Spring

A Java application framework

Oliver Drotbohm /odrotbohm odrotbohm@pivotal.io

Script

https://github.com/odrotbohm/lectures

http://static.olivergierke.de/lectures/spring

Guestbook Sample

https://github.com/st-tu-dresden/guestbook

Goals of Frameworks

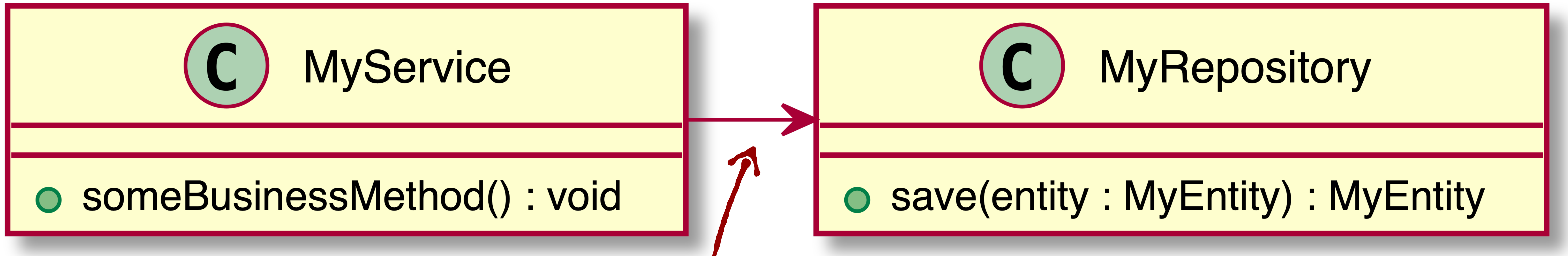
1. *Separation of concerns*
2. *Raising the abstraction level*
3. *Removal of boilerplate code*

Spring Framework

1. *Dependency Injection*
2. *Portable service abstraction*

Dependency Injection

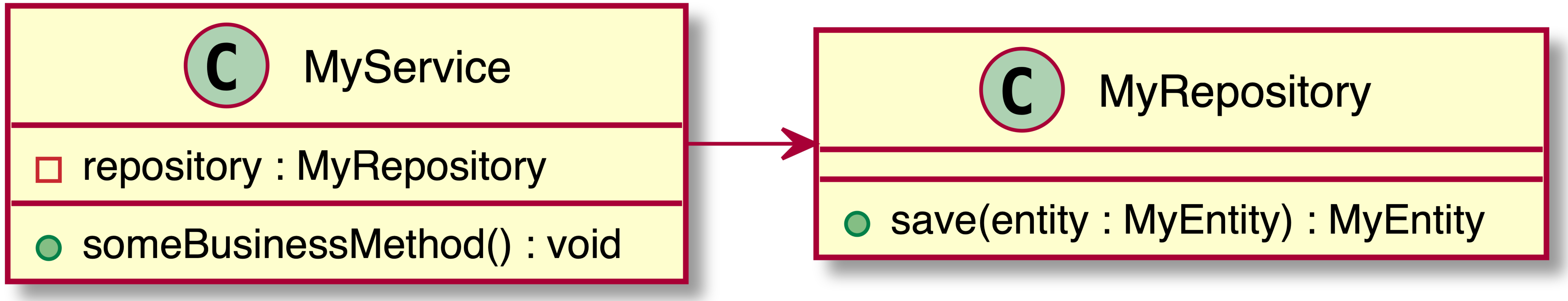
About constructing a net of objects...


MyService
someBusinessMethod() : void
MyRepository
save(entity : MyEntity) : MyEntity

```
class MyService {

  void someBusinessMethod() {
    // MyRepository.save(…) ?
  }
}
```

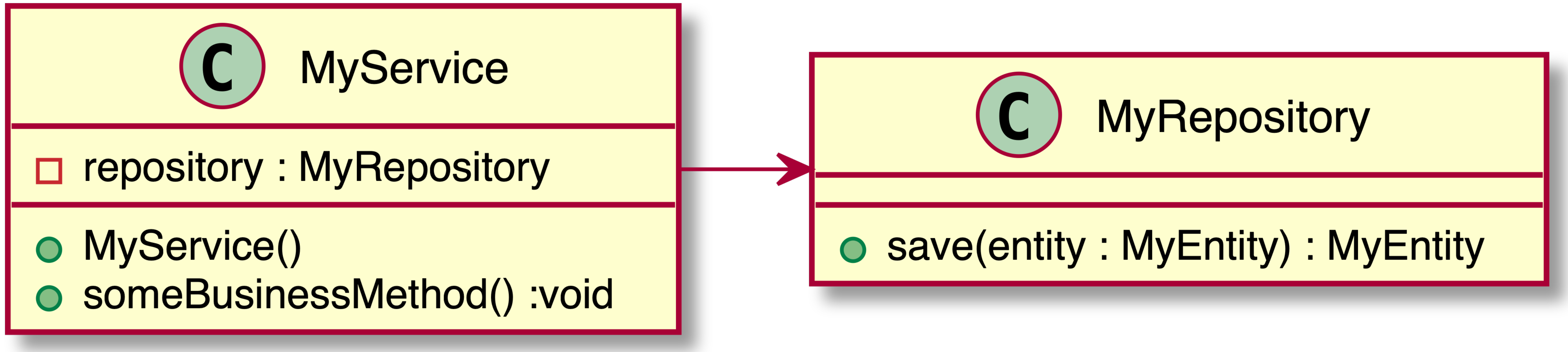
```
class MyRepository {

  MyEntity save(MyEntity entity) {
    …
  }
}
```

How do we establish this relationship?


C
MyService
repository : MyRepository
someBusinessMethod() : void
C
MyRepository
save(entity : MyEntity) : MyEntity

```
class MyService {

  private final MyRepository repo;

  void someBusinessMethod() {
    repo.save(new MyEntity());
  }
}
```

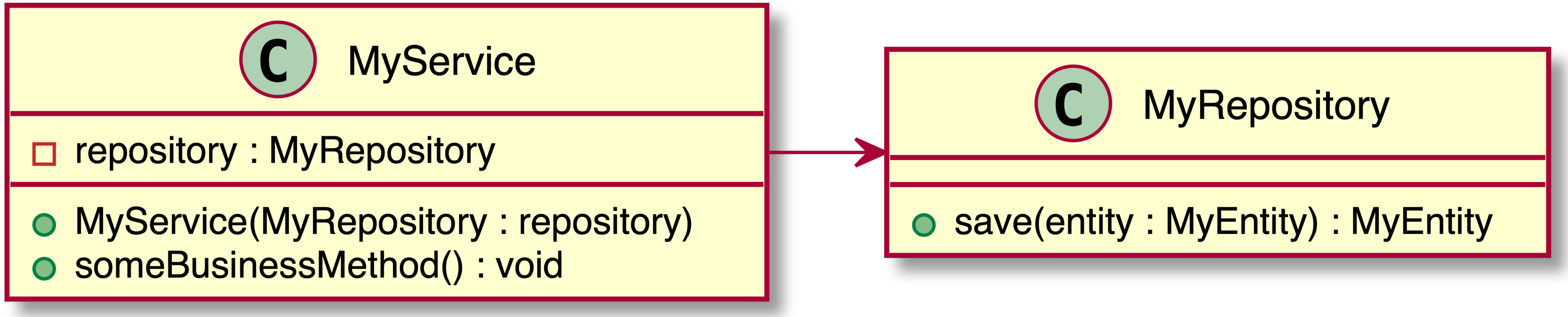
```
class MyRepository {

  MyEntity save(MyEntity entity) {
    …
  }
}
```


MyService
repository : MyRepository
MyService()
someBusinessMethod() :void
MyRepository
save(entity : MyEntity) : MyEntity

```
class MyService {

  private final MyRepository repo;

  MyService() {
    … // ?
  }

  void someBusinessMethod() {
    repo.save(new MyEntity());
  }
}
```

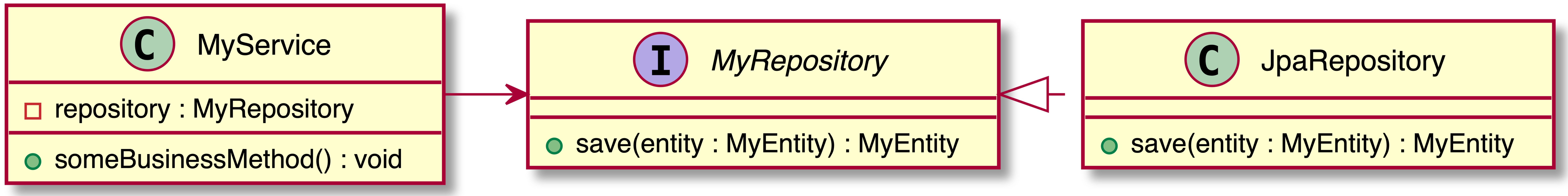
```
class MyRepository {

  MyEntity save(MyEntity entity) {
    …
  }
}
```

How do we obtain a repository instance?


MyService
repository : MyRepository
MyService(MyRepository : repository)
someBusinessMethod() : void
MyRepository
save(entity : MyEntity) : MyEntity

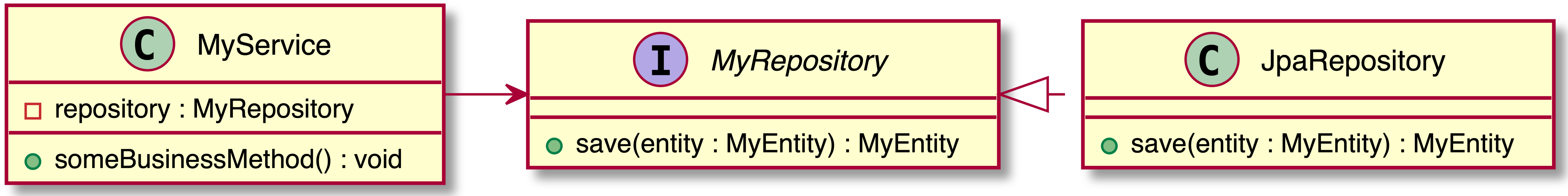
```
class MyService {

  private final MyRepository repo;

  MyService(MyRepository repo) {
    this.repo = repo
  }

  void someBusinessMethod() {
    repo.save(new MyEntity());
  }
}
```

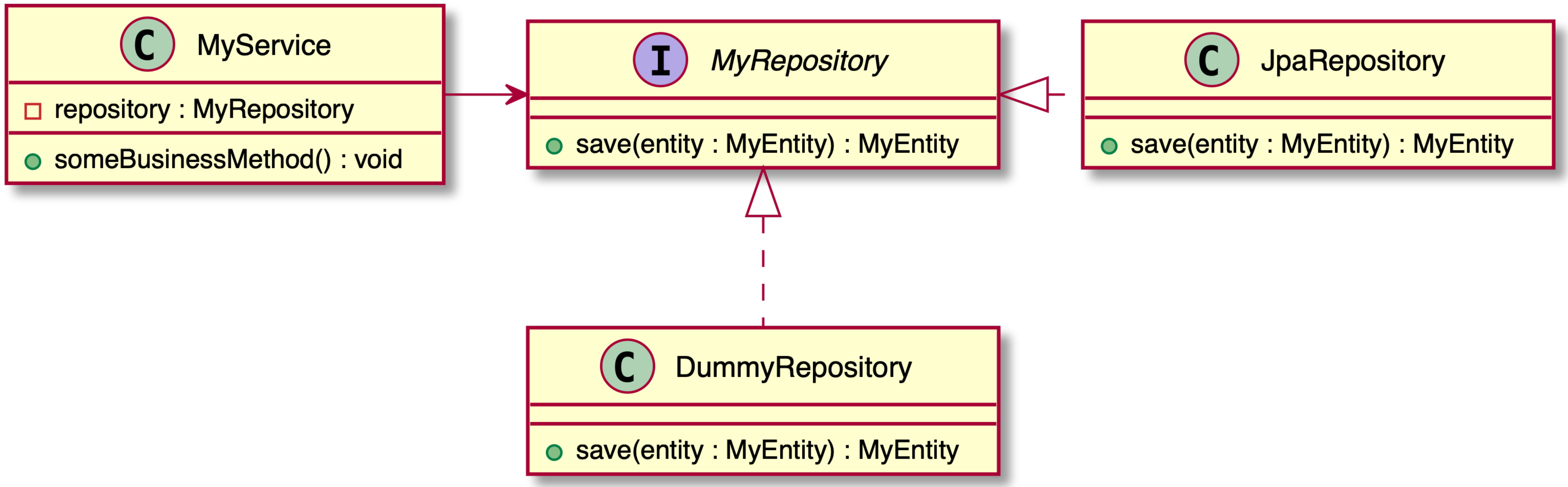
```
class MyRepository {

  MyEntity save(MyEntity entity) {
    …
  }
}
```


C MyService
repository : MyRepository
someBusinessMethod() : void
I MyRepository
save(entity : MyEntity) : MyEntity
C JpaRepository
save(entity : MyEntity) : MyEntity

```
class MyService {

  private final MyRepository repo;

  MyService(MyRepository repo) {
    this.repo = repo
  }

  void someBusinessMethod() {
    repo.save(new MyEntity());
  }
}
```

```
interface MyRepository {
  MyEntity save(MyEntity entity);
}

class JpaRepository
          implements MyRepository {

  @Override
  MyEntity save(MyEntity entity) {
    …
  }
}
```


C MyService
repository : MyRepository
someBusinessMethod() : void
I MyRepository
save(entity : MyEntity) : MyEntity
C JpaRepository
save(entity : MyEntity) : MyEntity


C MyService
repository : MyRepository
someBusinessMethod() : void
I MyRepository
save(entity : MyEntity) : MyEntity
C JpaRepository
save(entity : MyEntity) : MyEntity
C DummyRepository
save(entity : MyEntity) : MyEntity

```java
// For production
MyRepository repository = new JpaRepository();
MyService service = new MyService(repository);

// For tests
MyRepository repository = new DummyRepository();
MyService service = new MyService(repository);
```

Different implementations for production and test!

Manual dependency injection

Dependency Injection allows to select the actual implementation at construction time.

```
@Component
class MyService {
  MyService(MyRepository repository) { … }
}

@Component
class JpaRepository implements MyRepository { … }

// For production
ApplicationContext context =
                   new AnnotationConfigApplicationContext();
MyService service = context.getBean(MyService.class);
```

Declare classes as framework components

Bootstrap framework

Spring-based Dependency Injection

Portable Service Abstraction

Transactions

Security

…

Transactions

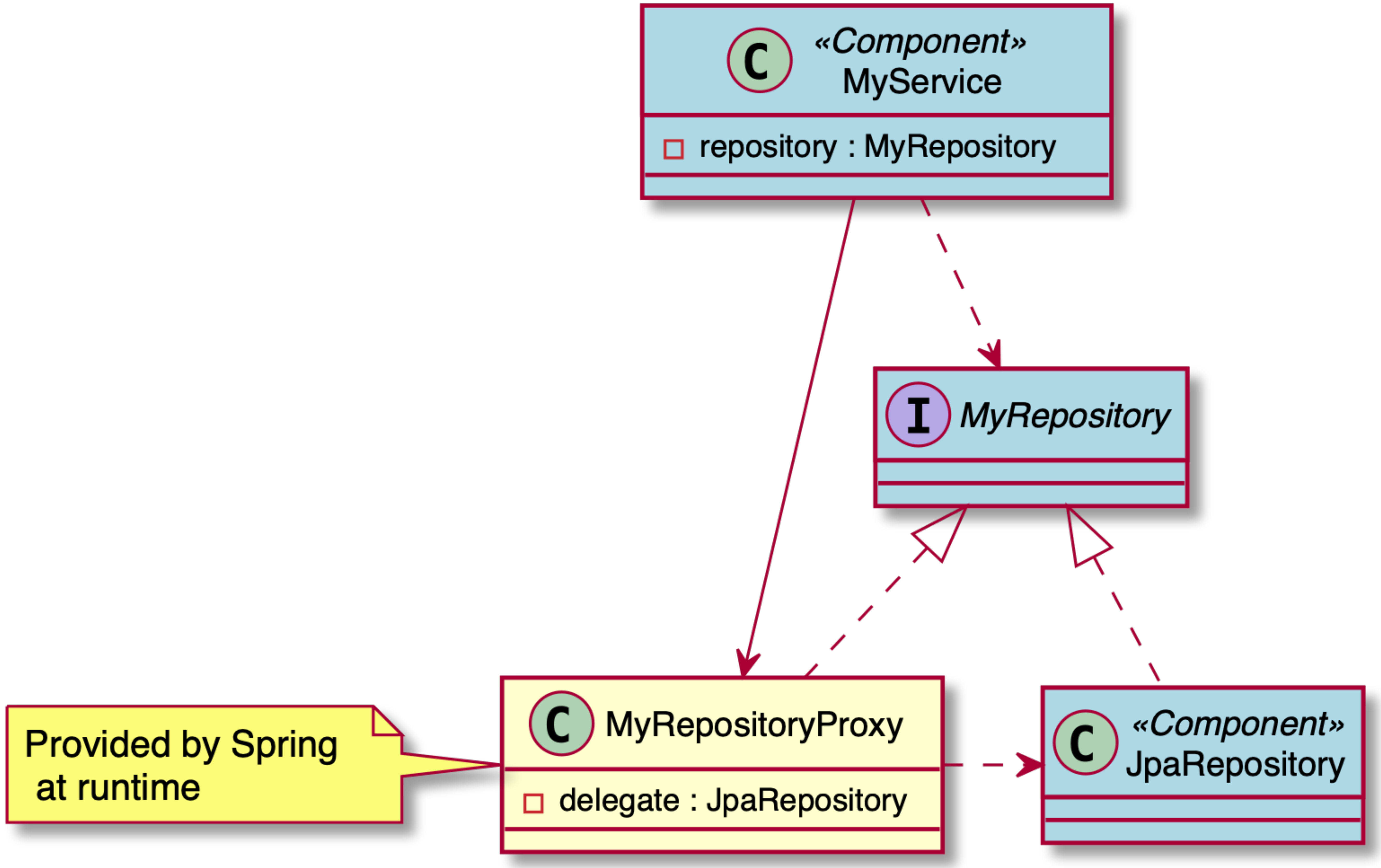
```
class JpaRepository implements MyRepository {

  @Override
  @Transactional
  MyEntity save(MyEntity entity) {

  }
}
```

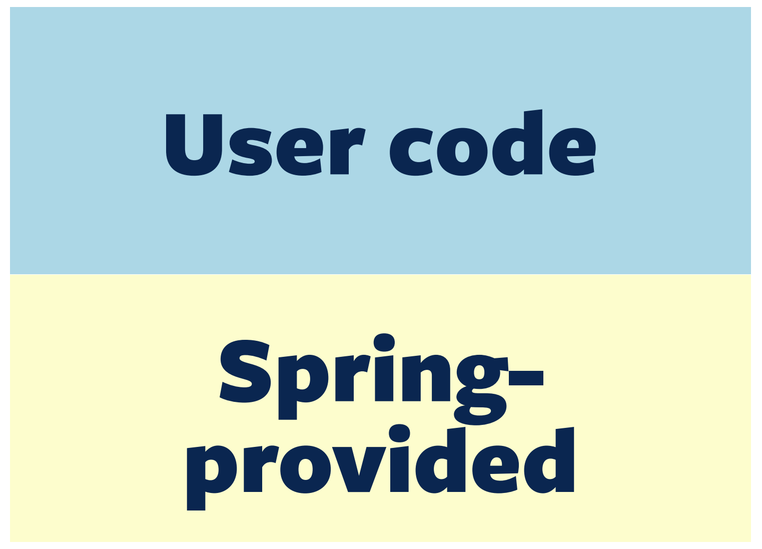
The magic happens here!

Declarative transactions

```
// Wrapping a component into a transactional proxy

JpaRepository repository = new JpaRepository();

ProxyFactory factory = new ProxyFactory(repository);
factory.addAdvice(new TransactionInterceptor(…));

MyRepository proxy = factory.getProxy();
MyService service = new MyService(proxy);
```

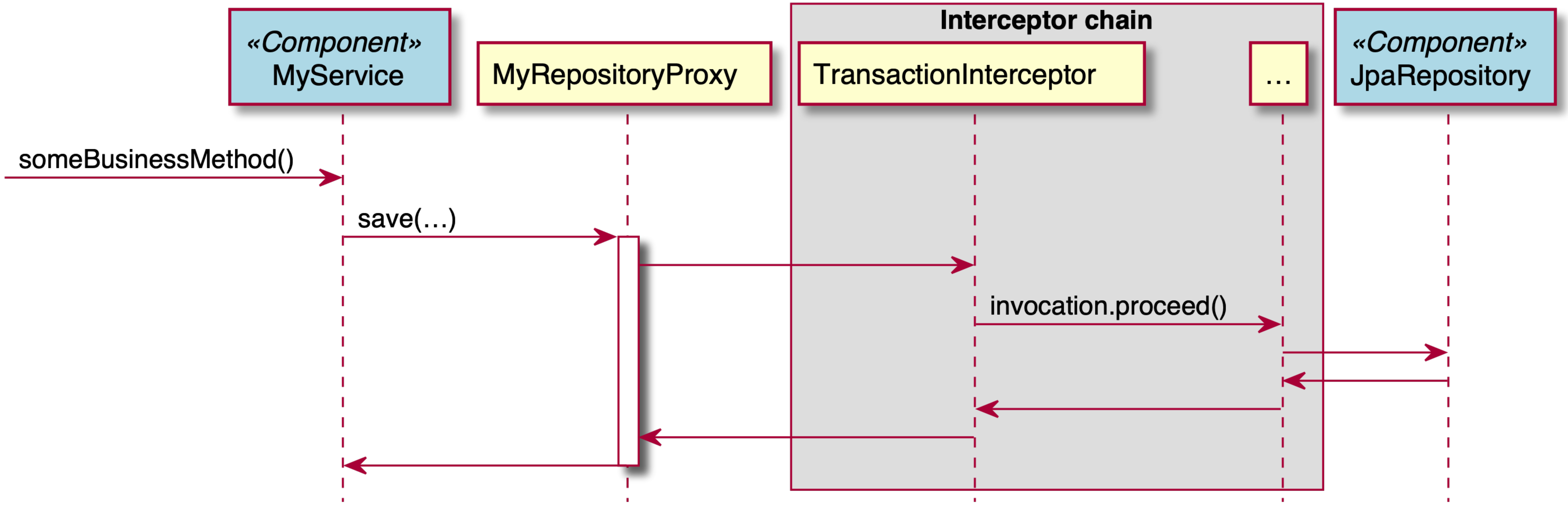
Proxy creation


C «Component» MyService
repository : MyRepository
I MyRepository
C MyRepositoryProxy
delegate : JpaRepository
C «Component» JpaRepository
Provided by Spring at runtime



Runtime component setup

User code	Spring-provided



Proxy invocation flow

Spring MVC

```
class MyController {



}
```

Spring MVC Controller

```
@RestController
class MyController {




}
```

Declares what kind of class that is

Spring MVC Controller

```
@RestController
class MyController {


  String sayHelloTo(                                  ) {

  }
}
```

Spring MVC Controller

```java
@RestController
class MyController {

  @GetMapping("/hello")
  String sayHelloTo(@RequestParam Optional<String> name) {

  }
}
```

Which URI to map to?

Spring MVC Controller

```
@RestController
class MyController {

  @GetMapping("/hello")
  String sayHelloTo(@RequestParam Optional<String> name) {

  }
}
```

What parts of the request are we interested in?

Spring MVC Controller

```java
@RestController
class MyController {

  @GetMapping("/hello")
  String sayHelloTo(@RequestParam Optional<String> name) {
    return String.format("Hello, %s!", name.orElse("world"));
  }
}
```

Spring MVC Controller

Using Frameworks

Reuse VS. Coupling

Java and the Enterprise

1. Security of investment
2. Backwards compatibility
3. Availability of support

Thank you!

Oliver Drotbohm / odrotbohm odrotbohm@pivotal.io