Phase 1 : Java

Phase 2 : Spring Framework

Phase 3 : DevOps Tools

Java - 8 : IoT

Lambda Expression

base64 API

Streams

Functional Interface

default

method reference

Optional

DateTime API

Concurrent API enhanced

Nashorn Engine (JS engine)

Functional Programming

functions as first class citizens + OOPs

Imperative style of programming

Classical/Traditional

Focus : how to perform operation

write steps on how to achieve an objective

Object mutability

Declarative style of programming

Focus : what is the result

Object immutability

SQL style

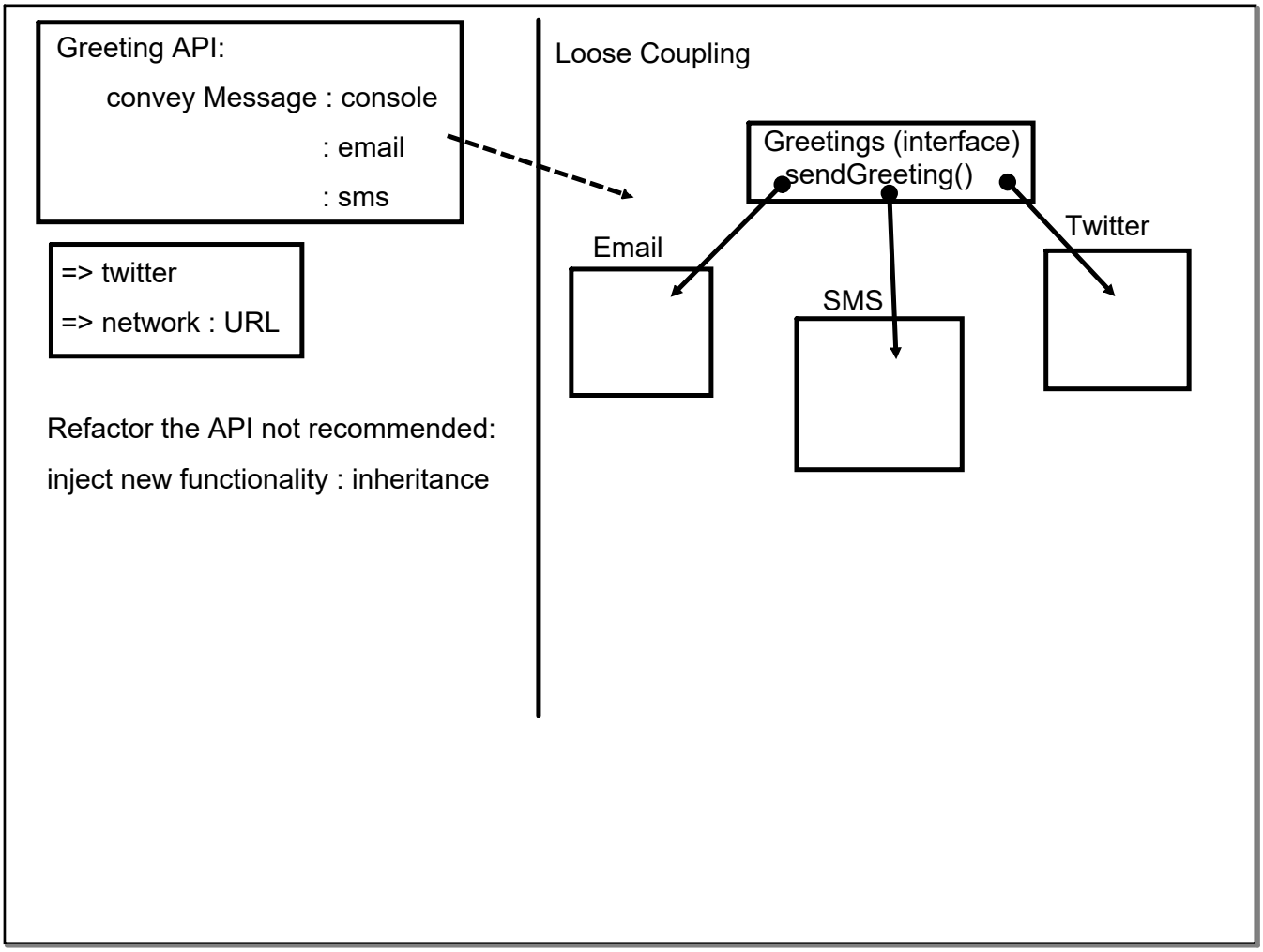
Collection of numbers

fetch the unique numbers

wf.com

com.wf

<reverse domain of org><training>


Greeting API:
convey Message : console
: email
: sms
=> twitter
=> network : URL
Refactor the API not recommended:
inject new functionality : inheritance
Loose Coupling
Greetings (interface)
sendGreeting()
Email
SMS
Twitter

Declarative:

Inject only functionality (pure function), not wrapped inside an object

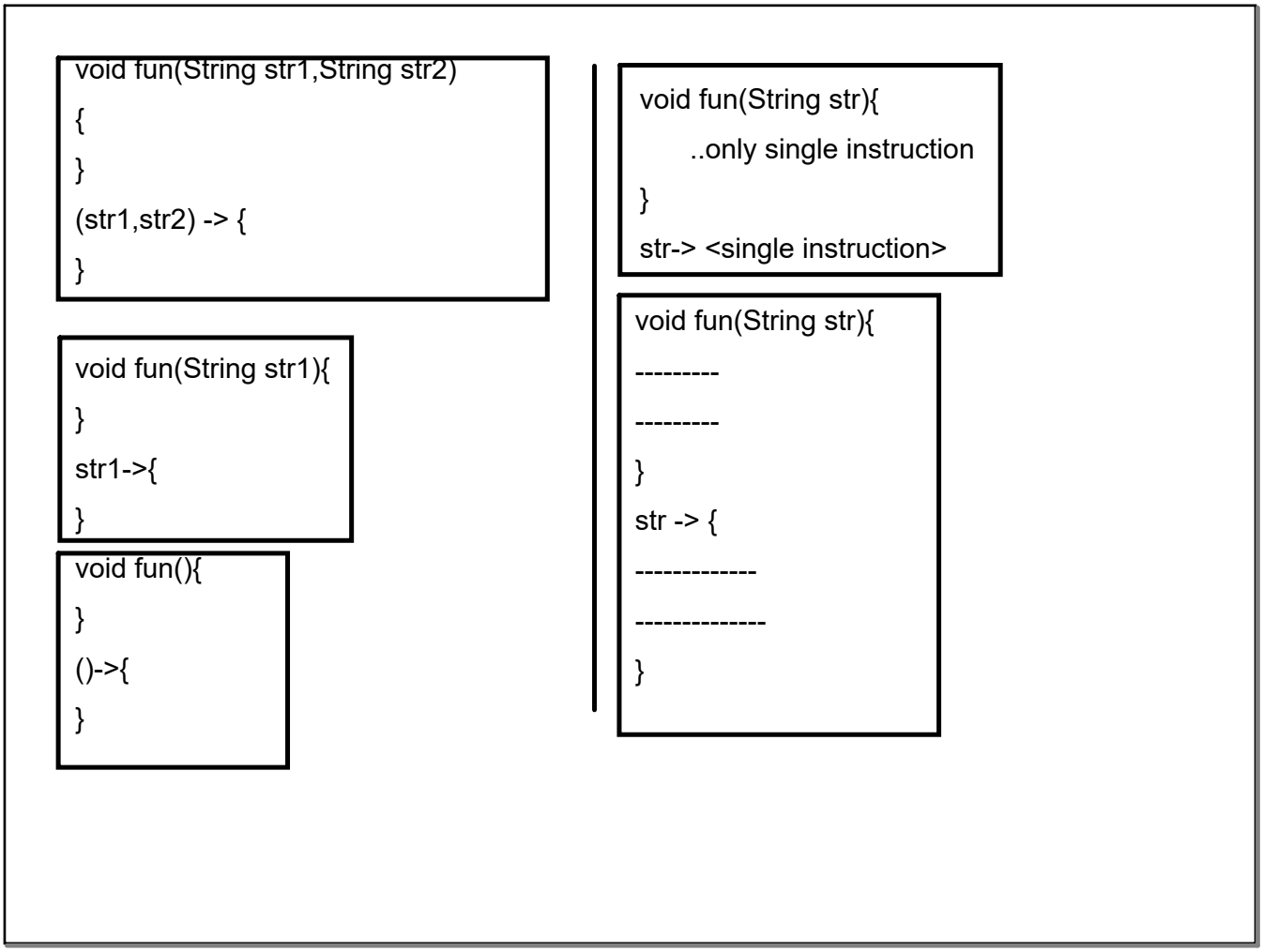
Java should expose a datatype : Function

New Datatype : would not be backward compatible

interface : Function datatype

Syntax :

1. not have any access modifier
2. Anonymous function (no name)
3. return type is not mentioned
4. no param types
5. <praram name> -> {<definition>}


void fun(String str1,String str2)
{
}
(str1,str2) -> {
}
void fun(String str1){
}
str1->{
}
void fun(){
}
()->{
}
void fun(String str){
..only single instruction
}
str-> <single instruction>
void fun(String str){

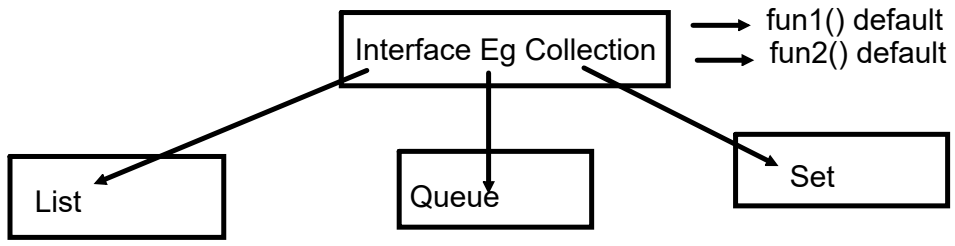
}
str -> {

}

```
int add(int a,int b){
    return a+b;
}
(a,b) -> a+b;    // single instruction return is by default associated
(a,b) -> {
    return a+b;
}
```

interface :

1. default functions : interfaces can have functions with definition



Functional Interface :

An interface containing only a single abstract method

it may have multiple default and static method

Lambdas/Method Reference can be assigned to only functional interface reference

Lambda Expression/ Method Reference signature must match with the only abstract method of FI

An reference of functional interface can refer to any method as long as its signature matches with the only abstract method (other than lambdas also)

More Practical Usage....

Streams

Specialized lib/api

Existing interface

- # Runnable
- # Comparator
- # Comparable

Lambdas with local variables

Effectively final : Local variable declared outside the Lambdas are effectively final inside the Lambda expression

Not allowed to use the same local variable name as param or inside the lambda body

Not restriction for instance variable

-> Easy to perform concurrency operations

-> immutability

A Special Library of Functional Interfaces

Common prototypes are exposed

java.util.function.*

Consumer : BiConsumer

void accept(<>) : Consume the data

Predicate : BiPredicate

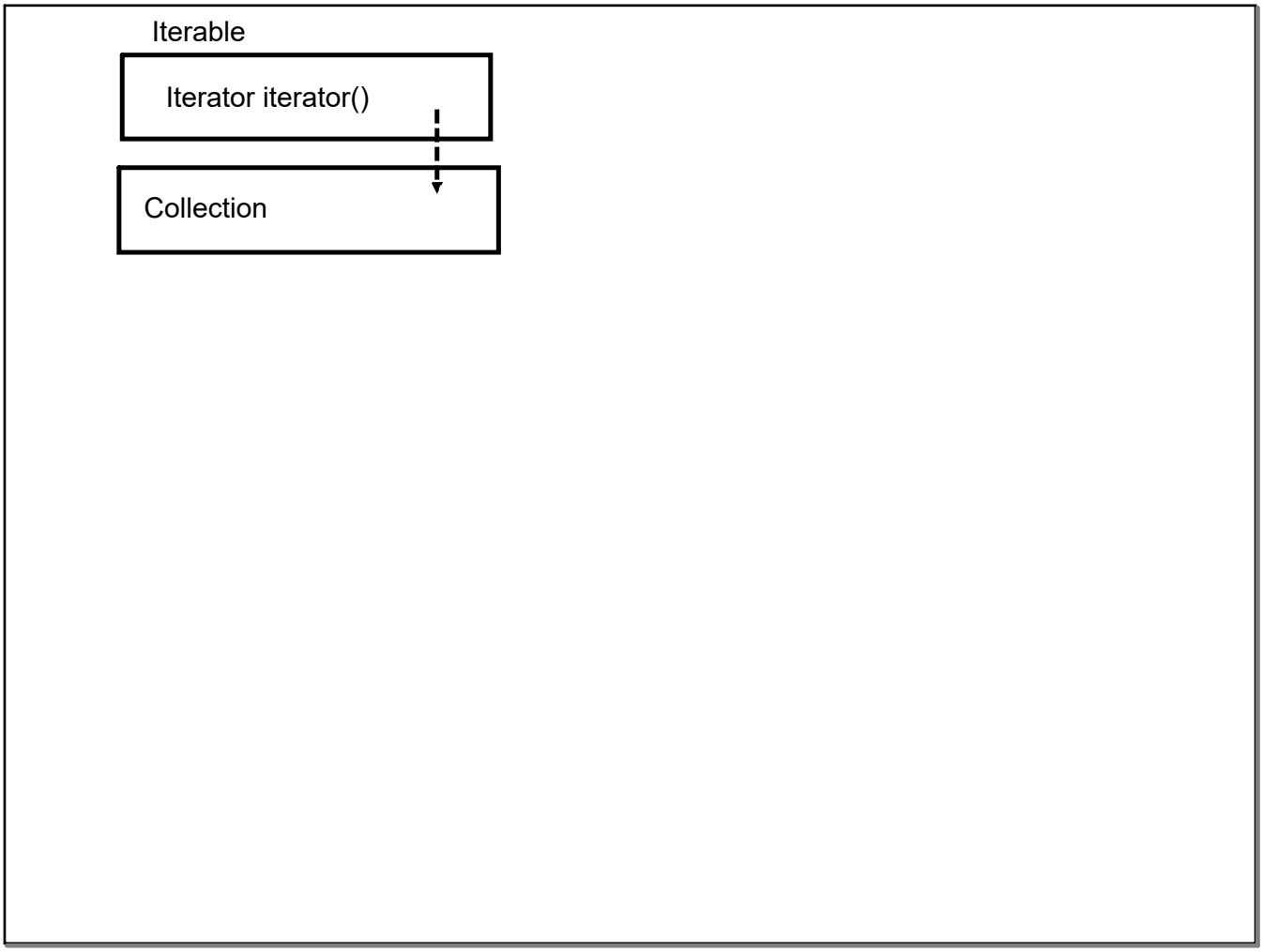
boolean test (<>) : Use data to check some condition

Function : BiFunction, UnaryOperator, BinaryOperator()

<> apply(<>) : Transformation

Supplier:

<> get() : Generate some data and return data back


Iterable
Iterator iterator()
Collection

Stream : Sequence of elements created out of collections / IO resource

Add a stream of activity

Stream : Safe and Efficient way

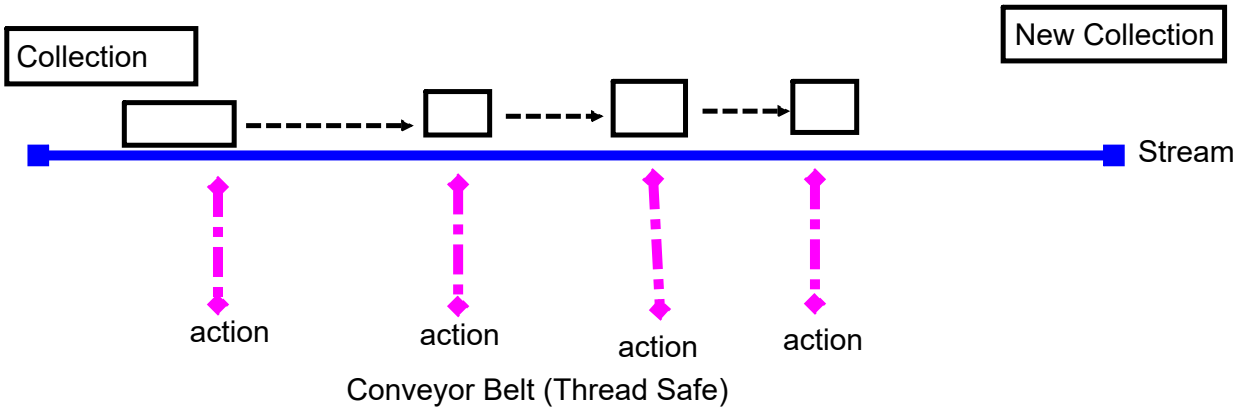
Safe :

immutability : Thread safe implementation : Concurrency

Sequential/ Parallel

Efficient :

Not a Data Structure : not going to store any data : Lazy processing model



Phase1

Phase1 : SBA1 :

Phase2 : SBA2

Phase3 : SBA3

Use case :

End-to-end : Milestone (weekly)

Team based implementation

Every Stream must have a terminal activity

1. initiate a stream
2. intermediate operation (optional)
3. Terminal operation

Stream does not initiates if no terminal activity is there

Sequential Stream

Parallel Stream : Parallel operations without having to spawn thread

1. Stream is using a mutable resource/service : Parallel is not recommended
2. inherently complex activities which consumes more time in parallel processing

1,2,3,4,5....

1,4,2

result = result + v

result = 25 + 5

result = 25 + 2

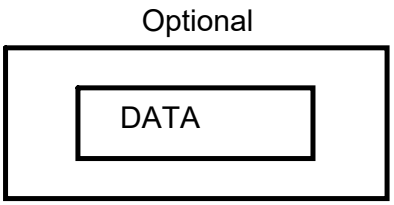
Optional

DateTime API

Servlet API

Optional : used to resolve issue related with Null Reference

1. Revert data encapsulated as Optional



unwrap, validation can take place

Traditionally :

Date

Calander

DateTimeAPI

LocalDate : only Date

LocalTime : only Time

LocalDateTime : both

Servlet API

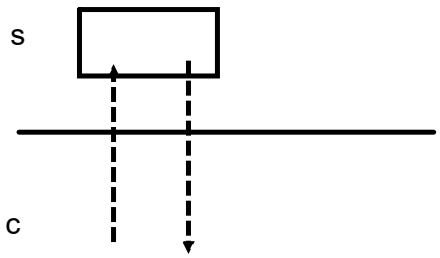
Most popular API to create web app using java

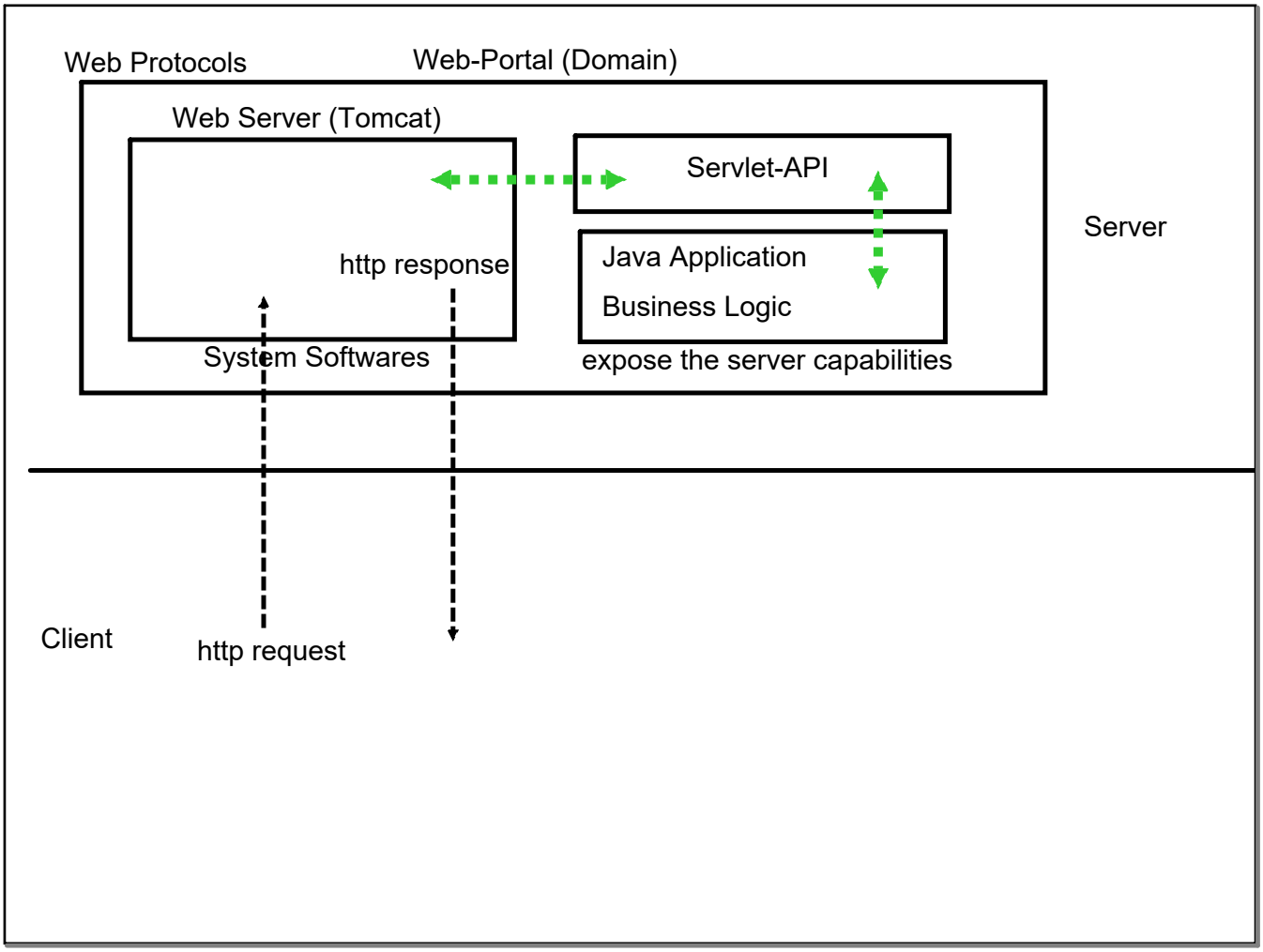
Core Java

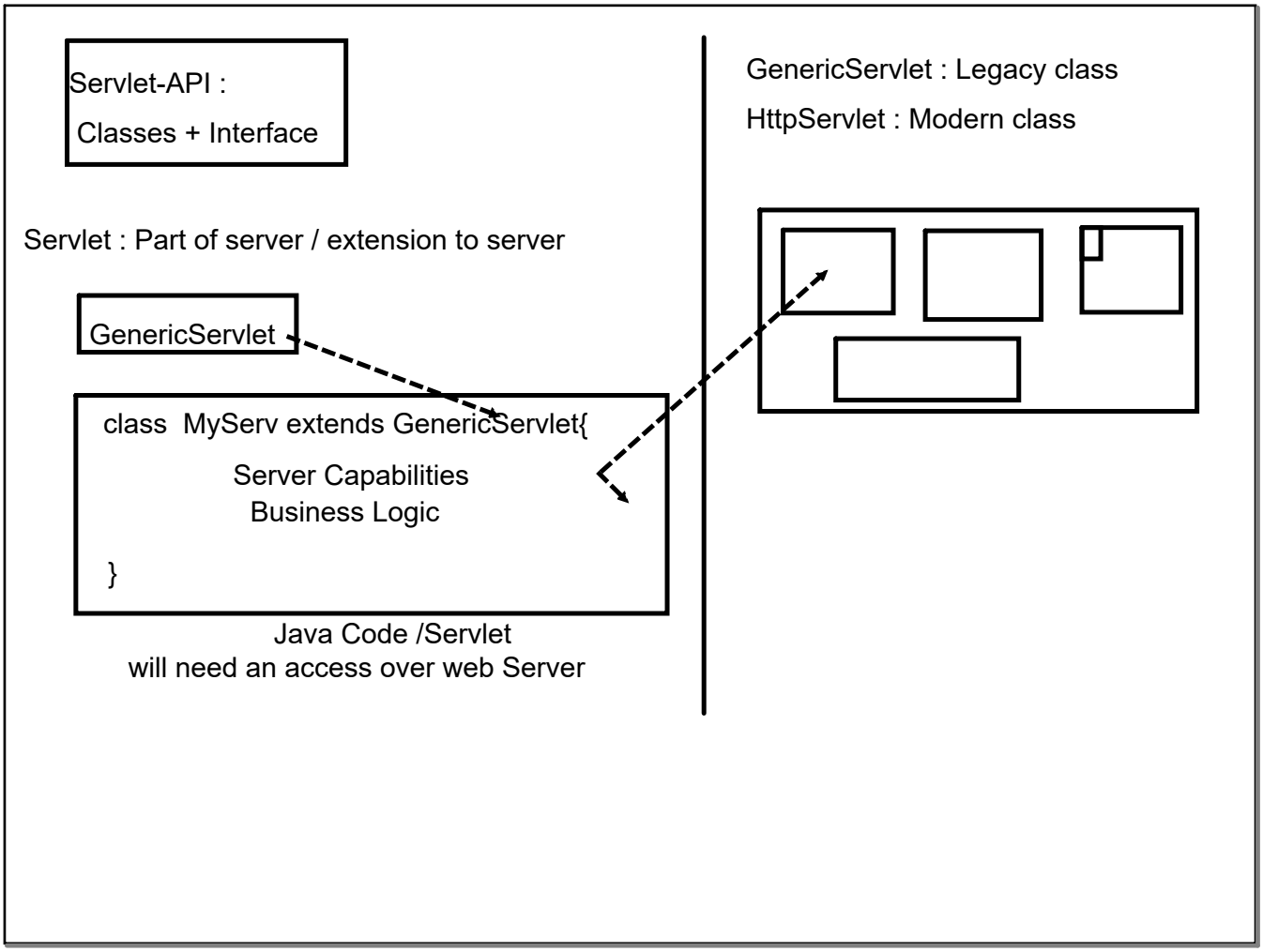
Popular Frameworks

JavaEE, Spring, Struts, EJB....

Web based Application (HTTP Protocols/Web Protocols)




Web Protocols
Web-Portal (Domain)
Web Server (Tomcat)
Servlet-API
Server
http response
Java Application
Business Logic
System Softwares
expose the server capabilities
Client
http request


Servlet-API :
Classes + Interface
GenericServlet : Legacy class
HttpServlet : Modern class
Servlet : Part of server / extension to server
GenericServlet
class MyServ extends GenericServlet{
Server Capabilities
Business Logic
}
Java Code /Servlet
will need an access over web Server

Team member1 name :

Team member2 name :

---

eg :

user

id (PK) int AI

name varchar(50)

email varchar(50)

age int

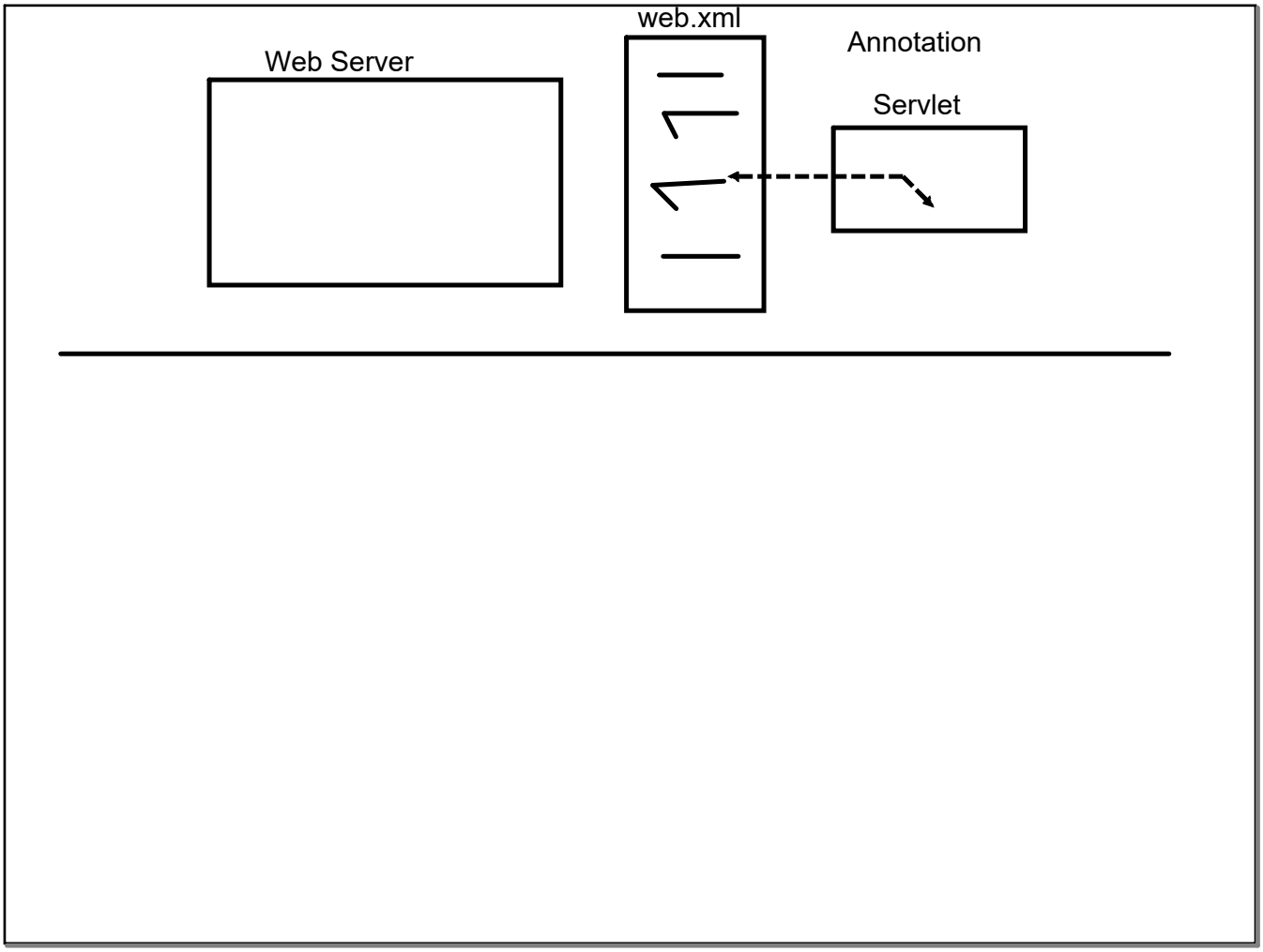
```
class User{
        private int id;
        private String name;
        private String email;
        private int age;
        ----getter
        ----setter
}
```

GenericServlet :

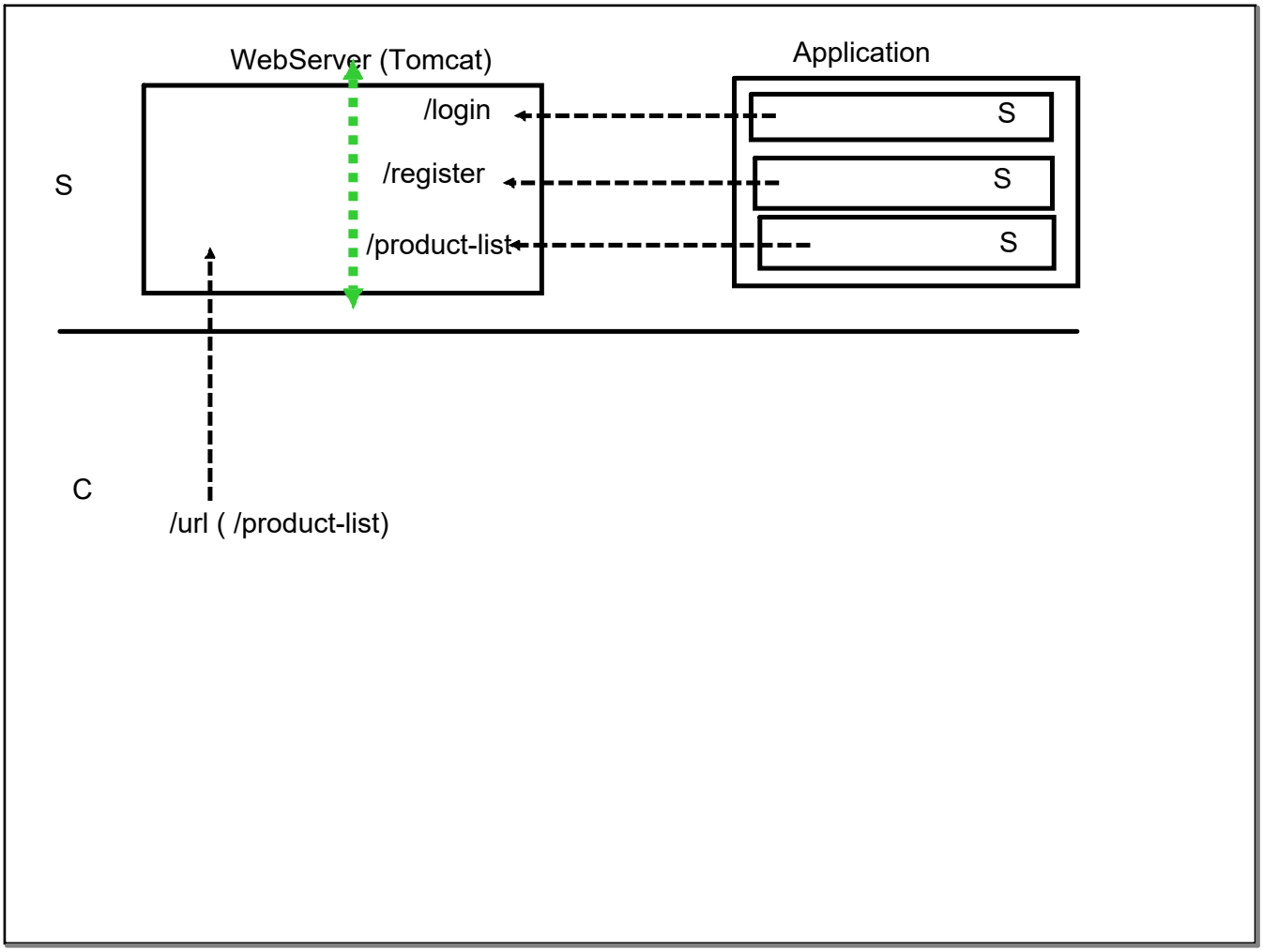
does not diffrentiate between http verbs : GET/PUT/POST/DELETE/PATCH

HttpServlet : can diffrentiate

1. Each Servlet class is req to be registered with WebServer
2. Registration is done using manifest file : deployment descriptor
   web.xml (de-facto std)
3. Each Servlet exposes a url
4. WebServer manages the lifecycle of reg. servlets


web.xml
Annotation
Web Server
Servlet

Phase1


WebServer (Tomcat)
Application
/login
/register
/product-list
S
S
S
S
C
/url (/product-list)

When a url request of a servlet is recieved by webserver

1. Creates an object of that class
2. Launching lifecycle phases
    a. init() : hooks : prepare for handling the request
    b. service() : hooks : recieve the request/data, processing, responding
    c. destroy() : hooks : release the resources
3. make object available for garbage collector

First time request :

1---->2(a)------>2(b)----------> cache the object

Next request

2(b)---------->cache

Cache :

1. time
2. capacity

Whenever Object would be req to remove from cache/new version of servlet/server is restarted

2(c)----> 3

Sequential

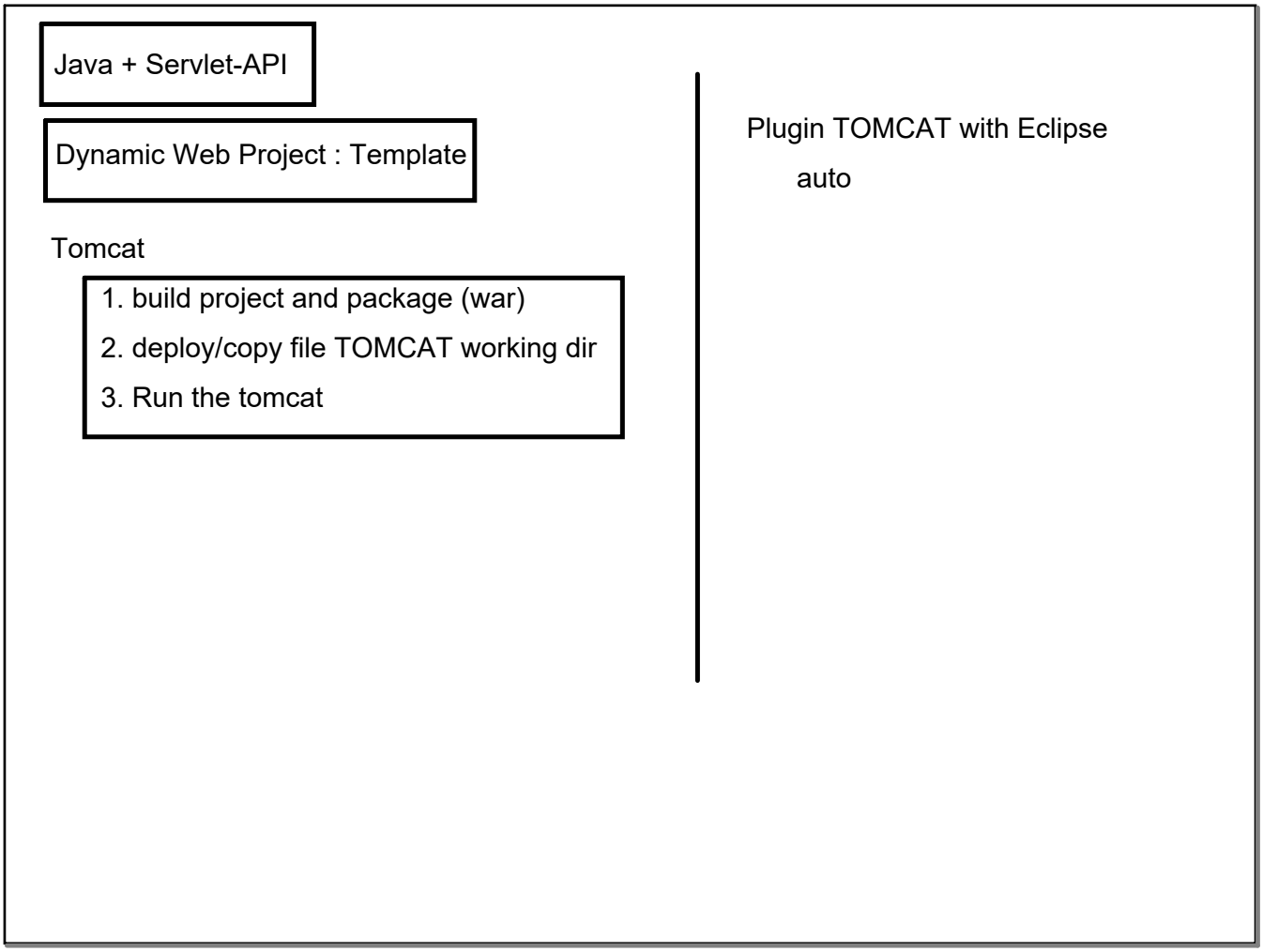
Parallel (n)

=> create n instance of servlet (overhead)

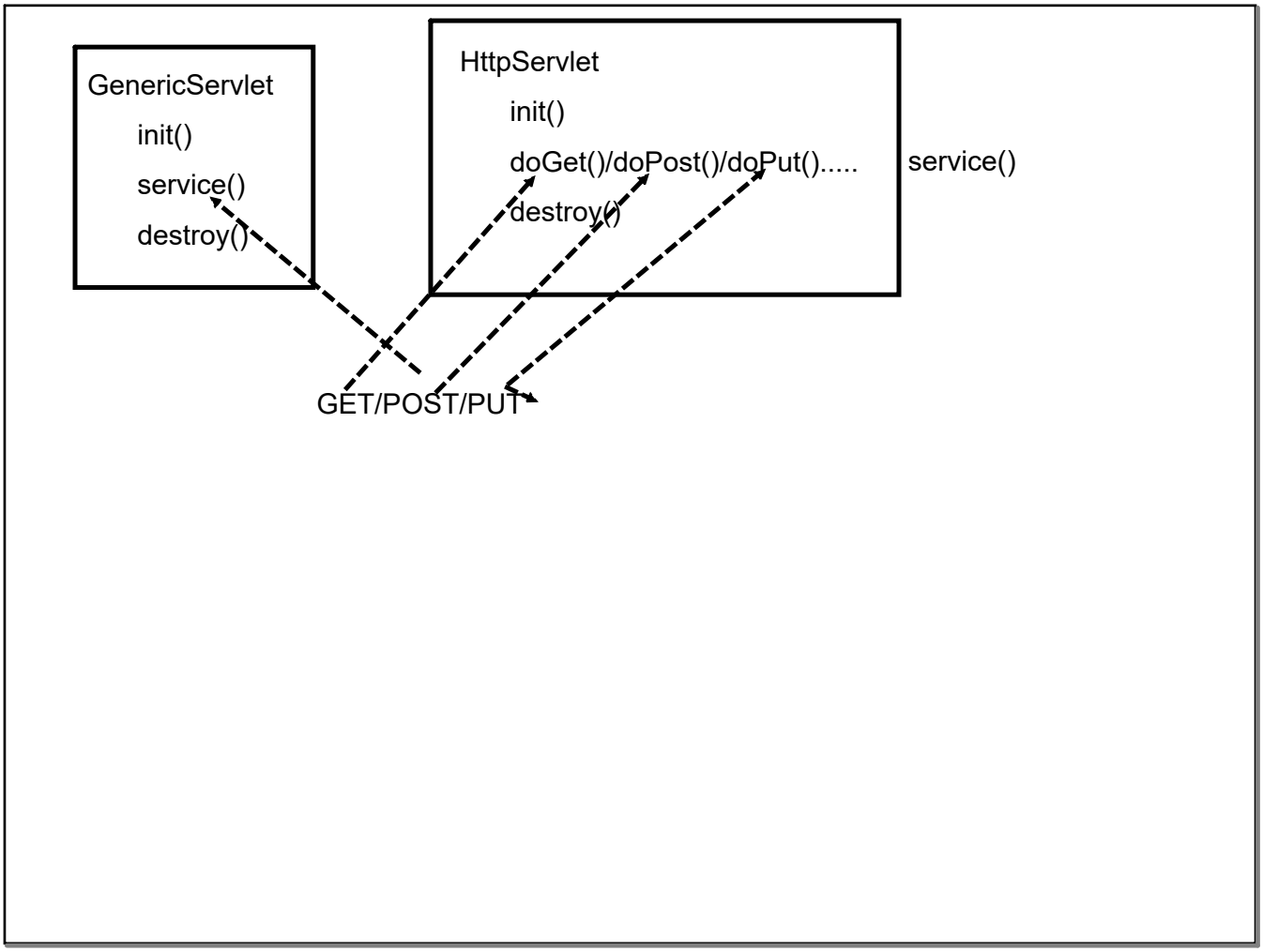
=> create a queue (performance lag)

=> Multithread (create threads of 2(b))

Servlet : Thread Safe resources


Java + Servlet-API
Dynamic Web Project : Template
Tomcat
1. build project and package (war)
2. deploy/copy file TOMCAT working dir
3. Run the tomcat
Plugin TOMCAT with Eclipse
auto

Phase1


GenericServlet
init()
service()
destroy()
HttpServlet
init()
doGet()/doPost()/doPut().....
destroy()
service()
GET/POST/PUT

Request

Response

```
HttpServletRequest request  ==> get any info

HttpServletResponse response ==> respond

response => HTML Page
JAVA CODE  ==> HTML
response object provide a writer : write HTML back to client
```

JSP : Java Server Pages

JAVA + HTML : Dynamic presentation

<%= %>

substitution tag

Substitute dynamic value in HTML

<%

JAVA CODE

%>

scriptlet tag

<%--

--%>

JSP Comment

<%!

%>

Declaration

Each JSP exposes as URL : filename is the url

JSP :

=>Concept

=>Runtime JSP does not exits

=> JSP another way of writing Servlet

JSP--->Servlet

compile

All Code of JSP

HTML,scriptlet tag, substitution : service

init()

destroy()

anything part of servlet class outside of service

Declaration Tag

```
class JSPSERVLETCLASS{
    String data = "";
    service(){
        String str = "Hello";
    }
}
```