Lecture 4

Syntax, keywords, variables and operators

Syntax = Rules of a programming languages

Algorithm and Pseudocode remains the same but your syntax changes for every programming language.

Syntax

Case Sensitivity

- Class names always start with an Uppercase letter
- Function names always start with lowercase letters
- Hello is not equal to hello

Comments

- int a = 5 // a is now equal to 5
- Single line comments - start with //
- Multiline comments - start with /* and end with /*
- /* This is a multiline
- comment */

Identifiers

Identifiers - Names in your program

- Class Name
- Function Name
- Variable Name

All identifiers should begin with a letter (A - Z, a - z), ampersand (&) or underscore (_)

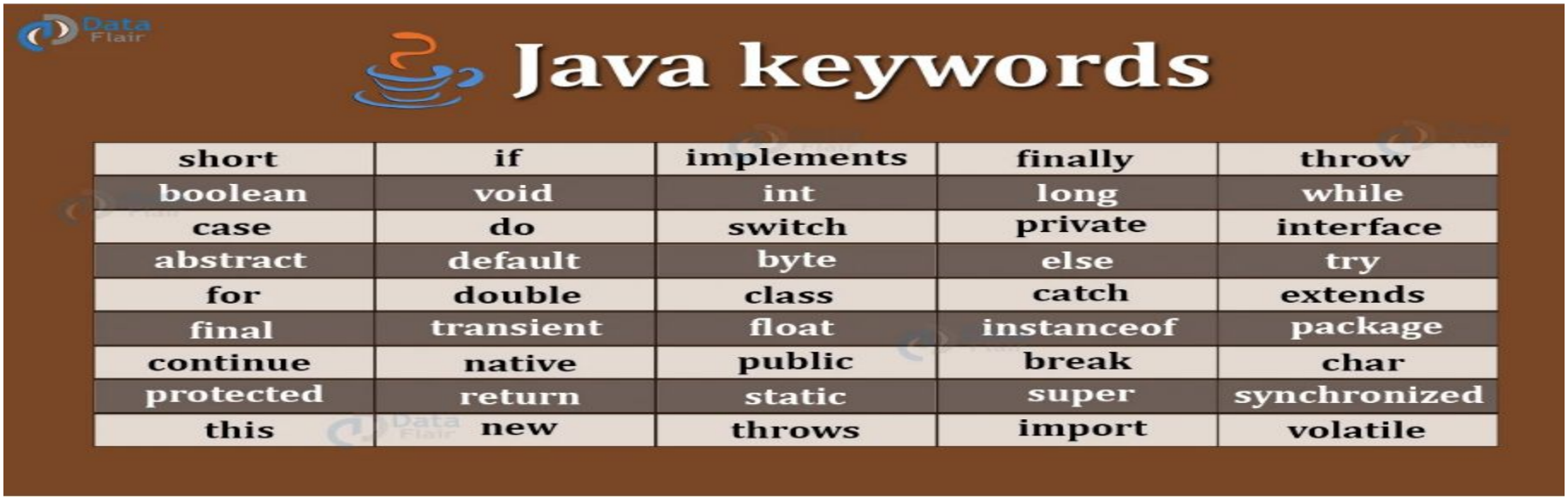
Identifiers are case sensitive

Hello vs hello

Keywords

Keywords - Reserved words which have special meaning in Java

public, static, void, for, while



Variables

Variables are containers which store some data value

- int a = 5
- float b = 19.9
- char c = ‘z’
- boolean d = true
- String e = “hello world”

Area of rectangle = L x B

L = 4,B = 2 Area = 8

L = 10, B = 5 Area = 50

Variables

Variable Declaration

- <variable type> <variable name>;
- int a;

Variable Assignment

- <variable name> = <some value>;
- a = 5;

You can do both in a single instruction (*Initialization*)

- <variable type> <variable name> = <some value>;
- int a = 5;

Variables

You can also assign the result of an expression to a variable

- int a = (5 * 4) + 5; // what's the value of a?
- right-hand-side(RHS) expression is evaluated first and is assigned to the variable

Variables

DO NOT REDEFINE/RE-DECLARE VARIABLES

```
class Main {
    public static void main(String[] args) {
      int a;
      int a; //not Ok
      int a=20; //not OK
    }
}
```

Datatypes

Datatype - Defines the type of data to be stored in a variable.

- Different datatypes have different storage size
- Primitive vs Non-Primitive data type
- **Primitive** - int, char, float, boolean, long
- **Non-Primitive** - String

```
class Main {
   public static void main(String args[]) {
       int a = 5;
       char b = 'Z';
       boolean c = true;
       String d =  "Hello World";

       System.out.println( "a is " + a);
       System.out.println( "b is " + b);
       System.out.println( "c is " + c);
       System.out.println( "d is " + d);

   }
}
```

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

String vs Character

String is a group of characters stored together

char a = ‘A’;

String b = “This stores multiple characters together”;

Java Operators

Operator : symbol that performs the specified operations

Arithmetic Operators

+

-

*

/

%

++ (a = a + 1….. ++a)

– (b = b - 1 ……. –b)

++a vs a++

Pre-Increment (++a) vs Post-Increment (a++)

Pre-Decrement (--a) vs Post-Decrement (a–)

Assignment Operator

= …… a = 5

+=..... a += 5…….. a = a + 5

-= ……. a -=5 …….. a = a - 5;

/=

%=

Bitwise Operators

Bitwise operators are used to performing the manipulation of individual bits of a number. They can be used with any integral type (char, short, int, etc.)

int a = 5 // This will store the binary value of 5 in a

Bitwise OR (|)

Returns bit-by-bit OR of input values

0 | 0 = 0

1 | 1 = 1

1 | 0 = 1

0 | 1 = 1

---

0 1 0 1 (5)

| 0 1 1 0 (6)

= 0 1 1 1 (7)

Bitwise AND (&)

Returns bit-by-bit AND of input values

0 & 0 = 0

1 & 1 = 1

1 & 0 = 0

0 & 1 = 0

0 1 0 1 (5)

& 0 1 1 0 (6)

= 0 1 0 0 (4)

1 1 0 0

1 0 1 0

1 0 0 0

Bitwise XOR (^)

Returns bit-by-bit XOR of input values - if bits are different return 1 else return 0

0 ^ 0 = 0

1 ^ 1 = 0

1 ^ 0 = 1

0 ^ 1 = 1

```
  0 1 0 1
^ 0 1 1 0
= 0 0 1 1
```

Bitwise Complement (~)

Reverses the bits - i.e turns 1 to 0 and 0 to 1

~ 0 = 1

~ 1 = 0

~ 0 1 0 1

= 1 0 1 0

Logical Operators

&& - Return true if all the conditions are true

|| - Return true if any one of the condition is true

(true && true && false) == false

(true && true && true) == true

(false && true && true) == false

(false || true || true) == true

(true || true || false) == true

(false || false || false) == false