Spring Exercise

Spring

Spring Framework is an integration framework that integrates many different Java open source projects into a common programming model.

It builds on Dependency Injection. This is an exercise about Dependency Injection and the use of Interfaces in Dependency Injection.

1 – Create project

Create a new normal Java project and add a **Main** class with a **public static void main** method.

Also, create these two classes:

```
class Bank
{
    public void deposit(int amount) {

        Logger log = new Logger();
        log.logMessage("Deposited " + amount + " Sek");

        System.out.println("Deposited " + amount + " Sek");
    }
}

class Logger
{
    public void logMessage(String message)
    {
        System.out.println("Logged:" + message);
    }
}
```

Now you can create a new **Bank** object in the **main** method and call the **deposit** method on the bank object and for example deposit 100 Sek, like this:

```
Bank bank = new Bank();
bank.deposit(100);
```

Try this out and see that it works.

It should work but we have a problem here with coupling. Testing will be hard and it will be hard to replace the Logger with another Logger implementation since it is created with a **new** keyword inside the **deposit** method.

The use of the **new** keyword inside methods could be replaced with Dependency Intection

2 – Add more loggers

What if we sometimes want to log with another logger implementation, like a DBLogger?

Create this class to the project:

```
class DBLogger
{
    public void logMessage(String message)
    {
        System.out.println("DBLogged:" + message);
    }
}
```

We could use a flag for indicating which logger implementation we want to use by sending in a Boolean flag like this:

```
class Bank
{
    public void deposit(int amount, boolean logToDB) {

        if(logToDB) {
            DBLogger log = new DBLogger();
            log.logMessage("Deposited " + amount + " Sek");
        }
        else
        {
            Logger log = new Logger();
            log.logMessage("Deposited " + amount + " Sek");
        }

        System.out.println("Deposited " + amount + " Sek");
    }
}
```

But what if we sometimes want to log to XML? Now we have a third alternative like this:

```
class XMLLogger
{
  public void logMessage(String message)
  {
    System.out.println("XML Logged:" + message);
  }
}
```

Using a flag just won't work!

3 – Dependency Injection

Let's add a constructor instead. Change the Bank class like this:

```
class Bank
{
    private Logger logger;
    private DBLogger dblogger;

    public Bank(Logger logger, DBLogger dblogger) {
        this.logger = logger;
        this.dblogger = dblogger;
    }

    public void deposit(int amount) {

        if(logger != null) {
            logger.logMessage("Deposited " + amount + " Sek");
        }

        if(dblogger != null) {
            dblogger.logMessage("Deposited " + amount + " Sek");
        }

        System.out.println("Deposited " + amount + " Sek");
    }
}
```

By injecting the dependency to the correct logger implementation in the constructor, the correct logger implementation will be used in the **deposit** method and we no longer need a flag.

But we still need to change the constructor to be able to use the XMLLogger. We can build a better solution using an Interface and dependency inject the Interface instead. Let's try this out.

4 – Dependency Injection with an Interface

Create an Interface like this:

```
interface ILogger {
   void logMessage(String message);
}
```

Change all loggers to implement this Interface.

Now you can dependency inject the Interface instead, and then you can add as many new logger implementations as you want without needing to change the constructor.

Try this version of the **Bank** class, and create it in the **main** method and send any of the loggers in to the constructor method. Then call the **deposit** method, and the correct logger that you sent in the constructor will be used in the **deposit** method!

```
class Bank
{
   private ILogger logger;

   public Bank(ILogger logger) {
      this.logger = logger;
   }

   public void deposit(int amount) {

      logger.logMessage("Deposited " + amount + " Sek");

      System.out.println("Deposited " + amount + " Sek");
   }
}
```

Now the bank's dependencies are injected, it is easy to test and flexible!

We can even support JSONLogging in the future!

Stretch Tasks (if you have time)

Read about Spring and Dependency Injection here:

https://www.tutorialspoint.com/spring/spring_dependency_injection.htm

General information about Dependency Injection:

https://en.wikipedia.org/wiki/Dependency_injection