Core Java:

Difference between JDK, JRE, and JVM: JDK is for development, JRE is for running, and JVM executes Java code.

Class vs. Object: A class is like a blueprint, and an object is an instance of a class.

Inheritance: It's inheriting properties from another class.

Constructor: It's a method used for initializing objects.

Polymorphism: Many forms of a method.

Static Keyword: Used for variables/methods that belong to the class, not objects.

Access Modifiers: Control access to class members.

Interface vs. Abstract Class: Interface specifies behavior, abstract class can have some code.
Exception Handling: Handling errors in a program.

Final Keyword: Makes a variable or method unchangeable.

Garbage Collection: Automatically cleans up unused objects.

ArrayList vs. LinkedList: ArrayList is resizable, LinkedList is node-based.

hashCode and equals: Used for object comparison.

Synchronized: Makes methods thread-safe.

== vs. equals: Compares objects' references vs. contents.

SQL:

SQL Importance: Manages databases.

SQL vs. NoSQL: Structured vs. unstructured data.

CRUD Operations: Create, Read, Update, Delete data.

INNER JOIN vs. LEFT JOIN: Combine data from tables.

Normalization: Organizing data efficiently.

Indexes: Speed up data retrieval.

Primary Key: Uniquely identifies rows.

GROUP BY: Group and aggregate data.

SQL Injection: Prevent hacking.

UNION vs. UNION ALL: Combine query results.

Subquery: Nested query.

ACID Properties: Data transaction reliability.

Optimizing Queries: Improve query speed.

View vs. Table: Virtual vs. physical data.

HAVING Clause: Filters grouped data.

Data Structures:

Data Structure: Organizes and stores data.

Array vs. Linked List: Fixed vs. flexible size.

Stack: Last in, first out (LIFO) structure.

Queue: First in, first out (FIFO) structure.

Binary Trees: Hierarchical data structure.

Hashing: Quick data retrieval.

ArrayList vs. LinkedList: Resizable vs. node-based list.

Hash Table: Key-value data storage.

Linked List vs. Doubly Linked List: One-way vs. two-way connections.

Graphs: Complex data relationships.

Bubble Sort: Simple sorting algorithm.

Dynamic Programming: Efficient problem solving.

Heap Data Structure: Priority queue.

Sorting Algorithms: Different ways to sort data.

Big O Notation: Analyzing algorithm efficiency.

Mobile Application Development using Android Studio:

Android: Popular for mobile apps.

App Components: Building blocks.

Activity: Represents a screen.

Fragment: Reusable UI part.

AndroidManifest.xml: App configuration.

Screen Sizes: Adapt to devices.

AsyncTask vs. ThreadPoolExecutor: Background tasks.

Android App Bundle (AAB): App packaging.

SharedPreferences: Store app data.

Responsive UI: Fit different screens.

Lifecycle: App state management.

RecyclerView: Efficient list display.

Intent: Communication between components.

Runtime Permissions: User access control.

Gradle: Build and dependency management.

Spring Boot:

Spring Boot: Simplifies Java apps.
Spring Boot vs. Spring: Easier setup.
Annotation: Metadata for config.
@RestController: RESTful APIs.
Data Source Configuration: Connect to a database.
Dependency Injection: Provides dependencies.
Spring Boot Starter: Pre-configured dependencies.
Actuator: Monitor app.
Auto-Configuration: Automatic setup.
Application Properties: Configure app.
DevTools: Faster development.
Security: Protects app.
Testing: Unit tests.
Deployment: App launch.
Profiles: Environment-specific config.

Servlets:

Servlet: Java web component.
Servlet vs. JSP: Logic vs. UI.
Initialize and Destroy: Lifecycle methods.
HttpServletRequest and HttpServletResponse: Client-server interaction.
Form Data Handling: User input.
doGet() vs. doPost(): HTTP methods.
HttpSession: User session data.
URL Rewriting and Cookies: Data persistence.
ServletContext: App-wide data.
Servlet Filters: Pre/post-processing.
Pass Data to JSP: Attributes.
Servlet Listeners: Event handling.

Hibernate:

Hibernate: Database interaction.
Hibernate vs. JDBC: Object-oriented vs. SQL.
Mapping Files (hbm.xml): Connect Java to DB.
CRUD Operations: Create, Read, Update, Delete data.
Session and SessionFactory: DB interaction.
Lazy Loading: Load data when needed.
Database Connection: Configuration.
Caching: Faster data access.
Object States: Transient, persistent, detached.
HQL Query: SQL-like queries.
Criteria API: Programmatic queries.
Annotations: Simplify mapping.
Transaction Management: Data consistency.
Inheritance Mapping: Table design.
Best Practices: Efficient coding.

HTML:

HTML: Web content structure.
HTML Structure: Tags organize content.
Tags and Attributes: Define content.
Hyperlinks: Navigate the web.
HTML5: Latest HTML version.
Forms: User input.
Multimedia: Images and videos.
Tables: Data presentation.
Semantic HTML: Meaningful tags.
Responsive Design: Adapts to screens.

Collection Framework:

What is the Java Collection Framework?
It's a library of classes and interfaces for working with collections like lists and maps.

What is the difference between ArrayList and LinkedList?
ArrayList uses dynamic arrays, LinkedList uses nodes. ArrayList is faster for random access, LinkedList for insertions/deletions.

Explain the Set interface and give an example of a Set implementation.
Set doesn't allow duplicate elements. Example: HashSet.

What is the difference between List and Set?
List allows duplicates, maintains order; Set doesn't allow duplicates, no specific order.

What is the Map interface and give an example of a Map implementation.
Map stores key-value pairs. Example: HashMap.

What is the difference between HashMap and HashTable?
HashMap is not synchronized (not thread-safe), Hashtable is synchronized (thread-safe).

Explain the purpose of the Iterator in the Collection Framework.
Iterator allows you to traverse through a collection and perform operations on its elements.

What is the difference between HashSet and TreeSet?
HashSet doesn't maintain order, TreeSet sorts elements in natural order or by a provided comparator.

What is the difference between ArrayList and Vector?
Vector is synchronized (thread-safe), ArrayList is not.

Explain the concept of generics in Java collections.
Generics allow you to specify the type of elements a collection can hold.

Exception Handling:

What is an exception in Java?
An exception is an unexpected event that disrupts the normal flow of a program.

Explain the difference between checked and unchecked exceptions.
Checked exceptions must be handled (e.g., IOException), unchecked exceptions (e.g., NullPointerException) need not be handled.

What is the purpose of the try-catch block?
It's used to catch and handle exceptions.

What is the "finally" block used for in exception handling?
The "finally" block is executed whether an exception occurs or not, typically used for cleanup.

What is the "throw" keyword used for?
It's used to explicitly throw an exception.

Explain the "throws" keyword in Java.
It's used in a method signature to declare that the method might throw a particular type of exception.

What is the difference between "throw" and "throws"?
"throw" is used to throw an exception, "throws" is used to declare exceptions that a method may throw.

What is the purpose of the "try-with-resources" statement in Java?
It's used for automatic resource management, like closing files or sockets.

What is a custom exception in Java, and how do you create one?
Custom exceptions are user-defined exceptions. You create one by extending the Exception class.

Explain the role of the "catch" block in multiple catch blocks.
It allows you to catch different types of exceptions and handle them differently.

Arrays:

What is an array in Java?
An array is a data structure that stores a fixed-size sequence of elements of the same type.

How do you declare an array in Java?
Example: int[] numbers = new int[5];

What is the difference between an array and an ArrayList?
Arrays have a fixed size, ArrayLists can grow dynamically.

How do you access elements in an array?
By using the index, e.g., numbers[0] accesses the first element.

Explain the enhanced for loop (for-each) in Java.
It simplifies iterating through arrays and collections.

What is the length of an array in Java?
It's the number of elements in the array, accessed with array.length.

Can you change the size of an array once it's declared?
No, arrays have a fixed size.

What is a multi-dimensional array?
It's an array of arrays, used for representing tables or grids.

How do you find the maximum element in an array?
By iterating through the array and keeping track of the maximum value.

What is the difference between int[] array and int array[] in Java?
There's no difference; both syntax forms are allowed.

String:

What is a String in Java?
A String is a sequence of characters.

How do you create a String in Java?
By assigning a string literal or using the new keyword, e.g., String str = "Hello";.

Can you modify a String in Java?
No, Strings are immutable (cannot be changed).

Explain the difference between == and .equals() for comparing strings.
== checks if references are the same; .equals() checks if content is the same.

What is the length() method used for in Java Strings?
It returns the number of characters in a String.

How do you concatenate Strings in Java?
You can use the + operator or the concat() method.

What is the purpose of the StringBuilder class in Java?
It's used for efficient String manipulation as it's mutable.

What is a substring in Java?
A part of a String extracted using the substring() method.

Explain the compareTo() method for comparing Strings.
It's used to compare two Strings lexicographically.

What is the purpose of the trim() method in Java Strings?
It removes leading and trailing white spaces from a String.

OOPs (Object-Oriented Programming):

What is OOP?
It's a programming paradigm based on objects and classes.

Explain the four pillars of OOP.
Encapsulation, Inheritance, Polymorphism, Abstraction.

What is a class in OOP?
It's a blueprint for creating objects.

What is an object in OOP?
An instance of a class.

What is encapsulation?
It's the bundling of data and methods that operate on the data into a single unit (class).

What is inheritance?
It's the mechanism that allows one class to inherit properties and behaviors from another class.

What is polymorphism?
It's the ability of different objects to respond to the same method call in their own way.

What is abstraction?
It's simplifying complex reality by modeling classes based on real-world entities.

What is a constructor in Java?
A special method used for object initialization.

Explain the concept of method overloading.
It's when a class has multiple methods with the same name but different parameters.

LinkedList:

What is a LinkedList in Java?
It's a data structure that stores elements in a linear order using nodes.

How is a LinkedList different from an ArrayList?
LinkedList uses nodes and is more efficient for insertions/deletions, while ArrayList uses arrays and is faster for random access.

Explain the difference between singly linked lists and doubly linked lists.
Singly linked lists have nodes with one reference (next), while doubly linked lists have nodes with two references (next and previous).

How do you add elements to the beginning and end of a LinkedList?
You can use addFirst() and addLast() methods.

What is the purpose of the remove() method in a LinkedList?
It's used to remove elements by value or index.

How do you traverse a LinkedList?
You can use a loop and the next reference of nodes.

What is the time complexity for inserting an element in a LinkedList?
O(1) for inserting at the beginning, O(n) for inserting at other positions.

What is the time complexity for deleting an element in a LinkedList?
O(1) for deleting at the beginning, O(n) for deleting at other positions.

What is a circular LinkedList?
It's a LinkedList where the last node points to the first node.

Explain the concept of a doubly ended LinkedList.
It's a LinkedList where you can add and remove elements from both ends efficiently.

Hashing:

What is hashing?
It's a process of converting data (like a key) into a fixed-size value (a hash code).

What is a hash code in Java?
It's a unique identifier generated for objects.

Explain collision in hashing.
It's when two different keys produce the same hash code.

What is the purpose of a hash function?
It's used to convert data into a hash code.

What is an ideal hash function?
One that distributes keys evenly, minimizing collisions.

How are collisions resolved in hashing?
Techniques like chaining (linked lists) or open addressing (rehashing) are used.

What is load factor in hashing?
It's the ratio of the number of elements to the number of buckets in a hash table.

What is hashing used for in Java Collections?
HashMaps and HashSets use hashing for efficient data retrieval.

Explain the hashCode() method in Java.
It returns the hash code for an object and is used in hashing-based collections.

What is the importance of a good hash function in hashing?
It reduces collisions and improves the efficiency of hash-based data structures.

HashMap:

What is a HashMap in Java?
It's a data structure that stores key-value pairs.

How do you add elements to a HashMap?
You use the put() method.

How do you retrieve values from a HashMap?
You use the get() method, providing the key.

What happens if you try to add a duplicate key to a HashMap?
It will replace the old value with the new one.

How does a HashMap handle collisions?
It uses linked lists (chaining) to store multiple values at the same hash code.

What is the default initial capacity of a HashMap?
It's 16.

What is the load factor of a HashMap, and why is it important?
The load factor is a threshold; when reached, the HashMap is resized to maintain efficiency.

How do you remove a key-value pair from a HashMap?
You use the remove() method, providing the key.

What is the time complexity for adding, retrieving, and removing elements in a HashMap?
O(1) on average, but O(n) in the worst case.

What is the difference between HashMap and HashTable?
HashMap is not synchronized (not thread-safe), Hashtable is synchronized (thread-safe).

Tree:

What is a tree in data structures?
A tree is a hierarchical data structure consisting of nodes connected by edges.

What is a binary tree in Java?
It's a tree in which each node has at most two children.

What is a root node in a tree?
It's the topmost node, the starting point.

What is a leaf node in a tree?
It's a node with no children.

What is a binary search tree (BST)?
It's a binary tree with the left subtree containing nodes with values less than the root and the right subtree containing nodes with values greater than the root.

What is the height of a tree?
It's the length of the longest path from the root to a leaf.

Explain the concept of a balanced tree.
A tree in which the height of the left and right subtrees of any node differ by at most one.

What is a self-balancing tree, and why is it important?
It's a tree that automatically maintains balance, ensuring efficient operations.

What is a binary search tree traversal?
It's the process of visiting and processing each node of a BST in a specific order (in-order, pre-order, post-order).

How do you insert and search for an element in a binary search tree?
Insertion maintains the BST property, and searching is done by comparing values at nodes while traversing the tree.

CODE

1. Print "Hello, World!”

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

2. Swap two numbers without using a temporary variable:

```
int a = 5, b = 10;
a = a + b;
b = a - b;
a = a - b;
```

3. Find the factorial of a number:

```
int number = 5;
int factorial = 1;
for (int i = 1; i <= number; i++) {
factorial *= i;
}
```

4. Check if a number is prime:

```
int number = 17;
boolean isPrime = true;
for (int i = 2; i <= Math.sqrt(number); i++) {
if (number % i == 0) {
isPrime = false;
break;
}
}
```

5. Reverse a string:

```
String input = "Hello, World!";
String reversed = new StringBuilder(input).reverse().toString();
```

6. Find the maximum element in an array:

```
int[] arr = {5, 3, 9, 1, 7};
int max = arr[0];
for (int i = 1; i < arr.length; i++) {
if (arr[i] > max) {
max = arr[i];
}
}
```

7. Calculate the sum of natural numbers up to a given number:

```
int n = 10;
int sum = 0;
for (int i = 1; i <= n; i++) {
sum += i;
}
```

8. Check if a string is a palindrome:

```
String input = "racecar";
boolean isPalindrome = true;
for (int i = 0; i < input.length() / 2; i++) {
if (input.charAt(i) != input.charAt(input.length() - 1 - i)) {
isPalindrome = false;
break;
}
}
```

9. Find the Fibonacci series up to a given number:

```
int n = 10;
int a = 0, b = 1;
System.out.print(a + " " + b + " ");
for (int i = 2; i < n; i++) {
int next = a + b;
System.out.print(next + " ");
a = b;
b = next;
}
```

10. Check if a year is a leap year:

```
int year = 2024;
boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
```

11. Total count of the vowel in string or in an array:

```
public class CountVowels {
public static void main(String[] args) {
String input = "Hello, World!";
int vowelCount = countVowels(input);
System.out.println("Total vowels in the string: " + vowelCount);
}

public static int countVowels(String str) {
str = str.toLowerCase(); // Convert the string to lowercase to handle both upper and lower
case vowels
int count = 0;
for (int i = 0; i < str.length(); i++) {
char ch = str.charAt(i);
if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
count++;
}
}

return count;
 }
}
```

12. Print the total even number in the given array.

```
public class CountEvenNumbers {
public static void main(String[] args) {
int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int evenCount = countEvenNumbers(numbers);
System.out.println("Total even numbers in the array: " + evenCount);
}

public static int countEvenNumbers(int[] arr) {
int count = 0;

for (int num : arr) {
if (num % 2 == 0) {
count++;
}
}

return count;
}
}
```