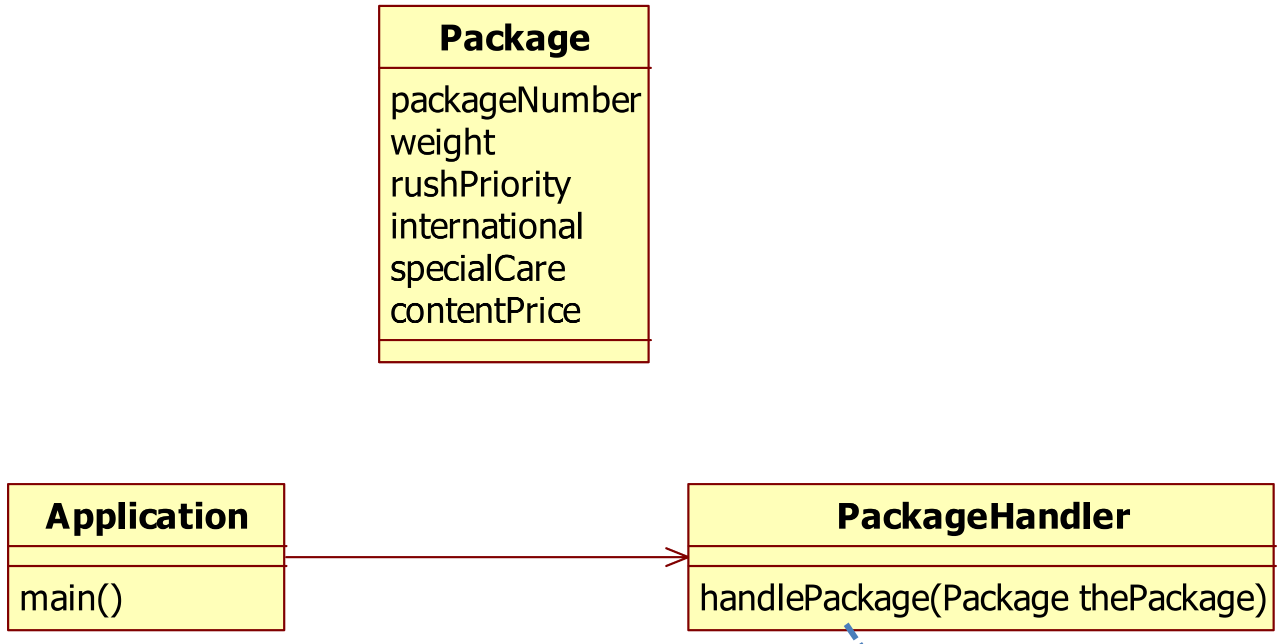
LESSON 7
CHAIN OF RESPONSIBILITY

Handle a package



```
public void handlePackage(Package thePackage) {
    if (thePackage.isInternational()) {
      if (thePackage.isSpecialCare()) { ... } else {... }
    } else if (thePackage.isSpecialCare()) {
      if (thePackage.getWeight()>100) { ... } else {... }
    } else if (thePackage.isRushPriority()) {
      if (thePackage.getWeight()>30) { ... } else {... }
    } else if (thePackage.getContentPrice()>10000.0) { ... } else {... }
  }
```

Package handler application

```
public class Application {
  public static void main(String[] args) {
    PackageHandler packageHandler = new PackageHandler();
    packageHandler.handlePackage(new Package(1543, 56, false, true, true, 300.0));
    packageHandler.handlePackage(new Package(1223, 156, true, false, true, 154.45));
    packageHandler.handlePackage(new Package(545, 12, false, false, false, 30.0));
  }
}
```

```
public class Package {
  private int packageNumber;
  private int weight;
  private boolean rushPriority;
  private boolean international;
  private boolean specialCare;
  private double contentPrice;
  ...
}
```

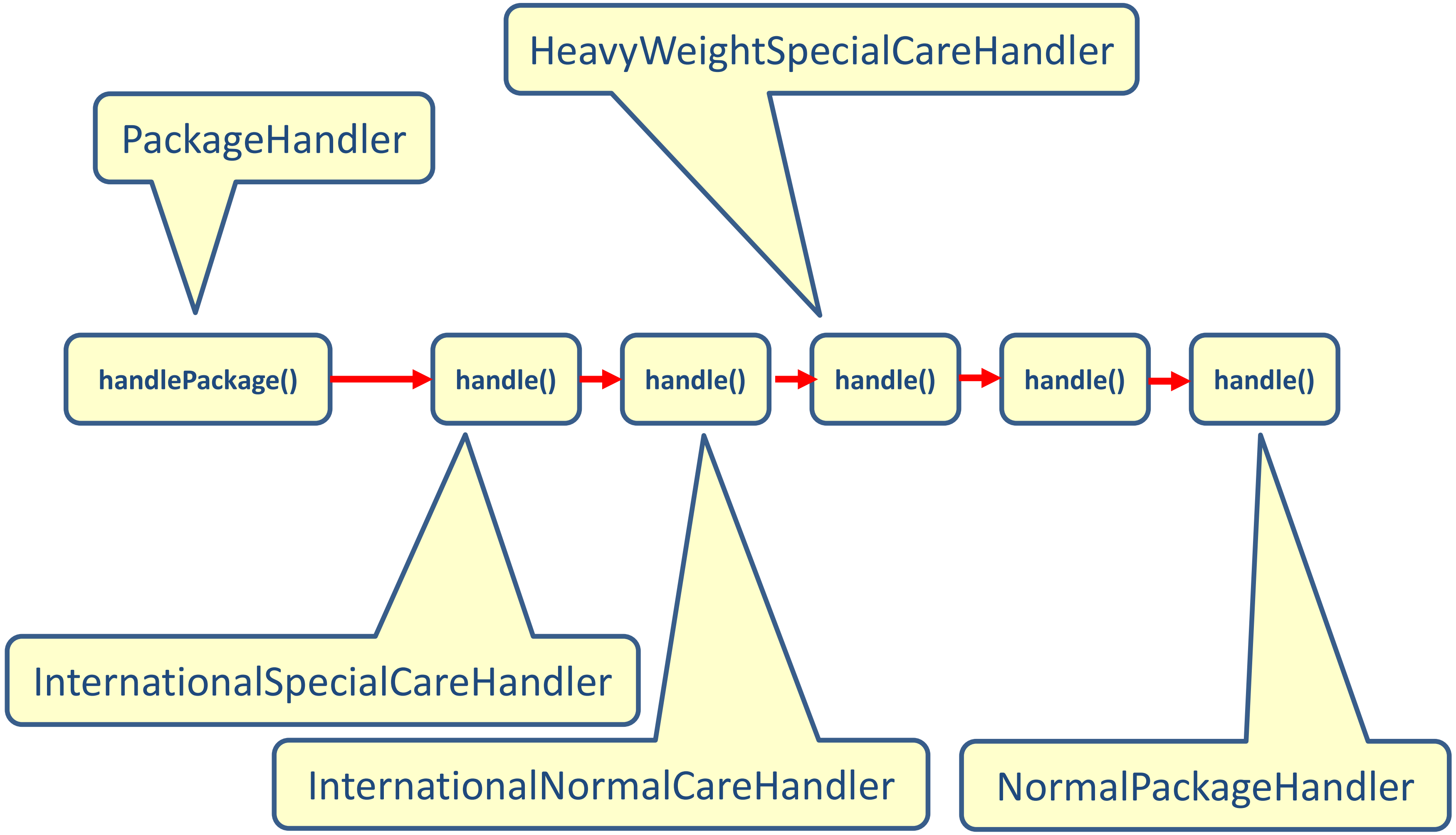
```
Handle international special care package
Handle special care package larger than 100 pounds
Handle normal package
```

PackageHandler

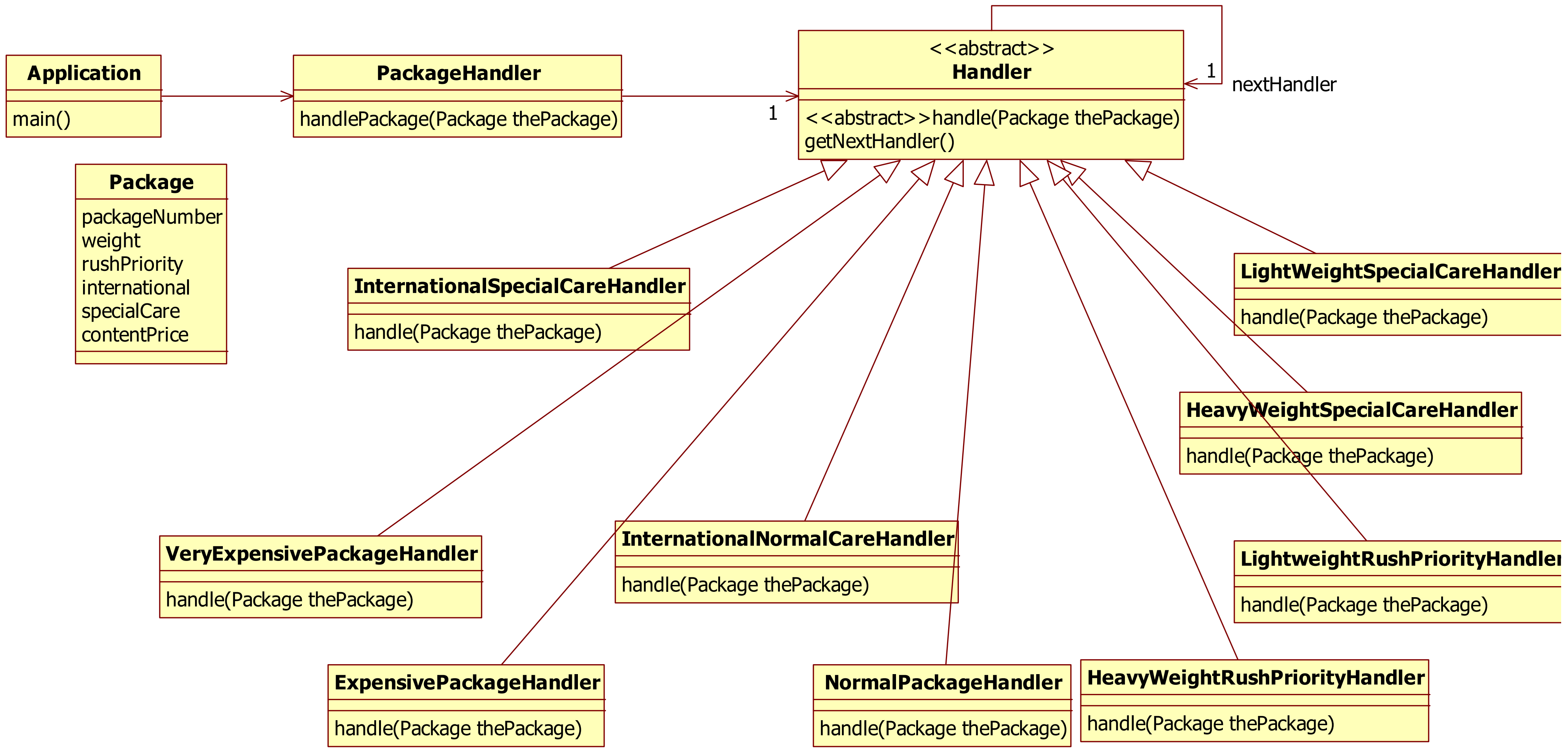
```
public class PackageHandler {
  public void handlePackage(Package thePackage) {
    if (thePackage.isInternational()) {
      if (thePackage.isSpecialCare()) {
        System.out.println("Handle international special care package");
      } else {
        System.out.println("Handle international package");
      }
    } else if (thePackage.isSpecialCare()) {
      if (thePackage.getWeight()>100) {
        System.out.println("Handle special care package larger than 100 pounds");
      } else {
        System.out.println("Handle special care package smaller than 100 pounds");
      }
    } else if (thePackage.isRushPriority()) {
      if (thePackage.getWeight()>30) {
        System.out.println("Handle rush package larger than 30 pounds");
      } else {
        System.out.println("Handle rush package smaller than 30 pounds");
      }
    } else if (thePackage.getContentPrice()>10000.0) {
      if (thePackage.getContentPrice()>1000000.0) {
        System.out.println("Handle expensive package with price > 1000000.0");
      } else {
        System.out.println("Handle expensive package with price > 10000.0");
      }
    }
    else {
      System.out.println("Handle normal package");
    }
  }
```

NOT OK

Chain of responsibility


HeavyWeightSpecialCareHandler
PackageHandler
handlePackage()
handle()
handle()
handle()
handle()
handle()
InternationalSpecialCareHandler
InternationalNormalCareHandler
NormalPackageHandler

Chain of responsibility



Application
main()
PackageHandler
handlePackage(Package thePackage)
1
<<abstract>>
Handler
<<abstract>>handle(Package thePackage)
getNextHandler()
1
nextHandler
Package
packageNumber
weight
rushPriority
international
specialCare
contentPrice
InternationalSpecialCareHandler
handle(Package thePackage)
LightWeightSpecialCareHandler
handle(Package thePackage)
HeavyWeightSpecialCareHandler
handle(Package thePackage)
VeryExpensivePackageHandler
handle(Package thePackage)
InternationalNormalCareHandler
handle(Package thePackage)
LightweightRushPriorityHandle
handle(Package thePackage)
ExpensivePackageHandler
handle(Package thePackage)
NormalPackageHandler
handle(Package thePackage)
HeavyWeightRushPriorityHandler
handle(Package thePackage)

PackageHandler and Handler

```java
public class PackageHandler {
  private Handler chainOfHandlers;

  public void setChainOfHandlers(Handler chainOfHandlers) {
    this.chainOfHandlers = chainOfHandlers;
  }

  public void handlePackage(Package thePackage) {
    chainOfHandlers.handle(thePackage);
  }
}
```

```java
public abstract class Handler {
  protected Handler nextHandler;


  public Handler(Handler nextHandler) {
    this.nextHandler = nextHandler;
  }

  public Handler getNextHandler() {
    return nextHandler;
  }

  public abstract void handle(Package thePackage);
}
```

InternationalSpecialCareHandler

```
public class InternationalSpecialCareHandler extends Handler {

  public InternationalSpecialCareHandler(Handler nextHandler) {
    super(nextHandler);
  }

  @Override
  public void handle(Package thePackage) {
    if (thePackage.isInternational() && thePackage.isSpecialCare()) {
      System.out.println("Handle international special care package");
    } else {
      nextHandler.handle(thePackage);
    }
  }
}
```

NormalPackageHandler

```java
public class NormalPackageHandler extends Handler {

  public NormalPackageHandler(Handler nextHandler) {
    super(nextHandler);
  }

  @Override
  public void handle(Package thePackage) {
    System.out.println("Handle normal package");
  }

}
```

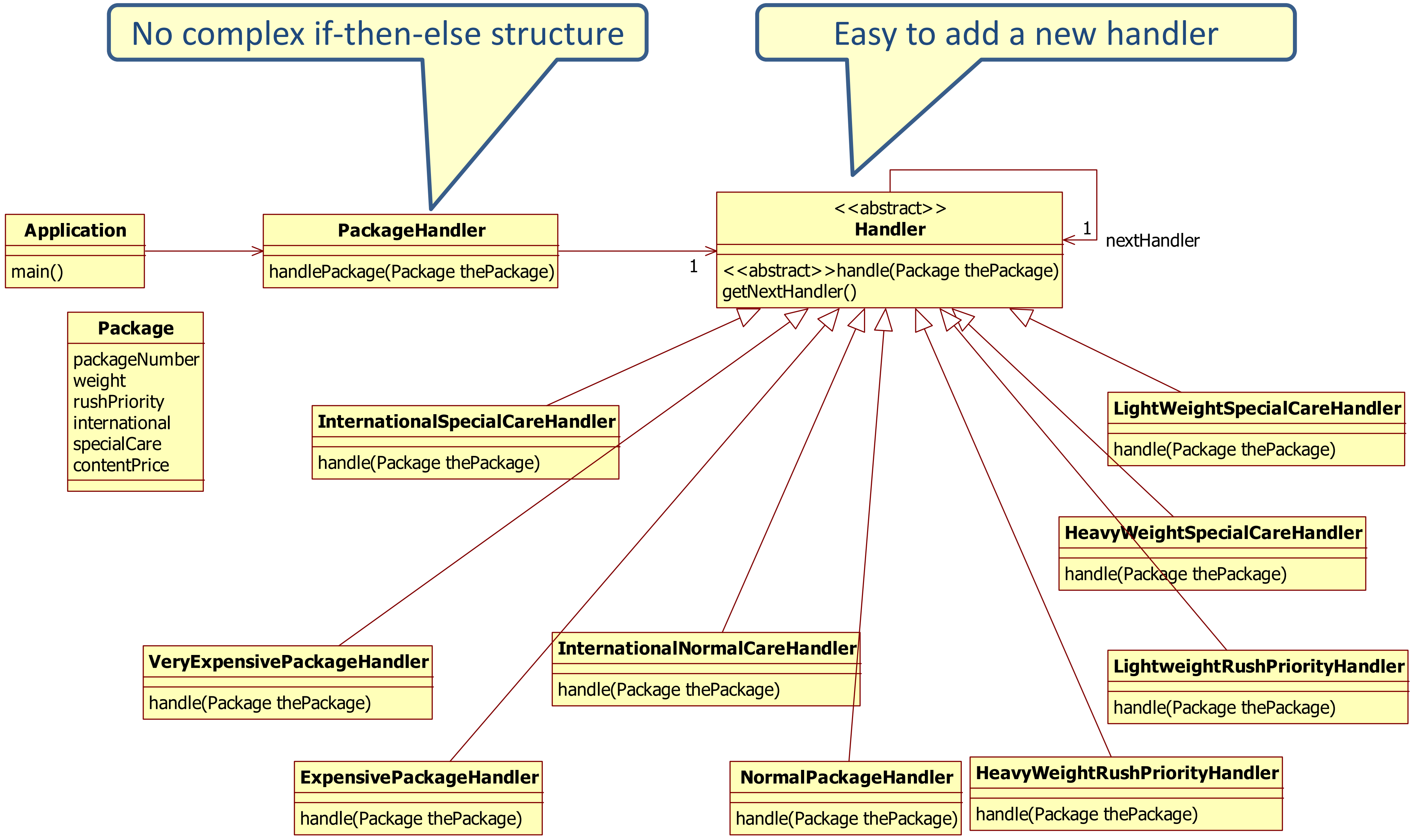
Package handler application

```java
public class Application {
  public static void main(String[] args) {
    PackageHandler packageHandler = new PackageHandler();
    NormalPackageHandler normalPackageHandler = new NormalPackageHandler(null);
    HeavyWeightSpecialCareHandler heavyWeightSpecialCareHandler = new
                              HeavyWeightSpecialCareHandler(normalPackageHandler);
    InternationalNormalCareHandler internationalNormalCareHandler= new
                              InternationalNormalCareHandler(heavyWeightSpecialCareHandler);
    InternationalSpecialCareHandler internationalSpecialCareHandler = new
                             InternationalSpecialCareHandler(internationalNormalCareHandler);

    packageHandler.setChainOfHandlers(internationalSpecialCareHandler);

    packageHandler.handlePackage(new Package(1543, 56, false, true, true, 300.0));
    packageHandler.handlePackage(new Package(1223, 156, true, false, true, 154.45));
    packageHandler.handlePackage(new Package(545, 12, false, false, false, 30.0));
  }
}
```

```
Handle international special care package
Handle special care package larger than 100 pounds
Handle normal package
```

```java
public class Package {
  private int packageNumber;
  private int weight;
  private boolean rushPriority;
  private boolean international;
  private boolean specialCare;
  private double contentPrice;
  ...
}
```

Chain of responsibility


No complex if-then-else structure
Easy to add a new handler
Application
main()
PackageHandler
handlePackage(Package thePackage)
1
<<abstract>>
Handler
<<abstract>>handle(Package thePackage)
getNextHandler()
1
nextHandler
Package
packageNumber
weight
rushPriority
international
specialCare
contentPrice
InternationalSpecialCareHandler
handle(Package thePackage)
LightWeightSpecialCareHandler
handle(Package thePackage)
HeavyWeightSpecialCareHandler
handle(Package thePackage)
VeryExpensivePackageHandler
handle(Package thePackage)
InternationalNormalCareHandler
handle(Package thePackage)
LightweightRushPriorityHandler
handle(Package thePackage)
ExpensivePackageHandler
handle(Package thePackage)
NormalPackageHandler
handle(Package thePackage)
HeavyWeightRushPriorityHandler
handle(Package thePackage)

Handle orders

order1.txt

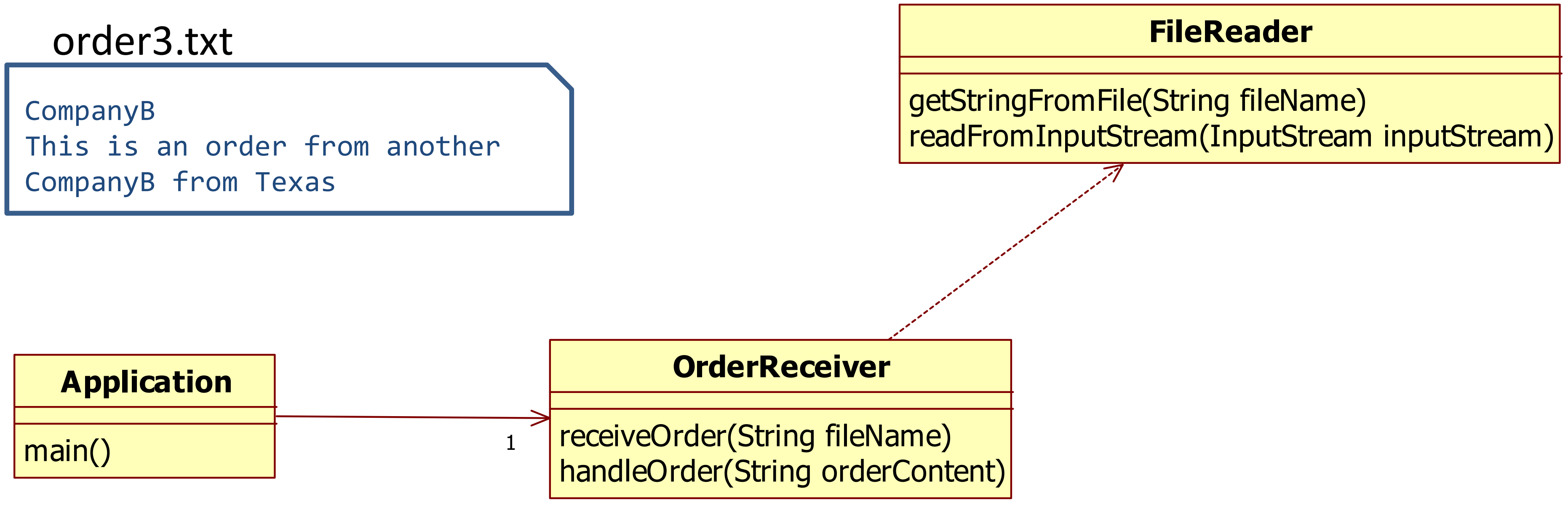
```
CompanyA
This is an order from CompanyA
```

order2.txt

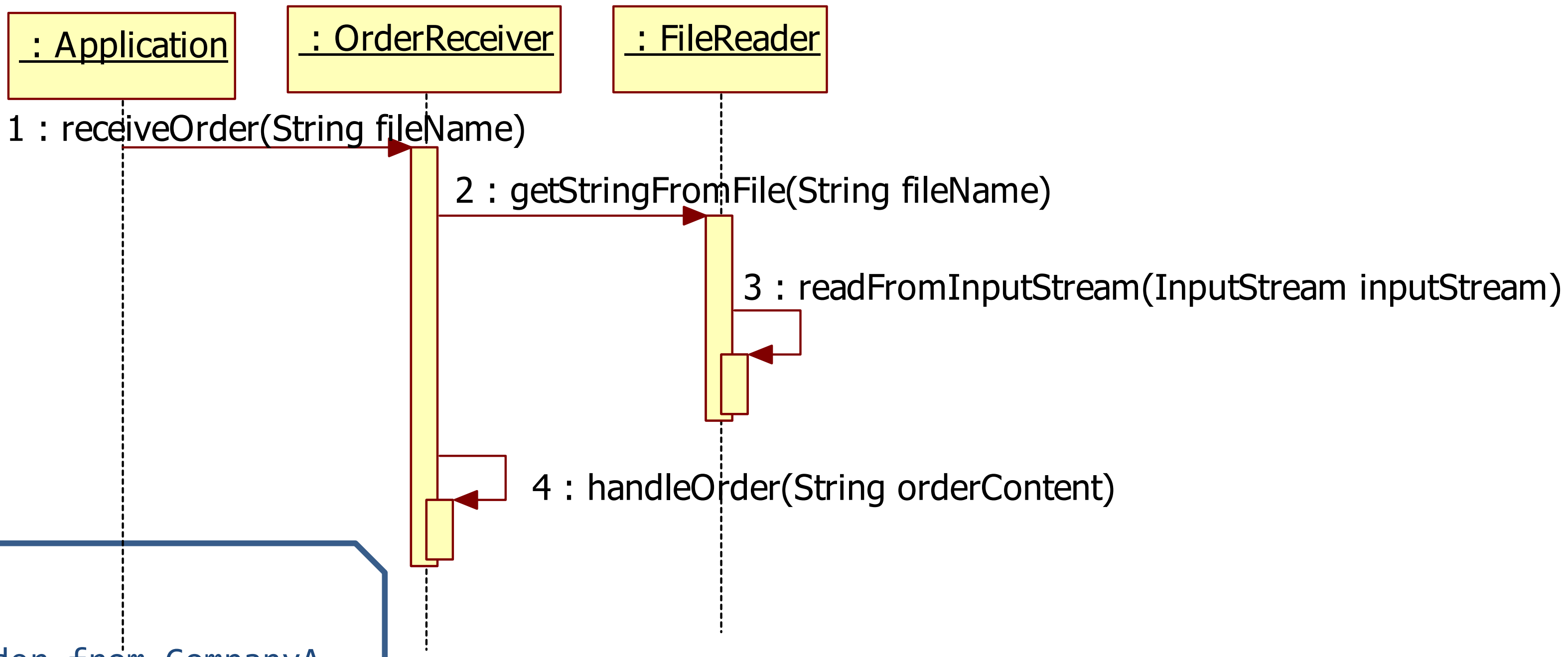
```
This is an order from CompanyB
from New York
CompanyB
```

order3.txt

```
CompanyB
This is an order from another
CompanyB from Texas
```


FileReader
getStringFromFile(String fileName)
readFromInputStream(InputStream inputStream)
Application
main()
1
OrderReceiver
receiveOrder(String fileName)
handleOrder(String orderContent)

Handle orders



: Application
: OrderReceiver
: FileReader
1 : receiveOrder(String fileName)
2 : getStringFromFile(String fileName)
3 : readFromInputStream(InputStream inputStream)
4 : handleOrder(String orderContent)

order1.txt

```
CompanyA
This is an order from CompanyA
```

order2.txt

```
This is an order from CompanyB
from New York
CompanyB
```

order3.txt

```
CompanyB
This is an order from another
CompanyB from Texas
```

```
handle order for CompanyA
handle order for CompanyB from New York
handle order for CompanyB from Texas
```

Application

```
public class Application {

  public static void main(String[] args) {
    OrderReceiver orderReceiver = new OrderReceiver();
    try {
      orderReceiver.receiveOrder("order1.txt");
      orderReceiver.receiveOrder("order2.txt");
      orderReceiver.receiveOrder("order3.txt");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

order1.txt

```
CompanyA
This is an order from CompanyA
```

order2.txt

```
This is an order from CompanyB
from New York
CompanyB
```

order3.txt

```
CompanyB
This is an order from another
CompanyB from Texas
```

```
handle order for CompanyA
handle order for CompanyB from New York
handle order for CompanyB from Texas
```

FileReader

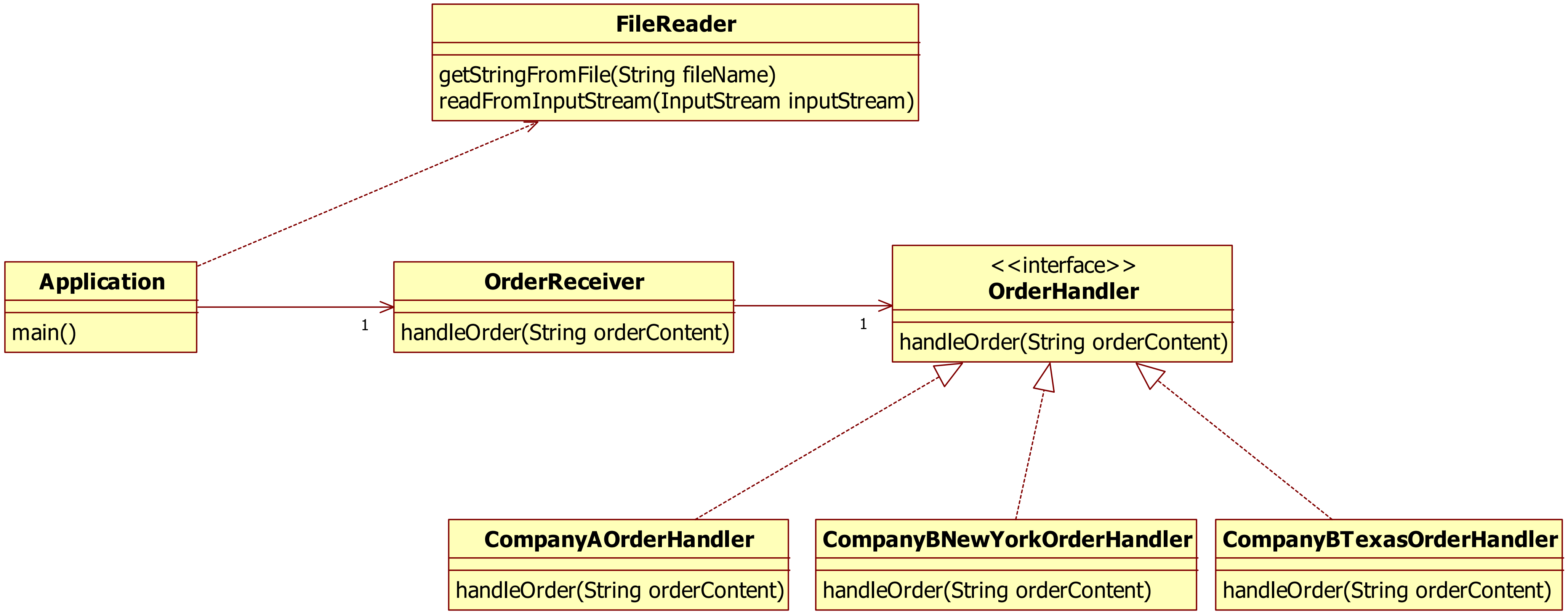
```java
public class FileReader {
  public String getStringFromFile(String fileName) throws IOException {
    ClassLoader classLoader = getClass().getClassLoader();
    InputStream inputStream = classLoader.getResourceAsStream(fileName);
    String content = readFromInputStream(inputStream);
    return content;
  }


  private String readFromInputStream(InputStream inputStream) throws IOException {
    StringBuilder resultStringBuilder = new StringBuilder();
    try (BufferedReader br = new BufferedReader(new InputStreamReader(inputStream))) {
      String line;
      while ((line = br.readLine()) != null) {
        resultStringBuilder.append(line).append("\n");
      }
    }
    return resultStringBuilder.toString();
  }
}
```

Package handler application

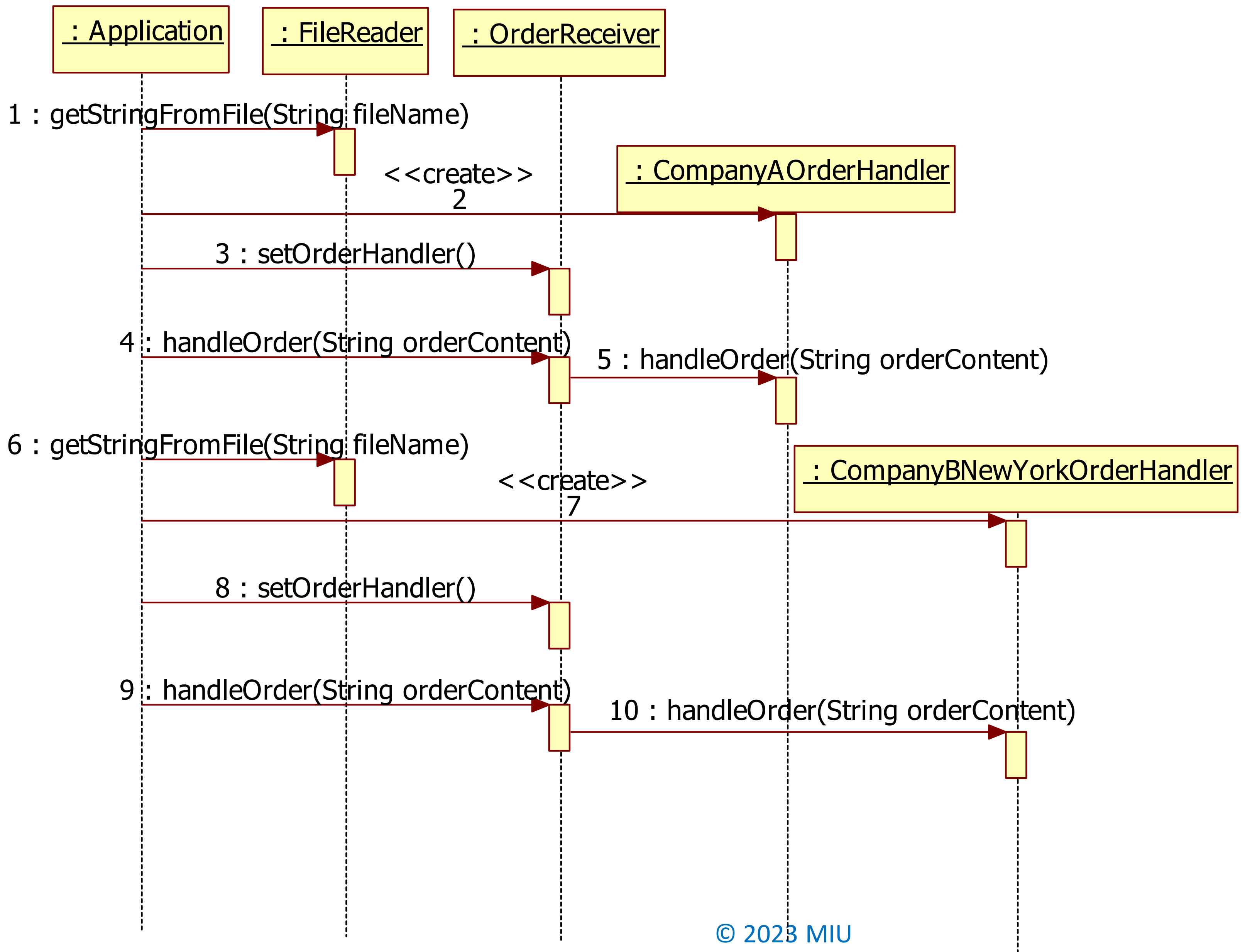
```java
public class OrderReceiver {

  public void receiveOrder(String fileName) throws IOException {
    FileReader fileReader = new FileReader();
    String orderContent = fileReader.getStringFromFile(fileName);
    handleOrder(orderContent);
  }

  public void handleOrder(String orderContent) {
    if (orderContent.startsWith("CompanyA")) {
      System.out.println("handle order for CompanyA");
    } else if (orderContent.lastIndexOf("CompanyB") != -1) {
      if (orderContent.lastIndexOf("New York") != -1) {
        System.out.println("handle order for CompanyB from New York");
      } else if (orderContent.lastIndexOf("Texas") != -1) {
        System.out.println("handle order for CompanyB from Texas");
      }
    }
  }
}
```

NOT OK

Handle orders with strategy


FileReader
getStringFromFile(String fileName)
readFromInputStream(InputStream inputStream)
Application
main()
1
OrderReceiver
handleOrder(String orderContent)
1
<<interface>>
OrderHandler
handleOrder(String orderContent)
CompanyAOrderHandler
handleOrder(String orderContent)
CompanyBNewYorkOrderHandler
handleOrder(String orderContent)
CompanyBTexasOrderHandler
handleOrder(String orderContent)

Handle orders with strategy



: Application
: FileReader
: OrderReceiver
1 : getStringFromFile(String fileName)
<<create>>
2
: CompanyAOrderHandler
3 : setOrderHandler()
4 : handleOrder(String orderContent)
5 : handleOrder(String orderContent)
6 : getStringFromFile(String fileName)
<<create>>
7
: CompanyBNewYorkOrderHandler
8 : setOrderHandler()
9 : handleOrder(String orderContent)
10 : handleOrder(String orderContent)

Order handler strategies

```java
public interface OrderHandler {
  public void handleOrder(String orderContent);
}
```

```java
public class CompanyAOrderHandler implements OrderHandler{

  @Override
  public void handleOrder(String orderContent) {
    System.out.println("handle order for CompanyA");
  }
}
```

```java
public class CompanyBNewYorkOrderHandler implements OrderHandler {

  @Override
  public void handleOrder(String orderContent) {
    System.out.println("handle order for CompanyB from New York");
  }
}
```

```java
public class CompanyBTexasOrderHandler implements OrderHandler {

  @Override
  public void handleOrder(String orderContent) {
    System.out.println("handle order for CompanyB from Texas");
  }
}
```

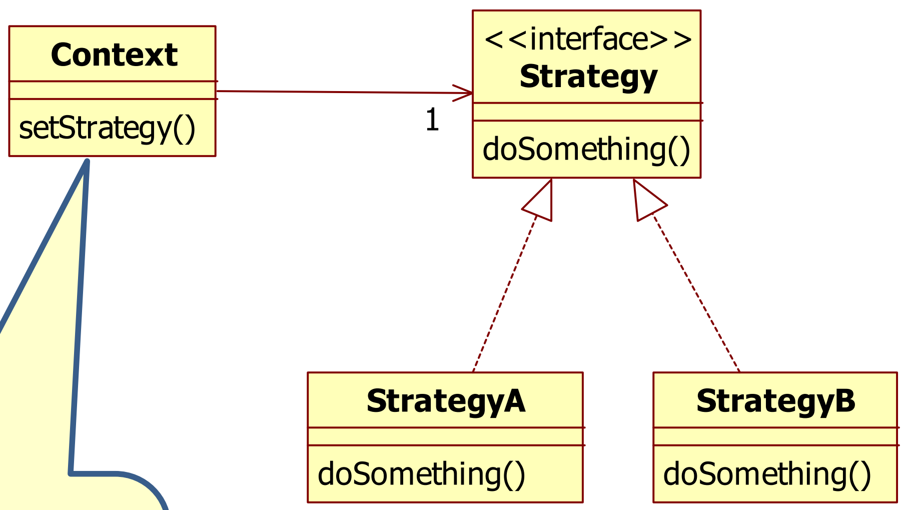
OrderReceiver

```
public class OrderReceiver {
 private OrderHandler orderHandler;

  public void setOrderHandler(OrderHandler orderHandler) {
    this.orderHandler = orderHandler;
  }

  public void handleOrder(String orderContent) {
    orderHandler.handleOrder(orderContent);
  }
}
```

```java
public class Application {

  public static void main(String[] args) {
    OrderReceiver orderReceiver = new OrderReceiver();
    FileReader fileReader = new FileReader();
    try {
      String orderContent = fileReader.getStringFromFile("order1.txt");
      setOrderHandler(orderReceiver, orderContent);
      orderReceiver.handleOrder(orderContent);

       orderContent = fileReader.getStringFromFile("order2.txt");
       setOrderHandler(orderReceiver, orderContent);
       orderReceiver.handleOrder(orderContent);

       orderContent = fileReader.getStringFromFile("order3.txt");
       setOrderHandler(orderReceiver, orderContent);
       orderReceiver.handleOrder(orderContent);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  private static void setOrderHandler(OrderReceiver orderReceiver, String orderContent) {
    if (orderContent.startsWith("CompanyA")) {
      orderReceiver.setOrderHandler(new CompanyAOrderHandler());
    } else if (orderContent.lastIndexOf("CompanyB") != -1) {
      if (orderContent.lastIndexOf("New York") != -1) {
        orderReceiver.setOrderHandler(new CompanyBNewYorkOrderHandler());
      } else if (orderContent.lastIndexOf("Texas") != -1) {
        orderReceiver.setOrderHandler(new CompanyBTexasOrderHandler());
      }
    }
  }
}
```

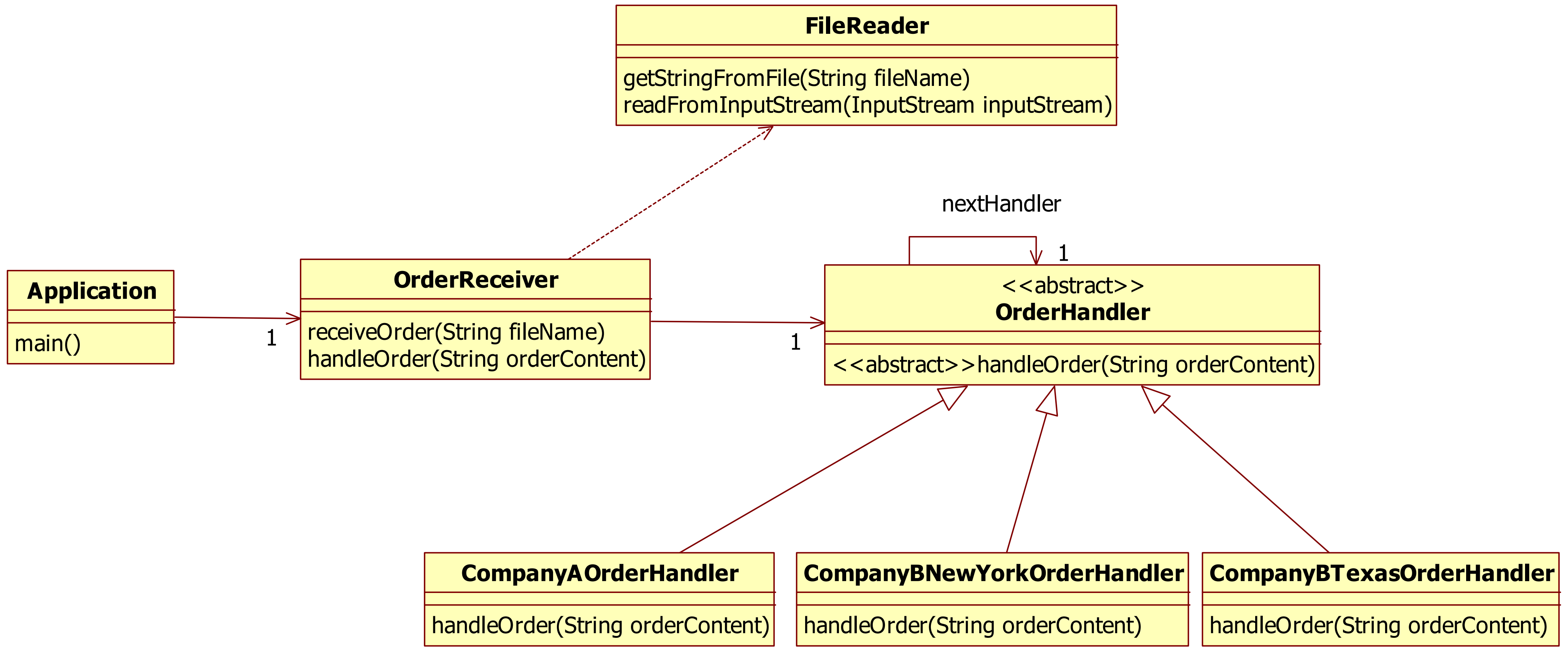
NOT OK

Strategy pattern

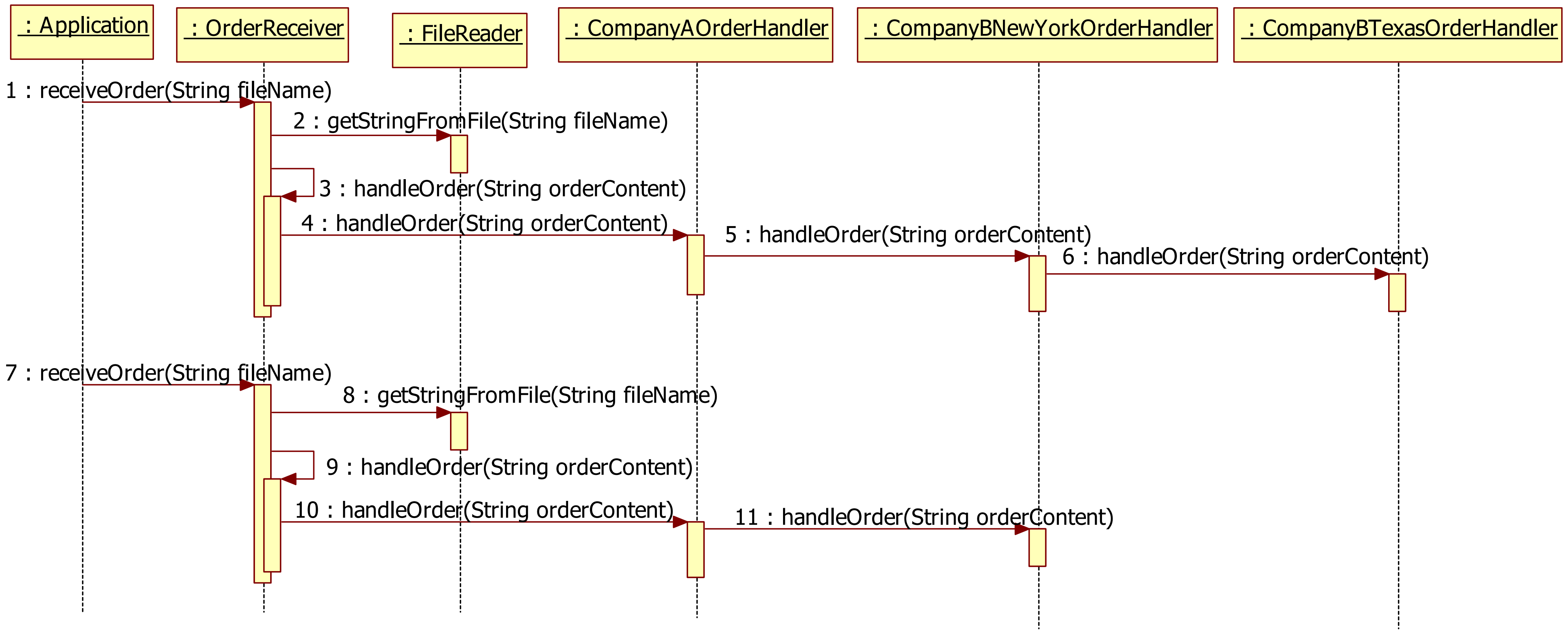


Someone has to decide which strategy to use and then call setStrategy()

Chain of responsibility


FileReader
getStringFromFile(String fileName)
readFromInputStream(InputStream inputStream)
nextHandler
1
Application
main()
1
OrderReceiver
receiveOrder(String fileName)
handleOrder(String orderContent)
1
<<abstract>>
OrderHandler
<<abstract>>handleOrder(String orderContent)
CompanyAOrderHandler
handleOrder(String orderContent)
CompanyBNewYorkOrderHandler
handleOrder(String orderContent)
CompanyBTexasOrderHandler
handleOrder(String orderContent)

Chain of responsibility


: Application
: OrderReceiver
: FileReader
: CompanyAOrderHandler
: CompanyBNewYorkOrderHandler
: CompanyBTexasOrderHandler
1 : receiveOrder(String fileName)
2 : getStringFromFile(String fileName)
3 : handleOrder(String orderContent)
4 : handleOrder(String orderContent)
5 : handleOrder(String orderContent)
6 : handleOrder(String orderContent)
7 : receiveOrder(String fileName)
8 : getStringFromFile(String fileName)
9 : handleOrder(String orderContent)
10 : handleOrder(String orderContent)
11 : handleOrder(String orderContent)

Handlers

```java
public abstract class OrderHandler {
  protected OrderHandler nextHandler;

  public OrderHandler(OrderHandler nextHandler) {
    this.nextHandler = nextHandler;
  }

  public OrderHandler getNextHandler() {
    return nextHandler;
  }
  public abstract void handleOrder(String orderContent);
}
```

```java
public class CompanyAOrderHandler extends OrderHandler{

  public CompanyAOrderHandler(OrderHandler nextHandler) {
    super(nextHandler);
  }

  @Override
  public void handleOrder(String orderContent) {
    if (orderContent.startsWith("CompanyA")) {
      System.out.println("handle order for CompanyA");
    } else {
      nextHandler.handleOrder(orderContent);
    }
  }
}
```

Handlers

```java
public class CompanyBNewYorkOrderHandler extends OrderHandler {

  public CompanyBNewYorkOrderHandler(OrderHandler nextHandler) {
    super(nextHandler);
  }

  @Override
  public void handleOrder(String orderContent) {
    if (orderContent.lastIndexOf("New York") != -1) {
      System.out.println("handle order for CompanyB from New York");
    } else {
      nextHandler.handleOrder(orderContent);
    }
  }
}
```

```java
public class CompanyBTexasOrderHandler extends OrderHandler {

  public CompanyBTexasOrderHandler(OrderHandler nextHandler) {
    super(nextHandler);
  }

  @Override
  public void handleOrder(String orderContent) {
    if (orderContent.lastIndexOf("Texas") != -1) {
      System.out.println("handle order for CompanyB from Texas");
    }
  }
}
```

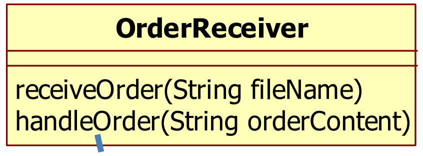
OrderReceiver

```java
public class OrderReceiver {
  private OrderHandler orderHandler;

  public void setOrderHandler(OrderHandler orderHandler) {
    this.orderHandler = orderHandler;
  }

  public void receiveOrder(String fileName) throws IOException {
    FileReader fileReader = new FileReader();
    String orderContent = fileReader.getStringFromFile(fileName);
    handleOrder(orderContent);
  }

  public void handleOrder(String orderContent) {
    orderHandler.handleOrder(orderContent);
  }
}
```

Application

```java
public class Application {

  public static void main(String[] args) {
    OrderReceiver orderReceiver = new OrderReceiver();
    // create the chain
    CompanyBTexasOrderHandler companyBTexasOrderHandler = new CompanyBTexasOrderHandler(null);
    CompanyBNewYorkOrderHandler companyBNewYorkOrderHandler = new
                                      CompanyBNewYorkOrderHandler(companyBTexasOrderHandler);
    CompanyAOrderHandler companyAOrderHandler = new
                                      CompanyAOrderHandler(companyBNewYorkOrderHandler);

    orderReceiver.setOrderHandler(companyAOrderHandler);
    try {
      orderReceiver.receiveOrder("order1.txt");
      orderReceiver.receiveOrder("order2.txt");
      orderReceiver.receiveOrder("order3.txt");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

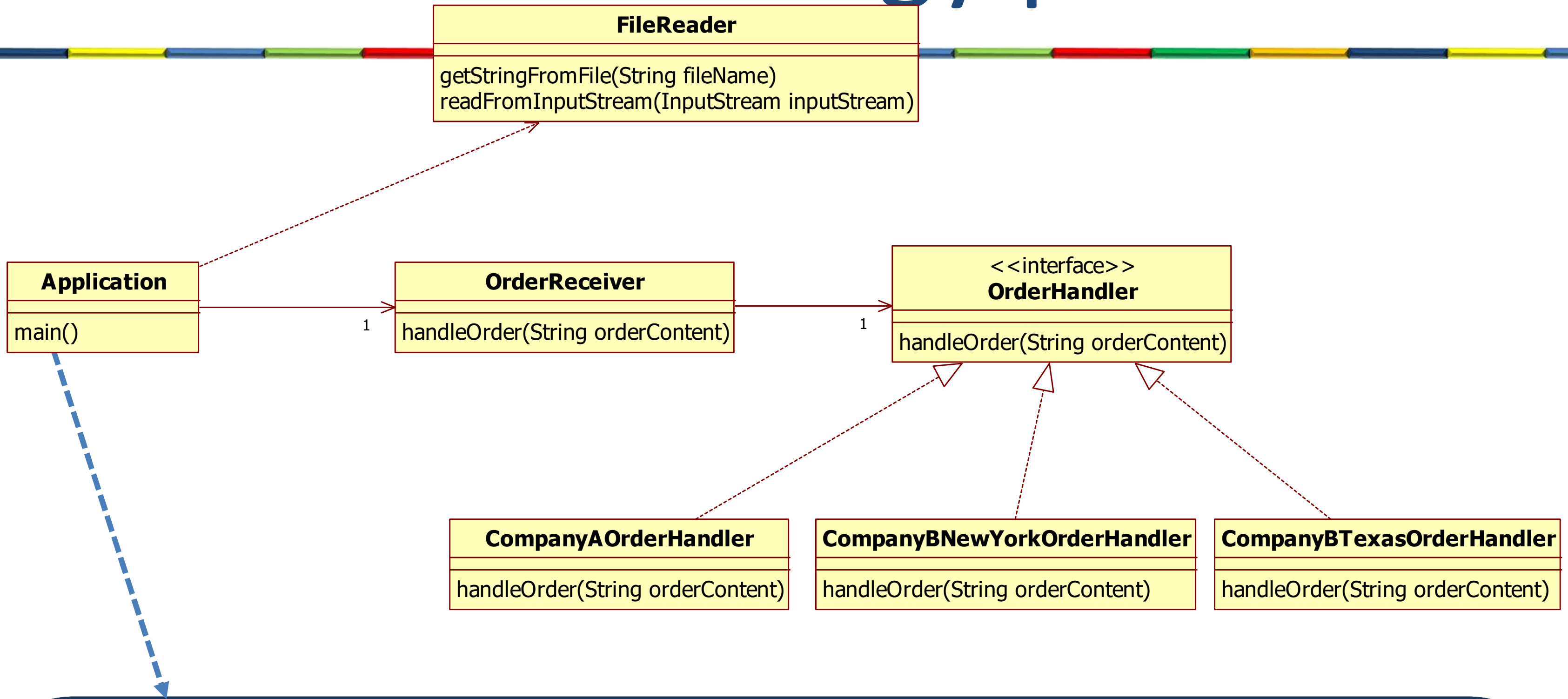
Without chain of responsibility


OrderReceiver
receiveOrder(String fileName)
handleOrder(String orderContent)

```
public void handleOrder(String orderContent) {
  if (orderContent.startsWith("CompanyA")) {
    System.out.println("handle order for CompanyA");
  } else if (orderContent.lastIndexOf("CompanyB") != -1) {
    if (orderContent.lastIndexOf("New York") != -1) {
      System.out.println("handle order for CompanyB from New York");
    } else if (orderContent.lastIndexOf("Texas") != -1) {
      System.out.println("handle order for CompanyB from Texas");
    }
  }
}
```

Large and complex if-then-else structure

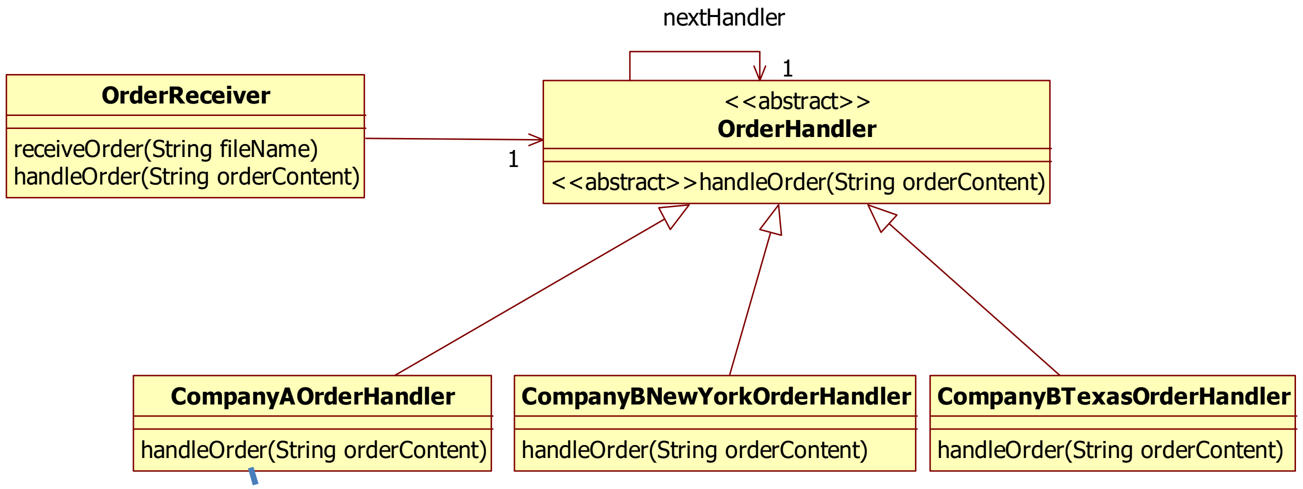
With strategy pattern


FileReader
getStringFromFile(String fileName)
readFromInputStream(InputStream inputStream)
Application
main()
1
OrderReceiver
handleOrder(String orderContent)
1
<<interface>>
OrderHandler
handleOrder(String orderContent)
CompanyAOrderHandler
handleOrder(String orderContent)
CompanyBNewYorkOrderHandler
handleOrder(String orderContent)
CompanyBTexasOrderHandler
handleOrder(String orderContent)

```
private static void setOrderHandler(OrderReceiver orderReceiver, String
orderContent) {
    if (orderContent.startsWith("CompanyA")) {
      orderReceiver.setOrderHandler(new CompanyAOrderHandler());
    } else if (orderContent.lastIndexOf("CompanyB") != -1) {
      if (orderContent.lastIndexOf("New York") != -1) {
        orderReceiver.setOrderHandler(new CompanyBNewYorkOrderHandler());
      } else if (orderContent.lastIndexOf("Texas") != -1) {
        orderReceiver.setOrderHandler(new CompanyBTexasOrderHandler());
      }
    }
  }
```

Large and complex if-then-else structure

With chain of responsibility


nextHandler
1
OrderReceiver
receiveOrder(String fileName)
handleOrder(String orderContent)
1
<<abstract>>
OrderHandler
<<abstract>>handleOrder(String orderContent)
CompanyAOrderHandler
handleOrder(String orderContent)
CompanyBNewYorkOrderHandler
handleOrder(String orderContent)
CompanyBTexasOrderHandler
handleOrder(String orderContent)

```
@Override
public void handleOrder(String orderContent) {
  if (orderContent.startsWith("CompanyA")) {
    System.out.println("handle order for CompanyA");
  } else {
    nextHandler.handleOrder(orderContent);
  }
}
```

Small and simple if-then-else structure

COR issues

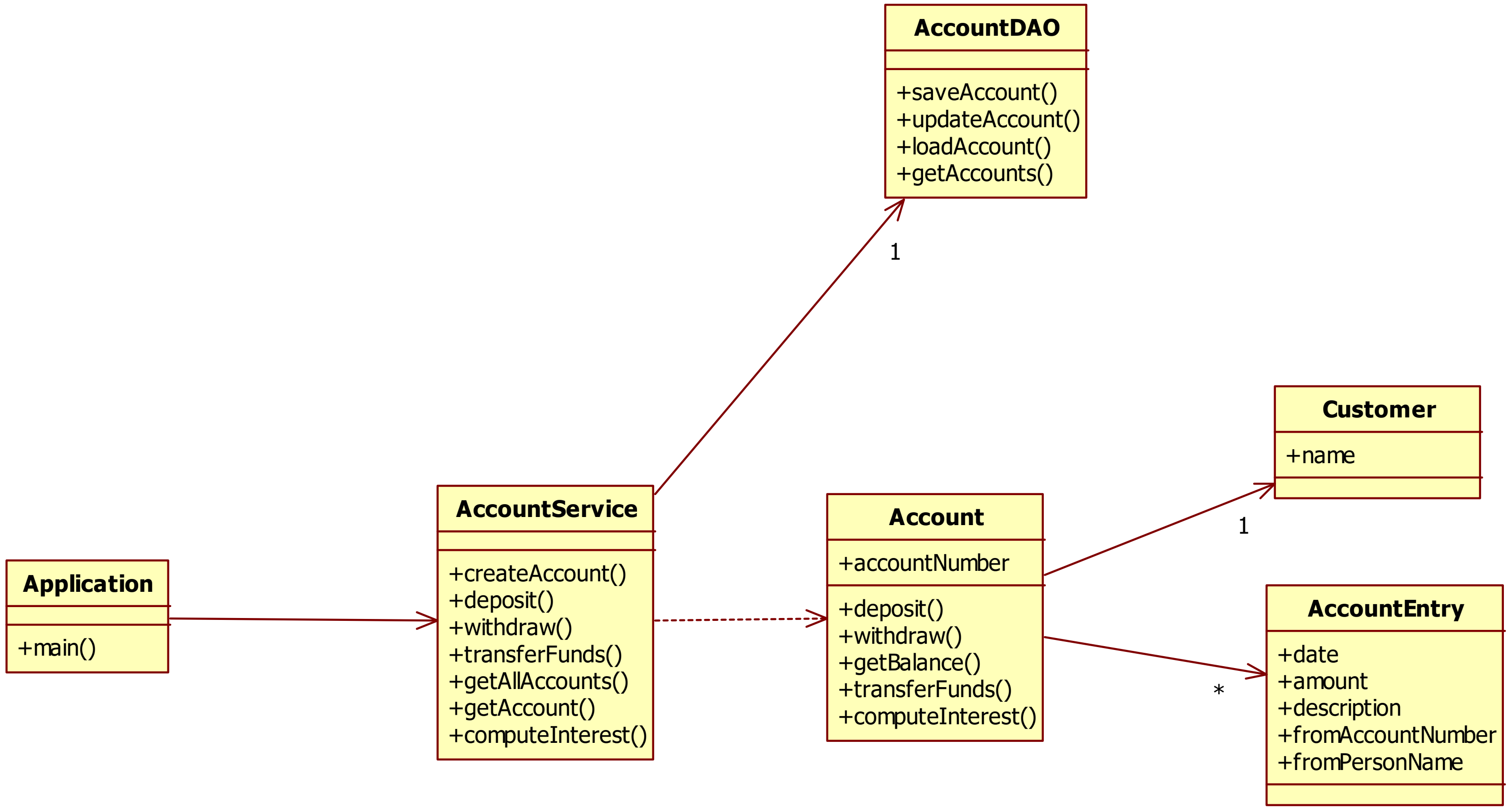
- Who creates the chain?
  - Factory class (later)
- What if no handler will handle the request?
- Does always 1 handler handle the request?

Main point

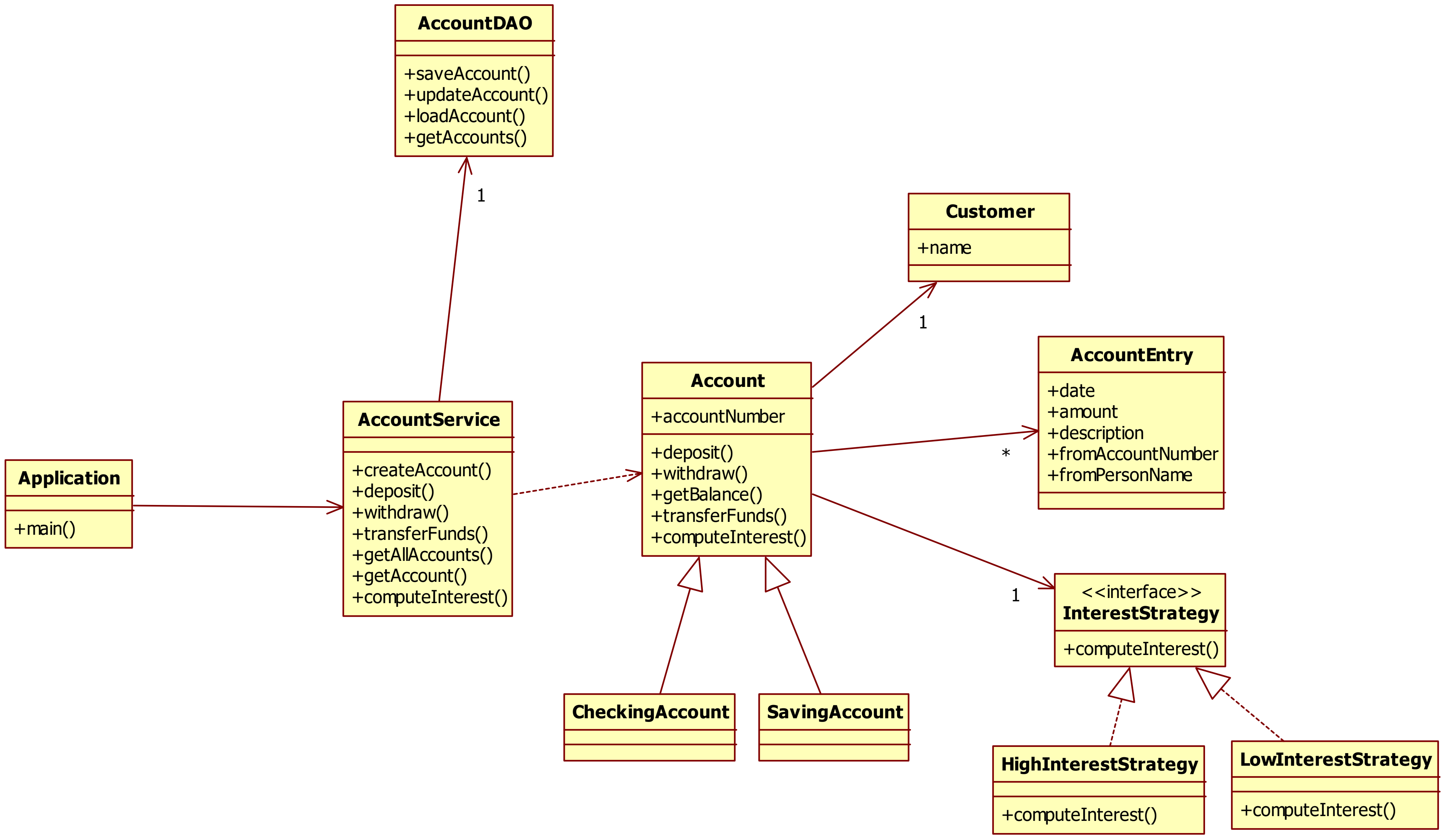
- In the Chain Of Responsibility pattern we connect a number of handlers in a chain.
- In creation, everything is connected to everything else at the transcendental field of pure consciousness

COMBINING PATTERNS

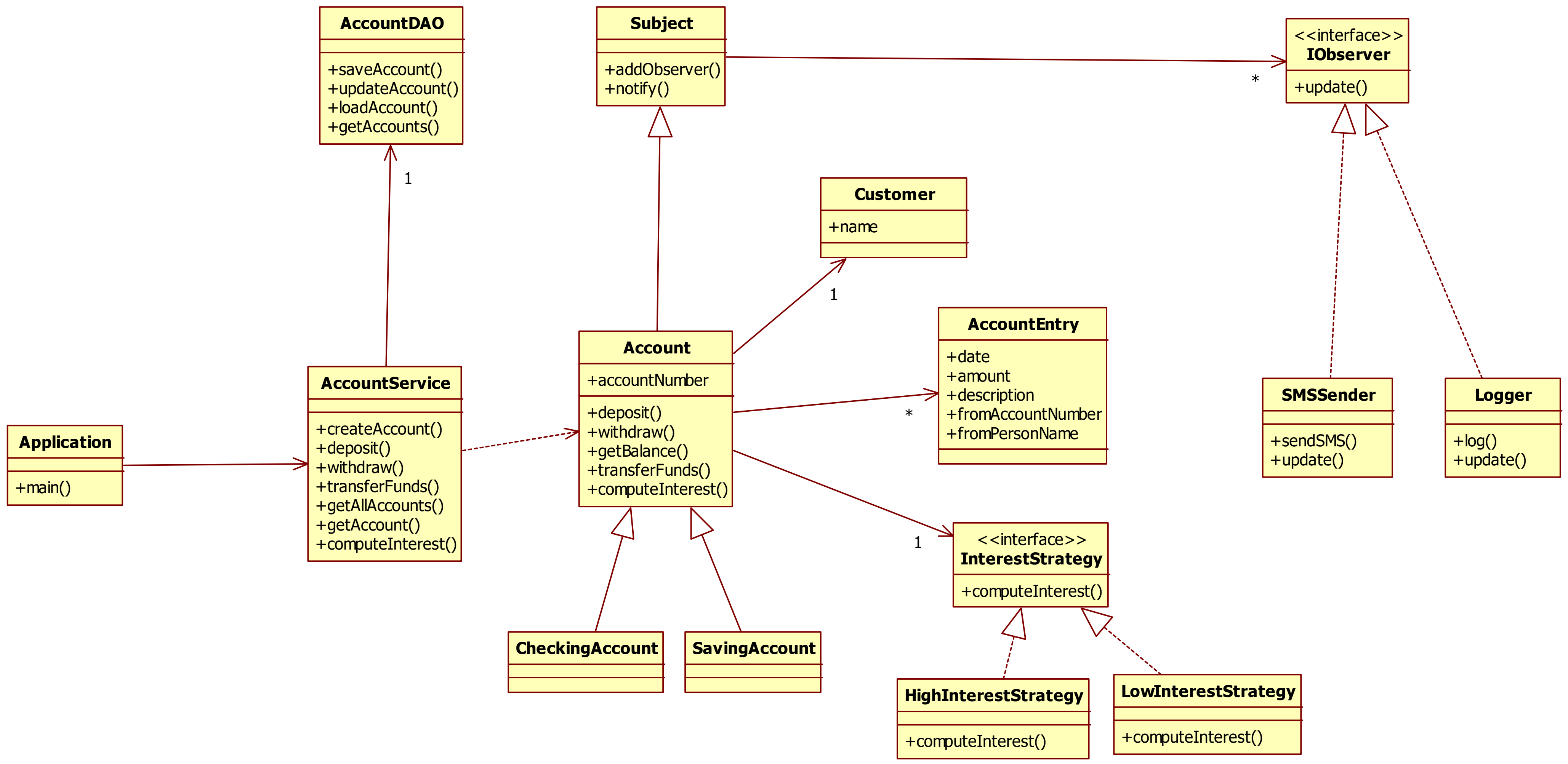
Bank application



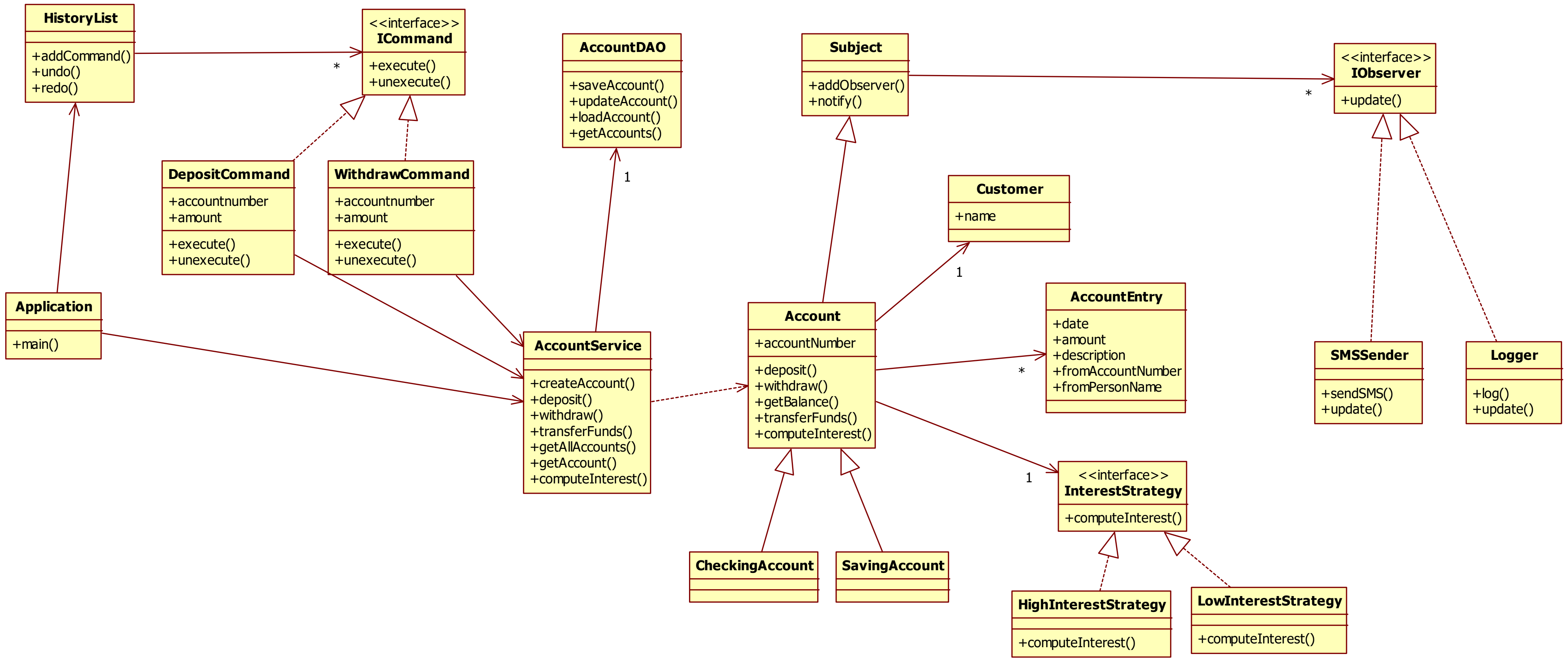
Strategy pattern



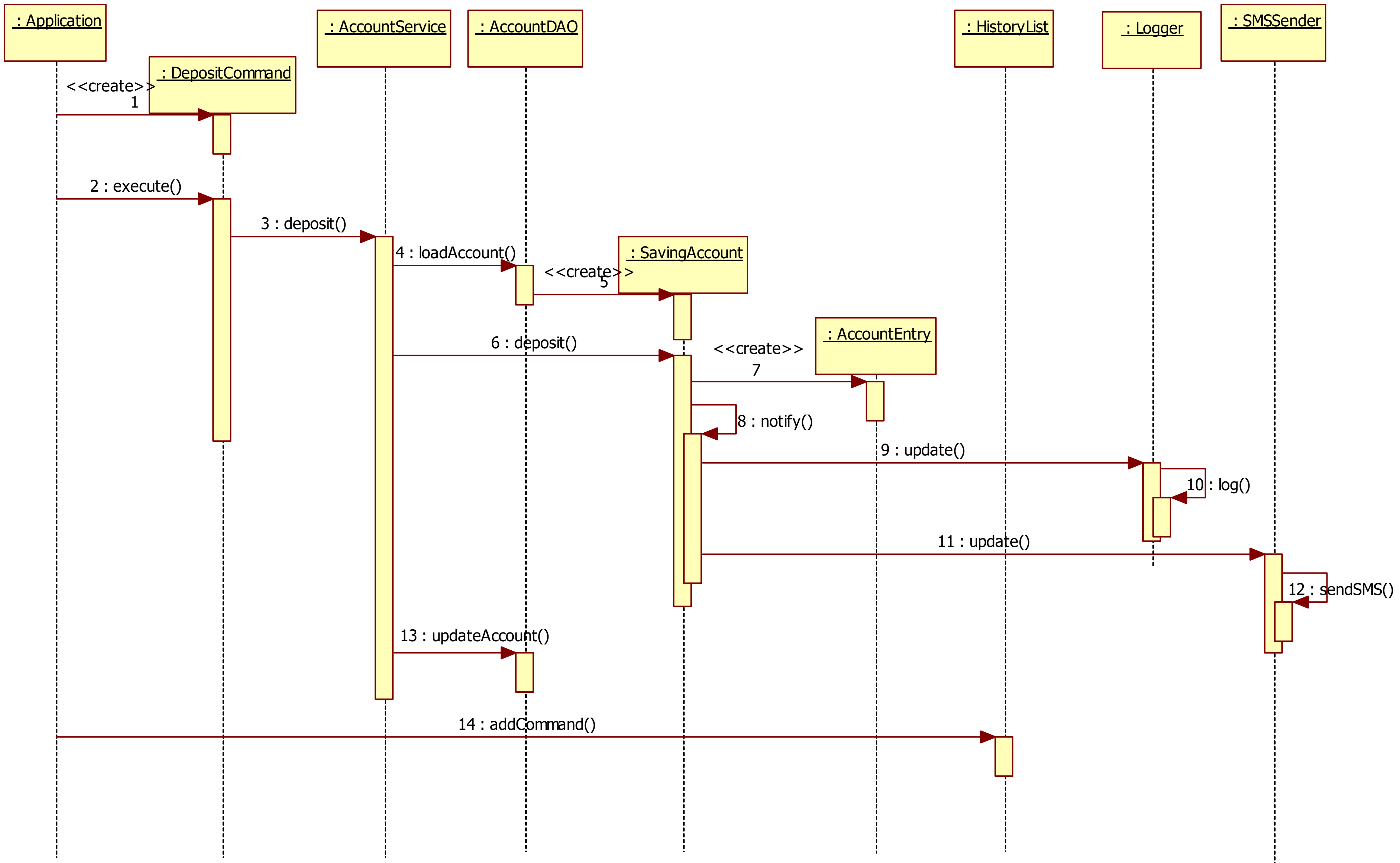
Strategy + observer



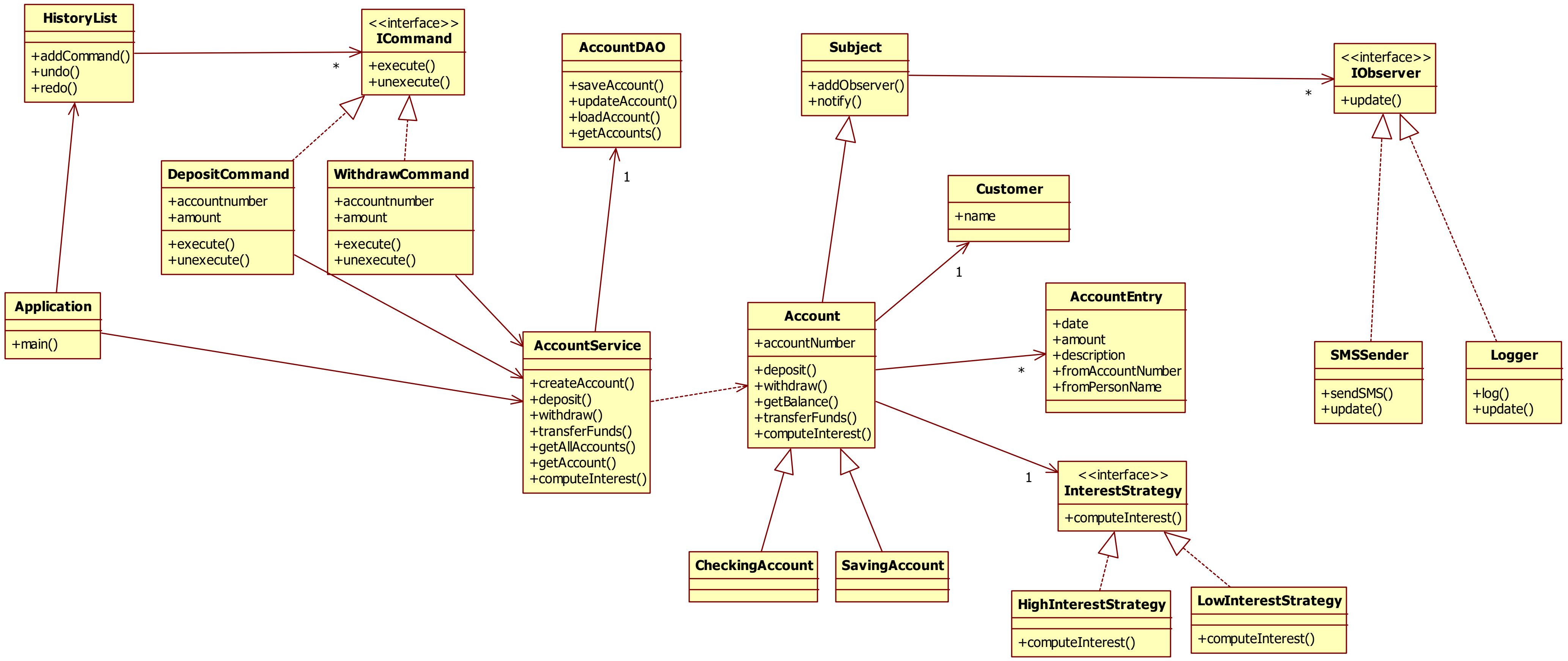
Strategy + observer + command



Strategy + observer + command



Strategy + observer + command



Connecting the parts of knowledge with the wholeness of knowledge

1. With the chain of responsibility pattern we chain different handlers together.
2. The chain of responsibility pattern transforms complex if-then-else logic into many simpler if-then-else structures.

---

3. **Transcendental consciousness** is the source off all activity.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes the unity between everything around you.