Komal Rawat
Session: Spring Framework

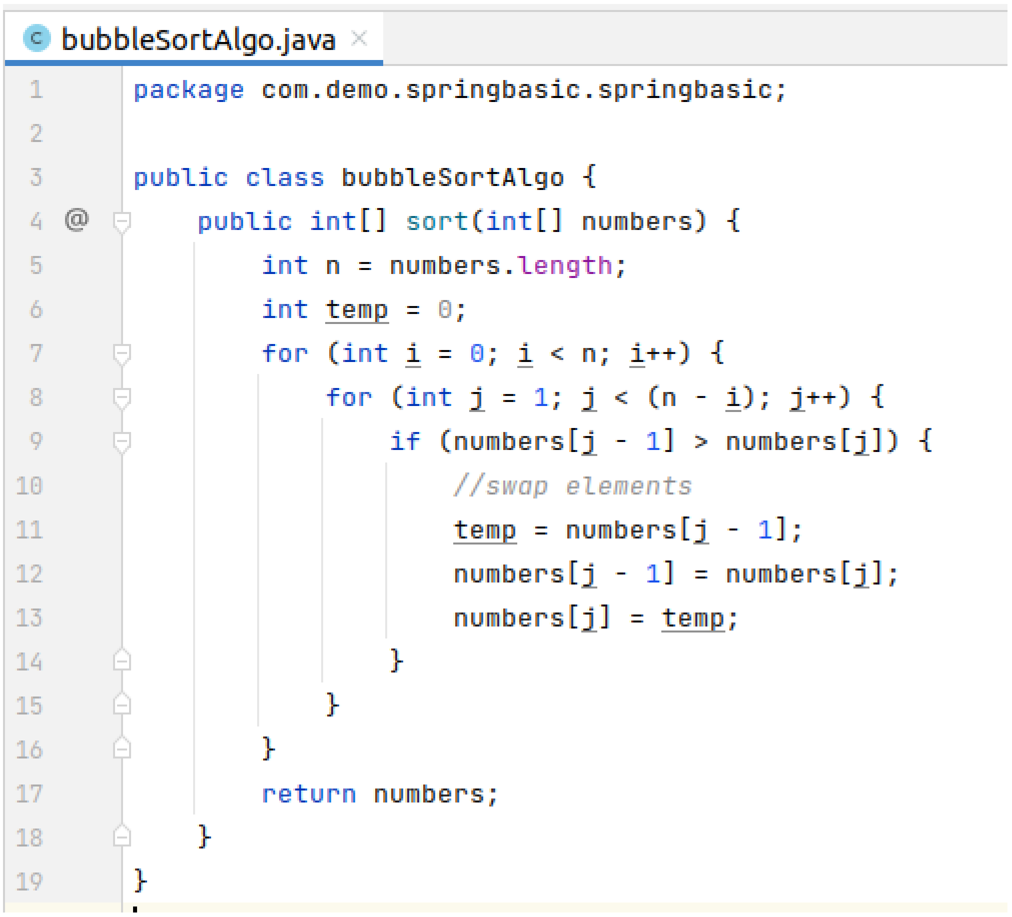
---

1) Write a program to demonstrate Tightly Coupled code.

Sol. Tightly Coupled: It is when classes or group of classes are highly dependent on each other.

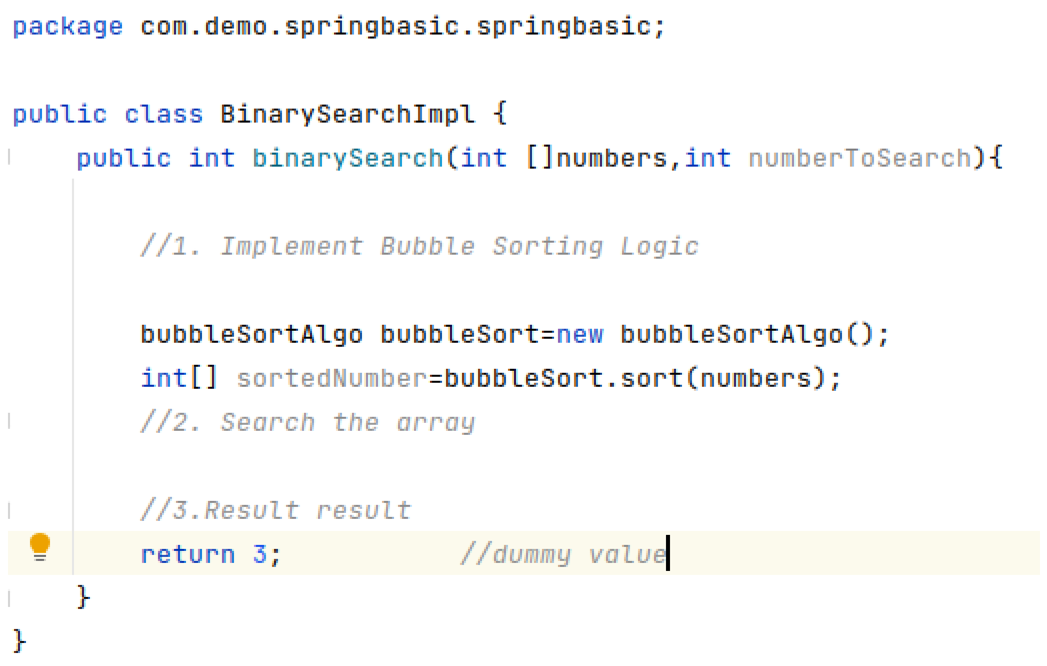
For example. Let's say I want to do binary search on an array. So, to implement Binary search the array has to be sorted first. Considering we use Bubble sort for sorting. I can either put the sorting logic in the binary search class as follows or create a separate class and use its implementation. But what if we want to change the logic or say we want to use another sorting algorithm instead of bubble sort, then we have to change the logic manually and multiple times in case of a big project. So, this is the concept of tight coupling. Here Binary search is highly dependent on Bubble sort.

bubbleSortAlgo.java



```java
package com.demo.springbasic.springbasic;

public class bubbleSortAlgo {
    public int[] sort(int[] numbers) {
        int n = numbers.length;
        int temp = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (numbers[j - 1] > numbers[j]) {
                    //swap elements
                    temp = numbers[j - 1];
                    numbers[j - 1] = numbers[j];
                    numbers[j] = temp;
                }
            }
        }
        return numbers;
    }
}
```

binarySearchImpl.java



```java
package com.demo.springbasic.springbasic;

public class BinarySearchImpl {
    public int binarySearch(int []numbers,int numberToSearch){

        //1. Implement Bubble Sorting Logic

        bubbleSortAlgo bubbleSort=new bubbleSortAlgo();
        int[] sortedNumber=bubbleSort.sort(numbers);
        //2. Search the array

        //3.Result result
        return 3;           //dummy value
    }
}
```

(2) Write a program to demonstrate Loosely Coupled code.

Sol. Loosely Coupled: Loose coupling in Java means that the classes are independent of each other. The only knowledge one class has about the other class is what the other class has exposed through its interfaces in loose coupling.

Now as seen above, Binary Search is tightly coupled with bubble Sort. The way to make it loosely coupled is by using interfaces.

SortAlgorithm.java

```java
package com.demo.springbasic.springbasic;

public interface SortAlgorithm {

   public int[] sort(int[] numbers);

}
```

We'll make BubbleSortAlgo and QuickSortAlgorithm to implement the SortAlgorithm Interface. Through this we can dynamically decide which algorithm to use. Here I have used Constructor Injection by passing the sorting algorithm I want to use through the constructor.

BinarySearchImpl.java

```java
package com.demo.springbasic.springbasic;
```

```java
public class BinarySearchImpl {
   private SortAlgorithm sortAlgorithm;

   public BinarySearchImpl(SortAlgorithm sortAlgorithm) {
       this.sortAlgorithm = sortAlgorithm;
   }

   public int binarySearch(int []numbers,int numberToSearch){
       //1. Implement Bubble Sorting Logic
       // BubbleSortAlgo bubbleSort=new BubbleSortAlgo();    //Tight Coupling
       int[] sortedNumber=sortAlgorithm.sort(numbers);       //Loose Coupling
       System.out.println(sortAlgorithm);
       //2. Search the array
       //3.Result result
       return 3;           //dummy value
   }
}
```

SpringBasicApplication.java

```java
package com.demo.springbasic.springbasic;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringBasicApplication {
  public static void main(String[] args) {
     BinarySearchImpl binarySearch=new BinarySearchImpl(new BubbleSortAlgo());
    //BinarySearchImpl binarySearch1=new BinarySearchImpl(new
QuickSortAlgorithm());
     int result=binarySearch.binarySearch(new int[]{4,7,5,3,41},3);
     System.out.println(result);
     //SpringApplication.run(SpringBasicApplication.class, args);
  }
}
```

We have made Sorting algo as a separate dependency which is a dependency of Binary Search.

(3) Use @Compenent and @Autowired annotations in Loosely Coupled code for dependency management.

Sol. We have created a lot of objects in the program and we are wiring them together. But all this can be controlled by Spring framework which can manage the dependencies and beans and to wire them.

What are the Beans?

The **@Component** annotation is used to load a java class as a bean.

Here, BinarySearchImpl, BubbleSortAlgo and QuickSortAlgo are the beans.

What are the dependencies of the beans?

The annotation **@Autowired** in spring boot is used to auto-wire a bean into another bean for dependency injection.

Here, SortAlgorithm is the dependency.

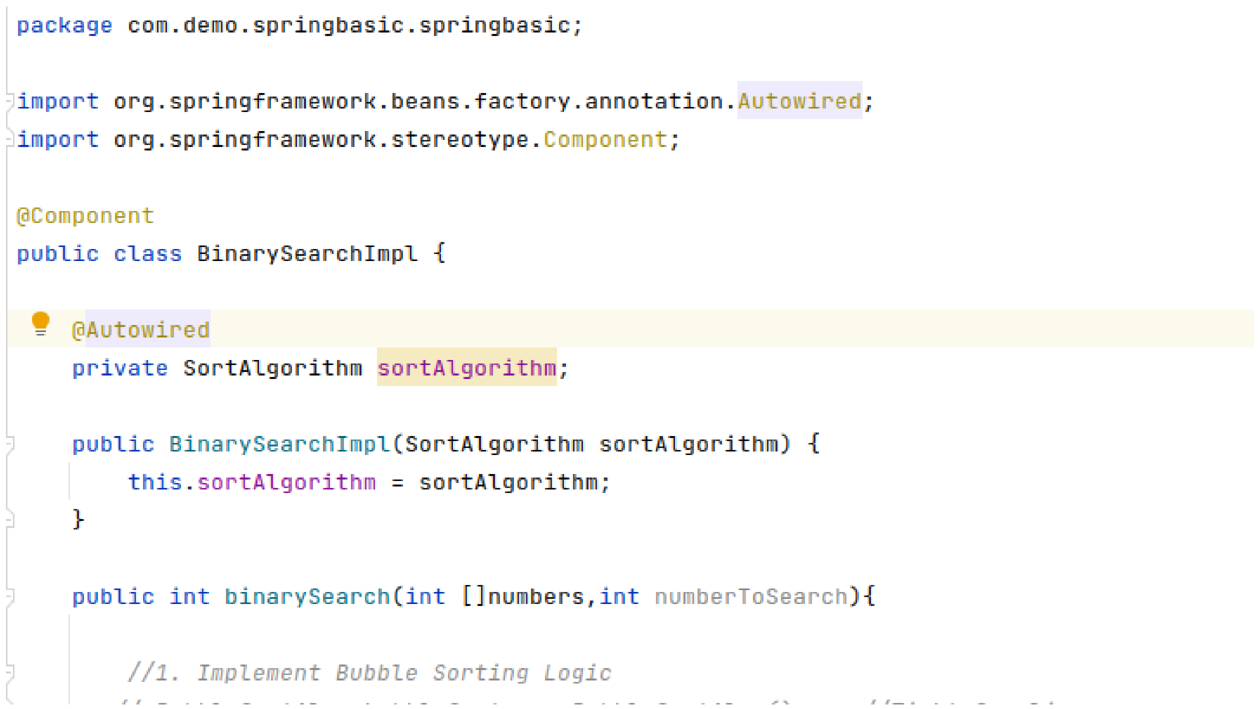
Where to search for Beans?

```
@SpringBootApplication
```

It checks for @Component annotation in the current package and its subpackages.

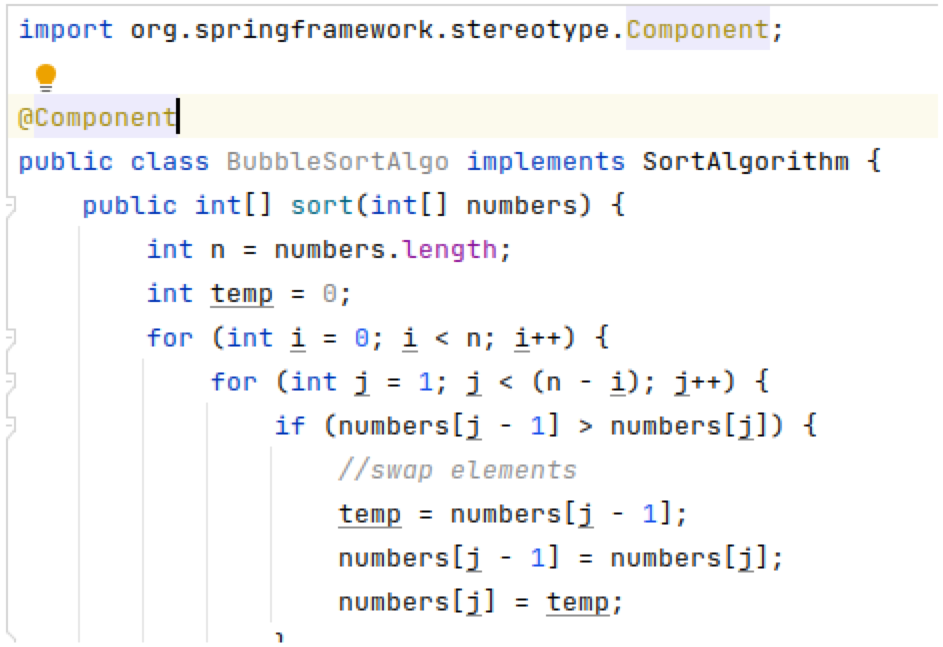
Spring does component scan in the specified package by spring boot which automatically scans the package and subpackages.

BinarySearchImpl.java



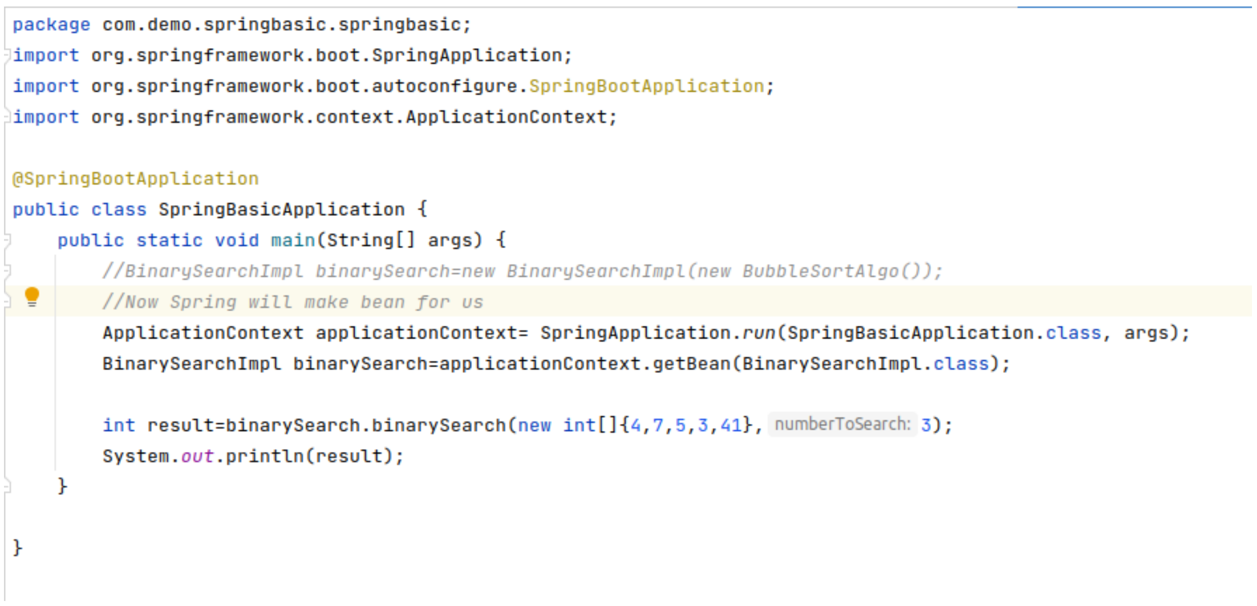
```java
package com.demo.springbasic.springbasic;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class BinarySearchImpl {

    @Autowired
    private SortAlgorithm sortAlgorithm;

    public BinarySearchImpl(SortAlgorithm sortAlgorithm) {
        this.sortAlgorithm = sortAlgorithm;
    }

    public int binarySearch(int []numbers,int numberToSearch){

        //1. Implement Bubble Sorting Logic
```

BubbleSortAlgo.java



```java
import org.springframework.stereotype.Component;

@Component
public class BubbleSortAlgo implements SortAlgorithm {
    public int[] sort(int[] numbers) {
        int n = numbers.length;
        int temp = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (numbers[j - 1] > numbers[j]) {
                    //swap elements
                    temp = numbers[j - 1];
                    numbers[j - 1] = numbers[j];
                    numbers[j] = temp;
                }
```

SpringBasicApplication.java



```
package com.demo.springbasic.springbasic;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class SpringBasicApplication {
    public static void main(String[] args) {
        //BinarySearchImpl binarySearch=new BinarySearchImpl(new BubbleSortAlgo());
        //Now Spring will make bean for us
        ApplicationContext applicationContext= SpringApplication.run(SpringBasicApplication.class, args);
        BinarySearchImpl binarySearch=applicationContext.getBean(BinarySearchImpl.class);

        int result=binarySearch.binarySearch(new int[]{4,7,5,3,41}, numberToSearch: 3);
        System.out.println(result);
    }

}
```

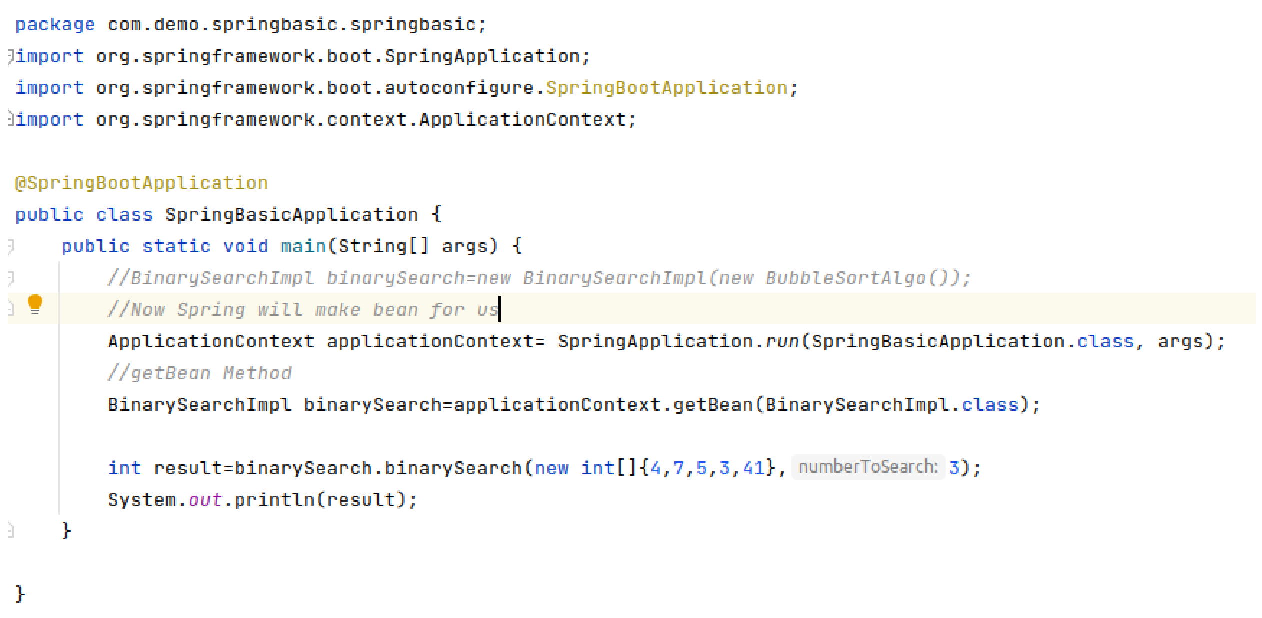
(4) Get a Spring Bean from the application context and display its properties.

Sol. All the beans are managed by **Spring Application Context.**

To get the beans from the application context we use the getBean method by passing the bean name. E.x

BinarySearchImpl binarySearch=applicationContext.**getBean**(BinarySearchImpl.class);

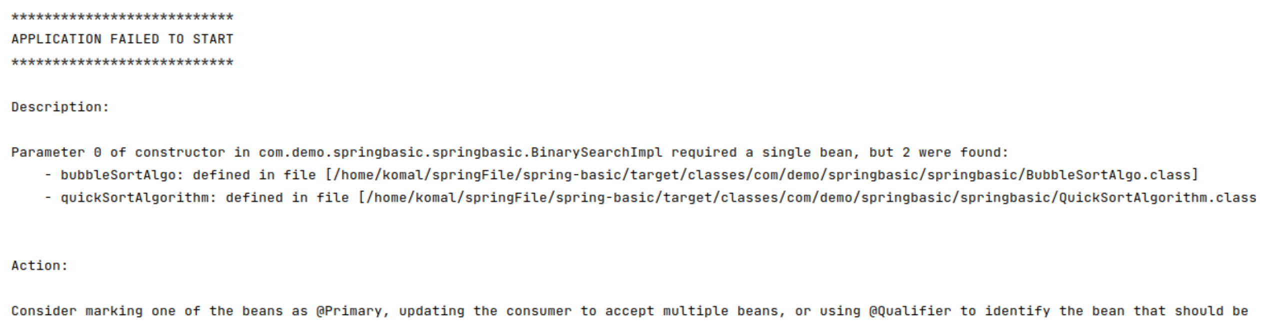
Now, we can use the BinarySearchImpl properties through the binarySearch object.



```java
package com.demo.springbasic.springbasic;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class SpringBasicApplication {
    public static void main(String[] args) {
        //BinarySearchImpl binarySearch=new BinarySearchImpl(new BubbleSortAlgo());
        //Now Spring will make bean for us
        ApplicationContext applicationContext= SpringApplication.run(SpringBasicApplication.class, args);
        //getBean Method
        BinarySearchImpl binarySearch=applicationContext.getBean(BinarySearchImpl.class);

        int result=binarySearch.binarySearch(new int[]{4,7,5,3,41}, numberToSearch: 3);
        System.out.println(result);
    }

}
```

(5) Demonstrate how you will resolve ambiguity while autowiring bean (Hint : @Primary).

Sol. SortingAlgo is a dependency of BinarySearchImpl. We have auto-wired BinarySearchImpl and BinarySearchAlgo through @auto-wired annotation. But let's say now we have to use QuickSortAlgorithm so we will make it a bean using @component annotation. But if we run our application it will lead to ambiguity.



```
***************************
APPLICATION FAILED TO START
***************************

Description:

Parameter 0 of constructor in com.demo.springbasic.springbasic.BinarySearchImpl required a single bean, but 2 were found:
    - bubbleSortAlgo: defined in file [/home/komal/springFile/spring-basic/target/classes/com/demo/springbasic/springbasic/BubbleSortAlgo.class]
    - quickSortAlgorithm: defined in file [/home/komal/springFile/spring-basic/target/classes/com/demo/springbasic/springbasic/QuickSortAlgorithm.class


Action:

Consider marking one of the beans as @Primary, updating the consumer to accept multiple beans, or using @Qualifier to identify the bean that should be
```

So, in order to remove the ambiguity we use **@Primary** annotation. It is used to give higher preference to a bean among multiple beans.

So, if we want to give preference to BubbleSortAlgo we will use the @Primary annotation.

```java
import org.springframework.stereotype.Component;

@Component
@Primary
public class BubbleSortAlgo implements SortAlgorithm {
    public int[] sort(int[] numbers) {
        int n = numbers.length;
        int temp = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (numbers[j - 1] > numbers[j]) {
                    //swap elements
                    temp = numbers[j - 1];
                    numbers[j - 1] = numbers[j];
                    numbers[j] = temp;
                }
            }
        }
        return numbers;
    }
}
```

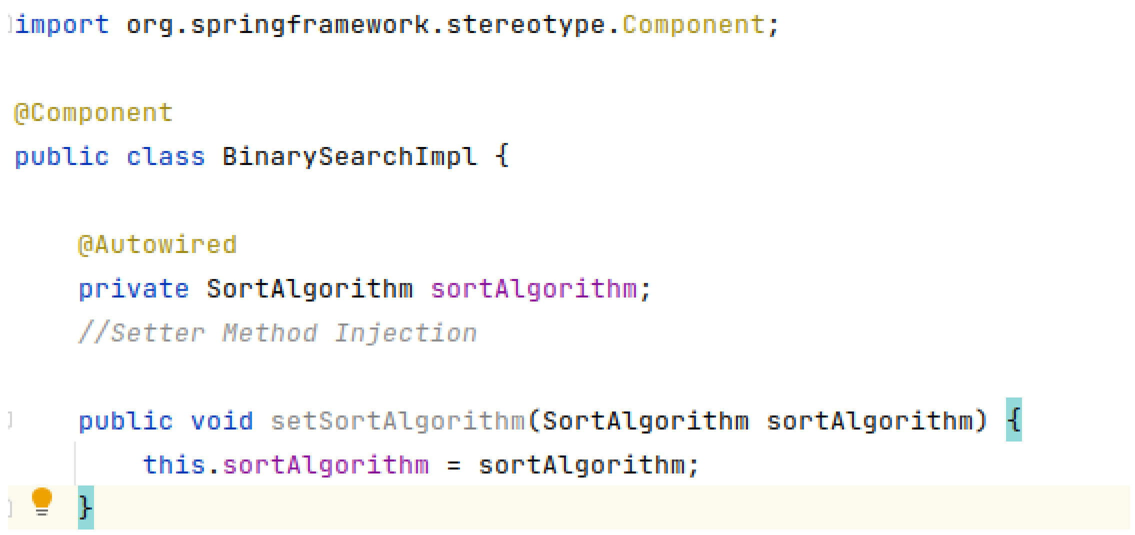
(6) Perform Constructor Injection in a Spring Bean.

Sol. Dependency Injection can be done through Constructor Injection and setter method.

Using Constructor Injection

```java
import org.springframework.stereotype.Component;

@Component
public class BinarySearchImpl {

    @Autowired
    private SortAlgorithm sortAlgorithm;

    //Constructor Injection
    public BinarySearchImpl(SortAlgorithm sortAlgorithm) {
        this.sortAlgorithm = sortAlgorithm;
    }
```

Using Setter method



```java
import org.springframework.stereotype.Component;

@Component
public class BinarySearchImpl {

    @Autowired
    private SortAlgorithm sortAlgorithm;
    //Setter Method Injection

    public void setSortAlgorithm(SortAlgorithm sortAlgorithm) {
        this.sortAlgorithm = sortAlgorithm;
    }
```