1.

1.1.1

* Spring employs four key strategies:
  1. Lightweight development with POJOs
  2. Loose coupling through DI and interface orientation
  3. Declarative programming through aspects and common conventions
  4. Eliminating boilerplate code with aspects and templates

1.1.2 Injecting dependencies

* In a spring application, an application context loads bean definitions and wires them together. The spring application context is fully responsible for the creation and wiring of the objects that make up the application. Spring comes with several implementations of its application context, each differing in how it loads its configuration.
* ClassPathXmlApplicationContext loads the Spring context from one or more XML files located in the application’s classpath

1.2.1 Working with an application context

* AnnotationConfigApplicationContext: Loads a Spring application context from one or more Java-based config classes
* AnnotationConfigWebApplicationContext: Loads a Spring web application context from one or more Java-based config
* The difference between using FileSystemXmlApplicationContext and ClassPathXmlApplicationContext is that FileSystemXmlAppliationContext looks for a config file in a specific location within the filesystem, whereas ClassPathXmlApplicationContext looks for its config file anywhere in the classpath

2.2 Automatically wiring beans

* Component scanning – Spring automatically discovers beans to be created in the application context
* Autowiring – spring automatically satisfies bean dependencies

2.2.1 Creating discoverable beans

* With no further configuration, @ComponentScan will default to scanning the same package as the configuration class
* Use <context:component-scan base-package = “whatever”> for xml

2.4.4 Setting Properties

* The <property> element does for property setter methods what the <constructor-arg> element does for constructors, e.g. <property name=”compactDisc” ref=”compactDisc” />
* <value> vs <value-ref> in lists