Employee Class

- In an organization, an employee belongs to a department.
- When modeling an application for the organization, the relationship between the classes must be determined.
- The employee is a class that is dependent on the department class.

```java
public class Employee {
    private int empId;
    private String empName;
    private String address;
    private Department department;

    public Employee(int empId, String empName,
                    String address,
                    Department department) {
        this.empId = empId;
        this.empName = empName;
        this.address = address;
        this.department = department;
    }
    public int getEmpId() { return empId; }
    public void setEmpId(int empId) { this.empId = empId; }
    public String getEmpName() { return empName; }
    public void setEmpName(String empName) { this.empName = empName; }
    public String getAddress() { return address; }
    public void setAddress(String address) { this.address = address; }
    public Department getDepartment() { return department; }
    public void setDepartment(Department department) { this.department = department; }
}
```

Slide Note

```java
public class Department {
    private int deptId;
    private String deptName;

    public Department() {
    }

    public Department(int deptId, String deptName) {
        this.deptId = deptId;
        this.deptName = deptName;
    }

    @Override
    public String toString() {
        return "Department{" +
                "deptId=" + deptId +
                ", deptName='" + deptName + '\'' +
                '}';
    }
}
```

Department Class

- The attributes of the `Department` class are defined.

Slide Note

The Implementation Class

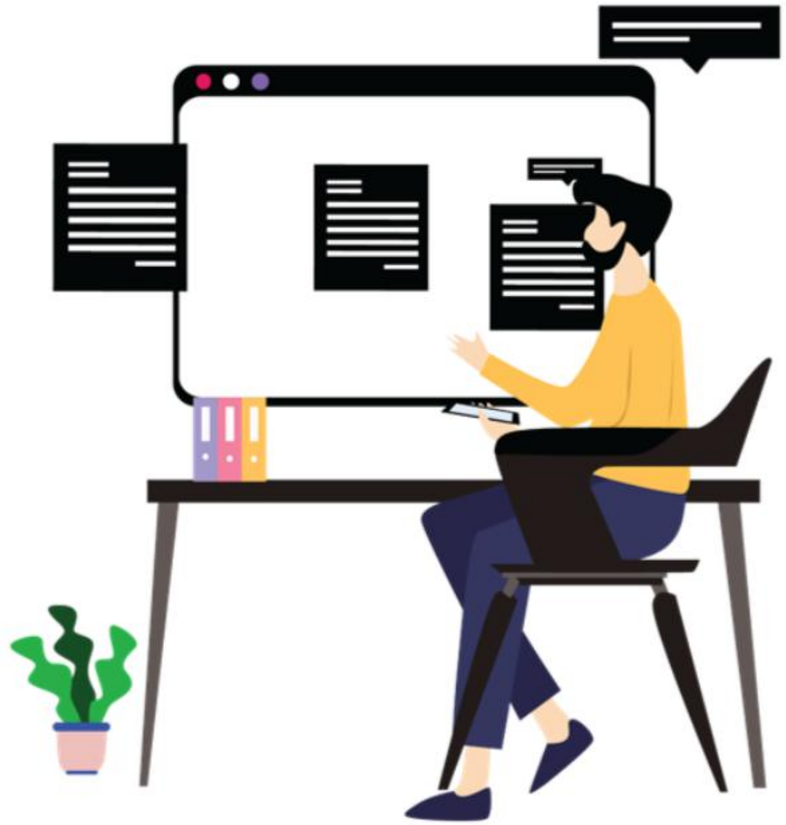
- How can you specify the `Department` class object in the employee class?
- Is the new keyword used to create the object of the `Department` inside the `Employee` class?
- If there are any changes in the `Department` class attributes, will they affect the `Employee` class also?
- Do you think this will be easier for large applications that deal with multiple objects?

```
public class EmployeeImpl {
    public static void main(String[] args) {
            Employee employee = new Employee( empId: 101, empName: "Helen",
                    address: "223 main Harlow",
                    new Department( deptId: 301, deptName: "Sales"));
            System.out.println(employee);
    }
}
```

The `Department` object must be manually created and provided by the programmer to `Employee`.

Slide Note

Implement Inversion of Control (IoC) Inside the Spring Application by Using Annotations



Slide Note

Learning Objectives

- Manage Spring Objects
- Perform Different Types of Autowiring
- Autowire a Property
- Autowire a Setter
- Autowire a Constructor

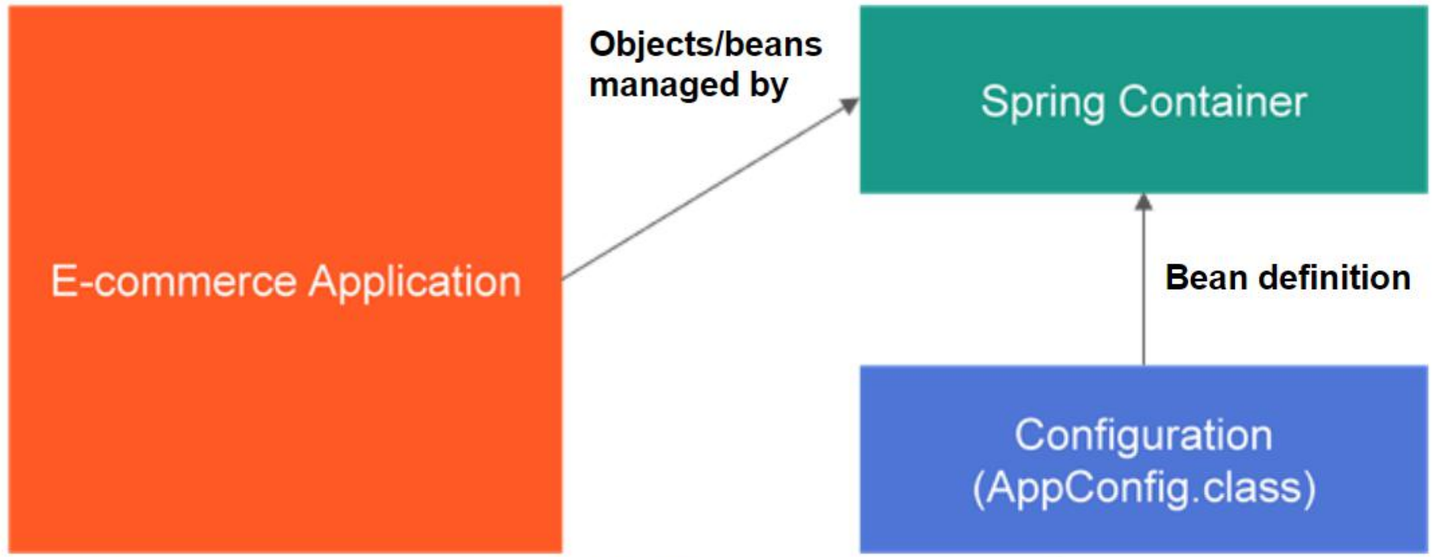
Slide Note

Managing Spring Objects

Slide Note

Spring-Managed Objects

- The process of creating and managing associations among application objects or beans forms the core of DI.



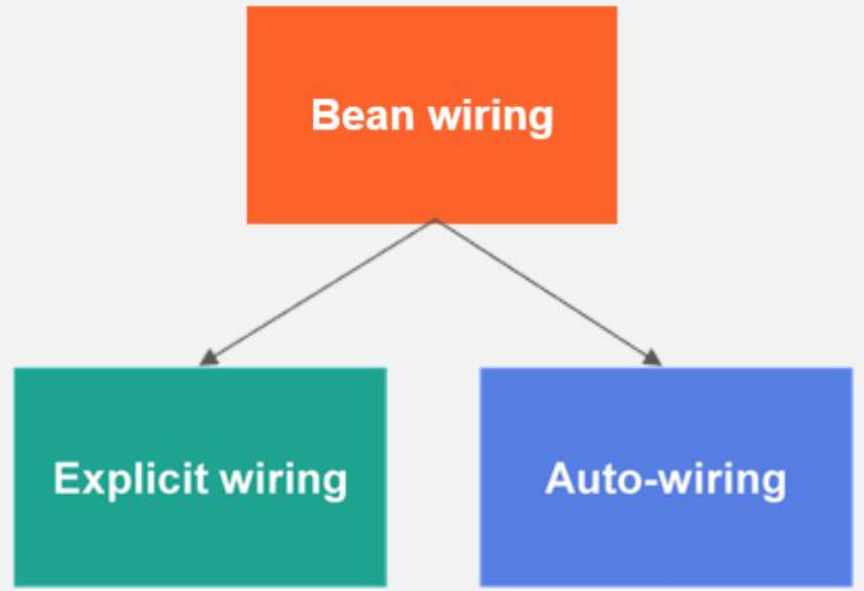
- Now that the objects are managed by the Spring core container, the need to create objects using the new keyword is eliminated.
- The program can make a reference to the beans available in the configuration as required. This is achieved by means of autowiring provided by the Spring Framework.

Slide Note

Wiring

- The process of creating associations between application objects, or beans, is commonly referred to as wiring. For example, a customer has an address.
- Beans can be wired in two ways by the Spring container:
  - Explicit wiring – The bean dependencies are explicitly wired in the configuration file.
  - Autowiring – The bean dependencies are not stated explicitly; the container automatically injects the appropriate dependencies.

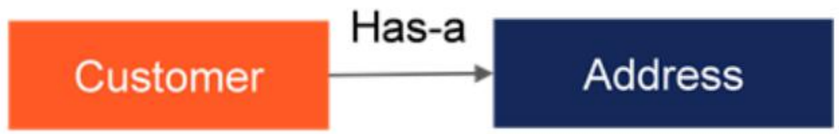
Note: Explicit wiring is done using an XML configuration file, which is not in this course's scope.



Slide Note

Autowiring

- Autowiring injects the object dependency implicitly.
- The Spring container finds the type and name of the property and matches the same with other beans within the container to resolve dependencies.
- For example, a Customer can have an Address. The 'Customer' and 'Address' are two objects that have a relationship between them.
- When the Customer object is created, the 'Address' object will be injected into the 'Customer' by the Spring container.



Slide Note

Autowiring – Annotation-based Configuration

- The `@Autowired` annotation in the Spring automatically injects the dependent beans into the associated references of a POJO class.
- The `@Autowired` annotation can be used on the following:
  - Property
  - Setter method
  - Constructor

Slide Note

Quick Check

The `@Autowired` annotation cannot be used on _____.

1. Class
2. Property
3. Setter
4. Constructor

Slide Note

Quick Check: Solution

The `@Autowired` annotation cannot be used on _____.

1. **Class**
2. **Property**
3. **Setter**
4. **Constructor**

Slide Note

Performing Different Types of Autowiring

Slide Note

```
public class Customer
{

    private int customerId;
    private String customerName;
    private Address address;
```

```
public class Address {
    private int houseNo;
    private String street;
    private String city;
    private int zipCode;
```

Create the Classes

- Define the Customer and Address classes with appropriate attributes.

Slide Note

```
public class Customer
{

    private int customerId;
    private String customerName;
    @Autowired
    private Address address;
```

@Autowired

- Use the `@Autowired` annotation on the Address object in the Customer class.

Slide Note

The Main Class

- Call the customer bean in the main class and print the values.

```
public class Main
{
    public static void main(String[] str){

        ApplicationContext ctx =
                new AnnotationConfigApplicationContext(AppConfig.class);
            Customer customer = ctx.getBean( s: "customer", Customer.class);
        System.out.println();
        System.out.println();
            System.out.println(customer);


    }
}
```

- The output, along with the address, is displayed.

```
Customer{customerId=101, customerName='Henry',
```

```
address=Address{houseNo=23, street='Dimora Street', city='New Jersey', zipCode=7831}}
```

Slide Note

Autowiring a Property

Slide Note

@Autowired – Property

- `@Autowired` **can be used on the property of the class.**
- `Address` **is a property of the** `Customer` **class.**

```
public class Customer
{

    private int customerId;
    private String customerName;
    @Autowired
    private Address address;
```

Autowiring a property

Slide Note

Customer and Address – Autowire Property

Customer credentials for logging into an e-commerce application need to be captured. A customer can have an associated address. Manage the beans of the application in the Spring container.

- Add appropriate dependencies to the pom.xml.
- Use the annotation-based configuration to manage the beans.
- Use autowiring to inject dependencies.
- Display the customer's name on the console.
- Click here for the solution.

DEMO

Slide Note

Autowiring a Setter

Slide Note

@Autowired – Setter

- `@Autowired` can be used on the setter property of the class.
- `Address` is a property of the `Customer` class and has a setter.

```
@Autowired
public void setAddress(Address address) {
    this.address = address;
}
```

Autowiring a setter

Slide Note

Note: Autowire the setter now. Use the example from the previous demo.

Customer and Address – Autowire Setter

Customer credentials for logging into an e-commerce application need to be captured. A customer can have an associated address. Manage the beans of the application in the Spring container.

- Add appropriate dependencies to the pom.xml.
- Use the annotation-based configuration to manage the beans.
- Use autowiring to inject dependencies.
- Display the customer's name and address on the console.
- Click here for the solution.

DEMO

Slide Note

Autowiring a Constructor

Slide Note

@Autowired – Constructor

- `@Autowired` can be used on the constructor of the class.

- The `Address` object must be passed in the constructor of the `Customer` class object.

```java
@Autowired
public Customer(Address address) {
    this.address = address;
}
```

Autowiring a constructor

```java
@Bean("customer")
public Customer getCustomerDetails()
{
    Customer customer = new Customer(getAddress());
    customer.setCustomerId(101);
    customer.setCustomerName("Henry");
    return customer;
}

@Bean
public Address getAddress()
{
    return new Address( houseNo: 23,
            street: "Dimora Street",
            city: "New Jersey", zipCode: 7831);
}
```

Slide Note

Constructor Autowiring

- Constructor autowiring is preferred over other autowiring methods.
- This type of injection is safer as the objects will not be created if the dependencies are not available or cannot be resolved.

Slide Note

Customer and Address – Autowire Constructor

Customer credentials for logging into an e-commerce application need to be captured. A customer can have an associated address. Manage the beans of the application in the Spring container.

- Add appropriate dependencies to the pom.xml.
- Use the annotation-based configuration to manage the beans.
- Use autowiring to inject dependencies.
- Display the customer's name on the console.
- Click here for the solution.

DEMO