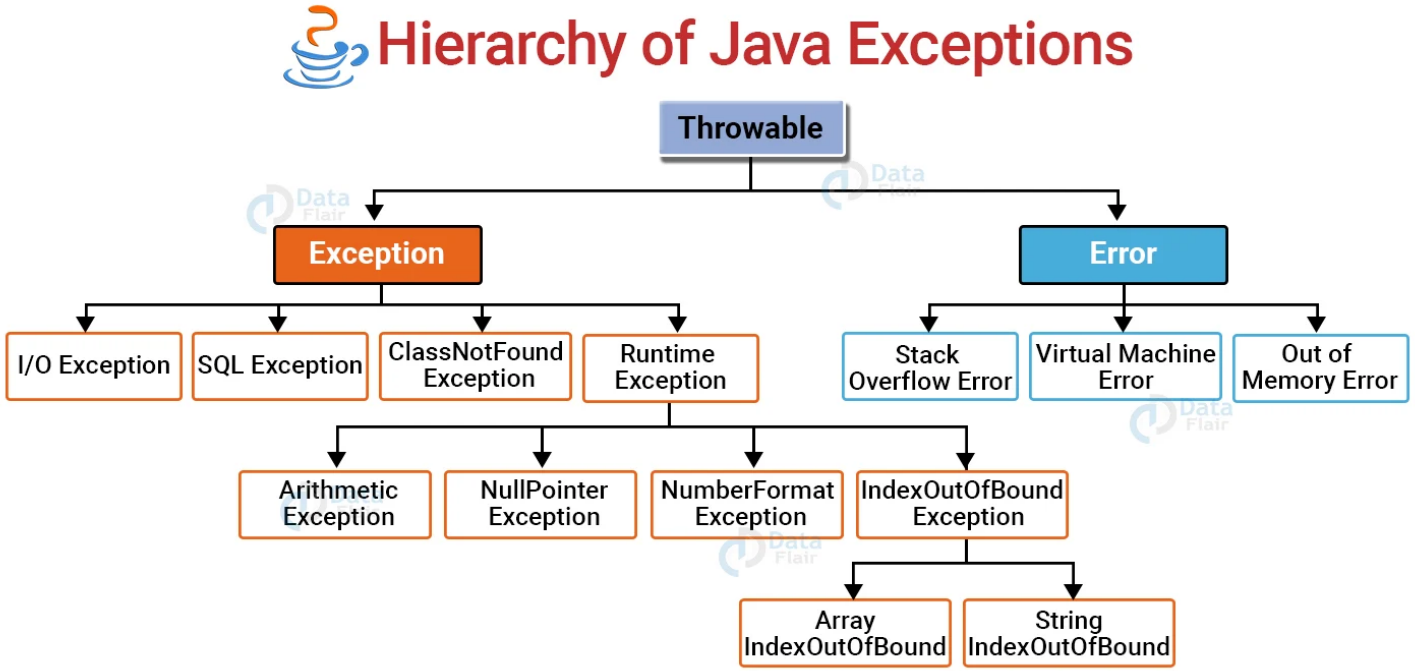
برنامه نویسی پیشرفته

رفع اشکال: جلسه ۷

Exceptions
Streams
Try-with-Resources
Generics

Exception Handling – Basics

- **Definition**: Mechanism to handle runtime errors
- **Types**:
  - Checked Exceptions (e.g., IOException)
  - Unchecked Exceptions (e.g., NullPointerException)



try/catch/finally

```
try {
    // risky code
} catch (ExceptionType e) {
    // handle error
} finally {
    // always executes
}
```

- **try:** Wraps code that might throw exceptions
- **catch:** Handles exceptions
- **finally:** Executes regardless of exception

Custom Exceptions

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

- Why? For meaningful, domain-specific error handling

Exception Handling – Questions

Q1. What is the difference between checked and unchecked exceptions?

Q2. Will the finally block execute if there's a return in the try block?

Answers – Exception Handling

A1.

- Checked exceptions are checked at compile-time (e.g., IOException) and must be declared or handled.

- Unchecked exceptions are subclasses of RuntimeException (e.g., NullPointerException) and are not checked at compile time.

A2.

- Yes, the finally block will always execute, even if there is a return in the try or catch block, unless the JVM exits or the thread is killed.

Try-with-Resources

- Introduced in Java 7
- Automatically closes resources

```
try (FileReader fr = new FileReader("file.txt")) {
    // use resource (calls fr.close() automatically at the end)

} // (calls fr.close() automatically at the end)
```

- Resource must implement AutoCloseable

Try-with-Resources – Questions

Q1. What are the advantages of using try-with-resources?

Q2. What happens if both the try block and the close() method throw exceptions?

Answers – Try-with-Resources

A1.

- Automatic resource management
- Cleaner and less error-prone code
- Ensures resources are closed properly even if an exception is thrown

A2.

- The exception from the try block is thrown.
- The exception from close() is suppressed and can be retrieved using getSuppressed() on the thrown exception.

File Streams in Java

Java I/O File Streams:

- Handle reading and writing of files
- Work with bytes (binary) or characters (text)

```
try (BufferedReader reader = new BufferedReader(new FileReader("data.txt"))) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Byte Streams vs Character Streams

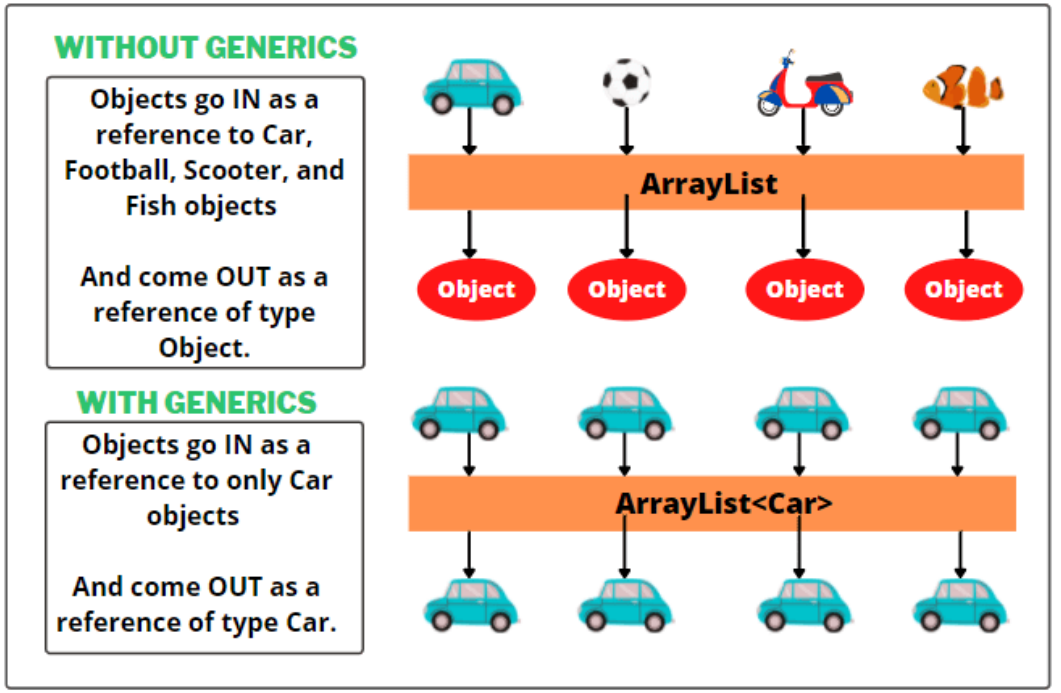
Feature	Byte Streams	Character Streams
Used For	Binary data (images, PDFs, etc.)	Text data (characters, strings)
Base Classes	InputStream / OutputStream	Reader / Writer
Encoding Awareness	Not aware of character encoding	Handles encoding (e.g., UTF-8, UTF-16)
Examples	FileInputStream, BufferedOutputStream	FileReader, BufferedWriter
Use Case Example	Reading a .jpg file	Reading a .txt file

- Use Buffered variants (BufferedReader, BufferedInputStream, etc.) for better performance!

Stream API Basics

- Introduced in Java 8 for functional-style operations on collections
- Stream pipeline: Source → Intermediate Ops → Terminal Op

```
List<String> names = List.of("Alice", "Bob");
names.stream()
     .filter(n -> n.startsWith("A"))
     .forEach(System.out::println);
```



Generics – What and Why?

- Allow type parameters for classes & methods
- Prevents runtime ClassCastException

```
List<String> list = new ArrayList<>();
```

Generic Methods & Classes

```
class Box<T> {
    T value;
    void set(T value) { this.value = value; }
    T get() { return value; }
}


<T> void printArray(T[] array) {
    for (T item : array) System.out.println(item);
}
```

Generics – Questions

Q1. What are the benefits of using generics in Java?

Q2. How do generic methods differ from regular methods?

Answers – Generics

A1.

- Type safety (compile-time checks)
- Eliminates need for casting
- Reusability of code for different data types

A2.

- Generic methods define their own type parameters (<T>)
- These can be used independently of the class's generic type (if any)
- They're more flexible for utility-type operations

Time to Code

We'll look at a short program that demonstrates:

- **Custom exception** for invalid age
- **Reading from a file** using try-with-resources
- **Generic method** to print a list of valid ages