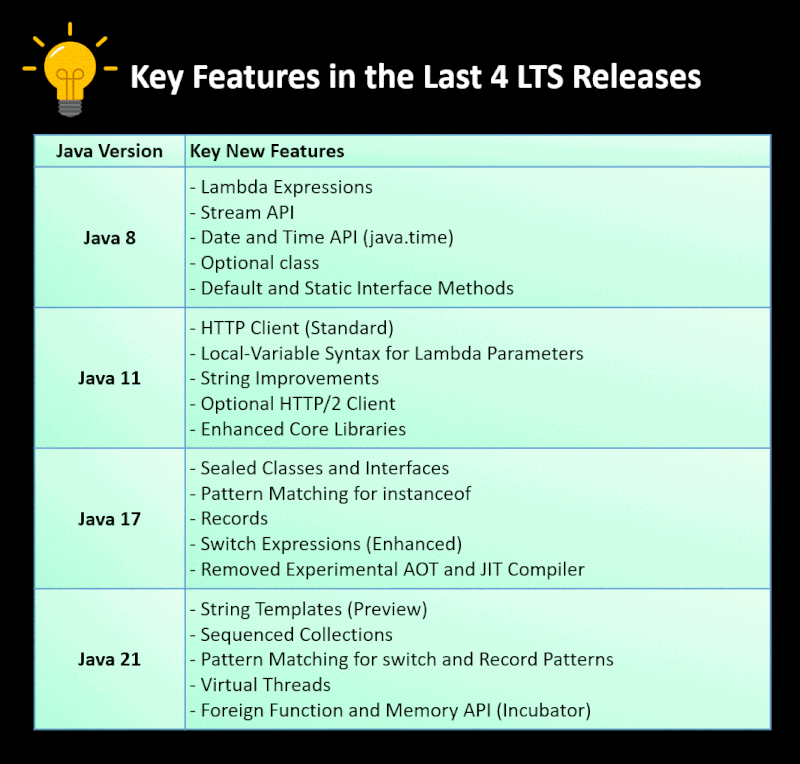
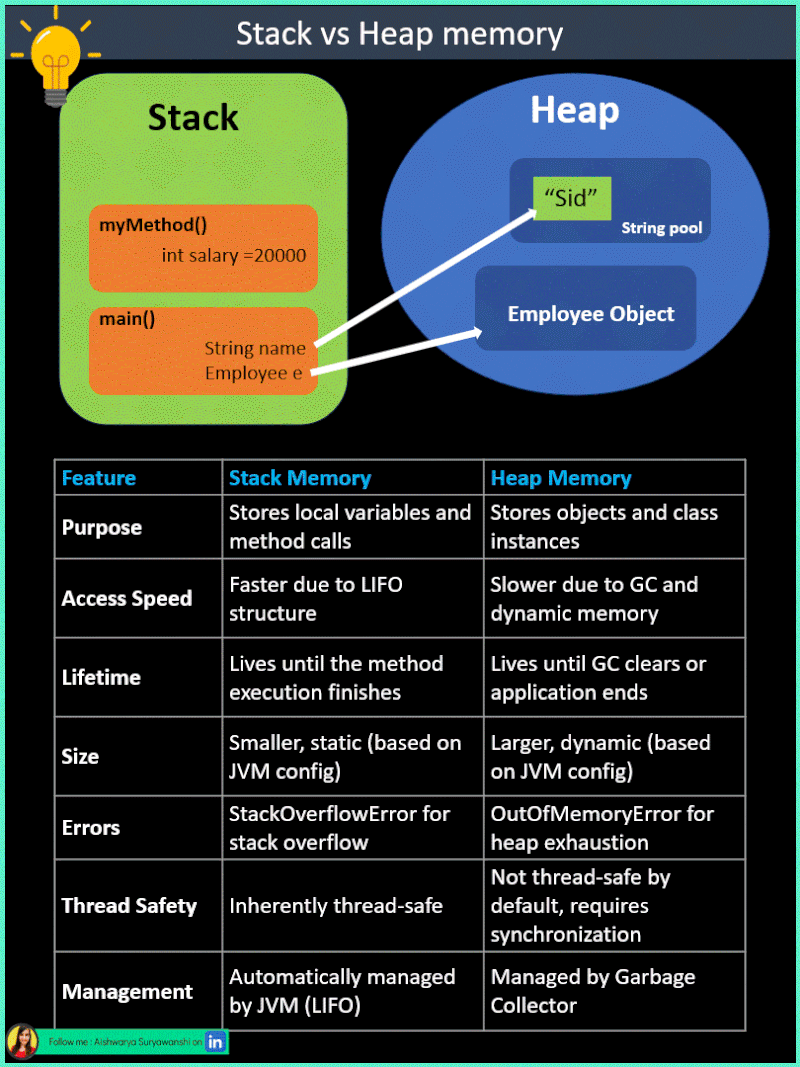


⭐ Classes & Objects:  
Classes are blueprints that define the properties and behaviors of objects. An object is an instance of a class, representing a specific realization of the blueprint with its own unique state and behavior. This allows for creating and managing multiple instances with similar structures but different data.  
⭐ Encapsulation:  
Encapsulation involves bundling the data (attributes) and the methods that operate on the data into a single unit (a class). It restricts direct access to some of the object's components, which can prevent unintended interference and misuse of the data. By providing access methods (getters and setters), encapsulation ensures that data is accessed and modified in a controlled manner.  
⭐ Inheritance:  
Inheritance allows a new class to inherit properties and behaviors from an existing class. This promotes code reuse by enabling the creation of a new class based on an existing class, called the parent or superclass. The new class, known as the child or subclass, can extend or modify the inherited features and add new ones. This hierarchical relationship helps in organizing code and building upon existing functionality.  
⭐ Polymorphism:  
Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. It provides the ability to use a single method name to perform different tasks based on the object’s actual class. This is achieved through:  
Method Overloading: Defining multiple methods with the same name but different parameter lists within the same class. This enables the use of the same method name to perform various functions based on the arguments passed.  
Method Overriding: Redefining a method in a subclass that has already been defined in its superclass. This allows a subclass to provide a specific implementation of a method that is already defined in its superclass, enhancing the method’s functionality for the subclass.  
⭐ Abstraction:  
Abstraction involves defining abstract classes or interfaces that provide a common interface for a group of related classes. It focuses on exposing only the relevant features of an object while hiding the complex implementation details. This helps in designing systems where the internal workings are hidden, and only the essential aspects are exposed to the user, facilitating easier management and scalability.  
  
🚀 Benefits of OOP:  
-Reusability: Leverage existing classes to build new functionalities, reducing redundancy and saving time.  
-Maintainability: Simplify updates and fixes by managing code in modular units (classes).  
-Flexibility: Adapt and extend code with minimal disruption through inheritance and polymorphism.

The Long-Term Support (LTS) versions of Java are released every three years, and as of 2024, the four most recent LTS versions are Java 21 (2023), Java 17 (2021), Java 11 (2018), and Java 8 (2014).  
  
LTS refers to a version of the software that is supported for an extended period, usually several years. During this time, the software receives regular updates, including security patches and bug fixes, ensuring stability and reliability for users who rely on it for critical systems and applications.  
Java's LTS releases are designed to offer a stable and robust platform for developers, allowing them to use these versions for a longer time without needing to upgrade frequently. LTS versions are ideal for enterprises and projects that prioritize long-term stability over adopting the latest features immediately. Typically, Java releases an LTS version every three years.  
  
🚀Java 8 (2014)  
Lambda Expressions  
Stream API  
Date and Time API (java.time)  
Optional class  
Default and Static Interface Methods  
🚀 Java 11 (2018)  
HTTP Client (Standard)  
Local-Variable Syntax for Lambda Parameters  
String Improvements  
Optional HTTP/2 Client  
Enhanced Core Libraries  
🚀 Java 17 (2021)  
Sealed Classes and Interfaces  
Pattern Matching for instanceof  
Records  
Enhanced Switch Expressions  
Removal of Experimental AOT and JIT Compiler  
🚀 Java 21 (2023)  
String Templates (Preview)  
Sequenced Collections  
Pattern Matching for switch and Record Patterns  
Virtual Threads  
Foreign Function and Memory API (Incubator)  
  
As Java continues to grow, it’s important to stay updated, as features introduced in one version might be refined or even removed in later versions.   
  
Whether you're maintaining legacy systems or building new applications, understanding these evolving features ensures that your codebase remains robust, maintainable, and future-proof.





Have you ever wondered how a HashMap works behind the scenes? 🤔 It's one of the most widely used data structures in Java!  
  
💡 Hashing: Each key-value pair is stored in a bucket. The bucket is determined by the hash code of the key. Java’s hashCode() method is used to compute an integer hash code which is then transformed into a bucket index.  
💡Buckets: A HashMap consists of an array of buckets. Initially, this array has a default size of 16. Each bucket can hold multiple entries in case of collisions.  
💡Handling null Key: HashMap allows one null key. When the key is null, it is always stored in the first bucket (bucket index 0). This special handling ensures that null keys do not interfere with the normal hashing mechanism.  
💡Collision Handling: When two keys have the same hash code, they end up in the same bucket. HashMap handles collisions using linked lists (up to Java 7) or balanced trees (from Java 8 onwards if the number of items in a bucket exceeds a certain threshold).  
💡Entry Objects: Each bucket contains an Entry object. An Entry stores key-value pairs and a reference to the next Entry in case of a collision, forming a linked list or tree.  
💡Load Factor and Resizing: The load factor (default 0.75) determines when to resize the HashMap. When the number of entries exceeds the product of load factor and current capacity, the capacity is doubled, and all existing entries are rehashed to the new buckets.

