Analysis and Implementation of Database Using Aspects

Murat Kurtcephe
Oğuzcan Oğuz
Mehmet Özçelik

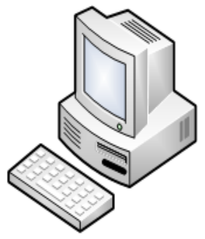
Third Turkish Aspect-Oriented Software Development Workshop (Bilkent Ankara 12/2008)

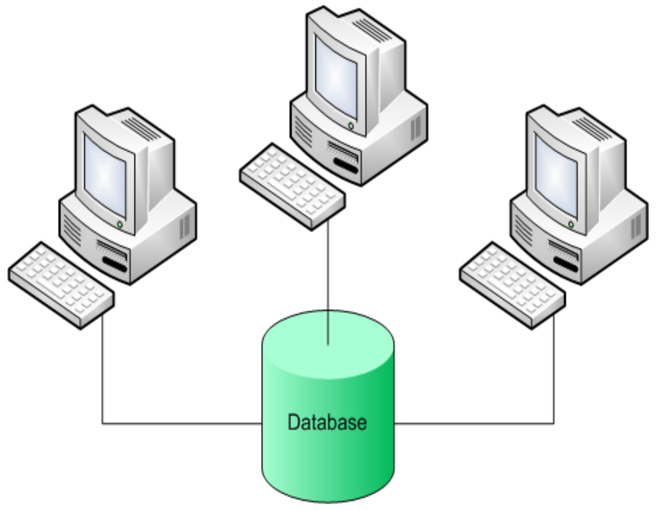
Introduction

- Database implementation on legacy code
  - Persistence
  - Synchronization
  - Failure Management
  - Refresh from database

Case Study: Car Rental System

- Case: Existing “Car Rental System” coded in Java
- Multi-user design without persistency concern

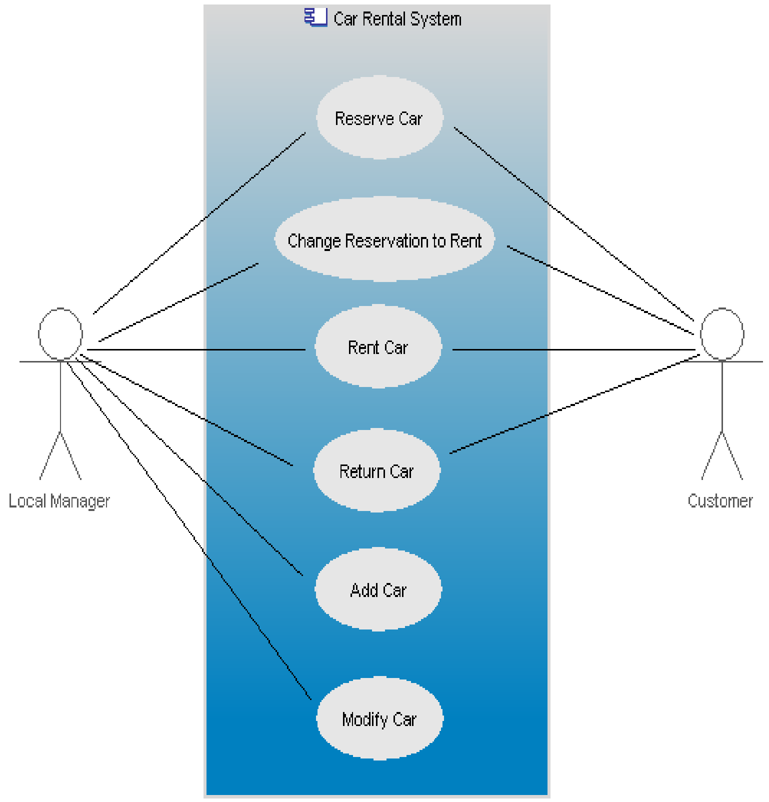

The Initial Single User System



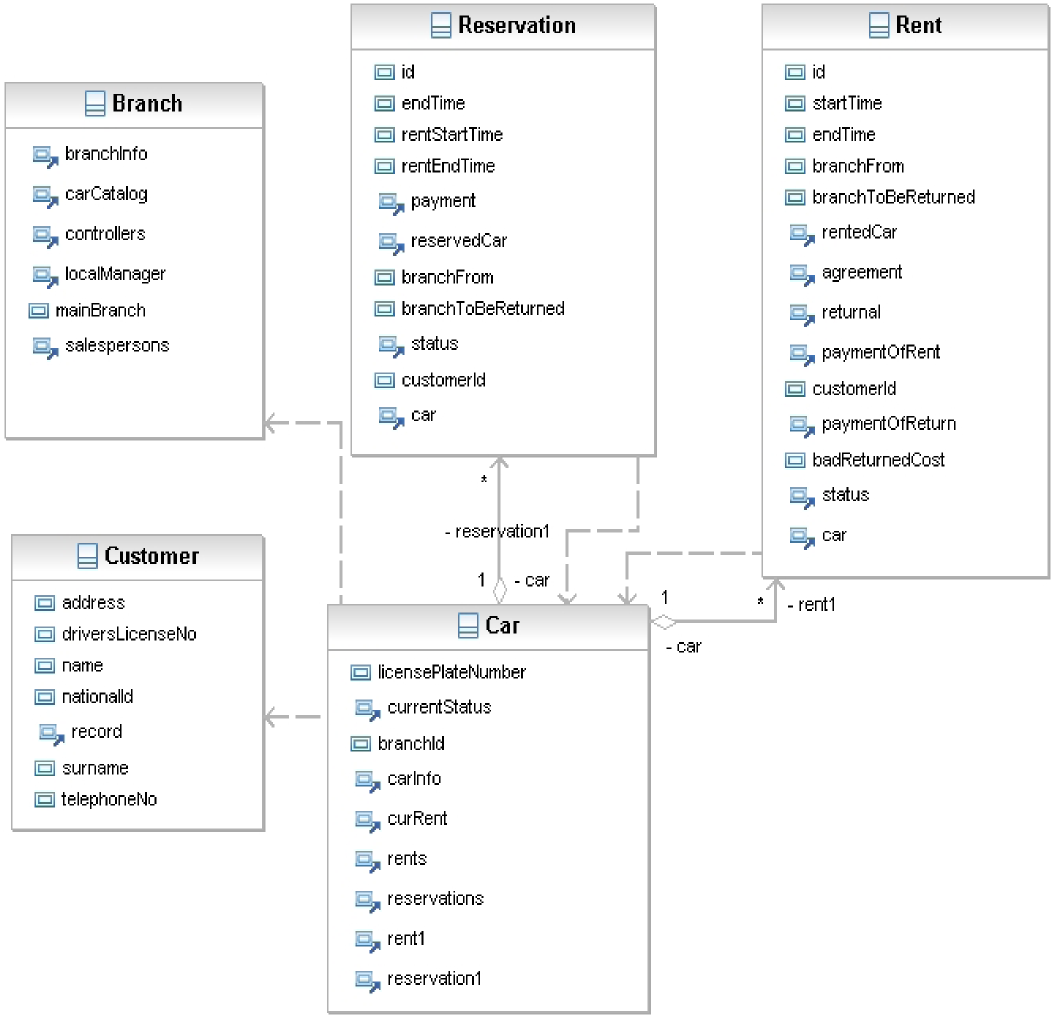
Multi-User Persistent System

Car Rental System

- Users perform rents, reservations and returns
- Information of branches, cars, customers etc. can be modified.
- Different user profiles with different priviliges

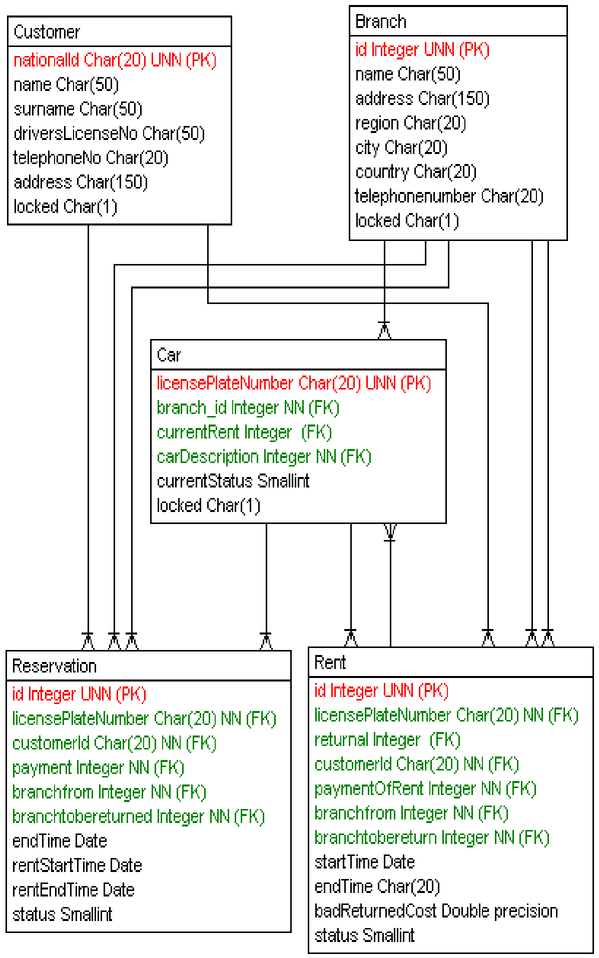


Class Diagram

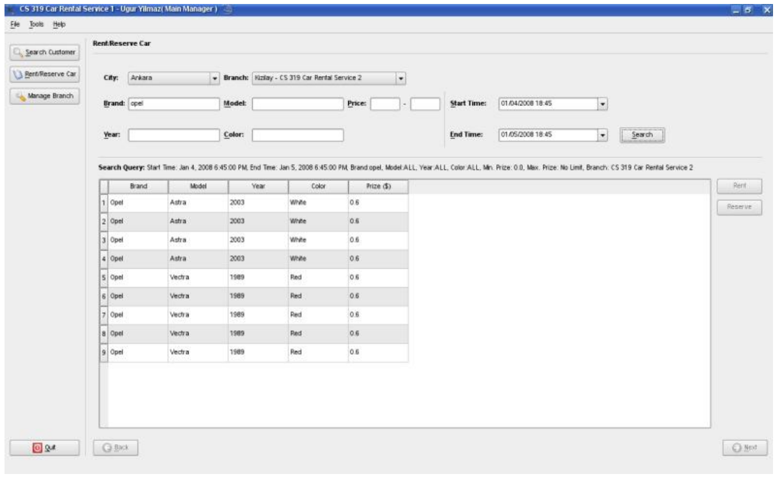


Entity Relationship Design

- Relational database assumption.
- ER model capable of storing all states and application data.



Requirement Analysis

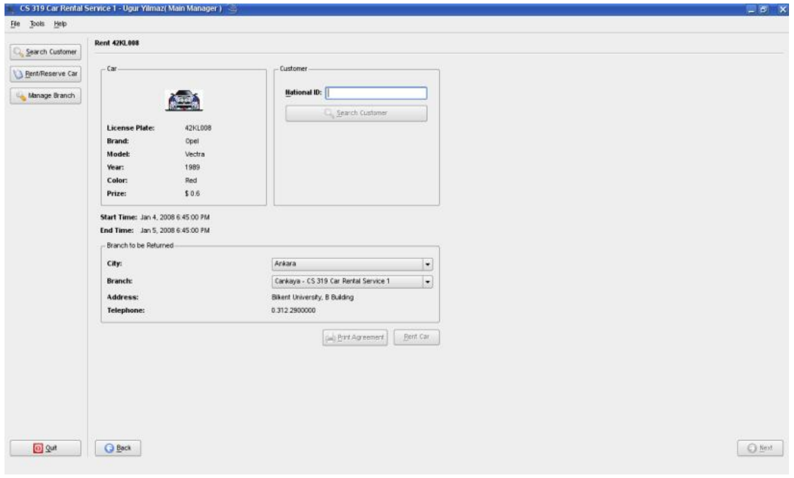


Rent/Reserve Car

City: Ankara
Branch: Kizilay - CS 319 Car Rental Service 2
Brand: opel
Model:
Price: -
Start Time: 01/04/2008 18:45
Year:
Color:
End Time: 01/05/2008 18:45

Search Query: Start Time: Jan 4, 2008 6:45:00 PM, End Time: Jan 5, 2008 6:45:00 PM, Brand:opel, Model:ALL, Year:ALL, Color:ALL, Min. Prize: 0.0, Max. Prize: No Limit, Branch: CS 319 Car Rental Service 2

	Brand	Model	Year	Color	Prize ($)
1	Opel	Astra	2003	White	0.6
2	Opel	Astra	2003	White	0.6
3	Opel	Astra	2003	White	0.6
4	Opel	Astra	2003	White	0.6
5	Opel	Vectra	1989	Red	0.6
6	Opel	Vectra	1989	Red	0.6
7	Opel	Vectra	1989	Red	0.6
8	Opel	Vectra	1989	Red	0.6
9	Opel	Vectra	1989	Red	0.6



Rent 42KL008

Car

License Plate:	42KL008
Brand:	Opel
Model:	Vectra
Year:	1989
Color:	Red
Prize:	$ 0.6

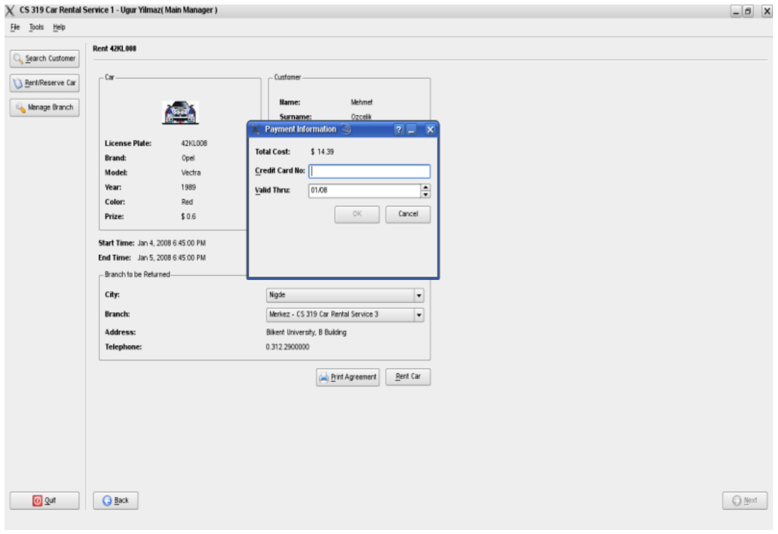
Customer

National ID:

Start Time: Jan 4, 2008 6:45:00 PM
End Time: Jan 5, 2008 6:45:00 PM

Branch to be Returned

City:	Ankara
Branch:	Cankaya - CS 319 Car Rental Service 1
Address:	Bilkent University, B Building
Telephone:	0.312 2900000



Rent 42KL008

Car

License Plate:	42KL008
Brand:	Opel
Model:	Vectra
Year:	1989
Color:	Red
Prize:	$ 0.6

Customer

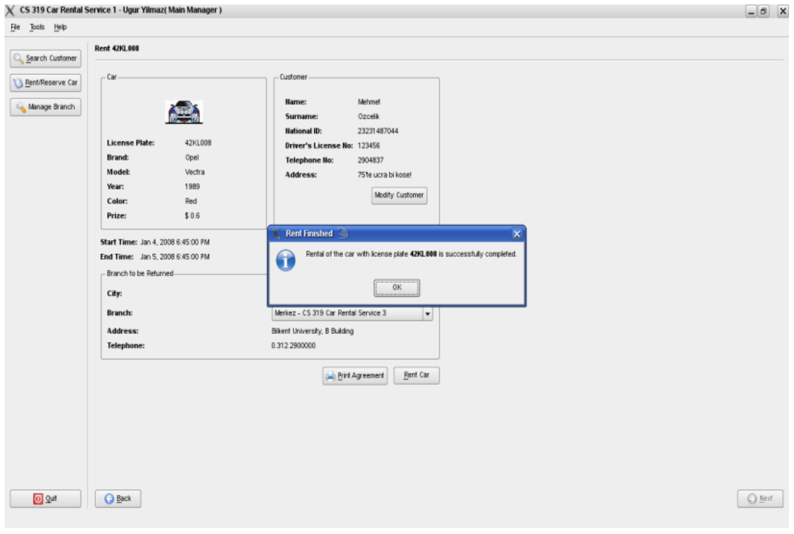
Name: Mehmet
Surname: Ozcelik

Total Cost: $ 14.39
Credit Card No:
Valid Thru: 01/08

Start Time: Jan 4, 2008 6:45:00 PM
End Time: Jan 5, 2008 6:45:00 PM

Branch to be Returned

City:	Nigde
Branch:	Merkez - CS 319 Car Rental Service 3
Address:	Bilkent University, B Building
Telephone:	0.312 2900000



Rent 42KL008

Car

License Plate:	42KL008
Brand:	Opel
Model:	Vectra
Year:	1989
Color:	Red
Prize:	$ 0.6

Customer

Name:	Mehmet
Surname:	Ozcelik
National ID:	23231487044
Driver's License No:	123456
Telephone No:	2904837
Address:	75'te ucra bi kose!

Rental of the car with license plate **42KL008** is successfully completed.

Start Time: Jan 4, 2008 6:45:00 PM
End Time: Jan 5, 2008 6:45:00 PM

Branch to be Returned

City:	
Branch:	Merkez - CS 319 Car Rental Service 3
Address:	Bilkent University, B Building
Telephone:	0.312 2900000

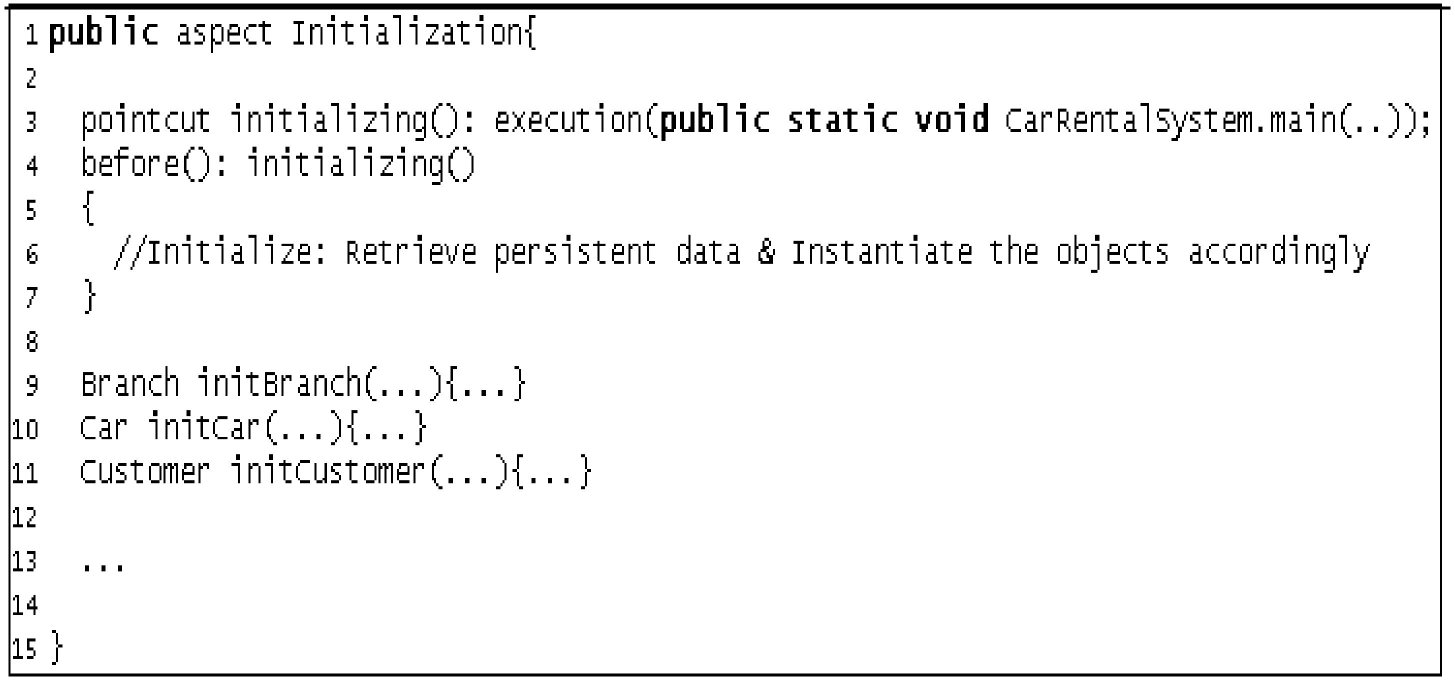
Crosscutting Concerns

- Persistence
  - Initialization
  - Add
  - Update
  - Delete
- Synchronization
- Faul management
- Refresh

Persistence – Initialization

- Retrieves the persistent data & instantiates required objects
- Modularization of instantiation of every object
- Order of instantiation to minimize revisiting
- Care need to be taken not to trigger other aspects trapping store and update operations

Persistence – Initialization



```
public aspect Initialization{

  pointcut initializing(): execution(public static void CarRentalSystem.main(..));
  before(): initializing()
  {
    //Initialize: Retrieve persistent data & Instantiate the objects accordingly
  }

  Branch initBranch(...){...}
  Car initCar(...){...}
  Customer initCustomer(...){...}

  ...

}
```

Persistence – Add operations

- Objects held in the arrays
- Access add methods of the arrays
- Change in the legacy code
- Requires code scattering
- Lowers modularity

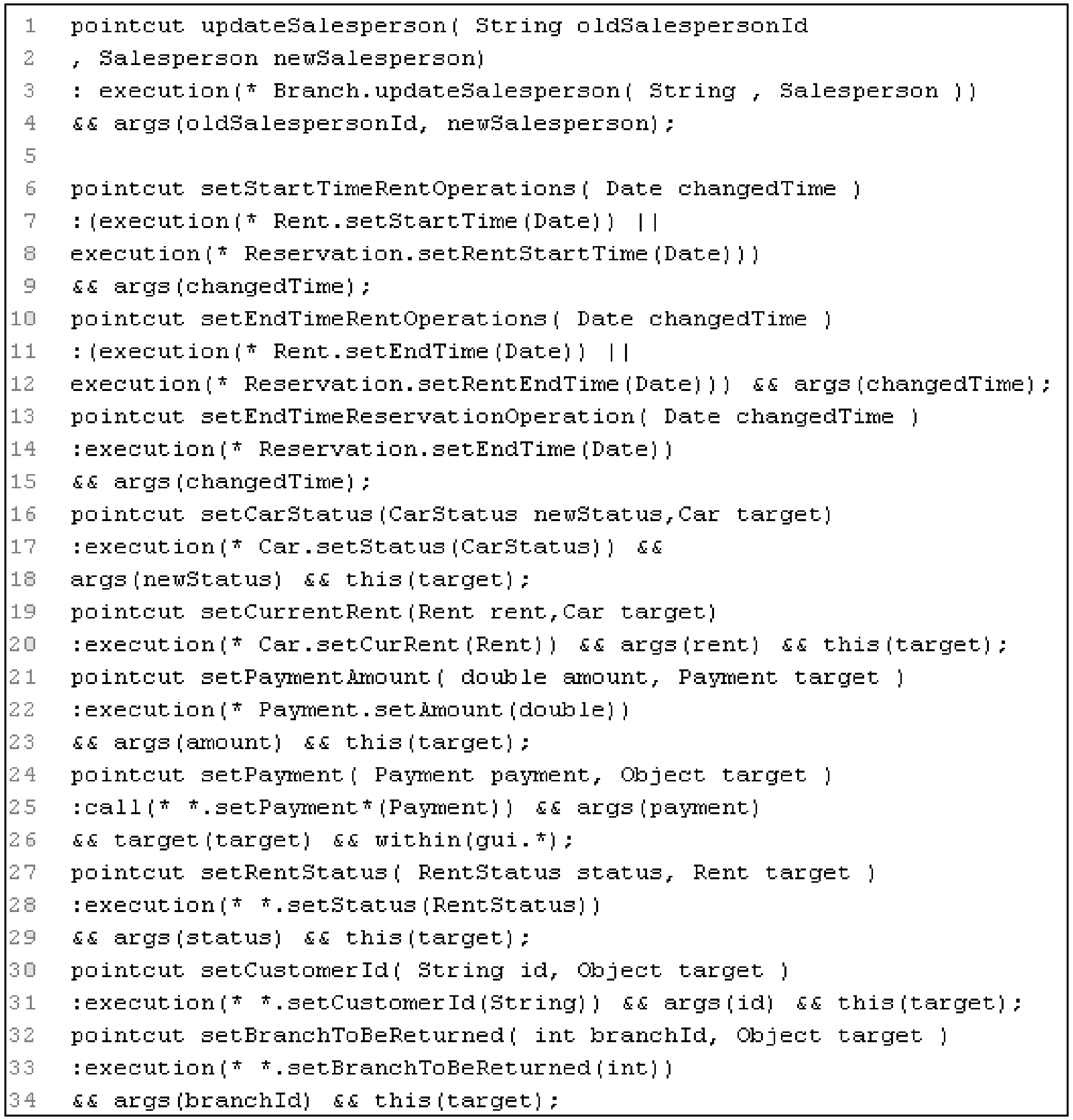
Add operations Aspect Code

```
pointcut putMethods( Branch branch, Object addedObject ) :
    call( * *.put( Object, Object ) ) && this( branch )
    && args( Object, addedObject ) && !withincode( * *.get*(..) )
    && within( core.* );

after( Branch branch, Object addedObject ) : putMethods ( branch, addedObject ) {
    if ( addedObject instanceof Salesperson ) {
        ...
    }
    else if ( addedObject instanceof Customer ) {
        ...
    }
    else if ( addedObject instanceof Controller ) {
        ...
    }
}
```

Persistence – Update operations

- For atomic operations such as updates
- Some of the set operations
- Not very easy to find pointcuts

Update Operations Aspect Code



```
pointcut updateSalesperson( String oldSalespersonId
, Salesperson newSalesperson)
: execution(* Branch.updateSalesperson( String , Salesperson ))
&& args(oldSalespersonId, newSalesperson);

pointcut setStartTimeRentOperations( Date changedTime )
:(execution(* Rent.setStartTime(Date)) ||
execution(* Reservation.setRentStartTime(Date)))
&& args(changedTime);
pointcut setEndTimeRentOperations( Date changedTime )
:(execution(* Rent.setEndTime(Date)) ||
execution(* Reservation.setRentEndTime(Date))) && args(changedTime);
pointcut setEndTimeReservationOperation( Date changedTime )
:execution(* Reservation.setEndTime(Date))
&& args(changedTime);
pointcut setCarStatus(CarStatus newStatus,Car target)
:execution(* Car.setStatus(CarStatus)) &&
args(newStatus) && this(target);
pointcut setCurrentRent(Rent rent,Car target)
:execution(* Car.setCurRent(Rent)) && args(rent) && this(target);
pointcut setPaymentAmount( double amount, Payment target )
:execution(* Payment.setAmount(double))
&& args(amount) && this(target);
pointcut setPayment( Payment payment, Object target )
:call(* *.setPayment*(Payment)) && args(payment)
&& target(target) && within(gui.*);
pointcut setRentStatus( RentStatus status, Rent target )
:execution(* *.setStatus(RentStatus))
&& args(status) && this(target);
pointcut setCustomerId( String id, Object target )
:execution(* *.setCustomerId(String)) && args(id) && this(target);
pointcut setBranchToBeReturned( int branchId, Object target )
:execution(* *.setBranchToBeReturned(int))
&& args(branchId) && this(target);
```

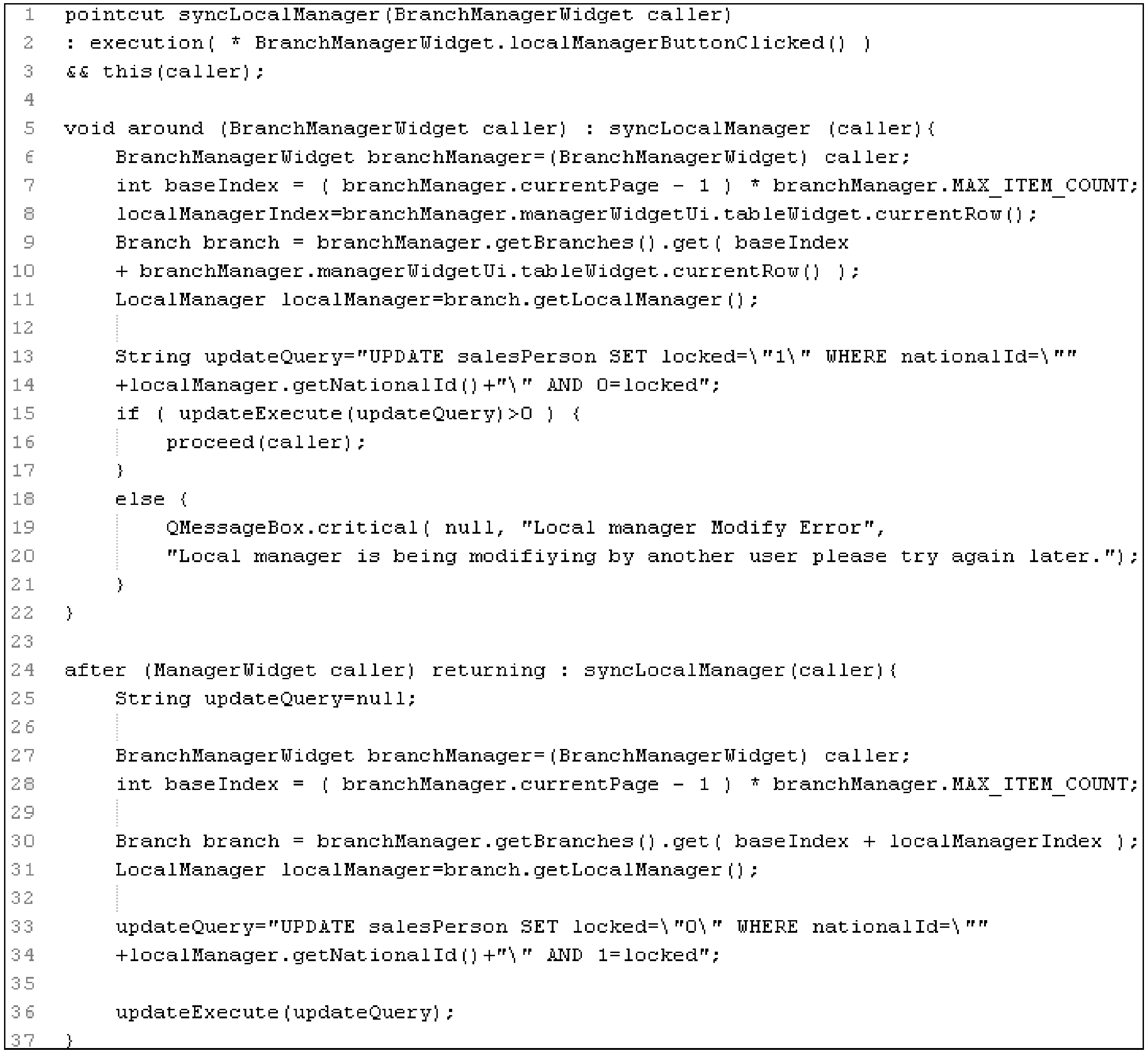
Persistence – Delete operations

- Case specific concerns:
  - Application data is a local copy of persistent data
  - Every object is stored in main memory for once
  - Only rents and reservations are removed
- Not always the case:
  - Non-fuzzy deletions requires deletions to be specifically declared
  - Shared persistent data should not be removed

Synchronization

- Takes each modify operation as a pointcut
- For checking when modifying finishes it listens GUI operations

Synronization Aspect Code

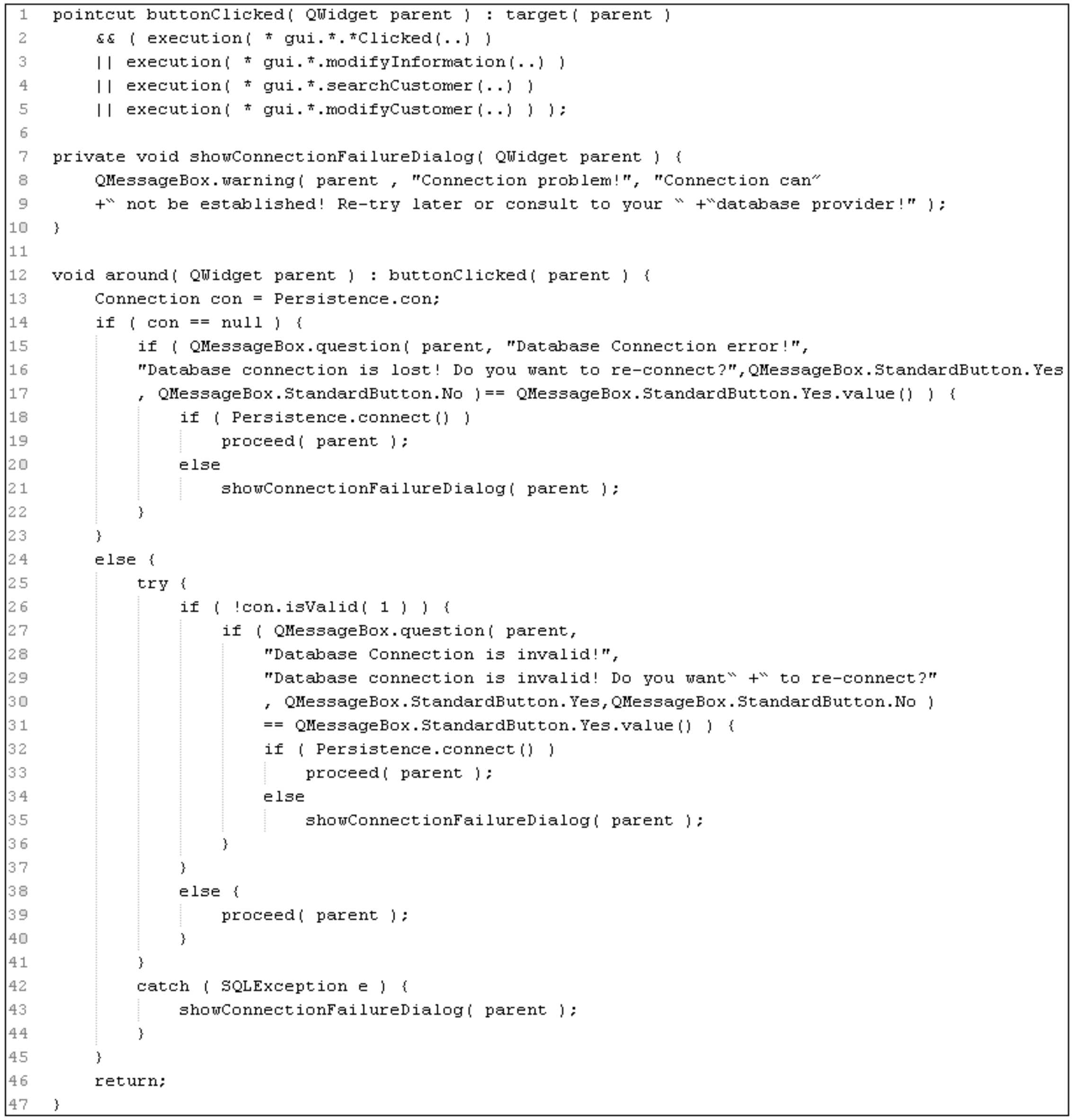


```
pointcut syncLocalManager(BranchManagerWidget caller)
: execution( * BranchManagerWidget.localManagerButtonClicked() )
&& this(caller);

void around (BranchManagerWidget caller) : syncLocalManager (caller){
    BranchManagerWidget branchManager=(BranchManagerWidget) caller;
    int baseIndex = ( branchManager.currentPage - 1 ) * branchManager.MAX_ITEM_COUNT;
    localManagerIndex=branchManager.managerWidgetUi.tableWidget.currentRow();
    Branch branch = branchManager.getBranches().get( baseIndex
    + branchManager.managerWidgetUi.tableWidget.currentRow() );
    LocalManager localManager=branch.getLocalManager();

    String updateQuery="UPDATE salesPerson SET locked=\"1\" WHERE nationalId=\""
    +localManager.getNationalId()+"\" AND 0=locked";
    if ( updateExecute(updateQuery)>0 ) {
        proceed(caller);
    }
    else {
        QMessageBox.critical( null, "Local manager Modify Error",
        "Local manager is being modifiying by another user please try again later.");
    }
}

after (ManagerWidget caller) returning : syncLocalManager(caller){
    String updateQuery=null;

    BranchManagerWidget branchManager=(BranchManagerWidget) caller;
    int baseIndex = ( branchManager.currentPage - 1 ) * branchManager.MAX_ITEM_COUNT;

    Branch branch = branchManager.getBranches().get( baseIndex + localManagerIndex );
    LocalManager localManager=branch.getLocalManager();

    updateQuery="UPDATE salesPerson SET locked=\"0\" WHERE nationalId=\""
    +localManager.getNationalId()+"\" AND 1=locked";

    updateExecute(updateQuery);
}
```

Failure Management

- Need control after button clicks
- Need access to database connection object
- Produces scattering codes
- Lowers the modularity

Failure Management Aspect Code

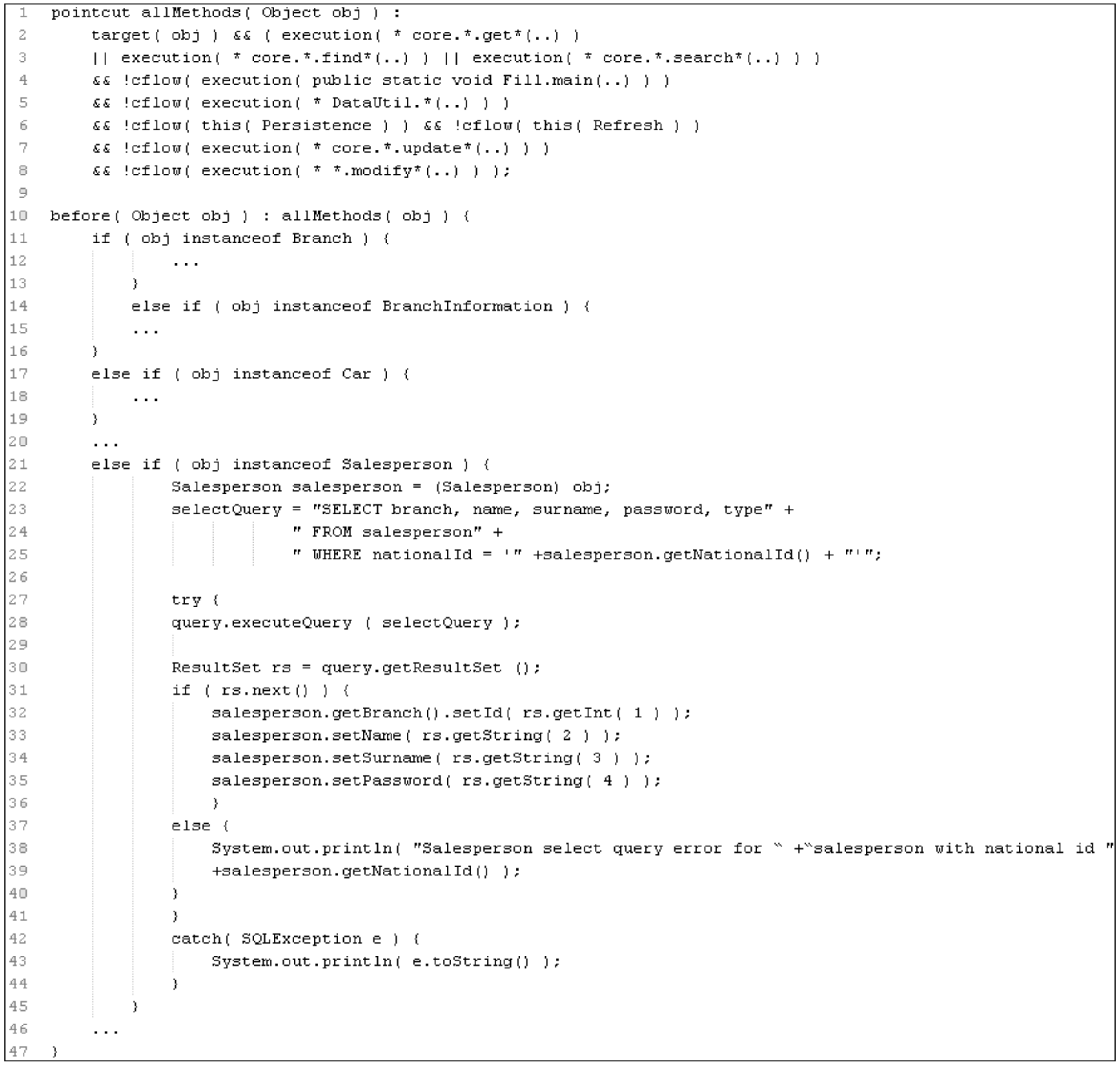


```
pointcut buttonClicked( QWidget parent ) : target( parent )
    && ( execution( * gui.*.*Clicked(..) )
    || execution( * gui.*.modifyInformation(..) )
    || execution( * gui.*.searchCustomer(..) )
    || execution( * gui.*.modifyCustomer(..) ) );

private void showConnectionFailureDialog( QWidget parent ) {
    QMessageBox.warning( parent , "Connection problem!", "Connection can"
    +" not be established! Re-try later or consult to your " +"database provider!" );
}

void around( QWidget parent ) : buttonClicked( parent ) {
    Connection con = Persistence.con;
    if ( con == null ) {
        if ( QMessageBox.question( parent, "Database Connection error!",
        "Database connection is lost! Do you want to re-connect?",QMessageBox.StandardButton.Yes
        , QMessageBox.StandardButton.No )== QMessageBox.StandardButton.Yes.value() ) {
            if ( Persistence.connect() )
                proceed( parent );
            else
                showConnectionFailureDialog( parent );
        }
    }
    else {
        try {
            if ( !con.isValid( 1 ) ) {
                if ( QMessageBox.question( parent,
                    "Database Connection is invalid!",
                    "Database connection is invalid! Do you want" +" to re-connect?"
                    , QMessageBox.StandardButton.Yes,QMessageBox.StandardButton.No )
                    == QMessageBox.StandardButton.Yes.value() ) {
                    if ( Persistence.connect() )
                        proceed( parent );
                    else
                        showConnectionFailureDialog( parent );
                }
            }
            else {
                proceed( parent );
            }
        }
        catch ( SQLException e ) {
            showConnectionFailureDialog( parent );
        }
    }
    return;
}
```

Refresh from Database

- Update objects regularly
- Code change in each get method
- Requires scattering code
- Decreases the modularity

Refresh from Database Aspect Code



```
pointcut allMethods( Object obj ) :
    target( obj ) && ( execution( * core.*.get*(..) )
    || execution( * core.*.find*(..) ) || execution( * core.*.search*(..) ) )
    && !cflow( execution( public static void Fill.main(..) ) )
    && !cflow( execution( * DataUtil.*(..) ) )
    && !cflow( this( Persistence ) ) && !cflow( this( Refresh ) )
    && !cflow( execution( * core.*.update*(..) ) )
    && !cflow( execution( * *.modify*(..) ) );

before( Object obj ) : allMethods( obj ) {
    if ( obj instanceof Branch ) {
            ...
        }
        else if ( obj instanceof BranchInformation ) {
        ...
    }
    else if ( obj instanceof Car ) {
        ...
    }
    ...
    else if ( obj instanceof Salesperson ) {
            Salesperson salesperson = (Salesperson) obj;
            selectQuery = "SELECT branch, name, surname, password, type" +
                        " FROM salesperson" +
                        " WHERE nationalId = '" +salesperson.getNationalId() + "'";

            try {
            query.executeQuery ( selectQuery );

            ResultSet rs = query.getResultSet ();
            if ( rs.next() ) {
                salesperson.getBranch().setId( rs.getInt( 1 ) );
                salesperson.setName( rs.getString( 2 ) );
                salesperson.setSurname( rs.getString( 3 ) );
                salesperson.setPassword( rs.getString( 4 ) );
                }
            else {
                System.out.println( "Salesperson select query error for " +"salesperson with national id "
                +salesperson.getNationalId() );
            }
            }
            catch( SQLException e ) {
                System.out.println( e.toString() );
            }
        }
    ...
}
```

Program Demonstration

JBoss AOP vs. AspectJ

- Language mechanisms
- Development Environments
- Ready to use aspects

- Hot Deployment of the Aspects

Dynamic AOP

- What is it ?
  - Dynamically inserting and deleting aspects from system.
- How could we used that in our system ?
  - For debugging.
  - For disabling an aspect on the runtime.

- AspectJ is planning to support it.

Discussion

- Hard to analysis the code without tools
- Some of the aspects lowers performance
  - Refresh of the objects
- Some of the aspects are not reusable
  - Update operation of persistence

Conclusion

- Which aspects are focused
  - Persistence
  - Synchronization
  - Failure Management
  - Refresh the objects from database

Conclusion (cont.)

- Existing Car Rental System
  - Works on files
  - Only a single user can work on the program
- New Car Rental System
  - Working on MySQL database system.
  - Allows multi-user usage.

Conclusion (cont.)

- Why was AOSD necessary ?
  - No need to change the legacy code
  - New concerns added and the system is still modular and maintainable

Conclusion (cont.)

- Future Work
  - Transactions
  - SQL query builder
  - More reusable aspects