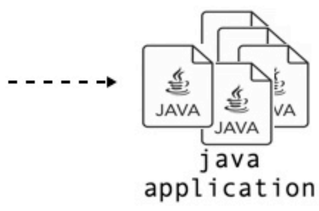
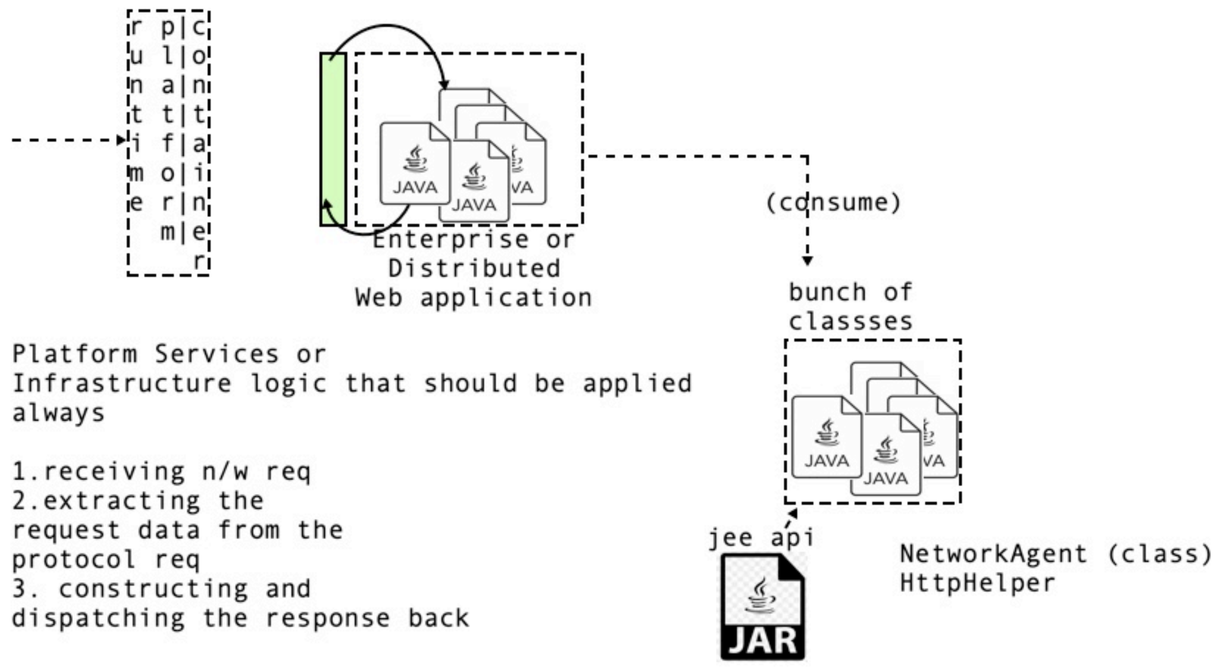
Standalone Application



In a typical standalone java applications, there is always an single-entry point into the application (like a main method) in which the programmer pretty much instantiate the objects of the classes within the application and invoke the methods to perform the operations required.

Enterprise/Distributed Application

Infrastructure logic or platform logic



Platform Services or Infrastructure logic that should be applied always

1.receiving n/w req
2.extracting the request data from the protocol req
3. constructing and dispatching the response back

Many of the times like above the runtime environment or framework has to be executed first-in place in performing the mandatory action/activity or platform functionality and there after has to pass the control to our application components allowing us to perform further operation.

Which means the framework or runtime environment should be able to instantiate the components of our application and invoke the methods of our classes by passing the input data that it has computed first-in place. our application components by using the data should perform operation and compute the output data and return back to the framework or runtime environment which takes care perform further operation on our data in building the response and dispatching.

How does the runtime environments / containers knows the information about our application components?
The developers of the application has to pass information pertaining to the components of our application as an input to the framework or runtime environment, so that it would be able instantiate and callback our application component. There are 2 ways we can pass the information about our application components to the underlying container or platform or runtime environment or framework

1. Metadata approach
2. programmatic api approach

1. Metadata approach
The developer has to write configuration file like an XML describing the entire information about the component/class, its methods, the parameters these methods take and the returnType they return should declared aspart of an XML-Based configuration file. This should be return based on the container/runtime environment understandable or standard format.
Then the underlying containers or platforms or frameworks would read this configuration metadata in understanding our application components in calling our methods with necessary inputs we defined and will collect the output in providing the platform services.

There are lot of challenges in writing the configuration describing the details of our application components:
1. Writing the configuration information describing the classes, methods, parameters and returnTypes is going to be huge and takes lot of time in writing this configuration information, that kills the development time and productivity

2. memorizing these XML-based configurations and writing them is going to be very tedious job

3. There is no intermediate compilation process like java in verifying the configuration that is produced by the developer is valid or not. The only way to verify such configuration is over runtime, which results in repeated deployments/re-runs of the application in deriving the configuration errors that wastes lot of development time in building the application

4. even though the syntactically the configuration that is produced is correct, it would be hard to identify the mis-configuration issues and debug

5. always any change in the component and their methods needs to be quickly reflected in the configuration which is always difficult.

To overcome the challenges in working with metadata approach, the apis, containers or runtime env, frameworks has introduced programming api approach.

---

2. Programmatic api approach
Instead of defining the details of our application components in metadata configuration file, The frameworks or the api would provide standard interfaces declared with bunch of pre-defined methods declared with parameters/returnTypes. Per each functionality that we want to use from the underlying framework or runtime environment we need to implement an appropriate interface provided by the Framework/Api.

By implementing the interface we are defining #2 things
1. What type of component it is
2. Contractual methods implemented

since the framework/api knows what methods are there aspart of our components, they dont need metadata information describing the methods, parameters and returnTypes. But to let the framework/runtime env know the information about the classes we still need to write configuration information registering the components to the runtime env/framework.

By using programmatic api approach, it looks like most of the challenges with metadata-driven approach has been resolved ending up with very little or no configuration information. But still this approach has lot of problems:

problems:
1. There are numerous number of libraries/frameworks and apis are available, all of these provides different interfaces declared with methods in enforcing us to implement inorder to consume the functionality. For which functionality to acomoplish what interface needs to be implemented should be memorized that makes difficult build the application

2. since the classes of our application are enforced to be written implementing the interfaces provided by the api/framework, we dont have choice of writing the classes withour own methodNames depicting the functionality we are performing. Looks like the methods represents the technical names rather than the business functionality we are implementing due to this the readability of the code is poor.

3. The components of our application are tighly coupled with the underlying api/framework we are using, so if we want to switch from one api/framework to another we need to rewrite the components with different set of interfaces provided by others

4. The developers are left with no choice of choosing their own parameterTypes and returntypes for the methods of the classes while implementing from the api/framework. due to which there is lack of flexibility in choosing the signature of the methods

From the above we can understand the programmatic api/interface-driven programming lacks flexibility in developing the application and makes the programmer find complex in memorizing and building the solutions.
To overcome this problem the annotation-driven programmatic approach has been introduced!