**Cheat Sheet**

This cheat sheet contains the most important takeaways that lead up to section one.

**1. Getting Started**

* class: contains all of your code.
* main(): entry point of your app.
* javac <file-name>.java: compiles your code.
* java <file-name>: runs the compiled code.

**Coding Pitfalls**

* None yet.

**Good coding habits**

* The name of a class must be CamelCase.

**Tips and tricks**

**Terminal**

* Use the **up** key to run previous terminal commands.
* Use the **tab** key to autocomplete.

\*\* The cheat sheet updates after each section.

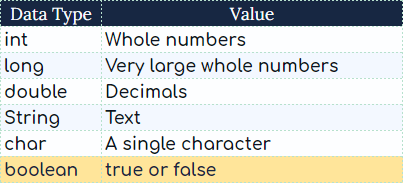
**1. Getting Started**

* class: contains all of your code.
* main(): entry point of your app.
* javac <file-name>.java: compiles your code.
* java <file-name>: runs the compiled code.

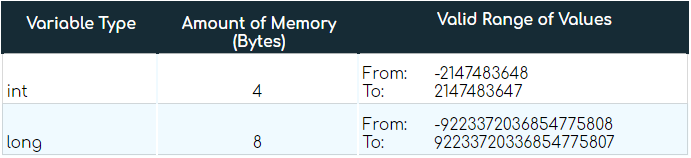
**2. Variables**

**Data types**

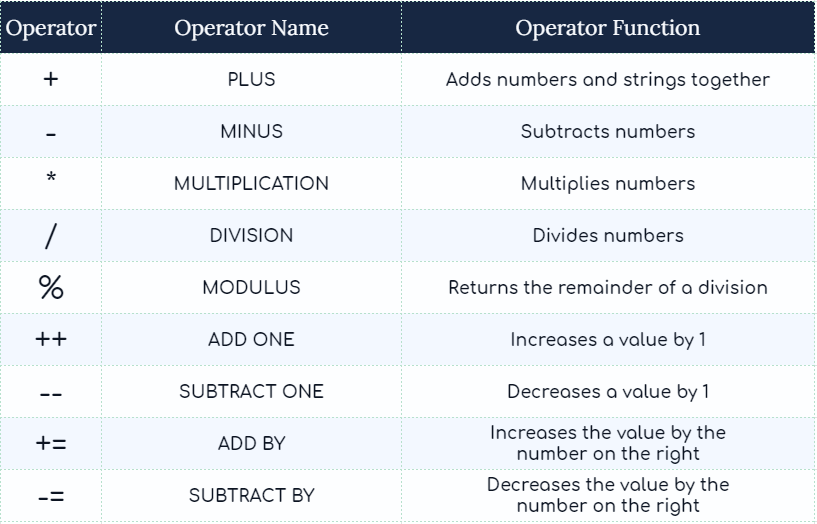
There are 6 main data types (we didn't cover boolean yet).



long can store very big numbers. But, int is faster and takes less memory.



**Operators**



**Scanner**

nextLine(): scans for a String value.

nextDouble(): scans for a double value.

nextLong(): scans for a long value.

nextInt(): scans for an int value.

**Coding Pitfalls**

* Put math operations in brackets if you wish to embed them inside a String.
  + "Example String " + (10 - 2) + " Example String " ;
* Trap: Putting nextLine() ahead of nextInt(), nextDouble(), nextLong(), or next().
  + Solution: add a throwaway nextLine() before the real nextLine().

\*\*The reason this happens will be covered in the lesson: Delimiters.

**Good coding habits**

* class: CamelCase.
* variable: lowerCamelCase.

**Tips and tricks**

**Terminal**

* Use the **up** key to run previous terminal commands.
* Write clear to clear the terminal output.
* Press the tab key for auto-complete.

**Escape characters**

* \n adds a new line of space.
* \t adds a new tab of space.

**Shortcut keys**

* Use CMD/Ctrl+/ to comment a highlighted piece of code.

**Cheat Sheet**

This cheat sheet contains takeaways leading up to section three.

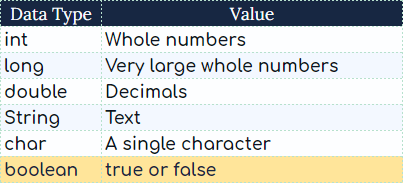
**1. Getting Started**

* class: contains all of your code.
* main(): entry point of your app.
* javac <file-name>.java: compiles your code.
* java <file-name>: runs the compiled code.

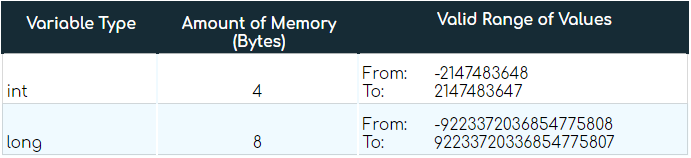
**2. Variables**

**Data types**

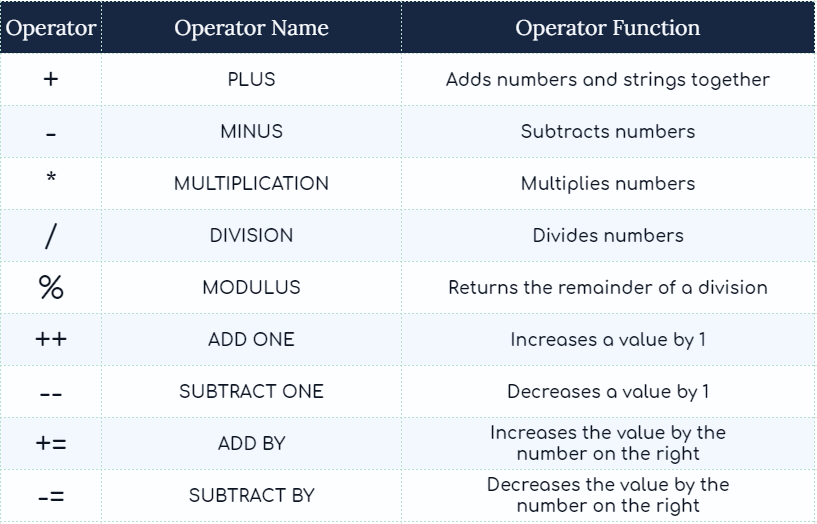
There are 6 main data types.



long can store very big numbers. But, int is faster and takes less memory.



**Math Operators**

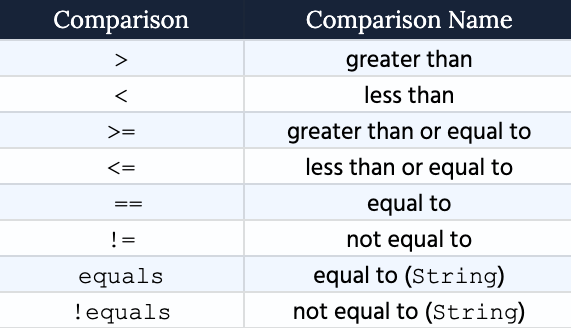


Put math operations in brackets if you wish to embed them inside a String.

* Example: "5 + 2 is " + (5 + 2);

**3. Booleans and Conditionals**

**Comparison Operators**



Put comparison operations in brackets if you wish to embed them inside a String.

* Example: "Five is not smaller than one. I'm certain this is " + (5 < 1);

**Rules of thumb**

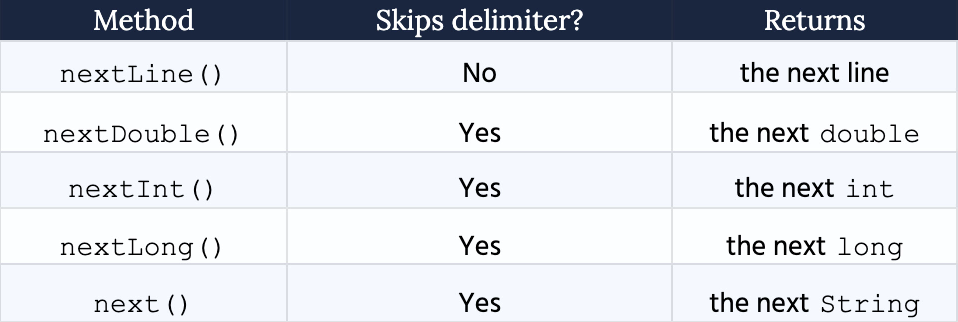
When controlling how your code runs:

* use switch to compare one value against a list of values.
* in any other scenario, use if-else.

Do not confuse = with ==

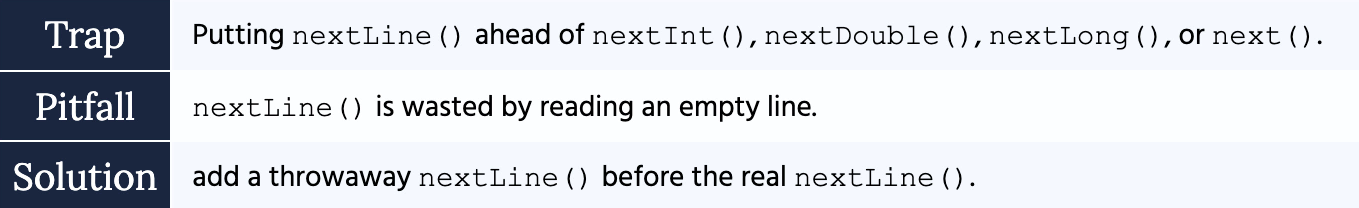
* use = to set a value equal to another.
* use == to compare two values and return a boolean.

**Scanner**



\*\*The default delimiter is white space.

**Coding Pitfalls**



**Good coding habits**

**Conventions**

* class: CamelCase.
* variable: lowerCamelCase.

**Tips and tricks**

**Terminal**

* Use the **up** key to run previous terminal commands.
* Write clear to clear the terminal output.
* Press the tab key for auto-complete.

**Escape characters**

* \n adds a new line of space.
* \t adds a new tab of space.

**Shortcut keys**

* Use CMD/Ctrl+/ to comment a highlighted piece of code.
* In Visual Studio Code, use sysout as a shortcut to System.out.println()

**Cheat Sheet**

This cheat sheet contains the most important takeaways that lead up to section four.

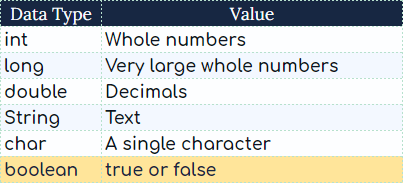
**1. Getting Started**

* class: contains all of your code.
* main(): entry point of your app.
* javac <file-name>.java: compiles your code.
* java <file-name>: runs the compiled code.

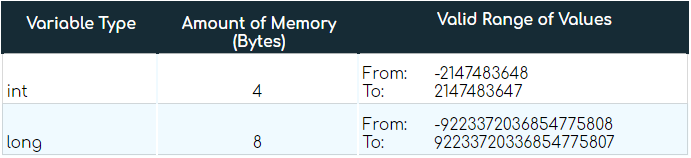
**2. Variables**

**Data types**

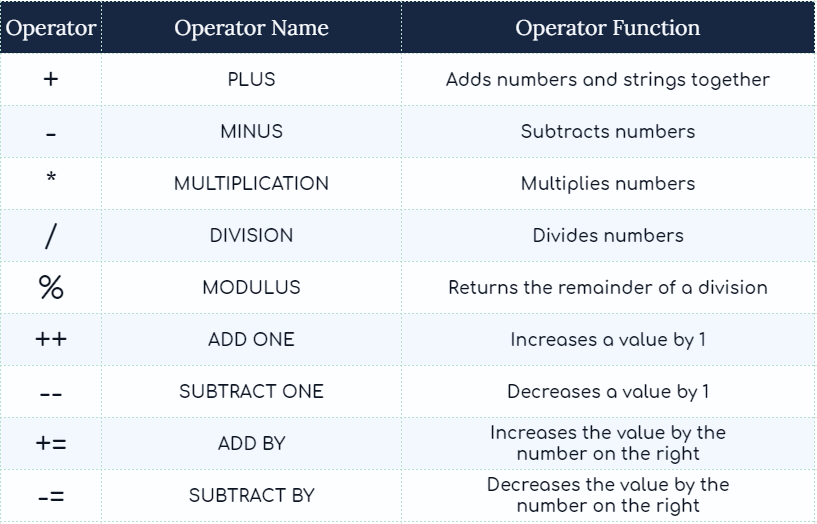
There are 6 main data types.



long can store very big numbers. But, int is faster and takes less memory.



**Math Operators**

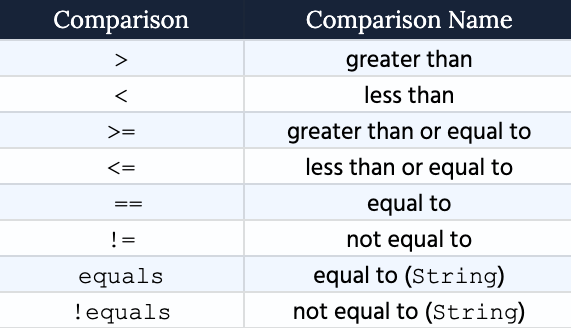


Put math operations in brackets if you wish to embed them inside a String.

* Example: "5 + 2 is " + (5 + 2);

**3. Booleans and Conditionals**

**Comparison Operators**



Put comparison operations in brackets if you wish to embed them inside a String.

* Example: "Five is not smaller than one. I'm certain this is " + (5 < 1);

**Rules of thumb**

When controlling how your code runs:

* use switch to compare one value against a list of values.
* in any other scenario, use if-else.

Do not confuse = with ==

* use = to set a value equal to another.
* use == to compare two values and return a boolean.

**4. Functions**

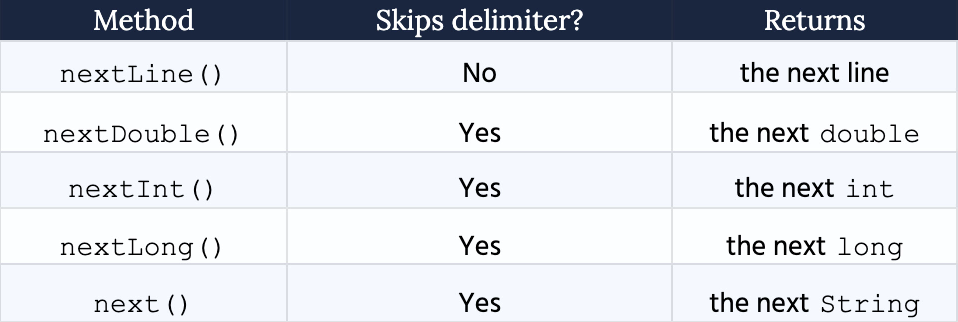
**Function Parts**

1. **Level of access**: private, public.
2. **Return type**: double, int, boolean, char, String, long.
3. **Function** **name**: lowerCamelCase (singChorus, kickBall).
4. **Parameter**: value received by the function.
5. **Argument**: value passed into the function.
6. **Code**: performs your task.

* return breaks the entire function. Nothing after it can run.

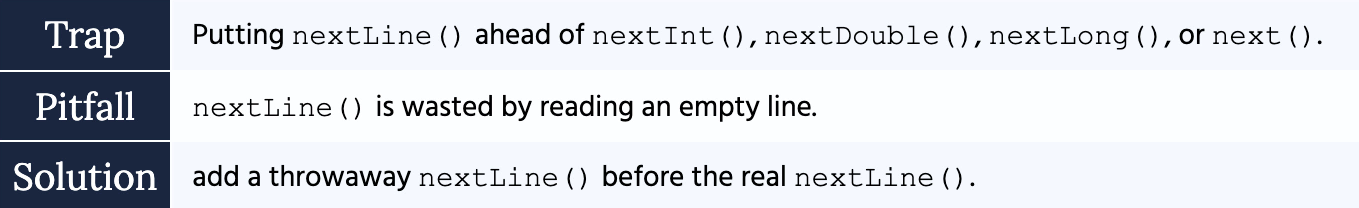
**Rule of thumb:** if a function calculates a value, return it.

**Scanner**



^\*\*The default delimiter is white space.^

**Coding Pitfalls**



**Good coding habits**

**Conventions**

* class: CamelCase.
* variable: lowerCamelCase.
* function: lowerCamelcase.

**Tips and tricks**

**Terminal**

* Use the **up** key to run previous terminal commands.
* Write clear to clear the terminal output.
* Press the tab key for auto-complete.

**Escape characters**

* \n adds a new line of space.
* \t adds a new tab of space.

**Shortcut keys**

* Use CMD/Ctrl+/ to comment a highlighted piece of code.
* In Visual Studio Code, use sysout as a shortcut to System.out.println()

**Cheat Sheet**

This cheat sheet contains the most important takeaways that lead up to section five.

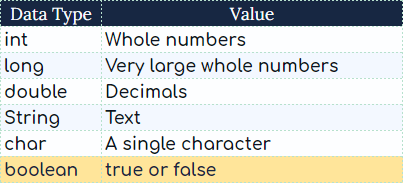
**1. Getting Started**

* class: contains all of your code.
* main(): entry point of your app.
* javac <file-name>.java: compiles your code.
* java <file-name>: runs the compiled code.

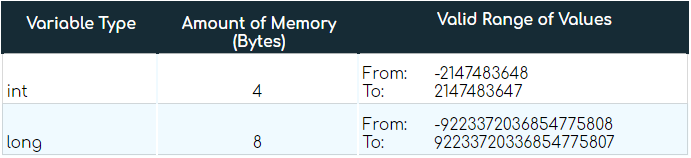
**2. Variables**

**Data types**

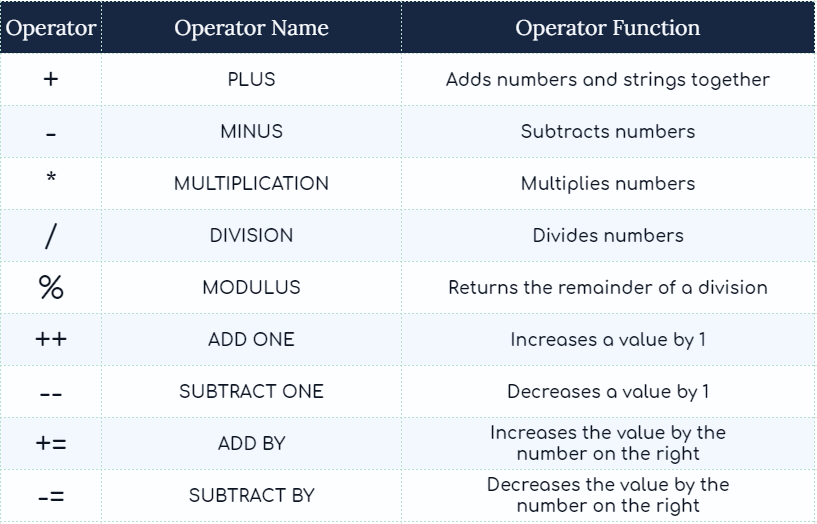
There are 6 main data types.



long can store very big numbers. But, int is faster and takes less memory.



**Math Operators**

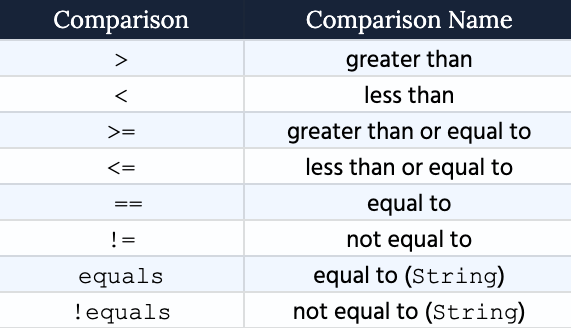


Put math operations in brackets if you wish to embed them inside a String.

* Example: "5 + 2 is " + (5 + 2);

**3. Booleans and Conditionals**

**Comparison Operators**



Put comparison operations in brackets if you wish to embed them inside a String.

* Example: "Five is not smaller than one. I'm certain this is " + (5 < 1);

**Rules of thumb**

When controlling how your code runs:

* use switch to compare one value against a list of values.
* in any other scenario, use if-else.

Do not confuse = with ==

* use = to set a value equal to another.
* use == to compare two values and return a boolean.

**4. Functions**

**Function Parts**

1. **Level of access**: private, public.
2. **Return type**: double, int, boolean, char, String, long.
3. **Function** **name**: lowerCamelCase (singChorus, kickBall).
4. **Parameter**: value received by the function.
5. **Argument**: value passed into the function.
6. **Code**: performs your task.

* return breaks the entire function. Nothing after it can run.

**Rule of thumb:** if a function calculates a value, return it.

**5. Loops**

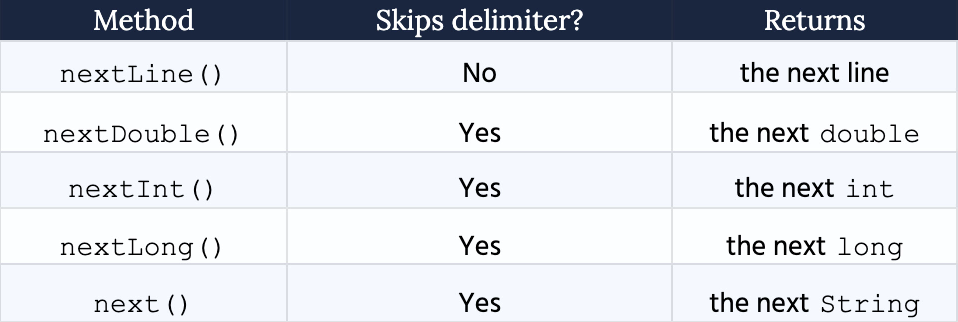
**Rule of thumb:**

* use for loops when you know in advance how many times your code should run.
* use while loops to keep running code while a condition is true.

break**:** breaks a loop and stops it from running.

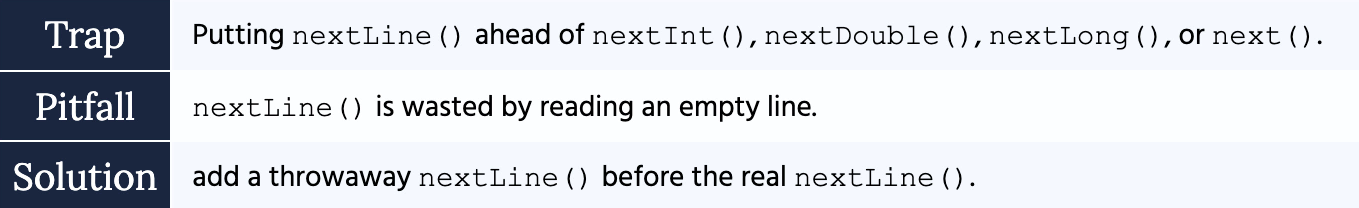
continue: skips the current run, and continues to the next one.

**Scanner**



The default delimiter is white space.

**Coding Pitfalls**



**Good coding habits**

**Conventions**

* class: CamelCase.
* variable: lowerCamelCase.
* function: lowerCamelcase.

**Tips and tricks**

**Terminal**

* Use the **up** key to run previous terminal commands.
* Write clear to clear the terminal output.
* Press the tab key for auto-complete.

**Escape characters**

* \n adds a new line of space.
* \t adds a new tab of space.

**Shortcut keys**

* Use CMD/Ctrl+/ to comment a highlighted piece of code.
* In Visual Studio Code, use sysout as a shortcut to System.out.println()
* Use Ctrl/control + C to interrupt the terminal output.

**Cheat Sheet**

This cheat sheet contains the most important takeaways that lead up to section six.

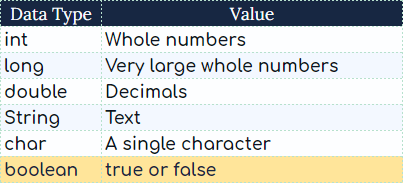
**1. Getting Started**

* class: contains all of your code.
* main(): entry point of your app.
* javac <file-name>.java: compiles your code.
* java <file-name>: runs the compiled code.

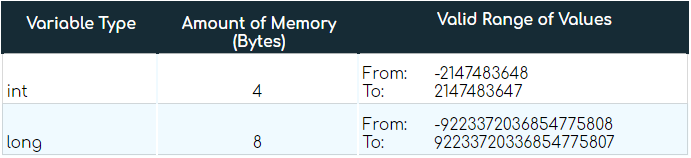
**2. Variables**

**Data types**

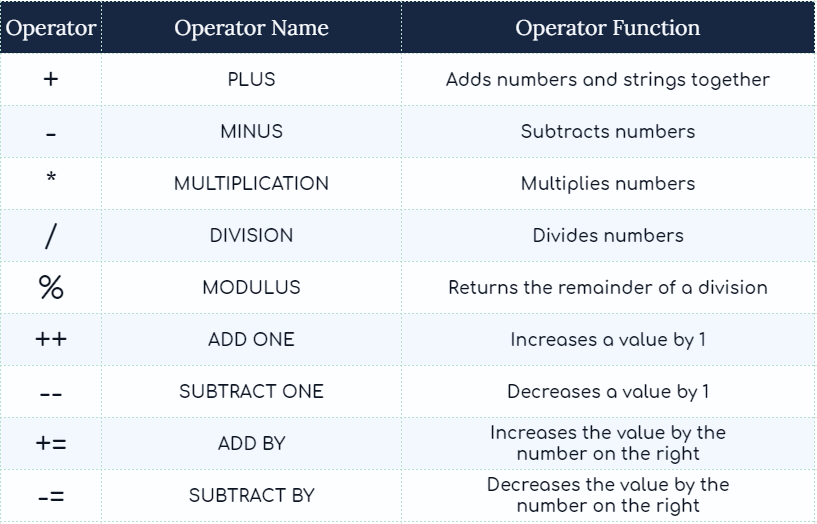
There are 6 main data types.



long can store very big numbers. But, int is faster and takes less memory.



**Math Operators**

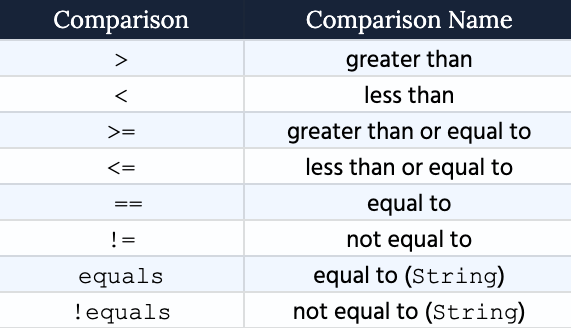


Put math operations in brackets if you wish to embed them inside a String.

* Example: "5 + 2 is " + (5 + 2);

**3. Booleans and Conditionals**

**Comparison Operators**



Put comparison operations in brackets if you wish to embed them inside a String.

* Example: "Five is not smaller than one. I'm certain this is " + (5 < 1);

**Rules of thumb**

When controlling how your code runs:

* use switch to compare one value against a list of values.
* in any other scenario, use if-else.

Do not confuse = with ==

* use = to set a value equal to another.
* use == to compare two values and return a boolean.

**4. Functions**

**Function Parts**

1. **Level of access**: private, public.
2. **Return type**: double, int, boolean, char, String, long.
3. **Function** **name**: lowerCamelCase (singChorus, kickBall).
4. **Parameter**: value received by the function.
5. **Argument**: value passed into the function.
6. **Code**: performs your task.

* return breaks the entire function. Nothing after it can run.

**Rule of thumb:** if a function calculates a value, return it.

**5. Loops**

**Rule of thumb:**

* use for loops when you know in advance how many times your code should run.
* use while loops to keep running code while a condition is true.

break: breaks a loop and stops it from running.

continue: skips the current run, and continues to the next one.

**6. Arrays**

**6.1** **Defining an array**

You can define an array and its values in one line.

Type[﻿] array = { element1, element2, element3 }﻿;

You can also define the length of the array and populate it later.

Type[﻿] array = new Type﻿[﻿3﻿]﻿;

array[﻿0﻿] = element1;

array[﻿1﻿] = element2;

array[﻿2﻿] = element3;

In both cases:

* The variable does not store the array directly.
* It stores a **reference** that points to it.

**6.2** **Accessing values from an array**

You can access values from an array by referring to its index, such that:

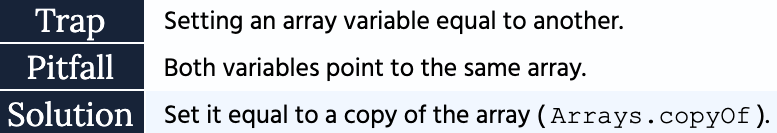
* The first index is 0.
* The last index is one less the length of the array.

**6.3** **Looping an array**

for (int i = 0﻿; i < array.length; i++﻿) {

}

**6.4** **Reference Trap**



**6.5** **Defining a 2D array**

Method 1:

Type[﻿]﻿[﻿] array = {

{element1, element2}﻿,

{element3, element4}﻿,

}﻿;

Method 2:

Type[﻿]﻿[﻿] array = new Type﻿[﻿2﻿]﻿[﻿2﻿]﻿;

array[﻿0﻿]﻿[﻿0﻿] = element1;

array[﻿0﻿]﻿[﻿1﻿] = element2;

array[﻿1﻿]﻿[﻿0﻿] = element3;

array[﻿1﻿]﻿[﻿1﻿] = element4;

**6.6** **Accessing values from a 2D array**

* The first bracket indexes the row.
* The second bracket indexes the element in that row.

**6.7** **Looping a 2D array**

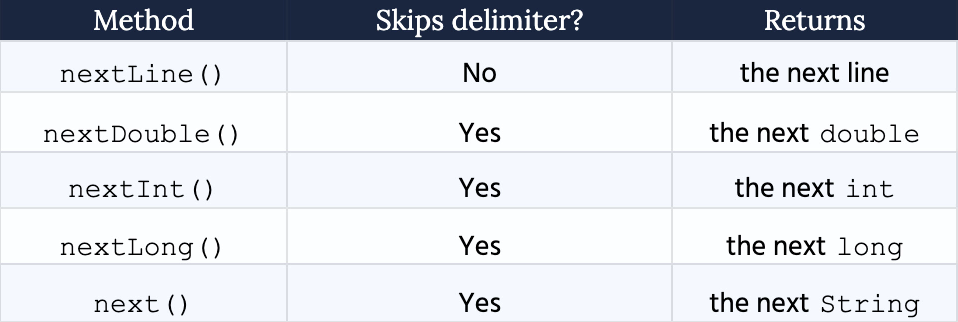
for (int i = 0﻿; i < array.length; i++﻿) {

for (int j = 0﻿; j < array[i]﻿.length; j++﻿) {

}

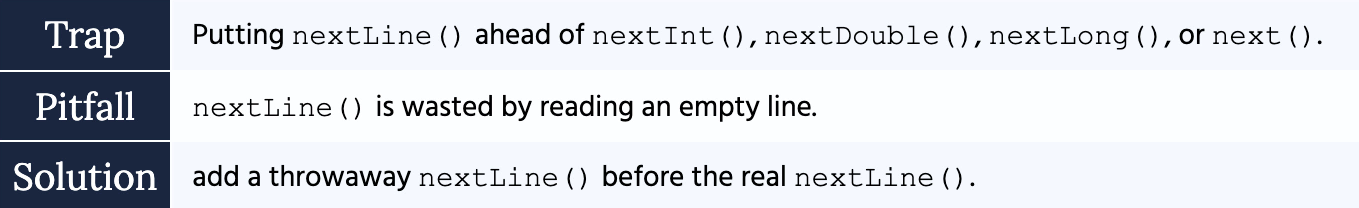
}

**Scanner**



The default delimiter is white space.

**Scanner Pitfall**



**Good coding habits**

**Conventions**

* class: CamelCase.
* variable: lowerCamelCase.
* function: lowerCamelcase.

**Tips and tricks**

**Terminal**

* Use the **up** key to run previous terminal commands.
* Write clear to clear the terminal output.
* Press the tab key for auto-complete.

**Escape characters**

* \n adds a new line of space.
* \t adds a new tab of space.

**Shortcut keys**

* Use CMD/Ctrl+/ to comment a highlighted piece of code.
* In Visual Studio Code, use sysout as a shortcut to System.out.println()
* Use Ctrl/control + C to interrupt the terminal output.
* Highlight and press tab for right indentation.
* Highlight and press shift + tab for left indentation.

**Arrays**

* return an array on the fly using: return new Type[] { element1, element2 };

**Cheat Sheet**

This cheat sheet contains the most important takeaways that lead up to section seven.

**Definitions**

* **Class**: blueprint from which you can create objects.
* **Object**: an object is a *thing* that you can see or describe.
* **Field**: class variable that describes an object.
* **Action**: method (function) that represents what the object can do.
* **Constructor**: runs when you create a new object.
  + Purpose: update every field of the newly created object.
* **this**: refers to the current object that's calling the constructor or method.
  + Purpose: useful when fields and parameters have conflicting names.
* **Copy Constructor**: runs when you create an object.
  + Purpose: copy every value from a source object into a new object.
* **Getter**: method (function) that returns a copy of a private field's value.
* **Setter**: method (function) that updates the value of a private field.
* **toString**: method that returns a String representation of every field in an object.
  + Purpose: When you print an object, Java internally calls the toString method.

**public vs. private**

* public: provides public access to a field or method from anywhere.
  + Constructors and methods tend to be public.
* private: prevents direct access of a field or method outside of its class.
  + To protect the state of an object, fields tend to be private.

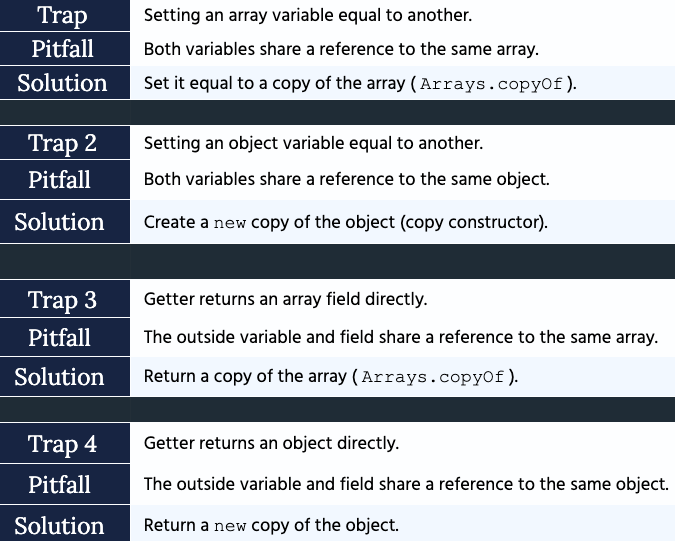
**The Big Three**

If a class has fields, you need to add:

1. **Constructor** -- to update the fields of a new object.
2. **Getters** -- to get the value of a private field.
3. **Setters** -- to set the value of a private field.

**Reference Traps**

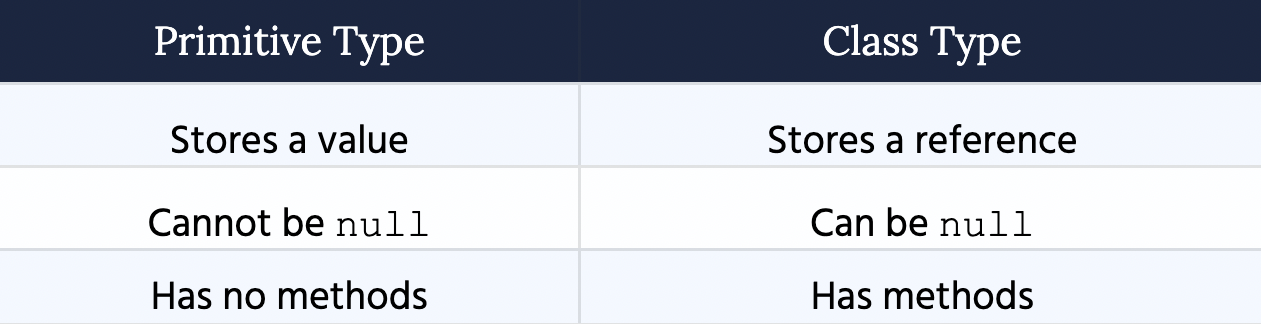
The state of a variable should not change because you updated another.



Falling into a reference trap can:

1. lead to unpredictable behavior.
2. lead to variables changing when you don't expect them to.
3. leave you wondering: *why is my code doing this?*

**Primitive vs. Class Types.**



Examples of primitives types are int, long, char and boolean.

Examples of class types that we created objects from are: Scanner, Dealership, Car.

**Tips and Tricks.**

* Syntax to directly pass an array into a method: new String[] { element, element }
* Syntax to directly return an array: return new String[] { element, element }
* You can autocomplete getters and setters in Visual Studio Code.