Java 8 / 11

Lambda Expressions

Lambda Expressions

- Why?
  - Alternate way of creating anonymous class instances
- Advantages
  - Easier creation of anonymous class instances
  - More readable anonymous class instances
- Ex: Runnable, FileFilter, Comparator interfaces and …
- The type of Lambda Expression : a Functional Interface
- Lambdas can be stored in a variable
- Can be used along with method references

Lambda Expressions

- Collections processed using Lambda
- The forEach of Iterable<E>
  - Added in java 8 without breaking existing implementations
  - Default method : forEach
- The functional interface toolbox
  - Has several default methods

The Functional Interfaces Utilities

- New Package
  - java.util.function
- Categories
  - Consumer
  - Predicate
  - Function
  - Supplier

Stream API

Stream API

- Stream is a
  - Typed interface
  - An object
    - on which operations are defined
    - Which does not hold data
    - Does not change data during computation
    - Processes data in a single pass
    - That processes data in parallel
    - With optimized algorithms

Stream API

- Used for
  - Processing voluminous data
  - Processing smaller data too
- Processing mechanism
  - Parallel using multicore CPUs
  - Pipelined
  - Unnecessary intermediary computations are avoided
- Stream is totally new
  - Collections work in the same old fashion.

Backward Compatibility

- *Default methods* for interfaces
  - can still override it
  - don't have to
  - Available through implementing class
  - forEach() in Iterable interface
- reuse interfaces
  - as a type of lambda expressions
  - Runnable, FileFilter, Comparator etc…
- Static methods
  - Available through interface

Backward Compatibility

- Method Reference (::)
  - Used with
    - Instance & static methods
    - New keyword
- Loads of Functional Interfaces
  - The java.util.function package
  - The @FunctionalInterface annotation
- Streams
  - Works on existing collections
    - Sequential stream
    - Parallel stream

Date & Time API

The Design Principles

- Immutability:
  - All the classes in the new Date-Time API are immutable and good for multithreaded environments.
- Separation of Concerns:
  - The new API separates clearly between human-readable date time and machine time (Unix timestamp).
  - It defines separate classes for
    - Date,
    - Time,
    - DateTime,
    - Timestamp,
    - Timezone, etc.

The Design Principles

- Clarity:
  - Clearly defined methods perform the same action in all the classes.
  - For example, to get the current instance we have *now()* method.
  - There are format() and parse() methods defined in all these classes rather than having a separate class for them.
- Utility operations:
  - All the new Date-Time API classes come with methods to perform common tasks, such as plus, minus, format, parsing, getting the separate part in date/time, etc.

Commonly Used Classes

Class	Description
`LocalDate`	Represents a date (year, month, day (yyyy-MM-dd))
`LocalTime`	Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))
`LocalDateTime`	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)
`DateTimeFormatter`	Formatter for displaying and parsing date-time objects

Java 11

Java 11 New Features

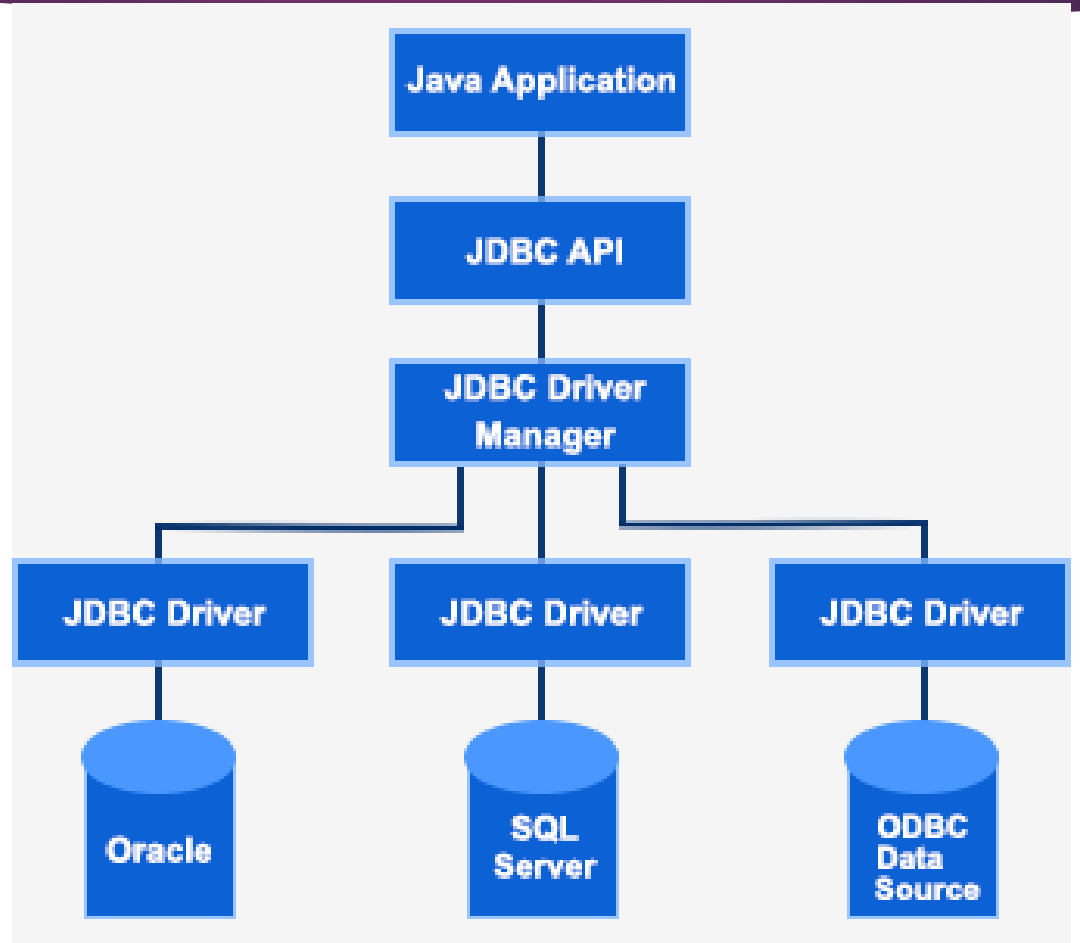
- The HTTP API
  - *HTTP API is a Java library to execute HTTP requests.*
  - Supports common HTTP methods like GET, POST, PUT, DELETE
  - Can handle synchronous and asynchronous communication
- Single-File Java Program
  - *execution of single file Java program with a single command.*
- *New String Methods*
  - String.repeat(Integer)
  - String.isBlank()
  - String.lines()

Java 11 New Features

- The HTTP API
  - *HTTP API is a Java library to execute HTTP requests.*
  - Supports common HTTP methods like GET, POST, PUT, DELETE
  - Can handle synchronous and asynchronous communication
- Single-File Java Program
  - *execution of single file Java program with a single command.*
- *New String Methods*
  - String.repeat(Integer)
  - String.isBlank()
  - String.lines()

JDBC

JDBC Architecture



JDBC Introduction

- Java Database Connectivity (JDBC) is an API specification
- Helps connecting applications written in Java to data in popular databases.
- The JDBC API lets you encode SQL statements that are then passed to the application that manages the database.
- It returns the results through a similar interface.

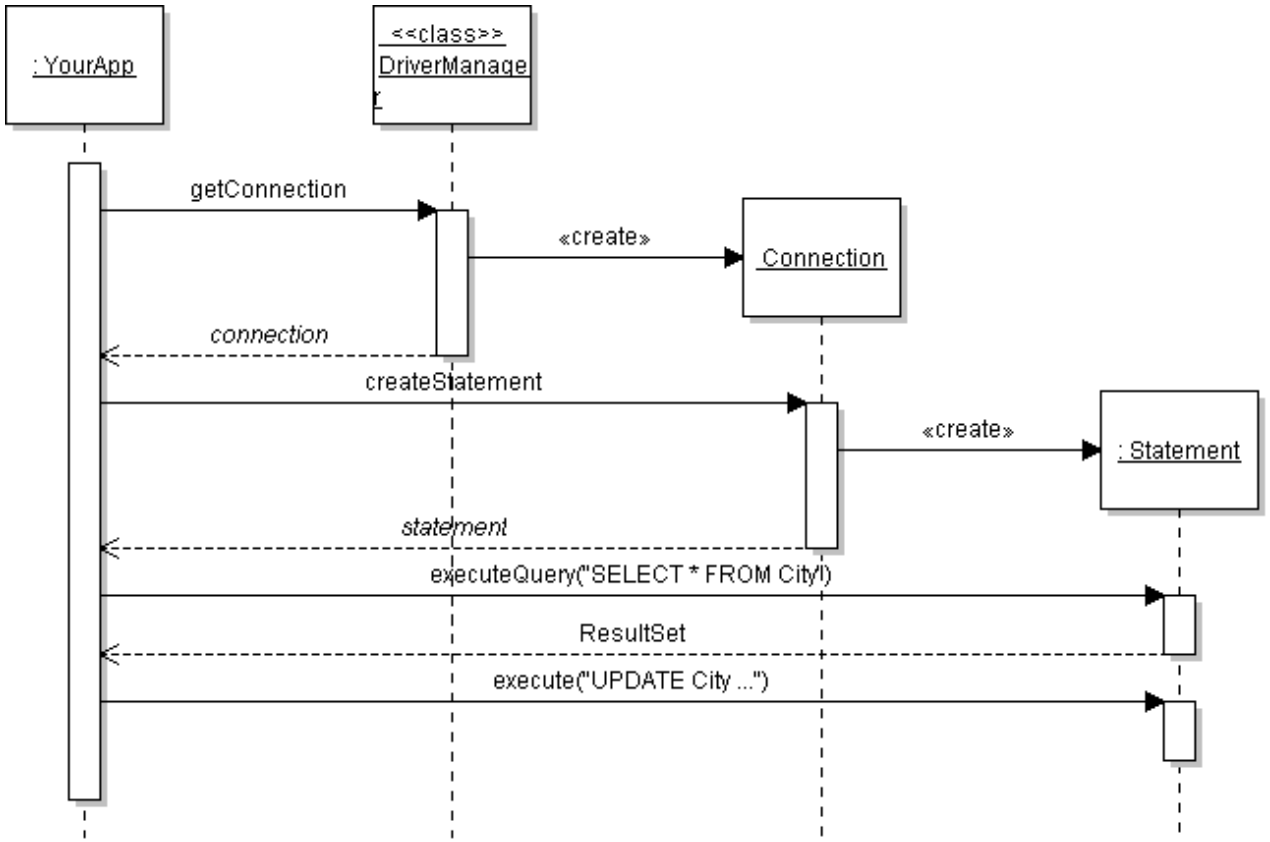
JDBC API

- Commonly used interfaces of the JDBC API:
  - Connection interface
  - Statement interface
  - PreparedStatement interface
  - CallableStatement interface
  - ResultSet interface
  - ResultSetMetaData interface
  - DatabaseMetaData interface

JDBC Steps

- Get a Connection to the database.
- Create a Statement using the Connection.
- Execute the Statement with SQL string.
- Use the results.

JDBC Sequence Diagram



Thank you