Java created in 1995, owned by Oracle, open-source and free, close to C++ and C#

case-sensitive, strongly-typed

A screen shot of a computer code

Description automatically generatedEvery app begins with class name matching file name, so Main.java + public class Main {…}

terminal command ‘javac filename’ to compile code, ‘java filename’ to run code

Every LoC must be in a class, with classes starting with a capital letter

main(String[] args) function is required in every Java program

comment = // and /\* \*/, grouped with {…}, ends with ;

variables = [final] type name = value 🡪 int x = 0, y = 0, z = 0; == int x, y, z; x = y = z = 0; 🡪 final = read-only, cannot overwrite

primitive: boolean (1b), byte (1B), short (2B), int (4B), long (8B), float (4B), double (8B), char (2B)

append f/d for float/double to represent 6-7/15 decimal digits, can use e/E for scientific notation

type casting – assign value of one primitive type to another

widening = convert smaller type to larger 🡪 automatic

narrowing = convert larger type to smaller 🡪 (int) myDouble

non-primitive: String, Array, Class  
 value can be null

operators:

arithmetic: +, -, \*, /, %, ++, --

assignment: =, +=, -+, \*=, /=, %=, &=, |=, ^=, >>=, <<=

comparison: ==, !=, >, <, >=, <=

logical: &&, ||, !

bitwise: &, |, ^, >>, <<

escape sequences: \', \", \\, \n, \r, \t, \b, \f

conditional statements: if… else if… else, switch… case… break… default, (cond) ? true : false

loops: while (cond), do {…} while (cond), for (iter; cond; incr), for (type var : array)

keyword: break = jump out of loop, continue = jump one iteration

Arrays: type[] var = {…}; 🡪 can be multidimensional 🡪 returns void or datatype

Method/Function 🡪 params = declaration, arg = use

static = belongs to Main class, not an object of Main class 🡪 can be called without making object, but cannot access anything outside

overloading: multiple methods with same name but different type/num of params, usually doing the same thing

recursion: needs a halting condition

Class: template for Object, which has attributes/fields and methods

constructor: called when object is made, usually to initialize values 🡪 must match class name

access modifiers: classes = public, default 🡪 attribute, method, constructor = public, private, default, protected

non-access mod: classes = final, abstract 🡪 attr, meth, construct = final, static, abstract, transient, synchronized, volatile

abstract = accessible only when inherited, with abstract methods with no bodies

nesting: group related classes, created with outer.inner name = outer.new inner(); 🡪 inner can be privated/protected, static

Interface: group related methods with empty bodies 🡪 keyword implements to inherit (since Java doesn’t allow multiple inheritance)

Enum: “class” that groups constants = enum myEnum { ENUM1, ENUM2, … }

myEnum.values() to access constants

can have attr and meths, but are public, static, and final

can’t create objects or extend classes, but can implement interfaces

OOP:

Abstraction = show essential information only to user = use abstract classes or interfaces

Encapsulation = “sensitive” data hidden from users = declare private AND provide getters/setters

- control class attr and meths, class can be read- or write-only, code change wont affect others, better data security

Inheritance = code reusability = keyword extends creates subclass from superclass

- keyword final to prevent class inheritance

Polymorphism = code reusability = use inheritance for different tasks

Packages: group related classes to avoid name conflicts and to have maintainable code

Java API: included in Java Dev Env 🡪 <https://docs.oracle.com/javase/8/docs/api/> 🡪 use import package.

User-defined: create with keyword package myPackage 🡪 compile file, then compile package

package named with all-lowercase

root.mypack.myPackClass.java

java.util.Scanner = get user input class

nextLine(), nextX(): X = primitive num type

java.util.Collections = sort lists alphabetically or numerically

sort()

java.util.ArrayList = resizable array class (implements List 🡪 uses objects) auto-creates new array if need space (use for storing/accessing data)

add(), get(), set(), remove(), clear(), size()

java.util.LinkedList = containers with links class (implements List) that auto-links items (use for manipulating data)

addFirst(), addLast(), removeFirst(), removeLast(), getFirst(), getLast()

java.util.HashMap = stores items in key/value pair using objects

put(), remove(), clear(), size(), keySet(), .values()

java.util.HashSet() = HashMap with unique items

add(), contains(), remove(), clear(), size()

java.util.Iterator = object to loop through Collections

iterator(): constructor, next(), hasNext(), remove()

java.time = datetime API

LocalDate, LocalTime, LocalDateTime, DateTimeFormatter, now()

java.util.regex = text sequence for search pattern

Pattern, compile(): constructor, CASE\_INSENSITIVE, LITERAL, UNICODE\_CASE

[abc], [^abc], [0-9], |, ., ^, $, \d, \s, \b, \uxxxx, n+, n\*, n?, n{x}, n{x, y}, n{x, }

Matcher, matcher(): constructor, find(), PatternSyntaxException

java.io.File = File handling

canRead(), canWrite(), createNewFile(), delete(), exists(), getName(), getAbsolutePath(), length(), list(), mkdir()

use java.io.IOException to handle errors

use java.io.FileWriter with write(), close() to write to file

use java.util.Scanner to read files

Wrapper Class: allow primitive data to be represented as objects

xValue(): x = primitive, toString()

Exceptions: throwing an error 🡪 use try { … } catch() {…} finally {…}

Use throw statement to create custom error

Thread: to use, extend Thread OR implement Runnable, then override run()

to run, call start() to run

to avoid concurrency problems, use isAlive()

Lambda Exp: unnamed functions that can be implemented in method body = anon functions

syntax: (params) -> exp OR code block

System, Math

Reserved Keywords - https://www.w3schools.com/java/java\_ref\_keywords.asp