DIGITAL NOTES
ON
JAVA PROGRAMMING

FOR

COMPUTER ENGG. STUDENT

PROVIEDED BY: -

SHAKTI RAJ SINGH, PROGRAMMER

UNIT V

Exception Handling in Java

1. Definition of Exception Handling

Exception Handling in Java is a mechanism that allows a program to deal with runtime errors, enabling the program to continue its execution without crashing. An **exception** is an event that disrupts the normal flow of the program, such as an error in input, output, or even a logical error in the program.

Java provides a powerful mechanism to handle such errors through the **Exception Handling** mechanism, which helps in controlling the flow of the program and ensures that errors are caught and properly dealt with. This is done by using specific keywords like try, catch, finally, throw, and throws.

Key Concepts in Exception Handling:

1. **Exception**: An object that describes an abnormal condition in a program.
2. **Error**: A serious issue, usually outside the control of the program (e.g., OutOfMemoryError), which the program cannot typically recover from.
3. **Throwable**: The superclass of all errors and exceptions in Java. Both exceptions and errors are subclasses of the Throwable class.

2. Keywords Used in Exception Handling

There are several important keywords used in Java to implement exception handling. These keywords help in managing exceptions effectively:

i. try:

The try block is used to wrap the code that might cause an exception. If an exception occurs within the try block, the corresponding catch block (if provided) will handle it.

- **Syntax**:

```
try {
   // Code that may throw an exception
}
```

ii. catch:

The catch block is used to handle exceptions thrown by the try block. It can catch specific types of exceptions or all exceptions.

- **Syntax**:

```
try {
   // Code that may throw an exception
} catch (ExceptionType e) {
   // Code to handle the exception
}
```

iii. finally:

The finally block is optional and is used to execute code that must run regardless of whether an exception occurred or not. It is typically used to clean up resources (e.g., closing files or database connections).

- **Syntax**:

```
try {
   // Code that may throw an exception
} catch (ExceptionType e) {
   // Code to handle the exception
} finally {
   // Code that will always run, regardless of exception occurrence
}
```

iv. throw:
The throw keyword is used to explicitly throw an exception from within a method or block of code. It is used to signal the occurrence of an exception.

- **Syntax**:

```
throw new ExceptionType("Message");
```

v. throws:
The throws keyword is used in a method signature to declare that a method may throw an exception. It doesn't handle the exception but delegates it to the calling method for handling.

- **Syntax**:

```
public void methodName() throws ExceptionType {
   // Code that might throw an exception
}
```

3. Importance of Exception Handling in Practical Implementation of Live Projects
Exception handling is extremely important in real-world applications for the following reasons:

1. **Program Stability**: Handling exceptions ensures that your program doesn't crash unexpectedly. This makes the application more stable and user-friendly.
2. **Graceful Degradation**: If an exception occurs, instead of crashing the application, exception handling allows the program to either handle the exception or provide the user with meaningful error messages.
3. **Resource Management**: Exception handling allows proper management of system resources (like file handles, database connections) by using finally blocks to release resources regardless of whether an exception occurred or not.
4. **Debugging**: Proper exception handling provides a way to capture error details, including the stack trace, which aids in debugging the application.
5. **Maintainability**: By clearly defining exception handling mechanisms, it becomes easier to modify and maintain the code, especially when dealing with unforeseen errors or adding new features.

4. Advantages of Exception Handling

1. **Improved Program Flow Control**: Exception handling improves program control by allowing you to define how to handle errors and prevent them from propagating and causing program termination.
2. **Separation of Error-Handling Code from Regular Code**: It keeps error-handling code separate from normal code, making the code cleaner and easier to read.
3. **Enhanced Debugging and Monitoring**: Exceptions provide useful information such as stack traces, which helps in debugging. You can also use logging mechanisms to monitor exceptions.
4. **Prevention of Program Crashes**: By catching and handling exceptions, you can prevent the entire program from crashing, even when unexpected situations occur.
5. **Clearer Error Reporting**: Exception handling provides a way to throw meaningful error messages to users or other parts of the system, improving user experience and maintainability.

5. Disadvantages of Exception Handling

1. **Performance Overhead**: Exception handling introduces some performance overhead, especially when using try-catch blocks extensively, as it requires additional processing.
2. **Complexity**: In complex applications, handling multiple exceptions can lead to more complicated code structures, making it difficult to maintain or understand.
3. **Error Propagation**: Improperly handled exceptions may lead to unintended behavior, and exceptions that are not caught may propagate up to higher levels of the program, which can be dangerous if not handled properly.

4. **Unnecessary Handling**: Sometimes, developers overuse exception handling for situations where simple condition checks would suffice, which can complicate the code without adding much value.

6. Differentiation Between throw and throws

Feature	throw	throws
Purpose	Used to explicitly throw an exception in a method or block of code.	Used to declare that a method may throw one or more exceptions, without handling them.
Usage	Inside the method or block where the exception is thrown.	Inside the method signature to declare possible exceptions.
Syntax	throw new Exception("Message");	public void method() throws Exception { ... }
Control	Throws an exception and transfers control to the catch block or the calling method.	Does not handle exceptions, just informs the calling method that it may throw an exception.
Example	throw new ArithmeticException("Division by zero");	public void myMethod() throws IOException { ... }

7. Example with Explanation and Output

Example 1: Basic Exception Handling Using try, catch, and finally

```
public class Example1 {
   public static void main(String[] args) {
      try {
         int result = 10 / 0;  // This will throw ArithmeticException
      } catch (ArithmeticException e) {
         System.out.println("Error: Division by zero");
      } finally {
         System.out.println("This is the finally block");
      }
   }
}
```

Output:

```
Error: Division by zero
This is the finally block
```

- **Explanation**: In the above example, the try block contains code that may throw an exception (dividing by zero). When the exception occurs, the catch block handles the exception by printing an error message. The finally block runs regardless of whether an exception occurred or not, which is useful for cleanup operations.

Example 2: Throwing and Handling Custom Exception Using throw

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

public class Example2 {
```

```
    public static void validateAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or older");
        } else {
            System.out.println("Age is valid");
        }
    }

    public static void main(String[] args) {
        try {
            validateAge(15);  // This will throw InvalidAgeException
        } catch (InvalidAgeException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Output:
Exception: Age must be 18 or older

- **Explanation**: The custom exception InvalidAgeException is thrown using the throw keyword. The validateAge method checks if the age is less than 18 and throws the exception if the condition is met. The catch block then catches and prints the exception message.

Example 3: Declaring Exceptions with throws

```
class FileHandler {
    public void readFile() throws Exception {
        throw new Exception("File not found");
    }
}

public class Example3 {
    public static void main(String[] args) {
        FileHandler fileHandler = new FileHandler();
        try {
            fileHandler.readFile();  // Calling a method that throws an exception
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:
Error: File not found

- **Explanation**: The readFile method is declared with throws Exception, which indicates that it may throw an exception. The method is called in the try block, and the exception is caught in the catch block.