Instantiation

Static Type vs. Dynamic Type

```java
Collection<User> users = new ArrayList<User>();
```

**Static Type**
- Type Declaration
- Defines
  - Methods
  - Intention
  - Limits Behavior
- Could be
  - Class
  - Abstract Class
  - Interface

**Dynamic Type**
- Instantiation
- Actual Type
- Determine Behavior
  - instanceof Static Type
    - implements
    - extends
  - Can be Instantiated.
Instantiation

Abstraction

Liskov Substitution Principle

New Operator

Notes
**new Operator**

```java
Rectangle r = new Rectangle(0,0,1,1);
/* where
Rectangle(double p1x,double p1y,..){
    this.p1 = new Point(p1x,p1y); ... 
} */
//or
Point p1 = new Point(0,0);
Point p2 = new Point(1,1);
Rectangle r = new Rectangle(p1,p2);
//or
Rectangle r = new Rectangle(
    new Point(0,0),
    new Point(1,1);
);
```

---

**Instantiation vs. Usage**

<table>
<thead>
<tr>
<th>Instantiation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Costly</strong></td>
<td>Use Object via Methods</td>
</tr>
<tr>
<td><strong>Resource Allocation</strong></td>
<td>Depends on Expected Behavior</td>
</tr>
<tr>
<td><strong>Memory Allocation</strong></td>
<td>Static Type</td>
</tr>
<tr>
<td><strong>Takes Time</strong></td>
<td>Interface Required</td>
</tr>
<tr>
<td><strong>Establish Network Connection</strong></td>
<td>Should not Create Object.</td>
</tr>
<tr>
<td><strong>Could be Difficult</strong></td>
<td>Dynamic Type could be Changed</td>
</tr>
<tr>
<td><strong>Tons of Parameters</strong></td>
<td>Reusability</td>
</tr>
<tr>
<td><strong>JDBC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Defines Dynamic Type</strong></td>
<td></td>
</tr>
</tbody>
</table>

Object creation and usage should be separated!

---

**Consequences**

**Pro**
- Instantiation could be Centralized.
- Testing Become Easier.
  - See "Software Projects & Testing" subject.
- Increase Reusability.
- More General Code

**Question**
- How will we get the dependencies?
Participants

service an object that will be used
client object that uses the service
interface defines how the client uses the service
injector creates the service and gives it into the client.

Goal
Separation of the creation of the client's dependencies from the client's behavior.

Types of Dependency Injection

Field Injection
- Field Injection Frameworks know
- Should not be used.
- Reflection

Constructor Injection
- Dependencies are Passed as Constructor Arguments
- Cannot Create object when a Dependency is Missing
- Mandatory Dependencies

Setter Injection
- Setter Methods are Invoked After Constructor Used
- Optional Dependencies
## Constructor Injection

- Mandatory Dependencies
- Part of Client Initialization
- Block the Client's Functionality
- Examples
  - Database Access

```java
class Client{
    private MandatoryService ms;
    private OptionalService os;
    public Client(MandatoryService ms){
        this.ms = ms;
    }
}
```

## Setter Injection

- Optional Dependencies
- Not Required for the Client
- Extends, Enhance its Functions
- Examples
  - Logging
  - Publishing Event

```java
class Client{
    private MandatoryService ms;
    private OptionalService os;
    public setOptionalService(OptionalService os ){
        this.os = os;
    }
}
```

## Typical Service Categories

- Storage
  - Database Access Objects
- Business Logic
  - Other Functions
  - Micro Services
- 3rd Party Services
  - System Integration
  - "Distributed Systems" subject in MSc.
  - Social Media
  - Currency Change
  - www.programmableweb.com
Consequences

+ Flexibility and Configurability
+ Refactoring
+ Increase Abstractness
+ Cleaner Code
+ Independent Development
+ Facilitates Testing and Mocking
  - Configuration Required
  - Difficult to Understand
  - Dependency Injection Framework is Required

Notes
Core Elements

**Core**
- Basic Tasks
- Resource Finding

**Beans**
- Bean Definition
- XML Mapping

```
#{systemProperties['user']}
#{T(java.lang.Math).random()}
```

**Context**
- ApplicationContext

**SpEL**
- Spring Expression Language
  - Text Evaluation
  - XML based Bean Definition
  - #{string expr.}

**Super Interfaces**
- BeanFactory
- ResourceLoader

**Implementations**
- Annotation Config Application Context
- ClassPath Xml Application Context
- Xml Web Application Context
**BeanFactory**

- interface
- Bean Container
- Registry of Application Components
- Implementations (27)
  - XML
  - Annotation

**Bean**

- Object
- String ID
- Dependency
- Types
  - Prototype
  - Singleton

**interface**

- containsBean(String)
- getBean(String)
- getBean(Class<T>)
- isPrototype(String)
- isSingleton(String)
- isTypeMatch(String, Class<T>)

**Bean Life-cycle**

1. BeanNameAware’s `setBeanName`
2. BeanClassLoaderAware’s `setBeanClassLoader`
3. BeanFactoryAware’s `setBeanFactory`
4. EnvironmentAware’s `setEnvironment`
5. EmbeddedValueResolverAware’s `setEmbeddedValueResolver`
6. ResourceLoaderAware’s `setSelfResourceLoader` (only applicable when running in an application context)
7. ApplicationEventPublisherAware’s `setApplicationEventPublisher` (only applicable when running in an application context)
8. MessageSourceAware’s `setMessageSource` (only applicable when running in an application context)

**ApplicationContextAware’s** `setApplicationContext` (only applicable when running in an application context)

**ServletContextAware’s** `setServletContext` (only applicable when running in a web application context)

**InitializingBean’s** `afterPropertiesSet`

**postProcessBeforeInitialization** methods of BeanPostProcessors
Application Context - XML

- AbstractXmlApplicationContext
  - ClassPathXmlApplicationContext
  - FileSystemXmlApplicationContext
- XML
  - Text based
  - Human Readable
  - Security
  - Reconfigurable

Location
- $basedir/src/main/resources
- WEB-INF/

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="...
xmlns:xsi="...
xmlns:p="...
xmlns:context="...
xmlns:aop="...
xmlns:mvc="...
xsi:schemaLocation="...">
  <import resource="si-config-measurement.xml"/>
  ...
</beans>
```

```xml
<bean id="wifiDistanceCalculator"
  class="<packages>.VectorIntersectionWiFiRSSIDistance">
</bean>

<bean id="measurementDistanceCalculator"
  class="<packages>.MeasurementDistanceCalculatorImpl">
  <constructor-arg ref="wifiDistanceCalculator"/>
  <constructor-arg name="wifiDistanceWeight" value="1.0"/>
  <constructor-arg name="magnetometerDistanceWeight" value="0.5"/>
  <constructor-arg name="gpsDistanceWeight" value="0.0"/>
</bean>
```
Spring Core

Application Context - XML

```xml
<bean id="positioningService"
     class="<packages>.KNNSimplePositioning">
  <constructor-arg name="distanceCalculator" ref="measurementDistanceCalculator"/>
  <constructor-arg name="measurementGateway" ref="MeasurementGateway"/>
  <constructor-arg name="k" value="3"/>
</bean>
</beans>
```

Application Context - Annotation

Sprint Annotations
- Java Based
- XML is not required.
- Not Readable
- Configuration ???
- Annotations
  - @Component
  - @Configuration
  - @Bean

@Bean
- Method
  - Returns with a Bean
- Bean Name
  - Method Name
  - Specified names
  - Alias
- Optional Parameters
  - `@Autowired`: Autowire
  - `initMethod`: String
  - `destroyMethod`: String
  - `name`: String
  - `value`: String

Notes
```java
@Configuration
public class ExampleConfiguration {

    @Bean(name = "greetingsService")
    public GreetingsService initGreetingsService(){
      return new GreetingsHU();
    }
}
```