

### String Methods

.toUpperCase()	.equals(str)
.toLowerCase()	.indexOf(e)
.substring(i,j) <i>j is excluded</i>	.concat(str)
.length()	.charAt(i)
.compareTo(str)	.contains(e)

Integer.parseInt(intString)

Double.parseDouble(doubleString)

```
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

Math.pow(a, b)	Math.PI()
----------------	-----------

Math.log(x), Math.log10(x)	Math.sqrt(x)
-------------------------------	--------------

Math.floor <i>rounds down</i>	Math.ceil() <i>rounds up</i>
-------------------------------	------------------------------

Math.random()	Math.min(), Math.max()
---------------	---------------------------

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

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

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

### Escape Sequences

\t	tab
\n	newline
\"	double quote
\\	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	Circle() { } r
constructor	Circle (double radius) { this.radius=radius;} r
getter	double getArea() { return 2 x radius x radius x Math.PI; } r
setter	void setRadius(double radius) { this.radius=radius;} r
instanceof	tests whether an object is an instance of a class
super();	calls no arg constructor of superclass
super(arg);	calls matching arg constructor of superclass

### Objects (cont)

array of objects for (int i, i<thing.length, i++)  
array[i]= new Thing(param);

"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

```
public abstract class ClassName {
```

```
java.lang.Comparable
public interface comparable <E>{
    public int compareTo(E o); }
    returns -1 for less than, 0 for equals,
    1 for greater than
```

```
java.lang.Cloneable
public interface clonable {}
    use .clone()
```

### Loops

while	int x=n; while (x>1) { change x; }
-------	--

for	for (int i, i<variable, i++){
-----	-------------------------------

for each (arrays)	for (int i: list){
-------------------	--------------------

boolean	(boolean ? true : false)
---------	--------------------------

### Characters

.isDigit(ch)	.isLetter(ch)
.isLowerCase(ch), .isUpperCase(ch)	.toLowerCase(ch), .toUpperCase(ch)

### ArrayList Methods

create      `ArrayList<type> name = new ArrayList<type>();`

access element      `list.get(i)`

update element      `list.set(i, e)`

return size      `list.size()`

add element      `list.add((i), e)`

remove element      `list.remove(i or e)`

remove all elements      `list.clear()`

`import java.util.ArrayList;`

### Important methods

modifier returnType  
methodName(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

### Vocabulary (cont)

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)`

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