JAVA TUTORIAL 1

General Java Knowledge

Who developed Java?

Java was primarily developed by James Gosling and his team at Sun Microsystems (later acquired by Oracle Corporation).

Java Version History

- **Java 1.0 (January 1996):**
  - Initial release of Java.
  - Introduced applets for interactive web content.
  - Included AWT (Abstract Window Toolkit) for basic GUI development.
  - Featured a robust set of standard libraries for networking, I/O, and data structures.
- **Java 1.1 (February 1997):**
  - Enhanced event handling with the introduction of the Swing toolkit, providing a more powerful GUI framework.
  - Added inner classes, which enabled better encapsulation and organization of code.
  - Introduced JDBC (Java Database Connectivity) for database access.
- **Java 1.2 (Java 2, December 1998):**
  - Renamed to Java 2 Standard Edition (J2SE).
  - Introduced major enhancements:
    - Collections framework (java.util package) for data structure manipulation.
    - Java Naming and Directory Interface (JNDI) for accessing directory services.
    - Swing improvements and support for pluggable look-and-feel.
- **Java 1.3 (May 2000):**
  - Improved performance with HotSpot JVM (Java Virtual Machine).
  - Added JavaSound API for audio playback and MIDI (Musical Instrument Digital Interface) support.
  - Enhanced networking capabilities and XML processing with the introduction of SAX (Simple API for XML) parsing.
- **Java 1.4 (February 2002):**
  - Introduced `assert` keyword for programmatic assertions.
  - Added regular expressions support through `java.util.regex` package.
  - Included NIO (New I/O) API for scalable and non-blocking I/O operations.
- **Java 5 (September 2004):**
  - Renamed to Java 5 (or J2SE 5.0).
  - Introduced generics (parameterized types) for type-safe collections and APIs.
  - Added metadata annotations for embedding data and configurations within source code.
  - Enhanced for loop (for-each loop) for simplified iteration over collections.
- **Java 6 (December 2006):**
  - Focus on performance enhancements and web services improvements.
  - Introduced scripting API (`javax.script` package) for embedding scripting languages.
  - Added JDBC 4.0 with enhanced features for database connectivity.
- **Java 7 (July 2011):**
  - Introduced several language enhancements:
  - Try-with-resources statement for automatic resource management.
  - Diamond operator (<>) for improved type inference with generics.
  - Strings in switch statement for enhanced switch-case functionality.
- **Java 8 (March 2014):**
  - Major release focusing on functional programming capabilities:
  - Introduced Lambda expressions for concise syntax in functional interfaces.
  - Added Stream API for processing collections using functional-style operations.
  - Included Date and Time API (`java.time` package) for modern date/time handling.
- **Java 9 (September 2017):**
  - Modularized JDK with Project Jigsaw, introducing Java Platform Module System (JPMS).
  - Introduced JShell, an interactive REPL (Read-Eval-Print Loop) tool for experimenting with Java code.
  - Added enhancements to Stream API, Process API, and CompletableFuture.
- **Java 10 (March 2018):**
  - Introduced local variable type inference with `var` keyword for enhanced readability.
  - Added improvements to the Garbage-Collector interface and JDK APIs.
- **Java 11 (September 2018):**
  - Long-term support (LTS) release.
  - Introduced HTTP client API (`java.net.http` package) for improved handling of HTTP requests and responses.
  - Removed Java EE and Corba modules from JDK, focusing on core Java SE functionalities.
- **Java 12 - Java 17 (March 2019 - September 2021):**
  - These releases continued to add incremental improvements, including new language features, enhancements to the JDK libraries, and performance optimizations.
  - Highlights include preview features such as switch expressions, text blocks, sealed classes, and pattern matching.

Java Basics - For Each

for each loop (enhanced for loop) simplifies iterating over collections and arrays in Java.

```
for (elementType element : collection) {
    // statements
}
```

elementType: Type of elements in the collection/array.

element: Variable to hold each element during iteration.

collection: Iterable collection or array over which iteration is performed.

Benefits:

- Streamlines code for iterating over elements without manually managing indices.
- Improves readability and reduces potential errors related to index handling.
- Supports iterating over any type of iterable collection, including arrays, lists, sets, and more.

Example:

```
int[] numbers = {1, 2, 3, 4, 5};
for (int number : numbers) {
    System.out.println(number);
}
```

Reading Java Code

Understanding Java code involves grasping the syntax, conventions, and logic implemented within the code base.

Key aspects include:

- Identifying class and method declarations.
- Understanding variable declarations and their scopes.
- Analyzing control flow structures (loops, conditionals).
- Recognizing usage of Java libraries and APIs.
- Interpreting comments and documentation within the code for clarity.

Array of Object and their Memory Layout

Array of Objects: In Java, an array can hold objects of a class. The array elements are references (or pointers) to the objects, not the objects themselves.

Memory Layout:

- The array itself is stored as a contiguous block of memory.
- Each element in the array holds a reference to an object stored elsewhere in the heap memory.
- References in the array occupy a fixed amount of memory, regardless of the object size.
- Objects themselves are allocated dynamically in the heap and can vary in size.

Key Java Collections

HashMap

- Implements the Map interface, storing key-value pairs.
- Allows null values and one null key.
- Provides constant-time performance for basic operations (get and put), assuming the hash function disperses elements properly.
- Not synchronized; use ConcurrentHashMap for thread-safe operations.
- Iteration order is not guaranteed.

HashMap

- Access (get): O(1) - Average case, assuming a good hash function.
- Search (containsKey): O(1) - Average case, assuming a good hash function.
- Insertion (put): O(1) - Average case, assuming a good hash function.
- Deletion (remove): O(1) - Average case, assuming a good hash function.

Example:

```


Map map = new HashMap<>();
map.put("one", 1);
map.put("two", 2);
map.put("three", 3);

for (Map.Entry entry : map.entrySet()) {
    System.out.println(entry.getKey() + " = " + entry.getValue());
}

```

HashSet

- Implements the Set interface, backed by a HashMap.
- Stores unique elements; duplicates are not allowed.
- Allows null values.
- Provides constant-time performance for basic operations (add, remove, contains), assuming the hash function disperses elements properly.
- Not synchronized; use Collections.synchronizedSet for thread-safe operations.
- Iteration order is not guaranteed.

HashSet

- Access (get): O(1) - Direct access is not applicable as HashSet does not provide access by index.
- Search (contains): O(1) - Average case, assuming a good hash function.
- Insertion (add): O(1) - Average case, assuming a good hash function.
- Deletion (remove): O(1) - Average case, assuming a good hash function.

Example:

```
Set set = new HashSet<>();
set.add("one");
set.add("two");
set.add("three");

for (String s : set) {
    System.out.println(s);
}

```

ArrayList

- Implements the List interface, backed by a dynamically resizable array.
- Allows duplicate elements and maintains insertion order.
- Provides constant-time performance for positional access (get and set) and amortized constant-time for adding elements.
- Allows null values.
- Not synchronized; use Collections.synchronizedList for thread-safe operations.
- Performance can degrade if elements are frequently added/removed from the middle of the list.

Access (get): O(1) - Direct access using the index.

- Search (contains): O(n) - Needs to iterate through the list to find the element.
- Insertion (add):
- O(1) - Amortized constant time when adding at the end (considering occasional resizing).
- O(n) - When adding at a specific position, as it may require shifting elements.
- Deletion (remove):
- O(n) - Needs to shift elements after the removed element.

Example:

```
List list = new ArrayList<>();
list.add("one");
list.add("two");
list.add("three");

for (String s : list) {
    System.out.println(s);
}
```

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

Popular posts from this blog