Java Lab Viva Questions with Detailed Answers

1. What is Java?

Java is a high-level, object-oriented programming language developed by Sun Microsystems (now Oracle). It is platform-independent due to its ability to compile code into bytecode which can run on any machine via the Java Virtual Machine (JVM).

2. What is JVM, JRE, and JDK?

- **JVM (Java Virtual Machine)**: Executes Java bytecode and provides platform independence.
- **JRE (Java Runtime Environment)**: Contains JVM and libraries to run Java applications.
- **JDK (Java Development Kit)**: Includes JRE and development tools like compiler (javac), debugger, etc.

3. Features of Java

- **Simple**: Easy to learn with familiar syntax
- **Object-Oriented**: Everything is based on objects and classes
- **Platform Independent**: "Write Once, Run Anywhere"
- **Secure**: No pointer manipulation, has a security manager
- **Robust**: Strong memory management and exception handling
- **Multithreaded**: Built-in support for multithreading
- **Portable**: Bytecode is platform-independent

4. What is bytecode?

Bytecode is the intermediate code generated by the Java compiler. It is platform-independent and is executed by the JVM.

5. Difference between == and .equals()

- **==**: Compares reference (memory address)
- **.equals()**: Compares actual content of objects

6. What is an object and class in Java?

- **Class**: A blueprint or template for creating objects.
- **Object**: An instance of a class.

7. What are constructors?

Constructors are special methods used to initialize objects. They have the same name as the class and no return type. They can be overloaded.

8. What is inheritance?

Inheritance allows a class (child class) to inherit properties and methods from another class (parent class). It supports code reuse and method overriding.

9. What is method overloading and overriding?

- **Overloading**: Same method name with different parameters (compile-time polymorphism).
- **Overriding**: Subclass provides specific implementation of a superclass method (runtime polymorphism).

Explanation of Method Overriding

Method Overriding allows a subclass to provide a specific implementation of a method already defined in its superclass. It is used to achieve runtime polymorphism.

Rules of Method Overriding:

1. The method must have the **same name**, **return type**, and **parameters** as in the parent class.
2. The method must be **inherited** from the superclass.
3. The access modifier of the overriding method must be the **same or more accessible** than the method in the superclass.
4. **Final**, **static**, or **private** methods cannot be overridden.
5. If the superclass method throws exceptions, the subclass can only throw the same or subclass exceptions.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

public class Test {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound();  // Output: Dog barks
    }
}
```

Here, the `sound()` method of `Dog` overrides the `sound()` method of `Animal`.

10. What is super keyword?

The `super` keyword in Java is used to refer to the immediate parent class object. It allows a subclass to access methods and variables of its superclass, even if they are overridden or hidden.

Uses of `super` keyword:

1. **Call Parent Class Constructor**:

```
super();
```

This must be the first statement in the subclass constructor.

1. **Access Parent Class Method** (especially if overridden in subclass):

```
super.methodName();
```

1. **Access Parent Class Variable** (if hidden in subclass):

```
System.out.println(super.variableName);
```

Example:

```
class Animal {
    String type = "Animal";

    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    String type = "Dog";

    void sound() {
        super.sound();  // Calls Animal's sound method
        System.out.println("Dog barks");
    }

    void printType() {
        System.out.println(super.type);  // Accesses Animal's type
```

```
    }
}
```

Output:

```
Animal makes a sound
Dog barks
Animal
```

11. What is encapsulation?

Encapsulation is the process of wrapping data (variables) and code (methods) into a single unit (class). It uses private fields and public getters/setters.

12. What is abstraction?

Abstraction hides implementation details and shows only essential features. It is achieved using abstract classes and interfaces.

13. What is interface in Java?

An interface contains only abstract methods (until Java 8 which introduced default and static methods). It is used to achieve 100% abstraction and multiple inheritance.

14. Difference between interface and abstract class

Feature	Abstract Class	Interface
Methods	Can have concrete methods	Only abstract (pre-Java 8)
Constructors	Can have	Cannot have
Multiple Inheritance	Not supported	Supported

15. What is polymorphism?

Definition: Polymorphism means "many forms". It allows a single interface or method to operate on different types or behave differently in different situations.

Types of Polymorphism in Java:

1. **Compile-time Polymorphism (Static Binding)**:

2. Achieved through **method overloading** (same method name with different parameters).

3. Decision is made at compile-time.

4. Use case: Adding flexibility to method definitions.

5. **Runtime Polymorphism (Dynamic Binding)**:

6. Achieved through **method overriding** (subclass provides its own implementation of a superclass method).

7. Decision is made at runtime.
8. Use case: Providing specific implementation in subclass.

16. What are access modifiers?

- **private**: Accessible within class only
- **default**: Accessible within package
- **protected**: Package + subclass
- **public**: Accessible everywhere

17. What is static keyword?

Definition: The `static` keyword in Java is used to indicate that a particular member (method, variable, block) belongs to the class rather than instances of the class.

Types and Use Cases:

- **Static Variables**:

- Shared by all instances of the class.

- Use case: Keeping track of data that should be common for all objects.

- **Static Methods**:

- Can be called without creating an object.

- Cannot access non-static members directly.
- Use case: Utility methods (e.g., `Math.pow()`).

- **Static Blocks**:

- Run once when the class is loaded.

- Use case: Initialization of static data.

- **Static Classes**:

- Inner static classes do not need an instance of the outer class.

Example:

```
class Example {
    static int count = 0;

    Example() {
        count++;
    }

    static void displayCount() {
        System.out.println("Count: " + count);
    }
}
```

18. What is final keyword?

Definition: The `final` keyword in Java is a non-access modifier used to restrict the user from modifying classes, methods, or variables once defined.

Types and Use Cases:

- **Final Variable**:
- A constant value that cannot be reassigned once initialized.
- Use case: Declaring fixed values such as `PI` or configuration constants.
- **Final Method**:
- A method that cannot be overridden by any subclass.
- Use case: Preserving the original behavior of the method in inheritance.
- **Final Class**:
- A class that cannot be subclassed.
- Use case: Preventing inheritance for security or design reasons (e.g., `String` class).
- **Final variable**: Constant
- **Final method**: Cannot be overridden
- **Final class**: Cannot be extended

19. Difference between throw and throws

Definition: Both `throw` and `throws` are used in Java exception handling but serve different purposes.

• ``:

• Used to actually throw an exception manually inside a method or block.

• Use case: When you want to throw a custom or specific exception.

• ``:

• Used in method declaration to inform the caller that this method might throw exceptions.

• Use case: Propagating exception responsibility to the calling method.

Keyword	Purpose	Used In
throw	Actually throws an exception	Inside method body
throws	Declares potential exceptions	In method declaration

Keyword	Purpose	Used In
throw	Actually throws an exception	Inside method body
throws	Declares potential exceptions	In method declaration

20. Exception Handling in Java

Definition: Exception handling in Java is a mechanism to handle runtime errors so that the normal flow of the application can be maintained. It helps prevent the program from crashing and provides meaningful messages.

Key Keywords and Their Roles:

- **try**: Defines a block of code to be tested for exceptions.
- **catch**: Defines a block of code to handle the exception.
- **finally**: Defines a block of code, always executed regardless of exception.
- **throw**: Used to explicitly throw an exception.
- **throws**: Declares exceptions that a method might throw.

Types of Exceptions:

1. **Checked Exceptions** (Compile-time):

2. Handled using try-catch or declared using `throws`.

3. Examples: `IOException`, `SQLException`.

4. **Unchecked Exceptions** (Runtime):

5. Not checked at compile time.

6. Examples: `ArithmeticException`, `NullPointerException`, `ArrayIndexOutOfBoundsException`.

Use Cases:

- Preventing application crash during file operations.
- Handling divide-by-zero or null pointer scenarios.
- Providing user-friendly error messages.

21. What is a package in Java?

Package is a namespace that organizes classes and interfaces. Declared using `package` keyword.

22. What is multithreading?

Multithreading allows concurrent execution of two or more threads. Achieved by:

- Extending `Thread` class
- Implementing `Runnable` interface

23. Difference between Array and ArrayList

Feature	Array	ArrayList
Size	Fixed	Dynamic
Type	Primitives & Objects	Objects only
Performance	Faster	Slower

24. What is garbage collection?

Automatic memory management process where JVM reclaims memory used by unreferenced objects. Triggered by `System.gc()` or automatically.

25. What is synchronization?

Definition: Synchronization is a technique that ensures only one thread can access a shared resource (e.g., a variable or block of code) at a time to avoid data inconsistency in multithreading.

Types of Synchronization:

1. **Synchronized Methods**:
2. Entire method is synchronized.
3. **Synchronized Blocks**:
4. Only specific section of code is locked.
5. **Static Synchronization**:
6. For static methods – locks on class level, not object level.

Use Cases:

- Banking applications (balance update)
- Logging systems
- Shared counters

Example:

```
public synchronized void increment() {
    count++;
}

// Block-level synchronization
synchronized(this) {
    count++;
}
```

26. What is wrapper class?

Converts primitives to objects:

- int → Integer
- float → Float
- char → Character

27. Explain the main() method

`public static void main(String[] args)` is the entry point:

- **public**: Called from anywhere
- **static**: Called without object
- **void**: Returns nothing

- **String[] args**: Command-line arguments

28. What are String, StringBuilder, and StringBuffer?

Definition: These are classes used in Java to handle and manipulate text (sequence of characters).

Types and Differences:

- **String**:
- Immutable (cannot be changed after creation)
- Thread-safe due to immutability
- Use case: Constant or rarely changing text
- **StringBuilder**:
- Mutable (can be modified)
- Not thread-safe
- Faster than StringBuffer
- Use case: Single-threaded dynamic text operations
- **StringBuffer**:
- Mutable and thread-safe (synchronized methods)
- Slower than StringBuilder
- Use case: Multi-threaded environments

29. Difference between == and .equals() for strings

- **==**: Compares references
- **.equals()**: Compares values

30. What is this keyword?

Refers to the current object in a method or constructor.

This document covers all essential Java viva topics with detailed answers suitable for lab exams and interviews.