

 Target.java ×

 Adapter.java ×

 Adaptee.java ×

```
1 package org.example.adapter;
2
3 /** Target */
4 public interface Target {
5     void methodWithoutArguments();
6 }
```

```
1 package org.example.adapter;
2
3 import java.util.Random;
4
5 /** Adapter */
6 public class Adapter implements Target {
7     public Adapter(Adaptee adaptee) {
8         this.adaptee = adaptee;
9         this.random = new Random();
10    }
11
12    @Override
13    public void methodWithoutArguments() {
14        int argument = random.nextInt();
15        adaptee.methodWithArgument(argument)
16    }
17
18    private final Adaptee adaptee;
19    private Random random;
20 }
```

```
1 package org.example.adapter;
2
3 /** Adaptee */
4 public class Adaptee {
5     public void methodWithArgument(int argument) {
6         System.out.println("Был передан аргумент " + argument);
7     }
8 }
```

```

1 package org.example.builder;
2
3 /** Product */
4 public class Product {
5     private final String first;
6     private final String second;
7     private final int third;
8
9     public Product(String first, String second, int
10         | this.first = first;
11         | this.second = second;
12         | this.third = third;
13     }
14
15     public String getFirst() {
16         | return first;
17     }
18
19     public String getSecond() {
20         | return second;
21     }
22
23     public int getThird() {
24         | return third;
25     }
26
27     @Override
28     public String toString() {
29         | return "Product{"
30         |     | ...+ "first="
31         |     | ...+ first
32         |     | ...+ '\ '
33         |     | ...+ ", second="
34         |     | ...+ second
35         |     | ...+ '\ '
36         |     | ...+ ", third="
37         |     | ...+ third

```

```

36 public class Product {
37     private final String first;
38     private final String second;
39     private final int third;
40
41     public Product(String first, String second, int
42         | this.first = first;
43         | this.second = second;
44         | this.third = third;
45     }
46
47     public String getFirst() {
48         | return first;
49     }
50
51     public String getSecond() {
52         | return second;
53     }
54
55     public int getThird() {
56         | return third;
57     }
58
59     @Override
60     public String toString() {
61         | return "Product{"
62         |     | ...+ "first="
63         |     | ...+ first
64         |     | ...+ '\ '
65         |     | ...+ ", second="
66         |     | ...+ second
67         |     | ...+ '\ '
68         |     | ...+ ", third="
69         |     | ...+ third
70         |     | ...+ '\ '
71         |     | ...+ ";
72     }
73 }

```

```

1 package org.example.builder;
2
3 /** Builder */
4 public class ProductBuilder {
5     private String first;
6     private String second;
7     private int third;
8
9     public ProductBuilder setFirst(String first) {
10         | this.first = first;
11         | return this;
12     }
13
14     public ProductBuilder setSecond(String second) {
15         | this.second = second;
16         | return this;
17     }
18
19     public ProductBuilder setThird(int third) {
20         | this.third = third;
21         | return this;
22     }
23
24     public Product build() {
25         | return new Product(first, second, third);
26     }

```

```
1 package org.example.chain;
2
3 /** Concrete Handler A */
4 public class ConcreteHandlerA extends AbstractHandler {
5     @Override
6     public void handle(Object request) {
7         if (request instanceof Integer) {
8             System.out.println("ConcreteHandlerA обработал int: " + request);
9         } else {
10             handleNext(request);
11         }
12     }
13 }
```

```
1 package org.example.chain;
2
3 /** Concrete Handler B */
4 public class ConcreteHandlerB extends AbstractHandler {
5     @Override
6     public void handle(Object request) {
7         if (request instanceof String) {
8             System.out.println("ConcreteHandlerB обработал String: " + request);
9         } else {
10             handleNext(request);
11         }
12     }
13 }
```

```
1 package org.example.chain;
2
3 /** Abstract Handler */
4 public abstract class AbstractHandler implements Handler {
5     private Handler next;
6
7     @Override
8     public void setNext(Handler handler) {
9         this.next = handler;
10     }
11
12     protected void handleNext(Object request) {
13         if (next != null) {
14             next.handle(request);
15         }
16     }
17 }
```

```
1 package org.example.chain;
2
3 /** Handler */
4 public interface Handler {
5     void setNext(Handler handler);
6
7     void handle(Object request);
8 }
```

```
1 package org.example.decorator;
```

```
1  
2 /** Component */
```

```
3 public interface Component {  
4     void operation();  
5 }
```

```
1 package org.example.decorator;
```

```
1  
2 /** Concrete Component */
```

```
3 public class ConcreteComponent implements Component {  
4  
5     @Override  
6     public void operation() {  
7         System.out.println("ConcreteComponent выполняет операцию");  
8     }  
9 }
```

```
1 package org.example.decorator;
```

```
1  
2 /** Decorator */
```

```
3 public abstract class Decorator implements Component {  
4  
5     protected Decorator(Component component) {  
6         this.component = component;  
7     }  
8  
9     @Override  
10    public void operation() {  
11        component.operation();  
12    }  
13  
14    protected final Component component;  
15 }
```

```
1 package org.example.decorator;
```

```
1  
2 /** Concrete Decorator */
```

```
3 public class ConcreteDecorator extends Decorator {  
4  
5     public ConcreteDecorator(Component component) {  
6         super(component);  
7     }  
8  
9     @Override  
10    public void operation() {  
11        component.operation();  
12        System.out.println("ConcreteDecorator добавляет поведение");  
13    }  
14 }
```

```
1 package org.example.proxy;  
2 /**·Service·*/  
3 public interface Service {  
4     void execute();  
5 }
```

```
1 package org.example.proxy;  
2 /**·Concrete·Service·*/  
3 public class ConcreteService implements Service {  
4  
5     @Override  
6     public void execute() {  
7         System.out.println("ConcreteService выполняет операцию");  
8     }  
9 }
```

```
1 package org.example.proxy;  
2 /**·Service·Proxy·*/  
3 public class ServiceProxy implements Service {  
4  
5     public ServiceProxy(Service service) {  
6         this.service = service;  
7     }  
8  
9     @Override  
10    public void execute() {  
11        System.out.println("ServiceProxy разрешает выполнение операции");  
12        service.execute();  
13    }  
14  
15    private final Service service;  
16 }
```

```
1 package org.example.strategy;  
2 /** Concrete Strategy A */  
3 public class ConcreteStrategyA implements Strategy {  
4  
5     @Override  
6     public void execute() {  
7         System.out.println("ConcreteStrategyA выполняет операцию");  
8     }  
9 }
```

```
1 package org.example.strategy;
```

```
1  
2 /** Strategy */  
3 public interface Strategy {  
4     void execute();  
5 }
```

```
1 package org.example.strategy;  
2 /** Concrete Strategy A */  
3 public class ConcreteStrategyA implements Strategy {  
4  
5     @Override  
6     public void execute() {  
7         System.out.println("ConcreteStrategyA выполняет операцию");  
8     }  
9 }
```

```
1 package org.example.strategy;
```

```
1  
2 /** Context */  
3 public class Context {  
4  
5     public Context(Strategy strategy) {  
6         this.strategy = strategy;  
7     }  
8  
9     public void setStrategy(Strategy strategy) {  
10        this.strategy = strategy;  
11    }  
12  
13    public void executeStrategy() {  
14        strategy.execute();  
15    }  
16  
17    private Strategy strategy;  
18 }
```

```

1  /*
2  * This source file was generated by the Gradle 'init' task
3  */
4  package org.example;
5
6  import org.example.adapter.*;
7  import org.example.builder.*;
8  import org.example.chain.*;
9  import org.example.decorator.*;
10 import org.example.proxy.*;
11 import org.example.strategy.*;
12
13 public class App {
14     public static void main(String[] args) {
15         run();
16     }
17
18     private static void run() {
19         checkAdapterPattern();
20         checkBuilderPattern();
21         checkChainPattern();
22         checkDecoratorPattern();
23         checkProxyPattern();
24         checkStrategyPattern();
25     }
26
27     private static void checkAdapterPattern() {
28         Adaptee adaptee = new Adaptee();
29         Target target = new Adapter(adaptee);
30
31         target.methodWithoutArguments();
32     }
33
34
35
36
37

```

```

37
36     System.out.println(product);
35 }
34
33     private static void checkChainPattern() {
32         Handler handler = new ConcreteHandlerA();
31         handler.setNext(handler: new ConcreteHandlerB());
30
29         handler.handle(request: 42);
28         handler.handle(request: "Hello");
27     }
26
25     private static void checkDecoratorPattern() {
24         Component component = new ConcreteComponent();
23         Component decorator = new ConcreteDecorator(component);
22
21         decorator.operation();
20     }
19
18     private static void checkProxyPattern() {
17         Service service = new ConcreteService();
16         Service proxy = new ServiceProxy(service);
15
14         proxy.execute();
13     }
12
11     private static void checkStrategyPattern() {
10         Context context = new Context(strategy: new ConcreteStrategyA());
9
8         boolean crayfishWhistled = true; // Если рак на горе свистнул
7
6         if (crayfishWhistled) {
5             context.setStrategy(strategy: new ConcreteStrategyB());
4         }
3
2         context.executeStrategy();
1     }
79 }

```

1 < ./gradlew run > output.txt

2 Reusing configuration cache.

3 > Task :app:processResources NO-SOURCE

4 > Task :app:compileJava UP-TO-DATE

5 > Task :app:classes UP-TO-DATE

6
7 > Task :app:run

8 Был передан аргумент -568655464

9 Product{first='first', second='second', third=3}

10 ConcreteHandlerA обработал int: 42

11 ConcreteHandlerB обработал String: Hello

12 ConcreteComponent выполняет операцию

13 ConcreteDecorator добавляет поведение

14 ServiceProxy разрешает выполнение операции

15 ConcreteService выполняет операцию

16 ConcreteStrategyB выполняет операцию

17
18 BUILD SUCCESSFUL in 837ms

19 2 actionable tasks: 1 executed, 1 up-to-date

20 Configuration cache entry reused.