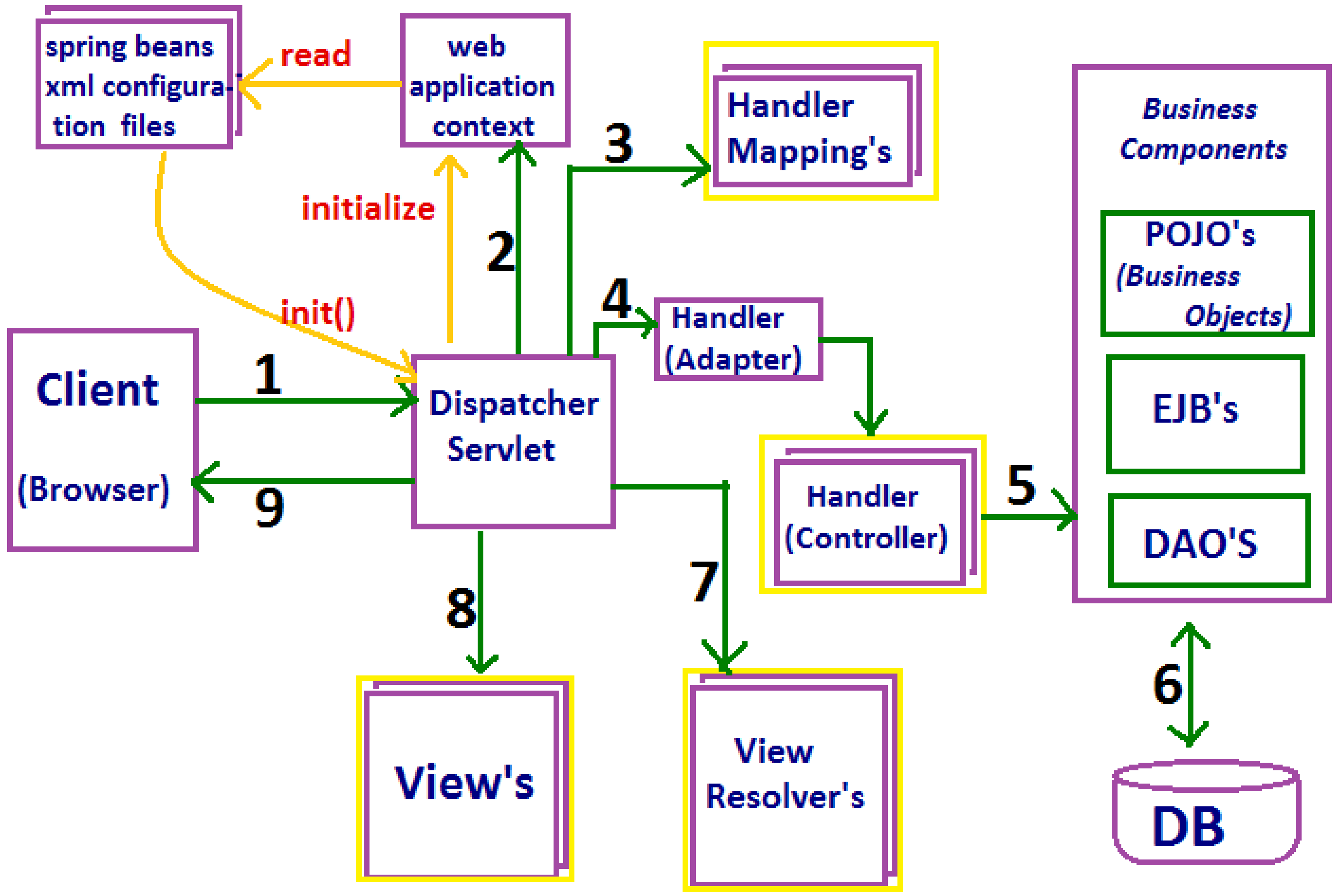
My Project Title is : VISTA

Spring Web MVC Architecture :

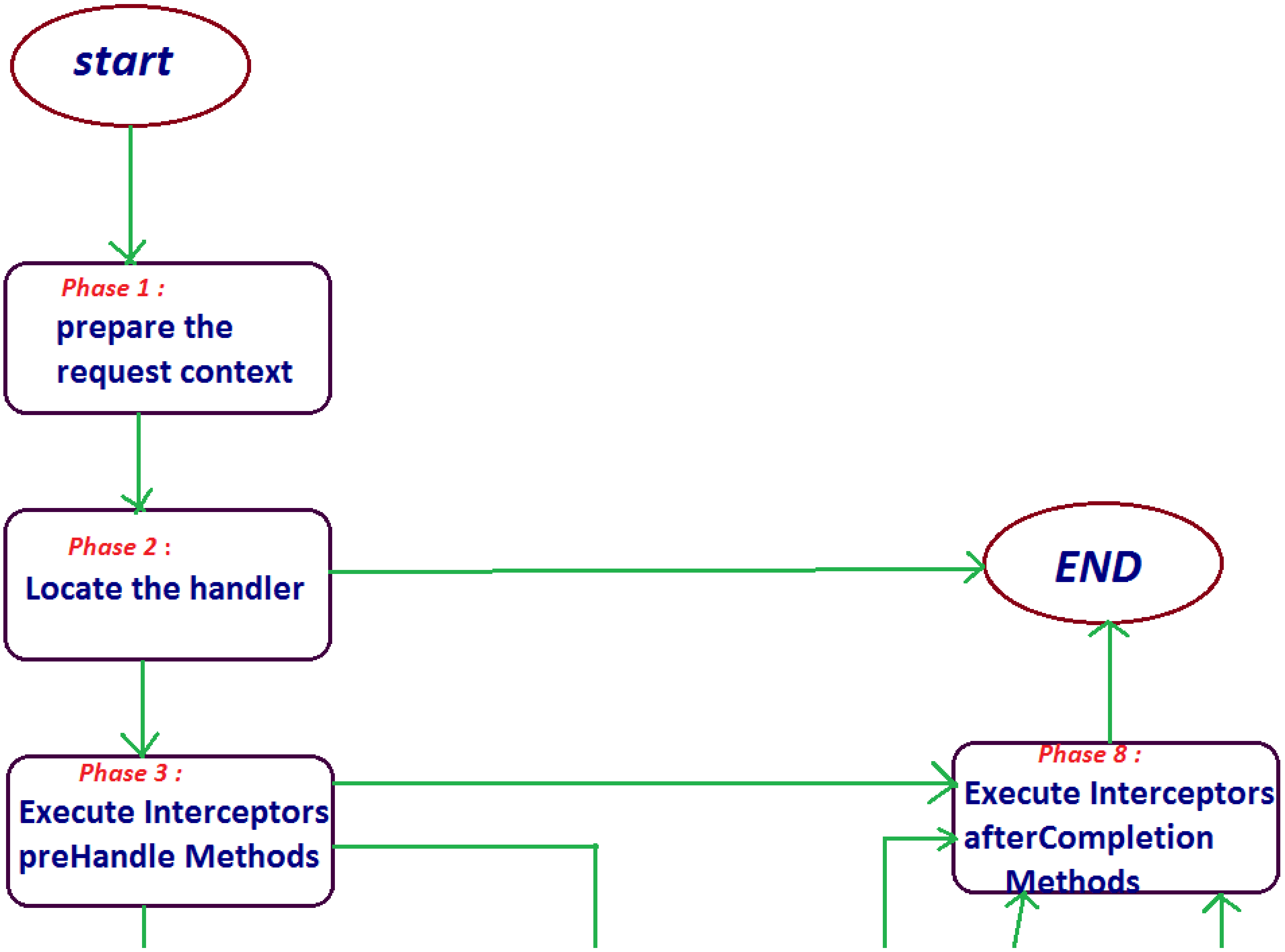
- According to spring MVC , DispatcherServlet is the front controller for the spring Web MVC application, providing a centralized access for various requests to the application and collaborating with various other objects to complete the request handling and present the response to client.
- DispatcherServlet uses the WebApplicationContext object to locate the various objects configured in the Spring Bean XML configuration file.
- The WebApplicationContext is instantiated and initialized as a part of the DispatcherServlet's initialization process.
- The WebApplicationContext object is responsible to locate the spring beans XML configuration file and load its details to prepare the context for handling the requests.

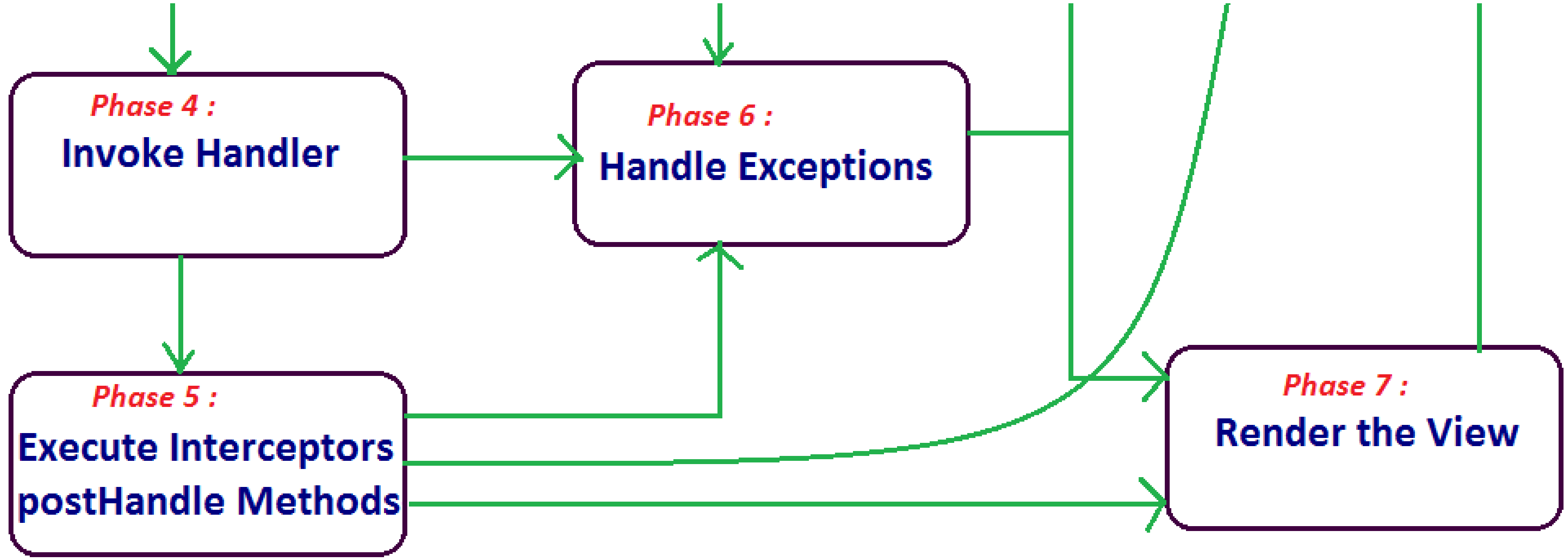


Now, when a client request is given to DispatcherServlet, it performs the following operations:

Types of Phases :

1. Prepare the request context
2. Locate The Handler
3. Execute Interceptors preHandle methods
4. Invoke Handler
5. Execute Interceptors postHandle methods
6. Handle Exceptions
7. Render the View
8. Execute Interceptors afterCompletion methods


start
Phase 1 :
prepare the request context
Phase 2 :
Locate the handler
END
Phase 3 :
Execute Interceptors preHandle Methods
Phase 8 :
Execute Interceptors afterCompletion Methods

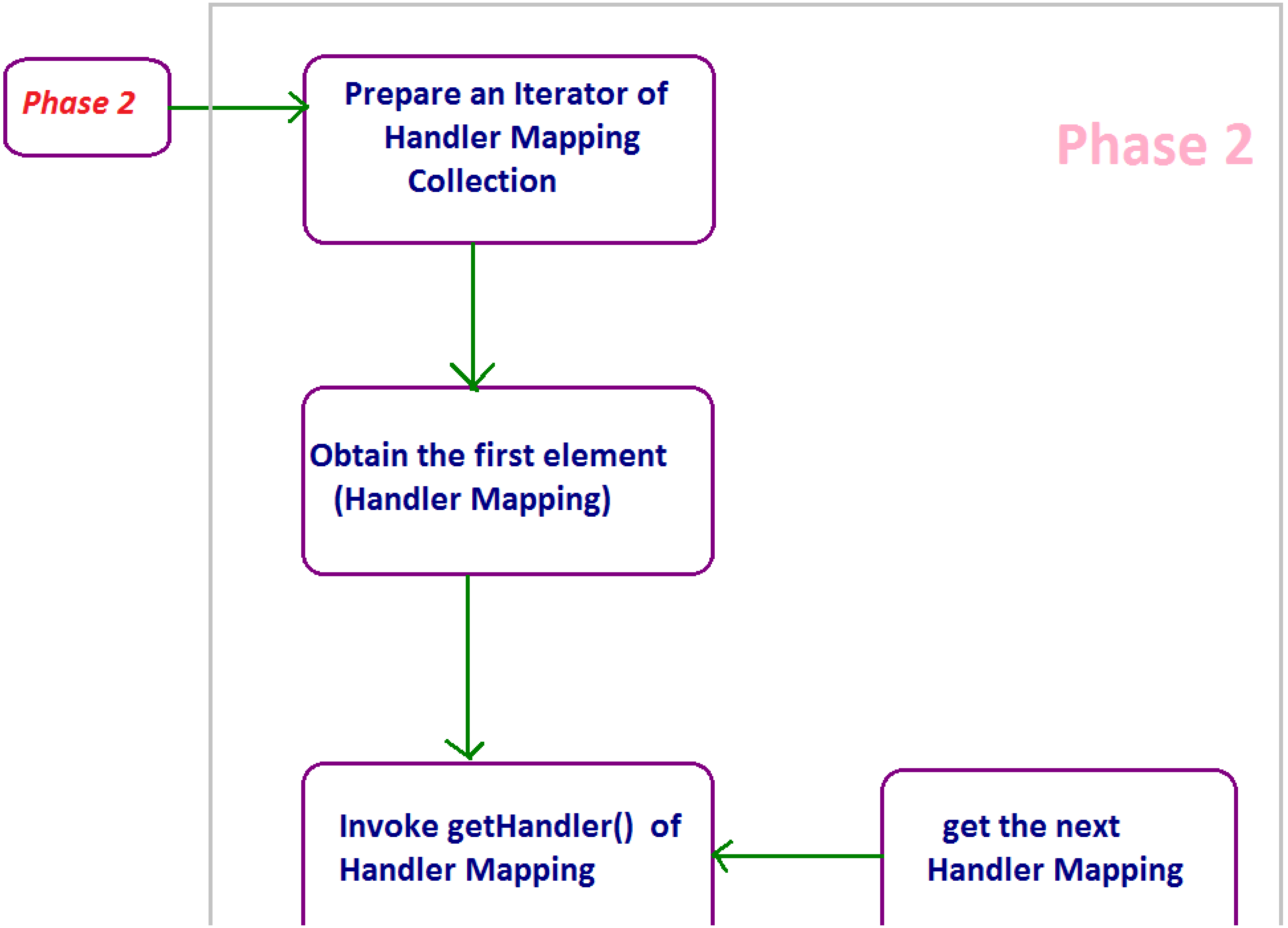


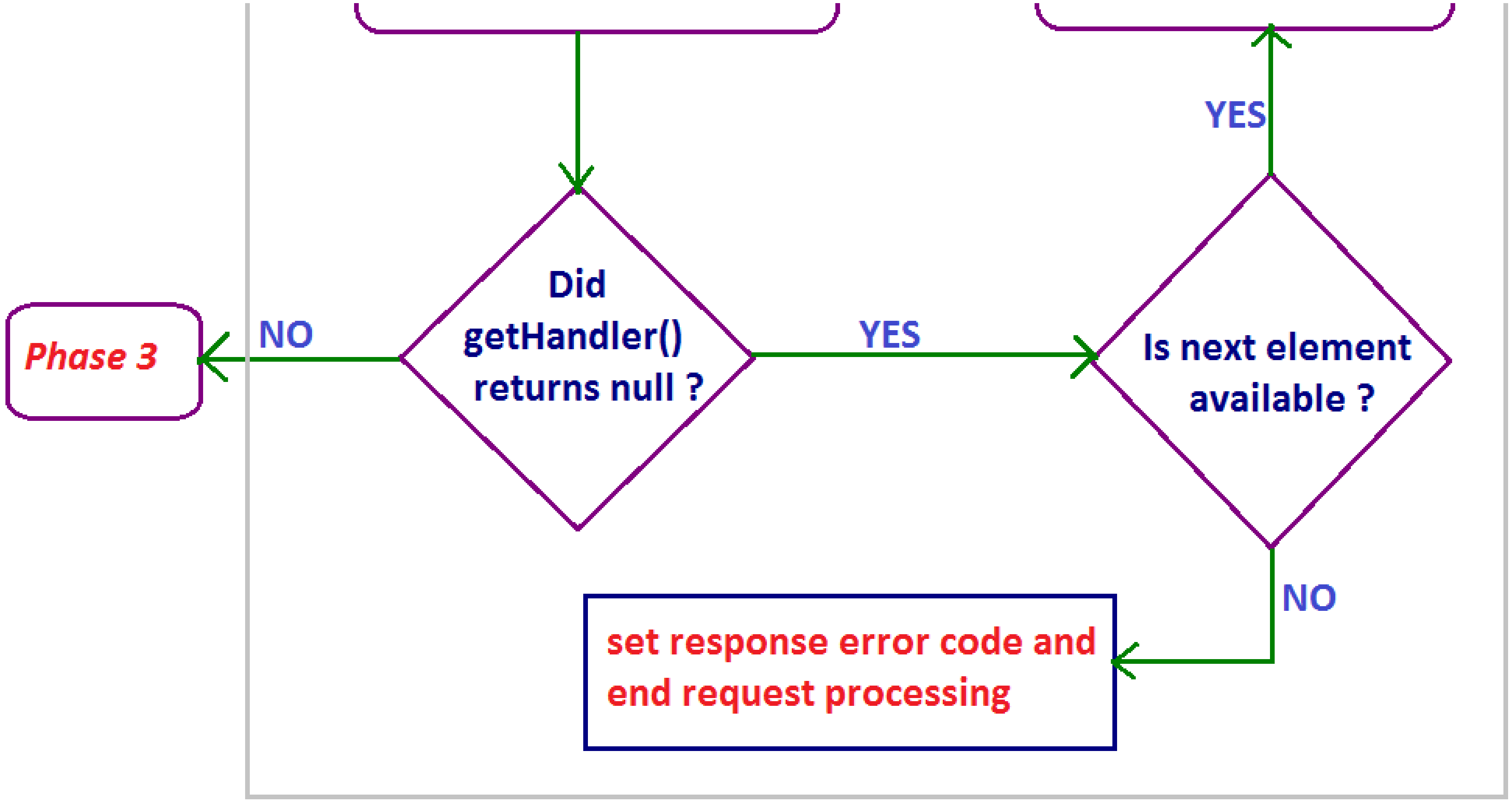
Phase 1 : Prepare the request context

1. DispatcherServlet prepare the request context by setting the framework objects into the request scope.
2. Here framework objects are WebApplicationContext, LocaleResolver, ThemeResolver and ThemeSource.
3. These objects are set into the request scope to make them available to handler and view objects.
4. DispatcherServlet resolves the request using MultipartResolver , so that if the request contains contains multi part data , then it wraps the request in a MultipartHttpRequest type object.
5. In case , if there is any problem in this process the request processing is terminated by throwing an Exception.

6. Once if this process is done successfully the request process workflow continues to the next phase, i.e., Locate the Handler.

Phase 2 : Locate The Handler


Phase 2
Prepare an Iterator of Handler Mapping Collection
Phase 2
Obtain the first element (Handler Mapping)
Invoke getHandler() of Handler Mapping
get the next Handler Mapping


Phase 3
NO
Did getHandler() returns null ?
YES
Is next element available ?
YES
NO
set response error code and end request processing

1. After preparing the request context , the DispatcherServlet locates handler that can handle this request.
2. The DispatcherServlet uses the registered HandlerMapping's and collects the HandlerExecutionChain object.
3. The HandlerExecutionChain object encapsulates the HandlerIntercepter's and the HandlerObject (i.e., controller).
4. Preparing an Iterator object of the Collection, storing the HandlerMapping objects.

5. This Collection object is created while the DispatcherServlet is being initialized, i.e., in its initialization phase.
6. Thereafter, the first element is received from the iterator. This can't be an empty collection since if there is no HandlerMapping configured in the context the DispatcherServlet uses BeanNameUrlHandlerMapping as default HandlerMapping.
7. Invoke the getHandler() of the current HandlerMapping object in the iteration.
   - If getHandler() returns null, then next element is available the next HandlerMapping. Otherwise set response error code and end request processing.
   - If getHandler() returns a valid HandlerExecutionChain object reference then the control delegates to next phase.(i.e., Execute Interceptors preHandle methods)

The HandlerMappings :

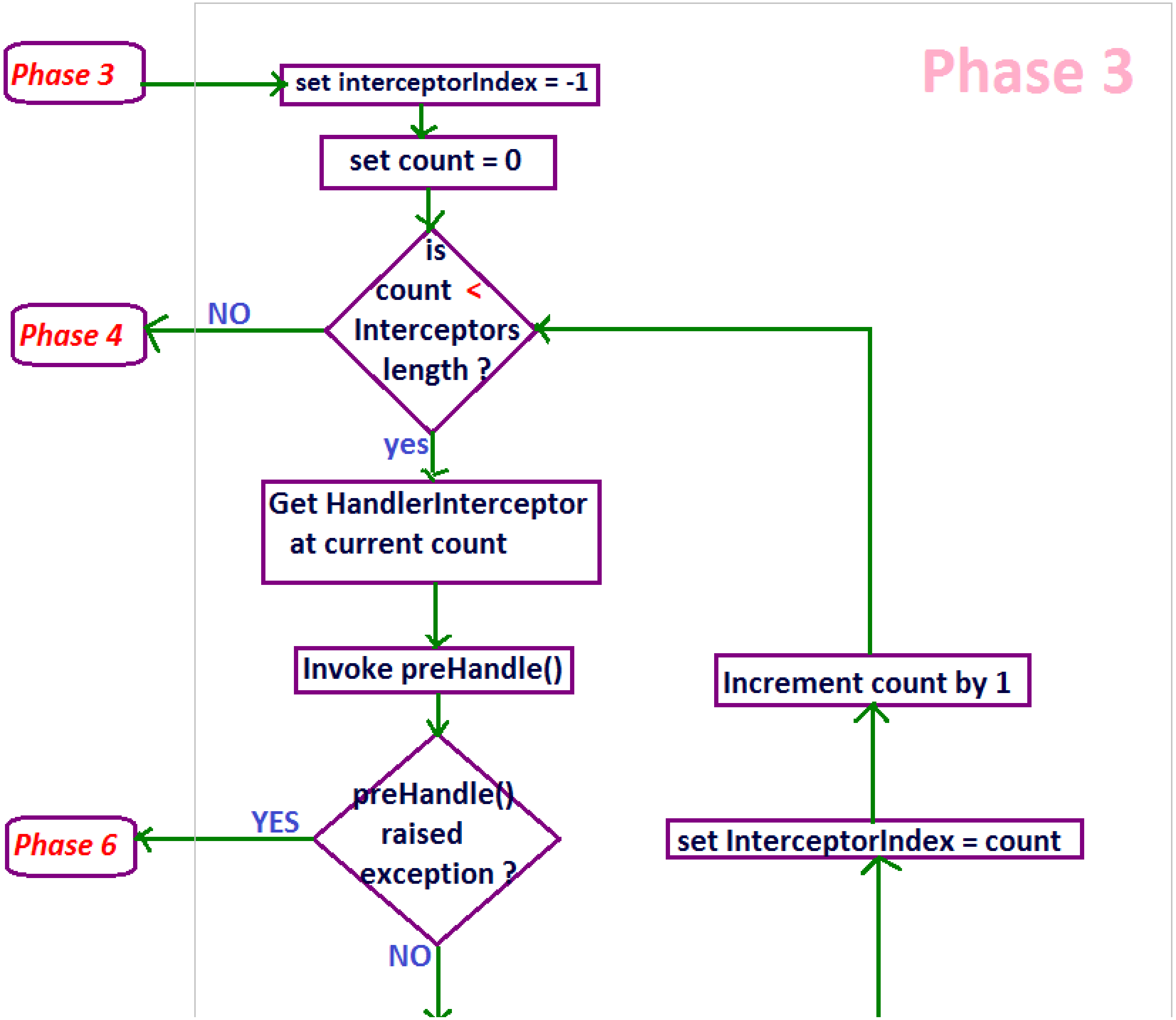
1. The HandlerMapping is responsible for mapping the incoming request to the handler that can handle the request.
2. As discussed in the above section, When the DispatcherServlet receives the request it delegates the request to the HandlerMapping, which identifies the appropriate HandlerExecutionChain that can handle the request.
3. The Spring Web MVC framework provides customizable navigation strategies. Spring provides built-in navigation strategies as determining the handler based on the request URL mapping , which is again based on the bean name.

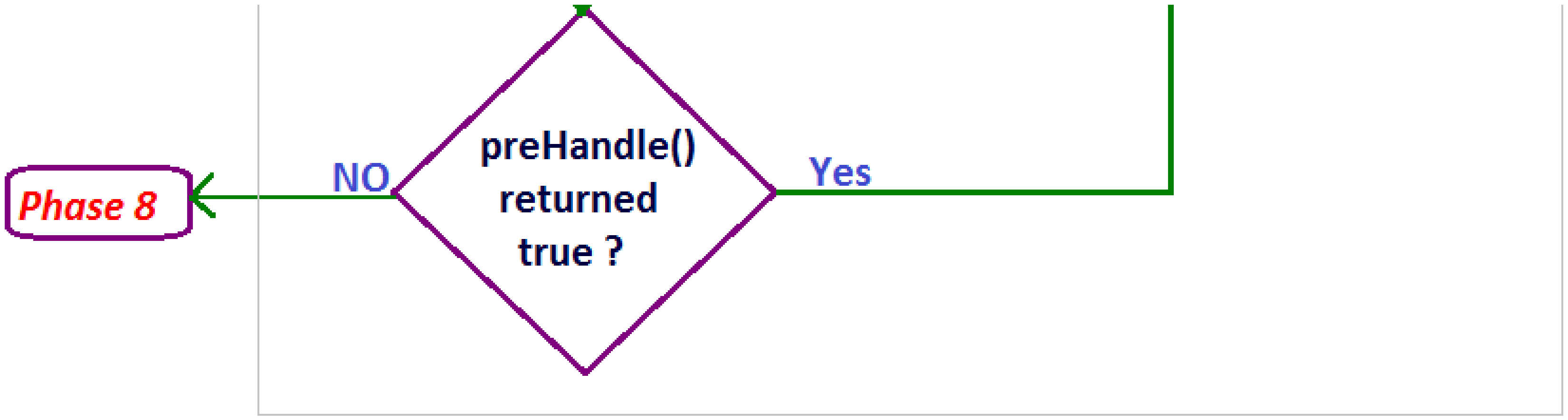
The Spring built-in HandlerMapping implementations are :

1. BeanNameUrlHandlerMapping
2. SimpleUrlHandlerMapping
3. ControllerClassNameHandlerMapping
4. CommonsPathMapHandlerMapping

Phase 3 : Execute Interceptors preHandle methods

1. After successfully locating the HandlerExcecutionChain the DispatcherServlet executes the HandlerInterceptors described by the HandlerExecutionChain returned by the HandlerMapping.
2. The HandlerInterceprors gives us an opportunity to add common pre and post-processing behavior without needing to modify each handler implementation.
3. The HandlerInterceptors is basically similar to a Servlet Filter(2.3v) .


Phase 3
Phase 3
set interceptorIndex = -1
set count = 0
is count < Interceptors length ?
NO
Phase 4
yes
Get HandlerInterceptor at current count
Invoke preHandle()
preHandle() raised exception ?
YES
Phase 6
NO
Increment count by 1
set InterceptorIndex = count



4. The HandlerInterceptors can be used for implementing pre-processing aspects, for example , for authorization checks, or common handler behavior like locale or theme changes.

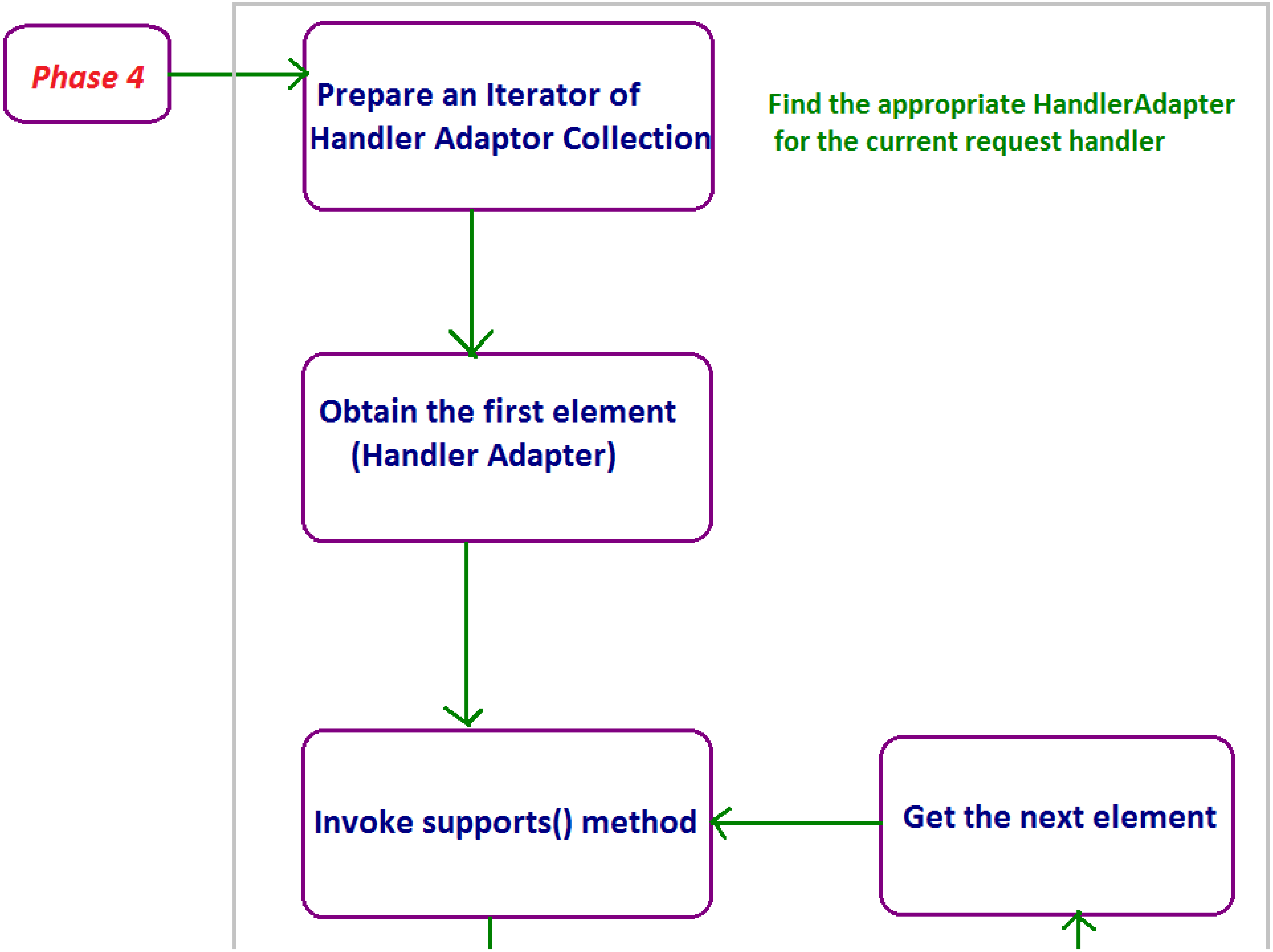
Phase 4 : Invoke Handler

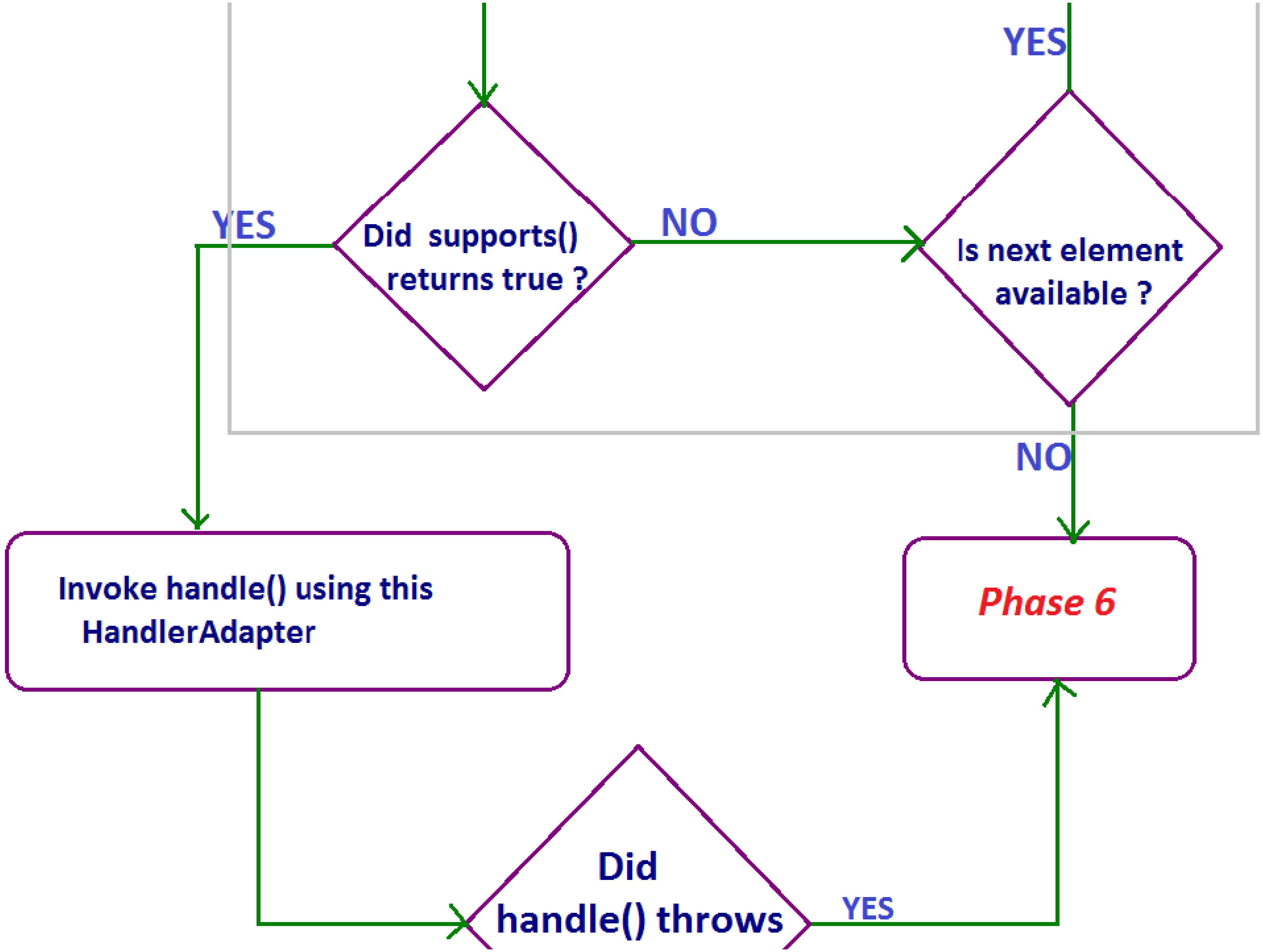
1. In this phase of Spring web MVC request processing workflow the DispatcherServlet delegates the request to the handler that is located by the HandlerMapping in Phase 2.
2. DispatcherServlet uses HandlerAdapter to delegate the request to the handler located to handle this request.

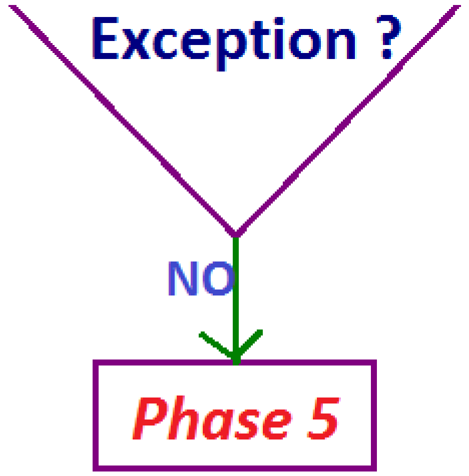
- **step 1 : Prepare a iterator of HandleAdapter collection :**
  1. In this Phase workflow starts with preparing an iterator of the collection representing the HandlerAdapter objects configured in the application.
  2. The handlerAdapters collection is initialized in the initialization phase of the DispatcherServlet where it finds all HandlerAdapters in the ApplicationContext.

   iii. If no HandlerAdapter beans are defined in the application then the default is considered as SimpleControllerHandlerAdapter. Thus the handlerAdapters collection contains at least one element.

- **step 2 : Get HandlerAdapter :**
   i. In this step the workflow obtains the next element from the iterator in step 1 and casts it into HandlerAdapter type reference.
- **step 3 : Find is HandlerAdapter compatible :**
   i. After getting the HandlerAdapter of the current iteration the workflow continues with finding whether this HandlerAdapter is suitable for the handler located in phase 2.
   ii. This is done by invoking supports() method on the HandlerAdapter.
   iii. If it returns "true" then it indicates that this HandlerAdapter supports the handler. In such a case the workflow continues to the next step.
   iv. If the supports() returns "false" then the work flow continues finding whether there is a next element in the handlerAdapter Collection , if found then it moves to step 2. If not then ServletException is thrown, delegating the workflow to phase 6.
- **step 4 : Execute handler :**
   i. After successfully locating the HandlerAdapter that supports the handler that is located in phase 2, the workflow proceeds to invoke handle() method of HandlerAdapter which further delegates the request to the handler(may be controller).
   ii. If the handle() method throws any exception then the workflow proceeds to phase 6 , if not it proceeds to phase 5 after collecting the ModelAndView object reference returned by the handle() method.


Phase 4
Prepare an Iterator of Handler Adaptor Collection
Find the appropriate HandlerAdapter for the current request handler
Obtain the first element (Handler Adapter)
Invoke supports() method
Get the next element


YES
Did supports() returns true ?
NO
Is next element available ?
YES
NO
Invoke handle() using this HandlerAdapter
Phase 6
Did handle() throws
YES



The HandlerAdapter :

1. The HandlerAdapter implementation takes the responsibility of identifying the type of handler and invokes its appropriate methods.
2. The use of HandlerAdapter facilitates us to use Plain Old Java Objects (POJOs) with any method encapsulating the handler behavior , as a handler.
3. Spring provides the built-in HandlerAdapter implementations supporting different types of handlers to work with.
4. The various types of handlers supported by the Spring built-in HandlerAdapter are controller types, HttpRequestHandler types, Servlet types, and ThrowawayController types.
5. The org.springframework.web.servlet.HandlerAdapter interface declares three methods (supports(), handle(), getLastModified()) that have to be implemented by the HandlerAdapter

implementations to help DiapatcherServlet in delegating the request to the handlers.

Spring built-in HandlerAdapter implementations :

1. SimpleControllerHandlerAdapter
2. ThrowawayControllerHandlerAdapter
3. HttpRequestHandlerAdapter
4. SimpleServletHandlerAdapter

Note :

- By default only the SimpleControllerHandlerAdapter and ThrowawayControllerHandlerAdapter handler adapters are available to the DispatcherServlet.
- Other handler adapters need to be explicitly configured, as normal as other beans in the context Spring Beans XML configuration file of DispatcherServlet.
- Even though Spring supports implementing the custom handler adapters that enables handling the request using user defined type of handlers (without making them depend on the Spring API), however it is recommended to use the Spring Controller infrastructure to implement the handler.
- The custom handler adapter option is given to support some situation where we want to use the existing handlers without rewriting them as spring defined controllers.

The HandlerAdapter uses handler to handle the request and results returning the ModelAndView object.

ModelAndView :

- The ModelAndView is a value object designed to hold model and view making it possible for a handler to return both model and view in a single value.
- The ModelAndView object represents a model and view specified by the handler, which is resolved by the DispatcherServlet using the special framework objects as ViewResolver and View.
- The view is an object that can describe a view name in the form of String which will be resolved by a ViewResolver object to locate View object, alternatively, a View object directly.
- The Model is a Map, enabling to specify multiple objects.

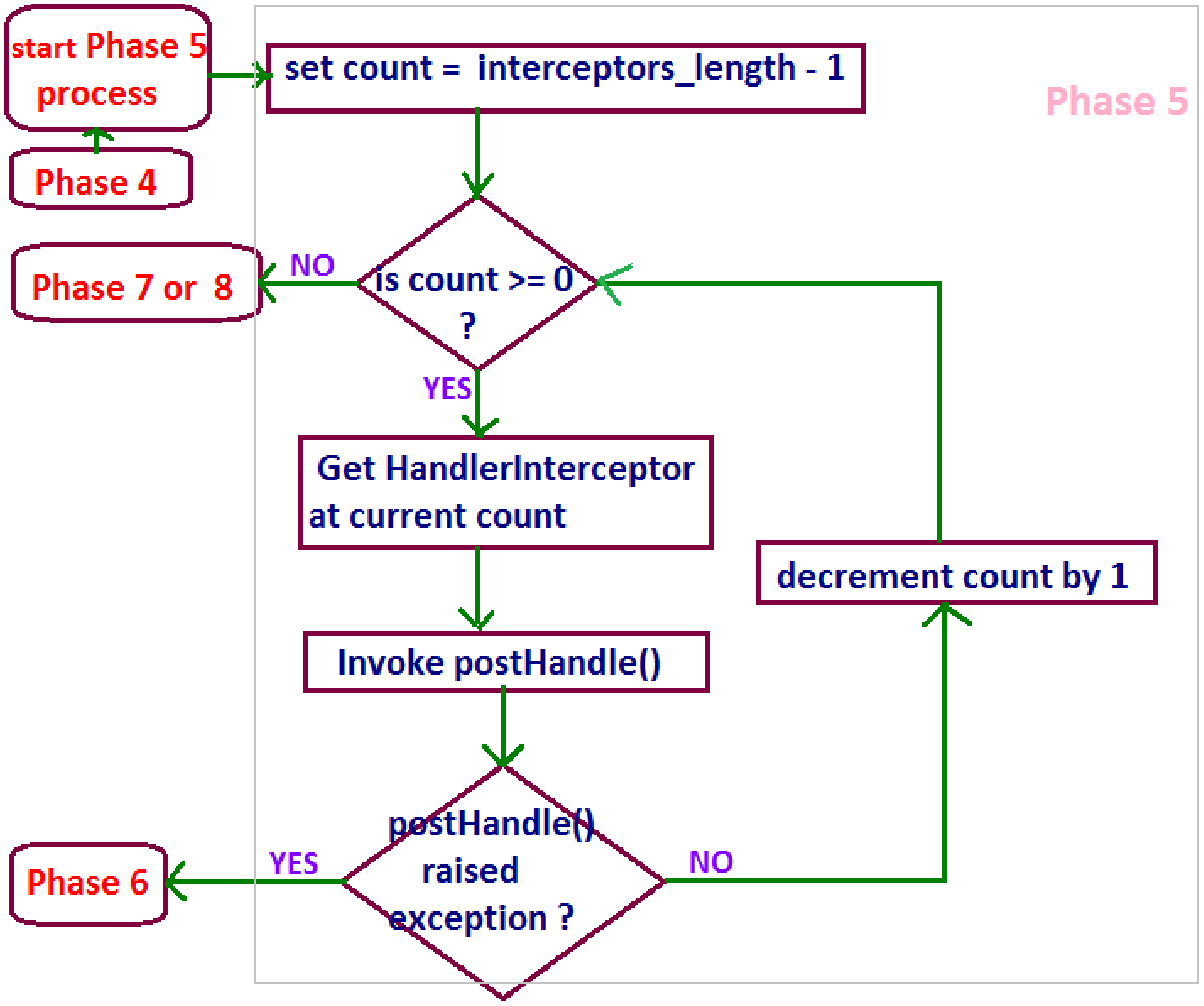
Phase 5 : Execute Interceptors postHandle methods

- HandlerInterceptor gives us an opportunity to add common pre- and post-processing behavior without needing to modify each handler implementation.
- The post-processing operations like changing the logical view name in the ModelAndView based on some inputs/output to support different types of views.

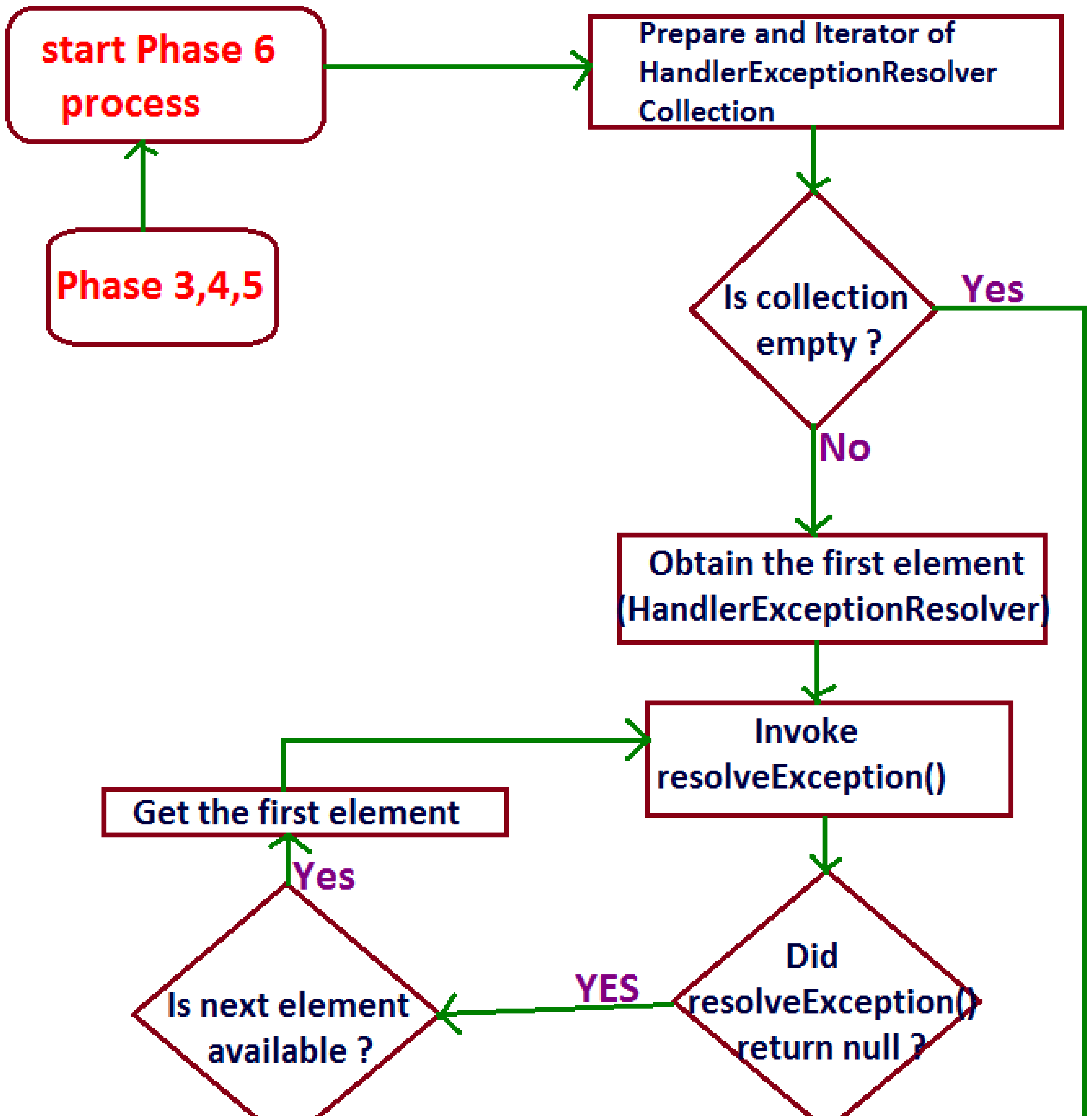
Step 1 : Prepare the Counter :

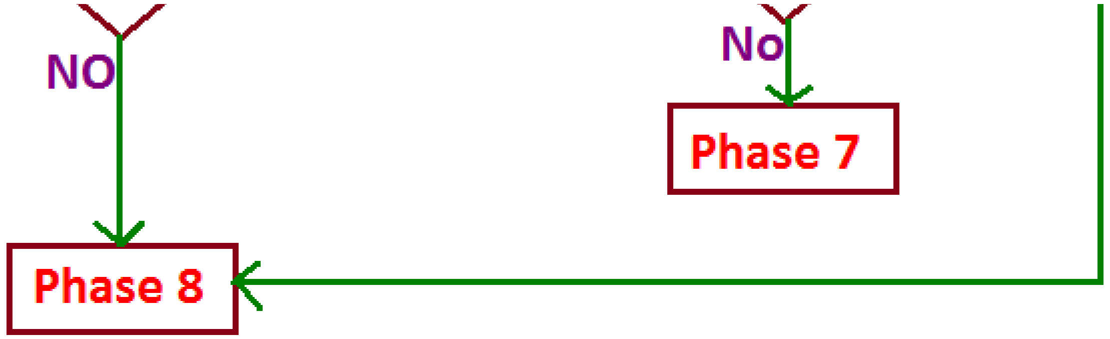
- In this phase workflow starting with setting the count to one minus the interceptor's length so that the postHandle method can be invoked on the interceptor's in the reverse order.
- If the

Step 2 : Invoke postHandle() method :

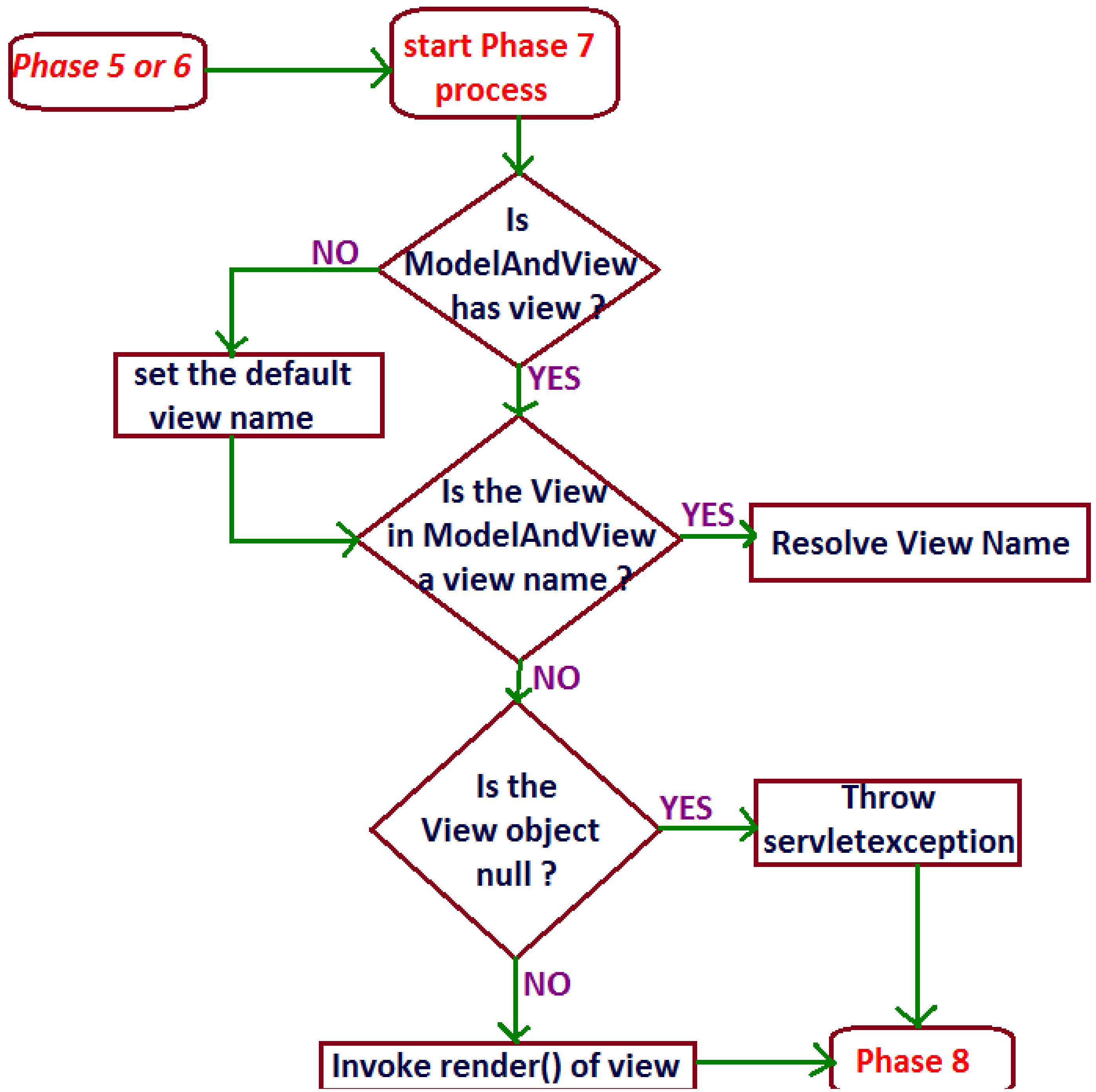


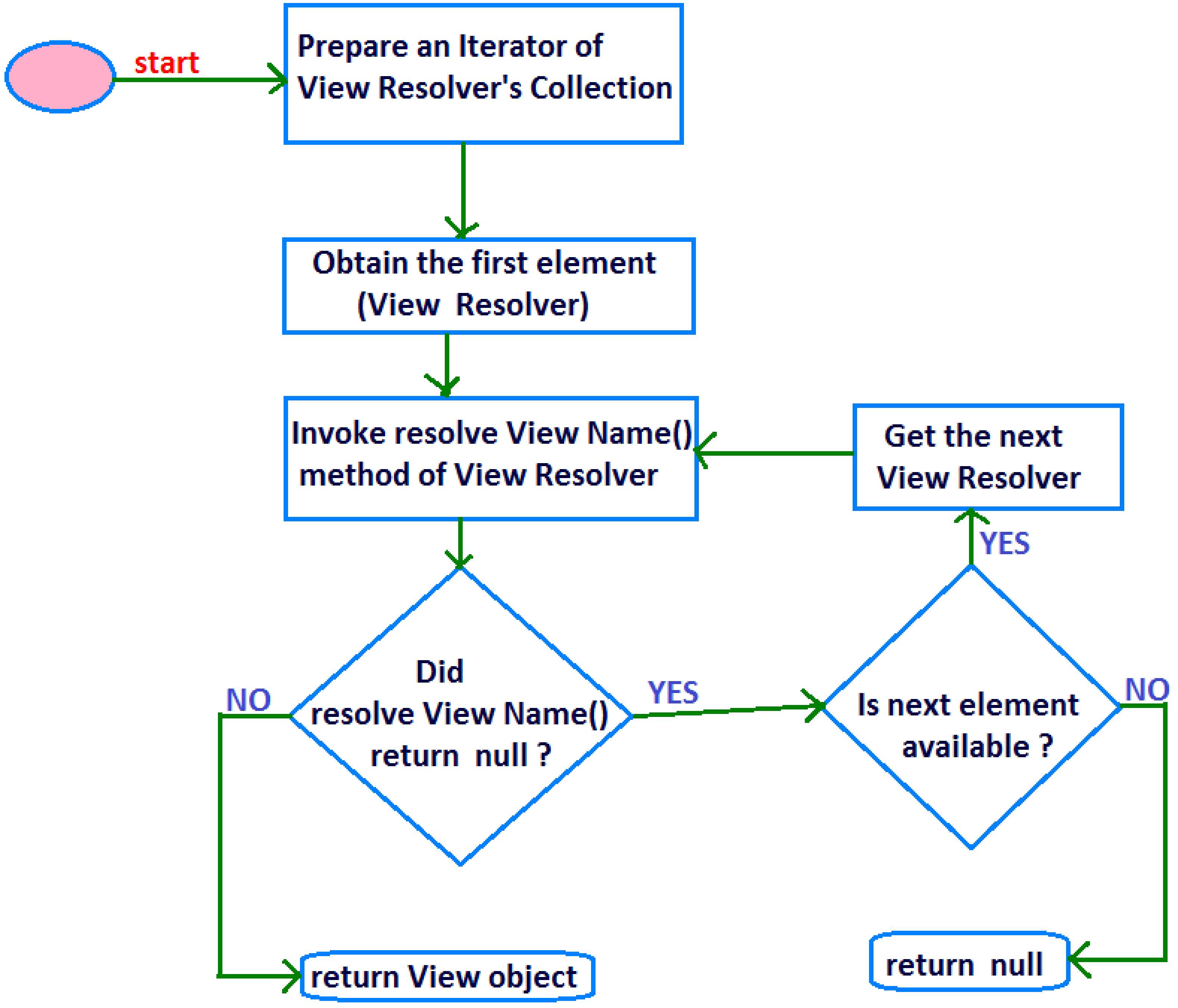
Phase 6 : Handle Exceptions


start Phase 6 process
Phase 3,4,5
Prepare and Iterator of HandlerExceptionResolver Collection
Is collection empty ?
Yes
No
Obtain the first element (HandlerExceptionResolver)
Invoke resolveException()
Get the first element
Yes
Is next element available ?
YES
Did resolveException() return null ?

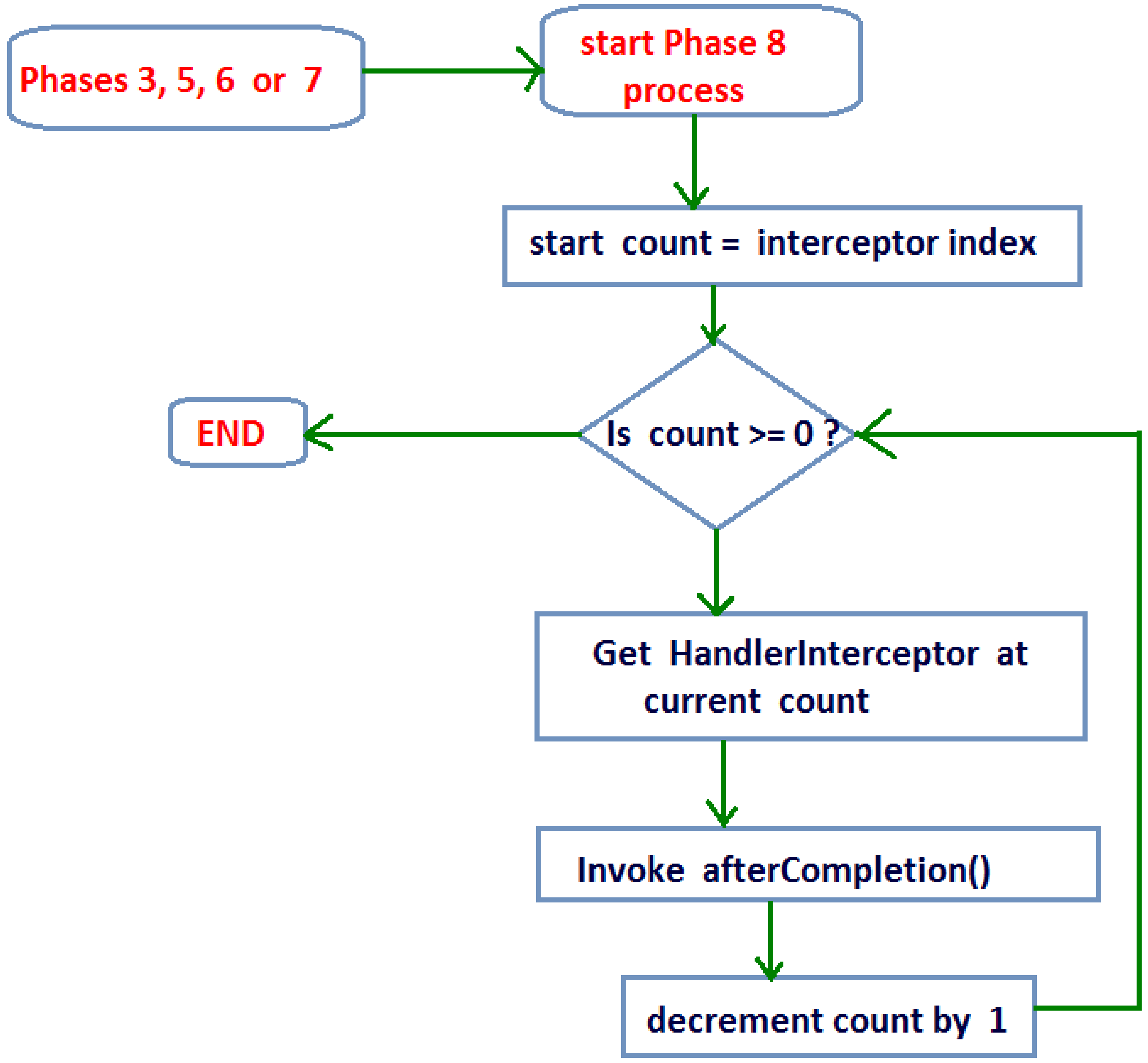


Phase 7 : Render the View


Phase 5 or 6
start Phase 7 process
Is ModelAndView has view ?
NO
set the default view name
YES
Is the View in ModelAndView a view name ?
YES
Resolve View Name
NO
Is the View object null ?
YES
Throw servletexception
NO
Invoke render() of view
Phase 8


start
Prepare an Iterator of View Resolver's Collection
Obtain the first element (View Resolver)
Invoke resolve View Name() method of View Resolver
Get the next View Resolver
YES
Did resolve View Name() return null ?
NO
YES
Is next element available ?
NO
return View object
return null

Phase 8 : Execute Interceptors afterCompletion methods


Phases 3, 5, 6 or 7
start Phase 8 process
start count = interceptor index
Is count >= 0 ?
END
Get HandlerInterceptor at current count
Invoke afterCompletion()
decrement count by 1