CYUSA GASHUGI Euclide
25916

1. IOException

Scenario: Attempting to read a file that does not exist, simulating an input/output error.

```
import java.io.*;

public class IOExceptionExample {
    public static void main(String[] args) {
        try {
            // Attempt to read a non-existent file
            FileReader file = new FileReader("nonexistentFile.txt");
            BufferedReader reader = new BufferedReader(file);
            reader.readLine();  // Will cause an exception if file doesn't exist
            reader.close();
        } catch (IOException e) {
            // Handle IOException (e.g., file not found or read error)
            System.out.println("IOException occurred: " + e.getMessage());
        } finally {
            // Always runs, even if exception is thrown
            System.out.println("Finally block: Cleaning up resources.");
        }
    }
}
```

- Explanation: The program attempts to read a file that doesn't exist, triggering an IOException.

2. FileNotFoundException

Scenario: Trying to open a file that doesn’t exist.

```
import java.io.*;

public class FileNotFoundExceptionExample {
    public static void main(String[] args) {
        try {
            // Try to open a non-existent file
            FileReader file = new FileReader("missingFile.txt");
            BufferedReader reader = new BufferedReader(file);
            reader.readLine();
            reader.close();
        } catch (FileNotFoundException e) {
            // Specific handling for a missing file
            System.out.println("FileNotFoundException occurred: " + e.getMessage());
        } catch (IOException e) {
            // Catch other IOExceptions
            System.out.println("IOException occurred: " + e.getMessage());
        }
    }
}
```

- Explanation: `FileNotFoundException` is a subclass of `IOException`. It occurs when the program tries to open a file that doesn't exist.

3. EOFException

Scenario: Reaching the end of the file unexpectedly while reading data.

```
import java.io.*;

public class EOFExceptionExample {
    public static void main(String[] args) {
        try {
            // Simulate reading past the end of a file
            FileInputStream file = new FileInputStream("testFile.txt");
            ObjectInputStream objectStream = new ObjectInputStream(file);
            objectStream.readObject();
        } catch (EOFException e) {
```

```
            // Handle EOFException when file reading ends prematurely
            System.out.println("EOFException occurred: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IOException occurred: " + e.getMessage());
        }
    }
}
```

- Explanation: An EOFException occurs when you attempt to read beyond the end of the file.

4. SQLException

Scenario: Trying to connect to a non-existent database or running an invalid query.

```
import java.sql.*;

public class SQLExceptionExample {
    public static void main(String[] args) {
        try {
            // Simulating a database connection error
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/nonexistentDB",
"user", "password");
            Statement stmt = conn.createStatement();
            stmt.executeQuery("SELECT * FROM invalidTable");  // Invalid SQL query
        } catch (SQLException e) {
            // Handle database-related errors such as connection issues or invalid queries
            System.out.println("SQLException occurred: " + e.getMessage());
        }
    }
}
```

- Explanation: The program tries to connect to a non-existent database, which triggers a SQLException.

5. ClassNotFoundException

Scenario: Trying to load a class at runtime that is missing.

```
public class ClassNotFoundExceptionExample {
    public static void main(String[] args) {
        try {
            // Trying to load a non-existent class at runtime
            Class.forName("com.example.NonExistentClass");
        } catch (ClassNotFoundException e) {
            // Handle ClassNotFoundException when the class can't be found
            System.out.println("ClassNotFoundException occurred: " + e.getMessage());
        }
    }
}
```

- Explanation: This exception is triggered when Java attempts to load a class using `Class.forName()` and the class is missing.

6. ArithmeticExceptionScenario: Attempting division by zero.

```
public class ArithmeticExceptionExample {
    public static void main(String[] args) {
        try {
            // Deliberate division by zero to trigger ArithmeticException
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            // Handle division by zero error
            System.out.println("ArithmeticException occurred: " + e.getMessage());
        }
    }
}
```

- Explanation: The program tries to divide a number by zero, which triggers an `ArithmeticException`.

7. NullPointerException

Scenario: Accessing a method or field on a null reference.

```
public class NullPointerExceptionExample {
    public static void main(String[] args) {
        try {
            // Deliberate null reference
            String str = null;
            // Attempting to access a method on a null object triggers NullPointerException
            int length = str.length();
        } catch (NullPointerException e) {
            // Handle null pointer exception
            System.out.println("NullPointerException occurred: " + e.getMessage());
        }
    }
}
```

- Explanation: This exception occurs when the program attempts to use a null reference as if it were an object. The `catch` block handles the exception and provides an error message.

---

8. ArrayIndexOutOfBoundsException

Scenario: Attempting to access an array element beyond its valid index.

```
public class ArrayIndexOutOfBoundsExceptionExample {
    public static void main(String[] args) {
        try {
            // Accessing an invalid array index
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]);  // This index is out of bounds
        } catch (ArrayIndexOutOfBoundsException e) {
            // Handle array index out-of-bounds error
```

```
            System.out.println("ArrayIndexOutOfBoundsException occurred: " + e.getMessage());
        }
    }
}
```

- **Explanation**: The program tries to access an index (5) that is out of bounds for the array. The exception is caught and an error message is printed.

9. ClassCastException

Scenario: Attempting an invalid type cast.

```
public class ClassCastExceptionExample {
    public static void main(String[] args) {
        try {
            // Attempting to cast an Integer to a String (which is an invalid cast)
            Object obj = new Integer(10);
            String str = (String) obj;
        } catch (ClassCastException e) {
            // Handle invalid type cast
            System.out.println("ClassCastException occurred: " + e.getMessage());
        }
    }
}
```

- **Explanation**: The program attempts to cast an `Integer` object to a `String`, which is invalid and causes a `ClassCastException`. The exception is caught, and a message is displayed.

10. IllegalArgumentException

Scenario: Passing an invalid argument to a method.

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

```
        try {
            // Passing a negative value to a method that only accepts positive numbers
            setAge(-5);
        } catch (IllegalArgumentException e) {
            // Handle the invalid argument passed to the method
            System.out.println("IllegalArgumentException occurred: " + e.getMessage());
        }
    }

    // Method that throws IllegalArgumentException for invalid arguments
    public static void setAge(int age) {
        if (age < 0) {
            throw new IllegalArgumentException("Age cannot be negative.");
        }
        System.out.println("Age set to: " + age);
    }
}
```

- Explanation: This program throws an IllegalArgumentException when a negative age is passed to the setAge method. The exception is caught, and the error message is displayed.

11. NumberFormatException

Scenario: Attempting to convert a non-numeric string to a number.

```
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
 try
{
 // Attempting to convert a non-numeric string to an integer int number =
Integer.parseInt("abc123");
}
 catch (NumberFormatException e)
 { // Handle invalid number format
 System.out.println("NumberFormatException occurred: " + e.getMessage());
 }
```

```
}
}
```

- **Explanation**: The program tries to convert the string `"abc123"` into an integer, which is not a valid number format. This causes a `NumberFormatException`, and the exception is caught and handled.