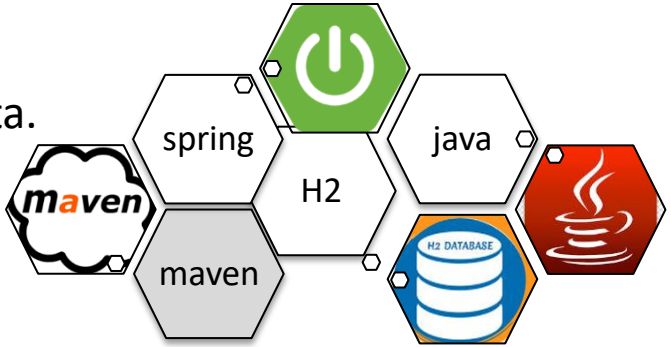
TRAINING BASKET
QUALIFY FOR TOMORROW

Writing continuously integrate-able code :Spring Boot

Dheeraj Singh (Trainer Software)

MCP, MCTS, Perform Cloud Data Science with Azure Machine Learning

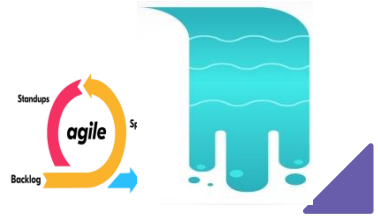
- Java 8 Features Quick Review .
- Spring Boot, Spring MVC, Spring Data.
- Automated Build(s) with Maven.
- H2Database.
- Overview of CI, CD pipelines.



Agenda :

1. Use Case -1
2. Obstructs in practicing Agile.
3. Agile vs waterfall model identify and bridge the gap.
4. Introduction to java8, it helps practicing Agile.
5. Code has value, maintain it through SOLID principles by Uncle Bob.
6. Data Modeling | Data Structures By Java (demo).
7. Apache Maven (Automated Builds).
8. Introduction to Spring Framework, MVC, JPA, H2 Database .
9. What is CD CI also IAC (Infrastructure As Code)?
10. DevOps to extend agility.
11. Host the a application on Heroku (PaaS) .

TAKE AWAY FROM THE SESSION



Agile

Waterfall

DevOps

Java8 Lambdas

Improvements

SOLID Principles

maven

Maven

Java JEE

Spring JPA
hibernate
Spring Boot

H2 Database

Why CI CD pipelines are important

Hosting on Heroku Cloud Application platform

Lets have a quick look at the Use-case to know what stops us delivering the application incrementally by comparing waterfall with agile;

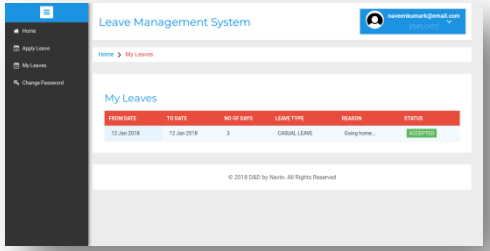
Well known Software Development methodologies.

Application Today

todo App

Heroku Cloud

www.myapp.herokuapp.com



Web Layer

Service Layer

Repository Layer

DTOS

- DTOs
- Domain Models

MONGO

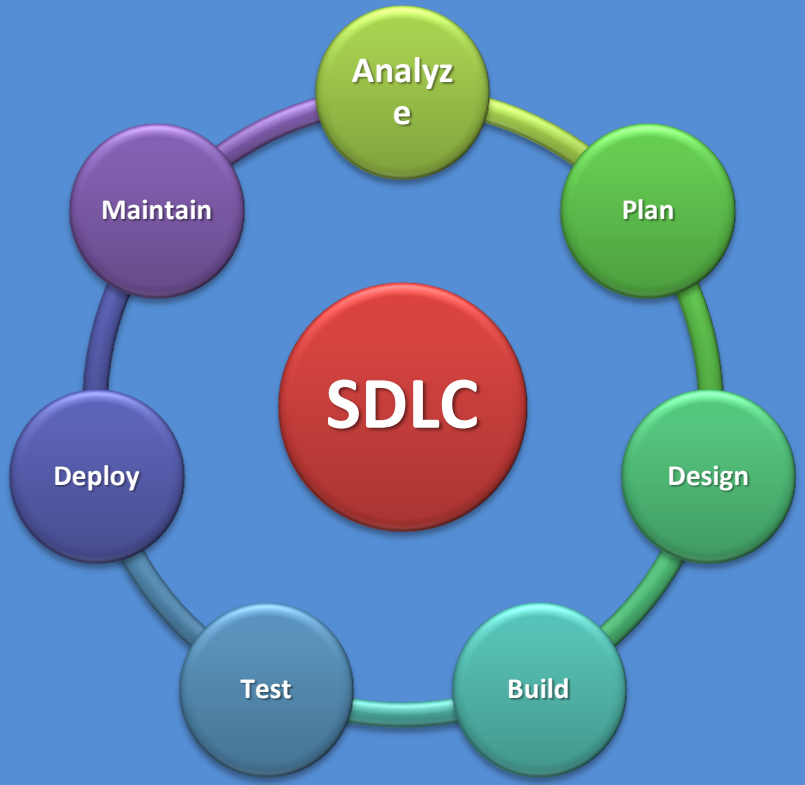
Kibana

MySQL

MS-SQL

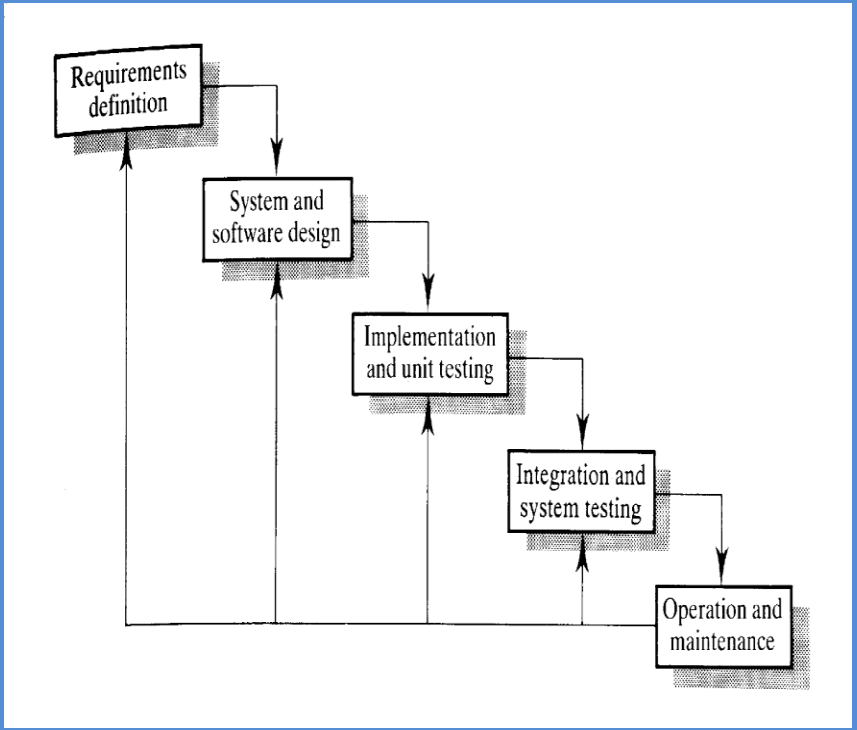

PLEASE
Make it live In a day, How it is even possible

WHAT IS WATERFALL?

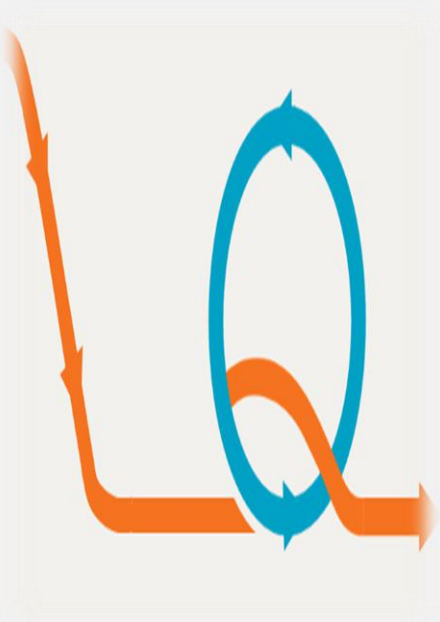


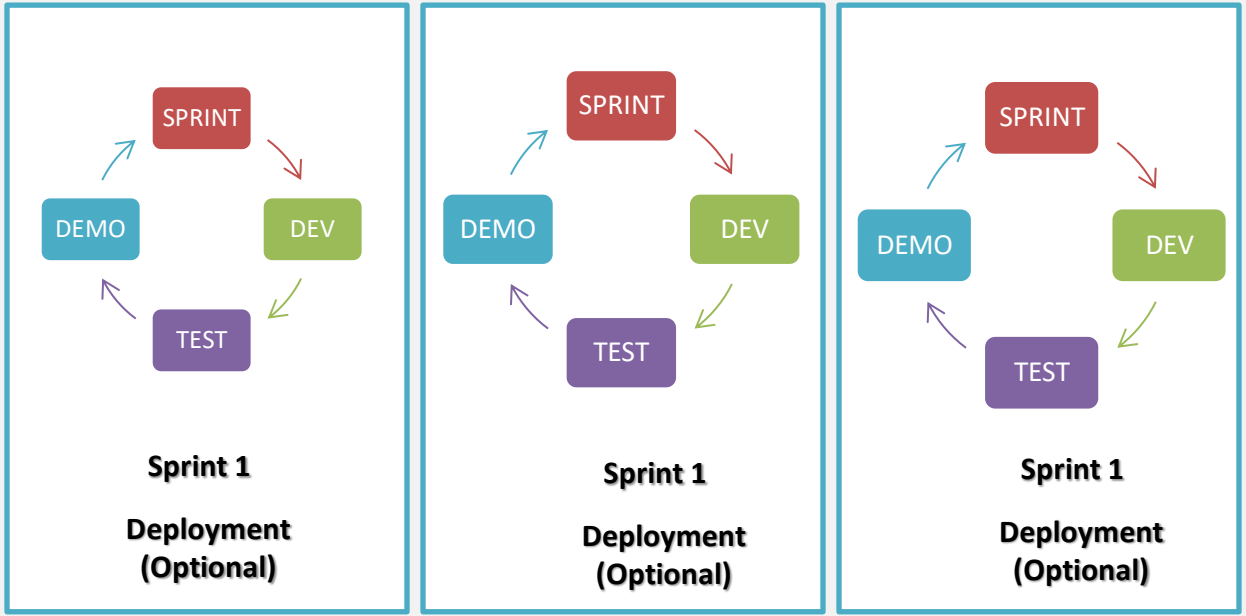
PROBLEMS IN WATERFALL?

1. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
2. No working software is produced until late during the life cycle.
3. High amounts of risk and uncertainty.
4. Not a good model for complex and object-oriented projects.
5. Poor model for long and ongoing projects.
6. Not suitable for the projects where requirements are at a moderate to high risk of changing.



PROBLEMS IN WATERFALL?

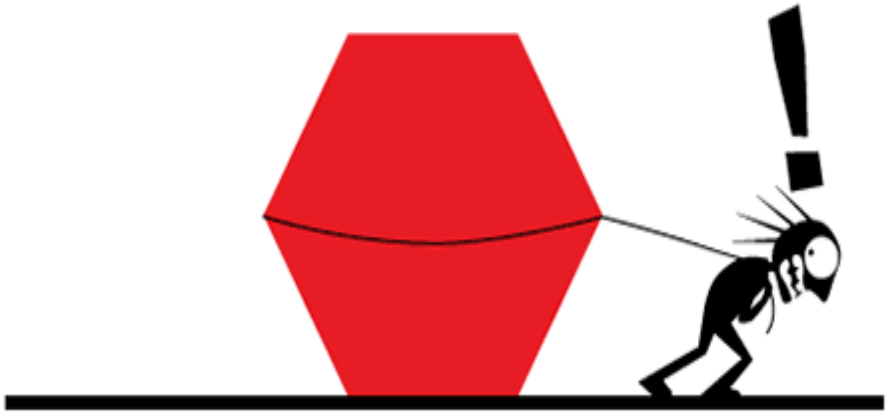




Divide and Conquer works well in computers

THE BIG DIFFERENCE

THE WATERFALL PROCESS

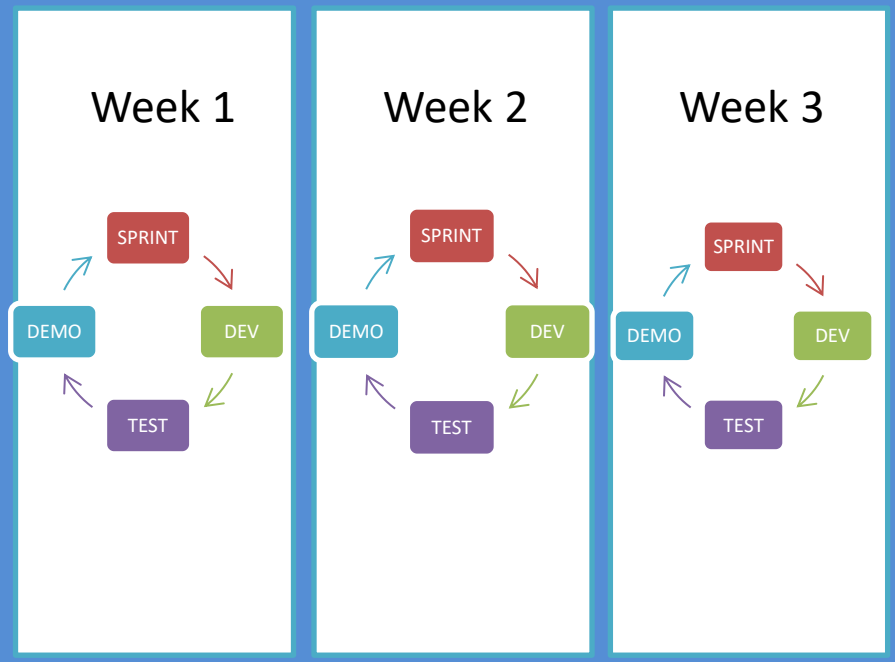


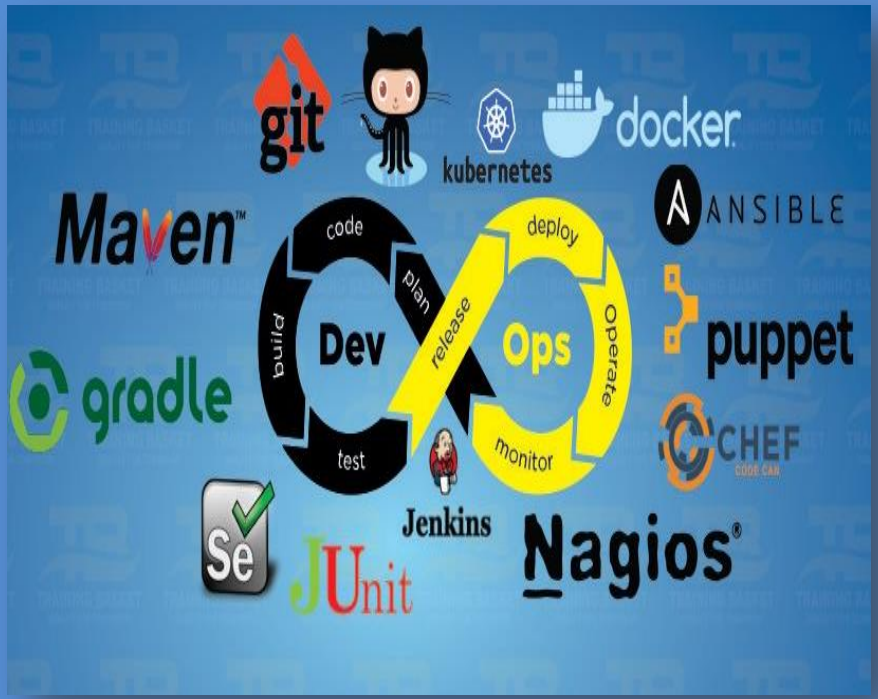
'This project has got so big, I'm not sure I'll be able to deliver it!'

THE AGILE PROCESS

'It's so much better delivering this project in bite-sized sections'

DEVOPS MINDSET TO ENHANCE AGILITY





Introduction to java8, it helps practicing Agile.

WHAT IS JAVA ?

Java is a set of several computing platform & programming language specifications from Sun Microsystems (which has since merged with Oracle Corporation)

WHAT IS JAVA ?

Java is a fully functional, platform independent and object oriented programming language it has powerful set of machine independent libraries, including windowing (GUI) libraries.

THIS ACTUALLY IS JAVA

- Development Environment
- Application Environment
- Deployment Environment
- Programming Language

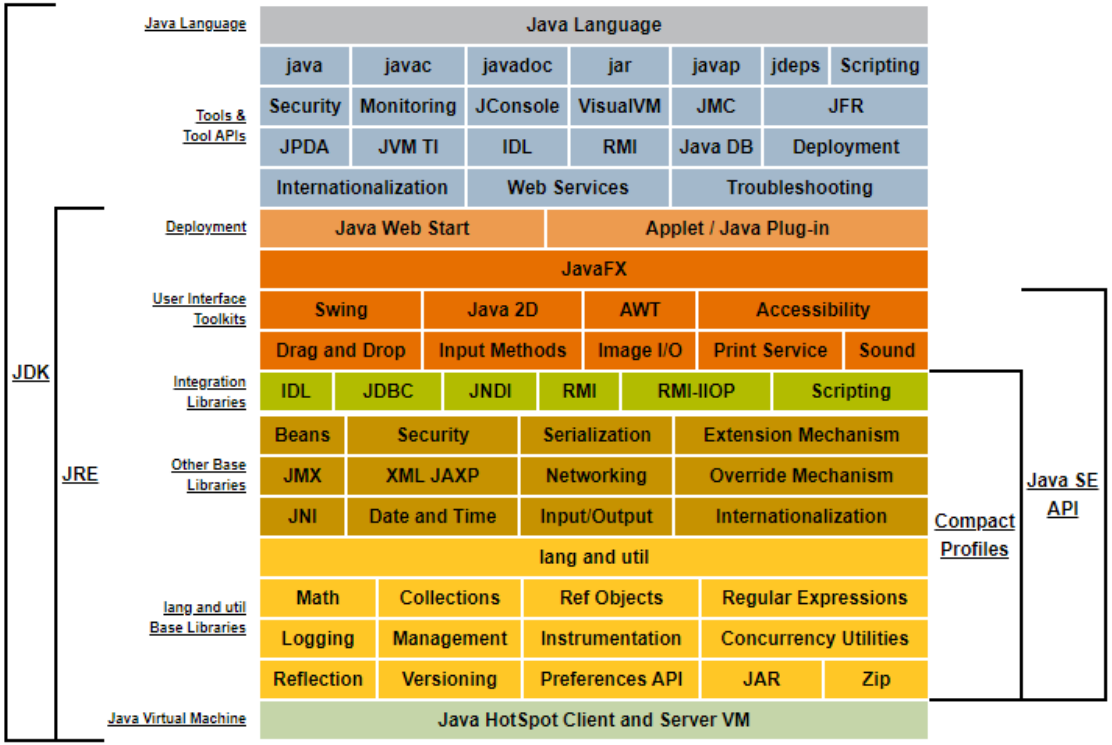
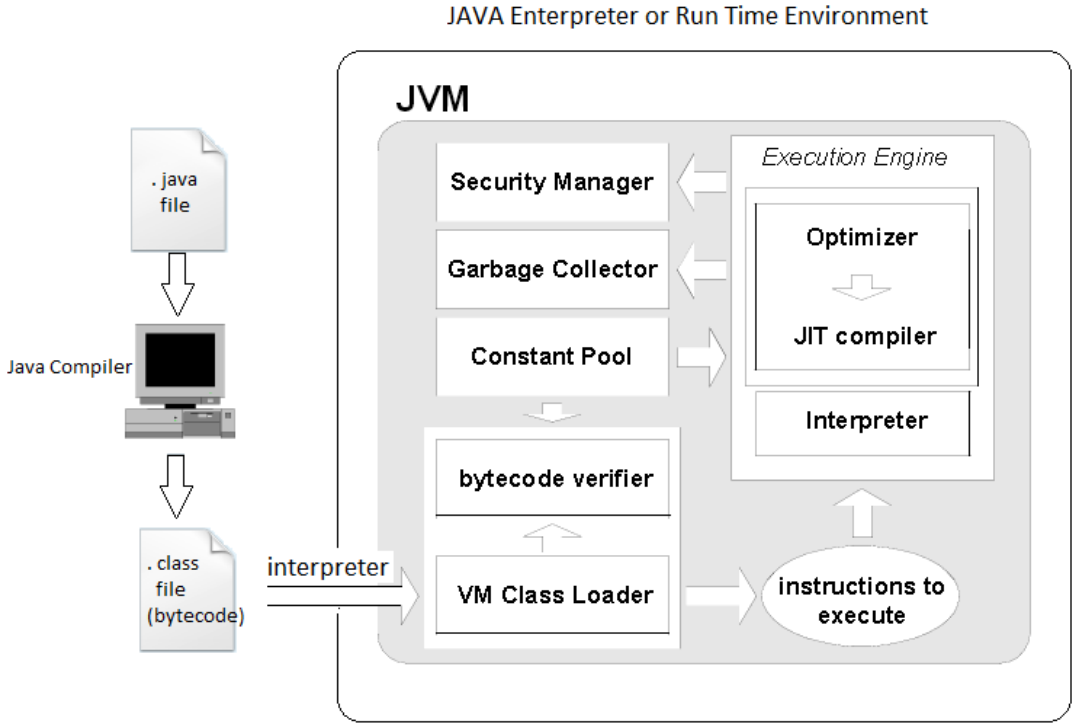


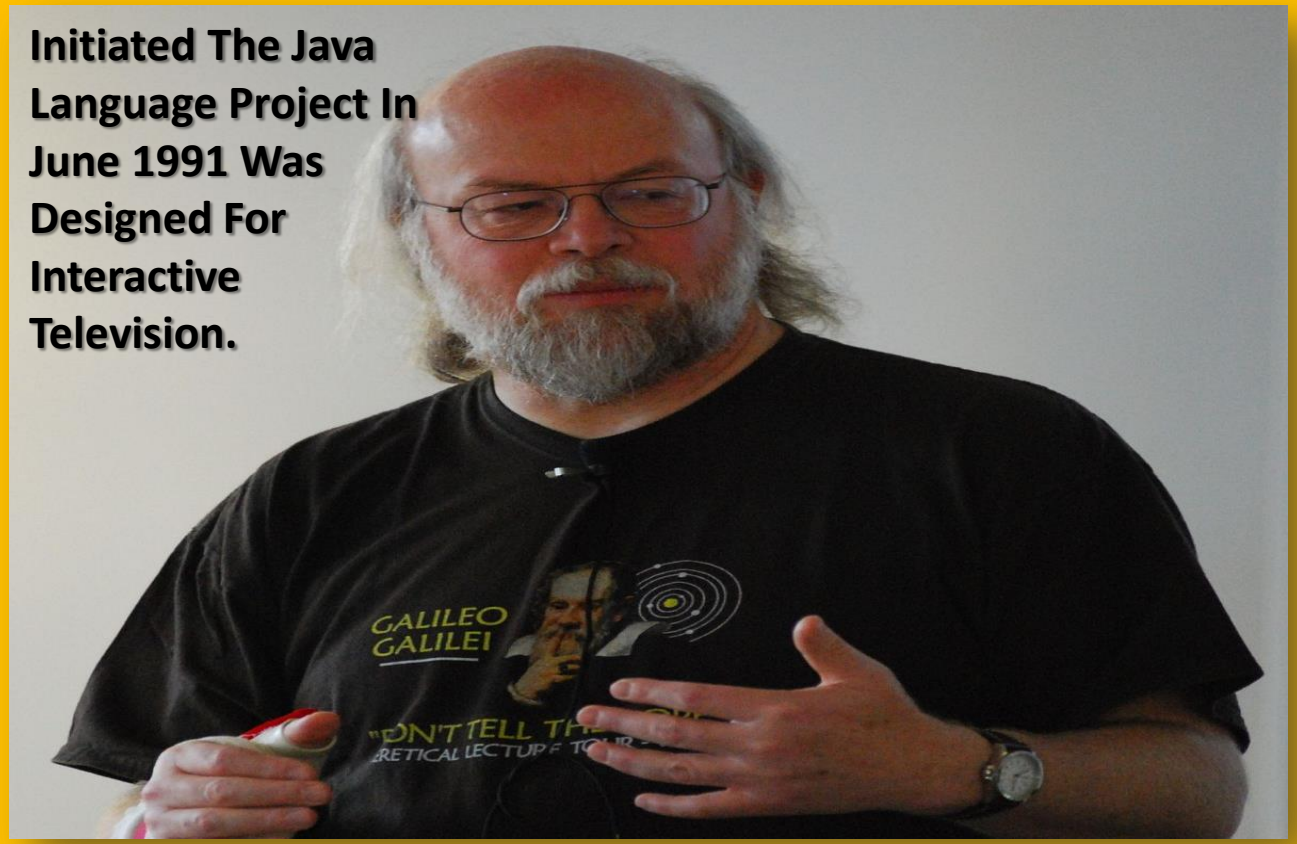
Image Credits : https://docs.oracle.com/javase/8/docs/

HOW DOES JAVA WORK ?

Java applications are typically compiled to the byte code(class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.



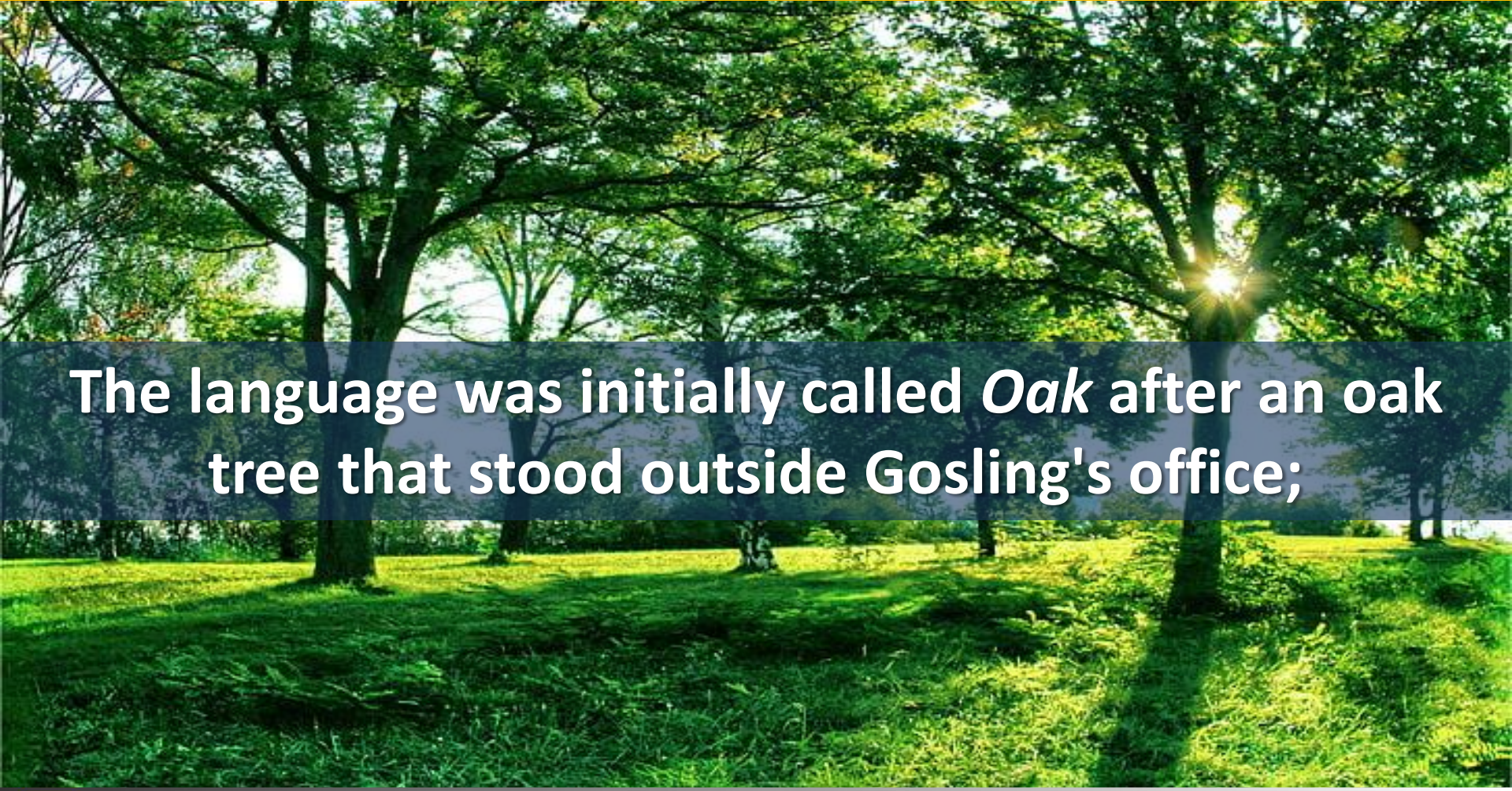
JVM Disassembled

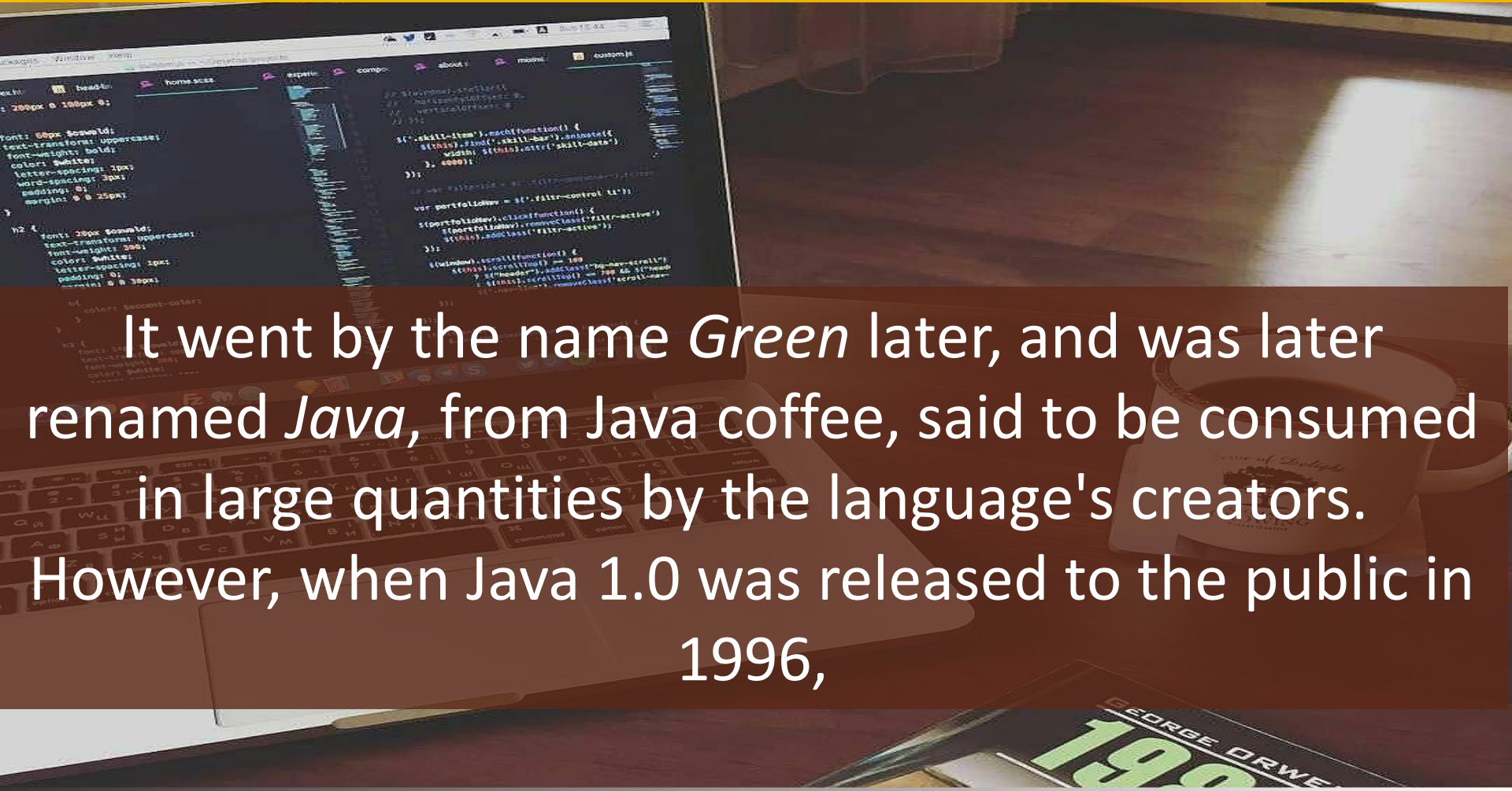


James Gosling

Mike Sheridan

Patrick Naughton


The language was initially called *Oak* after an oak tree that stood outside Gosling's office;



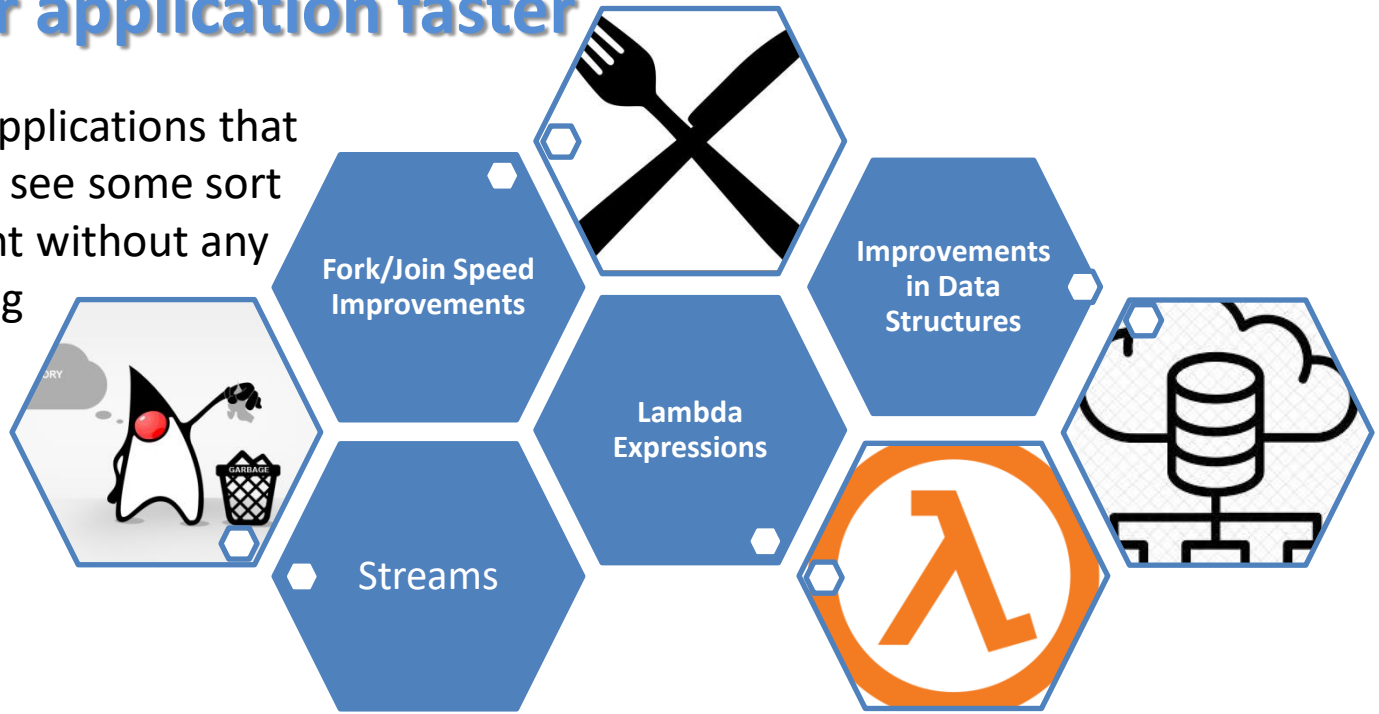
It went by the name *Green* later, and was later renamed *Java*, from Java coffee, said to be consumed in large quantities by the language's creators. However, when Java 1.0 was released to the public in 1996,

USE JAVA IT IS REALLY FAST:

features that make java8 is preferred for computationally expensive use cases.

Java 8 runs your application faster

Generally speaking, applications that have moved to Java 8 see some sort of speed improvement without any specific work or tuning



WHY JAVA :

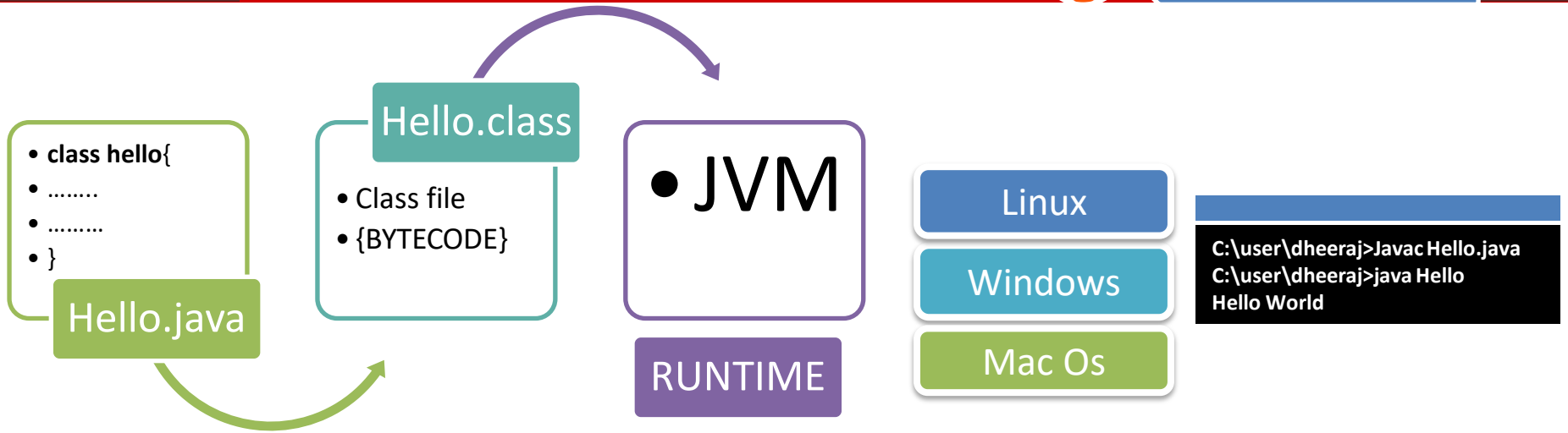
Reason # 2

It has been #1 or #2 among most popular languages since its creation in the mid 90s.

Many of the world's biggest companies use Java to build desktop apps and backend web systems

WHY JAVA :

Reason # 3



Thanks to the platform-agnostic Java Virtual Machine (JVM), Java can run on nearly every system. Java is also the most popular Android language, so the vast majority of Android apps are built in Java.

WHY JAVA :

Reason # 4

“When web companies grow up, they become Java shops”.

Java is built for scalability in mind, which is why it is so popular among enterprises and scaling startups (Twitter moved from Ruby to Java for scaling purposes). Since Java is a statically-typed language, it’s faster and easier to maintain with less bugs. It is also backwards compatible, which means old versions of the language will still run perfectly even after new versions are released. This is a big relief for businesses who would otherwise worry about rewriting their code every time a new version comes out.

James Governor

WHY JAVA :

Reason # 5

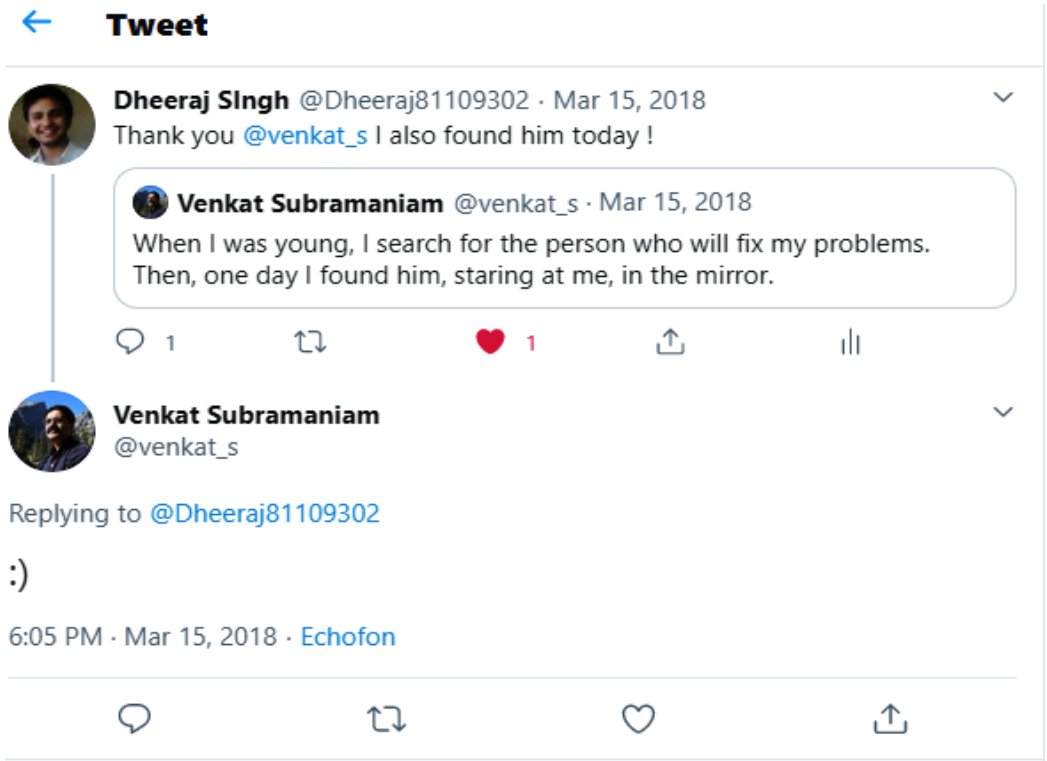
The popularity of Java helps to ensure its future popularity, thanks to a huge community of users. With massive Stack Overflow and GitHub communities, developers can find help on virtually any problem they might encounter. Coupled with its portability, developers know that investing in Java will pay dividends for a long, long time.

stackoverflow

Only Releasable Code Has Value

Maintain it through SOLID principles by Uncle Bob.

WHY JAVA :



If you have written SOLID compliant code the that is only (you) who can fix the code very easily.

Lets' know more....


Robert C. Martin
(You may also remember him as the author of Clean Code)

ADVANTAGE OF OOAD

What is object oriented design?

What is it all about?

What are it's benefits?

What are it's costs?

It may seem silly to ask these questions in a day and age when virtually every software developer is using an object oriented language of some kind. Yet the question is important because, it seems to me, that most of us use those languages without knowing why, and without knowing how to get the most benefit out of them.

DEMO TIME

Data Modeling | Data Structures By Java

DEMO TIME

USE CASE -1

Make a program to emulate leave management system in java

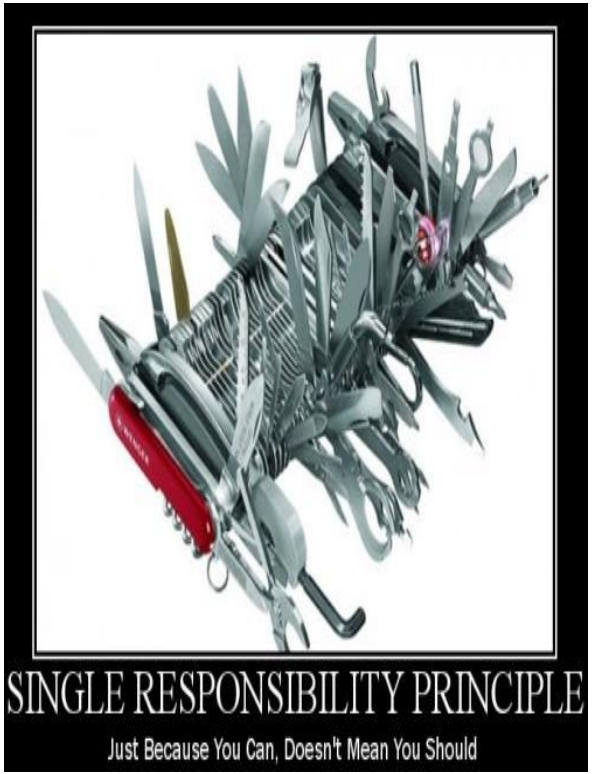
Lets test some code in java8 digesting the updates and features of the language Need to write a program that stores employee into a collection , being a Java-SE practitioner need to make a Data-Model and store some employees make them sort on various properties like first name, last name, marks, score in games etc..

We will make a web app for this but now we just need to think of Data Model .

WHAT IS SOLID:

1. S : SRP (Single Responsibility Principle)
2. O : OCP (Open Close Principle)
3. L : LSP (Liskov Substitution Principle)
4. I : Interface Segregation Principle
5. D : Dependency Inversion Principle

SRP - SINGLE RESPONSIBILITY PRINCIPLE



SRP - Single responsibility Principle

"The Single Responsibility Principle states that a class should have one, and only one, reason to change. "

SRP - SINGLE RESPONSIBILITY PRINCIPLE

When the Single Responsibility Principle is followed, testing is easier. With a single responsibility, the class will have fewer test cases. Less functionality also means fewer dependencies to other classes. It leads to better code organization since smaller and well-purposed classes are easier to search.

SRP - SINGLE RESPONSIBILITY PRINCIPLE

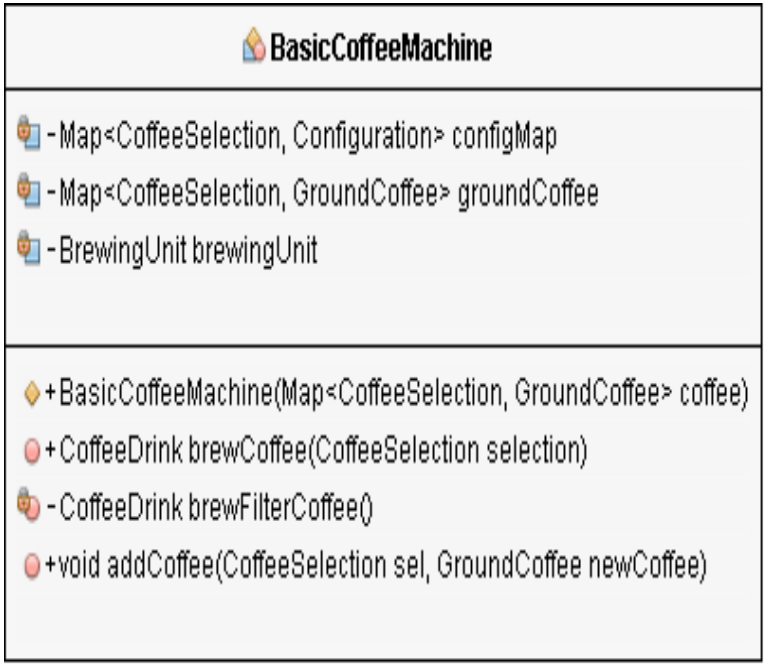
```
public class UserSettingService
{
   public void changeEmail(User user)
   {
      if(checkAccess(user))
      {/ /Grant option to change }
   }
   public boolean checkAccess(User user)
   {
            //Verify if the user is valid.
   }
}
```

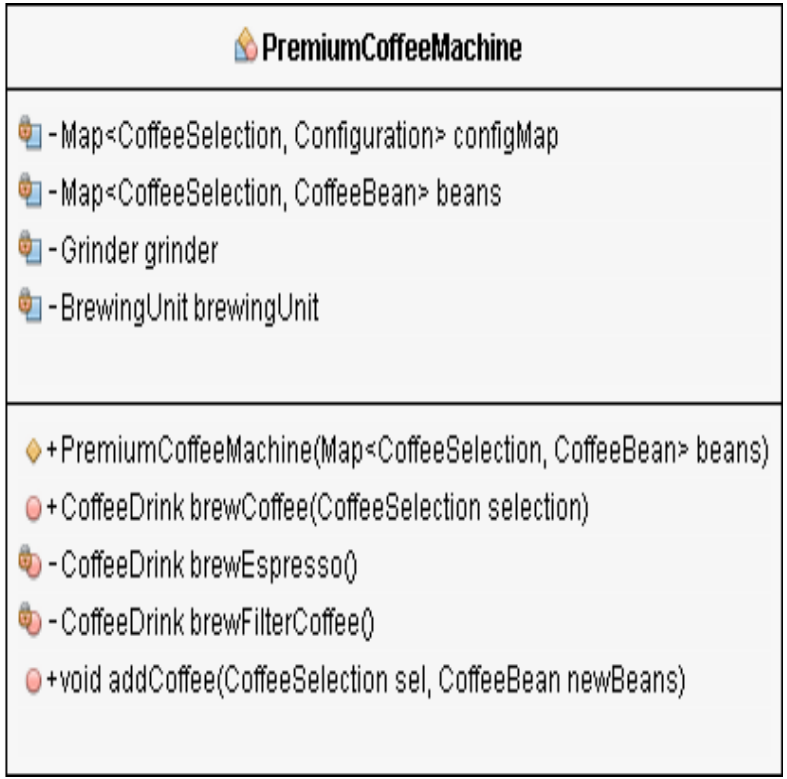
```
public class UserSettingService
{
   public void changeEmail(User user)
   {
      if(SecurityService.checkAccess(user))
      {  //Grant option to change   }
   }
}
public class SecurityService
{
   public static boolean checkAccess(User user)
   {  //check the access.  }
}
```

OPEN CLOSED PRINCIPLE

Software components should be open for extension, but closed for modification.

LOSKOV SUBSTITUTION PRINCIPLE

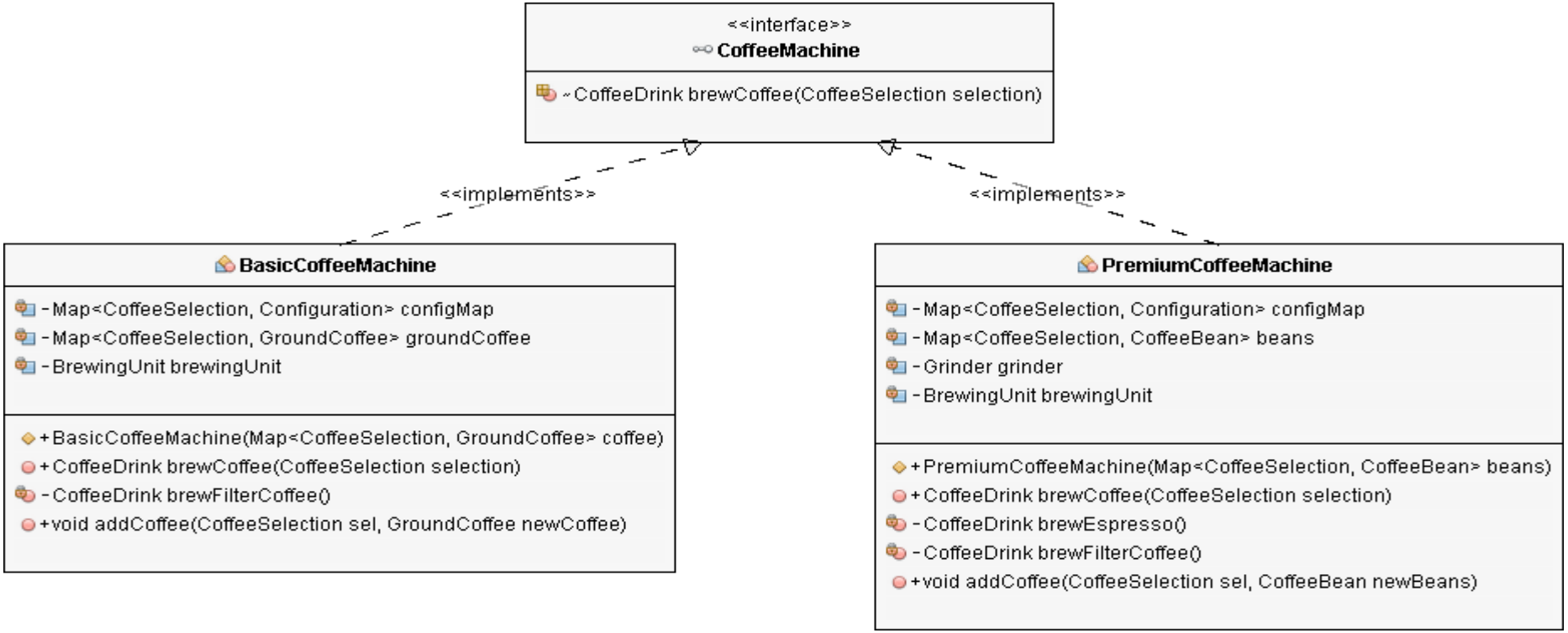




LOSKOV SUBSTITUTION PRINCIPLE

"This principle states that objects should be replaceable with instances of their subtypes without altering the correctness of that program"

LOSKOV SUBSTITUTION PRINCIPLE



ISP - INTERFACE SEGREGATION PRINCIPLE

"This principle states that no implementation of an interface should be forced to depend on methods it does not use."

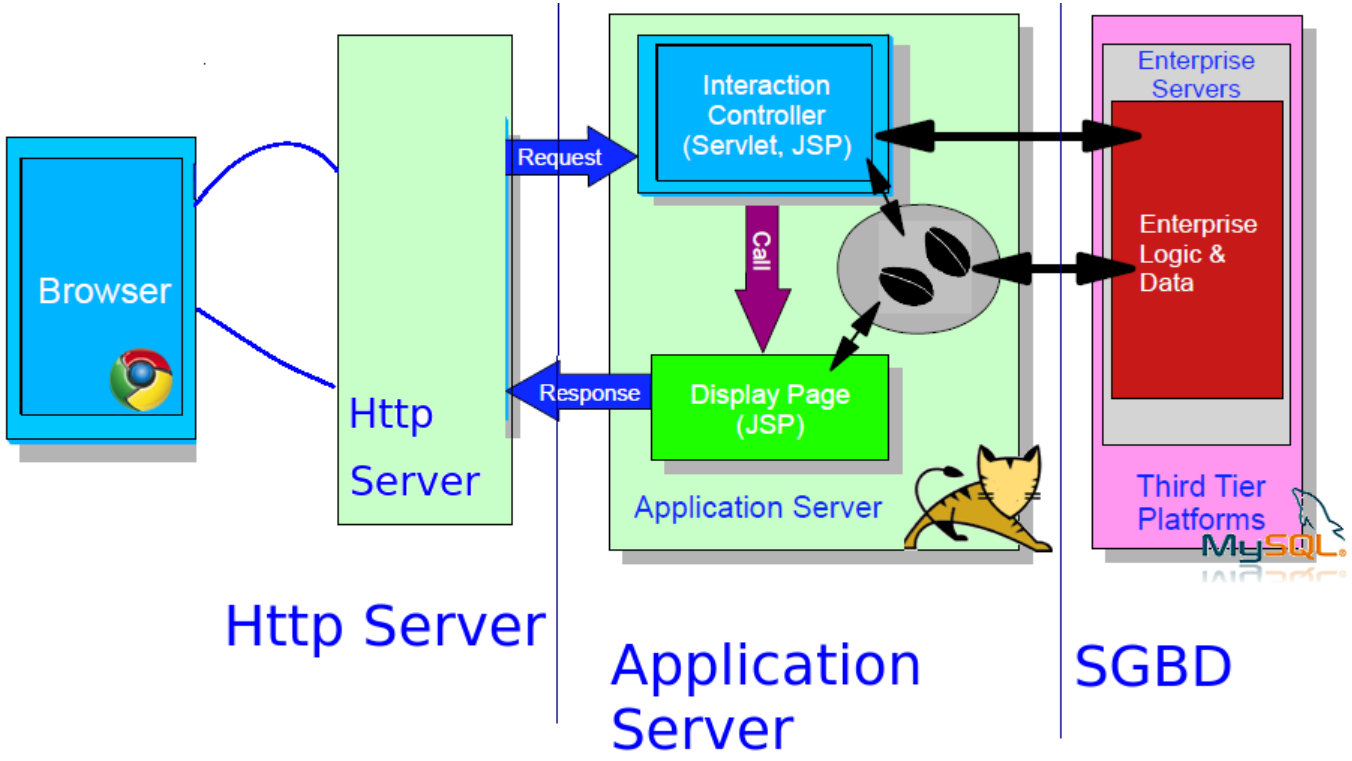
DEPENDENCY INVERSION PRINCIPLE

"This principle states that high-level code should not depend on low-level code, and that abstractions should not depend upon details."

It allows a programmer to remove hardcoded dependencies so that the application becomes loosely coupled and extendable.

THE JEE APPLICATION

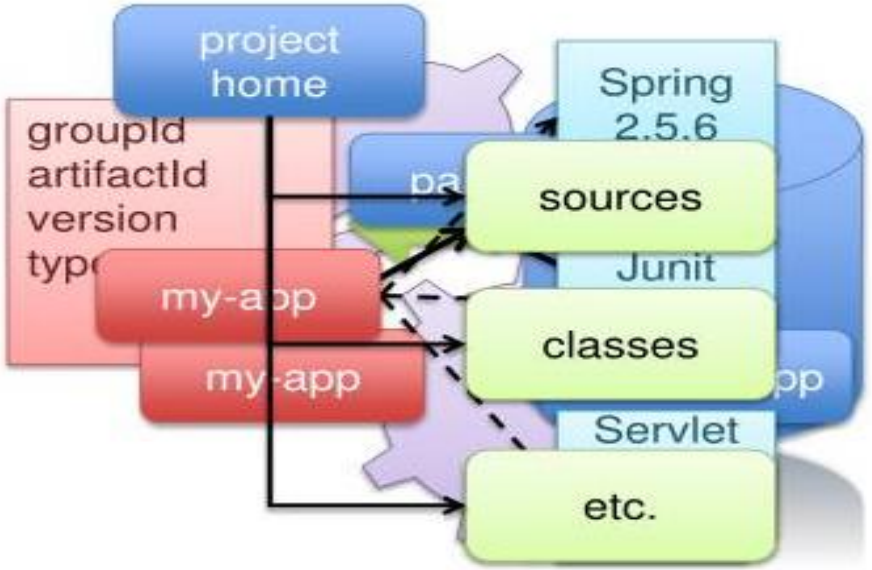
How would I remember this, I am an absolute beginner ?



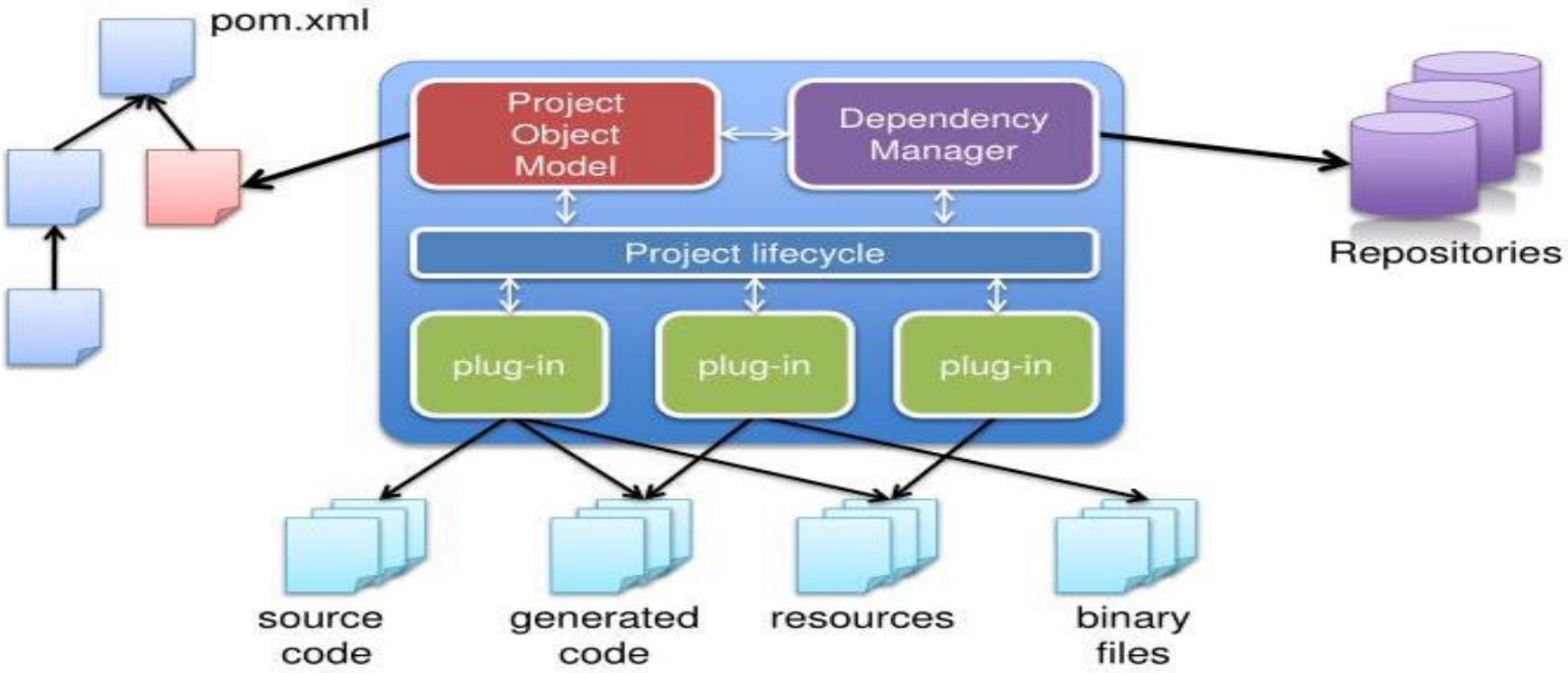
- Open Source: Apache License 2.0
- Uniform build system
- Knowledge accumulator
- Project management tool
- Declarative logic instead of imperative

MAVEN STRUCTURE

- Identification
- Hierarchy
- Structure
- Dependencies
- Build settings



MAVEN STRUCTURE



WHAT WE WILL COVER?

1. Overview
2. Spring IO Platform
3. Spring Framework
4. Environment Setup
5. Inversion of Control
6. Dependency injection
7. Spring Web MVC Framework
8. Spring Security

OVERVIEW

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, reusable code.

BENEFITS

- Spring enables developers to develop enterprise-class applications using POJOs.
- Spring is organized in a modular fashion.
- Spring does not reinvent the wheel instead, it truly makes use of some of the existing technologies.
- Testing an application written with Spring is simple because environment-dependent code is moved into this framework.
- Spring's web framework is a well-designed web MVC framework.
- Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers.
- Spring provides a consistent transaction management.

SPRING IO PLATFORM

SPRING IO PLATFORM - CORE

Spring Framework	Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.
Spring Security	Protects your application with comprehensive and extensible authentication and authorization support.

SPRING IO PLATFORM - DATA

Spring IO addresses modern data landscape — whether it be document, graph, key-value, relational, or simply unstructured files.

SPRING IO PLATFORM - WORKLOADS

Integration	Channels, Adapters, Filters, Transformers
Batch	Jobs, Steps, Readers, Writers
Big data	Ingestion, Export, Orchestration, Hadoop
Web	Controllers, REST, WebSocket

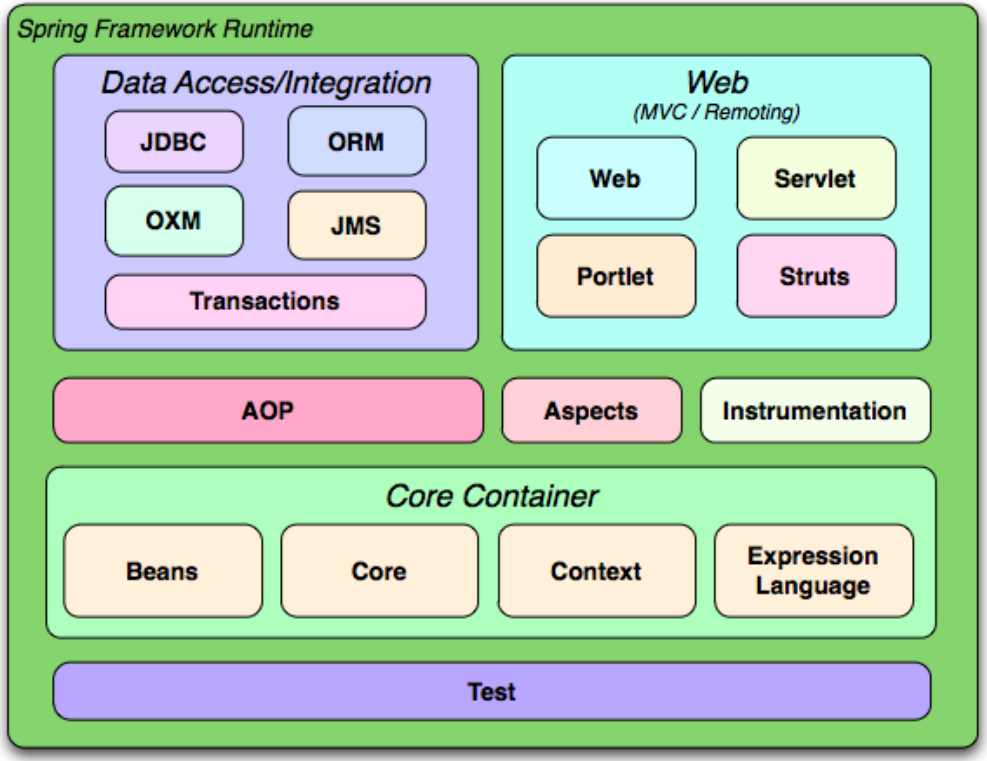
SPRING FRAMEWORK

Spring is the most popular application development framework for enterprise Java.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

SPRING FRAMEWORK - COMPONENTS



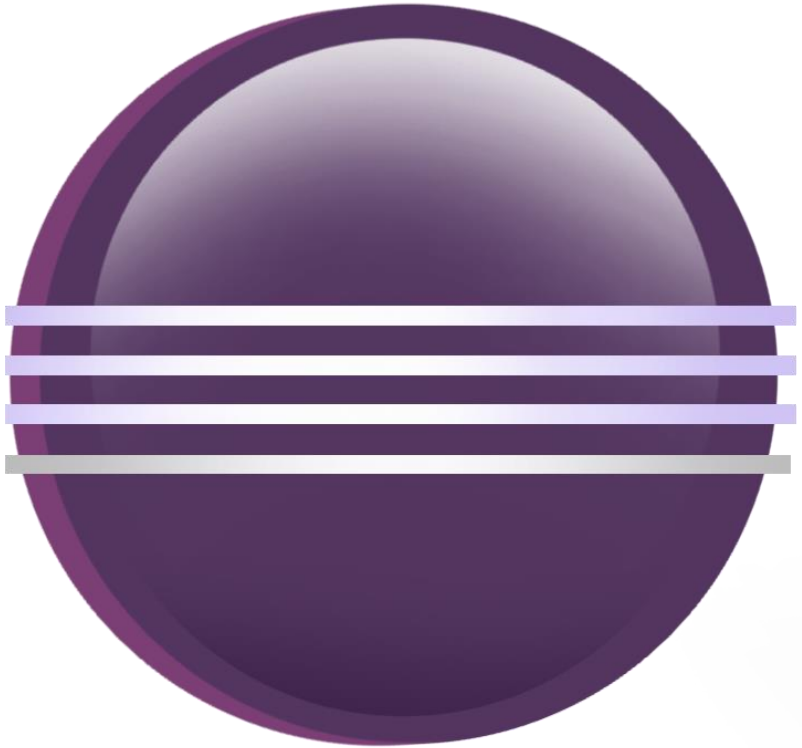
SPRING FRAMEWORK - CORE CONTAINER

- The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Bean** module provides BeanFactory which is a sophisticated implementation of the factory pattern.
- The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured.
- The **Expression Language** module provides a powerful expression language for querying and manipulating an object graph at runtime.

SPRING FRAMEWORK - WEB

- Spring's **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The **Web-Servlet** module contains Spring's model-view-controller (MVC) implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms, and integrates with all the other features of the Spring Framework.
- The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.
- The **Web-Struts** module contains the support classes for integrating a classic Struts web tier within a Spring application.

ENVIRONMENT SETUP



ECLIPSE

- Download Eclipse from http://www.eclipse.org/downloads
- Decompress downloaded file into the directory of your choice (e.g. "C:\Eclipse" on Windows)
- ???
- PROFIT!!!

MAVEN INTEGRATION FOR ECLIPSE

- Open Eclipse
- Go to Help -> Eclipse Marketplace
- Search by Maven
- Click "Install" button at "Maven Integration for Eclipse" section
- Follow the instruction step by step

HELLO WORLD

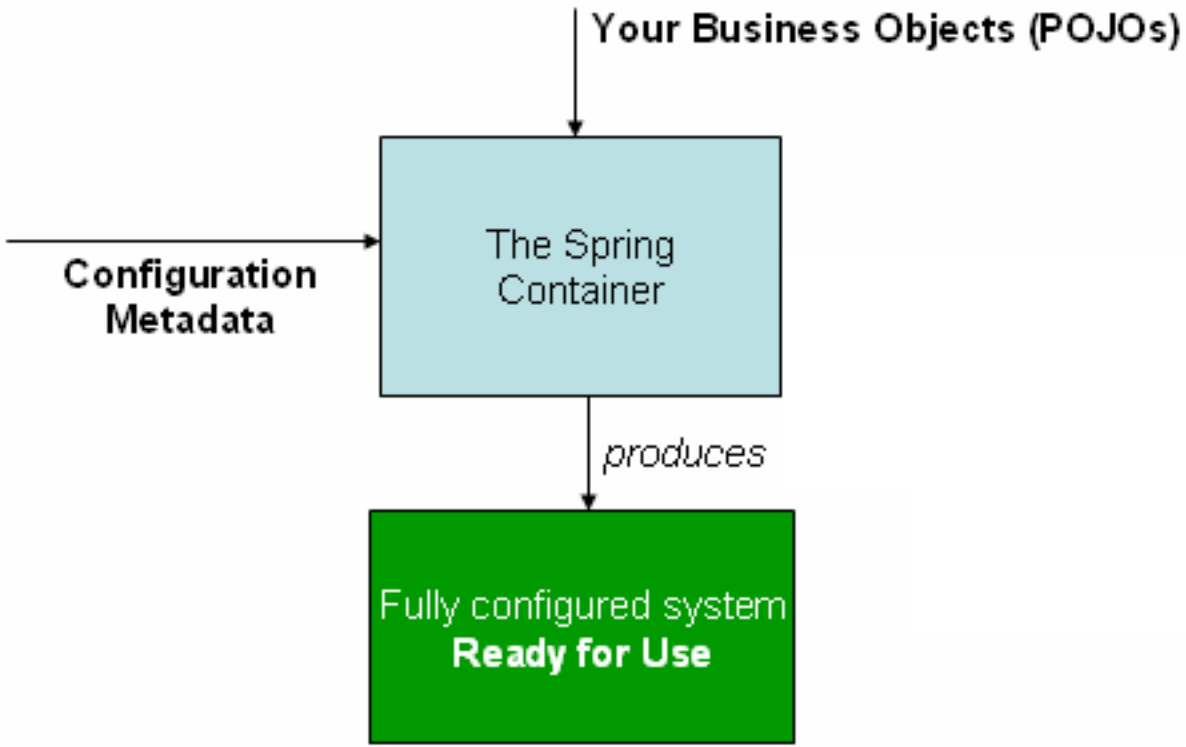
https://github.com/dheeraj-thedev/java-webminar-repo.git

ECLIPSE

Inversion of control (IoC) is a programming technique in which object coupling is bound at run time by an assembler object and is typically not known at compile time using static analysis. In order for the assembler to bind objects to one another, the objects must possess compatible abstractions.

- is a concept in application development
- "don’t call me, I’ll call you“
- one form is Dependency Injection (DI)

INVERSION OF CONTROL



INVERSION OF CONTROL

The Spring container is at the core of the Spring Framework.
The Spring container uses dependency injection (DI) to manage the components that make up an application.
The container will create the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction.
The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code.

DEPENDENCY INJECTION CONTAINERS

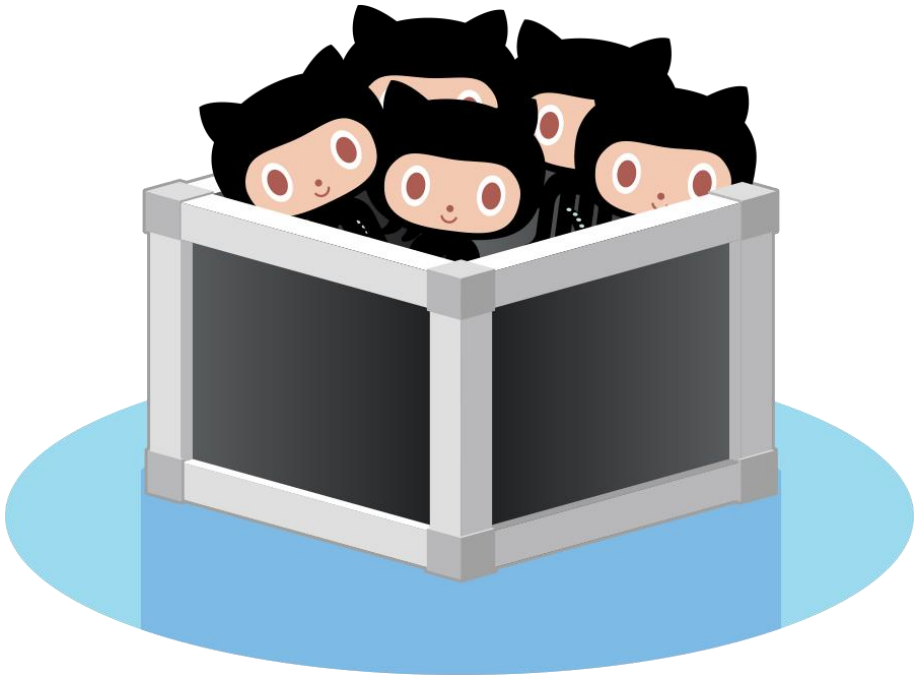
- Spring **BeanFactory** Container

This is the simplest container providing basic support for DI. There are a number of implementations of the BeanFactory interface that come supplied straight out-of-the-box with Spring. The most commonly used BeanFactory implementation is the XmlBeanFactory class.

- Spring **ApplicationContext** Container

The ApplicationContext includes all functionality of the BeanFactory, it is generally recommended over the BeanFactory. It adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

CONTAINERS EXAMPLE



BEANS

https://github.com/dheeraj-thedev/java-webminar-repo.git

BEANS

The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.

A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions which you have already seen in previous chapters.

BEANS - DEFINITION

The bean definition contains the information called configuration metadata which is needed for the container to know the followings:

- How to create a bean
- Bean's lifecycle details
- Bean's dependencies

BEANS - DEFINITION

Property	Description
class*	The bean class to be used to create the bean.
name	The unique bean identifier.
scope	The scope of the objects created from a particular bean definition.
autowiring mode	Used to specify autowire mode for a bean definition
lazy-initialization mode	Tells the IoC container to create a bean instance when it is first requested, rather than at startup.

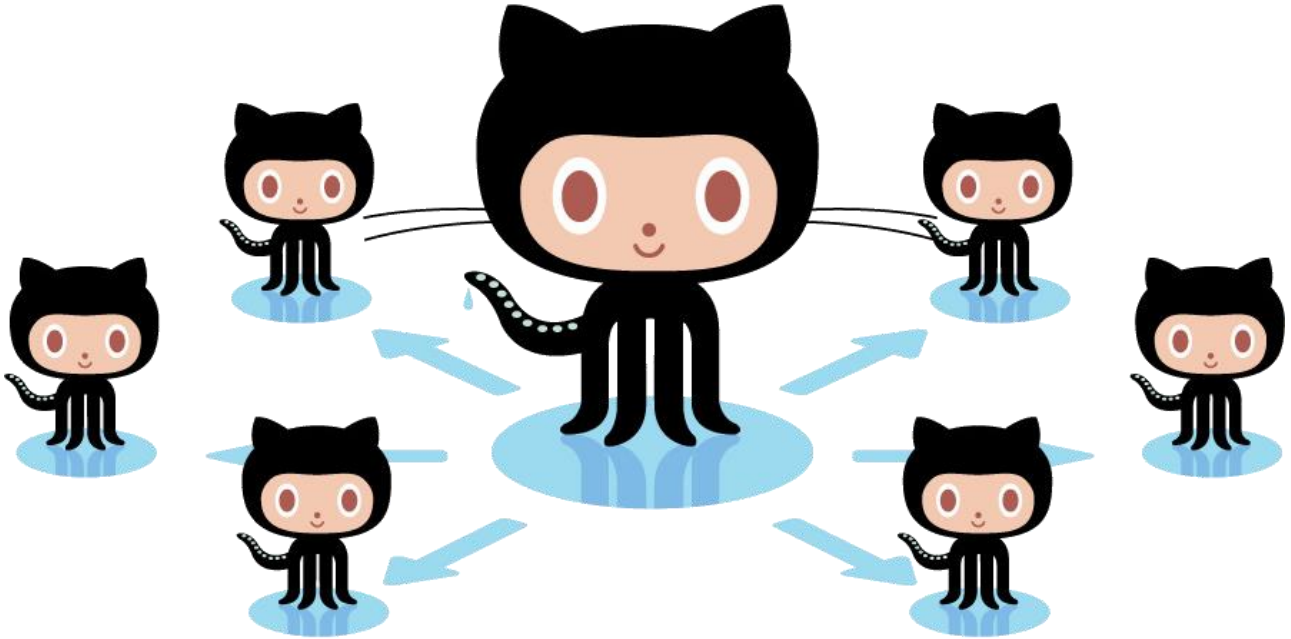
BEANS - DEFINITION

Property	Description
constructor-arg	Used to inject the dependencies into the class through a class constructor
properties	Used to inject the dependencies into the class through setter methods

BEANS - DEFINITION

Property	Description
initialization method	A callback to be called just after all necessary properties on the bean have been set by the container.
destruction method	A callback to be used when the container containing the bean is destroyed.

BEAN DEFINITION EXAMPLE



https://github.com/dheeraj-thedev/java-webminar-repo.git

BEANS - SCOPES

Scope	Description
singleton	This scopes the bean definition to a single instance per Spring IoC container (default).
prototype	This scopes a single bean definition to have any number of object instances.
request *	This scopes a bean definition to an HTTP request.
session *	This scopes a bean definition to an HTTP session.
global-session *	This scopes a bean definition to a global HTTP session.

BEANS - SCOPES EXAMPLE

https://github.com/dheeraj-thedev/java-webminar-repo.git

BEANS - DEFINITION INHERITANCE

- Spring Bean definition inheritance has nothing to do with Java class inheritance.
- You can define a parent bean definition as a template and other child beans can inherit required configuration from the parent bean.
- A child bean definition inherits configuration data from a parent definition. The child definition can override some values, or add others, as needed.
- When you use XML-based configuration metadata, you indicate a child bean definition by using the parent attribute, specifying the parent bean as the value of this attribute.

BEANS - INHERITANCE EXAMPLE

https://github.com/dheeraj-thedev/java-webminar-repo.git

BEANS - LIFECYCLE

- The life cycle of a Spring bean is easy to understand.
- When a bean is instantiated, it may be required to perform some initialization to get it into a usable state.
- When the bean is no longer required and is removed from the container, some cleanup may be required.

BEANS - LIFECYCLE - INITIALIZATION

The org.springframework.beans.factory.InitializingBean interface specifies a single method:

```
void afterPropertiesSet() throws Exception;
```

In the XML-based configuration metadata, you can use the init-method attribute to specify the name of the method that has a void no-argument signature.

```
<bean id="..." class="..." init-method="init"/>
```

Annotate the method with @PostConstruct

```
@PostConstruct public void init() { ... }
```

BEANS - LIFECYCLE - DESTRUCTION

The org.springframework.beans.factory.DisposableBean interface specifies a single method:

```
void destroy() throws Exception;
```

In the XML-based configuration metadata, you can use the destroy-method attribute to specify the name of the method that has avoid no-argument signature.

```
<bean id="..." class="..." destroy-method="destroy"/>
```

Annotate the method with @PreDestroy

```
@PreDestroy public void destroy() { ... }
```

BEANS - MULTIPLE LIFECYCLE MECHANISMS

Multiple lifecycle mechanisms configured for the same bean are called in the following order:

Initialization:

- Methods annotated with @PostConstruct
- afterPropertiesSet() as defined by the InitializingBean callback interface
- A custom configured init() method

Destruction:

- Methods annotated with @PreDestroy
- destroy() as defined by the DisposableBean callback interface
- A custom configured destroy() method

BEANS - LIFECYCLE EXAMPLE

https://github.com/dheeraj-thedev/java-webminar-repo.git

DEPENDENCY INJECTION & AUTOWIRING

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while doing unit testing.

Dependency Injection (or sometime called wiring) helps in gluing these classes together and same time keeping them independent.

DEPENDENCY INJECTION TYPES

Type	Description
Constructor-based	Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
Setter-based	Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

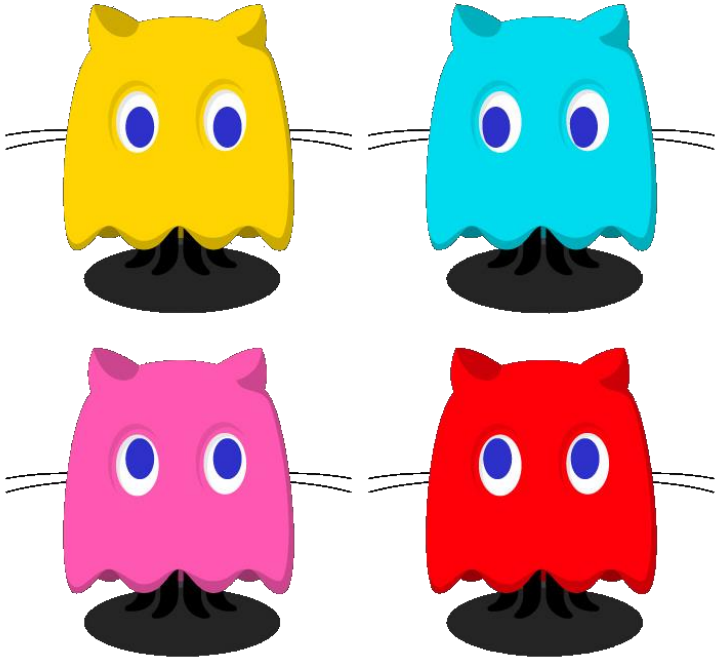
CONSTRUCTOR-BASED DEPENDENCY INJECTION

Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

SETTER-BASED DEPENDENCY INJECTION

Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

DEPENDENCY INJECTION TYPES EXAMPLE

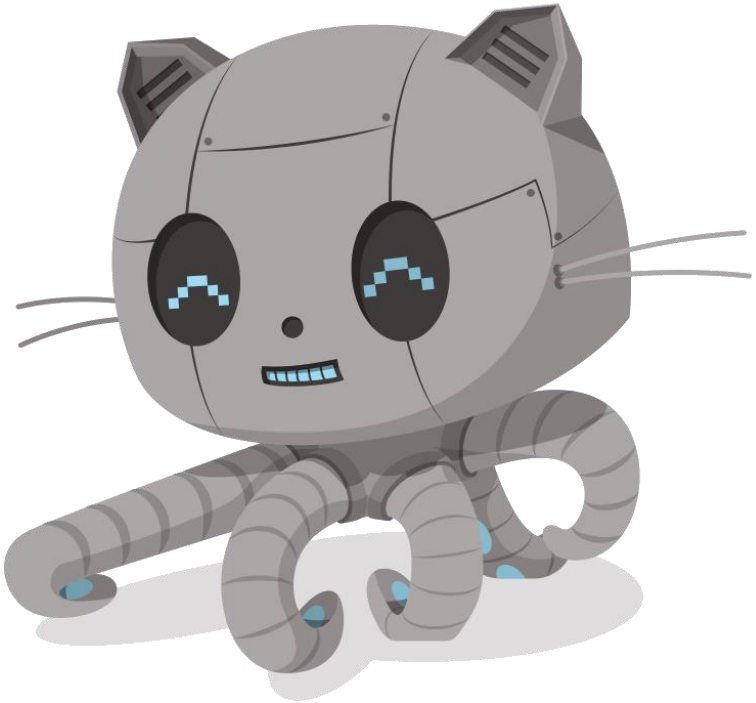


https://github.com/dheeraj-thedev/java-webminar-repo.git

AUTOWIRING MODES

Scope	Description
no	No autowiring. You should use explicit bean reference for wiring.
byName	Autowiring by property name. Spring container looks at the properties of the beans on which autowire attribute is set to byName in the XML configuration file.
byType	Autowiring by property datatype. Spring container looks at the properties of the beans on which autowire attribute is set to byType in the XML configuration file.
constructor	Similar to byType, but type applies to constructor arguments.
autodetect	Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

AUTOWIRING EXAMPLE



https://github.com/dheeraj-thedev/java-webminar-repo.git

ANNOTATION BASED CONFIGURATION

Starting from Spring 2.5 it became possible to configure the dependency injection using annotations. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Annotation injection is performed before XML injection, thus the latter configuration will override

the former for properties wired through both approaches.

ANNOTATION BASED CONFIGURATION

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider to have following configuration file in case you want to use any annotation in your Spring application.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<context:annotation-config/>
</beans>
```

ANNOTATION BASED CONFIGURATION

@Autowired	The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.
@Qualifier	The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
@Required	The @Required annotation applies to bean property setter methods.
JSR-250 Annotations	Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

JAVA BASED CONFIGURATION

Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations explained below.

Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions.

The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

SPRING WEB MVC FRAMEWORK

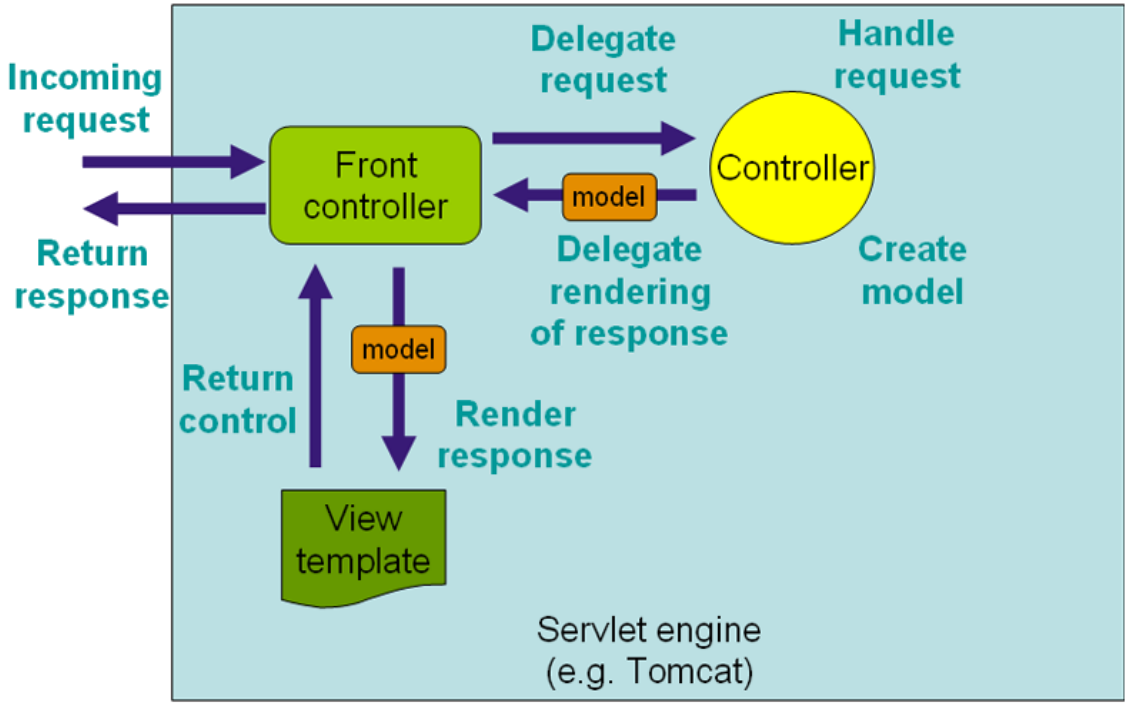
The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications.

The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

SPRING WEB MVC FRAMEWORK

- The Model encapsulates the application data and in general they will consist of POJO.
- The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The Controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

SPRING WEB MVC FRAMEWORK



BASIC CONFIGURATION

AppInitializer.java

```
public class AppInitializer implements WebApplicationInitializer {
	@Override
public void onStartup(ServletContext servletContext) throws ServletException {
		WebApplicationContext context = getContext();
		ServletRegistration.Dynamic dispatcher = servletContext.addServlet(
				"DispatcherServlet",
				new DispatcherServlet(context)
		);
		dispatcher.setLoadOnStartup(1);
		dispatcher.addMapping("*.do");
		servletContext.addListener(new ContextLoaderListener(context));
	}
private AnnotationConfigWebApplicationContext getContext() {
		AnnotationConfigWebApplicationContext context =
new AnnotationConfigWebApplicationContext();
		context.setConfigLocation("teach.webhello.config");
return context;
	}
}
```

BASIC CONFIGURATION

AppConfig.java

```
@Configuration
@ComponentScan("teach.webhello.config")
public class AppConfig {
}
```

WebConfig.java

```
@Configuration
@EnableWebMvc
@ComponentScan("teach.webhello.controller")
public class WebConfig extends WebMvcConfigurerAdapter {
@Bean
public ViewResolver viewResolver() {
                InternalResourceViewResolver resolver = new InternalResourceViewResolver();
                resolver.setPrefix("/WEB-INF/views/");
                resolver.setSuffix(".jsp");
return resolver;
        }
}
```

CONTROLLERS

DispatcherServlet delegates the request to the controllers to execute the functionality specific to it.

The @Controller annotation indicates that a particular class serves the role of a controller.

The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

The class-level usage of@RequestMapping indicates that all handling methods on this controller are relative to his path.

CONTROLLERS

HelloController.java

```
@Controller
public class HelloController {

	@RequestMapping("/index")
public String hello(ModelMap model) {
		model.addAttribute("date", new Date());
return "hello";
	}
}
```

VIEWS

Spring MVC supports many types of views for different presentation technologies. These include - HTML, PDF, Excel, XML, Velocity, JSON, Atom/RSS feeds, JasperReports etc. But most commonly we use JSP templates written with JSTL.

hello.jsp:

```
<html>
   <head>
      <title>Home page</title>
   </head>
   <body>
      <h1>${message}</h1>
      <p>Today is: ${date}</p>
   </body>
</html>
```

@REQUESTMAPPING DETAILS

URI templates can be used for convenient access to selected parts of a URL in a @RequestMapping method.

A URI Template is a URI-like string, containing one or more variable names. When you substitute values for these variables, the template becomes a URI.

For example, the URI Template http://www.example.com/users/{userId} contains the variable userId. Assigning the value fred to the variable yields http://www.example.com/users/fred.

@REQUESTMAPPING DETAILS

HelloController.java

```
@RequestMapping(value = "/users/{userId}", method = RequestMethod.GET)
public String findUser(@PathVariable String userId, Model model) {
        User user =  userService.findById(userId);
        model.addAttribute("user", user);
        return "displayUser";
}
@Controller
@RequestMapping(value = "/users/{userId}")
public class UsersController {
        @RequestMapping(value = "/docs/{docId}", method = RequestMethod.GET) public String
        findDoc(@PathVariable String userId,
                @PathVariable String docId,
                Model model) {
    ...
  }
}
```

@REQUESTMAPPING DETAILS

```
@RequestMapping(value = "/users/{userId:[a-z-]+}")
public String findUser(@PathVariable String userId, Model model) {
        ...
}
```

@REQUESTMAPPING DETAILS

MATRIX VARIABLES

```
// GET /pets/42;q=11;r=22
@RequestMapping(value = "/pets/{petId}", method = RequestMethod.GET)
public void findPet(@PathVariable String petId, @MatrixVariable int q) {
        // petId == 42
        // q == 11
 }

// GET /owners/42;q=11/pets/21;q=22
@RequestMapping(value = "/owners/{ownerId}/pets/{petId}")
public void findPet(@MatrixVariable(value="q" pathVar="ownerId") int q1,
                @MatrixVariable(value="q" pathVar="petId") int q2) {
  // q1 == 11
  // q2 == 22
}
```

CONSUMABLE MEDIA TYPE

You can narrow the primary mapping by specifying a list of consumable media types. The request will be matched only if the Content-Type request header matches the specified media type.

```
@Controller
@RequestMapping(value = "/pets", method = RequestMethod.POST,
            consumes = "application/json")
public void addPet(@RequestBody Pet pet, Model model) {
   ...
}
```

FORM HANDLING

Form handling is the day-to-day task in general web development as well as in Spring MVC development.

A typical scenario would be like this: the user fills in a web form and click Submit button.

The server receives the user's request, validates inputs, processes some business logic and finally returns a response/message back to the user.

HIBERNATE

Hibernate is a high-performance Object/Relational persistence and query service. Hibernate ORM facilitated the storage and retrieval of Java domain objects via Object/Relational Mapping.

- Natural Programming Model: Hibernate lets you develop persistent classes following natural Object-oriented idioms including inheritance, polymorphism, association, composition, and the Java collections framework.

- Transparent Persistence: Hibernate requires no interfaces or base classes for persistent classes and enables any class or data structure to be persistent.

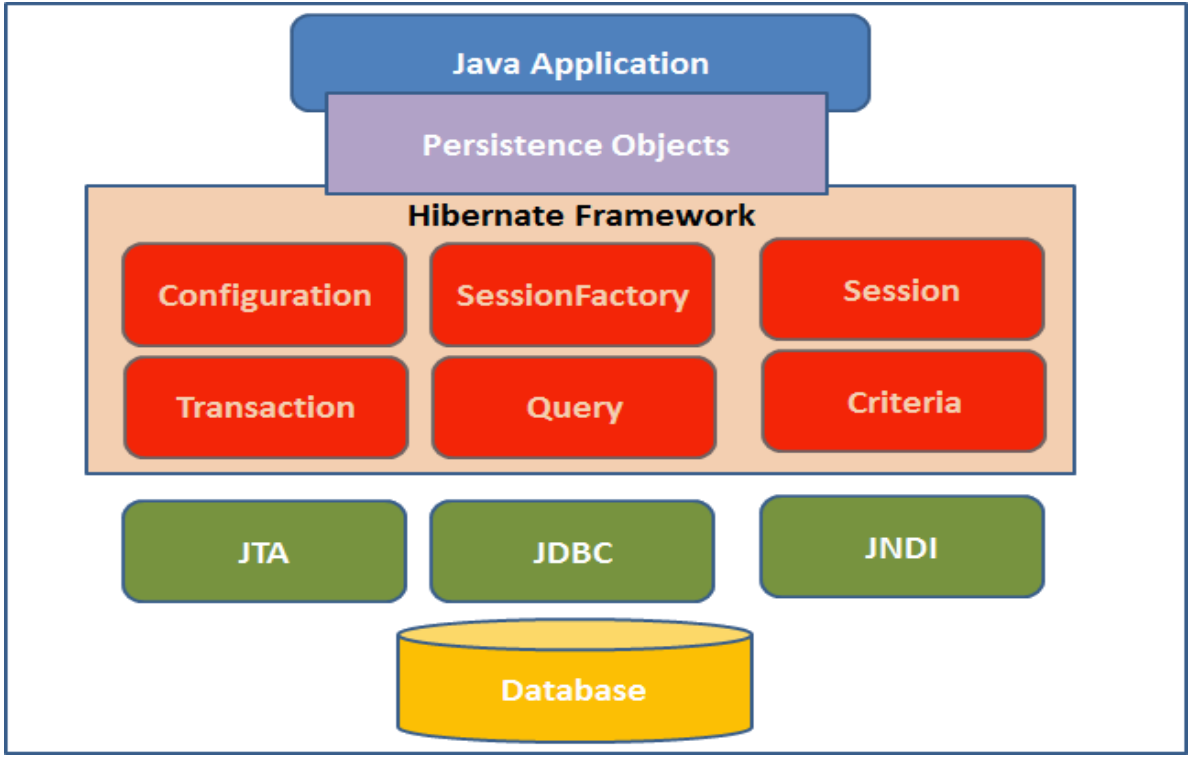
- High Performance: Hibernate supports lazy initialization, many fetching strategies, and optimistic locking with automatic versioning and time stamping.

HIBERNATE ADVANTAGES

- Hibernate takes care of mapping Java classes to database tables without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides Simple querying of data.

HIBERNATE ARCHITECTURE

Hibernate Architecture



HIBERNATE ARCHITECTURE

CONFIGURATION OBJECT

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

SESSIONFACTORY OBJECT

Configuration object is used to create a SessionFactory object which inturn configures Hibernate and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

HIBERNATE ARCHITECTURE

SESSION OBJECT

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.

TRANSACTION OBJECT

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA)

HIBERNATE ARCHITECTURE

QUERY OBJECT

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

CRITERIA OBJECT

Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

HIBERNATE AND SPRING MVC INTEGRATION

```
@Configuration
@EnableTransactionManagement
public class PersistenceConfig {

    @Value("${jdbc.driverClassName}")       private String driverClassName;
    @Value("${jdbc.url}")                   private String url;
    @Value("${jdbc.username}")              private String username;
    @Value("${jdbc.password}")              private String password;
    @Value("${hibernate.dialect}")          private String hibernateDialect;
    @Value("${hibernate.show_sql}")         private String hibernateShowSql;
    @Value("${hibernate.hbm2ddl.auto}")     private String hibernateHbm2ddlAuto;
…
}
```

HIBERNATE AND SPRING MVC INTEGRATION

```
@Bean
public Properties getHibernateProperties() {
   Properties properties = new Properties();
   properties.put("hibernate.dialect", hibernateDialect);
   properties.put("hibernate.show_sql", hibernateShowSql);
   properties.put("hibernate.hbm2ddl.auto", hibernateHbm2ddlAuto);
   return properties;
}

@Bean()
public DataSource getDataSource() {
   BasicDataSource dataSource = new BasicDataSource();
   dataSource.setDriverClassName(driverClassName);
   dataSource.setUrl(url);
   dataSource.setUsername(username);
   dataSource.setPassword(password);
   return dataSource;
}
```

HIBERNATE AND SPRING MVC INTEGRATION

```
@Bean
public LocalSessionFactoryBean getSessionFactory() {
   LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
   sessionFactory.setDataSource(getDataSource());
   sessionFactory.setHibernateProperties(getHibernateProperties());
   sessionFactory.setPackagesToScan(new String[]{"teach.blog.model"});
   return sessionFactory;
}

@Bean
@Autowired
public HibernateTransactionManager transactionManager(SessionFactory sessionFactory) {
   HibernateTransactionManager htm = new HibernateTransactionManager();
   htm.setSessionFactory(sessionFactory);
   return htm;
}
```

SPRING SECURITY

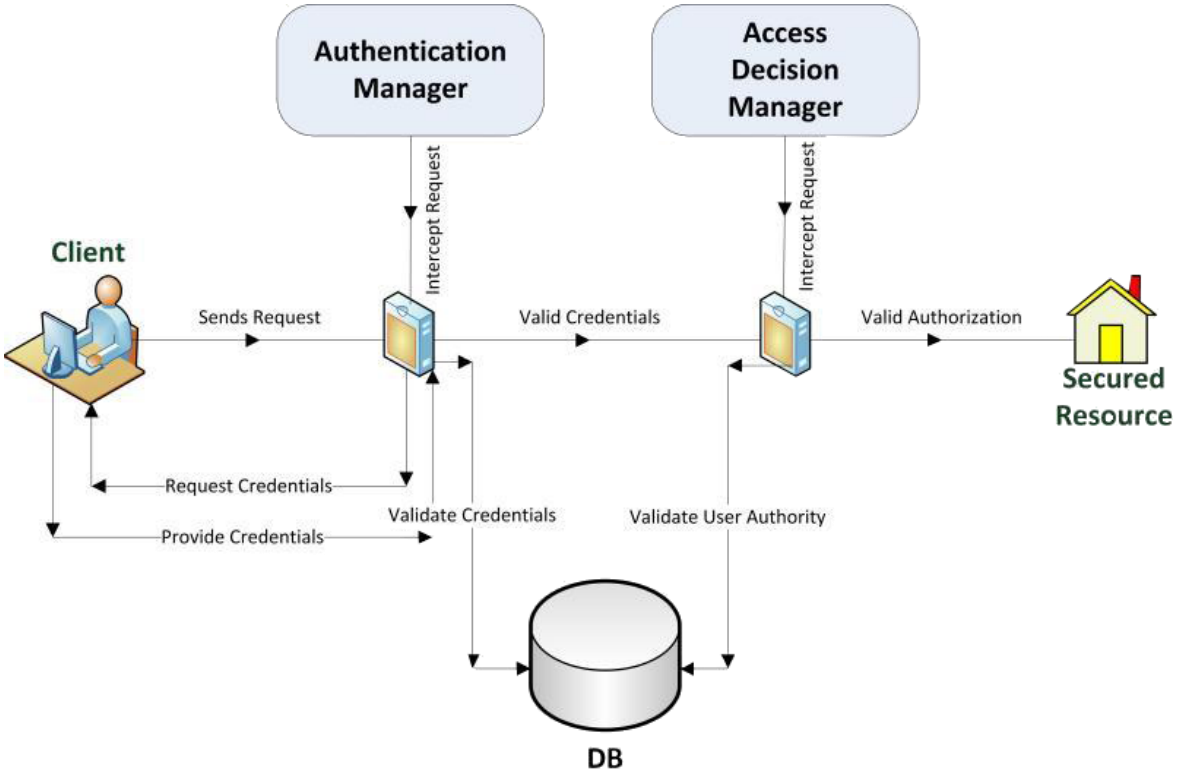
- A powerful and highly customizable authentication and access-control framework
- de-facto standard for securing Spring-based applications
- build on top of Spring Framework

SPRING SECURITY

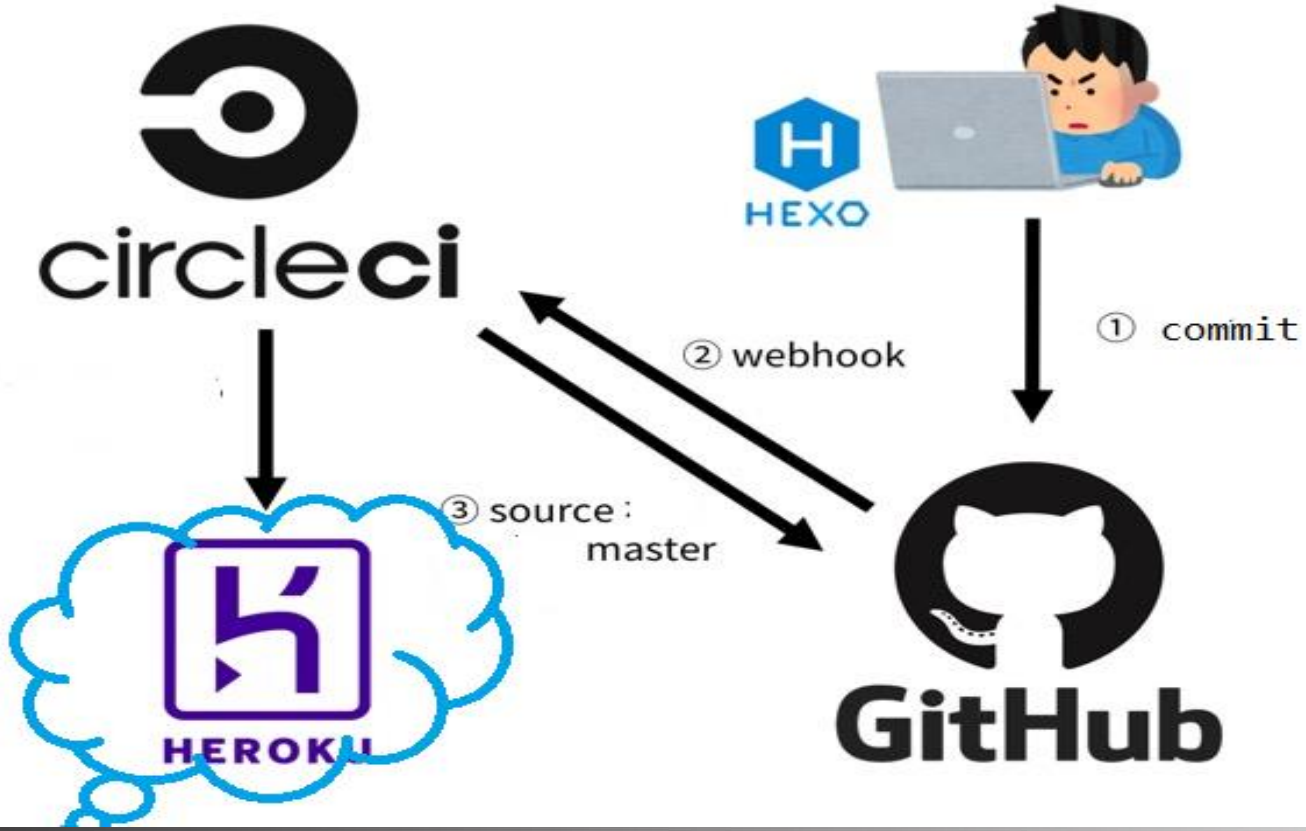
Specifically, Spring Security currently supports authentication integration with all of these technologies:

- HTTP BASIC authentication headers (an IETF RFC-based standard)
- HTTP Digest authentication headers (an IETF RFC-based standard)
- HTTP X.509 client certificate exchange (an IETF RFC-based standard)
- LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
- Form-based authentication (for simple user interface needs)
- OpenID authentication
- Authentication based on pre-established request headers (such as Computer Associates Siteminder)
- JA-SIG Central Authentication Service (otherwise known as CAS, which is a popular open source single sign-on system)
- Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker (a Spring remoting protocol}

SPRING SECURITY



HEROKU DEPLOYMENT: LIVE CLOUD



For **Enquiry**

support@trainingbasket.in

9015-887-887

SUBSCRIBE

TRAINING BASKET

QUALIFY FOR TOMORROW

THANK YOU

Visit : www.trainingbasket.in

VISIT OUR
PLACEMENT PORTAL

www.trainingbasket.in

For Reviews & Video Testimonials of our
Recently Placed Students

TRAINING BASKET
QUALIFY FOR TOMORROW

Thank You

For more information please visit our website
www.online.trainingbasket.in