Spring Integration - 1

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.
www.fandsindia.com

Ground Rules

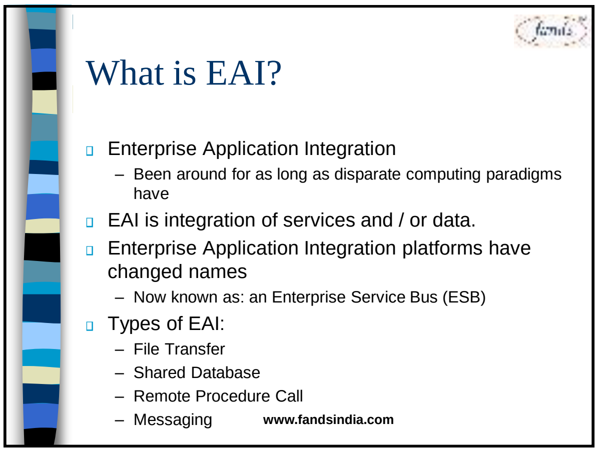
- **Turn off cell phone. If you cannot, please keep it on silent mode. You can go out and attend your call.**
- **If you have questions or issues, please let me know immediately.**
- **Let us be punctual.**

www.fandsindia.com

Agenda

EAI and Spring Integration

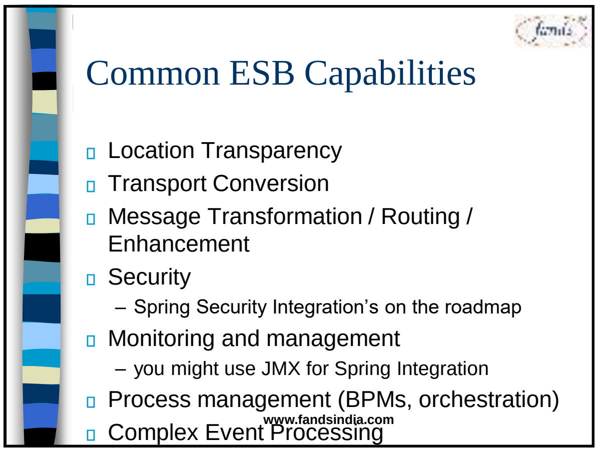
www.fandsindia.com



What is EAI?

- Enterprise Application Integration
  - Been around for as long as disparate computing paradigms have
- EAI is integration of services and / or data.
- Enterprise Application Integration platforms have changed names
  - Now known as: an Enterprise Service Bus (ESB)
- Types of EAI:
  - File Transfer
  - Shared Database
  - Remote Procedure Call
  - Messaging

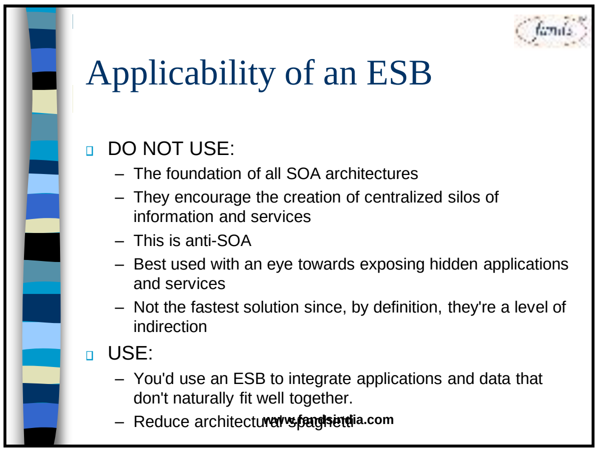
www.fandsindia.com



Common ESB Capabilities

- Location Transparency
- Transport Conversion
- Message Transformation / Routing / Enhancement
- Security
  - Spring Security Integration's on the roadmap
- Monitoring and management
  - you might use JMX for Spring Integration
- Process management (BPMs, orchestration)
- Complex Event Processing

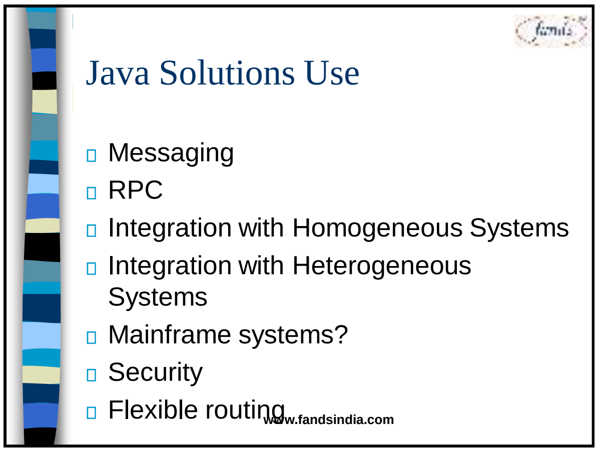
www.fandsindia.com



Applicability of an ESB

- DO NOT USE:
  - The foundation of all SOA architectures
  - They encourage the creation of centralized silos of information and services
  - This is anti-SOA
  - Best used with an eye towards exposing hidden applications and services
  - Not the fastest solution since, by definition, they're a level of indirection
- USE:
  - You'd use an ESB to integrate applications and data that don't naturally fit well together.
  - Reduce architectural spaghetti

www.fandsindia.com



Java Solutions Use

- Messaging
- RPC
- Integration with Homogeneous Systems
- Integration with Heterogeneous Systems
- Mainframe systems?
- Security
- Flexible routing

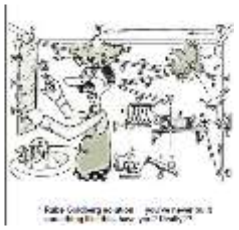
www.fandsindia.com

Spring Integration

- New addition to the Spring Portfolio
- Provides support for SOA, EDA and EAI
  – It went live with a 1.0 late in 2008.
- Provides philosophical consistency with core Spring principles.
- Spring Integration is an API geared towards building ESB - centric solutions, not another name for the Spring remoting APIs.

www.fandsindia.com

The ESB Landscape



- Traditional EAI solutions from the likes of TIBCO, Axway, WebMethods
- Open source / Java-centric solutions like ServiceMix, PEtALS, OpenESB, JBossESB, Mule (and of course Spring Integration)
- The alternatives: solutions strung together with bailing wire and tape, a veritable Rube Goldberg machine * Rube Goldberg solution -- you've never built something like

www.fandsindia.com

Spring Integration

- Uses standard Spring XML
- Idiomatic configuration: annotations, schema
- Spring Integration is embedded: deploy Spring
- Integration with your app - you don't deploy your application to Spring Integration

www.fandsindia.com

Spring Integration

- You'll typically deal with three things in an integration solution
  – Channels
  – Endpoints
  – Adapters

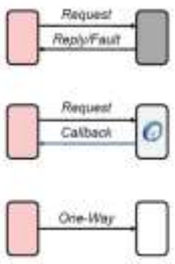
www.fandsindia.com

Synchronous and Asynchronous Interactions

Synchronous request/response
- Real-time response or error feedback
- Client in waiting mode

Asynchronous request/callback
- Client free after request submission
- Separate service invocation for response

Asynchronous request only
- Also known as "fire and forget"
- Client free after request submission
- No response



www.fandsindia.com

Before Spring Integration

www.fandsindia.com

Overview of java.util.concurrent

www.fandsindia.com

Processes and Threads

- two basic units of execution: *processes* and *threads*.
- In Java - concurrent programming is mostly concerned with threads, processes are also important.
- It's becoming more and more common for computer systems to have multiple processors or processors with multiple execution cores. This greatly enhances a system's capacity for concurrent execution of processes and threads — but concurrency is possible even on simple systems, without multiple processors or execution cores.

www.fandsindia.com

Processes

- A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.
- Processes are often seen as synonymous with programs or applications. However, what the user sees as a single application may in fact be a set of cooperating processes. To facilitate communication between processes, most operating systems support Inter Process Communication (IPC) resources, such as pipes and sockets. IPC is used not just for communication between processes on the same system, but processes on different systems.
- Most implementations of the Java virtual machine run as a single process. A Java application can create additional processes using a ProcessBuilder object.

Threads

- Threads are sometimes called *lightweight processes*. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process.
- Threads exist within a process - every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.

We know multithreading

- Join
- Synchronization
- ThreadPool
- DeadLock
- Starvation
- LiveLock
  – they are simply too busy responding to each other to resume work.

High Level Concurrency Objects

- Lock objects support locking idioms that simplify many concurrent applications.
- Executors define a high-level API for launching and managing threads. Executor implementations provided by java.util.concurrent provide thread pool management suitable for large-scale applications.
- Concurrent collections make it easier to manage large collections of data, and can greatly reduce the need for synchronization.
- Atomic variables have features that minimize synchronization and help avoid memory consistency errors.

Starting Threads

- Executor framework

```
public interface Executor {
   void execute(Runnable command);
}
```

Executor Example

```
class MyTask implements Runnable {
  public void run() {…}
  public static void main(…) {
    Runnable task1 = new MyTask();
    Executor exec = /* */;
    exec.execute(task1);
  }
}
```

Thread/ Task

```
class OnePer implements Executor {
  public void Execute(Runnable r){
    new Thread(r).start();
  }
}
```

Single Threaded

```
class Just1 implements Executor {
  public void Execute(Runnable r){
    r.run();
  }
}
```

Provided Thread Pool Executors

- `newFixedThreadPool`
  - Bounded size
  - Replace if thread dies
- `newCachedThreadPool`
  - Demand driven variable size
- `newSingleThreadExecutor`
  - Just one thread
  - Replace if thread dies
- `newScheduledThreadPool`
  - Delayed and periodic task execution
  - Good replacement for class `Timer`

Executor Shut-down

- Executor is a serice provider
- Executor abstracts thread management
- To maintain the abstraction, the service should generally provide methods for shutting down the service
- JVM cannot shut down until threads do

ExecutorService

```
public
interface ExecutorService extends Executor {
  void shutdown();
  List<Runnable> shutdownNow();
  boolean isShutdown();
  boolean isTerminated();
  boolean
  awaitTermination(long timeout, TimeUnit unit)
                    throws InterruptedException;
}
```

ExecutorService Notes

- Implies three states:
  - Running
  - Shutting down
  - Terminated
- `shutdown()` runs tasks in queue
- `shutdownNow()` returns unstarted tasks
- `awaitTermination()` blocks until the service is in the terminated state

ExecutorService implementation

- ExecutorService is an interface
- How do you implement shut down and cancellation?

Cancellation in Java

- *Cooperative*, not mandated
- Traditional method: *interruption*

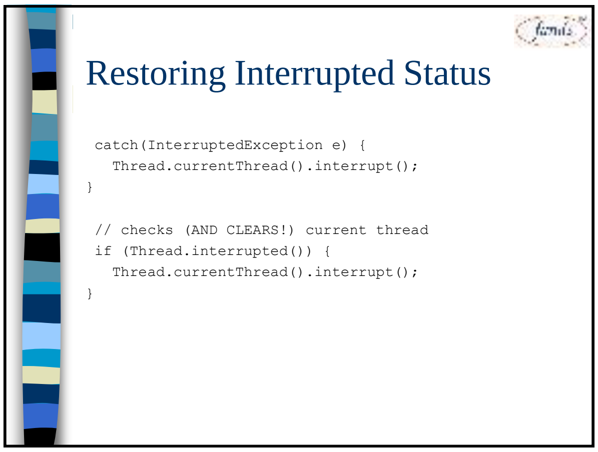
```
Public class Thread {
  public void interrupt();
  public void isInterrupted();
  public static boolean interrupted();
  …
}
```

Interruption

- Just a *request*!
- Sets a flag that must be explicitly cleared!
- Some methods clear the flag & throw *InterruptedException*
- Others *ignore* it, leaving other code to handle it

IMPORTANT!

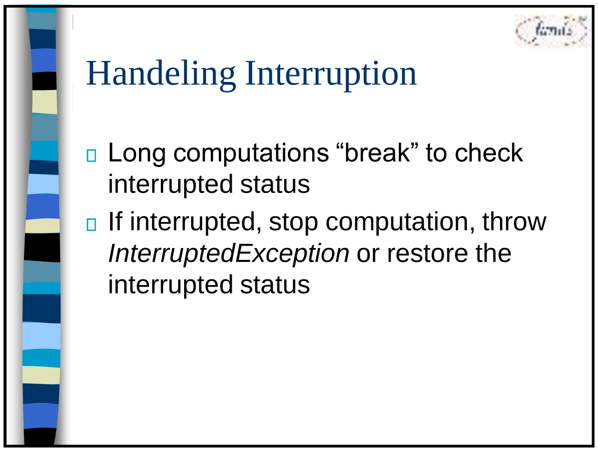
- Your code should follow protocol when interrupted
- If *interrupted()*,
  - Throw exception
  - Reset the flag for other code
- If *InterruptedException*
  - Pass it along
  - Reset the flag for other code
- Don’t swallow, unless your code is handles the interruption policy



Restoring Interrupted Status

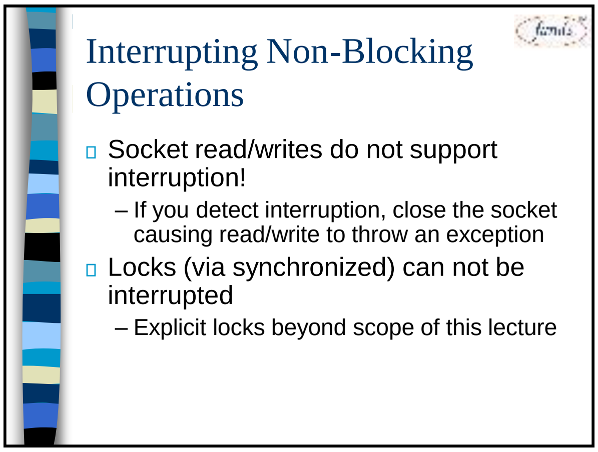
```
catch(InterruptedException e) {
  Thread.currentThread().interrupt();
}

// checks (AND CLEARS!) current thread
if (Thread.interrupted()) {
  Thread.currentThread().interrupt();
}
```



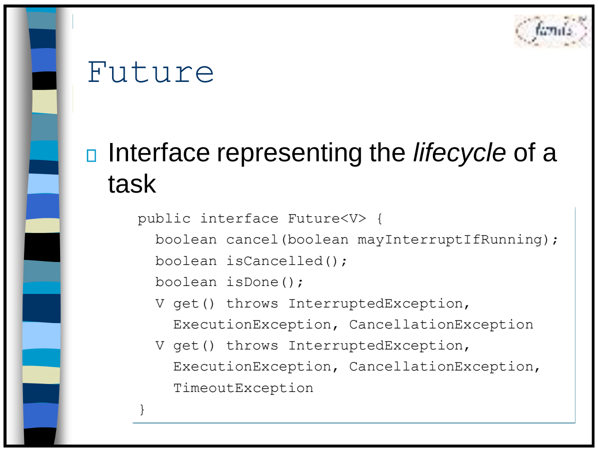
Handeling Interruption

- Long computations “break” to check interrupted status
- If interrupted, stop computation, throw *InterruptedException* or restore the interrupted status



Interrupting Non-Blocking Operations

- Socket read/writes do not support interruption!
  - If you detect interruption, close the socket causing read/write to throw an exception
- Locks (via synchronized) can not be interrupted
  - Explicit locks beyond scope of this lecture



Future

- Interface representing the *lifecycle* of a task

```
public interface Future<V> {
  boolean cancel(boolean mayInterruptIfRunning);
  boolean isCancelled();
  boolean isDone();
  V get() throws InterruptedException,
    ExecutionException, CancellationException
  V get() throws InterruptedException,
    ExecutionException, CancellationException,
    TimeoutException
}
```

CompletionService

- Combines `Executor` with `BlockingQueue`
- `ExecutorCompletionService`

```
public interface CompletionService<V> {
  Future<V> poll();
  Future<V> poll(long timeout, TimeUnit unit);
  Future<V> submit(Callable<V> task);
  Future<V> submit(Runnable<V> task);
  Future<V> take();
}
```

Futures Again

- *ExecutorService* interface also provides

```
public interface ExecutorService {
  …
  Future<T> submit(Callable<T> task);
  Future<?> submit(Runnable task);
  Future<T> submit(Runnable task, T result);
}
```

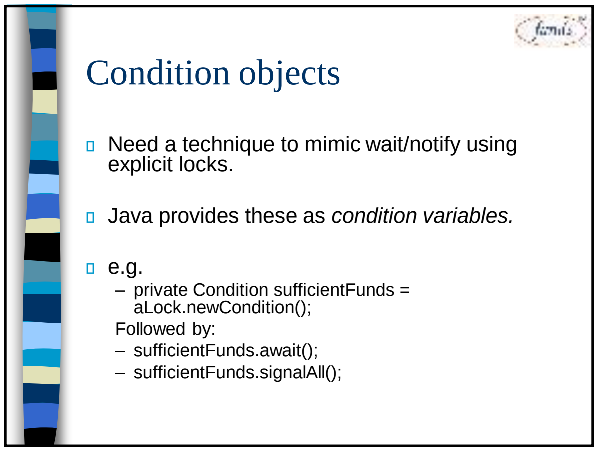
Higher level concurrency strategies

- Rather than relying on implicitly obtained locks with synchronized code blocks, since JDK1.5 Java now allows use of explicit lock objects.
- Key classes are in java.util.concurrent.locks.
- *ReentrantLock* is key class to focus on.
- Does not enable new capabilities, just simplifies programming model.

Basics of *Lock* interface --example

```
private Lock aLock = new ReentrantLock();
myLock.lock();
try{
 // synchronized code here
}
finally {
 myLock.unlock();
}
```

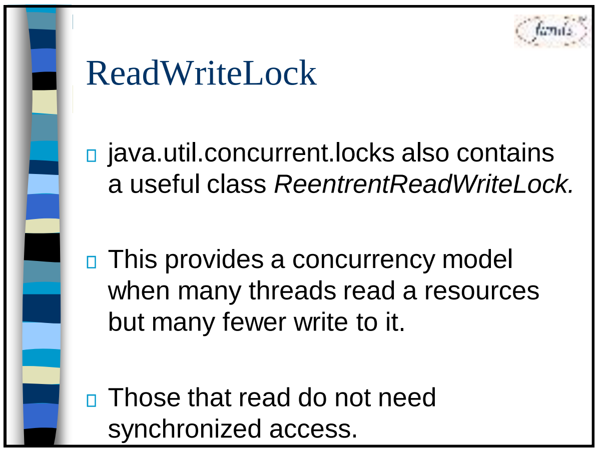
• lock() method blocs until lock is available
• This is same as synchronized code block but attempt to acquire lock
 is explicit.
• It is reentrant because a thread can reacquire a lock it already owns.



Condition objects

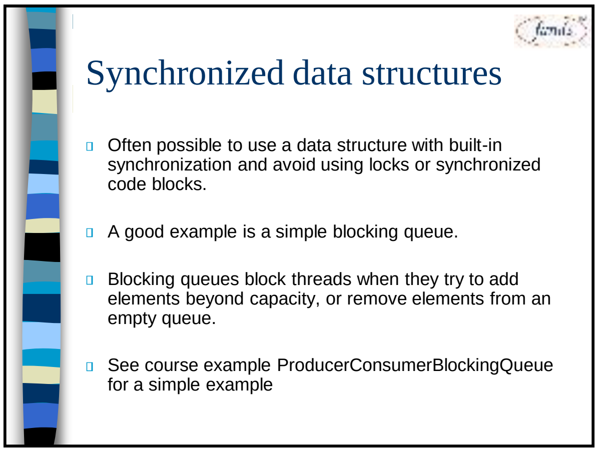
- Need a technique to mimic wait/notify using explicit locks.
- Java provides these as *condition variables.*
- e.g.
  - private Condition sufficientFunds = aLock.newCondition();

  Followed by:
  - sufficientFunds.await();
  - sufficientFunds.signalAll();



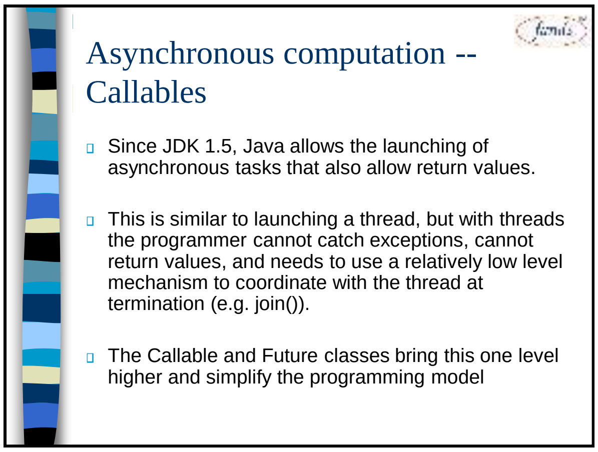
ReadWriteLock

- java.util.concurrent.locks also contains a useful class *ReentrentReadWriteLock.*
- This provides a concurrency model when many threads read a resources but many fewer write to it.
- Those that read do not need synchronized access.



Synchronized data structures

- Often possible to use a data structure with built-in synchronization and avoid using locks or synchronized code blocks.
- A good example is a simple blocking queue.
- Blocking queues block threads when they try to add elements beyond capacity, or remove elements from an empty queue.
- See course example ProducerConsumerBlockingQueue for a simple example



Asynchronous computation -- Callables

- Since JDK 1.5, Java allows the launching of asynchronous tasks that also allow return values.
- This is similar to launching a thread, but with threads the programmer cannot catch exceptions, cannot return values, and needs to use a relatively low level mechanism to coordinate with the thread at termination (e.g. join()).
- The Callable and Future classes bring this one level higher and simplify the programming model

Using Callable Objects

- Callable classes implement the Callable interface and are similar to threads:

```
Callable<T>{
  public T call(){

  }
}
```

- Rather than passed to Thread objects, Callable classes are launch using the submit(Callable) method of an ExecutorService, e.g.

```
ExecutorService pool = Executors.newFixedThreadPool(3);
Future f = pool.submit(myCallable);
And later …
retunVal = f.get(); //a blocking operation!
```

Task Execution and Scheduling

www.fandsindia.com

Introduction

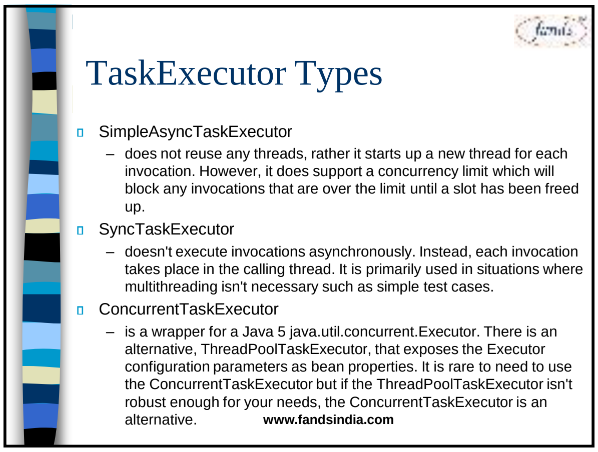
- The Spring Framework provides abstractions for asynchronous execution and scheduling of tasks with the TaskExecutor and TaskScheduler interfaces, respectively.
- Spring also features implementations of those interfaces that support thread pools or delegation to CommonJ within an application server environment.
- Ultimately the use of these implementations behind the common interfaces abstracts away the differences between Java SE 5, Java SE 6 and Java EE environments.

www.fandsindia.com

TaskExecutor

- Simple task executor interface that abstracts the execution of a Runnable.
- Implementations can use all sorts of different execution strategies, such as: synchronous, asynchronous, using a thread pool, and more.
- Equivalent to JDK 1.5's Executor interface; extending it now in Spring, so that clients may declare a dependency on an Executor and receive any TaskExecutor implementation.

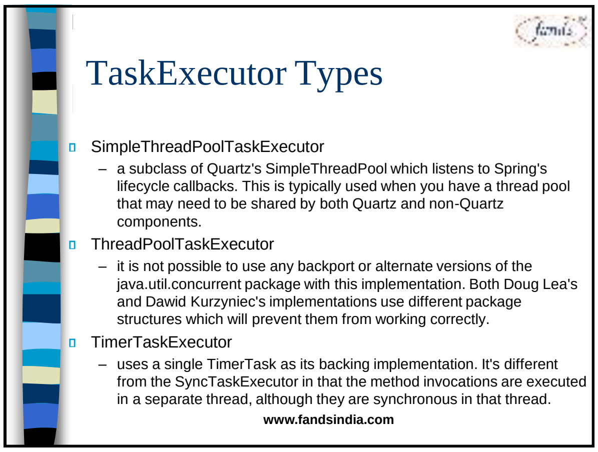
www.fandsindia.com



TaskExecutor Types

- SimpleAsyncTaskExecutor
  - does not reuse any threads, rather it starts up a new thread for each invocation. However, it does support a concurrency limit which will block any invocations that are over the limit until a slot has been freed up.
- SyncTaskExecutor
  - doesn't execute invocations asynchronously. Instead, each invocation takes place in the calling thread. It is primarily used in situations where multithreading isn't necessary such as simple test cases.
- ConcurrentTaskExecutor
  - is a wrapper for a Java 5 java.util.concurrent.Executor. There is an alternative, ThreadPoolTaskExecutor, that exposes the Executor configuration parameters as bean properties. It is rare to need to use the ConcurrentTaskExecutor but if the ThreadPoolTaskExecutor isn't robust enough for your needs, the ConcurrentTaskExecutor is an alternative.

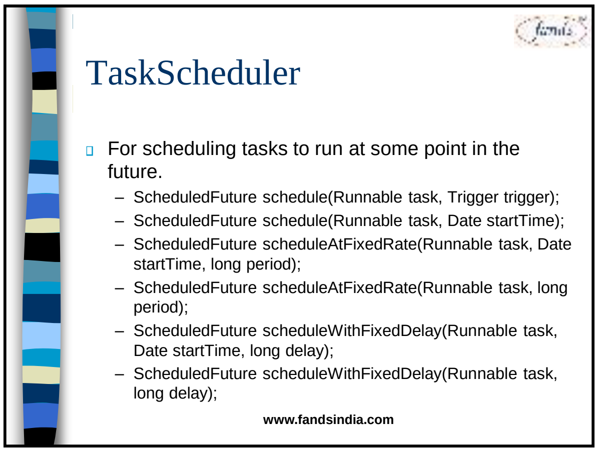
www.fandsindia.com



TaskExecutor Types

- SimpleThreadPoolTaskExecutor
  - a subclass of Quartz's SimpleThreadPool which listens to Spring's lifecycle callbacks. This is typically used when you have a thread pool that may need to be shared by both Quartz and non-Quartz components.
- ThreadPoolTaskExecutor
  - it is not possible to use any backport or alternate versions of the java.util.concurrent package with this implementation. Both Doug Lea's and Dawid Kurzyniec's implementations use different package structures which will prevent them from working correctly.
- TimerTaskExecutor
  - uses a single TimerTask as its backing implementation. It's different from the SyncTaskExecutor in that the method invocations are executed in a separate thread, although they are synchronous in that thread.

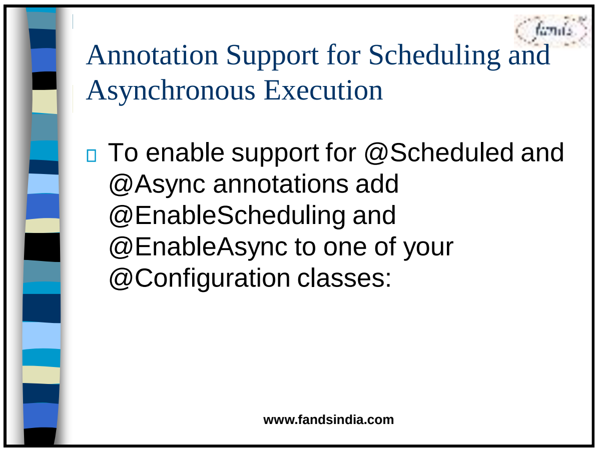
www.fandsindia.com



TaskScheduler

- For scheduling tasks to run at some point in the future.
  - ScheduledFuture schedule(Runnable task, Trigger trigger);
  - ScheduledFuture schedule(Runnable task, Date startTime);
  - ScheduledFuture scheduleAtFixedRate(Runnable task, Date startTime, long period);
  - ScheduledFuture scheduleAtFixedRate(Runnable task, long period);
  - ScheduledFuture scheduleWithFixedDelay(Runnable task, Date startTime, long delay);
  - ScheduledFuture scheduleWithFixedDelay(Runnable task, long delay);

www.fandsindia.com



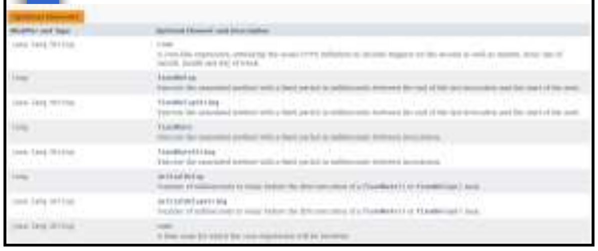
Annotation Support for Scheduling and Asynchronous Execution

- To enable support for @Scheduled and @Async annotations add @EnableScheduling and @EnableAsync to one of your @Configuration classes:

www.fandsindia.com

@Scheduled

- Exactly one of the cron(), fixedDelay(), or fixedRate() attributes must be specified.



[illegible]	[illegible]
[illegible]	cron [illegible]
[illegible]	fixedDelay [illegible]
[illegible]	fixedDelayString [illegible]
[illegible]	fixedRate [illegible]
[illegible]	fixedRateString [illegible]
[illegible]	initialDelay [illegible]
[illegible]	initialDelayString [illegible]
[illegible]	zone [illegible]

The @Async Annotation

- The @Async annotation can be provided on a method so that invocation of that method will occur asynchronously.
- In other words, the caller will return immediately upon invocation and the actual execution of the method will occur in a task that has been submitted to a Spring TaskExecutor.
- In the simplest case, the annotation may be applied to a void-returning method.

Enable Support(*@EnableAsync*)

- ***annotation*** – by default, *@EnableAsync* detects Spring's *@Async* annotation and the EJB 3.1 *javax.ejb.Asynchronous*; this option can be used to detect other, user-defined annotation types as well
- ***mode*** – indicates the type of *advice* that should be used – JDK proxy-based or AspectJ weaving
- ***proxyTargetClass*** – indicates the type of *proxy* that should be used – CGLIB or JDK; this attribute has effect only if the ***mode*** is set to *AdviceMode.PROXY*
- ***order*** – sets the order in which *AsyncAnnotationBeanPostProcessor* should be applied; by default, it runs last, just so that it can take into account all existing proxies

@Async Annotation

- Must be applied to *public* methods only and Self-invocation – calling the async method from within the same class won't work (why?)
- Can return
  - Void
  - Future
- Exception Handling
  - *Void* - exceptions will not be propagated to the calling thread.
  - *Future.get()* - method will throw the exception.

RESTful Web Services

Basic SOA Using REST

- REST stands for Representational State Transfer. It was first introduced by Roy Fielding in his 2000 doctoral dissertation (Fielding is one of the principal authors of the HTTP specification and a co-founder of the Apache HTTP Server project.).

What Is REST?

- REST-style services (i.e., RESTful services) adhere to a set of constraints and architectural principles that include the following
  - Stateless
  - Uniform Interface
  - built from resources
  - manipulate resources by exchanging representations of the resources

What Is REST?

- Stateless – each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server."
- RESTful services have a uniform interface. This constraint is usually taken to mean that the only allowed operations are the HTTP operations: GET, POST, PUT, and DELETE

What Is REST?

- REST-based architectures are built from resources (pieces of information) that are uniquely identified by URIs. For example, in a RESTful purchasing system, each purchase order has a unique URI.
- REST components manipulate resources by exchanging representations of the resources. For example, a purchase order resource can be represented by an XML document. Within a RESTful purchasing system, a purchase order might be updated by posting an XML document containing the changed purchase order to its URI.

Rest Vs RPC

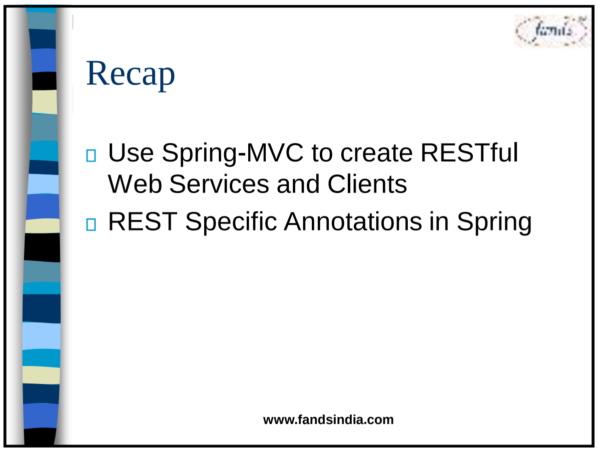
- REST-based architectures communicate primarily through the transfer of representations of resources" This is fundamentally different from the RPC approach that encapsulates the notion of invoking a procedure on the remote server. Hence, RPC messages typically contain information about the procedure to be invoked or action to be taken. This information is referred to as a verb in a Web service request. In the REST model, the only verbs allowed are GET, POST, PUT, and DELETE. In the RPC approach, typically many operations are invoked at the same URI. This is to be contrasted with the REST approach of having a unique URI for each resource.

Is it really REST?

- For example, among REST advocates, keeping shopping cart data on the server and maintaining a session related to the shopping process that is using the cart is acceptable.
- Storing session information or shopping cart data on the server is a clear violation of Fielding's original REST concept since it violates the requirement that a service be stateless

Uniform Interface?

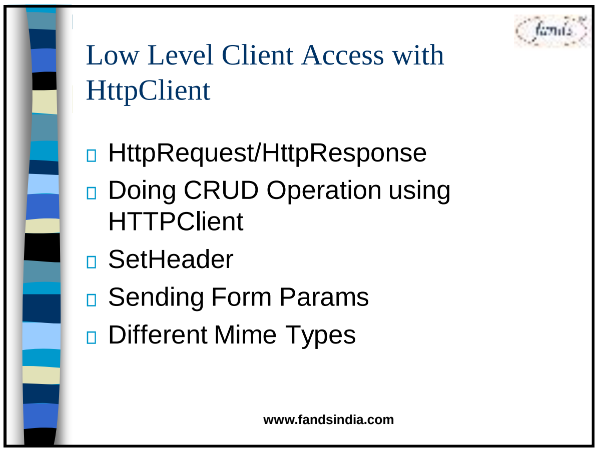
- More significant deviations from Fielding's definition of REST involve getting around the "uniform interface" constraint by embedding verbs and parameters inside URLs. The Amazon.com REST interface, for example, includes verbs in query strings and doesn't have unique URIs for each resource. Systems like this, although labeled as RESTful, are really starting to look very much like RPC using XML over HTTP without SOAP.



Recap

- Use Spring-MVC to create RESTful Web Services and Clients
- REST Specific Annotations in Spring

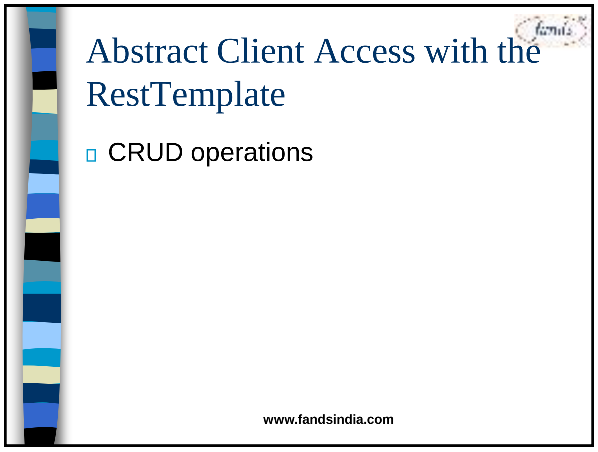
www.fandsindia.com



Low Level Client Access with HttpClient

- HttpRequest/HttpResponse
- Doing CRUD Operation using HTTPClient
- SetHeader
- Sending Form Params
- Different Mime Types

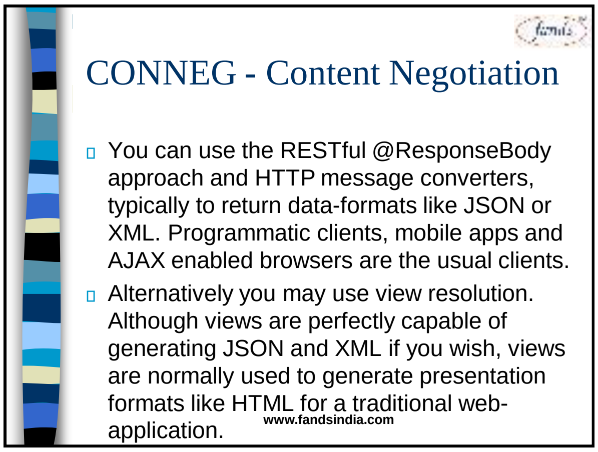
www.fandsindia.com



Abstract Client Access with the RestTemplate

- CRUD operations

www.fandsindia.com



CONNEG - Content Negotiation

- You can use the RESTful @ResponseBody approach and HTTP message converters, typically to return data-formats like JSON or XML. Programmatic clients, mobile apps and AJAX enabled browsers are the usual clients.
- Alternatively you may use view resolution. Although views are perfectly capable of generating JSON and XML if you wish, views are normally used to generate presentation formats like HTML for a traditional web-application.

www.fandsindia.com

Http Header

- There are three situations where we need to know what type of data-format to send in the HTTP response:
  - HttpMessageConverters: Determine the right converter to use.
  - Request Mappings: Map an incoming HTTP request to different methods that return different formats.
  - View Resolution: Pick the right view to use.
- Determining what format the user has requested relies on a ContentNegotationStrategy (either out of the box / custom)

Enabling Content Negotiation in Spring MVC

- Supported Options
  – URL suffixes and/or a URL parameter. These work alongside the use of Accept headers. As a result, the content-type can be requested in any of three ways. By default they are checked in this order
  1 - Add a path extension (suffix) in the URL.
    - For url like http://.../list.html - HTML is required.
    - For http://../list.xls. The suffix to media-type mapping is automatically defined via the JavaBeans Activation Framework or JAF (so

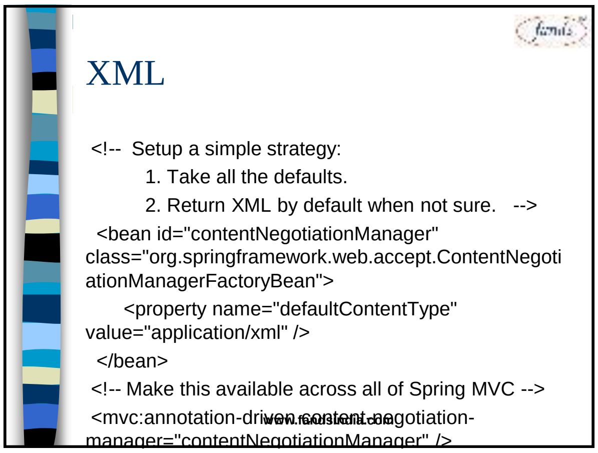
Enabling Content Negotiation in Spring MVC

- 2 - A URL parameter like this: http://myserver/myapp/accounts/list?format=xls. The name of the parameter is format by default, but this may be changed. Using a parameter is disabled by default, but when enabled, it is checked second.
- Finally the Accept HTTP header property is checked. This is how HTTP is actually defined to work, but, as previously mentioned, it can be problematic to use.

Java Configuration

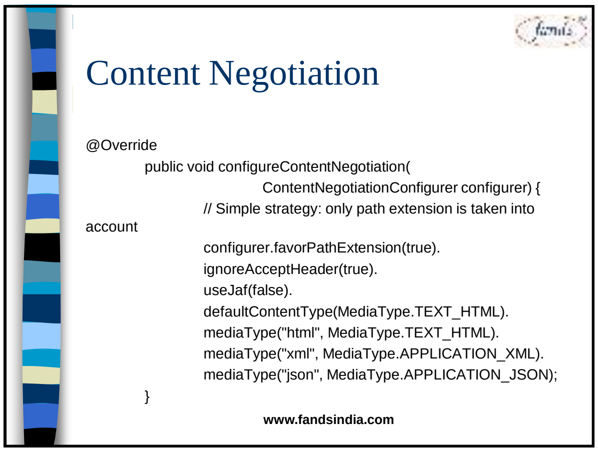
```
@Configuration
@EnableWebMvc
public class WebConfig extends
WebMvcConfigurerAdapter {

  /*  Setup a simple strategy: use all the defaults and
return XML by default when not sure.  */
  @Override
  public void configureContentNegotiation
(ContentNegotiationConfigurer configurer) {
configurer.defaultContentType(MediaType.APPLICATI
ON_XML);
```



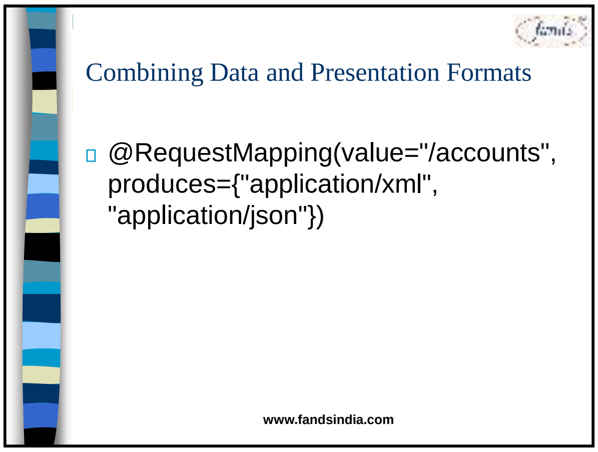
XML

```
<!--  Setup a simple strategy:
         1. Take all the defaults.
         2. Return XML by default when not sure.   -->
 <bean id="contentNegotiationManager"
class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean">
      <property name="defaultContentType"
value="application/xml" />
 </bean>
<!-- Make this available across all of Spring MVC -->
<mvc:annotation-driven content-negotiation-
manager="contentNegotiationManager" />
```



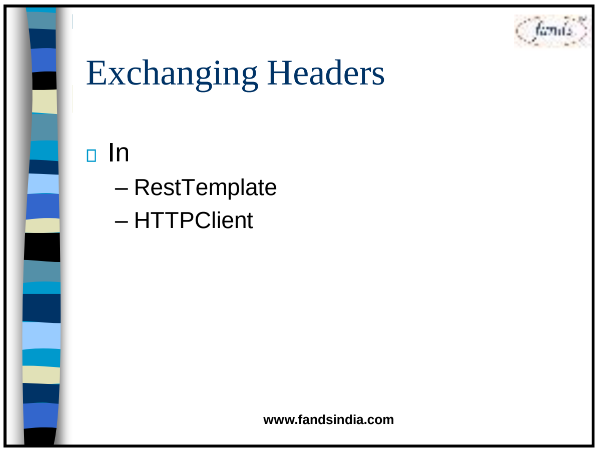
Content Negotiation

```
@Override
          public void configureContentNegotiation(
                              ContentNegotiationConfigurer configurer) {
                    // Simple strategy: only path extension is taken into
account
                    configurer.favorPathExtension(true).
                    ignoreAcceptHeader(true).
                    useJaf(false).
                    defaultContentType(MediaType.TEXT_HTML).
                    mediaType("html", MediaType.TEXT_HTML).
                    mediaType("xml", MediaType.APPLICATION_XML).
                    mediaType("json", MediaType.APPLICATION_JSON);
          }
```



Combining Data and Presentation Formats

- @RequestMapping(value="/accounts", produces={"application/xml", "application/json"})



Exchanging Headers

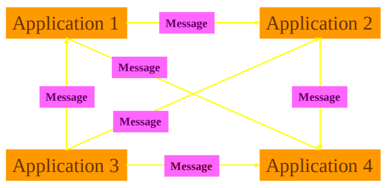
- In
  – RestTemplate
  – HTTPClient

JMS

What is a Messaging System?

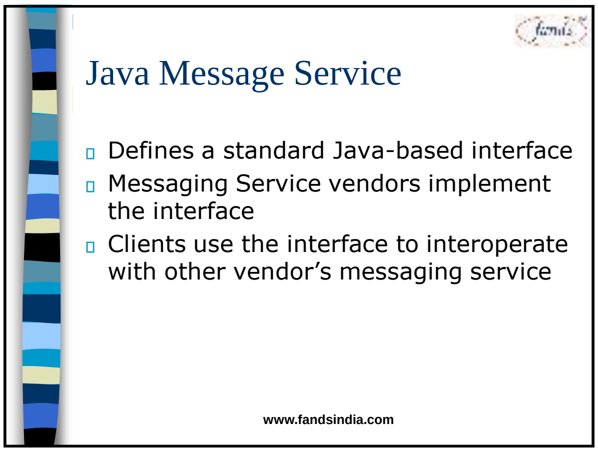
- A system that allows separate, uncoupled applications to reliably communicate asynchronously
- Supports peer-to-peer communication between components

Application Distribution



Problems?

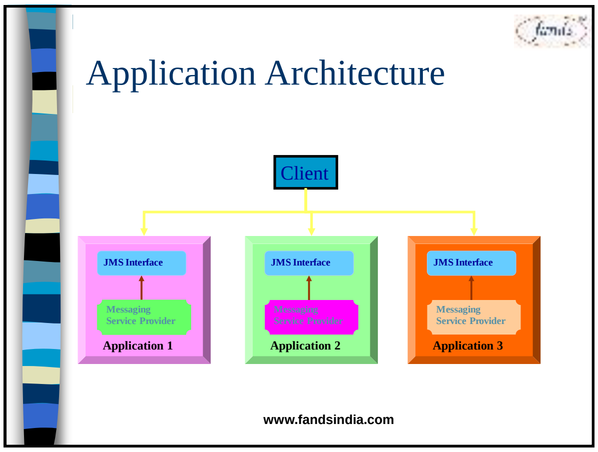
- Vendor specific Messaging API
- Non-interoperable
- A standard API required for interoperability



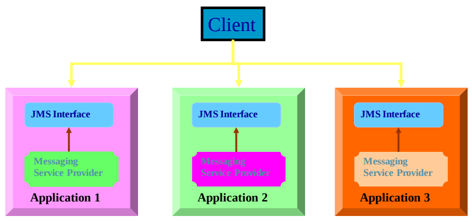
Java Message Service

- Defines a standard Java-based interface
- Messaging Service vendors implement the interface
- Clients use the interface to interoperate with other vendor's messaging service

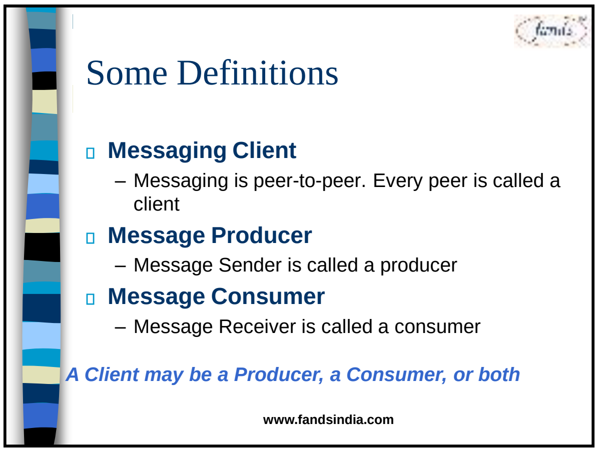
www.fandsindia.com



Application Architecture



www.fandsindia.com

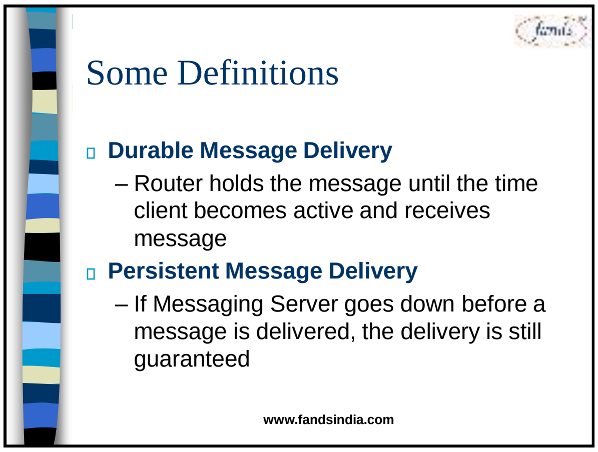


Some Definitions

- **Messaging Client**
  - Messaging is peer-to-peer. Every peer is called a client
- **Message Producer**
  - Message Sender is called a producer
- **Message Consumer**
  - Message Receiver is called a consumer

A Client may be a Producer, a Consumer, or both

www.fandsindia.com



Some Definitions

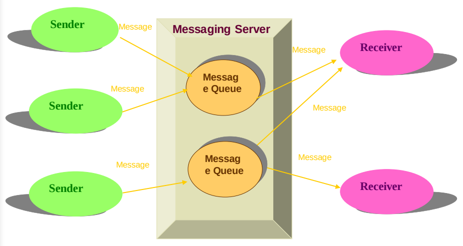
- **Durable Message Delivery**
  - Router holds the message until the time client becomes active and receives message
- **Persistent Message Delivery**
  - If Messaging Server goes down before a message is delivered, the delivery is still guaranteed

www.fandsindia.com

Communication Models

- **Point-To-Point (PTP)**
  – Used if there is one and only one receiver for each message
- **Publish-Subscribe (PS)**
  – Used for general broadcast kind of applications

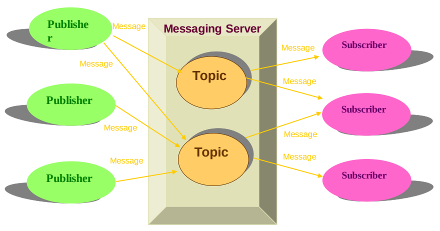
Point-To-Point Model



PTP Features

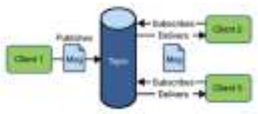
- Producer sends Messages to the queue
- Consumer receives message from the queue
- Multiple senders may send messages to same queue
- One and only one consumer for every message
- System may provide multiple queues.

Publish-Subscribe Model



PS Features

- Producer publishes messages to a topic
- Consumer subscribes to interested topic
- A message may be consumed by multiple subscribers
- Multiple publishers and subscribers to a single topic



Message Delivery Model

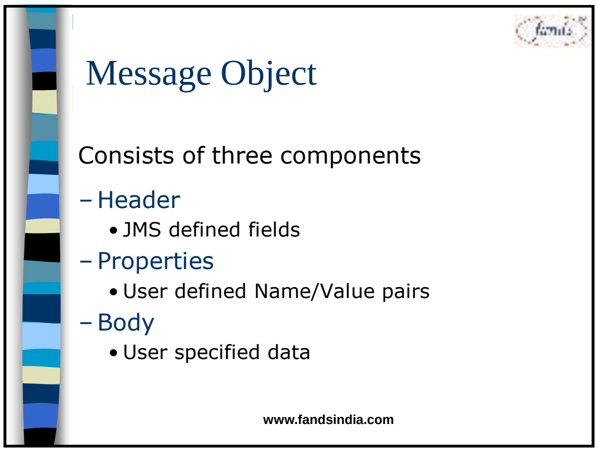
- **Synchronous**
  - Consumer is blocked until
    - Message is received
    - Timeout set by consumer expires
    - Consumer closes
- **Asynchronous**
  - Messaging system responsible for callback on consumer

Steps for using PTP Model

- Obtain a reference to *QueueConnectionFactory* or instantiate the provider specific class
- Open a communication session
- Obtain a reference to *Queue* object or Create a Queue object
- Send/Receive messages on the queue

Steps for using PS Model

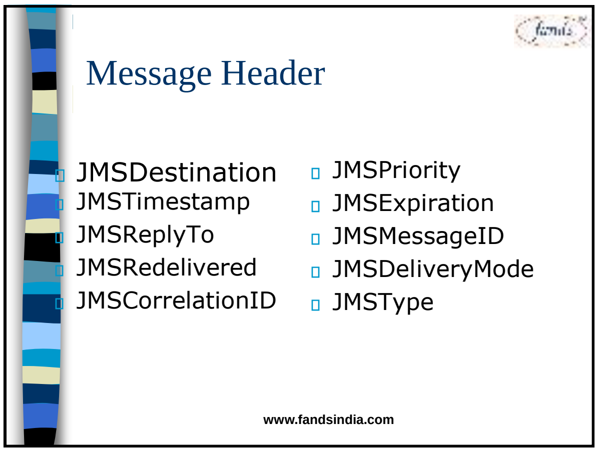
- Obtain a reference to *TopicConnectionFactory* or instantiate the provider specific class
- Open a communication session
- Obtain a reference to *Topic* object or Create a Topic object
- *Publish*/*Subscribe* messages for the topic



Message Object

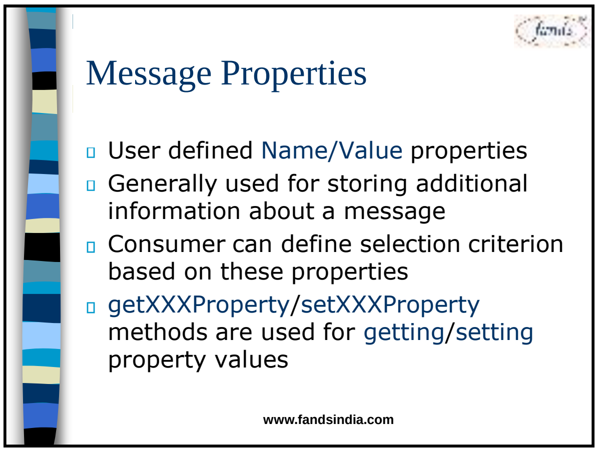
Consists of three components

- Header
  - JMS defined fields
- Properties
  - User defined Name/Value pairs
- Body
  - User specified data



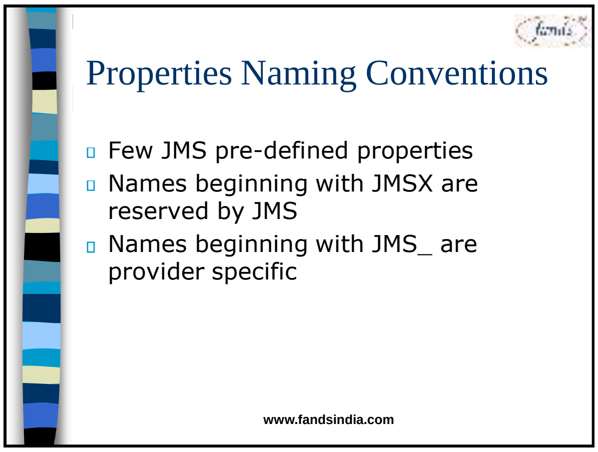
Message Header

- JMSDestination
- JMSTimestamp
- JMSReplyTo
- JMSRedelivered
- JMSCorrelationID
- JMSPriority
- JMSExpiration
- JMSMessageID
- JMSDeliveryMode
- JMSType



Message Properties

- User defined Name/Value properties
- Generally used for storing additional information about a message
- Consumer can define selection criterion based on these properties
- getXXXProperty/setXXXProperty methods are used for getting/setting property values



Properties Naming Conventions

- Few JMS pre-defined properties
- Names beginning with JMSX are reserved by JMS
- Names beginning with JMS_ are provider specific

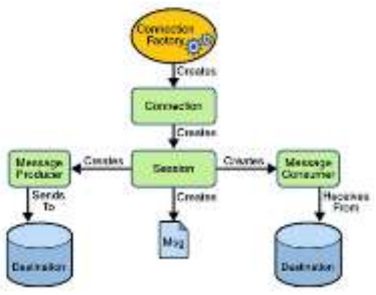
Message Body

- Encapsulates data
- JMS provides several pre-defined message types

Message Types

- TextMessage
  - Represents String messages
- MapMessage
  - Provides Name/Value pairs
- BytesMessage
  - Stream of bytes
- StreamMessage
  - Series of primitive data types
- ObjectMessage
  - Serialized Java object

Messaging Model



Spring and JMS

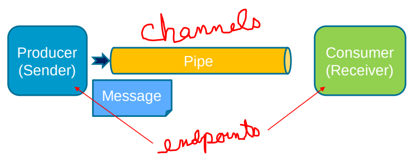
www.fandsindia.com

Spring Integration

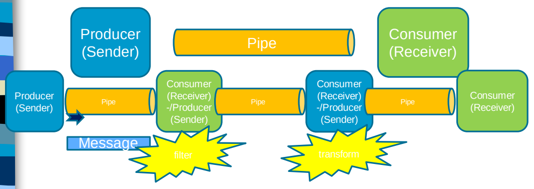
www.fandsindia.com

Spring Integration

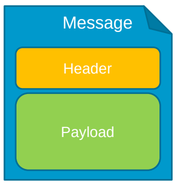
The main components



Spring Integration Applications



Messages



Message Endpoints

- Adapters (connect your channel to some other system)
- Filter (remove some messages from channels based on header, content, etc.)
- Transformer (convert a message content or structure)
- Enricher (add content to the message header or payload)
- Service activator (invoke service operations based on the arrival of a message)
- Gateway (connect your channels without SI coupling)

Message Channels

- Two general classifications of message channels
  - **Pollable Channel**
  - **Subscribable Channel**
- While there are many subtypes, they all implement at least one of these SI channel interfaces
  - see http://docs.spring.io/spring-integration/reference/html/messaging-channels-section.html

Pollable Channels

- May buffer its messages
  - Requires a queue to hold the messages
  - The queue has a designated capacity
- Waits for the consumer to get the messages
  - Consumers actively poll to receive messages
- Typically a point-to-point channel
  - Only one receiver of a message in the channel
- Usually used for sending information or "document" messages between endpoints

Subscribable

- Allows multiple subscribers (or consumers) to register for its messages.
  - Messages are delivered to all registered subscribers on message arrival
  - It has to manage a list or registry of subscribers.
- Doesn't buffer its messages
- Usually used for "event" messages
  - Notifying the subscribers that something happened and to take appropriate action.

Lab – Channels – Setup

- Pom
  - spring-boot-starter-integration
  - spring-integration-stream
- Application.java
  - @ImportResource("mycomponents.xml")
  - public class Startup {
  - Main method
    - new SpringApplication(Startup.class).run(args);
    - While(true){}
- Mycomponents.xml

www.fandsindia.com

Lab – Channels – Working

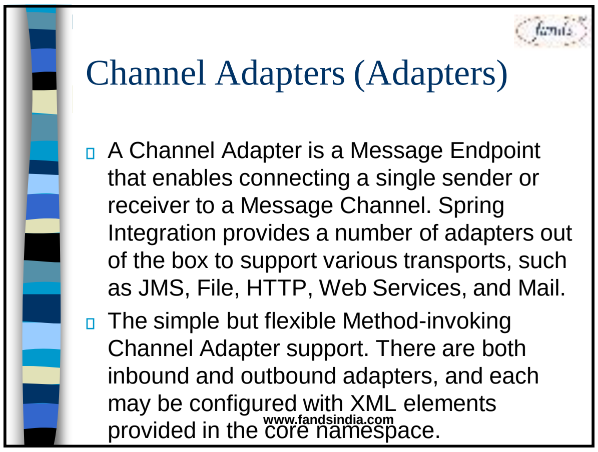
- *Replace the Subscribable Channel with a Pollable Channel -> Test*
- Remove the Producer and Consumer. In order to see the buffering of messages, it is necessary to change the producers and consumers
  - Comment out producer/consumer
  - Send bulk messages
    - MessageChannel channel = context.getBean("messageChannel", MessageChannel.class); Message<String> message1 = MessageBuilder.withPayload("Hello world - one!").build();
  - Test

www.fandsindia.com

Lab – Channels – Working

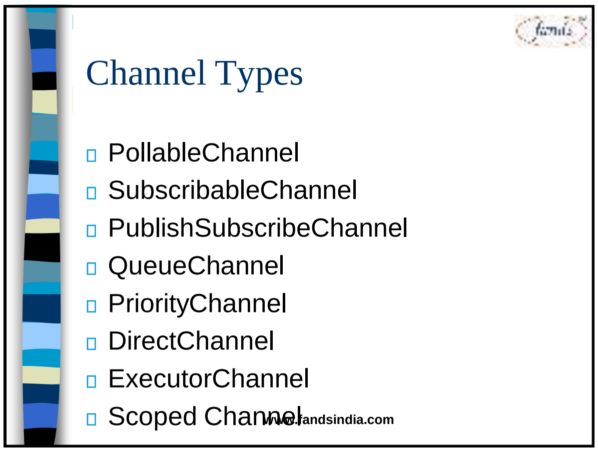
- *Replace the Pollable Channel with a Direct Channel*
  - remove the queue from the pollable channel (the "messageChannel") in order to create a direct channel. Direct channels are the default implementation for <int:channel>

www.fandsindia.com



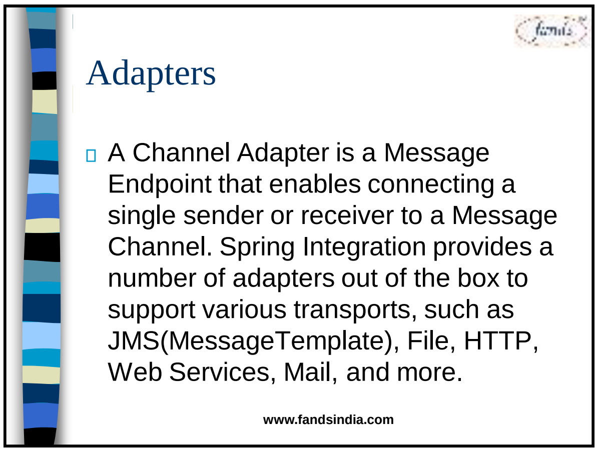
Channel Adapters (Adapters)

- A Channel Adapter is a Message Endpoint that enables connecting a single sender or receiver to a Message Channel. Spring Integration provides a number of adapters out of the box to support various transports, such as JMS, File, HTTP, Web Services, and Mail.
- The simple but flexible Method-invoking Channel Adapter support. There are both inbound and outbound adapters, and each may be configured with XML elements provided in the core namespace.



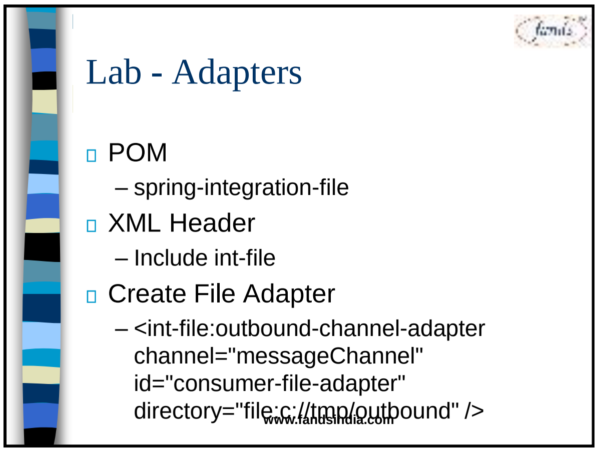
Channel Types

- PollableChannel
- SubscribableChannel
- PublishSubscribeChannel
- QueueChannel
- PriorityChannel
- DirectChannel
- ExecutorChannel
- Scoped Channel



Adapters

- A Channel Adapter is a Message Endpoint that enables connecting a single sender or receiver to a Message Channel. Spring Integration provides a number of adapters out of the box to support various transports, such as JMS(MessageTemplate), File, HTTP, Web Services, Mail, and more.



Lab - Adapters

- POM
  - spring-integration-file
- XML Header
  - Include int-file
- Create File Adapter
  - <int-file:outbound-channel-adapter channel="messageChannel" id="consumer-file-adapter" directory="file:c://tmp/outbound" />

Lab - Adapters

- Replace inbound adapter
  - <int-file:inbound-channel-adapter id="producer-file-adapter" channel="messageChannel" directory="file:c://tmp/inbound" prevent-duplicates="true"> <int:poller fixed-rate="5000" /> </int-file:inbound-channel-adapter>
- MyIntegration.xml

Filters

- <int:filter ref="selector" output-channel="outboundChannel" input-channel="inboundChannel"/>
- <bean id="selector" class="com.intertech.lab3.FileSelector"/>
- public class FileSelector implements MessageSelector {

```
      public boolean accept(Message<?> message) {
      if (message.getPayload() instanceof File
      && ((File) message.getPayload()).getName().startsWith("msg")) {
            return false;
      }
      return true;
```

Filters for XML

- <int-xml:xpath-filter>
- <int-xml:xpath-expression>
- <int-xml:validating-filter schema-location=“po.xsd">

Splitters and Aggregators

- The Splitter is a component whose role is to partition a message in several parts, and send the resulting messages to be processed independently. Aggregator is a type of Message Handler that receives multiple Messages and combines them into a single Message

Spring Batch

www.fandsindia.com

Batch Process?

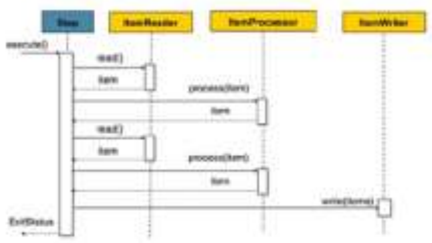
- Bulk Processessing
- Long Running Processes
- Mostly Sequential Processes
- Routine - onetime, daily, monthly, yearly, ...

"The lack of a standard, reusable batch architecture has resulted in the proliferation of many one-off, in-house solutions developed within client enterprise IT functions."

- spring batch documentation

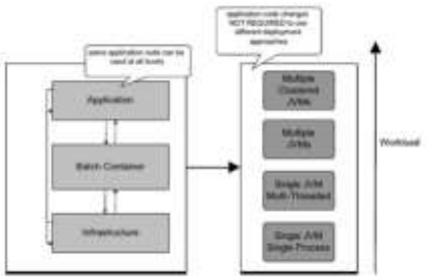
www.fandsindia.com

Item Oriented Processing

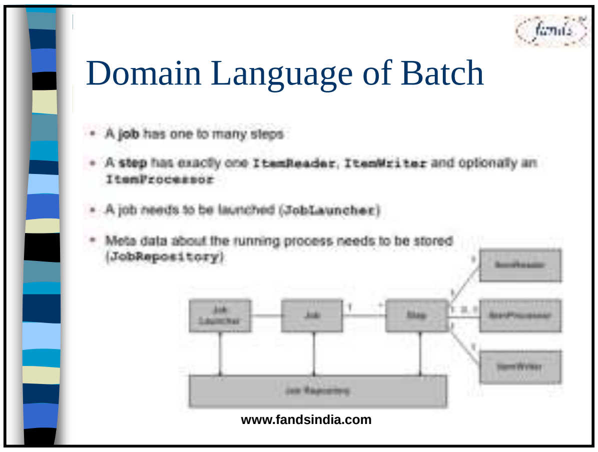


www.fandsindia.com

Layered Architecture


same application code can be used at all levels
application code changes NOT REQUIRED to use different deployment approaches
Application
Batch Container
Infrastructure
Multiple Clustered JVMs
Multiple JVMs
Single JVM Multi-Threaded
Single JVM Single Process
Workload

www.fandsindia.com

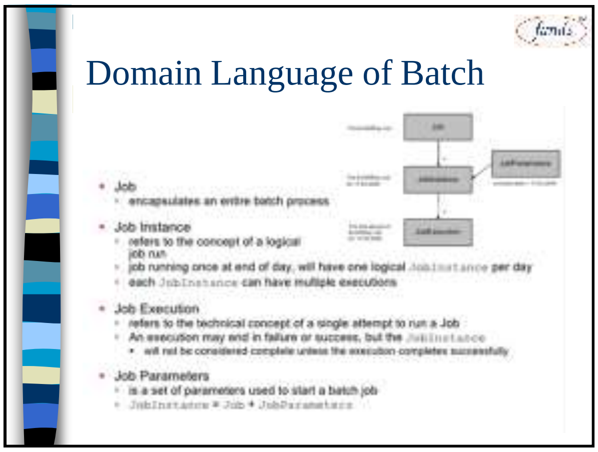


Domain Language of Batch

- A **job** has one to many steps
- A **step** has exactly one `ItemReader`, `ItemWriter` and optionally an `ItemProcessor`
- A job needs to be launched (`JobLauncher`)
- Meta data about the running process needs to be stored (`JobRepository`)

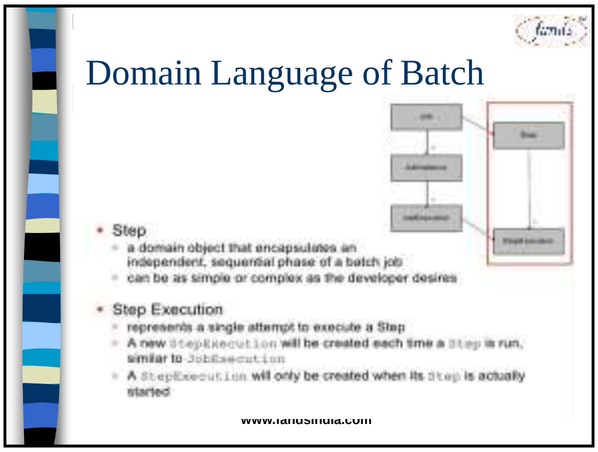
Job Launcher
Job
1
*
Step
1
1
1..1
1
1
ItemReader
ItemProcessor
ItemWriter
Job Repository

www.fandsindia.com



Domain Language of Batch

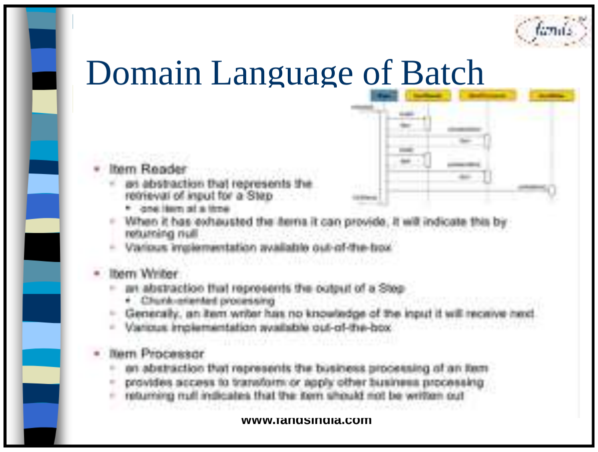
- Job
  - encapsulates an entire batch process
- Job Instance
  - refers to the concept of a logical job run
  - job running once at end of day, will have one logical `JobInstance` per day
  - each `JobInstance` can have multiple executions
- Job Execution
  - refers to the technical concept of a single attempt to run a Job
  - An execution may end in failure or success, but the `JobInstance`
    - will not be considered complete unless the execution completes successfully
- Job Parameters
  - is a set of parameters used to start a batch job
  - `JobInstance = Job + JobParameters`



Domain Language of Batch

- Step
  - a domain object that encapsulates an independent, sequential phase of a batch job
  - can be as simple or complex as the developer desires
- Step Execution
  - represents a single attempt to execute a Step
  - A new `StepExecution` will be created each time a `Step` is run, similar to `JobExecution`
  - A `StepExecution` will only be created when its `Step` is actually started

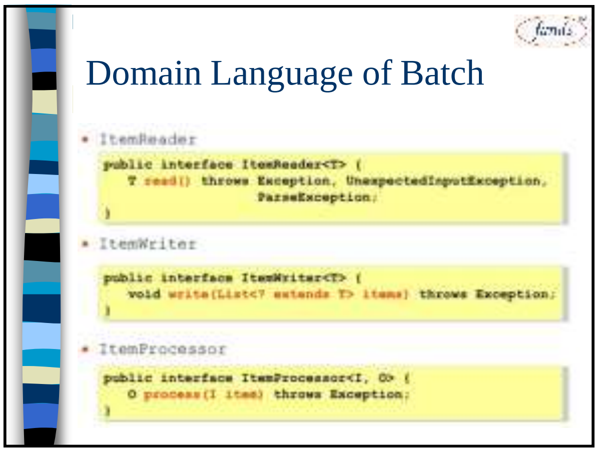
www.fandsindia.com



Domain Language of Batch

- Item Reader
  - an abstraction that represents the retrieval of input for a Step
    - one item at a time
  - When it has exhausted the items it can provide, it will indicate this by returning null
  - Various implementation available out-of-the-box
- Item Writer
  - an abstraction that represents the output of a Step
    - Chunk-oriented processing
  - Generally, an item writer has no knowledge of the input it will receive next
  - Various implementation available out-of-the-box
- Item Processor
  - an abstraction that represents the business processing of an item
  - provides access to transform or apply other business processing
  - returning null indicates that the item should not be written out

www.fandsindia.com



Domain Language of Batch

- ItemReader

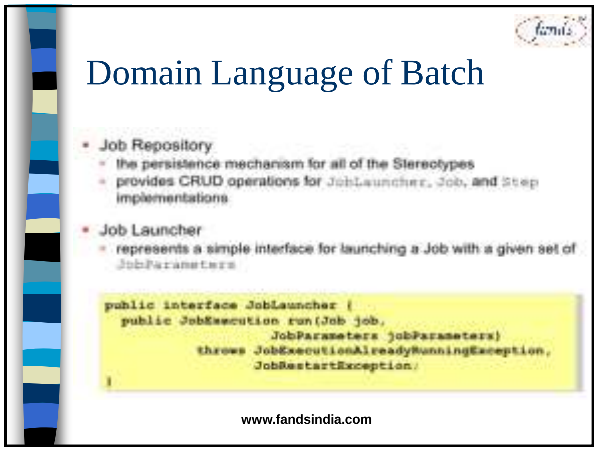
```
public interface ItemReader<T> {
    T read() throws Exception, UnexpectedInputException,
                    ParseException;
}
```

- ItemWriter

```
public interface ItemWriter<T> {
    void write(List<? extends T> items) throws Exception;
}
```

- ItemProcessor

```
public interface ItemProcessor<I, O> {
    O process(I item) throws Exception;
}
```



Domain Language of Batch

- Job Repository
  - the persistence mechanism for all of the Stereotypes
  - provides CRUD operations for JobLauncher, Job, and Step implementations
- Job Launcher
  - represents a simple interface for launching a Job with a given set of JobParameters

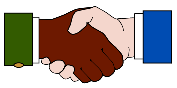
```
public interface JobLauncher {
  public JobExecution run(Job job,
                          JobParameters jobParameters)
              throws JobExecutionAlreadyRunningException,
                     JobRestartException;
}
```

www.fandsindia.com

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com