Autowiring-

Why?

Suppose I have two classes Employee and address both are the dependent on each other so in that case we should go for autowiring concepts.

What is Autowiring?

Autowiring feature of spring framework enables you to inject the object dependency implicitly.

Note-

1. We can inject only secondary types.
2. Autowiring can not apply to primitives types. Spring container will take care of automatic dependency injection.
3. You can apply autowiring through constructor or setter method.

There are several types of autowiring are as follows.

- ByType-(internally it uses setter base injection)
- ByName--(internally it uses setter base injection)
- Constructor-(internally it uses constructor base injection)
- autodetect-(it uses both setter and constructor base injection.)

ByType-

In this case, spring framework attempts to find out a bean in the configuration file, whose bean id (policy) is matching with the property type (categories.java private Policy policy) to be wired. If a bean found with class as property type then that class object will be injected into that property by calling setter injection.

If no class found then that property remains un-wired, but never throws any exception just like before.

Policy.java

```
package com.spring.auto;

public class Policy {

    private String planName;
    private int planAmount;
```

```
	public String getPlanName() {
		return planName;
	}

	public void setPlanName(String planName) {
		this.planName = planName;
	}

	public int getPlanAmount() {
		return planAmount;
	}

	public void setPlanAmount(int planAmount) {
		this.planAmount = planAmount;
	}

}
```

Categories.java

```
package com.spring.auto;

public class Categories {

	private String name;
	private Policy policy;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Policy getPolicy() {
		return policy;
	}
```

```
	public void setPolicy(Policy policy) {
		this.policy = policy;
	}

	public void show() {
		System.out.println("Categories name>>" + name);
		System.out.println("Policy Name>>" +
policy.getPlanName());
		System.out.println("Policy Amount>>" +
policy.getPlanAmount());
	}

}
```

Client.java

```
package com.test;


import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;


public class Client {

	public static void main(String[] args) {

		Resource res = new ClassPathResource("spring.xml");

		BeanFactory factory = new XmlBeanFactory(res);

		Categories categories = (Categories) factory.getBean("categories");
```

```
            categories.show();
      }
}
```

Spring.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/schem
a/beans
    http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">

     <bean id="categories" class="com.spring.auto.Categories"
autowire="byType">
          <property name="name" value="Life Insurance" />
     </bean>
     <bean id="policy" class="com.spring.auto.Policy">
          <property name="planName" value="Term Plan" />
          <property name="planAmount" value="8500" />
     </bean>
</beans>
```

Pom.xml (same as last program).

Output-

Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500

Internal working of byType

We called `categories` from Test.java [line number 14], then it will navigate to spring.xml file and check the bean id as categories, also read the autowire=byType, so spring container will checks for the bean id with class attribute policy in xml file

then navigate to Categories.java and check the policy type whether it is exactly matching then inserts policy objects into categories.

Test.java we used to type cast to get our output.

ByName

In this case, spring framework attempts to find out a bean in the configuration file, whose bean id (policy) is matching with the property type (categories.java private Policy policy) to be wired. If a bean found with class as property type then that class object will be injected into that property by calling setter injection.

If no class found then that property remains un-wired, but never throws any exception just like before.

Policy.java

```
package com.spring.auto;

public class Policy {

	private String planName;
	private int planAmount;

	public String getPlanName() {
		return planName;
	}

	public void setPlanName(String planName) {
		this.planName = planName;
	}

	public int getPlanAmount() {
		return planAmount;
	}

	public void setPlanAmount(int planAmount) {
		this.planAmount = planAmount;
	}
```

```
}
```

Categories.java

```
package com.spring.auto;

public class Categories {

	private String name;
	private Policy policy;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Policy getPolicy() {
		return policy;
	}

	public void setPolicy(Policy policy) {
		this.policy = policy;
	}

	public void show() {
		System.out.println("Categories name>>" + name);
		System.out.println("Policy Name>>" +
policy.getPlanName());
		System.out.println("Policy Amount>>" +
policy.getPlanAmount());
	}

}
```

Test.java

package com.test;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

```
package com.spring.auto;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class Test {

     public static void main(String[] args) {

          Resource res = new ClassPathResource("spring.xml");
          BeanFactory factory = new XmlBeanFactory(res);

          Categories categories = (Categories)
factory.getBean("categories");
          categories.show();
     }
}
```

Note-

1. In byName,constructor and autodetect, Categories object name(categories) and getBean("categories)" must be same and also bean id="categories" should be same.

2. In spring.xml, bean id=”policy” and Categories.java(private Policy policy) must be same.

Spring.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schem
a/beans
    http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd"><bean id="policy"
class="com.spring.auto.Policy">
          <property name="planName" value="Term Plan" />
          <property name="planAmount" value="8500" />
     </bean>

     <bean id="categories" class="com.spring.auto.Categories"
          autowire="byName">
          <property name="name" value="Life Insurance" />
     </bean>
</beans>
```

Internal working byName

We called `categories` from Test.java [line number 14], then it will navigate to spring.xml file and check the bean id as categories, also read the autowire=byType, so spring container will checks for the bean id with class attribute policy in xml file then navigate to Categories.java and check the policy type whether it is exactly matching then inserts policy objects into categories,

and then injects value “Life Insurance” into name property of Categories class.

Test.java we used to type cast to get our output.

Pom.xml (same as last program).

Output-

Categories name>>Life Insurance
Policy Name>>Term Plan
Policy Amount>>8500

Constructor-

Spring Autowiring by constructor is similar to spring autowiring byType [internally it will considers as byType only] but with little difference, in byType we used setter injection here we have to use constructor injection.

Policy.java

```
package com.spring.auto;

public class Policy {

	private String planName;
	private int planAmount;

	public String getPlanName() {
		return planName;
	}

	public void setPlanName(String planName) {
		this.planName = planName;
	}

	public int getPlanAmount() {
		return planAmount;
	}

	public void setPlanAmount(int planAmount) {
		this.planAmount = planAmount;
	}

}
```

Categories.java

```
package com.spring.auto;

public class Categories {

	private String name;
```

```
	private Policy policy;

	public Categories(Policy policy) {
		this.policy = policy;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Policy getPolicy() {
		return policy;
	}

	public void setPolicy(Policy policy) {
		this.policy = policy;
	}

	public void show() {
		System.out.println("Categories name>>" + name);
		System.out.println("Policy Name>>" +
policy.getPlanName());
		System.out.println("Policy Amount>>" +
policy.getPlanAmount());
	}

}
```

Test.java

```
package com.spring.auto;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
```

```
import org.springframework.core.io.Resource;

public class Test {

	public static void main(String[] args) {

		Resource res = new ClassPathResource("spring.xml");
		BeanFactory factory = new XmlBeanFactory(res);

		Categories categories = (Categories)
factory.getBean("categories");
		categories.show();

	}
}
```

Spring.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schem
a/beans
    http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">



	<bean id="policy" class="com.spring.auto.Policy">
		<property name="planName" value="Term Plan" />
		<property name="planAmount" value="8500" />
	</bean>

	<bean id="categories" class="com.spring.auto.Categories"
		autowire="constructor">
		<property name="name" value="Life Insurance" />
	</bean>

</beans>
```

Output-

Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500

autodetect

Actually spring autowire=”autodetect” first will works as Spring Autowiring constructor if not then works as Spring Autowiring byType, byType means setter injection.

Note- same as above example except one change in spring.xml file

<bean id="id1" class="com.test.Categories" autowire="autodetect">

Policy.java

```
package com.spring.auto;

public class Policy {

	private String planName;
	private int planAmount;

	public String getPlanName() {
		return planName;
	}

	public void setPlanName(String planName) {
		this.planName = planName;
	}

	public int getPlanAmount() {
		return planAmount;
	}

	public void setPlanAmount(int planAmount) {
		this.planAmount = planAmount;
	}
```

```
}
```

Categories.java

```
package com.spring.auto;

public class Categories {

	private String name;
	private Policy policy;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Policy getPolicy() {
		return policy;
	}

	public void setPolicy(Policy policy) {
		this.policy = policy;
	}

	public void show() {
		System.out.println("Categories name>>" + name);
		System.out.println("Policy Name>>" +
policy.getPlanName());
		System.out.println("Policy Amount>>" +
policy.getPlanAmount());
	}

}
```

Test.java

```java
package com.spring.auto;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class Test {

	public static void main(String[] args) {

		Resource res = new ClassPathResource("spring.xml");
		BeanFactory factory = new XmlBeanFactory(res);

		Categories categories = (Categories)
factory.getBean("categories");
		categories.show();

	}
}
```

Spring.xml

```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/schem
a/beans
    http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">

	<bean id="categories" class="com.spring.auto.Categories"
autowire="autodetect">
		<property name="name" value="Life Insurance" />
	</bean>
	<bean id="policy" class="com.spring.auto.Policy">
		<property name="planName" value="Term Plan" />
		<property name="planAmount" value="8500" />
	</bean>
```

```
</beans>
```

Output-

Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500