Here are five different examples of exception handling in Java, each showcasing a different type of exception:

1. Handling `NullPointerException`

```java
public class NullPointerExceptionExample {
    public static void main(String[] args) {
        String str = null;
        try {
            int length = str.length(); // This line will throw a NullPointerException
        } catch (NullPointerException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

In this example, we attempt to call the `length()` method on a null string, leading to a `NullPointerException`.

2. Handling `ArrayIndexOutOfBoundsException`

```java
public class ArrayIndexOutOfBoundsExample {
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3 };
        try {
            int value = arr[5]; // This line will throw an ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

This example tries to access an element at an index that is out of bounds in the array, resulting in an `ArrayIndexOutOfBoundsException`.

3. Handling `NumberFormatException`

```java
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
```

```
        String str = "abc";
        try {
            int num = Integer.parseInt(str); // This line will throw a NumberFormatException
        } catch (NumberFormatException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

Here, we attempt to parse a non-integer string into an integer using `Integer.parseInt()`, which will throw a `NumberFormatException`.

4. Handling Custom Exception

```java
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}

public class CustomExceptionExample {
    public static void main(String[] args) {
        try {
            throw new CustomException("Custom Exception Example");
        } catch (CustomException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

In this example, we define a custom exception `CustomException` that extends `Exception` and then throw and catch this custom exception.

5. Handling `IOException` (Checked Exception)

```java
import java.io.*;

public class IOExceptionExample {
    public static void main(String[] args) {
        try {
```

```
            FileReader fileReader = new FileReader("nonexistentfile.txt"); // This line may throw an IOException
            // Code to read from the file
            fileReader.close();
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

This example attempts to read from a file that does not exist, resulting in an `IOException`. `IOException` is a checked exception, so it must be either caught or declared in the method's throws clause.

These examples demonstrate various scenarios where exception handling is necessary in Java, including handling built-in exceptions and custom exceptions.