Here are detailed and structured notes based on the uploaded document:

Exception Handling in Java

Definition

- Exception handling is a mechanism to handle runtime errors and ensure the normal flow of the program.
- **Key Points:**
  - Exceptions occur at runtime, while syntax errors occur at compile time.
  - Exceptions are identified at runtime, not compile time.
  - The Java Virtual Machine (JVM) creates an exception object when a runtime error occurs.
  - If not handled, JVM terminates the program abnormally with an error message.
  - Proper handling allows for meaningful error messages and smooth program execution.

Advantages of Exception Handling

1. Maintains the normal flow of the program.
2. Provides flexibility in error handling.
3. Allows user-friendly error messages.
4. Separates "error-handling code" from "regular code."

Types of Exceptions

1. **Predefined Exceptions (Built-in Exceptions):**

   - Created by JVM for standard errors.
   - Organized as subclasses under `java.lang.Throwable`.
   - Example: `IOException`, `ArithmeticException`.

2. **Custom Exceptions (User-defined Exceptions):**

   - Created by programmers to handle specific cases.

Hierarchy of Java Exception Classes

1. **`Object`**:
   - Root of all Java classes.
2. **`Throwable`**:
   - Superclass for all exceptions and errors.
   - Two main subclasses:
     - **`Error`**:
       - Represents JVM failures (e.g., `StackOverflowError`, `OutOfMemoryError`).
       - Cannot be handled.
     - **`Exception`**:
       - Represents programmer logic failures (e.g., `ArithmeticException`).
       - Can be handled.

Differences Between `Error` and `Exception`

Aspect	Error	Exception
Source	JVM/system-related	Programmer logic-related
Handling	Cannot be handled	Can be handled
Examples	`VirtualMachineError`, `OutOfMemoryError`	`ArithmeticException`, `NullPointerException`

Predefined Exceptions

1. **Checked Exceptions:**
   - Identified at compile time, occur at runtime.
   - Example: `IOException`, `ClassNotFoundException`.
   - Must be handled in the program.
2. **Unchecked Exceptions (Runtime Exceptions):**
   - Identified and occur at runtime.
   - Example: `NullPointerException`, `ArrayIndexOutOfBoundsException`.
   - Handling is optional.

Exception Handling Process

1. Create an exception object for the logical error.
2. Throw the exception to the appropriate handler.
3. Catch the exception.
4. Take corrective actions.

Keywords for Exception Handling

1. **`try`**: Defines a block of code to monitor for exceptions.
2. **`catch`**: Handles exceptions thrown by the `try` block.
3. **`finally`**: Executes code regardless of exception occurrence.
4. **`throw`**: Throws a specific exception.
5. **`throws`**: Declares exceptions a method might throw.

Try-Catch Block

- **Syntax:**

```
try {
    // code that might throw an exception
} catch (ExceptionType e) {
    // exception handling code
}
```

- **Rules:**
  1. Must be within a method.
  2. Must be followed by at least one `catch` or `finally` block.
  3. No statements between `try` and `catch`.

4. Multiple `catch` blocks are allowed.

Finally Block

- Contains crucial cleanup code (e.g., closing resources).
- Always executed, regardless of exception occurrence.
- **Use Cases:**
    - Prevent resource leaks.
    - Perform cleanup tasks.

Throw vs Throws

Aspect	Throw	Throws
Purpose	Throws a specific exception	Declares potential exceptions
Usage	Within method body	In method signature
Multiple Exceptions	Not supported	Supported

User-Defined Exceptions

- Created by extending the `Exception` class.
- Example:

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

Chained Exceptions

- Technique to relate one exception with another.
- **Methods in `Throwable` for Chained Exceptions:**
    1. `getCause()` : Returns the cause of the exception.
    2. `printStackTrace()` : Prints the exception and its cause.

Assertions (JDK 1.4)

- **Purpose:**
    - Test program assumptions.
    - Debugging and identifying invalid logic.
- **Syntax:**

```java
assert condition;
assert condition : errorMessage;
```

These notes provide a concise yet comprehensive overview of exception handling in Java. Let me know if additional details or formatting adjustments are needed!