**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG**

**Department of Computer science and Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025-2026** | **Estimated Time** | **02 Hours** |
| **Course & Semester** | **S.E. CSE** | **Subject Name** | **Object Oriented**  **Programming with Java Lab** |
| **Module No.** | **06** | **Chapter Title** | Handling exceptions in Java |
| **Experiment Type** | **Software Performance** | **Subject Code** | 25PCC12CS07 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student** | Atharva Dharmendra Jagtap | **Roll No.** | 10937 |
| **Date of Performance:** | 05-10-2025 | **Date of Submission:** | 05-10-2025 |
| **CO Mapping** | CO3: Explore multithreading, File I/O, and exception handling | | |

**Objective of Experiment:** Exception Handling: Handling exceptions in Java (try-catch-throw- throws-finally), User defined Exceptions

**Pre-Requisite:** Any programming language like C,C++

**Tools:** Java IDE

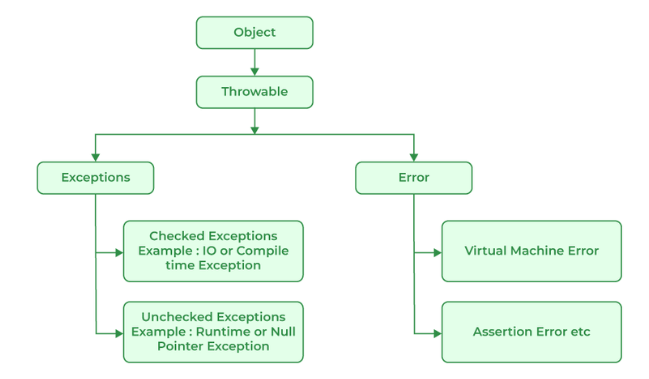
**Theory:** Exception Handling in Java is one of the effective means to handle runtime errors so that the regular flow of the application can be preserved. In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program’s instructions. Exceptions can be caught and handled by the program.

**Some Major reasons why an exception Occur:**

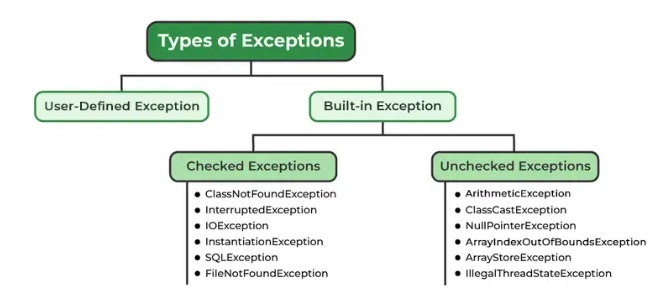
* Invalid user input
* Out of bound
* Null reference
* Type mismatch
* Opening an unavailable file
* Database errors
* Arithmetic errors

**Exception Hierarchy**

All exception and error types are subclasses of the class Throwable. Exception class is used for exceptional conditions that user programs should catch. Error is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself (JRE).



**Types of Exceptions:**

****

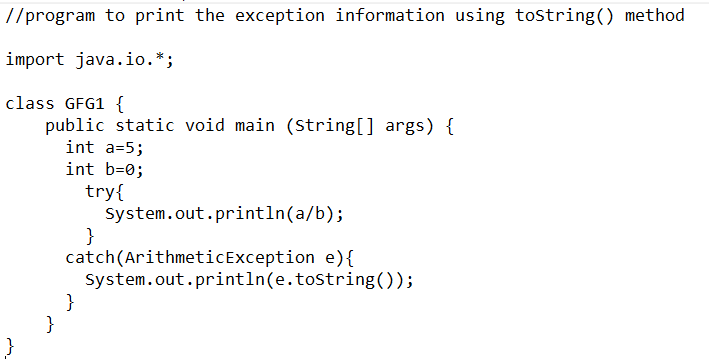
* ***Built-in exceptions*** are the exceptions that are available in Java libraries.
* ***Checked Exceptions:*** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
* The ***unchecked exceptions*** are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time.
* ***User-Defined Exceptions:*** Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called ‘user-defined Exceptions

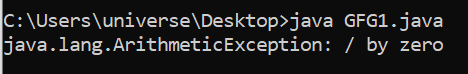
**Advantages of Exception Handling in Java**

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Propagation of Errors
4. Meaningful Error Reporting
5. Identifying Error Types

**Methods to print the Exception information**

* printStackTrace()
* toString() : The toString() method prints exception information in the format of the Name of the exception: description of the exception.
* getMessage()



****

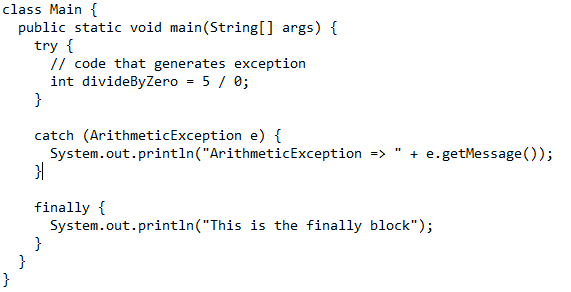
**Java Exception Keywords**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Keyword** | **Description** |
| **1.** | **try** | **The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.** |
| **2.** | **catch** | **The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.** |
| **3.** | **finally** | **The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.** |
| **4.** | **throw** | **The "throw" keyword is used to throw an exception.** |
| **5.** | **throws** | **The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.** |

**Common Scenarios of Java Exceptions:**

* If we divide any number by zero, there occurs an ***ArithmeticException***.
* If we have a null value in any variable, performing any operation on the variable throws a ***NullPointerException***.
* If the formatting of any variable or number is mismatched, it may result into ***NumberFormatException***. E.g. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.
* When an array exceeds to it's size, the ***ArrayIndexOutOfBoundsException*** occurs.

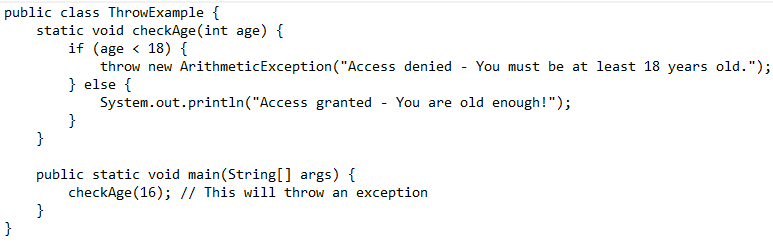
**Java Program using try, catch and finally keywords:**

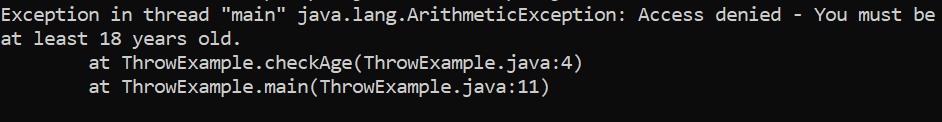




**Java Program using throw keyword:**

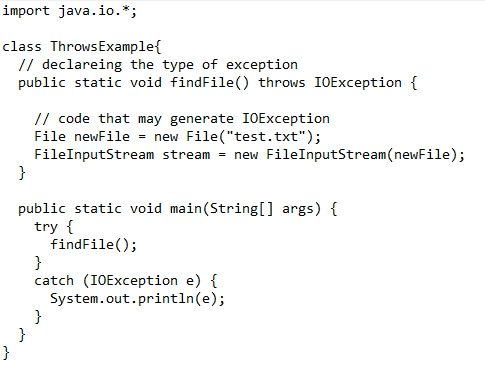
You can explicitly throw an exception using the throw keyword.





**Java Program using throws keyword:**

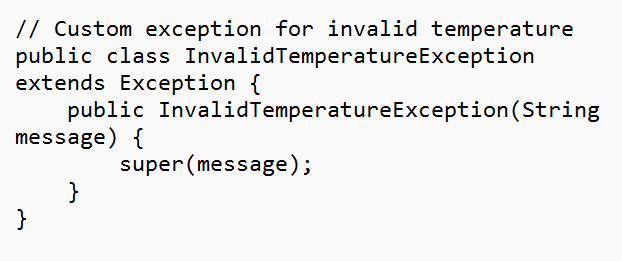
The throws keyword is used in method declarations to indicate that a method might throw an exception. This way, the method doesn’t need to handle the exception internally.

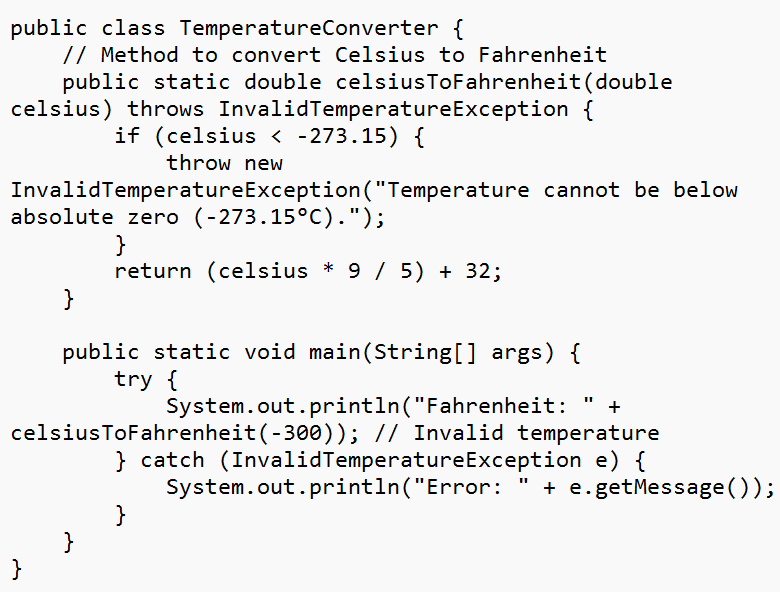




**Java Program with user-defined exception:**

User-Defined Exception for Temperature Validity

****

****

****

**Problem Description:**

|  |
| --- |
| ***Book store management System:*** Develop a program for a book store management system that handles exceptions such as invalid book id and book availability. |

**Code:**

import java.io.\*;

import java.util.\*;

// User-defined Exception for invalid book ID

class InvalidBookIdException extends Exception {

    public InvalidBookIdException(String message) {

        super(message);

    }

}

// User-defined Exception for unavailable books

class BookUnavailableException extends Exception {

    public BookUnavailableException(String message) {

        super(message);

    }

}

// Book Class

class Book {

    int id;

    String title;

    boolean available;

    public Book(int id, String title, boolean available) {

        this.id = id;

        this.title = title;

        this.available = available;

    }

    @Override

    public String toString() {

        return id + "," + title + "," + (available ? "Available" : "Unavailable");

    }

}

// BookStore Class (with Exception Handling and File I/O)

class BookStore {

    private List<Book> inventory = new ArrayList<>();

    public void addBook(Book b) {

        inventory.add(b);

    }

    public void checkBook(int id) throws InvalidBookIdException, BookUnavailableException {

        boolean found = false;

        for (Book b : inventory) {

            if (b.id == id) {

                found = true;

                if (!b.available) {

                    throw new BookUnavailableException("Book ID " + id + " is not available.");

                }

                System.out.println("Book found: " + b.title + " is available.");

                return;

            }

        }

        if (!found) {

            throw new InvalidBookIdException("Invalid Book ID: " + id);

        }

    }

    public void saveInventoryToFile(String filename) throws IOException {

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filename))) {

            for (Book b : inventory) {

                bw.write(b.toString());

                bw.newLine();

            }

            System.out.println("Inventory successfully saved to file.");

        }

    }

    public void loadInventoryFromFile(String filename) throws IOException {

        inventory.clear();

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

            String line;

            while ((line = br.readLine()) != null) {

                String[] parts = line.split(",");

                inventory.add(new Book(

                        Integer.parseInt(parts[0]),

                        parts[1],

                        parts[2].equalsIgnoreCase("Available")

                ));

            }

            System.out.println("Inventory loaded from file successfully.");

        }

    }

}

// Thread class for simulating concurrent access

class BookCheckThread extends Thread {

    private BookStore store;

    private int bookId;

    public BookCheckThread(BookStore store, int bookId) {

        this.store = store;

        this.bookId = bookId;

    }

    @Override

    public void run() {

        try {

            store.checkBook(bookId);

        } catch (InvalidBookIdException | BookUnavailableException e) {

            System.err.println("Thread " + Thread.currentThread().getName() + ": " + e.getMessage());

        }

    }

}

// Main class

public class BookStoreManagement {

    public static void main(String[] args) {

        BookStore store = new BookStore();

        store.addBook(new Book(1, "Cybersecurity Essentials", true));

        store.addBook(new Book(2, "Java Programming", false));

        store.addBook(new Book(3, "Data Structures", true));

        try {

            store.saveInventoryToFile("books.txt");

            store.loadInventoryFromFile("books.txt");

            Thread t1 = new BookCheckThread(store, 1);

            Thread t2 = new BookCheckThread(store, 2);

            Thread t3 = new BookCheckThread(store, 5); // Invalid ID

            t1.start();

            t2.start();

            t3.start();

            t1.join();

            t2.join();

            t3.join();

        } catch (IOException e) {

            System.err.println("File I/O Error: " + e.getMessage());

        } catch (InterruptedException e) {

            System.err.println("Thread interrupted: " + e.getMessage());

        } finally {

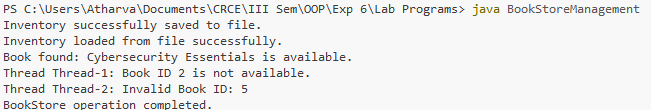
            System.out.println("BookStore operation completed.");

        }

    }

}

**Output:**

****

**PostLabQuestions:**

|  |
| --- |
| **Add code implemented during practical session.** |

**Code:**

// Main class demonstrating all three exceptions

public class PostLabBookExceptionDemo {

    // User-defined exception for invalid book ID

static class BookIdInvalid extends Exception {

    public BookIdInvalid(String message) {

        super(message);

    }

}

// User-defined exception for book not found

static class BookNotFound extends Exception {

    public BookNotFound(String message) {

        super(message);

    }

}

// User-defined exception for unavailable book

static class BookUnavailable extends Exception {

    public BookUnavailable(String message) {

        super(message);

    }

}

// Book class for testing

static class Book {

    int id;

    String title;

    boolean available;

    public Book(int id, String title, boolean available) {

        this.id = id;

        this.title = title;

        this.available = available;

    }

}

    // Method to check book details and raise user-defined exceptions

    static void checkBook(Book[] books, int id)

            throws BookIdInvalid, BookNotFound, BookUnavailable {

        // Check for invalid ID

        if (id <= 0) {

            throw new BookIdInvalid("Book ID cannot be zero or negative: " + id);

        }

        boolean found = false;

        for (Book b : books) {

            if (b.id == id) {

                found = true;

                if (!b.available) {

                    throw new BookUnavailable("Book '" + b.title + "' is currently unavailable.");

                } else {

                    System.out.println("Book Found: " + b.title + " (Available)");

                }

                break;

            }

        }

        // If book not found

        if (!found) {

            throw new BookNotFound("Book with ID " + id + " not found in the system.");

        }

    }

    public static void main(String[] args) {

        Book[] books = {

            new Book(1, "Java Programming", true),

            new Book(2, "Network Security", false),

            new Book(3, "Operating Systems", true)

        };

        int[] testIds = {1, 2, 5, -3};

        for (int id : testIds) {

            try {

                System.out.println("\nChecking Book ID: " + id);

                checkBook(books, id);

            } catch (BookIdInvalid e) {

                System.err.println("Error: " + e.getMessage());

            } catch (BookNotFound e) {

                System.err.println("Error: " + e.getMessage());

            } catch (BookUnavailable e) {

                System.err.println("Error: " + e.getMessage());

            } finally {

                System.out.println("Check operation completed for Book ID: " + id);

            }

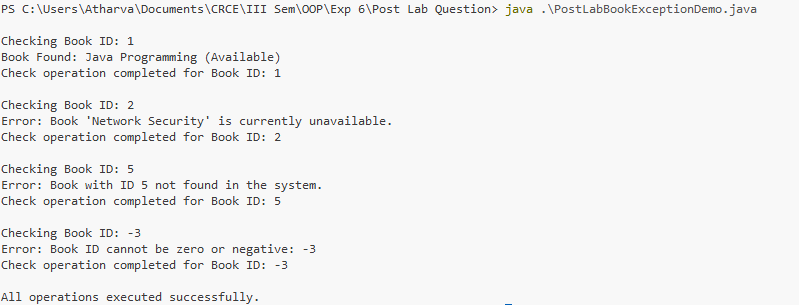
        }

        System.out.println("\nAll operations executed successfully.");

    }

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| **On time Completion and Submission (2)** | **Knowledge of the topic (4)** | **Implementation and Output (4)** | **Total (10)** |
|  |  |  |  |

**References:**

|  |  |
| --- | --- |
| **Study Materials**  [**https://www.w3schools.com/java/**](https://www.w3schools.com/java/)  [**https://www.geeksforgeeks.org/java/**](https://www.geeksforgeeks.org/java/)  https://www.codecademy.com/learn/learn-java | **Video Channels**:  [**https://www.youtube.com/user/programmingwithmosh**](https://www.youtube.com/user/programmingwithmosh)  [**https://www.youtube.com/c/TheNetNinja**](https://www.youtube.com/c/TheNetNinja)  [**https://www.youtube.com/c/Freecodecamp**](https://www.youtube.com/c/Freecodecamp)  [**https://www.youtube.com/user/Simplilearn**](https://www.youtube.com/user/Simplilearn) |
| **Study Materials used for Demo**  <Add links here> | |

**Note:-students are expected to paste screen shot of the program with output**