<epam>

SOLIDный чистый код

на простых примерах

Александр Бармин

TechTrain 2019

<epam>

Александр Бармин

Lead Software Engineer

Ментор лаборатории EPAM

Контакты

Email: Aleksandr_Barmin@epam.com

Twitter: @AlexBarmin

План

- Стандарты программистов
- Как писать хороший код
- Соглашения разработчиков
- Вопросы и ответы

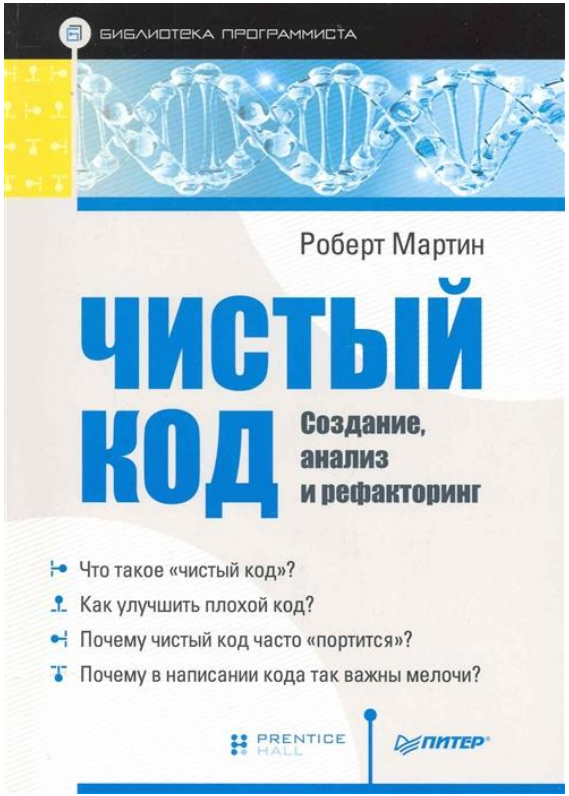
Стандарты программистов

- У инженеров есть стандарты, а у программистов?
- Стандарты есть:
  - Внутренние и отраслевые стандарты
  - Стандарты на основе лучших практик
  - Есть инструменты для проверки качества, но голову они не заменяют

Чистый код

Чистый код

- Роберт Мартин “Чистый код”
- Набор практик, принципов, приемов и паттернов



Золотые цитаты

https://twitter.com/codinghorror/status/506010907021828096

Чистый код - осмысленные имена

```
int d;

String n;

List<String> getNames() { ... }

void setValues(List<String> values) { ... }

int calc() { ... }
```

Чистый код - осмысленные имена

```
int fileAgeInDays;

String currentUserName;

List<String> getAvailableVendorNames() { ... }

void setSynonyms(List<String> values) { ... }

int findNextAvailableIndex() { ... }
```

Чистый код - известные исключения

```
for (int i = 0; i < 10; i++) {
 for (int j = 0; j < 20; j++) {
   for (int k = 0; k < 30; k++) {
     // ... do something here
   }
 }
}
```

Золотые цитаты

Talk is cheap. Show me the code.

Линус Торвальдс

Сообщение в список рассылки ядра linux. Получено 28.08.2006.

https://en.wikiquote.org/wiki/Linus_Torvalds

Train-o-Gram

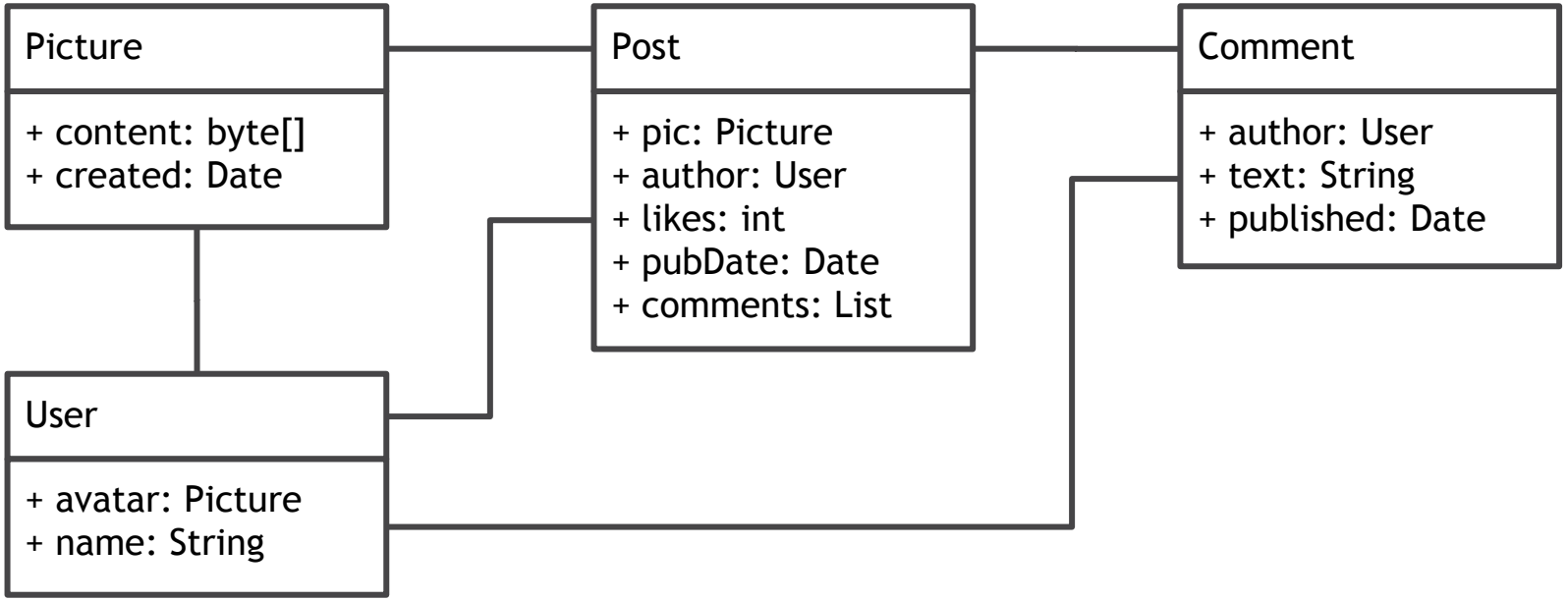
- Постить фоточки
- Собирать лайки
- Писать комменты
- Добавлять в друзья
- Публиковать спонсорские посты
- Делать отчеты

100500 лайков

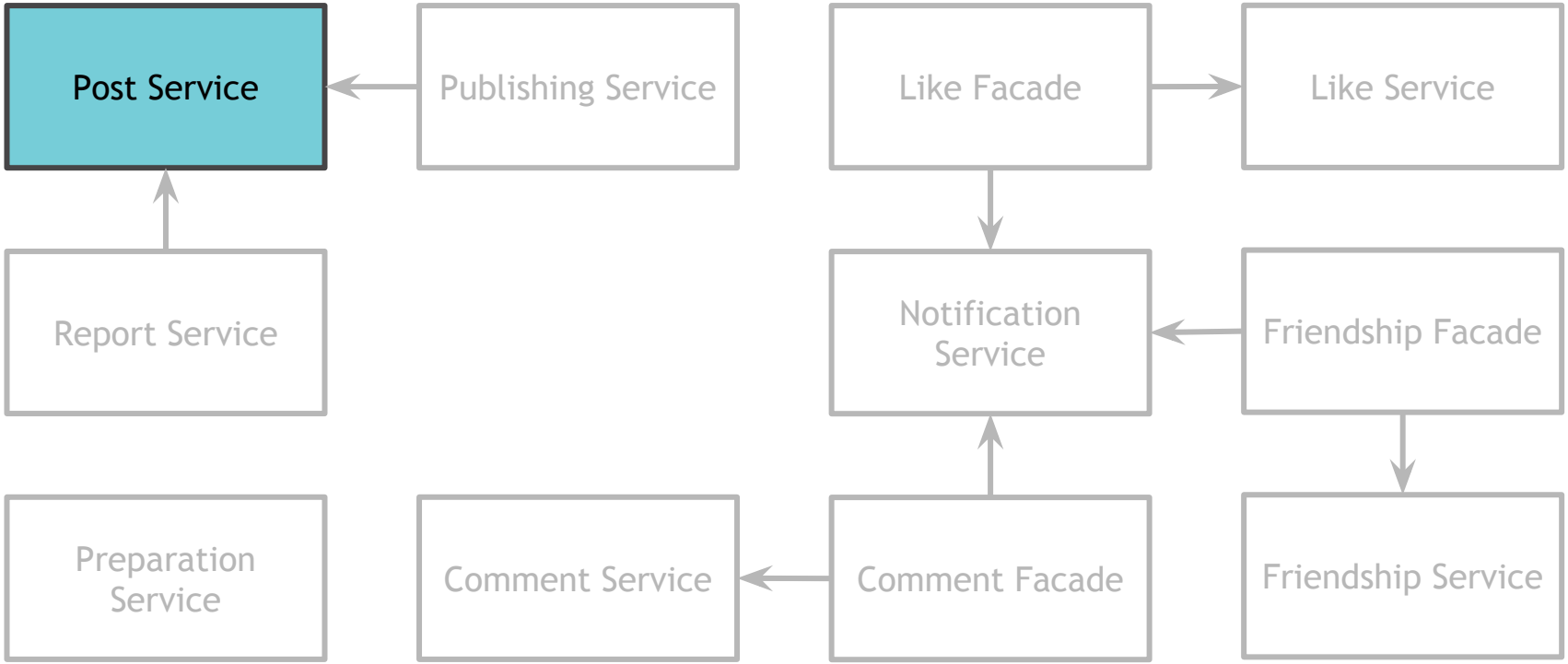
Коммент от Васи
Коммент от Пети
Коммент от Ксюши

Запостить в Train-o-Gram

Train-o-Gram


Picture
+ content: byte[]
+ created: Date
Post
+ pic: Picture
+ author: User
+ likes: int
+ pubDate: Date
+ comments: List
Comment
+ author: User
+ text: String
+ published: Date
User
+ avatar: Picture
+ name: String

Архитектура решения



Чистый код - единообразие

```
public interface PostService {
  List<Post> findAll();
  List<Post> getByUser(User user);
  List<Post> retrieveRecommendations(User user);
}
```

Чистый код - единообразие

```
public interface PostService {
  List<Post> findAll();
  List<Post> findAll(User user);
  List<Post> findRecommendations(User user);
}
```

Чистый код - функции

Рекомендательная система:

- Получить все посты пользователя
- Получить комментарии на каждый пост
- Взять пять первых постов у каждого автора комментария

```java
public class PostServiceImpl implements PostService {
 @Override
 public List<Post> findRecommendations(User user) {
   List<Post> suggestions = new ArrayList<>();
   List<Post> allPosts = findAll(user);
   for (Post post : allPosts) {
     List<Comment> comments = post.getComments();
     for (Comment comment : comments) {
       User commentAuthor = comment.getAuthor();
       List<Post> commenterPosts = findAll(commentAuthor);
       int count = 0;
       for (Post commenterPost : commenterPosts) {
         if (count == 5) { break; }
         count++;
         suggestions.add(commenterPost);
       }
     }
   }
   return suggestions;
 }
}
```

```
public class PostServiceImpl implements PostService {
 @Override
 public List<Post> findRecommendations(User user) {
   List<Post> suggestions = new ArrayList<>();
   List<Post> allPosts = findAll(user);
   for (Post post : allPosts) {
     List<Comment> comments = post.getComments();
     for (Comment comment : comments) {
       User commentAuthor = comment.getAuthor();
       List<Post> commenterPosts = findAll(commentAuthor);
       int count = 0;
       for (Post commenterPost : commenterPosts) {
         if (count == 5) { break; }
         count++;
         suggestions.add(commenterPost);
       }
     }
   }
   return suggestions;
 }
}
```

Все посты пользователя

```
public class PostServiceImpl implements PostService {
 @Override
 public List<Post> findRecommendations(User user) {
   List<Post> suggestions = new ArrayList<>();
   List<Post> allPosts = findAll(user);
   for (Post post : allPosts) {
     List<Comment> comments = post.getComments();
     for (Comment comment : comments) {
       User commentAuthor = comment.getAuthor();
       List<Post> commenterPosts = findAll(commentAuthor);
       int count = 0;
       for (Post commenterPost : commenterPosts) {
         if (count == 5) { break; }
         count++;
         suggestions.add(commenterPost);
       }
     }
   }
   return suggestions;
 }
}
```

Все посты пользователя

Все комментарии

```
public class PostServiceImpl implements PostService {
 @Override
 public List<Post> findRecommendations(User user) {
   List<Post> suggestions = new ArrayList<>();
   List<Post> allPosts = findAll(user);
   for (Post post : allPosts) {
     List<Comment> comments = post.getComments();
     for (Comment comment : comments) {
       User commentAuthor = comment.getAuthor();
       List<Post> commenterPosts = findAll(commentAuthor);
       int count = 0;
       for (Post commenterPost : commenterPosts) {
         if (count == 5) { break; }
         count++;
         suggestions.add(commenterPost);
       }
     }
   }
   return suggestions;
 }
}
```

Все посты пользователя

Все комментарии

У автора комментария все посты

```
public class PostServiceImpl implements PostService {
 @Override
 public List<Post> findRecommendations(User user) {
   List<Post> suggestions = new ArrayList<>();
   List<Post> allPosts = findAll(user);
   for (Post post : allPosts) {
     List<Comment> comments = post.getComments();
     for (Comment comment : comments) {
       User commentAuthor = comment.getAuthor();
       List<Post> commenterPosts = findAll(commentAuthor);
       int count = 0;
       for (Post commenterPost : commenterPosts) {
         if (count == 5) { break; }
         count++;
         suggestions.add(commenterPost);
       }
     }
   }
   return suggestions;
 }
}
```

Все посты пользователя

Все комментарии

У автора комментария все посты

Берем первые пять

```
public class PostServiceImpl implements PostService {
 @Override
 public List<Post> findRecommendations(User user) {
   List<Post> suggestions = new ArrayList<>();
   List<Post> allPosts = findAll(user);
   for (Post post : allPosts) {
     List<Comment> comments = post.getComments();
     for (Comment comment : comments) {
       User commentAuthor = comment.getAuthor();
       List<Post> commenterPosts = findAll(commentAuthor);
       int count = 0;
       for (Post commenterPost : commenterPosts) {
         if (count == 5) { break; }
         count++;
         suggestions.add(commenterPost);
       }
     }
   }
   return suggestions;
 }
}
```

Все посты пользователя

Все комментарии

У автора комментария все посты

Берем первые пять

Читабельно?

```java
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        User commentAuthor = comment.getAuthor();
        List<Post> commenterPosts = findAll(commentAuthor);
        suggestions.addAll(getTopPosts(commenterPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getTopPosts(List<Post> posts, int number) {
    if (number > posts.size()) { number = posts.size(); }
    return posts.subList(0, number);
  }
}
```

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        User commentAuthor = comment.getAuthor();
        List<Post> commenterPosts = findAll(commentAuthor);
        suggestions.addAll(getTopPosts(commenterPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getTopPosts(List<Post> posts, int number) {
    if (number > posts.size()) { number = posts.size(); }
    return posts.subList(0, number);
  }
}
```

Возвращает последние посты

```java
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        User commentAuthor = comment.getAuthor();
        List<Post> commenterPosts = findAll(commentAuthor);
        suggestions.addAll(getTopPosts(commenterPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getTopPosts(List<Post> posts, int number) {
    if (number > posts.size()) { number = posts.size(); }
    return posts.subList(0, number);
  }
}
```

Берем последние пять

Возвращает последние посты

Лучше? Немного...

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        User commentAuthor = comment.getAuthor();
        List<Post> commenterPosts = findAll(commentAuthor);
        suggestions.addAll(getTopPosts(commenterPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getTopPosts(List<Post> posts, int number) {
    if (number > posts.size()) { number = posts.size(); }
    return posts.subList(0, number);
  }
}
```

Берем последние пять

Возвращает последние посты

```java
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getCommentAuthorPosts(Comment comment) {
    User author = comment.getAuthor();
    return findAll(author);
  }
}
```

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getCommentAuthorPosts(Comment comment) {
    User author = comment.getAuthor();
    return findAll(author);
  }
}
```

Возвращает посты автора комментария

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }

  private List<Post> getCommentAuthorPosts(Comment comment) {
    User author = comment.getAuthor();
    return findAll(author);
  }
}
```

Посты автора комментария

Возвращает посты автора комментария

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }
}
```

Чистый код - функции

Цикл по постам

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }
}
```

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }
}
```

Цикл по постам

Цикл по комментариям

Чистый код - функции

Цикл по постам

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    List<Post> suggestions = new ArrayList<>();
    List<Post> allPosts = findAll(user);
    for (Post post : allPosts) {
      List<Comment> comments = post.getComments();
      for (Comment comment : comments) {
        List<Post> commentAuthorPosts = getCommentAuthorPosts(comment);
        suggestions.addAll(getTopPosts(commentAuthorPosts, 5));
      }
    }
    return suggestions;
  }
}
```

Код-то не про циклы!

Цикл по комментариям

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Чистый код - функции

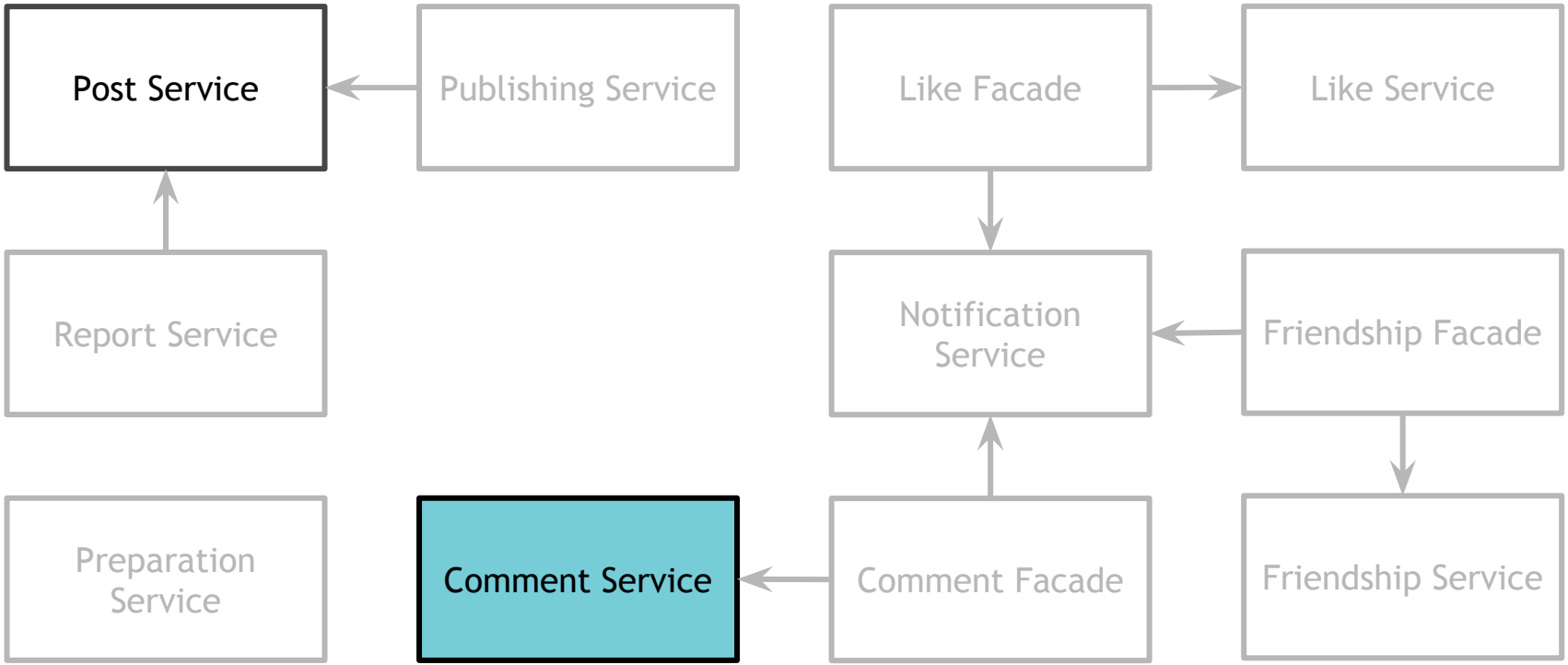
```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Чистый код - функции

```
public class PostServiceImpl implements PostService {
  @Override
  public List<Post> findRecommendations(User user) {
    return findAll(user).stream()
        .flatMap(post -> post.getComments().stream())
        .map(comment -> comment.getAuthor())
        .flatMap(commentAuthor -> findAll(commentAuthor).stream().limit(5))
        .collect(Collectors.toList());
  }
}
```

Очевидно!

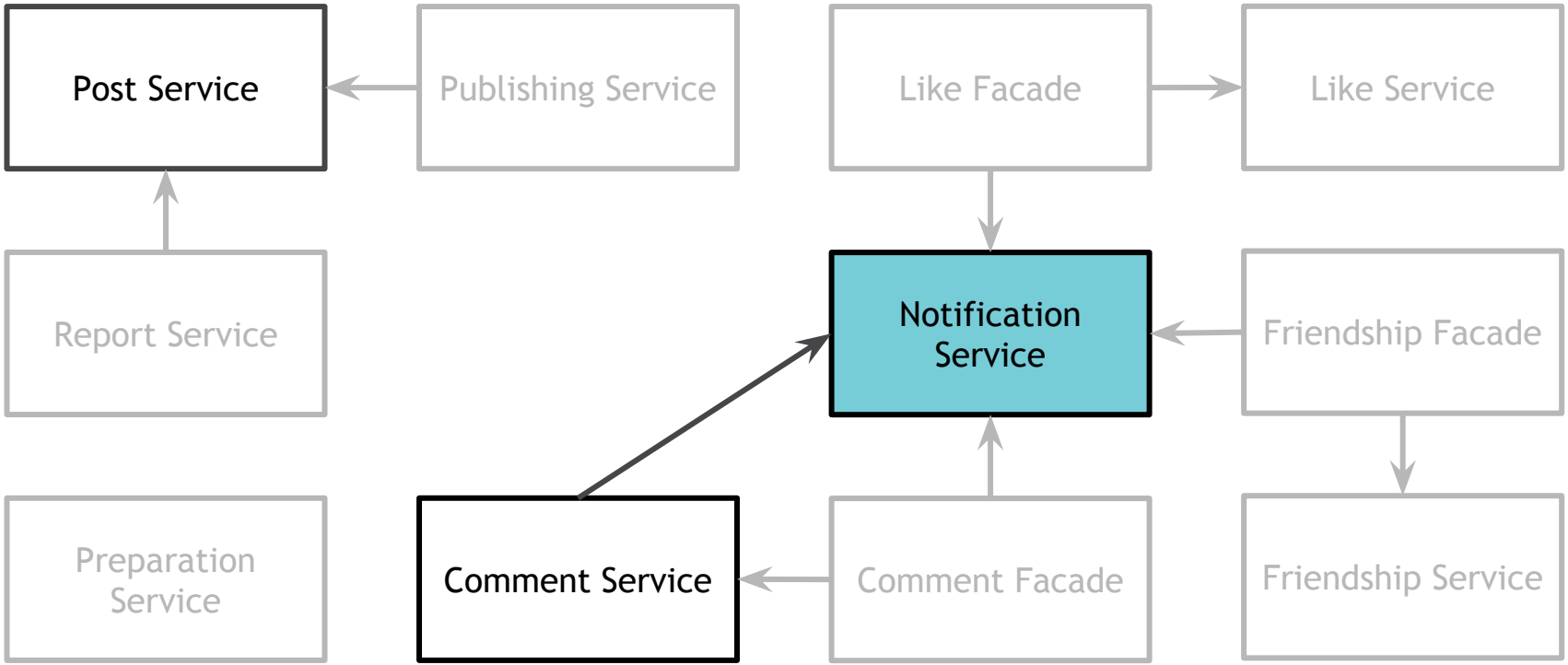
Архитектура решения



Чистый код - чистые функции

```
public interface CommentService {

  void addComment(Post post, Comment comment);

}
```

Архитектура решения



Чистый код - чистые функции

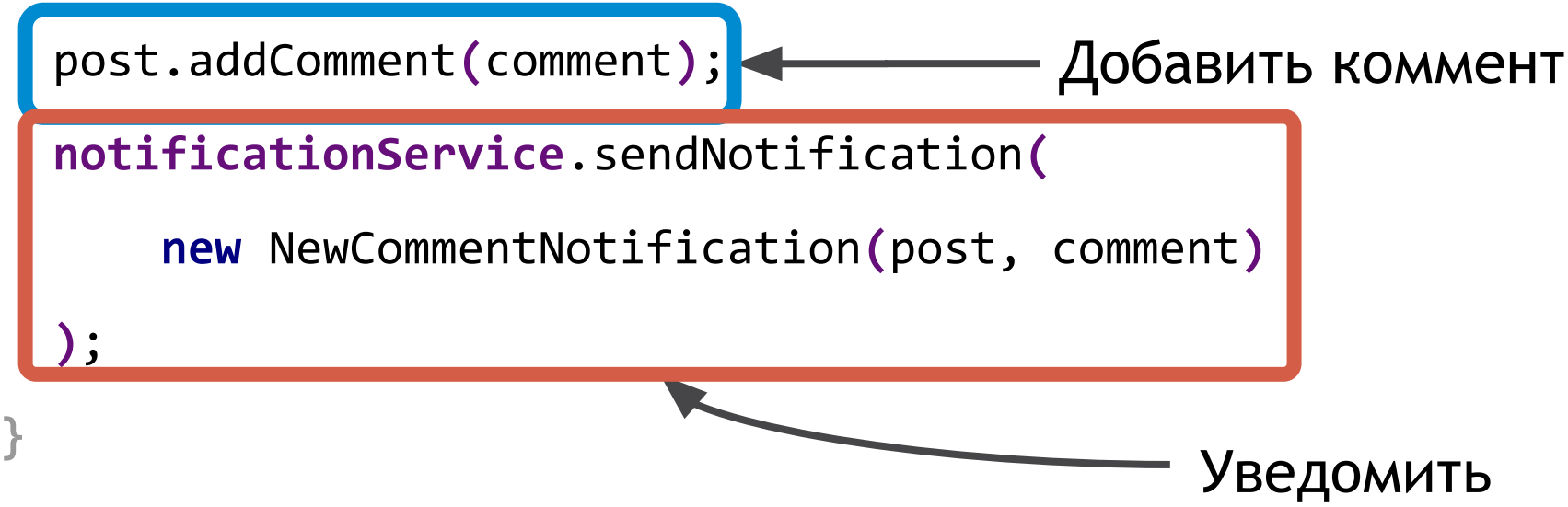
```
public class CommentServiceImpl implements CommentService {
  private NotificationService notificationService;
  public void addComment(Post post, Comment comment) {
    post.addComment(comment);
    notificationService.sendNotification(
        new NewCommentNotification(post, comment)
    );
  }
}
```

Чистый код - чистые функции

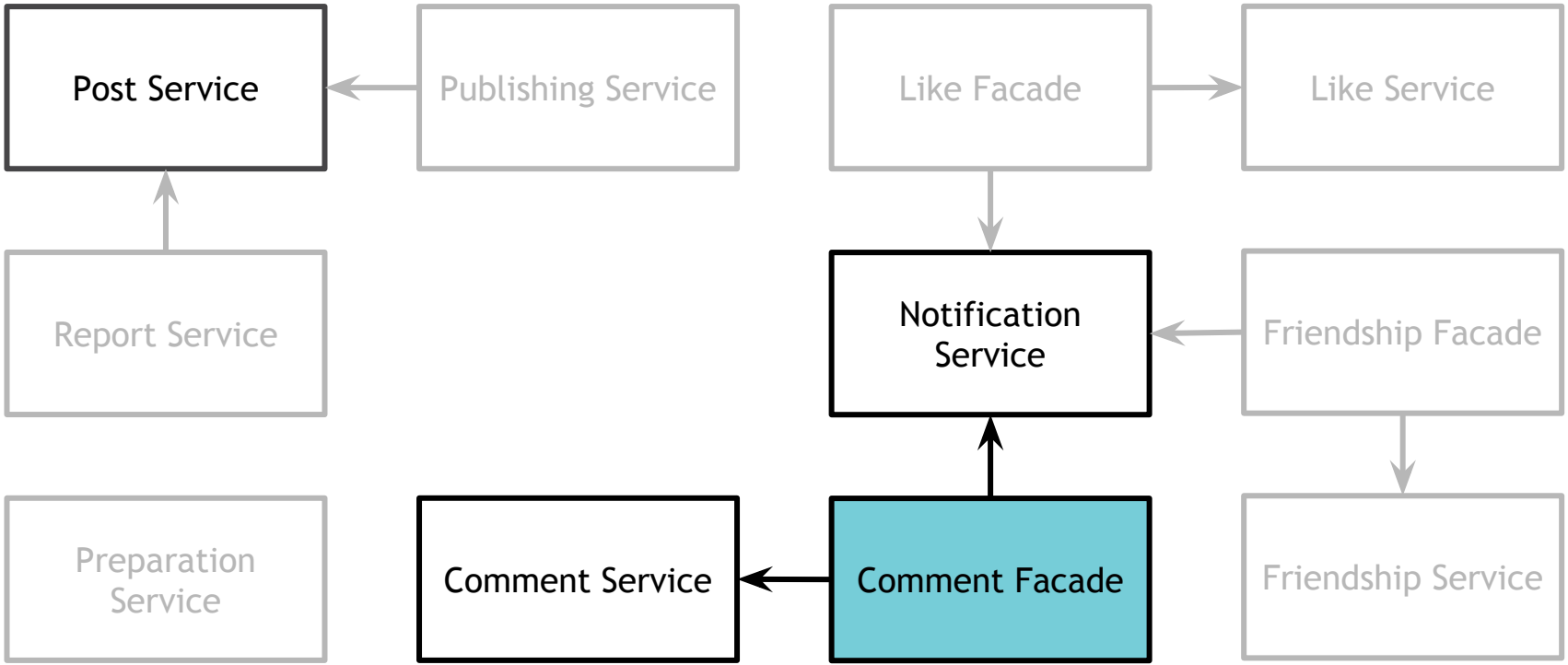
```
public class CommentServiceImpl implements CommentService {
  private NotificationService notificationService;
  public void addComment(Post post, Comment comment) {
    post.addComment(comment);
    notificationService.sendNotification(
        new NewCommentNotification(post, comment)
    );
  }
}
```

Чистый код - чистые функции

```
public class CommentServiceImpl implements CommentService {
  private NotificationService notificationService;
  public void addComment(Post post, Comment comment) {
    post.addComment(comment);
    notificationService.sendNotification(
        new NewCommentNotification(post, comment)
    );
  }
}
```



Архитектура решения



Чистый код - чистые функции

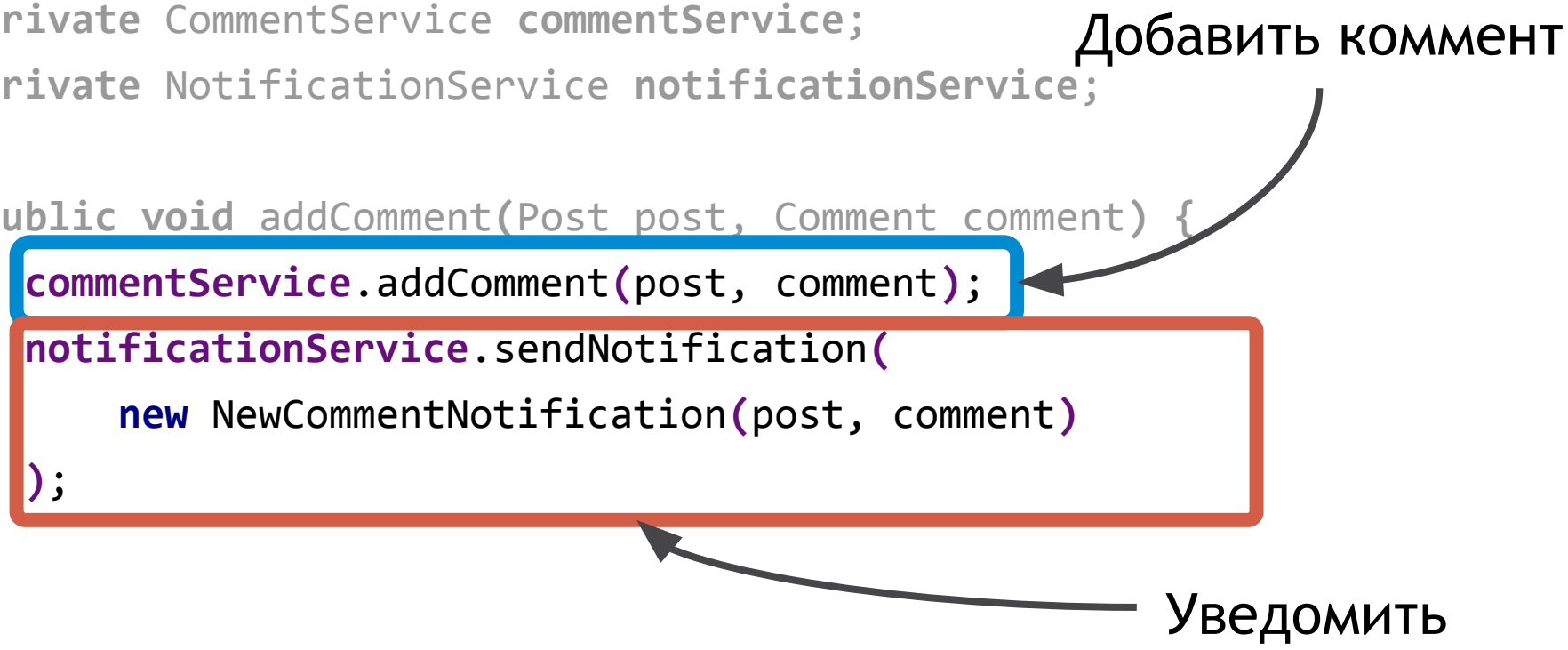
```
public class CommentsFacadeImpl implements CommentsFacade {
 private CommentService commentService;
 private NotificationService notificationService;

 public void addComment(Post post, Comment comment) {
   commentService.addComment(post, comment);
   notificationService.sendNotification(
       new NewCommentNotification(post, comment)
   );
 }
}
```

Чистый код - чистые функции

```
public class CommentsFacadeImpl implements CommentsFacade {
 private CommentService commentService;
 private NotificationService notificationService;

 public void addComment(Post post, Comment comment) {
   commentService.addComment(post, comment);
   notificationService.sendNotification(
       new NewCommentNotification(post, comment)
   );
 }
}
```

Чистый код - чистые функции

```
public class CommentsFacadeImpl implements CommentsFacade {
 private CommentService commentService;
 private NotificationService notificationService;

 public void addComment(Post post, Comment comment) {
   commentService.addComment(post, comment);
   notificationService.sendNotification(
       new NewCommentNotification(post, comment)
   );
 }
}
```



Золотые цитаты

Брейнуальное тестирование — это как мануальное, только с мозгами.

Барух Садогурский

Гейзенбаг 2018

Чистый код

- Брейнуальное программирование
- Имена отражают намерение
- Единообразное наименование
- Сложные методы разделять на маленькие
- Выразительные средства языка

SOLID — еще одна аббревиатура

SOLID

- Роберт Мартин “Design Principles and Design Patterns”
- Майкл Фезерс ввел термин SOLID

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

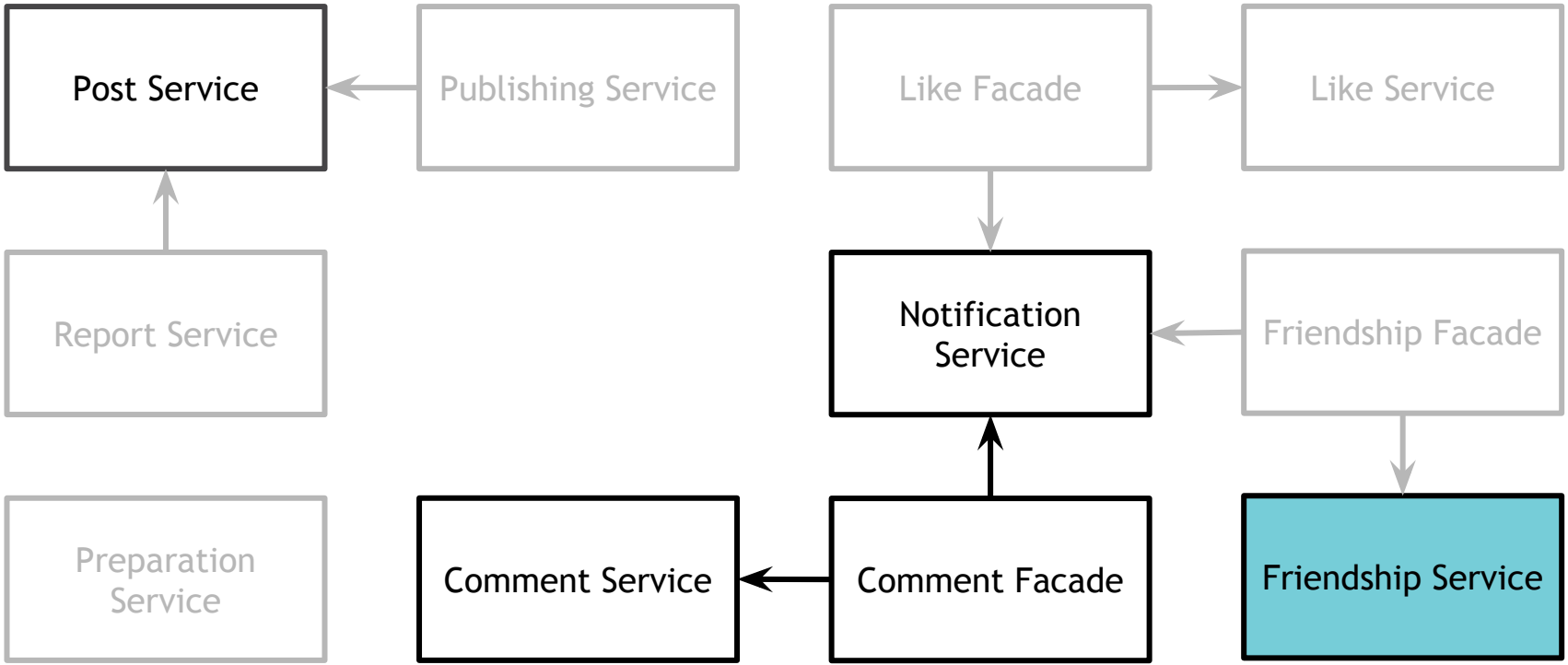
Interface Segregation Principle

Dependency Inversion Principle

У класса должна быть только одна причина для изменения

Роберт Мартин

Архитектура решения



Принцип единственной ответственности

```
public interface FriendshipService {

 void addToFriends(User owner, User newFriend);

}
```

Принцип единственной ответственности

```
public class FriendshipServiceImpl implements FriendshipService {
 private FriendshipDao friendshipDao;
 private NotificationService notificationService;
 public void addToFriends(User owner, User newFriend) {
   friendshipDao.save(new Friendship(owner, newFriend));
   friendshipDao.save(new Friendship(newFriend, owner));
   notificationService.sendNotification(
       new NewFriendshipNotification(owner, newFriend)
   );
 } }
```

Принцип единственной ответственности

```
public class FriendshipServiceImpl implements FriendshipService {
 private FriendshipDao friendshipDao;
 private NotificationService notificationService;
 public void addToFriends(User owner, User newFriend) {
   friendshipDao.save(new Friendship(owner, newFriend));
   friendshipDao.save(new Friendship(newFriend, owner));
   notificationService.sendNotification(
       new NewFriendshipNotification(owner, newFriend)
   );
 } }
```

Добавить в друзья

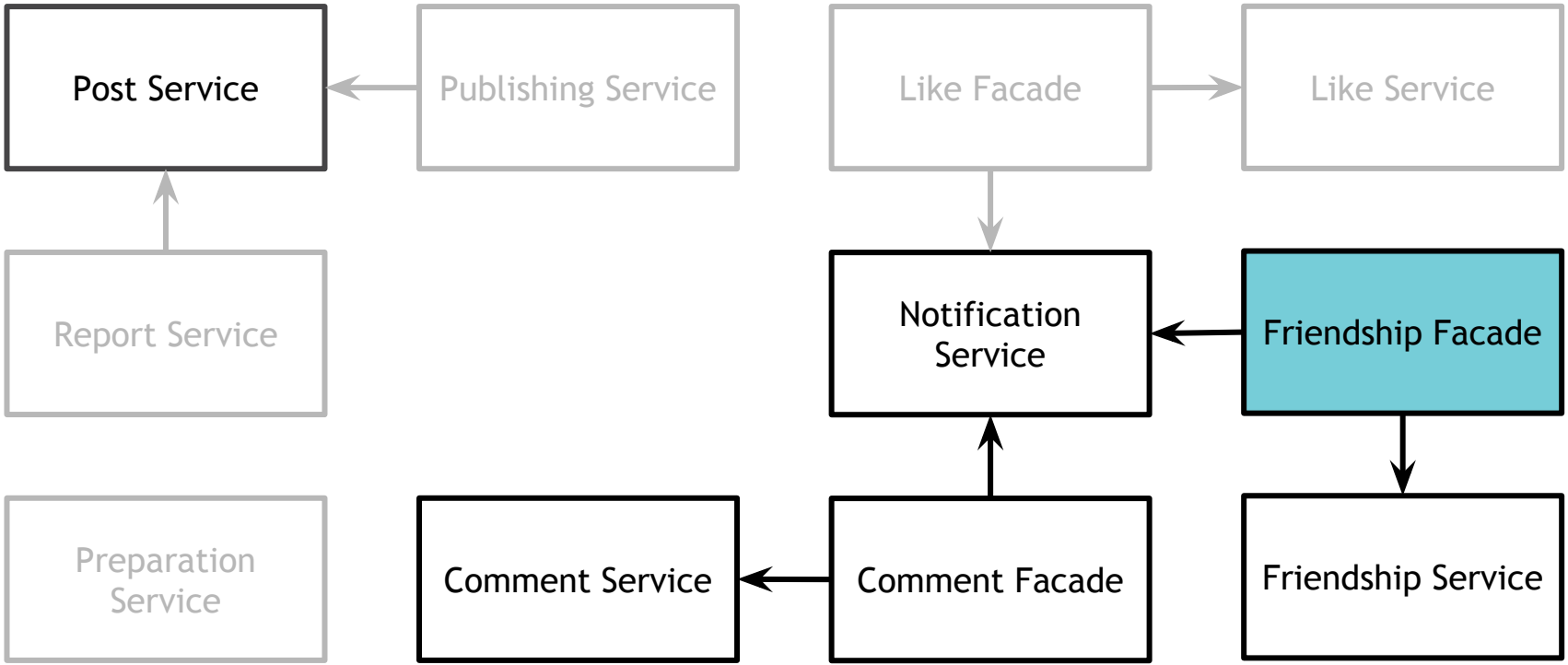
Принцип единственной ответственности

```
public class FriendshipServiceImpl implements FriendshipService {
 private FriendshipDao friendshipDao;
 private NotificationService notificationService;
 public void addToFriends(User owner, User newFriend) {
   friendshipDao.save(new Friendship(owner, newFriend));
   friendshipDao.save(new Friendship(newFriend, owner));
   notificationService.sendNotification(
       new NewFriendshipNotification(owner, newFriend)
   );
 } }
```

Добавить в друзья

Уведомить

Архитектура решения



Принцип единственной ответственности

```
public class FriendshipFacadeImpl implements FriendshipFacade {
  private FriendshipService friendshipService;
  private NotificationService notificationService;

  public void addToFriends(User owner, User newFriend) {
    friendshipService.addToFriends(owner, newFriend);
    notificationService.sendNotification(
        new NewFriendshipNotification(owner, newFriend)
    );
  }
}
```

Принцип единственной ответственности

```
public class FriendshipFacadeImpl implements FriendshipFacade {
  private FriendshipService friendshipService;
  private NotificationService notificationService;

  public void addToFriends(User owner, User newFriend) {
    friendshipService.addToFriends(owner, newFriend);
    notificationService.sendNotification(
        new NewFriendshipNotification(owner, newFriend)
    );
  }
}
```

Добавить в друзья

Принцип единственной ответственности

```
public class FriendshipFacadeImpl implements FriendshipFacade {
  private FriendshipService friendshipService;
  private NotificationService notificationService;

  public void addToFriends(User owner, User newFriend) {
    friendshipService.addToFriends(owner, newFriend);
    notificationService.sendNotification(
        new NewFriendshipNotification(owner, newFriend)
    );
  }
}
```

Добавить в друзья

Уведомить

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

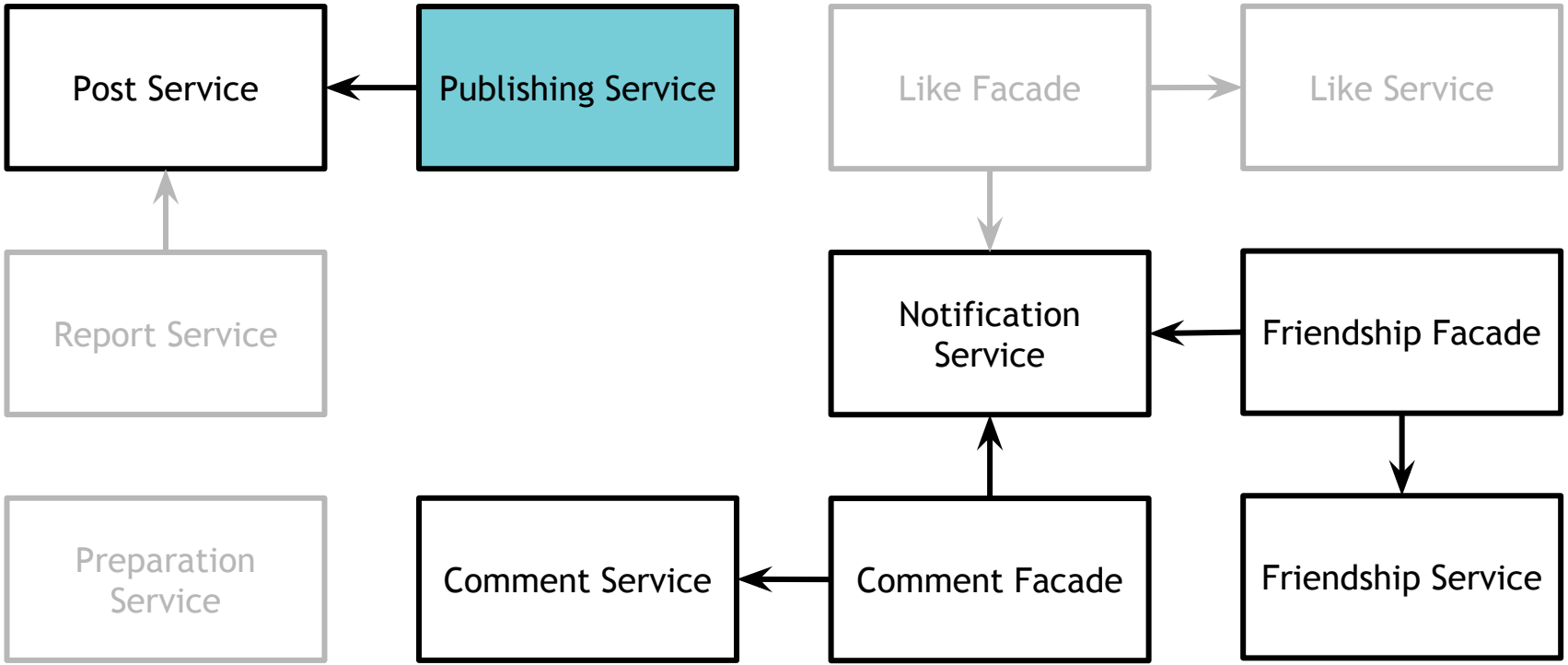
Interface Segregation Principle

Dependency Inversion Principle

Программные сущности (классы, модули, функции) должны быть открыты для расширения, но закрыты для модификации

Бертранд Майер

Архитектура решения



Принцип открытости-закрытости

```
public interface PublishingService {

 Post publishPost(User author, Picture picture);

}
```

Принцип открытости-закрытости

```
public class PublishingServiceImpl implements PublishingService {
 private PostService postService;
 public Post publishPost(User author, Picture picture) {
   Post newPost = new Post();
   newPost.setPicture(picture);
   newPost.setAuthor(author);
   newPost.setPublicationDate(LocalDate.now());
   return postService.save(newPost);
 }
}
```

Принцип открытости-закрытости

```
public class PublishingServiceImpl implements PublishingService {
 private PostService postService;
 public Post publishPost(User author, Picture picture) {
   Post newPost = new Post();
   newPost.setPicture(picture);
   newPost.setAuthor(author);
   newPost.setPublicationDate(LocalDate.now());
   return postService.save(newPost);
 }
}
```

Подготовить пост

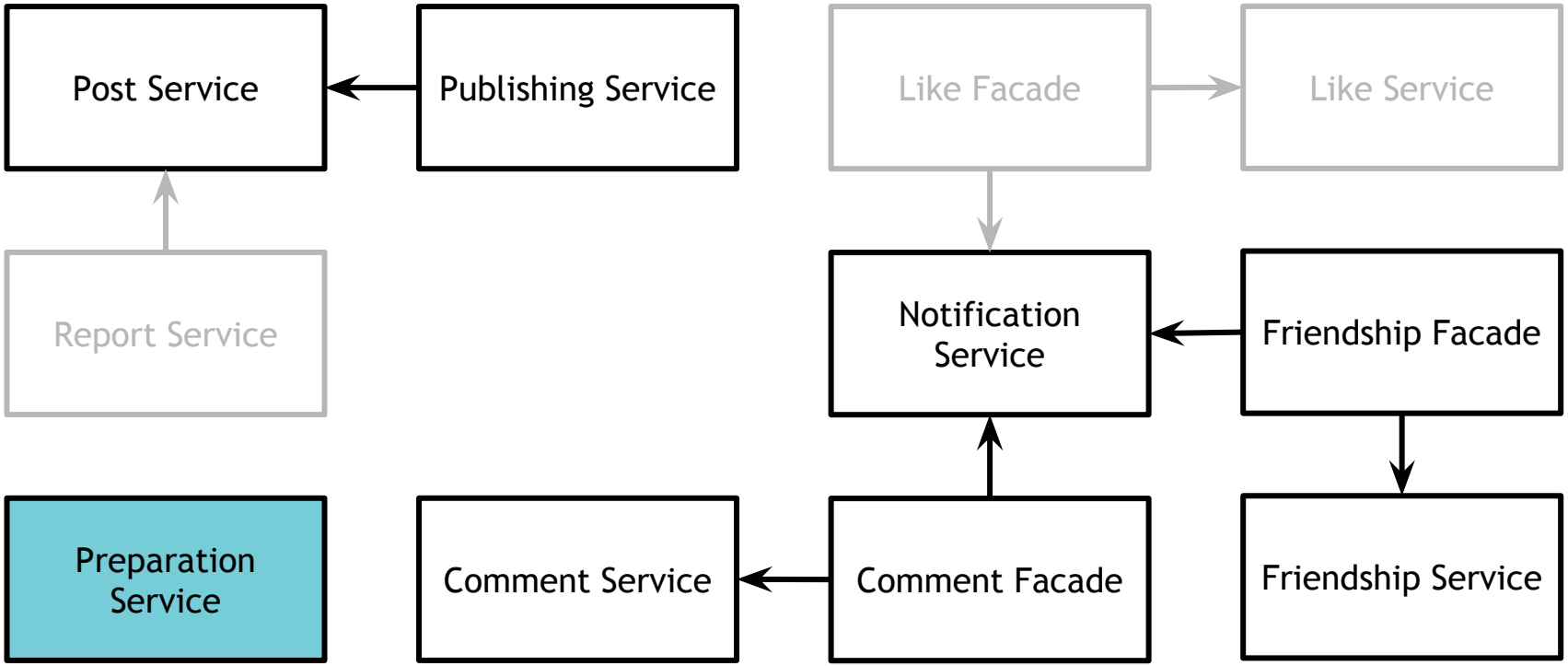
Принцип открытости-закрытости

```
public class PublishingServiceImpl implements PublishingService {
 private PostService postService;
 public Post publishPost(User author, Picture picture) {
  Post newPost = new Post();
  newPost.setPicture(picture);
  newPost.setAuthor(author);
  newPost.setPublicationDate(LocalDate.now());
  return postService.save(newPost);
 }
}
```

Подготовить пост

Сохранить пост

Архитектура решения



Принцип открытости-закрытости

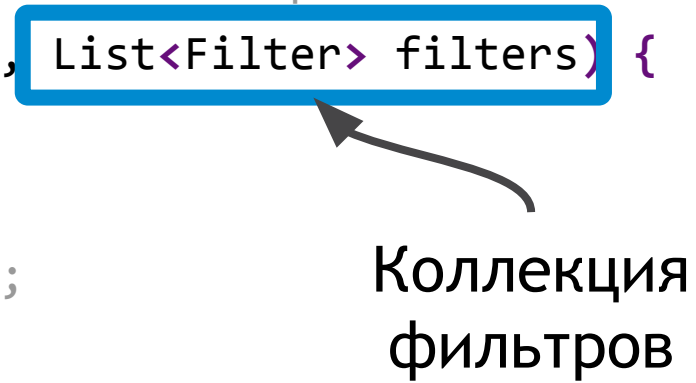
```
public interface PreparationService {

  Picture prepare(Picture source, List<Filter> filters);

}
```

Принцип открытости-закрытости

```
public class PreparationServiceImpl implements PreparationService {
  public Picture prepare(Picture source, List<Filter> filters) {
    Picture prepared = source;
    for (Filter filter : filters) {
      prepared = filter.apply(prepared);
    }
    return prepared;
  }
}
```

Принцип открытости-закрытости

```
public class PreparationServiceImpl implements PreparationService {
  public Picture prepare(Picture source, List<Filter> filters) {
    Picture prepared = source;
    for (Filter filter : filters) {
      prepared = filter.apply(prepared);
    }
    return prepared;
  }
}
```



Принцип открытости-закрытости

```
public class PreparationServiceImpl implements PreparationService {
  public Picture prepare(Picture source, List<Filter> filters) {
    Picture prepared = source;
    for (Filter filter : filters) {
      prepared = filter.apply(prepared);
    }
    return prepared;
  }
}
```

Коллекция фильтров

Последовательно применяем

Принцип открытости-закрытости

```
public interface Filter {
  Picture apply(Picture source);
}
```

Принцип открытости-закрытости

```
public class ClarendonFilter implements Filter {
  public Picture apply(Picture source) {
    // some implementation here
  }
}

public class MoonFilter implements Filter {
  public Picture apply(Picture source) {
    // some implementation here
  }
}
```

Принцип открытости-закрытости

```
public class PreparationDelegate {
  public void runPreparation(User user, Picture picture) {
    List<Filter> filters = Arrays.asList(
        new ClarendonFilter(),
        new MoonFilter()
    );
    Picture preparedPicture = preparationService.prepare(picture, filters);
    publishingService.publishPost(user, preparedPicture);
  }
}
```

Принцип открытости-закрытости

Настраиваем фильтры

```
public class PreparationDelegate {
  public void runPreparation(User user, Picture picture) {
    List<Filter> filters = Arrays.asList(
        new ClarendonFilter(),
        new MoonFilter()
    );
    Picture preparedPicture = preparationService.prepare(picture, filters);
    publishingService.publishPost(user, preparedPicture);
  }
}
```

Принцип открытости-закрытости

```
public class PreparationDelegate {
  public void runPreparation(User user, Picture picture) {
    List<Filter> filters = Arrays.asList(
        new ClarendonFilter(),
        new MoonFilter()
    );
    Picture preparedPicture = preparationService.prepare(picture, filters);
    publishingService.publishPost(user, preparedPicture);
  }
}
```

Настраиваем фильтры

Применяем и сохраняем пост

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

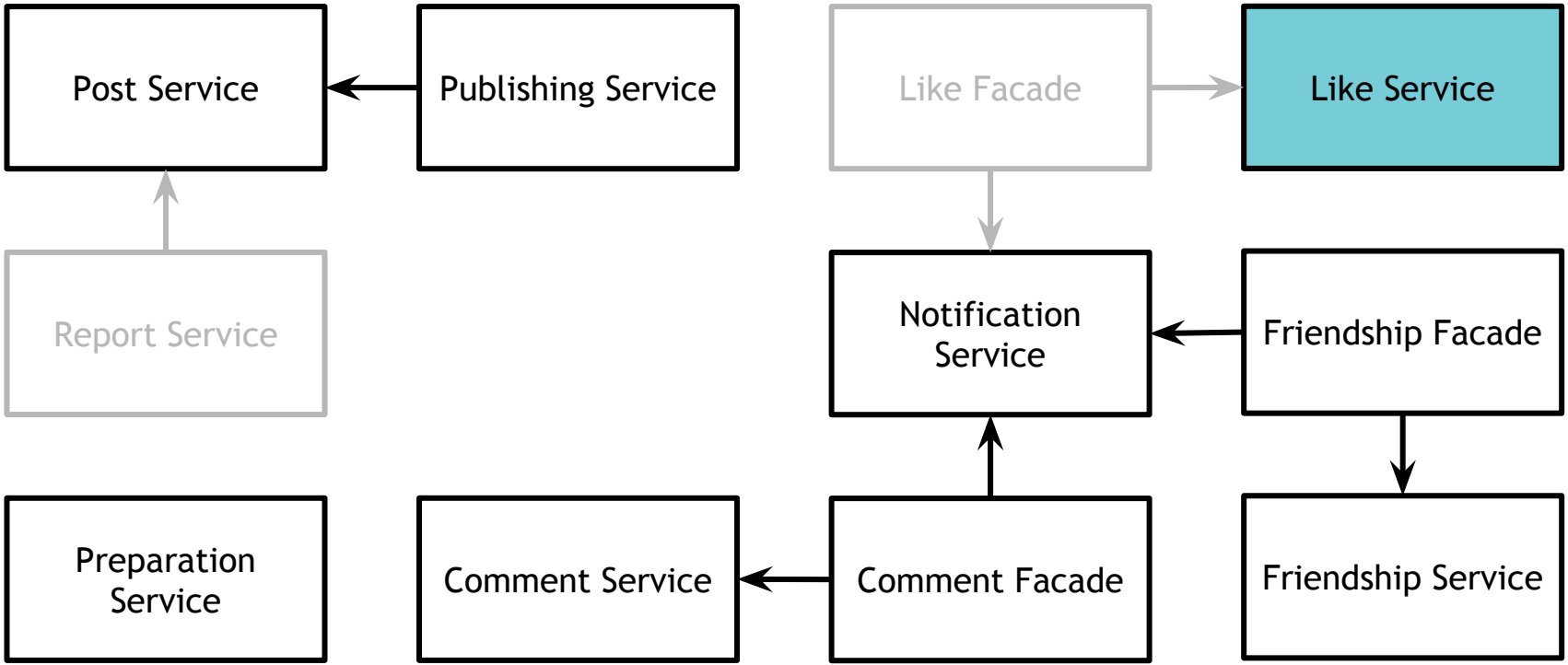
Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом

Барбара Лисков

Принцип подстановки Лисков

```
public class SponsoredPost extends Post {
   private User sponsor;
}
```

Архитектура решения



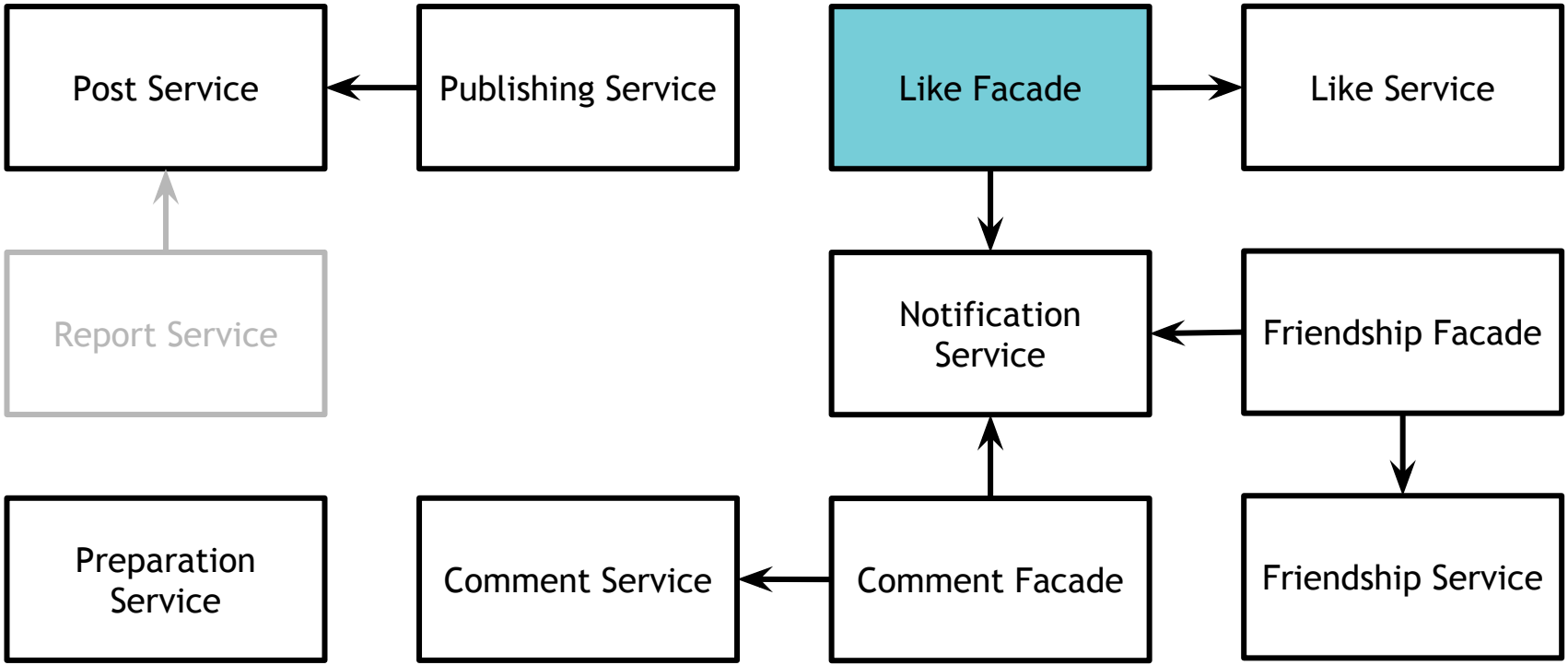
Принцип подстановки Лисков

```
public interface LikeService {
  void likePost(Post post, User whoLikes);
}
```

Принцип подстановки Лисков

```
class LikeServiceImpl implements LikeService {
  private LikeDao likeDao;
  public void likePost(Post post, User whoLikes) {
    likeDao.save(new Like(post, whoLikes));
  }
}
```

Архитектура решения



Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
 private LikeService likeService;
 private NotificationService notificationService;
 public void likePost(Post post, User whoLikes) {
   likeService.likePost(post, whoLikes);
   notificationService.sendNotification(
       new NewLikeNotification(post.getAuthor(), whoLikes)
   );
   // ... еще чуть-чуть кода
 }
```

Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
 private LikeService likeService;
 private NotificationService notificationService;
 public void likePost(Post post, User whoLikes) {
   likeService.likePost(post, whoLikes);
   notificationService.sendNotification(
       new NewLikeNotification(post.getAuthor(), whoLikes)
   );
   // ... еще чуть-чуть кода
 }
```

Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
 private LikeService likeService;
 private NotificationService notificationService;
 public void likePost(Post post, User whoLikes) {
   likeService.likePost(post, whoLikes);
   notificationService.sendNotification(
       new NewLikeNotification(post.getAuthor(), whoLikes)
   );
   // ... еще чуть-чуть кода
 }
```

Сохранить лайк

Уведомить

Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
 public void likePost(Post post, User whoLikes) {
   // ... код с предыдущего слайда здесь
   if (post instanceof SponsoredPost) {
     SponsoredPost sponsoredPost = SponsoredPost.class.cast(post);
     notificationService.sendNotification(
       new NewLikeNotification(sponsoredPost.getSponsor(), whoLikes)
     );
   }
 }
```

Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
 public void likePost(Post post, User whoLikes) {
   // ... код с предыдущего слайда здесь
   if (post instanceof SponsoredPost) {
     SponsoredPost sponsoredPost = SponsoredPost.class.cast(post);
     notificationService.sendNotification(
       new NewLikeNotification(sponsoredPost.getSponsor(), whoLikes)
     );
   }
 }
```

Спонсорский пост

Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
 public void likePost(Post post, User whoLikes) {
   // ... код с предыдущего слайда здесь
   if (post instanceof SponsoredPost) {
     SponsoredPost sponsoredPost = SponsoredPost.class.cast(post);
     notificationService.sendNotification(
       new NewLikeNotification(sponsoredPost.getSponsor(), whoLikes)
     );
   }
 }
}
```

Спонсорский пост

Уведомить спонсора

Принцип подстановки Лисков

```
public interface HasLikeNotificationRecipients {
  List<User> getLikeNotificationRecipients();
}
```

Принцип подстановки Лисков

```
public class Post implements HasLikeNotificationRecipients {
  public List<User> getLikeNotificationRecipients() {
    return Arrays.asList(getAuthor());
  }
}
```

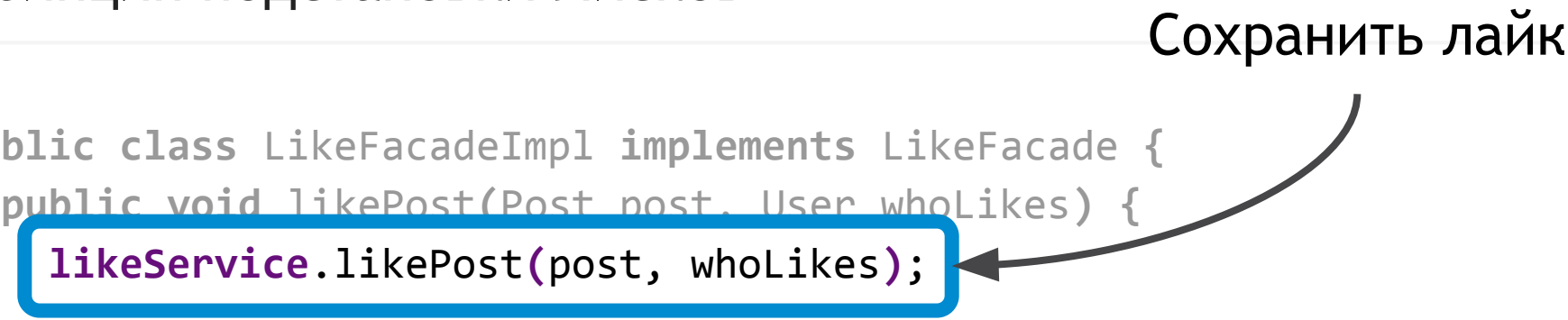
Принцип подстановки Лисков

```
public class Post implements HasLikeNotificationRecipients {
  public List<User> getLikeNotificationRecipients() {
    return Arrays.asList(getAuthor());
  }
}

public class SponsoredPost extends Post {
  public List<User> getLikeNotificationRecipients() {
    return Arrays.asList(getAuthor(), getSponsor());
  }
}
```

Принцип подстановки Лисков

```
public class LikeFacadeImpl implements LikeFacade {
  public void likePost(Post post, User whoLikes) {
    likeService.likePost(post, whoLikes);

    for (User recipient : post.getLikeNotificationRecipients()) {
      notificationService.sendNotification(
          new NewLikeNotification(recipient, whoLikes)
      );
    }
  }
}
```

Принцип подстановки Лисков




```
public class LikeFacadeImpl implements LikeFacade {
  public void likePost(Post post, User whoLikes) {
    likeService.likePost(post, whoLikes);

    for (User recipient : post.getLikeNotificationRecipients()) {
      notificationService.sendNotification(
          new NewLikeNotification(recipient, whoLikes)
      );
    }
  }
}
```

Принцип подстановки Лисков

Сохранить лайк

```
public class LikeFacadeImpl implements LikeFacade {
  public void likePost(Post post, User whoLikes) {
    likeService.likePost(post, whoLikes);

    for (User recipient : post.getLikeNotificationRecipients()) {
      notificationService.sendNotification(
          new NewLikeNotification(recipient, whoLikes)
      );
    }
  }
}
```

Уведомления всем получателям

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

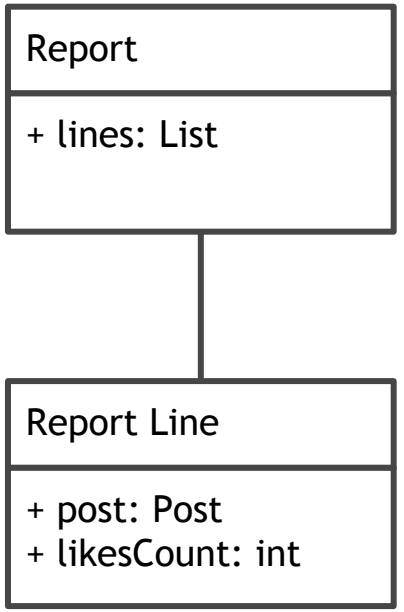
Dependency Inversion Principle

Программные сущности не должны зависеть от методов, которые они не используют

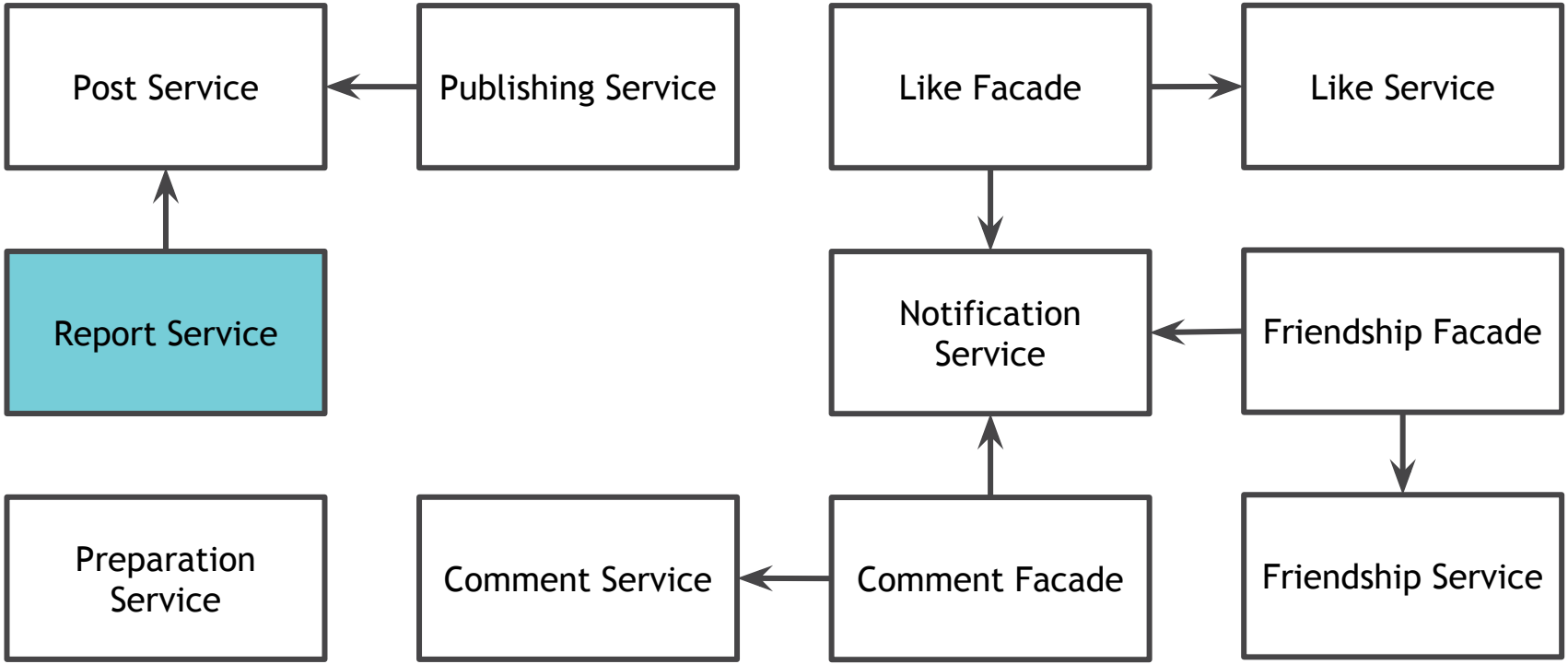
Роберт Мартин

Принцип разделения интерфейсов

```
public class Report {
  private List<ReportLine> lines;
}

public class ReportLine {
  private Post post;
  private int likesCount;
}
```



Архитектура решения



Принцип разделения интерфейсов

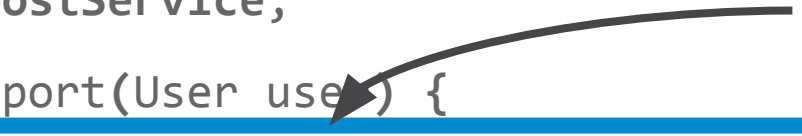
```
public interface ReportService {
   Report buildReport(User user);
}
```

Принцип разделения интерфейсов

```
public class ReportServiceImpl implements ReportService {
 private PostService postService;
 public Report buildReport(User user) {
   List<ReportLine> reportLines = postService.findAll(user).stream()
       .map(post -> new ReportLine(post, post.getLikesCount()))
       .collect(Collectors.toList());
   return new Report(reportLines);
 }
}
```

Принцип разделения интерфейсов

```
public class ReportServiceImpl implements ReportService {
 private PostService postService;
 public Report buildReport(User user) {
   List<ReportLine> reportLines = postService.findAll(user).stream()
       .map(post -> new ReportLine(post, post.getLikesCount()))
       .collect(Collectors.toList());
   return new Report(reportLines);
 }
}
```



Принцип разделения интерфейсов

```
public class ReportServiceImpl implements ReportService {
 private PostService postService;
 public Report buildReport(User user) {
   List<ReportLine> reportLines = postService.findAll(user).stream()
       .map(post -> new ReportLine(post, post.getLikesCount()))
       .collect(Collectors.toList());
   return new Report(reportLines);
 }
}
```

Берем посты

И делаем отчет

Принцип разделения интерфейсов

```
public class ReportServiceImpl implements ReportService {
  public Report buildReport(User user) {
    return isSponsor(user) ?
        buildReport(postService.findSponsored(user)) :
        buildReport(postService.findAll(user));
  }

  private boolean isSponsor(User user) { ... }
  private Report buildReport(List<Post> posts) { ... }
}
```

Принцип разделения интерфейсов

```
public class ReportServiceImpl implements ReportService {
  public Report buildReport(User user) {
    // build a report for an ordinary user
  }

}
```

Принцип разделения интерфейсов

```
public class ReportServiceImpl implements ReportService {
  public Report buildReport(User user) {
    // build a report for an ordinary user
  }

  public Report buildSponsoredReport(User user) {
    // build a report for a sponsor
  }
}
```

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

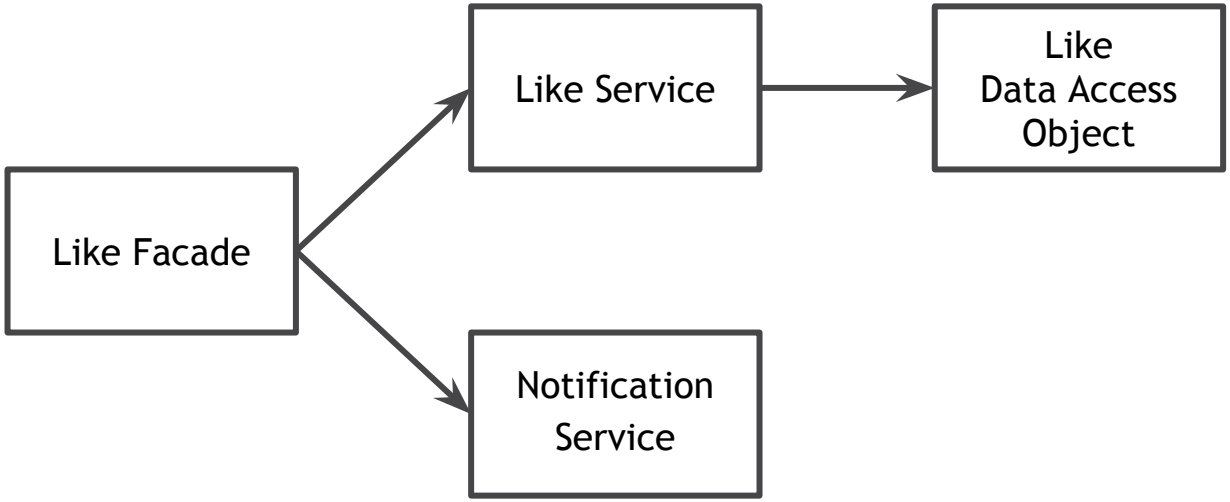
Interface Segregation Principle

Dependency Inversion Principle

Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.

Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций

Принцип инверсии зависимостей



Принцип инверсии зависимостей

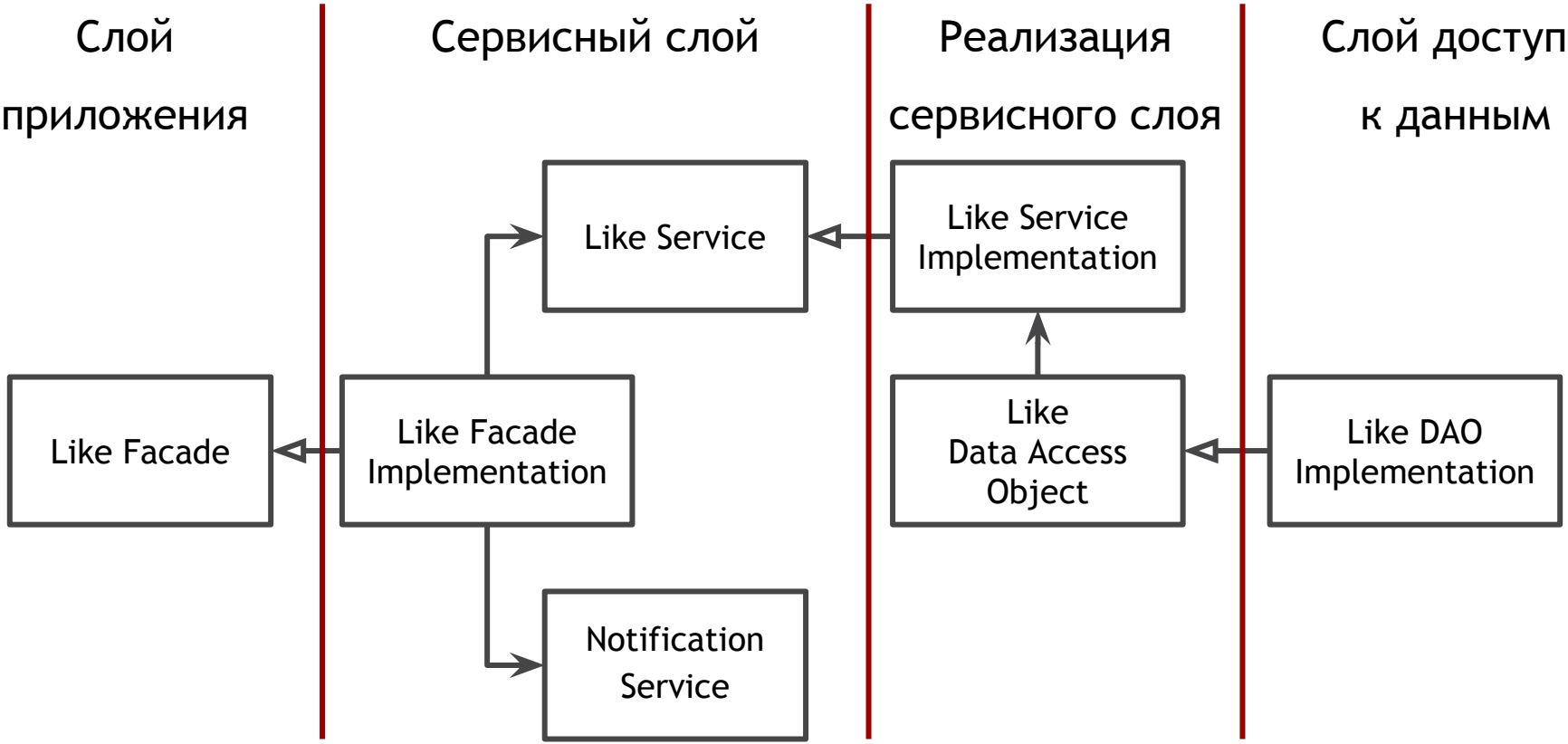
```
public class LikeFacadeImpl implements LikeFacade {
 private final LikeService likeService;
 private final NotificationService notificationService;

 public LikeFacadeImpl() {
   likeService = new LikeServiceImpl();
   notificationService = new NotificationServiceImpl();
 }
}
```

Принцип инверсии зависимостей

```
public class LikeFacadeImpl implements LikeFacade {
 private final LikeService likeService;
 private final NotificationService notificationService;

 public LikeFacadeImpl() {
   likeService = new LikeServiceImpl();
   notificationService = new NotificationServiceImpl();
 }
}
```

Зависимости

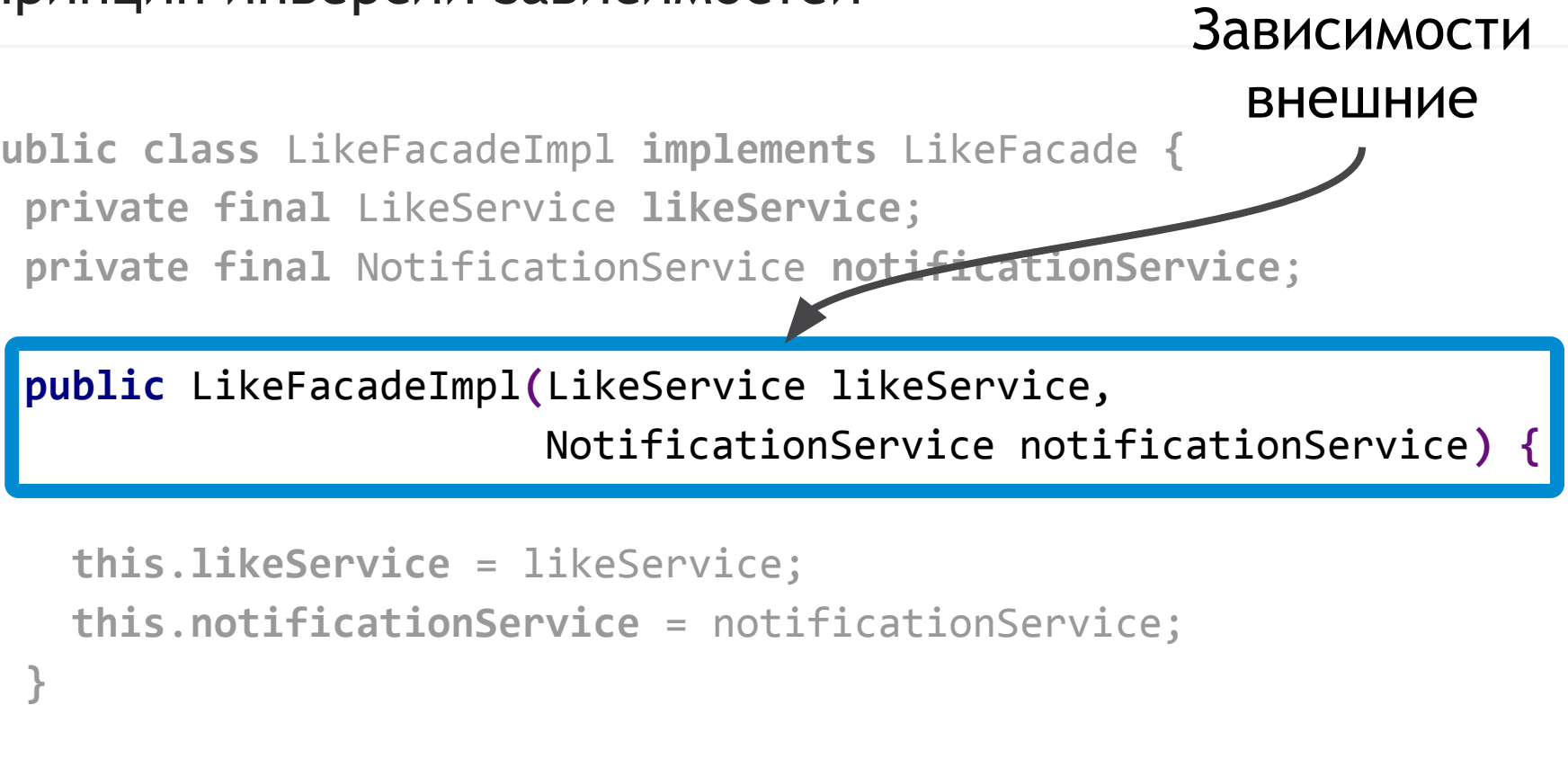
Принцип инверсии зависимостей



Принцип инверсии зависимостей

```
public class LikeFacadeImpl implements LikeFacade {
  private final LikeService likeService;
  private final NotificationService notificationService;

  public LikeFacadeImpl(LikeService likeService,
                        NotificationService notificationService) {

    this.likeService = likeService;
    this.notificationService = notificationService;
  }
}
```

Принцип инверсии зависимостей




```
public class LikeFacadeImpl implements LikeFacade {
  private final LikeService likeService;
  private final NotificationService notificationService;

  public LikeFacadeImpl(LikeService likeService,
                        NotificationService notificationService) {

    this.likeService = likeService;
    this.notificationService = notificationService;
  }
}
```

SOLID

- **Принцип единственной ответственности —** у класса должна быть одна обязанность и причина для изменений.
- **Принцип открытости-закрытости** — для расширения нужно писать новый код, а не редактировать старый.
- **Принцип подстановки Лисков** — добавление дочерних классов не требует полной переработки приложения.
- **Принцип разделения интерфейсов** — каждому клиенту свой интерфейс, клиенты не зависят друг от друга.
- **Принцип инверсии зависимостей** — реализация зависит от абстракции.

Заключение

- Чаще приходится код читать, чем писать
- Чистый код
- SOLID
- Практика, больше практики!

<epam>

Конец?

aabarmin

AlexBarmin

ABarmin