integer valueOf(int i)`
Returns an Integer instance representing the specified int value.

`static Integer valueOf(String s)`
Returns an Integer object holding the value of the specified String.

Double Class

`static double MAX_VALUE`
A constant holding the largest positive finite value of type double.

`static double MIN_VALUE`
A constant holding the smallest positive nonzero value of type double.

static int compare(double d1, double d2)
Compares the two specified double values.

int compareTo(Double anotherDouble)
Compares two Double objects numerically.

boolean equals(Object obj)
Compares this object against the specified object.

static double parseDouble(String s)
Returns a new double initialized to the value represented by the specified String, as performed by the valueOf method of class Double.

String toString()
Returns a string representation of this Double object.

static String toString(double d)
Returns a string representation of the double argument.

static Double valueOf(String s)
Returns a Double object holding the double value represented by the argument string s.

Character Class

`static int compare(char x, char y)`
Compares two char values numerically.

`int compareTo(Character otherCharacter)`
Compares two Character objects numerically.

`boolean equals(Object obj)`
Compares this object against the specified object.

`static boolean isDigit(char ch)`
Determines if the specified character is a digit.

`static boolean isLetter(char ch)`
Determines if the specified character is a letter.

`static boolean isLowerCase(char ch)`

Joanna Klukowska
joannakl@cs.nyu.edu

Determines if the specified character is a lowercase character.

`static boolean isSpaceChar(char ch)`

Determines if the specified character is a Unicode space character.

`static boolean isUpperCase(char ch)`

Determines if the specified character is an uppercase character.

`static boolean isWhitespace(char ch)`

Determines if the specified character is white space according to Java.

`static char toLowerCase(char ch)`

Converts the character argument to lowercase using case mapping information from the UnicodeData file.

`static char toUpperCase(char ch)`

Converts the character argument to uppercase using case mapping information from the UnicodeData file.

`Random` Class

`Random()`

Creates a new random number generator.

`Random(long seed)`

Creates a new random number generator using a single long seed.

`boolean nextBoolean()`

Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.

`double nextDouble()`

Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.

`float nextFloat()`

Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.

`int nextInt()`

Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.

`int nextInt(int n)`

Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

`long nextLong()`

Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.

`Arrays` Class

`static void sort(Object[] a)`

Sorts the specified array of objects into ascending order, according to the natural ordering (using compareTo()) of its elements.

`ArrayList` Class

`E` stands for the type of the elements in the ArrayList object.

`boolean add(E e)`

Appends the specified element to the end of this list.

`void add(int index, E element)`

Inserts the specified element at the specified position in this list.

`E get(int index)`

Returns the element at the specified position in this list.

`boolean isEmpty()`

Returns true if this list contains no elements.

`E remove(int index)`

Removes the element at the specified position in this list.

`protected void removeRange(int fromIndex, int toIndex)`

Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

`E set(int index, E element)`

Replaces the element at the specified position in this list with the specified element.

`int size()`

Returns the number of elements in this list.

`Scanner` Class

`Scanner(File source)`

Constructs a new Scanner that produces values scanned from the specified file.

`boolean hasNext()`

Returns true if this scanner has another token in its input.

`String next()`

Finds and returns the next complete token from this scanner.

`double nextDouble()`

Scans the next token of the input as a double.

`int nextInt()`

Scans the next token of the input as an int.

`String nextLine()`

Advances this scanner past the current line and returns the input that was skipped.

`PrintWriter` Class

`PrintWriter(File file)`

Creates a new PrintWriter, without automatic line flushing, with the specified file.

`PrintWriter(String fileName)`

Creates a new PrintWriter, without automatic line flushing, with the specified file name.

`void close()`

Closes the stream and releases any system resources associated with it.

`void print(TYPE b)`

Prints a value of specified TYPE.

`PrintWriter printf(String format, Object... args)`

A convenience method to write a formatted string to this writer using the specified format string and arguments.

`void println(TYPE x)`

Prints a value of specified TYPE and then terminates the line.

`void write(char[] buf)`

Writes an array of characters.

`File` Class

`File(String pathname)`

Creates a new File instance by converting the given pathname string into an abstract pathname.

`boolean canRead()`

Tests whether the application can read the file denoted by this abstract pathname.

`boolean canWrite()`

Tests whether the application can modify the file denoted by this abstract pathname.

`boolean exists()`

Tests whether the file or directory denoted by this abstract pathname exists.

`boolean isDirectory()`

Tests whether the file denoted by this abstract pathname is a directory.

`boolean isFile()`

Tests whether the file denoted by this abstract pathname is a normal file.

Java Keywords

`abstract`	`final`	`return`
`assert`	`finally`	`short`
`boolean`	`float`	`static`
`break`	`for`	`strictfp`
`byte`	`if`	`super`
`case`	`implements`	`switch`
`catch`	`import`	`synchronized`
`char`	`instanceof`	`this`
`class`	`int`	`throw`
`const`	`interface`	`throws`
`continue`	`long`	`transient`
`default`	`native`	`try`
`do`	`new`	`void`
`double`	`package`	`volatile`
`else`	`private`	`while`
`enum`	`protected`	
`extends`	`public`	

Operator Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
relational	`< > <= >= instanceof`
equality	`== !=`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %=`