

Primitive Data types

Name	Size	Values
byte	1 B	-128 → 127
short	2 B	-32768 → 32767
int	4 B	-2•10 ⁹ → 2•10 ⁹
long	8 B	-9•10 ¹⁵ → 9•10 ¹⁵
float	4 B	6-7 decimal digits
double	8 B	15 decimal digits
boolean	1 b	true or false
char	2 B	Characters or ASCII

Comparison Operators

Equal	<code>x == y</code>	Not equal	<code>x != y</code>
Less	<code>x < y</code>	Less or equal	<code>x <= y</code>
Greater	<code>x > y</code>	Greater or equal	<code>x >= y</code>

String Methods

Usage	Description
<code>s.length();</code>	Length of a string
<code>s.charAt(x);</code>	Extract the ⁿ th character
<code>s.toUpperCase();</code>	Returns a copy of a string in ALL CAPS
<code>s.toLowerCase();</code>	Returns a copy of a string in lowercase
<code>s.indexOf(c);</code>	Returns the first occurring index of <i>c</i>
<code>s.replace(old, new);</code>	Search and Replace
<code>s.split(Regex);</code>	Splits string into tokens
<code>s.equals(s2);</code>	Compares strings (true if equal)
<code>s.compareTo(s2);</code>	Returns "0" if equal, "+" if <i>s</i> > <i>s2</i> , "-" if <i>s</i> < <i>s2</i>

IF Statement

```
// conditional handler
if (condition) {
    // pose a condition ...
} else if (other condition) {
    // if first condition not met pose another one ...
} else {
    // if second condition not met finally ...
}
```

FOR EACH Statement

```
// self-iterating loop for iterable objects
for (var: collection) {
    statements
}
```

FOR Statement

```
// structured iterator for non-iterable objects
for (int i = 0; i < max; i++) {}
```

Continue

```
for (int i = 0; i < max; i++) {
    // jump to the next iteration
    continue;
}
```

Break

```
while (true) {
    // exit early a loop
    break;
}
```



By [lucamazza](#)

Not published yet.

Last updated 2nd October, 2023.

Page 1 of 4.

Sponsored by [Readable.com](#)

Measure your website readability!

<https://readable.com>

TRY-CATCH Statement

```
// exception handler
try {
    // try to do something
} catch (exception e) {
    // when you fail do...
} finally {
    // when you are done do...
}
```

Enumerals

```
// "class" containing a group of constants
enum Name {
    NAME1,
    NAME2,
    NAME3
}

// constant are accessible per dot notation
// they are typed as the declared enum
Name var = Name.NAME1;
```

Object Example

```
public class Object {
    private <type> attr1;
    public Object (<type> attr1) {
        this.attr1 = attr1;
    }
    public void method() {
        // ...
    }
}
```

Interfaces

```
// An interface manages accessibility of object
// it is an abstract object
public interface Intrf {
    String const = " CIA O";
    public void method1();
}

public class Obj implements Intrf {
    public void Obj() { }
    @ov erride
    public void method1() {}
}
```

Parent Class

```
// parent class
public class Parent implements Intrf {}

// child class, inherits Parents characteristics
public class Child extends Parent {}
```

ArrayList Methods

Usage	Description
<code>list.add(item);</code>	Add <i>item</i> to the list
<code>list.get(n);</code>	Return the <i>n</i> th item
<code>list.size();</code>	Return number of item
<code>list.remove(n);</code>	Remove the <i>n</i> th item
<code>list.set(n, value);</code>	Put at the <i>n</i> th position value

Non-Primitive Data Types

Name	Declaration
String	<code>String name = " Luc a";</code>
Array	<code>type[] = {x, y};</code>
Class	<code>access class Obj { ... }</code>
Interface	<code>public interface ObjInterface { ... }</code>



By [lucamazza](#)

Not published yet.

Last updated 2nd October, 2023.

Page 2 of 4.

Sponsored by [Readable.com](#)

Measure your website readability!

<https://readable.com>

Boolean Operators

NOT	<code>! x</code>
AND	<code>x && y</code>
OR	<code>x y</code>
XOR	<code>x ^ y</code>

Arithmetic Operators

Addition	<code>x + y</code>
Subtraction	<code>x - y</code>
Multiplication	<code>x * y</code>
Division	<code>x / y</code>
Increment	<code>x++ ++x</code>
Decrement	<code>x-- --x</code>
Modulus	<code>x % y</code>

WHILE Statement

```
// pre-conditioned iterator ...
while (condition) {}
```

Scanner

```
import java.util.Scanner;
public class Obj{
    public static void main (String[] args) {
        Scanner input = new Scanner(System.in);

        String str = input.nextLine();
        int i = input.nextInt();
        input.close();
    }
}
```

DO-WHILE Statement

```
// post-conditional iterator
do { } while (condition);
```

SWITCH Statement

```
// case-oriented condition handler
switch (condition) {
    case value0:
        statements
        break;
    case value1:
        statements
        break;
    default:
        statements
}
```

Ternary operator

```
// same as an if, but inline!
condition ? iftrue : iffalses;
```

Type conversion

String to any number `int i = Integer.parseInt(str);`

Any type to String `String s = String.valueOf(value);`

Between numeric types `int i = (int) number;`

Array Utilities

Usage	Description
<code>Arrays.copyOf(orig, orig.length);</code>	Copy array onto another one



By [lucamazza](#)

Not published yet.

Last updated 2nd October, 2023.

Page 3 of 4.

Sponsored by [Readable.com](#)

Measure your website readability!

<https://readable.com>

Recursion

```
// method that sums all the number 1..10
// attention! recursion can incur in an infinite
loop
// handle it well
public static int sum(int start, int end) {
    if (end > start) {
        return end + sum(start, end - 1);
    } else {
        return end;
    }
}
```

Access Definitions

	public	protected	package	private
Class	•	•	•	•
Package	•	•	•	
Subclasses	•	•		
Classes	•			

Usage of an Object

```
// to use an object
public static void main (String[] args) {
    // declar ation meets constr uctor method
    Object obj = new Object (at tr1);
    obj.me thod();
}
```

Dynamic Data Types

```
import java.util.*;
List <type> names = new ArrayList <type>();
Vector<type> vec = new Vector<type>();
Stack<type> stk= new Stack< >();
```

Dynamic Data Types Methods

Vector

<code>vec.add(value);</code>	Add an element to the end
<code>vec.add(i, value);</code>	Add an element to a specific index
<code>vec.capacity();</code>	Returns the current capacity
<code>vec.clear();</code>	Empty vector
<code>vec.clone();</code>	Clone the vector

Stack

Inherits from Vector

<code>stk.empty();</code>	Returns true if head is empty
<code>stk.peek();</code>	Returns head of stack
<code>stk.pop();</code>	Delete head of stack
<code>stk.push(value);</code>	Add an element at head
<code>stk.search(value);</code>	Returns index of element