Lazy Initialization, using @Lazy

By default Spring container instantiates all configured beans at startup (eager loading). In some situations, however, beans might rarely be used during application life cycle. Loading them at startup will not be a good idea if they are going to use considerable resources/memory to get initialized. In those situations we may decide to initialize such beans only when they are first accessed by application code (i.e. initialize on demand). We can achieve that by using @Lazy annotation on bean configuration method.

Example

An example bean which is used rarely

```
public class RarelyUsedBean {

  @PostConstruct
  private void initialize() {
      System.out.println("RarelyUsedBean initializing");
  }

  public void doSomething() {
      System.out.println("RarelyUsedBean method being called");
  }
}
```

An example bean which is used frequently

```
public class AlwaysBeingUsedBean {

  @PostConstruct
  public void init() {
      System.out.println("AlwaysBeingUsedBean initializing");
  }
}
```

Using @Lazy annotation and running main class

```
@Configuration
public class AppConfig {

  @Bean
  public AlwaysBeingUsedBean alwaysBeingUsedBean() {
      return new AlwaysBeingUsedBean();
  }

  @Bean
  @Lazy
  public RarelyUsedBean rarelyUsedBean() {
      return new RarelyUsedBean();
```

```
  }

  public static void main(String... strings) {
      AnnotationConfigApplicationContext context =
              new AnnotationConfigApplicationContext(AppConfig.class);
      System.out.println("Spring container started and is ready");
      RarelyUsedBean bean = context.getBean(RarelyUsedBean.class);
      bean.doSomething();
  }
}
```

Output

```
AlwaysBeingUsedBean initializing
Spring container started and is ready
RarelyUsedBean initializing
RarelyUsedBean method being called
```

2 - Using @Lazy at Injection points

Starting Spring 4.0.0, @Lazy can be used at injection points as well. That means we can place @Lazy along with @Autowired or @Inject or @Resource. That will delay the target bean initialization until it is used by the injection point class, even though the injection point class has been initialized earlier.

Example

An eagerly initialized bean

Following is MyEagerBean which has an @Autowired annotation (an injection point). This bean will eagerly initialized and that will cause to initialize all it's dependencies as well, unless we use @Lazy at a given injection point:

```
public class MyEagerBean {

  @Autowired
  @Lazy
  private MyLazyBean myLazyBean;

  @PostConstruct
  public void init () {
      System.out.println(getClass().getSimpleName() + " has been initialized");
  }

  public void doSomethingWithLazyBean () {
      System.out.println("Using lazy bean");
      myLazyBean.doSomething();
  }
}
```

The lazily initialized bean

```
public class MyLazyBean {
```

```
  @PostConstruct
  public void init () {
      System.out.println(getClass().getSimpleName() + " has been initialized");
  }

  public void doSomething () {
      System.out.println("inside lazy bean doSomething()");
  }
}
```

Main class

```
public class LazyExampleMain {

  public static void main (String[] args) {
      ApplicationContext context =
              new AnnotationConfigApplicationContext(
                      MyConfig.class);
      System.out.println("--- container initialized ---");
      MyEagerBean bean = context.getBean(MyEagerBean.class);
      System.out.println("MyEagerBean retrieved from bean factory");
      bean.doSomethingWithLazyBean();
  }
```

```
}
```

Output

```
MyEagerBean has been initialized
--- container initialized ---
MyEagerBean retrieved from bean factory
Using lazy bean
MyLazyBean has been initialized
inside lazy bean doSomething()
```

Now if we comment out @Lazy from MyEagerBean:

```
public class MyEagerBean {

  @Autowired
  //@Lazy
  private MyLazyBean myLazyBean;
  .....
}
```

Output:

```
MyLazyBean has been initialized
MyEagerBean has been initialized
--- container initialized ---
```

MyEagerBean retrieved from bean factory

Using lazy bean

inside lazy bean doSomething()

When MyEagerBean is loaded, it will cause to initialize all it's dependencies unless we use @Lazy at injection point too. The usage of @Lazy in factory class MyConfig prevents loading bean during 'startup' until it's used first time in the application.

In short:

@Lazy with @Bean (or @Lazy with @Component): don't load eagerly during application start up, until it's used in the application

@Lazy with @Autowired : don't load during outer class initialization, until it's first used by the application.

How that works?

Spring creates and uses a lazy-resolution proxy object for the injection point (the one annotated with @Autowired and @Lazy) instead of initializing the object directly.