Projekt p21

```
public class AppMain {
    public static void main(String[] args) {
        DataProcessor processor = new DataProcessor();

        boolean cont=true;
        while(cont) {
            cont = processor.processData();
        }
    }
}

public class DataProcessor {
    private DataSource source;

    public DataProcessor() {
        source = new DataSource();
    }

    public boolean processData() {
        String s = source.getData();
        System.out.println(s);
        return s!=null;
    }
}

public class DataSource {
    public String getData() {
        return "asos hello";
    }
}
```

Verzia s interfejsom umoznuje pouzit rozne implementacie zdroja dat.

```
public class DataProcessor {
    private DataSourceIfc source;

    public DataProcessor() {
//          source = new DataSource();
        source = new DataSourceMock();
    }

    public boolean processData() {
        String s = source.getData();
        System.out.println(s);
        return s!=null;
    }
}


public interface DataSourceIfc {
    public String getData();
}

public class DataSourceMock implements DataSourceIfc{
    public String getData() {
        return null;
    }
}
```

Stale vsak treba menit kod (aj ked len jediny riadok).
Riesenie: pre definovanie komponent, ktoré tvoria aplikáciu použiť **konfiguračný súbor.**
myConfig.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
">
    <bean id="mysource" class="asos.DataSourceMock"/>
</beans>

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

public class DataProcessor {
    private DataSourceIfc source;

    public DataProcessor() {
//          source = new DataSourceMock();
        ApplicationContext context =
            new ClassPathXmlApplicationContext(new String[]{"myConfig.xml"});
        source = context.getBean("mysource", DataSourceIfc.class);
    }

    public boolean processData() {
        String s = source.getData();
        System.out.println(s);
        return s!=null;
    }
}
```

Konstruktor DataProcessora si potrebuje nacitat konfiguraciu aby sa dostal k referencii na zdroj dat.
Ten isty zdroj dat by vsak mohli potrebovat aj ine objekty, preto by bolo vhodne **vytiahnut incializaciu kontextu a vytvorenie zdroja dat do hlavneho programu.**

Referenciu na source mozno zadat processoru dvom sposobmi
- do konstruktoru

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

        ApplicationContext context =
            new ClassPathXmlApplicationContext(new String[]{"myConfig.xml"});
        DataSourceIfc source = context.getBean("mysource", DataSourceIfc.class);

        DataProcessor processor = new DataProcessor(source);

        boolean cont=true;
        while(cont) {
            cont = processor.processData();
        }
    }
}
```

processor musi mat konstruktor s parametrom source

```
public class DataProcessor {
    private DataSourceIfc source;

    public DataProcessor(DataSourceIfc source) {
        this.source = source;
    }

    public boolean processData() {
        String s = source.getData();
        System.out.println(s);
        return s!=null;
    }
}
```

- alebo setterom:

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

        ApplicationContext context =
            new ClassPathXmlApplicationContext(new String[]{"myConfig.xml"});
        DataSourceIfc source = context.getBean("mysource", DataSourceIfc.class);

        DataProcessor processor = new DataProcessor();
        processor.setSource(source);

        boolean cont=true;
        while(cont) {
            cont = processor.processData();
        }
    }
}
```

processor musi mat setter pre source

```
public class DataProcessor {

    private DataSourceIfc source;

    public DataProcessor() {
    }

    public void setSource(DataSourceIfc source) {
        this.source = source;
    }

    public boolean processData() {
        String s = source.getData();
        System.out.println(s);
        return s!=null;
    }
}
```

Aj processor moze vytvorit a spravovat IoC kontainer

```
public class AppMain {
    public static void main(String[] args) {
        ApplicationContext context
            = new ClassPathXmlApplicationContext(new String[]{"myConfig.xml"});
        DataSourceIfc source = context.getBean("mysource", DataSourceIfc.class);

//        DataProcessor processor = new DataProcessor();
        DataProcessor processor =
          context.getBean("myprocessor",DataProcessor.class);
        processor.setSource(source);

        boolean cont=true;
        while(cont) {
            cont = processor.processData();
        }
        System.out.println("hotovo");
   }
```

myConfig.xml

```
    <bean id="mysource" class="asos.DataSourceMock"/>
    <bean id="myprocessor" class="asos.DataProcessor"/>
```

Dependency Injection - DI

Pozri Projekt p23.
IoC kontainer moze inicializovat aj referenciu medzi komponetami.
Teraz už referenciu na source ani volanie processor.setSource(source) uz v maine nepotrebujeme.

```
public class AppMain {
    public static void main(String[] args) {
        ApplicationContext context
                = new ClassPathXmlApplicationContext(new String[]{"myConfig.xml"});
        DataProcessor processor = context.getBean("myprocessor", DataProcessor.class);
//        DataSourceIfc source = context.getBean("mysource", DataSourceIfc.class);
//        processor.setSource(source);

        boolean cont=true;
        while(cont) {
            cont = processor.processData();
        }
        System.out.println("hotovo");
   }
}
```

Konfiguráciu IoC-kontainera musíme upraviť podľa toho, či komponenta processor používa pre inicializáciu referencie na source setter metódu alebo konštruktor.
Setter based DI

```
    <bean id="mysource" class="asos.DataSourceMock"/>
    <bean id="myprocessor" class="asos.DataProcessor">
        <property  name="source" ref="mysource"/>
    </bean>
```

Constructor based DI:

```
    <bean id="mysource" class="asos.DataSourceMock"/>
    <bean id="myprocessor" class="asos.DataProcessor">
<!--        <property name="source" ref="mysource"/>-->
        <constructor-arg ref="mysource"/>
    </bean>
```

V oboch prípadoch sme kontaineru expicitne povedali, ktorú kompomentu má injektovať. Vyhladanie vhodnej komponenty pre injektovanie (pokiaľ je taká len jedna) môžeme prenechať aj kontaineru: **Autowire**

Pre **Constructor based DI** použijeme **autowire="constructor"**

```
    <bean id="mysource" class="asos.DataSourceMock"/>
    <bean id="myprocessor" class="asos.DataProcessor" autowire="constructor"/>
```

Setter based DI môžeme použiť
buď **byType – kontainer najde vhodnu komponentu podla typu argumentu settra**

```
    <bean id="mysource" class="asos.DataSourceMock"/>
    <bean id="myprocessor" class="asos.DataProcessor" autowire="byType"/>

public class DataProcessor {
    private DataSourceIfc source;
    public void setSource(DataSourceIfc source) {
        this.source = source;
    }
```

alebo **byName - kontainer najde vhodnu komponentu podla mena datoveho clena**

```
    <bean id="source" class="asos.DataSourceMock"/>
    <bean id="myprocessor" class="asos.DataProcessor" autowire="byName"/>

public class DataProcessor {
    private DataSourceIfc source;
```