

Algorithm Classes

1. Arrays.
2. Collections.

@ Provide common functionalities can be applied on Array `int []` and Collection

`Collections.concurrentList();` // Thread safe equivalent



Java - 8

Functional-Programming

functional interface

default methods

static methods

lambdas

streams

methods references

DateTime API

Optional

Nashorn engine (javascript engine)

Extension in collection API

Traditional :(Pure OOPs) : Imperative

Functional : Declarative

Imperative :

How : focus

Pure OOPs

Embraces data mutability

Declarative

What : focus

Functional Programming (pure function)

Data immutability

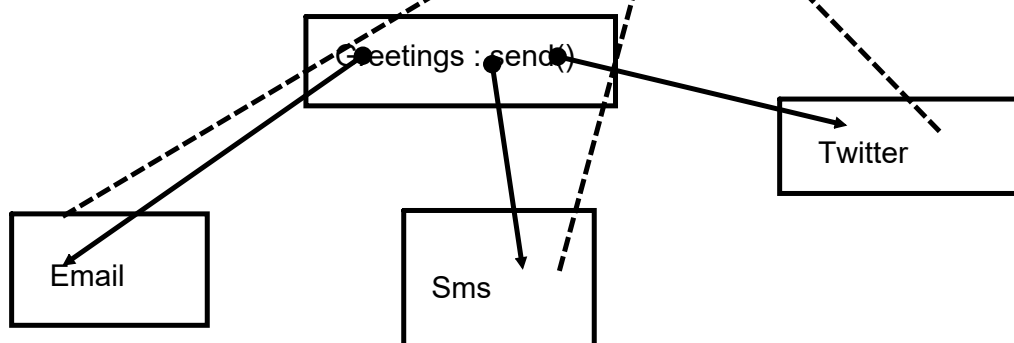
SQL style

Interfaces:

can have functions with definitions (default methods)

static methods with definitions

```
public void conveyMessage(String message, Greeting){  
    // call a single method send()  
}
```



Lambda Syntax for pure function

1. not have any accessibility modifier (nothing related to class)
2. Do not any name (anonymous function)
3. no return type
4. no param type
5. (<param>) -> { <definition> }

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

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

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

```
void fun(String str1){  
    // only single instruction  
}  
str1 -> single instruction
```

```
void fun(String str1){  
    // instruction  
    return a;  
}
```

```
str1 -> {  
    // instruction  
    return a;  
}
```

```
int add(int a, int b){  
    return a+b;  
}
```

```
// for single inst not bounded in braces return is default associated  
(a, b) -> a+b;
```

interface <reference> = <lambda expression>

only when interface is functional interface :

contains only one abstract methods

can have any number of default/static

refrence of an interface can refer to only those lambda expression whose signature matches with the only abstract method inside the interface

Runnable :

Comparator

Comparable

few protoypes have been identified which are common in use

api : functional interface

Consumer : void accept(<>), BiConsumer [two param], IntegerConsumer()

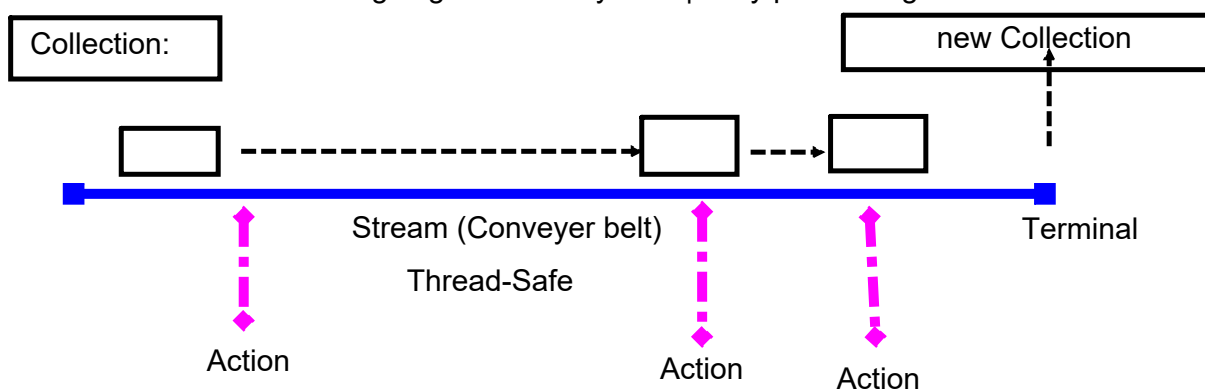
Predicate : boolean test(<>)

Function : <> apply(<>)

Supplier : <> get()

Streams : Safe/efficient way to process a collection

Not a data-structure/ not going to store any data | Lazy processing model



Stream method

1. action <intermediary methods>
2. Terminal method (stops the stream) cannot add any new activity after it

Stream are not going to initiate if no terminal activity is there

stream : external mutable resource

inherently complex (JVM)

want to use calculator to perform addition of stream of number

1 2 3 4 5..... result : mutable (not thread safe)

1 Data inconsistency
3

25 : result 5,6,7

result = result + v

result = 25 + 5

result = 25 + 6

Unboxing : buffer

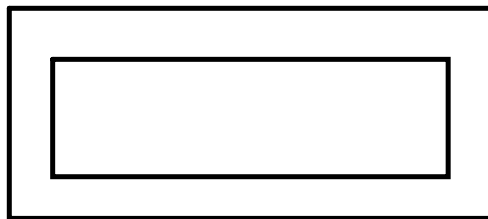
finalize() // gcm

Optional : to avoid null reference exception

if any object being initialized by some external logic (method)

if no explicit validation code is added to check null

Optional



wrapped instance

forced to put validation

functionality to handle situation

pre jdk 8

Date

Calander

Time API :

java.time

LocalDate

LocalDate

LocalDateTime

Multi-Threading

Multi-Programming :

O/S to reside more than 1 exe program in memory , not necessarily executing them

Multi-Tasking :

O/S execute more than 1 program simultaneously , interleaved fashion (time sharing)

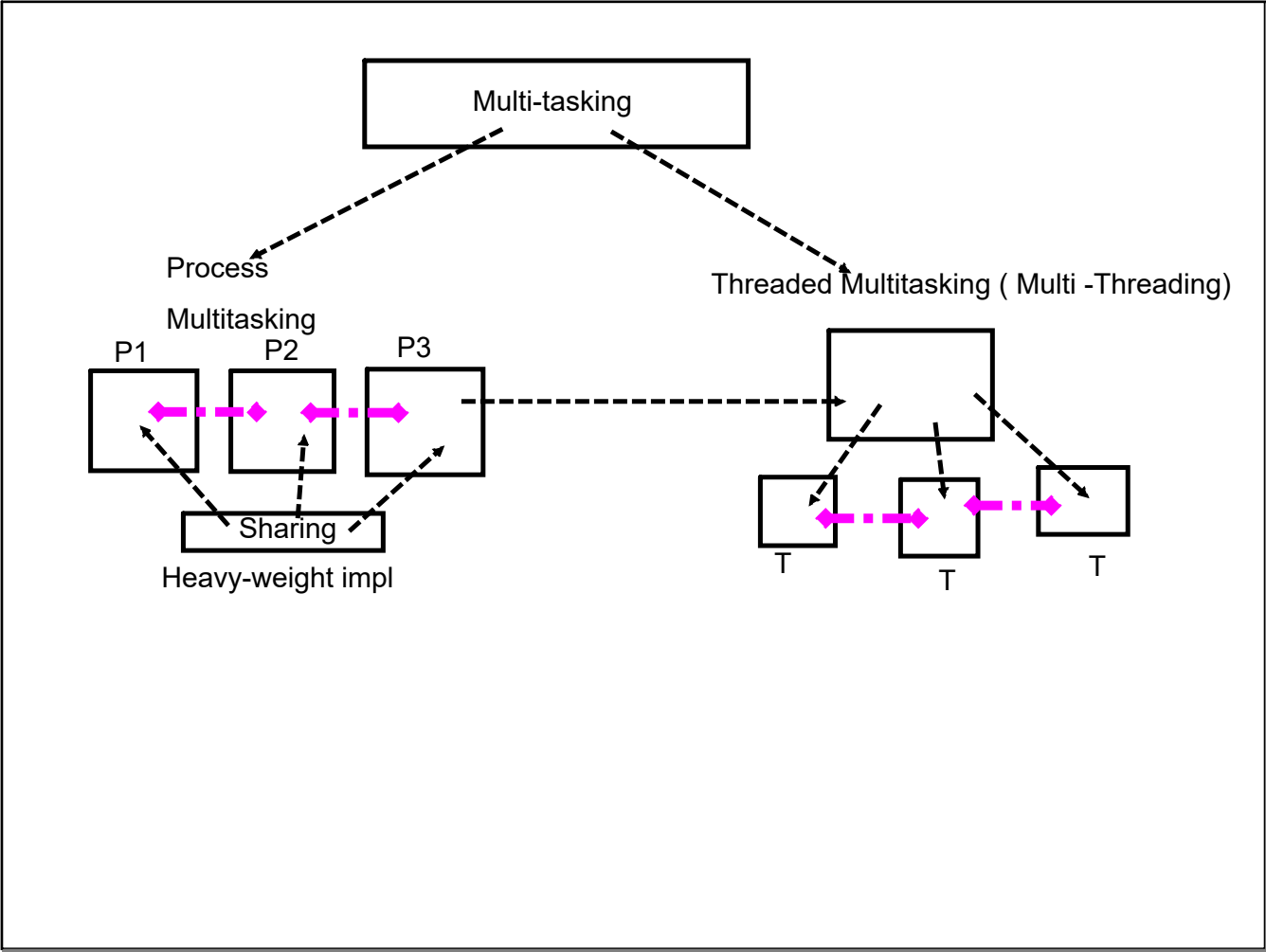
Multi-Processing

O/S to manage more than 1 processing units (parallel / array)

O/S

Multi-tasking :

multi-core processor : multi-processing

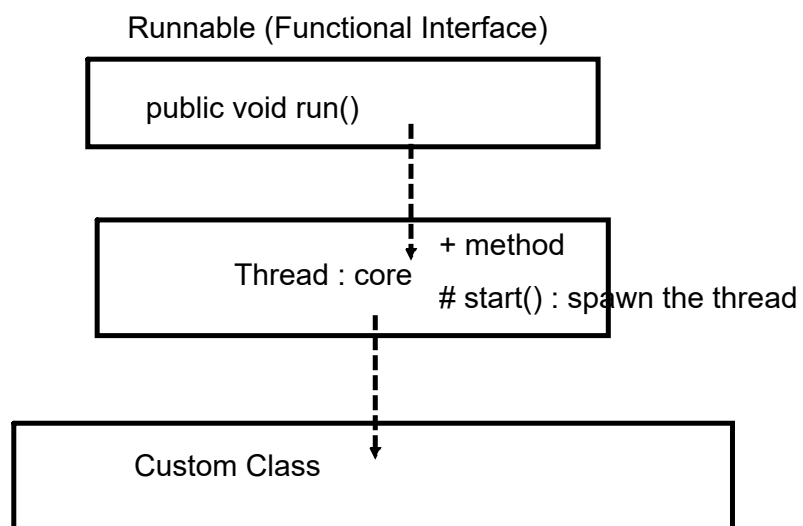


=> Thread class

=> Runnable

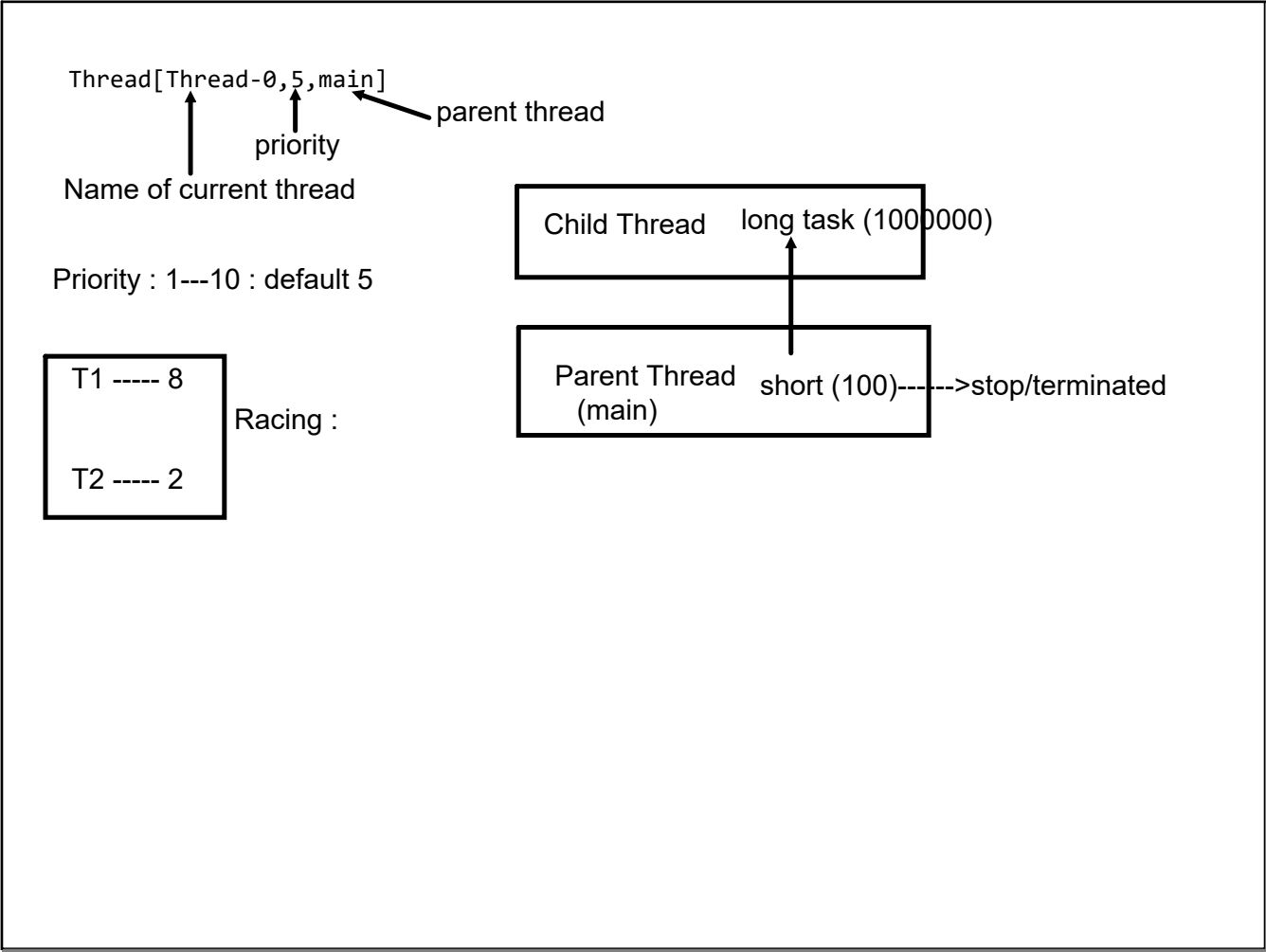
Sharable resources : data consistency

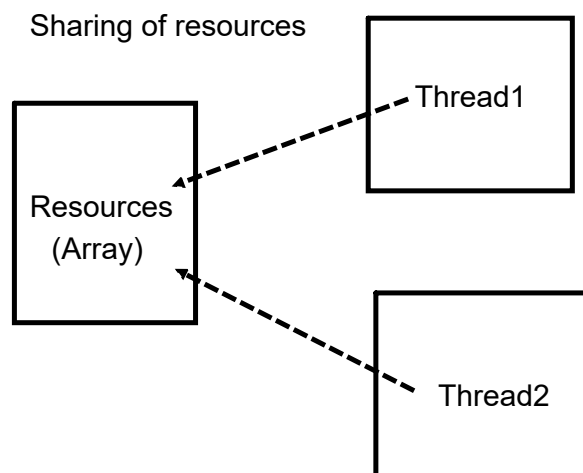
Inter-thread communication



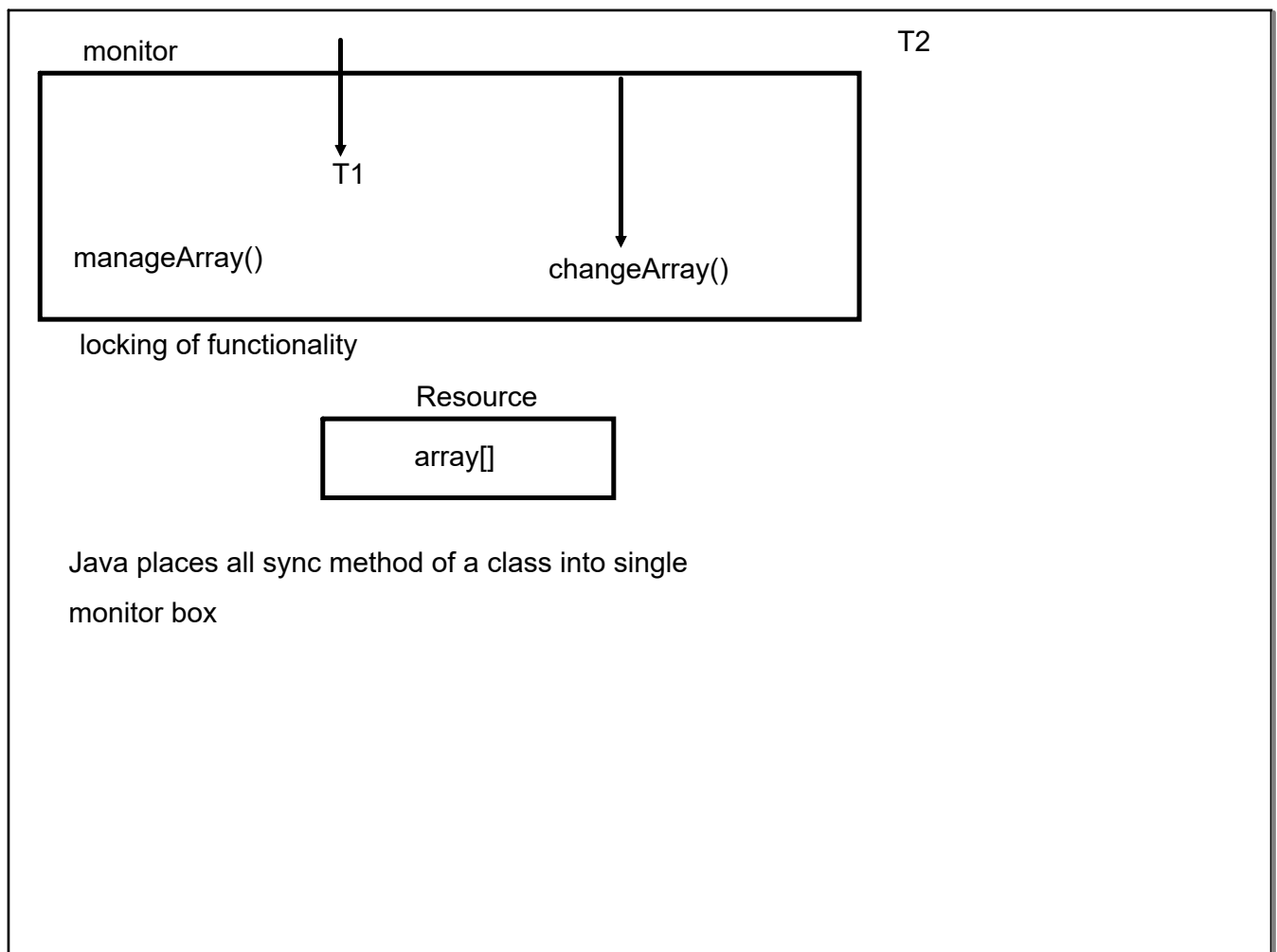
```
class Thread implements Runnable{  
    public void run(){} // empty  
    public final void start(){  
        // low level functionality to spawn a new thread  
        // native coding : C/C++  
        // call the run() : assigning thread capacity  
    }  
}
```

```
class MThread extends Thread {  
    public void run(){ // overriding run of thread  
        // thread activity  
    }  
}
```





Any method blocking the thread execution
will throw checked exception
InterruptedException



locking of functionality : sync method [permanent] : Resource owner

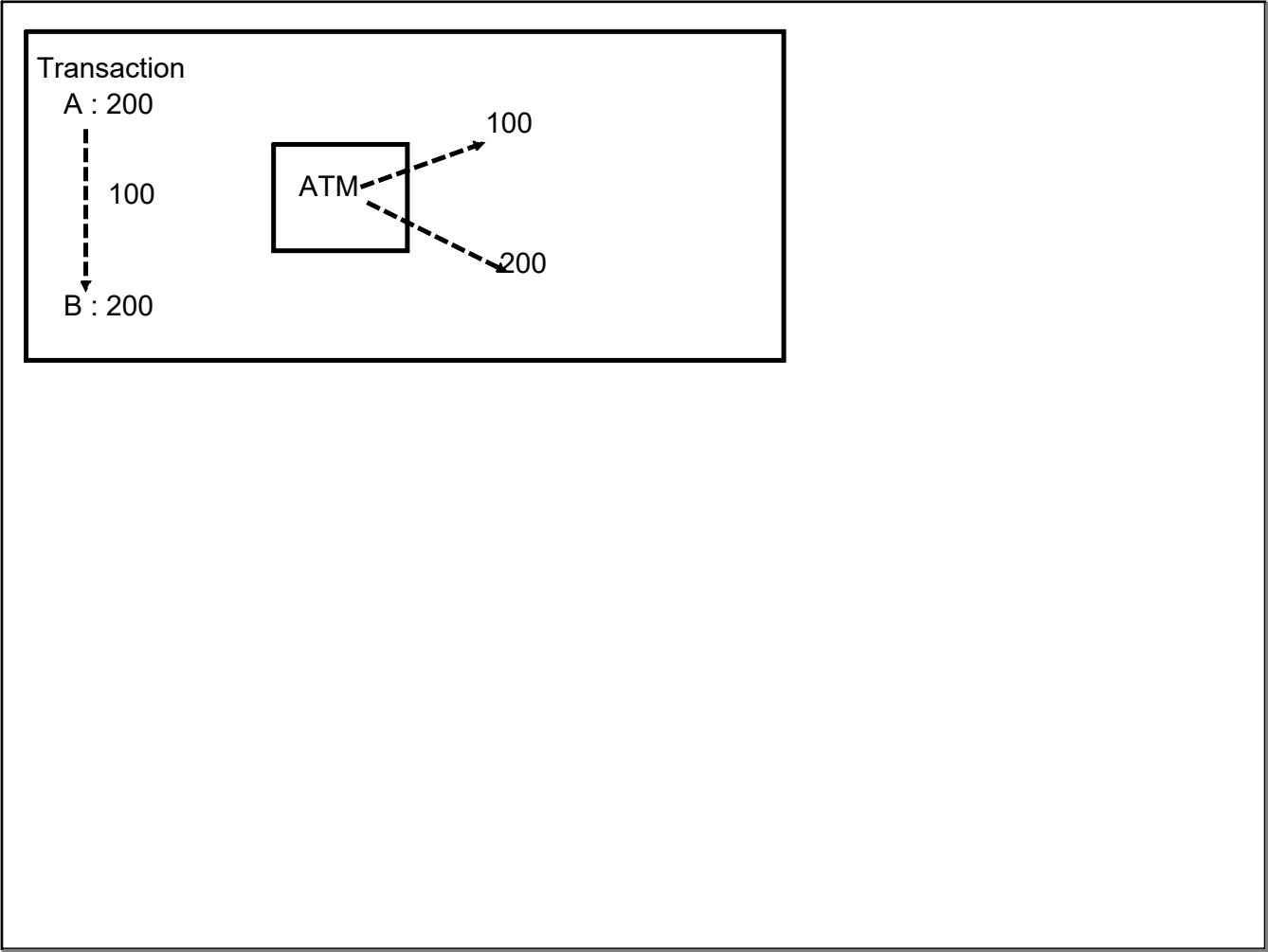
locking of objects : sync block [temp] : user

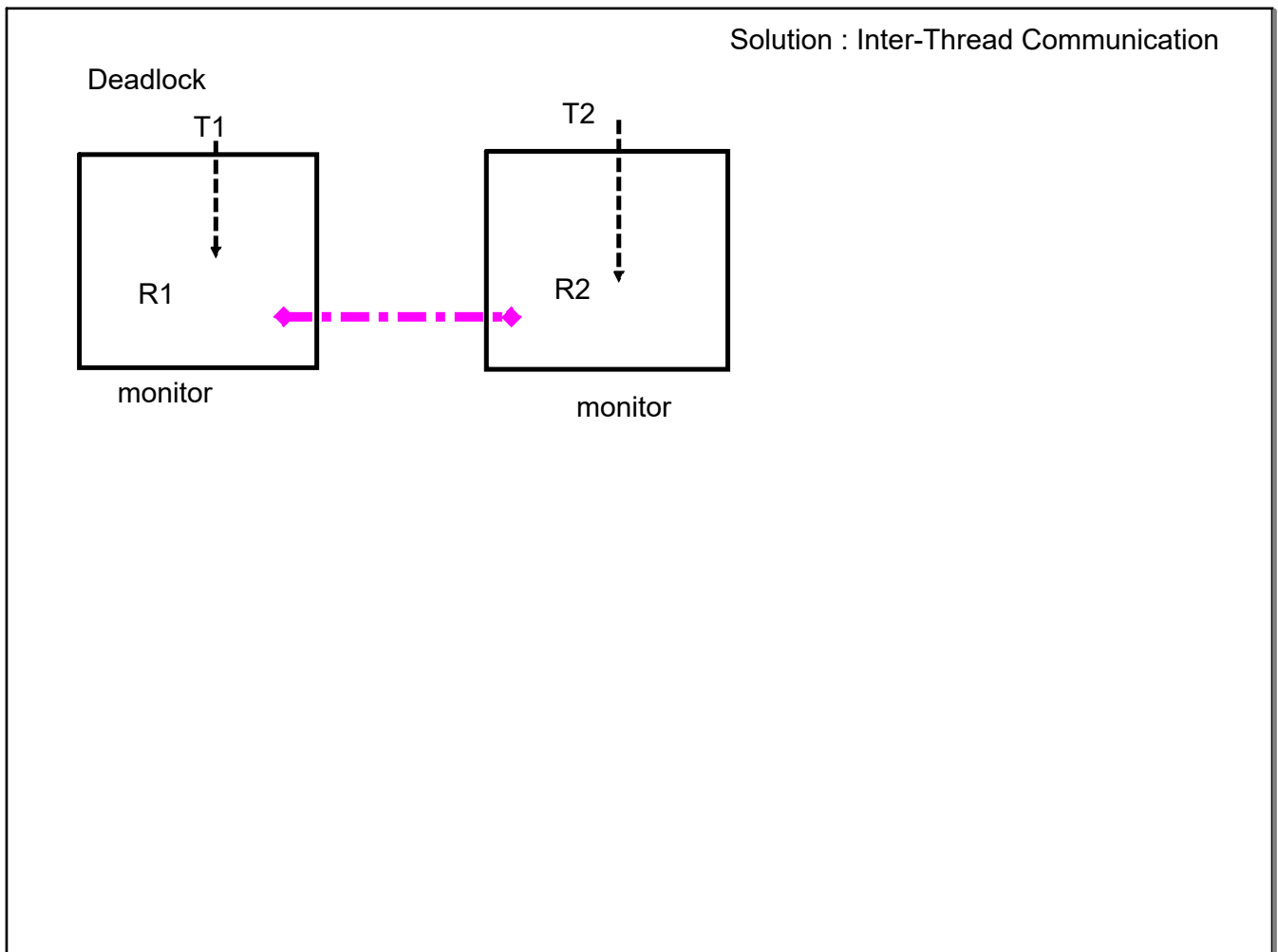
Concurrent-Collection API

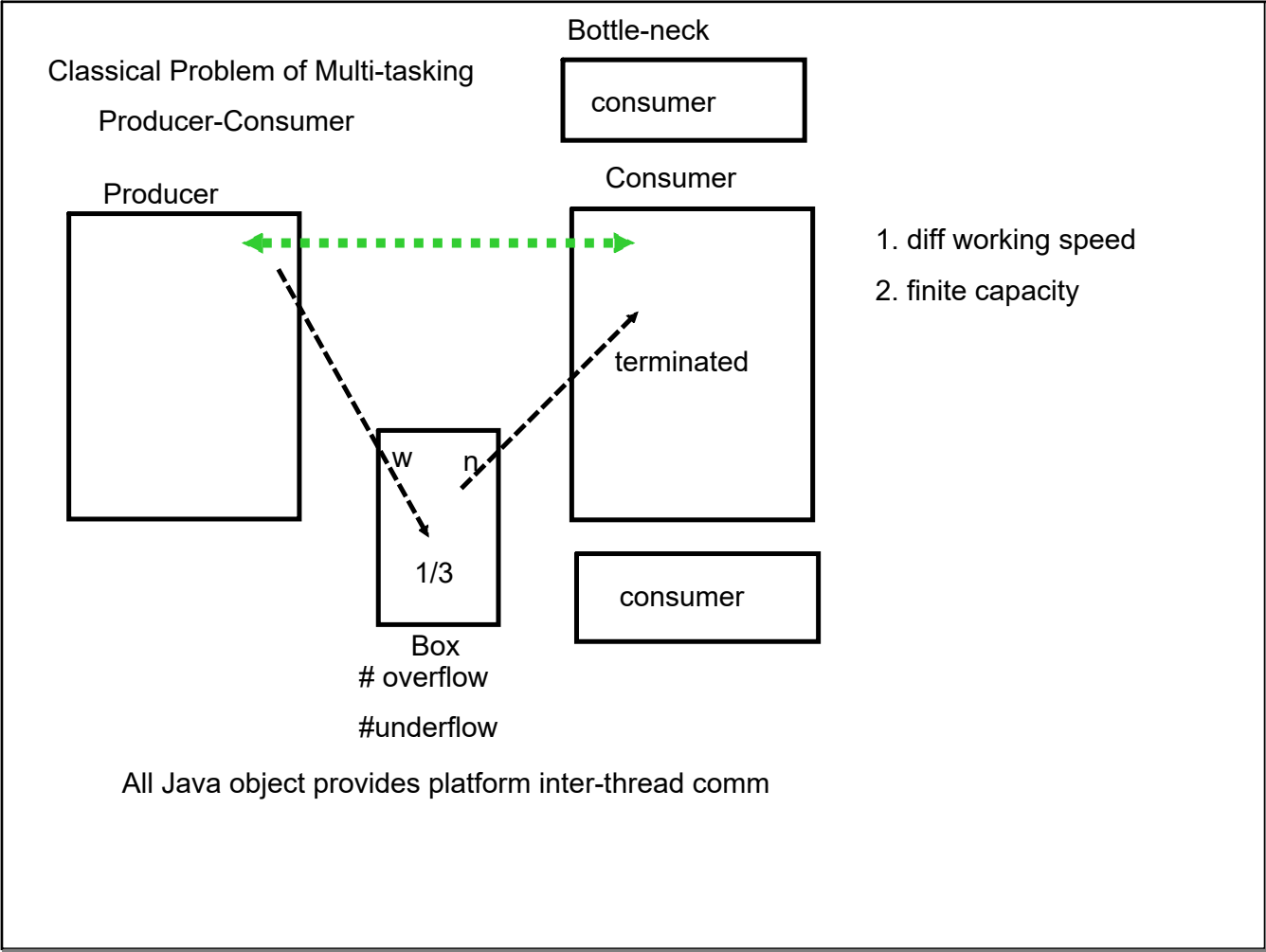
High level of concurrency with Thread Safety

synchronized : method/block : Wide Spectrum of locking

granular locking : ReentrantLock();

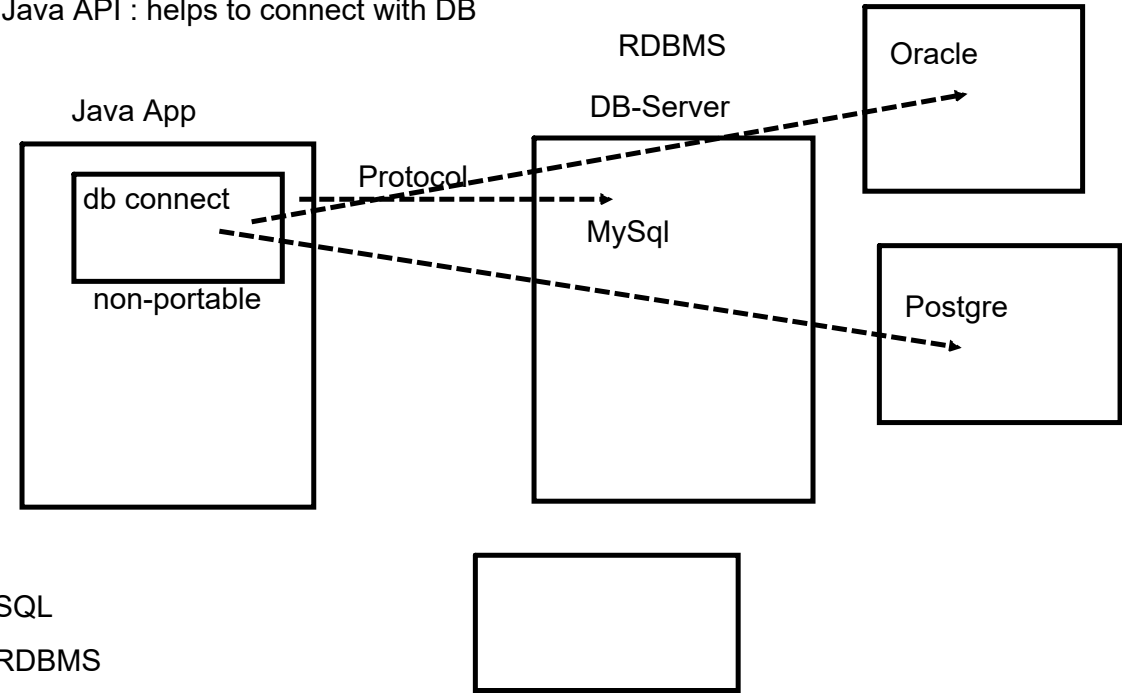


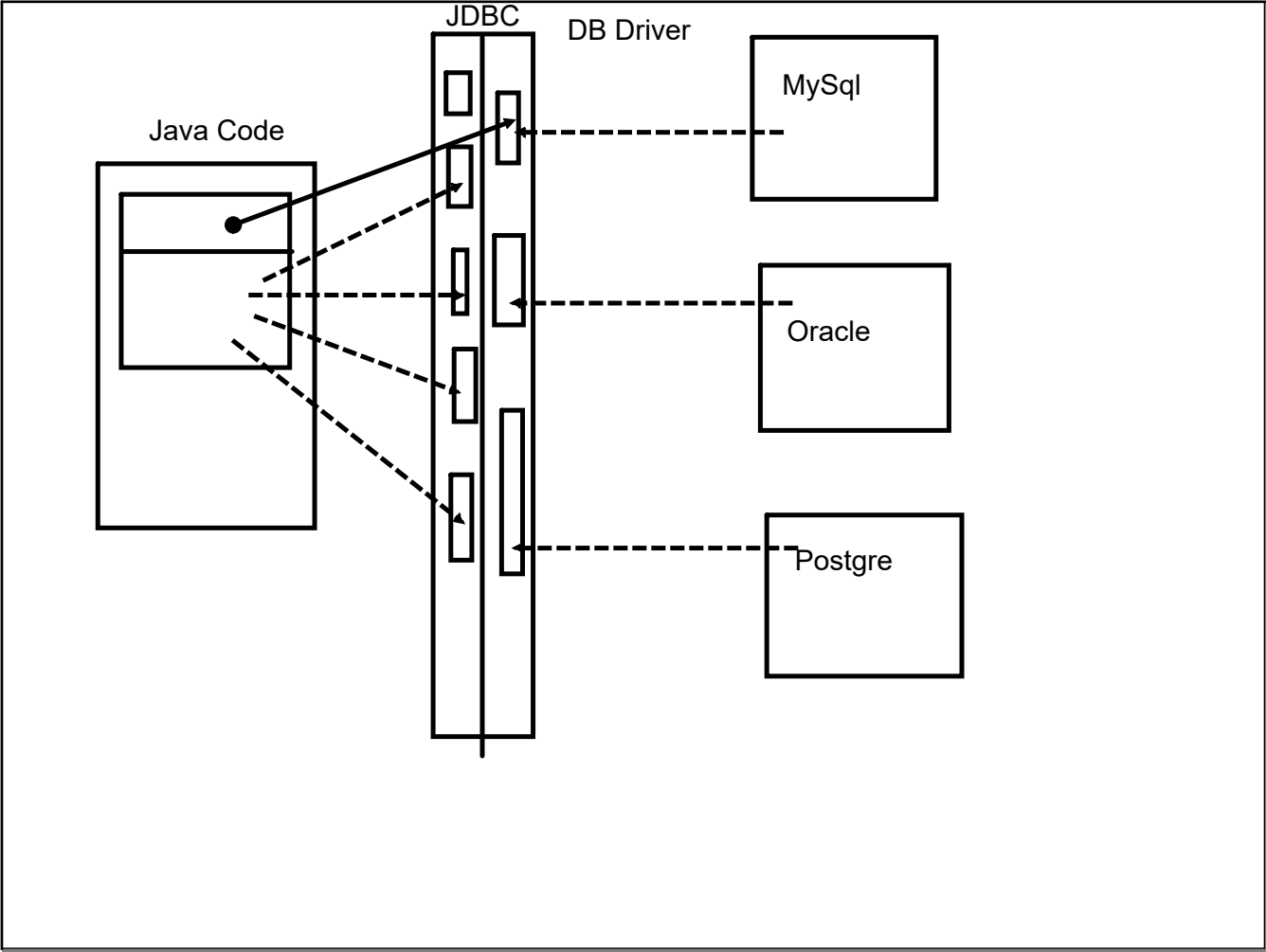


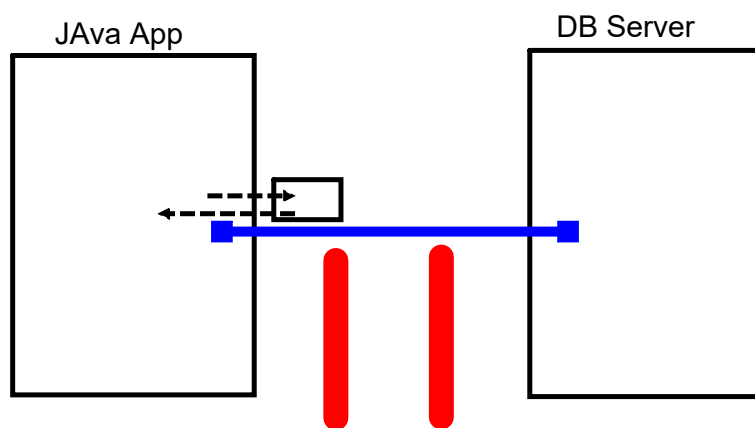


Thread Pool : Executor Framework

JDBC : implementation
Java API : helps to connect with DB

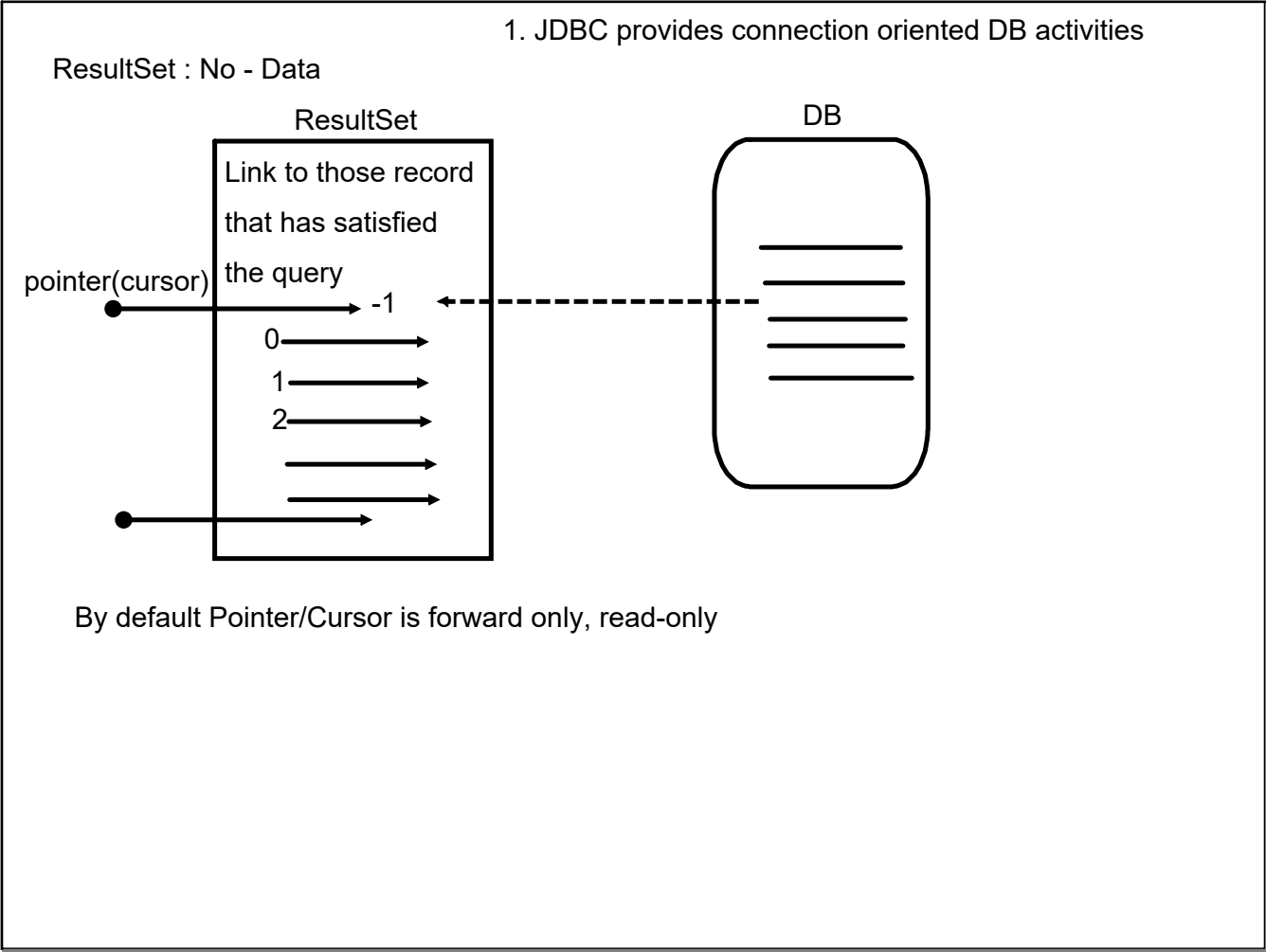


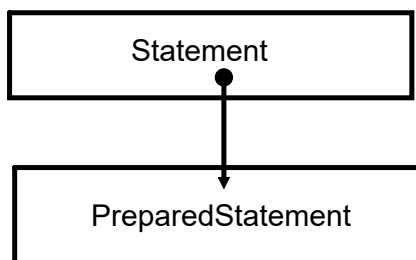




Steps:

1. Load the DB Driver
2. Setup a connection
3. Create a Statement





Avoid : SQL injections

Input from user

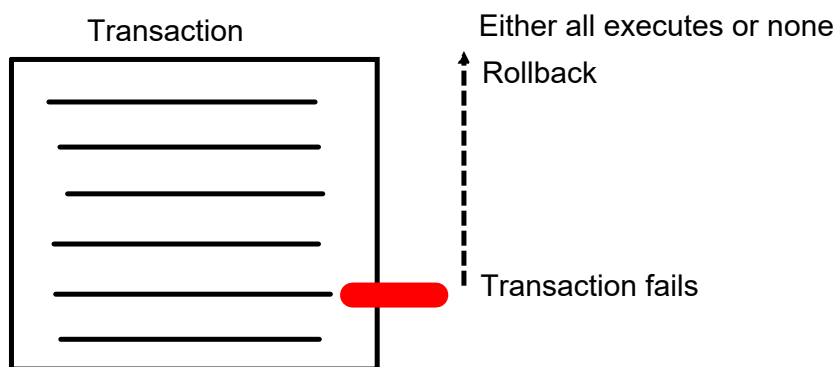
Batch Processing

Transactions

Java EE (Servlet Technology)

Batch Processing :

Transactions : Critical example of batch of queries



Views : Temporary status of Database

Java way to create web based Server resources

Server based API : can integrate with web server(Tomcat, Websphere..) and can deal with HTTP Request and Response

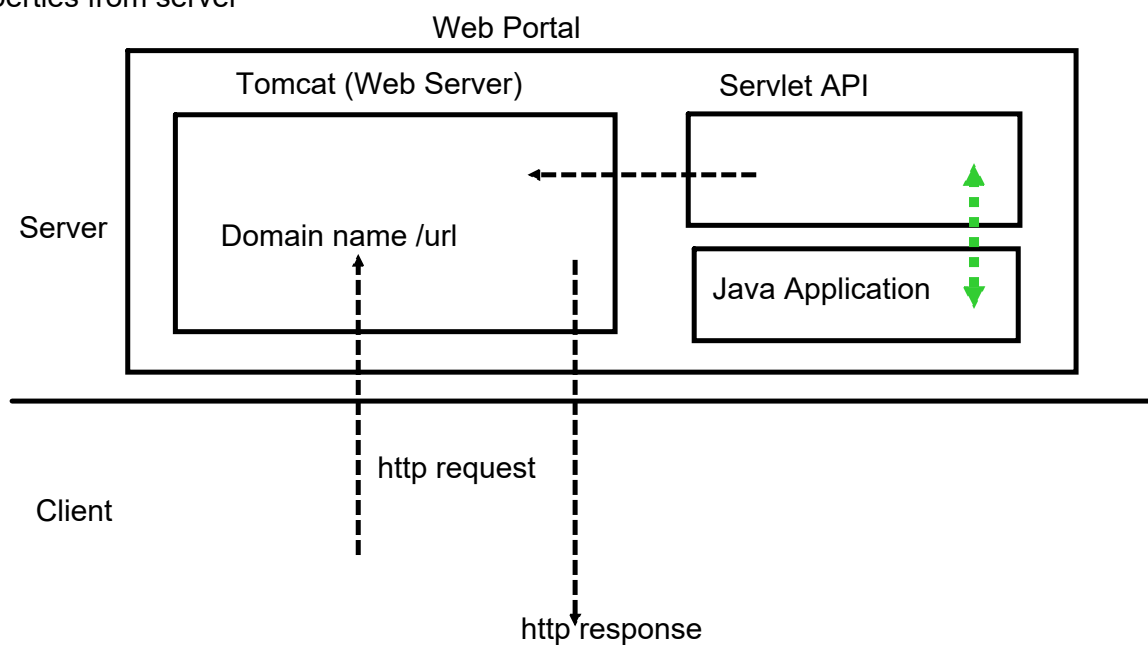
Servlet-API

Lot of frameworks uses this API

eg : Java EE, Spring

Servlet API :

It provides certain specialized classes that connect with webserver and can read the http properties from server



Core/Key class of Servlet-API :has complete access over Web Server

GenericServlet : old version : not able to differentiate between HTTP Verbs (GET,POST,PUT,DELETE,PATCH...)

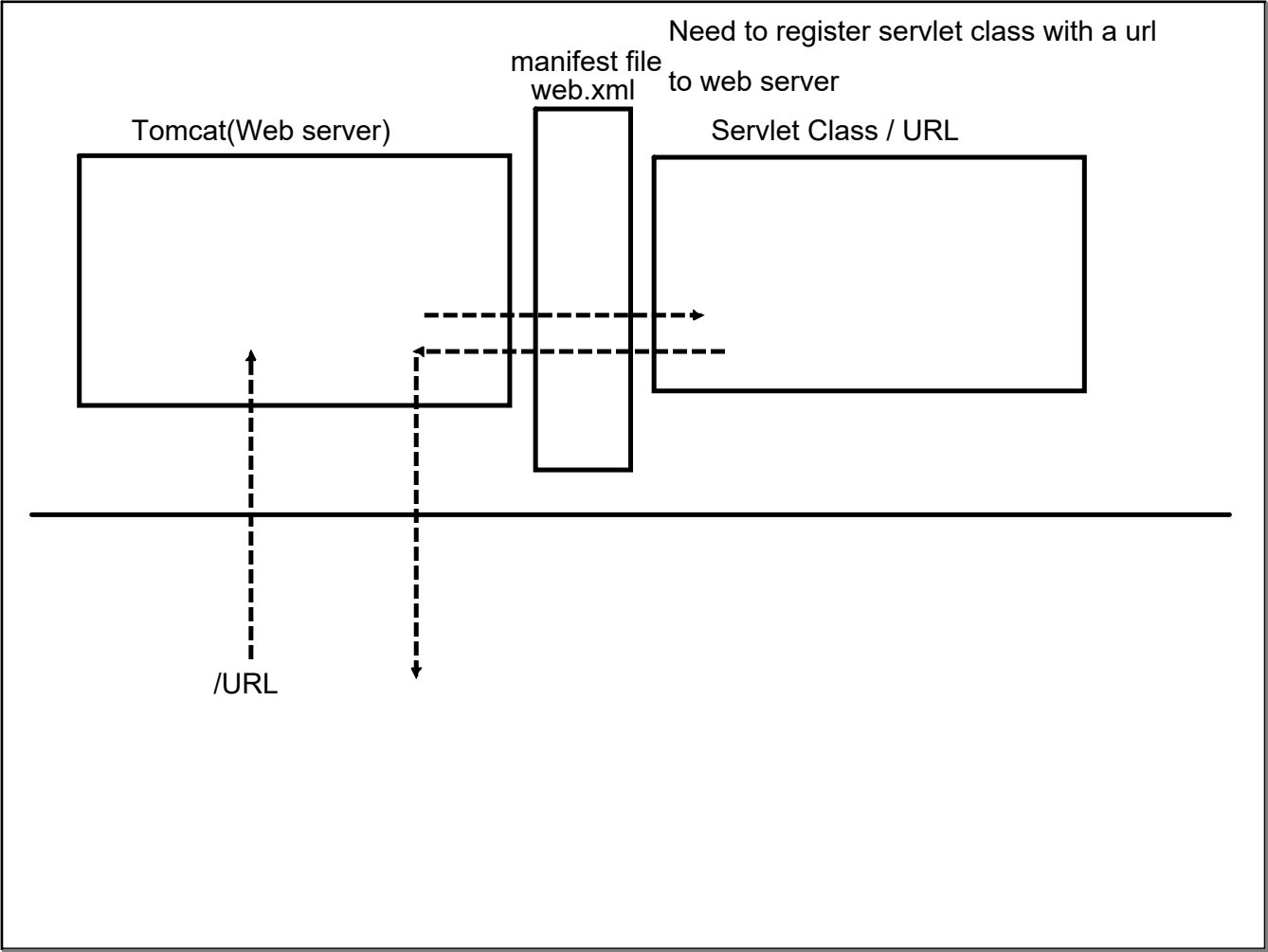
HttpServlet : new version (extention) : can identify

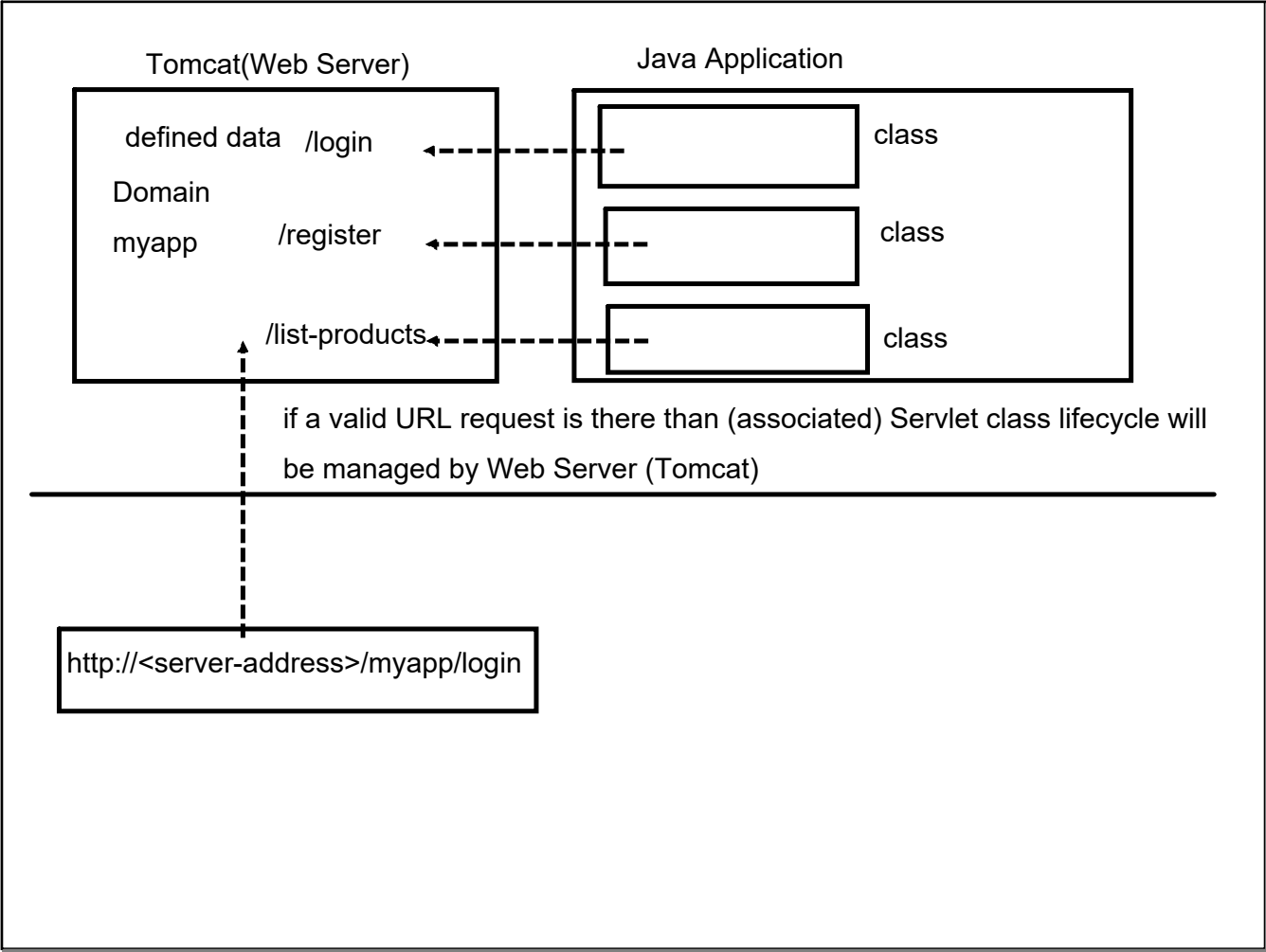
// Servlet class : part of the server

```
class <Custom Class> extends HttpServlet{
```

```
    // business logic and use the server info
```

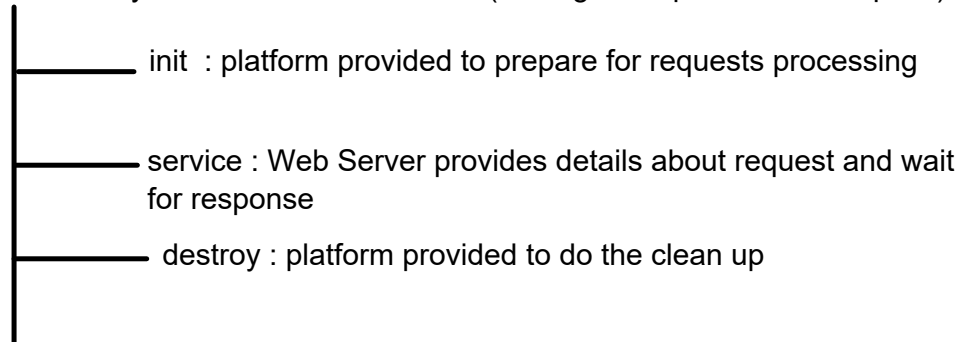
```
}
```





Lifecycle :

1. Server will create an instance of it
2. provide and runs life-cycle hooks to servlet class (manage and process the request)



service :

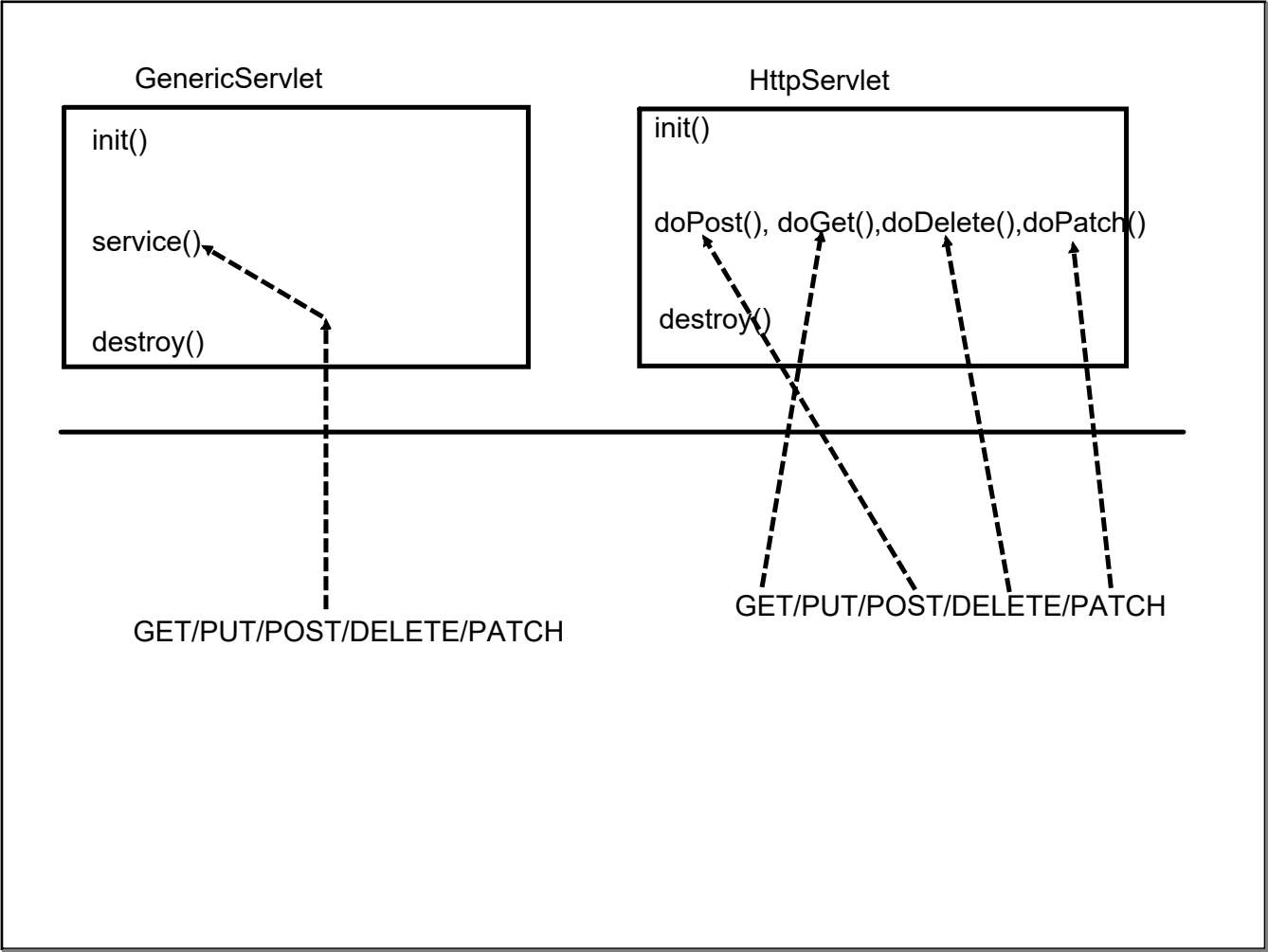
Web Server encapsulate the request related details in a Servlet-API class :
HttpServletRequest/GenericRequest

==> use request data for processing

Server Waiting for response

==> Respond back by encapsulating response in a Servlet-API class :

HttpServletResponse/GenrericResponse



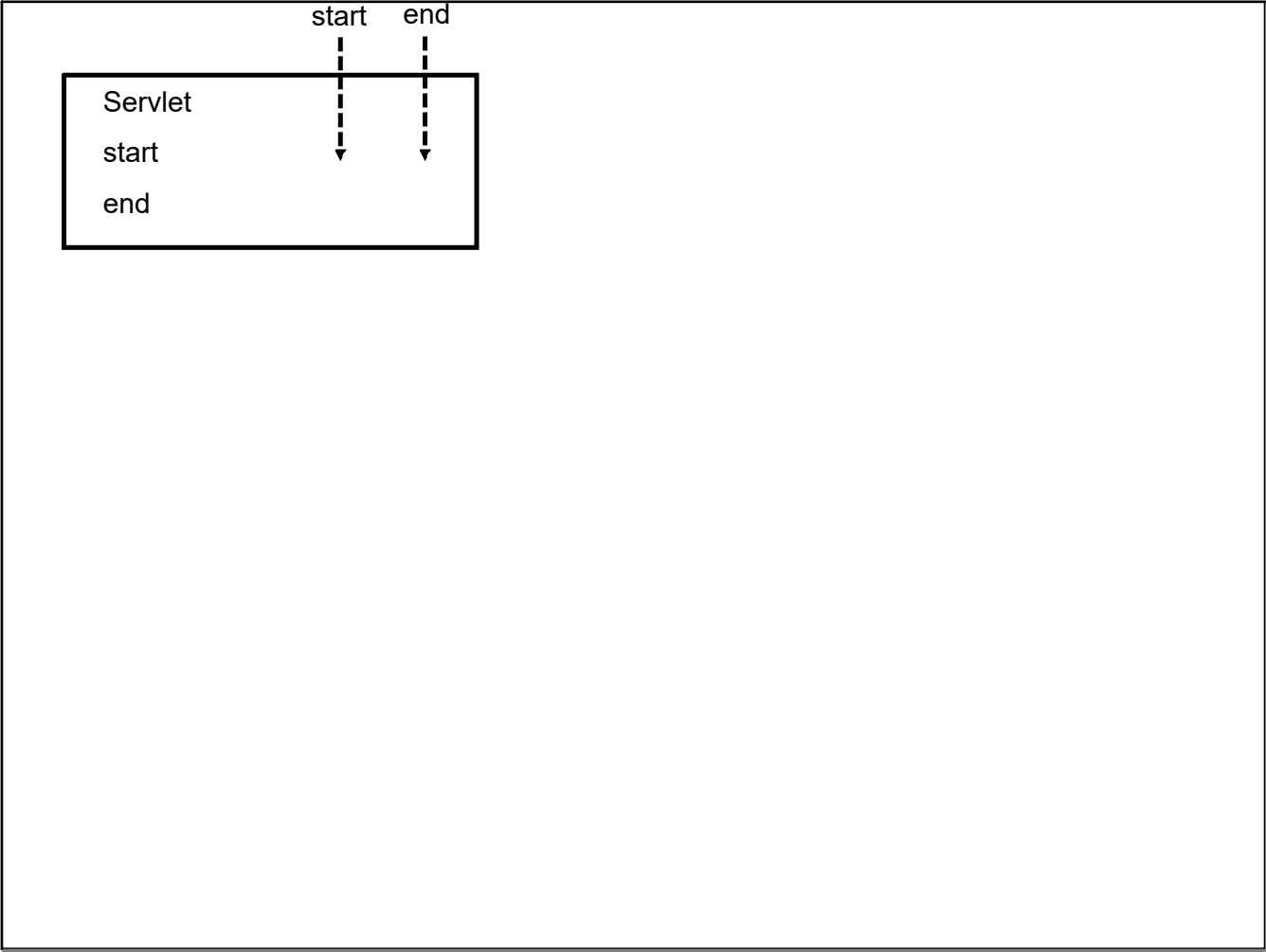
Tomcat (Apache Foundation) : download (9) and install

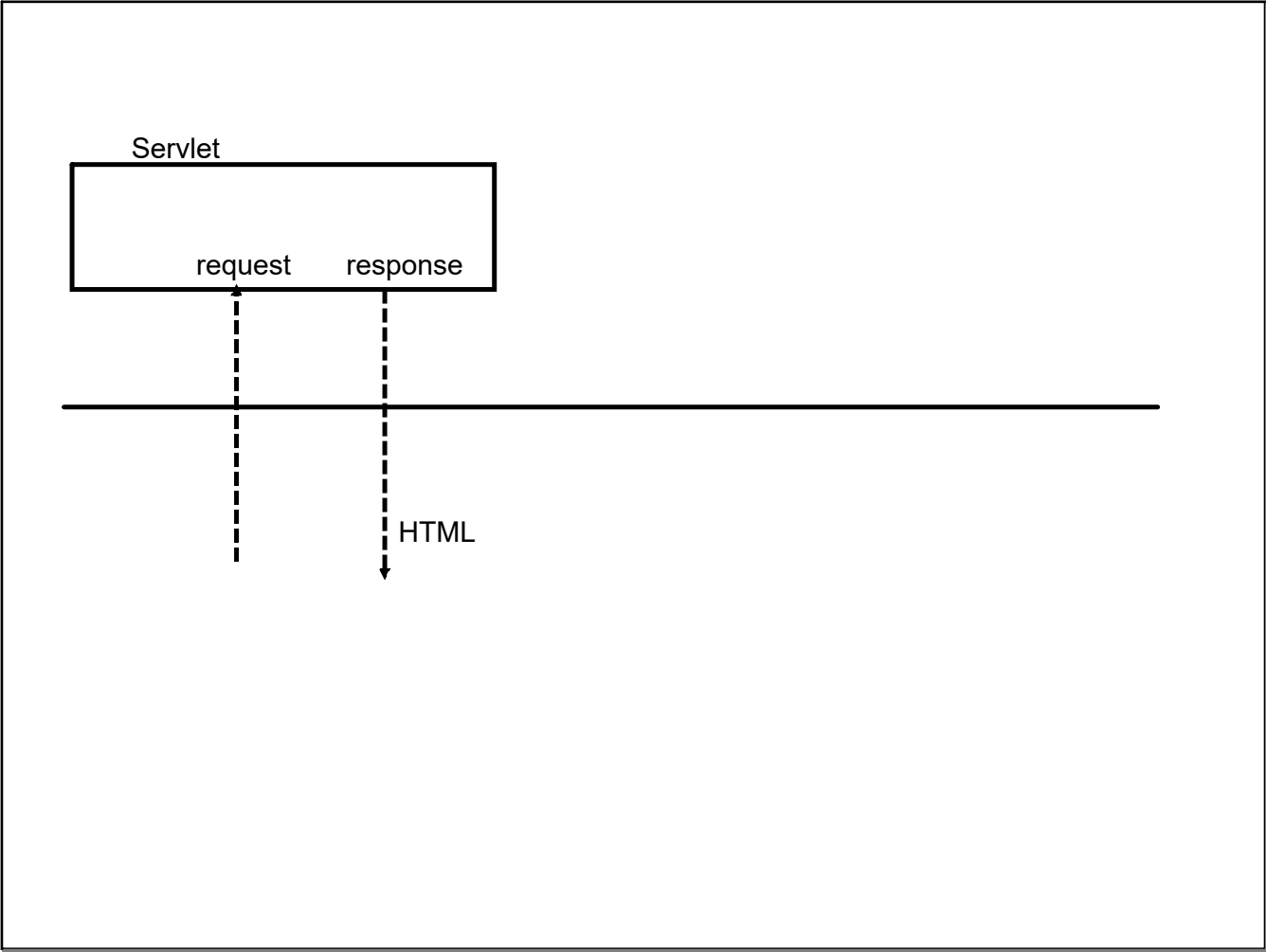
Explicitly to deploy an web-application

1. Create web application
2. build and package in war :
3. copy war file webapp folder of Tomcat

Eclipse can deploy in web server
can launch web server

Need to connect Tomcat with eclipse (workspace)

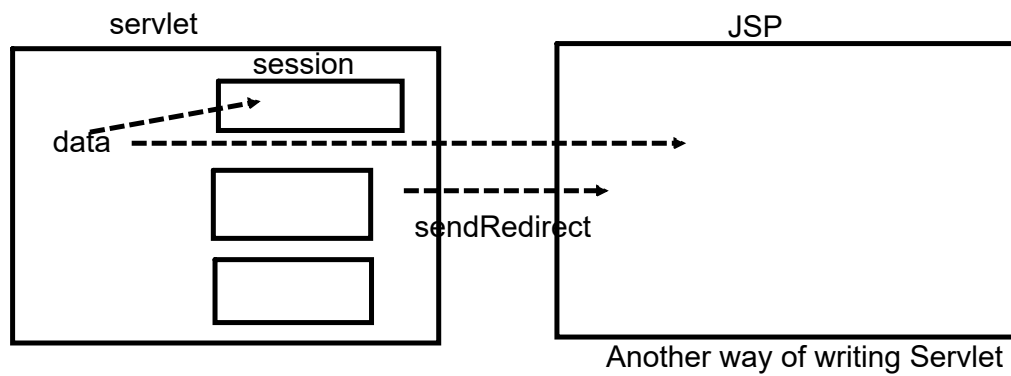




Feature of Servlet API : JSP : Java server pages

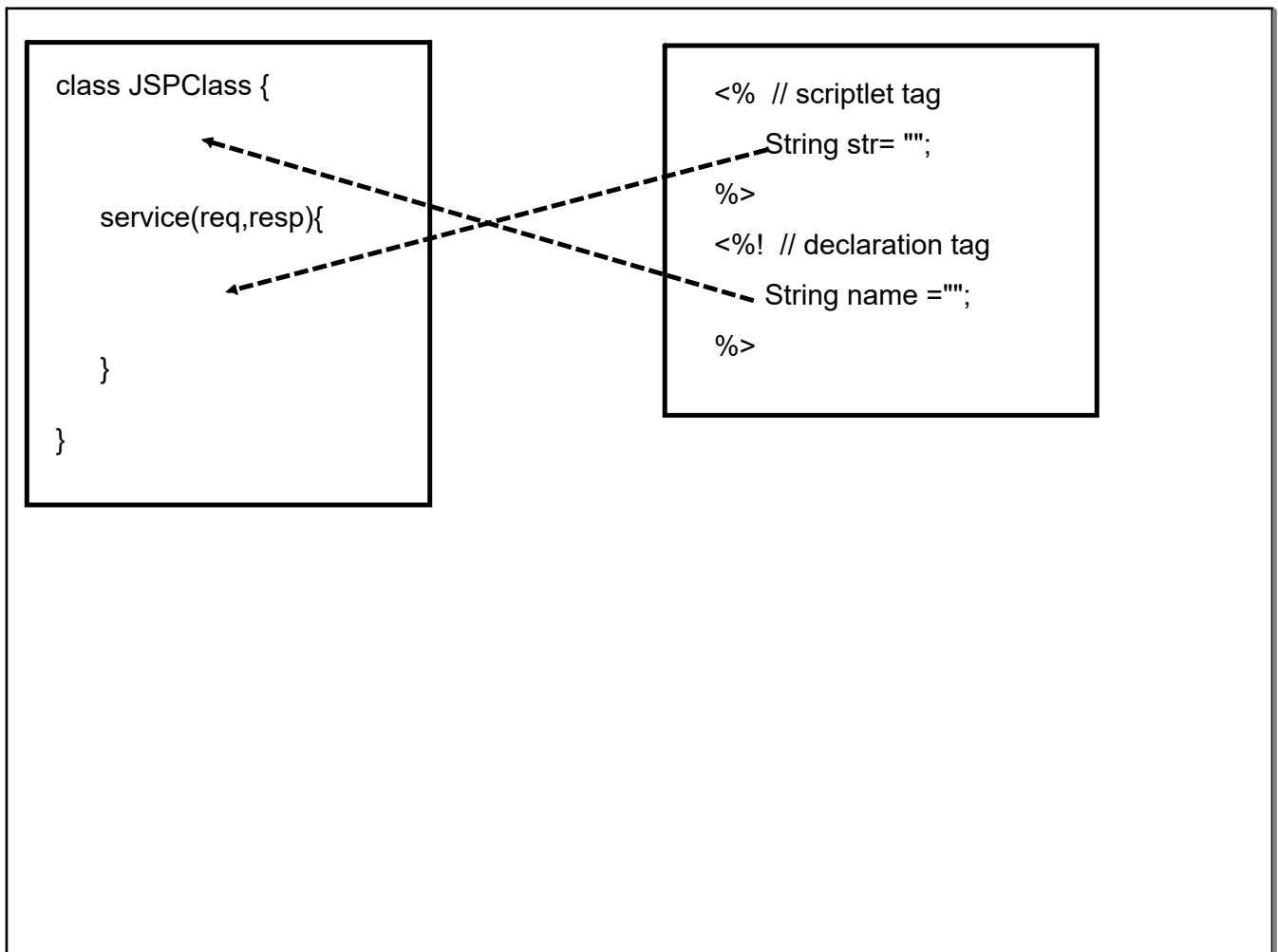
HTML : static in nature

JSP : Allow to integrate Java with HTML



#When we build projects : JSP--->Servlet

runtime .class (Servlet class)



Tag Library : allow to add dynamics in form of markup tag

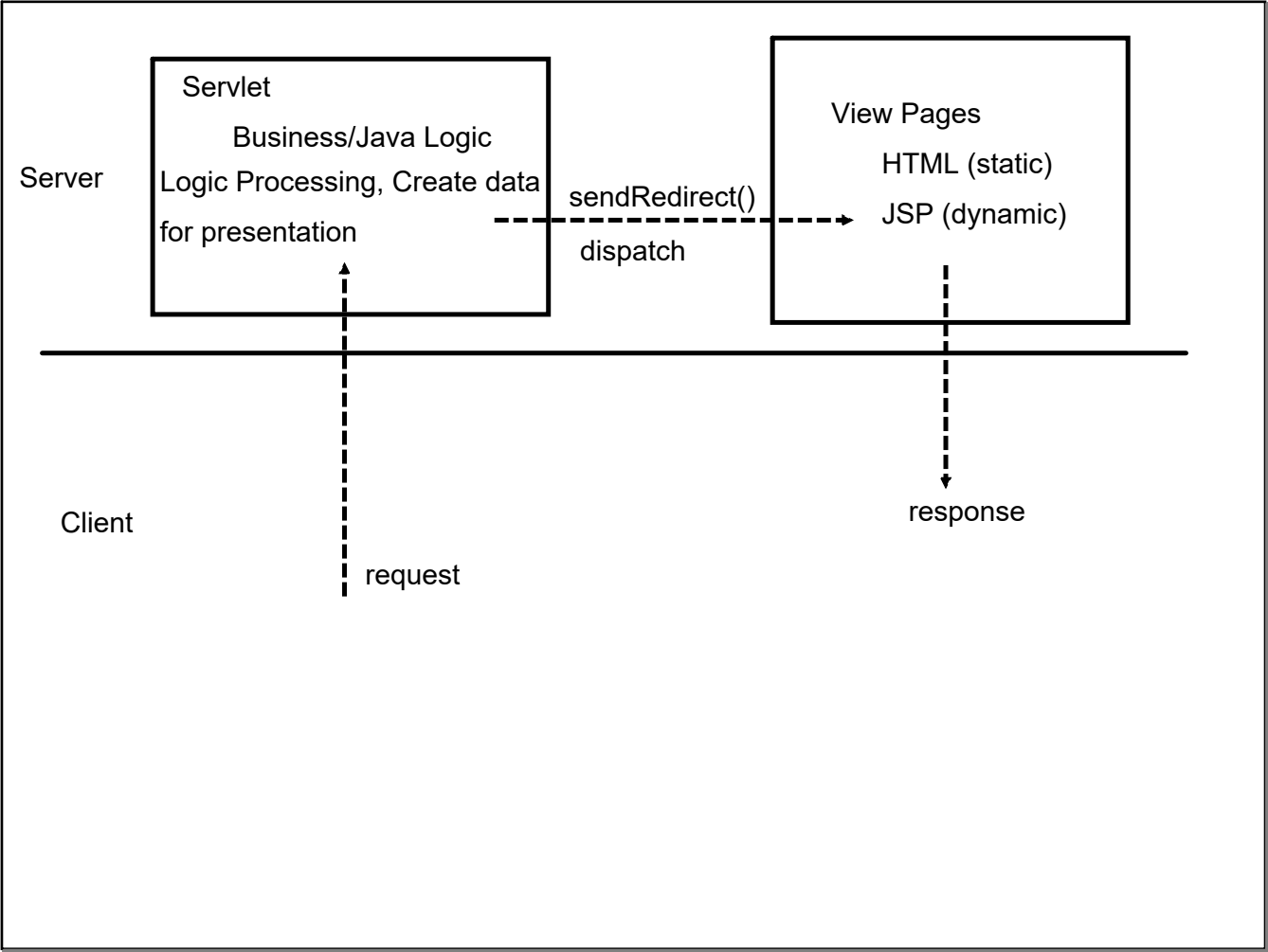
JSP : by default exposes lots of predefined object eg : PrintWriter, req, resp, session....

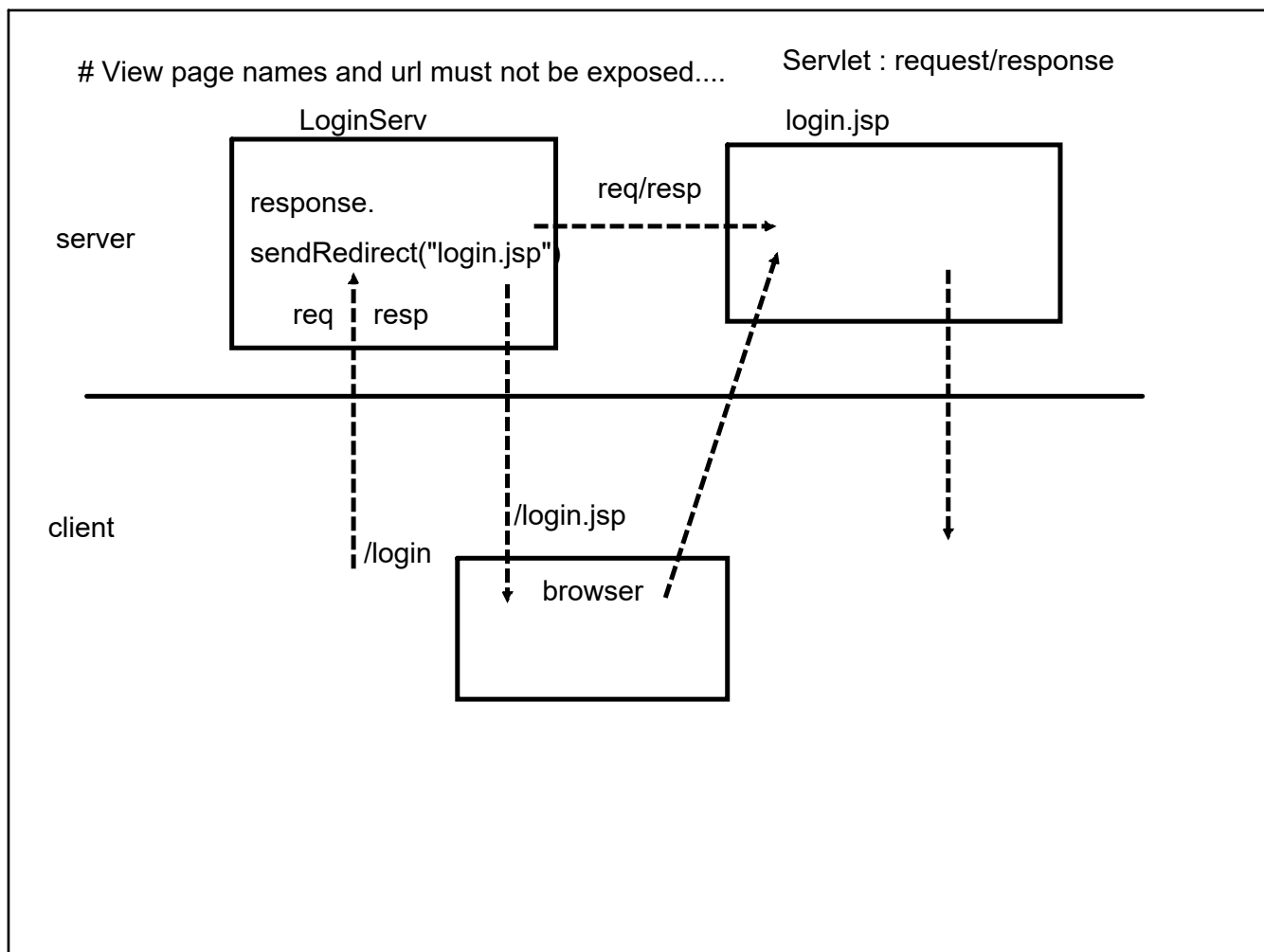
Employee Management

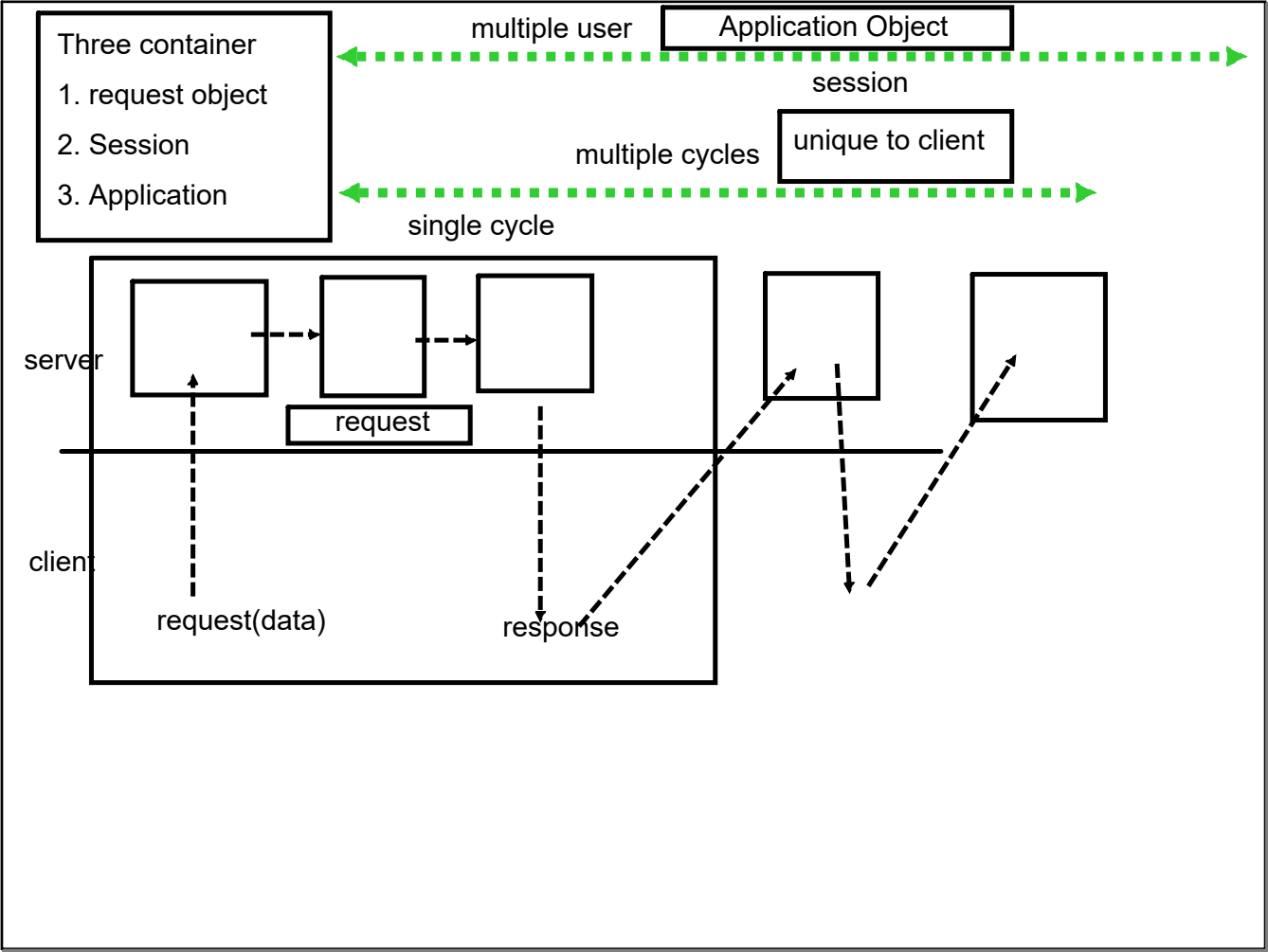
Add New Employee

List of All Employee

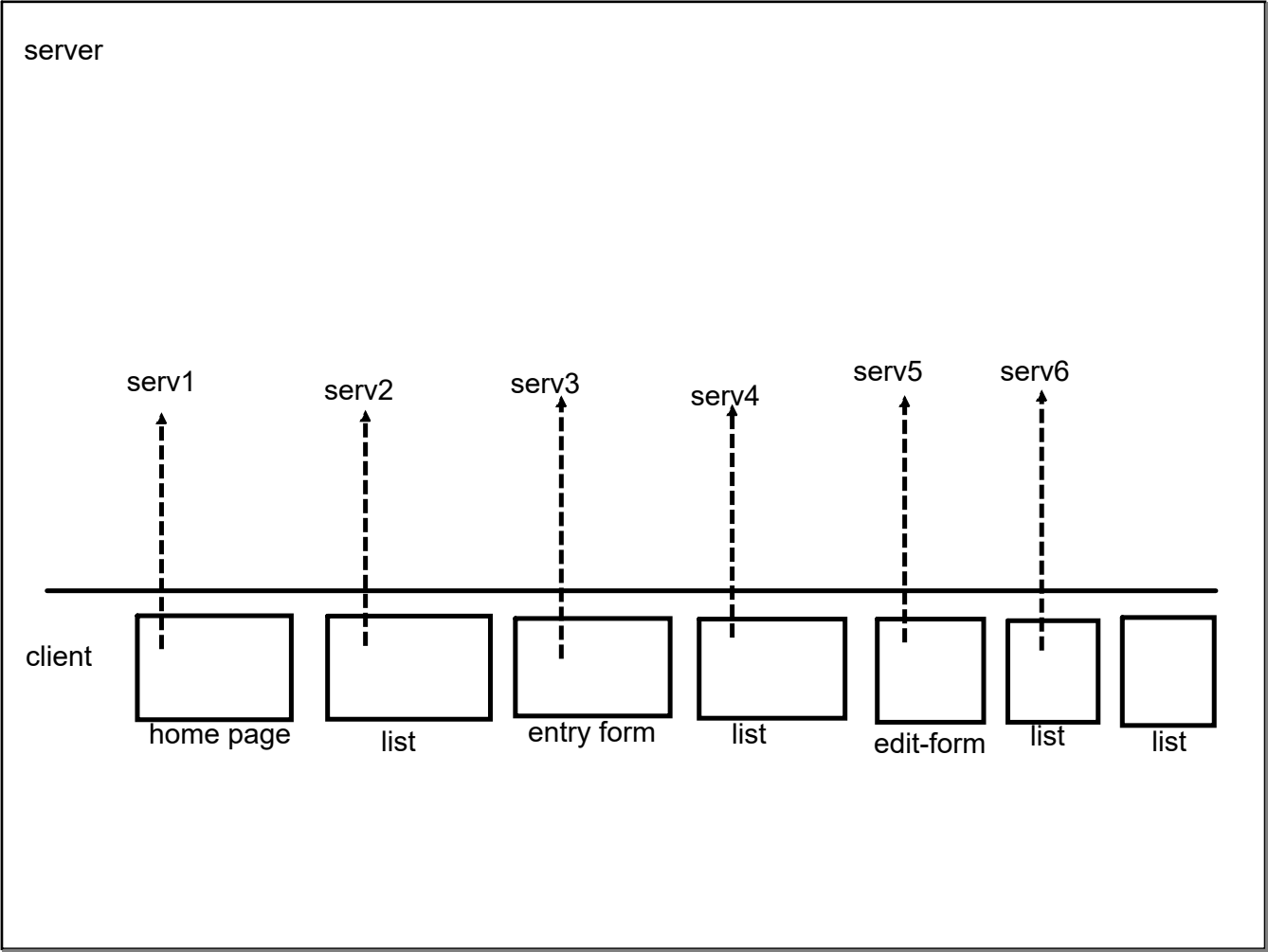
	Edit	Delete
	Edit	Delete
	Edit	Delete
	Edit	Delete
	Edit	Delete

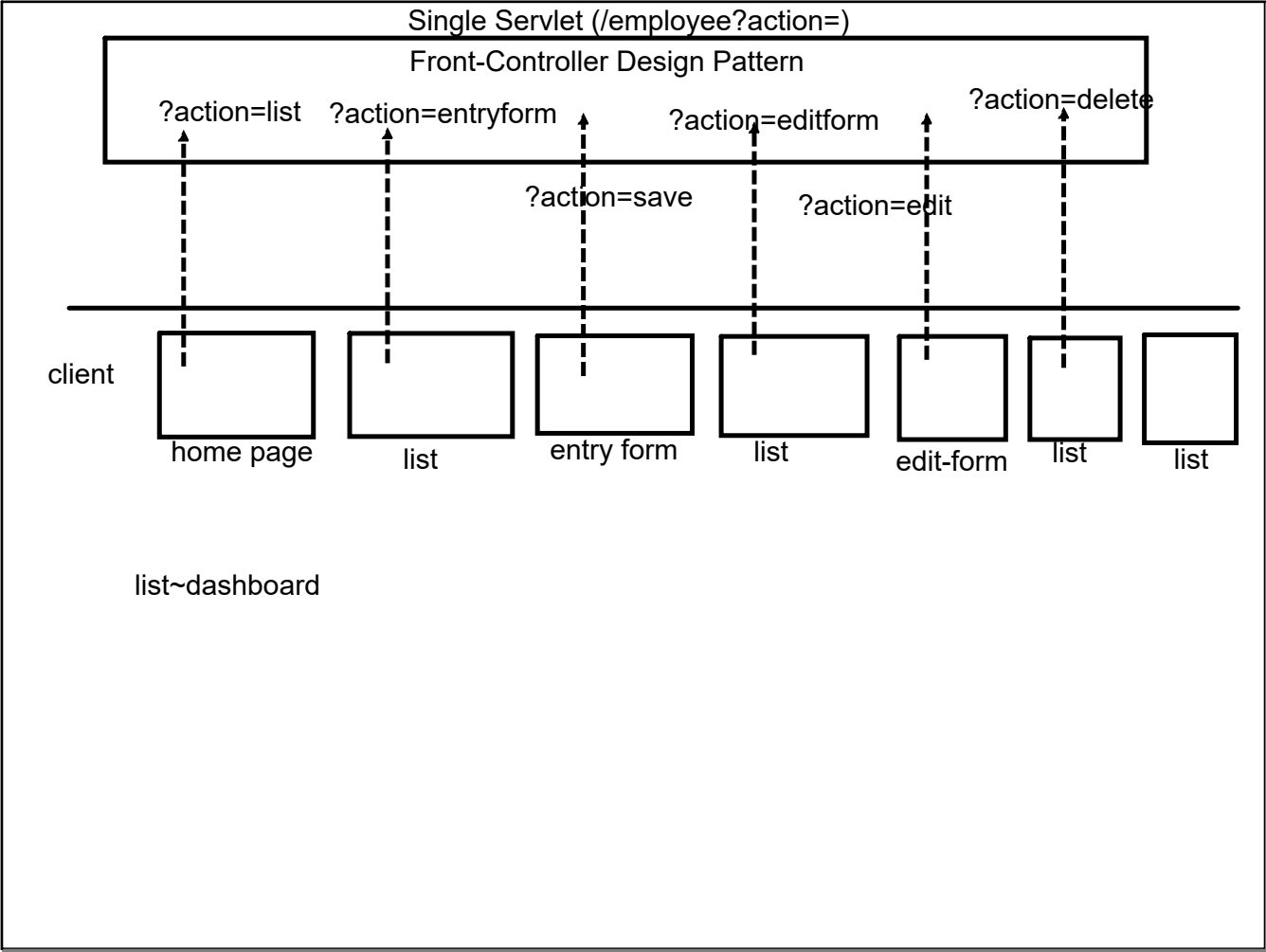


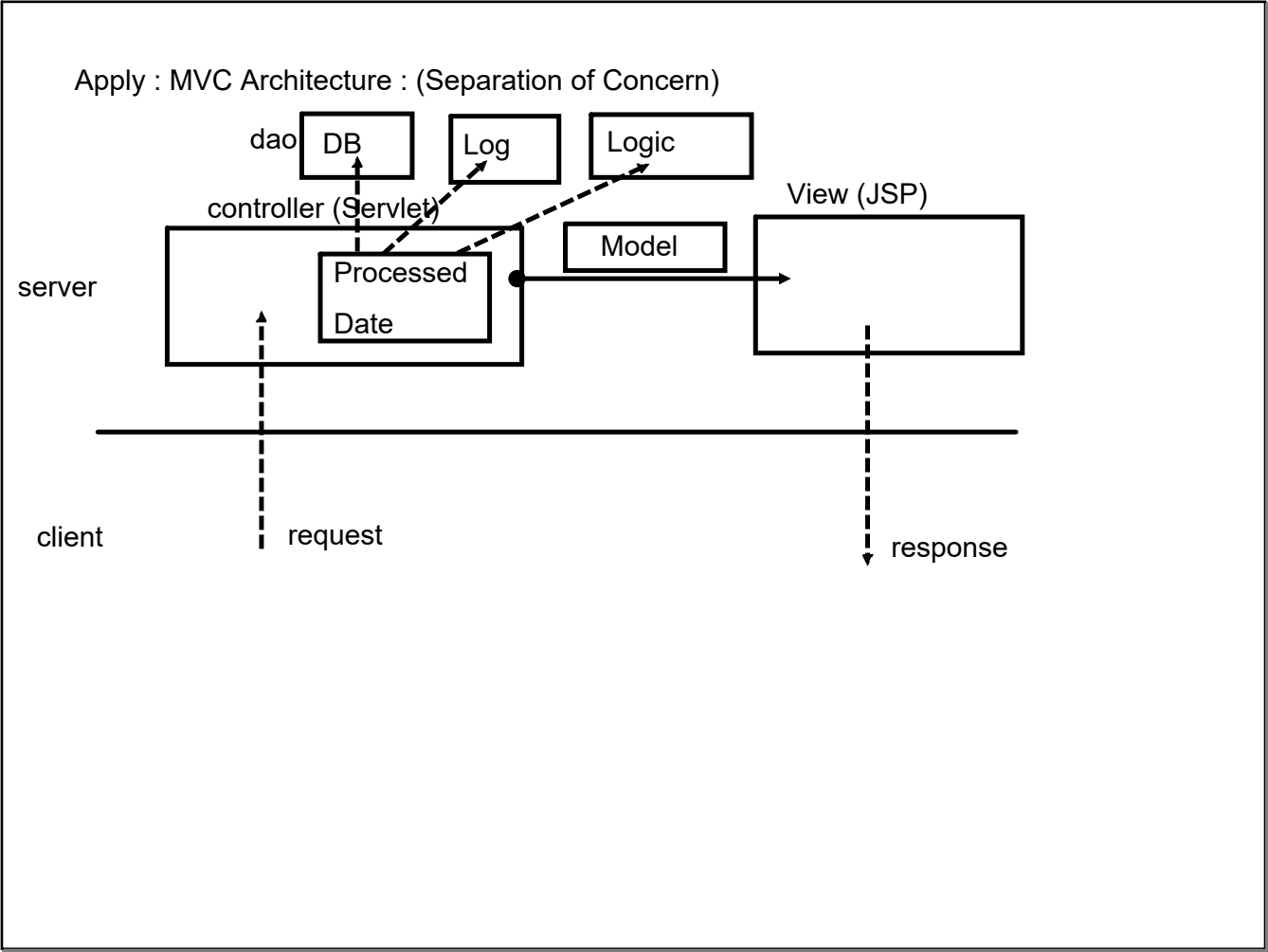


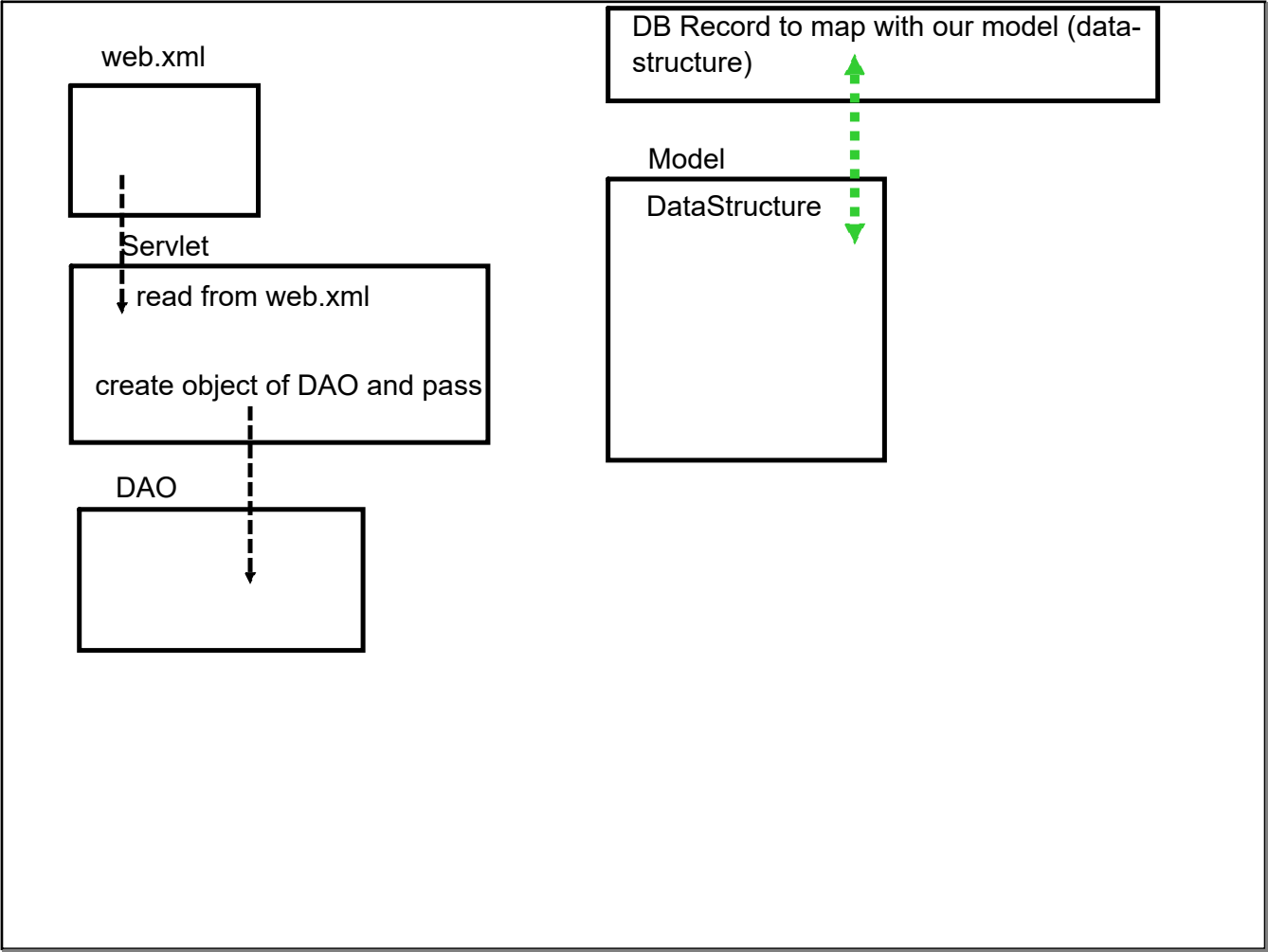


Web Server memory (application)
Separate memory space for each client (session)









Managing the Project :
Project Management Tool

Maven

Gradle

Ant

JUnit API : Unit test cases for Testing Java Code

- # Set of Annotation to organize the test case

- # Set of Assertion methods to test

Mockito API : Allow to create mock objects for testing purposes