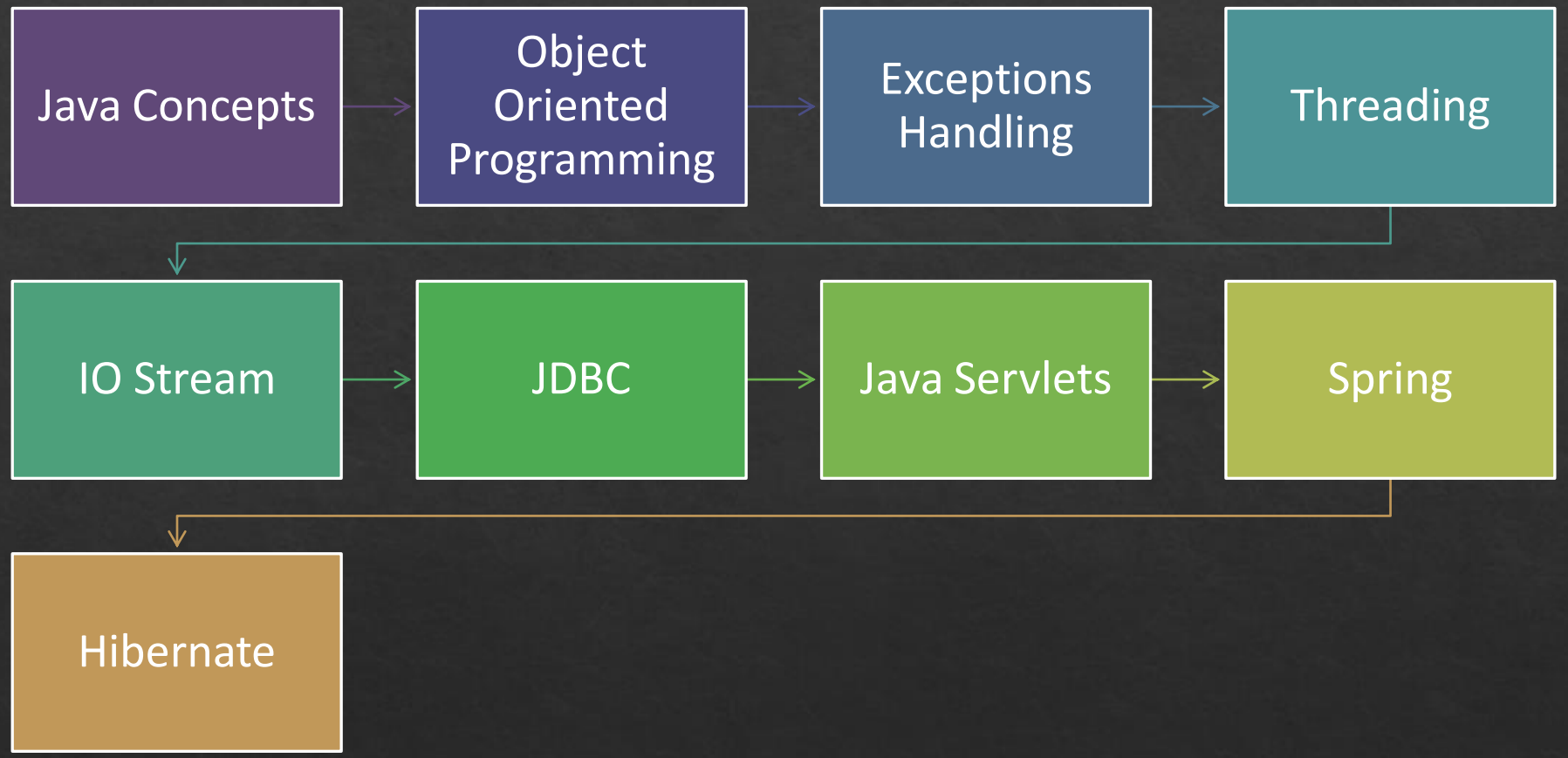
Advanced Java

ANIL JOSEPH

Agenda


Java Concepts
Object Oriented Programming
Exceptions Handling
Threading
IO Stream
JDBC
Java Servlets
Spring
Hibernate

Introduction

Anil Joseph

- Over 20 years of experience in Training and Development
- Technologies
  - C++
  - **Java, Enterprise Java and Java Frameworks**
  - .NET and .NET Core
  - Node, Node Express and other frameworks
  - UI Technologies: React, Angular, ExtJS, jQuery, Knockout, Redux etc
  - Mobile: Native Android and React Native
- Worked on numerous projects
- Conducted training for corporates(700+)

Software

JDK 1.8 or higher

Eclipse for JEE

Apache Tomcat

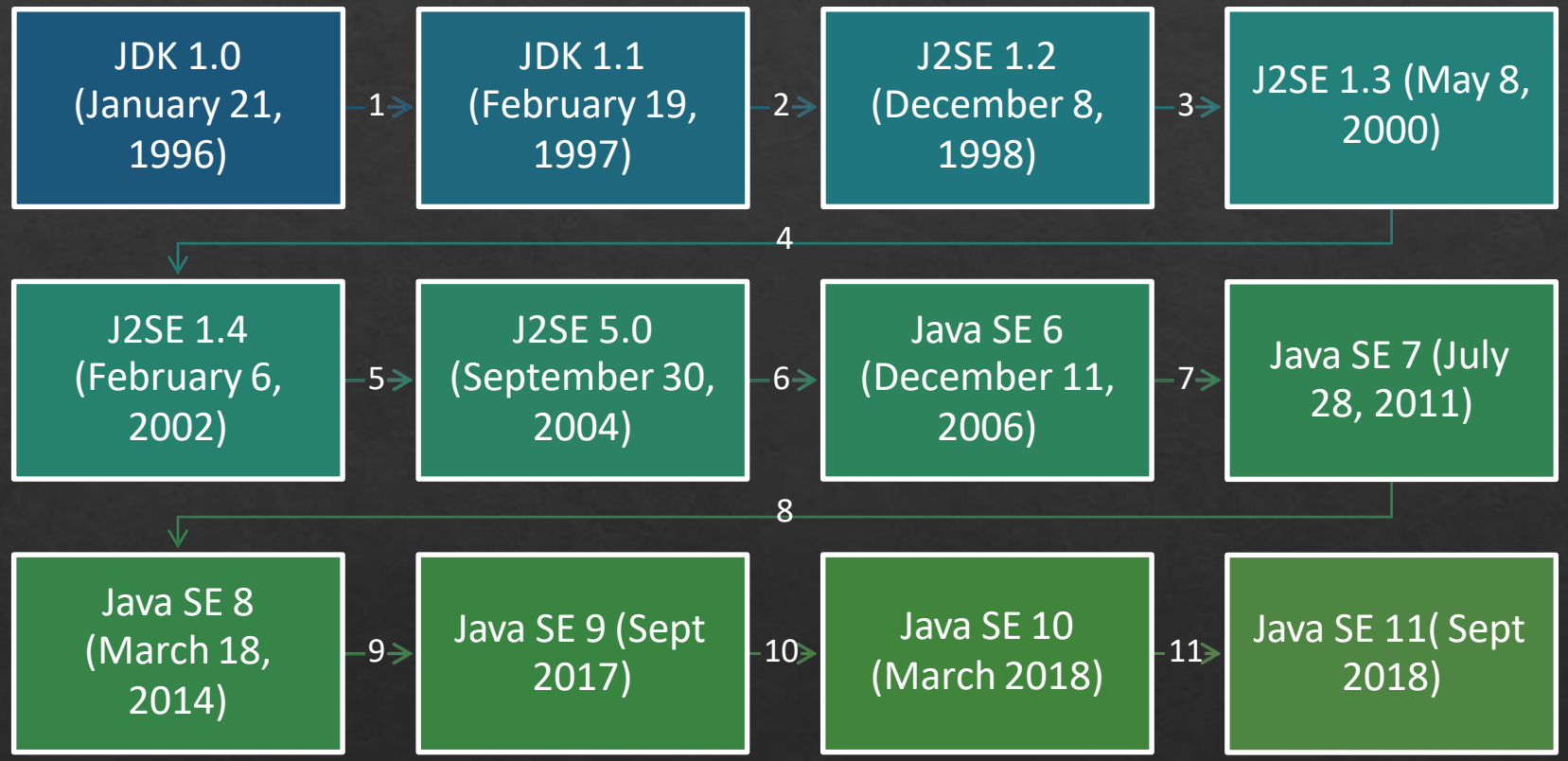
Maven

Apache Derby

Java

- Java is a general-purpose computer programming language
- Java software runs on every platform from laptops to data centers, gaming consoles to scientific computers
- Initially called OAK and then Project Green, Java was developed by James Gosling, Mike Sheridan and Patrick Naughton
- Sun Microsystem released the first public implementation as Java 1.0 in 1995
- Currently Java is maintained by Oracle corporation

Major Releases



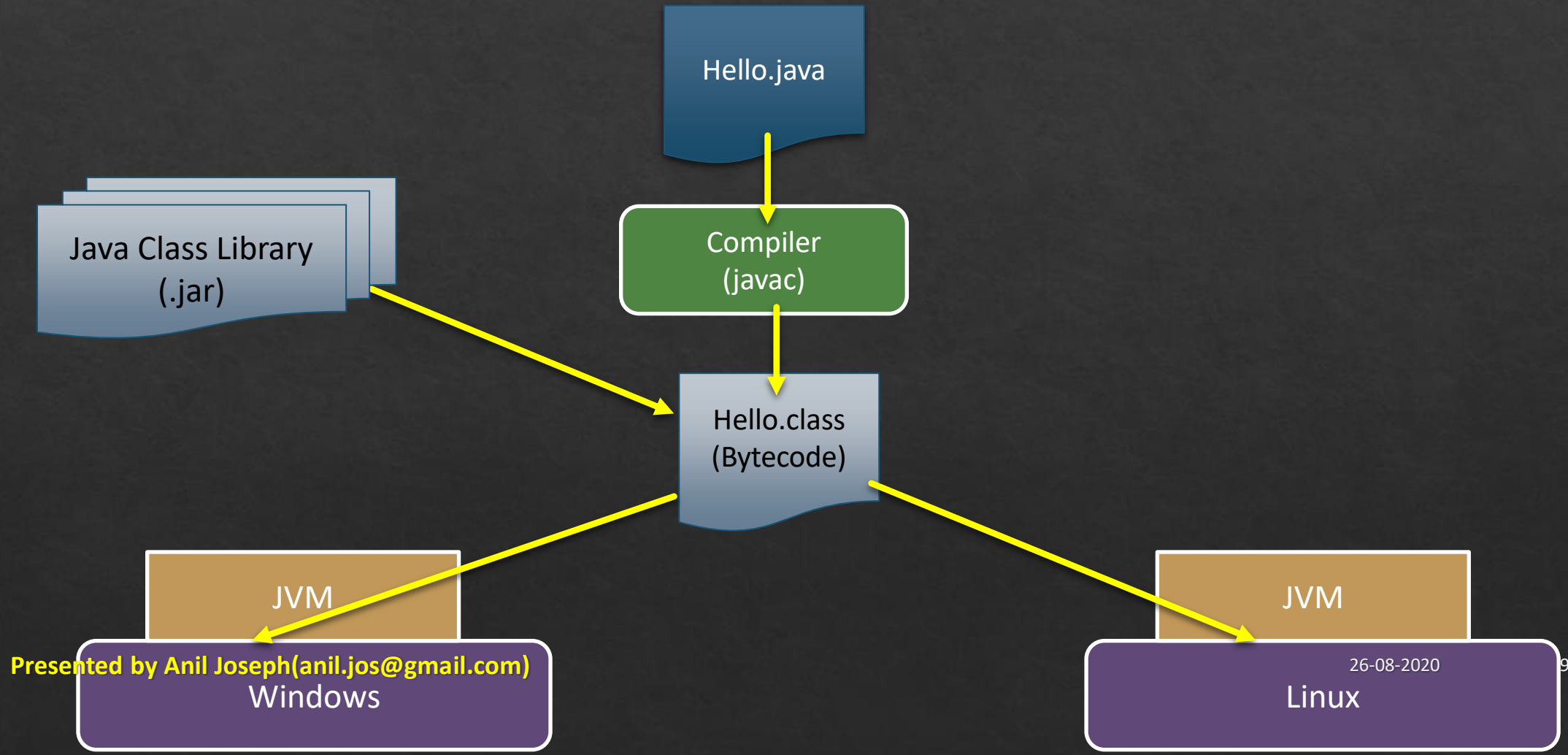
Java Language Features

Simple, Robust & Secure	Object Oriented	Automatic Object management	Platform Independent
Architecture Neutral	Distributed	Dynamic	Supported by a library(API)

Demo

- Hello World

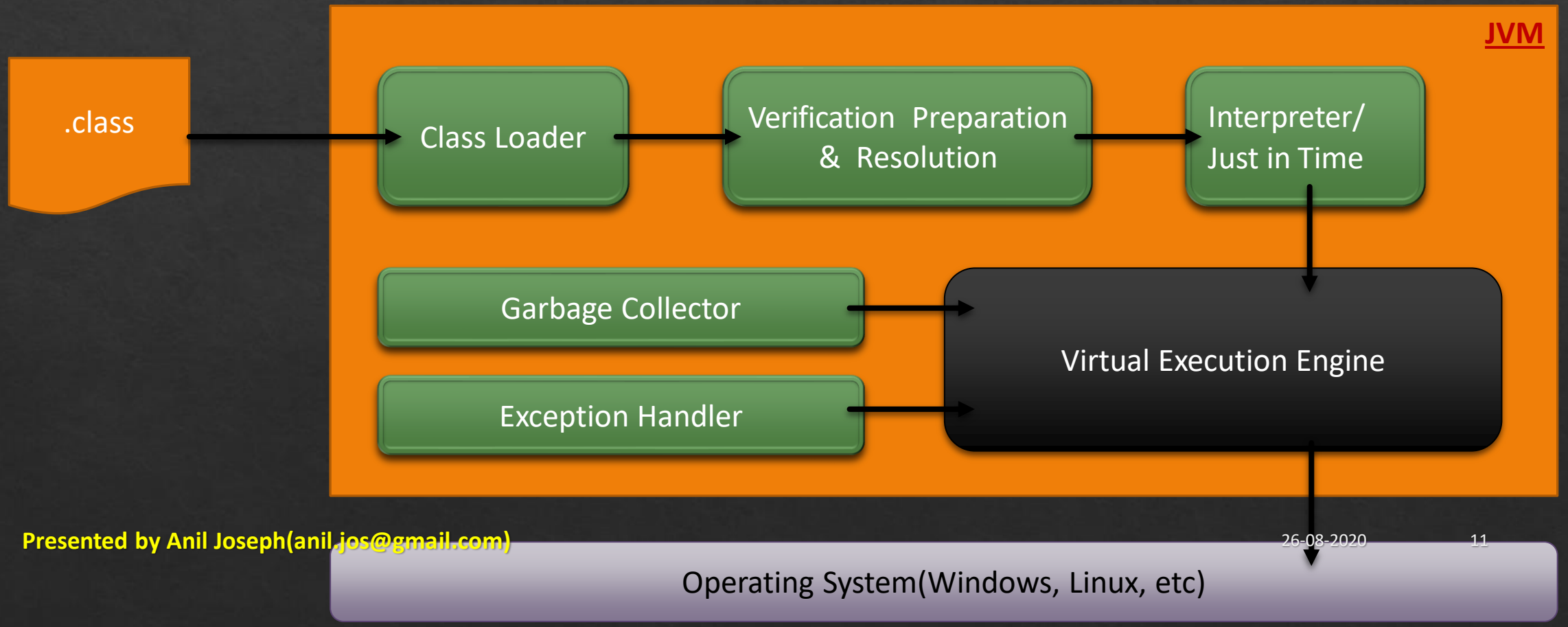
Compilation



Java Platform

- Java Runtime Environment
  - To execute Java Applications
  - Comprises of the JVM and the rt.jar
- Java Development Kit
  - To develop Java applications
  - Comprises of JRE and set of Java utilities
  - Also known as Java Standard Edition(JSE)
- Path and Classpath variables
  - Path variable is used to locate the Java tools executables like java and javac
  - Classpath is used by the JVM to locate the .class files at the time of execution

Java Execution Process



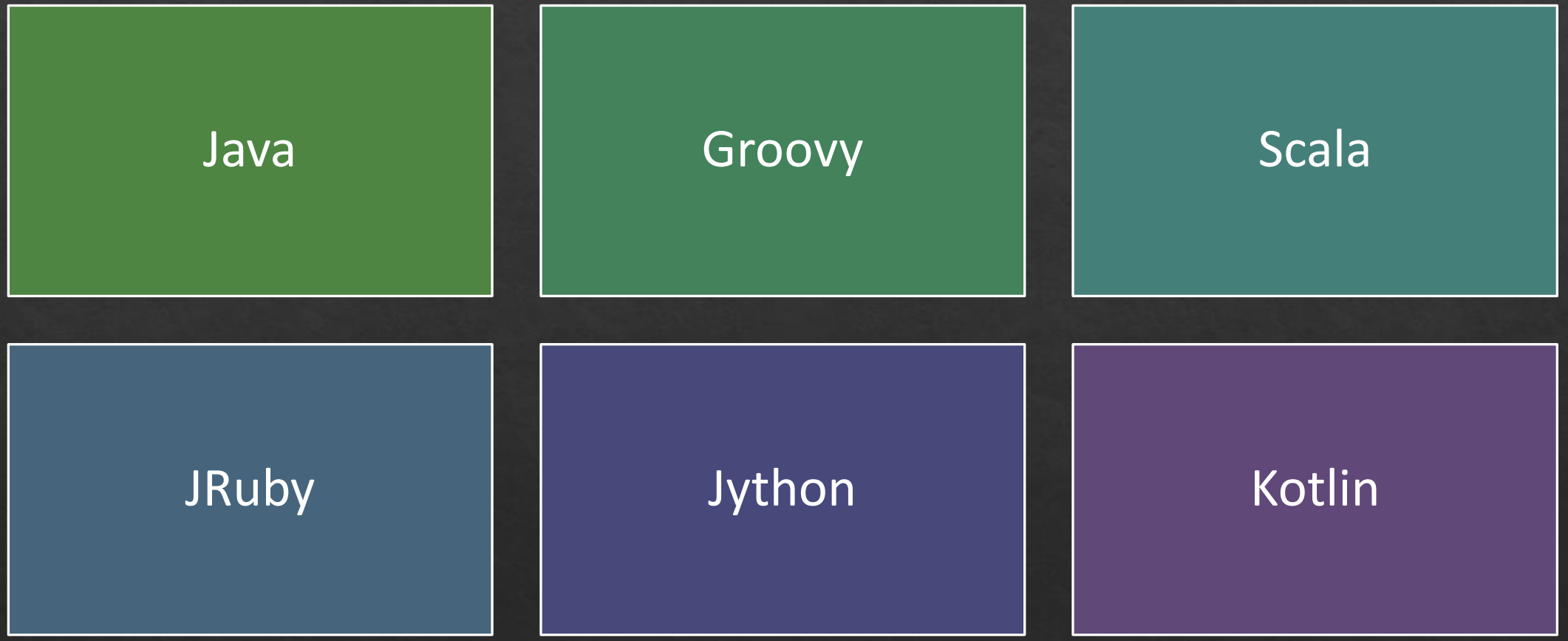
Java Execution Process

- Start of Execution
  - The Java command to execute a .class file
  - Java command loads the JVM
  - The JVM loads the .class into the memory
  - Verifies the code
  - Converts the bytecode to native code
  - Starts the execution from the entry point(main method)

Execution Process

- During Execution
  - Memory allocation happens on the stack and Heap
  - Garbage collector deallocates memory on the Heap(of any unused objects)
- End of Execution
  - When all active threads are over the application ends
- More on JVM
  - Supports custom class-loading
  - Supports several algorithms for Garbage Collection

JVM Languages



Java Types

- Primitive
  - boolean (true/false)
  - byte(1 byte)
  - short(2 bytes)
  - int(4 bytes)
  - long(8 bytes)
  - float(4 bytes)
  - double(8 bytes)
  - char(2 bytes)

Java Types

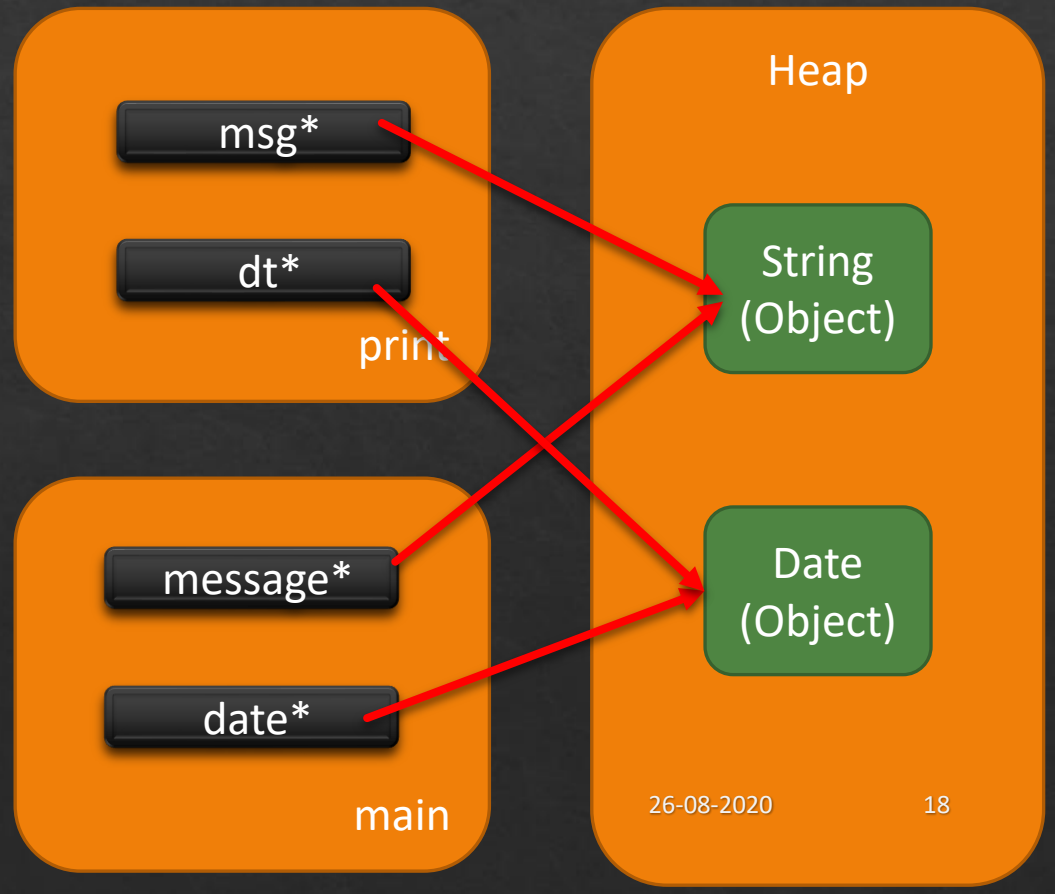
- Reference types(Heap based)
  - Objects(Instances of classes)
  - Anything apart from primitive types are references
  - All reference types inherit for java.lang.Object

Types

```
public class Types {

    public static void main(String[] args) {

        int x  = 100;
        System.out.println("X: "+  x);

        double interest = calcInterest(1000, 5, 10);
        System.out.println("interest: " + interest);

    }
    public static double calcInterest(double amt,
                            int roi, double term) {
        double interest =  (amt * roi * term)/100;
        return interest;
    }
}
```

Reference Types

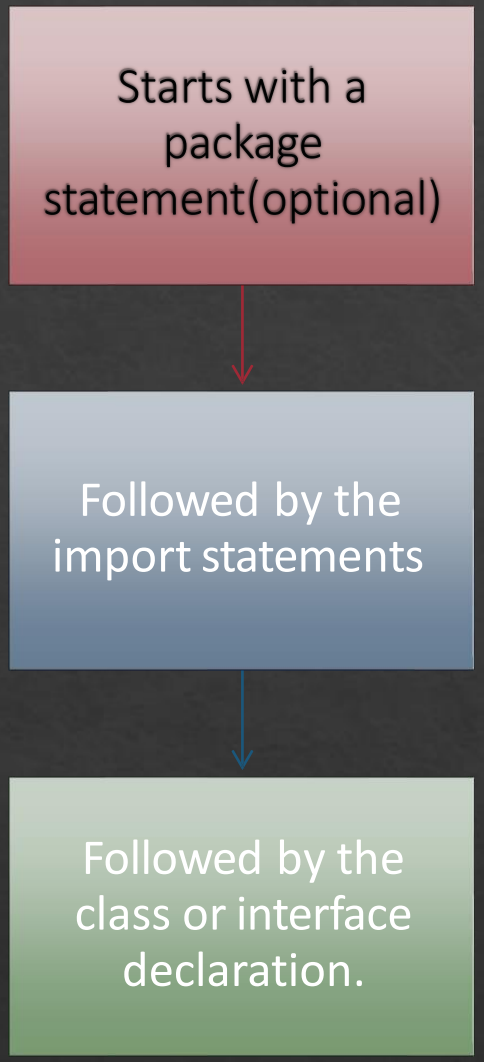
```java
public class Types {

    public static void main(String[] args) {

        String message = "Hello";
        LocalDate date = LocalDate.now();
        print(message, date);
    }

    public static void print(String message, LocalDate date) {
        System.out.println("Mesaage : "+ message);
        System.out.println("Date: "+ date.toString());
    }
}
```



Package

- A package in java allows to group similar class into the same package/folder.
- Used to differentiate between classes of the same name.
- A package is physical in java unlike C++ and .Net languages
- The import statement allows us to include a package in a .java file
- java.lang package is automatically imported.

Structure of a Java file



Object Oriented Programming

- Classes and Objects
  - Maps real time entities to objects
  - Class is a blueprint, object is an instance of a class
- Encapsulation
  - Combining state with behaviors
  - Hiding implementations and exposing interfaces
- Inheritance
  - Reusability of code
  - No or low maintenance
  - Easy extensibility
- Polymorphism
  - Different objects behaving differently to the same message
  - Different forms

Class

Data Members

- State

Member Functions

- Behaviors

Constructors

- Initialization

Finalize

- Cleanup

Access Modifiers

- Visibility

Static Members

- Static Fields
- Static Methods
- Static blocks

Access Modifiers

public

- Visible to all classes everywhere.

default(package-private)

- Visible only within its own package

private

- Can only be accessed in its own class.

protected

- Can only be accessed within its own package (as with *package-private*) and, in addition, by a subclass of its class in another package.

Static

Data members and methods of a class are available for every instance of the class. This means every instance has a copy of the members.

Static date members and methods are class-level, hence they are shared by all the instances of a class.

Since they are class-level they cannot be use the "***this***" keyword.

Static blocks can be used to initialize static data members

Class Keywords

abstract

- An abstract method does not have an implementation
- A class with an abstract method must be marked abstract
- An abstract class cannot be instantiated
- A class can be marked abstract even if it has not abstract methods

final

- final fields are constants
- final methods cannot be overridden
- final classes cannot have sub classes(cannot be inherited from)

Classes

Outer classes

- Can be either public or private-protected scope

Inner classes

- Can be declared with any access modifier
- Can access all members of the containing class
- Can be static

Local classes

- Class declared in a method
- Anonymous class
- Can access only the final parameters and final variables of the enclosing method

Inheritance

- In object-oriented programming, inheritance is the mechanism of basing an object or class upon another object or class, retaining similar implementation.
- Inheritance is based on an "**is a**" relationship
- A class that is derived from another class is called a subclass also a derived class, extended class, or child class.
- The class from which the subclass is derived is called a superclass also a base class or a parent class.
- A subclass inherits all the members (fields, methods, and nested classes) from its superclass.
- Constructors are not members, so they are not inherited by subclasses
- Constructors of the superclass can be invoked from the subclass.

Polymorphism

- Polymorphism is the concept that objects of different types can be accessed through the same interface.
- Two-types of polymorphism
  - Static or compile-time
  - Dynamic or run-time

Interfaces

- Interfaces are used to define contracts or specifications
- An interface is equivalent to an abstract class with only abstract methods
- Interfaces enable polymorphic behavior without inheritance
- A class can implement multiple interfaces

Generics

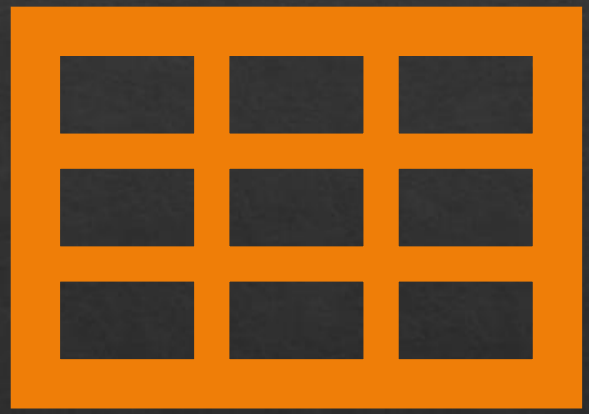
- Generics enable classes and interfaces to be parameters(type parameters) when defining classes, interfaces and methods.
- Type parameters provide a way for you to re-use the same code with different inputs.
- Advantages
  - Stronger type checks at compile time.
  - Elimination of casts
  - Enabling programmers to implement generic algorithms.

Generics: Bounded Type Parameters & Wildcards

- Bounded Type Parameters restrict the types that can be used as type arguments in a parameterized type.
- Example
  - <T extends A> : The type variable should be a subtype of A
  - <T extends A, B, C> : A type variable is a subtype of all the types listed in the bound.
- Wildcards
  - The question mark (?), called the wildcard, represents an unknown type.
  - Unbounded, Upper Bounded and Lower Bounded

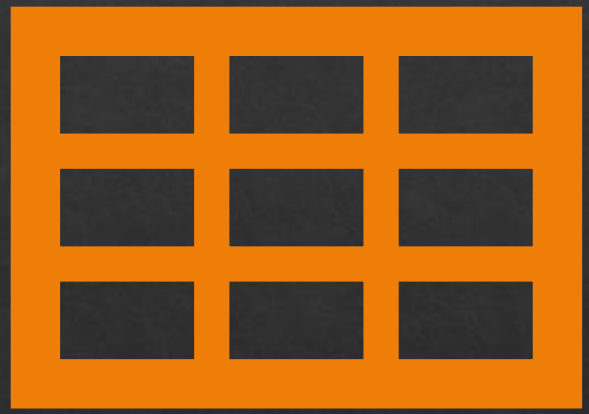
Collections

- A collection is an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve, manipulate, and communicate aggregate data.
- Typically, they represent data items that form a natural group, such as orders, products, customers etc.

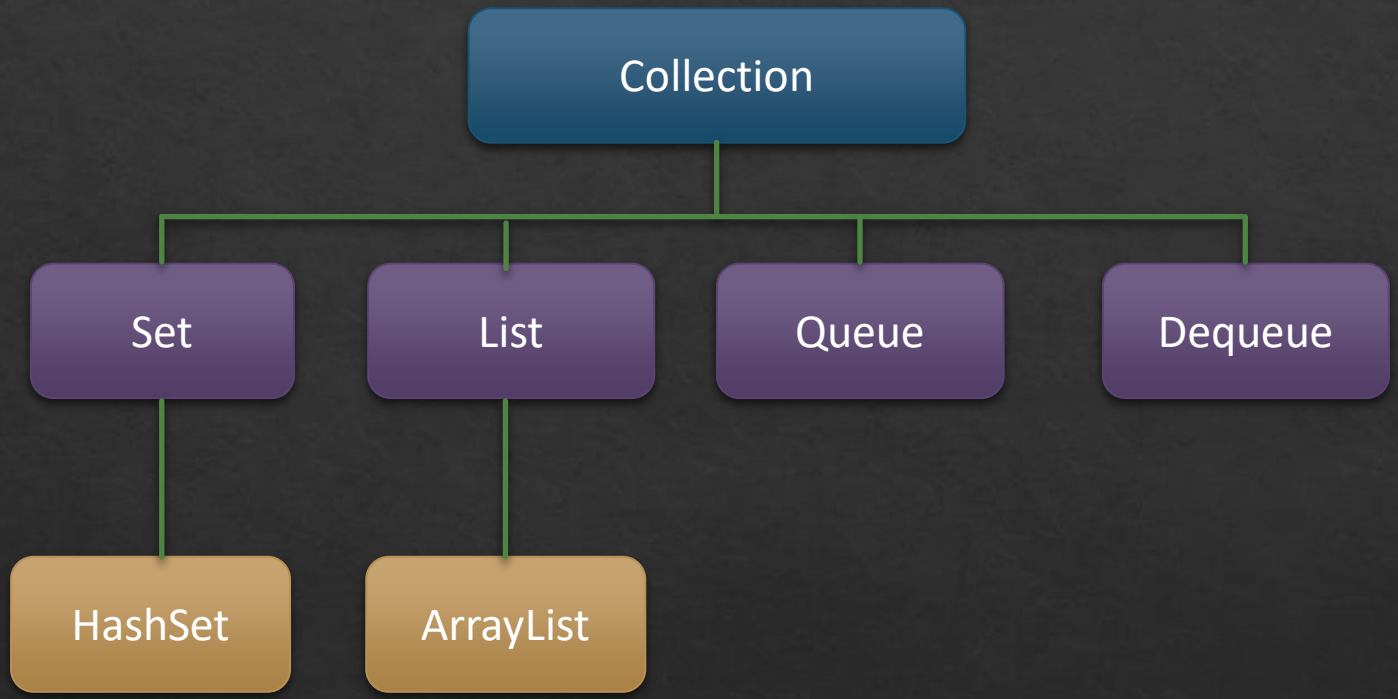


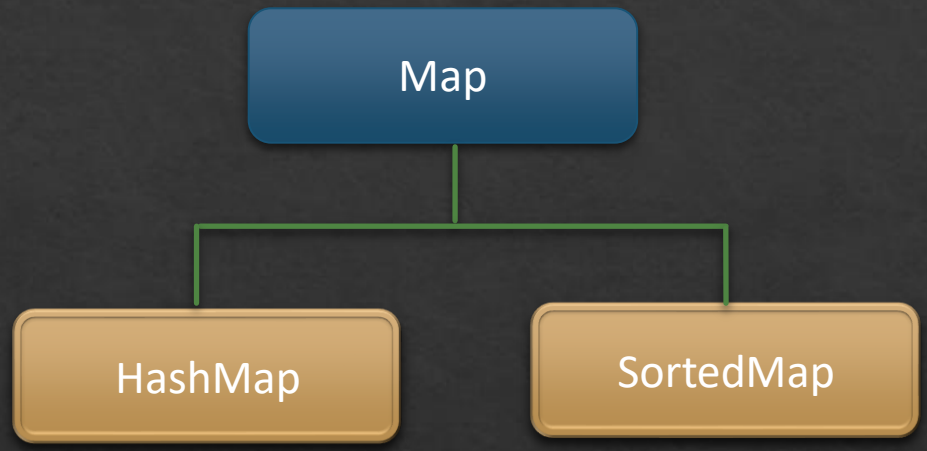
Java Collections Framework

- The Java Collection Frameworks consist of
  - Interfaces : Interfaces allow collections to be manipulated independently of the details of their representation
  - Implementations: These are the concrete reusable implementations of the collection interfaces.
  - Algorithms: These are polymorphic methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.



Java Collections Framework





Collection API Changes

Java 1.0

- Collections: Vector, Hashtable

Java 2.0

- Collection, Set, List, Map etc

Java 5.0

- Generics and Concurrent Collections

Java 8.0

- Stream and Parallel features

Exceptions

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
- The Java programming language uses exceptions to handle errors and other exceptional events.
- Advantages
  - Separating Error-Handling Code from "Regular" Code
  - Propagating Errors Up the Call Stack
  - Grouping and Differentiating Error Types

try-catch-finally

try

- The code that might generate an exception is placed in the try block.
- Every block should be immediately followed by the catch or finally block.

catch

- When an exception occurs, it is caught by the catch block
- For each try block, there can be zero or more catch blocks.
- The argument type of each catch block indicates the type of exception that can be handled by it.

finally

- The block is optional. However, if defined, it is always executed (even if the exception doesn't occur).
- For each try block, there can be only one finally block.

```
try {
  // code
} catch (ExceptionType e) {
  // catch block
} finally {
  // finally block
}
```

```
try {
  //code
} catch (ExceptionType1 e1) {
  // catch block
} catch (ExceptionType1 e2) {
 // catch block
} finally {
  // finally block always executes
}
```

Exceptions

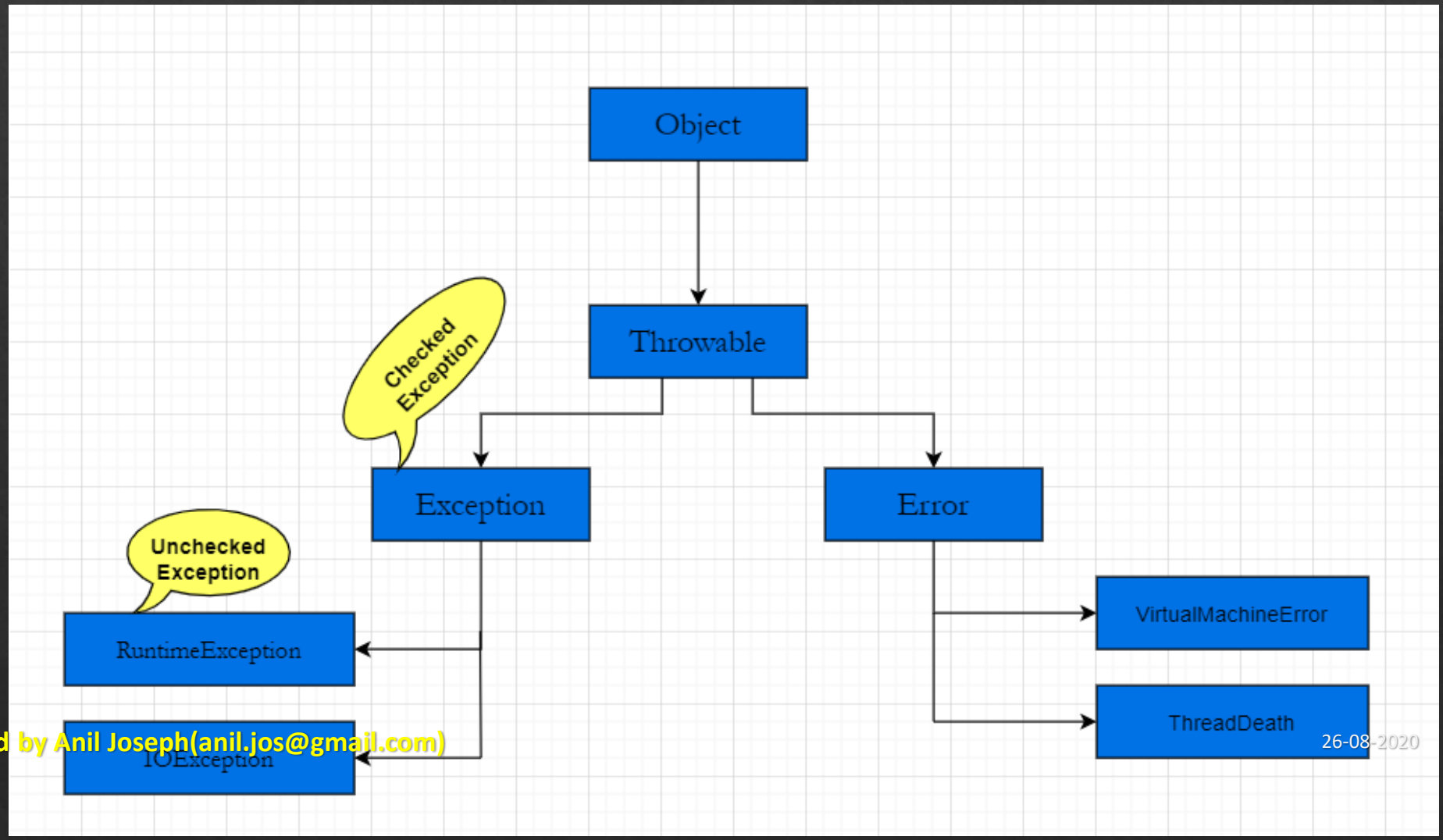
Checked

- The compiler check for the exception and a mandates a try-catch block

Unchecked

- The compiler does not check for the exception. try-catch is optional.

Exceptions


Object
Throwable
Checked Exception
Exception
Error
Unchecked Exception
RuntimeException
IOException
VirtualMachineError
ThreadDeath

try with resources

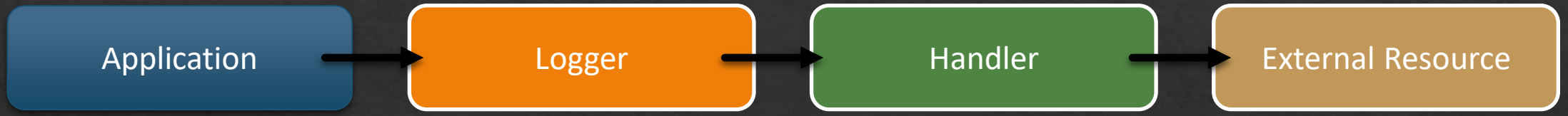
- A resource is an object that must be closed after the program is finished with it.
- The try-with-resources statement is a try statement that declares one or more resources.
- Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

```
try (Scanner scanner = new Scanner(new File("test.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException fnfe) {
    fnfe.printStackTrace();
}
```

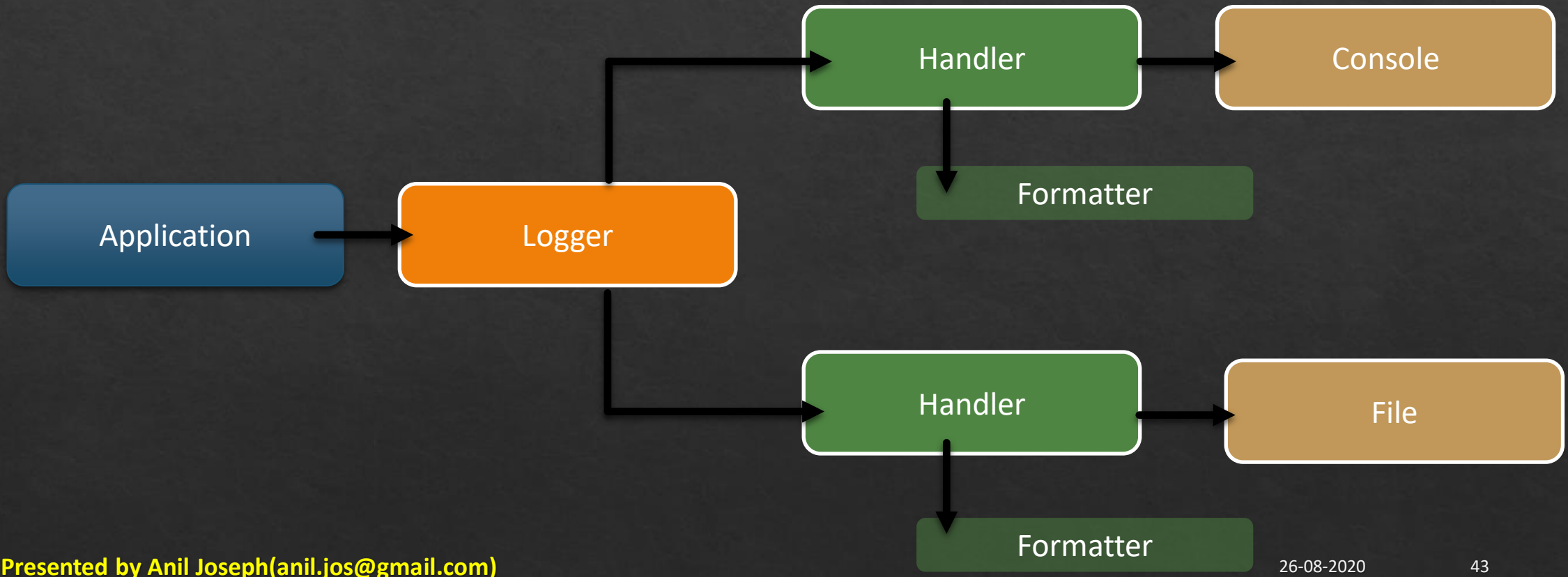
Logging

- Logging is the process of writing log messages during the execution of a program to a central place.
- It allows to report and persist messages(error, warning, info) to be retrieved and analyzed later.
- Java provided the Java Logging API

Logging



Logging


Application
Logger
Handler
Console
Formatter
Handler
File
Formatter

Logger Levels


SEVERE (highest)


WARNING


INFO


CONFIG


FINE

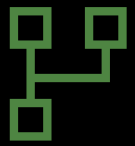

FINER


FINEST

Java IO Streams

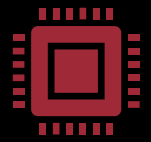
A stream is a conceptually endless flow of data.

You can either read from a stream or write to a stream.



A stream is connected to a data source or a data destination.

Byte Streams



Byte Streams perform input and output operations on 8-bit bytes.

All byte stream classes are inherit from InputStream and OutputStream.

Byte streams should only be used for the most primitive I/O.

Character Streams

Character stream I/O automatically translates this Unicode format to and from the local character set.

All character stream classes are descended from Reader and Writer.

Streams

BufferedStreams

- Optimize input and output by reducing the number of calls to the native API.

Data Streams

- Handle binary I/O of primitive data type and String values.

Object Streams

- Handle binary I/O of objects.

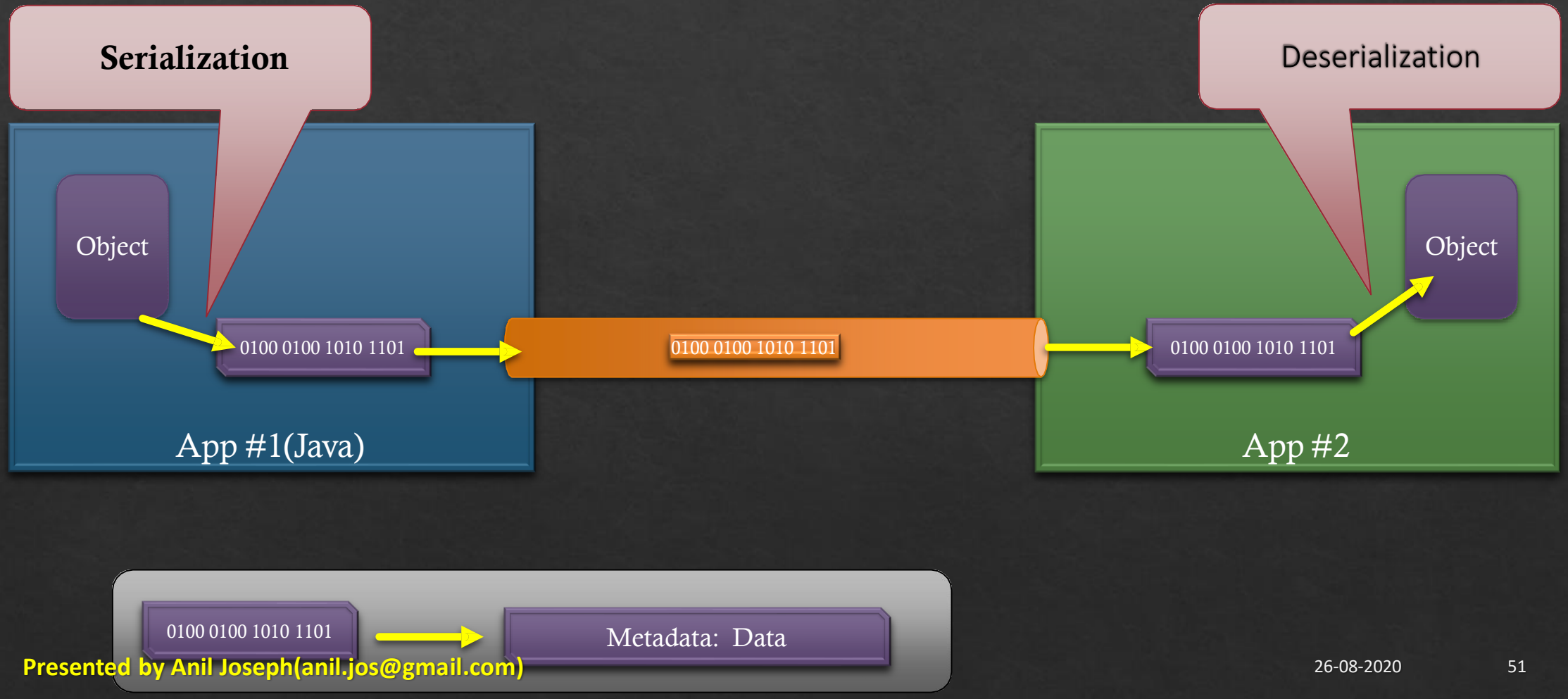
Java NIO

- The package Java New IO introduces more classes for IO Operations
  - Path
  - Files, Directory, and FileSystem
  - Random Access Files
  - Channels
  - Watch Directory for changes

Serialization

- Object Serialization supports the encoding of objects and the objects reachable from them, into a stream of bytes.
- Serialization also supports the complementary reconstruction of the object graph from a stream called deserialization.
- Use the transient keyword to mark a member as non-serializable
- Implement the Externalizable interface to customize serialization

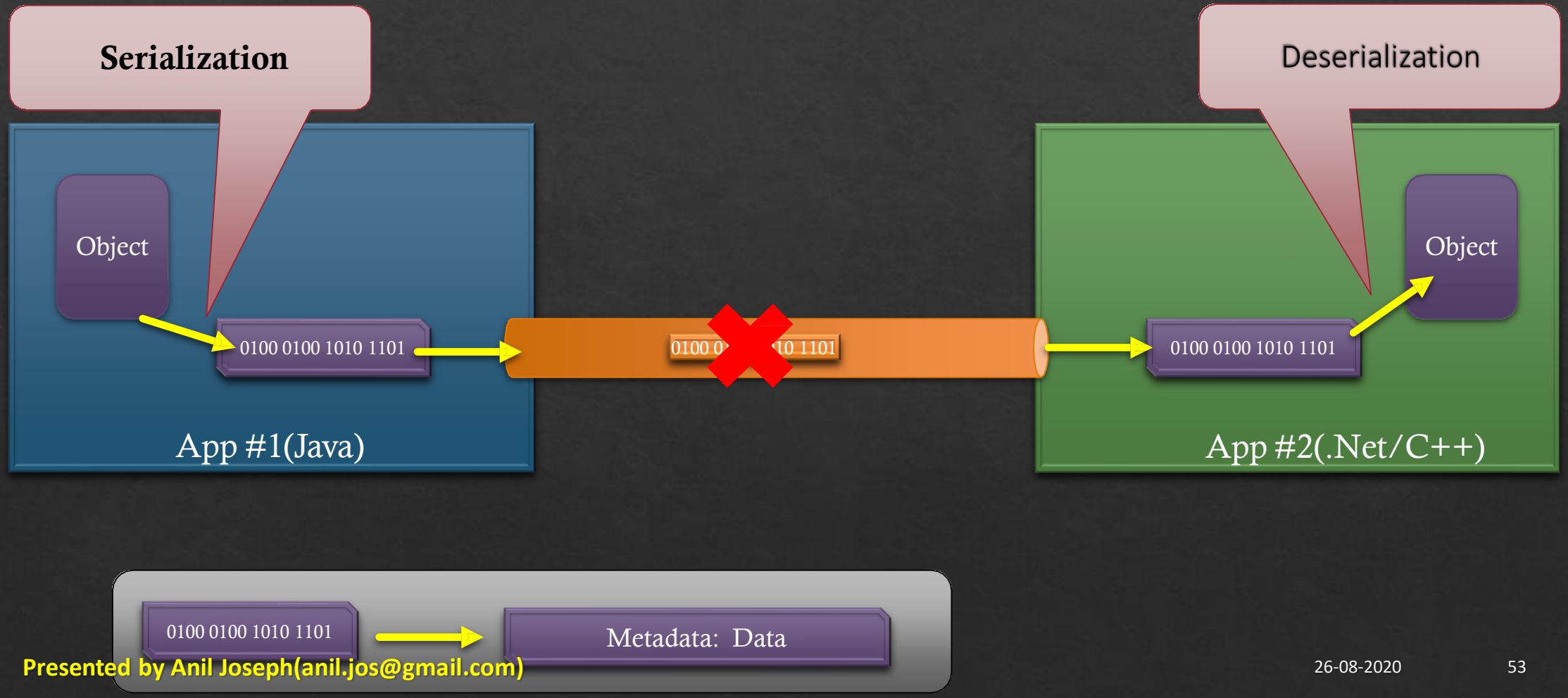
Serialization/Deserialization



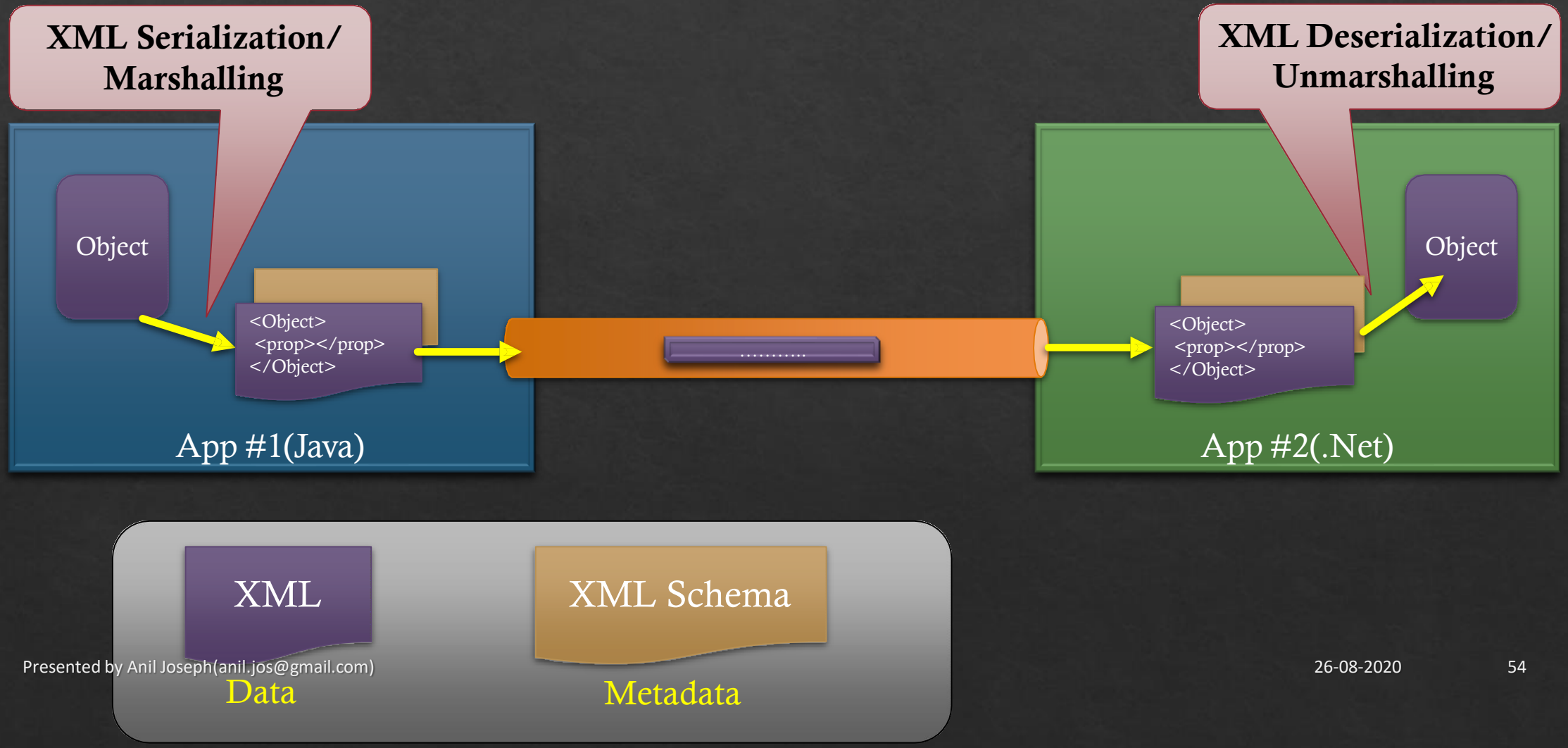
Custom Serialization

- Default Behavior
  - All members of class are serialized to stream
  - Access specifiers don't affect the serialization behavior
- Controlling data members
  - ***transient*** keyword
  - Only non transient members are serialized

Serialization/Deserialization



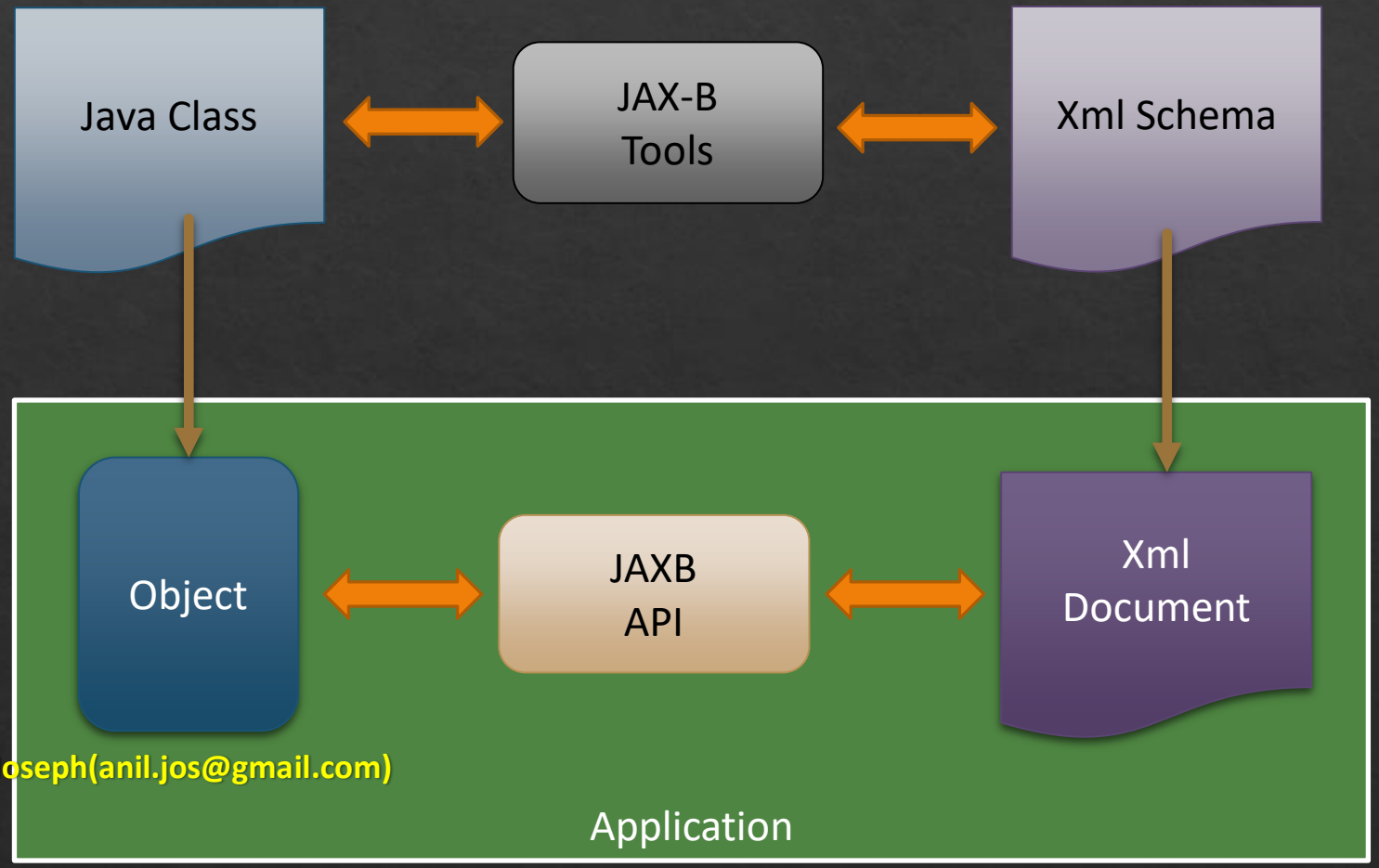
XML Serialization/Deserialization



JAXB

- Java Architecture for XML Binding
- JAXB a fast and convenient way to bind XML schemas and Java representations.
- JAXB provides
  - Unmarshalling : Converts XML documents into Java content trees
  - Marshalling: Converts Java objects back into XML instance documents.
- JAXB also provides a way to generate XML schema from Java objects.

JAXB


Java Class
JAX-B
Tools
Xml Schema
Object
JAXB
API
Xml
Document
Application

Reflection

- Reflection allows a Java program to introspect upon itself.
- Reflection can be used to inspect/or modify runtime attributes of a class, interface, field or methods
- Reflection can also be used to create objects dynamically
- Reflection can also be used to invoke a method dynamically

Annotations

- Annotations provide data about program that is not part of the program itself.
- Annotations have no direct effect of the code they annotate
- Advantages
  - Annotations can be used by the compiler to detect errors or suppress warmings
  - Software tools can process annotations to generate code, XML files, etc.
  - Some annotations are available to be examined at runtime
- Annotations can be applied to declarartions
  - classes, fields, methods, and other program elements.

Predefines Annotations

@Deprecated

- Annotation indicates that the marked element is deprecated and should no longer be used.

@Override

- The annotation informs the compiler that the element is meant to override an element declared in a superclass.

@SuppressWarnings

- The annotation tells the compiler to suppress specific warnings that it would otherwise generate.

@FunctionalInterface

- The annotation, introduced in Java SE 8,

Annotations on annotations

- @Retention annotation specifies how the marked annotation is stored
  - RetentionPolicy.SOURCE – The marked annotation is retained only in the source level and is ignored by the compiler.
  - RetentionPolicy.CLASS – The marked annotation is retained by the compiler at compile time, but is ignored by the Java Virtual Machine (JVM).
  - RetentionPolicy.RUNTIME – The marked annotation is retained by the JVM so it can be used by the runtime environment.
- @Documented
  - The annotation indicates that whenever the specified annotation is used those elements should be documented using the Javadoc tool.

Annotations on annotations

- @Target
  - The annotation marks another annotation to restrict what kind of Java elements the annotation can be applied to.
- @Inherited
  - The annotation indicates that the annotation type can be inherited from the super class.
- @Repeatable
  - The annotation, introduced in Java SE 8, indicates that the marked annotation can be applied more than once to the same declaration or type use.

Java 8

- More concise code.
- Extensibility.
- Simplified coding on multi core systems.

Interfaces

- Interfaces in Java 8 introduces 2 new types of methods apart from abstract methods
- Two new types
  - Default
  - Static

Interfaces: Default methods

- Default methods are methods in an interface with an implementation(can be overridden in the implementing class).
- Provides a mechanism to add new methods to existing interfaces
- It doesn’t break backwards compatibility
- Gives Java multiple inheritance of behavior, as well as types
  - but not state!

Interfaces: Default methods

```
public interface InterfaceA {

    public void doSomething();

    default void buildTask(){
        System.out.println("InterfaceA buildTask");
    }

}
```

Interfaces: Default methods Conflict Resolution

- The Java compiler follows inheritance rules to resolve the name conflict.
- Instance methods are preferred over interface default methods.
- Methods that are already overridden by other candidates are ignored.
  - This circumstance can arise when supertypes share a common ancestor.
- If two or more independently defined default methods conflict, or a default method conflicts with an abstract method, then the Java compiler produces a compiler error.
  - You must explicitly override the supertype methods.

Interfaces: static methods

- Java 8 static methods in an interface
- This is linked to default(extension) methods in that interfaces can now include behavior.
- Static methods, by definition, are not abstract

Interfaces: static methods

```
public interface InterfaceB {

    void doWork();

    default void defaultTask(){

        System.out.println("InterfaceB defaultTask...");

    }

    static void buildTask(){
        System.out.println("InterfaceB static buildTask");
    }
}
```

Functional Interfaces

- Many interfaces in the Java library declare only one method.
- Examples are java.lang.Runnable, java.awt.event.ActionListener, java.util.Comparator etc
- These interfaces are also called Single Abstract Method interfaces (SAM Interfaces).
- ***In Java 8 the same concept of SAM interfaces is recreated and are called Functional interfaces.***
- ***Functional interfaces*** can be represented using Lambda expressions, Method reference and constructor references.
- ***@FunctionalInterface*** *annotation* can be used for compiler level errors.

Functional Interfaces

```
@FunctionalInterface
public interface Simple {

    void doSomething();
}
```

Lambda Expressions

- Lambda expressions represent anonymous functions
  - Like a method, has a typed argument list, an inferred return type, a set of thrown exceptions, and a body
  - Not associated with a class
- Lambda expressions enable you to treat functionality as method argument, or code as data.
- Syntax
  - (parameters) -> {body}

Lambda Expressions

- **The Functional Interface**

```
@FunctionalInterface
public interface Simple {

    void doSomething();
}
```

- The Lambda Expression.

```
Simple simple = () -> System.out.println("Implemented using lambda");
simple.doSomething();
```

Lambda Expressions

- The Functional Interface

```
@FunctionalInterface
public interface Calculator {

    int calculate(int x, int y);

}
```

- The Lambda Expression.

```
Calculator calculator = (x, y) -> x + y;
System.out.println(calculator.calculate(10, 20));
calculator = (x, y) -> x - y;
System.out.println(calculator.calculate(10, 20));
calculator = (x, y) -> {

        System.out.println("Calculating");
        return x * y;

    };
System.out.println(calculator.calculate(10, 20));
```

Lambda Expressions

- Almost all machines now are multi-core, multi-processor, or both
- We need to make it simpler to write multi-threaded Java code
  - Java has always had the concept of threads
  - Even using the concurrency utilities and fork-join framework this is hard
- Lambda expressions and the streams API simplify the threading / parallelism code.

Method References

- Method references let us reuse a method as a lambda expression
- Four Types of references
- Reference to a static method:
  - **ContainingClass::staticMethodName**
- Reference to an instance method of a particular object:
  - **ContainingObject::instanceMethodName**
- Reference to an instance method of an arbitrary object of a particular type:
  - **ContainingType::methodName**
- Reference to a constructor:
  - **ClassName::new**

Optional

- Java 8 introduces a new class called java.util.Optional<T>.
- Used to indicate that reference may, or may not have a value.

Stream API's

- A ***stream*** represents a sequence of elements supporting sequential and parallel aggregate operations.
- A stream is like a pipeline with three parts
  - A source
  - Zero or more intermediate operations
  - A terminal operation
    - Producing a result or a side-effect

```
int sum = transactions.stream().
  filter(t -> t.getBuyer().getCity().equals("London")).
  mapToInt(Transaction::getPrice).
  sum();
```

Stream API's Sources

- There are many ways to create a Stream.
- From collections and arrays
  - `Collection.stream()`
  - `Collection.parallelStream()`
  - `Arrays.stream(T array)` or `Stream.of()`
- Static factories
  - `IntStream.range()`
  - `Files.walk()`
- Roll your own
  - `java.util.Spliterator()`

Stream API's
Intermediate Operations

- Intermediate Operations are executed lazily.
- The internal processing model of streams is designed in order to optimize the processing flow.
- Intermediate Operations available
  - Mapping
    - Mapping is a process of changing the form of the elements in a *stream*.
  - Filtering
    - Filtering is a process of selecting items depending upon some condition.
  - Unique Elements
    - A process to ensure there are a no duplicate elements. It's a type of filter
  - Skipping
    - A process to skip a number of elements. It's a type of filter
  - Sorting

Stream API's
Terminal methods

- Invoking a terminal operation executes the pipeline
- All operations can execute sequentially or in parallel
- Terminal operations can take advantage of pipeline characteristics

Collectors

- The collect method the Stream of is a terminal operation, that allows to transform a stream into another type(possibly a list or set).
- The argument passed to the collect method is an instance of java.util.stream.Collector.
- The Collector essentially describes a recipe for accumulating the elements of a stream into a final result.
- Collector operations
  - Grouping
  - Partitioning

Parallel Processing

- Java 8 Streams supports parallel processing.
- A collection can be turned into a parallel stream by invoking the method parallelStream on the collection source.
- A *parallel* stream is a stream that splits its elements into multiple chunks, processing each chunk with a different thread.
- It can automatically partition the workload of a given operation on all the cores of your multicore processor and keep all of them equally busy.
- Parallel streams use the threads for the ThreadPool
  - The default size is the equivalent to the number of processors.
  - System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", "12");

New Date and Time API

- Java 8 introduces a new set of classes to work with Date and Time
- The new classes are in the package java.time
- New Classes
  - LocalDate
  - LocalTime
  - LocalDateTime
  - Instant
  - Duration
  - ZonedDateTime

Modules

- Modules adds a higher level of aggregation above packages.
- *Module*—a uniquely named, reusable group of related packages, as well as resources (such as images and XML files) and a *module descriptor.*
- Module Descriptor
  - Module's *name*
  - Module's *dependencies* (that is, other modules this module depends on)
  - Packages it explicitly makes available to other modules
  - *Services it offers*
  - *Services it consumes*
  - Other modules it allows *reflection*

Module Benefits

- Reliable configuration
  - Modularity provides mechanisms for explicitly declaring dependencies between modules in a manner that's recognized both at compile time and execution time.
- Strong encapsulation
  - The packages in a module are accessible to other modules only if the module explicitly exports them.
- Scalable Java platform
  - Previously, the Java platform was a monolith consisting of a massive number of packages, making it challenging to develop, maintain and evolve.

Module Benefits

- Greater platform integrity
  - Before Java 9, it was possible to use many classes in the platform that were not meant for use by an app's classes.
- Improved performance
  - The JVM uses various optimization techniques to improve application performance.

Modules

- The JDK is modularized
- List Java Modules
  - java --list-modules

Concurrency

- Concurrency is the ability of a program to execute several computations simultaneously(in parallel).
- A modern computer has several CPU's or several cores within one CPU.
  - The ability to leverage these multi-cores can be the key for a successful high-volume application.
- Process
  - Processes are an execution environment provided by the operating system that has its own set of private resources
  - In Java, processes correspond to a running Java Virtual Machine (JVM).
- Thread
  - Threads are lightweight processes that live within a process and share their resources with the other threads of the process.
  - Threads live within the same JVM and can be created and stopped by the Java application dynamically during runtime

Thread

- Threads are light-weight processes within a process.
- They share the memory(Heap) with other threads.
- Each thread has its own call stack
- In Java the java.lang.Thread class is used to create a thread.
- Every thread has an
  - Identifier
  - Name
  - Priority
  - State

Creating Thread

Extend from the Thread class and override the run method

Implement the Runnable interface and override the run method

Thread States

New

- A thread that has not yet started is in this state.

Runnable

- A thread executing in the Java virtual machine is in this state.

Blocked

- A thread that is blocked waiting for a monitor lock is in this state.

Waiting

- A thread that is waiting indefinitely for another thread to perform a particular action is in this state.

Timed Waiting

- A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.

Terminated

- A thread that has exited is in this state.

Sleep, Join and Interrupt

sleep() puts the current Thread to sleep without consuming any processing time.

The current thread removes itself from the list of active threads and the scheduler doesn't schedule it for the next execution

Interrupts are a mechanism to send a message to a thread from another

join() lets the thread to wait for the termination of another thread.

Thread Local

- The *TheadLocal* construct allows us to store data that will be **accessible only** by **a specific thread**.
- This used to store context information like session or transaction identifies with the Thread.

Concurrency Issues

- Threads have their own call stack but can also access shared data.
- Therefore you have two basic problems
  - Visibility
  - Access problems.
- Visibility: Occurs if thread A reads shared data which is later changed by thread B and thread A is unaware of this change.
- Access: Occur if several thread access and change the same shared data at the same time.

Locks and Synchronization

- Java provides *locks* to protect certain parts of the code to be executed by several threads at the same time.
- To create a lock on an object use the ***synchronized*** keyword.
- Synchronization is necessary for mutually exclusive access to blocks of and for reliable communication between threads.

Object Monitors: wait and notify

The wait() method pauses the execution of the current thread.

The notify() methods wakes up a thread in the paused/wait mode.

The thread that calls the wait() method must hold a lock to a resource

Calling wait() the lock is released and the threads waits until notify is called.

Another thread that now owns the lock calls notify() on the same object instance

Java.util.concurrent package

- The java.util.concurrent packages provide a set of ready to use classes for concurrency.
- Thread Pool
- Executor
- ExecutorService
- ExecutorCompletionService
- Callable
- Future
- CompleteableFuture
- Semaphore
- CountDownLatch
- CyclicBarrier

Fork And Join

Fork (split) a large task into smaller subtasks which can be executed concurrently.

Each subtasks can be executed in parallel by different CPUs, or different threads on the same CPU.

Once the subtasks are completed the task may *join* (merge) all the results into one result.

The ForkJoinPool is a special thread pool which is designed to work well with fork-and-join task splitting

Jar(Java archive)

- A JAR is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file for distribution.
- JAR files are archive files that include a Java-specific manifest file.
- They are built on the ZIP format and typically have a .jar extension
- Executable JAR files
  - An executable Java program can be packaged in a JAR file, along with any libraries the program uses.
  - Executable JAR files have the manifest specifying the entry point class with Main-Class: com.MainClass

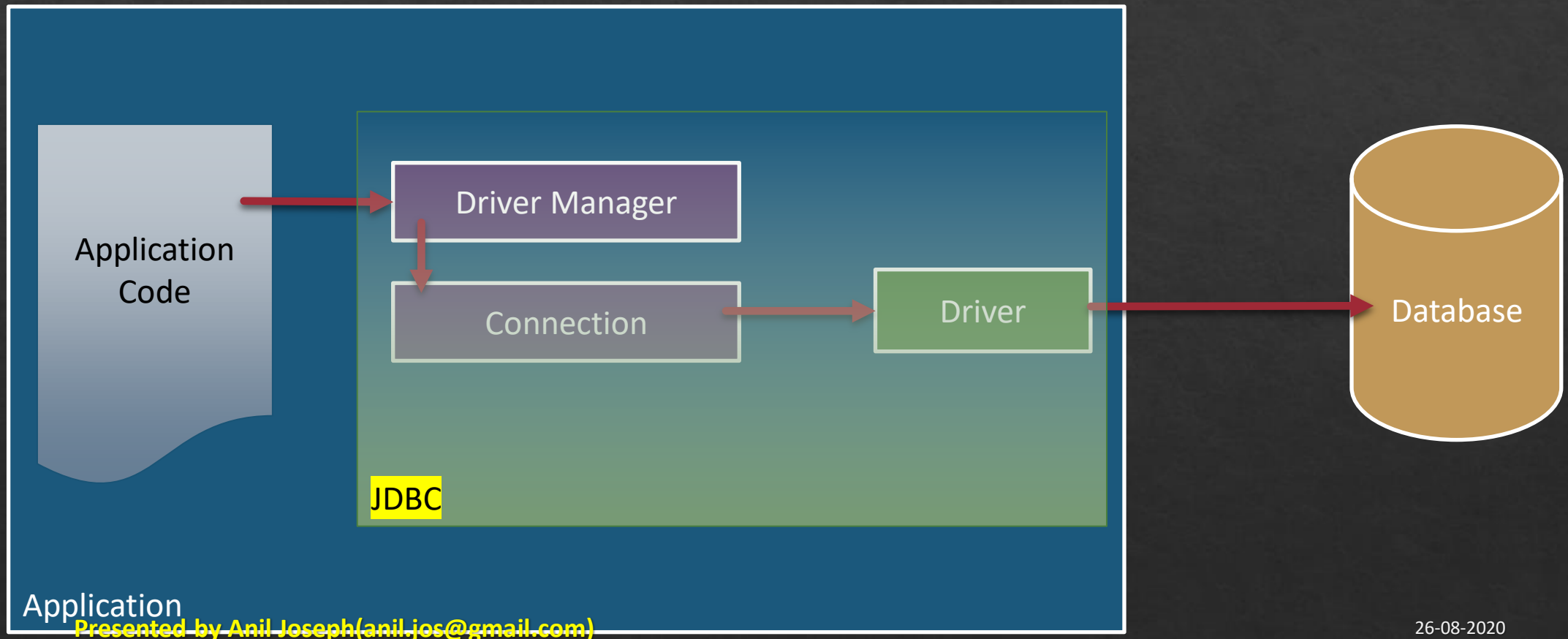
Java Database Connectivity(JDBC)

- JDBC is a Java standard that provides the interface for connecting from Java to relational databases.
- JDBC allows to
  - Connect to a wide range of Relational Database Management Systems (RDBMS) like Oracle, Microsoft SQL, MySQL and many more .
  - Manipulate the structure of a database by creating, deleting, or altering relations from that database.
  - Execute CRUD operations database table.
  - Querying a database
- The JDBC API consists of two Java packages: java.sql and javax.sql.
  - java.sql implements the JDBC Core API.
  - javax.sql implements the optional API's

JDBC Drivers

- A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database.
- JDBC Driver Types
  - Type 1: JDBC-ODBC Bridge Driver
  - Type 2: Native Driver(Partially Java)
  - Type 3: Middleware Driver
  - Type 4: Pure Java Driver
- DriverManager
  - DriverManager manages the set of JDBC drivers that are available for an application to use.

JDBC API's


Application Code
Driver Manager
Connection
Driver
Database
JDBC
Application

JDBC Connection

Represents a connection to the Database

The interface provides the methods for fetching the Database information using the method getMetaData().

Provides the methods for creating the Statements

createStatement, prepareStatement, prepareCall

Exposes the API's to manage transactions

setAutoComit, commit, rollback

Statement

A Statement is an interface that represents a SQL statement.

Three Types

Statement
PreparedStatement
CallableStatement

Statement

Used to implement simple SQL statements with no parameters

PreparedStatement

Used for precompiling SQL statements that might contain input parameters.

CallableStatement

Used to execute stored procedures that may contain both input and output parameters.

ResultSet

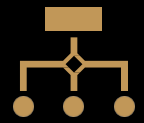
ResultSet is a table of data representing a database result set

Generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data.

Initially the cursor is positioned before the first row.

The next method moves the cursor to the next row



Three types

Forward only

Scroll insensitive

Scroll Sensitive

Transactions

A **transaction** is a logical unit of work that contains one or more SQL statements.

A transaction is an atomic unit.

The effects of all the SQL statements in a transaction can be either

All **committed** (applied to the database)

or all **rolled back** (undone from the database).

Globalization

- Globalization denotes that the application has been internationalized and localized.
- Internationalization(i18n)
  - The process of removing hard-coded region-specific information like texts, dates, time, currency, images
  - Using API to retrieves the regionalized data.
- Localization(l10n)
  - The process creating content for a specific locale or region

Internationalization

- Identity content that should change depending on the user's region
- This content is placed into an external resource called the resource bundle.
- Use API's to delegate the lookup and rendering of the content until runtime.

i18n: Locale

- A Locale is a Java Object that represents various types of information for a user or machine.
- Properties
  - A language
  - A country
  - A variant

Resource Bundles

- A resource bundle is a Java class or .properties file that contains locale-specific data.
- It is a way of internationalizing a Java application by making the code locale-independent.
- Resources bundle as a class
  - Inherit from java.util.ResourceBundle
  - Class name suffixed with the locale
- Resource bundle as a file
  - A Java properties file suffixed with e locale

Java Web Application

Java SE

- Java SE's API provides the core functionality of the Java programming language.
- It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.
- In addition to the core API, the Java SE platform consists of
  - A virtual machine
  - Development tools
  - Deployment technologies,
  - Class libraries
  - Toolkits

Java EE

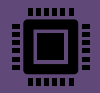
- The Java EE platform is built on top of the Java SE platform.
- The Java EE platform provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications.
- Java EE specification defines the
  - Web Server
  - Web API (Servlet, JSP, JSF etc)
  - Web Services(SOAP & REST)
  - Application Server
  - Enterprise Java Beans(EJB)
  - Enterprise API's(Messaging, Transactions, Security etc.)

Servlets

A servlet is a small Java program that runs within a Web server.

Servlets receive and respond to requests from Web clients, usually across HTTP.

The contract to write a servlet is defined in the javax.servlet.Servlet interface

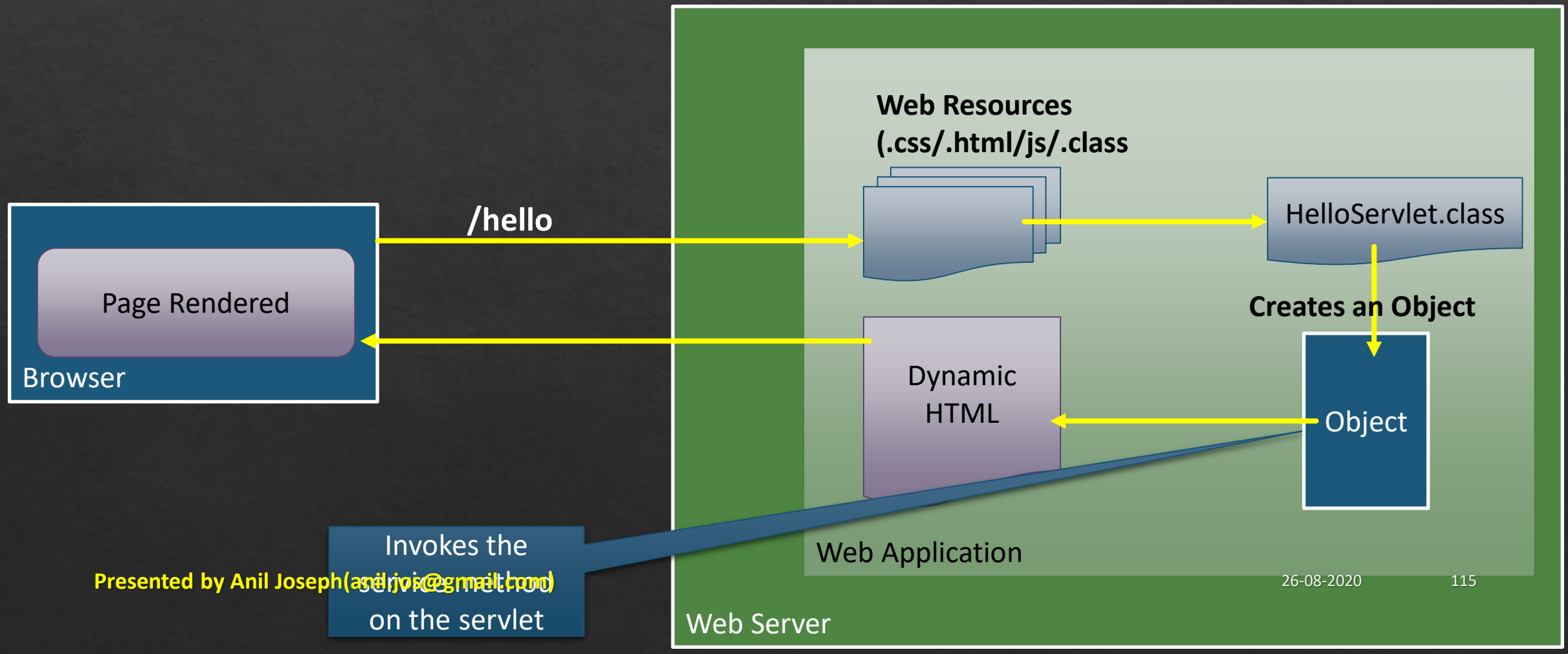


To implement a Servlet, extend from any one of these clases

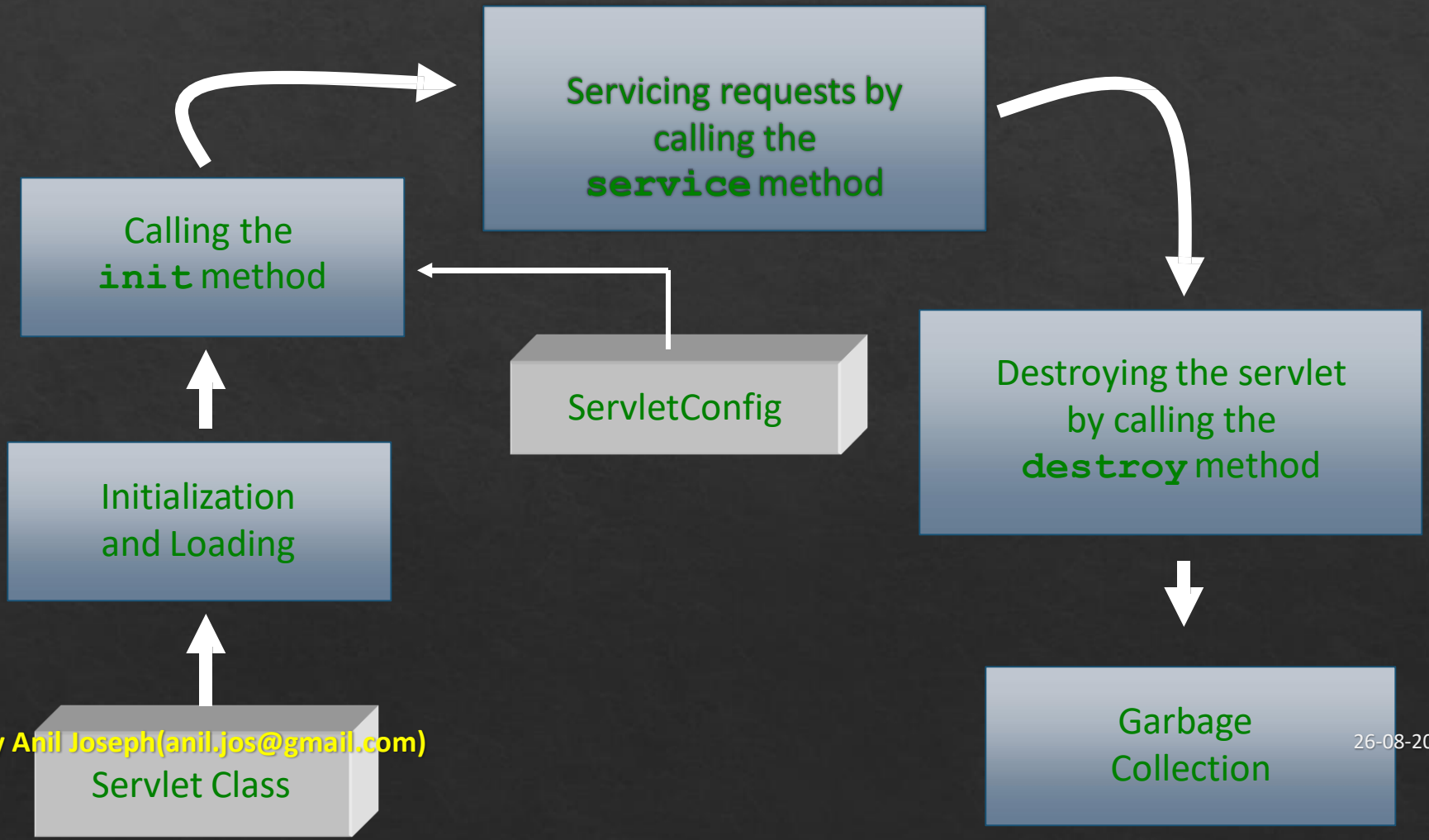
javax.servlet.GenericServlet

javax.servlet.http.HttpServlet.

Web Server: Servlets



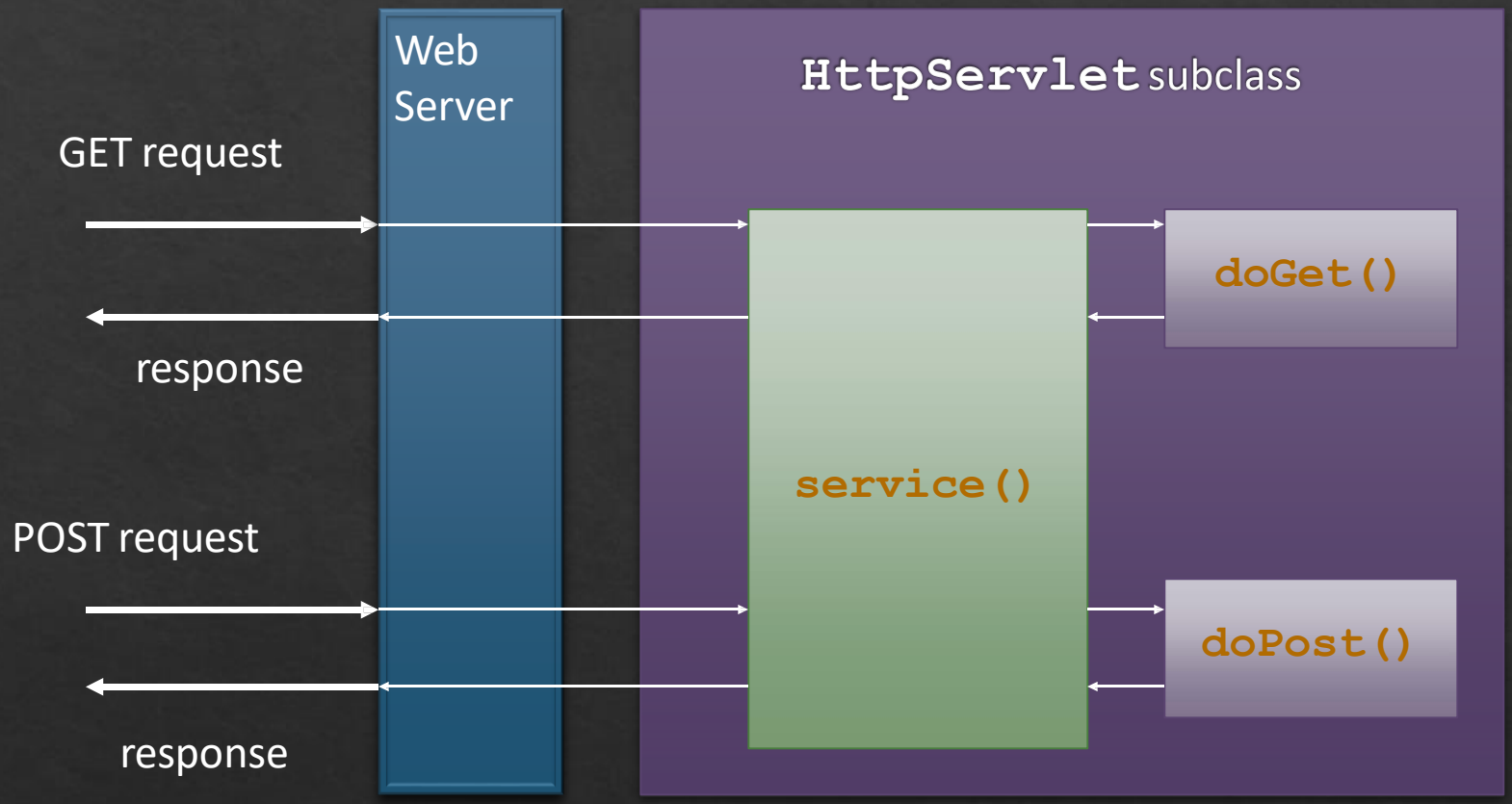
Servlet Life Cycle



HttpServlet

- Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site
- Methods
  - *doGet*
    - if the servlet supports HTTP GET requests
  - *doPost*
    - if the servlet supports HTTP POST requests
  - *doPut*
    - if the servlet supports HTTP PUT requests
  - *doDelete*
    - if the servlet supports HTTP DELETE requests
- The implementing class should not override the ***service*** method.

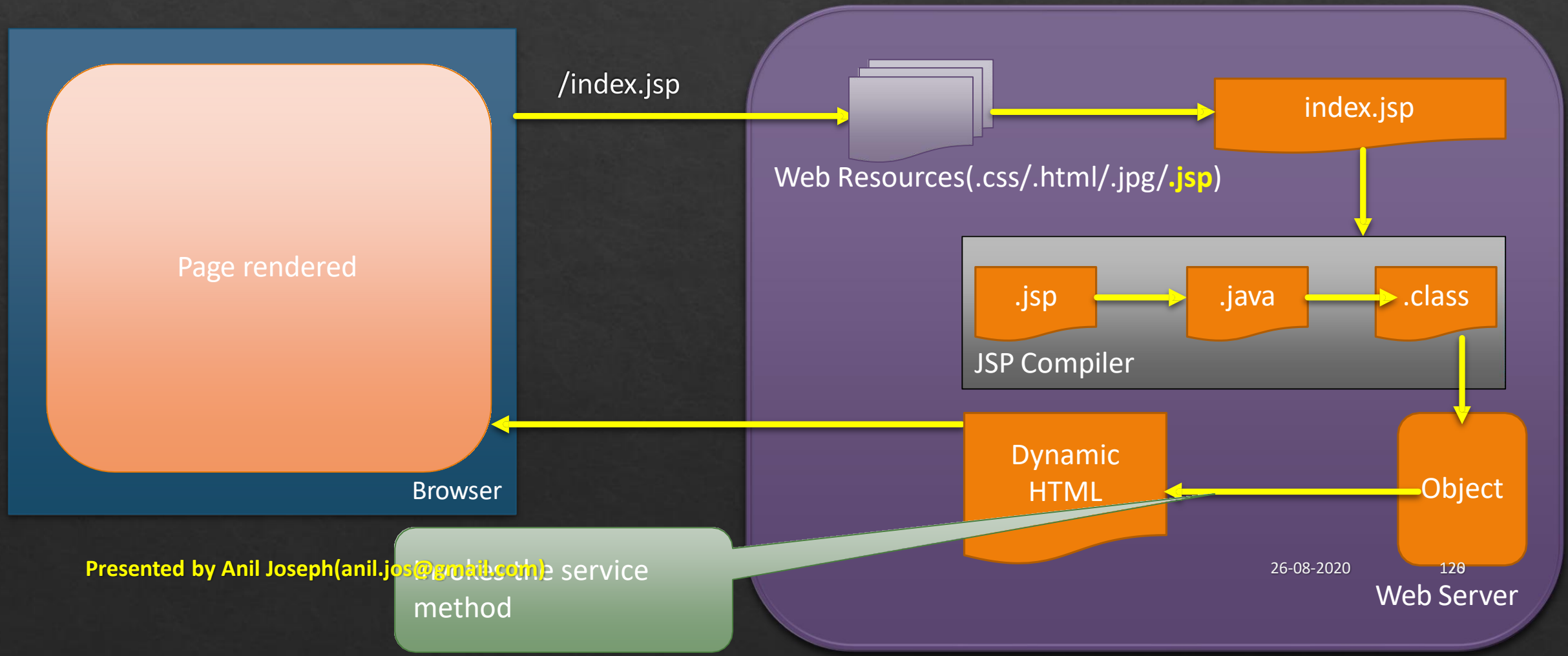
`HttpServlet` Request Handling



JSP(Java Server Pages)

- JSP facilitates the creation of dynamic content on the server
- JSP integrates numerous Java application technologies,
  - Java servlet
  - JavaBeans
  - JDBC
- It separates presentation from application logic and fosters a reusable-component model of programming.
- JSP as an extension of HTML that gives you the ability to seamlessly embed snippets of Java code within your HTML pages.
- JSP as a higher-level way to write servlets

Web Server: JSP



Spring 5.x

Spring

- Spring is a lightweight enterprise framework
- Open source and address the complexities of enterprise application development
- Simple, promotes TDD and loose coupling.
- Used to create standalone, web, enterprise, and cloud deployed applications
- Integrates with other frameworks like Hibernate, Struts, iBatis, Hessian etc.

What Spring offers?

Container

- Contains and Manages the lifecycle and configuration of application objects

Inversion of Control(Dependency Injection)

- Promotes loose coupling

Aspect-oriented programming

- Enables cohesive development by separating application business logic from system services

Data Access Framework

- Spring JDBC: Enables writing clean and simple data access code
- Spring ORM:Integrates with ORM's
- Spring Transaction: Transaction Management
- Spring Data: Simplified API for JPA

What Spring offers?

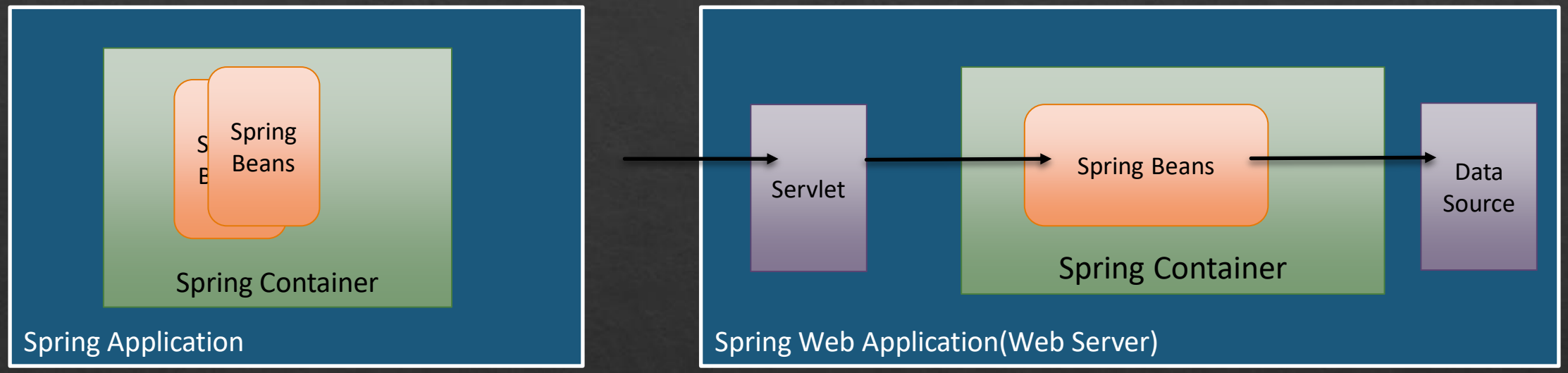
MVC Framework

- MVC framework for web and portal applications
- RESTful Web Services

Other Spring Projects

- Spring Security
- Spring Boot
- Spring WebFlow
- Spring Integration
- Any many more…

Spring Container



Spring Beans

Spring beans are objects managed by the Spring Container.

The follow the Java Bean Specification

Spring Containers

Containers are the core of spring framework

They manage the spring beans.

Application Context interface defines the container contract

Application Context Implementations

ClassPathXmlApplicationContext

- XML configuration in the class path

FileSystemXmlApplicationContext

- XML configuration in the file system

XmlWebApplicationContext

- XML configuration for a web application

AnnotationConfigApplicationContext

- Pure annotation configuration

AnnotationConfigWebApplicationContext

- Pure annotation configuration for web application

Component Scan

- A mechanism of scanning a package and its sub packages for spring beans.
- The beans get registered with the Spring Context .
- The class has to annotated with the @Component annotations or any of types
  - @Service
  - @Repository
  - @Controller

Inversion of Control

Principle: *“Don’t call us, we’ll call you.”*

IoC is a principle that is used to wire an application together

Defines how dependencies or object graphs are created.

In Spring, the IoC “flavor” is referred to as Dependency Injection

Dependency Injection Mechanisms

Property(Setter) Injection

- The dependency is injected through a property.

Constructor Injection

- The dependency is injected through a constructor.

Auto-wiring annotations

@Autowired

- Marks a constructor, field or setter method as to be autowired by Spring's dependency injection facilities.
- Only one constructor (at max) of any given bean class may carry this
- annotation.
- Resolves the dependency byType else byName

@Qualifier

- This annotation may be used on a field or parameter as a qualifier for candidate beans when autowiring.
- Used with @Autowired
- Resolves the dependency byName

Profiles

A profile is a named logical grouping that may be activated programmatically

@Profile

Indicated that a component is eligible for registration if one or more specified profiles are active.

The profile annotation may be used in any of the following ways

As a type-level annotation on any class directly of indirectly annotated with @Component, including @Configuration classes.

As a method level annotation on any @Bean method

Bean Scopes

Singleton	A single instance created per context
Prototype	Instances created whenever a bean is requested from the context
Request	Instance per Http request
Session	Instance per Http session

Lifecycle Methods

3 Mechanisms

- Implement interfaces InitializingBean and DisposableBean
- Provide custom initialization and cleanup methods
- Use the annotations @PostContruct and @PreDestroy

Order of Invocation

- Constructor
- Property Injection
- Initialization methods
- Destroy methods(based on the scope)

Spring JDBC

- A value addition provided by Spring as an abstraction over JDBC
- Spring JDBC takes care of the low-level details that makes JDBC a tedious API to develop with.
- Advantages
  - Opens the connection
  - Prepares and executes the statement
  - Handles the code iteration
  - Processes the exceptions
  - Handles transactions
  - Closes the connection, statement and resultset

Spring JDBC API

JdbcTemplate

- The classic Spring JDBC class

NamedParameterJdbcTemplate

- To provide named parameters instead of the traditional JDBC positional(?) parameters.

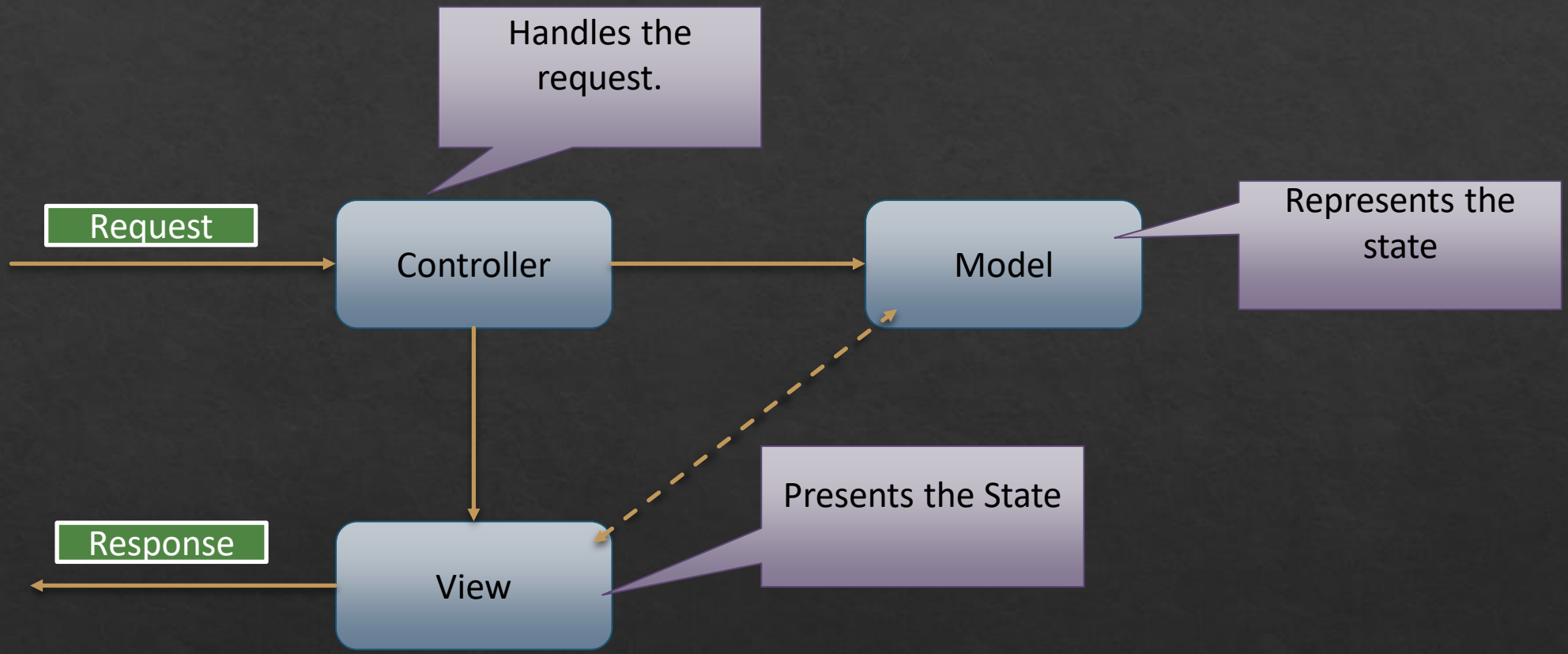
SimpleJdbcInsert and SimpleJdbcCall

- Optimize database metadata to limit the amount of necessary configuration.

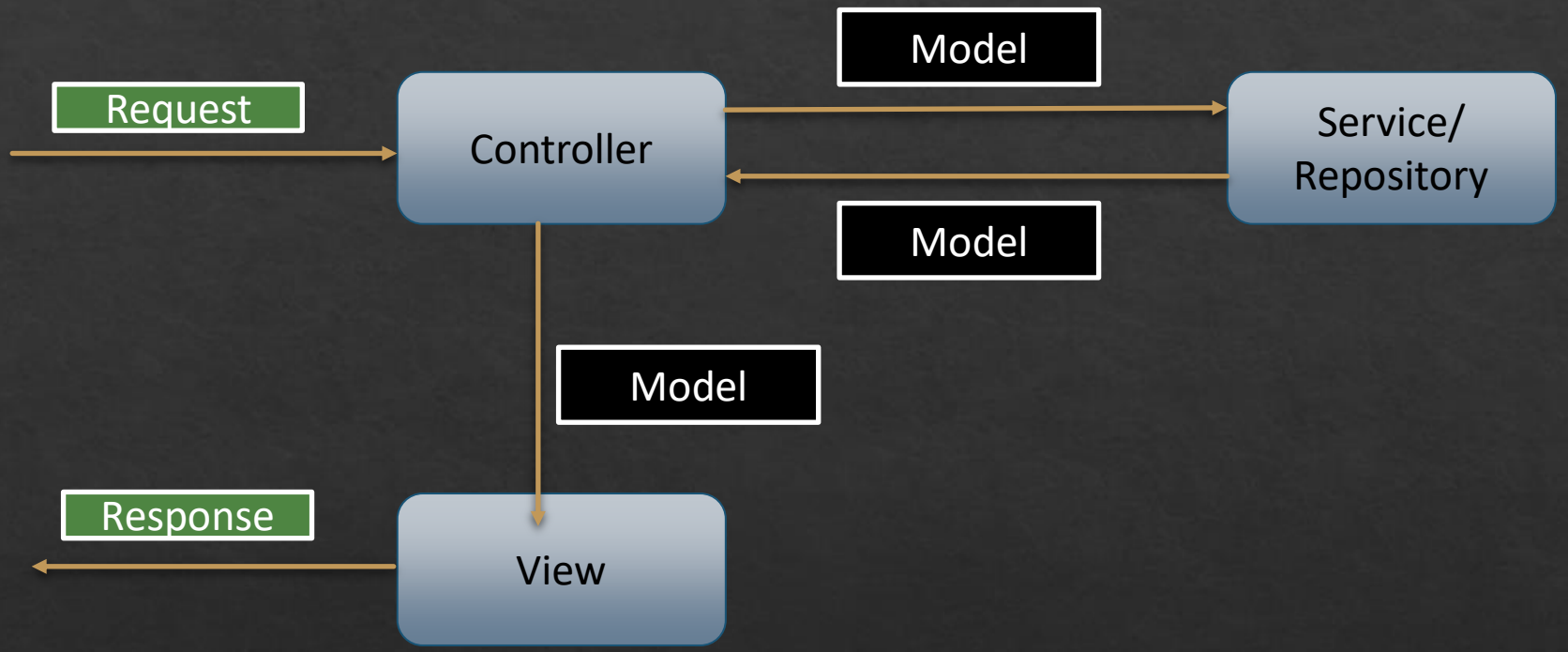
Spring MVC

- Spring MVC provides a framework to build web applications using the MVC design pattern
- Spring MVC is based in the Servlet specification.

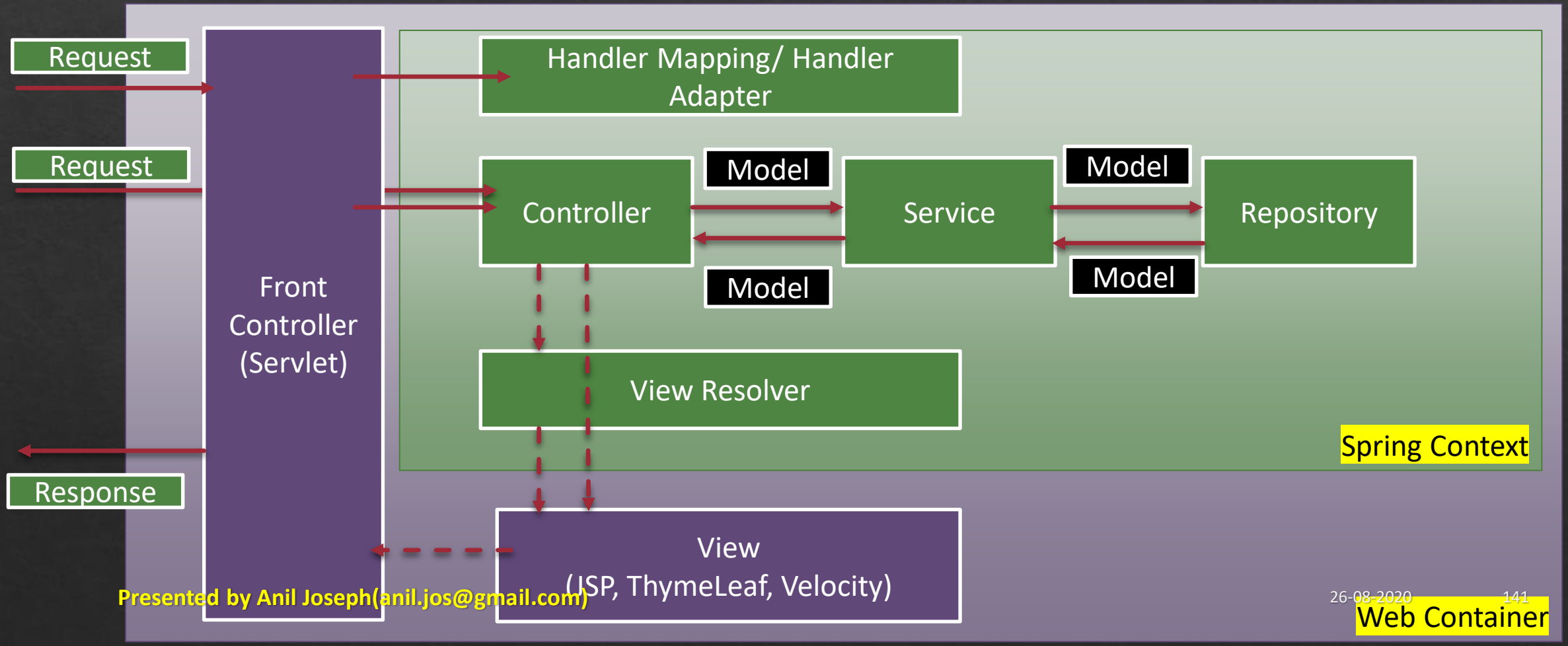
MVC


Handles the request.
Request
Controller
Model
Represents the state
Presents the State
Response
View

MVC


Request
Controller
Model
Service/
Repository
Model
Model
Response
View

Spring MVC Stack



Front Controller

The DispatcherServlet is the front Controller in the MVC stack of Spring.

The primary role of the servlet is to dispatch the requests to the controllers for processing

The servlet loads the spring application context and hence has access to all the spring beans and features

The implementing class is ***org.springframework.web.servlet.DispatcherServlet***

Controller

Controller is a Spring bean annotated with @Controller annotation.

Use the @RequestMapping annotation to map the requests to the controller

Defines the methods to handle request.

Request Handler Methods

Methods in the controller that handle the request can have flexible signatures.

The must be annotated with @RequestMapping to map the request to the handler methods.

Request Handler Method Arguments

- Request and Response Object
- Session Object
- InputStream or Reader
- OutputStream or Writer
- Annotated method parameters

MVC Annotations

@RequestParam	Maps a request(query) parameter to the method argument
@PathVariable	Maps a path on the request to the method argument
@CookieValue	Maps the value of a request cookie to the method argument
@RequestHeader	Maps a request header value to the method argument
@ModelAttribute	Maps request(query) parameters to a model object

Request Handler Return Types

- A string value that is interpreted as the logical view name or as the redirect URL
- An instance of ModelAndView
- An instance of View
- A String values that is the response
  - Use the @ResponseBody annotation

REST API(Services)

What is REST?

- Representational State Transfer (REST) is a style of architecture.
  - Describes how networked resources are defined and addressed.
- These principles were first described in 2000 by Roy Fielding
- REST has proved to be a popular choice for implementing Web Services.

Principles of REST

Client-Server

Cacheable

Stateless

Uniform Interface

Layered

Code on demand

Principles of REST

Client–server

- The client and the server both have a different set of concerns.
- The server stores and/or manipulates information and makes it available to the user in an efficient manner.
- The client takes that information and displays it to the user and/or uses it to perform subsequent requests for information.

Stateless

- A communication between the client and the server always contains all the information needed to perform the request.
- There is no session state in the server, it is kept entirely on the client's side.

Cacheable

- The client, the server and any intermediary components can all cache resources in order to improve performance.

Principles of REST

Uniform Interface

- Provides a **uniform interface** between components.
- Requests from different clients look the same

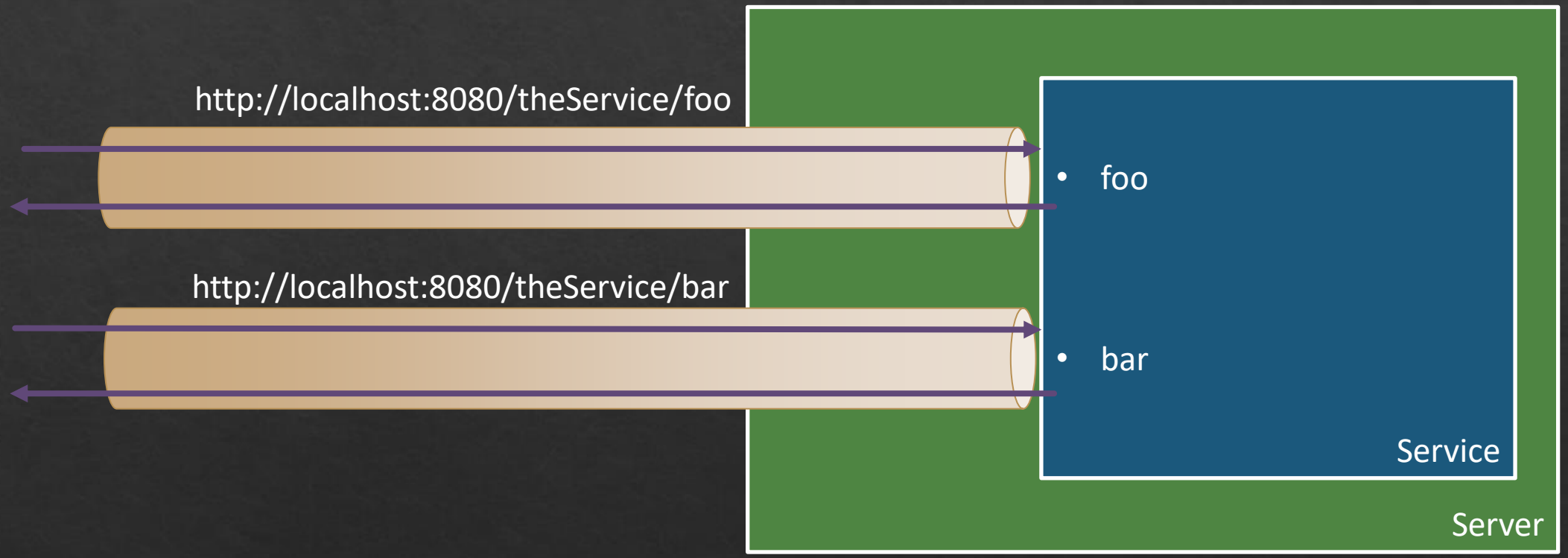
Layered System

- Between the client requests and the server response there can be a number of components/servers in the middle.
- The layers can provide security, caching, load-balancing or other functionality.
- The client is agnostic as to how many layers.

Code on demand

- This constraint is optional
- The client can request code from the server, and then the response from the server will contain some code

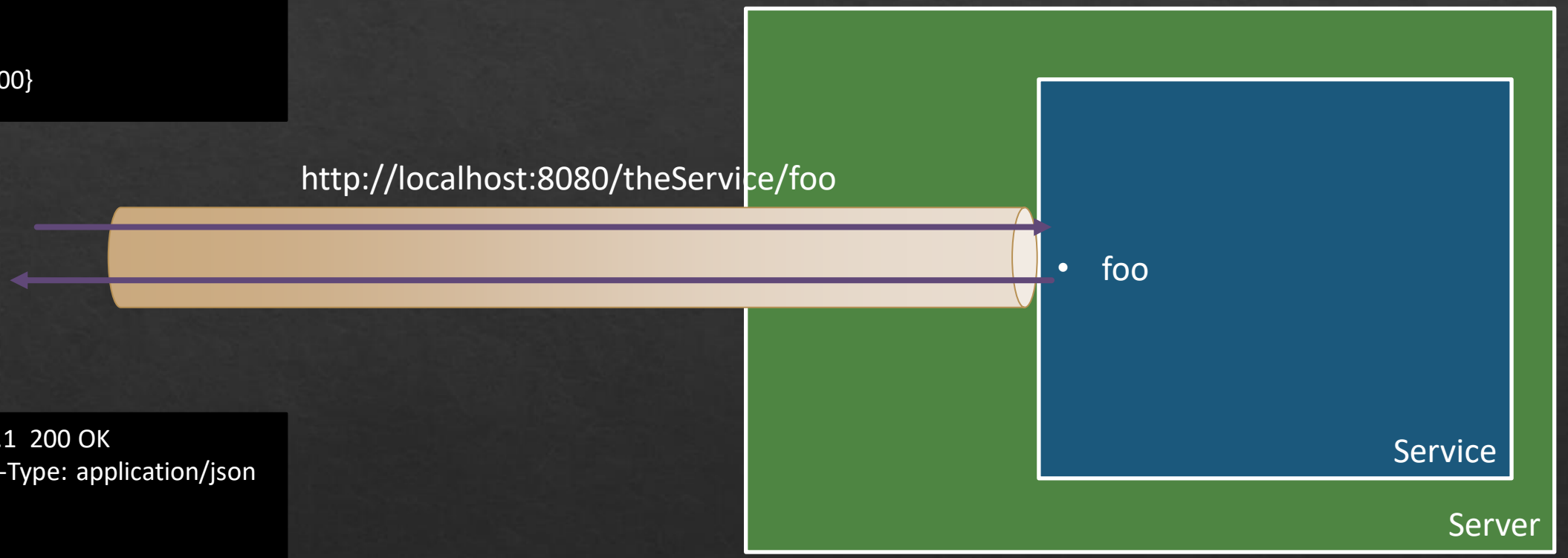
REST Service



REST Service

HTTP 1.1
Content-Type: application/json

{ “id”: 100}



HTTP 1.1 200 OK
Content-Type: application/json

{“name”: “Anil Joseph”}

Http Methods

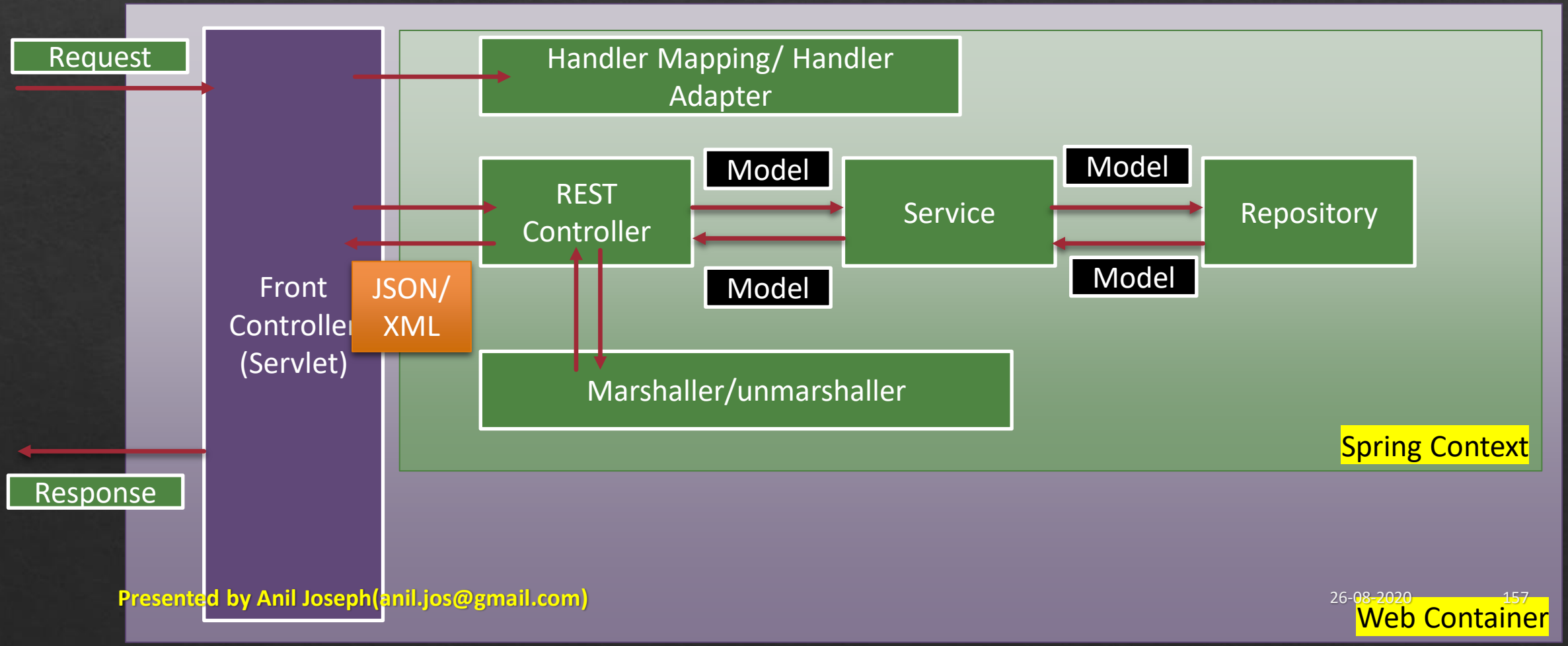
Get	To retrieve or read a resource.
Post	To create a new resource.
Delete	To delete/remove an existing resource.
Put	To update an existing resource.
Patch	Used to update a slice of the resource

Presented By Anil Joseph (anilj.j@gmail.com)

Spring MVC REST

- Spring MVC supports the creation of REST Services
- @RestController introduced in Spring 4.0
  - Defines a controller for REST
  - Combines @Controller and @ResponseBody
- @RequestMapping
  - Maps a method to a request URL

Spring MVC Stack



Spring Boot

- Provides a radically fast and widely accessible getting started experience for all Spring development.
- Automatically configures Spring whenever necessary
- Opinionated
  - Provides default configuration
- Customization when requirements diverge form the defaults
- Provides a range of non-functional features
  - Security, Metrics, externalized configuration
- Creates a standalone application
- Provides an embedded tomcat, Jetty.
  - Facilities easy deployments on the cloud
- Based on Maven and Gradle
- No requirements for XML configuration

Getting Started

- Spring CLI
  - A command line tool
  - Allows scripts (Groovy/Java)
  - Very less boiler point code
- Spring Initializer(https://start.spring.io/)
  - A web interface to generate the Maven or Gradle projects
- Eclipse or IntelliJ
  - Provides templates to create spring boot applications

Spring Boot Application

- @SpringBootApplication
  - The annotation is used on the main class of the application
  - Its an alternative to @Configuration, @ComponentScan & @EnableAutoConfiguration
- @Configuration
  - Indicates that a class declares on or more @Bean methods.
  - @Configuration classes are typically bootstrapped using
    - AnnotationConfigApplicationContext
    - AnnotationConfigWebApplicationContext
- @ComponentScan
  - Provides support parallel with Spring XML's <context:component-scan> element.

Spring Boot Application

- @EnableAutoConfiguration
  - Enable auto-configuration of the Spring Application Context
  - Automatically loads all the beans required by the application
- SpringApplication class
  - The SpringApplication class provides a convenient way to **bootstrap** a Spring application that will be started from a main() method
  - Will create an ***AnnotationConfigApplicationContext*** or ***AnnotationConfigEmbeddedWebApplicationContext***
  - Provides a set of events and listeners

Object Relation Mapping(ORM)

- An ORM provides an automated mechanism of persisting objects to the database.
- ORM aims to enable the business logic developer deal with underlying persistence only in terms of business entities.
- An ORM solution provides
  - Facility for specifying mapping metadata
  - API's for CRUD operations
  - Language of API for performing queries on objects
  - Facility for transactions, lazy associations fetching and other optimizations

ORM Implementations

Hibernate

EclipseLink

Data Nucleus

Java Persistent API(JPA)

- JPA is a specification that describes the management of relational data in applications using Java Platform.
- The Java Persistence API originated as part of the work of the Java Enterprise Edition.
- JPA comprises of 3 areas
  - The Java Persistent API's itself
  - The Query language JPQL
  - The mapping metadata
  - Using Annotations
  - XML mapping files

Implementations of JPA

- EclipseLink (formerly Oracle TopLink)
- JBoss with Hibernate
- ObjectDB
- Apache OpenJPA
- IBM, for WebSphere Application Server
- Batoo JPA
- DataNucleus (formerly JPOX)
- Kundera

JPA

- Applications use JPA through a set of interfaces.
- Persistence
  - A Bootstrapper class that is used to obtain an instance of the EntityManagerFactory.
- EntityManagerFactory
  - Manages the connections to the database
- EntityManager
  - Manages the entity and its associations
- EntityTransactions
  - Defines the transactions
- Query
  - Execute JPA queries

EntityManagerFactory

Represents a particular persistence unit.

There is typically a single instance for the an application per database(designed as a singleton).

The EntityManagerFactory caches generated SQL statements and other mapping metadata that is used at runtime.

This is a thread-safe object.

Entity Manager

- Primary interface used by the application when interacting with the database.
- Provides the methods to control the lifecycle of entities and creations of queries.
- Light weight interface that acts as a persistence manager.
- It is not thread safe and so should be used one per thread.
- Represents one unit of work.

- The EntityManger provides the first level(L1) of caching.

Entity

An entity is a lightweight persistence domain object.

Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table.

An object of entity type has its own database identity (primary key value).

An entity has its own lifecycle; it may exist independently of any other entity

Entity Requirements

The class must be annotated with the javax.persistence.Entity annotation.

The class must have a public or protected, no-argument constructor. The class may have other constructors.

The class, its methods and persistent instance variables must not be declared final.

Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.

Persistent Fields

The fields or properties of an entity must be of the following Java language types:

Java primitive types

java.lang.String

Wrappers of Java primitive types

java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.TimeStamp

ser defined Serializable types

Annotations

@Entity	Defined at the class level for an entity
@Table	Defined at the class level for an entity to map to the database table
@Column	Defined on a field or property to map to a database table column
@Id	Defined on a field or property to mark as the primary key
@Basic	Defined on a field or property
@Temporal	Defined on a field or property for date types Date, Time , DateTime

Embeddable Classes in Entities

Embeddable classes are used to represent the state of an entity but don't have a persistent identity of their own.

They share the identity of the entity that owns it.

Annotated with the javax.persistence.Embeddable

Example: JPA

```
Run | Debug
public static void main(String[] args) {

    EntityManagerFactory factory =
        Persistence.createEntityManagerFactory("manager1");
    EntityManager manager =
        factory.createEntityManager();
    EntityTransaction tx =
        manager.getTransaction();
    tx.begin();

    Message m = new Message();

    m.setId(100);
    m.setText("JPA Demo");
    manager.persist(m);

    tx.commit();
    manager.close();
}
```

Spring Data JPA

- Spring Data JPA makes it easy to easily implement JPA based repositories.
- Sophisticated support to build repositories based on Spring and JPA
- Support for Querydsl predicates and thus type-safe JPA queries
- JavaConfig based repository configuration by introducing @EnableJpaRepositories.
- Pagination support, dynamic query execution, ability to integrate custom data access code

JPA Repository (Spring Data JPA)

- JpaRepository is JPA specific extension of Repository.
- It contains the full API for the CRUD operations
- Its also provides the API for the sorting and pagination

JPA Mapping Associations

- JPA allows us to define the associations between entities as
- Unidirectional
- Bidirectional
- Multiplicity
  - One to One
  - One to many
  - Many to Many

Open API Specification

- The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs
- Allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.
- When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.
- An OpenAPI definition can then be used by
  - Documentation generation tools to display the API
  - Code generation tools to generate servers and clients in various programming languages
  - Testing tools, and many other use cases.

Swagger

- Swagger is a set of open-source tools built around the OpenAPI Specification
- Used to design, build, document and consume REST APIs.
- The major Swagger tools include:
  - Swagger Editor – browser-based editor where you can write OpenAPI specs.
  - Swagger UI – renders OpenAPI specs as interactive API documentation.
  - Swagger Codegen – generates server stubs and client libraries from an OpenAPI spec.

Thank You

ANIL JOSEPH