Cheat Sheet

Java

Java Cheat Sheet

Java Ultimate Cheat Sheet

🔷 Variables and Data Types

- 🔤 **String** : Text data
- 🔢 **int** : Whole numbers
- 🔢 **long** : Larger whole numbers
- 🔢 **double** : Decimal numbers
- 🔢 **float** : Smaller decimal numbers
- 🅾️ **boolean** : True or False
- 🔡 **char** : Single character
- 🔝 **byte** : Small numbers (-128 to 127)
- 📅 **Date** : Date and time

🔷 Operators

- ➕ **+** : Addition
- ➖ **-** : Subtraction
- ✖️ ***** : Multiplication
- ➗ **/** : Division
- 📈 **++** : Increment
- 📉 **--** : Decrement
- 👬 **&&** : And
- 🌗 **||** : Or
- 🚫 **!** : Not
- ⏪ **<** : Less than
- ⏩ **>** : Greater than
- ⏪= **<=** : Less than or equal to
- ⏩= **>=** : Greater than or equal to

Java Ultimate Cheat Sheet

🔷 Conditional Statements

- 🔄 `if` : Execute code if a condition is true
- ⏭️ `else` : Execute code if the condition is false
- 🔁 `else if` : Execute code if another condition is true
- 📊 `switch` : Execute code based on the value of a variable
- 🗑️ `case` : Define a case in a switch statement

🔷 Loops

- 🔁 `for` : Execute code a fixed number of times
- 🔁 `while` : Execute code while a condition is true
- 🔁 `do while` : Execute code at least once, then while a condition is true
- 🛑 `break` : Exit a loop

🔷 Arrays

- 🧮 `int[] myArray = new int[5];` : Declare an integer array
- 🧮 `String[] myArray = {"Apple", "Banana", "Orange"};` : Declare a string array
- 📊 `myArray.length` : Get the length of an array
- 🔤 `myArray[0]` : Access the first element of an array
- ✏️ `myArray[0] = 10;` : Set the first element of an array

Java Ultimate Cheat Sheet

🔷 Classes and Objects

├ 🏭 *public class MyClass {}* : **Declare a class**

├ 🧬 *public class MyClass extends MyParentClass {}* : **Inherit from a parent class**

├ 🎨 *public static void main(String[] args) {}* : **Declare a main method**

├ 📁 *private String name;* : **Declare a private instance variable**

├ 🏷️ *public void setName(String name) { this.name = name; }* : **Declare a public method**

├ 🏷️ *public String getName() { return this.name; }* : **Declare a method to get the value of an instance variable.**

├ 🏭 *MyClass myObject = new MyClass();* : **Create an object**

├ 📤 *myObject.setName("John");* : **Call a method on an object**

├ 📥 *String name = myObject.getName();* : **Get the value of an instance variable**

├ 🏷️ *public static void myStaticMethod() {}* : **Declare a static method**

├ 📥 *MyClass.myStaticMethod();* : **Call a static method**

├ 📥 *MyClass.myStaticVariable;* : **Access a static variable**

├ 🎁 *MyClass myObject = null;* : **Declare a null object**

└ 🌎 *System.out.println("Hello, World!");* : **Print a message to the console**

🔷 Inheritance and Polymorphism

├ 🧬 *public class MyChildClass extends MyParentClass {}* : **Inherit from a parent class**

├ 🍩 *super();* : **Call a parent constructor**

├ 🎨 *@Override* : **Override a method from a parent class**

├ 🏭 *MyParentClass myObject = new MyChildClass();* : **Create a child object and assign it to a parent variable**

├ 📜 *instanceof* : **Check if an object is an instance of a class**

└ 👨‍💻 *MyInterface myObject = new MyImplementation();* : **Implement an interface**

Java Ultimate Cheat Sheet

🔷 Interfaces

├ 🏭 *public interface MyInterface {}* : **Declare an interface**

├ 🏭 *public class MyImplementation implements MyInterface {}* : **Implement an interface**

├ 🏭 *public class MyImplementation extends MyParentClass implements MyInterface {}* : **Implement an interface and inherit from a parent class**

└ 🎨 *@Override* : **Override a method from an interface**

🔷 Exception Handling

├ 🕹️ *try {}* : **Try block**

├ 🚨 *catch (Exception e) {}* : **Catch block**

├ ⏭️ *finally {}* : **Finally block**

├ 🔍 *throw new Exception("Something went wrong.");* : **Throw an exception**

└ 🔍 *throws Exception* : **Declare a method that can throw an exception**

🔷 Generics

├ 🕹️ *ArrayList<String> myArrayList = new ArrayList<String>();* : *Declare a generic ArrayList*

├ 🎁 *public <T> T myMethod(T argument) {}* : **Declare a generic method**

└ 🎁 *public <T extends MyInterface> T myMethod(T argument) {}* : **Declare a bounded generic method**

Java Ultimate Cheat Sheet

🔷 Streams

├ 📥 `List<String> myList = Arrays.asList("Apple", "Banana", "Orange");` : **Create a list**

├ 🧰 `.stream()` : **Create a stream**

├ 🔍 `.filter(s -> s.startsWith("A"))` : **Filter elements**

├ 🔄 `.map(s -> s.toUpperCase())` : **Transform elements**

├ 📊 `.count()` : **Get the number of elements**

├ 📚 `.collect(Collectors.toList())` : **Collect elements into a list**

└ 📥 `int sum = myList.stream().mapToInt(Integer::parseInt).sum();` : **Calculate the sum of all elements in a list of integers.**

🔷 File Input and Output

├ 📥 `import java.io.File;`

├ 📥 `import java.io.FileWriter;`

├ 📥 `import java.io.FileReader;`

├ 📥 `import java.io.IOException;`

├ 📁 `File file = new File("filename.txt");` : **Create a file object**

├ 📤 `FileWriter writer = new FileWriter("filename.txt");` : **Create a file writer**

├ 📝 `writer.write("Hello, World!");` : **Write to a file**

├ 📙 `writer.close();` : **Close a file writer**

├ 📥 `FileReader reader = new FileReader("filename.txt");` : **Create a file reader**

├ 📖 `int character = reader.read();` : **Read a single character from a file**

├ 📜 `char[] buffer = new char[1024];` : **Create a character buffer**

├ 📖 `int bytesRead = reader.read(buffer);` : **Read multiple characters from a file**

└ 📙 `reader.close();` : **Close a file reader**

Java Ultimate Cheat Sheet

🔷 Threads and Concurrency

├ 📥 *import java.util.concurrent.ExecutorService;*

├ 📥 *import java.util.concurrent.Executors;*

├ 🧵 *Runnable task = () -> { // Code to execute in the thread };* **: Create a runnable task**

├ 🗂️ *ExecutorService executor = Executors.newFixedThreadPool(5);* **: Create an executor**

├ 🚀 *executor.submit(task);* **: Submit a task to the executor**

├ ⏸️ *executor.shutdown();* **: Shut down the executor**

└ 📥 *Thread.sleep(1000);* **: Pause a thread for a specified number of milliseconds**

🔷 Networking

├ 📥 *import java.net.Socket;*

├ 📥 *import java.io.PrintWriter;*

├ 📥 *import java.io.BufferedReader;*

├ 📥 *import java.io.InputStreamReader;*

├ 🌐 *Socket socket = new Socket("hostname", portNumber);* : **Create a socket**

├ 📤 *PrintWriter out = new PrintWriter(socket.getOutputStream(), true);* **: Create a print writer**

├ 📝 *out.println("Hello, World!");* : **Send data over the socket**

├ 📥 *BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));* : **Create a buffered reader**

├ 📖 *String data = in.readLine();* : **Read data from the socket**

└ 🚪 *socket.close();* : **Close the socket**

Java Ultimate Cheat Sheet

🔷 Collections

├ 📥 *import java.util.List;*

├ 📥 *import java.util.ArrayList;*

├ 📥 *import java.util.Map;*

├ 📥 *import java.util.HashMap;*

├ 📥 *import java.util.Set;*

├ 📥 *import java.util.HashSet;*

├ 📥 *List<String> myList = new ArrayList<>();* **: Create a list**

├ 📝 *myList.add("Apple");* **: Add an element to a list**

├ 📚 *String element = myList.get(0);* **: Get the element at index 0 from a list of Strings.**

├ 📝 *myList.set(0, "Orange");* **: Set an element in a list**

├ 📜 *myList.remove("Orange");* **: Remove an element from a list**

├ 📊 *int size = myList.size();* **: Get the size of a list**

├ 📝 *myList.addAll(anotherList);* **: Add elements from another list to a list**

├ 📥 *Map<String, Integer> myMap = new HashMap<>();* **: Create a map**

├ 📝 *myMap.put("Apple", 1);* **: Add a key-value pair to a map**

├ 📚 *int value = myMap.get("Apple");* **: Get the value for a key in a map**

├ 📋 *Set<String> mySet = new HashSet<>();* **: Create a set**

├ 📝 *mySet.add("Apple");* **: Add an element to a set**

├ 📜 *mySet.remove("Apple");* **: Remove an element from a set**

└ 📊 *int size = mySet.size();* **: Get the size of a set**

Java Ultimate Cheat Sheet

🔷 Regular Expressions

├ 📥 *import java.util.regex.Matcher;*

├ 📥 *import java.util.regex.Pattern;*

├ 📝 *String regex = "pattern";* **: Define a regular expression**

├ 🧬 *Pattern pattern = Pattern.compile(regex);* **: Compile a regular expression**

├ 📥 *Matcher matcher = pattern.matcher("text");* **: Create a matcher**

├ 📊 *boolean match = matcher.matches();* **: Check if a string matches a regular expression**

├ 📊 *boolean find = matcher.find();* **: Find the next occurrence of a regular expression in a string**

├ 📝 *String replace = matcher.replaceFirst("replacement");* **: Replace the first occurrence of a regular expression in a string**

└ 📝 *String[] split = pattern.split("text");* **: Split a string into an array of substrings based on a regular expression**

🔷 Serialization and Deserialization

├ 📥 *import java.io.Serializable;*

├ 📥 *import java.io.ObjectOutputStream;*

├ 📥 *import java.io.ObjectInputStream;*

├ 📤 *ObjectOutputStream out = new ObjectOutputStream(outputStream);* **: Create an object output stream**

├ 📥 *ObjectInputStream in = new ObjectInputStream(inputStream);* **: Create an object input stream**

├ 📤 *out.writeObject(myObject);* **: Serialize an object**

├ 📥 *MyObject myObject = (MyObject) in.readObject();* **: Deserialize an object**

└ 📥 *class MyObject implements Serializable {}* **: Implement the Serializable interface**

Java Ultimate Cheat Sheet

🔷 Advanced Concepts

├ 📥 *import java.lang.reflect.Method;*

├ 📥 *import java.lang.annotation.Annotation;*

├ 📥 *import java.util.concurrent.Callable;*

├ 📥 *import java.util.concurrent.Future;*

├ 📥 *import java.util.concurrent.TimeUnit;*

├ 🧬 **Reflection :** Accessing and modifying objects at runtime

├ 🎨 **Annotations :** Adding metadata to classes, methods, and fields

├ 🎁 **Callable and Future :** Running tasks asynchronously and returning results

├ 🎁 **Executors :** Creating and managing thread pools

└ 🔍 **Concurrency :** Synchronization, locks, and atomic variables

🔷 Frameworks and Libraries

├ 🌟 **Spring :** A popular framework for building web applications and services

├ 🌟 **Hibernate :** A library for working with relational databases in Java

├ 🌟 **Jackson :** A library for working with JSON in Java

├ 🌟 **Apache Commons :** A collection of reusable Java components

├ 🌟 **Guava :** A library of core utilities and extensions to the Java collections framework

├ 🌟 **Log4j :** A logging utility for Java applications

├ 🌟 **JPA :** The Java Persistence API for working with databases in Java

├ 🌟 **Apache POI :** A library for working with Microsoft Office documents in Java

└ 🌟 **Apache Tomcat :** A popular web server and servlet container for Java web applications

Java Ultimate Cheat Sheet

🔷 Design Patterns

├ 🎁 **Adapter :** Adapting an interface to work with a different interface

├ 🎁 **Bridge :** Separating an abstraction from its implementation

├ 🎁 **Composite :** Treating a group of objects as a single object

├ 🎁 **Facade :** Providing a simplified interface to a complex system

├ 🎁 **Flyweight :** Sharing objects to reduce memory usage

├ 🎁 **Proxy :** Providing a placeholder for another object

├ 🎁 **Chain of Responsibility :** Passing a request through a chain of objects until it is handled

├ 🎁 **Command :** Encapsulating a request as an object

├ 🎁 **Interpreter :** Defining a language and interpreting expressions in that language

├ 🎁 **Iterator :** Providing a way to traverse a collection of objects

├ 🎁 **Mediator :** Encapsulating communication between objects

├ 🎁 **Memento :** Capturing and restoring an object's internal state

├ 🎁 **Observer :** Notifying objects of changes in a subject

├ 🎁 **State :** Allowing an object to change its behavior based on its state

├ 🎁 **Strategy :** Defining a family of algorithms and selecting one at runtime

├ 🎁 **Template Method :** Defining a common structure for a task

├ 🎁 **Visitor :** Separating an algorithm from the objects it operates on

└ 🎁 **Decorator :** Adding functionality to an object dynamically

Java Ultimate Cheat Sheet

🔷 Best Practices

├ 🏷️ Use meaningful variable names

├ 🧹 Use comments to explain code

├ 🌐 Use appropriate data types for variables

├ 🛠️ Avoid magic numbers in code

├ 🧰 Use try-with-resources for input/output streams

├ 🧬 Use generics to make code more flexible

├ 🧑‍💻 Use interfaces to define behavior

├ 🔒 Use access modifiers to control access to variables and methods

├ 🛡️ Use exception handling to handle errors

├ 🏭 Use the factory method pattern to create objects

├ 📊 Use the builder pattern to create complex objects

├ 🤝 Use the observer pattern to notify objects of changes

├ 📖 Use the iterator pattern to traverse collections

├ 📋 Use the singleton pattern to ensure only one instance of a class is created

├ 🧹 Use the decorator pattern to add functionality to an object

├ 🧬 Use the strategy pattern to define different algorithms for a task

├ 🚀 Use the template method pattern to define a common structure for a task

└ 🎁 Use the command pattern to encapsulate a request as an object

Java Ultimate Cheat Sheet

🔷 Development Tools

- ├ 🌟 Eclipse : A popular Java IDE
- ├ 🌟 IntelliJ IDEA : Another popular Java IDE
- ├ 🌟 NetBeans : A third Java IDE option
- ├ 🌟 Maven : A build automation tool for Java projects
- ├ 🌟 Gradle : Another build automation tool for Java projects
- ├ 🌟 JUnit : A unit testing framework for Java
- └ 🌟 Mockito : A mocking framework for Java unit tests