Type of variables:

1. Instance variables (non-static fields)

Technically speaking, objects store their individual states in “non-static fields”, that is, fields declared without the **Static** keyword. Non-static fields are also known as instance variables because their values are unique to each instance of a class (to each object, in other words); for example, the **currentSpeed** of one bicycle is independent from the **currentSpeed** of another.

1. Class variables (Static fields)

A class variable is any field declared with the **static** modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. For example, a field defining the number of gears for a particular kind of bicycle could be marked as **Static** since conceptually the same number of gears will apply to all instances. **The code Static int numGears = 6;** would create such a static field. Additionally, the keyword **final** could be added to indicate that the number of gears will never change.

1. Local variables

Similar to how an object stores its state in fields, a method will often store its temporary state in local variables. The syntax for declaring a local variable is similar to declaring a field (for example, **int count = 0**). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared—which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

1. Parameters

You’ve already seen examples of parameters, both in **Bicycle** the class and in the **main** method of the “Hello World!” application. Recall that the signature for the main method is **public static void main(String[] args)**. Here, the **args** variable is the parameter to this method. The important thing to remember is that parameters are always classified as “variables,” not “fields”. This applies to other parameter-accepting constructs as well (such as constructors and exception handlers) that you’ll learn about later in the tutorial.

If your variable stores a constant value, such as **static final int NUM\_GEARS = 6**, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character.

Primitive data types

* byte,
* short,
* char,
* boolean,
* int,

decimal: base 10

hexadecimal: base 16, 0x1a, starts with 0x

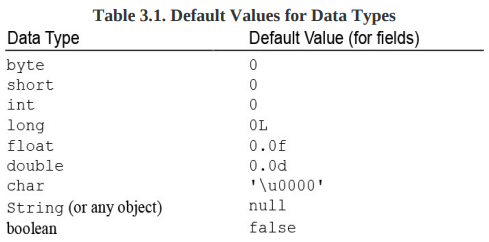
binary: base 2, 0b11010, stars with 0b

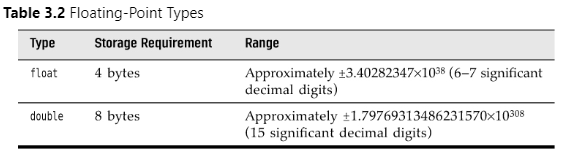
* long,
* float,

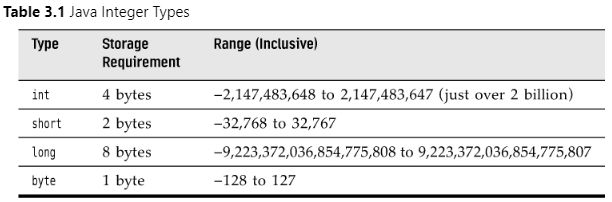
float f1= 1.234f, end with f

double d2 = 1.234

* double







declare variable:

* double salary;
* int days;
* boolean done;
* long earthPopulation;

initialize variable

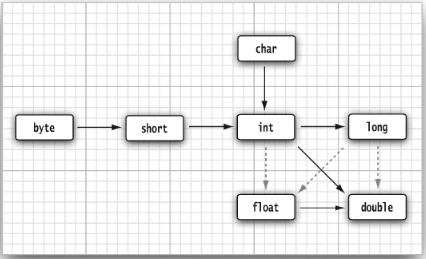
* int vacationDays;
* vacationDays = 12;

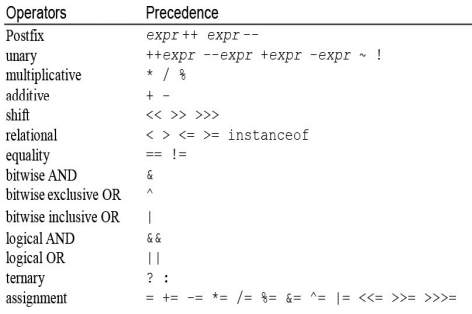
constants

* use final to declare constant
* final double CM\_PER\_INCH = 2.54;

enumerated types:

* restricted values
* enum ShirtSize = { S, M, L, XL };
* ShirtSize s = ShirtSize.XL;





casts

* double x = 9.997;

int nx = (int) x;

assignment

increment decrement

relational and boolean operator

* 1 && 1
* The && and || operators are evaluated in “short circuit” fashion: The second argument is not evaluated if the first argument already determines the value. If you combine two expressions with the && operator

Conditional operator

* x < y ? x: y

switch expressions

* int numLetters = switch (seasonName)

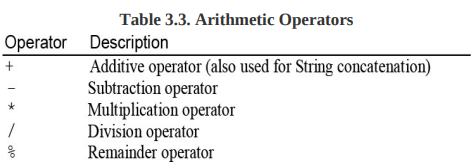
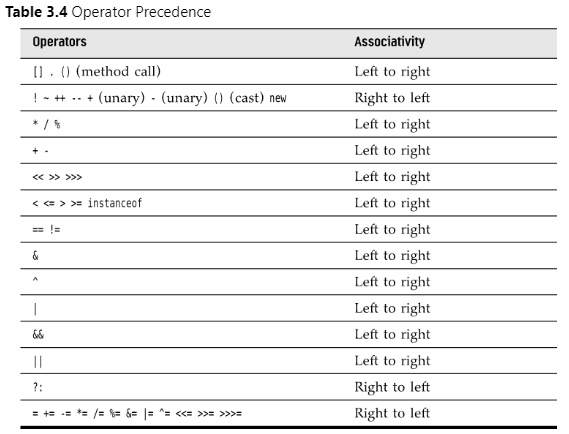
   {

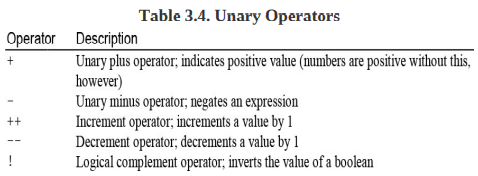
      case 'Spring', 'Summer', 'Winter' -> 6;

      case 'Fall' -> 4;

      default -> -1;

   };





Strings (Immutable)

* strings are immutable, to change string use substring
* String e = “”;

Substrings

* String greet = “hello”;
* String s = greet.substring(0, 3);

Concatenation

* String a = “a”;
* String b = “b”;
* String ab = a + b;

Testing strings for equality

* “hello”.equals(“hello”)
* Don’t use == to compare string. It only determines whether or not the strings are stored in same location

Empty and Null strings

* “jack”.length() == 0
* “jack”.equals(“”)

Code points and Code units

* int cpCount = greeting.codePointCount(0, greeting.length());
* char first = “Hello”.charAt(0); // first is 'H'

String API

Building strings

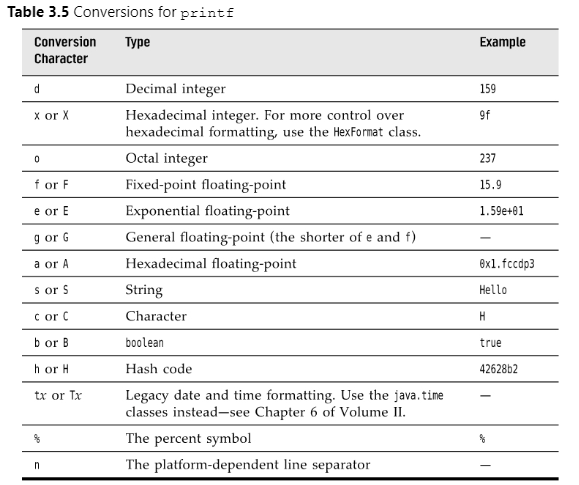
* build up strings from shorter strings
* It would be inefficient to use string concatenation
* Using the StringBuilder class avoids this problem

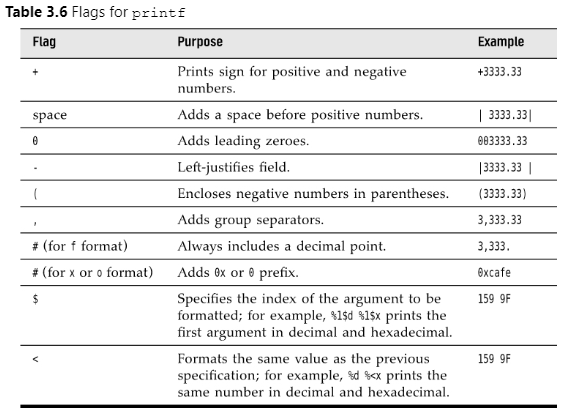
Input and output

* System.out.print('What is your name? ');
* String name = in.nextLine();
* System.out.println('Hello, ' + name)

Formatting output

* System.out.printf('%2.2f', x);
* System.out.printf('Hello, %s. Next year, you'll be %d', name, age);





String format without printing

* String message = String.format('Hello, %s. Next year, you'll be %d', name, age + 1);

File input and output

* Scanner in = new Scanner(Path.of('myfile.txt'), StandardCharsets.UTF\_8);
* Use PrintWriter to write file
* PrintWriter out = new PrintWriter('myfile.txt', StandardCharsets.UTF\_8);

Control flow

* If, else, elseif, while, for, foreach, break, continue, dowhile, switch

Arrays

* smallPrimes = new int[] { 17, 19, 23, 29, 31, 37 };
* int[] a = new int[100];
* for (int i = 0; i < 100; i++){

   a[i] = i; // fills the array with numbers 0 to 99

}

* System.out.println(Arrays.toString(a)); //easy way to print array

Foreach

* for (int element : a){

System.out.println(element);

}

Array copying

* you can copy one array variable to another, but then both variables refer to the same array
* int[] luckyNumbers = smallPrimes;
* luckyNumbers[5] = 12; // now smallPrimes[5] is also 12
* if you want copy all values of one array into new array, use copyOf method
* int[] copiedLuckyNumbers = Arrays.copyOf(luckyNumbers, luckyNumbers.length);

Array sorting

* Arrays.sort(a);

Multidimensional array

* Int[][] magicSquare = {

{16,2},

{2,4},

{3,44}

};