

String Methods

<code>.toUpperCase()</code>	<code>.equals(str)</code>
<code>.toLowerCase()</code>	<code>.indexOf(e)</code>
<code>.substring(i,j) j is excluded</code>	<code>.concat(str)</code>
<code>.length()</code>	<code>.charAt(i)</code>
<code>.compareTo(str)</code>	<code>.contains(e)</code>
<code>Integer.parseInt(intString)</code>	
<code>Double.parseDouble(doubleString)</code>	

```
import java.util.Scanner;
Scanner input= new Scanner(System.in);
```

Scanner Methods:

```
.nextLine() ends with line
.next() ends with white space
.nextDouble()
.nextInt()
```

Naming

keywords	lowercase	rule
variables	camelCase	convention
constants	ALL_CAPS	rule
class names	CamelCase	convention

Math Methods

<code>Math.pow(a, b)</code>	<code>Math.PI()</code>
<code>Math.log(x), Math.log10(x)</code>	<code>Math.sqrt(x)</code>
<code>Math.floor rounds down</code>	<code>Math.ceil() rounds up</code>
<code>Math.random()</code>	<code>Math.min(), Math.max()</code>

```
import java.lang.Math;
```

has sin, cos, tan, toRadians, toDegree, asin, acos, atan

low + Math.random()* high (non-inclusive)

Escape Sequences

<code>\t</code>	tab
<code>\n</code>	newline
<code>\"</code>	double quote
<code>\\</code>	backslash

Date Class

```
jav.util.Date date= new java.util.Date;

date.toString();
```

Point2D Class

```
import java.geometry.Point2D;

Point2D variable = new Point2D(x, y);
```

Objects

no variable constructor	<code>Circle() { }</code>
constructor	<code>Circle (double radius) { this.radius=radius; }</code>
getter	<code>double getArea() { return 2 x radius x radius x Math.PI; }</code>
setter	<code>void setRadius(double radius) { this.radius=radius; }</code>
instanceof	tests whether an object is an instance of a class
<code>super();</code>	calls no arg constructor of superclass
<code>super(-arg);</code>	calls matching arg constructor of superclass
array of objects	<code>for (int i, i<thing.length, i++) array[i]= new Thing(param);}</code>

"this.radius" is an instance variable, as is the original data field
"radius" is the local variable

constructors must have same name as class

constructors do not have a return type, not even void

constructors are invoked using the new operator when an object is created
default constructor goes to class with no other constructors defined

Abstract Classes and Interfaces

Abstract Classes	Interfaces
cannot use "new"	only has abstract methods
methods have no body	no constructors
mix of abstract/non-abstract methods	"implements"
"extends"	contains constants
has constructors	
contains contacts and variables	
<code>public abstract class ClassName {</code>	
<code>java.lang.Comparable</code>	
<code>public interface comparable <E>{</code>	
<code>public int compareTo(E o); }</code>	
<i>returns -1 for less than, 0 for equals, 1 for greater than</i>	
<code>java.lang.Cloneable</code>	
<code>public interface clonable {</code>	
<i>use .clone()</i>	

Loops

while	<code>int x=n;</code> <code>while (x>1) {</code> <code>change x; }</code>
for	<code>for (int i, i<variable, i++){</code>
for each (arrays)	<code>for (int i: list){</code>
boolean	<code>(boolean ? true : false)</code>

Characters

<code>.isDigit(ch)</code>	<code>.isLetter(ch)</code>
<code>.isLowerCase(ch),</code>	<code>.toLowerCase(ch),</code>
<code>.isUpperCase(ch)</code>	<code>.toUpperCase(ch)</code>

ArrayList Methods



By **sefergus**
cheatography.com/sefergus/

Published 16th October, 2016.
Last updated 16th October, 2016.
Page 1 of 3.

create	<code>ArrayList<type> name = new ArrayList<type>();</code>
access element	<code>list.get(i)</code>
update element	<code>list.set(i, e)</code>
return size	<code>list.size()</code>
add element	<code>list.add((i), e)</code>
remove element	<code>list.remove(i or e)</code>

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

ArrayList Methods (cont)

remove all elements list.clear()

import java.util.ArrayList;

Important methods

modifier returnType methodName(p-params){

public Class ClassName{
 public static void main (String[] args)

Scanner input= new Scanner(System.in)
System.out.println(line);

public static type name (type param){
 return type; }

public boolean equals (Object o){
 if (o instanceof Person){
 Person p= (Person) o;
 return this.name.equals(p.getName());
 }else{
 return false;
 }
}

public String toString(){
 return "String";}

to use a method from a different class:
Class.method(var);

Array methods

java.util.Arrays.sort(array) .length

java.util.Arrays.equal(a1, a2) *if corresponding elements are the same*

Arrays.toString(array) .reverse()

array[i] array[i]=e

import java.util.Arrays;
int[] values= new int[10]
default values: 0, /u0000, or false
printing gives reference
methods can modify arrays
import java.util.Arrays;
multi-dimensional arrays: arrays of arrays.
elementType [rows] [columns] arrayVar

Vocabulary

composition information belongs to one object

association/-segregation information can belong to many objects

public visibility can be seen anywhere in any package

private visibility can be seen within class

protected visibility in package and subclasses of this in any package

runtime error crash

compile error doesn't run

final static constant modifier

byte 8 bits*

block /* ... */

comment

line //

comment

javadoc /** ... */

comments

break; breaks out of a loop

continue; stays in loop

variable declaration creating a variable with its type

static shared by all instances of a class

relational operator <, <=, ==, !=, >, >=

logical operator !, &&, || (*inclusive*), ^ (*exclusive*)

Numeric Types (in order) byte, short, int, long, float, double

Variable Scope variables only exist within {}

assignment operators =, +=, -=, *=, /=, %=

operators +, -, %, / (*truncates for int*)

Vocabulary (cont)

increment/decrement operators ++, --

instance method a method that can only be invoked from a specific object

local variable within a method

instance variable dependent on the specific instance (class)

overloading methods methods can have the same name as long as their method signatures are different

binary operators are left-associative, assignment operators are right associative