Spring Core Container:

The Spring Framework is a popular framework for building enterprise-level Java applications. It provides various modules and features to simplify the development of complex applications. One of the fundamental parts of Spring is the Spring Core Container, which is responsible for managing and wiring the application's components. The core container consists of two main components: the **Bean Factory** and the **Application Context**. I'll explain both components with code examples.

1. Bean Factory:

The Bean Factory is the core part of the Spring Core Container. It manages the beans (Java objects) in the application, including creating, configuring, and wiring them together. Here's an example of how to define and use beans with the Bean Factory:

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

public class MainApp {
    public static void main(String[] args) {
        // Create a BeanFactory and load the configuration file
        BeanFactory factory = new XmlBeanFactory(new ClassPathResource("beans.xml"));

        // Retrieve a bean from the Bean Factory
        HelloWorldService service = (HelloWorldService) factory.getBean("helloWorldService");

        // Use the bean
        service.sayHello();
    }
}
```

In the above code, we create a `BeanFactory` by loading a configuration file called `beans.xml`. This configuration file defines a bean named `helloWorldService`. We then retrieve the `helloWorldService` bean from the Bean Factory and call the `sayHello()` method on it.

2. Application Context:

The Application Context is an advanced version of the Bean Factory and provides additional features such as internationalization, event propagation, and more. Here's an example of how to use the Application Context:

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

public class MainApp {
    public static void main(String[] args) {
        // Create an Application Context and load the configuration file
        ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

        // Retrieve a bean from the Application Context
        HelloWorldService service = (HelloWorldService) context.getBean("helloWorldService");

        // Use the bean
        service.sayHello();
    }
}
```

In this code, we create an `ApplicationContext` by loading the `beans.xml` configuration file. Then, we retrieve and use the `helloWorldService` bean from the Application Context. The main difference is that the Application Context offers more features compared to the basic Bean Factory.

In both examples, the `beans.xml` configuration file defines the `HelloWorldService` bean, which is a simple service that prints "Hello, World!" when its `sayHello()` method is called. The Spring Core Container manages the bean's lifecycle and dependencies.

These examples provide a basic understanding of the Spring Core Container and how to configure and use beans. Spring's real power comes when dealing with more complex configurations, such as dependency injection, aspect-oriented programming, and transaction management, which Spring provides out of the box.