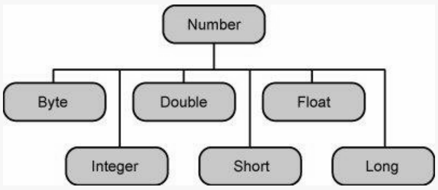
// import java.lang.Number;



Number Methods

Integer a = 20;	
a.byteValue(); a.shortValue(); a.intValue(); a.longValue(); a.floatValue(); a.doubleValue()	The method converts the value of the Number Object that invokes the method to the primitive data type that is returned from the method.
a.compareTopublic int compareTo(NumberSubClass same_type)	Compares this Number object to the argument.
a. equals(Object o)	Determines whether this number object is equal to the argument.
Number.valueOf(int i/String s/ String s, int radix)	The valueOf method returns the Integer holding the value of the argument passed.
a. toString()	string representation
Math.abs()	absolute value
Math.ceil()	The method ceil gives the smallest integer that is greater than or equal to the argument.
Math.rint()	Returns the integer that is closest in value to the argument. Returned as a double.
Math.round()	Returns the closest long or int, as indicated by the method's return type to the argument.
Math.min(a,b)	Returns the smaller of the two arguments.
Math.max(a,b)	Returns the larger of the two arguments.
Math.exp()	Returns the base of the natural logarithms, e, to the power of the argument.
Math.log()	Returns the natural logarithm of the argument.

Number Methods (cont)

Math.pow(a,b)	Returns the value of the first argument raised to the power of the second argument.
Math.sqrt()	Returns the square root of the argument.
Math.sin()	Returns the sine of the specified double value.
Math.cos(); Math.tan(); Math.asin(); Math.acos(); Math.atan(); Math.atan2();	
Math.toDegree()	Converts the argument to degrees.
Math.toRadians()	Converts the argument to radians.
Math.random()	Returns a random number.

StringBuilder

`StringBuilder sb = new StringBuilder(`*`capacity or string`*`);`	creat a StringBuilder
`sb.append(obj);`	append string representation of obj to this SB
`sb.capacity();`	return the current capacity
`sb.charAt(index);`	Returns the char value in this sequence at the specified index.
`sb.delete(start,end_exclusive);`	Removes the characters in a substring of this sequence.
`sb.deleteCharAt(index);`	Removes the char at the specified position in this sequence.
`getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin);`	Characters are copied from this sequence into the destination character array dst.
`sb.indexOf(string,`*`from`*`);`	Returns the index within this string of the first occurrence of the specified substring.
`sb.insert(index,obj)`	Inserts the string representation of the boolean argument into this sequence.
`sb.replace(int start, int end, String str);`	Replaces the characters in a substring of this sequence with characters in the specified String.

StringBuilder (cont)

`sb.reverse()`	Causes this character sequence to be replaced by the reverse of the sequence.
`sb.setChart(int index, char ch);`	The character at the specified index is set to ch.
`sb.subSequence(int start, int end)`	Returns a new character sequence that is a subsequence of this sequence.
`sb.substring(int start,`*end*`)`	Returns a new String that contains a subsequence of characters currently contained in this character sequence.
`sb.toString()`	Returns a string representing the data in this sequence.

Thread

Creating a thread in Java is done like this:

`Thread thread = new Thread();`

To start the thread

`thread.start();`

To join the thread

`thread.join()`

Thread subclass

```
public class MyThread extends Thread {
public void run(){System.out.println("MyThread running"); }
}
MyThread myThread = new MyThread();
myTread.start();
```

OR:

```
Thread thread = new Thread(){
public void run(){
System.out.println("Thread Running");}
}`
thread.start();
```

Runnable Interface Implementation

`public interface Runnable() {public void run();}`

To implement a Runnable

1. Java Class Implements Runnable

```
public class MyRunnable implements Runnable {
public void run(){
System.out.println("MyRunnable running"); }
}
```

2. Anonymous Implementation of Runnable

```
Runnable myRunnable =
```

Thread (cont)

```
new Runnable(){
public void run(){
System.out.println("Runnable running");}
}`
```

3. Java Lambda Implementation of Runnable

`Runnable runnable = () -> { System.out.println("Lambda Runnable running"); };`

Starting a Thread With a Runnable

`Runnable runnable = new MyRunnable();` // or an anonymous class, or lambda...

```
Thread thread = new Thread(runnable);
thread.start();
```

Synchronization

1. Synchronized Instance Methods

```
public synchronized void synchronisedCalculate()
{setSum(getSum() + 1);}
```

2. Synchronized Static Methods

```
public static synchronized void syncStaticCalculate()
{staticSum = staticSum + 1;}
```

**

3. Synchronized Blocks Within Methods

```
public void performSynchronisedTask() {
//unsynchronized part
// ...
synchronized (this/obj) {
setCount(getCount()+1);}}
```

import java.time

`import java.time.LocalDate;`	yyyy-MM-dd
`LocalDate myObj = LocalDate.now(); System.out.println(myObj);`	"2020-12-28"
`import java.time.LocalTime;`	HH-mm-ss-ns
`LocalTime myObj = LocalTime.now(); System.out.println(myObj);`	"00:01:02.290985"
`import java.time.LocalDateTime;`	yyyy-MM-dd-HH-mm-ss-ns
`LocalDateTime myDateObj = LocalDateTime.now(); System.out.println(myDateObj);`	"2020-12-28T00:01:02.311139"

source: https://www.w3schools.com/java/java_date.asp

import java.util.HashMap;

`public void clear()`	This method removes all of the mappings from this map.
`public boolean containsKey(Object key)`	This method returns true if this map contains a mapping for the specified key.
`public boolean containsValue(Object value)`	This method returns true if this map maps one or more keys to the specified value.
`public Set<Map.Entry<K,V>> entrySet()`	This method returns a Set view of the mappings contained in this map.

import java.util.HashMap; (cont)

public V get(Object key)	This method returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
public boolean isEmpty()	This method returns true if this map contains no key-value mapping.
public Set<K> keySet()	This method returns a Set view of the keys contained in this map.
public V put(K key, V value)	This method associates the specified value with the specified key in this map.
public void putAll(Map<? extends K,? extends V> m)	This method copies all of the mappings from the specified map to this map.
public V remove(Object key)	This method removes the mapping for the specified key from this map if present.
public int size()	This method returns the number of key-value mappings in this map.
public Collection<V> values()	This method returns a Collection view of the values contained in this map.

import java.util.PriorityQueue;

public boolean add(E e)	This method inserts the specified element into this priority queue.
public boolean offer(E e)	This method inserts the specified element into this priority queue.
public void clear()	This method removes all of the elements from this priority queue.
public boolean contains(Object o)	This method returns true if this queue contains the specified element.
public Iterator<E> iterator()	This method returns an iterator over the elements in this queue.

import java.util.PriorityQueue; (cont)

public E peek()	This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
This method retrieves and removes the head of this queue, or returns null if this queue is empty.	This method retrieves and removes the head of this queue, or returns null if this queue is empty.
public boolean remove(Object o)	This method removes a single instance of the specified element from this queue, if it is present.
public int size()	This method returns the number of elements in this collection.
public Object[] toArray()	This method returns an array containing all of the elements in this queue.
public <T> T[] toArray(T[] a)	This method returns an array containing all of the elements in this queue; the runtime type of the returned array is that of the specified array.

The java.util.PriorityQueue class is an unbounded priority queue based on a priority heap.Following are the important points about PriorityQueue

- The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

- A priority queue does not permit null elements.

- A priority queue relying on natural ordering also does not permit insertion of non-comparable objects.

import java.util.Stream;

https://developer.ibm.com/zh/languages/java/articles/j-lo-java8stream-api/

bye

Character class

Character c = '1';

Character.isLetter()	The method determines whether the specified char value is a letter.

Character class (cont)

Character.isDigit()	The method determines whether the specified char value is a digit.
Character.isLetterOrDigit(char ch)	Determines if the specified character is a letter or digit.
Character.digit(char ch, int radix)	Returns the numeric value of the character ch in the specified radix.
Character.forDigit(int digit, int radix)	Determines the character representation for a specific digit in the specified radix.
Character.isWhitespace()	Determines whether the specified char value is white space.
Character. isUpperCase()	Determines whether the specified char value is uppercase.
Character. toUpperCase()	Determines whether the specified char value is lowercase.
Character.toLowerCase()	Returns the lowercase form of the specified char value.
Character.toString()	Returns a String object representing the specified character value that is, a one-character string.
Character. codePointAt(seq,index)	return the ASCII value of char at index in seq
c.charValue();	return the ASCII value of this char.
Character.getNumericValue(char ch)	Returns the int value that the specified Unicode character represents.
Character.isSpaceChar(char ch)	Determines if the specified character is a Unicode space character.
Character.isWhitespace(char ch)	Determines if the specified character is white space according to Java.

String class

String s = "123"	
s.charAt()	This method returns the character located at the String's specified index. The string indexes start from zero.
s.compareTo()	This method compares two strings lexicographically.
s.compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences.
s.concat()	This method appends one String to the end of another. The method returns a String with the value of the String passed into the method, appended to the end of the String, used to invoke this method.
s.contentEqual(StringBuilder sb)	This method returns true if and only if this String represents the same sequence of characters as specified in StringBuffer.
s.endsWith()	This method tests if this string ends with the specified suffix.
s.equals()	This method compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.
s.equalsIgnoreCase()	This method compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case, if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

String class (cont)

s.getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	This method copies characters from this string into the destination character array.
s.indexOf(int ch)	Returns the index within this string of the first occurrence of the specified character.
s.indexOf(int ch, int fromIndex)	Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
s.lastIndexOf(int ch)	Returns the index within this string of the last occurrence of the specified character.
s.lastIndexOf(int ch, int fromIndex)	Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
s.length()	Returns the length of this string.
s.matches(regex)	Tells whether or not this string matches the given regular expression.
s.regionMatches(boolean ignoreCase,int this_offset,String other,int that_offset,int len))	This method has two variants which can be used to test if two string regions are equal.
s.replace(old,new)	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
s.split(del,*limit*)	split the string
s.subSequence(beg,end)	Returns a new character sequence that is a subsequence of this sequence.
s.substring(beg,*end*)	Returns a new string that is a substring of this string.
s.toCharArray()	Converts this string to a new character array.

String class (cont)

s.toLowerCase()	Converts all of the characters in this String to lower case using the rules of the default locale.
s.toUpperCase	Converts all of the characters in this String to upper case using the rules of the default locale.
s.trim()	Returns a copy of the string, with leading and trailing whitespace omitted.

import java.util.ArrayList;

`ArrayList<String> cars = new ArrayList<String>();`	create an Arraylist
`cars.get(index);`	get item at index
`cars.set(0, "Opel");`	set item at index
`cars.remove(0);`	remove item at index
`cars.clear();`	remove all items
`cars.size();`	get size of arraylist
`import java.util.Collections;`	Import the Collections class
`Collections.sort(myNumbers);`	Sort myNumbers

import java.util.Arrays;

`public static <T> List<T> asList(T... a)`	This method returns a fixed-size list backed by the specified array.
`public static int binarySearch(arr, `*`fromInd`*`,`*`toInd`*`, target,)`	This method searches the specified array of bytes for the specified value using the binary search algorithm.
`public static <T> T[] copyOf(T[] original, int newLength)`	This method copies the specified array, truncating or padding with nulls (if necessary) so the copy has the specified length.
`public static T[] copyOfRange(originalArr, int from, int to)`	This method copies the specified range of the specified array into a new array.
`public static boolean deepEquals(Object[] a1, Object[] a2)`	This method returns true if the two specified arrays are deeply equal to one another.

import java.util.Arrays; (cont)

public static boolean equals(Object[] a, Object[] a2)	This method returns true if the two specified arrays of Objects are equal to one another.
public static void fill(Object[] a, Object val)	This method assigns the specified Object reference to each element of the specified array of Objects.
public static void fill(object[] a, int fromIndex, int toIndex, object val)	This method assigns the specified Object reference to each element of the specified range of the specified array of Objects.
public static void sort(Object[] a)	This method sorts the specified array of objects into ascending order, according to the natural ordering of its elements.
public static void sort(Object[] a, int fromIndex, int toIndex)	This method sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements.
public static String toString(Object[] a)	This method returns a string representation of the contents of the specified array of ints.

import java.util.Collections;

public static <T> boolean addAll(Collection<? super T> c, T.. a)	This method adds all of the specified elements to the specified collection.
public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)	This method searches the specified list for the specified object using the binary search algorithm.
public static <T> void copy(List<? super T> dest,List<? extends T> src)	This method copies all of the elements from one list into another.
public static boolean disjoint(Collection<?> c1,Collection<?> c2)	This method returns true if the two specified collections have no elements in common.

import java.util.Collections; (cont)

public static <T> void fill(List<? super T> list, T obj)	This method returns an enumeration over the specified collection.
public static <T> void fill(List<? super T> list, T obj)	This method replaces all of the elements of the specified list with the specified element.
public static int indexOfSubList(List<?> source, List<?> target)	This method returns the starting position of the first occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence.
public static int lastIndexOfSubList(List<?> source, List<?> target)	This method returns the starting position of the last occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence.
public static <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll)	This method returns the maximum element of the given collection, according to the natural ordering of its elements.
public static <T extends Object & Comparable<? super T>> T min(Collection<? extends T> coll)	This method Returns the minimum element of the given collection, according to the natural ordering of its elements.
public static <T> boolean replaceAll(List<T> list,T oldVal,T newVal)	This method replaces all occurrences of one specified value in a list with another.
public static void shuffle(List<?> list)	This method randomly permutes the specified list using a default source of randomness.
public static void rotate(List<?> list,int distance)	This method rotates the elements in the specified list by the specified distance.
public static <T extends Comparable<? super T>> void sort(List<T> list)	This method sorts the specified list into ascending order, according to the natural ordering of its elements.

import java.util.Collections; (cont)

`public static void swap(List<?> list,int i,int j)`	This method swaps the elements at the specified positions in the specified list.

import java.util.HashSet;

`public boolean add(E e)`	This method adds the specified element to this set if it is not already present.
`public void clear()`	This method removes all of the elements from this set.
`public boolean contains(Object o)`	This method returns true if this set contains the specified element.
`public boolean isEmpty()`	This method returns true if this set contains no elements.
`public Iterator<E> iterator()`	This method returns an iterator over the elements in this set.
`public boolean remove(Object o)`	This method removes the specified element from this set if it is present.
`public int size()`	This method returns returns the number of elements in this set(its cardinality).

import java.util.Stack

`public boolean empty()`	This method tests if this stack is empty.
`public Object peek()`	This method looks at the object at the top of this stack without removing it from the stack.
`public Object pop()`	This method removes the object at the top of this stack and returns that object as the value of this function.
`public Object push(Object item)`	This method pushes an item onto the top of this stack.
`public int search(Object o)`	This method returns the 1-based position where an object is on this stack.

import java.util.Random

`protected int next(int bits)`	This method generates the next pseudorandom number.
`public boolean nextX()`	X = Boolean/Int/Double/Float/Long uniformly distributed between 0 and 1
`public double nextGaussian()`	This method returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
`public int nextInt(int n)`	This method returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
`public void setSeed(long seed)`	This method sets the seed of this random number generator using a single long seed.