Java Cheat sheet

plus main topics

Most Used Methods in Java.

System.out.println() → Prints output to the console.
String.length() → Returns the length of a string.
String.charAt() → Returns the character at a specified index in a string.
String.substring() → Returns a substring of a string.
String.split() → Splits a string into an array of substrings based on a specified delimiter.
String.valueOf() → Returns the string representation of the specified value.
Math.abs() → Returns the absolute value of a number.
Math.max() → Returns the maximum value between two numbers.
Math.min() → Returns the minimum value between two numbers.
Math.random() → Generates a random number between 0 and 1.
Integer.parseInt() → Converts a string representation of a number into an integer.
Double.parseDouble() → Converts a string representation of a number into a double.
ArrayList.add() → Adds an element to an ArrayList.
ArrayList.get() → Retrieves an element from an ArrayList at a specified index.
ArrayList.size() → Returns the size (number of elements) of an ArrayList.
ArrayList.remove() → Removes an element from an ArrayList.
Array.sort() → Sorts an array in ascending order.
Arrays.toString() → Converts an array to a string representation.
Scanner.nextLine() → Reads a line of input from the user.
Scanner.nextInt() → Reads an integer input from the user.

More on: https://docs.oracle.com/javase/8/docs/api/allclasses-frame.html

About Java:

- Simple: Java has a simple syntax and design, making it easy to learn and use.
- Object-Oriented: Java is designed around the concept of objects, allowing for modular and reusable code.
- Portable: Java code can be compiled once and run on any platform that supports Java.
- Platform-Independent: Java code is compiled into bytecode, which can be run on any platform that has a Java Virtual Machine (JVM) installed.
- Secure: Java provides a sandboxed environment for executing code, preventing malicious code from harming the system.
- Robust: Java has features like automatic memory management (garbage collection) and exception handling that make it less prone to errors and crashes.
- High-Performance: Java is designed to be efficient and optimized for performance.
- Multithreaded: Java supports multithreading, allowing for concurrent execution of code.

Primitive Data Types:
A primitive data type specifies the size and type of variable values, and it has no additional methods.

- byte: 8-bit signed two's complement integer (-128 to 127)
- short: 16-bit signed two's complement integer (-32,768 to 32,767)
- int: 32-bit signed two's complement integer (-2,147,483,648 to 2,147,483,647)
- long: 64-bit signed two's complement integer (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
- float: 32-bit IEEE 754 floating-point number
- double: 64-bit IEEE 754 floating-point number
- boolean: true/false values
- char: single 16-bit Unicode character (0 to 65,535)

Non-primitive Data Types:
Non-primitive data types are called reference types because they refer to objects.
The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

Operators:

- Arithmetic: +, -, *, /, %
- Comparison: ==, !=, >, <, >=, <=
- Logical: &&, ||, !
- Bitwise: &, |, ^, ~, <<, >>, >>>
- Assignment: =, +=, -= etc.

Variables:

- Declaration: type name; or type name = value;
- Assignment: name = value; or name += value;
- Constants: final type NAME = value;

Control Structures:

- if statement: if (condition) { code block; } else { code block; }
- switch statement: switch (expression) { case value: code block; break; }
- for loop: for (initialization; condition; increment) { code block; }
- while loop: while (condition) { code block; }
- do-while loop: do { code block; } while (condition);
- break statement: break; or break label;
- continue statement: continue; or continue label;

Arrays:
Objects, data structure that stores a fixed-size sequential collection of elements of the same type. Each element in the array is identified by its index, which is an integer value starting from zero.

- Declaration: type[] name; or type[] name = new type[length];
- Accessing elements: name[index];
- Length: name.length;

Class: definition.
A class in Java is a template for creating objects that share common properties and behavior. It encapsulates related data and methods into a single unit, making the code more organized, reusable, and easier to maintain.

Classes:
- Declaration: class Name { fields; constructors; methods; }
- Fields: access type name;
- Constructors: public Name(parameters) { code block; }
- Methods: access return_type name(parameters) { code block; }
- Inheritance: class Child extends Parent { code block; }
- Polymorphism: Parent p = new Child();

Access Modifiers:
- public: accessible to all classes in all packages.
- protected: accessible to the class itself, its subclasses, and other classes in the same package.
- default (package-private): accessible to the class itself and other classes in the same package.
- private: accessible only within the class itself.

Objects:
- An object is an instance of a class, created using the new keyword.
- Objects have state (fields) and behavior (methods).
- To access an object's fields or methods, use the dot operator (.).
- Objects are created on the heap and accessed through references.

Heap vs Stack:
- The heap is a region of memory used for dynamic memory allocation (e.g. creating objects) and is shared among all threads in a Java program.
- The stack is a region of memory used for method calls and local variable storage and is private to each thread in a Java program.
- Objects are stored on the heap and accessed through references.
- Primitive data types and method arguments/return values are stored on the stack.

Error vs Exception:
- Errors are problems that mainly occur due to the lack of system resources. It cannot be caught or handled. It occurs at run time. These are always unchecked. An example of errors is OutOfMemoryError, LinkageError, AssertionError, etc.
- Exception is an event that occurs during the execution of the program and interrupts the normal flow of program instructions. These are the errors that occur at compile time and run time. It can be recovered by using the try-catch block and throws keyword. There are two types of exceptions i.e. checked and unchecked.
- When an error is detected, an exception is thrown.
- Any exception that is thrown must be caught by the exception handler.
- If the programmer has forgotten to provide an exception handler, the exception will be caught by the catch-all exception handler provided by the system.
- Exception may be rethrown if exception handler is failure to handle it.

Exceptions:

- Handling: try { code block; } catch (Exception e) { code block; } finally { code block; }
- Throwing: throw new Exception("message");
- Declaring: void method() throws Exception { code block; }

Runtime vs Compile Time:

- Compile-time refers to the period during which the code is compiled (translated from source code to machine code).
- Runtime refers to the period during which the code is executed.
- Compile-time errors occur during the compilation of a program (e.g. syntax errors), preventing the program from being compiled successfully.
- Runtime errors occur during the execution of a program (e.g. NullPointerException), causing the program to terminate abnormally.
- Java is a compiled language, but also has a just-in-time (JIT) compiler that can optimize code at runtime.

Java Comments:

- Single-line comments start with two forward slashes (//).
- Multi-line comments start with /* and ends with */.

JVM, JRE, and JDK:

- JVM (Java Virtual Machine) is an abstract machine that provides a runtime environment for Java code.
- JRE (Java Runtime Environment) is a set of tools that provide a runtime environment for Java code, including the JVM and libraries.
- JDK (Java Development Kit) is a software development kit that includes the JRE, compiler, debugger, and other development tools for creating Java applications.
- The JVM interprets compiled Java code (bytecode) and executes it on the host machine.

Multithreading in Java is a process of executing multiple threads simultaneously.
A **thread** is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Keywords:

- static: used to declare a member as belonging to the class rather than an instance of the class.
- final: used to declare a variable or method as immutable (cannot be changed).
- abstract: used to declare a class or method as incomplete and intended to be subclassed or implemented by a concrete class.
- interface: used to declare a type that defines a set of abstract methods and constants that can be implemented by classes.
- extends: used to declare a subclass that inherits from a superclass.
- implements: used to declare a class that implements an interface.
- this: used to refer to the current object.
- super: used to refer to the superclass of the current object.

Constructor:
In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
It is a special type of method which is used to initialize the object.
Every time an object is created using the new() keyword, at least one constructor is called.
It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Compiler:
The compiler is a program that converts the high-level language to machine level code. The Java programming language uses the compiler named javac. It converts the high-level language code into machine code (bytecode). JIT is a part of the JVM that optimizes the performance of the application. JIT stands for Java-In-Time Compiler.

JIT compiler advantages:
- It requires less memory usages.
- The code optimization is done at run time.
- It uses different levels of optimization.
- It reduces the page faults.

Method Overloading:
If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

Method Overriding:
If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.
Method overriding is used for runtime polymorphism.

Polymorphism:
Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.
There are two types of polymorphism in Java: **compile-time polymorphism** and **runtime polymorphism**. We can perform polymorphism in java by method overloading and method overriding.
Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

Garbage Collector:
In java, garbage means unreferenced objects.
Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Java Lambda:
A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.
Ex:
parameter → expression
(parameter1, parameter2) → expression
(parameter1, parameter2) → { code block }

Encapsulation:
The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as private
- provide public get and set methods to access and update the value of a private variable

Why Encapsulation?

- Better control of class attributes and methods
- Class attributes can be made read-only (if you only use the get method), or write-only (if you only use the set method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

Java Packages & API
A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API, a library of prewritten classes, that are free to use, included in the Java Development Environment. The library is divided into packages and classes. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package. To use a class or a package from the library, you need to use the **import** keyword)
- User-defined Packages (create your own packages; to create a package, use the **package** keyword)

Abstraction in detail:
Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces.
The abstract keyword is a non-access modifier, used for classes and methods:

- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods

Interfaces in detail:
An interface is a completely "abstract class" that is used to group related methods with empty bodies.
To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class

Why And When To Use Abstract Classes and Methods?
To achieve security - hide certain details and only show the important details of an object.

Notes on Interfaces:

- Like abstract classes, interfaces cannot be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?
1) To achieve security - hide certain details and only show the important details of an object (interface).
2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces. Note: To implement multiple interfaces, separate them with a comma (see example below).

More on:

- https://www.w3schools.com/java/java_ref_keywords.asp
- https://docs.oracle.com/javase/8/docs/api/allclasses-frame.html
- https://www.w3schools.com/java/java_ref_string.asp
- https://www.w3schools.com/java/java_ref_math.asp