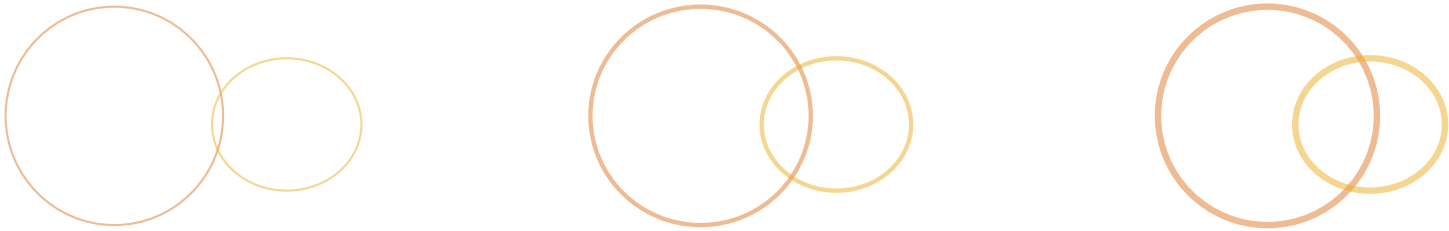
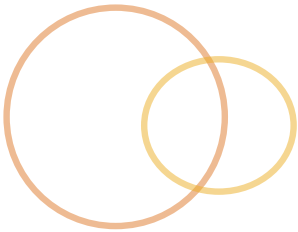


Lifecycle of Spring Beans

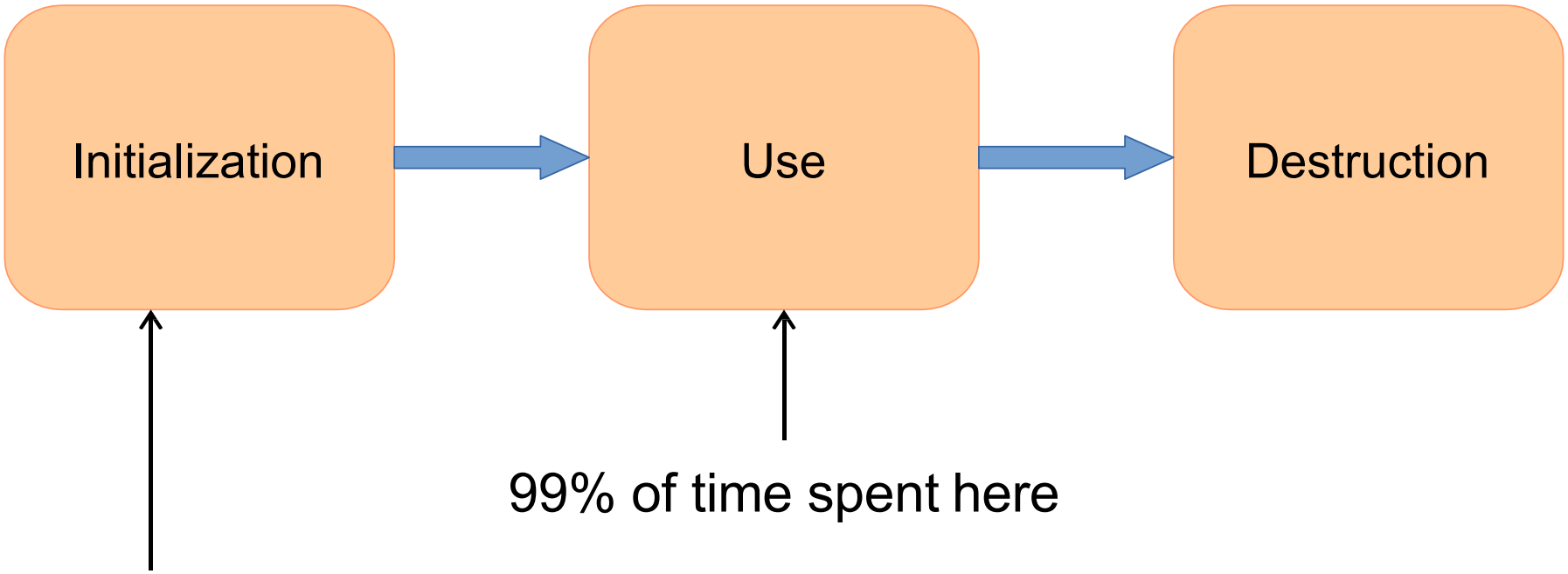
Objectives



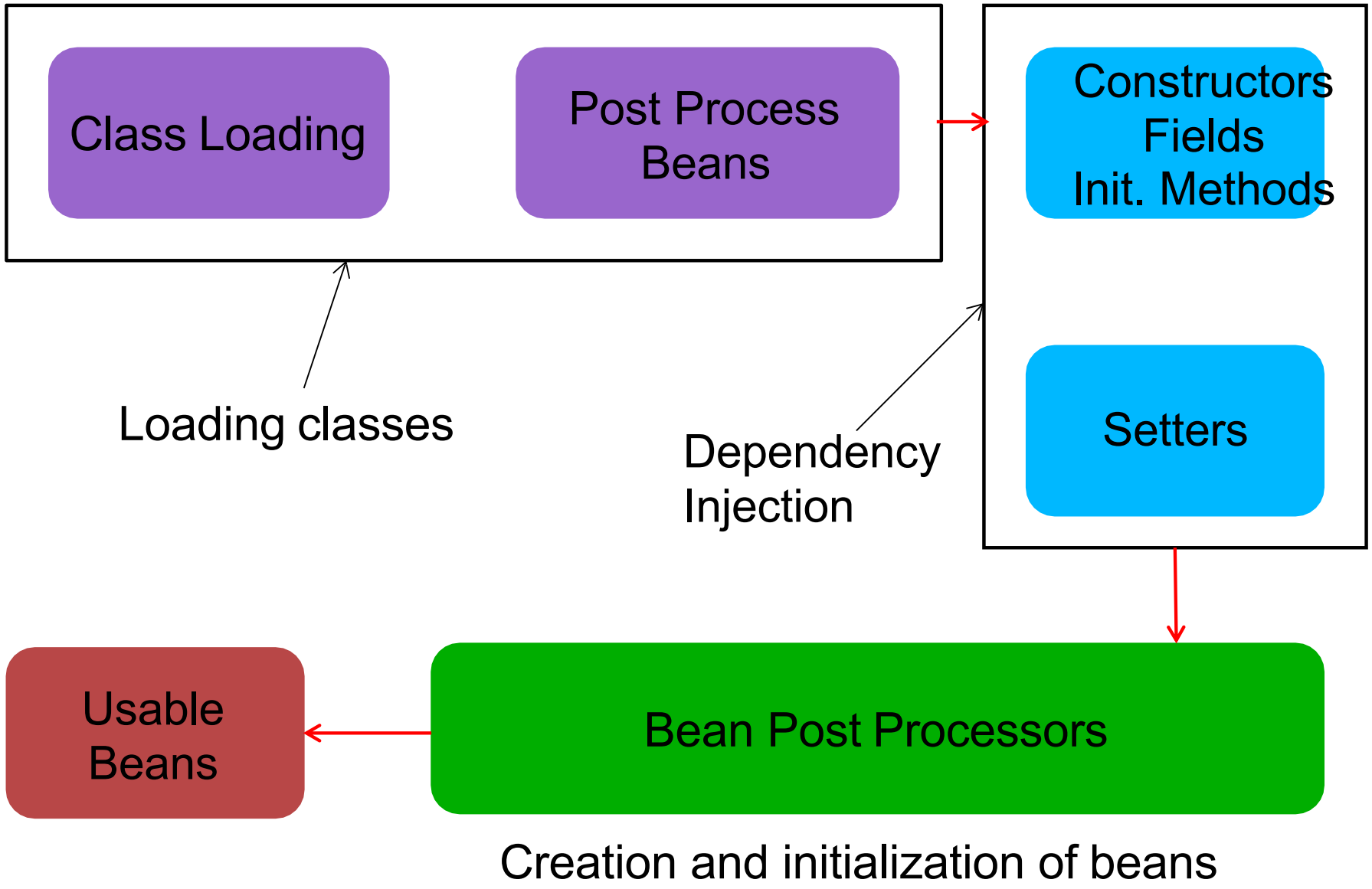
When we are done, you should be able to:

- Explain how the initialization phase works
- Understand the difference between the `BeanFactoryPostProcessor` and the `BeanPostProcessor`

Phases of Lifecycle



Initialization



Creation and initialization of beans

BeanFactoryPostProcessor

- These can modify the configuration of any bean
- We can write our own, but usually do not
- BFPPs allow us to:
  - Provide administrators a mechanism to monitor and modify beans
  - Handle tags from XML
  - `<context:component-scan/>`
  - `<context:annotation-config/>`
- In this phase, all beans are loaded and processed at once and all are completed before going on to the next phase

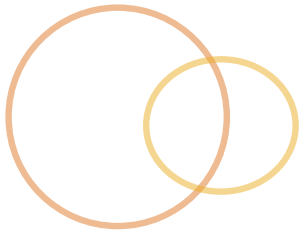
Dependency Injection of Beans

- Order of loading
  - Call constructors
  - Call field injections
  - Call set methods
- Each bean goes through entire phase one at a time

Creation and Initialization of Bean

- After all beans get through dependency injection
- Actually a `BeanPostProcessor` before and after initialization
  - Spring uses this, we rarely do
  - `@PostConstruct` called after second set of BPPs
  - Actually modify the bean instance
- Loaded Eagerly
  - Can be loaded lazily, but usually not appropriate

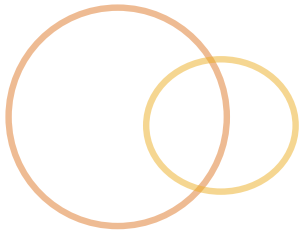
Destruction



- Context is responsible for handling
- Order of tear down:
  - Invokes cleanup methods, both those annotated with `@PreDestroy` and those defined in `@Bean`
  - Destroys any unreferenced singletons
  - Destroys itself (this context)
- Destruction of a context does NOT destroy
  - Referenced singletons
  - Non-singletons
  - Proxies bound to beans in use

Best Practices

Objectives



When we are done, you should be able to:

- Understand when to use what type of configuration
- Explain why to use setter injection over constructor injection
- Split configuration files into smaller files
- Utilize bean inheritance

Prefer Configuration Over Autowired

- Easier to organize
- Larger the application, more difficult it is to manage with autowiring
- Spring is a configuring framework
  - Best practice is to separate configuration from business logic
- If this is best practice, then why have annotations for non-configuration files?
  - XML has fallen out of favor
  - Annotations with XML was a step between XML and Java configuration
  - Sometimes configuration is easier at point of need

Prefer Setter Injection Over Constructor Injection

- Constructor Injection
  - Makes the bean thread safe
  - Ensures that everything that needs a bean gets its own instance
- Setter Injection
  - Better if there are multiple attributes to be set
  - Allows for attributes to be optional
    - `@Autowired` makes them required
    - `@Required` makes them required but you still have to call the method
  - Usually easier for testing

Use Multiple Configuration Files

- Easier to maintain, more flexible
- Too many types of beans
  - Business logic
  - Database connectivity
  - Different environments
- One file is poor separation of responsibility
- One file increases probability of error
- Usually easier to test

Combine Configuration Files In One

- In code (preferred)

```
public static void main(String[] args) {

  ApplicationContext appContext = new
   ClassPathXmlApplicationContext ("data-config.xml, business-
        config.xml");

  LibraryService service = appContext.getBean("library",
    LibraryService.class);

  Book book = service.getBook("An Artificial Night");
}
```

- In `application-config.xml`

```
<import resource "business-config.xml"/>
<import resource "data-config.xml"/>
```

Importing Java Configurations

- Two different ways
  - `@Import({OtherConfig1.class, OtherConfig2.class})`
  - `@ImportResource({"some-config.xml", "some-other-config.xml"})`

Miscellaneous Best Practices

- Externalize properties
  - Some change often, so putting them in a property file can make it easier to manage
- Use consistent naming conventions
- Use consistent version of tags
  - Use `@PostConstruct` or `@Bean(initMethod="...")`
- Use interfaces for all Spring beans
- Use short form of tags

Miscellaneous Best Practices [cont.]

- Do not overuse dependency injection
  - Is this a bean to be used by
    - Other beans?
    - Other users?
  - Is this something unique to this one user?
  - Does it have values that are not known until after a user actively interacts with it?