ADVANCE JAVA

Advance Java Features

- Generics
- Annotations
- Reflection
- NIO
- Multi Threading
- JDBC
- JNI / JNA

JAVA GENERICS

Generics

- Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods.
- Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to re-use the same code with different inputs.
- The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types
- Benefits
  - Stronger type checks at compile time
  - Elimination of casts
  - Enabling programmers to implement generic algorithms

Generic Types

- A *generic type* is a generic class or interface that is parameterized over types.
- Example:

```
public class Box<T> {
      // T stands for "Type"
      private T t;
      public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

- Type Parameter Naming Convention
  - E - Element (used extensively by the Java Collections Framework)
  - K – Key
  - N – Number
  - T – Type
  - V – Value
  - S,U,V etc. - 2nd, 3rd, 4th types

Generic Concepts

- Generic Types
- Raw Types
- Bounded Type Parameters
- Type Inference
- Wildcards
  - Upper bounded wildcards — *e.g: ? extends Number*
  - Lower bounded wildcards — *e.g: ? super Integer*
  - *Unbounded* — *e.g: ?*
- Type Erasure

JAVA ANNOTATIONS

Annotations

- Annotations, a form of metadata, provide data about a program that is not part of the program itself

- **Use cases**
  - Information for the compiler
  - Compile-time and deployment-time processing
  - Runtime Processing

JAVA REFLECTION

Reflection

- An API that represents ("reflects") the classes, interfaces, and objects in the current Java Virtual Machine.
- Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine
- **Use cases**
  - Extensibility Features
  - Class Browsers and Visual Development Environments
  - Debuggers and Test Tools
- **Limitations**
  - Performance Overhead
  - Security Restrictions
  - Exposure of Internals

JAVA NESTED / INNER CLASSES

Nested/Inner Classes

- A nested class is a member of its enclosing class.
- Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.
- Static nested classes do not have access to other members of the enclosing class

- **Why Nested Classes**
  - It is a way of logically grouping classes that are only used in one place
  - It increases encapsulation
  - It can lead to more readable and maintainable code

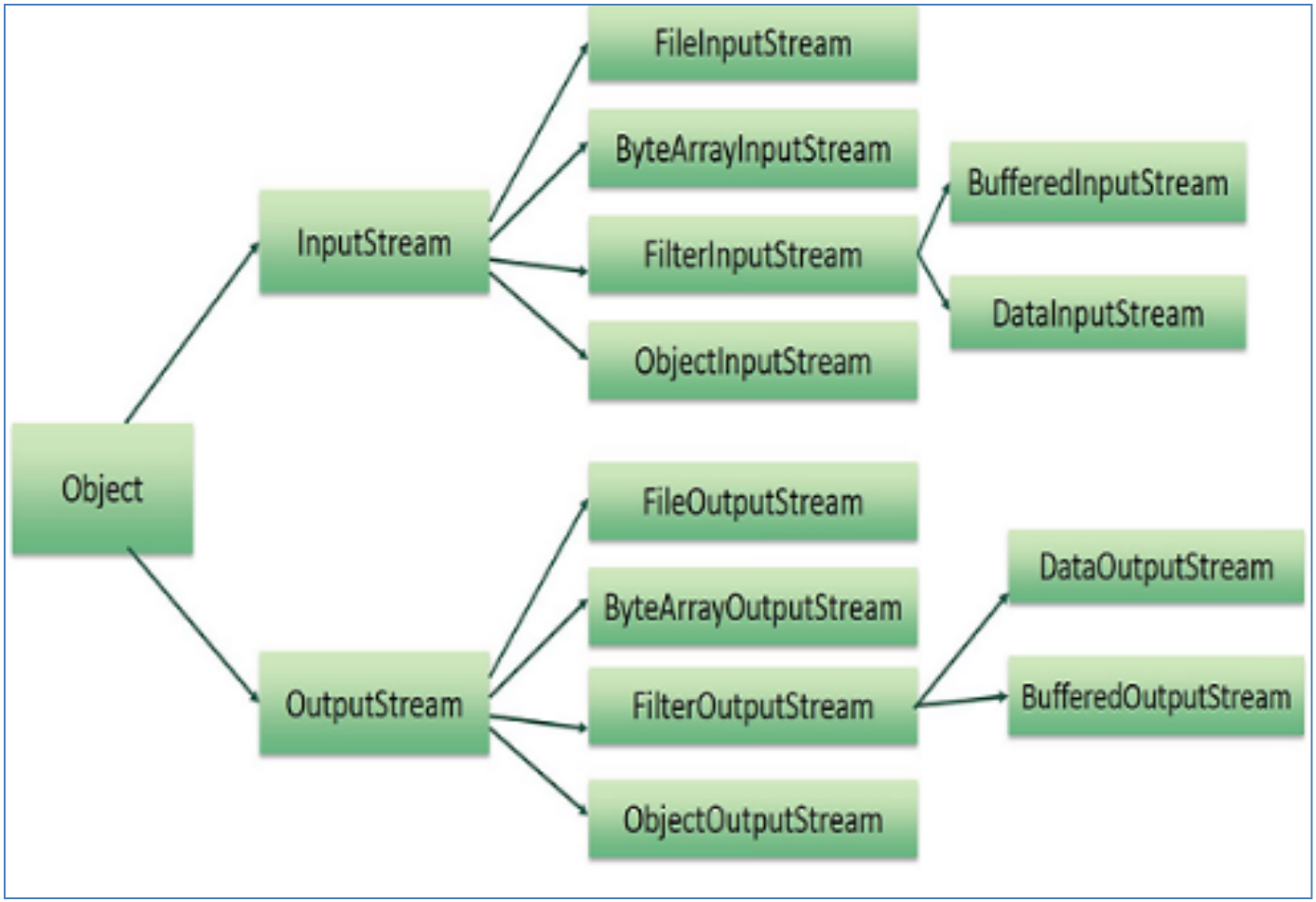
- **Types**
  - Static Nested Classes
  - Inner Classes (Non-static)
    - Local Inner Class -> declare an inner class within the body of a method
    - Anonymous Inner Class -> declare an inner class within the body of a method without naming the class

JAVA IO / NIO

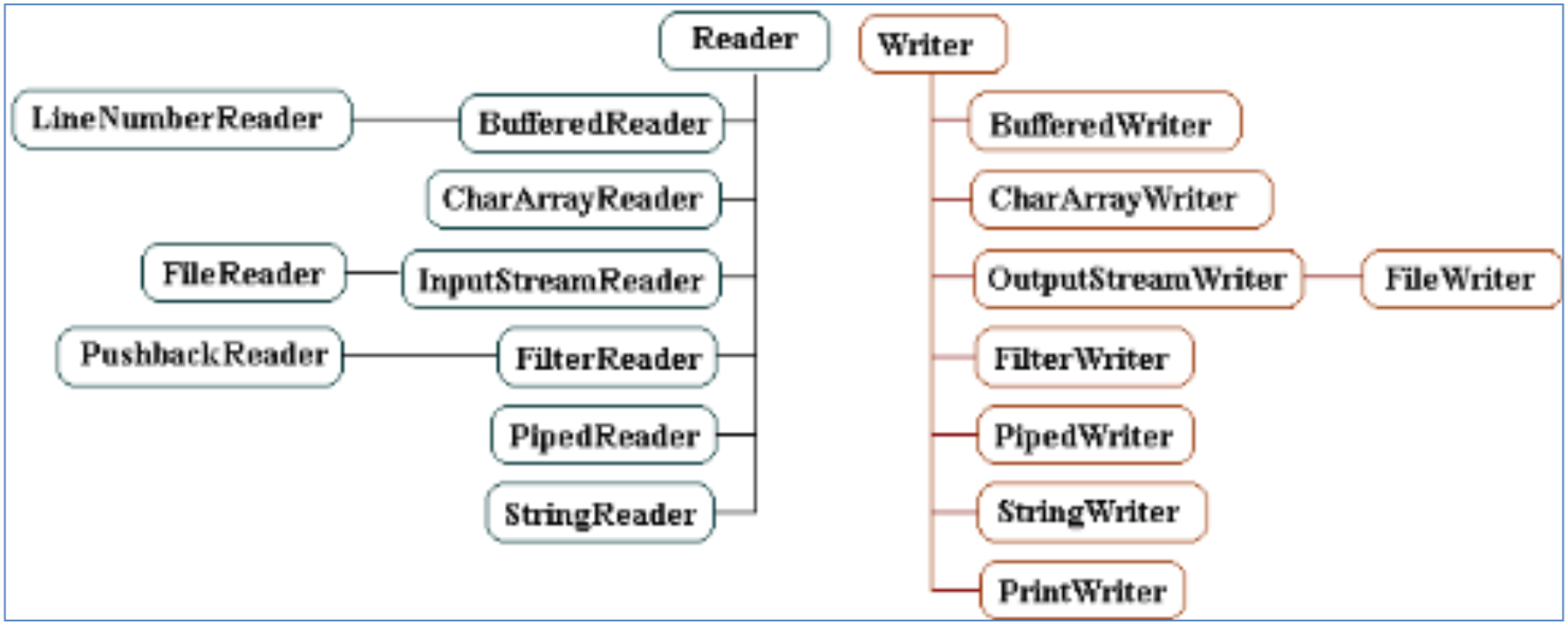
Java Serialization and I/O

- Serialization Overview
- I/O Streams Overview
- NIO (Non-blocking I/O Overview)

Byte Stream Hierarchy



Character Stream Hierarchy



Thank You!