Session: Spring Framework Introduction

Assignment

1. Write a program to demonstrate Tightly Coupled code.

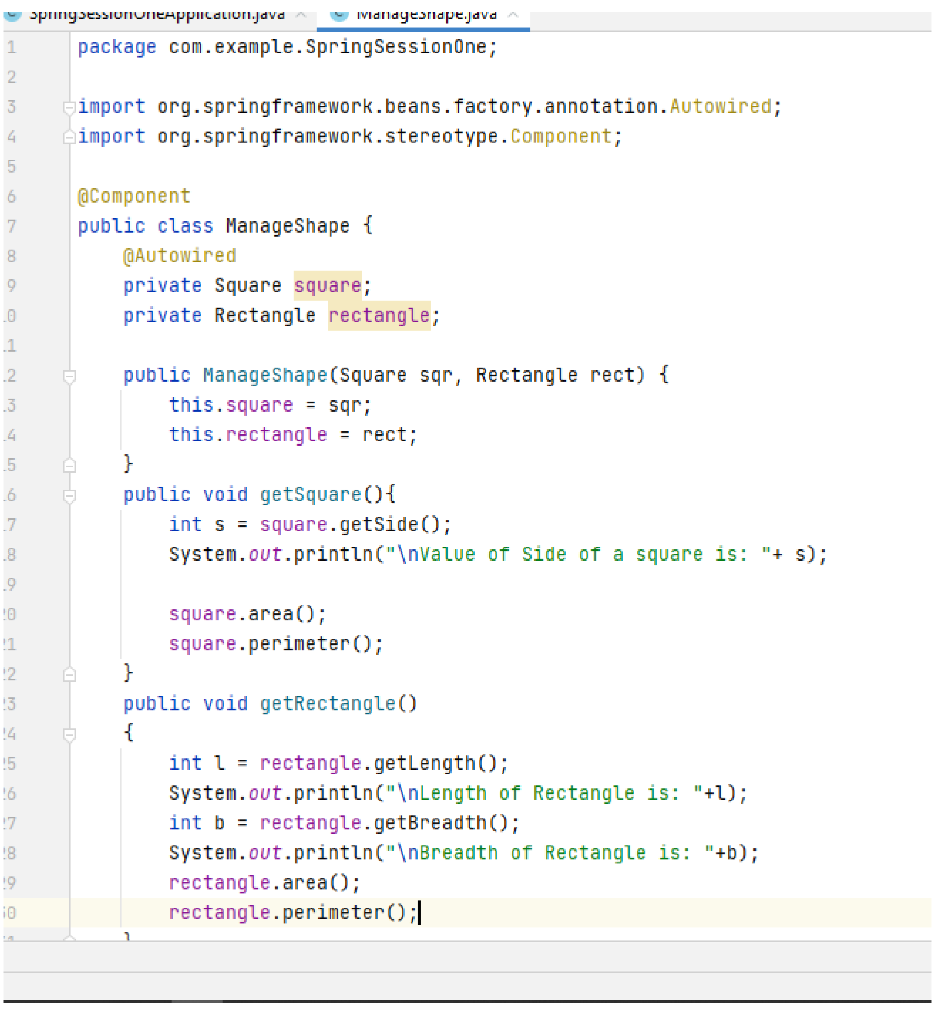
```java
interface Job {
    public void display();
}

class Doctor implements Job {

    @Override
    public void display() {
        System.out.println("\n This is Doctor");
    }
}

class Engineer implements Job {

    @Override
    public void display() {
        System.out.println("\nThis is Engineer");
    }

}

public class TightCoupling {

    Doctor doctor;
    Engineer engineer;

    public TightCoupling(Doctor doctor, Engineer engineer) {
        this.doctor = doctor;
        this.engineer = engineer;
    }
    public void display() {
        doctor.display();
        engineer.display();
    }

    public static void main(String[] args) {

        Engineer eng = new Engineer();
        Doctor doc = new Doctor();

        TightCoupling tightCoupling = new TightCoupling(doc,eng );
        tightCoupling.display();
    }
}
```

In the above code if we want to add a new job like lawyer for example, we first have to create a new class implementing the job interface and then we have to create a reference of the newly created class in TightCoupling, make changes in the constructor and in the display method as well. So the whole code is dependent to other things hence it is tightly coupled.

2. Write a program to demonstrate Loosely Coupled code.

```
interface Job1 {
    public void display();
}

class Lawyer implements Job1 {

    @Override
    public void display() {
        System.out.println("\nThis is Lawyer");
    }
}

class Scientist implements Job1 {

    @Override
    public void display() {
        System.out.println("\nThis is Scientist");
    }

}
public class LooseCoupling {

    Job1 job1;

    public LooseCoupling(Job1 job1) {
        this.job1 = job1;
    }
    public void display() {
        job1.display();
    }

    public static void main(String[] args) {

        Lawyer lawyer = new Lawyer();
        Scientist scientist = new Scientist();
        LooseCoupling looseCoupling = new LooseCoupling(lawyer);
        LooseCoupling looseCoupling1 = new
LooseCoupling(scientist);
        looseCoupling.display();
        looseCoupling1.display();

    }
}
```

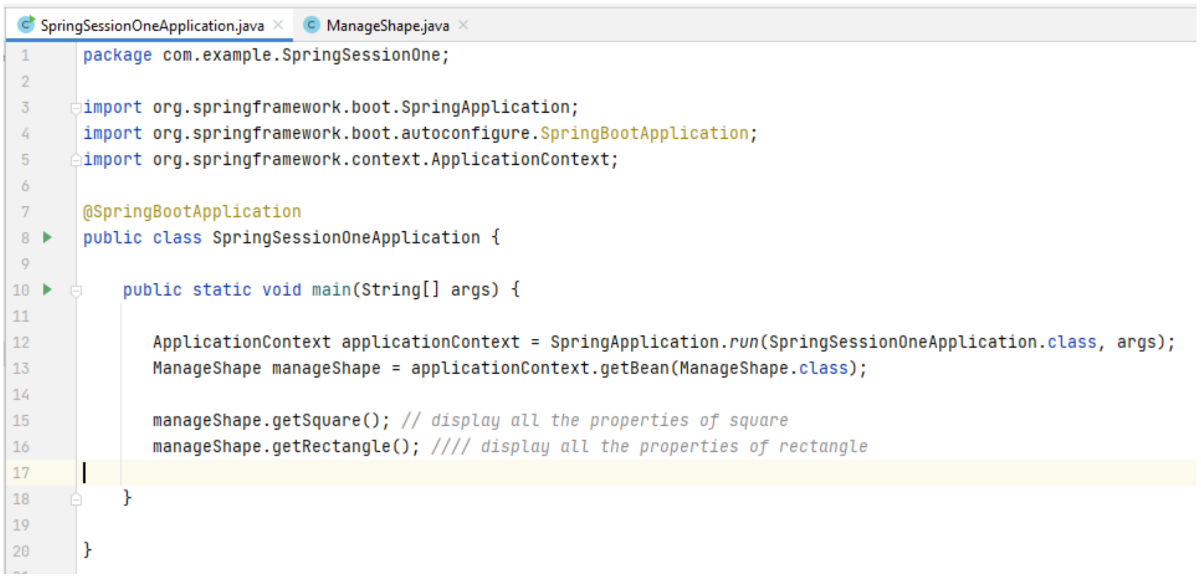
3. Use @Compenent and @Autowired annotations to in Loosely Coupled code for dependency management



SpringSessionOneApplication.java ManageShape.java

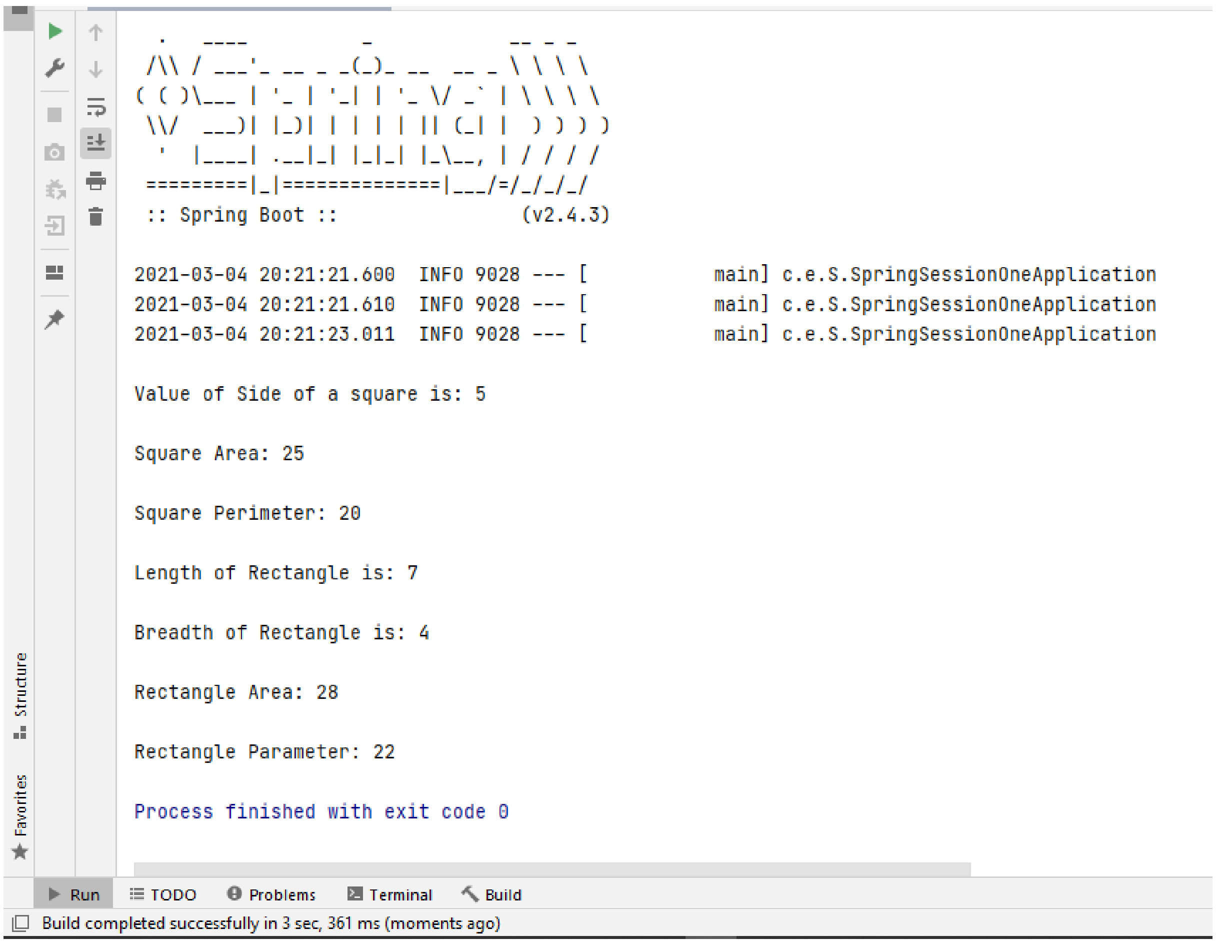
```java
package com.example.SpringSessionOne;

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

@Component
public class ManageShape {
    @Autowired
    private Square square;
    private Rectangle rectangle;

    public ManageShape(Square sqr, Rectangle rect) {
        this.square = sqr;
        this.rectangle = rect;
    }
    public void getSquare(){
        int s = square.getSide();
        System.out.println("\nValue of Side of a square is: "+ s);

        square.area();
        square.perimeter();
    }
    public void getRectangle()
    {
        int l = rectangle.getLength();
        System.out.println("\nLength of Rectangle is: "+l);
        int b = rectangle.getBreadth();
        System.out.println("\nBreadth of Rectangle is: "+b);
        rectangle.area();
        rectangle.perimeter();
```

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



```java
package com.example.SpringSessionOne;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class SpringSessionOneApplication {

    public static void main(String[] args) {

        ApplicationContext applicationContext = SpringApplication.run(SpringSessionOneApplication.class, args);
        ManageShape manageShape = applicationContext.getBean(ManageShape.class);

        manageShape.getSquare(); // display all the properties of square
        manageShape.getRectangle(); //// display all the properties of rectangle

    }

}
```

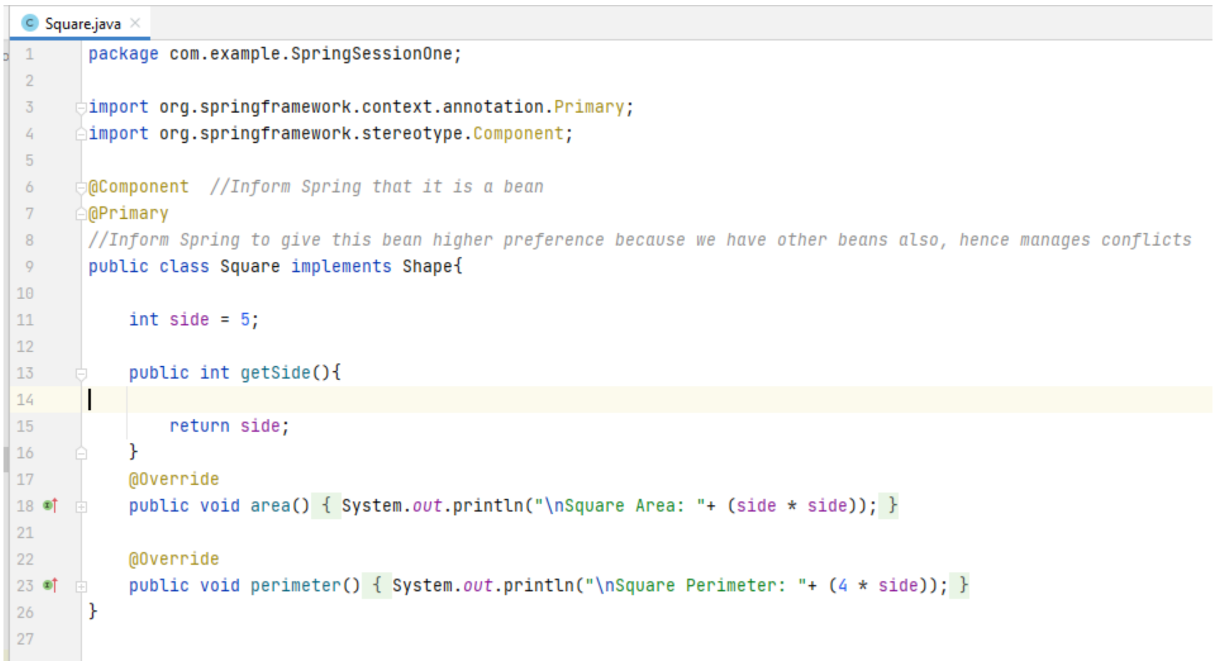
Output:



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.3)

2021-03-04 20:21:21.600  INFO 9028 --- [           main] c.e.S.SpringSessionOneApplication
2021-03-04 20:21:21.610  INFO 9028 --- [           main] c.e.S.SpringSessionOneApplication
2021-03-04 20:21:23.011  INFO 9028 --- [           main] c.e.S.SpringSessionOneApplication

Value of Side of a square is: 5

Square Area: 25

Square Perimeter: 20

Length of Rectangle is: 7

Breadth of Rectangle is: 4

Rectangle Area: 28

Rectangle Parameter: 22

Process finished with exit code 0
```

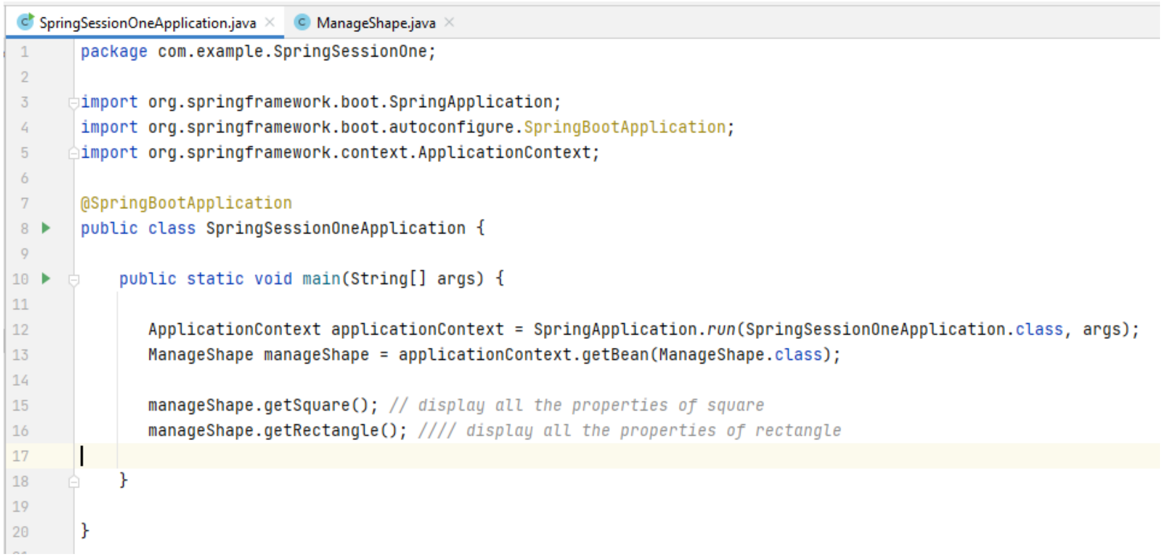
Build completed successfully in 3 sec, 361 ms (moments ago)

5. Demonstrate how you will resolve ambiguity while autowiring bean



```java
package com.example.SpringSessionOne;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;

@Component  //Inform Spring that it is a bean
@Primary
//Inform Spring to give this bean higher preference because we have other beans also, hence manages conflicts
public class Square implements Shape{

    int side = 5;

    public int getSide(){

        return side;
    }
    @Override
    public void area() { System.out.println("\nSquare Area: "+ (side * side)); }

    @Override
    public void perimeter() { System.out.println("\nSquare Perimeter: "+ (4 * side)); }
}

```

6. Perform Constructor Injection in a Spring Bean



```java
package com.example.SpringSessionOne;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class SpringSessionOneApplication {

    public static void main(String[] args) {

        ApplicationContext applicationContext = SpringApplication.run(SpringSessionOneApplication.class, args);
        ManageShape manageShape = applicationContext.getBean(ManageShape.class);

        manageShape.getSquare(); // display all the properties of square
        manageShape.getRectangle(); //// display all the properties of rectangle

    }

}
```

Note: All Java files are uploaded to github and the description is in the README.