Code Explanation and Documentation

1. Test.java

This is the main class of the Spring application. It demonstrates the usage of Spring's dependency injection using annotations. Let's go through the code step-by-step:

1. Import Statements: The required import statements are present at the beginning of the file to include the necessary classes.

2. `main` Method: The `main` method serves as the entry point of the application.

3. ApplicationContext: The `ApplicationContext` is the core Spring container responsible for managing beans and their dependencies. It is initialized here using the `ClassPathXmlApplicationContext` class, which loads the Spring configuration file named "autowire.xml" from the classpath.

4. Retrieving the Bean: The `Employee` bean is retrieved from the `ApplicationContext` using the `getBean()` method. The bean name used to retrieve the `Employee` object is "emp1".

5. Printing the Employee: Finally, the details of the employee are printed on the console using the `toString()` method of the `Employee` class.

2. Employee.java

This is the `Employee` class that represents an employee entity. It is a POJO (Plain Old Java Object) class with properties and getter/setter methods for dependency injection. Let's understand the code:

1. Properties: The class has three properties - `empName`, `empId`, and `address`.

2. `@Autowired`: The `@Autowired` annotation is used above the `address` property. This annotation tells Spring to automatically inject the `Address` bean into the `Employee` bean, making it possible for the `Employee` instance to use the `Address` instance without explicitly creating it.

3. Getter/Setter Methods: The getter and setter methods are defined for all properties, allowing Spring to set the property values.

4. Constructor: There is a default constructor defined.

5. `toString()` Method: An overridden `toString()` method is provided to display the details of the `Employee` object.

3. Address.java

This class represents the `Address` entity. It is also a POJO class with properties and getter/setter methods. The code is straightforward:

1. Properties: The class has two properties - `street` and `city`.

2. Getter/Setter Methods: The getter and setter methods are defined for both properties.

3. Constructor: There is a default constructor defined.

4. `toString()` Method: An overridden `toString()` method is provided to display the details of the `Address` object.

4. autowire.xml

This XML configuration file defines the Spring beans for the application using the `beans` element. Let's see the content of the file:

1. `xmlns` and `xsi:schemaLocation`: The XML file uses the `http://www.springframework.org/schema/beans` and `http://www.springframework.org/schema/context` namespaces with corresponding schema locations to define the Spring beans and annotations.

2. `context:annotation-config`: This element enables the support for Spring's annotations in the application.

3. Address Bean: A bean of type `Address` is defined with the name "address." It has properties `street` and `city` initialized using the `p:` namespace (property placeholder). This means that Spring will use the `setStreet` and `setCity` methods to set the values for these properties.

4. Employee Bean: A bean of type `Employee` is defined with the name "emp1." It has properties `empName` and `empId` initialized using the `p:` namespace. The `@Autowired` annotation in the `Employee` class will automatically inject the `Address` bean into this bean's `address` property.

Summary

In summary, the provided code demonstrates a simple Spring application that uses annotation-based dependency injection. The application defines two classes - `Employee` and `Address` - as POJOs. It also configures Spring beans using an XML configuration file (`autowire.xml`). The `Test` class serves as the main class that demonstrates the injection of the `Address` bean into the `Employee` bean using the `@Autowired` annotation. When executed, it prints the details of an employee, including the associated address.