Required dependency checking

For a large project where many developers work in parallel, it's desirable to have some framework level facilities to check that all mandatory dependencies have been set or not. This check should ideally be performed at compile time, if not possible then as early as start up time, before we start having NullPointerException on missing values. Spring offers various dependency checking mechanism during start up time.

Let's explore some aspects and options provided by Spring framework regarding dependency checking.

Constructor vs setter injection

As we discussed in the tutorial, different ways to do dependency injection, we should always use constructors based injection for the mandatory properties (with programmatic validation of arguments) and setter based injection for optional properties.

There might still be many reasons we want to use setters for mandatory properties rather than constructor. May be our constructor is getting too complex, may be we want to reconfigure some properties later (of course they cannot be final in that case but still are mandatory at wiring time).

Dependency injection via constructors

Spring throws UnsatisfiedDependencyException if the required dependency is missing for the target constructor

```
@Configuration
@ComponentScan(useDefaultFilters = false,
        includeFilters = @ComponentScan.Filter(type = FilterType.REGEX, pattern =
"ConstructorDiExample"))
public class ConstructorDiExample {


    public static void main(String[] args) {
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext(ConstructorDiExample.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }


    @Component
    private static class ClientBean {
        private final ServiceBean serviceBean;


        private ClientBean(ServiceBean serviceBean) {
            this.serviceBean = serviceBean;
        }


        void doSomething() {
            System.out.println("doing something with: " + serviceBean);
        }
```

```
    }


    //uncomment @service to get rid of UnsatisfiedDependencyException
   // @Service
    private class ServiceBean {


    }
}
```

Output

```
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'com.piseth.java.school.example.ConstructorDiExample$ClientBean': Unsatisfied dependency expressed through constructor parameter 0: No qualifying bean of type [com.piseth.java.school.example.ConstructorDiExample$ServiceBean] found for dependency [com.piseth.java.school.example.ConstructorDiExample$ServiceBean]: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {}; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.piseth.java.school.example.ConstructorDiExample$ServiceBean] found for dependency [com.piseth.java.school.example.ConstructorDiExample$ServiceBean]: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {}
```

How to check required properties in setter injection?

If we are doing setter injection for required dependencies then we have following options for dependency checking:

Setter injection with @Autowire

If we are using default method of autowiring i.e. Autowire.NO then we need to explicitly use @Autowire at injection point (except for constructor since Spring 4.3). The annotation @Autowire#required attribute is true by default. So Spring throws UnsatisfiedDependencyException if the dependency via setter is not provided.

```
@Configuration
@ComponentScan(useDefaultFilters = false,
    includeFilters = @ComponentScan.Filter(type = FilterType.REGEX, pattern =
"SetterAutowiredExample"))
public class SetterAutowiredExample {

  public static void main(String... strings) {
    AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(
        SetterAutowiredExample.class);
    ClientBean bean = context.getBean(ClientBean.class);
    bean.doSomething();
  }

  @Component
  public static class ClientBean {
    private ServiceBean serviceBean;

    @Autowired
```

```
        public void setServiceBean(ServiceBean serviceBean) {
            this.serviceBean = serviceBean;
        }

        public void doSomething() {
            System.out.println("doing something with : " + serviceBean);
        }
    }

    //uncomment following to  get rid of UnsatisfiedDependencyException
    //@Service
    public static class ServiceBean {
    }
}
```

Output

```
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'com.piseth.java.school.example.SetterAutowiredExample$ClientBean': Unsatisfied dependency expressed through method 'setServiceBean' parameter 0: No qualifying bean of type [com.piseth.java.school.example.SetterAutowiredExample$ServiceBean] found for dependency [com.piseth.java.school.example.SetterAutowiredExample$ServiceBean]: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {}; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.piseth.java.school.example.SetterAutowiredExample$ServiceBean] found for dependency [com.piseth.java.school.example.SetterAutowiredExample$ServiceBean]:
```

expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {}

Setter injection without @Autowire but with @PostConstruct validation

Using default autowiring mode Autowiring.NO:

```
@Configuration
public class PostConstructExample {

    @Bean
    public ClientBean clientBean (ServiceBean serviceBean) {
        ClientBean clientBean = new ClientBean();
        //uncomment following to get rid of IllegalArgumentException
        //clientBean.setServiceBean(serviceBean);
        return clientBean;
    }

    @Bean
    public ServiceBean serviceBean () {
        return new ServiceBean();
    }

    public static void main (String... strings) {
        AnnotationConfigApplicationContext context =
                        new AnnotationConfigApplicationContext(
```

```
                                        PostConstructExample.class);
        ClientBean bean = context.getBean(ClientBean.class);
        bean.doSomething();
    }

    public static class ClientBean {
        private ServiceBean serviceBean;

        @PostConstruct
        public void myPostConstructMethod () throws Exception {
            // we can do more validation than just checking null values here
            if (serviceBean == null) {
                throw new IllegalArgumentException("ServiceBean not set");
            }
        }

        public void setServiceBean (ServiceBean serviceBean) {
            this.serviceBean = serviceBean;
        }

        public void doSomething () {
            System.out.println("doing something with: " + serviceBean);
        }
```

```
    }


    public static class ServiceBean {

    }

}
```

Output

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'clientBean': Invocation of init method failed; nested exception is java.lang.IllegalArgumentException: ServiceBean not set
```

Using Autowire.BY_TYPE mode:

```
@Configuration

public class PostConstructAutoWiringByType {


    @Bean(autowire = Autowire.BY_TYPE)

    public ClientBean clientBean() {

        return new ClientBean();

    }


    //remove following comment to get rid of java.lang.IllegalArgumentException

    //@Bean

    public ServiceBean serviceBean() {

        return new ServiceBean();

    }
```

```
public static void main(String... strings) {
    AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(
                    PostConstructAutoWiringByType.class);
    ClientBean bean = context.getBean(ClientBean.class);
    bean.doSomething();
}

public static class ClientBean {
    private ServiceBean serviceBean;

    public void setServiceBean(ServiceBean serviceBean) {
        this.serviceBean = serviceBean;
    }

    public void doSomething() {
        System.out.println("doing something with: " + serviceBean);
    }

    @PostConstruct
    public void myPostConstructMethod() throws Exception {
        // we can do more validation  than just checking null values here
```

```
            if (serviceBean == null) {
                throw new IllegalArgumentException("ServiceBean not set");
            }
        }
    }

    public static class ServiceBean {
    }
}
```

Output

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'clientBean': Invocation of init method failed; nested exception is java.lang.IllegalArgumentException: ServiceBean not set
```