1 – Explain different types of errors in Java.

Ans - Compile-time Errors: These errors occur during the compilation of the code. They are also called syntax errors. They indicate that the code violates the rules of the Java language, such as misspelled keywords, missing semicolons, incorrect method signatures, etc. Since the code cannot be compiled successfully, it won't run until these errors are fixed.

Runtime Errors (Exceptions): These errors occur during the execution of the program. They are typically caused by unexpected conditions or inputs, like dividing by zero, accessing an out-of-bounds array index, or trying to use a null reference. Java provides mechanisms, like try-catch blocks, to handle these exceptions and prevent program crashes.

Logic Errors: Also known as semantic errors, these errors don't cause the program to crash, but they lead to incorrect behavior or unexpected output. Logic errors are the result of incorrect program logic or flawed algorithms. The program runs, but the outcome is not what the programmer intended. Detecting and fixing logic errors often involves careful code review and testing.

2 – What is an Exception in Java?

Ans - An exception in Java is a runtime event that disrupts the normal flow of a program due to unexpected circumstances. It occurs when an error or exceptional condition occurs during program execution, such as dividing by zero or attempting to access a null reference. Java provides a built-in exception handling mechanism using try-catch blocks to gracefully manage these exceptional situations and prevent program crashes.

3 – How can you handle exceptions in Java? Explain with an example

Ans - Exception handling can be performed using three main blocks:

Try Block: The set of statements or code that might throw an exception is enclosed within this block.

Catch Block: This block catches and handles the exceptions that are thrown within the corresponding try block. Different catch blocks can handle different types of exceptions.

Finally Block: This block contains code that is always executed, regardless of whether an exception was thrown or caught. It's used to perform cleanup tasks that must be done, such as closing files or releasing resources.

Example - try {

```
    // Code that might cause an exception
    int result = 10 / 0; // This line throws an ArithmeticException
}
 catch (ArithmeticException e) {
    // Handling the specific exception
    System.out.println("An error occurred: " + e.getMessage());
}
finally {
    // Cleanup code or actions that should always occur
    System.out.println("Finally block executed.");
}
```

4 – Why do we need Exception Handling in Java?

Ans - Exception handling in Java is essential because it ensures that programs can gracefully manage unexpected errors during runtime, preventing crashes and maintaining stability. It also aids in effective debugging by providing error details for developers. Moreover, it enhances user experience by allowing controlled responses to errors, guiding users through unforeseen situations.

5 – What is the difference between exception and errors in Java?

Ans - Exceptions: These are unexpected events during runtime, often due to program logic issues or external factors. They can be handled using try-catch blocks, allowing the program to continue execution.

Errors: These are severe issues that typically arise from system-level problems, such as running out of memory. They are not usually handled by the program and can lead to termination of the application.

6 – Name different types of exceptions in Java.

Ans - Based on how they are handled by the Java Virtual Machine (JVM), there are generally two types of exceptions in Java:

Checked Exceptions: These occur during compilation. The compiler verifies whether the exception is handled by the code and enforces appropriate error handling.

Unchecked Exceptions: These emerge during program execution. They aren't detected during compilation but surface while the program runs, often due to logic errors or unforeseen circumstances.

7 - Can we simply use a try block without including catch and finally blocks? Provide an example.

Ans - No, attempting to do so will result in a compilation error. A try block must always be accompanied by either a catch or a finally block. While it's possible to omit either the catch or the finally block, having both absent is not permissible.