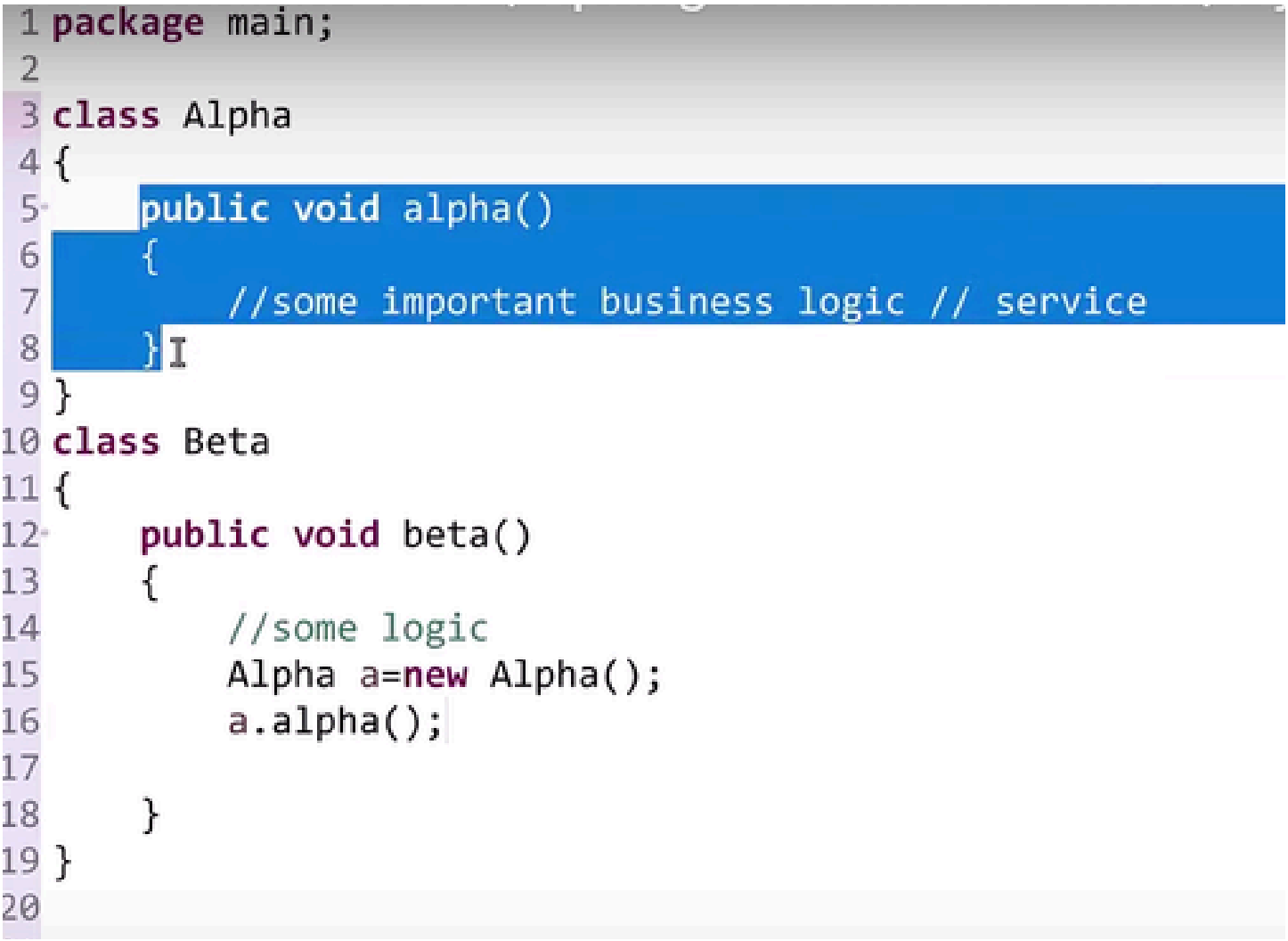
SPRING CORE:

1. Why Spring Framework?

```
1  package main;
2
3  class Alpha
4  {
5      public void alpha()
6      {
7          //some important business logic // service
8      }
9  }
10 class Beta
11 {
12     public void beta()
13     {
14         //some logic
15
16
17     }
18 }
19
```

- Suppose we have two classes, Alpha and Beta.
- Now, if we have to use Alpha class's method alpha() in the beta() method of Beta class.
- We can do this by creating object of Alpha class in beta() and then calling alpha method (consider below image).



```java
package main;

class Alpha
{
    public void alpha()
    {
        //some important business logic // service
    }
}
class Beta
{
    public void beta()
    {
        //some logic
        Alpha a=new Alpha();
        a.alpha();

    }
}
```

- We have achieved or task of using Alpha class service in Beta class service.
- We can either achieve this task by making class Beta to extend class Alpha (inheritance).
- But there is one problem → if something happens to our Alpha class then it will cause problem in beta class.
- Because Beta and Alpha class are Tightly coupled.
- If we want to achieve loose coupling then we have to do something which will help us to use Alpha class's

service in Beta class by not using object creation, inheritance and without using name of alpha class.
- Then we can say Alpha and Beta class our loosely coupled.

Note:

- Whenever you have to write some service logic which will be used by other classes then don’t write it in a class.
- Write it in a interface then use it in class.

How Can we achieve Loose Coupling?

- Understanding it with a project.

```
package services;

public class TShapedSkills
{

    public Boolean buyTheCourse(Double amount)
    {
        return true;
    }
}
```

- We will make more classes, whose code will be used in this TShapedSkills class without the name of those classes to achieve Loose Coupling.

```
package services;

public interface ICourse
{
        Boolean getTheCourse(Double price);
}
```

- We have created a Interface which will be implemented by other two classes (Java and SpringBoot).
- In ‘Java’ class, we have implemented the getTheCourse() and returned (“Java course Purchased of price” + price);
- In ‘SpringBoot’ class, we have implemented the getTheCourse() and returned (“SpringBoot course Purchased of price” + price);

- Now our task was to use service of Java and SpringBoot class into TShapedSkills without using their name.
- We can achieve this by using the Interface they are implementing.

```
package services;

public class TShapedSkills
{

    private ICourse course;



    public void setCourse(ICourse course)
    {
        this.course = course;
    }
          I


    public Boolean buyTheCourse(Double amount)
    {

        return course.getTheCourse(amount);

    }
}
```

- This setCourse() constructor will help us in setting what class's object to use.

```
public static void main(String[] args)
{


    TShapedSkills t=new TShapedSkills();
    //Java j=new Java();
    t.setCourse(new Java());
    Boolean status=t.buyTheCourse(4999.4);
    if(status)
        System.out.println("Sucess");
    else
        System.out.println("Failed to get the course");
```

- Here Java class's getTheCourse() method will be called from buyTheCourse() from TShapedSkills class.

```
27 public class LaunchApp
28 {
29
30     public static void main(String[] args)
31     {
32
33
34         TShapedSkills t=new TShapedSkills();
35         //Java j=new Java();
36         //t.setCourse(new Java());
37         t.setCourse(new SpringBoot());
38         Boolean status=t.buyTheCourse(4999.4);
39         if(status)
40             System.out.println("Sucess");
41         else
42             System.out.println("Failed to get the course");
43
```

- Here SpringBoot class's getTheCourse() method will be called from buyTheCourse() from TShapedSkills

class.

- We can see that control is shifted towards main method (it is responsible for creating objects of java and SpringBoot class).

Target class and Dependent class:

- Target class is that class where services of other class is used (TShapedSkills).
- Such Objects whose services will be used in Target class are called Dependent objects (Java and SpringBoot).
- Injecting dependent object in target class is called Dependency Injection.

```
TShapedSkills t=new TShapedSkills();
//Java j=new Java();
//t.setCourse(new Java());
t.setCourse(new SpringBoot());
```

- Inserting the dependent object into the target class using setter is called Setter Injection.
- As we have seen that we have made a setter setCourse() in TShapedSkills.
- Inserting the dependent object into the target class using constructor is called Constructor Injection.

Till now we have achieved loose coupling between TShapedSkills, Java and SpringBoot classs.

But still TShapedSkills and main class are tightly coupled (what if something happens to TShapedSkills class, we are manually implementing dependency injection).

Spring framework helps us in achieving loose coupling.