CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

Advanced Software Development

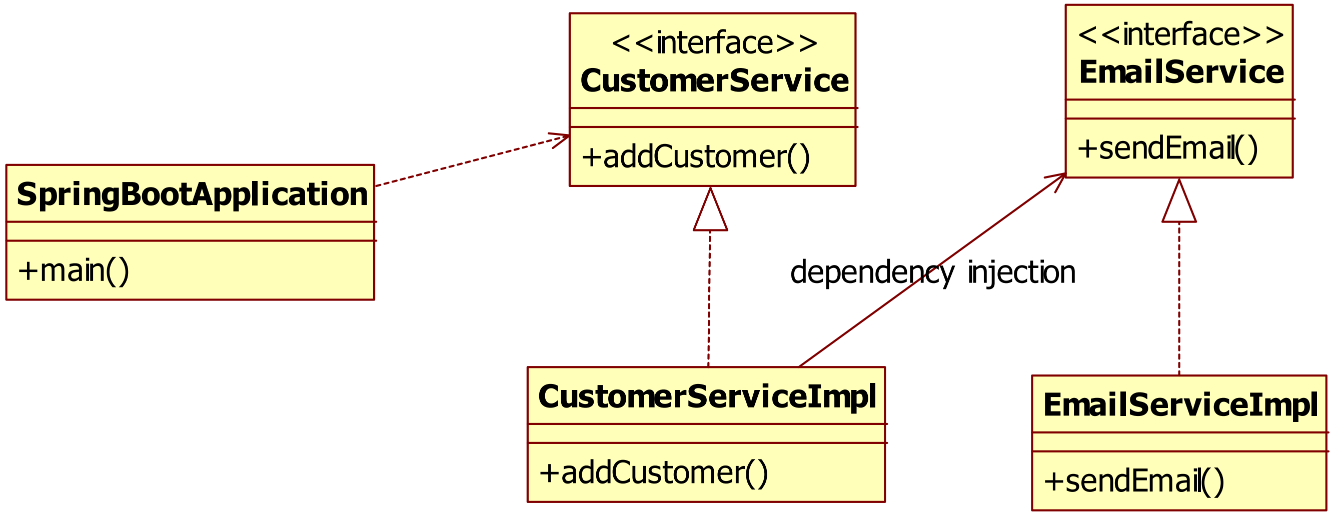
Maharishi University
OF MANAGEMENT

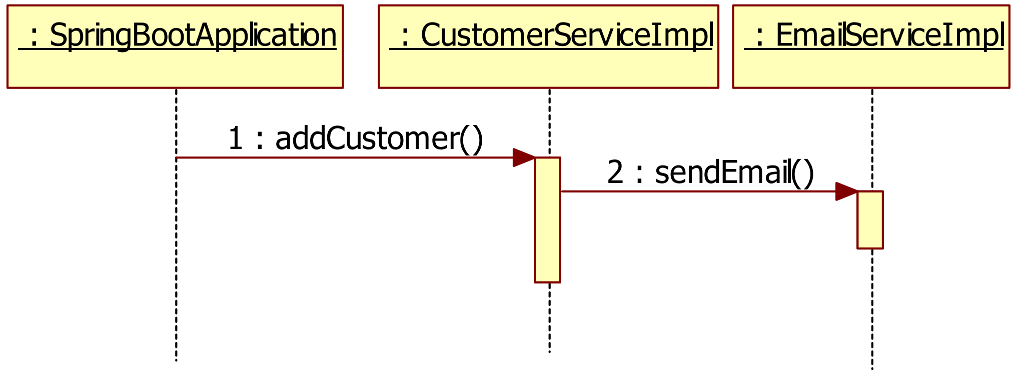
SPRING BOOT

Spring boot

- Framework that makes it easy to configure and run spring applications

Example application





Using annotations

```java
public interface EmailService {
  void sendEmail();
}
```

```java
@Service
public class EmailServiceImpl implements EmailService{
  public void sendEmail() {
    System.out.println("Sending email");
  }
}
```

```java
public interface CustomerService {
  void addCustomer();
}
```

```java
@Service
public class CustomerServiceImpl implements CustomerService{
  @Autowired
  private EmailService emailService;

  public void addCustomer() {
    emailService.sendEmail();
  }
}
```

Spring Boot option 1

Same as
@Configuration,
@ComponentScan
@EnableAutoConfiguration

```
@SpringBootApplication
public class SpringBootApplication {

  public static void main(String[] args) {
    ApplicationContext context = new
         AnnotationConfigApplicationContext(SpringBootApplication.class);
    CustomerService customerService =
    context.getBean("customerServiceImpl",CustomerService.class);
    customerService.addCustomer();
  }
}
```

Create an ApplicationContext based on the current configuration class

Spring Boot option 2

```
@SpringBootApplication
public class SpringBootApplication {

  public static void main(String[] args) {
    ApplicationContext context = new
         AnnotationConfigApplicationContext(AppConfig.class);
    CustomerService customerService =
    context.getBean("customerServiceImpl",CustomerService.class);
    customerService.addCustomer();
  }
}
```

Create an ApplicationContext based on an external configuration class

```
@Configuration
@ComponentScan("customers")
public class AppConfig {
}
```

Spring Boot option 3

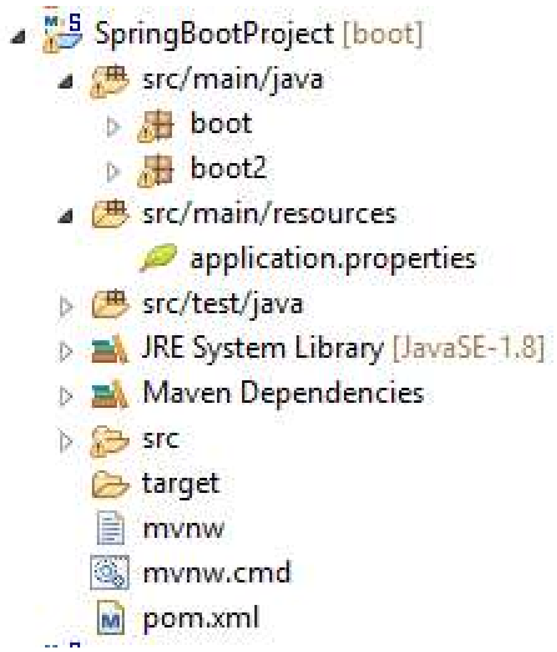
```
@SpringBootApplication
public class SpringBootProjectApplication implements CommandLineRunner {
  @Autowired
  private CustomerService customerService;

  public static void main(String[] args) {
    SpringApplication.run(SpringBootProjectApplication.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    customerService.addCustomer();
  }
}
```

Implement the CommandLineRunner

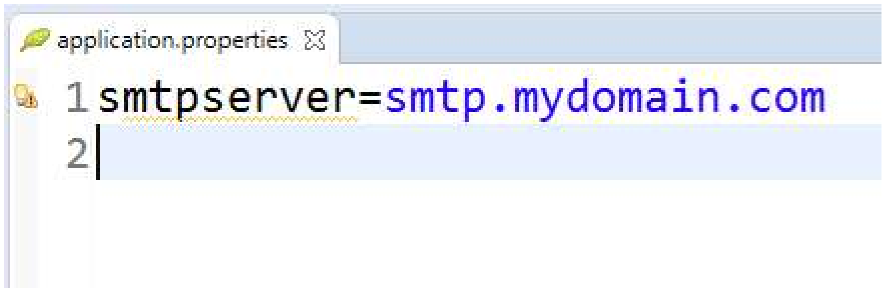
Implement the run() method

Spring Boot configuration

- Spring Boot uses **application.properties** as the default configuration file



```
SpringBootProject [boot]
  src/main/java
    boot
    boot2
  src/main/resources
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  mvnw
  mvnw.cmd
  pom.xml
```



application.properties

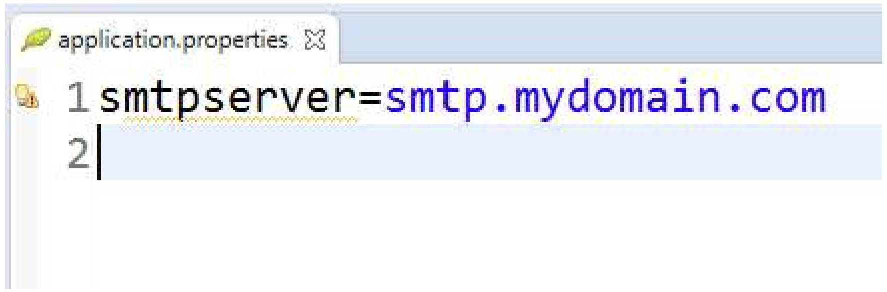
```
smtpserver=smtp.mydomain.com
```

application.properties

```java
public interface EmailService {
  void sendEmail();
}
```

```java
@Service
public class EmailServiceImpl implements EmailService{
  @Value(" ${smtpserver}")
  String smtpServer;

  public void sendEmail() {
    System.out.println("Sending email using smtp server "+smtpServer);
  }
}
```

Inject the value from the properties file



application.properties

```
smtpserver=smtp.mydomain.com
```

STRATEGY WITH SPRING

DI example

```
@Service
public class GreetingService {
  @Autowired
  private Greeting greeting;

  public String getTheGreeting() {
    return greeting.getGreeting();
  }
}
```

Spring does not know which class to inject

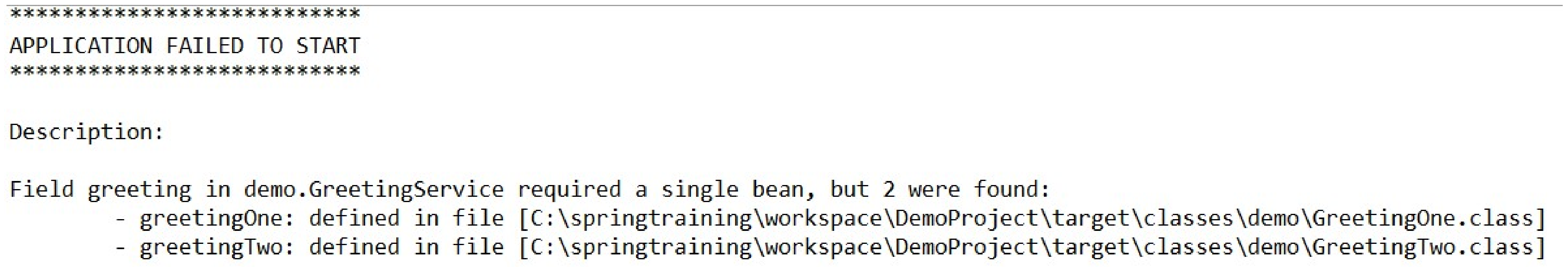
```
public interface Greeting {
  String getGreeting();
}
```

```
@Component
public class GreetingOne implements Greeting{

  public String getGreeting() {
    return "Hello World";
  }
}
```

```
@Component
public class GreetingTwo implements Greeting{

  public String getGreeting() {
    return "Hi World";
  }
}
```

DI example

```
@SpringBootApplication
public class DemoProjectApplication implements CommandLineRunner {

  @Autowired
  private GreetingService greetingService;

  public static void main(String[] args) {
    SpringApplication.run(DemoProjectApplication.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    System.out.println(greetingService.getTheGreeting());
  }
}
```



```
***************************
APPLICATION FAILED TO START
***************************

Description:

Field greeting in demo.GreetingService required a single bean, but 2 were found:
	- greetingOne: defined in file [C:\springtraining\workspace\DemoProject\target\classes\demo\GreetingOne.class]
	- greetingTwo: defined in file [C:\springtraining\workspace\DemoProject\target\classes\demo\GreetingTwo.class]
```

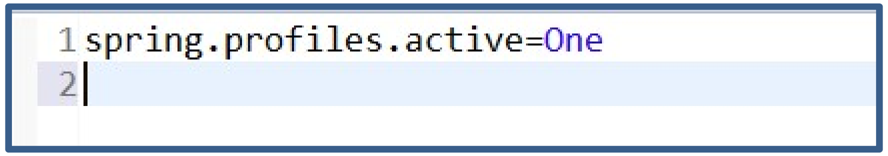
Solution 1: use qualifier

```
@Service
public class GreetingService {
  @Autowired
  @Qualifier(value="greetingOne")
  private Greeting greeting;

  public String getTheGreeting() {
    return greeting.getGreeting();
  }
}
```

Solution 2: use profiles

```java
@Service
public class GreetingService {
  @Autowired
  private Greeting greeting;

  public String getTheGreeting() {
    return greeting.getGreeting();
  }
}
```

Set the active profile in application.properties



```
spring.profiles.active=One
```

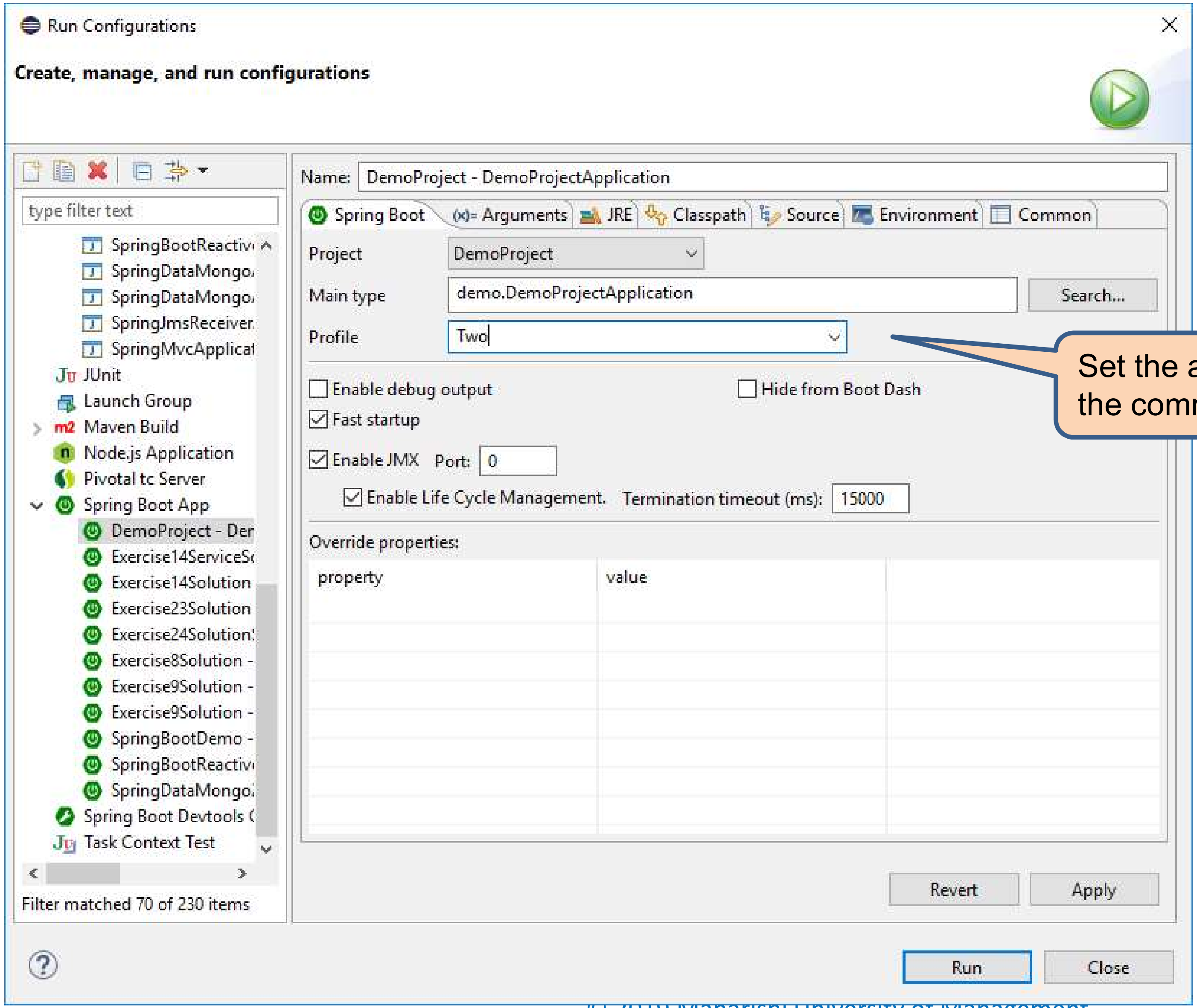
```java
@Component
@Profile("One")
public class GreetingOne implements Greeting{
  public String getGreeting() {
    return "Hello World";
  }
}
```

Define a profile

```java
@Component
@Profile("Two")
public class GreetingTwo implements Greeting{
  public String getGreeting() {
    return "Hi World";
  }
}
```

Define a profile

Activate a profile



Run Configurations

Create, manage, and run configurations

type filter text

- SpringBootReactiv
- SpringDataMongo
- SpringDataMongo
- SpringJmsReceiver
- SpringMvcApplicat
- JUnit
- Launch Group
- Maven Build
- Node.js Application
- Pivotal tc Server
- Spring Boot App
  - DemoProject - Der
  - Exercise14ServiceS
  - Exercise14Solution
  - Exercise23Solution
  - Exercise24Solution
  - Exercise8Solution -
  - Exercise9Solution -
  - Exercise9Solution -
  - SpringBootDemo -
  - SpringBootReactiv
  - SpringDataMongo
- Spring Boot Devtools (
- Task Context Test

Filter matched 70 of 230 items

Name: DemoProject - DemoProjectApplication

Tabs: Spring Boot | Arguments | JRE | Classpath | Source | Environment | Common

Project: DemoProject

Main type: demo.DemoProjectApplication

Profile: Two

- [ ] Enable debug output
- [ ] Hide from Boot Dash
- [x] Fast startup
- [x] Enable JMX Port: 0
- [x] Enable Life Cycle Management. Termination timeout (ms): 15000

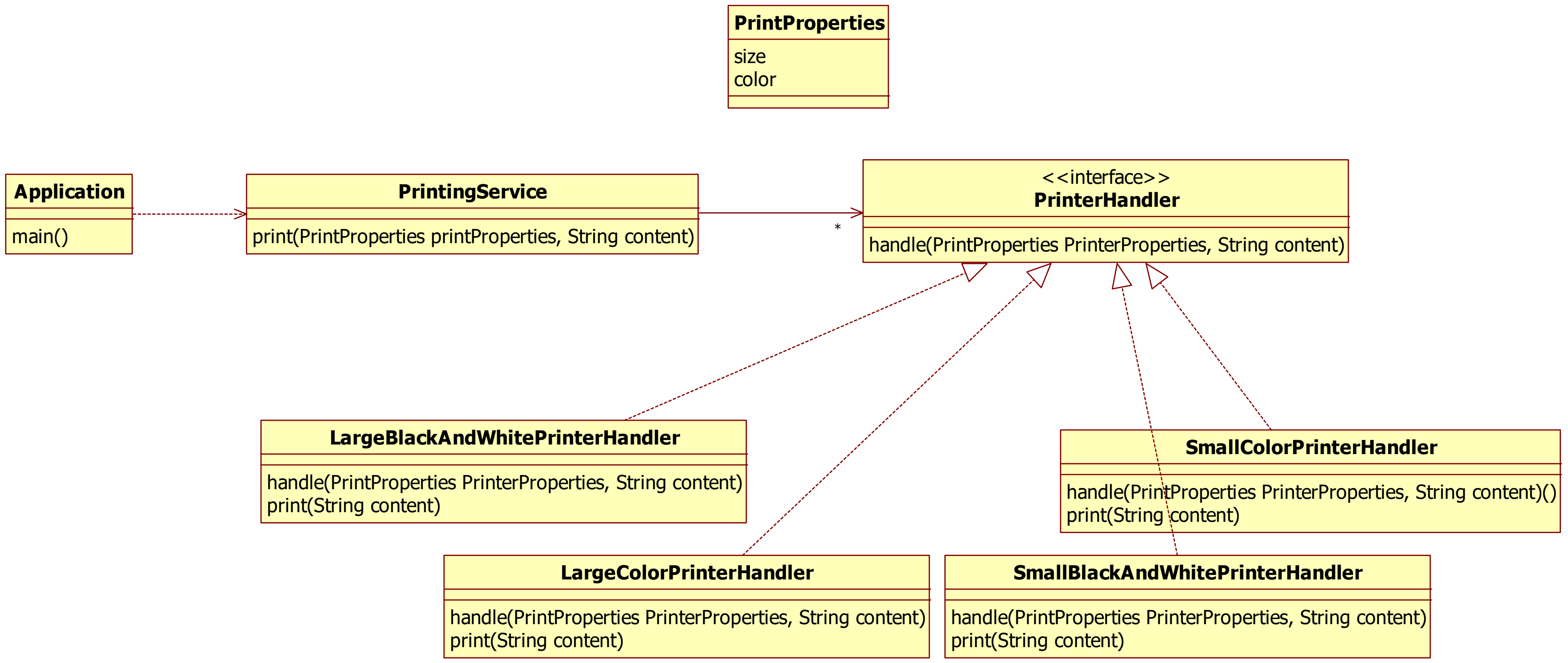
Override properties:

property	value

Set the active profile on the command line

CHAIN OF RESPONSIBILITY WITH SPRING

Chain of Responsibility with Spring


PrintProperties
size
color
Application
main()
PrintingService
print(PrintProperties printProperties, String content)
*
<<interface>>
PrinterHandler
handle(PrintProperties PrinterProperties, String content)
LargeBlackAndWhitePrinterHandler
handle(PrintProperties PrinterProperties, String content)
print(String content)
SmallColorPrinterHandler
handle(PrintProperties PrinterProperties, String content)()
print(String content)
LargeColorPrinterHandler
handle(PrintProperties PrinterProperties, String content)
print(String content)
SmallBlackAndWhitePrinterHandler
handle(PrintProperties PrinterProperties, String content)
print(String content)

LargeBlackAndWhitePrinterHandler

```
@Component
public class LargeBlackAndWhitePrinterHandler implements PrinterHandler{

  public void print(String content){
    System.out.println("Large black and white printer prints: "+content);
  }

  public boolean handle(PrintProperties printerProperties, String content) {
    if (isBlackAndWhitePrinter(printerProperties ) &&
        isLargePrinter(printerProperties)) {
      print(content);
      return true;
    }
    return false;
  }

  private boolean isLargePrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getSize()==Size.LARGE;
  }

  private boolean isBlackAndWhitePrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getColor()==Color.BLACKANDWHITE;
  }
}
```

LargeColorPrinterHandler

```
@Component
public class LargeColorPrinterHandler implements PrinterHandler{

  public void print(String content){
    System.out.println("Large color printer prints: "+content);
  }

  public boolean handle(PrintProperties printerProperties, String content) {
   if (isColorPrinter(printerProperties ) &&
        isLargePrinter(printerProperties)) {
      print(content);
      return true;
    }
    return false;
  }

  private boolean isLargePrinter(PrintProperties PrinterProperties) {
      return PrinterProperties.getSize()==Size.LARGE;
  }

  private boolean isColorPrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getColor()==Color.COLOR;
  }
}
```

SmallBlackAndWhitePrinterHandler

```java
@Component
public class SmallBlackAndWhitePrinterHandler implements PrinterHandler{

  public void print(String content){
    System.out.println("Large black and white printer prints: "+content);
  }

  public boolean handle(PrintProperties printerProperties, String content) {
   if (isBlackAndWhitePrinter(printerProperties ) &&
        isSmallPrinter(printerProperties)) {
      print(content);
      return true;
    }
    return false;
  }

  private boolean isSmallPrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getSize()==Size.SMALL;
  }

  private boolean isBlackAndWhitePrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getColor()==Color.BLACKANDWHITE;
  }
}
```

SmallColorPrinterHandler

```java
@Component
public class SmallColorPrinterHandler implements PrinterHandler{

  public void print(String content){
    System.out.println("Small color printer prints: "+content);
  }

  public boolean handle(PrintProperties printerProperties, String content) {
    if (isColorPrinter(printerProperties ) && isSmallPrinter(printerProperties)) {
      print(content);
      return true;
    }
    return false;
  }

  private boolean isSmallPrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getSize()==Size.SMALL;
  }

  private boolean isColorPrinter(PrintProperties PrinterProperties) {
    return PrinterProperties.getColor()==Color.COLOR;
  }
}
```

PrinterHandler and PrinterProperties

```
public interface PrinterHandler {
  public void handle(PrintProperties PrinterProperties, String content);
}
```

```
public class PrintProperties {
  enum Size {
    SMALL, LARGE
  }

  enum Color {
    BLACKANDWHITE, COLOR
  }

  private Size size;
  private Color color;

  public PrintProperties(Size size, Color color) {
    this.size = size;
    this.color = color;
  }

  ...

}
```

PrintingService

```java
@Service
public class PrintingService {
  @Autowired
  List<PrinterHandler> printerHandlers;

  public void print(PrintProperties printProperties, String content) {
    for (PrinterHandler printerHandler:  printerHandlers) {
      if (printerHandler.handle(printProperties, content))
        break;
    }
  }
}
```

Application

```java
@SpringBootApplication
public class Application implements CommandLineRunner {

  @Autowired
  private PrintingService printingService;

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    printingService.print(new PrintProperties(Size.SMALL, Color.COLOR), "Hello world");
    printingService.print(new PrintProperties(Size.LARGE, Color.COLOR), "Hello world");
    printingService.print(new PrintProperties(Size.SMALL, Color.BLACKANDWHITE), "Hello world");
    printingService.print(new PrintProperties(Size.LARGE, Color.BLACKANDWHITE), "Hello world");
  }
}
```

```
Small color printer prints: Hello world
Large color printer prints: Hello world
Small black and white printer prints: Hello world
Large black and white printer prints: Hello world
```

Main point

- The Chain of responsibility can be implemented in Spring with list injection

- We enliven those aspects in creation where we put our attention on.

Connecting the parts of knowledge with the wholeness of knowledge

1. The Spring framework instantiates your objects and wires them together with dependency injection.
2. Profiles allow you to change the wiring of beans using an external properties file

---

3. **Transcendental consciousness** is the field of pure consciousness which is home to all the laws of nature.
4. **Wholeness moving within itself:** In unity consciousness one enjoys a permanent state of fulfilment while spontaneously benefiting oneself and society