Here are 3 ways to do it, just peek the best one for your case:

Using @Autowired constructors

**Better when**: you have all you need to build your prototype bean in the context (even for properties such as @Value("${prop}"))

If you want an automatic way to do it, you will need to have everything that is needed to instantiate the bean in the context too (even for a prototype bean). In case you have everything needed in your context you could simply annotate the constructor as @Autowired and Spring will do the rest for your.

@Component

@Scope("prototype")

public class FooBar {

private Baz baz;

@Autowired

public FooBar(Baz baz) {

this.baz = baz;

}

}

Using FactoryBeans

**Better when**: if you are using an XML based context, you will prefer this way.

Another possibility, if you need a personalized way to do it, would be using FactoryBeans. From [documentation](http://docs.spring.io/spring-framework/docs/4.0.6.RELEASE/javadoc-api/org/springframework/beans/factory/FactoryBean.html):

Interface to be implemented by objects used within a BeanFactory which are themselves factories. If a bean implements this interface, it is used as a factory for an object to expose, not directly as a bean instance that will be exposed itself.

The FactoryBean is used by Spring just to build the object you requested (being it a prototype or singleton).

For your case you could have an implementation like:

@Component

public class FooBarFactory implements FactoryBean<FooBar> {

@Autowired

private Baz myContextProvidedObject;

@Override

public FooBar getObject() throws Exception {

return new FooBar(myContextProvidedObject, "my parameter");

}

@Override

public Class<?> getObjectType() {

return FooBar.class;

}

@Override

public boolean isSingleton() {

return false;

}

}

And you could simple @Autowired FooBar on other instances of your context.

Using @Configuration

**Better when**: if you already have your context configured using annotations you will definitely prefer this way.

A third way to do it, and this is my favorite, would be using your @Configuration class. From [documentation](http://docs.spring.io/spring-framework/docs/4.0.6.RELEASE/javadoc-api/org/springframework/context/annotation/Configuration.html):

public @interface Configuration: Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime, for example:

Within that class your could have a method like:

@Configuration

public class MyConfig {

@Bean

@Scope("prototype")

public FooBar fooBar(Baz myContextProvidedObject) {

return new FooBar(myContextProvidedObject, "my parameter");

}

}

<https://stackoverflow.com/questions/25232034/spring-how-to-use-non-default-constructor-when-auto-wiring-a-prototype-bean-usi>