Java Exception Handling

```
- What is an exception?
- What are the different types of exceptions?
- How do you handle exceptions in Java?
```

Answer 1- An **exception** is an unwanted or unexpected event that occurs during the execution of a program. Exceptions can be caused by a variety of factors, such as user input errors, hardware failures, and network problems.

Java provides a robust exception handling mechanism that allows programmers to handle exceptions in a controlled manner. This is important because unchecked exceptions can cause a program to terminate unexpectedly.

Answer 2- There are two main types of exceptions in Java:

- **Checked exceptions:** Checked exceptions are exceptions that must be handled explicitly by the programmer. If a checked exception is not handled, the compiler will generate an error. Examples of checked exceptions include `IOException` and `SQLException`.
- **Unchecked exceptions:** Unchecked exceptions are exceptions that do not have to be handled explicitly by the programmer. The compiler will not generate an error if an unchecked exception is not handled. However, it is still generally considered good practice to handle unchecked exceptions. Examples of unchecked exceptions include `NullPointerException` and `ArrayIndexOutOfBoundsException`.

Answer 3- To handle exceptions in Java, you can use the `try-catch` statement. The `try` block contains the code that you want to monitor for exceptions. The `catch` block contains the code that will be executed if an exception occurs.

For example, the following code shows how to handle a checked exception:

Java

```
try {
  // Code that may throw a checked exception.
} catch (IOException e) {
  // Handle the IOException.
}
```

The following code shows how to handle an unchecked exception:

Java

```
try {
  // Code that may throw an unchecked exception.
} catch (NullPointerException e) {
  // Handle the NullPointerException.
}
```

You can also use the `finally` block to execute code regardless of whether or not an exception occurs. The `finally` block is typically used to release resources, such as database connections or file streams.

For example, the following code shows how to use the `finally` block to release a database connection:

Java

```
try {
  // Open a database connection.
} catch (SQLException e) {
  // Handle the SQLException.
} finally {
  // Close the database connection.
}
```

Exception handling is an important part of Java programming. By handling exceptions in a controlled manner, you can prevent your programs from terminating unexpectedly and provide a better user experience.