Exceptions

Agenda

- What are they? What's the purpose?
- Examples of when exceptions are thrown in java stuff
- How to handle: throws vs try-catch
- Exceptions skip over code to catch block
- Checked vs unchecked
- How can we use exceptions?
- Changing specific exceptions to non-specific
  - Sub-classing RuntimeException

What are they

- **Throwable** is the super class of all exceptions
- All exceptions are throwable, they can be thrown, i.e. we can e.g.

```
throw new IllegalArgumentException("Cannot be null");
```

- Used when something goes wrong.

What does it look like?

```java
public class Test {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.get(1);
    }
}
```

Attempting to get object at index 1. List is empty, though.

Test

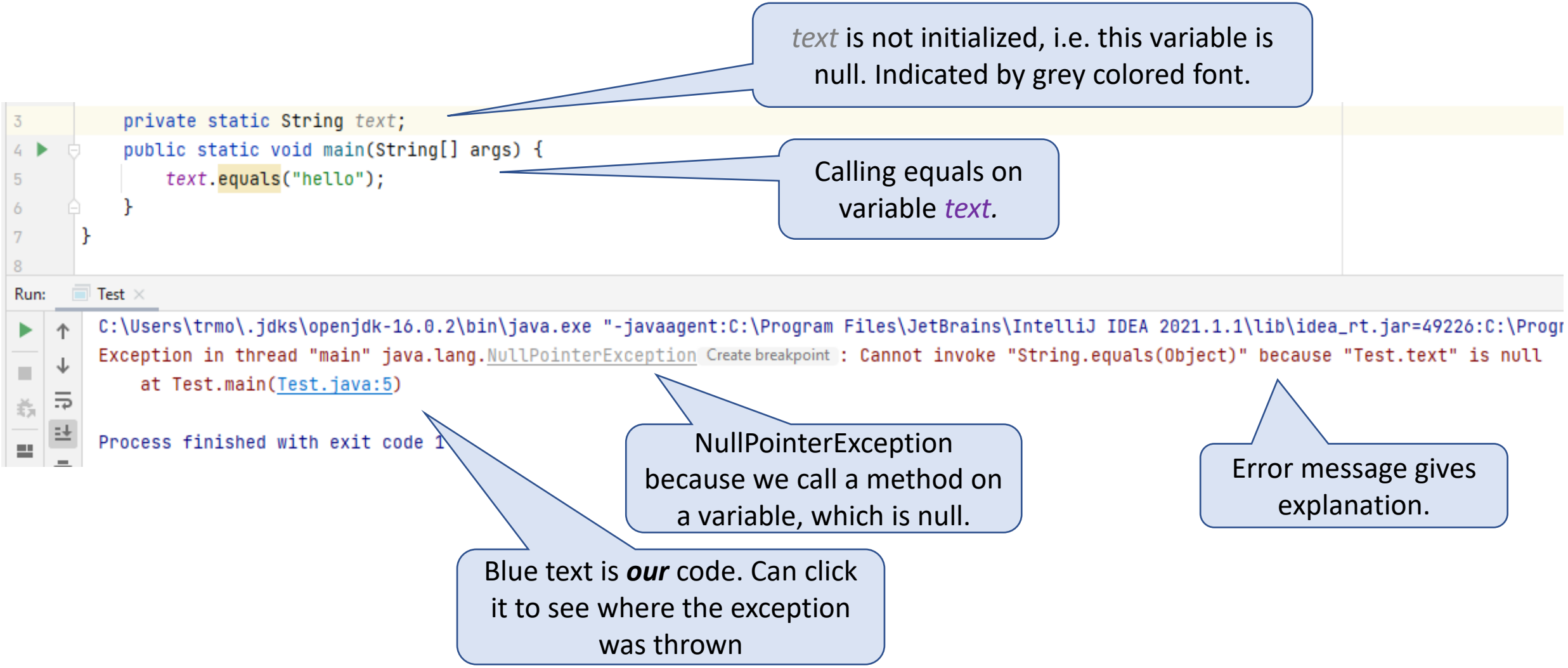
```
C:\Users\trmo\.jdks\openjdk-16.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.1.1\lib\idea_rt.jar=64471:C:
Exception in thread "main" java.lang.IndexOutOfBoundsException Create breakpoint : Index 1 out of bounds for length 0 <3 internal lines>
    at java.base/java.util.Objects.checkIndex(Objects.java:359)
    at java.base/java.util.ArrayList.get(ArrayList.java:427)
    at Test.main(Test.java:7)

Process finished with exit code 1
```

IndexOutOfBoundsException is thrown from get() method

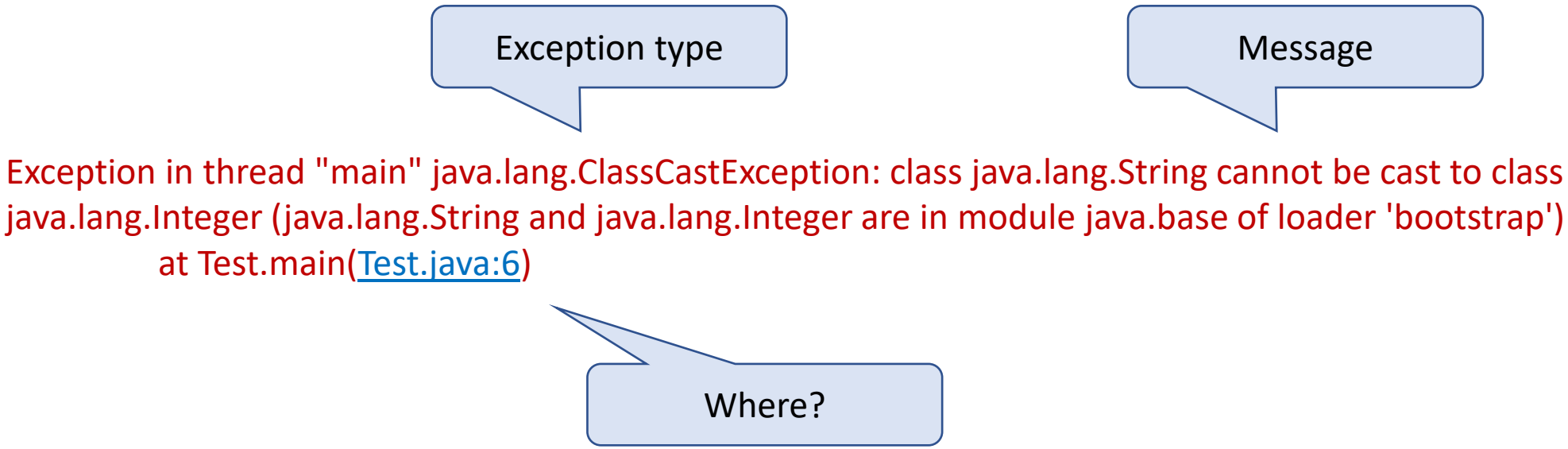
Error message gives explanation.

Blue text is ***our*** code. Can click it to see where the exception was thrown

What does it look like?



```
public static void main(String[] args) {
    Object s = "hello";
    Integer i = (Integer) s;
}
```



What does it look like?

- Often seen:
  - NullPointerExceptions in Controller or VM, because a field variable is not initiallised, or not passed through constructor parameter:
- Running javafx app:

Javafx app

Exception

Explanation

```
C:\Users\trmo\.jdks\openjdk-16.0.2\bin\java.exe ...
Oct 07, 2021 2:02:16 PM javafx.fxml.FXMLLoader$ValueElement   cessValue
WARNING: Loading FXML document with JavaFX API of versi   16 by JavaFX runtime of version 11.0.2
Exception in Application start method
Exception in thread "main" java.lang.RuntimeExcep  on Create breakpoint : Exception in Application start method
    at com.sun.javafx.application.LauncherImpl.launchApplication1(LauncherImpl.java:900)
    at com.sun.javafx.application.LauncherImpl.lambda$launchApplication$2(LauncherImpl.java:195) <1 internal line>
Caused by: java.lang.NullPointerException Create breakpoint : Cannot invoke "app.model.ThermostatModel.attachListener(String, java.beans.PropertyChangeListener)" because "this.model" is null
    at app.ui.ThermostatController.init(ThermostatController.java:33)
    at app.BaseThermostatApp.openView(BaseThermostatApp.java:26)
    at step3inheritance.InheritanceApp.start(InheritanceApp.java:14)
    at com.sun.javafx.application.LauncherImpl.lambda$launchApplication1$9(LauncherImpl.java:846)
    at com.sun.javafx.application.PlatformImpl.lambda$runAndWait$12(PlatformImpl.java:455)
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$10(PlatformImpl.java:428) <1 internal line>
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$11(PlatformImpl.java:427)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:96)
    at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
    at com.sun.glass.ui.win.WinApplication.lambda$runLoop$3(WinApplication.java:174)
    ... 1 more
```

My code. Click on top link

Red lines are java code, that I didn't write

Class A calls a method on class B.
Class B calls a method on class C.
Class C produces an exception.
We get this chain of calls. Aka a ***strack trace***

Javafx app

What's the problem?

```
26      private ThermostatModel model;
27
28      private ObservableList<Temperature> temps;
29
30      public void init(ThermostatModel m) {
31          temps = FXCollections.observableArrayList();
32
33          model.attachListener( evtName: "NewTemp", (evt) -> Platform.runLater(() ->
34                  onNewTemp(evt))
35          );
36          model.attachListener( evtName: "PowerChange", (evt) -> Platform.runLater(() ->
37                  onPowerChange(evt))
38          );
39
40          idColumn.setCellValueFactory(new PropertyValueFactory<>( s: "id"));
41          tempColumn.setCellValueFactory(new PropertyValueFactory<>( s: "value"));
42          unitColumn.setCellValueFactory(new PropertyValueFactory<>( s: "tempUnit"));
43          table.setItems(temps);
44      }
```

```
C:\Users\trmo\.jdks\openjdk-16.0.2\bin\java.exe ...
Oct 07, 2021 2:02:16 PM javafx.fxml.FXMLLoader$ValueElement processVal
WARNING: Loading FXML document with JavaFX API of version 16 by JavaFX runtime of version 11.0.2
Exception in Application start method
Exception in thread "main" java.lang.RuntimeException Create breakpoint : Exception in Application start method
    at com.sun.javafx.application.LauncherImpl.launchApplication1(LauncherImpl.java:900)
    at com.sun.javafx.application.LauncherImpl.lambda$launchApplication$2(LauncherImpl.java:195) <1 internal line>
Caused by: java.lang.NullPointerException Create breakpoint : Cannot invoke "app.model.ThermostatModel.attachListener(String, java.beans.PropertyChangeListener)" because "this.model" is null
    at app.ui.ThermostatController.init(ThermostatController.java:33)
    at app.BaseThermostatApp.openView(BaseThermostatApp.java:26)
    at step3inheritance.InheritanceApp.start(InheritanceApp.java:14)
    at com.sun.javafx.application.LauncherImpl.lambda$launchApplication1$9(LauncherImpl.java:846)
    at com.sun.javafx.application.PlatformImpl.lambda$runAndWait$12(PlatformImpl.java:455)
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$10(PlatformImpl.java:428) <1 internal line>
    at com.sun.javafx.application.PlatformImpl.lambda$runLater$11(PlatformImpl.java:427)
    at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:96)
    at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
    at com.sun.glass.ui.win.WinApplication.lambda$runLoop$3(WinApplication.java:174)
    ... 1 more
```

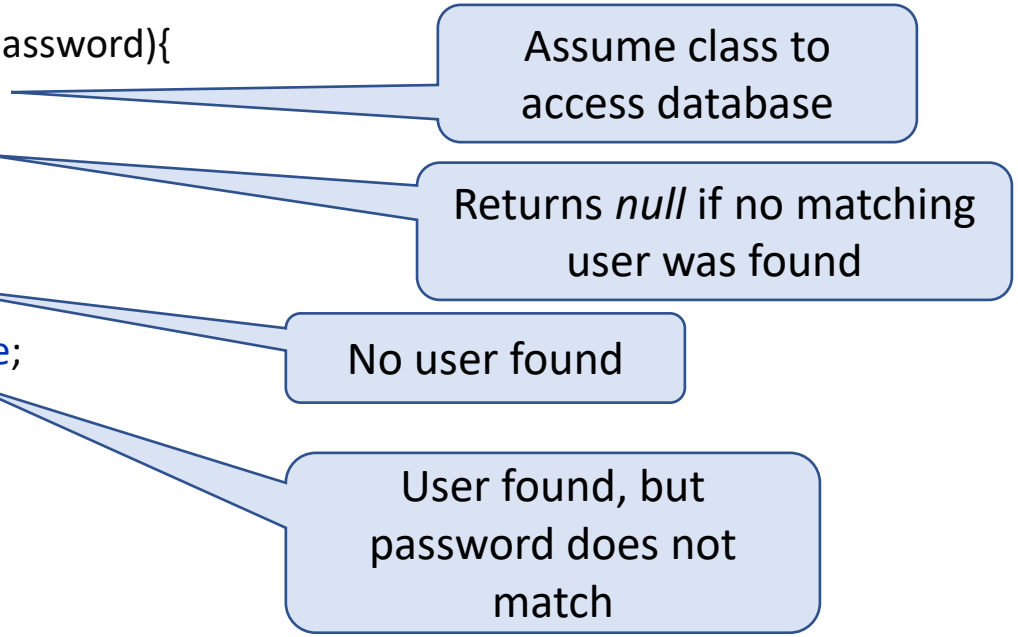
- ?

Purpose

- A good way to indicate
  - Something went wrong
  - Something was used incorrectly
  - Something should have been done, but wasn’t
  - Some rule was violated
  - Some logic was violated
  - You f’ed up

Example, logging in:

```
public boolean validateLogin(String username, String password){
    DatabaseAccess dba = new DatabaseAccess();
    User user = dba.getUserByUsername(username);

    if(user == null) return false;

    if(!user.getPassword().equals(password)) return false;

    return true;
}
```



Example, logging in:

```
public boolean validateLogin(String username, String password){
   DatabaseAccess dba = new DatabaseAccess();
   User user = dba.getUserByUsername(username);

   if(user == null) return false;

   if(!user.getPassword().equals(password)) return false;

   return true;
}
```

Now providing much more specific information

```
public User validateLogin2(String username, String password){
   DatabaseAccess dba = new DatabaseAccess();
   User user = dba.getUserByUsername(username);

   if(user == null) throw new RuntimeException("User not found");

   if(!user.getPassword().equals(password)) throw new RuntimeException("Incorrect password");

   return user;
}
```

Throw exception to indicate error

Return user object

Throw exception to indicate error

Password validation logic (Joseph's exercise)

```
public Password(String password) {

    if (password == null) {
        throw new IllegalArgumentException("Null password");
    }

    String message = Password.isLegal(password);

    if (message != null) {
        throw new IllegalArgumentException(message);
    }

    this.password = password;
}
```

Not allowing password to be null

Password not following rules, e.g. one upper, one lower, one digit.

Socket chat Server

Thread handling one chat-client

```
public class ServerSocketHandler implements Runnable {

    private ObjectInputStream inFromClient;
    private ObjectOutputStream outToClient;
    private Socket socket;
    private DatabaseAccess databaseAccess;
    private ChatModel chatModel;

    public ServerSocketHandler(Socket socket, DatabaseAccess databaseAccess, ChatModel chatModel) {
        this.socket = socket;
        this.databaseAccess = databaseAccess;
        this.chatModel = chatModel;
        try {
            inFromClient = new ObjectInputStream(socket.getInputStream());
            outToClient = new ObjectOutputStream(socket.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
```

Assume I can access a database for user info

Assume a chat model to keep track of chatting users

Socket connection can break, meaning streams cannot be created. IOException caught. Could probably be handled nicer

```
}

@Override
public void run() {
   try {
      stop : while (true) {
         Request req = (Request) inFromClient.readObject();
         String requestType = req.getRequest();
         Object arg = req.getArg();
         switch (requestType) {
            case "exit": {
               socket.close();
               chatModel.clientLoggedOff((User) arg);
               break stop;
            }
            case "login": {
               try {
                  User userInfo = (User) arg;
                  User user = databaseAccess.validateUser(userInfo.getUserName(), userInfo.getPassword());
                  outToClient.writeObject(user);
               } catch (RuntimeException e) {
                  outToClient.writeObject(new Response("ERROR", e.getMessage()));
               }
               break;
            }
            case "create_user": {
               try {
                  User userInfo = (User) arg;
```

Wrap everything in a try-catch. **Outside of while(true).** Loops can be named, here "stop"

Reading stuff from client.

What does the client want?

```
public class Request {
   private String request;
   private Object arg;

   public Request(String request, Object arg) {
      this.request = request;
      this.arg = arg;
   }
```

Breaking out of "stop"-while loop. Could also throw exception, to exit while loop. Or set a Boolean: while(running)

If ok, return the user object to client

Attempt to log in

If failure, return some kind of error object to client, with information about problem

```
            outToClient.writeObject(user);
        } catch (RuntimeException e) {
            outToClient.writeObject(new Response("ERROR", e.getMessage()));
        }
        break;
    }
    case "create_user": {
        try {
            User userInfo = (User) arg;
            databaseAccess.createUser(userInfo);
            outToClient.writeObject(new Response("SUCCESS","User created succesfully"));
        } catch (RuntimeException e) {
            outToClient.writeObject(new Response("ERROR", e.getMessage()));
        }
        break;
    }
    case "send_message": {
        try {
            Message msg = (Message) arg;
            chatModel.broadcast(msg);
            outToClient.writeObject(new Response("SUCCESS","Message sent"));
        } catch (RuntimeException e) {
            outToClient.writeObject(new Response("ERROR", "Could not send message"));
        }
        break;
    }
    case "private_message": {
        try {
            Message msg = (Message) arg;
```

Similar approach all the way down

May throw exception

Catch exception, return error message to client

- ?

Client side example

```
public class SocketClient implements Client{

   private Socket socket;
   private ObjectOutputStream outToServer;
   private ObjectInputStream inFromServer;

   public void start() {
      try {
         socket = new Socket("localhost", 2910);
      } catch (IOException e) {
         throw new RuntimeException("Couldn't connect to server, please try again later");
      }
   }

   private Object requestFromServer(Request req) {
      try {
         outToServer.writeObject(req);
         Response res = (Response)inFromServer.readObject();
         if("ERROR".equals(Response.getType())){
            throw new RuntimeException("Error: " + res.getArg());
         }
         return res.getArg();
      } catch (IOException | ClassNotFoundException e) {
         throw new RuntimeException("Problem contacting server: " + e.getMessage());
      }
   }

}
```

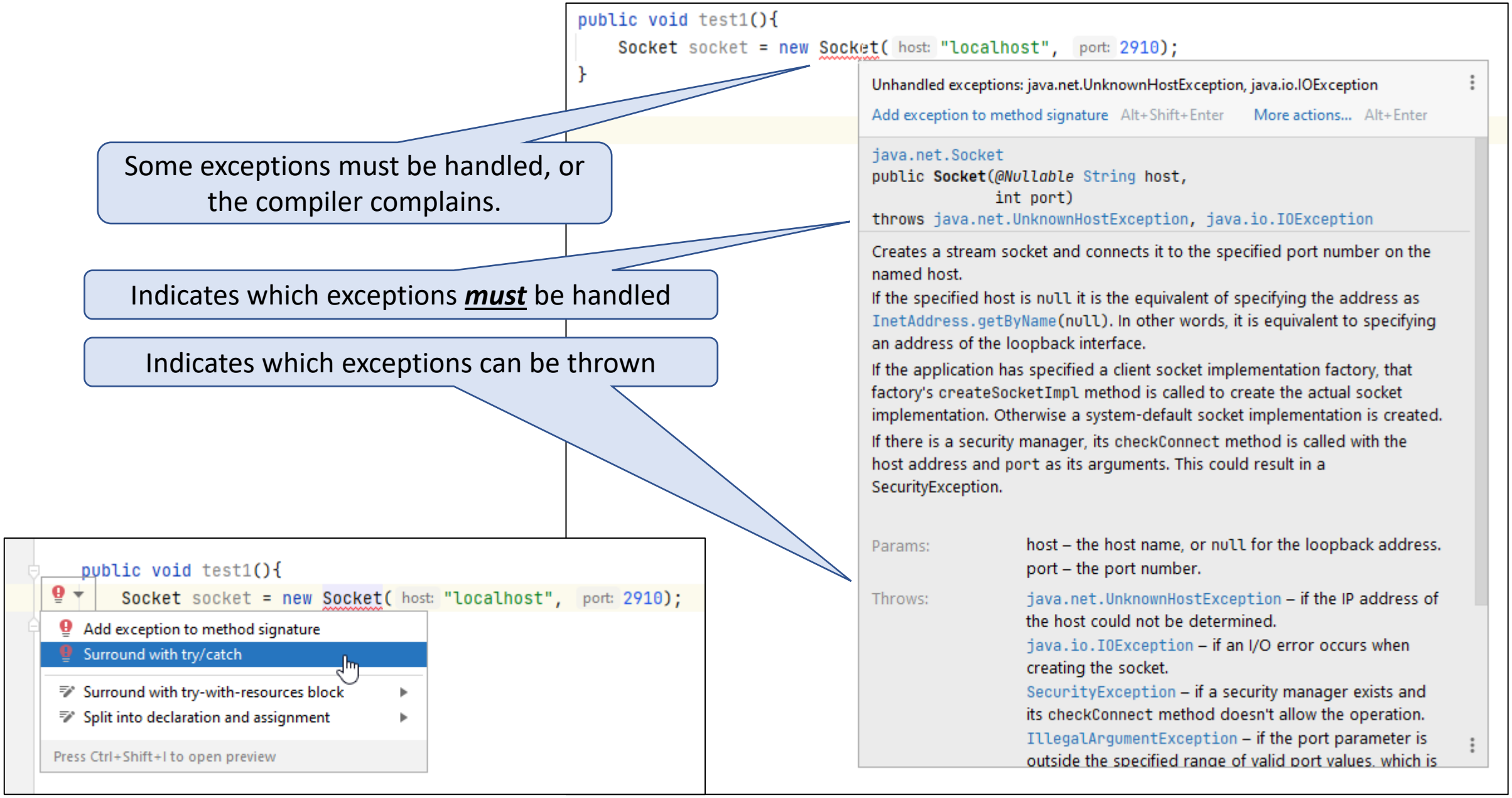
Changing from IOException to RuntimeException

Letting user know about error

If error happened on server

Letting user know about error

- ?


public void test1(){
Socket socket = new Socket(host: "localhost", port: 2910);
}
Some exceptions must be handled, or the compiler complains.
Indicates which exceptions must be handled
Indicates which exceptions can be thrown
Unhandled exceptions: java.net.UnknownHostException, java.io.IOException
Add exception to method signature Alt+Shift+Enter
More actions... Alt+Enter
java.net.Socket
public Socket(@Nullable String host,
int port)
throws java.net.UnknownHostException, java.io.IOException
Creates a stream socket and connects it to the specified port number on the named host.
If the specified host is null it is the equivalent of specifying the address as InetAddress.getByName(null). In other words, it is equivalent to specifying an address of the loopback interface.
If the application has specified a client socket implementation factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a system-default socket implementation is created.
If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.
Params:
host – the host name, or null for the loopback address.
port – the port number.
Throws:
java.net.UnknownHostException – if the IP address of the host could not be determined.
java.io.IOException – if an I/O error occurs when creating the socket.
SecurityException – if a security manager exists and its checkConnect method doesn't allow the operation.
IllegalArgumentException – if the port parameter is outside the specified range of valid port values, which is
public void test1(){
Socket socket = new Socket(host: "localhost", port: 2910);
Add exception to method signature
Surround with try/catch
Surround with try-with-resources block
Split into declaration and assignment
Press Ctrl+Shift+I to open preview

throws on method signature means another method must catch it

```
public void test1() throws IOException {
    Socket socket = new Socket("localhost", 2910);
}

public void test0(){
    try {
        test1();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void test2(){
    try {
        Socket socket = new Socket("localhost", 2910);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Like here

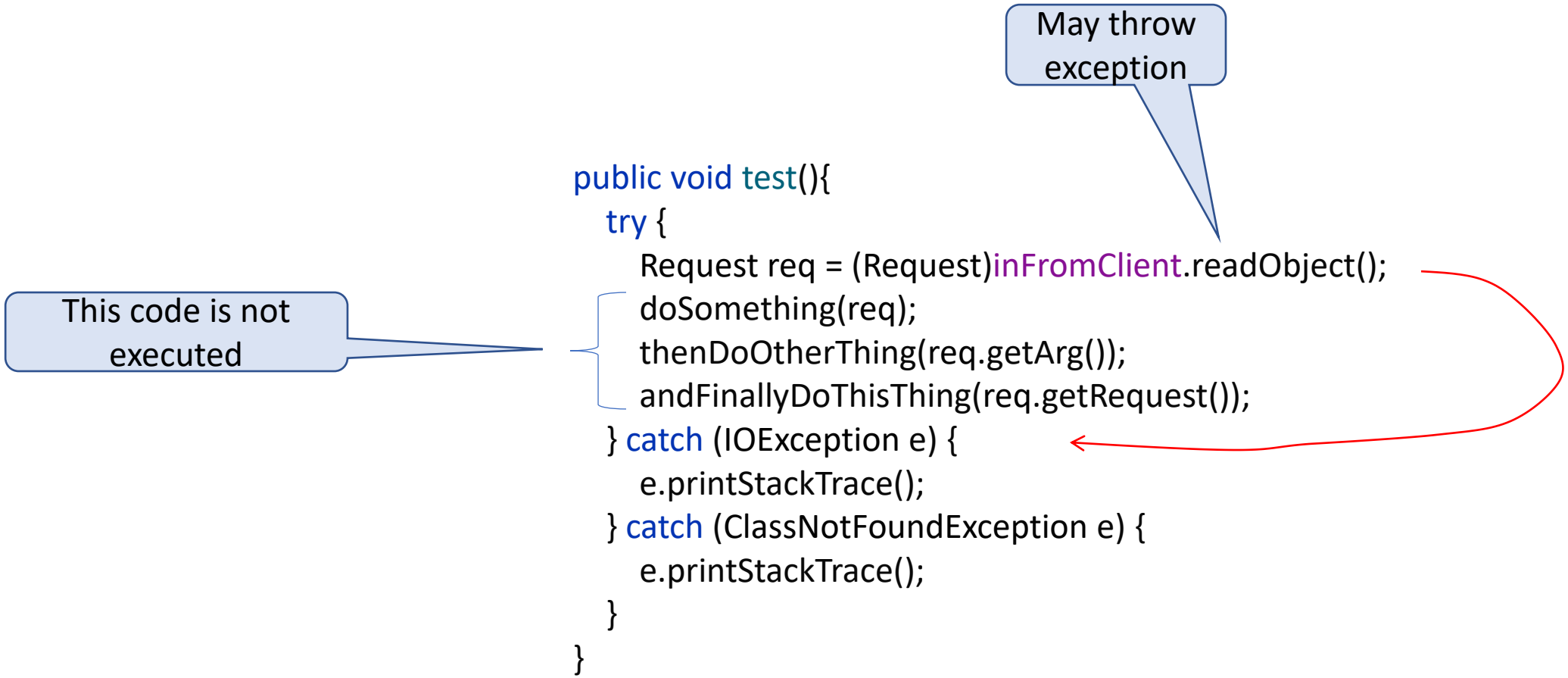
Adding throws just mean someone else has to deal with it. Rarely a good idea.

Alternatively

Like previously, could throw a new RuntimeException("…")

Lazy solution is to just print stack trace. This will, however, not inform the user about any problems

Exceptions skip over code



Checked vs Unchecked exceptions

- **Checked** exceptions *must* be handled either with *throws* in method signature, or surrounded with try-catch.
  Otherwise the compiler will complain.
  - IOException
  - RemoteException
  - ClassNotFoundException

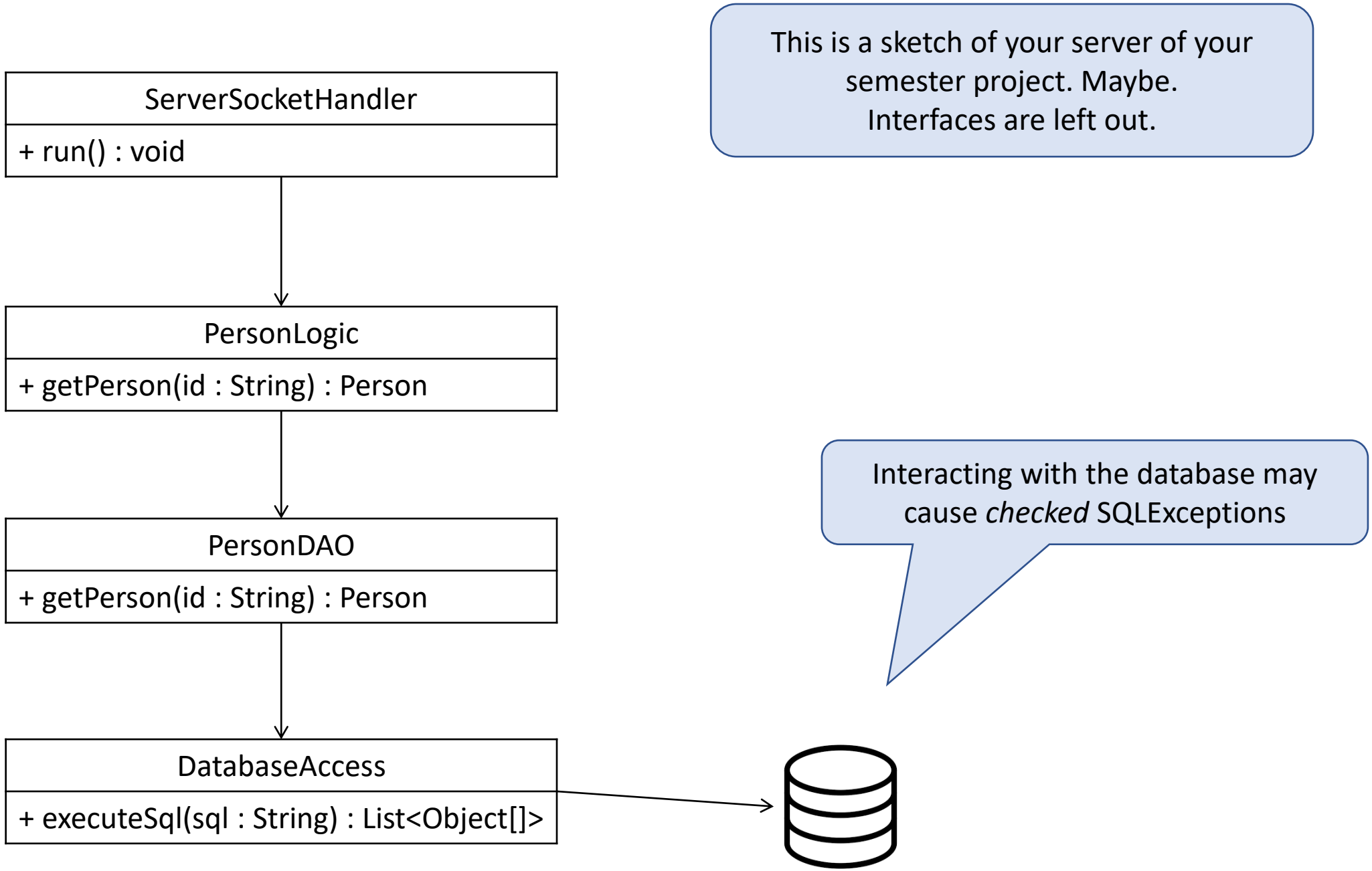
- **Unchecked** exceptions extends RuntimeException. The compiler will not require these to be handled:
  - NullpointerException
  - ArrayIndexOutOfBoundsException
  - IllegalArgumentException
  - IllegalStateException

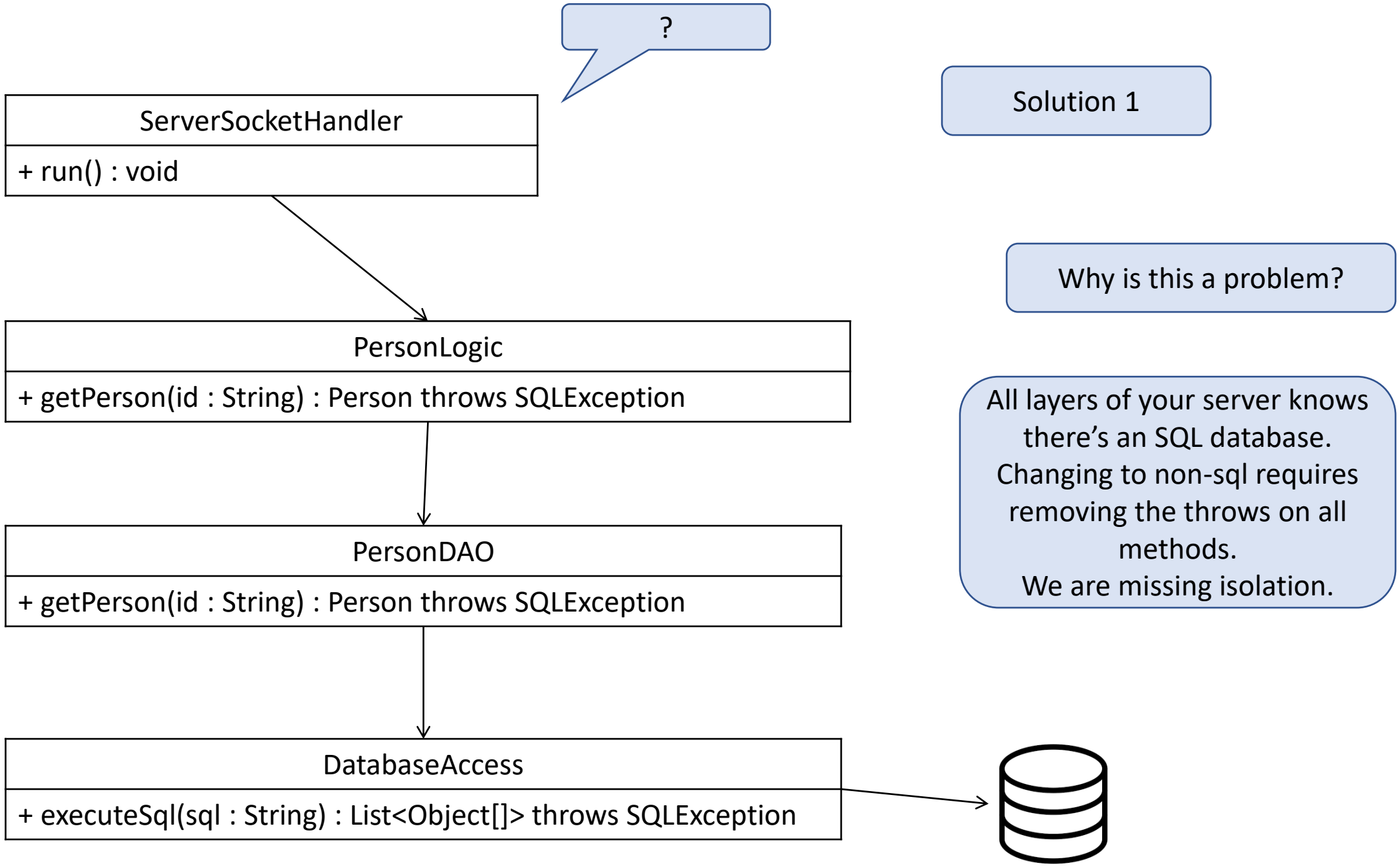
Throws vs try-catch

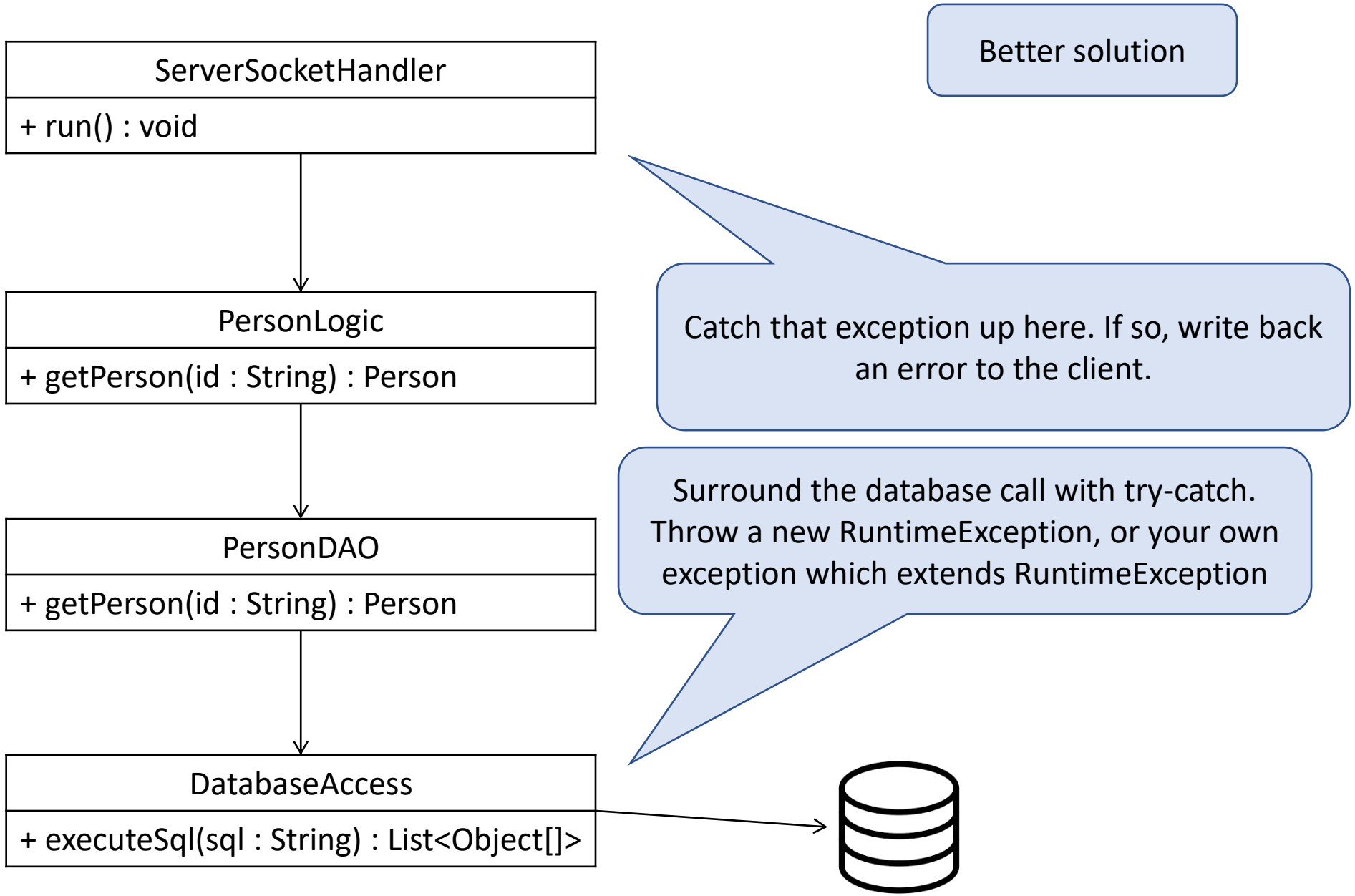
- In general, use try-catch, instead of adding *throws* to method signature

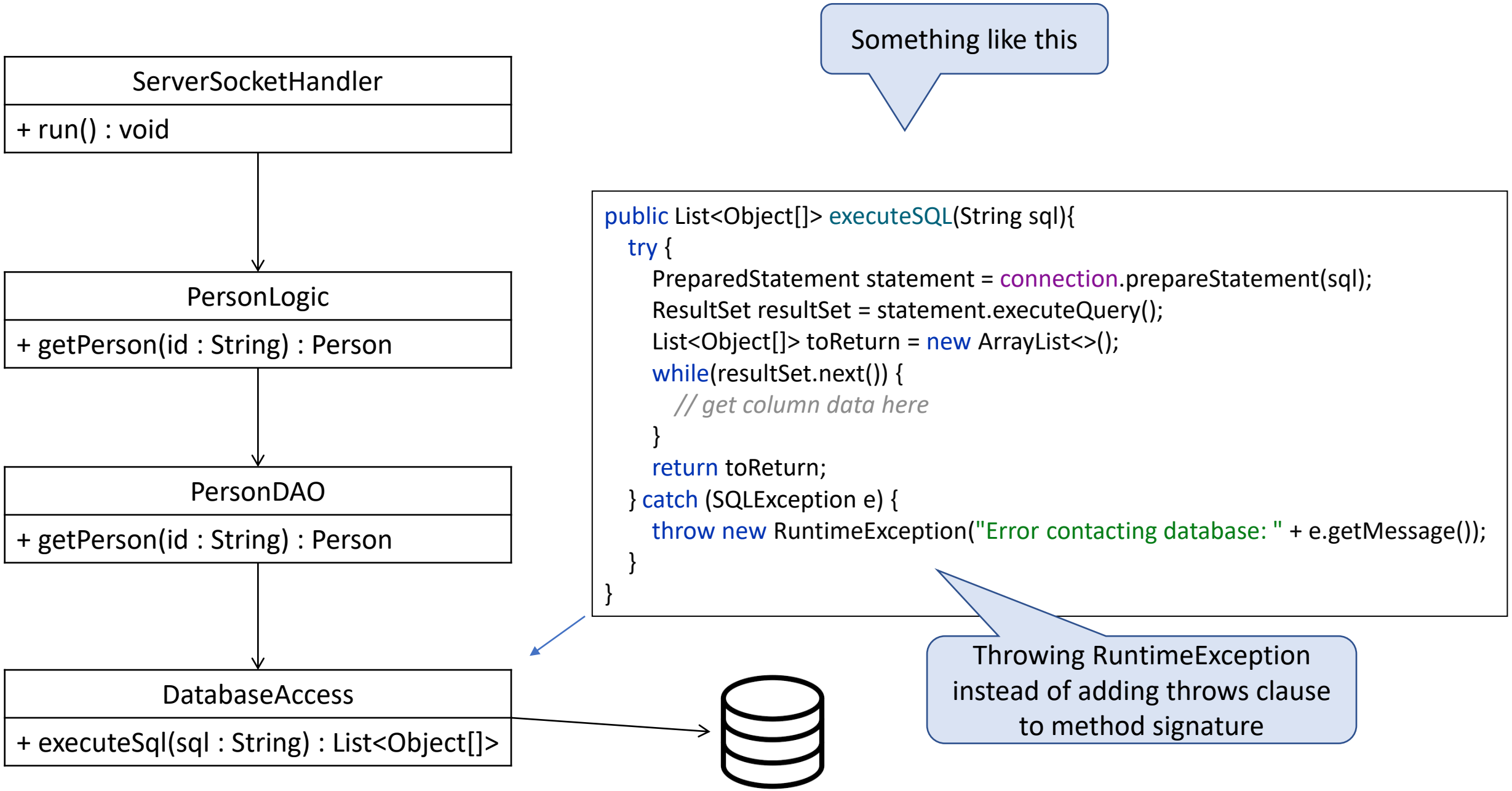
- Why? → let's see

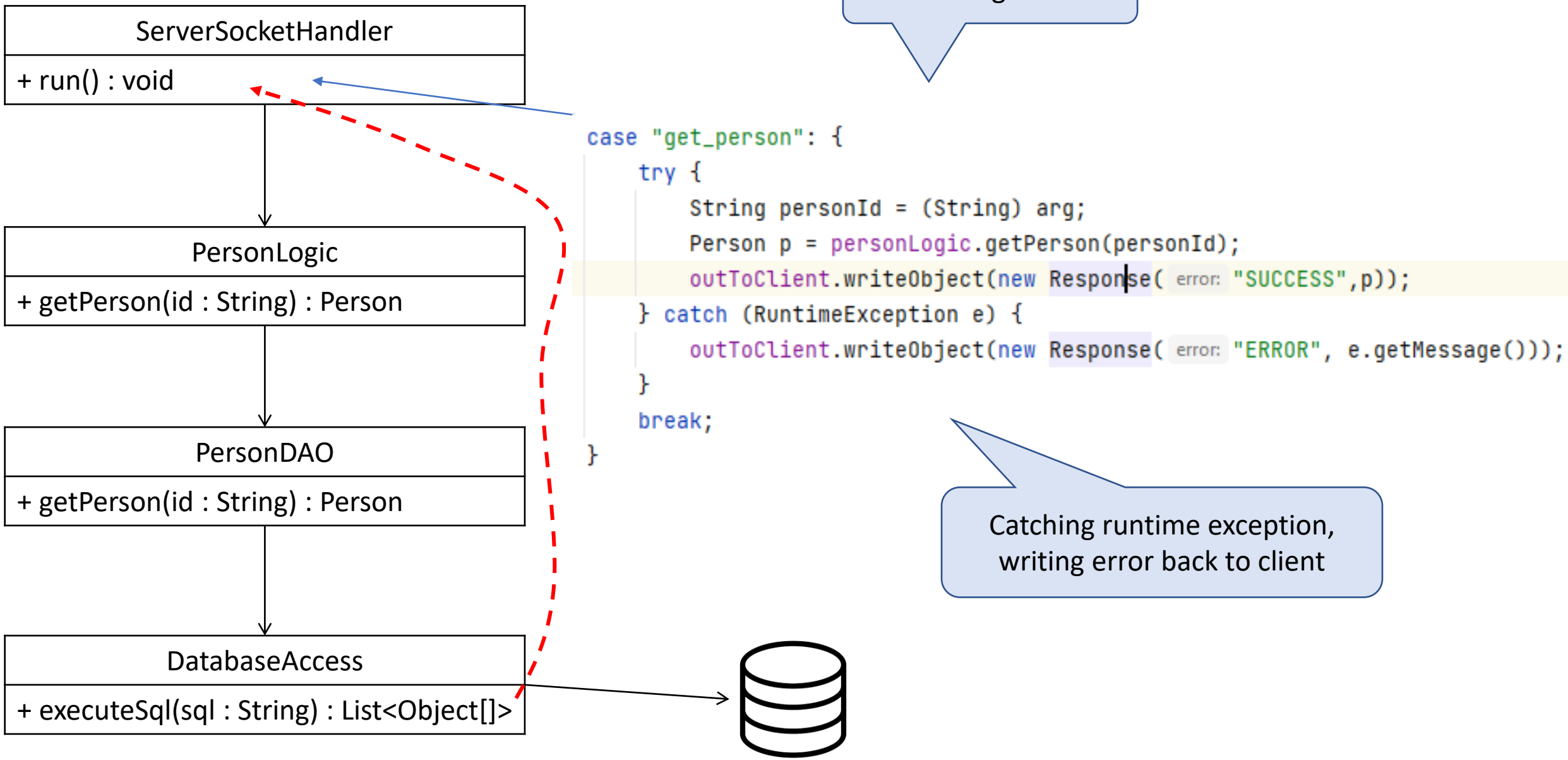
- ?


This is a sketch of your server of your semester project. Maybe.
Interfaces are left out.
ServerSocketHandler
+ run() : void
PersonLogic
+ getPerson(id : String) : Person
PersonDAO
+ getPerson(id : String) : Person
DatabaseAccess
+ executeSql(sql : String) : List<Object[]>
Interacting with the database may cause *checked* SQLExceptions


?
ServerSocketHandler
+ run() : void
PersonLogic
+ getPerson(id : String) : Person throws SQLException
PersonDAO
+ getPerson(id : String) : Person throws SQLException
DatabaseAccess
+ executeSql(sql : String) : List<Object[]> throws SQLException
Solution 1
Why is this a problem?
All layers of your server knows there's an SQL database. Changing to non-sql requires removing the throws on all methods.
We are missing isolation.

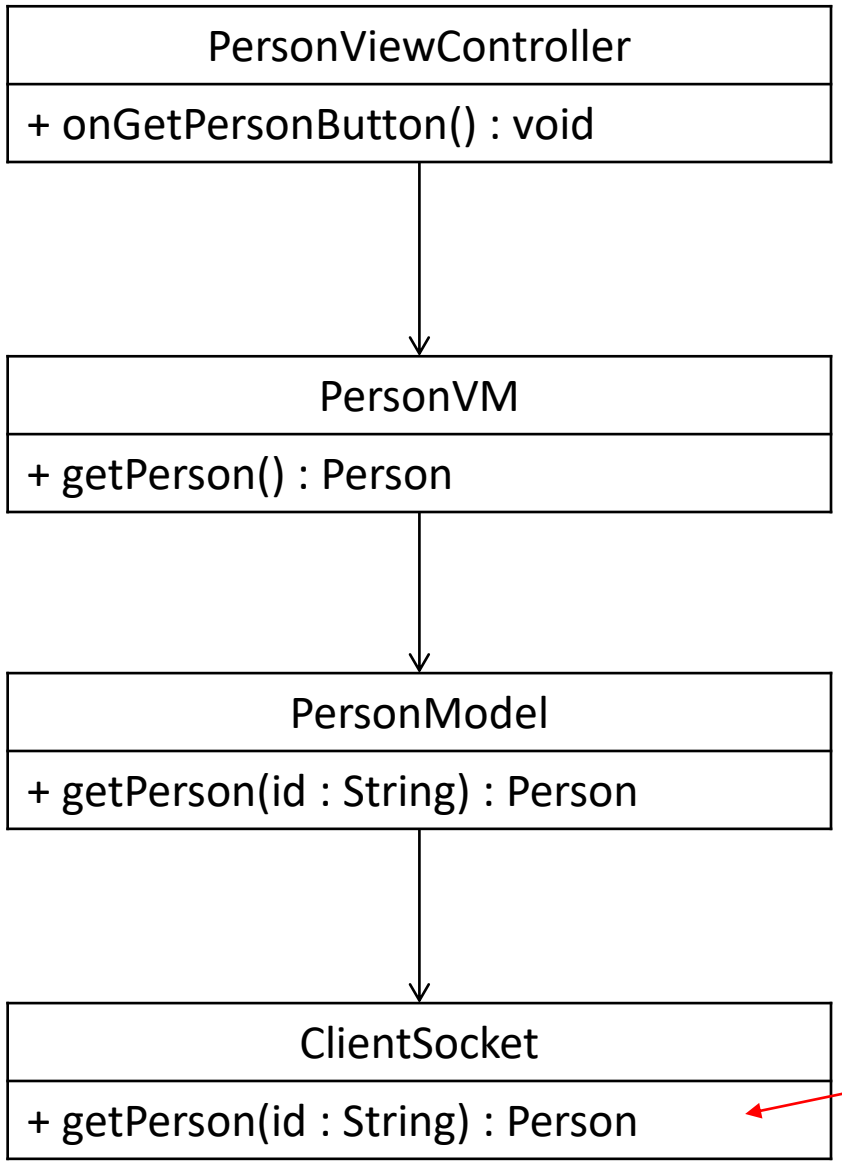

Better solution
ServerSocketHandler
+ run() : void
PersonLogic
+ getPerson(id : String) : Person
PersonDAO
+ getPerson(id : String) : Person
DatabaseAccess
+ executeSql(sql : String) : List<Object[]>
Catch that exception up here. If so, write back an error to the client.
Surround the database call with try-catch. Throw a new RuntimeException, or your own exception which extends RuntimeException


ServerSocketHandler
+ run() : void
PersonLogic
+ getPerson(id : String) : Person
PersonDAO
+ getPerson(id : String) : Person
DatabaseAccess
+ executeSql(sql : String) : List<Object[]>
Something like this
public List<Object[]> executeSQL(String sql){
try {
PreparedStatement statement = connection.prepareStatement(sql);
ResultSet resultSet = statement.executeQuery();
List<Object[]> toReturn = new ArrayList<>();
while(resultSet.next()) {
// get column data here
}
return toReturn;
} catch (SQLException e) {
throw new RuntimeException("Error contacting database: " + e.getMessage());
}
}
Throwing RuntimeException instead of adding throws clause to method signature


ServerSocketHandler
+ run() : void
PersonLogic
+ getPerson(id : String) : Person
PersonDAO
+ getPerson(id : String) : Person
DatabaseAccess
+ executeSql(sql : String) : List<Object[]>
Something like this
case "get_person": {
try {
String personId = (String) arg;
Person p = personLogic.getPerson(personId);
outToClient.writeObject(new Response(error: "SUCCESS",p));
} catch (RuntimeException e) {
outToClient.writeObject(new Response(error: "ERROR", e.getMessage()));
}
break;
}
Catching runtime exception,
writing error back to client

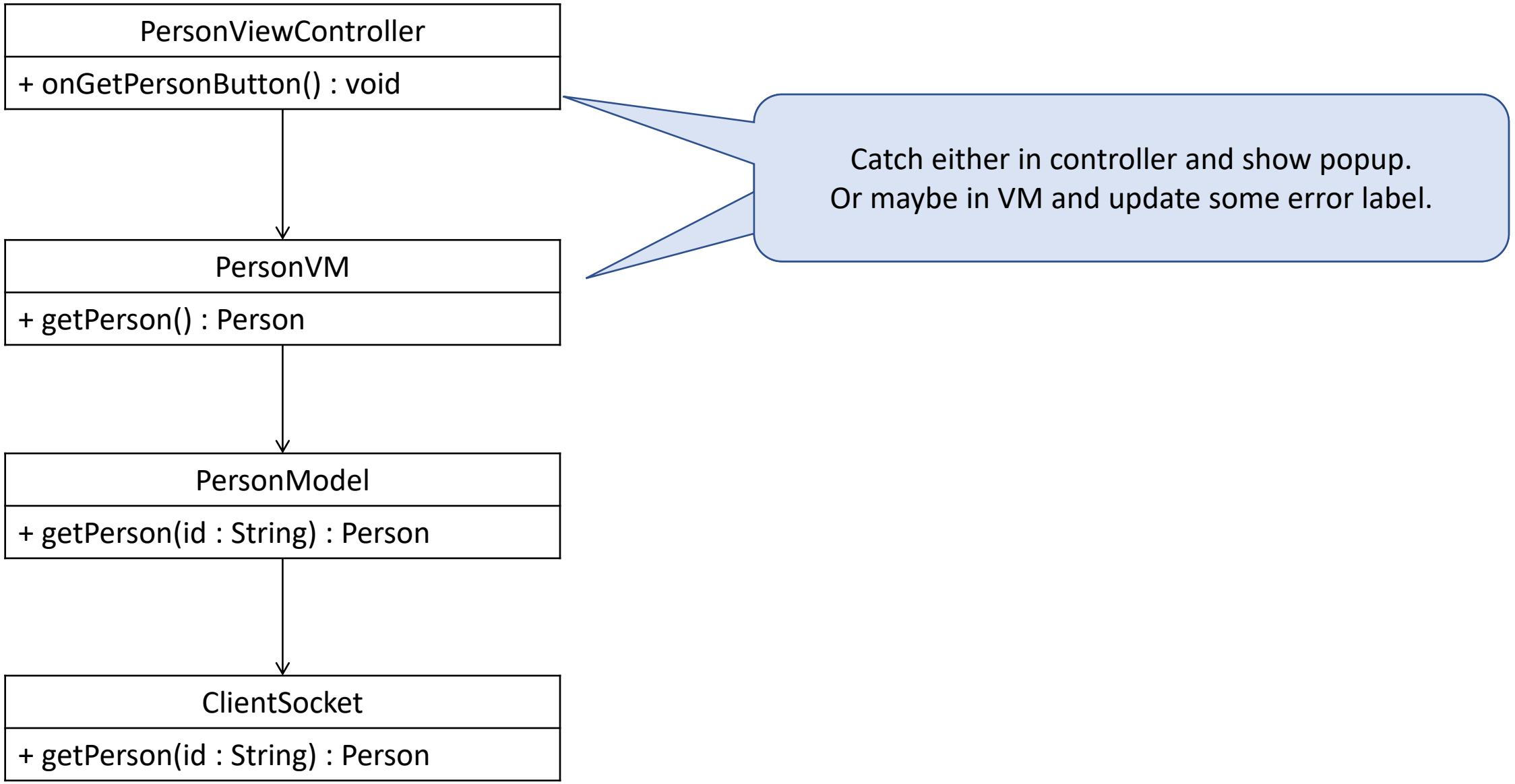
- ?

Client side




```
public Person getPerson(String id){
    Response response = null;
    try {
        Socket s = new Socket("localhost", 2910);
        ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(s.getInputStream());
        out.writeObject(new Request("get_person", id));
        response = (Response) in.readObject();

        if("ERROR".equals(response.getType())){
            throw new RuntimeException("Error: " + response.getArg());
        }
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
        throw new RuntimeException("Error: " + e.getMessage());
    }
    return (Person)response.getArg();
}
```

Client side


PersonViewController
+ onGetPersonButton() : void
PersonVM
+ getPerson() : Person
PersonModel
+ getPerson(id : String) : Person
ClientSocket
+ getPerson(id : String) : Person
Catch either in controller and show popup.
Or maybe in VM and update some error label.

- ?

When RuntimeException is not specific enough

```
public class PersonNotFoundException extends RuntimeException{

}

public class IncorrectIdFormatException extends RuntimeException {

}
```

You don't necessarily need to add more to these classes

```
public void getPerson(String id) {
   Person p;
   try {
      p = dataAccess.getPersonById(id);
      nameProperty.setValue(p.getName());
   } catch (PersonNotFoundException e) {
      // handle this case some way
   } catch (IncorrectIdFormatException e) {
      // catch this some other way
   }
}
```

- ?

Never leave a catch-block empty

Unless you really know what you’re doing