Phase 1 : Java + Servlet API

Phase 2: Spring Framework

Phase 3: DevOps

Java - 8

IoT

Java-8 : Functional Programming

Functional Interface

default method

static method

Lambdas

Streams

Method references

Optional

Concurrent Support in Collection API

DateTime API

Nashorn Engine (JS engine)

Phase1

Imperative style of programming

Classical style/Traditional style

pure OOPs

Focus how to perform operation

Object mutability : bugs

Declarative Style

Focus on result you want

Analogous SQL

Object Immutability
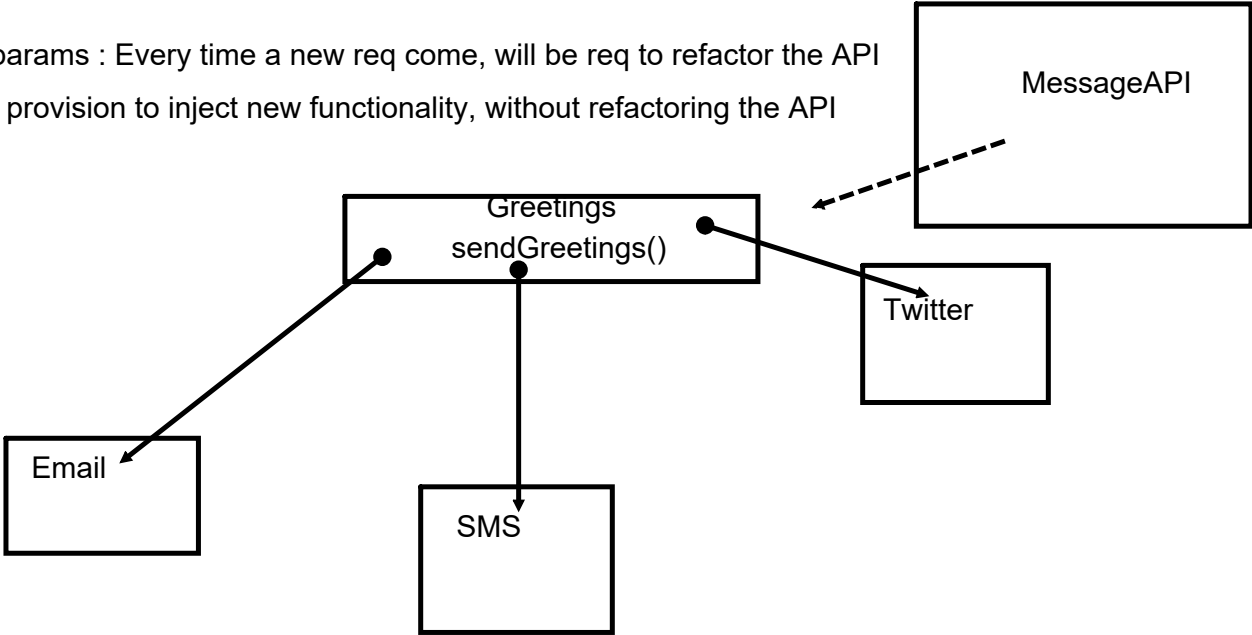
Functional Programming

List of numbers

fetch unique number

Reverse domain naming convention

Phase1

params : Every time a new req come, will be req to refactor the API

: provision to inject new functionality, without refactoring the API

MessageAPI

Greetings
sendGreetings()

Twitter

Email

SMS

Phase1

Functional Programming : Functions(pure) are first class citizens

No Object Overheads

variable/instance/reference : object
reference = function

New datatype would not have been
backward compatible

Expect a special datatype from JAVA : Function
Function twitter = ()

Extended the behavior of existing feature : interface

Syntax : Lambda

1. no access modifier : (not the part of any class)

2. no name (anonymous function)

3. no return type (can return values)

4. params : no param type

5. <param> -> {<definition>}

```
void fun(){
}
() -> {
}
```

```
void fun(String str){
     <single inst>
}
str-> <single inst>
```

```
void fun(String str1,String str2){
}
(str1,str2)->{
}
```

```
void fun (String str){
}
str -> {
}
```

```
void fun(String str){
-------------
-------------
}

str -> {
------------------
------------------
}
```

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

Functional Interface

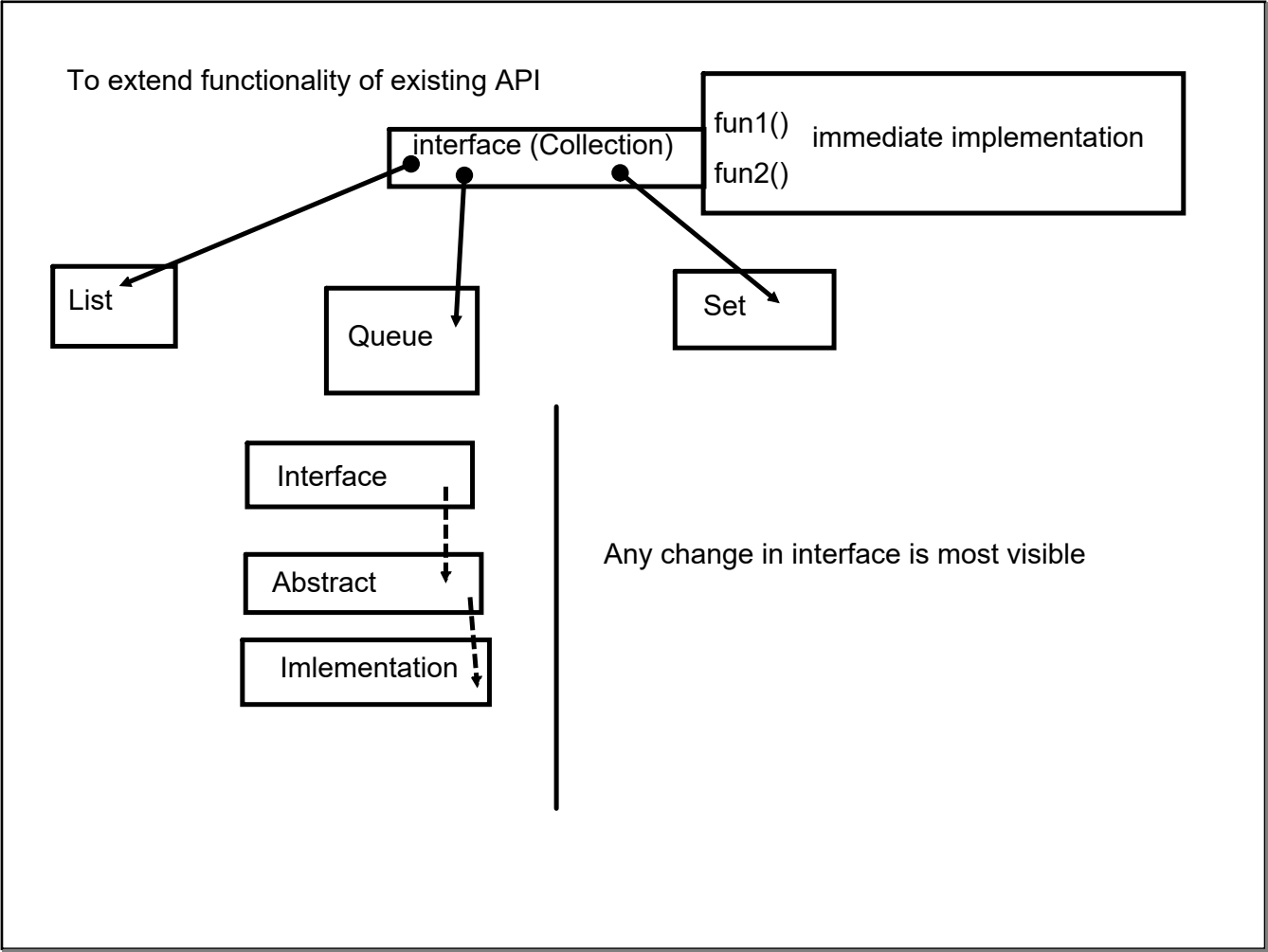   Contains only 1 abstract method, any number of default and static

\# Only Functional Interfaces can refer to lambda expression

\# Signature of Lambda expression must match with the only abstract method of FI

Interface :

     Define function inside an interface.

     Interface can have functions with definitions as well

To extend functionality of existing API

interface (Collection)

fun1()

fun2()

immediate implementation

List

Queue

Set

Interface

Abstract

Imlementation

Any change in interface is most visible

Phase1

Existing feature :

    #Functional Interface

Comparable

Comparator

Runnable

=> Specialized Libraries of Functional Interface

=> Streams

Lambdas with Local Variable

    1. Lambdas have an access over local variables and instance of enclosing scope

    2. Effectively final

    3. Not Allowed to use the same local variable name as param or redeclaration inside body

No restriction on instance variable

=> Easier to perform the concurrent operation : immutability

Collection of string : traverse through collection and display all strings in collection

iterable : iterator()

Collection
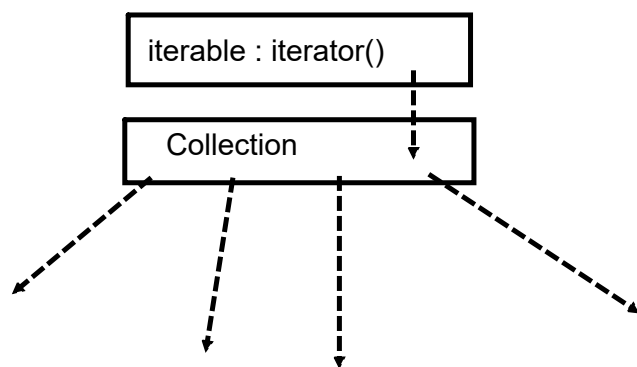
Functional API : Bunch of functional interface:

few prototypes have been identified with common usage

    java.util.function

Consumer: BiConsumer

    void accept(<>) : Consume the data

Predicate : BiPredicate

    boolean test(<>) : Add some condition and revert back

Function : BiFunction, UnaryOperation, BinaryOperator

    <> apply(<>) : Transformation

Supplier

    <> get() :

Streams : Pure Functional

    Perform operations on collections or I/O resource :
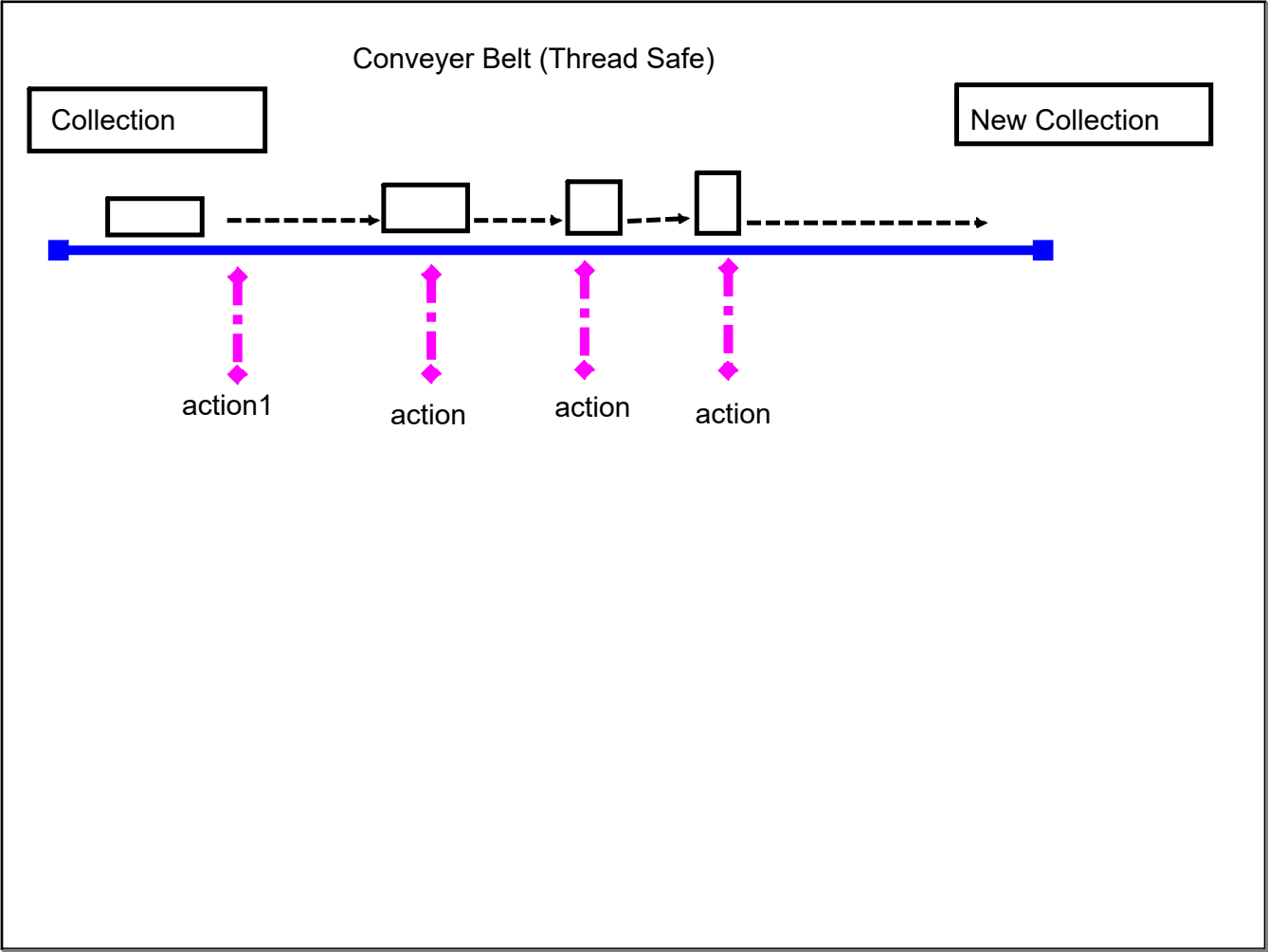
    Safe

        Immutability : Thread safe

    Efficient Way

        Not a data structure: not going to store any data

        Lazy processing model

    Parallel Stream : Parallel operation easily without spawning the thread

Phase1

Conveyer Belt (Thread Safe)

Collection

New Collection

action1

action

action

action

Phase1

SBA1 : use-case

SBA2 : use-case

SBA3 :

End-to-End

1. continous process

2. Milestone

3. walk-through (Friday)

4. Group based : group evaluation + individual eval

Every Stream must have a terminal activity

Else : Stream will not initiate

Every Stream

1. Initiate the stream

2. Intermediate activity (optional)

3. Terminal activity

Parallel Stream

Constraints in parallel streams

1. Order of records matter

2. where using a mutable service/data : Not a thread safe

3. activities, inherently complex, degrade performance

1,2,3,4,5,6,7.....

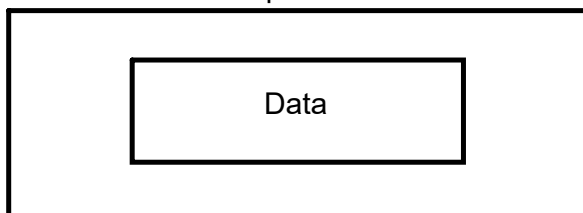4,2,6,3,1   result = 0

forEach(Consumer)

4 --> thread1
      = 0 + 4
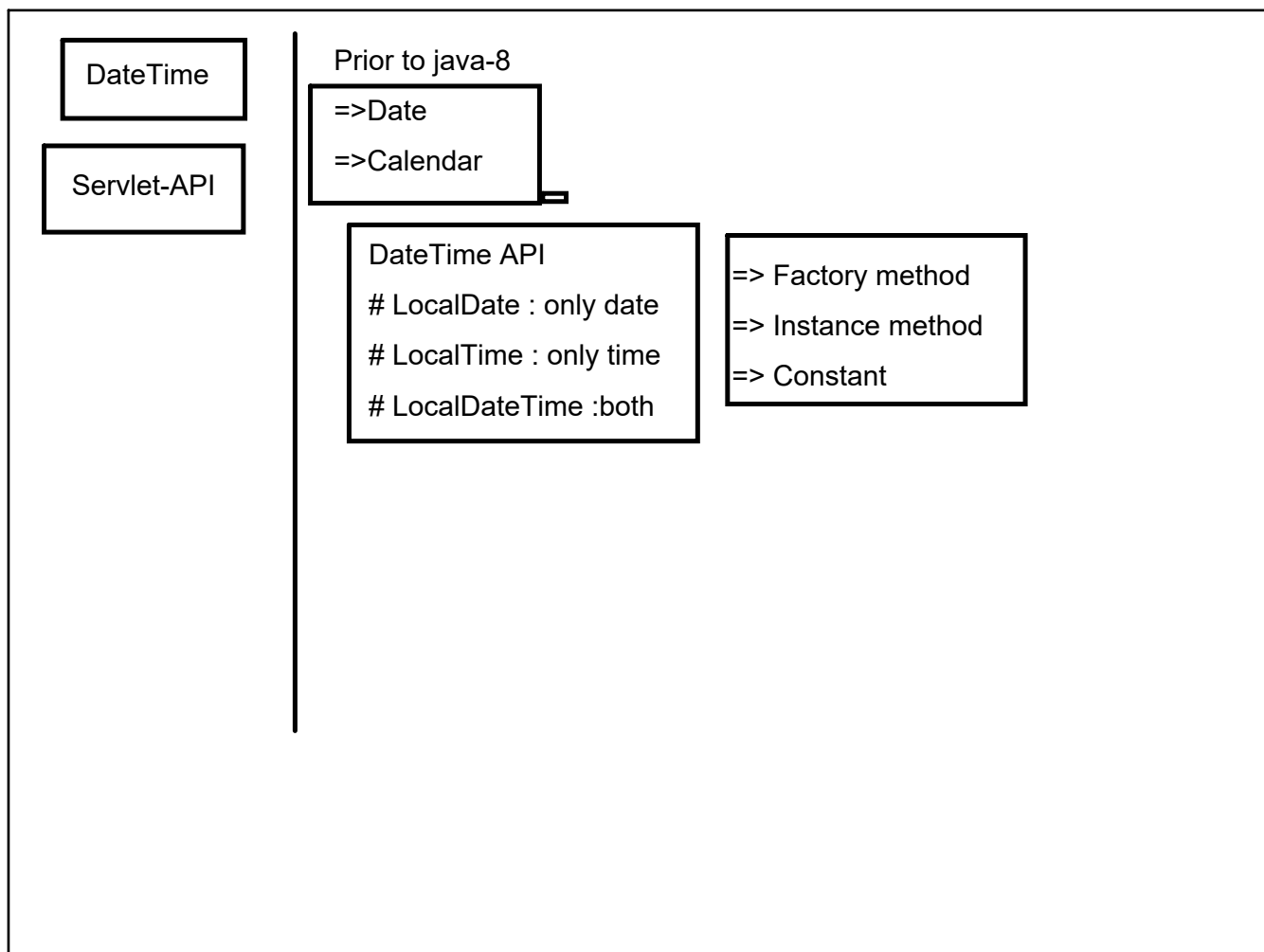   result = result + 4

2 --> thread2
      = 0 + 2
   result = result + 2

Optional : to avoid null reference exception

if any object being initialized by some external logic

need to check if it is

Suggested best Practise : never return raw data

Optional

Data

Phase1

DateTime

Servlet-API

Prior to java-8

=>Date

=>Calendar

DateTime API

LocalDate : only date

LocalTime : only time

LocalDateTime :both

=> Factory method

=> Instance method

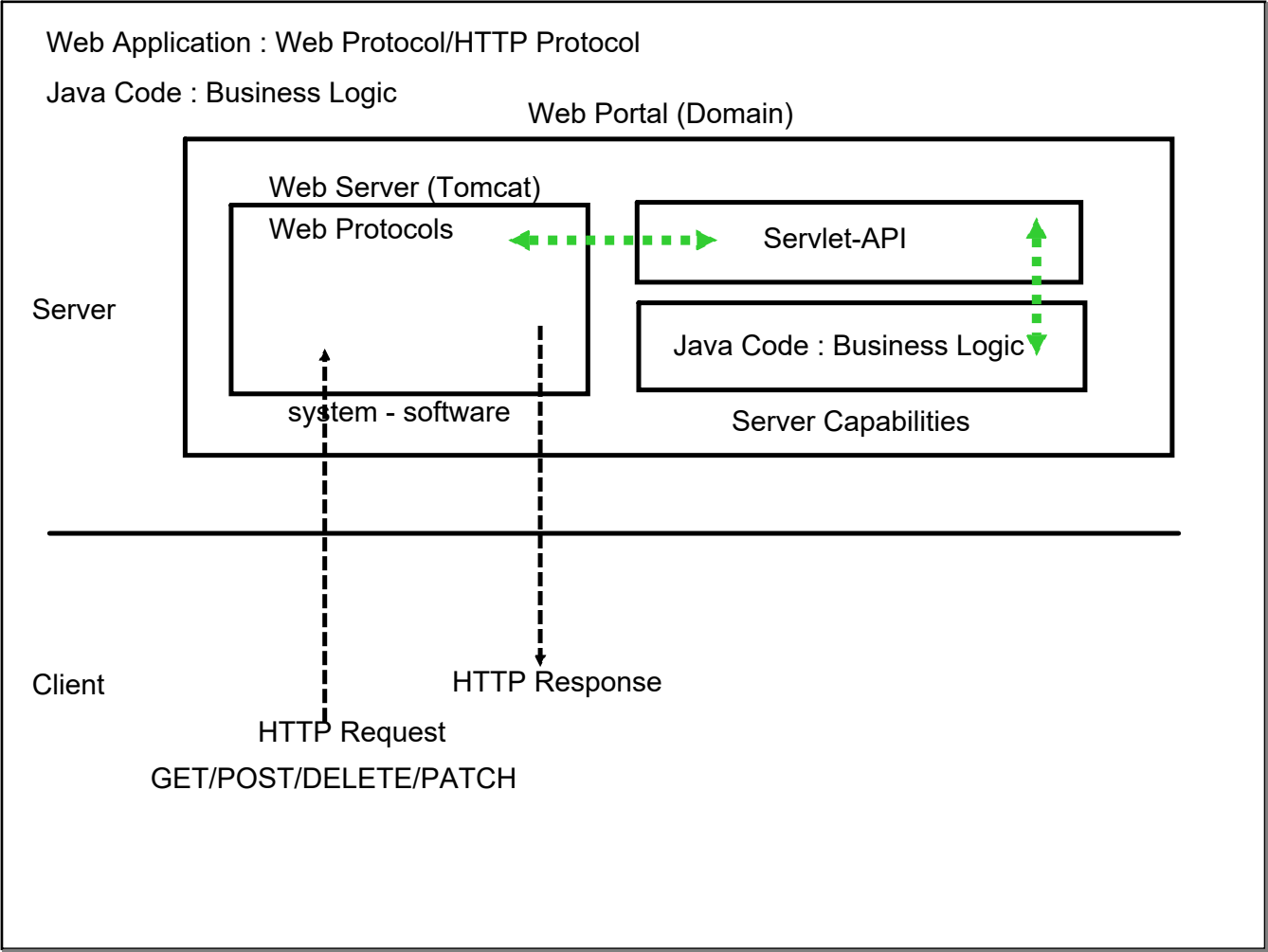=> Constant

Servlet-API:

    Popular API to create web application using Java

=> Core Java + Servlet-API

=> Complex Framework

Major Framework of JAva

    JavaEE, Spring, Struts, EJB....

Web Application : Web Protocol/HTTP Protocol

Java Code : Business Logic

Web Portal (Domain)

Web Server (Tomcat)

Web Protocols

Servlet-API

Server

Java Code : Business Logic

system - software

Server Capabilities

Client

HTTP Response

HTTP Request

GET/POST/DELETE/PATCH

Phase1

Servlet - API
=> Classes
=> Interface

=> GenericServlet(Legacy)/HttpServlet(Modern)

Java Class (Servlet)

Business Logic

S   S   S   S

GenericServlet : cannot differentiate among HTTP Verbs :

HttpServlet : can differentiate

class MyServ extends HttpServlet{

}

Servlet
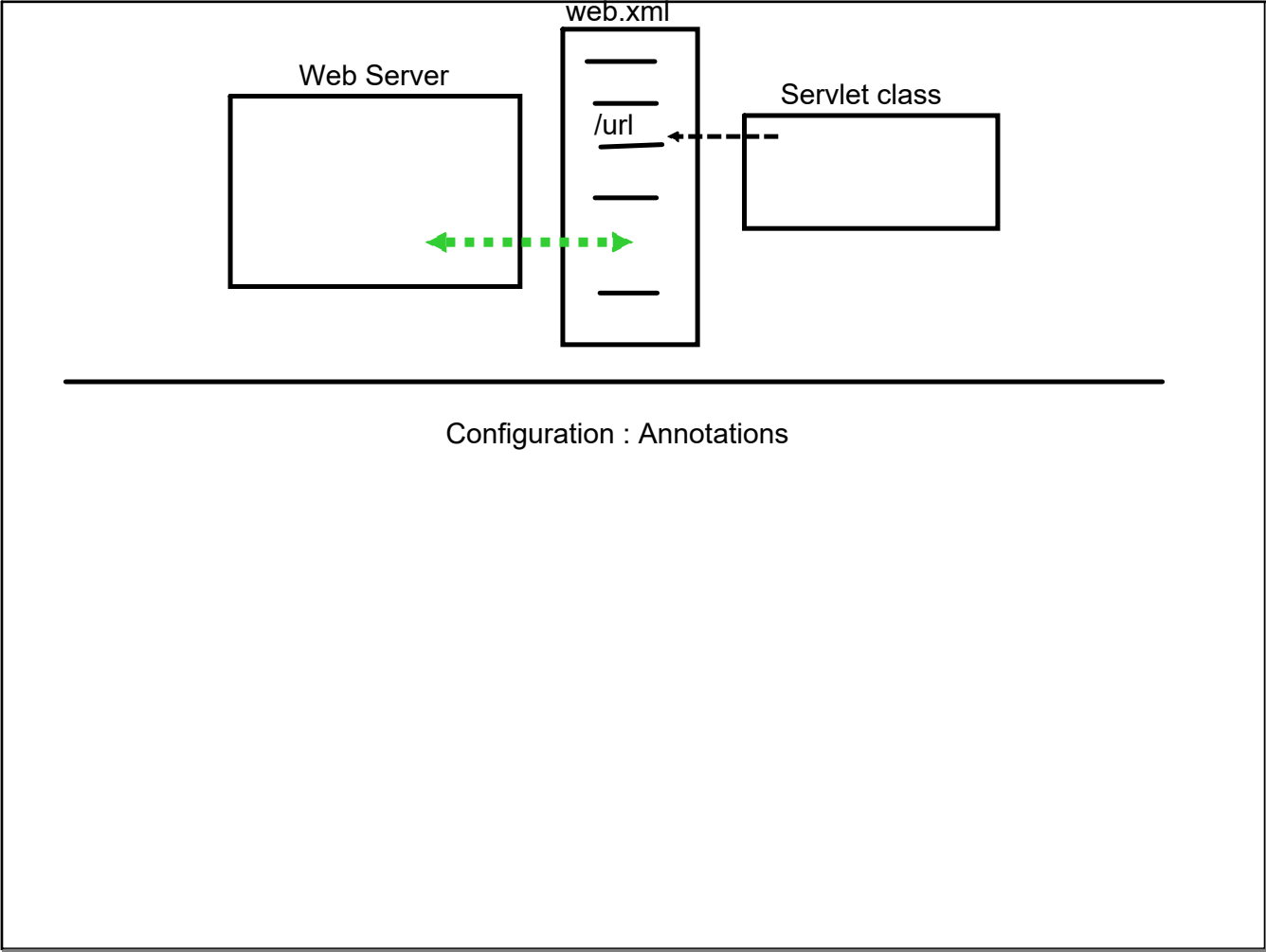
Every servlet needs to be registered with WebServer
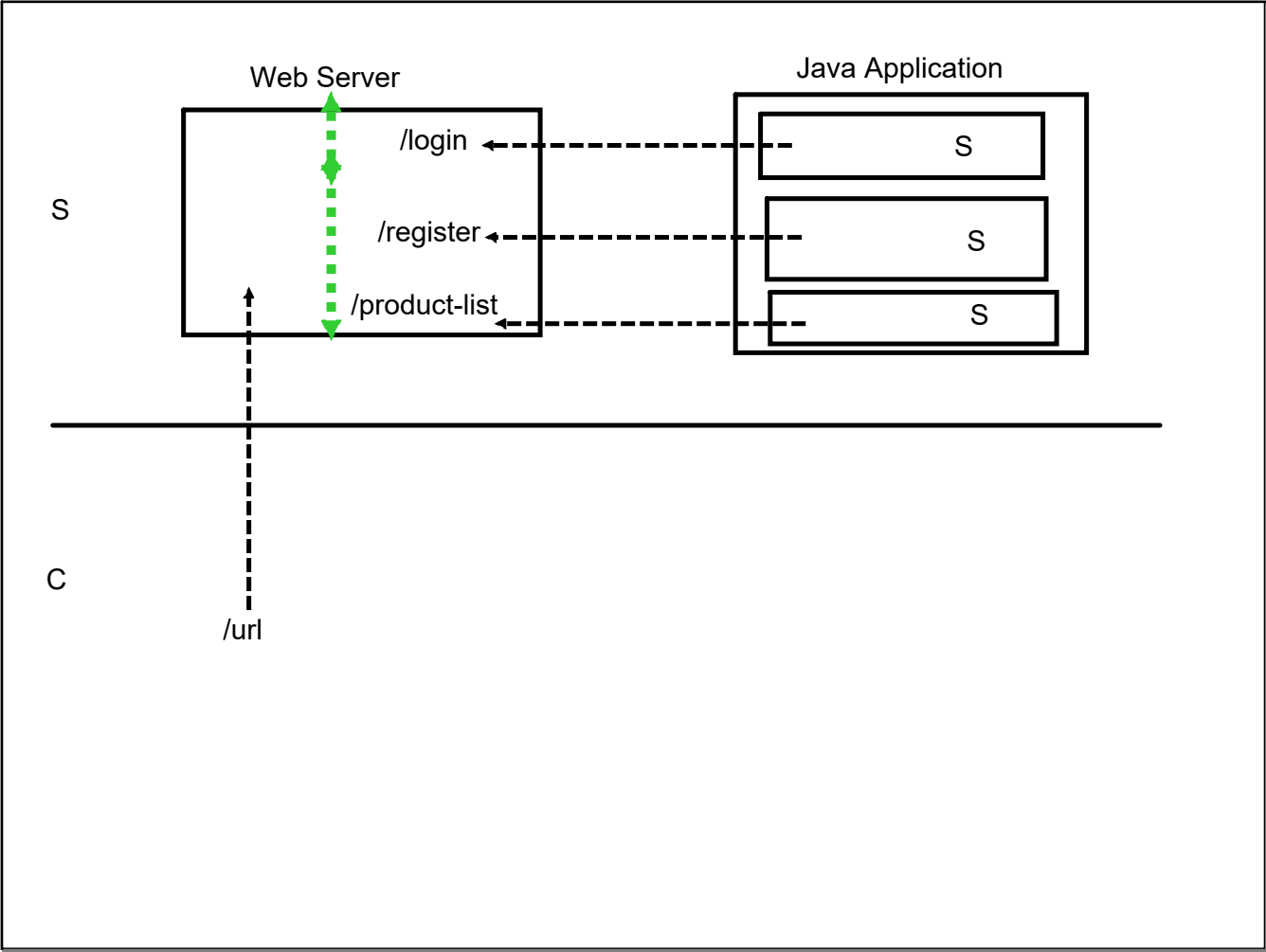
Registration will be based on URL

Registration will be done through a special manifest file

Deployment Descriptor : web.xml (de-facto std)

Once a servlet is registered , Web Server starts managing the lifecycle of Servlet

Phase1

web.xml

Web Server

/url

Servlet class

Configuration : Annotations

Phase1

MAnaging the life cycle:

    1. Create an object of that class

    2. launches life-cycle of servlet class

        a. init() : phase : <u>prepare for request processing</u>

        b. service() : phase : <u>access over request, processing, respond back</u>

        c. destroy() : phase : <u>release the resources</u>

    3. Make the servlet object available for garbage collection

First Request : 1->2(a)->2(b)----> cache the object

Next Request(s) : 2(b) ----> cache the object

Whenever :

    Servlet is not requested for long time/capacity :

        2(c)-->3

Sequential : 2(b)

Parallel : n user requesting same servlet

    Object cached :

=> n new object of servlet
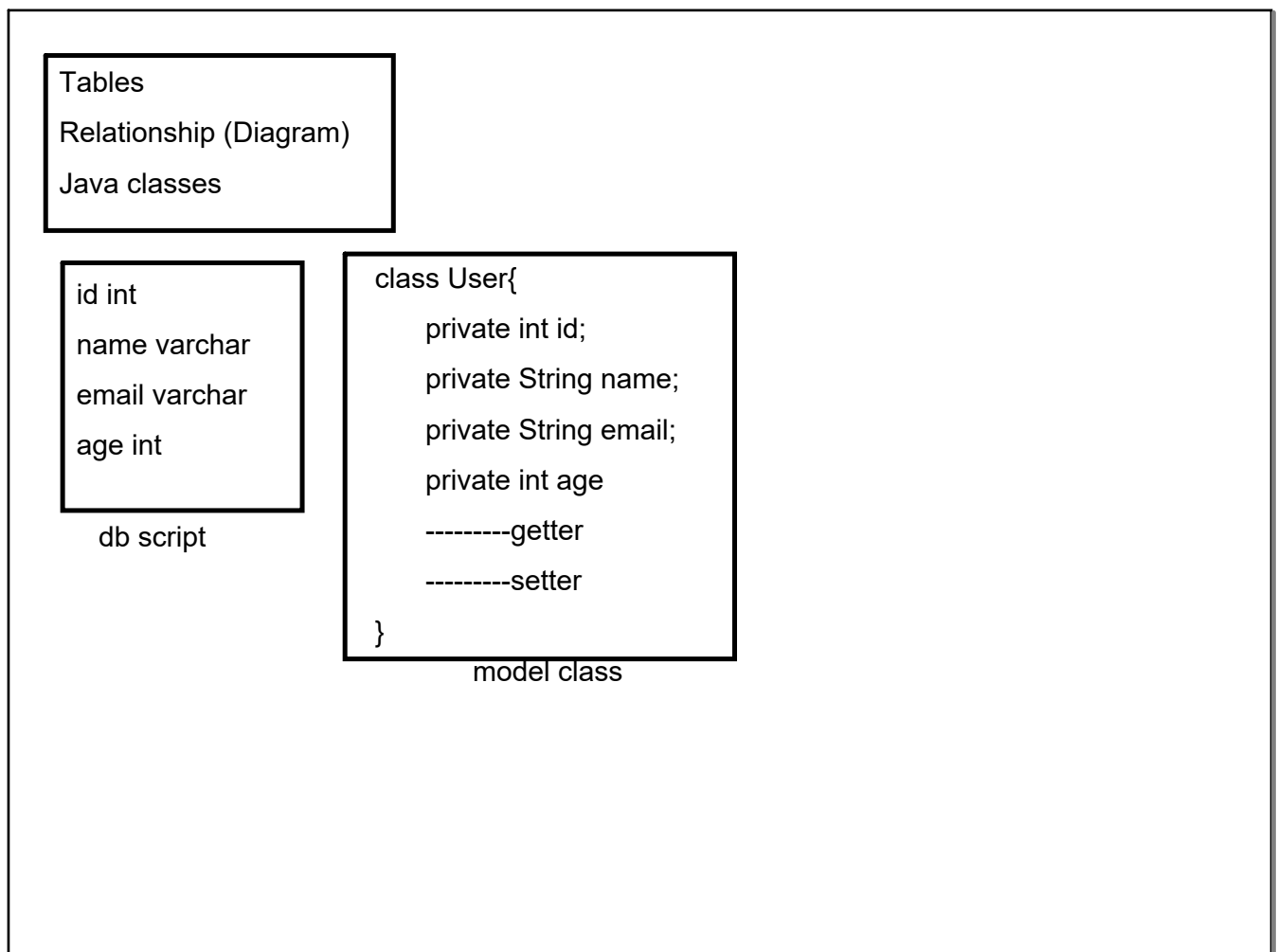
=> queue of request

=> multithreading : threads of service (phase(2(b))

Servlet must use Thread safe service

Names1 :

Name 2:

Phase1

Tables

Relationship (Diagram)

Java classes

id int

name varchar

email varchar

age int

db script

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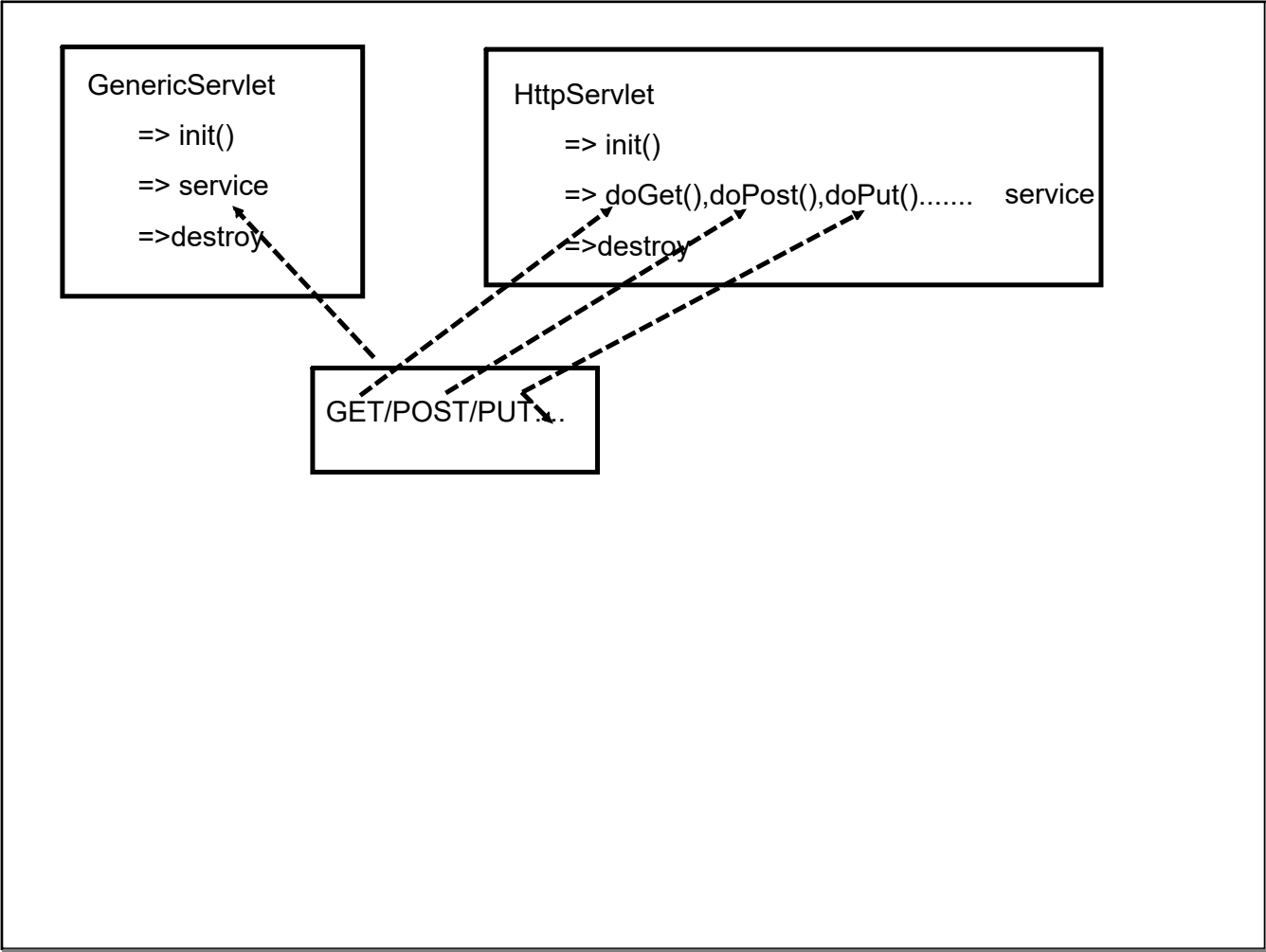model class

Java Application + Servlet-API

Template : Dynamic Web Project

Eclipse IDE : plugin the Tomcat

auto process

Tomcat Server

1. build and package (war) the project

2. deploy/copy to working dir TOMCAT

3. Launch the server

Phase1

GenericServlet
   => init()
   => service
   =>destroy

HttpServlet
   => init()
   => doGet(),doPost(),doPut().......   service
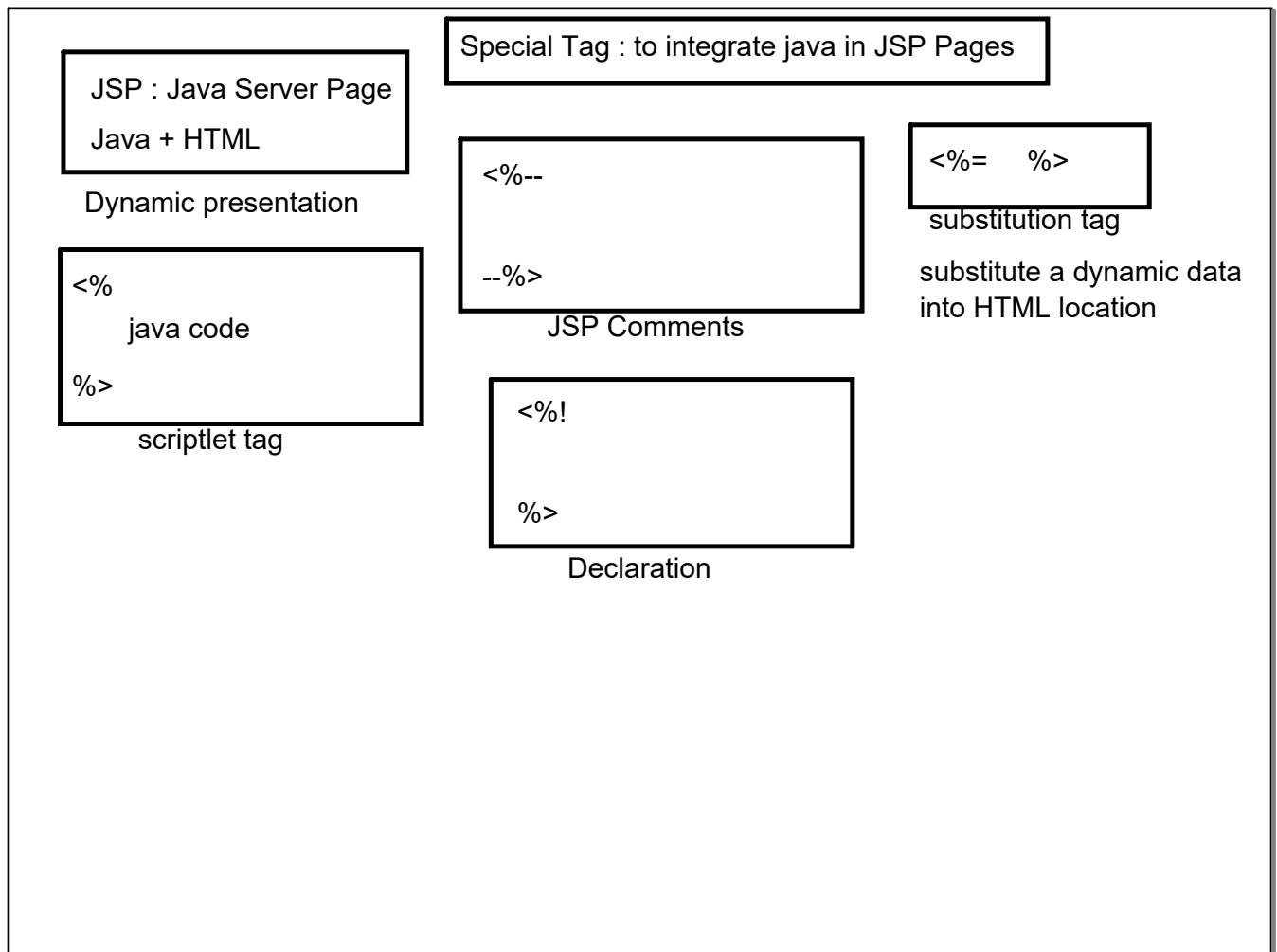   =>destroy

GET/POST/PUT...

```
HttpServletRequest request  => all info/functionality about request
HttpServletResponse response => all info/functionality about response
```

Response : HTML

JAVA CODE RESPOND BACK

1. Login Form (home-page)

2. Info as request ----------> Servlet

3. Fetch those info

4. business logic for credential

5. Welcome // Invalid

JSP : Java Server Page

Java + HTML

Dynamic presentation

```
<%
     java code
%>
```
scriptlet tag

Special Tag : to integrate java in JSP Pages

```
<%--


--%>
```
JSP Comments

```
<%!


%>
```
Declaration

```
<%=     %>
```
substitution tag

substitute a dynamic data into HTML location
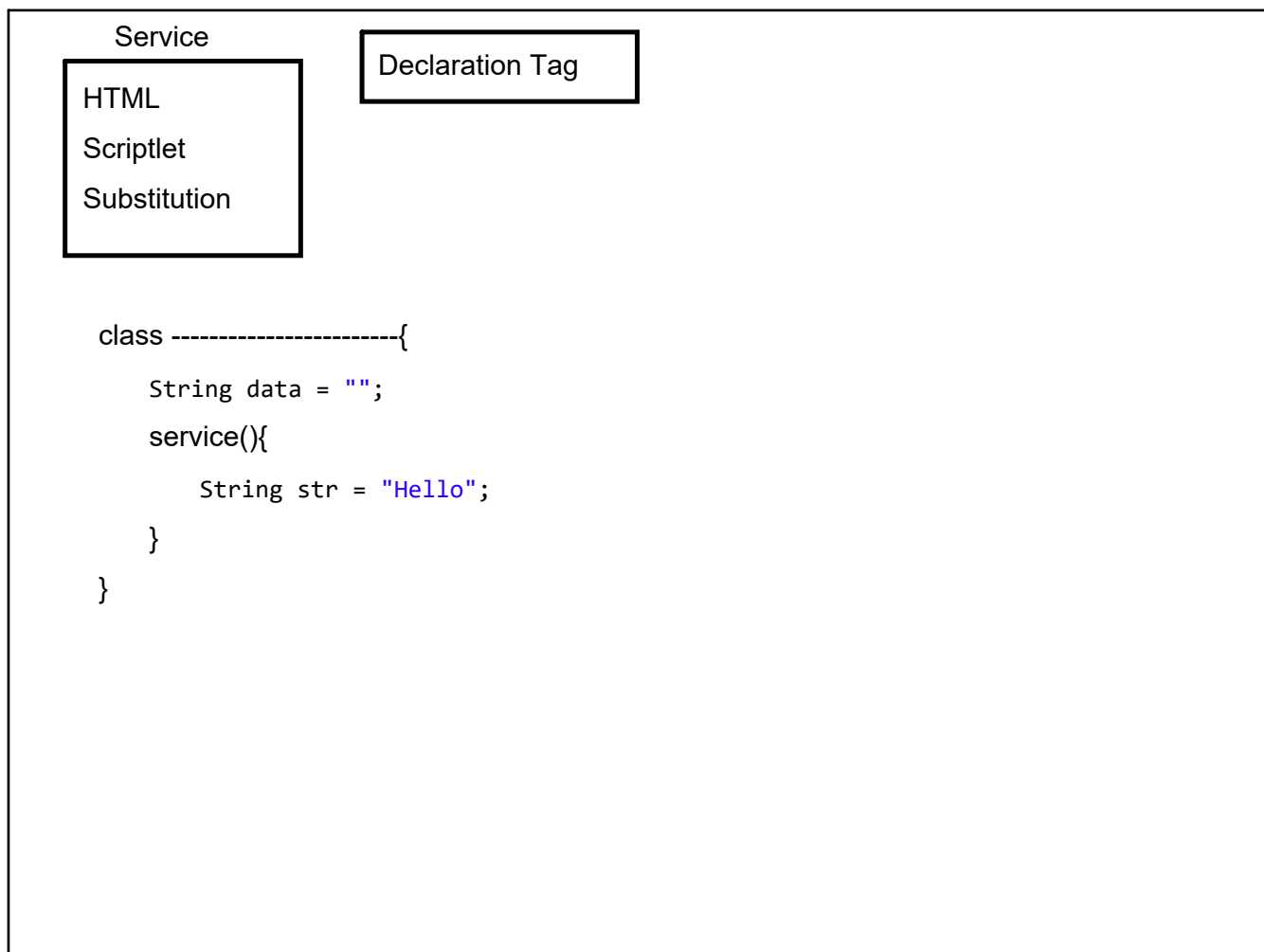
JSP : Concept

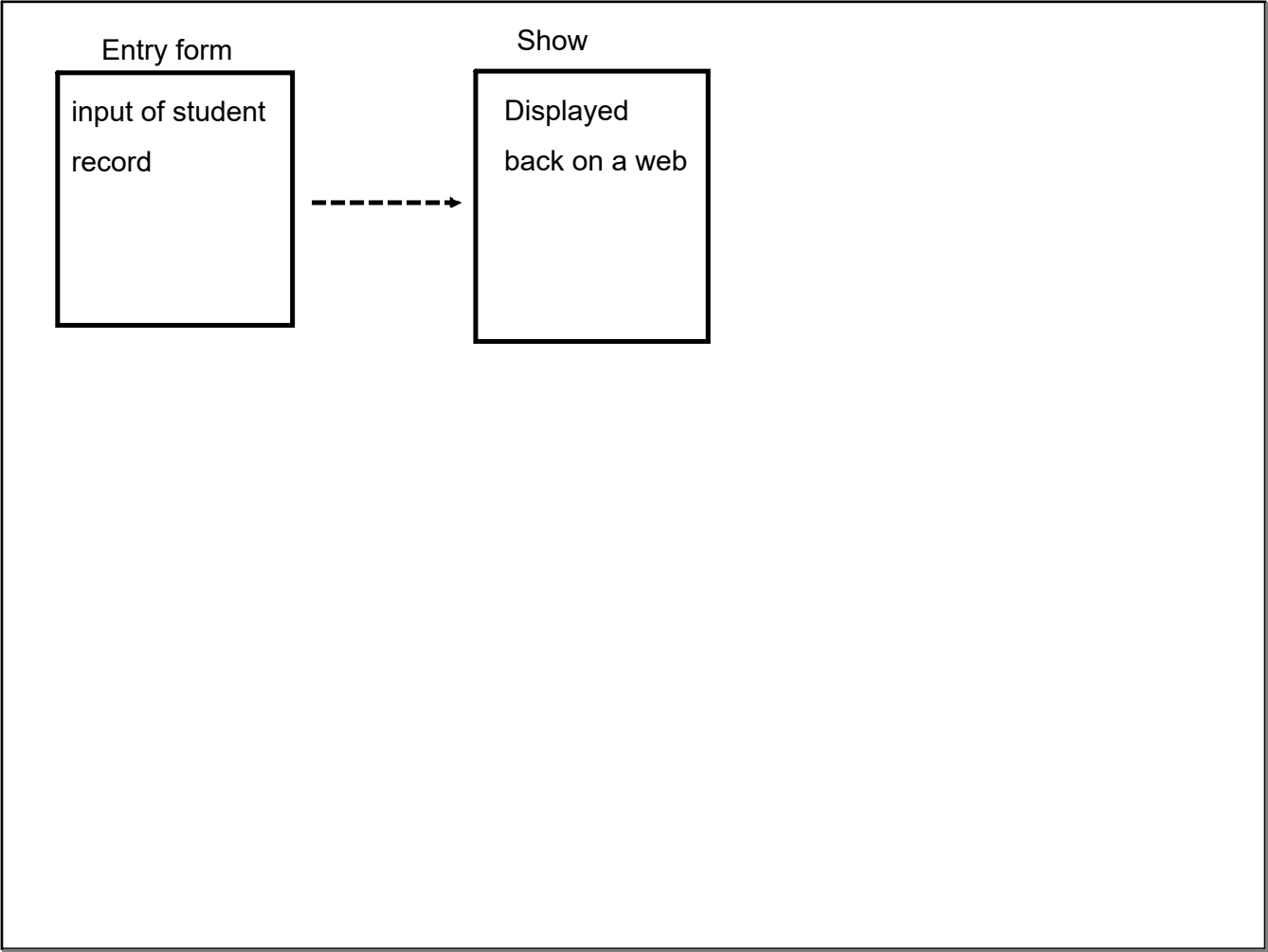=> Does not exists at runtime

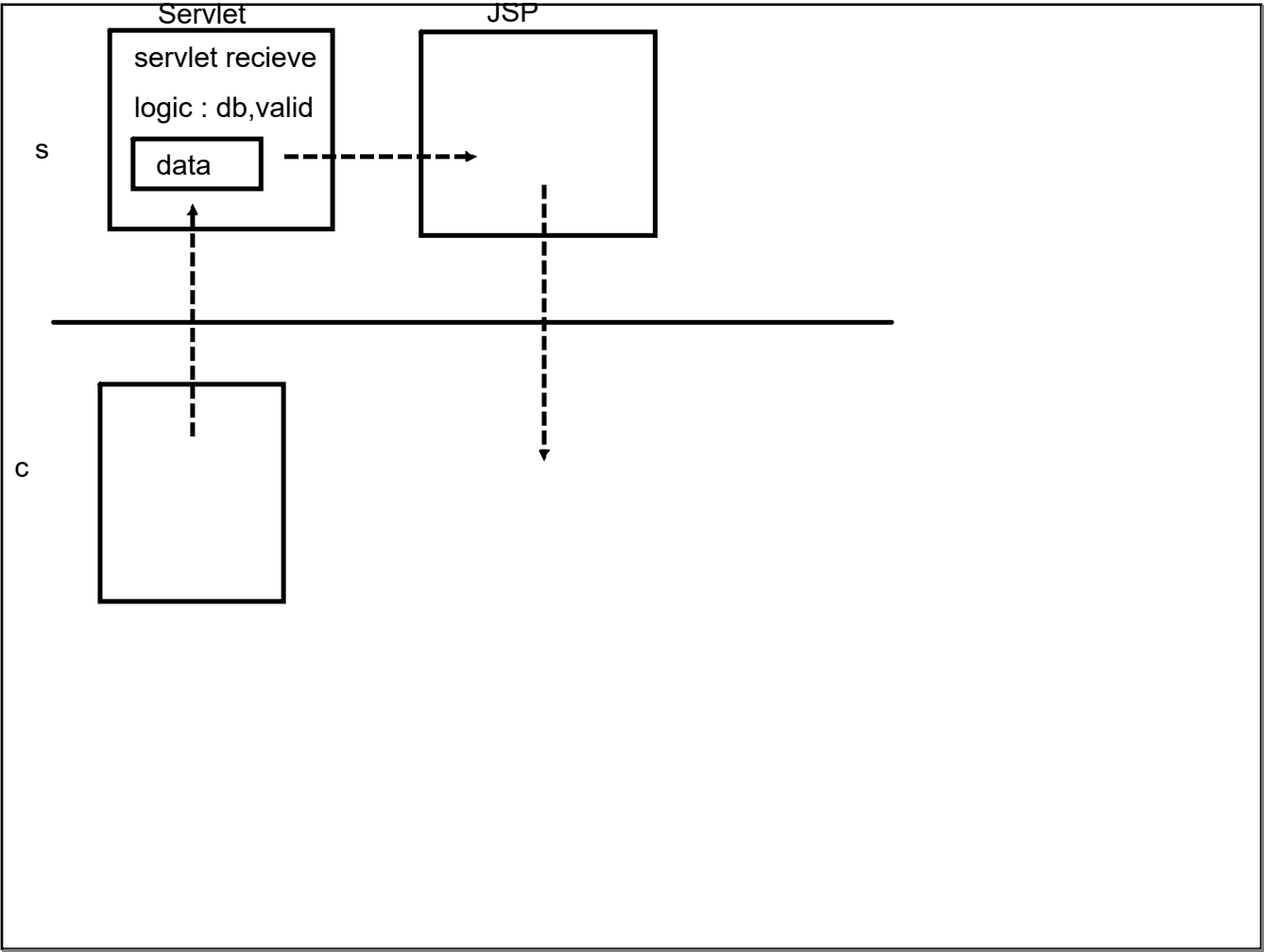=> Another way of writing the servlet

=> JSP---------------Servlet

Dynamic web Create :

Servlet : Java business logic (class instance)

Servlet : Heavy presentation (jsp instance)

Service

HTML
Scriptlet
Substitution

Declaration Tag

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

Phase1

Entry form

Show

```
┌────────────────┐          ┌────────────────┐
│ input of student│  - - - ->│ Displayed       │
│ record          │          │ back on a web   │
│                 │          │                 │
│                 │          │                 │
└────────────────┘          └────────────────┘
```

Servlet   JSP

servlet recieve

logic : db,valid

s

data

c

unique to a user

shareable/global

one cycle

session

application context

request

multiple cycle

multiple cycle

Servlet

JSP

Data

Phase1

Forward...

S1　　　S2　　　S3　　　JSP

Phase1

main-menu

Entry form

<add new>

<list all>

Product Inventory

add new products

list all product

delete

edit

lists all product
x
x
x
x

X

Edit
Page

Phase1

jsp/html   servlet   jsp   servlet

validate   list all   delete logic

save db   product

S

C

<add>   Entry

<list>   Form

main-menu   list product
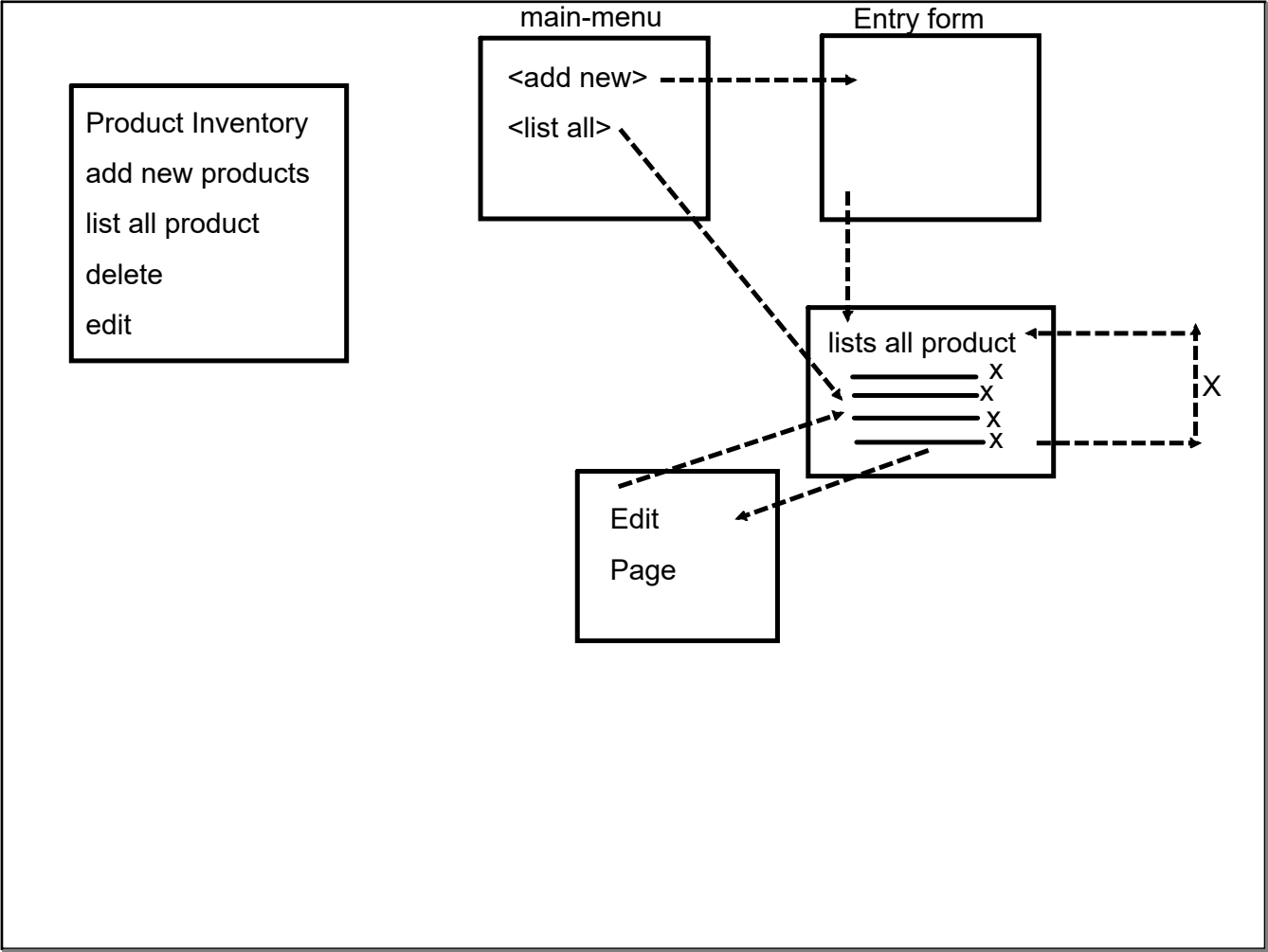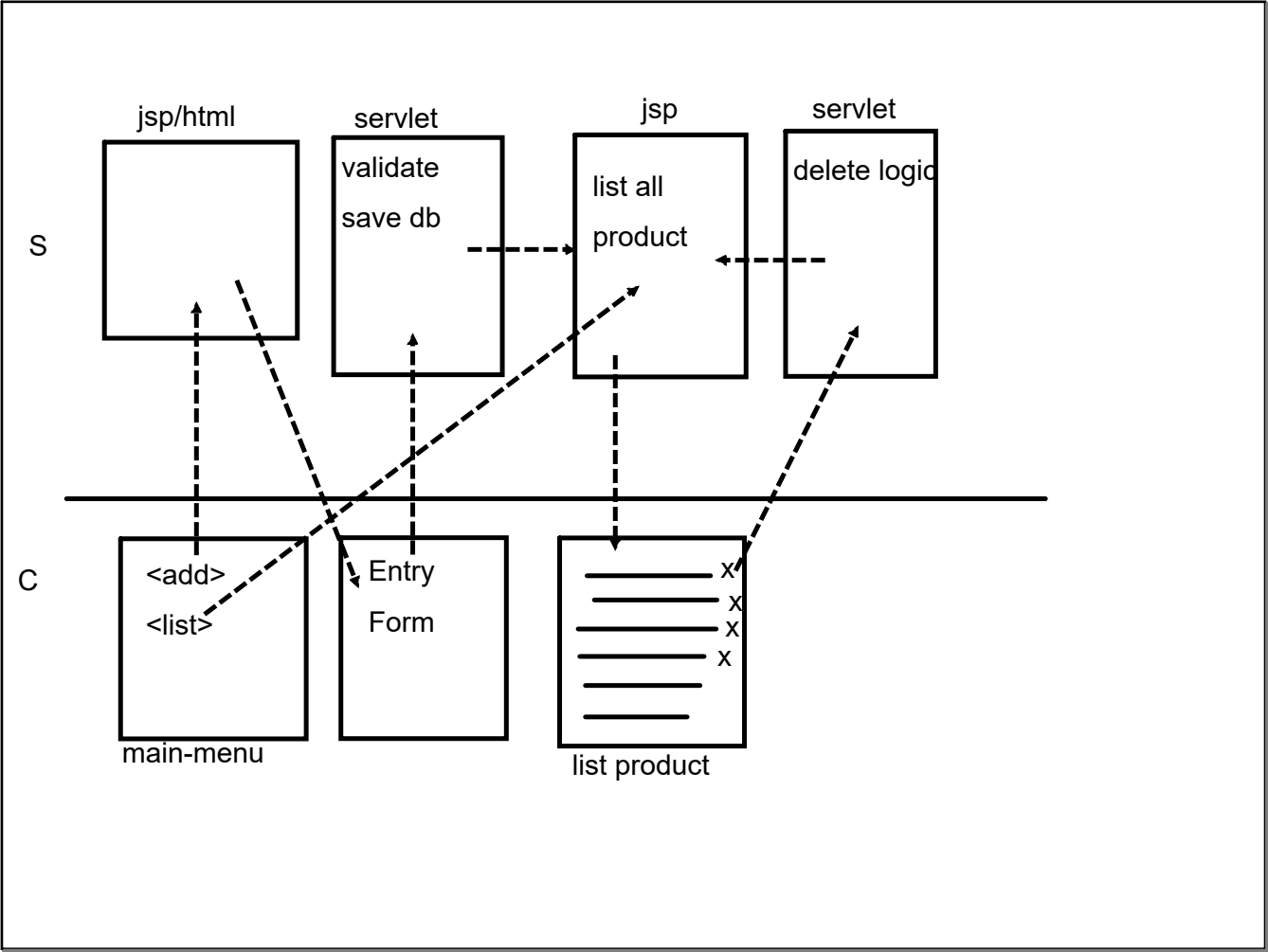
x
x
x
x

MVC Architecture

Best Practice

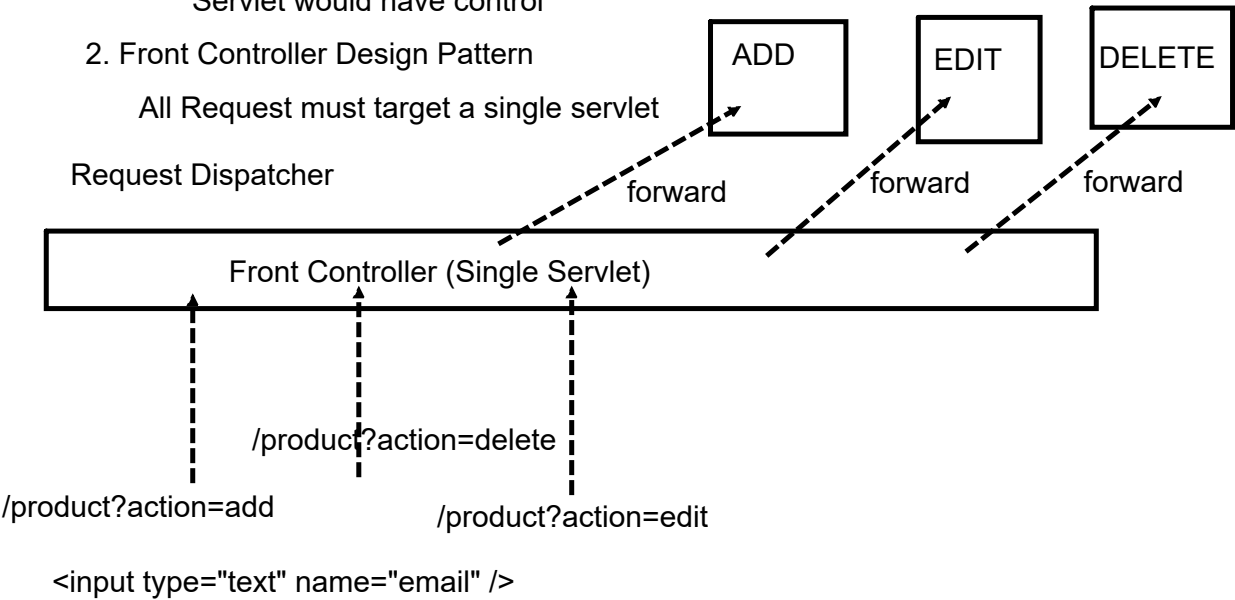   1. Never be calling a html/JSP

        All calls to presentation resource, should be through servlet only

        Servlet would have control

    2. Front Controller Design Pattern

        All Request must target a single servlet

Request Dispatcher

ADD	EDIT	DELETE

forward   forward   forward

Front Controller (Single Servlet)

/product?action=delete

/product?action=add

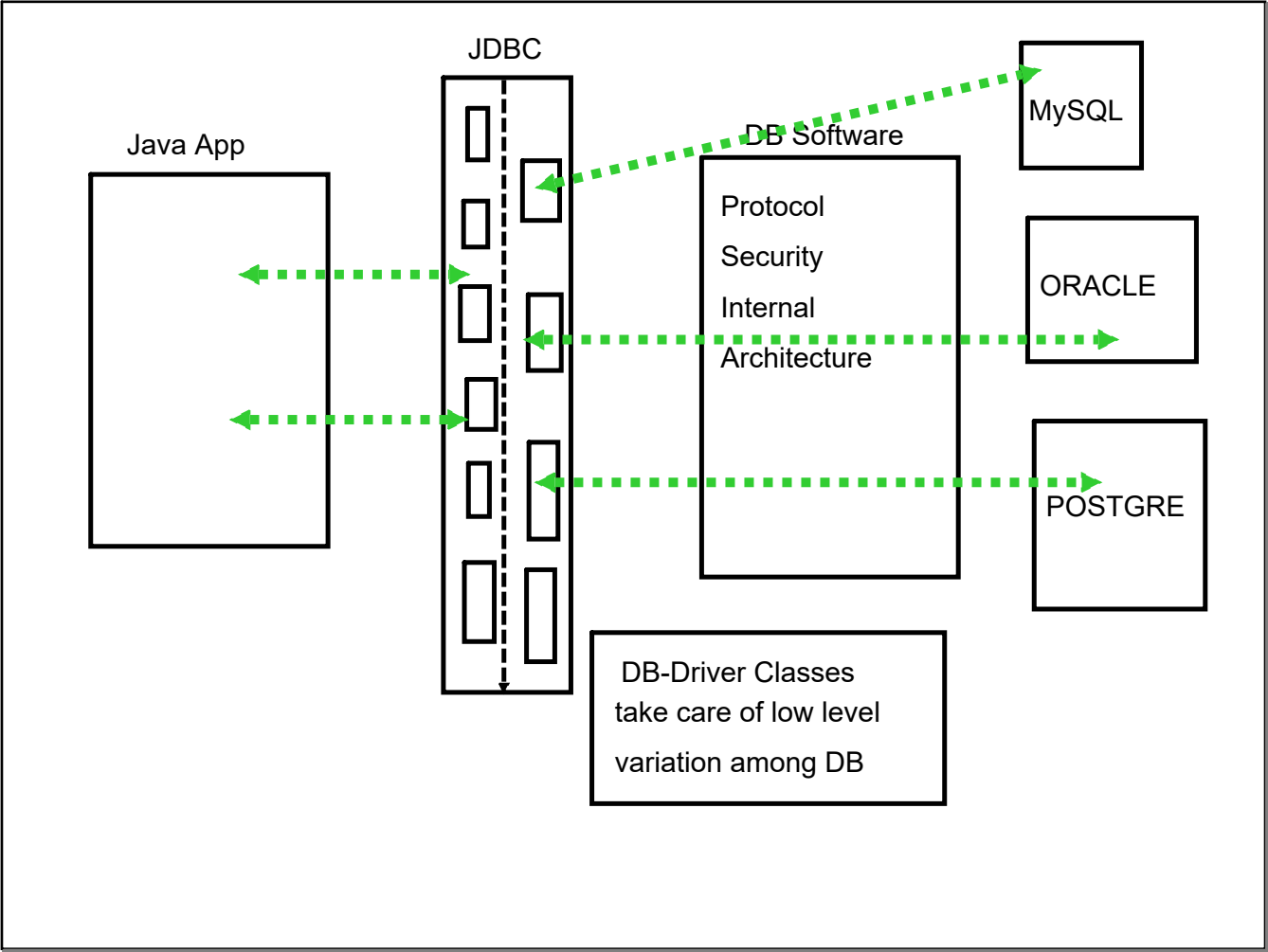/product?action=edit

<input type="text" name="email" />

M : Model : request/session/context

V : View : JSP/HTML

C : Controller : Servlet

Separation of concern

JDBC : Java Data Base Connectivity

Phase1

Java App

JDBC

DB Software

Protocol

Security

Internal

Architecture

MySQL

ORACLE

POSTGRE

DB-Driver Classes
take care of low level
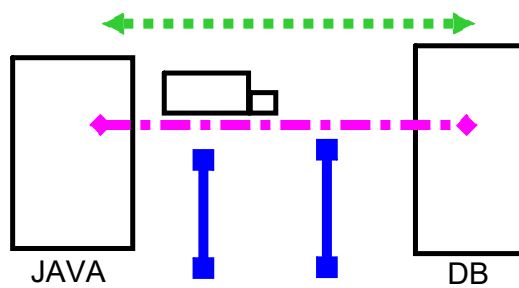variation among DB

Steps :
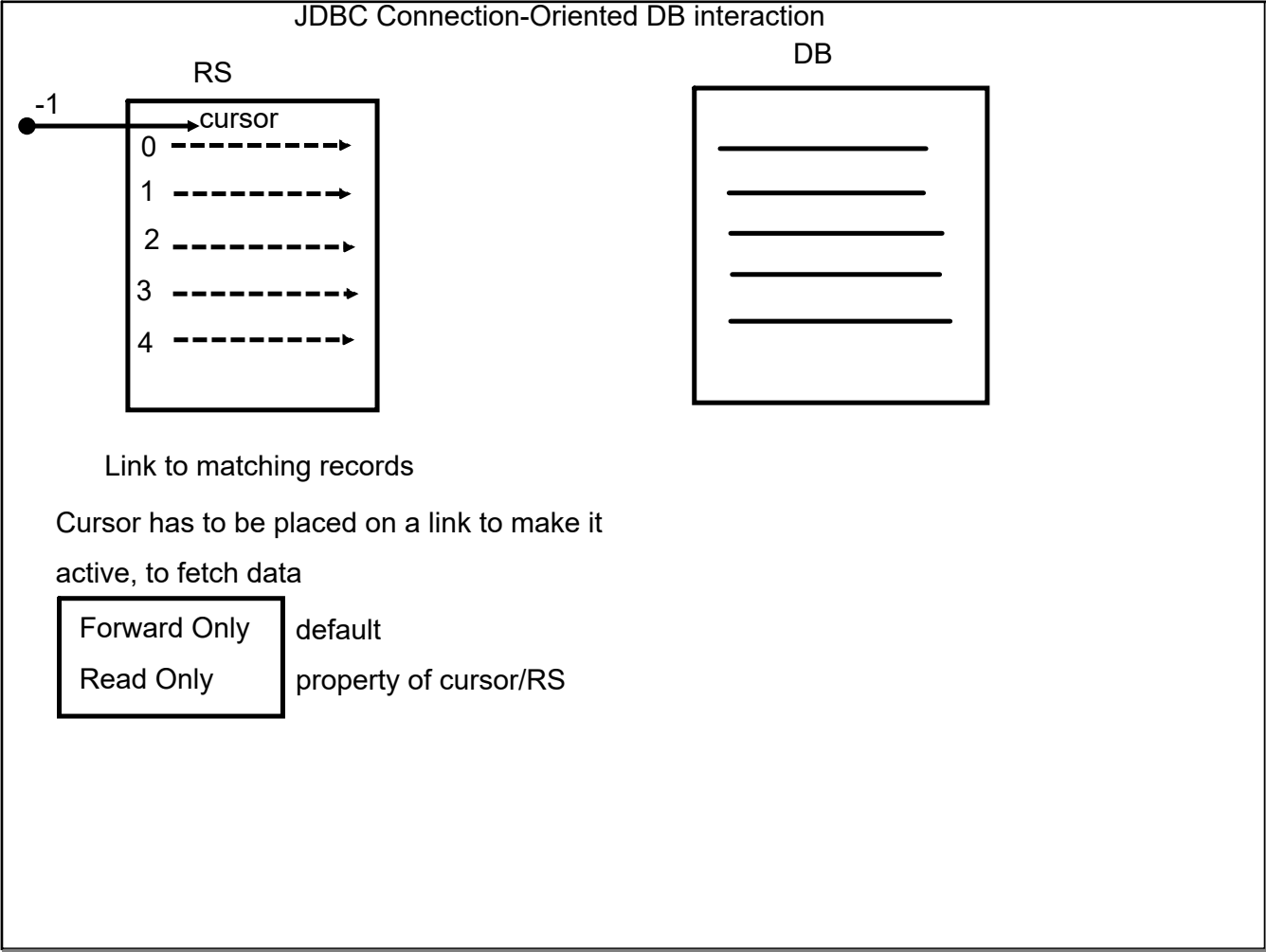1. <u>Load the driver class of that type</u>
2. <u>Create a connection</u>
3. <u>Create a statement</u>
4. Fire the query

JAVA   DB

MySQL : mysql-connector-java.jar

Load a Java Class in memory

Phase1

JDBC Connection-Oriented DB interaction

RS

DB

-1

cursor

0

1

2

3

4

Link to matching records

Cursor has to be placed on a link to make it

active, to fetch data

Forward Only   default

Read Only   property of cursor/RS

Phase1

Batch Processing

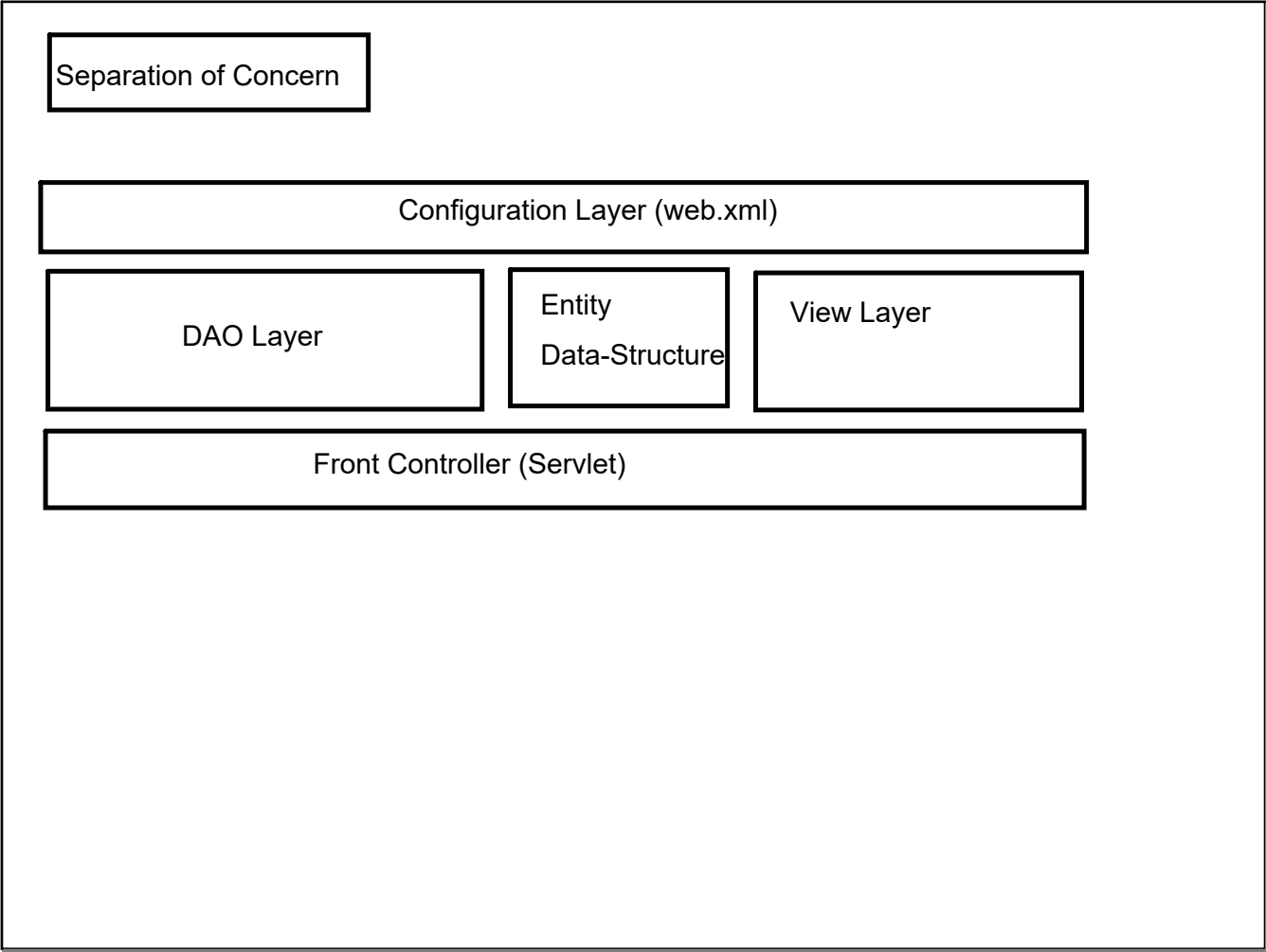　　# Allows to create a batch of queries

　　# Allow to run them at one go

Transactional Mode

　　# Group of query :

Either all of them shall execute or none of them, with the provision of rollback

A　　　　　　　　B

Separation of Concern

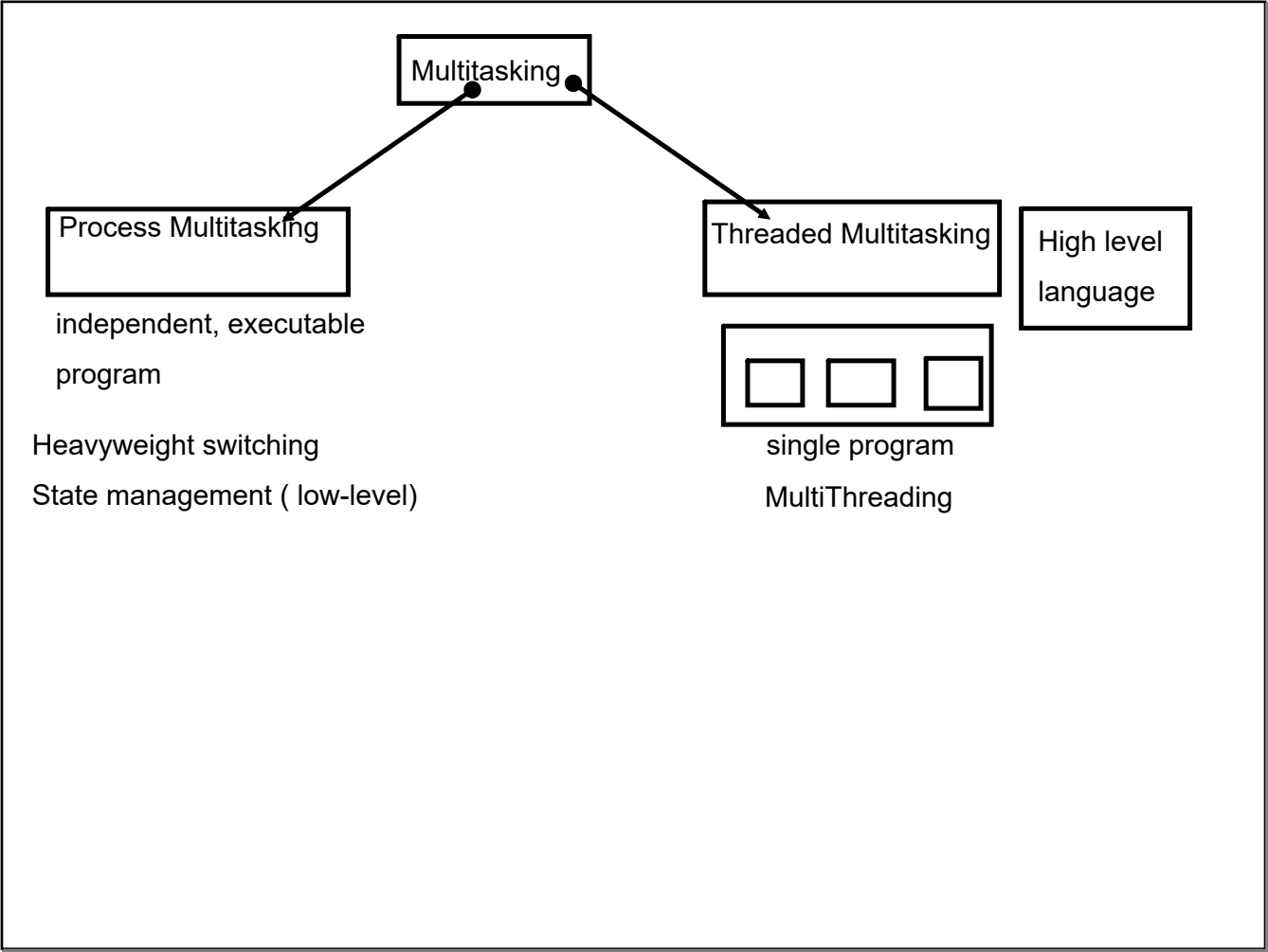| Configuration Layer (web.xml)		
DAO Layer	Entity Data-Structure	View Layer
Front Controller (Servlet)		

Phase1

Multithreading in Java:

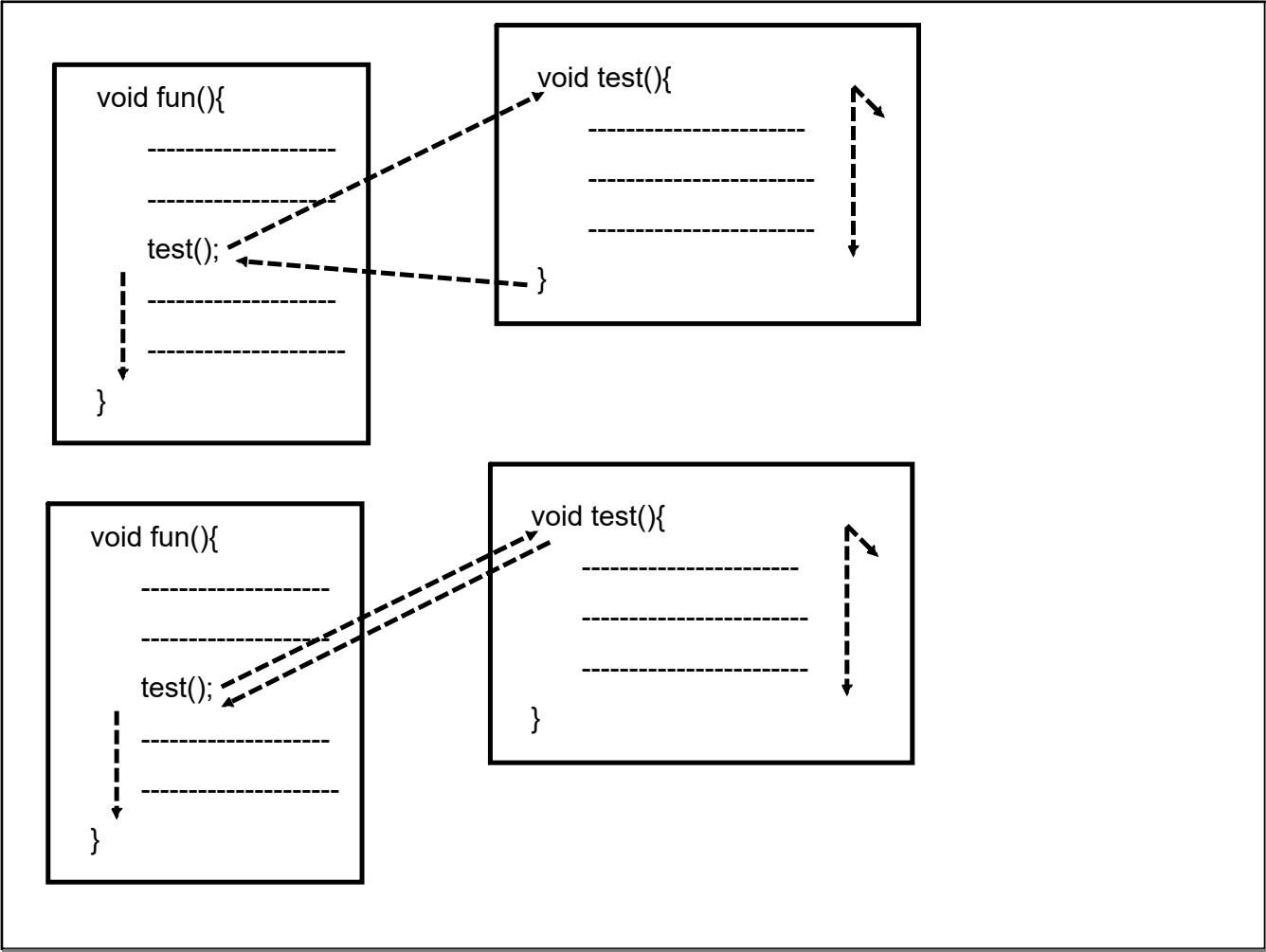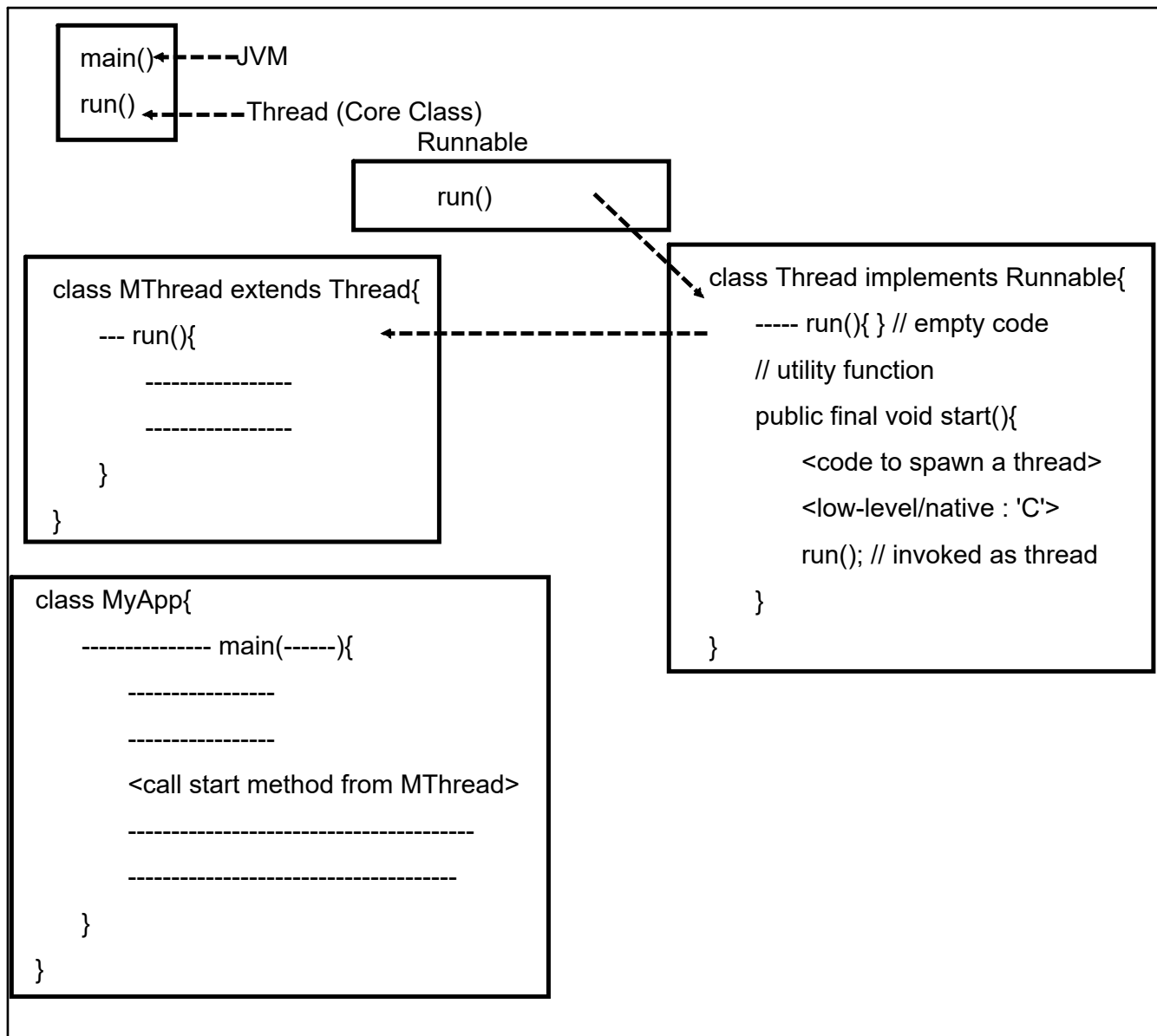Running more than 1 functions simultaneously in async manner

Multiprogramming :

 Multitasking : in interleaved fashion (time-sharing) : not parallel

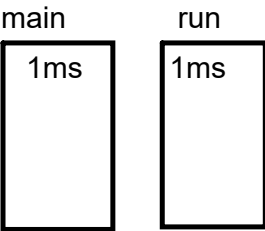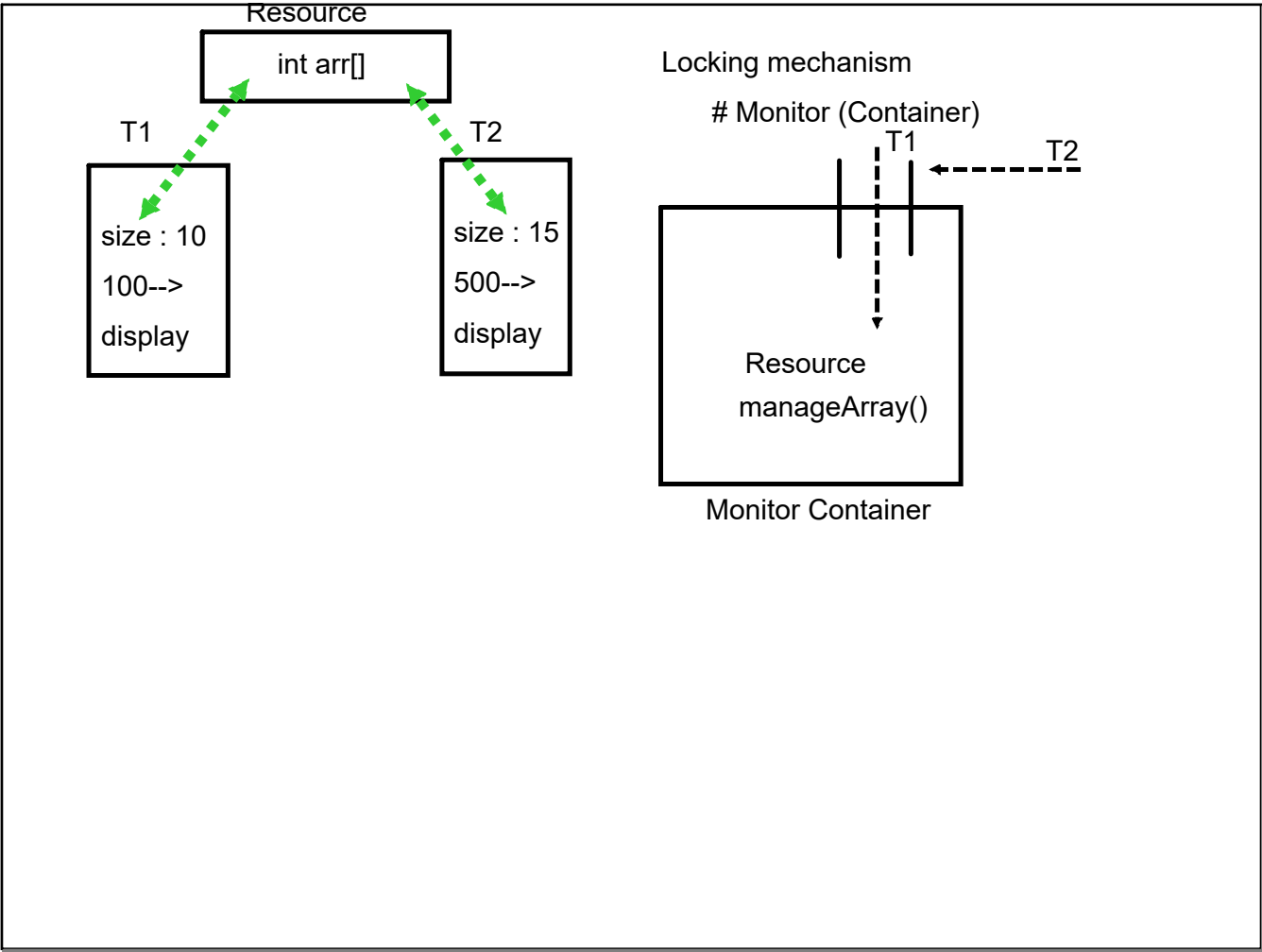MultiProcessing : parallel / batch processing

Multitasking

Process Multitasking

independent, executable

program

Heavyweight switching

State management (low-level)

Threaded Multitasking

High level language

single program

MultiThreading

Phase1

```
main() ◄ - - - - JVM
run() ◄ - - - - Thread (Core Class)
```

```
Runnable
┌─────────────────────────────┐
│          run()              │
└─────────────────────────────┘
```

```
class MThread extends Thread{
    --- run(){
        ----------------
        ----------------
    }
}
```

```
class Thread implements Runnable{
    ----- run(){ } // empty code
    // utility function
    public final void start(){
        <code to spawn a thread>
        <low-level/native : 'C'>
        run(); // invoked as thread
    }
}
```

```
class MyApp{
    -------------- main(------){
        -----------------
        -----------------
        <call start method from MThread>
        ----------------------------------------
        ------------------------------------
    }
}
```

Time Slice and the task alloted to a thread

main   run

main	run
1ms	1ms

Output device are slow

buffer memory

mixing in buffer memory

Sharing the common resources

     # Data in-consistancy

Phase1

Resource

| int arr[] |

T1

| size : 10
100-->
display |

T2

| size : 15
500-->
display |

Locking mechanism

Monitor (Container)

T1

T2

| Resource
manageArray() |

Monitor Container

manageArray()

monitor

changeArray()

monitor

manageArray()

changeArray()

Single Monitor Box for entire object

Phase1
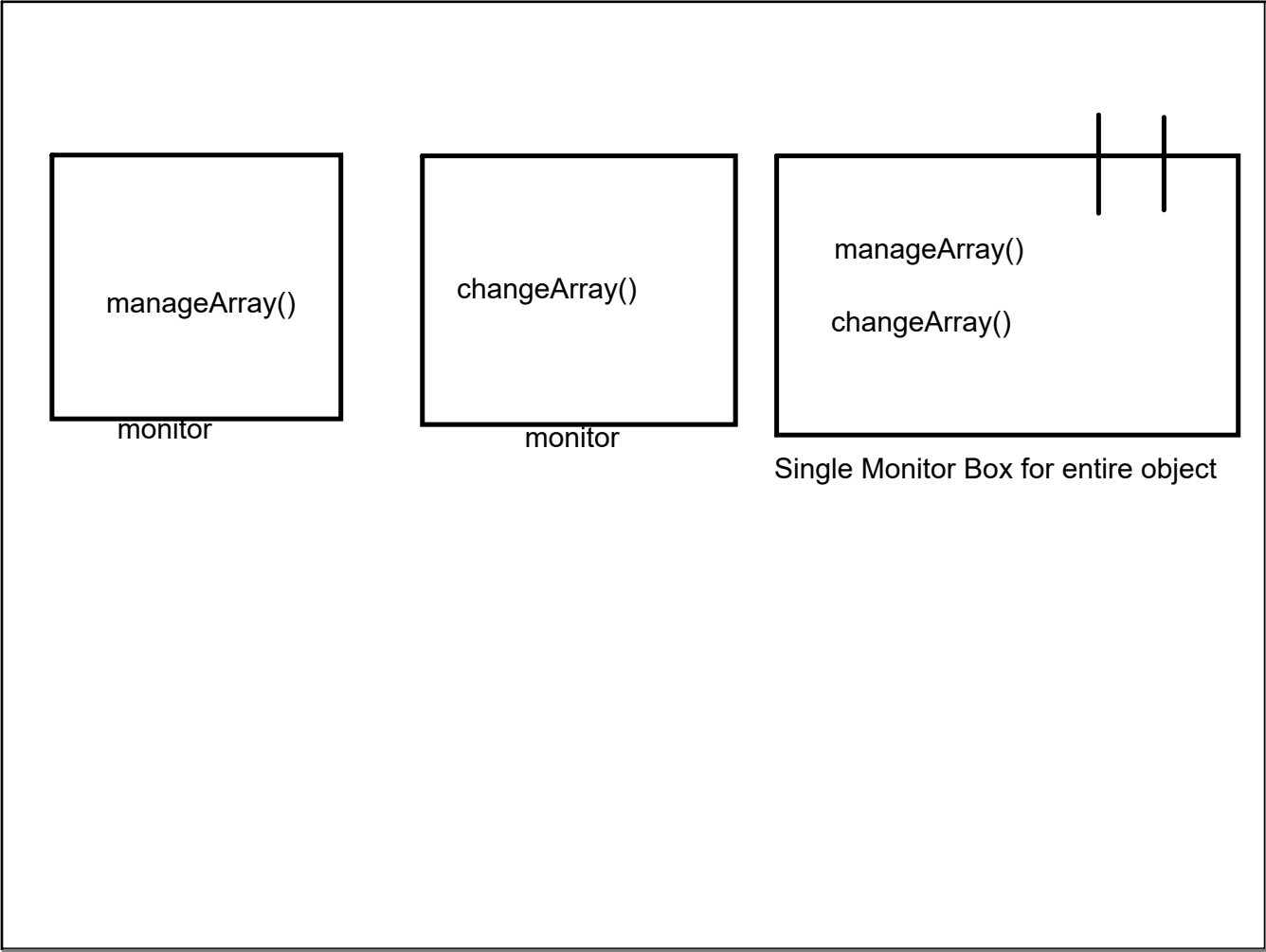
Functional Locking

 # Resource does the locking , not the client/thread
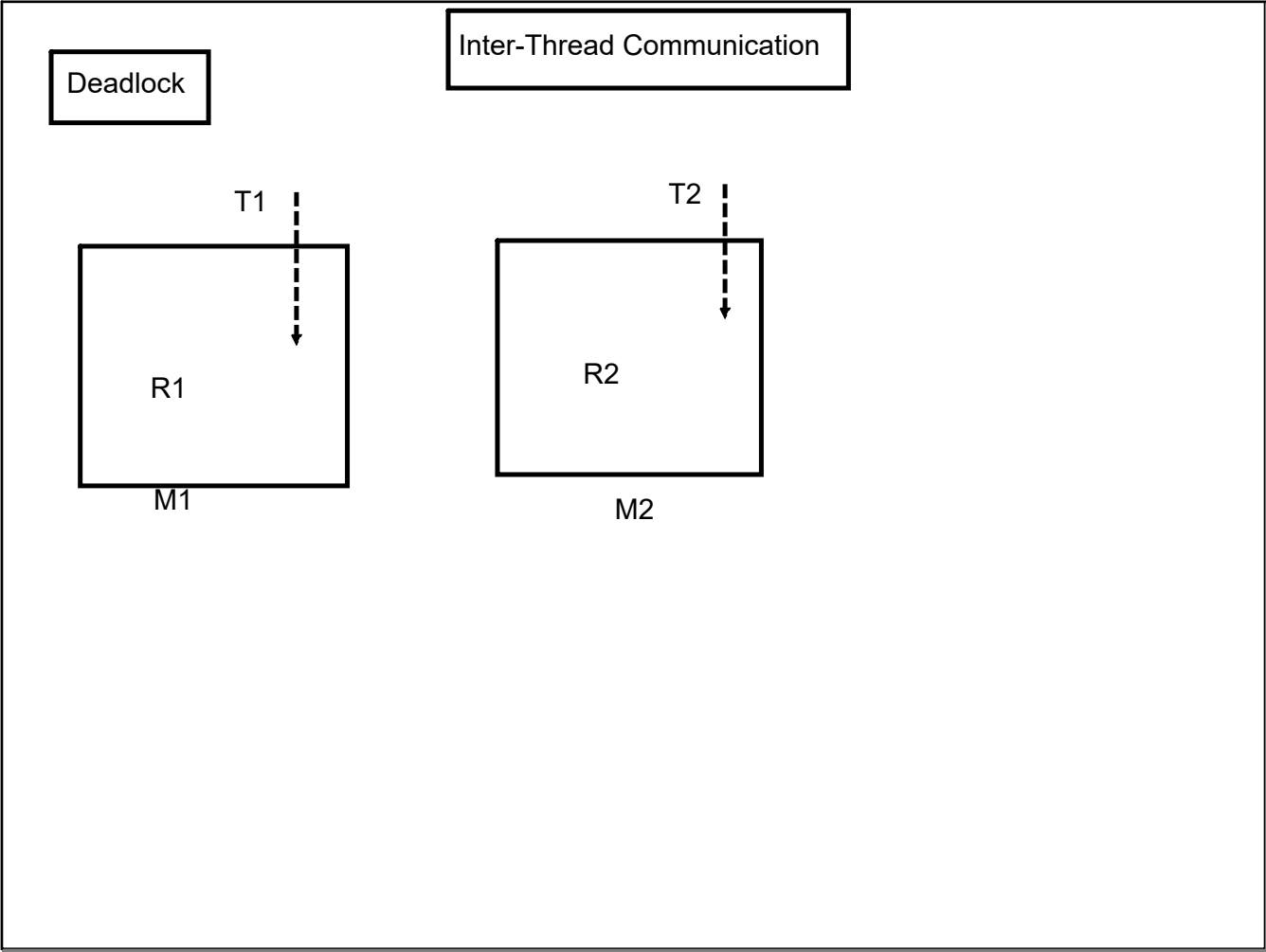
 # Permanent

Data Locking:

 # Cannot lock permanently, time frame based (block based)

 # Client/Thread can lock data
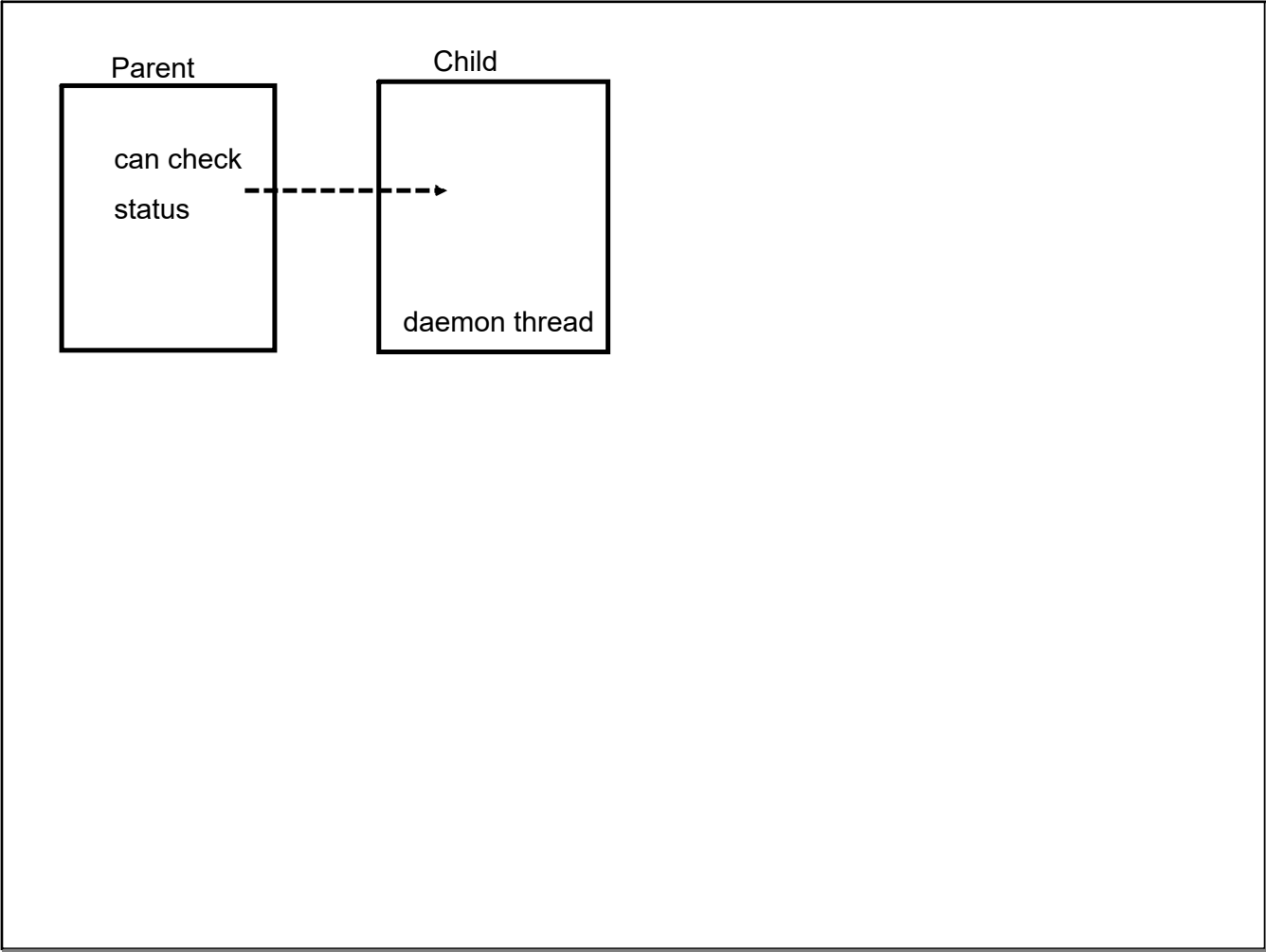
 # Lock Entire Object

Phase1

Deadlock

Inter-Thread Communication

T1

T2

R1

M1

R2

M2

Producer-Consumer : Classical Multitasking Use-Case

overflow   underflow

Producer   BOX   Consumer
sync   1

wait
notify

Inter-Thread Communication
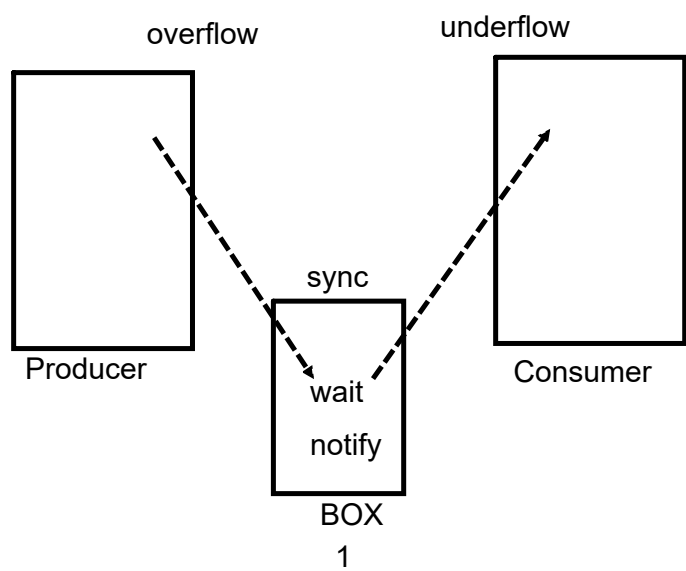
1. Different Speed
2. Box is of finite capacity

Phase1

Parent

can check
status

Child

daemon thread

Project Management Tool :

    Maven, Gradle, Ant...

Maven :

    1. Standardized Project Structure

    2. Dependency Management

Local Repository (.m2) c:/users

Project Object Model

App

pom

List

Maven : need to be installed

Eclipse IDE : plugin : m2eclipse

Cloud Repository

Phase1
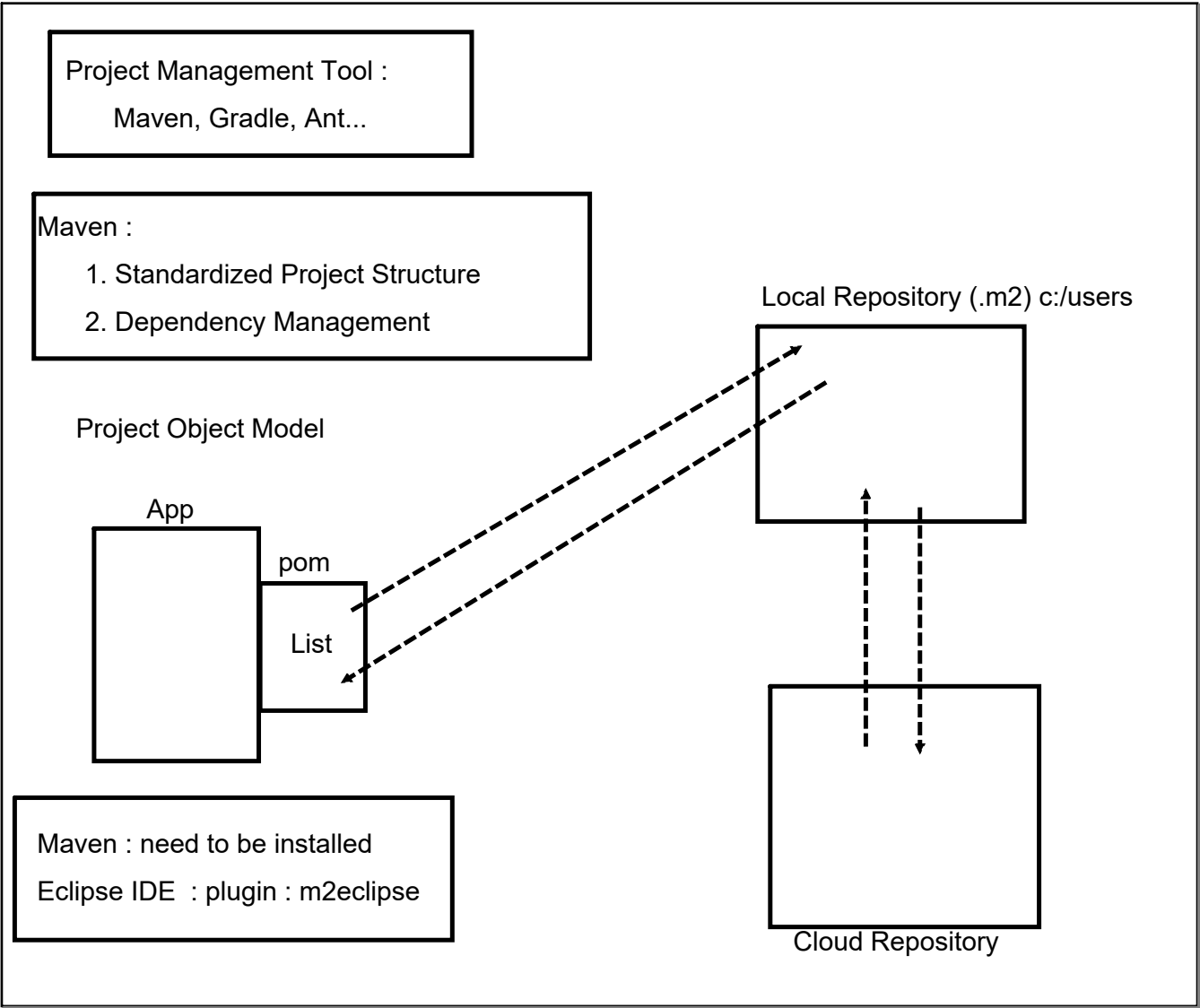
GAV Coordination

    GroupId : organization details (reverse domain)

    ArtifactId : Project name

    Version :

Some Configuration

    1. Compilation support for JSP

    2. Changing java 1.5 to 1.8

=> servlet-api

=> jsp-api

=> mysql-connector

Phase1

JUnit :

    # Allows to write test cases

    # Organize

    # Test Runners

Setup

Invoke the resource under test real/mock

Assert (test the expectation)

OO Design Patterns

Create an Object :

    <u><class> ref = new <class>([param]);</u>

Builder Patterns

    Complex Object : constructor : 10 param

    WE don't want to create an object at one go

    Instead : piecewise construction of object

    Builder allows to create API for step by step object creation

Phase1

Object Construction / Constructors can be confusing

Object construction cannot be piece-by-piece always

    we outsource object creation to separate method/class factory

Factory

Singleton Design Pattern

    Only one Object of that class is ever created

        Object Creation and exposing object will be done by class