Unit 2

Language Basics

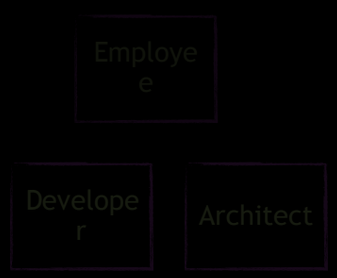
Unit 2

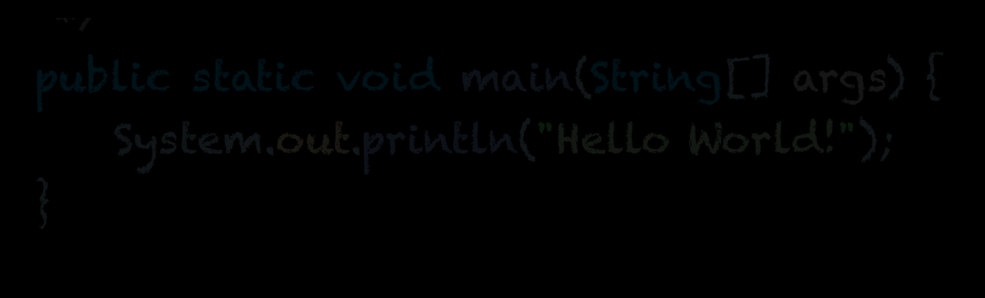
Language Basics

- **Primitives**
- **Variables**
- **Operators**
- **Conditional Statements**
- **Loops**

Tips on Learning

- Slow down
- Do the assignments
- Long-term memory
- Drink water
- Talk about it
- Listen to your brain
- Feel something
- Type and run the code



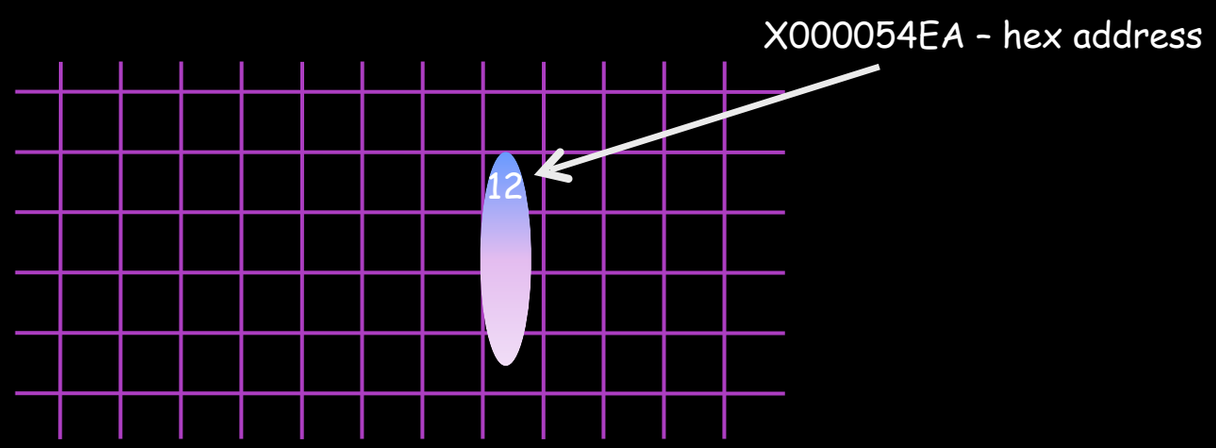


Java Variables

and Primitive Data Types

Java Variables

Location in memory to hold data.



Variables

What are variables?

```
int cadence = 0;
int speed = 0;
int gear = 1;
```

What are the rules and conventions for naming a field?

Besides int, what other data types are there?

Do fields have to be initialized when they are declared?

Are fields assigned a default value if they are not explicitly initialized?

Variables

Variable Declaration

```
// declare variable w/ value
int speedLimit = 65;

// declare variable w/o value
int speedLimit;
speedLimit = 65;

// change value
speedLimit = 75;

// You can’t declare a variable twice
int speedLimit;

// And you can’t change the data type, once it’s been declared
float speedLimit;
```

Variables

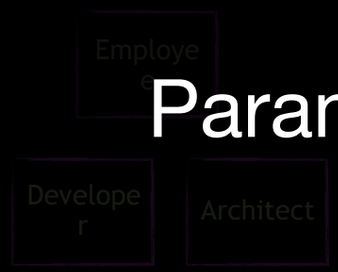
Kinds of Variables

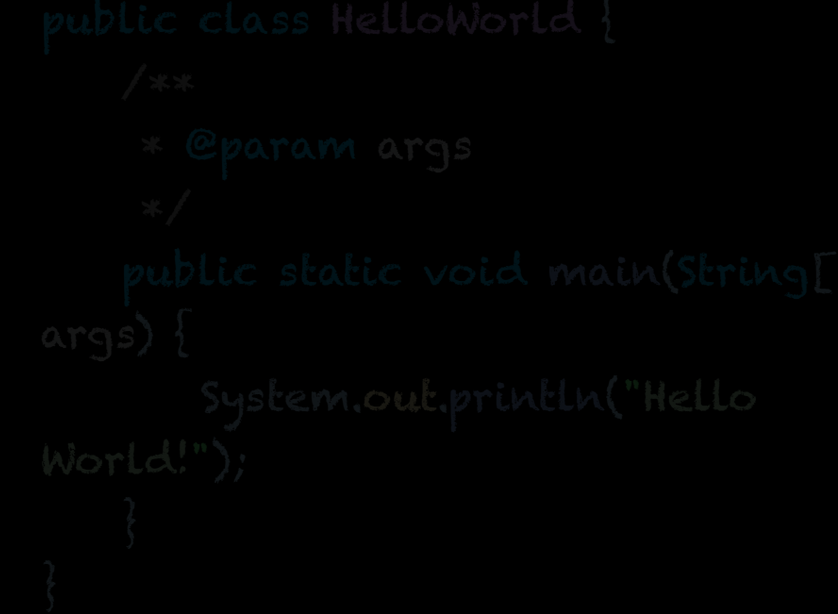
Instance Variables (Non-Static Fields)

Class Variables (Static Fields)

Local Variables

Parameters





Variables and Constants

A variable is a name for a memory location (address) that holds some piece of data. The value stored in that location may change during execution of the program; however, the type may not.

A constant is a name for a memory location (address) that holds some piece of data, where the value of the data cannot change during execution of the program.

Constants

Java does not have a unique keyword to define a constant. Constants are defined using the keywords static and final. For example

```
private static final int MAX_VALUE = 12;
```

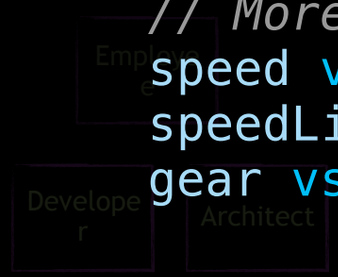
To make it easy to identify a constant in code, the name of a constant is written in all upper case letters. Use an underscore to separate words.

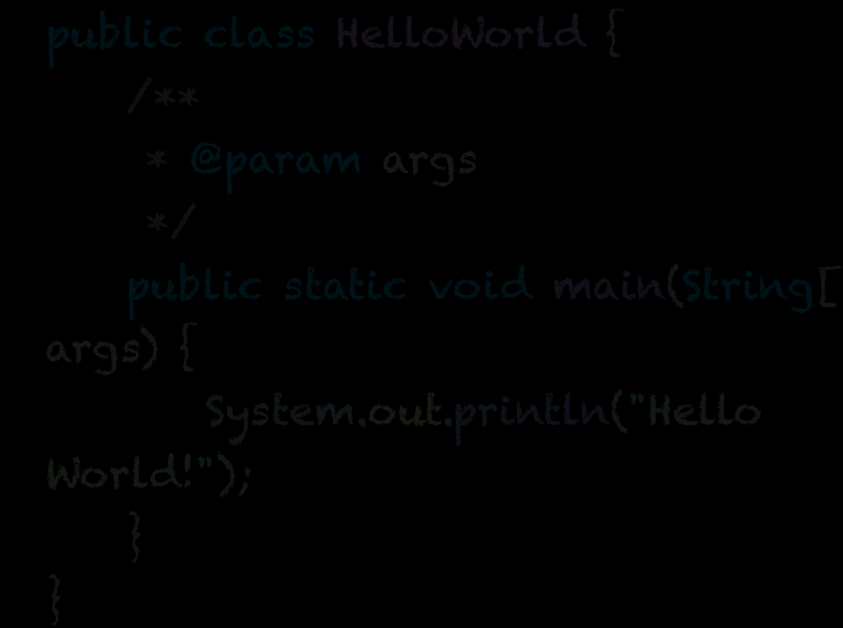
```
public class Constant {

    private static final int MAX_VALUE = 12;

    public static void main(String[] args) {
        System.out.format("Max Value:" + MAX_VALUE);
    }
}
```

Variables

Naming Rules

```
// Valid variable names
speed1
speedLimit
NUM_GEARS // For a constant value

// Valid but not recommended
_speed
$speed

// Invalid variable names
sp ed
spe”ed
1speed

// More Intuitive than abbreviate versions
speed vs s
speedLimit vs sl
gear vs g
```





Primitives

Programs deal with all kinds of data.
This data can be put into two broad categories.

- Primitive Data

  the most basic forms of data - numbers and characters

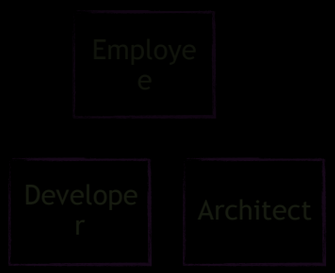
- Objects

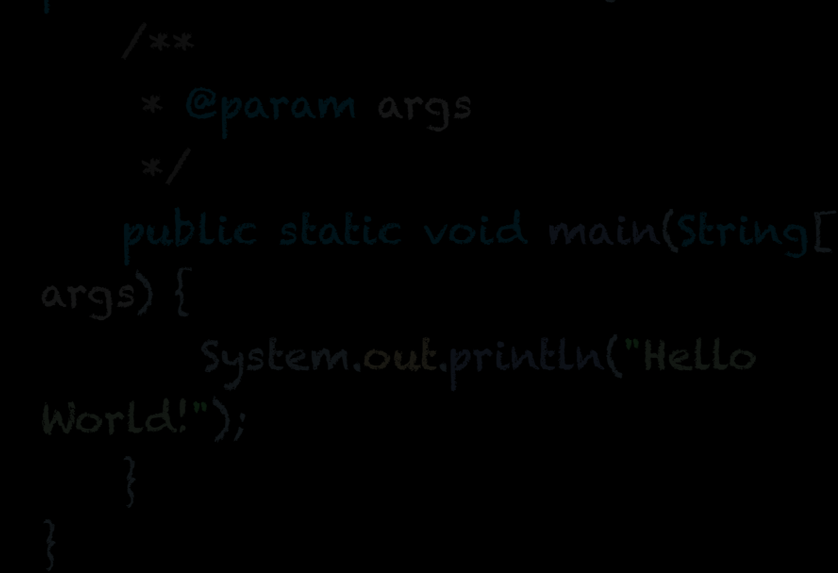
  more complex data – made up of many pieces of primitive data. For example, a person's address is usually made up of a house number, a street name, a city, state, and zip code.

Primitive Data Types

Java is a statically-type language

```
// doesn't have a declaration
speedLimit = 65;

// there we go, now it does
int speedLimit;
speedLimit = 65;
```

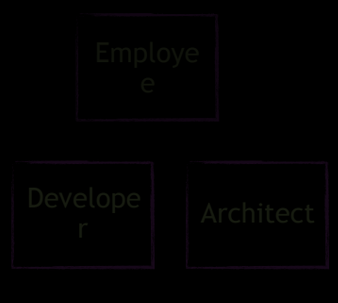


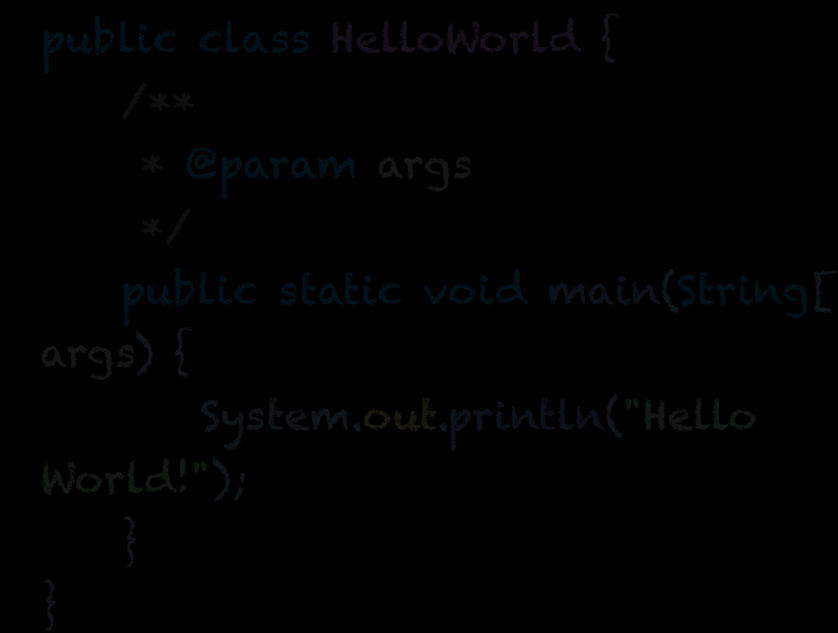


Primitive Data Types

8 of them

byte
short
int
long
double
float
boolean
char





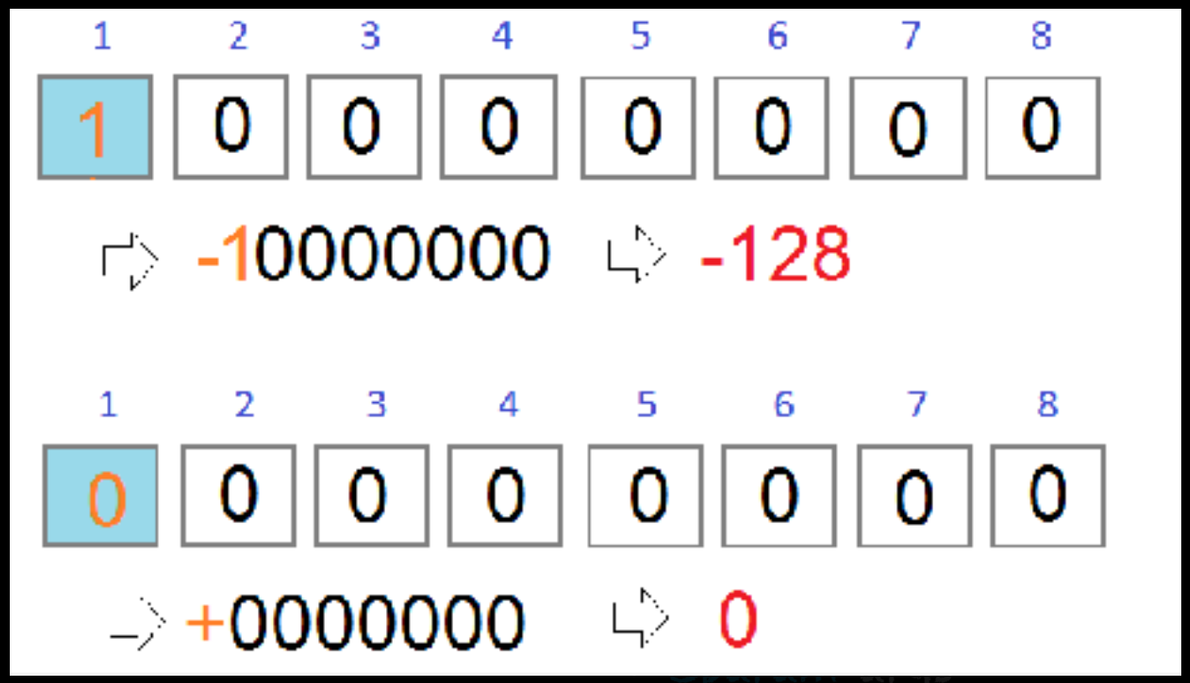
byte

8-bit signed two's complement integer

1 byte = 8 bits
signed = + or -

minimum value of -128
maximum value of 127
(inclusive)

Use it when memory savings actually manner.



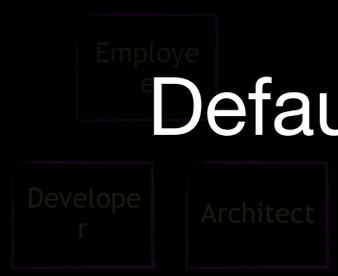
short

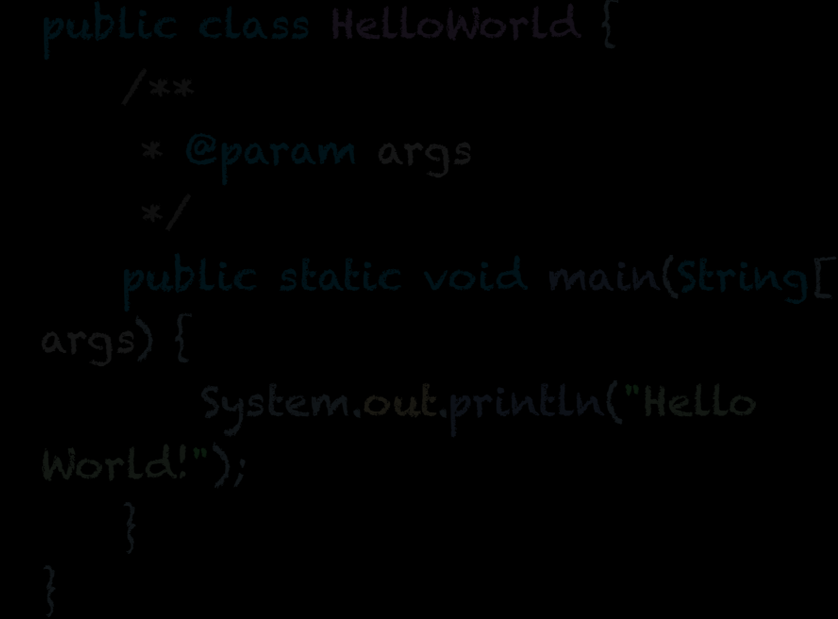
16-bit signed two's complement integer

The short data type can have values from -32768 to 32767.

It's used instead of other integer data types to save memory if it's certain that the value of the variable will be within [-32768, 32767].

Default value: 0





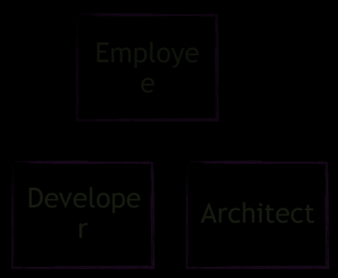
int

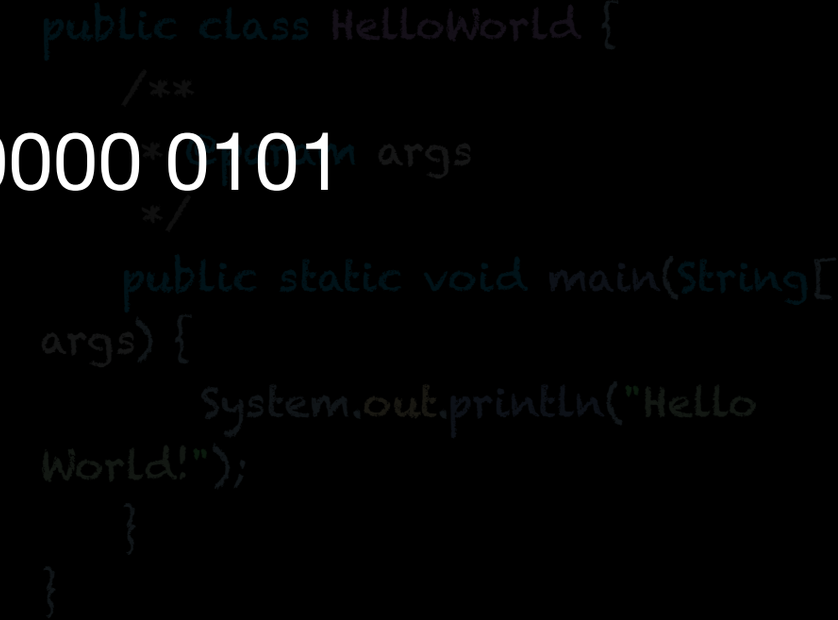
32-bit signed two's complement integer

In most modern digital computers, integer numbers are stored internally in binary. The number of bits used to store an integer in Java is 32 bits

Example: the integer 5 is

0000 0000 0000 0000 0000 0000 0000 0101

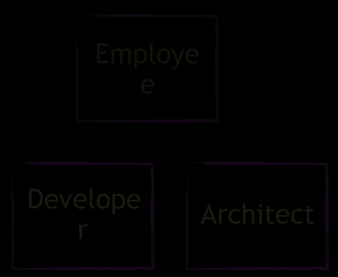


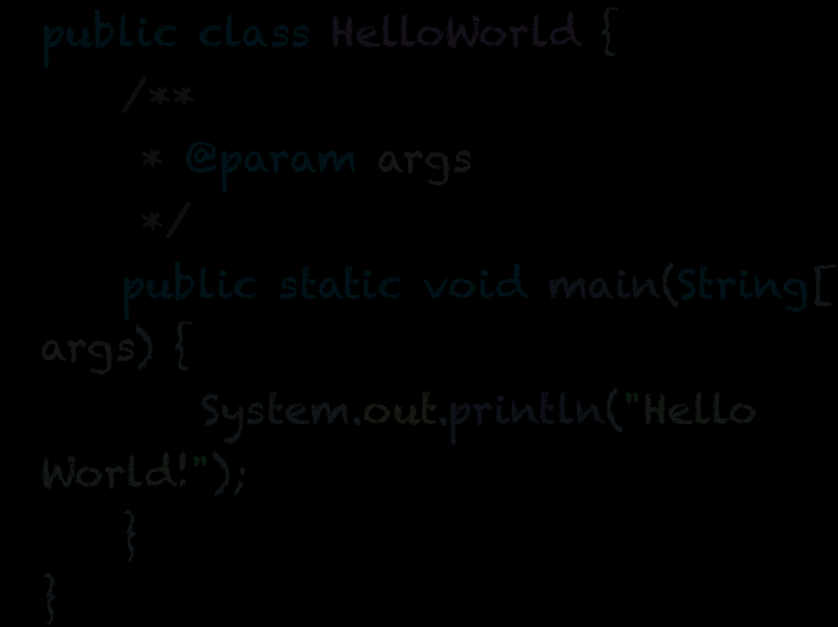


long

64-bit two's complement integer

minimum value of -2^{63} and a maximum value of $2^{63}-1$

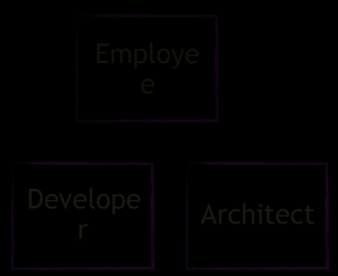




float

single-precision 32-bit IEEE 754 floating point

what are floats used for?

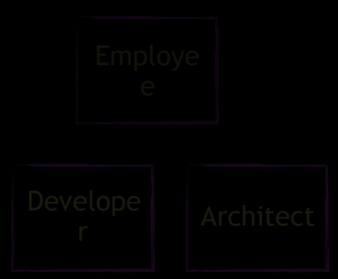


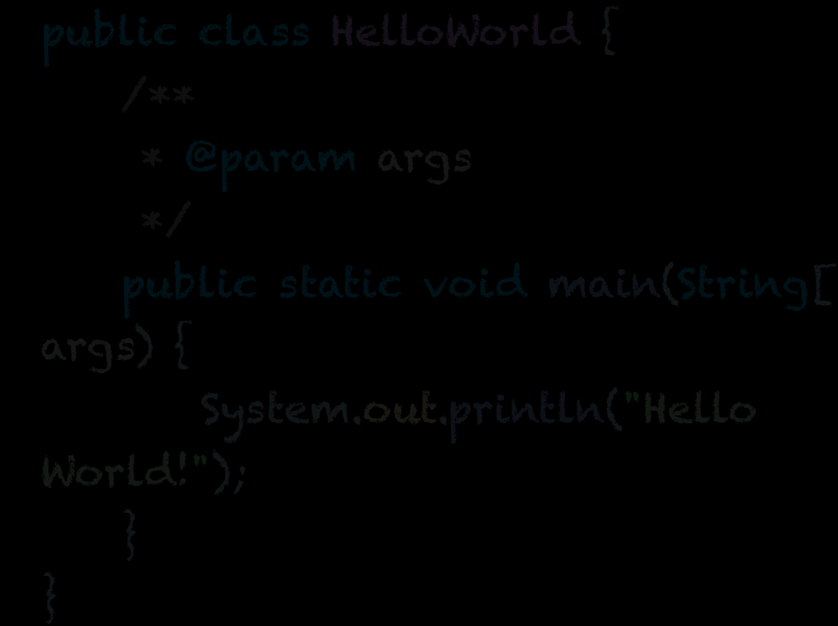

```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[
args) {
        System.out.println("Hello
World!");
    }
}
```

double

double-precision 64-bit IEEE 754 floating point

minimum value of -2^{63} and a maximum value of $2^{63}-1$



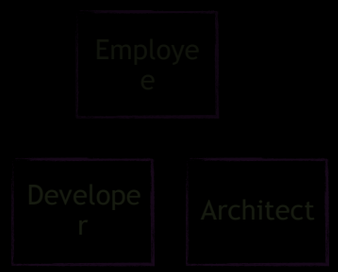


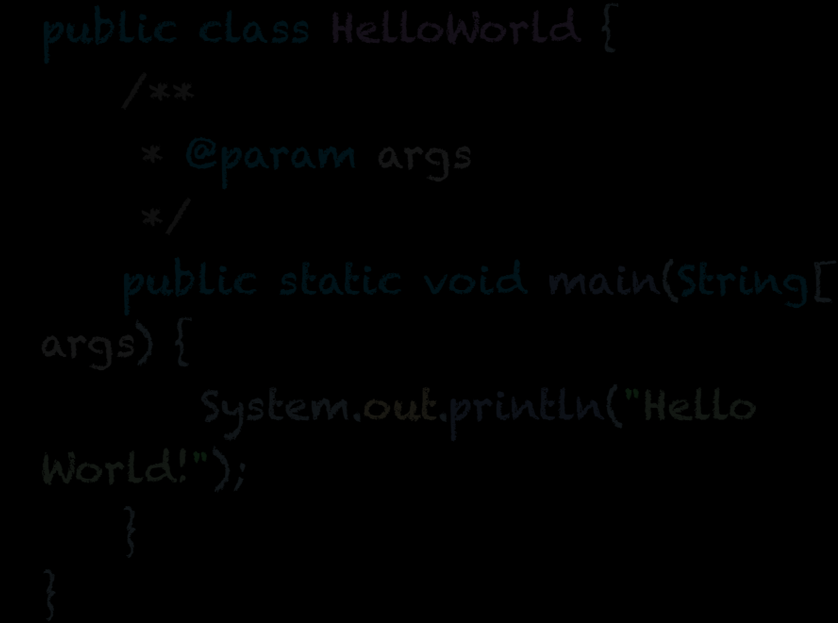
boolean

one bit

A piece of boolean data can only have one of two values:

true or false

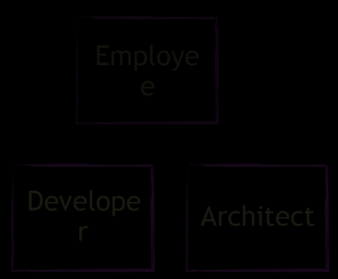


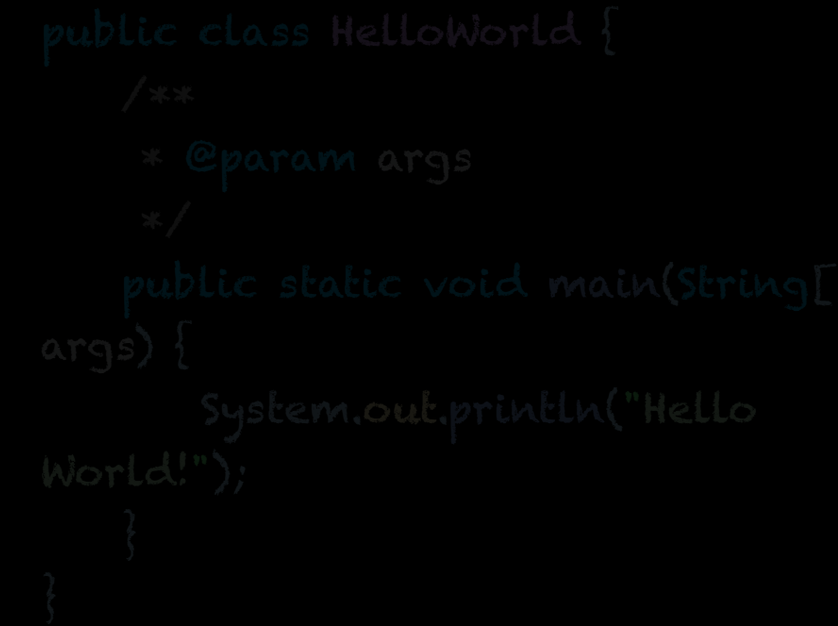


char

single 16-bit Unicode character

minimum value of -2^{63} and a maximum value of $2^{63}-1$





Java's primitive types

Ordered by descending upper range limit

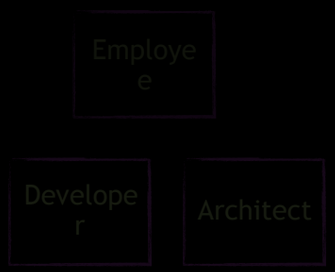
CATEGORIZATION	NAME	UPPER RANGE	LOWER RANGE	BITS
Floating point	double	Really huge positive	Really huge negative	64
	float	Huge positive	Huge negative	32
Integral	long	9,223,372,036,854,775,807	-9,223,372,036,854,775,808	64
	int	2,147,483,647	-2,147,483,648	32
	char	65,535	0	16
	short	32,767	-32,768	16
	byte	127	-128	8
Boolean	boolean	No numeric equivalent	true or false	1

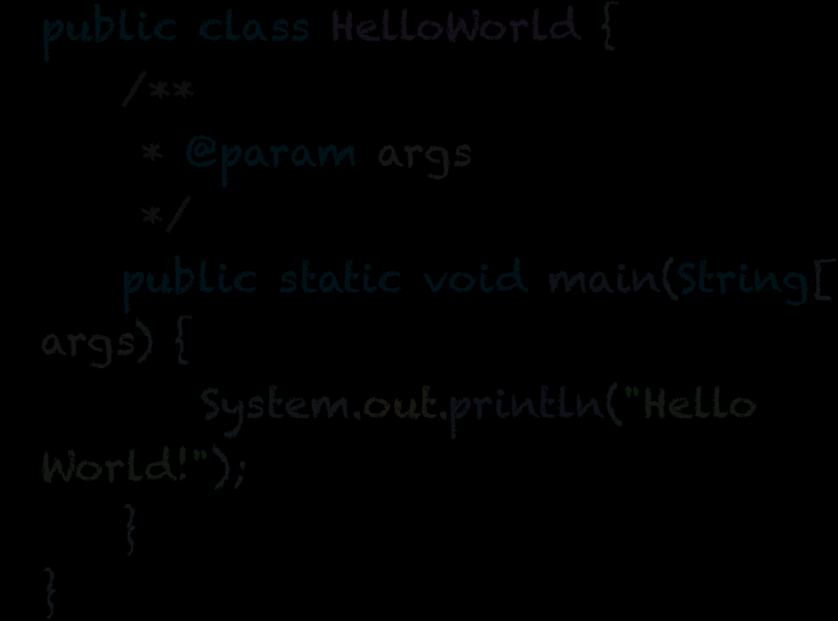
SOURCE: CAMERON MCKENZIE

java.lang.String

Title Text

```
String s = "this is a string";
```

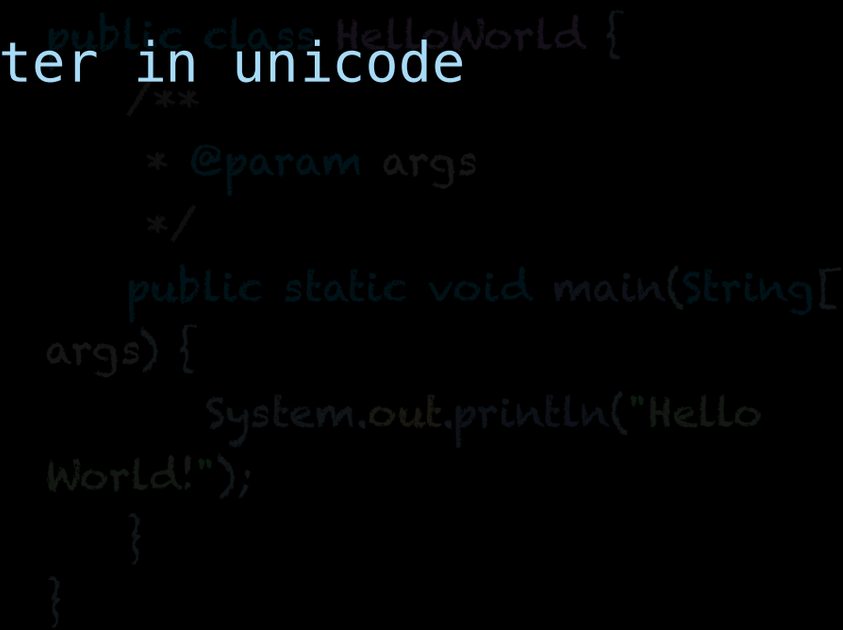




Default Values

When declaring uninitialized variables

```
int     = 0
byte    = 0
short   = 0
long    = 0
float   = 0.0
double  = 0.0
boolean = false
char    = ‘\u0000’ // which is the null character in unicode
string  = null
```



Default Values

Local Variables

Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable. If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it. Accessing an uninitialized local variable will result in a compile-time error.

```
public class LocalVariable {
    public static void main(final String[] args) {
        int speedLimit;
        System.out.println(speedLimit);
    }
}
```

```
LocalVariable.java:4: error: variable speedLimit might not have been
initialized
        System.out.println(speedLimit);
```

Literals

Literals

source code representation of a fixed value

```
boolean result = true;

char capitalC = 'C';

byte b = 100;

short s = 10000;

int i = 100000;
```

Integer Literals

int or long l,L

Decimal: Base 10, whose digits consists of the numbers 0 through 9; this is the number system you use every day

```
// The number 26, in decimal
int decVal = 26;
long longVal = 456L;
```

Hexadecimal: Base 16, whose digits consist of the numbers 0 through 9 and the letters A through F

```
//  The number 26, in hexadecimal
int hexVal = 0x1a;
```

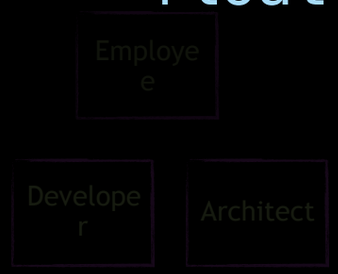
Binary: Base 2, whose digits consists of the numbers 0 and 1 (you can create binary literals in Java SE 7 and later)

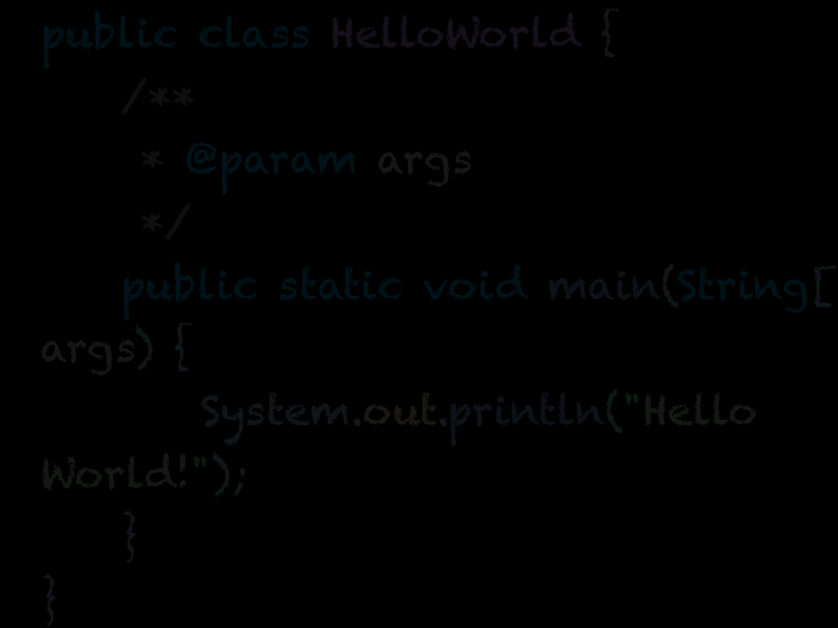
```
// The number 26, in binary
int binVal = 0b11010;
```

Floating-Point Literals

ends with f,F or d,D

```
double d1 = 123.4;

// same value as d1, but in scientific notation
double d2 = 1.234e2;

float f1  = 123.4f;
```





Character and String Literals

Any Unicode (UTF-16) characters

"Unicode escape”

```
'\u0108' (capital C with circumflex = Ĉ)
"S\u00ED Se\u00F1or" (Sí Señor in Spanish).
```

special escape sequences

```
\b (backspace)
\t (tab)
\n (line feed)
\f (form feed)
\r (carriage return)
\" (double quote)
\' (single quote)
\\ (backslash)
```

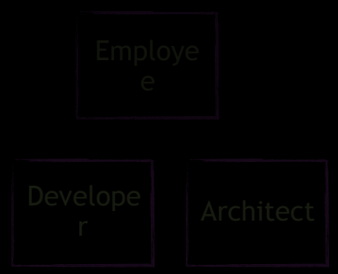
also

```
null
.class (String.class)
```

Numeric Literals

Underscore

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;s
```



Arrays

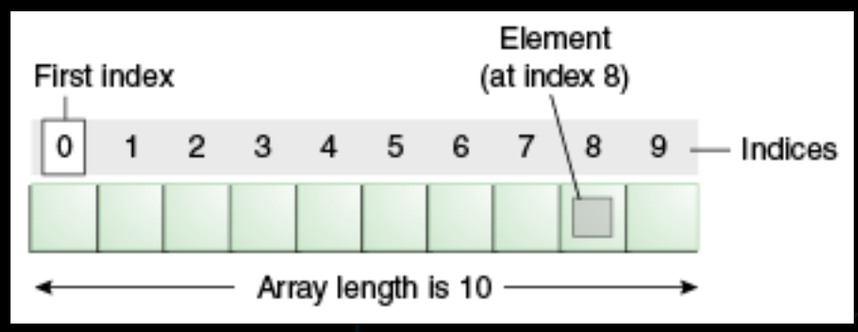
Arrays

Container Object

Declaring

Length

Index



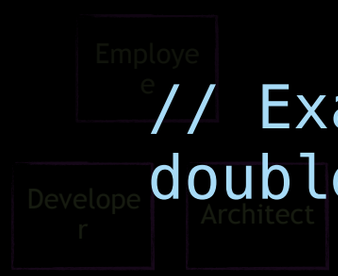
Java Arrays

Declaring an array

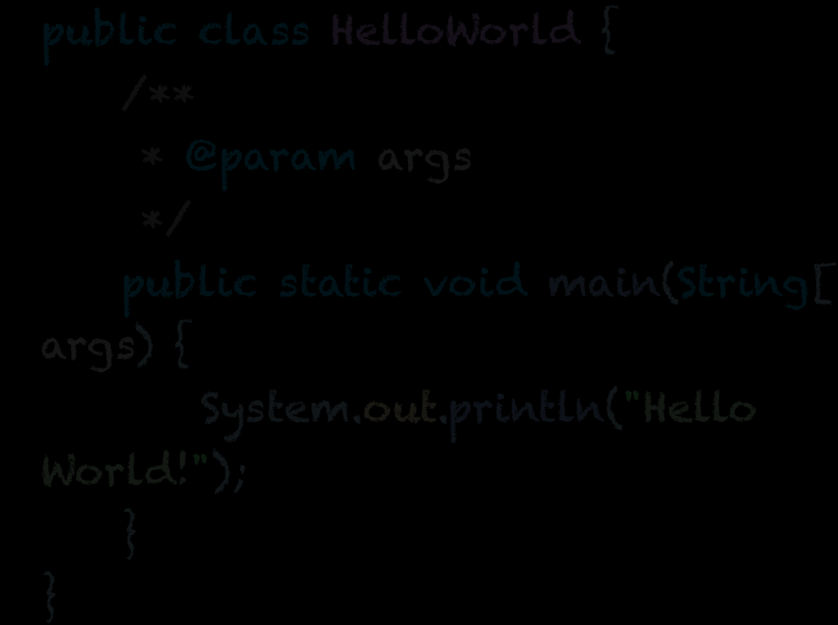
```
dataType[] arrayName;
```

dataType - it can be primitive data types like int, char, double, byte, etc. or Java objects

arrayName - it is an identifier




```
// Example
double[] data;
```



Java Arrays

How many elements can the array hold?

```
data = new Double[10];
```

Once the length of the array is defined, it cannot be changed in the program.

```
int[] age;
age = new int[5];
```

In a single statement

```
int[] age = new int[5];
```

Java Arrays

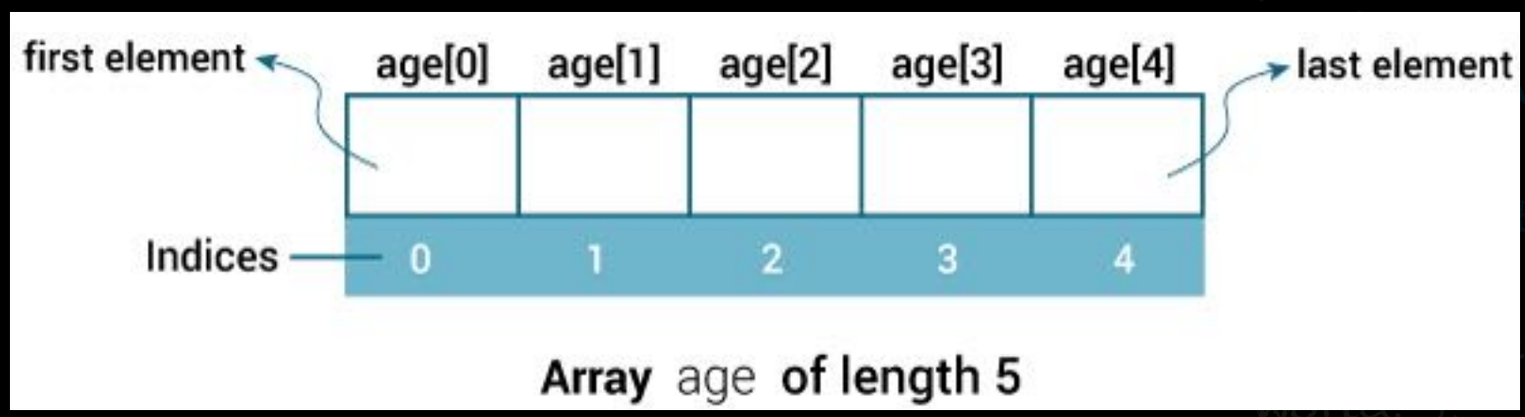
Array Index

Each element in an array is associated with a number. The number is known as an array index. We can access elements of an array by using those indices. For example,

```
int[] age = new int[5];
```

The array indices always start from 0.

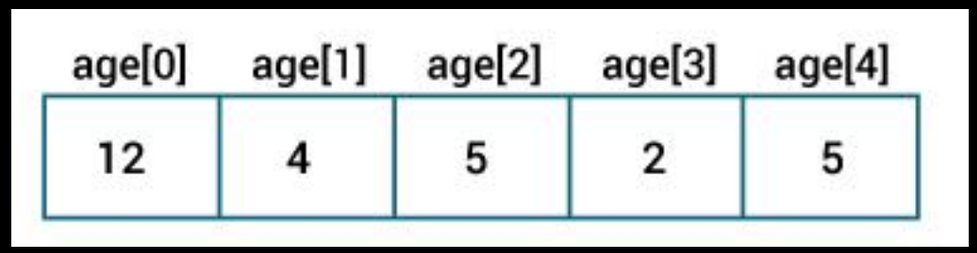
```
age[0]
age[1]
```



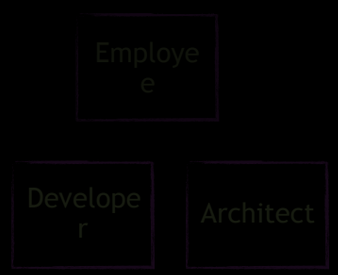
Java Arrays

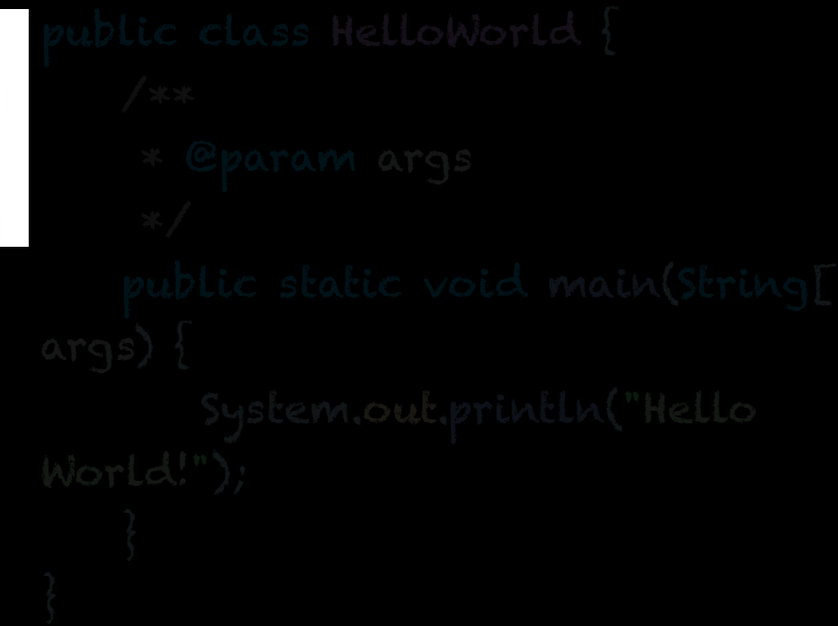
Initializing Arrays During Declaration

```
int[] age = {12, 4, 5, 2, 5};
```



age[0]	age[1]	age[2]	age[3]	age[4]
12	4	5	2	5

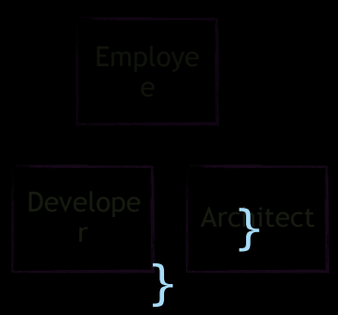




Java Arrays

Accessing Array Elements

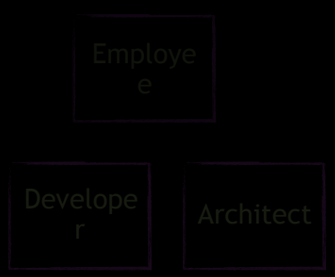
```
class ArrayExample {
    public static void main(String[] args) {

        int[] age = new int[5];

        // insert 14 to third element
        age[2] = 14;

        // insert 34 to first element
        age[0] = 34;

        System.out.println("Element at index 0: " + age[0]);
        System.out.println("Element at index 1: " + age[1]);
        System.out.println("Element at index 2: " + age[2]);
        System.out.println("Element at index 3: " + age[3]);
        System.out.println("Element at index 4: " + age[4]);
```

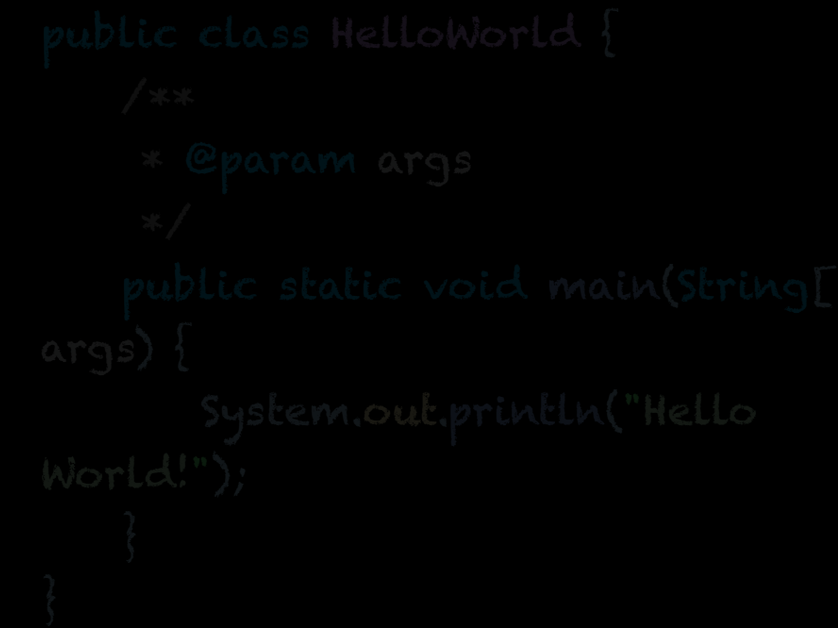


Java Arrays

Multidimensional

```
double[][] matrix = {{1.2, 3.4, 5.0}, {2.2, 3.1, 4.4}};
```



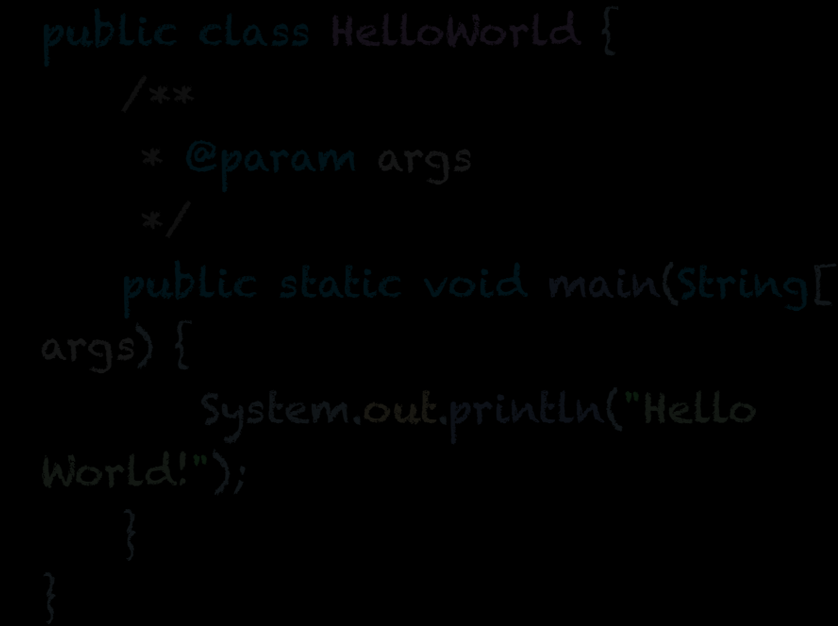


Code at Home

Try this at home

PrimitiveValues.java

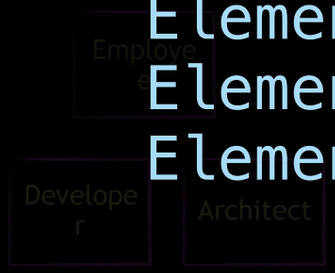
```
The output from this program is:
flag: true (boolean)
range: 129 (byte)
temperature: 200 (short)
range: -4250000 (int)
range: -42332200000 (long)
number: -42.3 (float)
number: -42.3 (double)
letter: Q (char)
name: <Your name> (string)
```

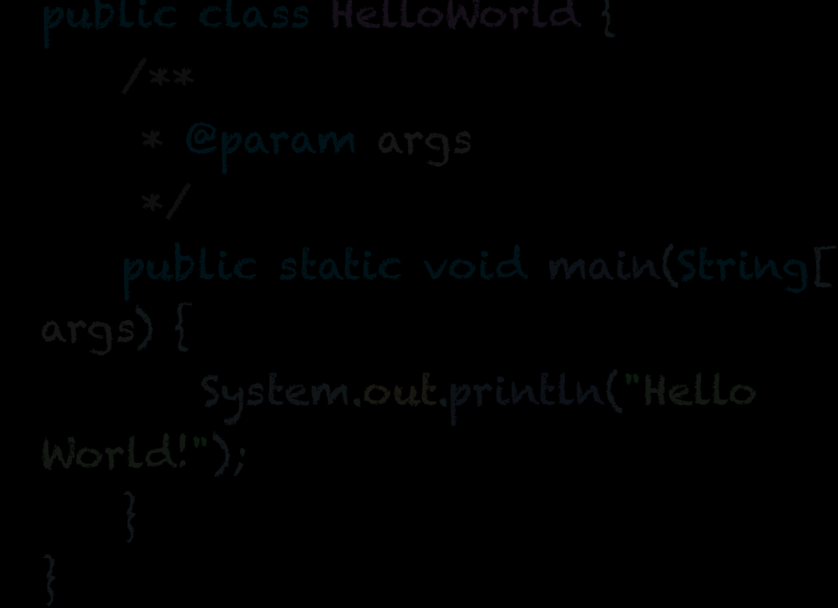


Try this at home

ArrayDemo.java

```
The output from this program is:
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```



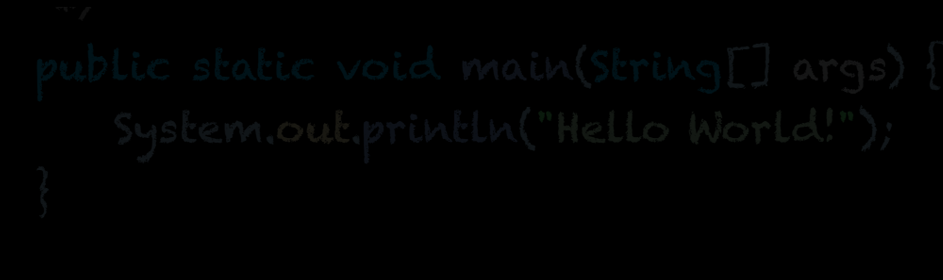


Recap

Variables

Summary

- “fields” and “variables” (terminology)
- Instance variables (non-static fields)
- Class variables (static fields)
- Local variables
- Parameters
- Naming Variables
- Primitive data types
  - byte, short, int, long, float, double, boolean, and char
- java.lang.String; character strings
- Array



The term "instance variable" is another name for ___.

The term "class variable" is another name for _____.

A local variable stores temporary state; it is declared inside a _____.

A variable declared within the opening and closing parenthesis of a method is called a _____.

What are the eight primitive data types supported by the Java programming language?

_____ _____ _____ _____ _____ _____ _____ _____

Character strings are represented by the class.

An _____ is a container object that holds a fixed number of values of a single type.

Assume that you are able to peek into the memory of your computer, and you see the bit pattern

0000 0000 0000 0000 0000 0000 0110 0010

What does this bit pattern mean?

Exercise:

Create fields for each data type leaving them uninitialized and print out their values.

PrimitivesUninitialized.java

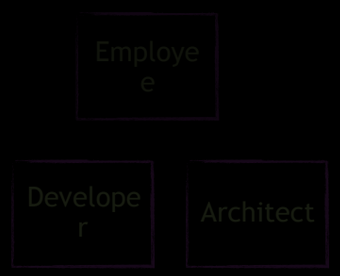
Operators

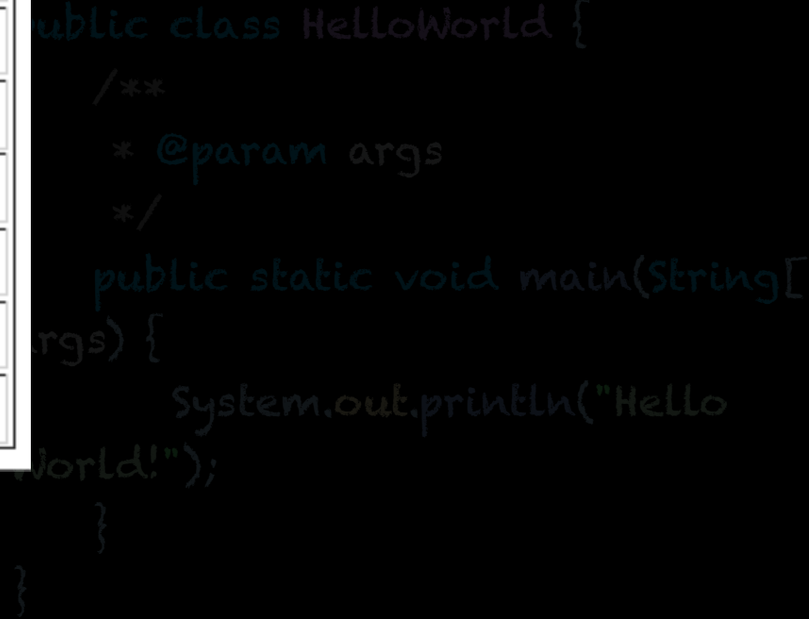
Operators

Doing something with variables

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

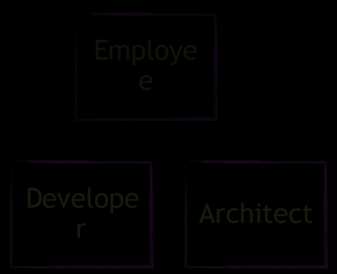


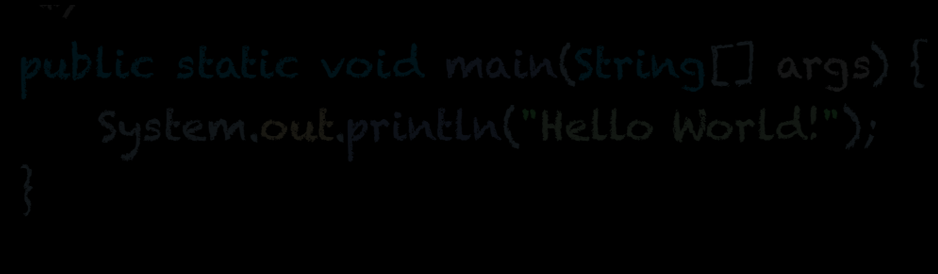


Operator Precedence

BODMAS

- **B**rackets
- **O**rder Of (2 power of 3 etc)
- **D**ivision
- **M**ultiplication
- **A**ddition
- **S**ubtraction





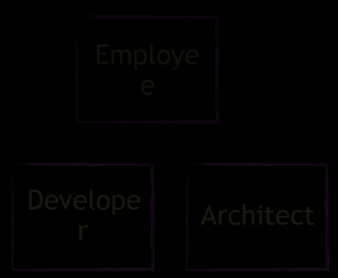
Assignment Operator

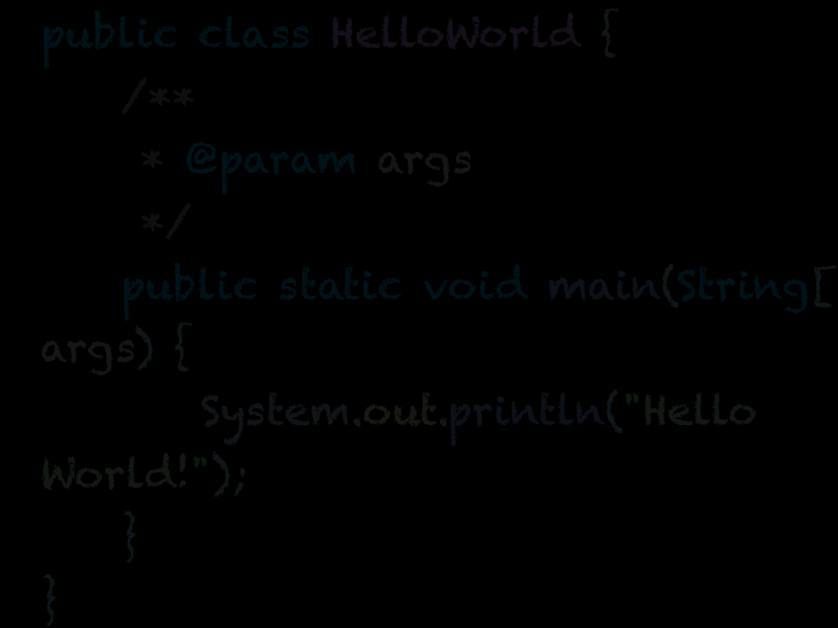
The Simple One

=

as in

```
int speed = 0;
```

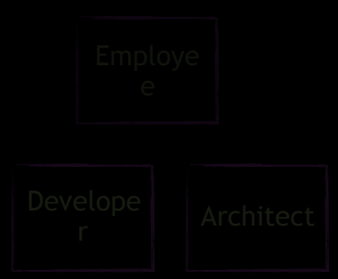


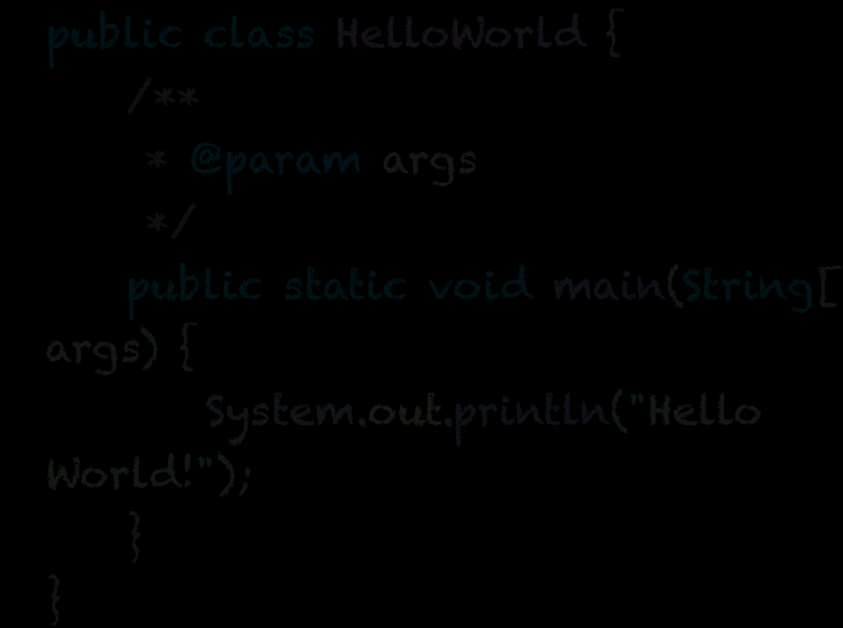


Arithmetic Operators

Basic Mathematics

```
// Additive (also used for String concatenation)
+

// Subtraction
-

// Multiplication
*

// Division
/

// Remainder
%
```





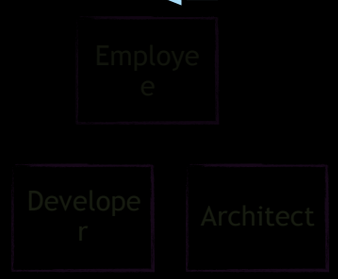
Unary Operators

Only require one operand

```
// Unary plus operator; indicates positive value
+

// Unary minus operator; negates an expression
-

// Increment operator; increments value by 1
++

// Decrement operator; decrements value by 1
--

// Logical complement operator; inverts the value of a boolean
!
```

Equality and Relational

```
==      equal to
!=      not equal to
>       greater than
>=      greater than or equal to
<       less than
<=      less than or equal to
```

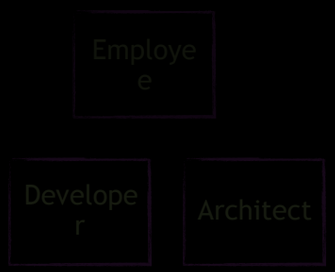



```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[
args) {
        System.out.println("Hello
World!");
    }
}
```

Conditional Operators

Title Text

```
&& Conditional-AND
|| Conditional-OR
?: Ternary Operator
```




```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[
args) {
        System.out.println("Hello
World!");
    }
}
```

Bitwise and Bit Shift Operators

Not Common

The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

The signed left shift operator "<<" shifts a bit pattern to the left, and the signed right shift operator ">>" shifts a bit pattern to the right. The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator ">>>" shifts a zero into the leftmost position, while the leftmost position after ">>" depends on sign extension.

The bitwise & operator performs a bitwise AND operation.

The bitwise ^ operator performs a bitwise exclusive OR operation.

The bitwise | operator performs a bitwise inclusive OR operation.

Recap

What operators does the code contain?

```
arrayOfInts[j] > arrayOfInts[j+1]
```

What are the values of `i` and `n` after the code is executed?

```
int i = 10;
int n = i++%5;
```

What are the final values of `i` and `n` if instead of using the postfix increment operator (`i++`), you use the prefix version (`++i`))?

```
int i = 10;
int n = ++i%5;
```

To invert the value of a `boolean`, which operator would you use?

Which operator is used to compare two values, = or == ?

Explain the following code sample:

```
result = someCondition ? value1 : value2;
```

Change the following program to use compound assignments:

```
class ArithmeticDemo {

    public static void main (String[] args){

        int result = 1 + 2; // result is now 3
        System.out.println(result);

        result = result - 1; // result is now 2
        System.out.println(result);

        result = result * 2; // result is now 4
        System.out.println(result);

        result = result / 2; // result is now 2
        System.out.println(result);

        result = result + 8; // result is now 10
        result = result % 7; // result is now 3
        System.out.println(result);

    }
}
```

In the following program, explain why the value "6" is printed twice in a row:

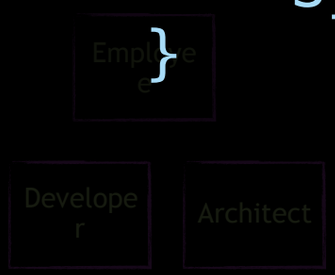
```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i);    // "4"
        ++i;
        System.out.println(i);    // "5"
        System.out.println(++i);  // "6"
        System.out.println(i++);  // "6"
        System.out.println(i);    // "7"
    }
}
```

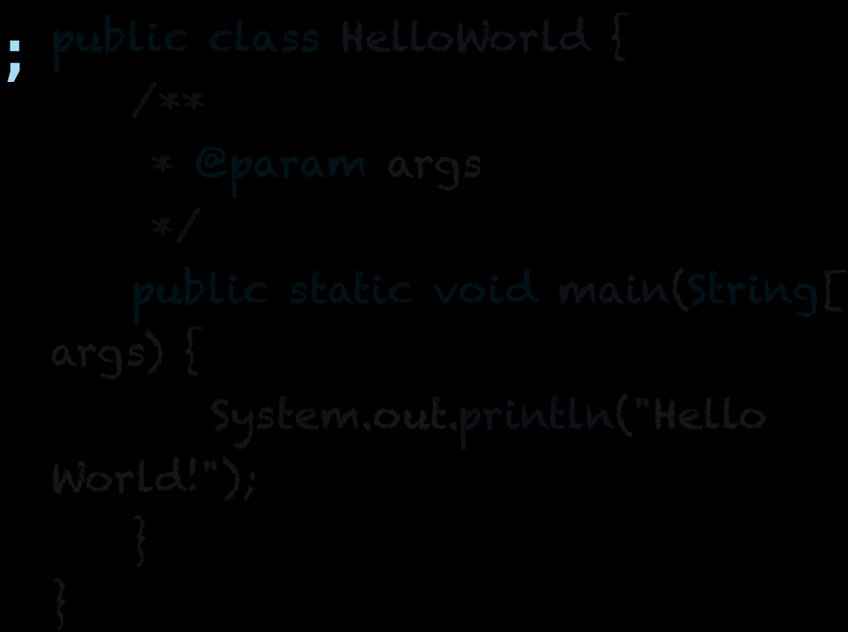
Expressions, Statements, Block

Expressions

core components of statements

```
int speedLimit = 0;
anArray[0] = 100;
System.out.println("Element 1 at index 0: " + anArray[0]);

int result = 1 + 2; // result is now 3
if (value1 == value2) {
    System.out.println("value1 == value2”);
}
```

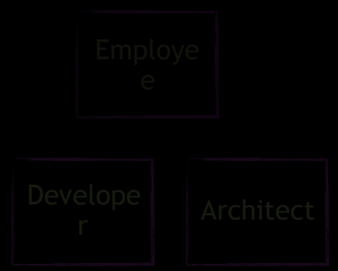




Expressions

Compound Expressions

```
1 * 2 * 3
```

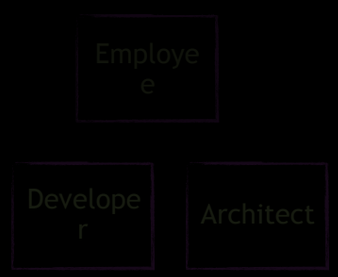


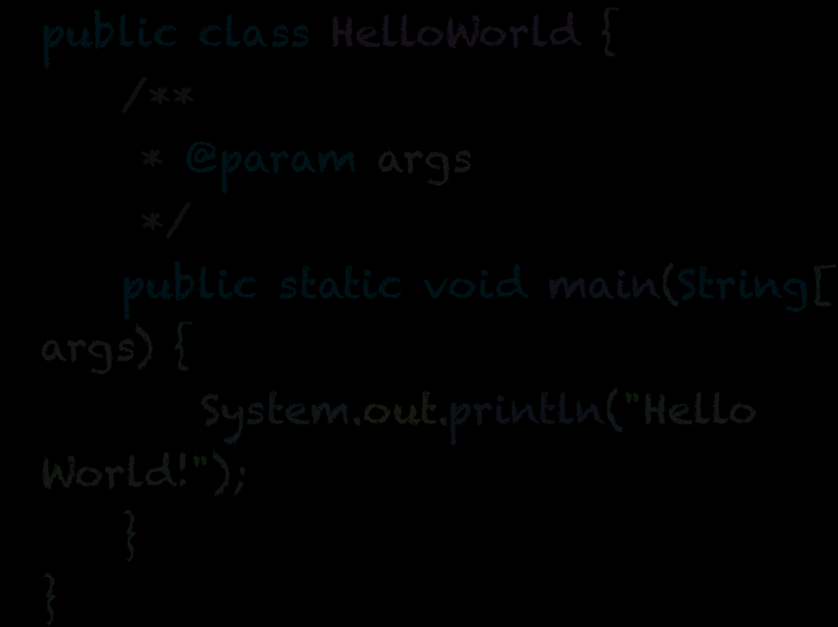

```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[
args) {
        System.out.println("Hello
World!");
    }
}
```

Expressions

Compound Expressions

```
x + y / 100    // ambiguous

(x + y) / 100  // unambiguous, recommended
```

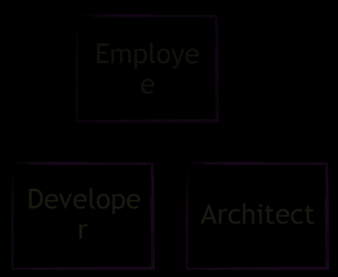


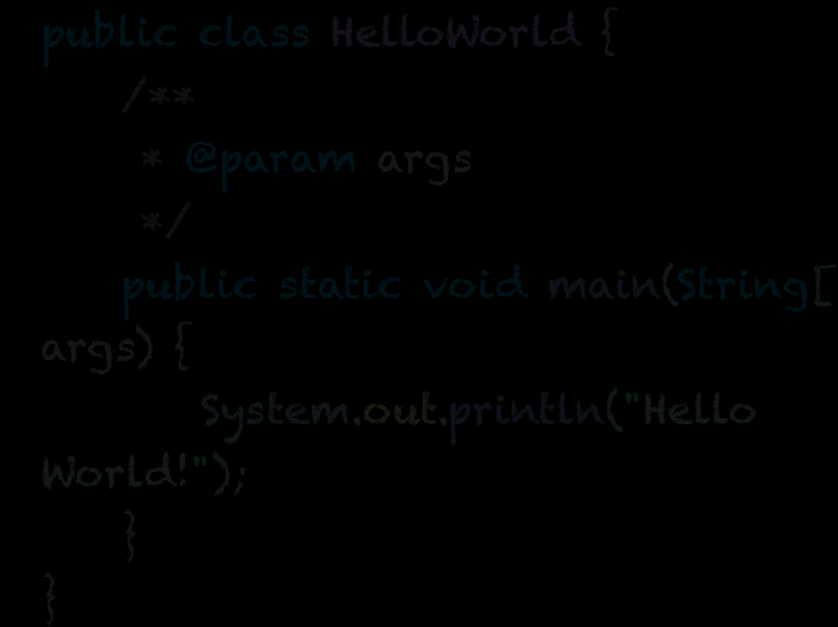


Expressions

Compound Expressions

```
x + y / 100    // ambiguous

x + (y / 100)  // unambiguous, recommended
```





Statements

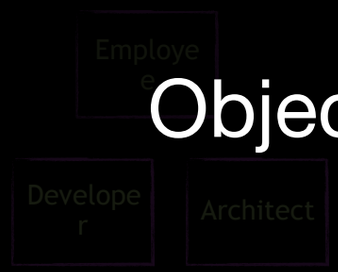
roughly equivalent to sentences in natural languages

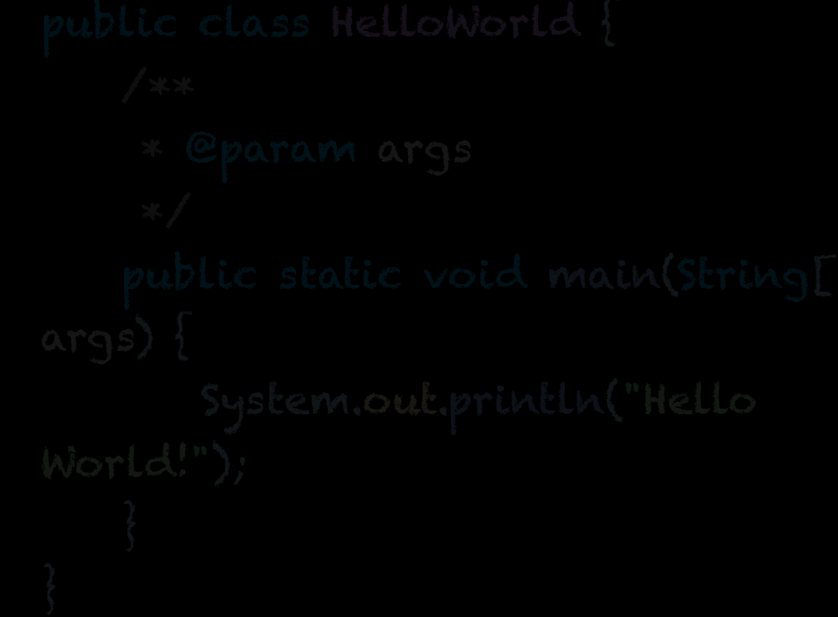
Assignment expressions

Any use of ++ or --

Method invocations

Object creation expressions

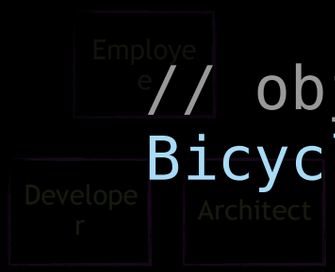


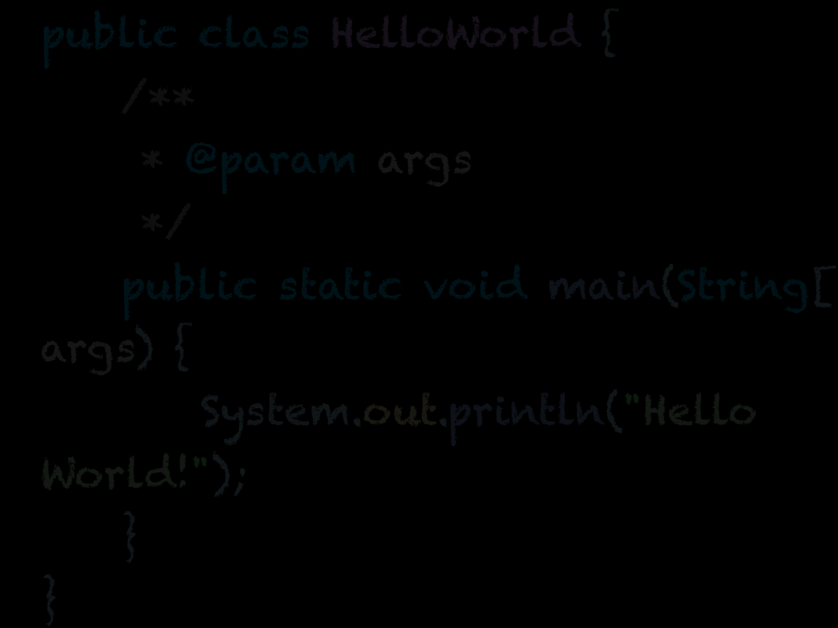


Statements

Title Text

```
// assignment statement
aValue = 8933.234;

// increment statement
aValue++;

// method invocation statement
System.out.println("Hello World!");

// object creation statement
Bicycle myBike = new Bicycle();
```

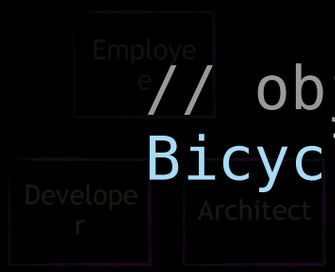


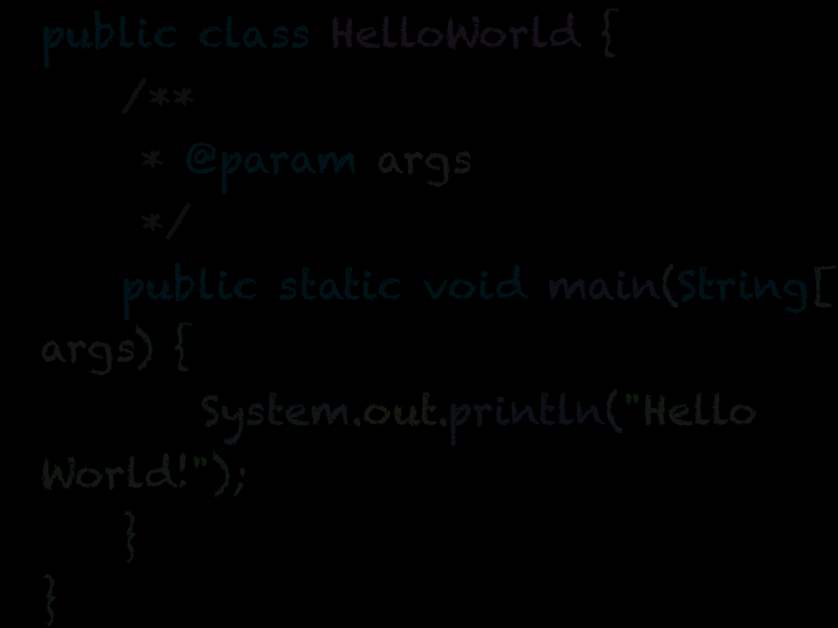


Statements

Title Text

```
// assignment statement
aValue = 8933.234;

// increment statement
aValue++;

// method invocation statement
System.out.println("Hello World!");

// object creation statement
Bicycle myBike = new Bicycle();
```

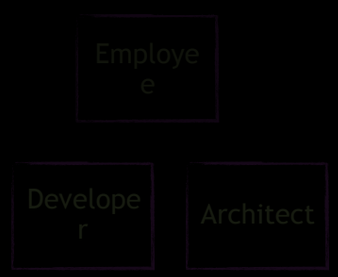


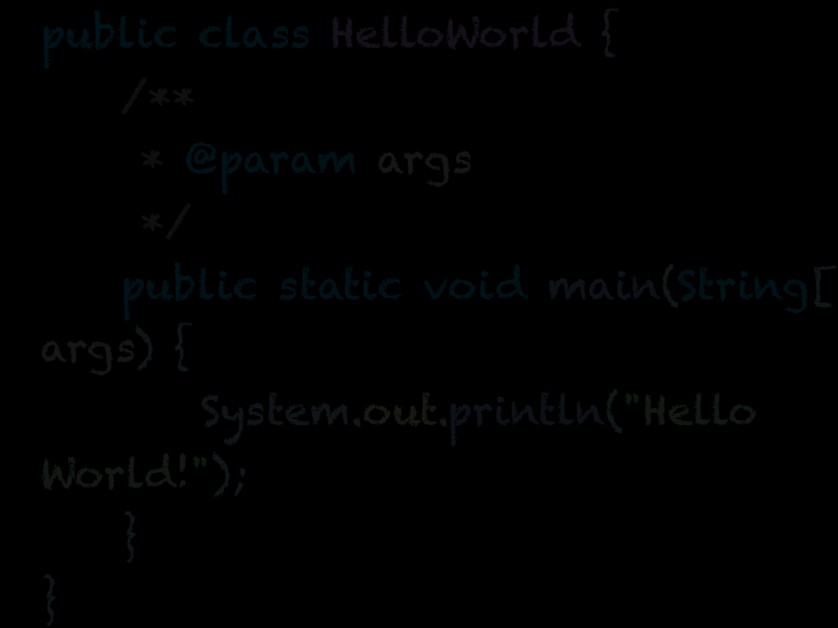


Statements

Title Text

```
// declaration statement
double aValue = 8933.234;

// control flow statements
if (true) {

}
```

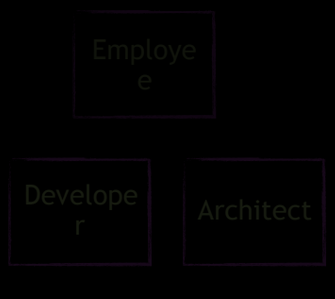




Blocks

braces

{ }

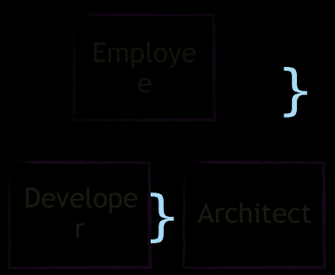


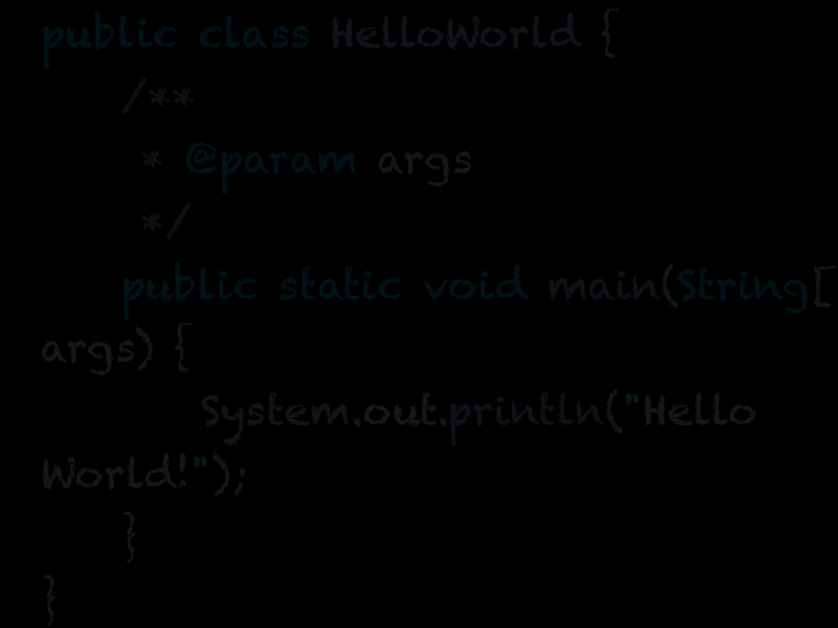

```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[
args) {
        System.out.println("Hello
World!");
    }
}
```

Blocks

braces

```
class BlockDemo {

    public static void main(String[] args) {
        boolean condition = true;

        if (condition) { // begin block 1
            System.out.println("Condition is true.");
        } // end block one
        else { // begin block 2
            System.out.println("Condition is false.");
        } // end block 2
```





Recap

Operators may be used in building _____, which compute values.

```
speedLimit = 65
```

Expressions are the core components of _____.

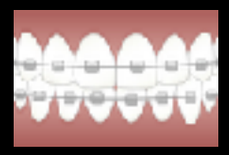
They are also roughly equivalent to sentences in natural languages;

Statements may be grouped into _____.

By using a set up these braces:

`{}`

`Not these:`



The following code snippet is an example
of a _____ expression,

```
1 * 2 * 3
```

from various smaller expressions.

Statements are roughly equivalent to sentences in natural languages, but instead of ending with a period, a statement ends with a _____.

A block is a group of zero or more statements between balanced _____ and can be used anywhere a single statement is allowed.

Identify the following kinds of expression statements:

```
// a________ statement
aValue = 8933.234;

// i_________ statement
aValue++;

// m______ i__________ statement
System.out.println("Hello World!”);

// o_____ c_______ statement
Bicycle myBike = new Bicycle();
```

Control Flow Statements

if-then and if-then-else

Basic

```
if (speed >= 65) {
    System.out.println(“Speeding”);
}
```

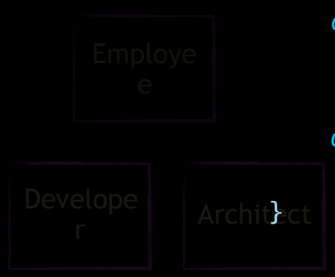
```
if (speed >= 65)
    System.out.println(“Speeding”);
```

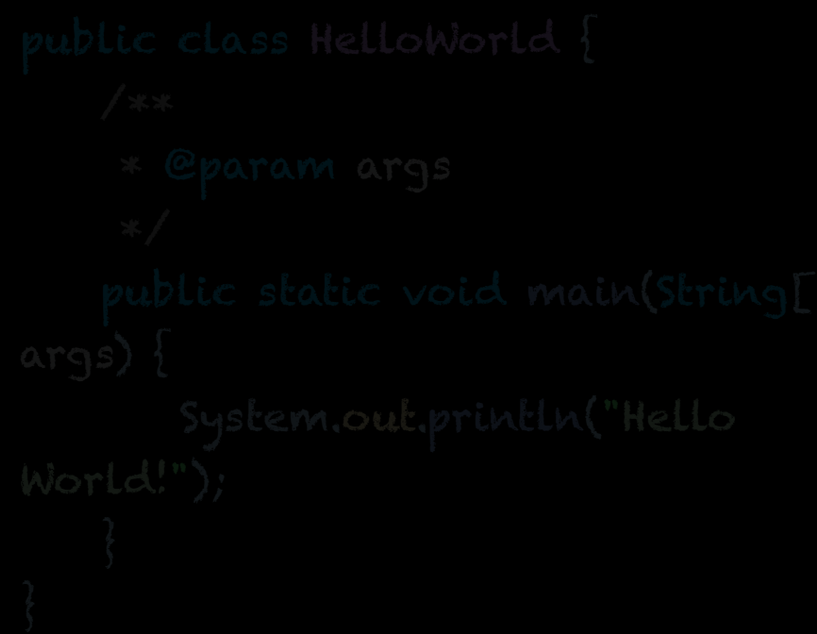
```
if (speed >= 65) {
    System.out.println(“Speeding”);
} else {
    System.out.println(“Not Speeding”);
}
```

```
if (speed >= 75) {
    System.out.println(“Speeding!”);
} else if (speed >= 65) {
    System.out.println(“Fast but not too fast...”);
} else {
    System.out.println(“Slow Driver!”);
}
```

switch

```java
public class SwitchDemoFallThrough {

    public static void main(String[] args) {

        int month = 3;
        String monthName = null;

        switch (month) {
            case 1:
                monthName = "January";
                break;
            case 2:
                monthName = "February";
                break;

            // Skipped a few months here to fit on slide!

            case 11:
                monthName = "November";
                break;

            case 12:
                monthName = "December";
                break;
            default:
                break;
        }

        System.out.println("Month Name: " + monthName);

    }
}
```





while and do-while

```
while (expression) {
    statement(s)
}
```

```
do {
    statement(s)
} while (expression)
```

Recap

The most basic control flow statement supported by the Java programming language is the _____ statement.

The _____ statement allows for any number of possible execution paths.

The _____ statement is similar to the `while` statement, but evaluates its expression at the _____ of the loop.

Question: How do you write an infinite loop using the `for` statement?

Question: How do you write an infinite loop using the `while` statement?

Code at Home

if-then-else

IfElseDemo.java

Exercises

```
// What output do you think the code will
produce if aNumber is 3?
```

```
if (aNumber >= 0)
    if (aNumber == 0)
        System.out.println("first string");
else
    System.out.println("second string");
System.out.println("third string");
```

```
// Use braces { and } to further clarify the
code and reduce the possibility of errors by
future maintainers of the code.
```

```
if (aNumber >= 0)
    if (aNumber == 0)
        System.out.println("first string");
else
    System.out.println("second string");
System.out.println("third string");
```

Objects and Classes

Just a little taste, more on it later

OBJECTS AND CLASSES

Object oriented languages give programmers the ability to model real-world objects.

Data

for example, a car has **attributes**

* it is black
* it has a 200 hp engine
* it has 2 doors
* it was built in 1943
* etc

Methods

it also has **behaviors**

* when you turn the key it starts
* when you press the brake it stops
* when you push the horn it beeps
* etc

object-oriented languages **encapsulate** the data and the methods that operate on that data into an **object**.

an object's methods (behaviors)
manage specific pieces
of data (attributes) inside the object.

methods outside of
the object cannot see
or manipulate the object's
data, which is **private**.
However, they can call
public methods inside
the object to access
the data.

Later on, we will spend much more time talking about objects and classes. For now, just think of a **class** as a **blueprint** that the computer uses when creating objects of that class. When we write an object oriented program, much of our time is devoted to designing and writing classes.

Java has many classes built into it that will make our programming tasks much easier. The first of these we will talk about is the **String** class.

A string is just a sequence of characters.

“hello”
“George”
“12 East State Road”

Declare a string this way:

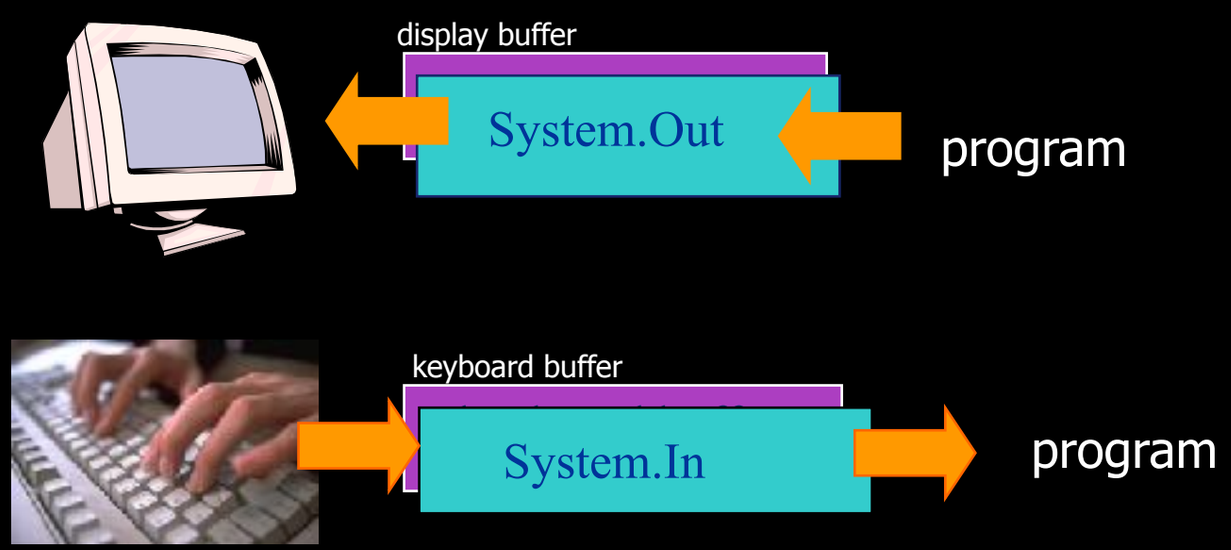
```
String myName;
```

Declare a string and give it an initial value this way:

String myName = “John Dolittle”;

Java Basic Input and Output

When a Console program executes, the Java runtime environment automatically creates these two stream objects to help manage Console input and output.



Java Input

The Scanner Class

The Scanner class in Java provides a nice way of getting input of various data types from the keyboard.

Create a Scanner Object

```
Scanner input = new Scanner(System.in);
```

This is the name of the Scanner object that gets created.

This part of the statement calls the "constructor" for the Scanner class. The parameter System.in tells the constructor that we want the Scanner to read input from the system console (the keyboard).

Scanner Methods

The Scanner class has lots of methods, but only a few of them will be of interest to us in this class. Tokens are strings of characters separated by white space.

nextInt()	returns the next token in the input stream as an integer. If the next token is not an integer, an exception is thrown.
nextDouble()	returns the next token in the input stream as a double. If the next token is not a double, an exception is thrown.
next()	returns the next token as a String object.
nextLine()	returns the next line, excluding any line separator at the end of the line.

```java
import java.util.Scanner;

public class ScannerTest {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        String name;
        name = in.nextLine();
        in.close();

        System.out.println(name);
    }
}
```

When using the Scanner class, be sure to
import java.util.Scanner;

in.nextInt()

When dealing with numbers, we have to use the correct method to read in the value of that data type.

For example:

```
Scanner in = new Scanner(System.in);
int age = 0;
age = in.nextInt();
```

in.nextDouble()

When dealing with numbers, we have to use the correct method to read in the value of that data type. For example:

```
Scanner in = new Scanner(System.in);
double weight = 0;
weight = in.nextDouble();
```

Java Output

Java Output

Title Text

In Java, you can simply use

```
System.out.println();
System.out.print();
System.out.printf();
```

to send output to standard output (screen).

```
// System is a class
// out is a public static field: it accepts output data.
```

Java Output

Difference between println(), print() and printf()

prints string inside the quotes.

`print()`

prints string inside the quotes similar like print() method. Then the cursor moves to the beginning of the next line.

`println()`

provides string formatting

`printf()`
`format()`

System.out.println()

System.out is an object of the PrintStream class.

The PrintStream class contains the println( ) method to write to the standard output device (the display console). This method takes a string as its parameter.

After writing to the display,
the cursor is moved to the next line.

System.out.println()

```
String name = “Joe Coder”;
System.out.println( name );

// Output
Joe Coder
```

System.out.println()

Numbers are automatically converted to strings by the println( ) method:

```
double money = 12.50
System.out.println(money);

// Output
12.50
```

System.out.format()

You can combine string literals and numerical data
using the format method. For example

```
double money = 12.50;
System.out.format("You owe me $%5.2f%n", money);
```

```
// Output
You own me $12.50
```

%5.2f is a format specifier.
It says to format the output as
a floating point number, in a field 5 digits
wide, with 2 decimal points of precision.

The format specifier %n says to generate a newline.

Format Specifiers

Format specifiers begin with a % symbol.
A % symbol followed by:

d – means display an integer value as a decimal number
f – means display a floating point value as a decimal number
s - means format any value as a string
n – outputs a platform specific line terminator

Formatting Strings

With an integer you can use a number to indicate how many digits to display

```
int number = 23;
System.out.format("The value is %4d", number);

// Output
// The value is   23
```

Formatting Strings

With a double you can use a number to indicate how many decimal digits to display

```
        double number = 23.98344;
        System.out.format("The value is %5.2f", number);

// Output
// The value is 23.98
```