Dependency Injection

Usa Sammapun

Dependency Injection (DI)

- one of very **<u>important</u>** software design principles and patterns
    - make code loosely coupled
    - can easily change implementation
- focus at composing objects (dependency)
    - client class does not instantiate dependent class within itself
    - but receive object instances via constructor or setter methods
        - —> inject dependency via constructor/setter
    - object instantiation now locates at only one place

Dependency Injection (DI)

- Dependency Injection (DI)
  - Specifically in the context of assembling dependencies between objects

inject

verb (SOMETHING NEW) UK US /ɪnˈdʒekt/ [T]

Definition

to introduce something new that is necessary or helpful to a situation or process

*A large amount of money will have to be injected **into** the company if it is to survive.*

*I tried to inject a little humour **into** the meeting.*

(Definition of inject verb (SOMETHING NEW) from the Cambridge Advanced Learner's Dictionary)

Source: http://martinfowler.com/articles/injection.html
http://dictionary.cambridge.org/

Dependency Injection (DI)

- Dependency Injection is also known as Inversion of Control (IoC).
  - Objects define their dependencies only through **constructor arguments**, arguments to a **factory method**, or **properties that are set** on the object instance after it is constructed
- (main method/container) then injects those when creating objects.
  - This process is fundamentally the inverse (hence the name, Inversion of Control) of the class itself controlling the instantiation or location of its dependencies by using direct construction of classes

Source: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

ATM with no Dependency Injection

(instantiate dependent object within itself)

ATM with no Dependency Injection

https://github.com/ladyusa/atm

```
public class Account {
   private double balance;

   public Account(double initialBalance) {
      balance = initialBalance;
   }
   // . . . code . . .
}

public class Customer {
   private int customerNumber;
   private int pin;
   private Account account;

   public Customer(int customerNumber, int pin, double initialBalance) {
      this.customerNumber = customerNumber;
      this.pin = pin;
      this.account = new Account(initialBalance);
   }
   // . . . code . . .
}
```

ATM with no Dependency Injection

https://github.com/ladyusa/atm

```
public class Bank {

   private Map<Integer,Customer> customers;
   private DataSource dataSource;

   public Bank() {
      dataSource = new DataSource("customers.txt");
      customers = new HashMap<Integer,Customer>();
   }
   public void initializeCustomers() throws IOException {
      customers = dataSource.readCustomers();
   }

   // . . . code . . .

}
```

ATM with no Dependency Injection

https://github.com/ladyusa/atm

```
public class DataSource {

    private String filename;

    public DataSource(String filename) {
        this.filename = filename;
    }

    public Map<Integer,Customer> readCustomers() throws IOException {

        // . . . code . . .

    }
}
```

ATM with no Dependency Injection

https://github.com/ladyusa/atm

```
public class ATM {
   public static final int START = 1;
   public static final int TRANSACT = 2;

   private int state;
   private int customerNumber;
   private Customer currentCustomer;
   private Account currentAccount;
   private Bank bank;

   public ATM() {
      this.bank = new Bank();
      this.customerNumber = -1;
      this.currentAccount = null;
      this.state = START;
   }

   public void validateCustomer(int customerNum, int pin) { . . . }

   // . . . code . . .
}
```

ATM with no Dependency Injection

https://github.com/ladyusa/atm

```
public class ATMSimulator {

   private ATM atm;

   public ATMSimulator() {
      atm = new ATM();
   }

   public void run() {
      atm.init();
      Scanner in = new Scanner(System.in);

      while (true) {
         int state = atm.getState();
         if (state == ATM.START) {
            System.out.print("Enter customer number: ");
            int number = in.nextInt();

      // . . . code . . .
}
```

ATM with no Dependency Injection

https://github.com/ladyusa/atm

```
public class Main {

    public static void main(String[] args) {

        ATMSimulator atmSimulator = new ATMSimulator();
        atmSimulator.run();

    }
}
```

ATM with Dependency Injection

(receive dependent object via constructor or setter)

ATM with Dependency Injection

https://github.com/ladyusa/atm-di

```
public class Bank {

   private Map<Integer,Customer> customers;
   private DataSource dataSource;

   public Bank(DataSource dataSource) {
      this.dataSource = dataSource;
      customers = new HashMap<Integer,Customer>();
   }
   public void initializeCustomers() throws IOException {
      customers = dataSource.readCustomers();
   }

   // . . . code . . .

}
```

ATM with Dependency Injection

https://github.com/ladyusa/atm-di

```
public class ATM {
   public static final int START = 1;
   public static final int TRANSACT = 2;

   private int state;
   private int customerNumber;
   private Customer currentCustomer;
   private Account currentAccount;
   private Bank bank;

   public ATM(Bank bank) {
      this.bank = bank;
      this.customerNumber = -1;
      this.currentAccount = null;
      this.state = START;
   }

   public void validateCustomer(int customerNum, int pin) { . . . }

   // . . . code . . .
}
```

ATM with Dependency Injection

https://github.com/ladyusa/atm-di

```
public class ATMSimulator {

   private ATM atm;

   public ATMSimulator(ATM atm) {
      this.atm = atm;
   }

   public void run() {
      atm.init();
      Scanner in = new Scanner(System.in);

      while (true) {
         int state = atm.getState();
         if (state == ATM.START) {
            System.out.print("Enter customer number: ");
            int number = in.nextInt();

      // . . . code . . .
}
```

ATM with Dependency Injection

https://github.com/ladyusa/atm-di

```
public class Main {

    public static void main(String[] args) {

        DataSource dataSource = new DataSource("customers.txt");
        Bank bank = new Bank(dataSource);
        ATM atm = new ATM(bank);
        ATMSimulator atmSimulator = new ATMSimulator(atm);
        atmSimulator.run();

    }
}
```

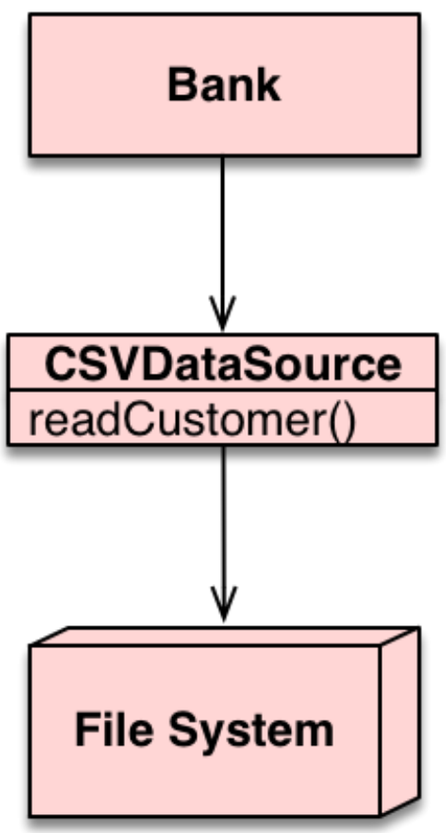
Dependency Injection with Interface

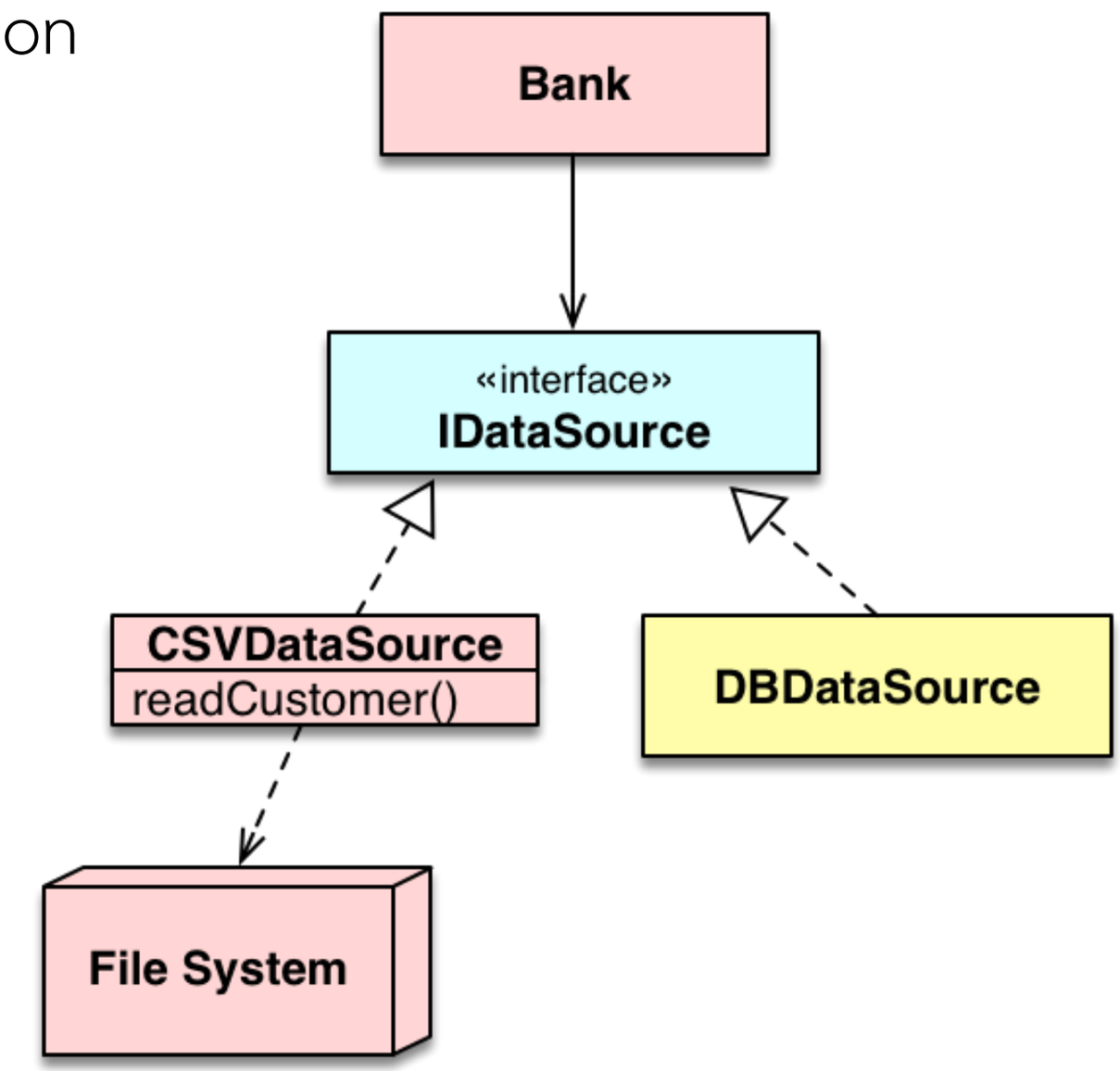
(allowing easier implementation change ---

Layer of Indirection)

Layer of Indirection

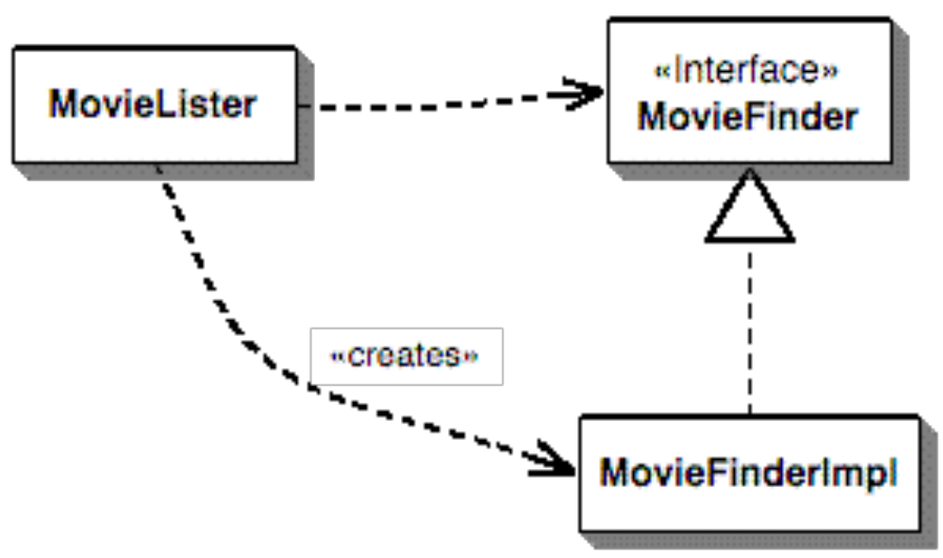
- insert interface between classes
  - can easily change implementation
  - testable design





MovieLister dependencies

- MovieLister has a following dependency



- **If used by others with different a MovieLister implementation**
  - This dependency should not be hardwired within the program
  - Depend only on the interface but still need to get to the object ?

Source: http://martinfowler.com/articles/injection.html

MovieLister example

```
public interface MovieFinder {
    List<Movie> findAll();
}
class MovieLister {
  private MovieFinder finder;
  public MovieLister() {
    finder = new CSVMovieFinder("movies1.txt");
  }
  public Movie[] moviesDirectedBy(String arg) {
      List<Movie> all = finder.findAll();

      Iterator<Movie> it = allMovies.iterator();
      while (it.hasNext()) {
         Movie movie = (Movie) it.next();
         if (!movie.getDirector().equals(arg))
            it.remove();
      }
      return (Movie[]) all.toArray(new Movie[all.size()]);
  }
}
```


Source: http://martinfowler.com/articles/injection.html

MovieLister example

- Or even if we push the object instantiation up to the caller / factory
  - the caller / factory still depends on a MovieFinder implementation

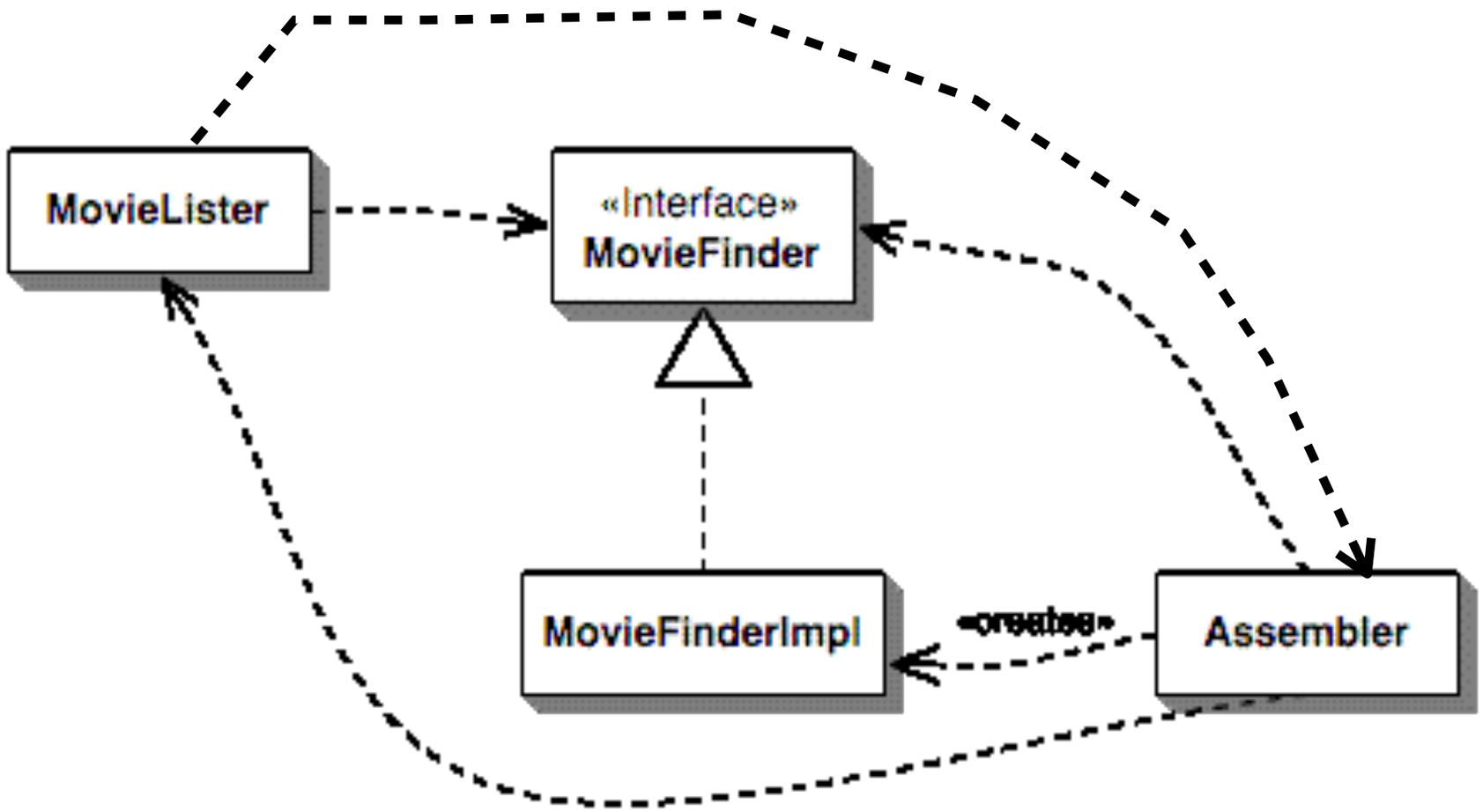
```
class MovieLister ...
  private MovieFinder finder;
  public MovieLister(MovieFinder finder) {
    this.finder = finder;
  }
  ...

class MovieEngine ...
  public MovieEngine() {
    MovieFinder finder = new CSVMovieFinder("movies1.txt");
    MovieLister lister = new MovieLister(finder)
```


Source: http://martinfowler.com/articles/injection.html

Dependency Injection

- Use a separate object, an assembler, to create desired objects



Source: http://martinfowler.com/articles/injection.html

Spring Framework

Spring Framework

- Framework : Libraries of classes
  - Facilitating database connectivity, transaction management, fault-tolerance, modular systems
  - Promote flexibility, extensibility, reusability
- Container : Object manager
  - Create objects (beans) according to your specification
  - Provide interface for accessing these objects
  - These objects are normally composed to build a business solution

Spring Container : Object manager

- Spring **instantiates** the objects (beans)
  - And injects the dependencies of your objects into the application
- Spring serves as a **life cycle manager of the objects** (beans).
  - Your objects do not have to worry about finding and establishing connections (association/dependency) with each other.

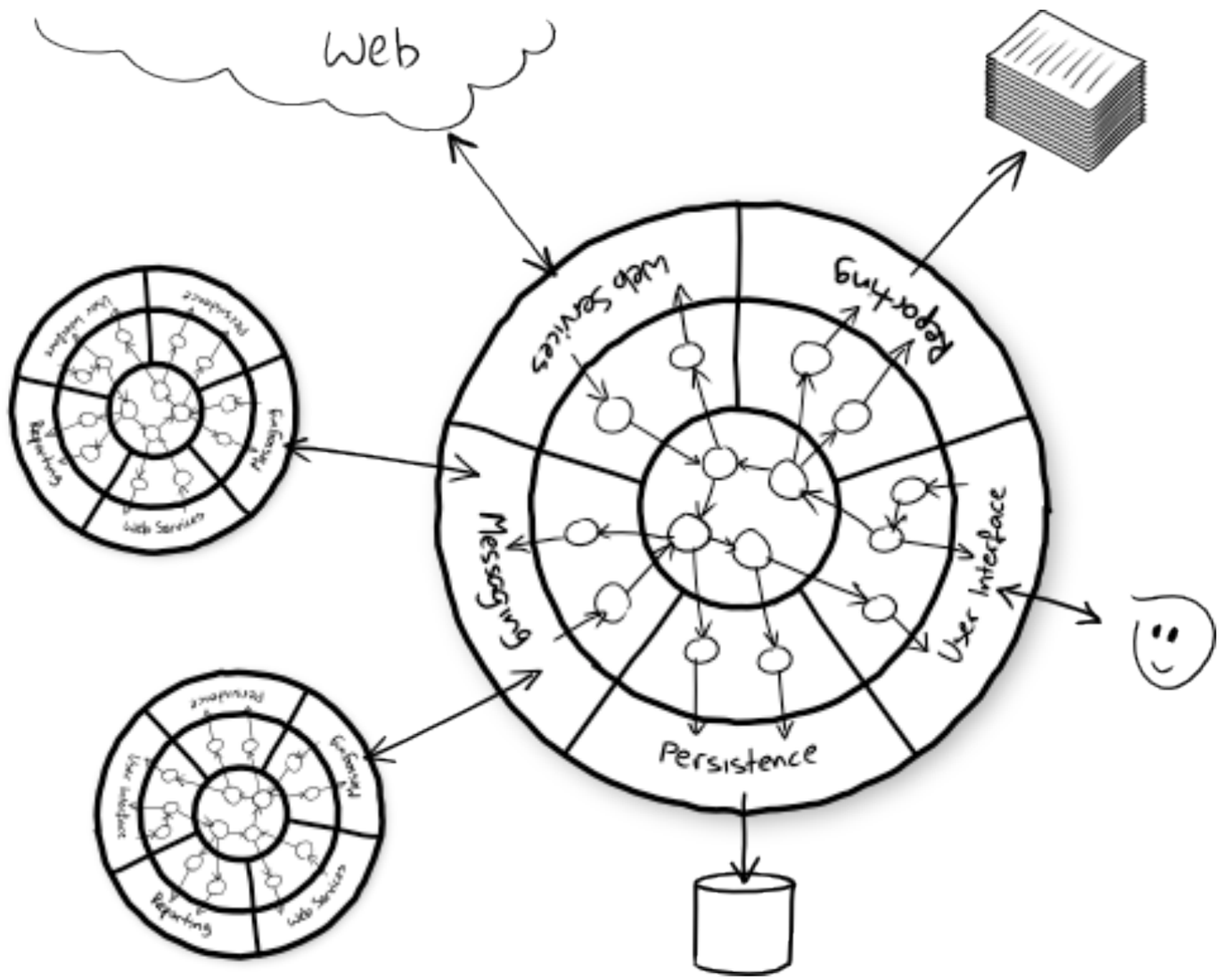
Source: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Spring Framework

- Reduce dependencies among objects
  - Promote using interface rather than implementation
  - Inversion of Control (IoC) / dependency injection
- Lightweight
  - Most classes can be independent of Spring

Spring

- Spring helps with “outer” classes



Source : http://www.natpryce.com/articles/000772.html

Spring

Spring helps with "outer" classes

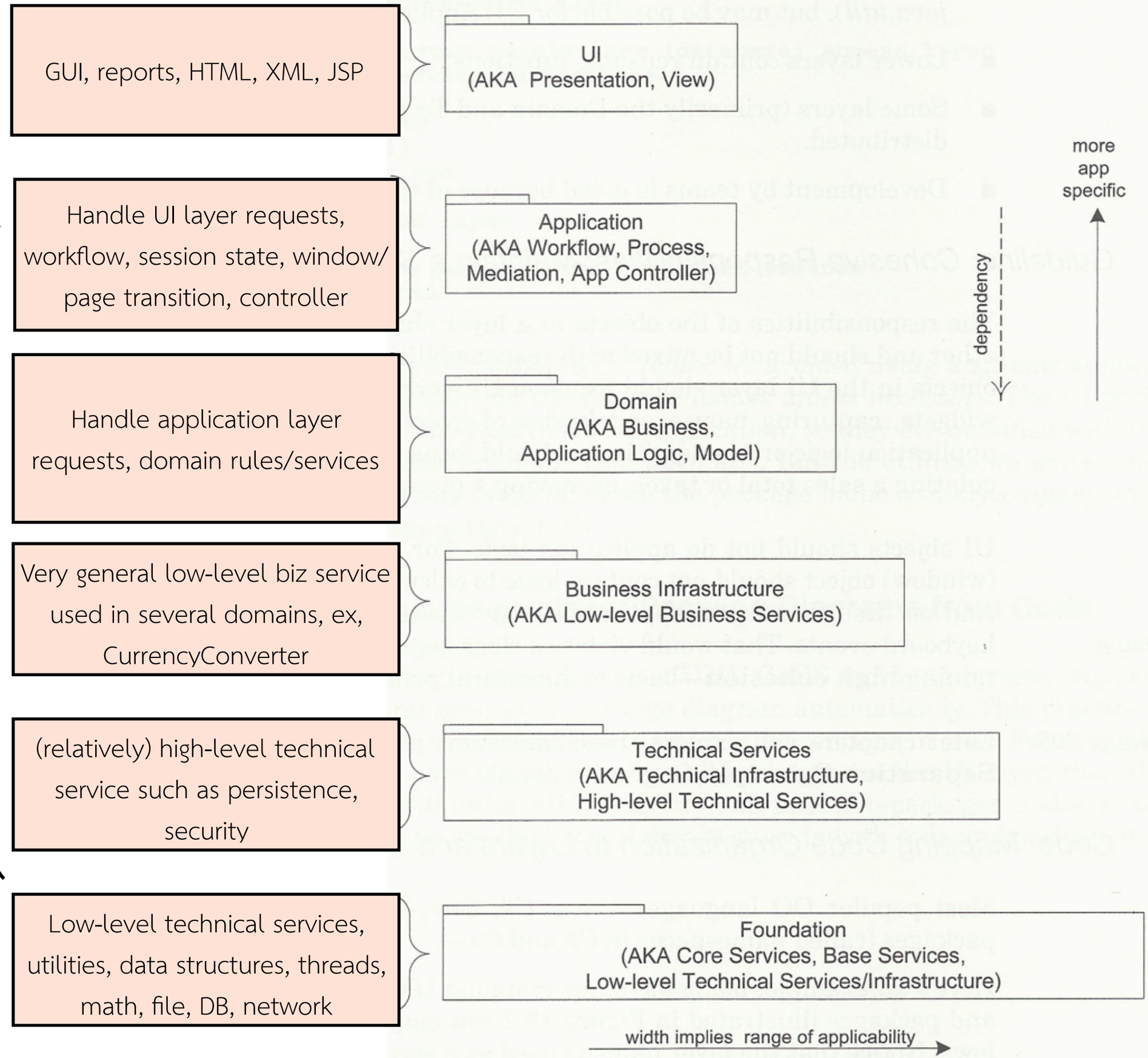


Figure 13.4 Common layers in an information system logical architecture.[1]

Source : Craig Larman. Applying UML and Patterns. 3rd ed. Prentice Hall, 2005.

Spring Framework

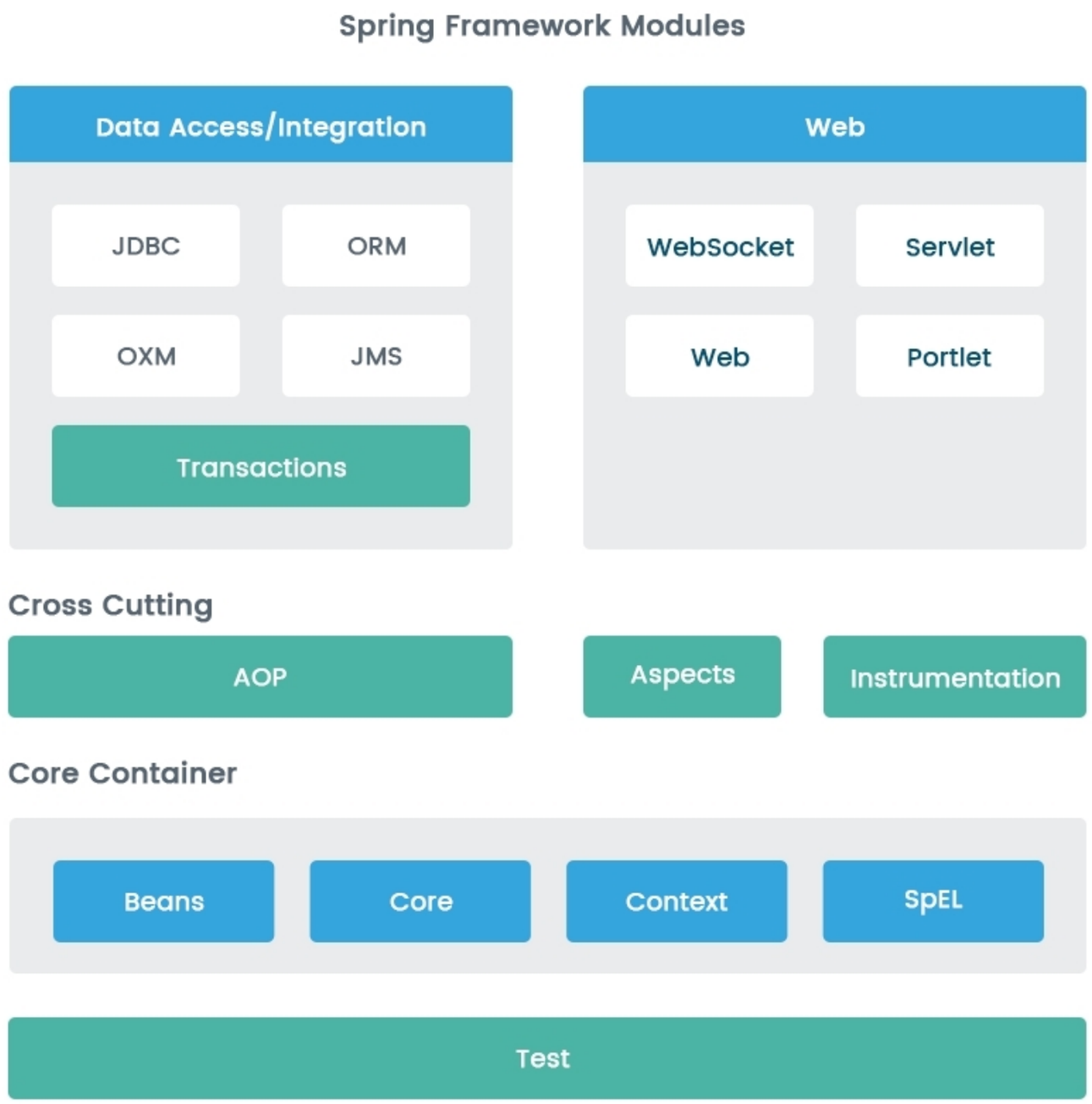


Image from Hands-On High Performance with Spring 5
by Dinesh Radadiya; Prashant Goswami; Subhash Shah; Chintan Mehta; Pritesh Shah, Packt Publishing, 2018

Spring Framework

- IoC Container
  - Very **generic** implementation of the **Factory** design pattern
  - Used to **create** any beans by giving its name
  - Core of IoC and dependency injection (DI)
  - Bean Factory / Application Context

Spring core container packages/modules

- **spring-core** : contains utilities used by other modules and managing bean life cycle operations.
- **spring-beans** : used to decouple code dependencies from actual business logic and eliminates the use of singleton classes using DI and IoC
- **spring-context** : provides features like resource loading and internationalization
- **spring-expression** : provides support for accessing properties of beans at runtime and also allows us to manipulate them.

Source: Hands-On High Performance with Spring 5
by Dinesh Radadiya; Prashant Goswami; Subhash Shah; Chintan Mehta; Pritesh Shah, Packt Publishing, 2018

Spring Framework

- Data Access/Integration
  - interacts with database and/or external interfaces. It consists of JDBC, ORM, OXM, JMS, and Transaction modules. These modules are **spring-jdbc**, **spring-orm**, **spring-oxm**, **spring-jms**, and **spring-tx**.
- Web
  - contains the Web, Web-MVC, Web-Socket, and other Web-Portlet modules. The respective module names are **spring-web**, **spring-webmvc**, **spring-websocket**, **spring-webmvc-portlet**.

Source: Hands-On High Performance with Spring 5
by Dinesh Radadiya; Prashant Goswami; Subhash Shah; Chintan Mehta; Pritesh Shah, Packt Publishing, 2018

Spring Framework

- Data Access Object (DAO)
  - JDBC abstraction layer
  - Hide complexity of JDBC programming
- Object-Relational Mapping (ORM)
  - Integration with ORM APIs such as Hibernate
- Aspect-Oriented Programming (AOP)
  - Provides support for aspect-oriented programming
  - Separation of responsibility

Spring Projects

- Spring provides different kinds of projects for different infrastructure needs,
  - also provides solutions to other problems in enterprise application: deployment, cloud, big data, and security, among others.
- Sample projects:
  - Spring Framework : IoC, dependency injection, DAO, JDBC, ORM, etc...
  - Spring Boot : support to create standalone, production-grade, Spring-based applications that you can just run. Embeds Tomcat, simplifies Maven by providing starter POMs. Support for application monitoring

Spring Projects

- Spring Data : provide easy access and manipulation of SQL-and NoSQL-based data stores. Also provide map-reduce and cloud-based data services.
- Spring Session : provides an API and implementations for managing a user's web session information.
- Spring HATEOAS : provides some APIs to ease creating REST representations that follow the HATEOAS principle when working with Spring. (HATEOAS — Hypermedia as the Engine of Application State)
- **Spring Cloud, Spring Security, etc**
- https://spring.io/projects/

Spring CORE Container

การนำ Spring Framework มาใช้

- เพิ่ม dependency นี้ใน pom.xml ของ Maven

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.1.1.RELEASE</version>
</dependency>
```

IoC/DI in Spring

- Core : IoC Container
  - Manage objects within Spring
  - Separate object creation and object execution
  - Developers describe how to create objects (via config file)
    - Without creating it yourself (no need for “new” keywords)
  - To reduce dependency between components
    - Thus, decoupling components

Different Forms of Dependency Injection

- Type of dependency injection
  - Constructor-based Dependency Injection
  - Setter-based Dependency Injection
- Type of bean creation configuration
  - XML-based
  - Annotation-based
  - Java-based

Source: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Constructor-based Dependency Injection

- Constructor-based DI is accomplished by the container invoking a constructor with a number of arguments, each representing a dependency.

```
package x.y;

public class ThingOne {

    // ThingOne depends on ThingTwo and ThingThree
    private ThingTwo thingTwo;
    private ThingThree thingThree;

    // a constructor so that dependencies (thingTwo, thingThree)
    // can be injected
    public ThingOne(ThingTwo thingTwo, ThingThree thingThree) {
        // ...
    }
}
```

Source: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Constructor-based Dependency Injection

- Spring creates all beans (objects) specified in a configuration
- XML-based configuration example below

```
<beans>
    <bean id="thingOne" class="x.y.ThingOne">
        <constructor-arg ref="thingTwo"/>
        <constructor-arg ref="thingThree"/>
    </bean>

    <bean id="thingTwo" class="x.y.ThingTwo"/>

    <bean id="thingThree" class="x.y.ThingThree"/>
</beans>
```

Source: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

MovieLister example

https://github.com/ladyusa/movie-ioc

```
public interface MovieFinder {
    List<Movie> findAll();
}

class MovieLister {

  private MovieFinder finder;

  public MovieLister() {
    finder = new CSVMovieFinder("movies1.txt");
  }

  public Movie[] moviesDirectedBy(String arg) {
      List<Movie> allMovies = finder.findAll();
      ...
```


Source: http://martinfowler.com/articles/injection.html

Constructor Injection (1)

```
public class CSVMovieFinder implements MovieFinder {

   private String filename;

   public CSVMovieFinder(String filename) {
      this.filename = filename;
   }

   @Override
   public List<Movie> findAll() {
      List<Movie> movies = new ArrayList<Movie>();
      // code that read csv file and create Movie object from file's data
      return movies;
   }
}
```

Constructor Injection (2)

```
public class MovieLister {

   private MovieFinder finder;

   public MovieLister(MovieFinder finder) {
      this.finder = finder;
   }

   public Movie[] moviesDirectedBy(String arg) {
      // code that return all movies directed by the given parameter
      ...
   }
}
```

Constructor Injection (3)

```
public class MovieTester {
    public static void main(String[] args) throws Exception {
        MovieLister lister = new MovieLister(new CSVMovieFinder("movies1.txt"));
        Movie[] ronHoward = lister.moviesDirectedBy("Ron Howard");
        for (Movie movieRH : ronHoward) {
            System.out.println(movieRH.getName());
        }
    }
}
```

ปกติเราทำแบบนี้

Constructor Injection (4)

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MovieTester {

   public static void main(String[] args) throws Exception {
      ApplicationContext context =
            new ClassPathXmlApplicationContext("movie-con-inject.xml");

      MovieLister lister = context.getBean("lister", MovieLister.class);

      Movie[] ronHoward = lister.moviesDirectedBy("Ron Howard");
      for (Movie movieRH : ronHoward) {
         System.out.println(movieRH.getName());
      }
   }
}
```

ใช้ Spring Framework

Constructor Injection (5)

xml document standard

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd">
```

schema for Spring XML

```
  <bean id="finder" class="springlesson.movie.CSVMovieFinder">
    <constructor-arg value="movies1.txt"/>
  </bean>

  <bean id="lister" class="springlesson.movie.MovieLister">
    <constructor-arg ref="finder"/>
  </bean>

</beans>
```

bean creation specification

Constructor Injection (5)

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="not-in-used-finder" class="springlesson.movie.CSVMovieFinder">
     <constructor-arg value="movies1.txt"/>
   </bean>
   <bean id="finder" class="springlesson.movie.DatabaseMovieFinder">
     <constructor-arg value="prod-database"/>
   </bean>
   <bean id="lister" class="springlesson.movie.MovieLister">
     <constructor-arg ref="finder"/>
   </bean>

</beans>
```

easy to change implementation

Constructor Injection (6)

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="not-in-used-finder" class="springlesson.movie.CSVMovieFinder">
    <constructor-arg value="movies1.txt"/>
  </bean>
  <bean id="finder" class="springlesson.movie.DatabaseMovieFinder">
    <constructor-arg value="prod-database"/>
  </bean>
  <bean id="lister" class="springlesson.movie.MovieLister">
    <constructor-arg ref="finder"/>
  </bean>

</beans>
```

primitive type param

dependency (object) param

Setter Injection (1)

```
public class MovieLister {

   private MovieFinder finder;

   public MovieLister() { ..... }

   public void setFinder(MovieFinder finder) {
      this.finder = finder;
   }

   public Movie[] moviesDirectedBy(String arg) {
      // code that return all movies directed by the given parameter
      ...
   }
}
```

รับ dependency
ผ่าน set method

Setter Injection (3)

```
<bean id="csv-finder" class="springlesson.movie.CSVMovieFinder">
  <constructor-arg value="movies1.txt"/>
</bean>

<bean id="lister" class="springlesson.movie.MovieLister">
  <property name="finder" ref="csv-finder"/>
</bean>
```

Constructor-based or Setter-based DI ??

- Since you can mix constructor-based and setter-based DI
  - Use constructors for mandatory dependencies
  - Use setter methods for optional dependencies.

Source: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html

Annotation-based Configuration

- Write configuration within component class itself by using annotations (ex. @Component, @Autowired) on the relevant class, method, or field
- XML only needs to specify **<context:annotation-config/>** and <context:component-scan base-package="....."/>

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

        <context:annotation-config/>
        <context:component-scan base-package="movie"/>
</beans>
```


Source: https://docs.spring.io/

MovieLister example

```java
@Component
public class MovieLister {

   private MovieFinder finder;

   @Autowired
   public MovieLister(MovieFinder finder) {
      this.finder = finder;
   }

   public Movie[] moviesDirectedBy(String arg) {
      // . . .
   }
}
```


Source: http://martinfowler.com/articles/injection.html

MovieLister example

```java
@Component
public class CSVMovieFinder implements MovieFinder {

   private String filename;

    public CSVMovieFinder() {
        this.filename = "movies1.txt";
    }

   public CSVMovieFinder(String filename) {
      this.filename = filename;
   }

   // . . .

}
```


Source: http://martinfowler.com/articles/injection.html

MovieLister example

no need for bean id

```
public class Main {

   public static void main(String[] args) throws Exception {
      ApplicationContext context =
            new ClassPathXmlApplicationContext("config.xml");

      MovieLister lister = context.getBean(MovieLister.class);

      Movie[] ronHoward = lister.moviesDirectedBy("Ron Howard");
      for (Movie movieRH : ronHoward) {
         System.out.println(movieRH.getName());
      }
   }
}
```


Source: http://martinfowler.com/articles/injection.html

ATM with Spring

Event with Spring (XML-based)

```java
package atm;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
   public static void main(String[] args) {
      ApplicationContext context =
            new ClassPathXmlApplicationContext("config.xml");

      ATMSimulator atmSimulator = context.getBean("atmSim",
                                                  ATMSimulator.class);
      atmSimulator.run();
   }
}
```

ใช้สร้าง object ที่ต้องการ แทนการใช้ new

https://github.com/fsciusa/atm-spring-xml
https://github.com/fsciusa/atm-spring-annotation

ไฟล์ config.xml

ใส่ในโฟลเดอร์ resources

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="dataSource" class="atm.DataSource">
        <constructor-arg value="customers.txt"/>
    </bean>
    <bean id="bank" class="atm.Bank">
        <constructor-arg ref="dataSource"/>
    </bean>
    <bean id="atm" class="atm.ATM">
        <constructor-arg ref="bank"/>
    </bean>
    <bean id="atmSim" class="atm.ATMSimulator">
        <constructor-arg ref="atm"/>
    </bean>

</beans>
```

https://github.com/fsciusa/atm-spring-xml
https://github.com/fsciusa/atm-spring-annotation