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

HIDE THE CONTEXT

Make it simpler

```
public class Application {

  public static void main(String[] args) {
    FWContext fWContext = new FWContext();

    BankService bankService =  (BankService) fWContext.getBeanOftype(BankService.class);
    if (bankService != null)
       bankService.deposit();
  }
}
```

Let's hide the context from the application code

Make it simpler

```java
public class Application {

  public static void main(String[] args) {
    FWContext fWContext = new FWContext();

    BankService bankService =  (BankService) fWContext.getBeanOftype(BankService.class);
    if (bankService != null)
       bankService.deposit();
  }
}
```

```java
public class Application implements Runnable{
  @Inject
  BankService bankService;

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

  @Override
  public void run() {
    bankService.deposit();
  }
}
```

Inject the application class(es)

Create the context and perform DI

Start the application

FWApplication

```java
public class FWApplication {

  public static void run(Class applicationClass) {
    // create the context
    FWContext fWContext = new FWContext();
    try {
      // create instance of the application class
      Object applicationObject = (Object) applicationClass.newInstance();
      // find annotated fields
      for (Field field : applicationObject.getClass().getDeclaredFields()) {
        if (field.isAnnotationPresent(Inject.class)) {
          // get the type of the field
          Class<?> theFieldType = field.getType();
          // get the object instance of this type
          Object instance = fWContext.getBeanOftype(theFieldType);
          // do the injection
          field.setAccessible(true);
          field.set(applicationObject, instance);
        }
      }
      //call the run() method
      if (applicationObject instanceof Runnable)
        ((Runnable)applicationObject).run();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

INJECTION OF PRIMITIVE TYPES

EmailServiceImpl

```
@Retention(RUNTIME)
@Target(FIELD)
public @interface Inject {
  String value() default "";
}
```

Add an attribute with name ‘**value**’ to the annotation

```
@Service
public class EmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending email: "+content+" , message="+theMessage);
  }
}
```

Inject the message specified in the config.properties file

config.properties

```
message=Hello
bankname=First National Bank
```

BankServiceImpl

```java
@Service
public class BankServiceImpl implements BankService{
  @Inject
  private EmailService emailService;

  @Inject(value="bankname")
  String bankName;

  public void setEmailService(EmailService emailService) {
    this.emailService = emailService;
  }

  public void deposit() {
    emailService.send("deposit to "+bankName);
  }
}
```

Inject the bankName specified in the config.properties file

```java
@Retention(RUNTIME)
@Target(FIELD)
public @interface Inject {
  String value() default "";
}
```

config.properties

```
message=Hello
bankname=First National Bank
```

ConfigFileReader

```java
public class ConfigFileReader {

  static Properties getConfigProperties() {
    Properties prop = null;

    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      prop = new Properties();
      prop.load(new FileInputStream(rootPath + "/config.properties"));
    } catch (Exception e) {
      e.printStackTrace();
    }
    return prop;
  }
}
```

config.properties

```
message=Hello
bankname=First National Bank
```

FWContext

```
public class FWContext {

  private static List<Object> objectMap = new ArrayList<>();

  public FWContext() {
    try {
      // find and instantiate all classes annotated with the @Service annotation
      Reflections reflections = new Reflections("");
      Set<Class<?>> types = reflections.getTypesAnnotatedWith(Service.class);
      for (Class<?> implementationClass : types) {
        objectMap.add((Object) implementationClass.newInstance());
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    performReferenceDI();
    performStringDI();
  }
...
```

Inject all object references

Inject all String attributes

FWContext

```java
private void performStringDI() {
  Properties properties = ConfigFileReader.getConfigProperties();
  try {
    for (Object theTestClass : objectMap) {
      // find annotated fields
      for (Field field : theTestClass.getClass().getDeclaredFields()) {
        if (field.isAnnotationPresent(Inject.class)) {
          // get the type of the field
          Class<?> theFieldType = field.getType();
          if (field.getType().getName().contentEquals("java.lang.String")) {
            // get attribute value
            String attrValue = field.getAnnotation(Inject.class).value();
            // get the property value
            String toBeInjectedString = properties.getProperty(attrValue);
            // do the injection
            field.setAccessible(true);
            field.set(theTestClass, toBeInjectedString);
          }
        }
      }
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Convention over configuration

```java
public class ConfigFileReader {

  static Properties getConfigProperties() {
    Properties prop = null;

    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      prop = new Properties();
      prop.load(new FileInputStream(rootPath + "/config.properties"));
    } catch (Exception e) {
      e.printStackTrace();
    }
    return prop;
  }
}
```

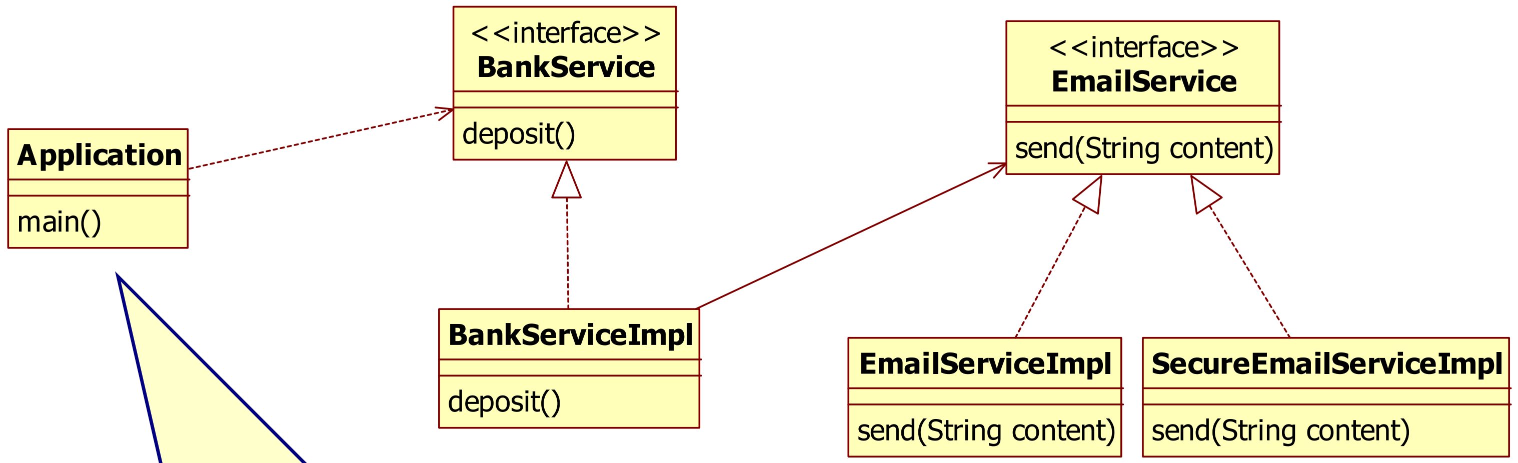
The framework will use this file by default

config.properties

```
message=Hello
bankname=First National Bank
```

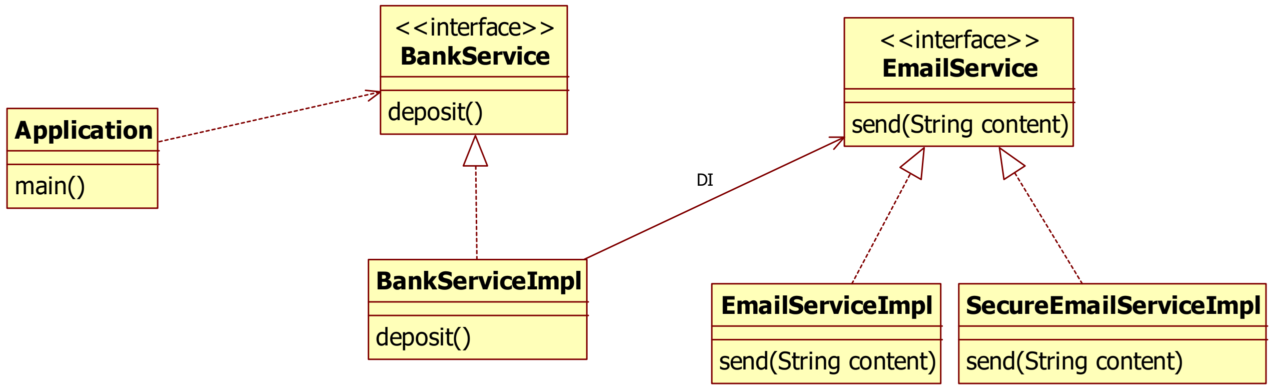
PROFILES

Strategy pattern



Someone needs to decide which concrete EmailService strategy to use

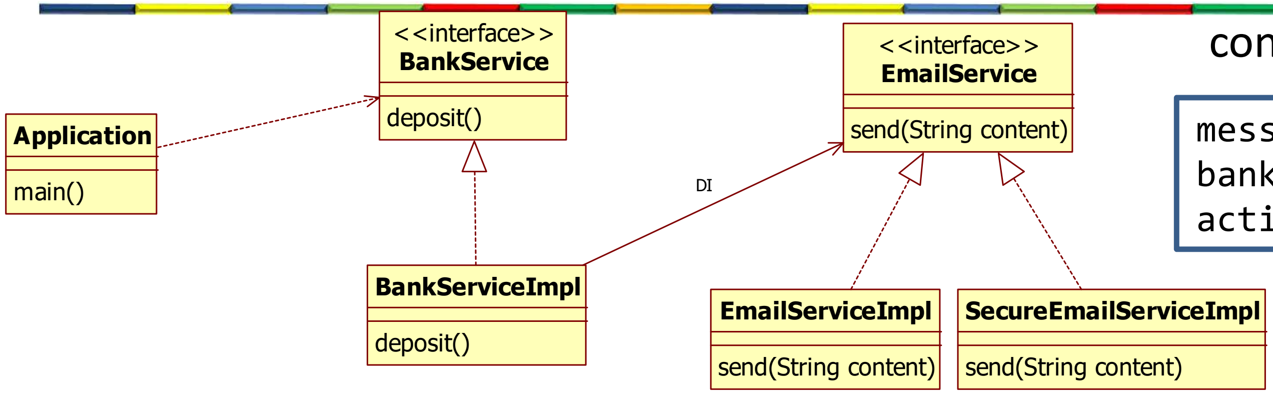
Two classes that implement the same interface



```
@Service
public class EmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending email: "+content+" , message="+theMessage);
  }
}
```

```
@Service
public class SecureEmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending secure email: "+content+" , message="+theMessage);
  }
}
```

Profiles



config.properties

```
message=Hello
bankname=First National Bank
activeprofile=Two
```

```java
@Service
@Profile(value="One")
public class EmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending email: "+content+" , message="+theMessage);
  }
}
```

```java
@Service
@Profile(value="Two")
public class SecureEmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending secure email: "+content+" , message="+theMessage);
  }
}
```

Working with profiles

```
public class FWContext {

  private static List<Object> objectMap = new ArrayList<>();
  Properties properties;
  String activeProfile;


  public FWContext() {
    try {
      properties = ConfigFileReader.getConfigProperties();
      activeProfile= properties.getProperty("activeprofile");
      ...
  }

  public Object getBeanOftype(Class interfaceClass) {
    ...
  }

}
```

Get the activeprofile from the configuration files

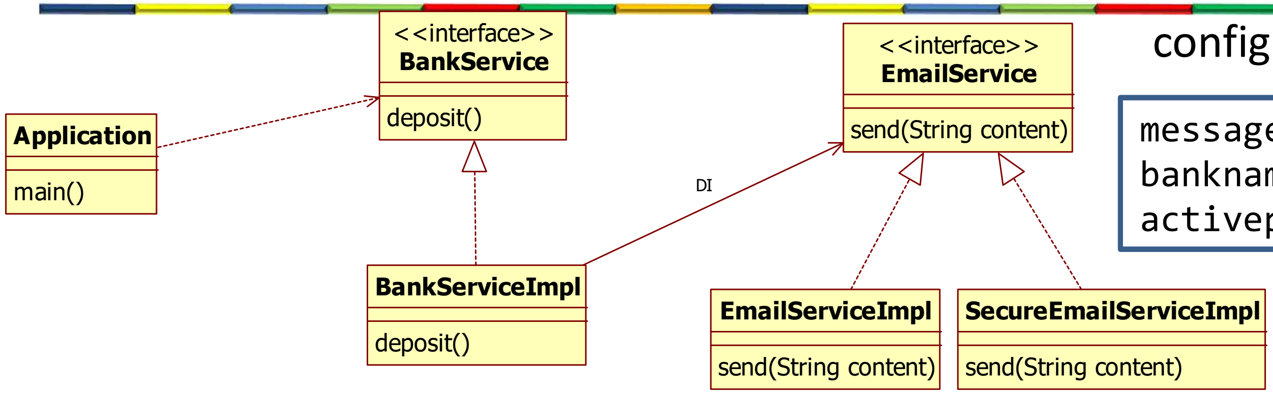
Working with profiles

```
public Object getBeanOftype(Class interfaceClass) {
  List<Object> objectList = new ArrayList<Object>();
  try {
    for (Object theTestClass : objectMap) {
      Class<?>[] interfaces = theTestClass.getClass().getInterfaces();

      for (Class<?> theInterface : interfaces) {
        if (theInterface.getName().contentEquals(interfaceClass.getName()))
          objectList.add(theTestClass);
        }
      }
  } catch (Exception e) {
    e.printStackTrace();
  }

  if (objectList.size() < 1) return null;
  if (objectList.size() == 1) return objectList.get(0);
  if (objectList.size() > 1) {
    for (Object theObject : objectList) {
      String profilevalue = theObject.getClass().getAnnotation(Profile.class).value();
      if (profilevalue.contentEquals(activeProfile)) {
        return theObject;
      }
    }
  }
  return null;
}
```

Get all classes that implement the provided interface

If multiple classes implement the provided interface, find the class with the activeprofile

Framework with profiles



config.properties

```
message=Hello
bankname=First National Bank
activeprofile=Two
```

We can configure which concrete EmailService strategy to use

```
@Service
@Profile(value="One")
public class EmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending email: "+content+" , message="+theMessage);
  }
}
```

```
@Service
@Profile(value="Two")
public class SecureEmailServiceImpl implements EmailService{
  @Inject(value="message")
  String theMessage;

  public void send(String content) {
    System.out.println("sending secure email: "+content+" , message="+theMessage);
  }
}
```

Main point

- Frameworks make heavily use of:
  - Inversion of Control
  - Classpath scanning
  - Dependency injection
  - Convention over configuration

- The Unified Field contains all the laws of nature.

Connecting the parts of knowledge with the wholeness of knowledge

1. Frameworks often use dependency injection to wire objects together
2. Dependency injection together with profiles gives us the open-closed principle

- **Transcendental consciousness** is the never changing field at the basis of all evolution.
- **Wholeness moving within itself:** In unity consciousness one realizes that the perfect underling structure of the entire creation is just the same structure of one's own pure consciousness.