Basics

21 March 2022 16:54

Java: It is a software platform for building (using JDK) *cross-platform* application and for *safely executing* (using JRE) binaries of such applications on an operating system for which it is supported. The implementation of Java platform includes support for

1. **Java Virtual Machine** (JVM) - Each Java *class* file contains the binary representation of a single Java type with matching name which includes the *meta-data* (compiler readable declaration) of that type along with *bytecodes* (machine neutral instructions) for its implemented method. The JVM manages execution of a Java application on a platform by
   (a) Loading class files into the memory when they are required by the executing application
   (b) Translating bytecodes into their equivalent native machine instructions with safety checks at the time of their execution.
2. **Java Runtime Library** - It is a set of packages containing class files with types which enable a Java application to consume services offered by the following in a portable manner:
   (a) **Runtime** which includes support for data-types, reflection and native-methods.
   (b) **Platform** which includes support for multi-threading, file i/o and network sockets.
3. **Java Programming Language** - It is a high-level programming language designed specifically for coding applications which can be executed by the JVM. It has following characteristics
   (a) It offers C++ like but more consistent syntax based on a type-system with eight *primitive value* types and support for implementing *user-defined reference* types.
   (b) It is primarily object oriented based on *common root single class* inheritance model with added support for generic programming and functional programming.

Java Primitive Types

Data Type	Allowed Value	Wrapper Class	Format
boolean	true|false	Boolean	%b
char	single character (Unicode)	Character	%c
byte	8-bit signed integer	Byte	%d
short	16-bit signed integer	Short	%d
int	32-bit signed integer	Integer	%d
long	64-bit signed integer	Long	%d
float	32-bit single precision real	Float	%f
double	64-bit double precision real	Double	%f

Inheritance

06 May 2022 10:57

Package: It is a named group of related user-defined types organized together to avoid collisions between the names of these types and other types not belonging to their group and to limit access to these types from other types not belonging to their group. A type **T** belonging to a package **p** has following characteristics:

1. It is accessible outside of **p** through its fully qualified name of **p.T** only if it is defined with public modifier (in T.java)
2. Its binary representation is loaded by default from path **p/T.class**

Object Identity: Two objects are considered to be identical if they refer to the same instance in the memory. Whether object **x** is identical to object **y** is determined from expression **x == y**.

Object Equality: Two object are considered to be equal if they refer to two instances of same class with matching states in the memory. Whether object **x** is equal to object **y** is determined from expression **x.hashCode() == y.hashCode() && x.equals(y)**

Visibility of a member outside of its declaring class

Access Modifier	Current Package	External Package
private	none	none
<default>	all	none
protected	all	sub-class
public	all	all

Abstract Class	Interface
It is a reference type which does not support instantiation but can define instance fields	It is a reference type which does not support instantiation and cannot define instance fields
It can define unimplemented	Its instance methods are implicitly

instance methods using *abstract* modifier	abstract and implemented methods must be defined with *default* modifier
It can include public as well as non-public members	It can only include public members
It can define constructor which is called by its sub-classes	It cannot define a constructor
It can define final as well as non-final static fields	It can only define final static fields
It can extend exactly one other class	It can extend multiple other interfaces
A non-abstract class can inherit from a single abstract class and it must override all the abstract methods of that class	A non-abstract class can inherit from multiple interfaces and it must implement all of their abstract methods
Generally defined for segregating common state associated behavior supported by different types of objects	Generally defined for segregating common state independent behavior supported by different types of objects

Multiple Inheritance in Java: JVM's type system does not allow a class to inherit from multiple other classes because an instance of such a class will require multiple sub-objects (to store values of instance fields defined by corresponding super class) which complicates its runtime type access required for *safe-casting* and *reflection*. Since interface cannot define instance fields it does require a sub-object within an instance of its inheriting class and as such Java allows a class to inherit from multiple interfaces.

Generics

11 May 2022 16:28

Java Generics: It is the syntactical support offered by Java for implementing common type-safe code patterns which can be reused with different reference types. It enables the Java compiler to identify matching reference types in a declaration and to perform type conversions which are otherwise explicit. A generic declaration contains at least one type parameter with following characteristics

1. It can be substituted with any reference type at compile-time by default since it is treated as java.lang.Object by the compiler but then only members of java.lang.Object can be applied to it by default (erasure).
2. It can be bounded by a reference type using *extends* statement so that it also supports members of that reference type but then it can only be substituted with a type which supports implicit conversion to (inherits from) that reference type.

Wild-card Substitution: A generic type **G** is *invariant* over its type-parameter **T** which means **G<U>** cannot be substituted by **G<V>** irrespective of relationship between **U** and **V**. However following variant forms of **G<T>** can be used in a declaration as

1. **G<? extends U>** which can be substituted by G<V> where V supports implicit conversion to U but then members of G in which its type-parameter T appears as an argument type cannot be applied to this form of G.
2. **G<? super U>** which can be substituted by G<V> where V supports implicit conversion from U but then members of G in which its type-parameter T appears as a return type cannot be applied to this form of G.

Generic Collection: It is an object of a generic class which supports grouping of elements of a given type and provides operations to access those elements in a type-safe manner. The *java.util* package includes *Collection*<E> interface which extends *java.lang.Iterable*<E> interface to define standard methods for adding/removing elements to/from a generic collection. This interface has following sub-interfaces in the same package

1. ***List*<E>** - it specifies support for implementing a sequential collection whose elements can be retrieved through their indexes. It is implemented by

ArrayList<E> [1 ref/entry and O(1) indexing] and LinkedList<E> [3 ref/entry and O(n) indexing] classes of java.util package .

2. ***Set*<E>** - it specifies support for implementing a collection which can only contain non-duplicate elements whose uniqueness can be determined from their behavior. It is implemented by HashSet<E> [1 ref/entry and O(n) operations] and TreeSet<E> [3 ref/entry and O(log n) operations] classes of java.util package.

   A Set<E> is commonly implemented to contain an implementation of Map<K, V> interface which groups pairs of unique keys and their associated values. This interface is implemented by HashMap<K, V> [2 ref/entry and O(n) operations] and TreeMap<K, V> [4 ref/entry and O(log n) operation] classes of java.util package.

Runtime

13 May 2022 09:28

Callback Function: It is a mechanism of passing a function of a specific type to some code in order to allow this code to indirectly call that function. Java provides syntactical support for implementing callbacks through

1. **Method Reference** - It is a reference type which can be bound to a method with a particular list of parameter types and a particular return type. It is produced by applying double-colon (::) operator to a class or its object along with the name of a static or an instance method defined by that class or from implementation of an anonymous method (defined by the compiler in the current class) known *as lambda expression*.
2. **Functional Interface** - It is an interface containing exactly one abstract method defined to be used as a type for a method reference which is compatible with that abstract method. It is automatically implemented at runtime to invoke the method referred by its identifier.

Functional Programming: It is *a declarative style* of processing data in which this data is passed through a *chain of calls* to functions which do not have side effects and accept other functions as their arguments when required. The *java.util.stream* package provides support for functional programming using

1. **Operation Pipeline** - A sequence of elements of a particular type known as a stream is processed by passing it through a chain of methods each applied to the stream returned by the previous method in that chain.
2. **Lazy Evaluation** - Each method in the operation pipe line performs its own *internal iteration* on the elements of its target stream and this iteration is executed only when the last method (terminal operation which does not return a stream) in the pipeline is called.

Reflection: It is a mechanism which enables a program to examine the structure of its own data at runtime. In Java the meta-data (description)

of a type **T** is provided by an instance of **java.lang.Class<T>** whose reference can be obtained at compile-time using expression **T.class** and at runtime from

1. An object obj of a reference type

   Class<?> c = obj.getClass();

   Note: c will refer to java.lang.Class of reference type which was used for constructing the instance referred by obj
2. The fully-qualified name N of the type

   Class<?> c = Class.forName(N);

   Note: The class with specified name N will be loaded using the built-in class-loader of the JVM. The built-in class loader loads the binary representation of type p.T from path p/T.class by searching for this path in each location (directory or archive) specified by the java.class.path property of the JVM which defaults to the current directory and implicitly includes Java runtime library.

Annotation: It is a *meta-data representing type* which can be applied as a custom modifier to a *declaration target* such as a field, a method, a type (class or interface) and a package. It is defined as an interface (with @ prefix) along with one of the following *retention policies*.

1. **SOURCE** - The annotation appears in the source code of its declaration target but is discarded by the compiler. Such an annotation can only be examined at compile-time using its annotation provider.
2. **CLASS** - The annotation is inserted by the compiler into the binary representation of its declaration target within the class file but is discarded by the runtime. Such an annotation can only be examined at compile-time using its annotation provider.
3. **RUNTIME** - The annotation is loaded into the memory by the class-loader along with the meta-data of its declaration target. Such an annotation can be examined at runtime using reflection.

Native Method: It is a method defined (with native modifier) in a Java class which on invocation executes the *machine specific instructions* of a *global function* exported by a *platform specific dynamically linkable library* (shared-object on Linux). The function called by a native method

is generally implemented in C/C++ and it interacts with the JVM using its *Java Native Interface* (exposed through JNIEnv type which addresses a struct containing pointers to C functions published by the JVM).

1. **Purpose**
   (a) to increase the performance of a Java program by skipping the excessive verification done by the JVM.
   (b) to reuse non-Java code available on the platform including system services which are not exposed by the Java runtime library.
2. **Disadvantages**
   (a) It can compromise the execution safety ensured by the JVM to protect a program from crashing at runtime.
   (b) It can limit the portability of a program to different platforms supported by Java.

Platform

18 May 2022 09:33

Concurrency: It is programming support offered by the platform for simultaneously executing multiple blocks of code. It is used for

1. **Asynchrony** - in which the caller of a procedure can resume execution before that procedure return in order to increase the responsiveness of the running program.
2. **Parallelism** - in which different iterations of a long running loop is executed on separate processors (or cores) available on the hardware in order to increase performance of the running program.

Thread: It is the basic unit of concurrency within an executing program and it has following characteristics.

1. A thread refers to a function which it executes concurrently with other threads including the main thread which is started automatically when the program is executed.
2. A thread shares values of variables with other threads of the program except for the local variables of the function it refers to.

Object Serialization: It is a mechanism of converting the entire state of an object (including the state of each object it references) into a stream (series) of bytes from which it can be deserialized (reconstructed). It is commonly used for

1. **Persistence** - in which the object is transferred to so a storage medium.
2. **Marshalling** - in which the object is transported from across process boundary.

Object Serialization in Java: The *java.io* package includes ObjectOutputStream/ObjectInputStream class to support serialization/deserialization of an object with following characteristics

1. Its class must inherit from *java.io.Serializable* interface and should define *static final long serialVersionUID* field with a value representing the set of its non-transient instance fields.

2. It must not refer to any non-serializable object through a field which is declared without transient modifier by its class.

Communication

20 May 2022 09:28

Socket: It is a *common logical interface* offered by the platform for transporting data between separate processes using different *communication schemes* supported by the operating system. Each socket is bound to a unique *endpoint* identifying its communication type specific *address* and provides operations to *send/receive* data to/from other such endpoints. A *connection oriented* socket which supports *streaming* of data between endpoints is used for implementing

(A) Server - It provides a service for processing or sharing some data over a *well-known endpoint address* using following steps

1. Open a listener socket and bind it to a particular endpoint address.
2. Use the above listener socket to accept the socket connected to the endpoint which has requested this connection.
3. Use the above accepted socket to exchange data with its connected endpoint.
4. Close the above accepted socket and go to step 2.

(B) Client - It consumes the service provided by the server from a *random endpoint address* using following steps

1. Open a socket and connect it to the endpoint address of the server.
2. Use the above socket to exchange data with its connected endpoint
3. Close the above socket.

Network Communication: It is a mechanism for supporting transfer of data between different processes running on separate machines linked to each other through some type of networking hardware. The platform provides sockets built on top of the network communication layer (protocol stack) of the operating system which commonly includes implementations for

1. **Internet Protocol** (IP v4/6) - It is *a network protocol* which provides a network linkage type (WiFi, 4G) independent scheme for identifying each *host* (communicating machine) on the network using a unique (32/128-bit) integer known as its *IP address* and for supporting transfer of data between such hosts using structured

blocks (containing a maximum of 65535 bytes each) known as *IP packets* each including the IP addresses of source and destination along with other headers.

2. **Transmission Control Protocol** (TCP) - It is a *transport protocol* which provides an IP based scheme for identifying each *peer* (communicating process) running on a host using a unique 16-bit integer called *port address* and for supporting a connection oriented mechanism for reliable streaming of data between two peers using IP packets which include synchronization and acknowledgement headers.

Distributed Computing: Dividing the implementation of a software into multiple parts so that each part executes within its own process on a separate machine and interacts with other parts using network communication is called distributed computing. It is commonly used for

1. Centralization of resources over the network by running some common code on a single machine which otherwise is supposed to execute on multiple machines. Large scale centralization of resources over the internet is called *cloud computing*.
2. Decentralization of resources over the network by running different parts of some code on multiple machines which otherwise is supposed to execute on a single machine. Large scale decentralization of resources over the internet is called *grid computing*.

Hyper-Text Transport Protocol (HTTP)**:** It is a standard TCP/IP based scheme (application protocol) for sharing a resource over the network using its *uniform resource identifier* (URI) containing the path of that resource along with the endpoint of its provider (server). HTTP offers a simple *request-response* based model for exchanging data between endpoints in a *stateless* manner (one request per connection) which involves

1. The client sends a request based on a URI in following form
   <VERB> <path> HTTP/<version>\r\n
   <Header-Name>: <value>\r\n
   …

\r\n
<body-content>

Standard HTTP Request verbs
POST - *Create* a new resource from the content of the request body.
GET - *Read* the resource identified in the requested path and send its content in the response body.
PUT - *Update* the resource identified in the requested path from the content sent in the request body.
DELETE - *Delete* the resource identified in the requested path.

2. The server sends the response based on the request in following form
   HTTP/<version> <status-code> <status-message>\r\n
   <Header-Name>: <value>\r\n
   …
   \r\n
   <body-content>

   Standard HTTP Response Status Codes
   2nn - The request was handled successfully and the content of the requested resource is in the response body.
   3nn - Redirect the request to the URI in the status message.
   4nn - The request has an error such as invalid path or verb or body content.
   5nn - The request cannot be handled because of some internal error.

Database

23 May 2022 09:30

Relational Data: It is a form of persistent data, logically structured into *tables* each created with its own set of *columns* which can be referenced by columns of another table to support *parent-child* relationship between those tables. The data is stored in the *rows* of the table whose columns are defined to specify

1. **Schema** - to indicate the type of the values that can be stored in the rows of that table.
2. **Constraints** - to indicate the restrictions on the values that can be stored in the rows of that table.

Transactional Data: It is a form of persistent data whose different parts can be updated within a single unit of work known as a *transaction* with support for a *commit* operation to persists the new state of data and a *rollback* operation to restore the data to its original state. A well-behaved transaction has following ACID characteristics

1. **Atomic** - A transaction should only perform an update whose effect can be cancelled by the rollback operation.
2. **Consistent** - A transaction should rollback as soon as any of its update violates a constraint on the target data.
3. **Isolated** - A transaction should not allow any data it accesses to be updated by an operation that does not belong to its scope.
4. **Durable** - A transaction should always end either with a commit or a rollback operation.

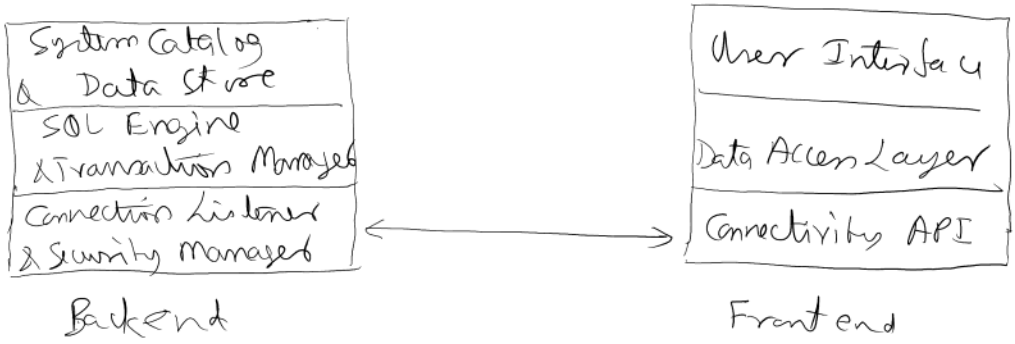
Relational Database System: It is a software that manages persistence of relational and transactional data with support for

1. Consuming this data using an API based on (a fourth generation declarative) *structured query language* (SQL).
2. Sharing this data in a secure manner based on *authentication* (confirming identity of a user) and *authorization* (granting permission to a user) schemes.

Database Application: It is software that automates a certain business

process by performing a set of transactional operations known as the *business logic* on some persistent data. Its implementation is generally divided to execute as at least two separate but communicating processes with following responsibilities

1. **Backend** - It manages persistence of transactional data for the application and it can optionally implement the business logic for that application to increase its *maintainability*. It is commonly supported using a relational database server installed on a central machine over the network.
2. **Frontend** - It provides a user-interface for the application and it can optionally implement the business logic for that application to increase its *scalability*. It is commonly supported using a client program which is installed on each of its user machine over the network.



Middle Tier: It is a separate process which hosts the implementation of a particular business logic for an application and is inserted between the frontend (presentation tier) and the backend (data tier) of that application to improve its maintainability as well as its scalability. It is commonly designed using *service oriented architecture* in which its implementation is divided into (small) sets of operations known as (micro)services each with following characteristics

1. A service is an *autonomous* software and as such it can be deployed

(and maintained) independently of other services.
2. A service only shares the *description* of its operations and the *schema* of the data exchanged by those operations.
3. A service has an *explicit boundary* and as such it can only be consumed by exchanging messages with the endpoint published by its host process which can execute on a separate machine (or in a separate container).

Java Database Connectivity (JDBC)**:** It provides a standard API (through *java.sql* package) for consuming data managed by a relational database system. The implementation of JDBC interfaces for a particular database system is called its JDBC driver and in includes support for following types of objects

1. **Connection** - it opens a communication session with the database system using its URL.
2. **Statement** - it sends SQL commands to the database system using the connection object.
3. **ResultSet** - it fetches the rows resulting from execution of a query command sent using the statement object.

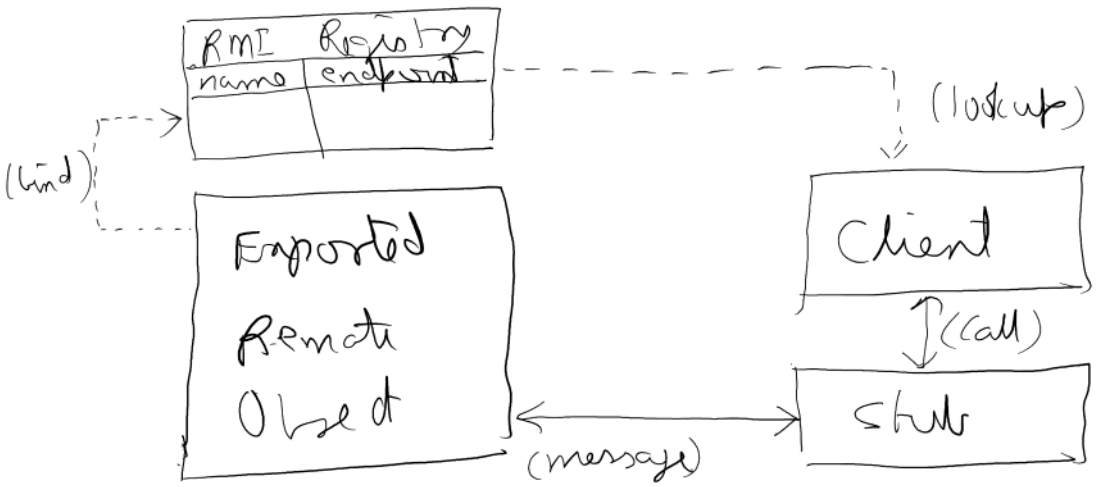
Java Persistence API (JPA)**:** It specifies standard support (through javax.persistence package) for accessing relational data using instances of data-context unaware Java classes. The implementation of this API (such as EclipseLink and Hibernate) is called the JPA provider and it includes support for

1. **Entity** - It is a *serializable plain old Java object* (POJO) containing at least one *identity* field which is mapped (in META-INF/orm.xml or through annotations) along with its other basic fields to columns of a database table mapped to its class.
2. **Entity Manager** - It loads entities specified using *Java persistence query language* (JPQL) from their mapped tables and handles persistence of these entities into their mapped tables in a transactional manner.
3. **Entity Manager Factory** - It creates an instance of entity manager from its *persistence unit* (configured in META-INF/persistence.xml) which includes information required by the JPA provider to connect

to the database and for reading/writing data within entities from/to tables of that database.

Remote Method Invocation (RMI)**:** It is programming support offered by Java which enables invocation of methods exposed by an object activated within one JVM from Java code running in another JVM across the network. It includes support for

1. **Remote Object** - It is an object whose class implements at least one *remote interface* (which extends java.rmi.Remote). When a remote object is exported (through java.rmi.server.UnicastRemoteObject) it is bound to a TCP/IP endpoint so that it can receive method invocation requests from remote endpoints.
2. **Remote Stub** - It is a *proxy* of an exported remote object in its client's JVM. The (runtime generated) class of a stub implements all the remote interfaces of its target remote object to transport invocation requests to the corresponding methods of that remote object.
3. **RMI Registry** - It is a TCP/IP *based name service* for discovering remote objects exported by its host. The server process *binds* the endpoint information of its exported remote object to a well-known name in the local RMI registry and the client process *looks-up* for a name in the remote RMI registry to acquire the stub of the remote object bound to that name.


RMI Registry
name
endpoint
(bind)
(lookup)
Exported
Remote
Object
Client
(call)
stub
(message)

Web

25 May 2022 17:44

Web Application: It is a software executed by a web-server to publish *dynamic content* over its HTTP endpoint so that this content can be presented to the user through a web-browser.

Classic Web Application	**Modern Web Application**
The data is acquired and output is rendered by the server-side code and this output is transported to the browser for presentation.	The data is acquired by the server side code and it transported to the browser where it is rendered by the client-side code for presentation.
Input is received by the browser and transported to the server where it is handled by the server side code.	Input is received by the browser where it is directly handled by client side code which calls server-side code if required.
Implementation requires familiarity with server-side frameworks such as Spring (Java) and ASP.NET (C#).	Implementation requires familiarity with server-side frameworks as well as client side frameworks such as Angular and React.
Application is more secure and is independent of type/version of the browser.	Application is less secure and may depend upon the type/version of the browser.
UI is less responsive because every interaction requires its re-rendering on the server.	UI is more responsive because interactions do not require any re-rendering.
It uses patterns suitable for large applications with multiple web-pages	It uses patterns suitable form a *single page application* (SPA)

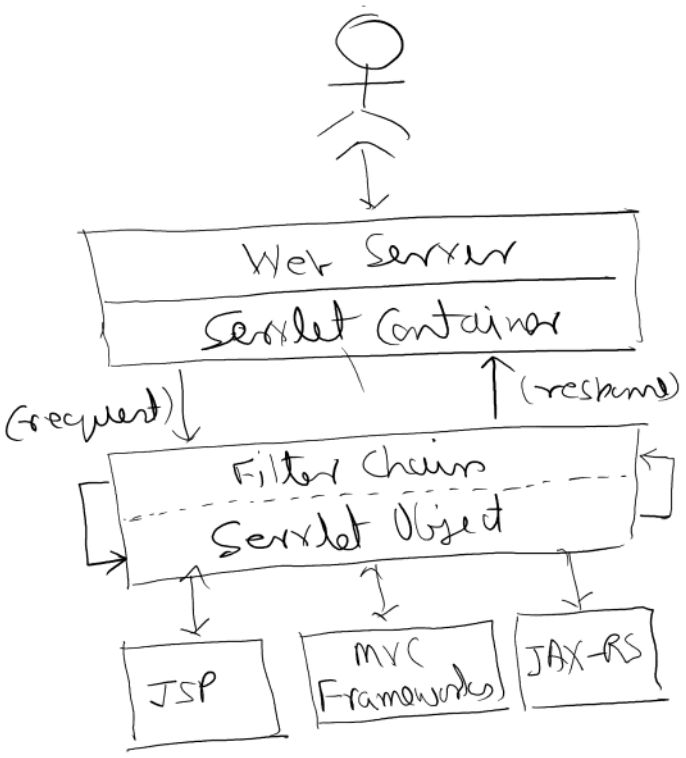
REST (REpresentational State Transfer)**:** It is an *architectural style* applied for enabling the *exchange* (transfer) of the *content* (state) of

some *resource* between *HTTP endpoints* using a mutually agreed upon *format* (representational form). A RESTful service (API) uses REST to publish methods of a *stateless* object so that each such method

1. Performs a *create, read, update* or *delete* operation indicated by the requested verb (POST, GET, PUT or DELETE) on a data-object identified within the requested path.
2. *Produces* (in response) or *Consumes* (from request) a data-object using a standard *media-format* (such as application/json) and returns a standard HTTP status code to indicate the success or the failure of its operation.

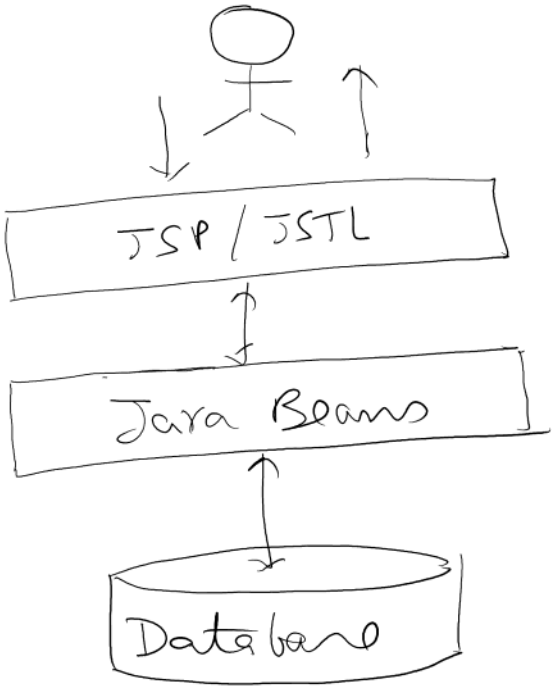
Java Servlet API: It specifies standard support through (*jakarta.servlet* package) for implementing a Java server-side object known as a *servlet* which generates a response to a request of a particular type received by the web-server. A web-server (such as TomCat, GlassFish, WebLogic, WebSphere) which supports deployment of Java web-applications includes an environment called a *servlet container* which provides the implementation of the Servlet API and handles each request received by the server using following steps:

1. Get the servlet object mapped (in WEB-INF/web.xml or through annotation) to the URL pattern matching with the requested path initializing a new object from the corresponding servlet class (which implements jakarta.servlet.Servlet interface) if required.
2. Invoke the *service* method of the above servlet object passing it the current *request* and *response* objects as arguments and when this method returns pass the content of the response object to the web-server so that it is transported back to the requesting client.



JSP (Java Server Pages)**:** It provides servlet based support for enabling a web-application to publish dynamic content by combining client-side markup (HTML) with server-side elements (expressions and tags) in a single web-page. The URL pattern of a JSP page (*.jsp) is mapped to the built-in JSP servlet which services the request for each such page by executing the server-side code in that page. The programming model of JSP includes support for:

1. **Java-Bean** - It is a serializable object whose class supports a parameter-less constructor and exposes instance properties using methods which follow standard get-set naming convention.
2. **Tag-Extension** - It is an object used specifically by a JSP page for handling a custom (user-defined) server-side markup element (tag) which is mapped to its class (which implements jakarta.servlet.jsp.tagext.SimpleTag interface) along with other information in a URI identified XML document called the *tag-library descriptor*. The JSP *standard tag library* (JSTL) provides tag extensions commonly required for composing JSP pages.



JAX-RS (Java API for Restful Services)**:** It specifies a standard support (through *jakarta.ws.rs* package) for enabling a web-application to publish RESTful services. It includes

1. **Servlet** to handle a request for the invocation of a method exposed by a RESTful service identified by the verb and path of that request.
2. **Annotations** to identify the path of the RESTful service and each of its methods along with the verb and the media-type of data-objects produced or consumed by that method.