1. **String Immutability in Java**

In Java, String objects are immutable, meaning once a String object is created, its value cannot be modified. This immutability is implemented by making the String class final and its internal character array private and final, preventing any alteration. When a String seems to change, a new object is actually created with the updated value, and the original object remains unaltered. Immutability provides security, thread safety, and optimization benefits through string pooling, which reuses String instances, saving memory and improving performance.

1. **Arrays and ArrayLists**

Arrays and ArrayList are two ways to handle collections in Java, with significant differences:Arrays are fixed-size data structures that hold elements of the same type. Once created, the length of an array cannot be modified. Arrays are more memory efficient and perform faster for basic operations due to their fixed size.

ArrayLists, part of the Collection Framework, are dynamic-sized arrays. They allow elements to be added or removed, automatically resizing as needed. ArrayList is a generic class, supporting various data types and providing many utility methods. However, ArrayList has a slightly slower performance than arrays because it manages resizing and elements through methods.

1. **Overriding equals() and hashCode() in the Context of the Collection Framework**

Equals() defines when two objects are considered equal. When overridden, it should check if objects have the same values in their fields.

hashCode() generates a hash code, an integer representing the object’s location in a hash table. If equals() is overridden, hashCode() should also be overridden to ensure that equal objects have the same hash code, maintaining consistency. Properly implementing these methods is essential for operations like searching, insertion, and deletion in collections like HashSet and HashMap.

**D) Database Connectivity in Java**

Java Database Connectivity (JDBC) is an API that enables Java applications to interact with databases. JDBC provides methods to connect to a database, execute SQL queries, retrieve and update data. Basic steps in JDBC include:

Loading the driver – Loading the database driver to establish communication.

Establishing a connection – Connecting to the database with a URL, username, and password.

Executing SQL queries – Using Statement or PreparedStatement objects to execute queries.

Processing results – Retrieving query results using ResultSet.

Closing the connection – Closing connections to free up resources.

JDBC is versatile, supporting multiple database types, and serves as the foundation for frameworks like Hibernate and JPA.

1. **Interfaces in Java: Types and Advanced Features**

Interfaces in Java are abstract types that specify a set of method signatures that implementing classes must define. They support multiple inheritance, allowing classes to implement multiple interfaces. Java interfaces have evolved with new features:

Types of Interfaces:

**Marker Interfaces**: Empty interfaces (like Serializable) used to signal specific capabilities.

**Functional Interfaces**: Interfaces with a single abstract method (SAM), used for lambda expressions. Common examples include Runnable and Comparator.

Advanced Features:

**Default Methods**: Introduced in Java 8, allowing interfaces to have method implementations, enabling backward compatibility without forcing all implementing classes to override these methods.

**Static Methods**: Java 8 also introduced static methods in interfaces, allowing methods to be accessed directly through the interface.

**Private Methods**: Java 9 added private methods, which can only be used within the interface to reduce code duplication in default methods.

**MYSQL**

**Datatypes**

1. CHAR

Definition: CHAR is a fixed-length character data type.

Characteristics:

It always reserves the specified number of characters, padding with spaces if the value is shorter.

Ideal for storing data with fixed sizes, such as codes or identifiers.

Storage: Uses 1 byte per character for ASCII data.

Example: CHAR(10) will always store 10 characters, padding with spaces if the input is shorter.

1. VARCHAR

Definition: VARCHAR is a variable-length character data type.

Characteristics:

It stores only the actual characters provided, up to a specified limit, which saves space compared to CHAR.

Commonly used for text fields where the length varies, like names or descriptions.

Storage: Uses 1 byte per character plus 1 or 2 bytes to store the length (1 byte if less than 255 characters, 2 bytes otherwise).

Example: VARCHAR(50) will store up to 50 characters, using storage based on the actual length of the string.

1. NVARCHAR

Definition: NVARCHAR is also a variable-length character data type but designed to store Unicode (UTF-8) characters.

Characteristics:

Supports international characters, making it suitable for multilingual data.

Unlike CHAR and VARCHAR, NVARCHAR ensures proper storage of special and accented characters.

Storage: Uses 2 or more bytes per character, depending on the encoding.

Example: NVARCHAR(100) allows storage of up to 100 Unicode characters.

Key Differences

Fixed vs. Variable Length: CHAR is fixed-length, while VARCHAR and NVARCHAR are variable-length.

Usage: CHAR is optimal for consistent-length fields, VARCHAR for variable-length ASCII text, and NVARCHAR for variable-length Unicode text.

Encoding Support: NVARCHAR provides better support for special characters and multiple languages, unlike CHAR and VARCHAR, which are limited to ASCII.

These types help optimize storage and retrieval efficiency in MySQL, depending on the type and variability of text.

**Types of Joins in MySQL**

**INNER JOIN**

Returns rows where there is a match in both tables.

Only the common data between the tables is included in the result.

Syntax: SELECT \* FROM table1 INNER JOIN table2 ON table1.column = table2.column;

LEFT JOIN (or LEFT OUTER JOIN)

Returns all rows from the left table and the matched rows from the right table.

If there is no match, NULL values are returned for columns from the right table.

**Syntax**: SELECT \* FROM table1 LEFT JOIN table2 ON table1.column = table2.column;

**RIGHT JOIN** (or RIGHT OUTER JOIN)

Returns all rows from the right table and the matched rows from the left table.

If there is no match, NULL values are returned for columns from the left table.

**Syntax**: SELECT \* FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;

**FULL JOIN (**or FULL OUTER JOIN)

Returns all rows from both tables, with NULL values where there is no match.

MySQL doesn’t directly support FULL JOIN, but it can be simulated using UNION.

**Syntax** (Simulated):

Sql

Copy code

SELECT \* FROM table1 LEFT JOIN table2 ON table1.column = table2.column

UNION

SELECT \* FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;

**CROSS JOIN**

Returns the Cartesian product of both tables, meaning every row from the first table is paired with every row from the second.

Useful for generating combinations but can produce large result sets.

**Syntax**: SELECT \* FROM table1 CROSS JOIN table2;

**SELF JOIN**

Joins a table with itself.

Useful for comparing rows within the same table.

**Syntax**: SELECT a.column, b.column FROM table a, table b WHERE a.common\_column = b.common\_column;

**Types of Constraints in MySQL**

PRIMARY KEY

Uniquely identifies each record in a table.

Ensures that no two rows have the same value in the primary key column(s) and prevents NULL values.

Syntax: PRIMARY KEY (column\_name)

FOREIGN KEY

Establishes a relationship between two tables, enforcing referential integrity.

Links a column in one table to the primary key of another table, ensuring valid references.

Syntax: FOREIGN KEY (column\_name) REFERENCES other\_table(other\_column)

UNIQUE

Ensures all values in a column or set of columns are unique, preventing duplicate values.

Unlike PRIMARY KEY, UNIQUE allows one NULL value.

Syntax: UNIQUE (column\_name)

NOT NULL

Prevents NULL values in a column, ensuring that every row has a value.

Often used in columns that are essential for the data’s integrity, such as identifiers or status fields.

Syntax: column\_name data\_type NOT NULL

CHECK

Ensures that a column’s value meets a specific condition.

MySQL started supporting CHECK constraints in version 8.0, allowing conditions like ranges or specific formats.

Syntax: CHECK (condition)

DEFAULT

Sets a default value for a column when no specific value is provided during insertion.

Useful for columns with common or expected values.

Syntax: DEFAULT value

These joins and constraints are essential for maintaining relational integrity and efficient data management in MySQL databases.

**ACID Properties of Databases**

1. Atomicity

Definition: Atomicity ensures that a database transaction is treated as a single, indivisible unit. Either all operations within the transaction are completed successfully, or none of them are applied.

Importance: This prevents partial updates in case of a failure. For instance, if a bank transfer operation fails midway, atomicity ensures that neither account is updated, avoiding any incorrect balance.

1. Consistency

Definition: Consistency ensures that a database starts in a valid state and ends in a valid state after a transaction.

Importance: Each transaction must bring the database from one consistent state to another, following all rules and constraints (e.g., data integrity, foreign key constraints). This prevents corrupted or invalid data from being saved, ensuring that all data meets the defined constraints.

1. Isolation

Definition: Isolation ensures that transactions are executed independently without interference. The operations in one transaction should be invisible to other transactions until it’s completed.

Importance: Isolation prevents issues like dirty reads, non-repeatable reads, and phantom reads. It ensures that concurrent transactions do not affect each other’s execution, resulting in stable and predictable outcomes even in multi-user environments.

1. Durability

Definition: Durability ensures that once a transaction is committed, its results are permanently stored in the database, even if the system crashes afterward.

Importance: This is achieved through mechanisms like transaction logs and backups. Durability guarantees that committed data is safe and can be recovered in case of power failures, crashes, or other failures.