



Desenvolvimento Back-End

2025

Rio de Janeiro – RJ

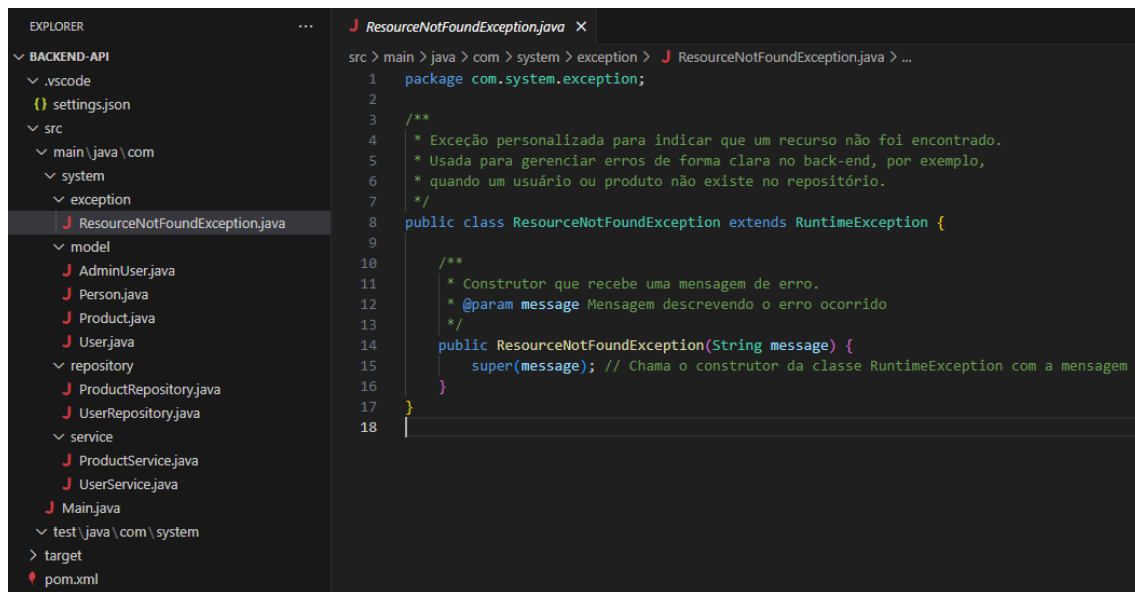
Higor Cosme de Jesus

Estrutura do Projeto:

```

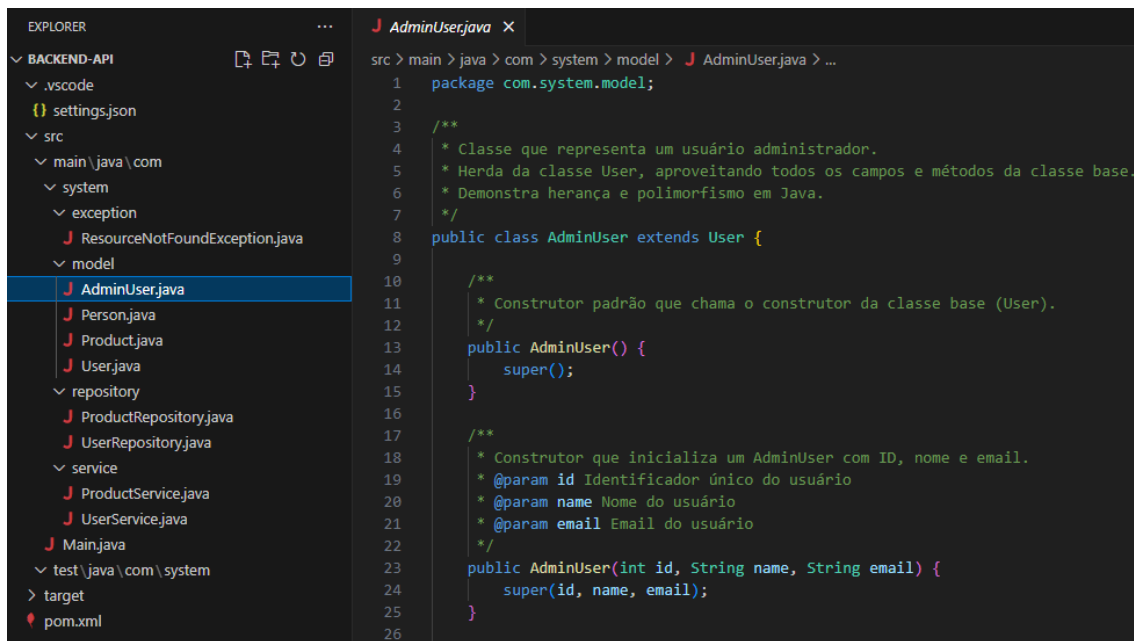
  ✓ BACKEND-API
  ✓ .vscode
    {} settings.json
  ✓ src
    ✓ main\java\com
      ✓ system
        ✓ exception
          J ResourceNotFoundException.java
        ✓ model
          J AdminUser.java
          J Person.java
          J Product.java
          J User.java
        ✓ repository
          J ProductRepository.java
          J UserRepository.java
        ✓ service
          J ProductService.java
          J UserService.java
      J Main.java
    ✓ test\java\com\system
  > target
  pom.xml
```

ResourceNotFoundException.java



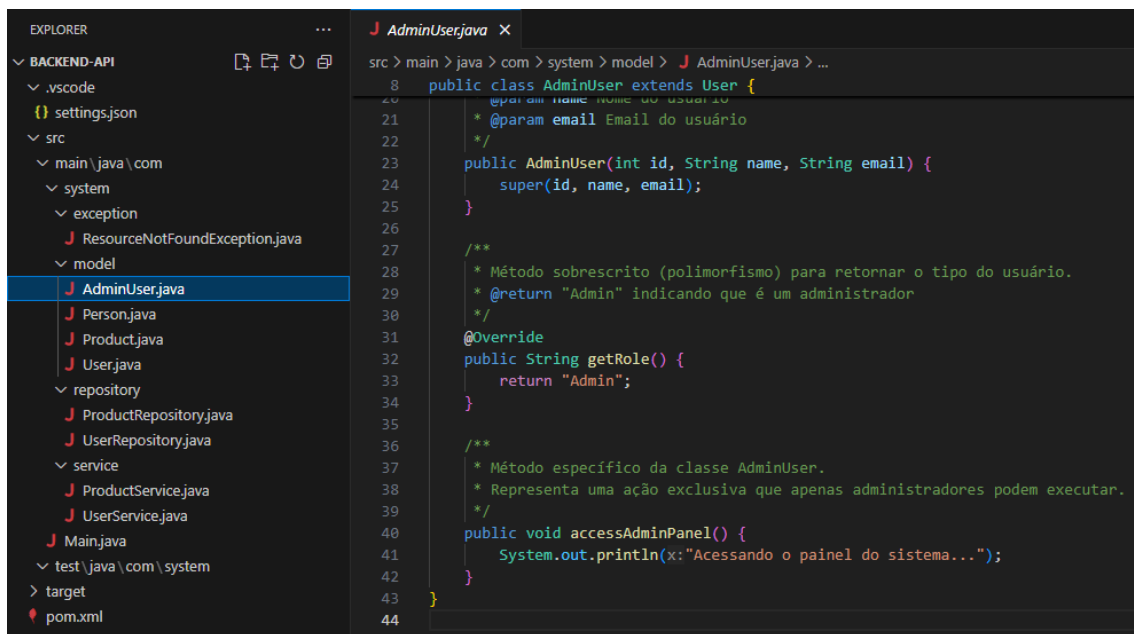
```
1 package com.system.exception;
2
3 /**
4  * Exceção personalizada para indicar que um recurso não foi encontrado.
5  * Usada para gerenciar erros de forma clara no back-end, por exemplo,
6  * quando um usuário ou produto não existe no repositório.
7  */
8 public class ResourceNotFoundException extends RuntimeException {
9
10     /**
11      * Construtor que recebe uma mensagem de erro.
12      * @param message Mensagem descrevendo o erro ocorrido
13      */
14     public ResourceNotFoundException(String message) {
15         super(message); // Chama o construtor da classe RuntimeException com a mensagem
16     }
17 }
18
```

AdminUser.java



```
EXPLORER
  BACKEND-API
    .vscode
    settings.json
    src
      main\java\com
        system
          exception
            ResourceNotFoundException.java
          model
            AdminUser.java
            Person.java
            Product.java
            User.java
        repository
            ProductRepository.java
            UserRepository.java
        service
            ProductService.java
            UserService.java
            Main.java
      test\java\com\system
    target
    pom.xml

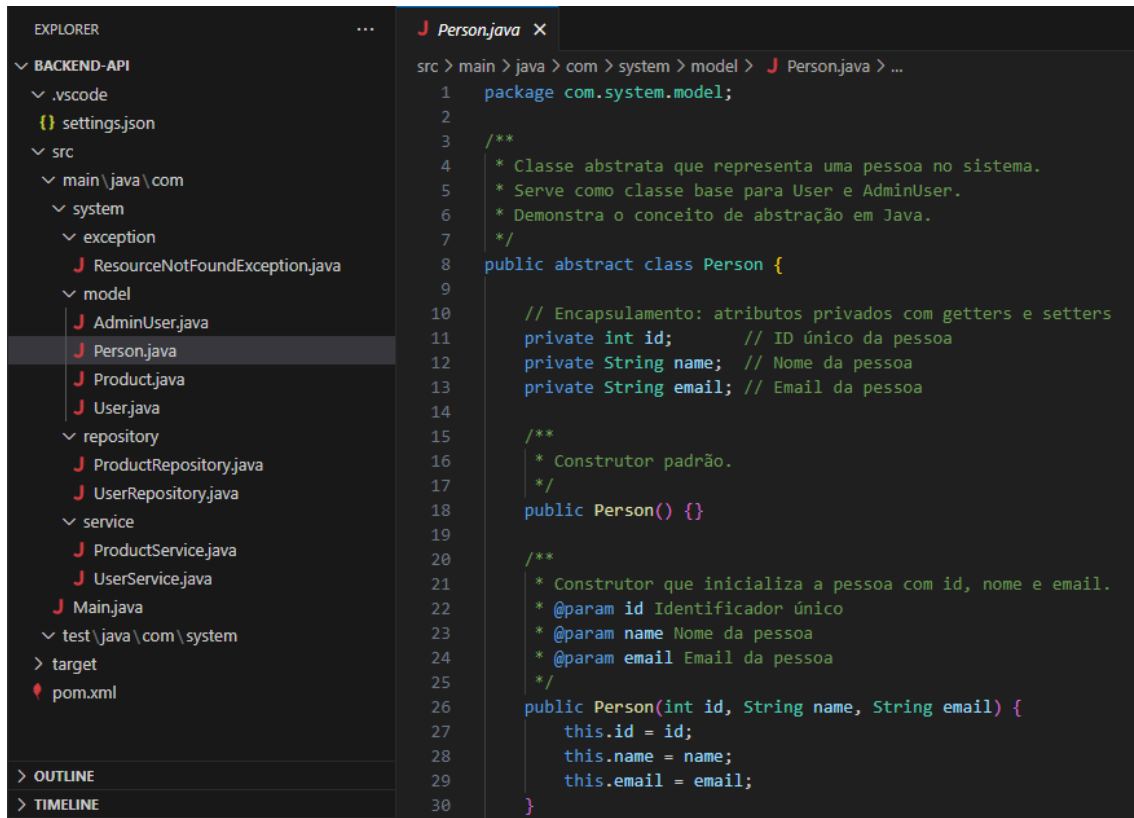
AdminUser.java
src > main > java > com > system > model > AdminUser.java > ...
1 package com.system.model;
2
3 /**
4  * Classe que representa um usuário administrador.
5  * Herda da classe User, aproveitando todos os campos e métodos da classe base.
6  * Demonstra herança e polimorfismo em Java.
7  */
8 public class AdminUser extends User {
9
10     /**
11      * Construtor padrão que chama o construtor da classe base (User).
12      */
13     public AdminUser() {
14         super();
15     }
16
17     /**
18      * Construtor que inicializa um AdminUser com ID, nome e email.
19      * @param id Identificador único do usuário
20      * @param name Nome do usuário
21      * @param email Email do usuário
22      */
23     public AdminUser(int id, String name, String email) {
24         super(id, name, email);
25     }
26 }
```



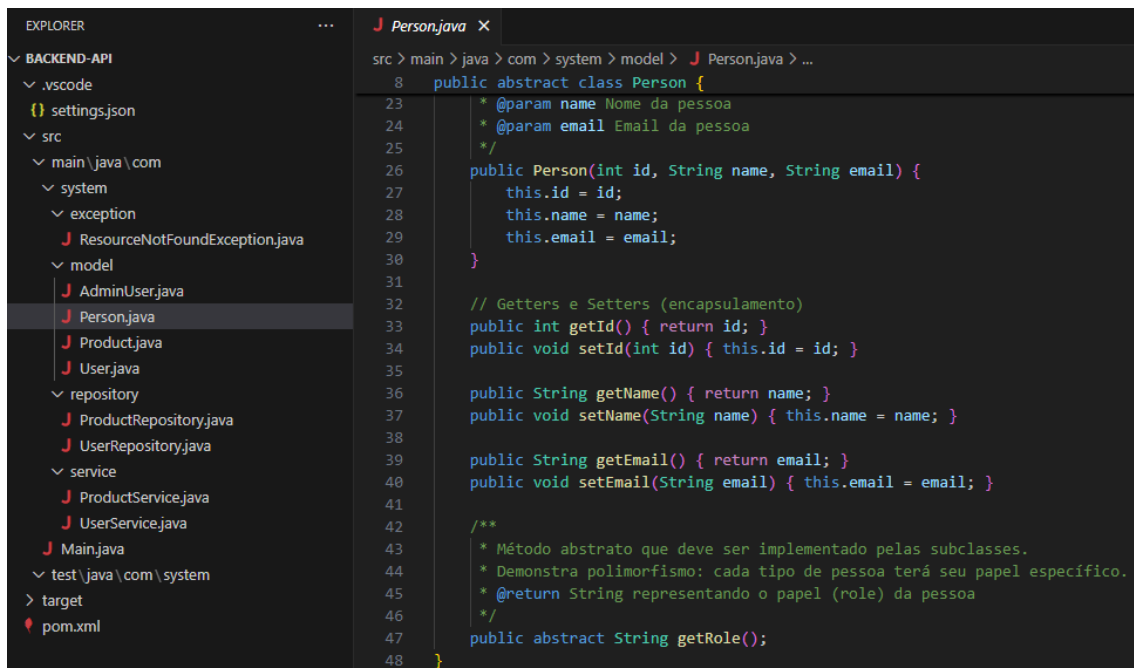
```
EXPLORER
  BACKEND-API
    .vscode
    settings.json
    src
      main\java\com
        system
          exception
            ResourceNotFoundException.java
          model
            AdminUser.java
            Person.java
            Product.java
            User.java
        repository
            ProductRepository.java
            UserRepository.java
        service
            ProductService.java
            UserService.java
            Main.java
      test\java\com\system
    target
    pom.xml

AdminUser.java
src > main > java > com > system > model > AdminUser.java > ...
8 public class AdminUser extends User {
20     @param name Nome do usuário
21     * @param email Email do usuário
22     */
23     public AdminUser(int id, String name, String email) {
24         super(id, name, email);
25     }
26
27     /**
28      * Método sobrescrito (polimorfismo) para retornar o tipo do usuário.
29      * @return "Admin" indicando que é um administrador
30      */
31     @Override
32     public String getRole() {
33         return "Admin";
34     }
35
36     /**
37      * Método específico da classe AdminUser.
38      * Representa uma ação exclusiva que apenas administradores podem executar.
39      */
40     public void accessAdminPanel() {
41         System.out.println("Acessando o painel do sistema...");
42     }
43 }
44 }
```

Person.java

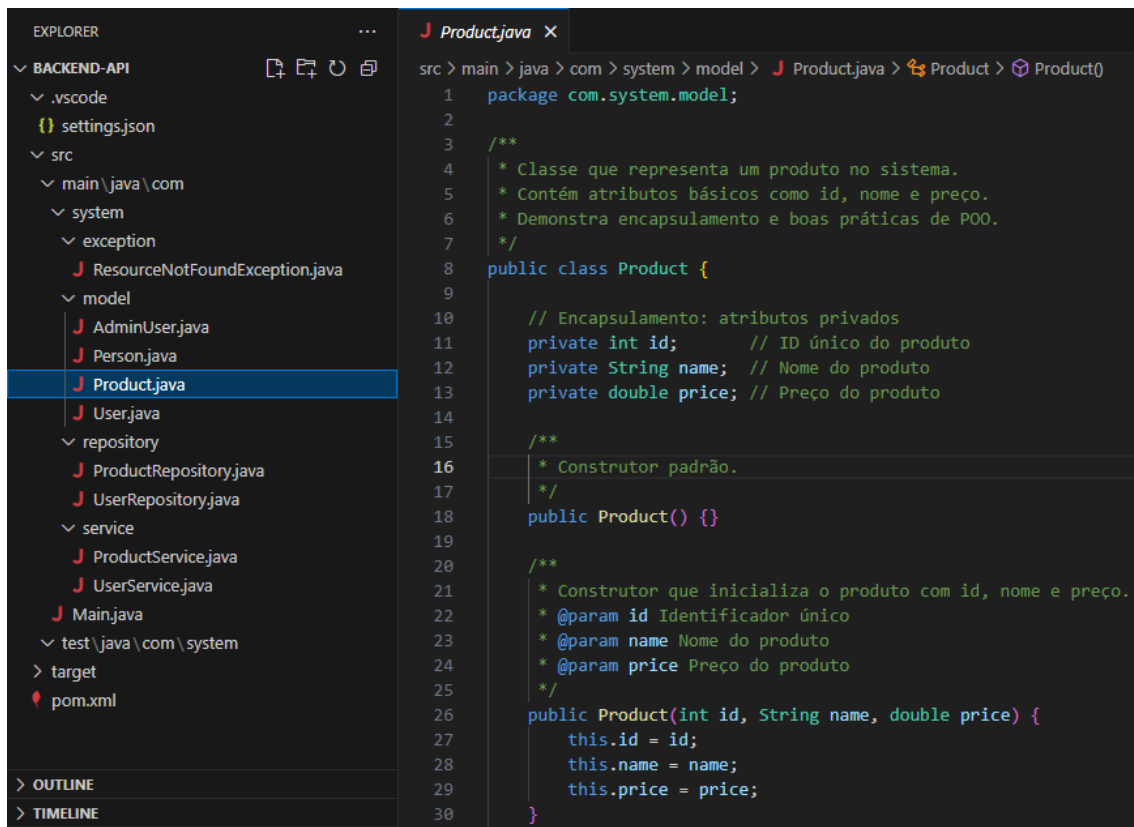


```
src > main > java > com > system > model > J Person.java > ...
1  package com.system.model;
2
3  /**
4   * Classe abstrata que representa uma pessoa no sistema.
5   * Serve como classe base para User e AdminUser.
6   * Demonstra o conceito de abstração em Java.
7   */
8  public abstract class Person {
9
10     // Encapsulamento: atributos privados com getters e setters
11     private int id;        // ID único da pessoa
12     private String name;   // Nome da pessoa
13     private String email;  // Email da pessoa
14
15     /**
16     * Construtor padrão.
17     */
18     public Person() {}
19
20     /**
21     * Construtor que inicializa a pessoa com id, nome e email.
22     * @param id Identificador único
23     * @param name Nome da pessoa
24     * @param email Email da pessoa
25     */
26     public Person(int id, String name, String email) {
27         this.id = id;
28         this.name = name;
29         this.email = email;
30     }
```

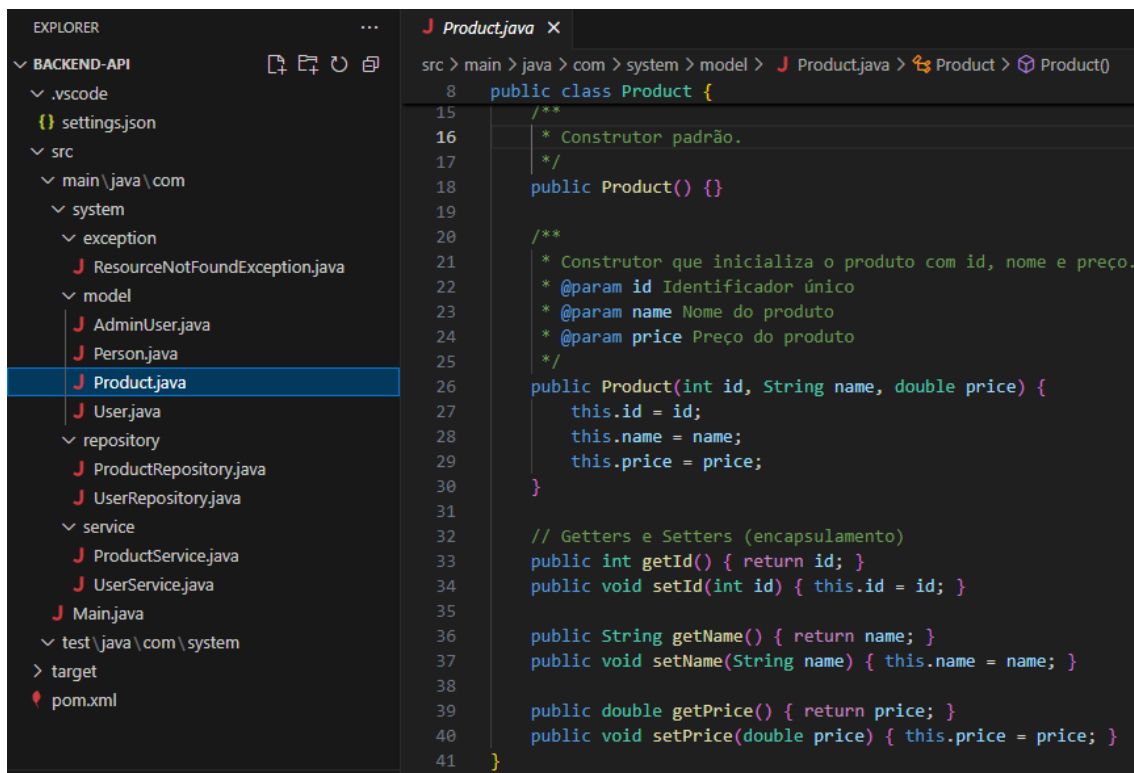


```
src > main > java > com > system > model > J Person.java > ...
8  public abstract class Person {
23     * @param name Nome da pessoa
24     * @param email Email da pessoa
25     */
26     public Person(int id, String name, String email) {
27         this.id = id;
28         this.name = name;
29         this.email = email;
30     }
31
32     // Getters e Setters (encapsulamento)
33     public int getId() { return id; }
34     public void setId(int id) { this.id = id; }
35
36     public String getName() { return name; }
37     public void setName(String name) { this.name = name; }
38
39     public String getEmail() { return email; }
40     public void setEmail(String email) { this.email = email; }
41
42     /**
43     * Método abstrato que deve ser implementado pelas subclasses.
44     * Demonstra polimorfismo: cada tipo de pessoa terá seu papel específico.
45     * @return String representando o papel (role) da pessoa
46     */
47     public abstract String getRole();
48 }
```

Product.java

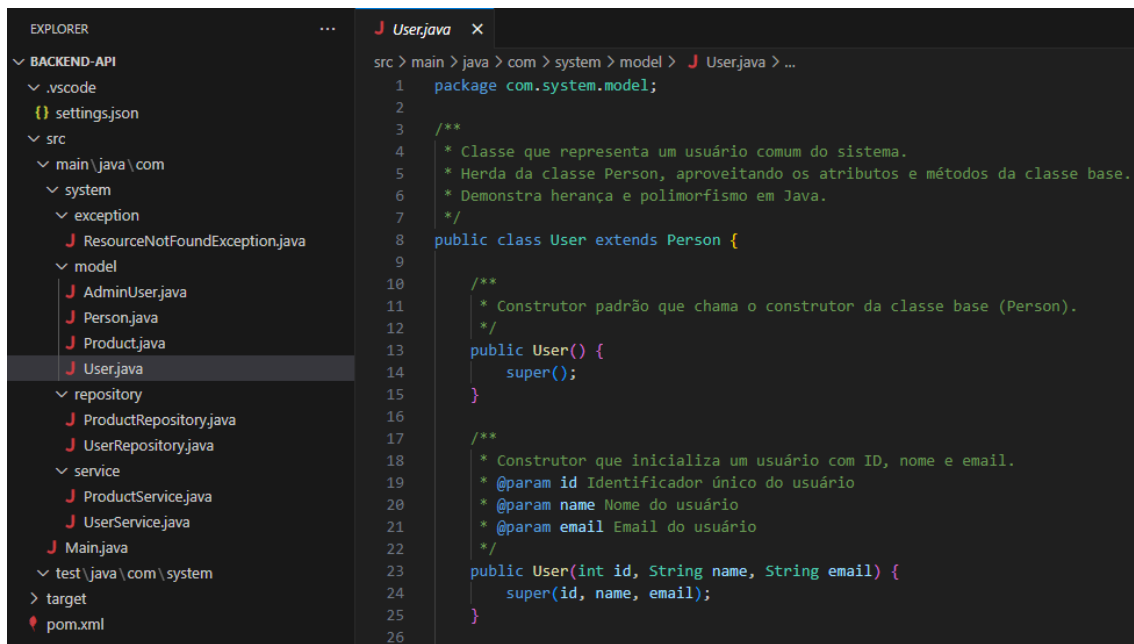


```
src > main > java > com > system > model > Product.java > Product > Product()
1 package com.system.model;
2
3 /**
4  * Classe que representa um produto no sistema.
5  * Contém atributos básicos como id, nome e preço.
6  * Demonstra encapsulamento e boas práticas de POO.
7  */
8 public class Product {
9
10     // Encapsulamento: atributos privados
11     private int id; // ID único do produto
12     private String name; // Nome do produto
13     private double price; // Preço do produto
14
15     /**
16     * Construtor padrão.
17     */
18     public Product() {}
19
20     /**
21     * Construtor que inicializa o produto com id, nome e preço.
22     * @param id Identificador único
23     * @param name Nome do produto
24     * @param price Preço do produto
25     */
26     public Product(int id, String name, double price) {
27         this.id = id;
28         this.name = name;
29         this.price = price;
30     }
}
```

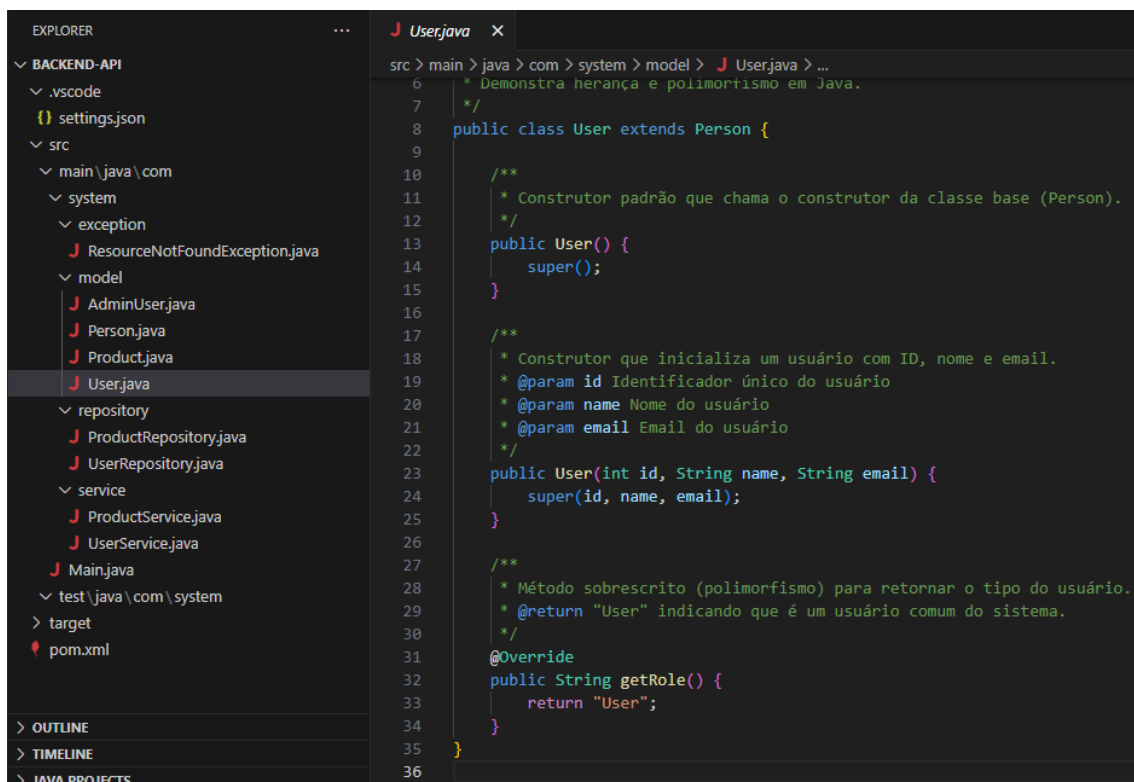


```
src > main > java > com > system > model > Product.java > Product > Product()
8 public class Product {
15     /**
16     * Construtor padrão.
17     */
18     public Product() {}
19
20     /**
21     * Construtor que inicializa o produto com id, nome e preço.
22     * @param id Identificador único
23     * @param name Nome do produto
24     * @param price Preço do produto
25     */
26     public Product(int id, String name, double price) {
27         this.id = id;
28         this.name = name;
29         this.price = price;
30     }
31
32     // Getters e Setters (encapsulamento)
33     public int getId() { return id; }
34     public void setId(int id) { this.id = id; }
35
36     public String getName() { return name; }
37     public void setName(String name) { this.name = name; }
38
39     public double getPrice() { return price; }
40     public void setPrice(double price) { this.price = price; }
41 }
}
```

User.java

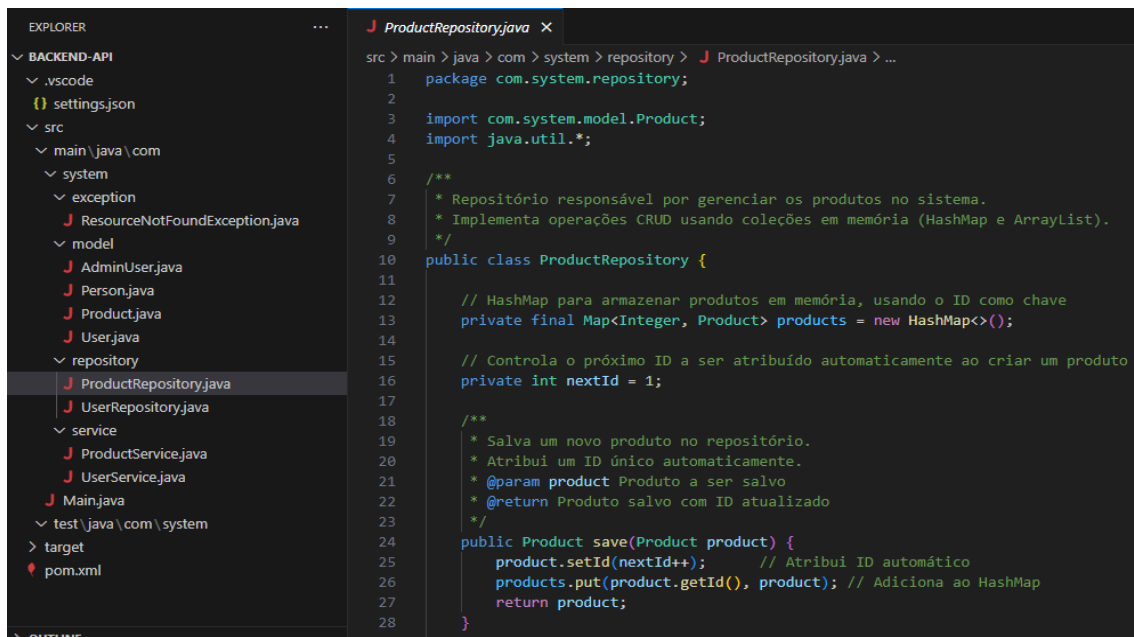


```
src > main > java > com > system > model > J User.java > ...
1  package com.system.model;
2
3  /**
4   * Classe que representa um usuário comum do sistema.
5   * Herda da classe Person, aproveitando os atributos e métodos da classe base.
6   * Demonstra herança e polimorfismo em Java.
7   */
8  public class User extends Person {
9
10     /**
11      * Construtor padrão que chama o construtor da classe base (Person).
12      */
13     public User() {
14         super();
15     }
16
17     /**
18      * Construtor que inicializa um usuário com ID, nome e email.
19      * @param id Identificador único do usuário
20      * @param name Nome do usuário
21      * @param email Email do usuário
22      */
23     public User(int id, String name, String email) {
24         super(id, name, email);
25     }
26 }
```

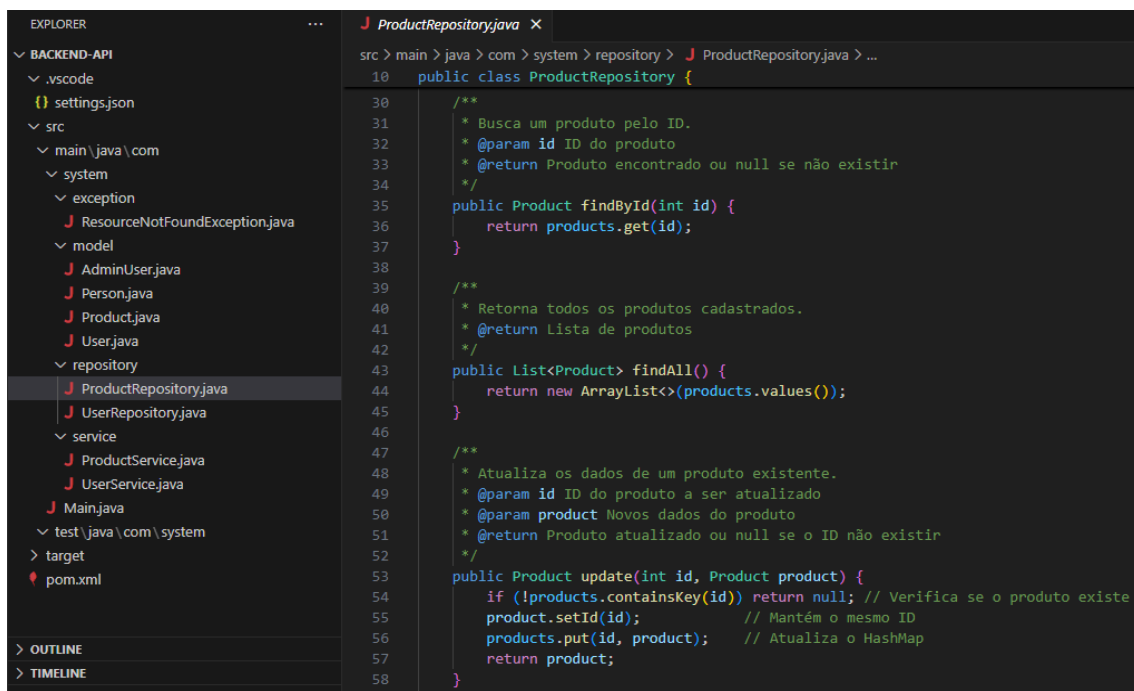


```
src > main > java > com > system > model > J User.java > ...
6  * Demonstra herança e polimorfismo em Java.
7  */
8  public class User extends Person {
9
10     /**
11      * Construtor padrão que chama o construtor da classe base (Person).
12      */
13     public User() {
14         super();
15     }
16
17     /**
18      * Construtor que inicializa um usuário com ID, nome e email.
19      * @param id Identificador único do usuário
20      * @param name Nome do usuário
21      * @param email Email do usuário
22      */
23     public User(int id, String name, String email) {
24         super(id, name, email);
25     }
26
27     /**
28      * Método sobrescrito (polimorfismo) para retornar o tipo do usuário.
29      * @return "User" indicando que é um usuário comum do sistema.
30      */
31     @Override
32     public String getRole() {
33         return "User";
34     }
35 }
36 }
```

ProductRepository.java



```
src > main > java > com > system > repository > J ProductRepository.java > ...
1 package com.system.repository;
2
3 import com.system.model.Product;
4 import java.util.*;
5
6 /**
7  * Repositório responsável por gerenciar os produtos no sistema.
8  * Implementa operações CRUD usando coleções em memória (HashMap e ArrayList).
9  */
10 public class ProductRepository {
11
12     // HashMap para armazenar produtos em memória, usando o ID como chave
13     private final Map<Integer, Product> products = new HashMap<>();
14
15     // Controla o próximo ID a ser atribuído automaticamente ao criar um produto
16     private int nextId = 1;
17
18     /**
19     * Salva um novo produto no repositório.
20     * Atribui um ID único automaticamente.
21     * @param product Produto a ser salvo
22     * @return Produto salvo com ID atualizado
23     */
24     public Product save(Product product) {
25         product.setId(nextId++); // Atribui ID automático
26         products.put(product.getId(), product); // Adiciona ao HashMap
27         return product;
28     }
29 }
```



```
src > main > java > com > system > repository > J ProductRepository.java > ...
10 public class ProductRepository {
30
31     /**
32     * Busca um produto pelo ID.
33     * @param id ID do produto
34     * @return Produto encontrado ou null se não existir
35     */
36     public Product findById(int id) {
37         return products.get(id);
38     }
39
40     /**
41     * Retorna todos os produtos cadastrados.
42     * @return Lista de produtos
43     */
44     public List<Product> findAll() {
45         return new ArrayList<>(products.values());
46     }
47
48     /**
49     * Atualiza os dados de um produto existente.
50     * @param id ID do produto a ser atualizado
51     * @param product Novos dados do produto
52     * @return Produto atualizado ou null se o ID não existir
53     */
54     public Product update(int id, Product product) {
55         if (!products.containsKey(id)) return null; // Verifica se o produto existe
56         product.setId(id); // Mantém o mesmo ID
57         products.put(id, product); // Atualiza o HashMap
58         return product;
59     }
60 }
```



```
src > main > java > com > system > repository > J ProductRepository.java > ...
10 public class ProductRepository {
40     /**
41      * Retorna todos os produtos cadastrados.
42      * @return Lista de produtos
43      */
44     public List<Product> findAll() {
45         return new ArrayList<>(products.values());
46     }
47
48     /**
49      * Atualiza os dados de um produto existente.
50      * @param id ID do produto a ser atualizado
51      * @param product Novos dados do produto
52      * @return Produto atualizado ou null se o ID não existir
53      */
54     public Product update(int id, Product product) {
55         if (!products.containsKey(id)) return null; // Verifica se o produto existe
56         product.setId(id); // Mantém o mesmo ID
57         products.put(id, product); // Atualiza o HashMap
58         return product;
59     }
60
61     /**
62      * Deleta um produto pelo ID.
63      * @param id ID do produto a ser removido
64      * @return true se removido com sucesso, false caso não exista
65      */
66     public boolean delete(int id) {
67         return products.remove(id) != null;
68     }
69 }
```

UserRepository.java

```
src > main > java > com > system > repository > J UserRepository.java > ...
1 package com.system.repository;
2
3 import com.system.model.User;
4 import java.util.*;
5
6 /**
7  * Repositório responsável por gerenciar os usuários no sistema.
8  * Implementa operações CRUD usando coleções em memória (HashMap e ArrayList).
9  */
10 public class UserRepository {
11
12     // HashMap para armazenar usuários em memória, usando o ID como chave
13     private final Map<Integer, User> users = new HashMap<>();
14
15     // Controla o próximo ID a ser atribuído automaticamente ao criar um usuário
16     private int nextId = 1;
17
18     /**
19      * Salva um novo usuário no repositório.
20      * Atribui um ID único automaticamente.
21      * @param user Usuário a ser salvo
22      * @return Usuário salvo com ID atualizado
23      */
24     public User save(User user) {
25         user.setId(nextId++); // Atribui ID automático
26         users.put(user.getId(), user); // Adiciona ao HashMap
27         return user;
28     }
29
30     /**
31      * Busca um usuário pelo ID.
32      * @param id ID do usuário
33      */
34 }
```

EXPLORER

...

UserRepository.java X

src > main > java > com > system > repository > J UserRepository.java > ...

10 public class UserRepository {

31 * Busca um usuário pelo ID.

32 * @param id ID do usuário

33 * @return Usuário encontrado ou null se não existir

34 */

35 public User findById(int id) {

36 return users.get(id);

37 }

38

39 /**

40 * Retorna todos os usuários cadastrados.

41 * @return Lista de usuários

42 */

43 public List<User> findAll() {

44 return new ArrayList<>(users.values());

45 }

46

47 /**

48 * Atualiza os dados de um usuário existente.

49 * @param id ID do usuário a ser atualizado

50 * @param user Novos dados do usuário

51 * @return Usuário atualizado ou null se o ID não existir

52 */

53 public User update(int id, User user) {

54 if (!users.containsKey(id)) return null; // Verifica se o usuário existe

55 user.setId(id); // Mantém o mesmo ID

56 users.put(id, user); // Atualiza o HashMap

57 return user;

58 }

EXPLORER

...

UserRepository.java X

src > main > java > com > system > repository > J UserRepository.java > ...

10 public class UserRepository {

43 public List<User> findAll() {

45 }

46

47 /**

48 * Atualiza os dados de um usuário existente.

49 * @param id ID do usuário a ser atualizado

50 * @param user Novos dados do usuário

51 * @return Usuário atualizado ou null se o ID não existir

52 */

53 public User update(int id, User user) {

54 if (!users.containsKey(id)) return null; // Verifica se o usuário existe

55 user.setId(id); // Mantém o mesmo ID

56 users.put(id, user); // Atualiza o HashMap

57 return user;

58 }

59

60 /**

61 * Deleta um usuário pelo ID.

62 * @param id ID do usuário a ser removido

63 * @return true se removido com sucesso, false caso não exista

64 */

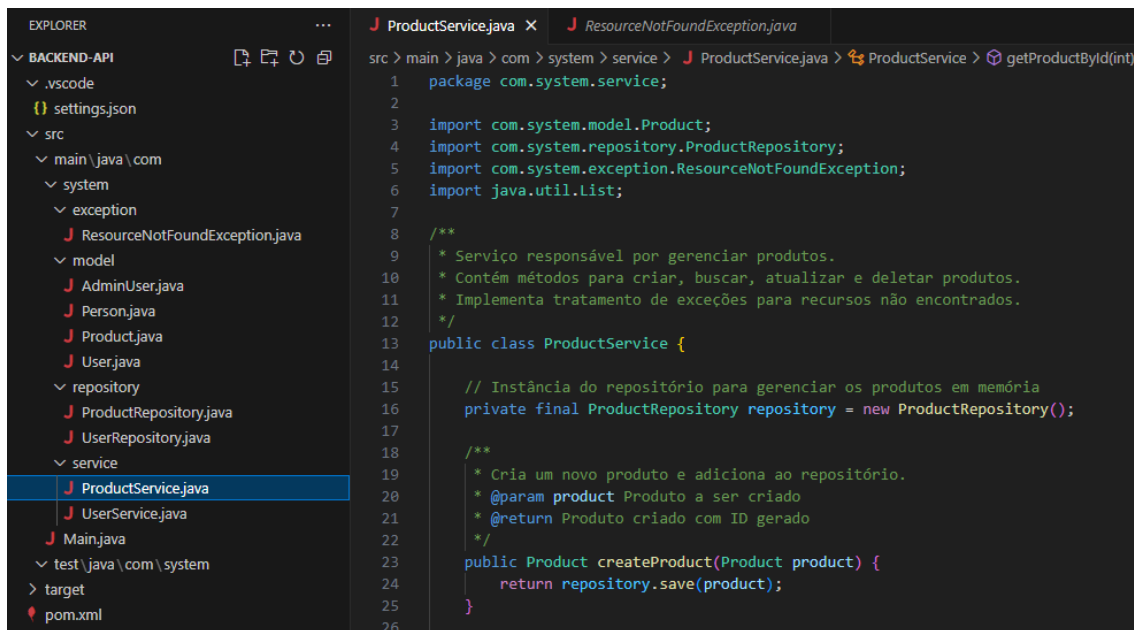
65 public boolean delete(int id) {

66 return users.remove(id) != null;

67 }

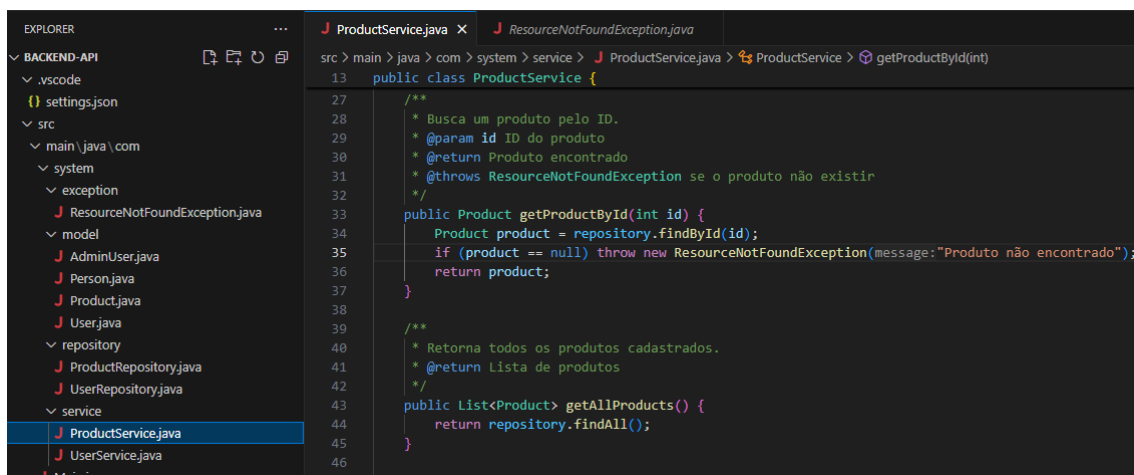
68 }

ProductService.java



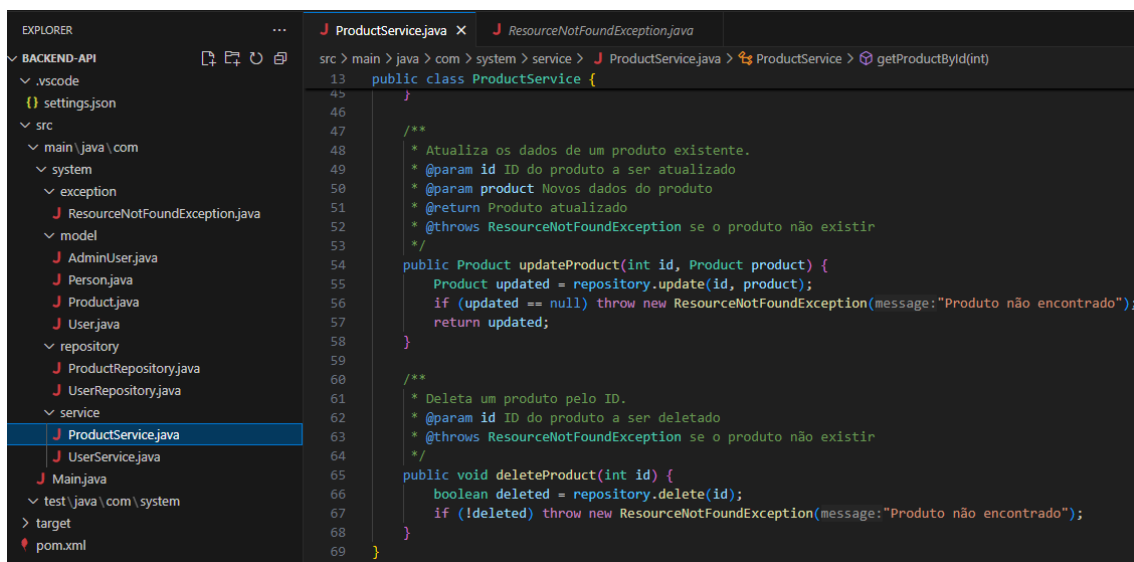
This screenshot shows the initial state of the `ProductService.java` file. The Explorer on the left shows the project structure with `ProductService.java` selected. The editor displays the package, imports, and the start of the `ProductService` class, including a comment about its responsibilities and the initialization of the `repository`.

```
1 package com.system.service;
2
3 import com.system.model.Product;
4 import com.system.repository.ProductRepository;
5 import com.system.exception.ResourceNotFoundException;
6 import java.util.List;
7
8 /**
9  * Serviço responsável por gerenciar produtos.
10  * Contém métodos para criar, buscar, atualizar e deletar produtos.
11  * Implementa tratamento de exceções para recursos não encontrados.
12  */
13 public class ProductService {
14
15     // Instância do repositório para gerenciar os produtos em memória
16     private final ProductRepository repository = new ProductRepository();
17
18     /**
19      * Cria um novo produto e adiciona ao repositório.
20      * @param product Produto a ser criado
21      * @return Produto criado com ID gerado
22      */
23     public Product createProduct(Product product) {
24         return repository.save(product);
25     }
26 }
```



This screenshot shows the addition of the `getProductById` method to `ProductService.java`. The Explorer and breadcrumb are the same as the previous screenshot. The editor now includes the `getProductById` method, which uses `repository.findById` and throws a `ResourceNotFoundException` if the product is not found.

```
13 public class ProductService {
27
28     /**
29      * Busca um produto pelo ID.
30      * @param id ID do produto
31      * @return Produto encontrado
32      * @throws ResourceNotFoundException se o produto não existir
33      */
34     public Product getProductById(int id) {
35         Product product = repository.findById(id);
36         if (product == null) throw new ResourceNotFoundException(message:"Produto não encontrado");
37         return product;
38     }
39
40     /**
41      * Retorna todos os produtos cadastrados.
42      * @return Lista de produtos
43      */
44     public List<Product> getAllProducts() {
45         return repository.findAll();
46     }
47 }
```



This screenshot shows the completion of `ProductService.java` with the addition of `updateProduct` and `deleteProduct` methods. The Explorer and breadcrumb are consistent. The editor now includes both methods, each using the repository and throwing exceptions for non-existent products.

```
13 public class ProductService {
45
46     /**
47      * Atualiza os dados de um produto existente.
48      * @param id ID do produto a ser atualizado
49      * @param product Novos dados do produto
50      * @return Produto atualizado
51      * @throws ResourceNotFoundException se o produto não existir
52      */
53     public Product updateProduct(int id, Product product) {
54         Product updated = repository.update(id, product);
55         if (updated == null) throw new ResourceNotFoundException(message:"Produto não encontrado");
56         return updated;
57     }
58
59     /**
60      * Deleta um produto pelo ID.
61      * @param id ID do produto a ser deletado
62      * @throws ResourceNotFoundException se o produto não existir
63      */
64     public void deleteProduct(int id) {
65         boolean deleted = repository.delete(id);
66         if (!deleted) throw new ResourceNotFoundException(message:"Produto não encontrado");
67     }
68 }
69 }
```

UserService.java

```
EXPLORER
...
BACKEND-API
  .vscode
  settings.json
  src
    main \ java \ com
      system
        exception
          ResourceNotFoundException.java
        model
          AdminUser.java
          Person.java
          Product.java
          User.java
        repository
          ProductRepository.java
          UserRepository.java
        service
          ProductService.java
          UserService.java
    Main.java
    test \ java \ com \ system
    target
    pom.xml

UserService.java
src > main > java > com > system > service > UserService.java > UserService > getUserId(int)
1 package com.system.service;
2
3 import com.system.model.User;
4 import com.system.model.AdminUser;
5 import com.system.repository.UserRepository;
6 import com.system.exception.ResourceNotFoundException;
7 import java.util.List;
8
9 /**
10  * Serviço responsável por gerenciar usuários do sistema.
11  * Contém métodos para criar, buscar, atualizar e deletar usuários.
12  * Inclui suporte para usuários comuns e administradores.
13  * Implementa tratamento de exceções para recursos não encontrados.
14  */
15 public class UserService {
16
17     // Instância do repositório para gerenciar os usuários em memória
18     private final UserRepository repository = new UserRepository();
19
20     /**
21      * Cria um novo usuário comum e adiciona ao repositório.
22      * @param user Usuário a ser criado
23      * @return Usuário criado com ID gerado
24      */
25     public User createUser(User user) {
26         return repository.save(user);
27     }
28
```

```
EXPLORER
...
BACKEND-API
  .vscode
  settings.json
  src
    main \ java \ com
      system
        exception
          ResourceNotFoundException.java
        model
          AdminUser.java
          Person.java
          Product.java
          User.java
        repository
          ProductRepository.java
          UserRepository.java
        service
          ProductService.java
          UserService.java
    Main.java
    test \ java \ com \ system

UserService.java
src > main > java > com > system > service > UserService.java > ...
15 public class UserService {
29
30     /**
31      * Cria um novo administrador e adiciona ao repositório.
32      * Demonstra polimorfismo, já que AdminUser é um User.
33      * @param admin AdminUser a ser criado
34      * @return AdminUser criado com ID gerado
35      */
36     public User createAdmin(AdminUser admin) {
37         return repository.save(admin);
38     }
39
40     /**
41      * Busca um usuário pelo ID.
42      * @param id ID do usuário
43      * @return Usuário encontrado
44      * @throws ResourceNotFoundException se o usuário não existir
45      */
46     public User getUserId(int id) {
47         User user = repository.findById(id);
48         if (user == null) throw new ResourceNotFoundException(message:"Usuário não encontrado");
49         return user;
50     }
51
```

```
15 public class UserService {
51
52     /**
53      * Retorna todos os usuários cadastrados.
54      * @return Lista de usuários
55      */
56     public List<User> getAllUsers() {
57         return repository.findAll();
58     }
59
60     /**
61      * Atualiza os dados de um usuário existente.
62      * @param id ID do usuário a ser atualizado
63      * @param user Novos dados do usuário
64      * @return Usuário atualizado
65      * @throws ResourceNotFoundException se o usuário não existir
66      */
67     public User updateUser(int id, User user) {
68         User updated = repository.update(id, user);
69         if (updated == null) throw new ResourceNotFoundException(message:"Usuário não encontrado");
70         return updated;
71     }
72
73     /**
74      * Deleta um usuário pelo ID.
75      * @param id ID do usuário a ser deletado
76      * @throws ResourceNotFoundException se o usuário não existir
77      */
78     public void deleteUser(int id) {
79         boolean deleted = repository.delete(id);
80         if (!deleted) throw new ResourceNotFoundException(message:"Usuário não encontrado");
81     }
}
```

Main.java

```
1 package com;
2
3 import io.javalin.Javalin;
4 import io.javalin.http.Context;
5 import com.system.model.User;
6 import com.system.model.AdminUser;
7 import com.system.model.Product;
8 import com.system.service.UserService;
9 import com.system.service.ProductService;
10 import com.system.exception.ResourceNotFoundException;
11 import com.fasterxml.jackson.databind.ObjectMapper;
12
13 /**
14  * Classe principal que inicializa a API usando Javalin.
15  * Define todas as rotas para usuários e produtos.
16  * Implementa tratamento de exceções e respostas em JSON.
17  */
18 public class Main {
19
20     // Serviços para gerenciar usuários e produtos
21     private static final UserService userService = new UserService();
22     private static final ProductService productService = new ProductService();
23
24     // Jackson ObjectMapper para converter JSON em objetos Java e vice-versa
25     private static final ObjectMapper objectMapper = new ObjectMapper();
26 }
```

EXPLORER

BACKEND-API

src

main

java

com

Main.java

src > main > java > com > J Main.java > ...

18 public class Main {

27 Run | Debug

28 public static void main(String[] args) {

29 // Inicializa a aplicação Javalin na porta 7000

30 Javalin app = Javalin.create().start(port:7000);

31 // Rota raiz para teste da API

32 app.get(path:"/", ctx -> ctx.result(resultString:"API funcionando 🚀"));

33

34 // Rotas de Usuários

35 app.get(path:"/users", ctx -> ctx.json(userService.getAllUsers())); // Lista todos os usuários

36 app.get(path:"/users/:id", Main::getUserById); // Busca usuário por ID

37 app.post(path:"/users", Main::createUser); // Cria usuário comum

38 app.post(path:"/admin", Main::createAdminUser); // Cria usuário administrador

39 app.put(path:"/users/:id", Main::updateUser); // Atualiza usuário

40 app.delete(path:"/users/:id", Main::deleteUser); // Deleta usuário

41

42 // Rotas de Produtos

43 app.get(path:"/products", ctx -> ctx.json(productService.getAllProducts())); // Lista todos os produtos

44 app.get(path:"/products/:id", Main::getProductById); // Busca produto por ID

45 app.post(path:"/products", Main::createProduct); // Cria produto

46 app.put(path:"/products/:id", Main::updateProduct); // Atualiza produto

47 app.delete(path:"/products/:id", Main::deleteProduct); // Deleta produto

48

49 // Tratamento de exceções para recursos não encontrados

50 app.exception(exceptionClass:ResourceNotFoundException.class, (e, ctx) -> {

51 ctx.status(arg0:404); // Retorna status HTTP 404

52 ctx.json(new ErrorResponse(e.getMessage())); // Retorna mensagem de erro em JSON

53 });

54 }

EXPLORER

BACKEND-API

src

main

java

com

Main.java

src > main > java > com > J Main.java > ...

18 public class Main {

56 // ===== Métodos auxiliares para Usuário =====

57

58 private static void getUserById(Context ctx) {

59 int id = Integer.parseInt(ctx.pathParam(arg0:"id")); // Extrai o ID da rota

60 ctx.json(userService.getUserById(id)); // Busca e retorna o usuário

61 }

62

63 private static void createUser(Context ctx) throws Exception {

64 // Converte JSON do corpo da requisição em objeto User

65 User user = objectMapper.readValue(ctx.body(), valueType:User.class);

66 ctx.json(userService.createUser(user)); // Salva e retorna o usuário

67 }

68

69 private static void createAdminUser(Context ctx) throws Exception {

70 // Converte JSON em AdminUser (herda de User)

71 AdminUser admin = objectMapper.readValue(ctx.body(), valueType:AdminUser.class);

72 ctx.json(userService.createAdmin(admin)); // Salva e retorna o administrador

73 }

74

75 private static void updateUser(Context ctx) throws Exception {

76 int id = Integer.parseInt(ctx.pathParam(arg0:"id"));

77 User user = objectMapper.readValue(ctx.body(), valueType:User.class);

78 ctx.json(userService.updateUser(id, user)); // Atualiza e retorna o usuário

79 }

80

81 private static void deleteUser(Context ctx) {

82 int id = Integer.parseInt(ctx.pathParam(arg0:"id"));

83 userService.deleteUser(id); // Deleta usuário

84 ctx.status(arg0:204); // Retorna HTTP 204 (No Content)

85 }

86 }

EXPLORER

BACKEND-API

.vscode

settings.json

src

main\java\com

system

exception

ResourceNotFoundException.java

model

AdminUser.java

Person.java

Product.java

User.java

repository

ProductRepository.java

UserRepository.java

service

ProductService.java

UserService.java

Main.java

test\java\com\system

target

pom.xml

OUTLINE

TIMELINE

JAVA PROJECTS

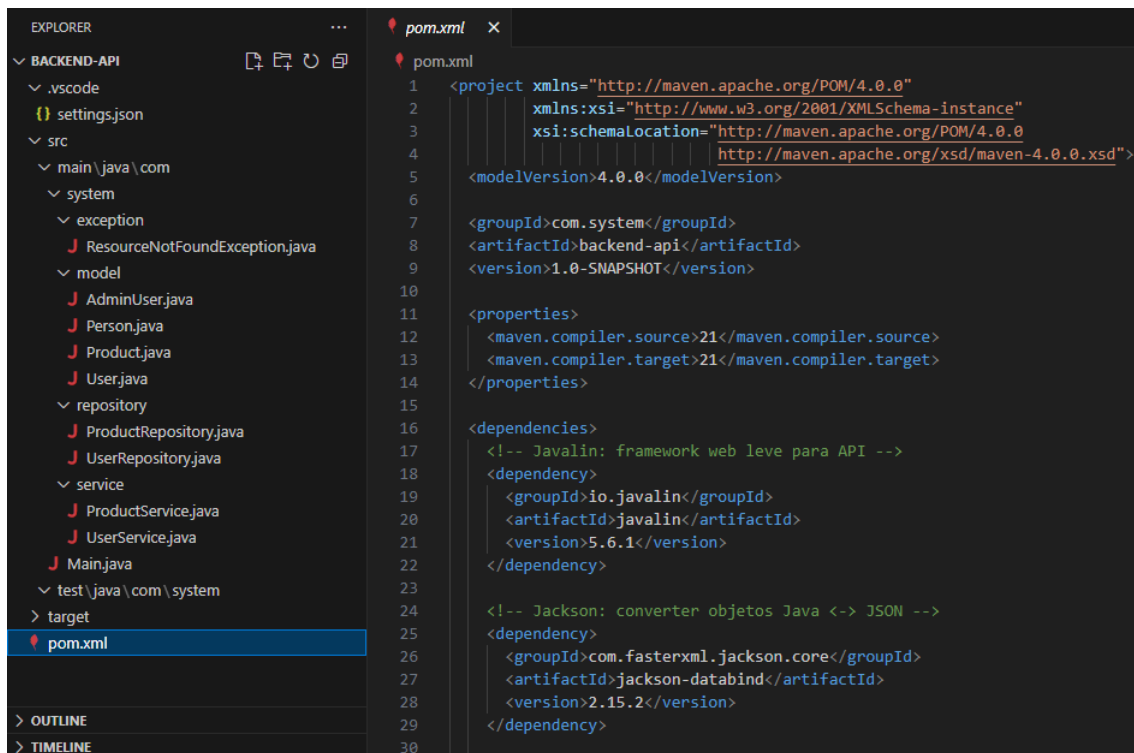
MAVEN

Main.java

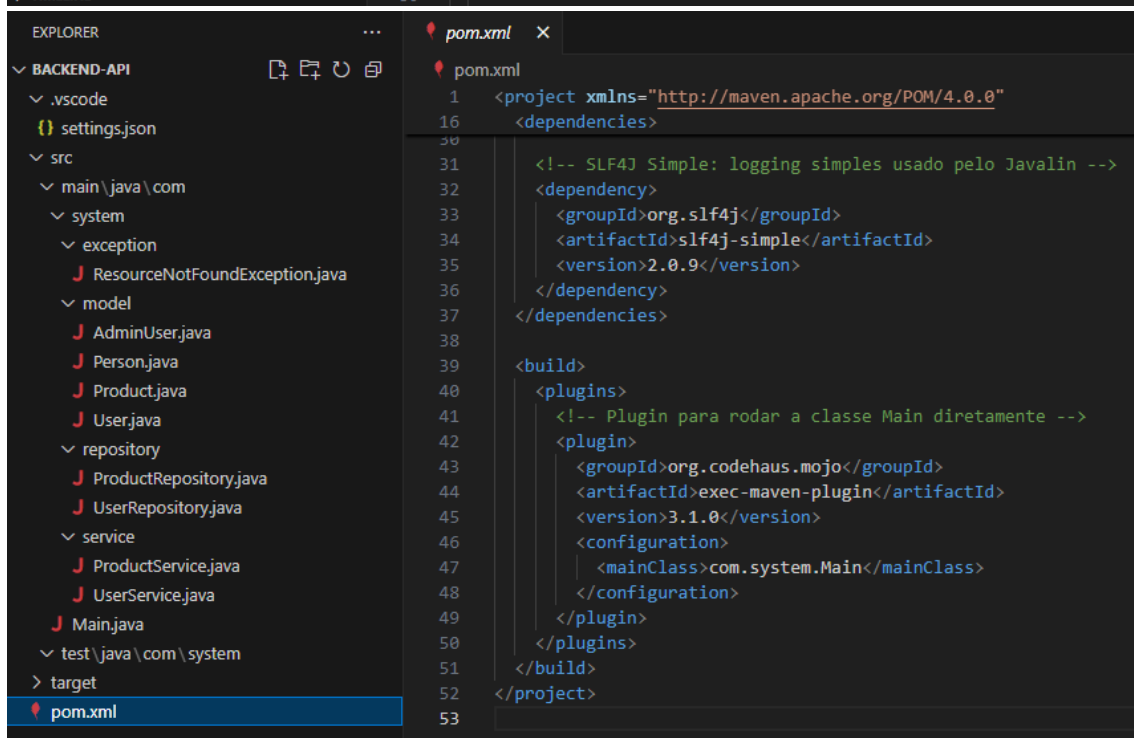
src > main > java > com > J Main.java > ...

18 public class Main {
87 // ===== Métodos auxiliares para Produto =====
88
89 private static void getProductById(Context ctx) {
90 int id = Integer.parseInt(ctx.pathParam(arg0:"id"));
91 ctx.json(productService.getProductById(id)); // Busca e retorna o produto
92 }
93
94 private static void createProduct(Context ctx) throws Exception {
95 Product product = objectMapper.readValue(ctx.body(), valueType:Product.class);
96 ctx.json(productService.createProduct(product)); // Salva e retorna o produto
97 }
98
99 private static void updateProduct(Context ctx) throws Exception {
100 int id = Integer.parseInt(ctx.pathParam(arg0:"id"));
101 Product product = objectMapper.readValue(ctx.body(), valueType:Product.class);
102 ctx.json(productService.updateProduct(id, product)); // Atualiza e retorna o produto
103 }
104
105 private static void deleteProduct(Context ctx) {
106 int id = Integer.parseInt(ctx.pathParam(arg0:"id"));
107 productService.deleteProduct(id); // Deleta produto
108 ctx.status(arg0:204); // Retorna HTTP 204 (No Content)
109 }
110
111 // ===== Classe interna para resposta de erro =====
112 static class ErrorResponse {
113 public String error; // Mensagem de erro
114
115 public ErrorResponse(String msg) { this.error = msg; }
116 }
117 }

pom.xml



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.system</groupId>
8     <artifactId>backend-api</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>21</maven.compiler.source>
13         <maven.compiler.target>21</maven.compiler.target>
14     </properties>
15
16     <dependencies>
17         <!-- Javalin: framework web leve para API -->
18         <dependency>
19             <groupId>io.javalin</groupId>
20             <artifactId>javalin</artifactId>
21             <version>5.6.1</version>
22         </dependency>
23
24         <!-- Jackson: converter objetos Java <-> JSON -->
25         <dependency>
26             <groupId>com.fasterxml.jackson.core</groupId>
27             <artifactId>jackson-databind</artifactId>
28             <version>2.15.2</version>
29         </dependency>
30     </dependencies>
```



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
16 <dependencies>
30
31     <!-- SLF4J Simple: logging simples usado pelo Javalin -->
32     <dependency>
33         <groupId>org.slf4j</groupId>
34         <artifactId>slf4j-simple</artifactId>
35         <version>2.0.9</version>
36     </dependency>
37 </dependencies>
38
39 <build>
40     <plugins>
41         <!-- Plugin para rodar a classe Main diretamente -->
42         <plugin>
43             <groupId>org.codehaus.mojo</groupId>
44             <artifactId>exec-maven-plugin</artifactId>
45             <version>3.1.0</version>
46             <configuration>
47                 <mainClass>com.system.Main</mainClass>
48             </configuration>
49         </plugin>
50     </plugins>
51 </build>
52 </project>
53
```


Documentação:

Pré-requisitos para execução do projeto:

1. Java 21 instalado
2. Maven instalado
3. VS Code com Extension Pack for Java instalado

Passos para rodar a aplicação:

1. Abrindo o projeto, executar no terminal do VS Code:

```
mvn clean install  
mvn exec:java -Dexec.mainClass="com.Main"
```

A API irá iniciar na porta 7000, você verá a mensagem da API funcionando.

Exemplos de Endpoints

Criar usuário:

```
POST http://localhost:7000/users  
Body (JSON):  
{  
  "name": "Higor",  
  "email": "higor@system.com"  
}
```

Criar AdminUser:

```
POST http://localhost:7000/admin  
Body (JSON):  
{  
  "name": "Admin Higor",  
  "email": "admin@system.com"  
}
```

Listar todos usuários:

```
GET http://localhost:7000/users
```

Atualizar usuário:

```
PUT http://localhost:7000/users/1  
Body (JSON):  
{  
  "name": "Higor Updated",  
  "email": "higor.updated@system.com"  
}
```

Deletar usuário:

```
DELETE http://localhost:7000/users/1
```

CRUD do Produto segue a mesma lógica usando /products.

Relatório: Arrays, Coleções e Exceções em Java

Introdução

No desenvolvimento de aplicações back-end, gerenciamento eficiente de dados é FUNDAMENTAL.

Em Java, arrays, coleções e tratamento de exceções oferecem poderosos meios para se organizar no código, armazenar e validar dados, facilitando a manutenção e a escalabilidade dos sistemas.

Arrays:

- **Arrays são estruturas de dados estáticas, com tamanho fixo.**
- **Permitem armazenar múltiplos elementos do mesmo tipo.**

Por exemplo:

```
int[] numeros = new int[5];  
numeros[0] = 10;  
numeros[1] = 20;
```

Limitação dos Arrays: tamanho fixo, não sendo muito flexível para inserções ou remoções dinâmicas.

Coleções:

- Coleções são estruturas de dados dinâmicas, parte do pacote `java.util`.

Exemplos: `ArrayList`, `HashMap`, `HashSet`.

No projeto:

- `UserRepository` e `ProductRepository` usam `HashMap` para armazenar entidades com IDs como chave:

```
private final Map<Integer, User> users = new HashMap<>();
```

- E retornamos listas com `ArrayList` para permitir iteração fácil:

```
public List<User> findAll() {  
    return new ArrayList<>(users.values());  
}
```

Vantagens das coleções:

- Crescimento dinâmico sem precisar definir tamanho fixo
- Busca rápida (`HashMap`)
- Fácil iteração e manipulação de dados (`ArrayList`)

Tratamento de Exceções:

- Exceções permitem gerenciar erros sem quebrar a aplicação.
- Em back-end, garante que operações críticas não falhem silenciosamente.

No projeto:

- Criação da exceção personalizada
ResourceNotFoundException:

```
public class ResourceNotFoundException extends RuntimeException {  
    public ResourceNotFoundException(String message) {  
        super(message);  
    }  
}
```

E utilização em serviços para validar operações
CRUD:

```
public User getUserById(int id) {  
    User user = repository.findById(id);  
    if (user == null) throw new ResourceNotFoundException("User not found");  
    return user;  
}
```

- O Main.java captura a exceção e retorna resposta HTTP 404, mantendo a API estável.

Mas como as coleções, as exceções e o uso de POO (Programação Orientada a Objetos) melhora a manutenção?

- **Coleções substituem arrays estáticos, permitindo manipulação dinâmica de dados.**
- **Exceções evitam falhas inesperadas e centralizam tratamento de erros.**
- **POO (herança e polimorfismo) nos permite reaproveitar código como no caso de User e AdminUser no projeto e facilita muito a criação de novas funcionalidades e manutenção do sistema.**

Conclusão

O uso de arrays, coleções e tratamento de exceções em Java:

- **Permite manipular dados de forma eficiente**
- **Melhora a robustez do back-end**
- **Facilita manutenção e evolução da aplicação**
- **Em conjunto com POO, torna o código mais modular, reutilizável, muito mais legível e prático de se trabalhar tanto sozinho quanto em equipe.**