1. Introduction

In Java, exceptions are events that disrupt the normal flow of the program's instructions.

Exceptions are broadly classified into two main categories:

1. Checked Exceptions
2. Unchecked Exceptions.

Checked Exceptions are the exceptions that are checked at compile-time, meaning that the compiler requires these exceptions to be either handled with a try-catch block or declared in the method's signature with a throws clause.

Unchecked Exceptions include RuntimeExceptions, which are not checked at compile-time, and the compiler does not require them to be declared or handled.

2. Key Points

1. Checked Exceptions must be caught or declared to be thrown in the method signature.

2. Unchecked Exceptions include *`RuntimeExceptions`* and *`Errors`*, and they do not need to be caught or declared.

3. Checked Exceptions are known to the Java compiler, while Unchecked Exceptions could be bugs in the program.

4. Unchecked Exceptions usually indicate programming errors such as bad casting, accessing out-of-bounds arrays, or null pointers.

3. Differences

Checked Exception	Unchecked Exception
Checked at compile-time.	Not checked at compile-time.

Must be caught or declared in a throws clause.	Do not need to be caught or declared.
Encourages error handling and increases robustness.	Indicate programmer errors and affect the progra behavior.

4. Example

```
// Example of Checked Exception
try {
    // Simulating a checked exception
    throw new Exception("This is a checked exception");
} catch (Exception e) {
    System.out.println(e.getMessage());
}

// Example of Unchecked Exception
try {
    // Simulating an unchecked exception
    throw new RuntimeException("This is an unchecked exception");
} catch (RuntimeException e) {
    System.out.println(e.getMessage());
}
```

Output:

```
This is a checked exception
This is an unchecked exception
```

Explanation:

1. A Checked Exception is thrown and caught within a try-catch block; this is mandatory unless it's declared with a throws keyword in the method signature.

2. An Unchecked Exception (a subclass of `RuntimeException`) is also thrown and caught, but this is not mandatory. The program would compile without the try-catch block, and the exception would cause the program to terminate if it were not caught.

5. When to use?

- Use Checked Exceptions to handle recoverable conditions and to ensure robustness in your application by making the Java compiler check for error handling.

- Use Unchecked Exceptions to handle programming errors, knowing that these are not conditions that the caller of the method can reasonably be expected to recover from.