25 Important Interview questions on JAVA specifically (OOPS portions are not included).

1. **Why is Java called platform-independent?**

   - **Answer:** Java is called platform-independent because its code, once compiled, is transformed into bytecode by the Java Compiler. This bytecode is platform-neutral and can be executed on any platform that has a Java Virtual Machine (JVM). The JVM interprets the bytecode into machine-specific code, enabling the same Java program to run on various operating systems without modification.
   - **Example:** A Java program compiled on Windows can run on Linux or Mac OS without any changes, provided the JVM is available on those platforms.

2. **Explain JVM, JRE, and JDK.**

   - **Answer:**
     - **JVM (Java Virtual Machine):** An abstract machine that enables a computer to run Java programs. It converts bytecode into machine code.
     - **JRE (Java Runtime Environment):** A package that includes the JVM and standard libraries, enabling the execution of Java applications.
     - **JDK (Java Development Kit):** A complete package for Java development, including the JRE, compiler, and various development tools.
   - **Example:** Developers use the JDK for coding and compiling Java programs, while end-users need the JRE to run these programs.

3. **What is Bytecode and how does it make Java programs platform-independent?**

   - **Answer:** Bytecode is an intermediate code generated by the Java compiler, which is platform-independent. The JVM on each platform interprets this bytecode into machine-specific instructions, allowing Java programs to run on any device with a JVM.
   - **Example:** Bytecode for a Java program compiled on Windows can be executed on Linux using the Linux JVM.

4. **Compare features of Java with C and Python.**

   - **Answer:**
     - **Java vs C:**
       - Memory Management: Java has automatic garbage collection, while C requires manual memory management.
       - Platform Independence: Java is platform-independent through the JVM, while C is platform-dependent.
       - Pointer Usage: Java does not use pointers explicitly, enhancing security and robustness, whereas C extensively uses pointers.
     - **Java vs Python:**
       - Performance: Java generally has better performance due to Just-In-Time (JIT) compilation, while Python is interpreted and tends to be slower.
       - Syntax: Java syntax is verbose and strict, requiring explicit declarations, whereas Python syntax is concise and easier to read.
       - Use Cases: Java is widely used for large-scale enterprise applications, whereas Python is favored for rapid development, scripting, and data science.

5. **Explain the `static` keyword in Java.**

   - **Answer:** The `static` keyword in Java indicates that a particular member (variable, method, or block) belongs to the class rather than instances of the class. This means static members are shared among all instances of the class.
   - **Example:**

```
class Example {
    static int count = 0;
    Example() {
        count++;
    }
}
public class Test {
    public static void main(String[] args) {
        new Example();
        new Example();
        System.out.println(Example.count); // Outputs: 2
    }
}
```

6. **What is the use of the `this` keyword in Java?**

   - **Answer:** The `this` keyword in Java is used to refer to the current object within a class. It helps to disambiguate instance variables from local variables when they have the same name.
   - **Example:**

```
class Example {
    int value;
    Example(int value) {
        this.value = value; // 'this.value' refers to the instance variable, 'value' to the parameter
    }
}
```

7. **Explain the various access specifiers in Java.**

   - **Answer:** Java provides four access specifiers:
     - **Private:** Accessible only within the same class.
     - **Default (no specifier):** Accessible within the same package.
     - **Protected:** Accessible within the same package and subclasses.
     - **Public:** Accessible from any other class.
   - **Example:**

```
class Example {
    private int a;
    int b; // default
    protected int c;
    public int d;
}
```

8. **What are collections in Java?**

   - **Answer:** Collections in Java are frameworks that provide architecture to store and manipulate groups of objects. They include classes like ArrayList, HashSet, and HashMap and interfaces like List, Set, and Map. Collections help to achieve dynamic data structures and enhance the performance of Java programs by providing algorithms for sorting, searching, and manipulating data.
   - **Example:**

```
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
System.out.println(list); // Outputs: [1, 2]
```

9. **What is garbage collection in Java?**

   - **Answer:** Garbage collection in Java is an automatic process of reclaiming memory by the JVM. When objects are no longer reachable, the garbage collector removes them to free up memory. This helps prevent memory leaks and optimizes resource usage.
   - **Example:**

```
public class GarbageCollectorDemo {
    public static void main(String[] args) {
        GarbageCollectorDemo obj = new GarbageCollectorDemo();
        obj = null; // Object is now eligible for garbage collection
        System.gc(); // Requesting JVM to perform garbage collection
    }
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Garbage collector called");
    }
}
```

10. **Difference between `Set`, `Map`, and `List`.**

    - **Answer:**
      - **Set:** A collection that does not allow duplicate elements. Example: `HashSet`.
      - **Map:** A collection of key-value pairs. Each key is unique. Example: `HashMap`.
      - **List:** An ordered collection that allows duplicate elements. Example: `ArrayList`.
    - **Example:**

```
Set<Integer> set = new HashSet<>();
set.add(1);
set.add(1); // No duplicate allowed

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

List<Integer> list = new ArrayList<>();
list.add(1);
list.add(1); // Duplicates allowed
```

11. **Why are strings immutable in Java?**

    - **Answer:** Strings in Java are immutable to enhance security, performance, and synchronization. Immutable objects are thread-safe and can be shared freely between threads without additional synchronization. Additionally, immutable strings allow for efficient memory usage through string pooling.
    - **Example:**

```
String s1 = "Hello";
String s2 = s1;
s1 = "World"; // s2 still references "Hello"
System.out.println(s2); // Outputs: Hello
```

12. **How to compare strings in Java?**

    - **Answer:** Strings in Java can be compared using the `equals()` method for content comparison and `==` operator for reference comparison.
    - **Example:**

```
String str1 = "Java";
String str2 = new String("Java");
System.out.println(str1.equals(str2)); // Outputs: true
System.out.println(str1 == str2); // Outputs: false
```

13. **What are wrapper classes in Java?**

    - **Answer:** Wrapper classes in Java provide a way to use primitive data types (int, char, etc.) as objects. Each primitive type has a corresponding wrapper class (Integer, Character, etc.). Wrapper classes are useful for collection frameworks that work with objects.
    - **Example:**

```
int num = 5;
Integer numObj = Integer.valueOf(num); // Boxing
int numAgain = numObj.intValue(); // Unboxing
```

14. **Explain the concept of Autoboxing and Unboxing.**

    - **Answer:** Autoboxing is the automatic conversion of primitive types to their corresponding wrapper classes. Unboxing is the reverse process. This feature simplifies the code by allowing seamless operations between primitives and their wrapper objects.
    - **Example:**

```
Integer obj = 10; // Autoboxing
int num = obj; // Unboxing
```

15. **What is the difference between `==` and `equals()` method in Java?**

    - **Answer:** The `==` operator compares references or memory addresses, while the `equals()` method compares the actual content of the objects.
    - **Example:**

```
String str1 = new String("Test");
String str2 = new String("Test");
System.out.println(str1 == str2); // Outputs: false
System.out.println(str1.equals(str2)); // Outputs: true
```

16. **How is the creation of a String using `new()` different from that of a literal, and explain String Pool in this context.**

    - **Answer:** When a string is created using a literal, it is added to the string pool, a special memory area in the JVM that stores unique string literals. If the same literal is used again, the JVM refers to the existing string in the pool. When a string is created using `new()`, a new object is created in the heap memory, even if an identical string exists in the pool.
    - **Example:**

```
String s1 = "Hello"; // Literal, added to string pool
String s2 = "Hello"; // Refers to the same object in string pool
String s3 = new String("Hello"); // New object in heap
System.out.println(s1 == s2); // Outputs: true
System.out.println(s1 == s3); // Outputs: false
```

17. **Explain the use of `break` and `continue` in loops.**

    - **Answer:**
        - **Break:** Terminates the loop or switch statement and transfers control to the statement immediately following the loop or switch.
        - **Continue:** Skips the current iteration of the loop and proceeds to the next iteration.

- **Example:**

```
for (int i = 0; i < 5; i++) {
    if (i == 2) break; // Exits loop when i is 2
    System.out.println(i);
}

for (int i = 0; i < 5; i++) {
    if (i == 2) continue; // Skips the iteration when i is 2
    System.out.println(i);
}
```

18. **Difference between checked and unchecked exceptions and examples of each.**

- **Answer:**
  - **Checked Exceptions:** Exceptions that are checked at compile-time. Example: `IOException`, `SQLException`.
  - **Unchecked Exceptions:** Exceptions that occur at runtime. Example: `NullPointerException`, `ArrayIndexOutOfBoundsException`.
- **Example:**

```
// Checked Exception
try {
    FileReader file = new FileReader("nonexistent.txt");
} catch (IOException e) {
    e.printStackTrace();
}

// Unchecked Exception
int[] array = new int[5];
int value = array[10]; // Throws ArrayIndexOutOfBoundsException
```

19. **Explain how exceptions propagate in the code.**

- **Answer:** When an exception occurs, it propagates up the call stack until it is caught by a corresponding `catch` block or terminates the program if uncaught. The JVM searches for the appropriate exception handler in the method where the exception occurred, then in the calling method, and so on.
- **Example:**

```
public class PropagationExample {
    public static void main(String[] args) {
        try {
            method1();
        } catch (Exception e) {
            System.out.println("Exception caught in main");
        }
    }

    static void method1() throws Exception {
        method2();
    }

    static void method2() throws Exception {
        throw new Exception("Exception in method2");
    }
}
```

20. **Explain the `final`, `finally`, and `finalize` keywords.**

- **Answer:**
  - **final:** Used to declare constants, prevent method overriding, and inheritance.
  - **finally:** A block that always executes after a `try-catch` block, used for cleanup activities.
  - **finalize:** A method invoked by the garbage collector before an object is reclaimed. It can be overridden to release resources.
- **Example:**

```
final int CONSTANT = 10;

try {
    // Code that may throw an exception
} catch (Exception e) {
    e.printStackTrace();
} finally {
    System.out.println("Cleanup code");
}

@Override
protected void finalize() throws Throwable {
    System.out.println("Finalize called");
}
```

21. **Difference between `throw` and `throws` in Java.**

- **Answer:**
  - **throw:** Used to explicitly throw an exception within a method or block.
  - **throws:** Used in method signatures to declare that a method can throw specified exceptions.
- **Example:**

```
void method() throws IOException {
    throw new IOException("Exception thrown");
}
```

22. **How does the `switch` statement work in Java?**

- **Answer:** The `switch` statement allows multi-way branching based on the value of an expression. It compares the expression with constant values and executes the corresponding block of code.
- **Example:**

```
int day = 3;
switch (day) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    default: System.out.println("Invalid day");
}
```

23. **Explain the difference between `ArrayList` and `LinkedList`.**

- **Answer:**
  - **ArrayList:** Uses a dynamic array, provides fast random access, and is efficient for read operations but slow for insertions and deletions.
  - **LinkedList:** Uses a doubly linked list, provides efficient insertions and deletions, and is slower for random access.
- **Example:**

```
List<Integer> arrayList = new ArrayList<>();
List<Integer> linkedList = new LinkedList<>();
```

24. **What is the `transient` keyword in Java?**

- **Answer:** The `transient` keyword is used in serialization to indicate that a field should not be serialized. This means the field will not be included in the serialized form of the object.
- **Example:**

```
class Example implements Serializable {
    private transient int transientField;
    private int regularField;
}
```

25. **What is the purpose of the `instanceof` operator?**

- **Answer:** The `instanceof` operator is used to test whether an object is an instance of a specific class or subclass. It helps in type checking and safe downcasting.
- **Example:**

```
if (obj instanceof String) {
    String str = (String) obj;
}
```