JAVA

CheatSheet

Java Basics

- **Hello World:** Printing to the console. example here

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
```

- **Variables and Data Types:** Integers, floating-point numbers, characters, strings, boolean.
- **Operators:** Arithmetic, assignment, comparison, logical, bitwise.
- **Control Flow:** if-else, switch-case, loops (for, while, do-while).
- **Arrays:** Declaration, initialization, accessing elements, array length.
- **Methods:** Declaration, parameters, return types, method overloading.
- **Classes and Objects:** Creating classes, instantiation, constructors, instance variables, methods.
- **Encapsulation:** Access modifiers (public, private, protected, default), getters and setters.

Object-Oriented Programming (OOP)

- **Inheritance:** Extending classes, super keyword, method overriding.
- **Polymorphism:** Method overloading, method overriding, dynamic method dispatch.
- **Abstraction:** Abstract classes, abstract methods, interfaces.
- **Encapsulation:** Access modifiers, getters, and setters, data hiding.

Exception Handling

- **try-catch Blocks:** Handling exceptions gracefully.
- **Multiple Catch Blocks:** Handling different types of exceptions separately.
- **throw and throws Keywords:** Throwing exceptions and propagating them.
- **Custom Exceptions:** Creating user-defined exceptions.

Collections Framework

- **Lists:** ArrayList, LinkedList, Vector.
- **Sets:** HashSet, LinkedHashSet, TreeSet.
- **Maps:** HashMap, LinkedHashMap, TreeMap.
- **Iterating Collections:** Using iterators, enhanced for loop.
- **Sorting Collections:** Comparable interface, Comparator interface.

Generics

- **Generic Classes:** Creating classes with type parameters.
- **Generic Methods:** Writing methods that work with different types.
- **Wildcard Types:** Unbounded wildcard, bounded wildcard.

Multithreading

- **Thread Basics:** Extending Thread class, implementing Runnable interface.
- **Thread Synchronization:** Synchronized blocks, synchronized methods.
- **Thread Pools:** Executors, ThreadPoolExecutor.
- **Concurrency Utilities:** AtomicInteger, CountDownLatch, CyclicBarrier.

Input/Output (I/O)

- **File Handling:** Reading from and writing to files.
- **Byte Streams:** FileInputStream, FileOutputStream.
- **Character Streams:** FileReader, FileWriter.
- **Buffered Streams:** BufferedReader, BufferedWriter.

Annotations

- **Built-in Annotations:** @Override, @Deprecated, @SuppressWarnings.
- **Custom Annotations:** Creating and using custom annotations.

Reflection

- **Class Objects:** Obtaining class objects, getClass() method.
- **Accessing Fields and Methods:** Field class, Method class.
- **Dynamic Loading:** Class.forName() method.

Advanced Topics

- **Lambda Expressions:** Writing concise anonymous functions.
- **Streams API:** Functional programming with streams.
- **Optional:** Dealing with potentially absent values.
- **Concurrency Utilities:** CompletableFuture, ForkJoinPool.
- **JDBC:** Database connectivity with Java.

Best Practices

- **Naming Conventions:** Follow standard naming conventions.
- **Code Organization:** Organize code into packages and classes logically.
- **Error Handling:** Use appropriate exception handling.
- **Memory Management:** Avoid memory leaks, and use resources efficiently.
- **Concurrency:** Ensure thread safety, and avoid deadlock.

Tools and IDEs

- **IDEs:** IntelliJ IDEA, Eclipse, NetBeans.
- **Build Tools:** Apache Maven, Gradle.
- **Version Control:** Git, SVN.

Debugging

- **Using Debuggers:** Setting breakpoints, and inspecting variables.
- **Logging:** Utilizing logging frameworks like Log4j, java.util.logging.

Testing

- Unit Testing: JUnit, TestNG.
- Mocking: Mockito, PowerMock.
- Integration Testing: Testing frameworks like Selenium for web applications.

Performance Tuning

- Profiling: Identifying performance bottlenecks using profilers.
- Memory Optimization: Reducing memory consumption.
- Algorithm Optimization: Improving algorithm efficiency.

Security

- Input Validation: Sanitizing user input to prevent injection attacks.
- Authentication and Authorization: Implementing secure login mechanisms.
- Encryption: Encrypting sensitive data using cryptographic algorithms.

Java EE (Enterprise Edition)

- Servlets: Handling HTTP requests and responses.
- JSP (JavaServer Pages): Dynamic web page generation.
- JDBC: Database connectivity.
- JPA (Java Persistence API): Object-relational mapping for databases.
- EJB (Enterprise JavaBeans): Component-based architecture.

Design Patterns

- Creational Patterns: Singleton, Factory, Builder.
- Structural Patterns: Adapter, Decorator, Proxy.
- Behavioral Patterns: Observer, Strategy, Command.