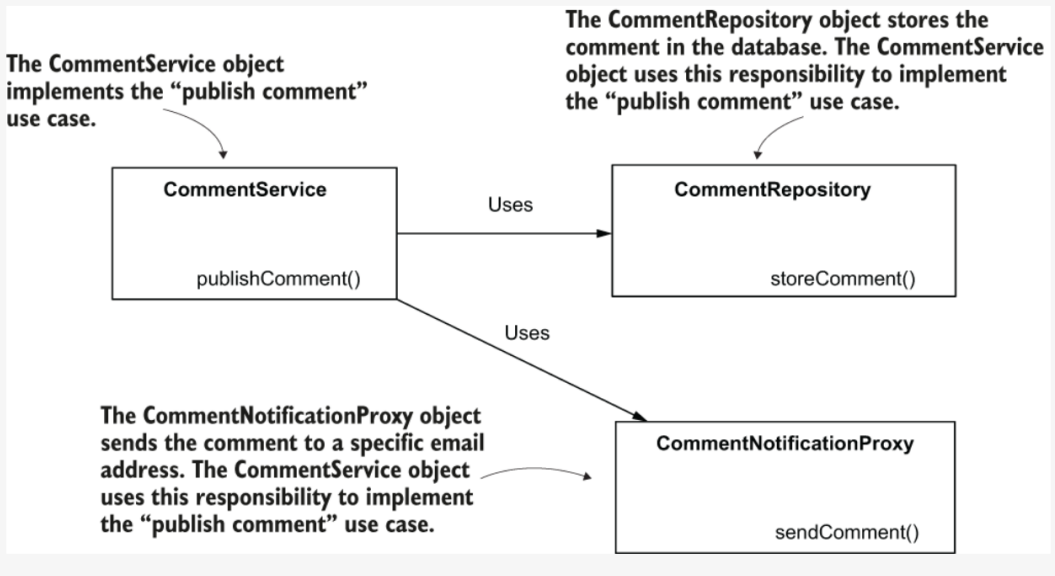
Using Abstraction

1) Review

- interface - what needs to happen
- class implements interface - show how it should happen

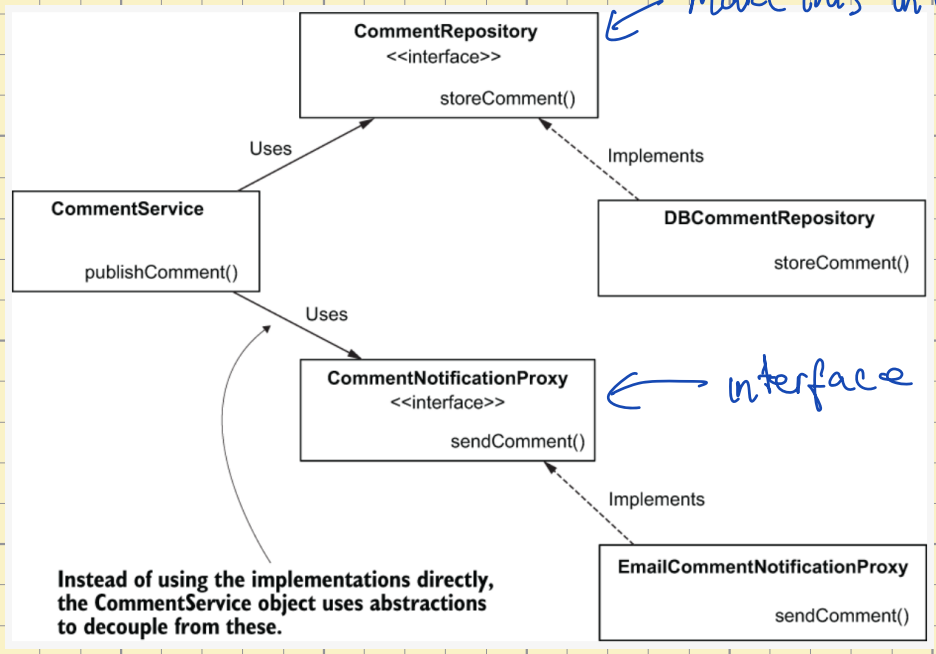
2) Without framework



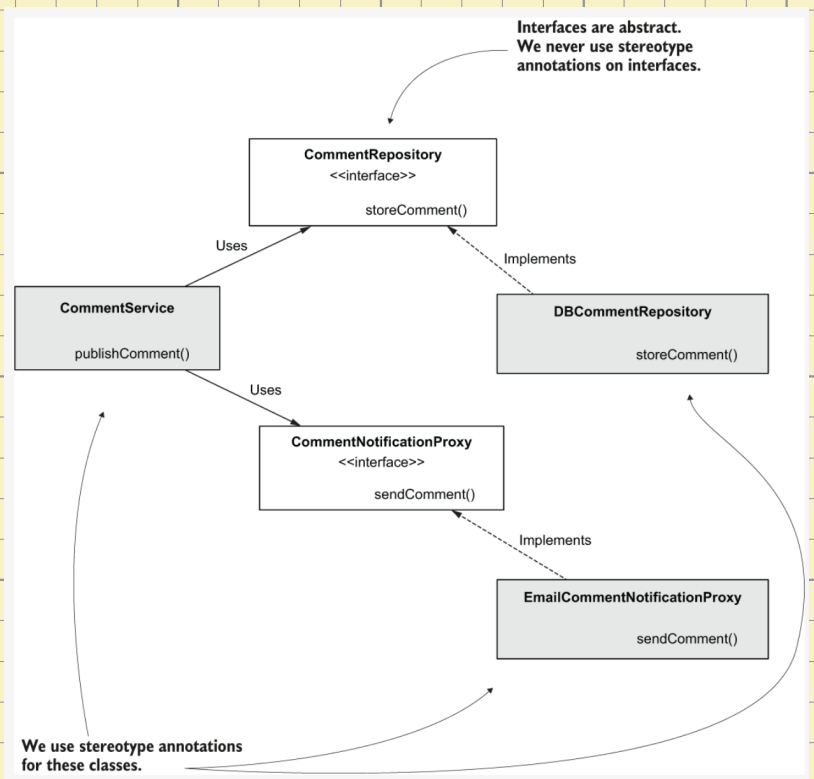
What if we want to communicate with other channels outside of the app / interact with other entities in the database?

→ Abstraction and decoupling

make this interface



3) With framework



graybox:
@Component
* We don't use stereotype annotation on abstract class/ interfaces cuz they are not instaintiated

- adding stereotype annotations instructs Spring to instantiate those objects and register them as beans in Spring context

```
@Component
public class CommentService {

  @Autowired
  private CommentRepository commentRepository;
  @Autowired
  private CommentNotificationProxy commentNotificationProxy;

  public void publishComment(Comment comment) {
    commentRepository.storeComment(comment);
    commentNotificationProxy.sendComment(comment);
  }
}
```

autowiring. method using field

autowire

```
@Configuration                                          ❶
public class ProjectConfiguration {

  @Bean                                                 ❷
  public CommentRepository commentRepository() {
    return new DBCommentRepository();
  }

  @Bean                                                 ❷
  public CommentNotificationProxy commentNotificationProxy() {
    return new EmailCommentNotificationProxy();
  }

  @Bean
  public CommentService commentService(
    CommentRepository commentRepository,                ❸
    CommentNotificationProxy commentNotificationProxy) {
    return new CommentService(commentRepository, commentNotificati
  }
}
```

wiring

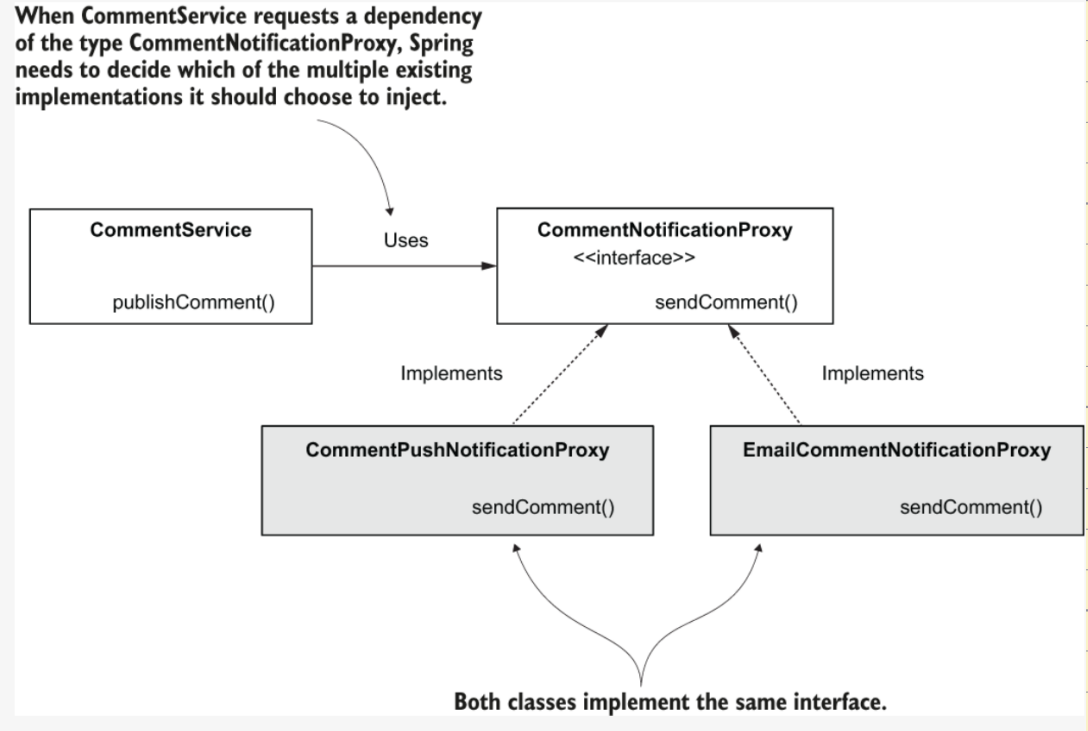
method

with Bean creation

How can we autowire when an interface has multiple implementations?

→ Using same approach ┌ same method identifier
                       └ @Primary, @Qualifier

Priority : @Qualifier > @Primary > same as method name



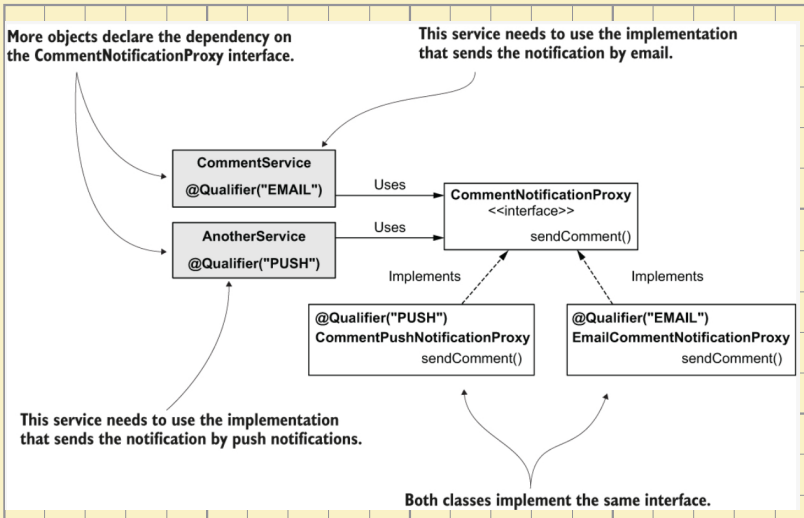
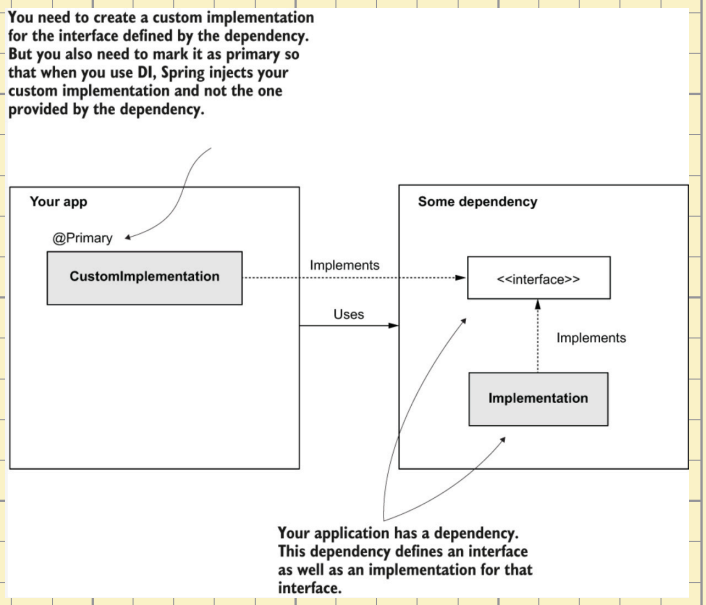


diagram using @Qualifier approach

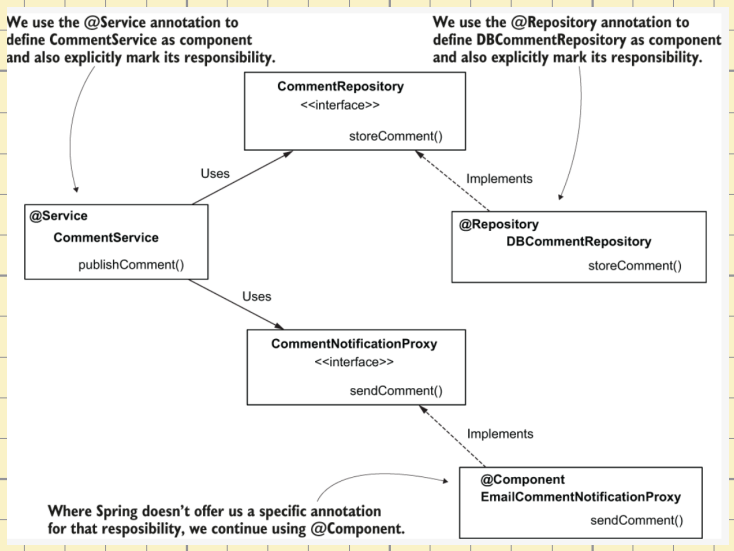
← @Qualifier not only can be used as part of the method arguements but can also be placed on top of the class ←

```
@Component
@Qualifier("PUSH")
public class CommentPushNotificationProxy
  implements CommentNotificationProxy {
  // Omitted code
}
```

diagram using @Primary approach →



4) More stereotype annotations @Service, @Repository



Terminology and Structure

1) objects work with database — repository / DAO (data object access)

2) objects communicate with sth outside of the app — proxies

3) simple class w/o dependencies POJO class — model

```
public class Comment {

  private String author;
  private String text;

  // Omitted getters and setters
}
```

4) mark a component that takes responsibility as a service — service @Service

5) mark a component that implements a repository - repository @Repository