Rational Software

Payroll System Use-Case Design Solution

Version 2004

Revision History

Date	Issue	Description	Author
09/01/2000	V2000	Generation for beta	Shawn Siemers
10/02/2000	V2000	Final release	Shawn Siemers
01/14/2003	V2003	Final Release	Alex Kutsick
05/20/2004	2004	Generation for beta	Alex Kutsick

Table of Contents

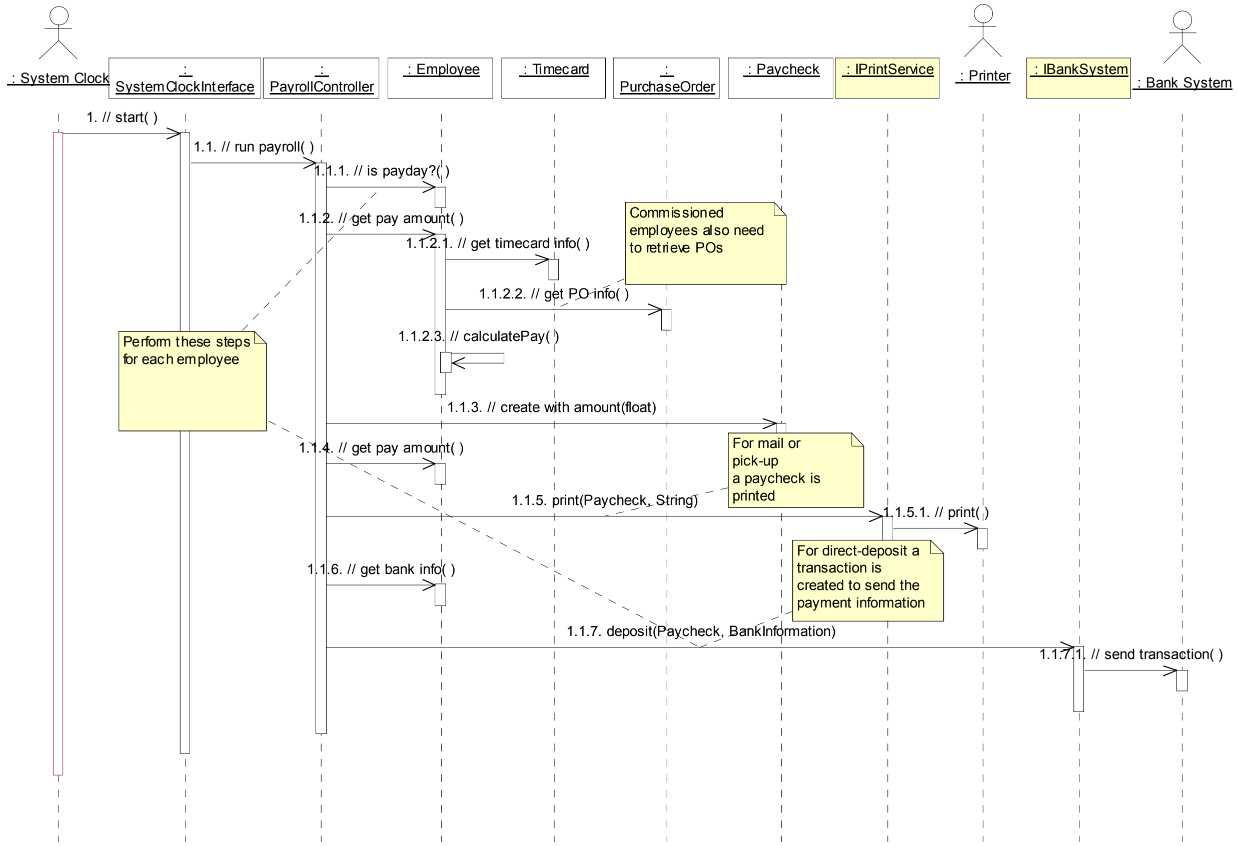
Payroll System Use-Case Design Solution

1. Exercise: Use-Case Design, Part 1

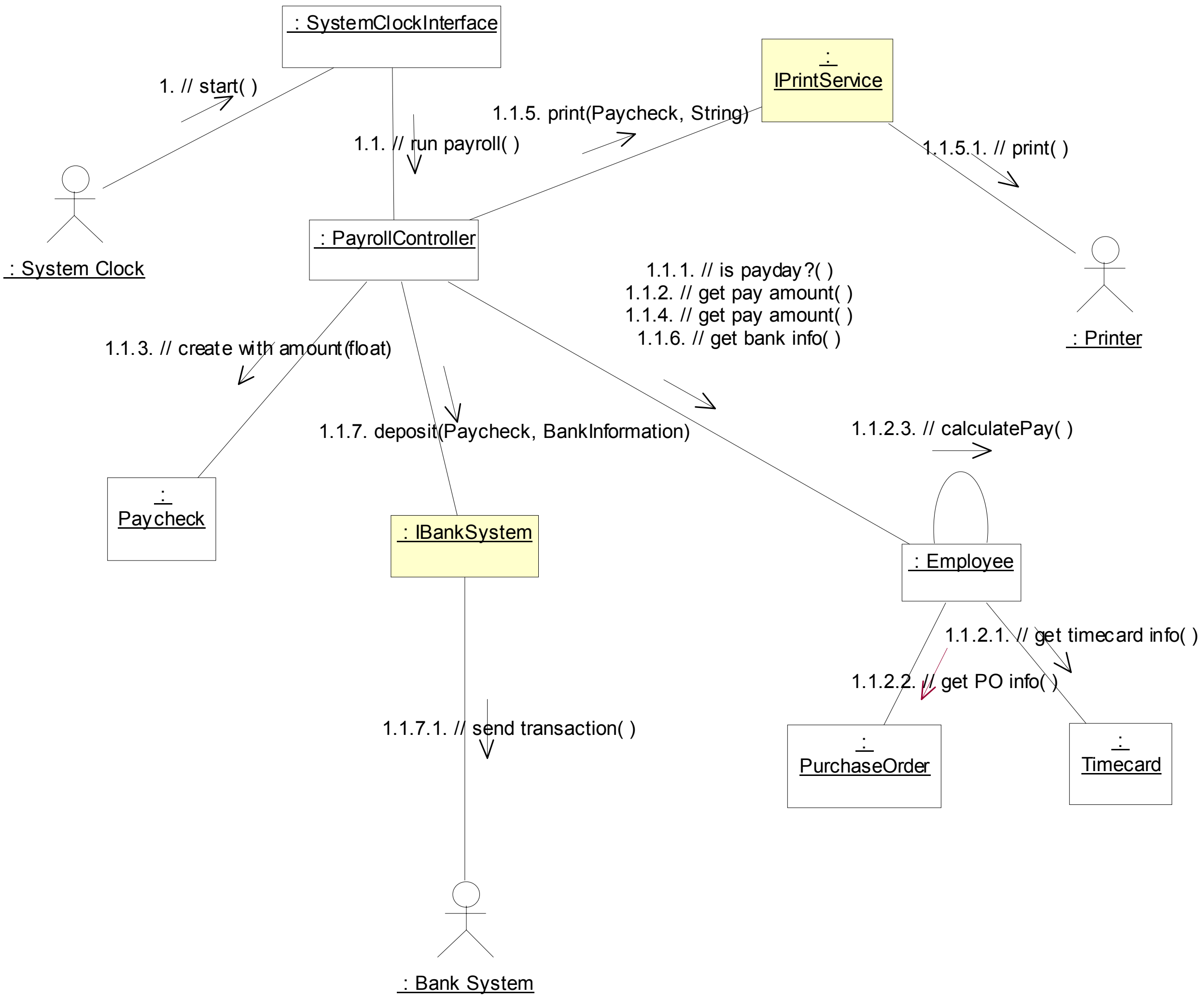
1.1 Use-Case Realization - Run Payroll

1.1.1 *Run Payroll (with ss interface)*

Run Payroll - Basic Flow (with ss interface)

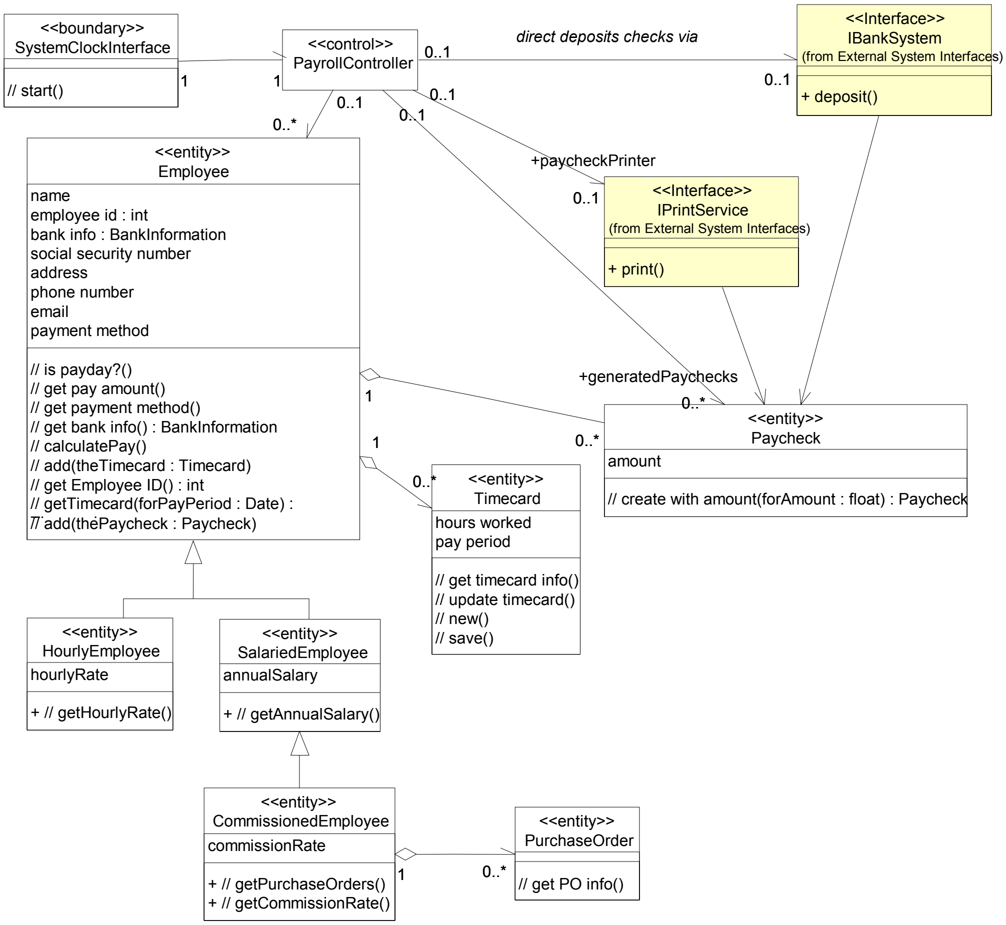

: System Clock
: SystemClockInterface
: PayrollController
: Employee
: Timecard
: PurchaseOrder
: Paycheck
: IPrintService
: Printer
: IBankSystem
: Bank System
1. // start( )
1.1. // run payroll( )
1.1.1. // is payday?( )
1.1.2. // get pay amount( )
1.1.2.1. // get timecard info( )
1.1.2.2. // get PO info( )
1.1.2.3. // calculatePay( )
1.1.3. // create with amount(float)
1.1.4. // get pay amount( )
1.1.5. print(Paycheck, String)
1.1.5.1. // print( )
1.1.6. // get bank info( )
1.1.7. deposit(Paycheck, BankInformation)
1.1.7.1. // send transaction( )
Commissioned employees also need to retrieve POs
Perform these steps for each employee
For mail or pick-up a paycheck is printed
For direct-deposit a transaction is created to send the payment information

Run Payroll - Basic Flow (with ss interface)



: SystemClockInterface
: IPrintService
1. // start( )
1.1.5. print(Paycheck, String)
1.1. // run payroll( )
1.1.5.1. // print( )
: System Clock
: PayrollController
: Printer
1.1.1. // is payday?( )
1.1.2. // get pay amount( )
1.1.4. // get pay amount( )
1.1.6. // get bank info( )
1.1.3. // create with amount(float)
1.1.7. deposit(Paycheck, BankInformation)
1.1.2.3. // calculatePay( )
: Paycheck
: IBankSystem
: Employee
1.1.2.1. // get timecard info( )
1.1.2.2. // get PO info( )
1.1.7.1. // send transaction( )
: PurchaseOrder
: Timecard
: Bank System

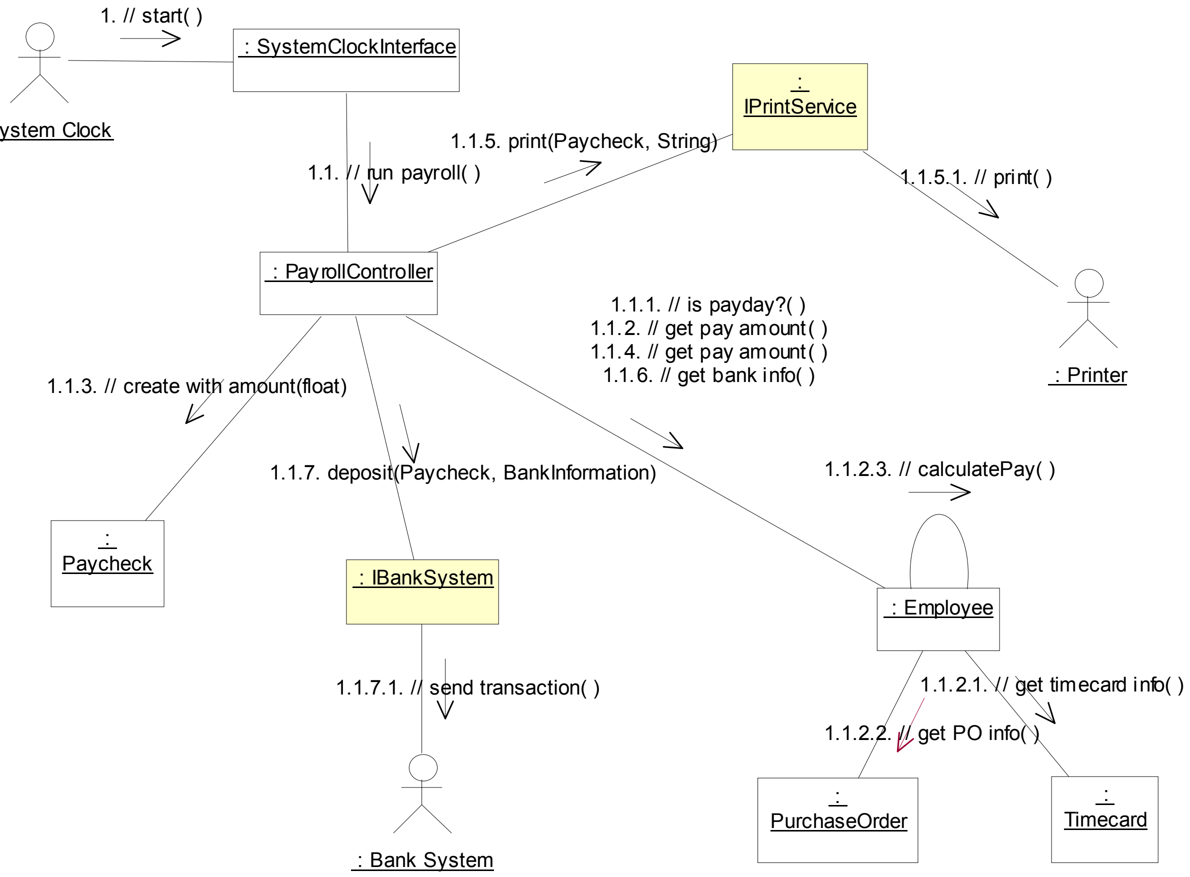
Run Payroll - VOPC (with ss interface)



1.1.2 Run Payroll (with Security)

Run Payroll - Basic Flow (with security)

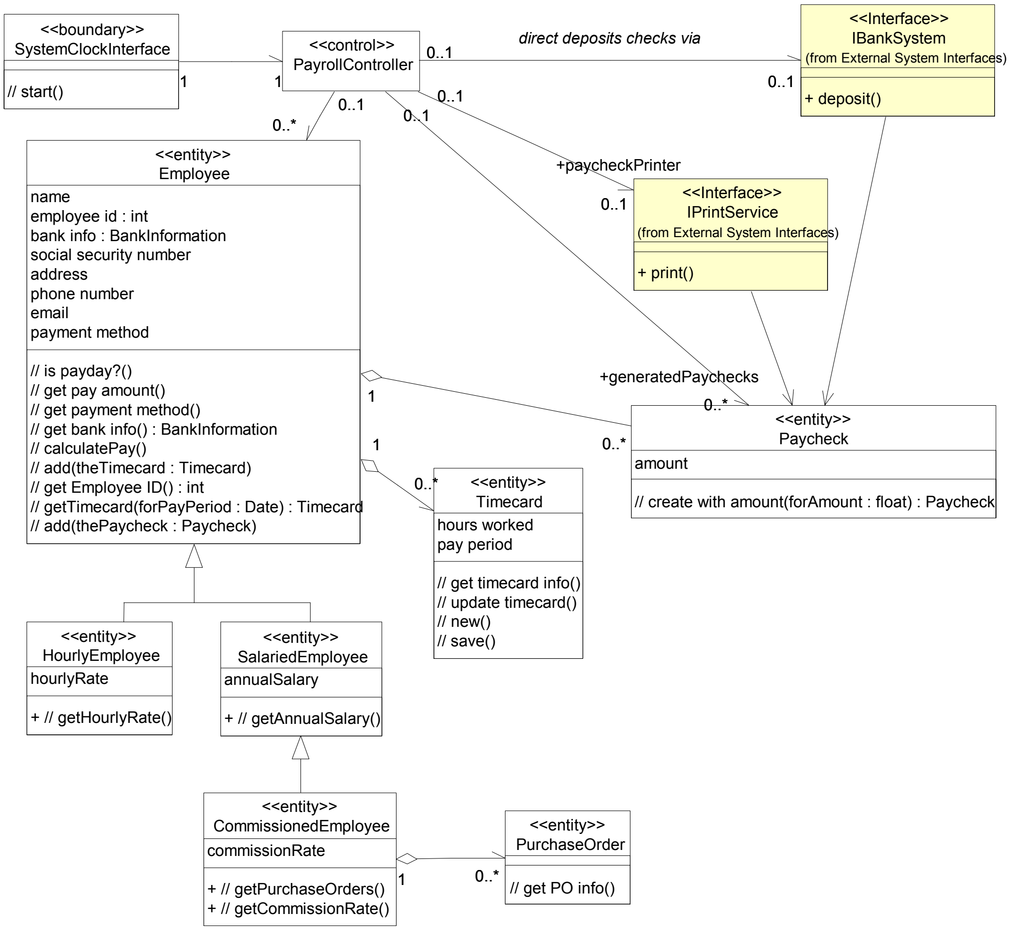
This is the same collaboration diagram as Run Payroll (with ss interface). There are no additional processing steps for Security for Run Payroll, as the PayrollController is meant to be "all-knowing" and "all-seeing" and thus, has open access to all secure data for Employees.



1. // start( )
: SystemClockInterface
: System Clock
: IPrintService
1.1.5. print(Paycheck, String)
1.1. // run payroll( )
1.1.5.1. // print( )
: PayrollController
1.1.1. // is payday?( )
1.1.2. // get pay amount( )
1.1.4. // get pay amount( )
1.1.6. // get bank info( )
: Printer
1.1.3. // create with amount(float)
1.1.7. deposit(Paycheck, BankInformation)
1.1.2.3. // calculatePay( )
: Paycheck
: IBankSystem
: Employee
1.1.7.1. // send transaction( )
1.1.2.1. // get timecard info( )
1.1.2.2. // get PO info( )
: Bank System
: PurchaseOrder
: Timecard

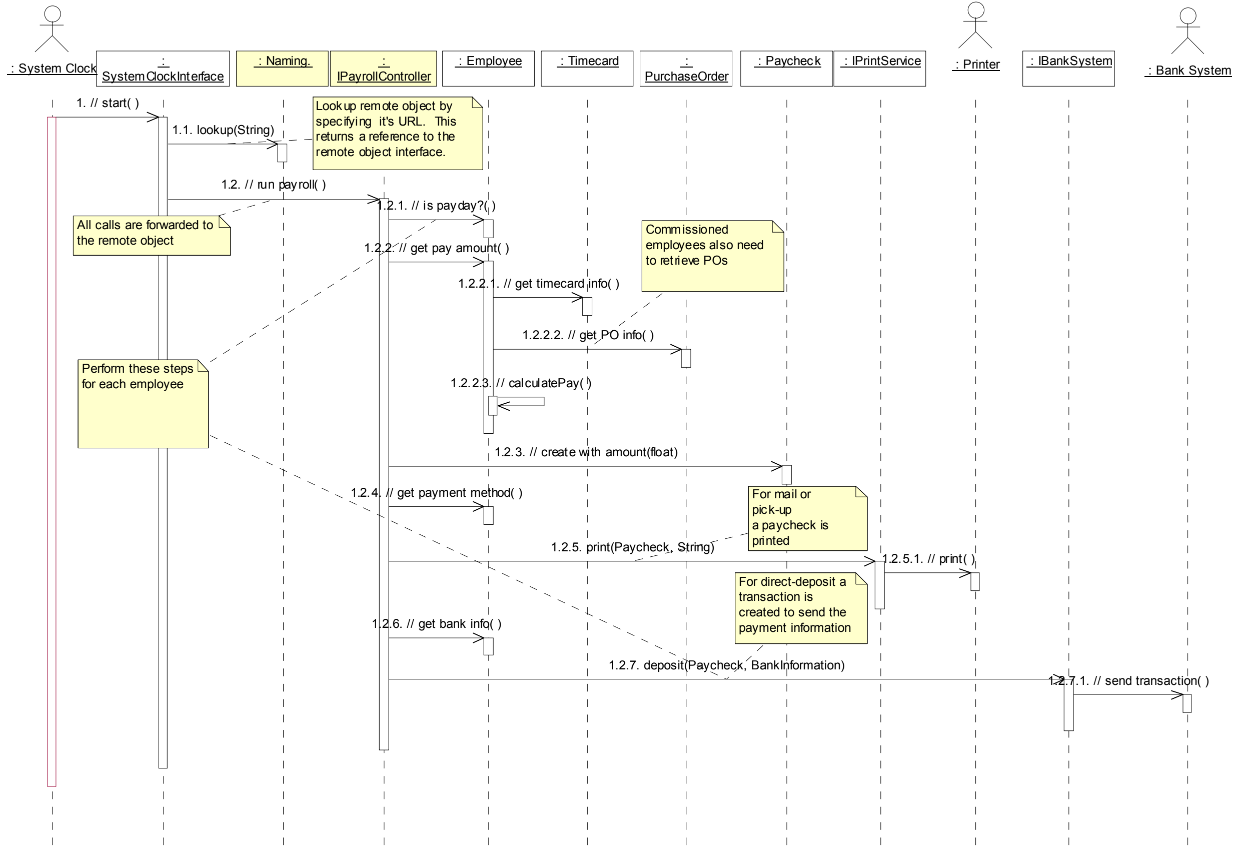
Run Payroll - VOPC (with Security)

This is the same VOPC as Run Payroll (with ss interface).
There are no additional processing steps for Security for Run Payroll, as the PayrollController is meant to be "all-knowing" and "all-seeing" and thus, has open access to all secure data for Employees.



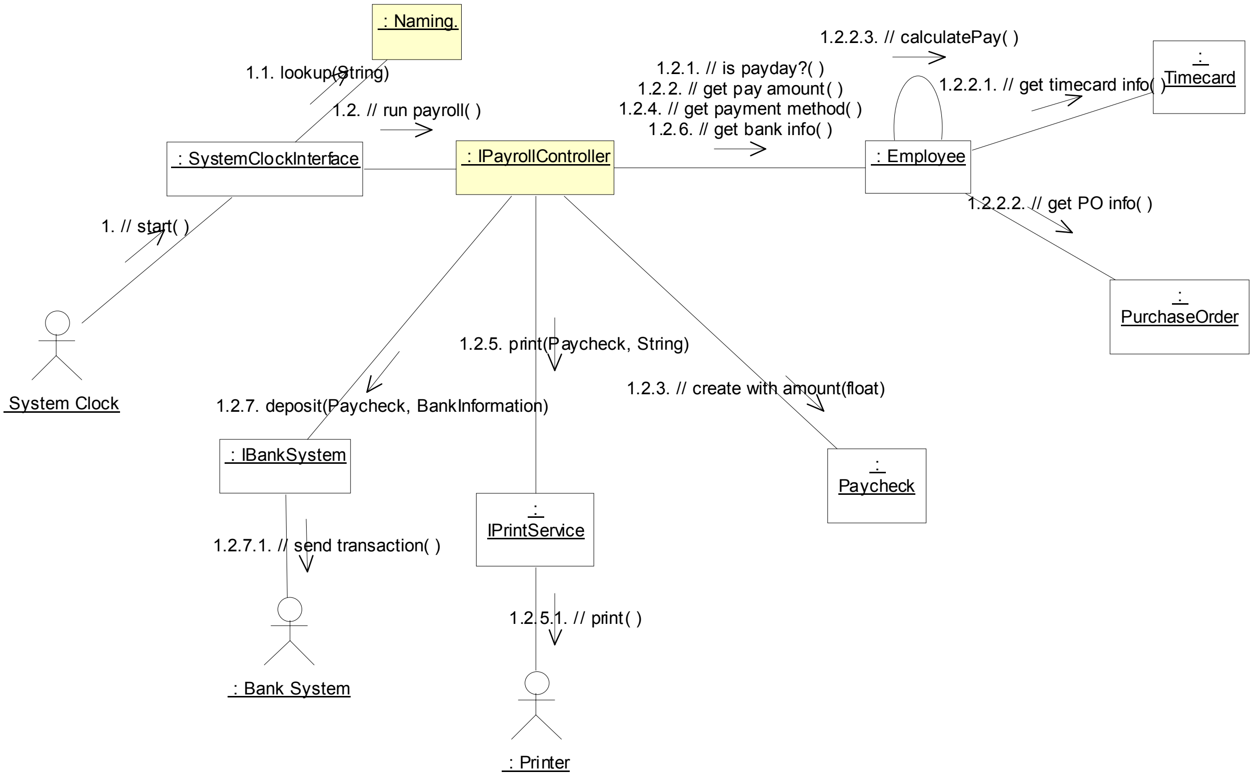
1.1.3 Run Payroll (with Distribution)

Run Payroll - Basic Flow (with Distribution)



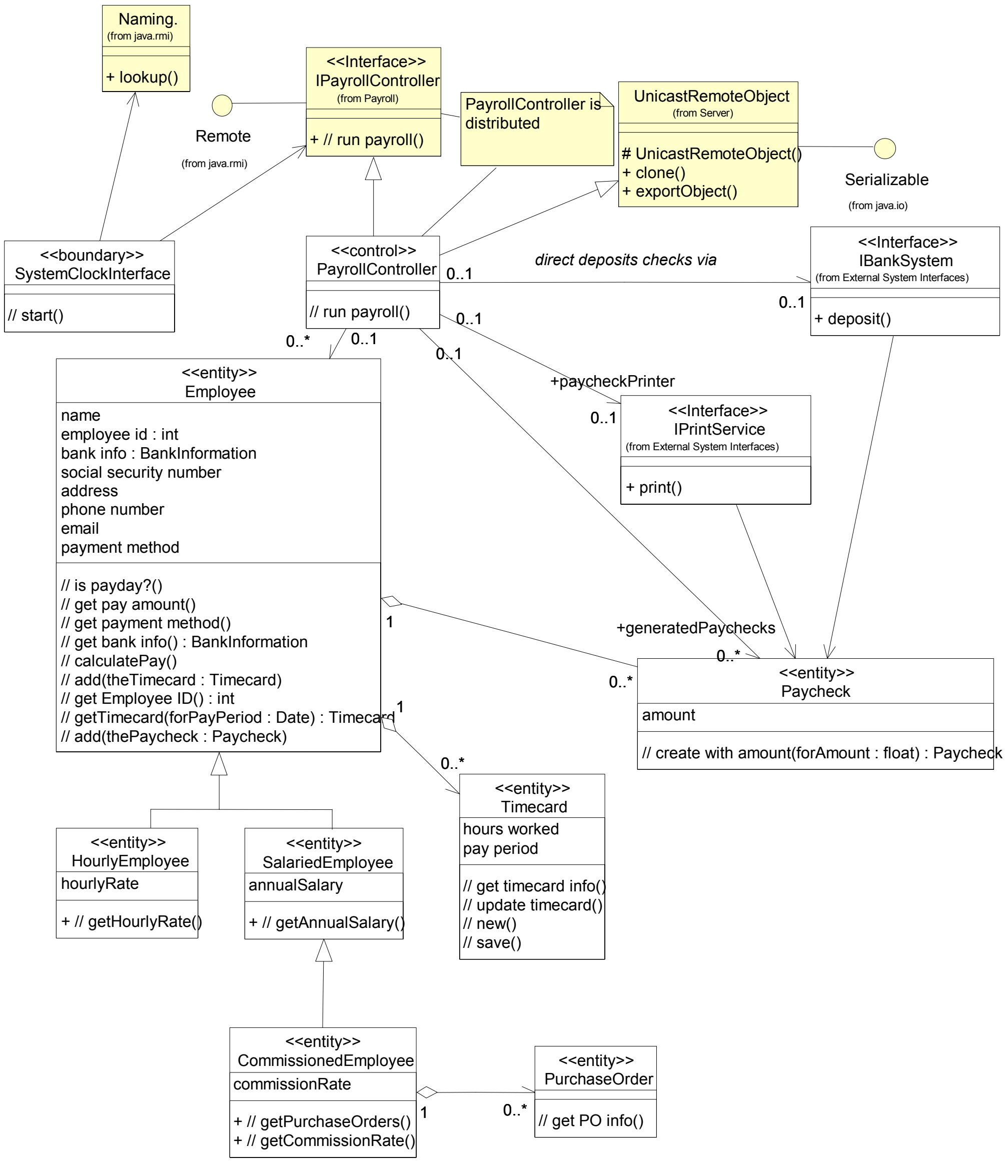
: System Clock
: SystemClockInterface
: Naming.
: IPayrollController
: Employee
: Timecard
: PurchaseOrder
: Paycheck
: IPrintService
: Printer
: IBankSystem
: Bank System
1. // start( )
1.1. lookup(String)
Lookup remote object by specifying it's URL. This returns a reference to the remote object interface.
1.2. // run payroll( )
1.2.1. // is payday?( )
All calls are forwarded to the remote object
1.2.2. // get pay amount( )
Commissioned employees also need to retrieve POs
1.2.2.1. // get timecard info( )
1.2.2.2. // get PO info( )
Perform these steps for each employee
1.2.2.3. // calculatePay( )
1.2.3. // create with amount(float)
1.2.4. // get payment method( )
For mail or pick-up a paycheck is printed
1.2.5. print(Paycheck, String)
1.2.5.1. // print( )
For direct-deposit a transaction is created to send the payment information
1.2.6. // get bank info( )
1.2.7. deposit(Paycheck, BankInformation)
1.2.7.1. // send transaction( )

Run Payroll - Basic Flow (with Distribution)



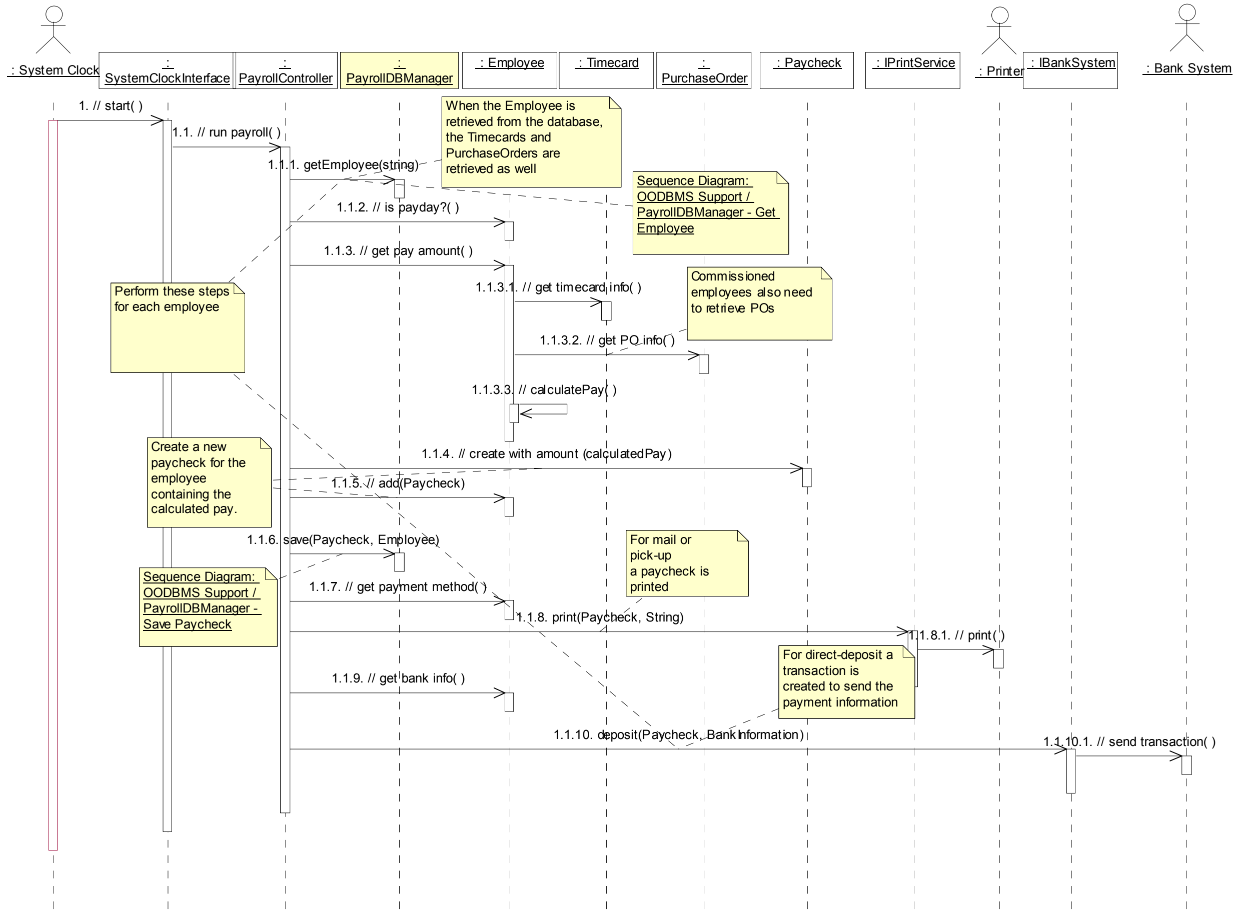
: Naming
1.1. lookup(String)
1.2. // run payroll( )
: SystemClockInterface
: IPayrollController
1.2.1. // is payday?( )
1.2.2. // get pay amount( )
1.2.4. // get payment method( )
1.2.6. // get bank info( )
1.2.2.3. // calculatePay( )
: Employee
1.2.2.1. // get timecard info( )
: Timecard
1.2.2.2. // get PO info( )
: PurchaseOrder
1. // start( )
System Clock
1.2.5. print(Paycheck, String)
1.2.3. // create with amount(float)
1.2.7. deposit(Paycheck, BankInformation)
: IBankSystem
: IPrintService
: Paycheck
1.2.7.1. // send transaction( )
: Bank System
1.2.5.1. // print( )
: Printer

Run Payroll - VOPC (with Distribution)



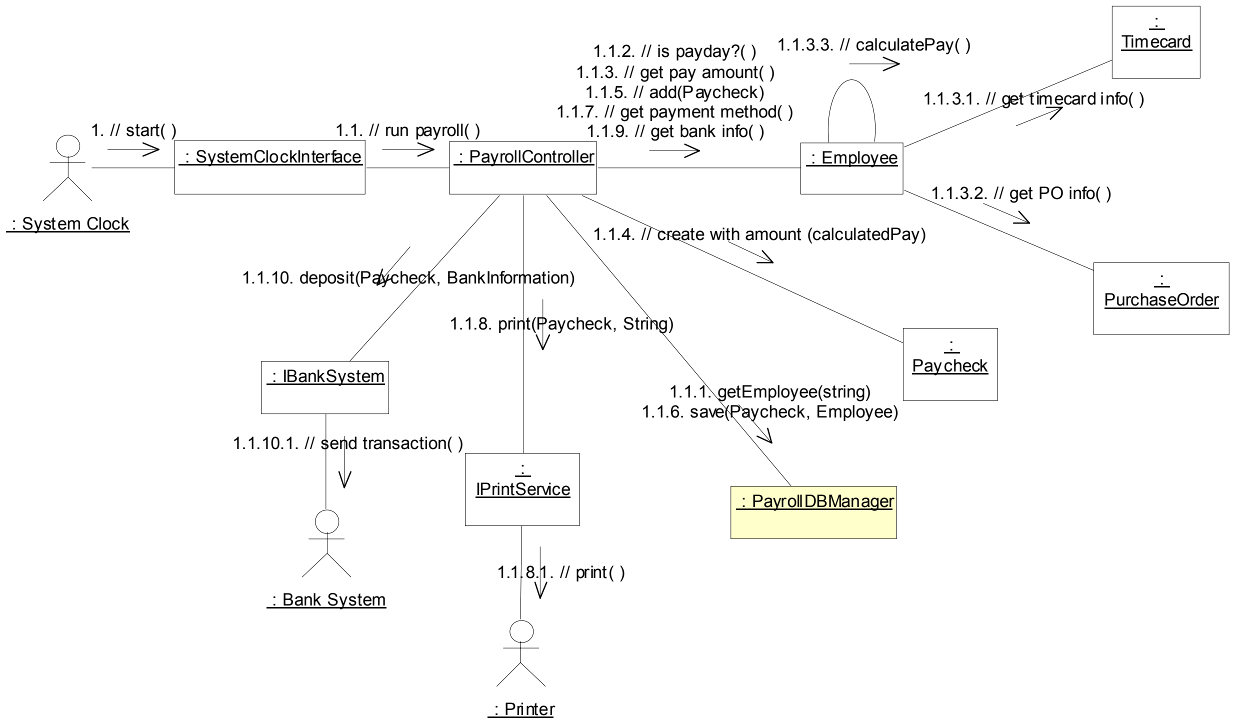
1.1.4 Run Payroll (with OODBMS Persistency)

Run Payroll - Basic Flow (with OODBMS Persistency)

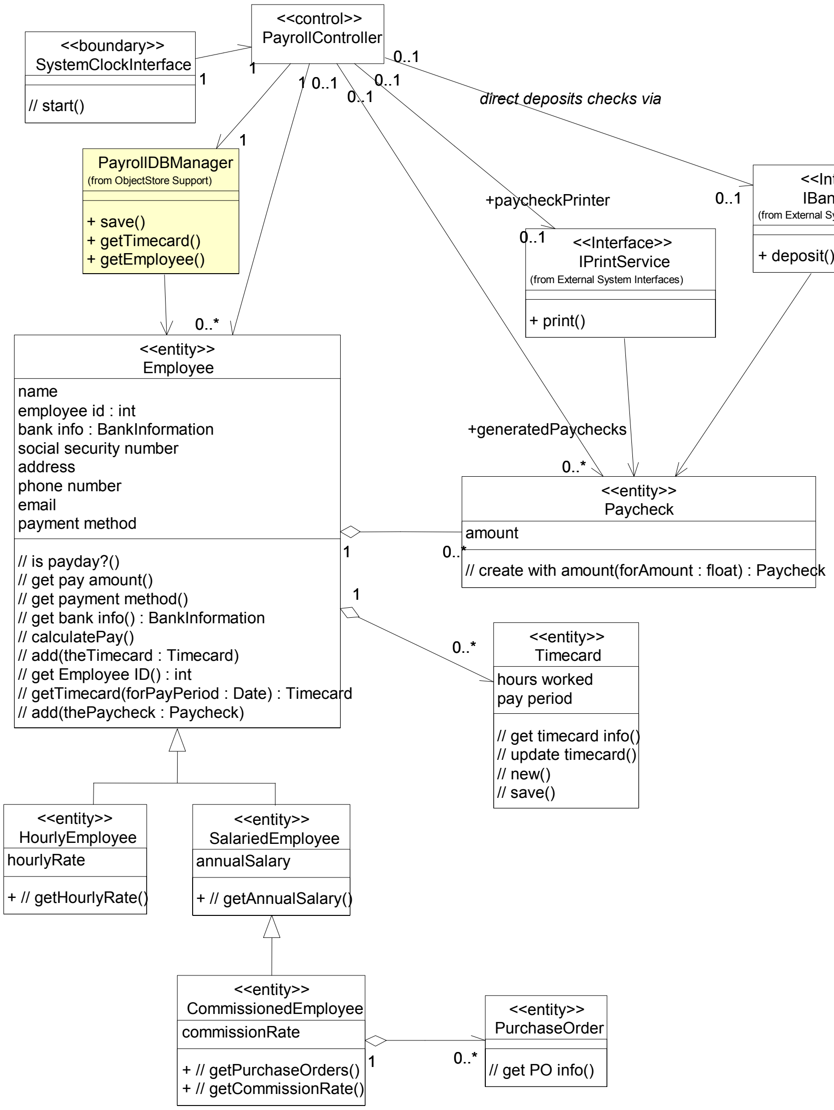


: System Clock
: SystemClockInterface
: PayrollController
: PayrollDBManager
: Employee
: Timecard
: PurchaseOrder
: Paycheck
: IPrintService
: Printer
: IBankSystem
: Bank System
1. // start( )
1.1. // run payroll( )
1.1.1. getEmployee(string)
When the Employee is retrieved from the database, the Timecards and PurchaseOrders are retrieved as well
Sequence Diagram: OODBMS Support / PayrollDBManager - Get Employee
1.1.2. // is payday?( )
1.1.3. // get pay amount( )
Perform these steps for each employee
1.1.3.1. // get timecard info( )
Commissioned employees also need to retrieve POs
1.1.3.2. // get PO info( )
1.1.3.3. // calculatePay( )
1.1.4. // create with amount (calculatedPay)
Create a new paycheck for the employee containing the calculated pay.
1.1.5. // add(Paycheck)
1.1.6. save(Paycheck, Employee)
Sequence Diagram: OODBMS Support / PayrollDBManager - Save Paycheck
1.1.7. // get payment method( )
For mail or pick-up a paycheck is printed
1.1.8. print(Paycheck, String)
1.1.8.1. // print( )
For direct-deposit a transaction is created to send the payment information
1.1.9. // get bank info( )
1.1.10. deposit(Paycheck, BankInformation)
1.1.10.1. // send transaction( )

Run Payroll - Basic Flow (with OODBMS Persistency)

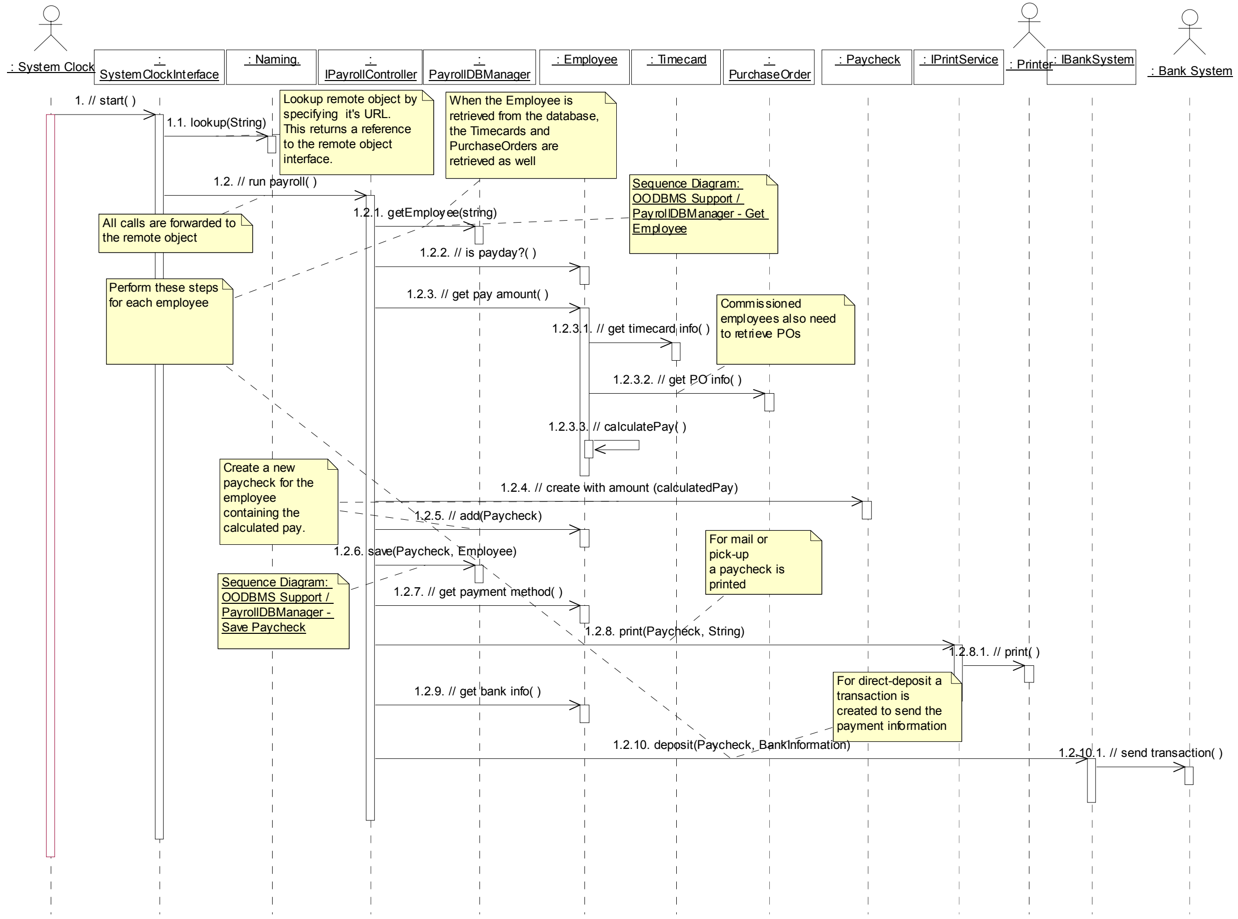

: System Clock
1. // start( )
: SystemClockInterface
1.1. // run payroll( )
: PayrollController
1.1.2. // is payday?( )
1.1.3. // get pay amount( )
1.1.5. // add(Paycheck)
1.1.7. // get payment method( )
1.1.9. // get bank info( )
: Employee
1.1.3.3. // calculatePay( )
: Timecard
1.1.3.1. // get timecard info( )
1.1.3.2. // get PO info( )
: PurchaseOrder
1.1.4. // create with amount (calculatedPay)
: Paycheck
1.1.10. deposit(Paycheck, BankInformation)
1.1.8. print(Paycheck, String)
: IBankSystem
1.1.1. getEmployee(string)
1.1.6. save(Paycheck, Employee)
1.1.10.1. // send transaction( )
: IPrintService
: PayrollDBManager
: Bank System
1.1.8.1. // print( )
: Printer

Run Payroll - VOPC (with OODBMS Persistency)

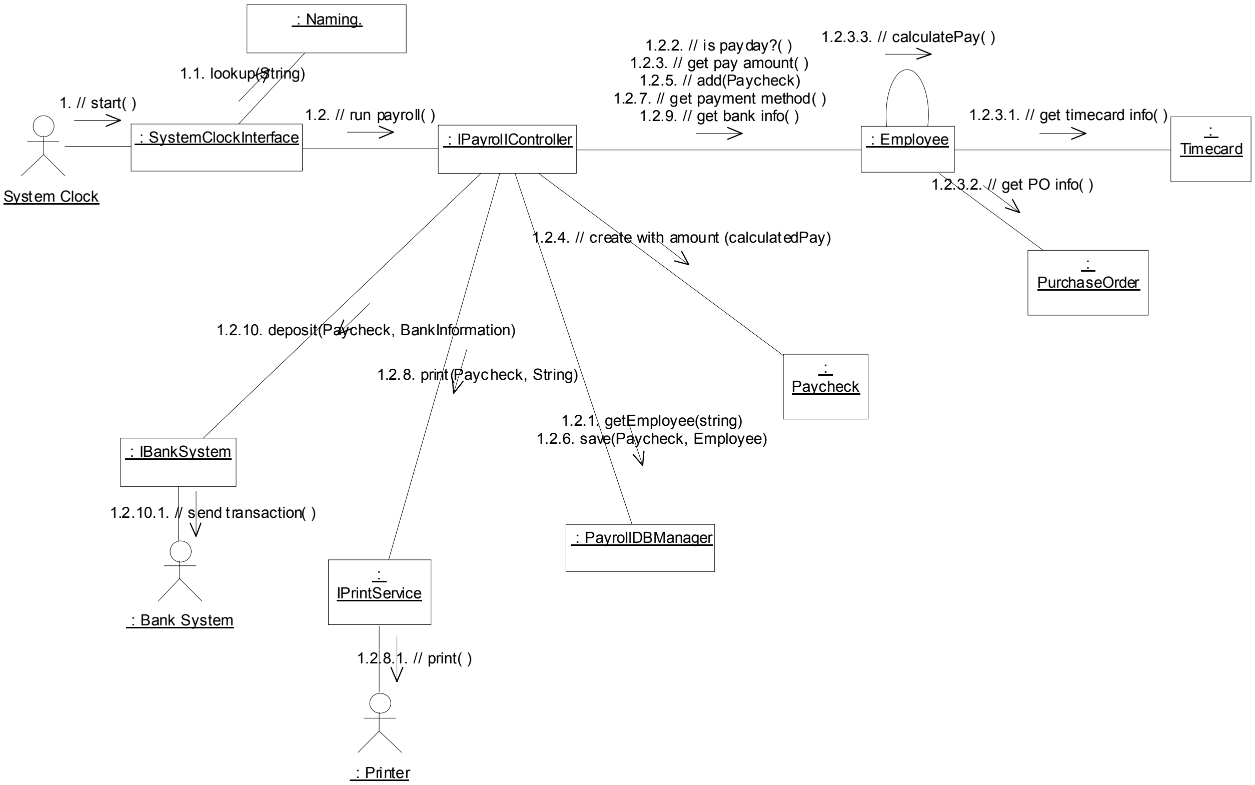


1.1.5 Run Payroll (with everything)

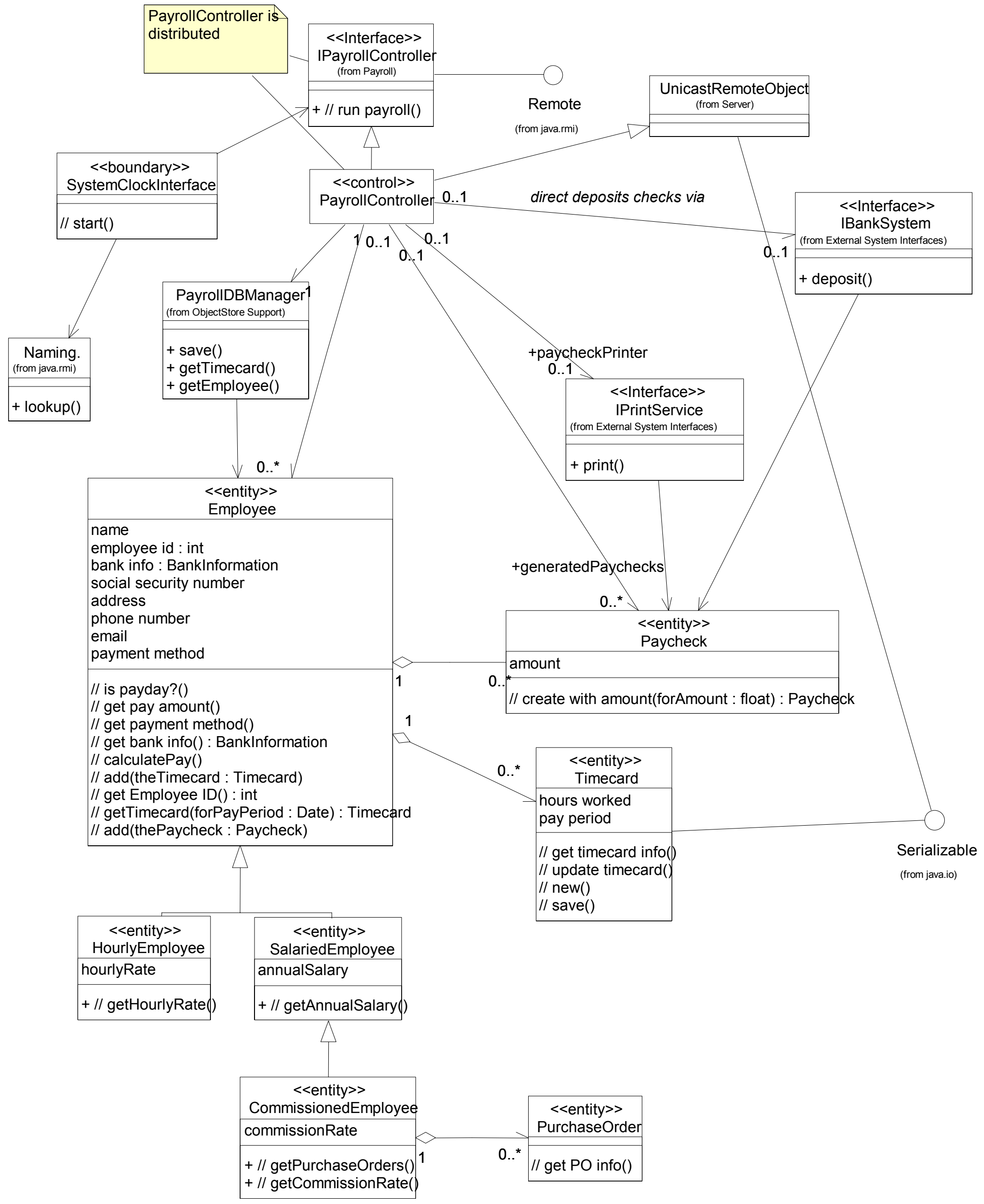
Run Payroll - Basic Flow (with everything)


: System Clock
: SystemClockInterface
: Naming.
: IPayrollController
: PayrollDBManager
: Employee
: Timecard
: PurchaseOrder
: Paycheck
: IPrintService
: Printer
: IBankSystem
: Bank System
1. // start( )
1.1. lookup(String)
Lookup remote object by specifying it's URL. This returns a reference to the remote object interface.
When the Employee is retrieved from the database, the Timecards and PurchaseOrders are retrieved as well
1.2. // run payroll( )
All calls are forwarded to the remote object
1.2.1. getEmployee(string)
Sequence Diagram: OODBMS Support / PayrollDBManager - Get Employee
1.2.2. // is payday?( )
Perform these steps for each employee
1.2.3. // get pay amount( )
Commissioned employees also need to retrieve POs
1.2.3.1. // get timecard info( )
1.2.3.2. // get PO info( )
1.2.3.3. // calculatePay( )
Create a new paycheck for the employee containing the calculated pay.
1.2.4. // create with amount (calculatedPay)
1.2.5. // add(Paycheck)
1.2.6. save(Paycheck, Employee)
For mail or pick-up a paycheck is printed
Sequence Diagram: OODBMS Support / PayrollDBManager - Save Paycheck
1.2.7. // get payment method( )
1.2.8. print(Paycheck, String)
1.2.8.1. // print( )
1.2.9. // get bank info( )
For direct-deposit a transaction is created to send the payment information
1.2.10. deposit(Paycheck, BankInformation)
1.2.10.1. // send transaction( )

Run Payroll - Basic Flow (with everything)


: Naming.
1.1. lookup(String)
1. // start( )
System Clock
: SystemClockInterface
1.2. // run payroll( )
: IPayrollController
1.2.2. // is payday?( )
1.2.3. // get pay amount( )
1.2.5. // add(Paycheck)
1.2.7. // get payment method( )
1.2.9. // get bank info( )
1.2.3.3. // calculatePay( )
: Employee
1.2.3.1. // get timecard info( )
: Timecard
1.2.3.2. // get PO info( )
: PurchaseOrder
1.2.4. // create with amount (calculatedPay)
: Paycheck
1.2.10. deposit(Paycheck, BankInformation)
1.2.8. print(Paycheck, String)
1.2.1. getEmployee(string)
1.2.6. save(Paycheck, Employee)
: IBankSystem
1.2.10.1. // send transaction( )
: Bank System
: IPrintService
: PayrollDBManager
1.2.8.1. // print( )
: Printer

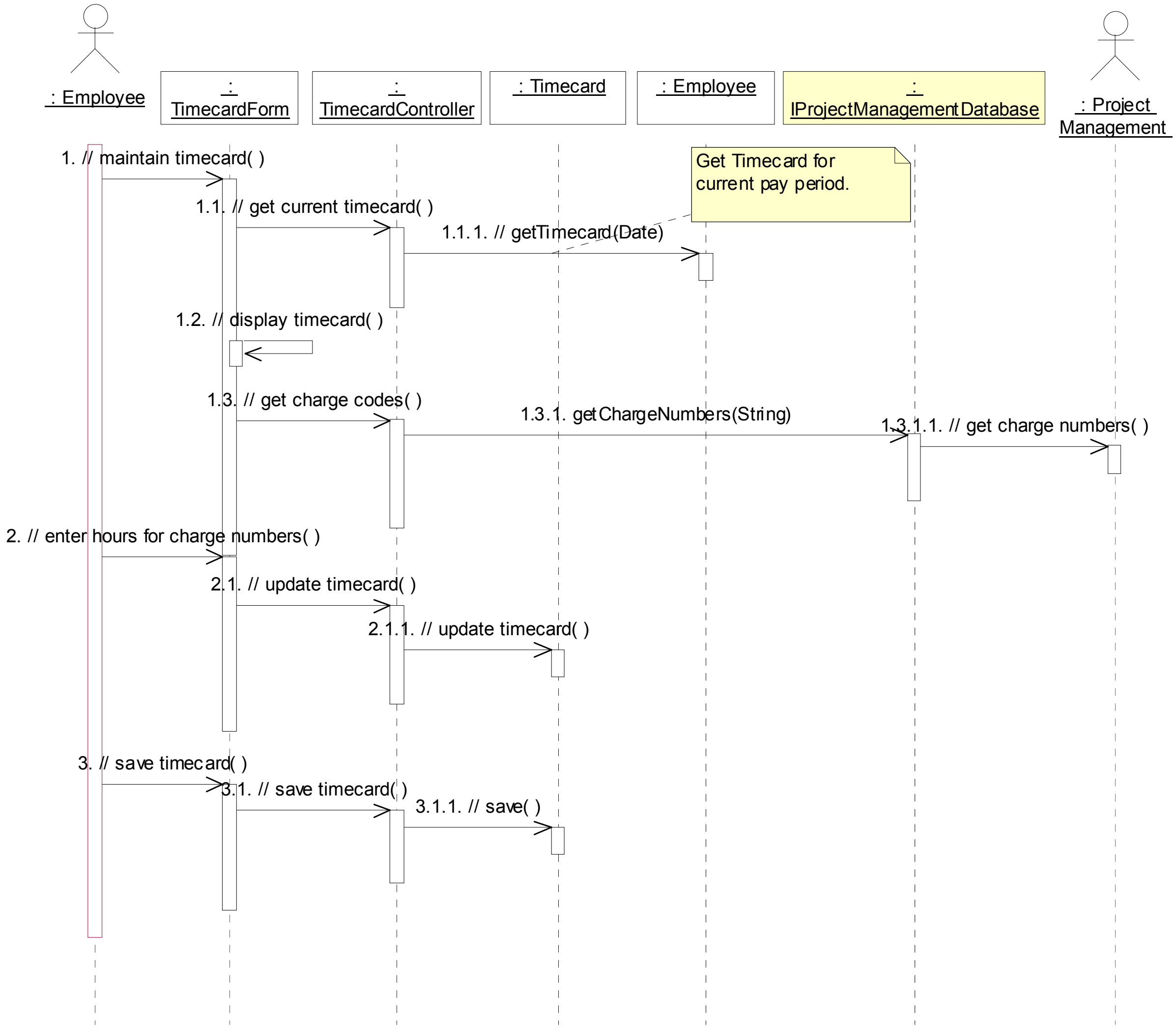
Run Payroll - VOPC (with everything)



1.2 Use-Case Realization - Maintain Timecard

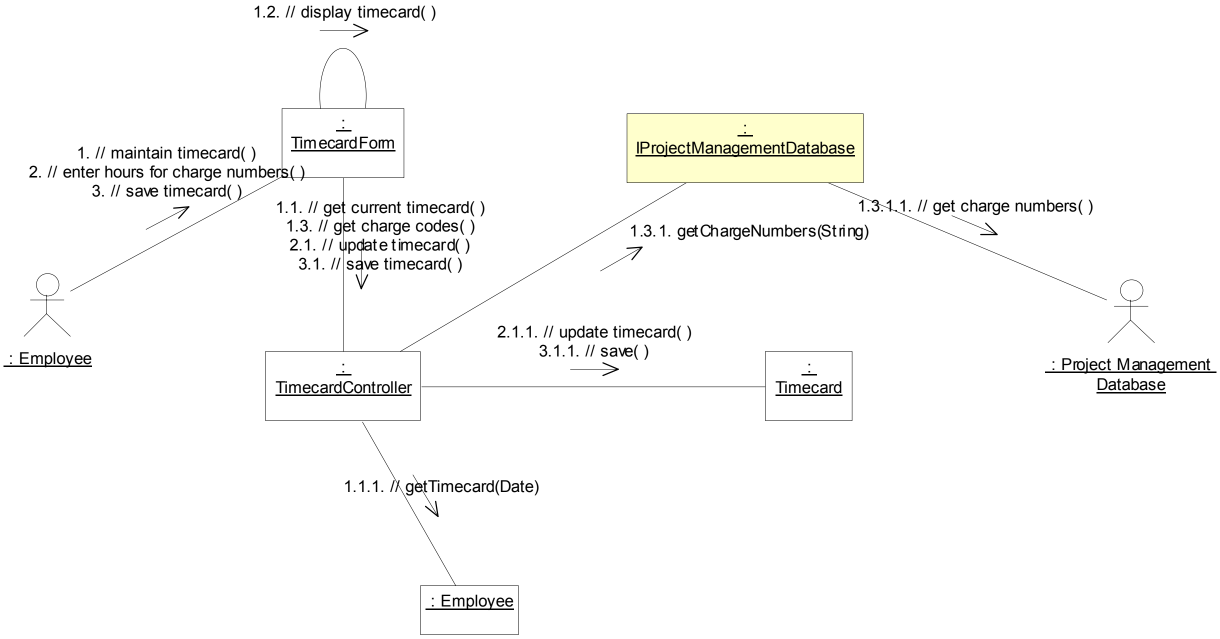
1.2.1 Maintain Timecard (with ss interface)

Maintain Timecard - Basic Flow (with ss interface)



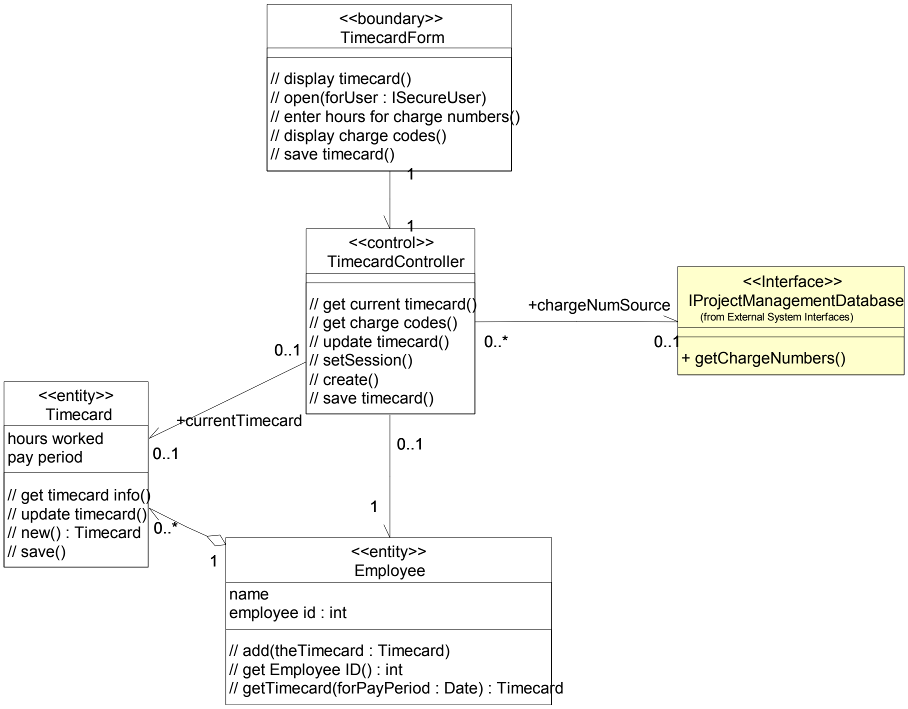
: Employee
: TimecardForm
: TimecardController
: Timecard
: Employee
: IProjectManagementDatabase
: Project Management
Get Timecard for current pay period.
1. // maintain timecard( )
1.1. // get current timecard( )
1.1.1. // getTimecard(Date)
1.2. // display timecard( )
1.3. // get charge codes( )
1.3.1. getChargeNumbers(String)
1.3.1.1. // get charge numbers( )
2. // enter hours for charge numbers( )
2.1. // update timecard( )
2.1.1. // update timecard( )
3. // save timecard( )
3.1. // save timecard( )
3.1.1. // save( )

Maintain Timecard - Basic Flow (with ss interface)



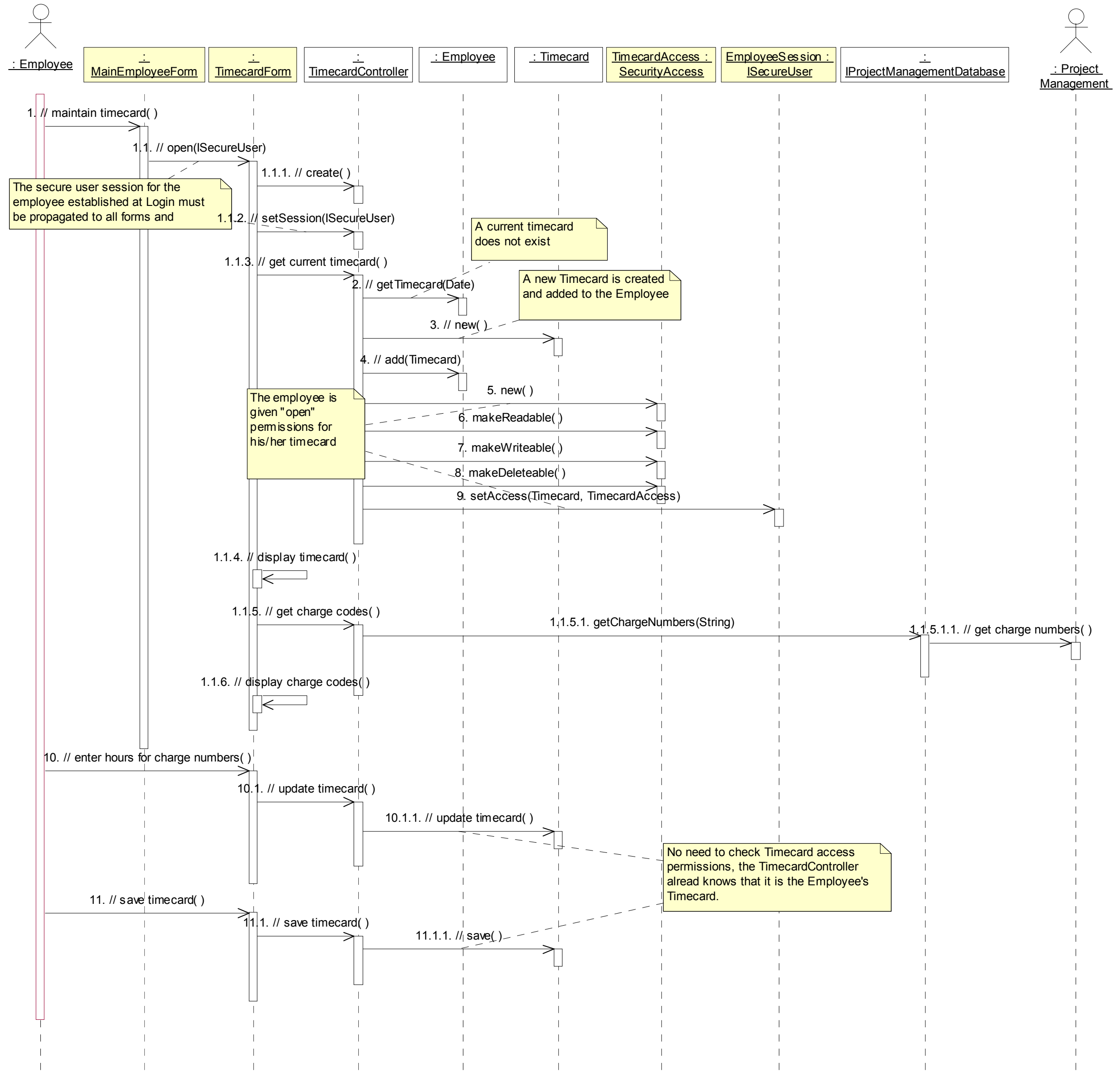
1.2. // display timecard( )
: TimecardForm
: IProjectManagementDatabase
1. // maintain timecard( )
2. // enter hours for charge numbers( )
3. // save timecard( )
1.1. // get current timecard( )
1.3. // get charge codes( )
2.1. // update timecard( )
3.1. // save timecard( )
1.3.1.1. // get charge numbers( )
1.3.1. getChargeNumbers(String)
: Employee
2.1.1. // update timecard( )
3.1.1. // save( )
: TimecardController
: Timecard
: Project Management Database
1.1.1. // getTimecard(Date)
: Employee

Maintain Timecard - VOPC (with ss interface)



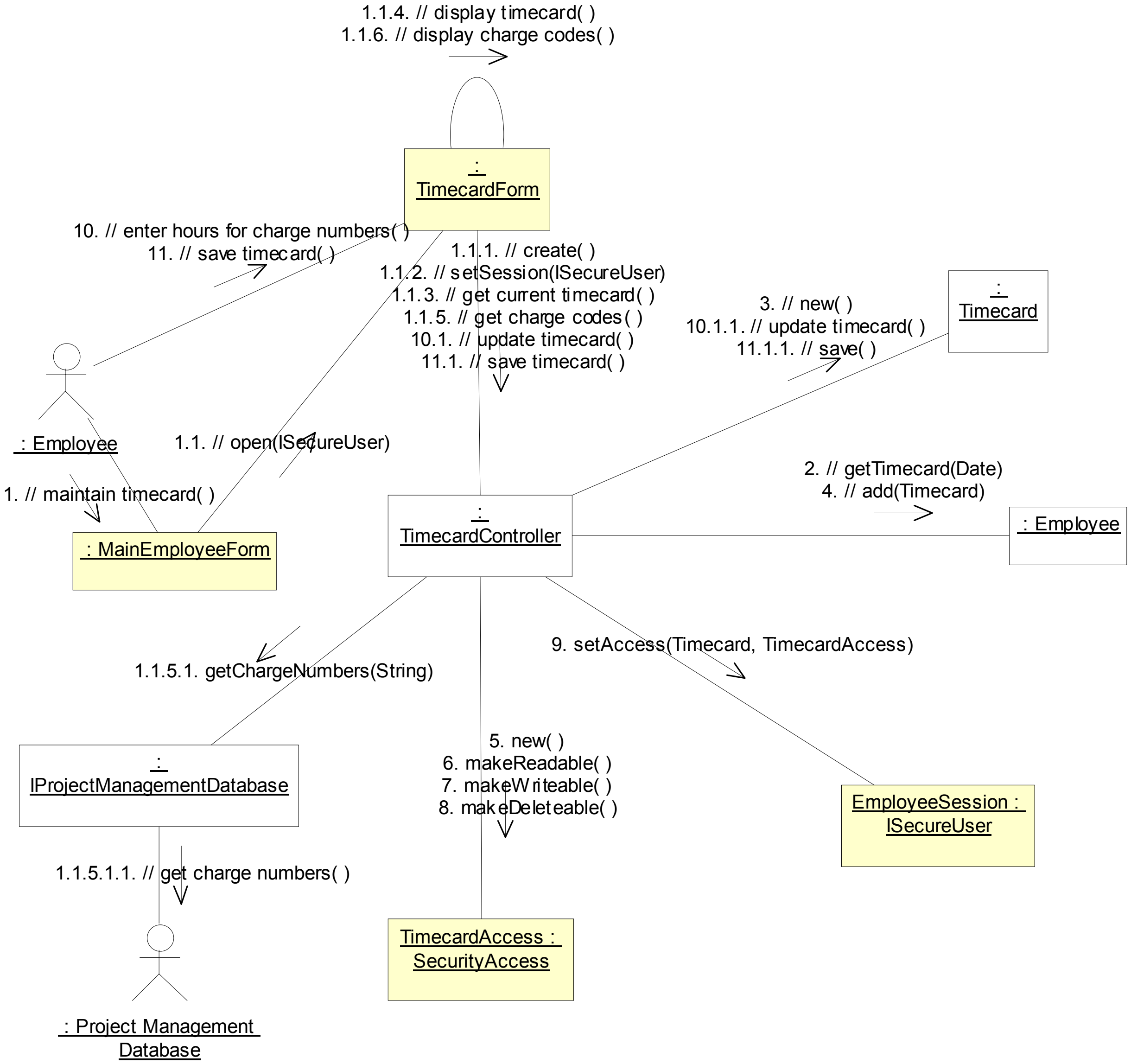
1.2.2 Maintain Timecard (with Security)

Maintain Timecard - Basic Flow (New Timecard with Security)

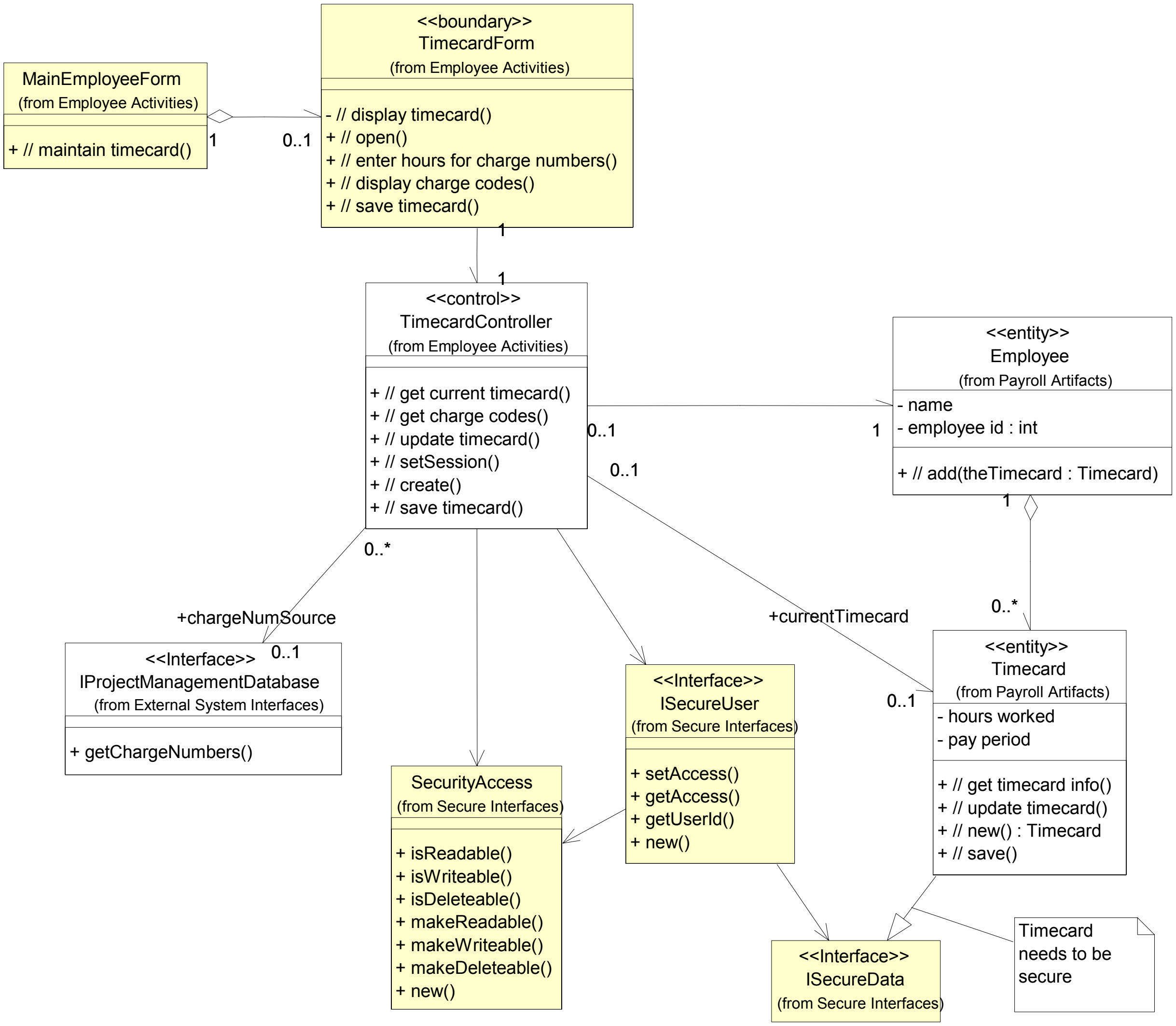


: Employee
: MainEmployeeForm
: TimecardForm
: TimecardController
: Employee
: Timecard
TimecardAccess : SecurityAccess
EmployeeSession : ISecureUser
: IProjectManagementDatabase
: Project Management
1. // maintain timecard( )
1.1. // open(ISecureUser)
1.1.1. // create( )
The secure user session for the employee established at Login must be propagated to all forms and
1.1.2. // setSession(ISecureUser)
A current timecard does not exist
1.1.3. // get current timecard( )
2. // getTimecard(Date)
A new Timecard is created and added to the Employee
3. // new( )
4. // add(Timecard)
5. new( )
The employee is given "open" permissions for his/her timecard
6. makeReadable( )
7. makeWriteable( )
8. makeDeleteable( )
9. setAccess(Timecard, TimecardAccess)
1.1.4. // display timecard( )
1.1.5. // get charge codes( )
1.1.5.1. getChargeNumbers(String)
1.1.5.1.1. // get charge numbers( )
1.1.6. // display charge codes( )
10. // enter hours for charge numbers( )
10.1. // update timecard( )
10.1.1. // update timecard( )
No need to check Timecard access permissions, the TimecardController alread knows that it is the Employee's Timecard.
11. // save timecard( )
11.1. // save timecard( )
11.1.1. // save( )

Maintain Timecard - Basic Flow (New Timecard with Security)

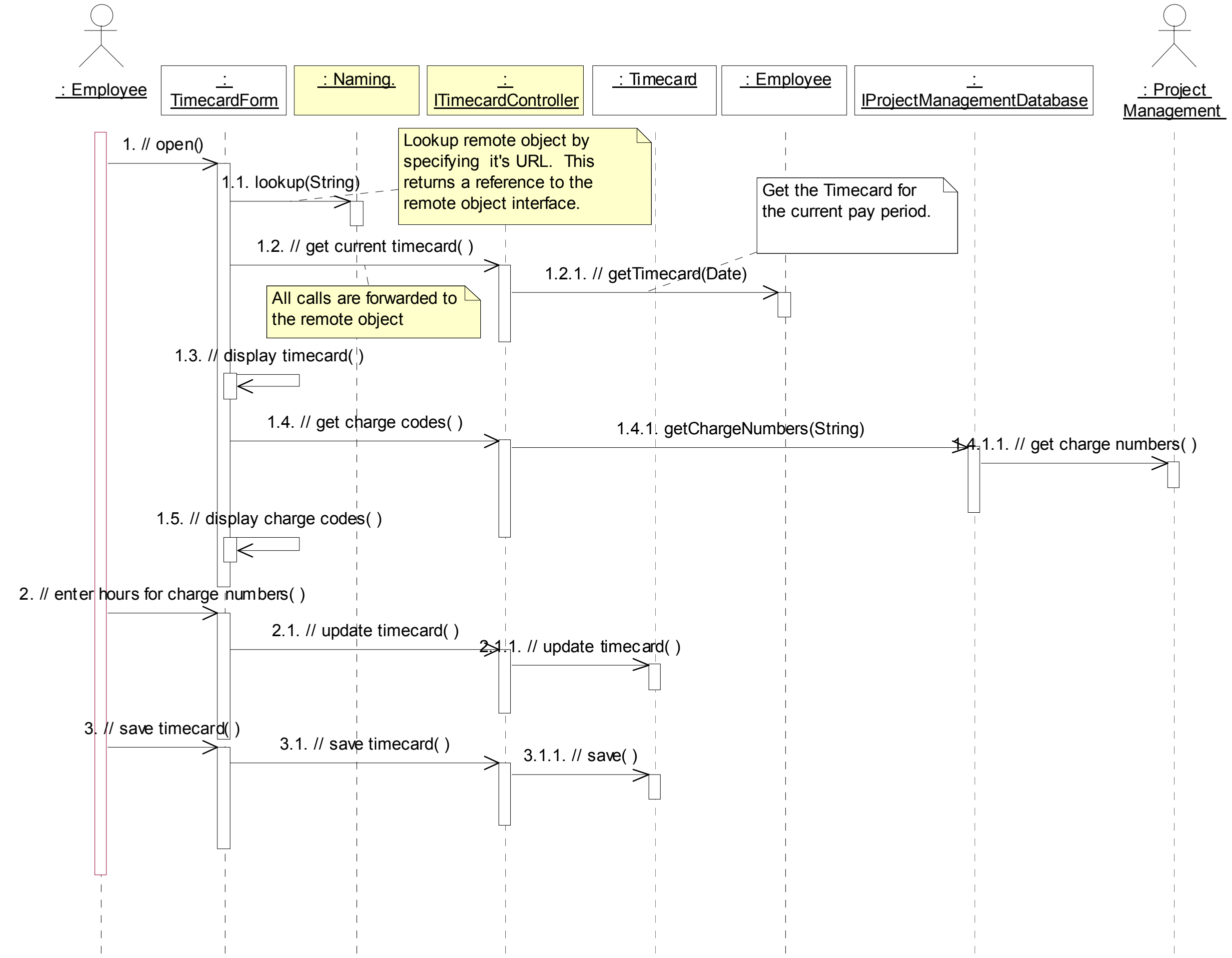

1.1.4. // display timecard( )
1.1.6. // display charge codes( )
: TimecardForm
10. // enter hours for charge numbers( )
11. // save timecard( )
1.1.1. // create( )
1.1.2. // setSession(ISecureUser)
1.1.3. // get current timecard( )
1.1.5. // get charge codes( )
10.1. // update timecard( )
11.1. // save timecard( )
3. // new( )
10.1.1. // update timecard( )
11.1.1. // save( )
: Timecard
: Employee
1.1. // open(ISecureUser)
1. // maintain timecard( )
: MainEmployeeForm
: TimecardController
2. // getTimecard(Date)
4. // add(Timecard)
: Employee
9. setAccess(Timecard, TimecardAccess)
1.1.5.1. getChargeNumbers(String)
: IProjectManagementDatabase
5. new( )
6. makeReadable( )
7. makeWriteable( )
8. makeDeleteable( )
EmployeeSession : ISecureUser
1.1.5.1.1. // get charge numbers( )
TimecardAccess : SecurityAccess
: Project Management Database

Maintain Timecard - VOPC (with Security)

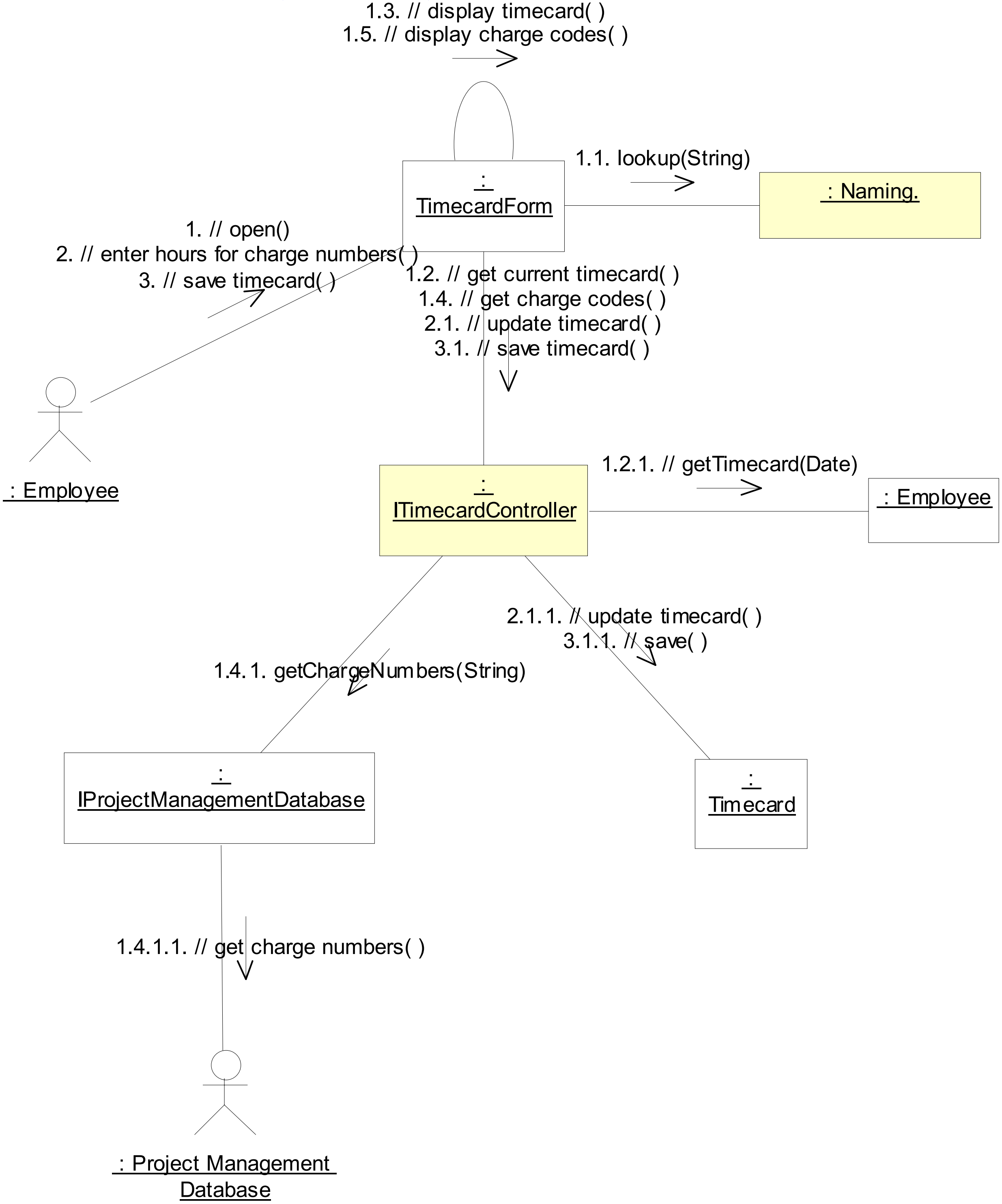


1.2.3 Maintain Timecard (with Distribution)

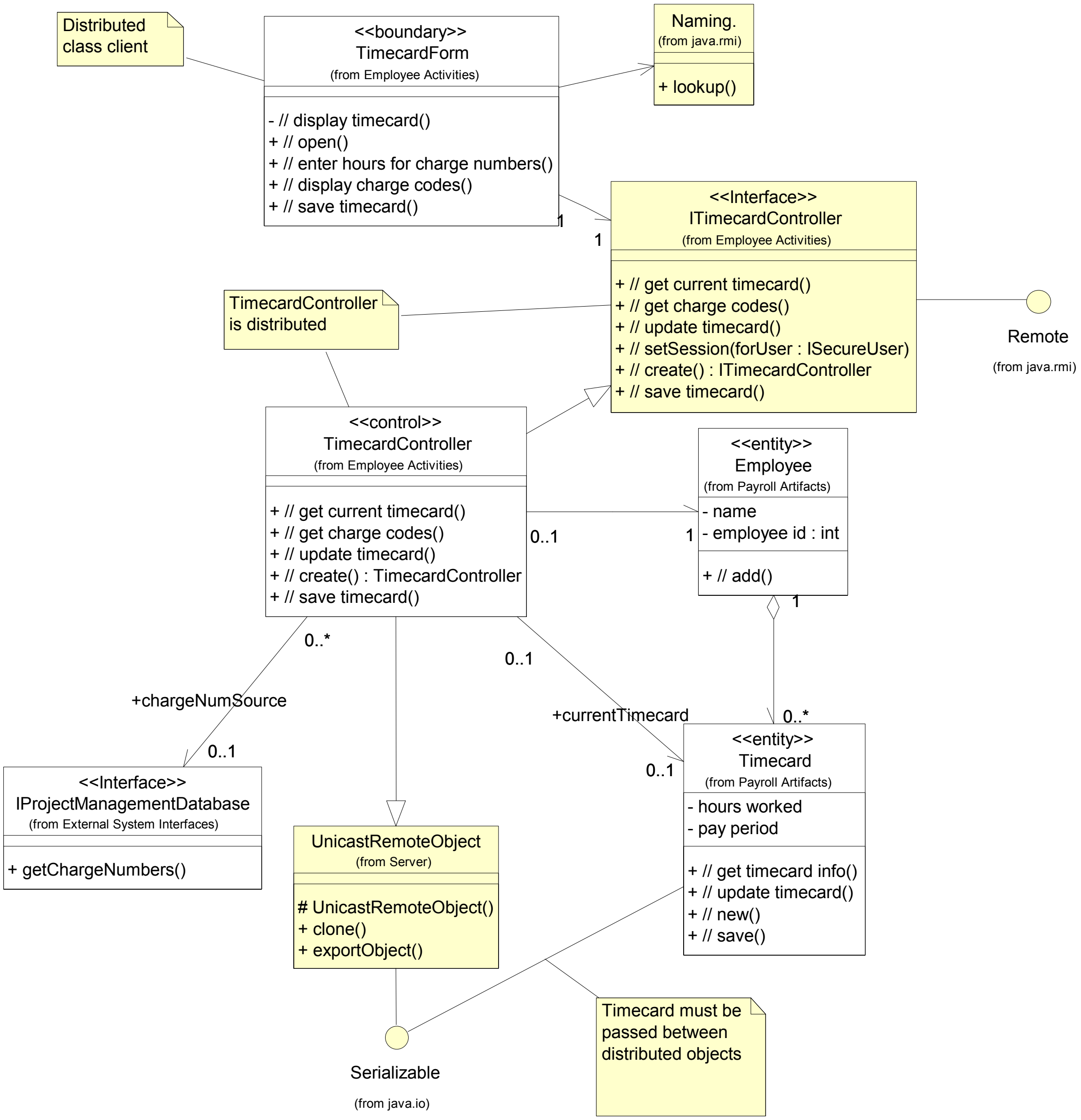
Maintain Timecard - Basic Flow (with distribution)


: Employee
: TimecardForm
: Naming.
: ITimecardController
: Timecard
: Employee
: IProjectManagementDatabase
: Project Management
1. // open()
1.1. lookup(String)
Lookup remote object by specifying it's URL. This returns a reference to the remote object interface.
Get the Timecard for the current pay period.
1.2. // get current timecard( )
1.2.1. // getTimecard(Date)
All calls are forwarded to the remote object
1.3. // display timecard( )
1.4. // get charge codes( )
1.4.1. getChargeNumbers(String)
1.4.1.1. // get charge numbers( )
1.5. // display charge codes( )
2. // enter hours for charge numbers( )
2.1. // update timecard( )
2.1.1. // update timecard( )
3. // save timecard( )
3.1. // save timecard( )
3.1.1. // save( )

Maintain Timecard - Basic Flow (with distribution)

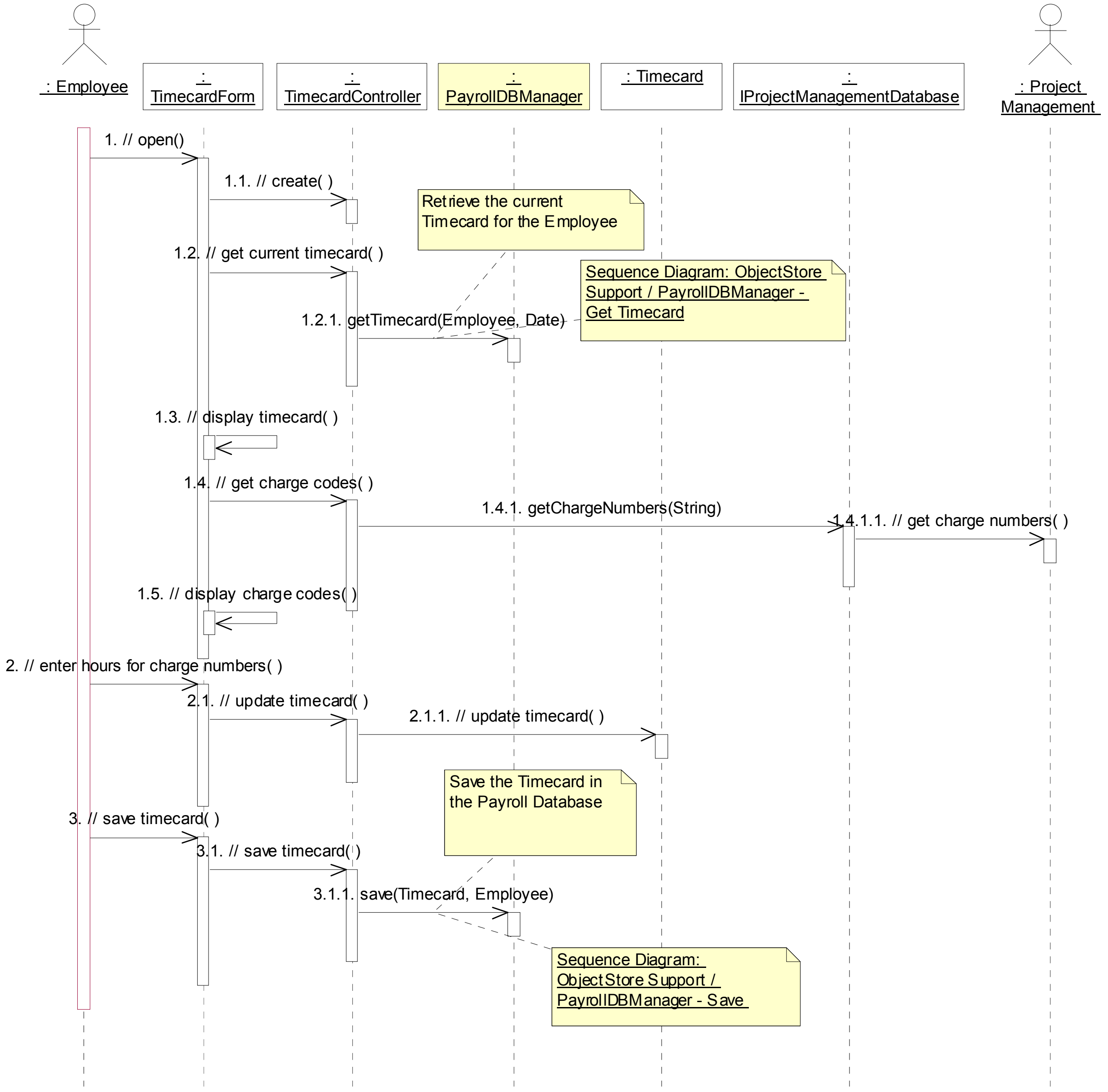

1.3. // display timecard( )
1.5. // display charge codes( )
: TimecardForm
1.1. lookup(String)
: Naming.
1. // open()
2. // enter hours for charge numbers( )
3. // save timecard( )
1.2. // get current timecard( )
1.4. // get charge codes( )
2.1. // update timecard( )
3.1. // save timecard( )
: Employee
1.2.1. // getTimecard(Date)
: ITimecardController
: Employee
2.1.1. // update timecard( )
3.1.1. // save( )
1.4.1. getChargeNumbers(String)
: IProjectManagementDatabase
: Timecard
1.4.1.1. // get charge numbers( )
: Project Management Database

Maintain Timecard - VOPC (with Distribution)



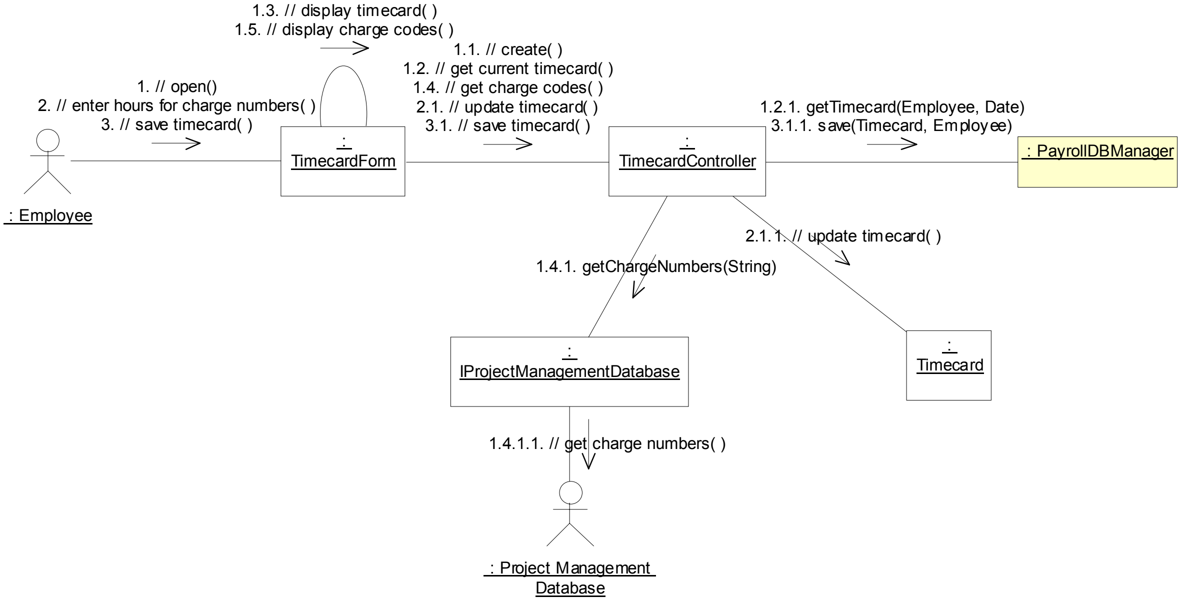
1.2.4 Maintain Timecard (with OODBMS Persistence)

Maintain Timecard - Basic Flow (with OODBMS Persistency)

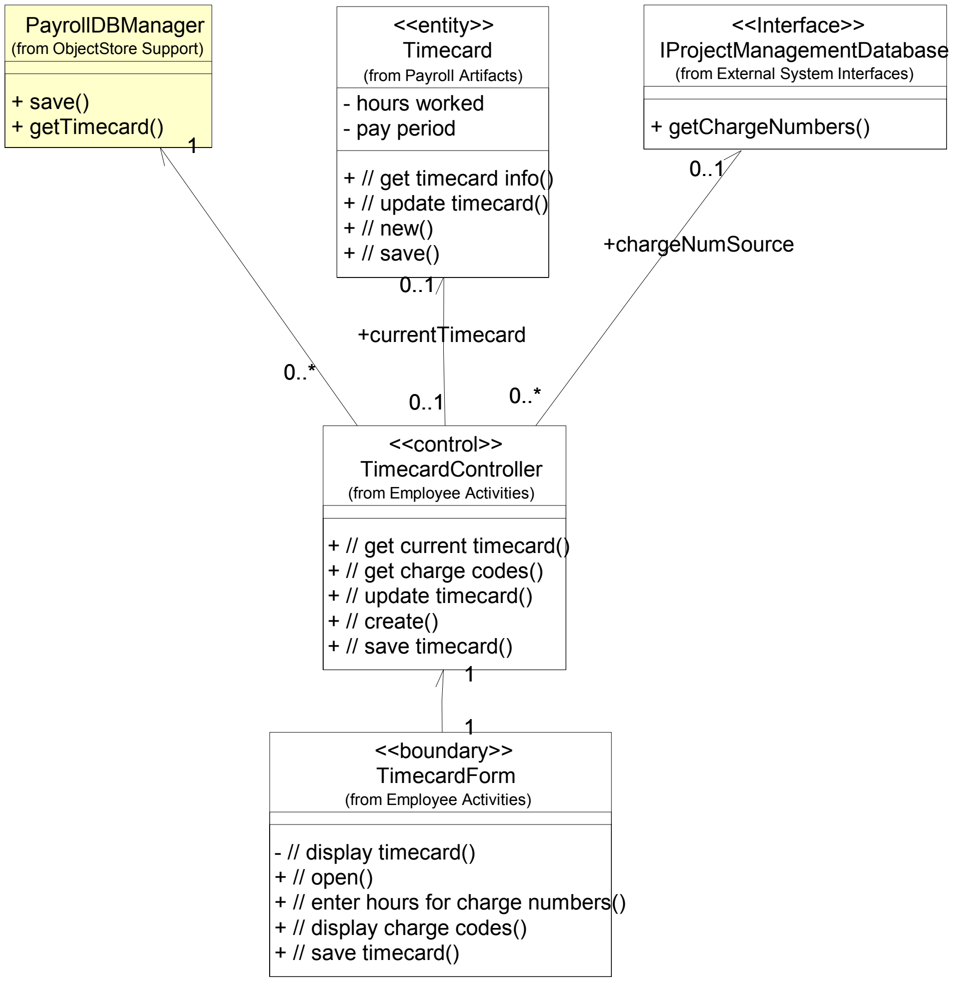


: Employee
: TimecardForm
: TimecardController
: PayrollDBManager
: Timecard
: IProjectManagementDatabase
: Project Management
1. // open()
1.1. // create( )
Retrieve the current Timecard for the Employee
1.2. // get current timecard( )
Sequence Diagram: ObjectStore Support / PayrollDBManager - Get Timecard
1.2.1. getTimecard(Employee, Date)
1.3. // display timecard( )
1.4. // get charge codes( )
1.4.1. getChargeNumbers(String)
1.4.1.1. // get charge numbers( )
1.5. // display charge codes( )
2. // enter hours for charge numbers( )
2.1. // update timecard( )
2.1.1. // update timecard( )
Save the Timecard in the Payroll Database
3. // save timecard( )
3.1. // save timecard( )
3.1.1. save(Timecard, Employee)
Sequence Diagram: ObjectStore Support / PayrollDBManager - Save

Maintain Timecard - Basic Flow (with OODBMS Persistency)

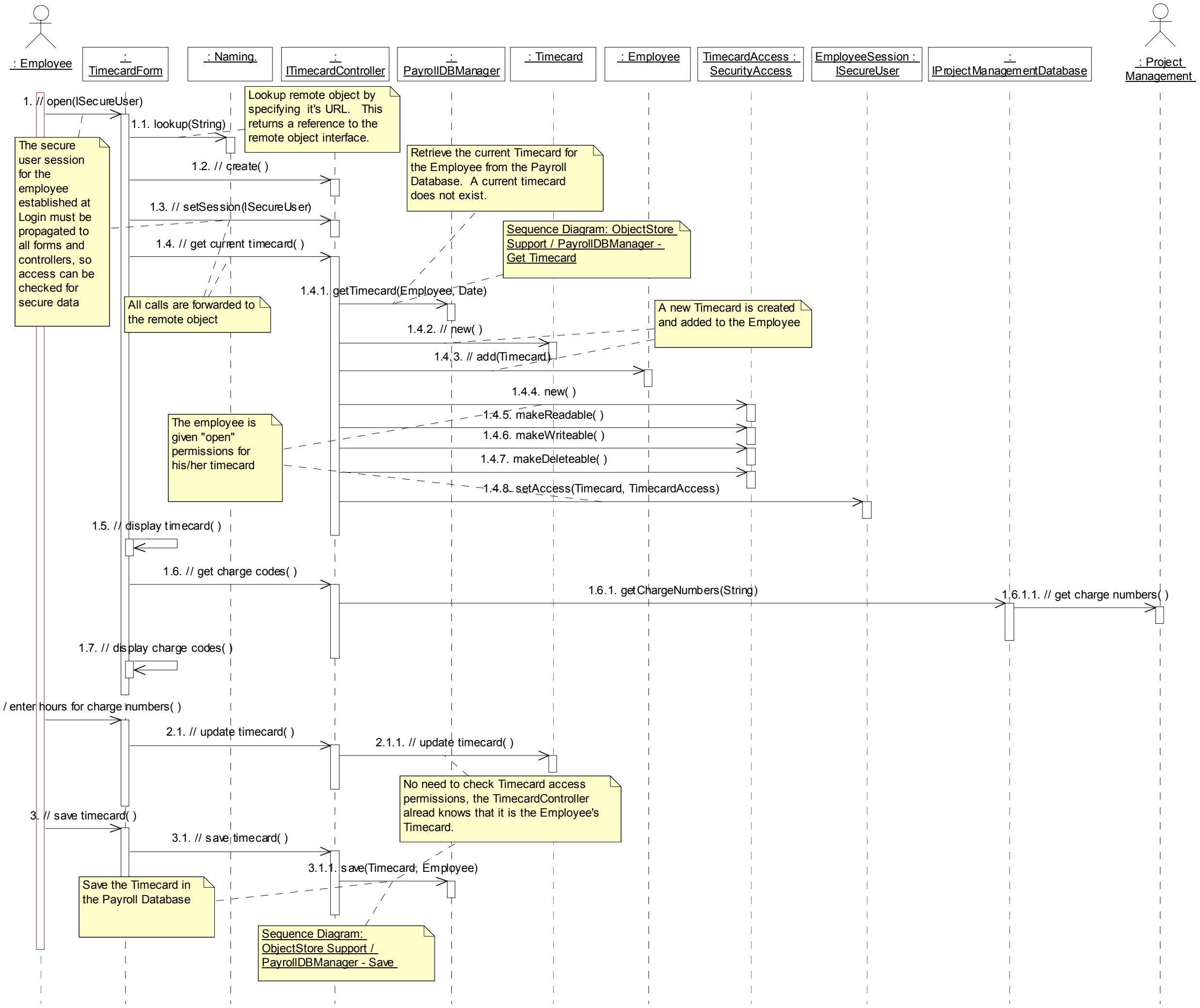

1.3. // display timecard( )
1.5. // display charge codes( )
1. // open()
2. // enter hours for charge numbers( )
3. // save timecard( )
1.1. // create( )
1.2. // get current timecard( )
1.4. // get charge codes( )
2.1. // update timecard( )
3.1. // save timecard( )
1.2.1. getTimecard(Employee, Date)
3.1.1. save(Timecard, Employee)
: Employee
: TimecardForm
: TimecardController
: PayrollDBManager
2.1.1. // update timecard( )
1.4.1. getChargeNumbers(String)
: IProjectManagementDatabase
: Timecard
1.4.1.1. // get charge numbers( )
: Project Management Database

Maintain Timecard - VOPC (with OODBMS Persistency)



1.2.5 Maintain Timecard (with everything)

Maintain Timecard - Basic Flow (with everything)


: Employee
: TimecardForm
: Naming.
: ITimecardController
: PayrollDBManager
: Timecard
: Employee
TimecardAccess : SecurityAccess
EmployeeSession : ISecureUser
: IProjectManagementDatabase
: Project Management
1. // open(ISecureUser)
1.1. lookup(String)
1.2. // create( )
1.3. // setSession(ISecureUser)
1.4. // get current timecard( )
1.4.1. getTimecard(Employee, Date)
1.4.2. // new( )
1.4.3. // add(Timecard)
1.4.4. new( )
1.4.5. makeReadable( )
1.4.6. makeWriteable( )
1.4.7. makeDeleteable( )
1.4.8. setAccess(Timecard, TimecardAccess)
1.5. // display timecard( )
1.6. // get charge codes( )
1.6.1. getChargeNumbers(String)
1.6.1.1. // get charge numbers( )
1.7. // display charge codes( )
/ enter hours for charge numbers( )
2.1. // update timecard( )
2.1.1. // update timecard( )
3. // save timecard( )
3.1. // save timecard( )
3.1.1. save(Timecard, Employee)

The secure user session for the employee established at Login must be propagated to all forms and controllers, so access can be checked for secure data

Lookup remote object by specifying it's URL. This returns a reference to the remote object interface.

Retrieve the current Timecard for the Employee from the Payroll Database. A current timecard does not exist.

Sequence Diagram: ObjectStore Support / PayrollDBManager - Get Timecard

All calls are forwarded to the remote object

A new Timecard is created and added to the Employee

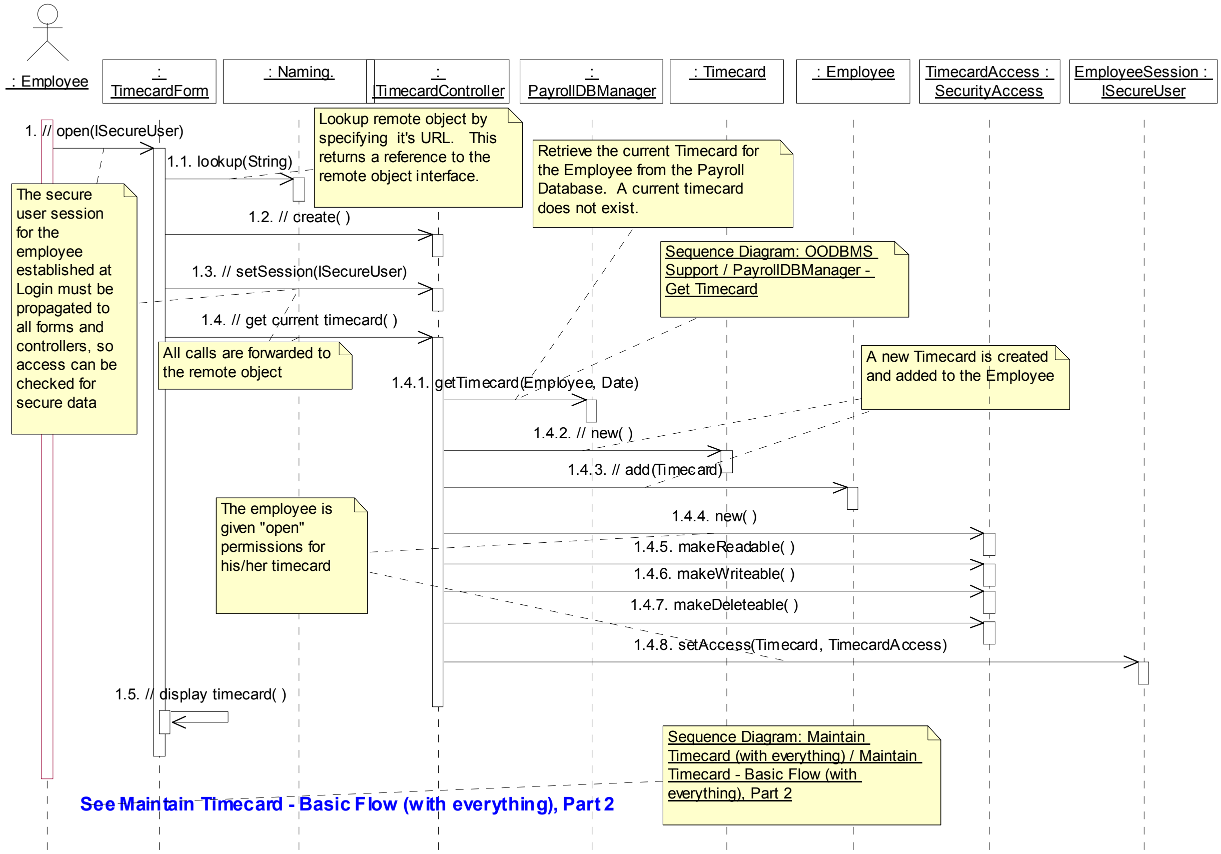
The employee is given "open" permissions for his/her timecard

No need to check Timecard access permissions, the TimecardController alread knows that it is the Employee's Timecard.

Save the Timecard in the Payroll Database

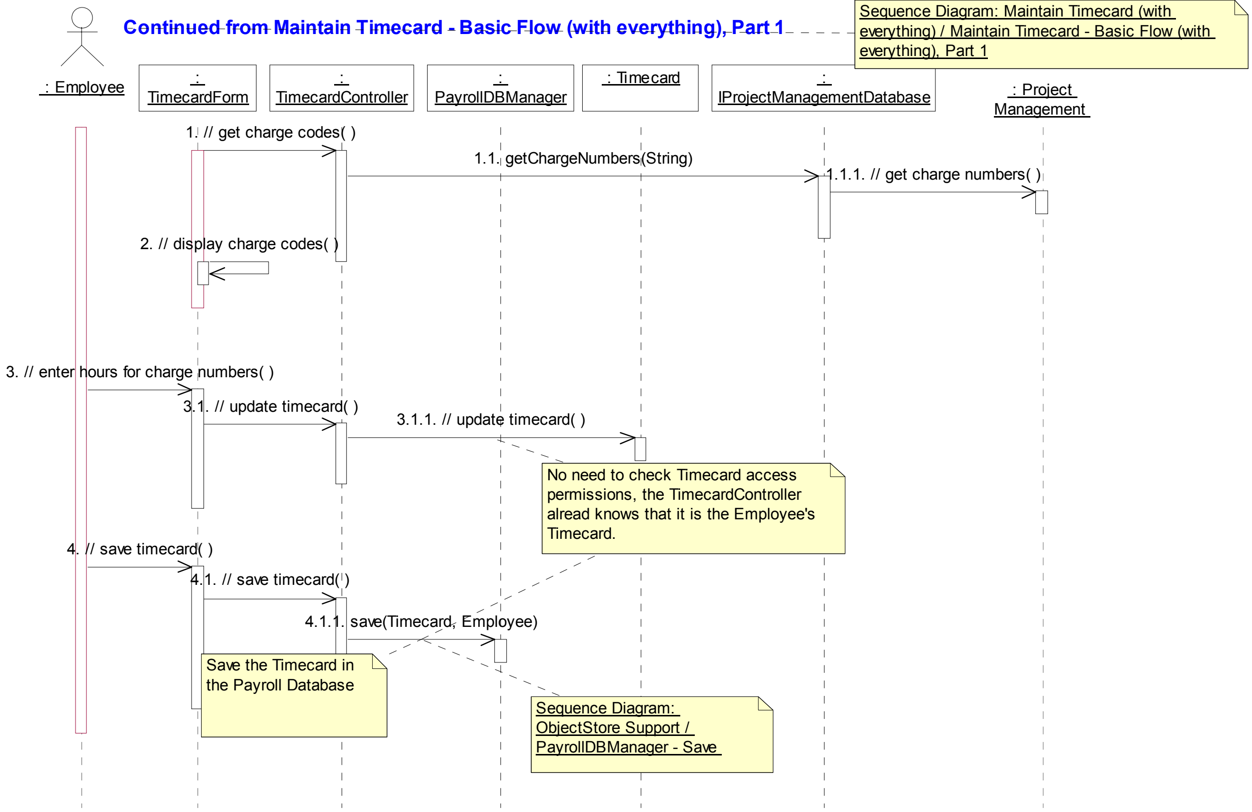
Sequence Diagram: ObjectStore Support / PayrollDBManager - Save

Maintain Timecard - Basic Flow (with everything), Part 1



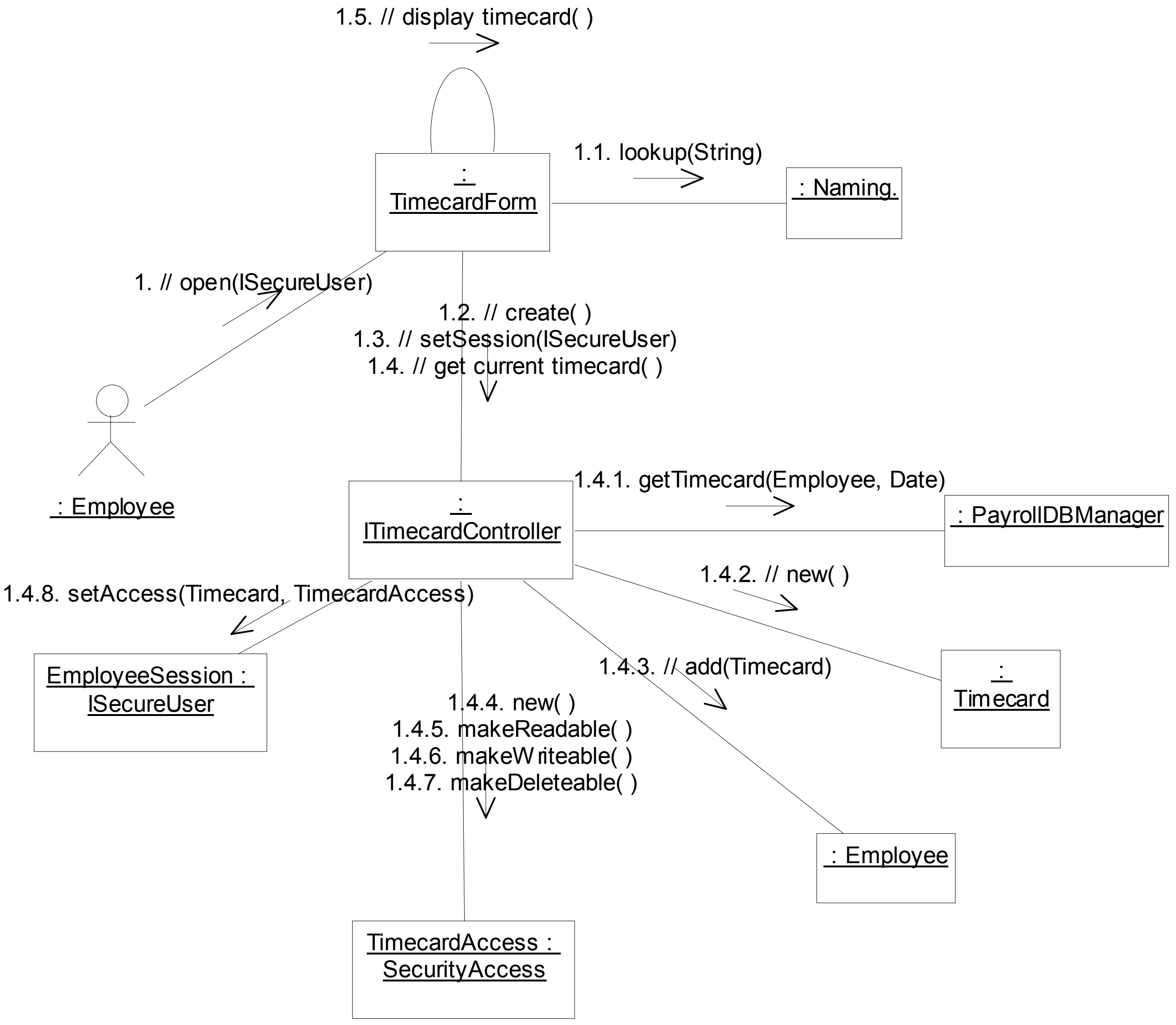
: Employee
: TimecardForm
: Naming.
: ITimecardController
: PayrollDBManager
: Timecard
: Employee
TimecardAccess : SecurityAccess
EmployeeSession : ISecureUser
1. // open(ISecureUser)
1.1. lookup(String)
Lookup remote object by specifying it's URL. This returns a reference to the remote object interface.
Retrieve the current Timecard for the Employee from the Payroll Database. A current timecard does not exist.
The secure user session for the employee established at Login must be propagated to all forms and controllers, so access can be checked for secure data
1.2. // create( )
Sequence Diagram: OODBMS Support / PayrollDBManager - Get Timecard
1.3. // setSession(ISecureUser)
1.4. // get current timecard( )
All calls are forwarded to the remote object
A new Timecard is created and added to the Employee
1.4.1. getTimecard(Employee, Date)
1.4.2. // new( )
1.4.3. // add(Timecard)
The employee is given "open" permissions for his/her timecard
1.4.4. new( )
1.4.5. makeReadable( )
1.4.6. makeWriteable( )
1.4.7. makeDeleteable( )
1.4.8. setAccess(Timecard, TimecardAccess)
1.5. // display timecard( )
Sequence Diagram: Maintain Timecard (with everything) / Maintain Timecard - Basic Flow (with everything), Part 2
See Maintain Timecard - Basic Flow (with everything), Part 2

Maintain Timecard - Basic Flow (with everything), Part 2

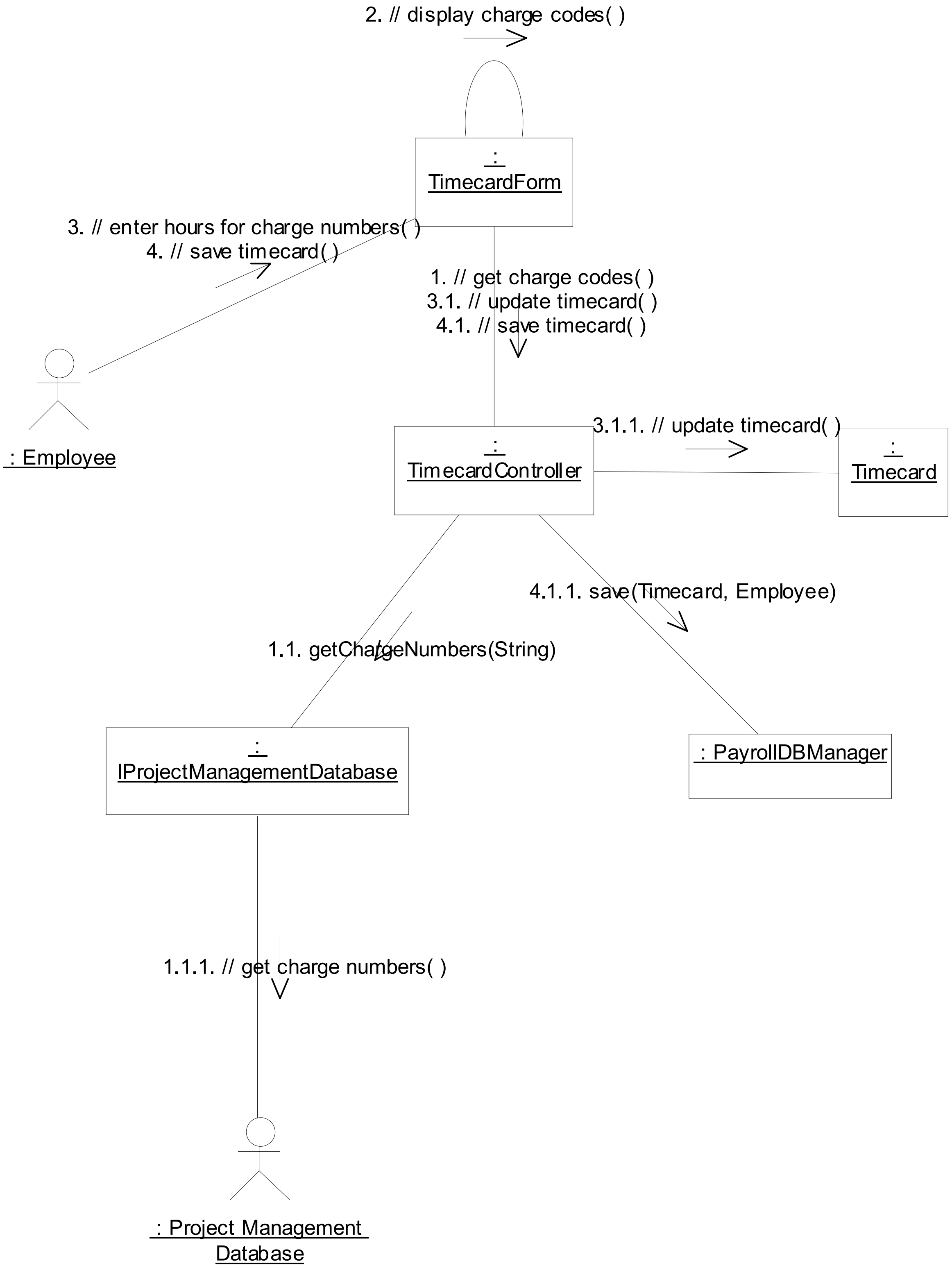


Continued from Maintain Timecard - Basic Flow (with everything), Part 1
Sequence Diagram: Maintain Timecard (with everything) / Maintain Timecard - Basic Flow (with everything), Part 1
: Employee
: TimecardForm
: TimecardController
: PayrollDBManager
: Timecard
: IProjectManagementDatabase
: Project Management
1. // get charge codes( )
1.1. getChargeNumbers(String)
1.1.1. // get charge numbers( )
2. // display charge codes( )
3. // enter hours for charge numbers( )
3.1. // update timecard( )
3.1.1. // update timecard( )
No need to check Timecard access permissions, the TimecardController alread knows that it is the Employee's Timecard.
4. // save timecard( )
4.1. // save timecard( )
4.1.1. save(Timecard, Employee)
Save the Timecard in the Payroll Database
Sequence Diagram: ObjectStore Support / PayrollDBManager - Save

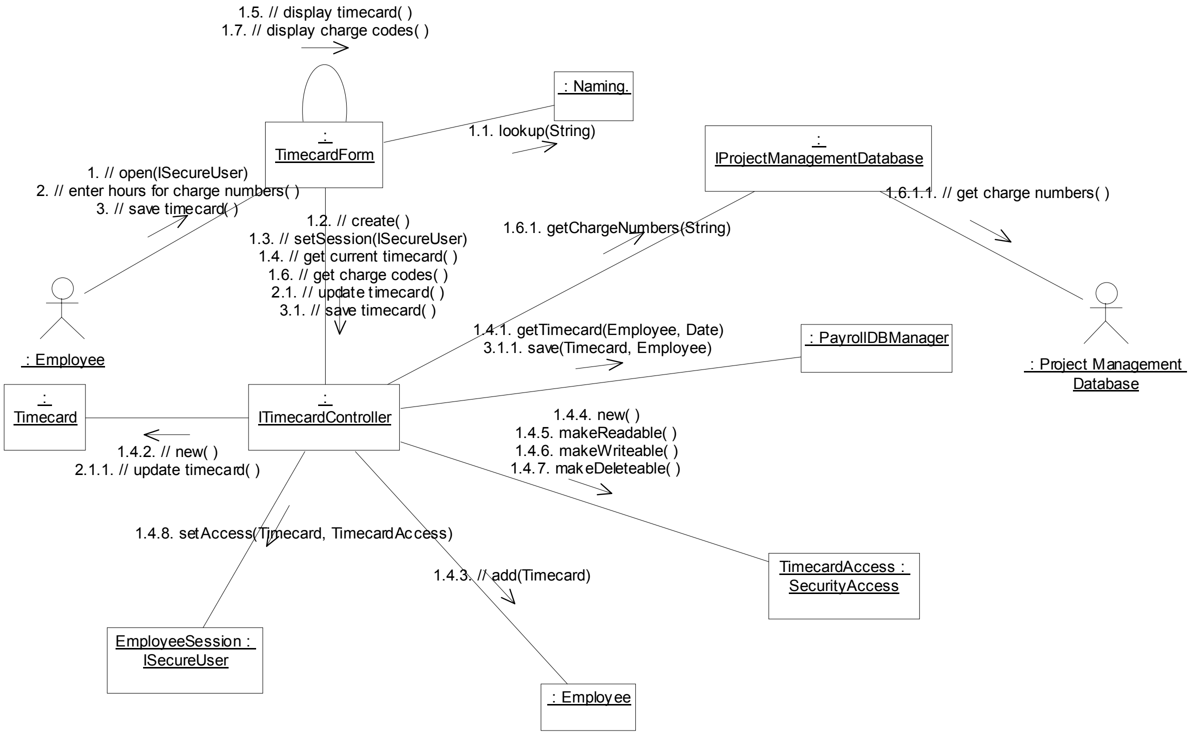
Maintain Timecard - Basic Flow (with everything), Part 1



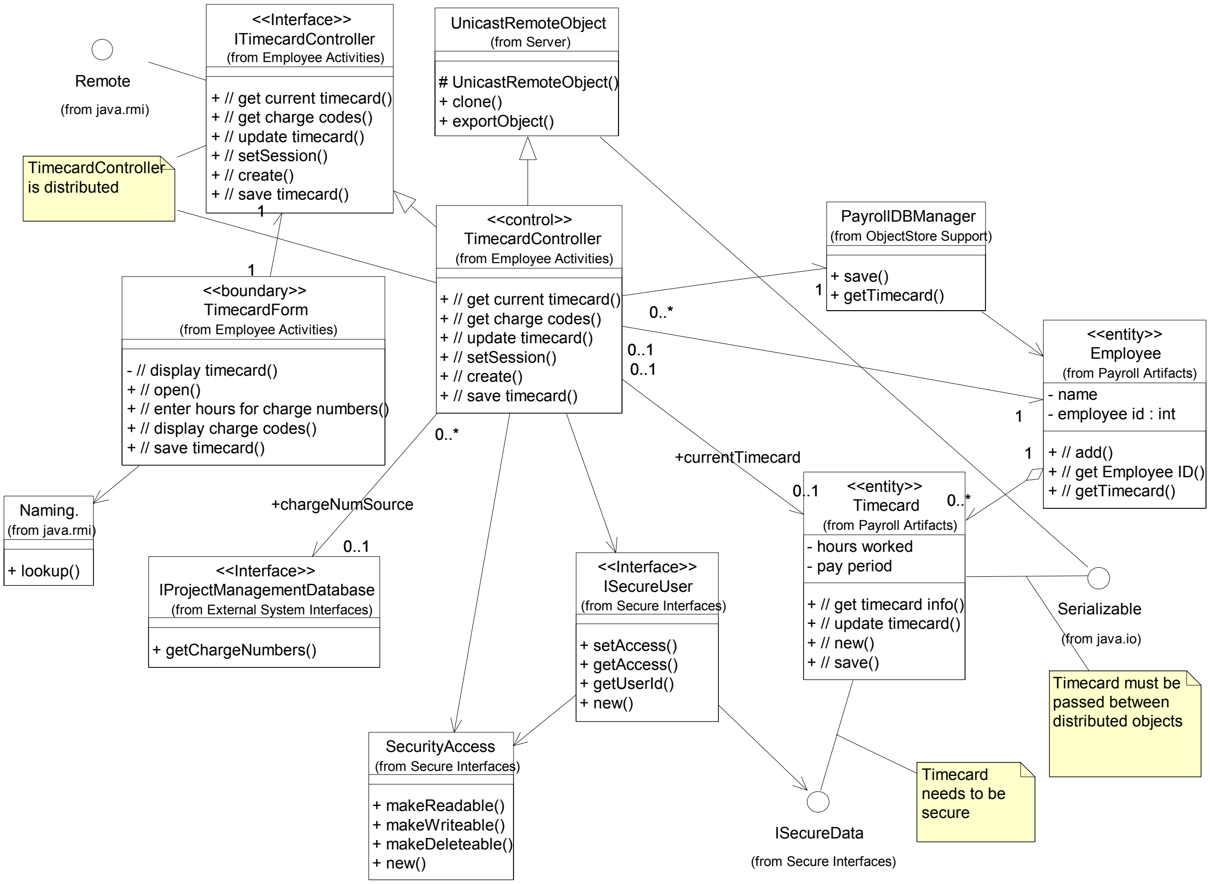
Maintain Timecard - Basic Flow (with everything), Part 2


2. // display charge codes( )
: TimecardForm
3. // enter hours for charge numbers( )
4. // save timecard( )
1. // get charge codes( )
3.1. // update timecard( )
4.1. // save timecard( )
: Employee
: TimecardController
3.1.1. // update timecard( )
: Timecard
4.1.1. save(Timecard, Employee)
1.1. getChargeNumbers(String)
: IProjectManagementDatabase
: PayrollDBManager
1.1.1. // get charge numbers( )
: Project Management Database

Maintain Timecard - Basic Flow (with everything)


1.5. // display timecard( )
1.7. // display charge codes( )
: Naming.
: TimecardForm
1.1. lookup(String)
: IProjectManagementDatabase
1. // open(ISecureUser)
2. // enter hours for charge numbers( )
3. // save timecard( )
1.6.1.1. // get charge numbers( )
1.2. // create( )
1.3. // setSession(ISecureUser)
1.4. // get current timecard( )
1.6. // get charge codes( )
2.1. // update timecard( )
3.1. // save timecard( )
1.6.1. getChargeNumbers(String)
1.4.1. getTimecard(Employee, Date)
3.1.1. save(Timecard, Employee)
: PayrollDBManager
: Employee
: Project Management Database
: Timecard
: ITimecardController
1.4.2. // new( )
2.1.1. // update timecard( )
1.4.4. new( )
1.4.5. makeReadable( )
1.4.6. makeWriteable( )
1.4.7. makeDeleteable( )
1.4.8. setAccess(Timecard, TimecardAccess)
1.4.3. // add(Timecard)
TimecardAccess : SecurityAccess
EmployeeSession : ISecureUser
: Employee

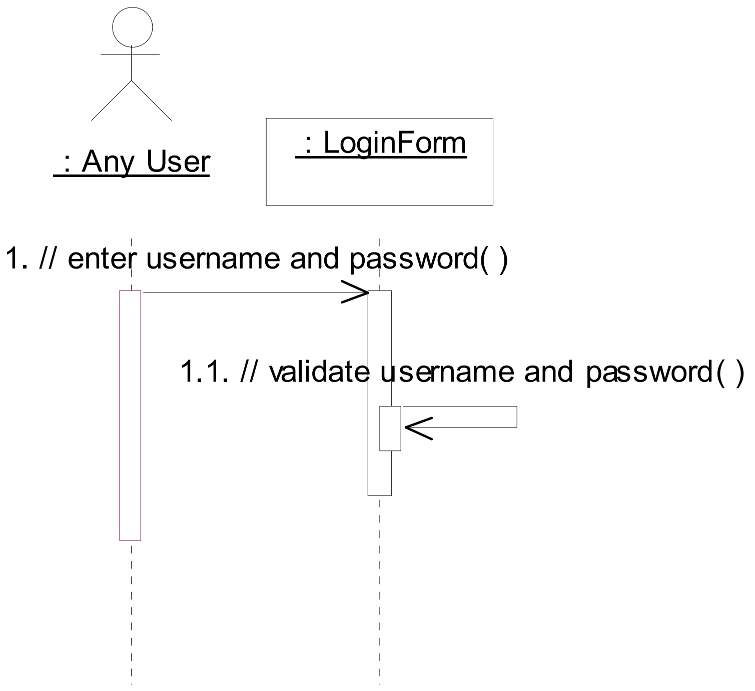
Maintain Timecard - VOPC (with everything)



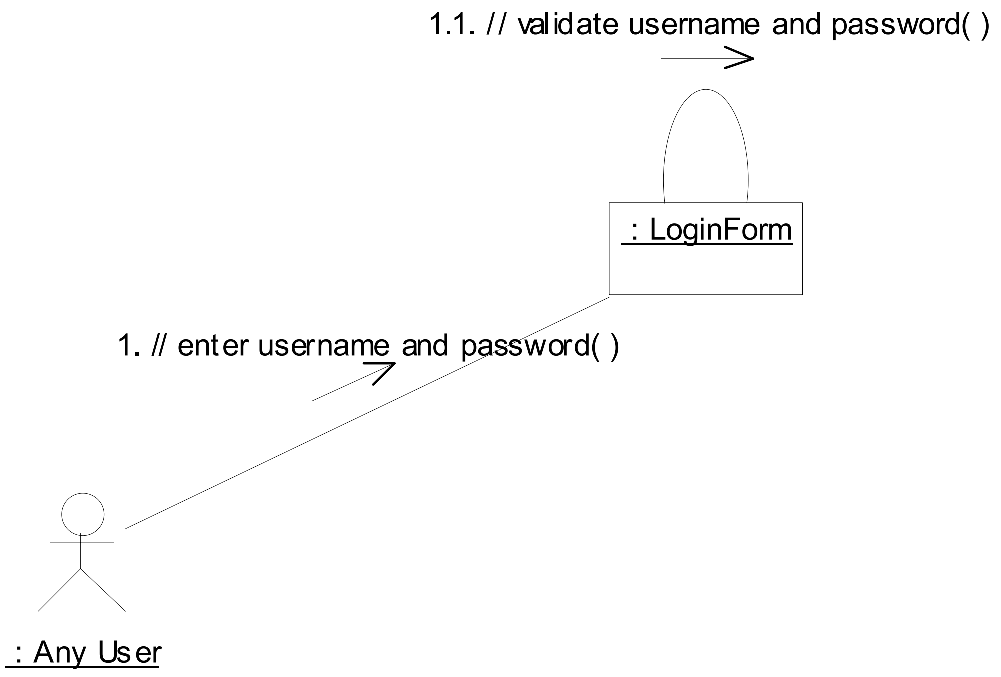
1.3 Use-Case Realization - Login

1.3.1 Login

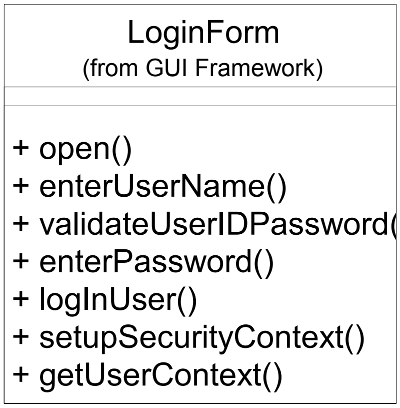
Login - Basic Flow


: Any User
: LoginForm
1. // enter username and password( )
1.1. // validate username and password( )

Login - Basic Flow


1.1. // validate username and password( )
: LoginForm
1. // enter username and password( )
: Any User

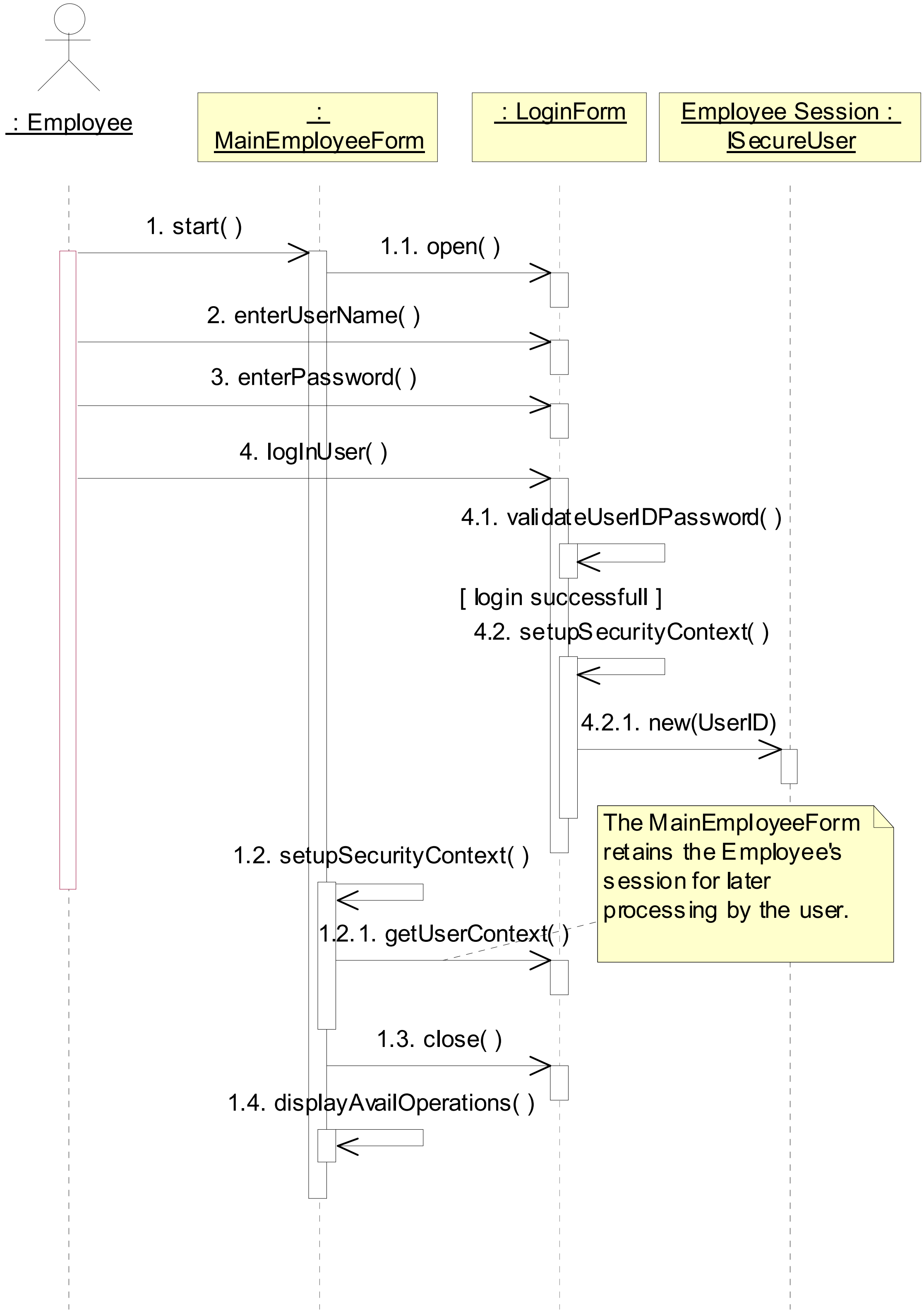
Login - VOPC



LoginForm
(from GUI Framework)
+ open()
+ enterUserName()
+ validateUserIDPassword(
+ enterPassword()
+ logInUser()
+ setupSecurityContext()
+ getUserContext()

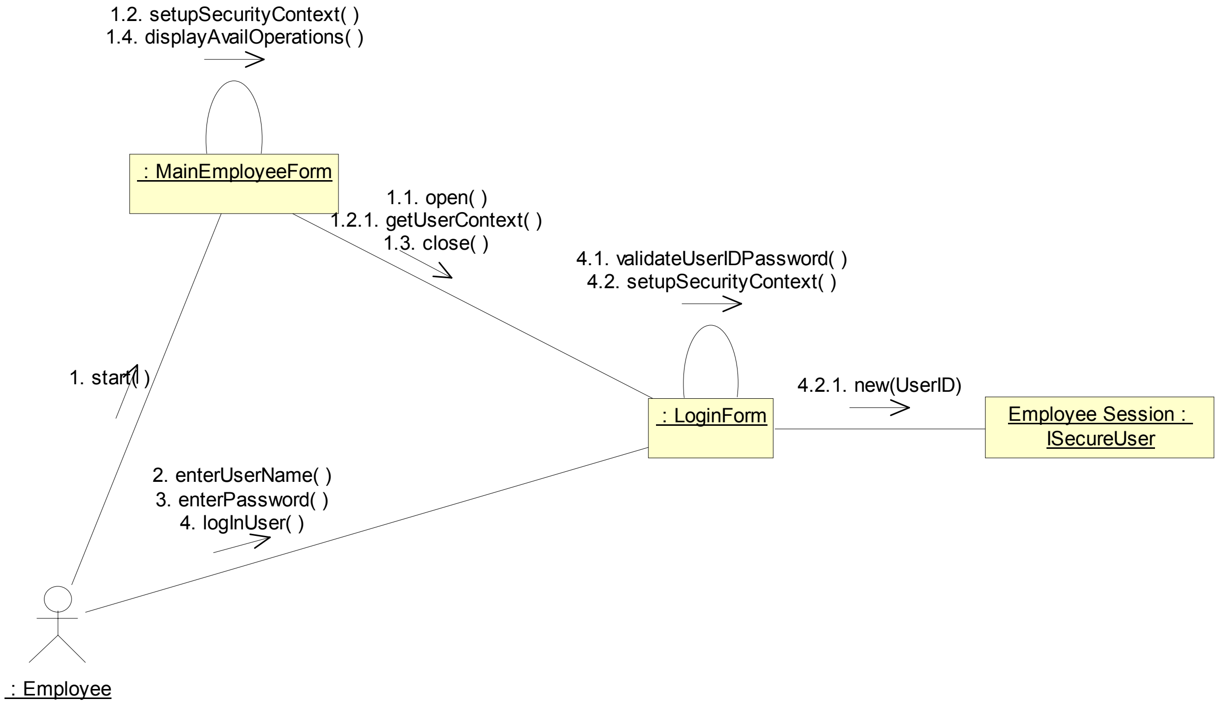
1.3.2 Login (with Security)

Login - Basic Flow (Employee Login with Security)

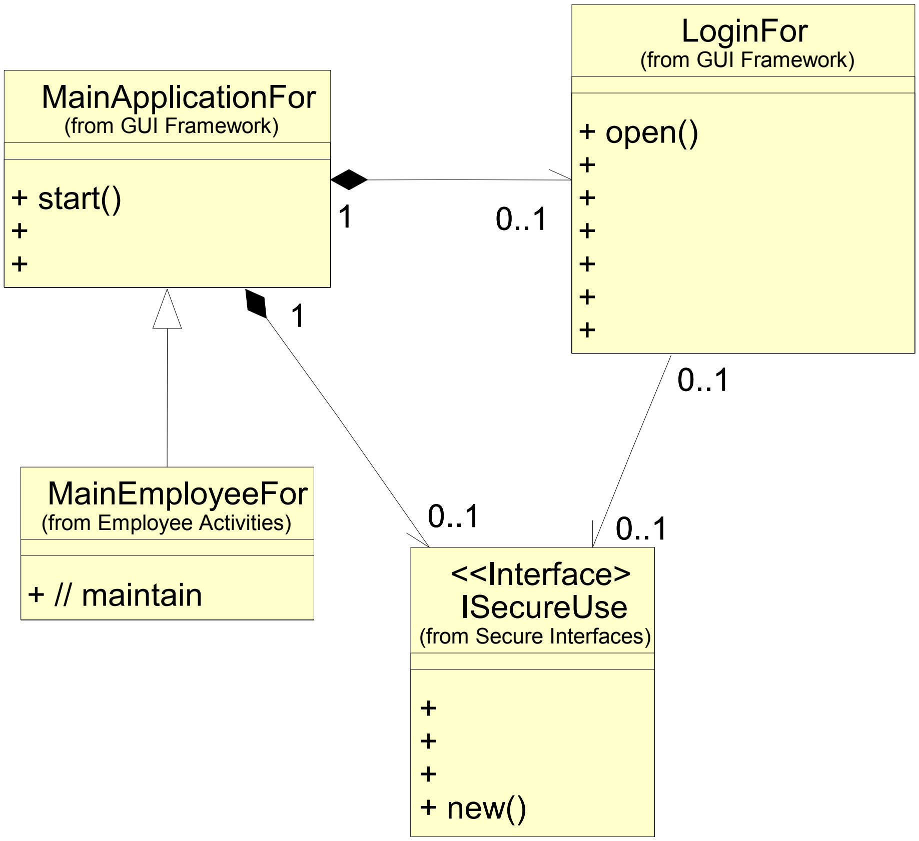


: Employee
: MainEmployeeForm
: LoginForm
Employee Session : ISecureUser
1. start( )
1.1. open( )
2. enterUserName( )
3. enterPassword( )
4. logInUser( )
4.1. validateUserIDPassword( )
[login successfull]
4.2. setupSecurityContext( )
4.2.1. new(UserID)
The MainEmployeeForm retains the Employee's session for later processing by the user.
1.2. setupSecurityContext( )
1.2.1. getUserContext( )
1.3. close( )
1.4. displayAvailOperations( )

Login - Basic Flow (Employee Login with Security)



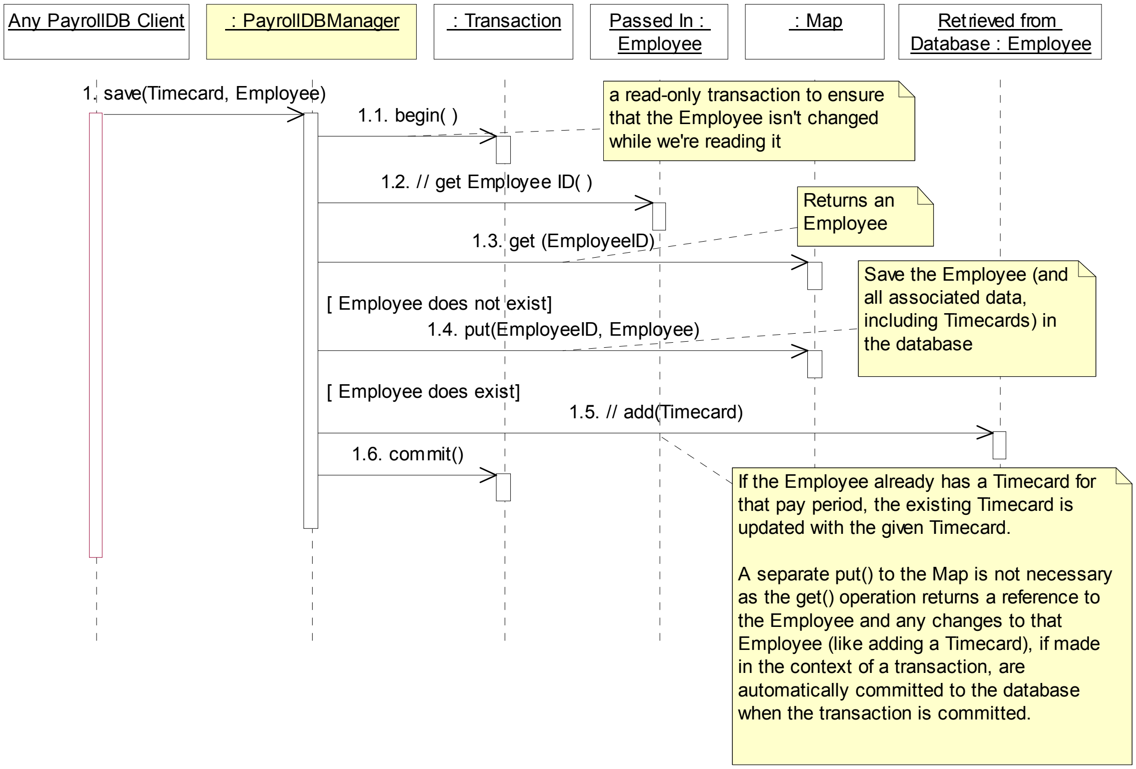
Login - VOPC (with Security)



1.4 ObjectStore Support

The following diagrams demonstrate the design of the PayrollDBManager class operations. These are included to supplement the use-case realization diagrams provided above. For the use-case realization diagrams that involve OODBMS persistency, there are references to the diagrams in this section.

PayrollDBManager - Save Timecard



a read-only transaction to ensure that the Employee isn't changed while we're reading it

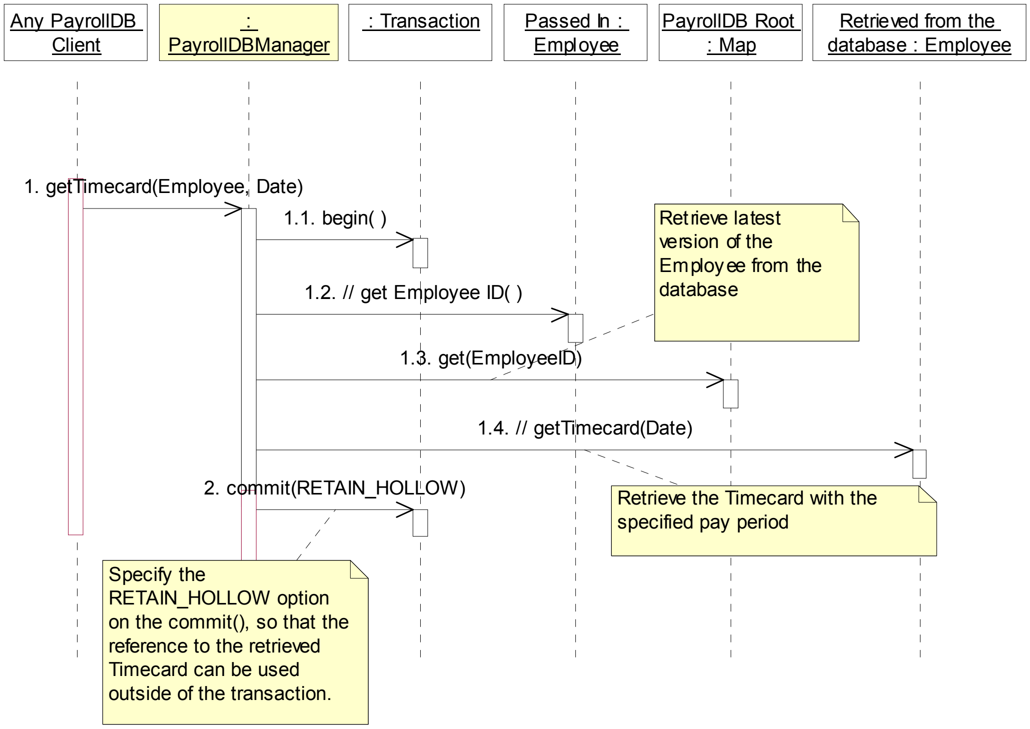
Returns an Employee

Save the Employee (and all associated data, including Timecards) in the database

If the Employee already has a Timecard for that pay period, the existing Timecard is updated with the given Timecard.

A separate put() to the Map is not necessary as the get() operation returns a reference to the Employee and any changes to that Employee (like adding a Timecard), if made in the context of a transaction, are automatically committed to the database when the transaction is committed.

PayrollDBManager - Get Timecard

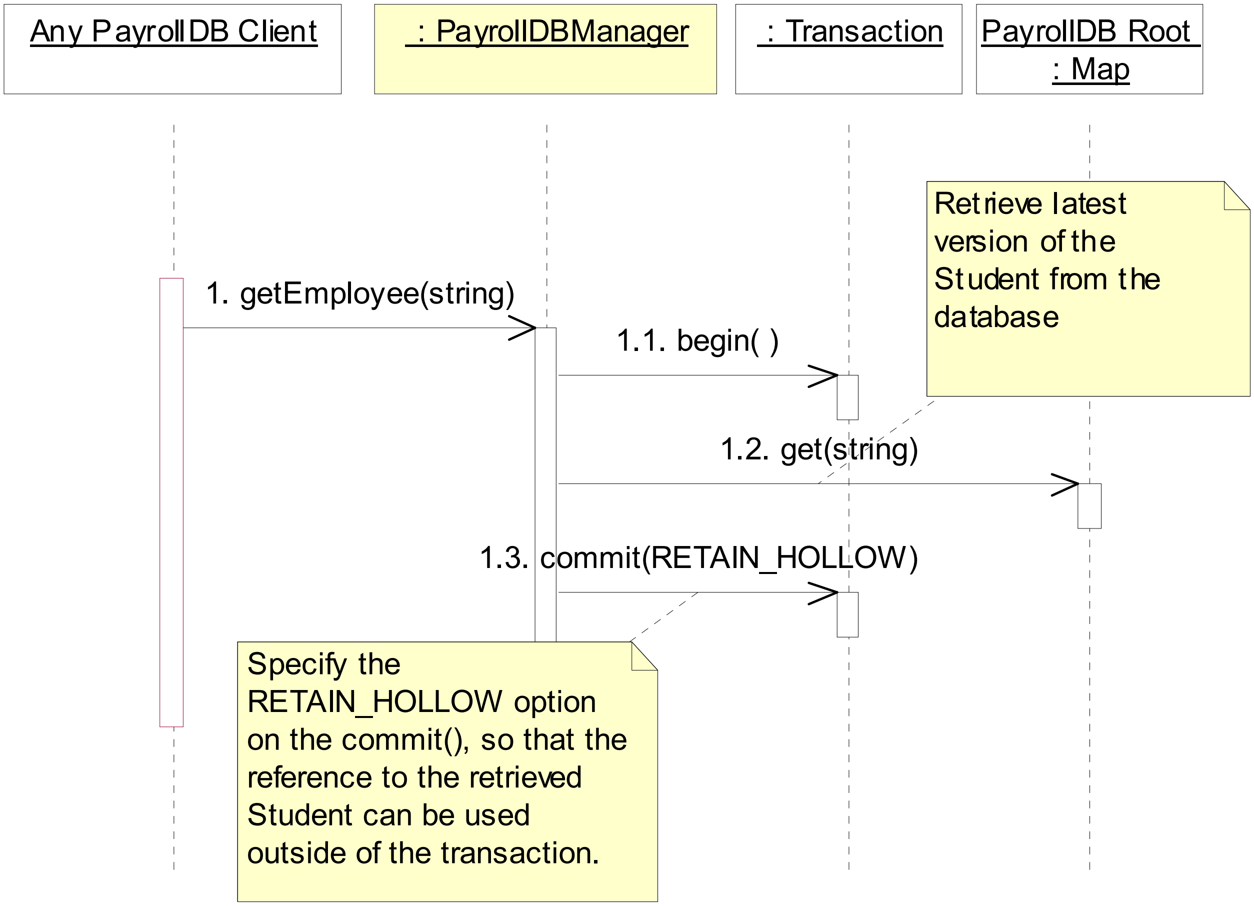


Retrieve latest version of the Employee from the database

Retrieve the Timecard with the specified pay period

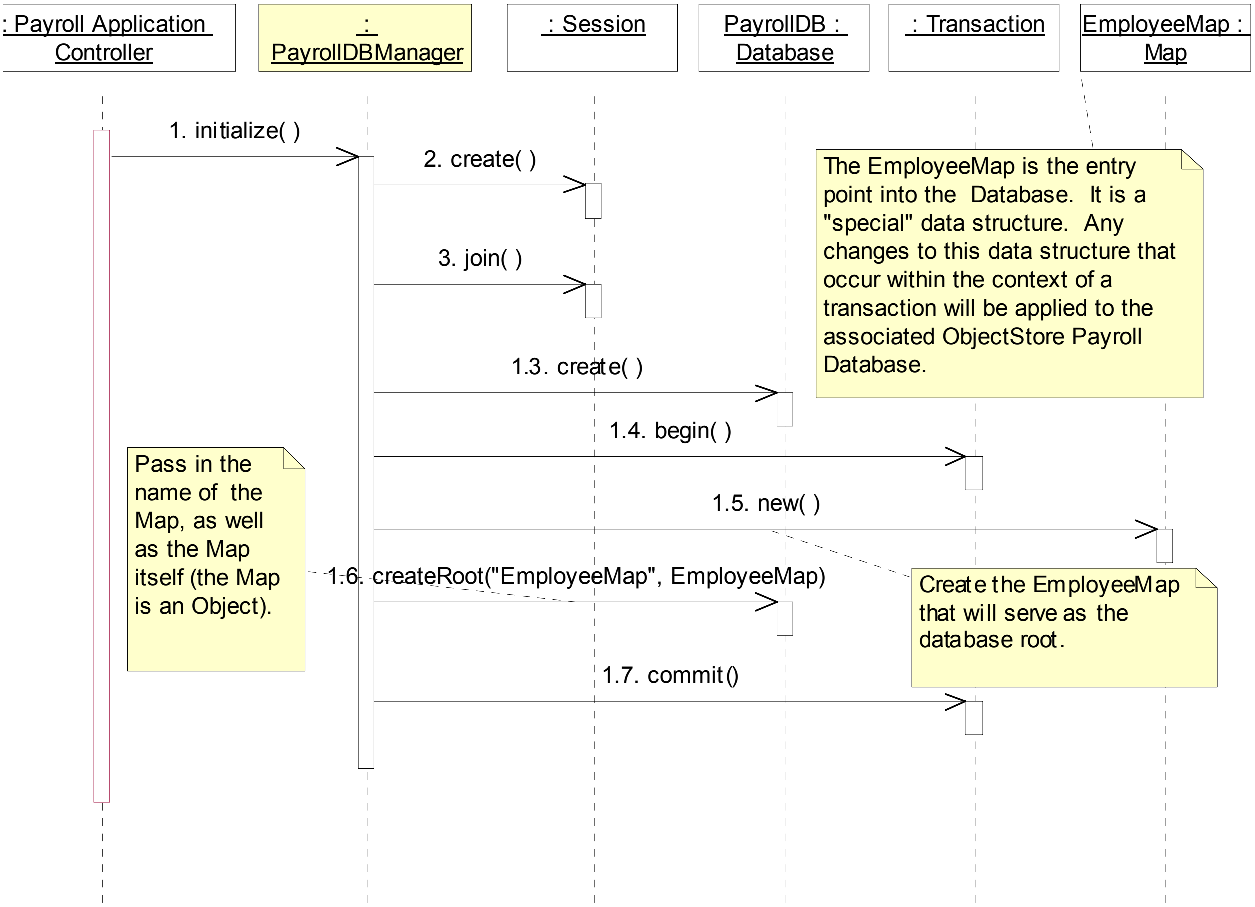
Specify the RETAIN_HOLLOW option on the commit(), so that the reference to the retrieved Timecard can be used outside of the transaction.

PayrollDBManager - Get Employee



Any PayrollDB Client
: PayrollDBManager
: Transaction
PayrollDB Root : Map
Retrieve latest version of the Student from the database
1. getEmployee(string)
1.1. begin( )
1.2. get(string)
1.3. commit(RETAIN_HOLLOW)
Specify the RETAIN_HOLLOW option on the commit(), so that the reference to the retrieved Student can be used outside of the transaction.

PayrollDBManager - Initialize


: Payroll Application Controller
: PayrollDBManager
: Session
PayrollDB : Database
: Transaction
EmployeeMap : Map
1. initialize( )
2. create( )
3. join( )
1.3. create( )
1.4. begin( )
1.5. new( )
1.6. createRoot("EmployeeMap", EmployeeMap)
1.7. commit()
The EmployeeMap is the entry point into the Database. It is a "special" data structure. Any changes to this data structure that occur within the context of a transaction will be applied to the associated ObjectStore Payroll Database.
Pass in the name of the Map, as well as the Map itself (the Map is an Object).
Create the EmployeeMap that will serve as the database root.

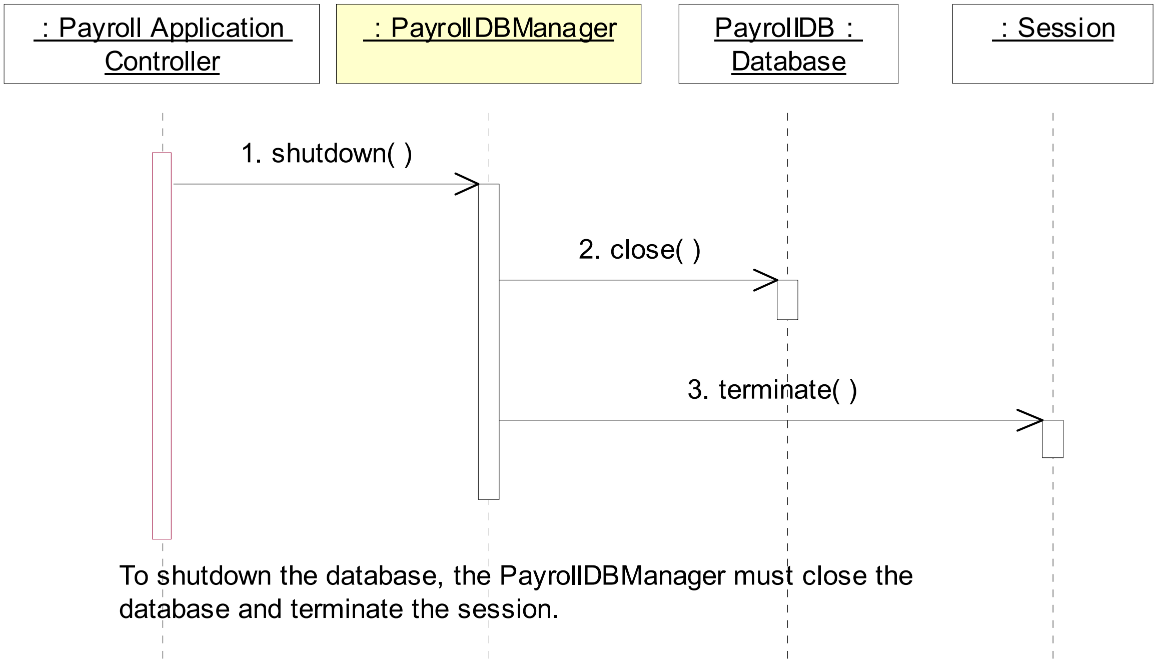
Initialization must occur before any persistent class can be accessed.

Once the session has been created and joined, the PayrollDBManager must open and create the new database.

To create the database, the PayrollDBManager creates a new transaction and creates the "root" of the database with the "createRoot()" operation. In our example, the root will be the EmployeeMap data structure. It will contain instances of the Employee class and all “reachable” classes (including Timecards and Purchase Orders). Remember, the root is the entry point into the Payroll Database. It is a "special" data structure. Any changes to this data structure that occur within the context of a transaction will be applied to the associated Payroll ObjectStore Database.

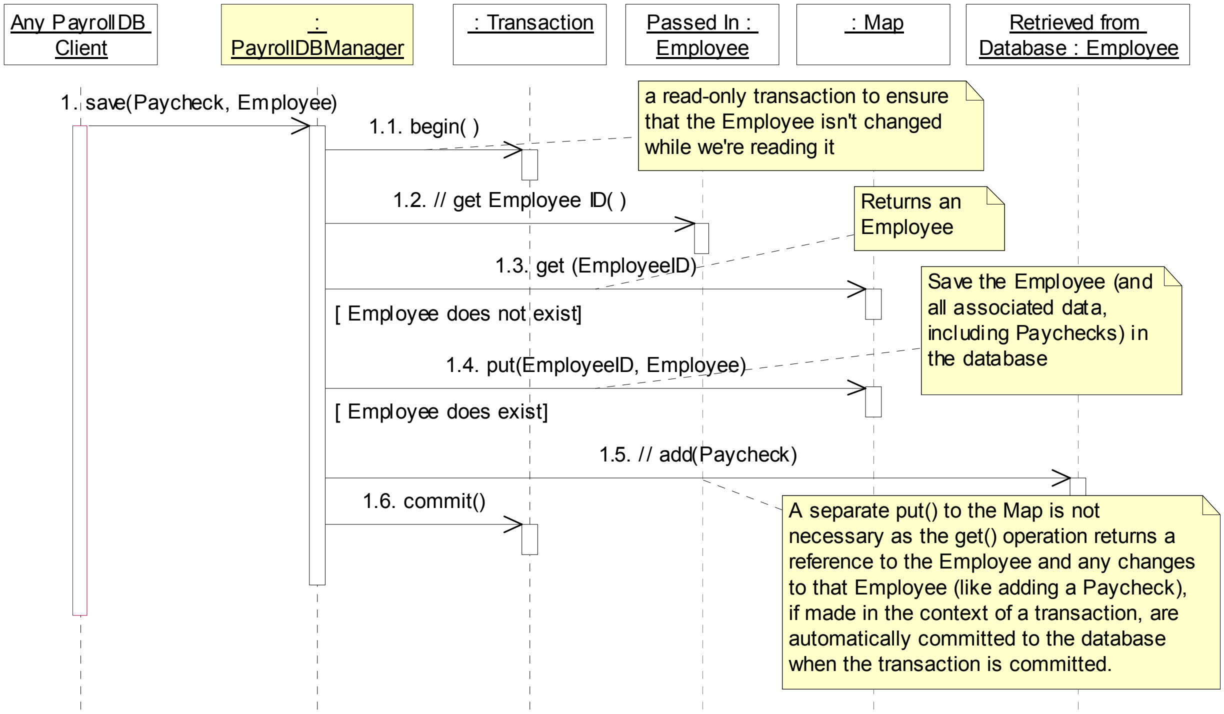
Once the root has been created, the transaction is committed

PayrollDBManager - Shutdown



To shutdown the database, the PayrollDBManager must close the database and terminate the session.

PayrollDBManager - Save Paycheck



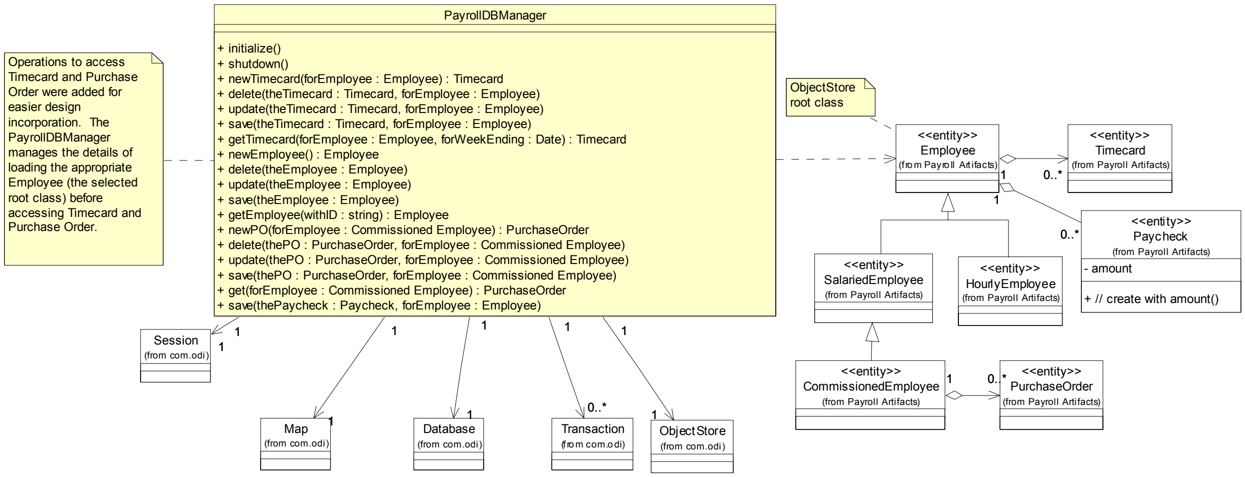
a read-only transaction to ensure that the Employee isn't changed while we're reading it

Returns an Employee

Save the Employee (and all associated data, including Paychecks) in the database

A separate put() to the Map is not necessary as the get() operation returns a reference to the Employee and any changes to that Employee (like adding a Paycheck), if made in the context of a transaction, are automatically committed to the database when the transaction is committed.

Main


Operations to access Timecard and Purchase Order were added for easier design incorporation. The PayrollDBManager manages the details of loading the appropriate Employee (the selected root class) before accessing Timecard and Purchase Order.
PayrollDBManager
+ initialize()
+ shutdown()
+ newTimecard(forEmployee : Employee) : Timecard
+ delete(theTimecard : Timecard, forEmployee : Employee)
+ update(theTimecard : Timecard, forEmployee : Employee)
+ save(theTimecard : Timecard, forEmployee : Employee)
+ getTimecard(forEmployee : Employee, forWeekEnding : Date) : Timecard
+ newEmployee() : Employee
+ delete(theEmployee : Employee)
+ update(theEmployee : Employee)
+ save(theEmployee : Employee)
+ getEmployee(withID : string) : Employee
+ newPO(forEmployee : Commissioned Employee) : PurchaseOrder
+ delete(thePO : PurchaseOrder, forEmployee : Commissioned Employee)
+ update(thePO : PurchaseOrder, forEmployee : Commissioned Employee)
+ save(thePO : PurchaseOrder, forEmployee : Commissioned Employee)
+ get(forEmployee : Commissioned Employee) : PurchaseOrder
+ save(thePaycheck : Paycheck, forEmployee : Employee)
ObjectStore root class
<<entity>> Employee (from Payroll Artifacts)
<<entity>> Timecard (from Payroll Artifacts)
<<entity>> Paycheck (from Payroll Artifacts)
- amount
+ // create with amount()
<<entity>> SalariedEmployee (from Payroll Artifacts)
<<entity>> HourlyEmployee (from Payroll Artifacts)
<<entity>> CommissionedEmployee (from Payroll Artifacts)
<<entity>> PurchaseOrder (from Payroll Artifacts)
Session (from com.odi)
Map (from com.odi)
Database (from com.odi)
Transaction (from com.odi)
ObjectStore (from com.odi)
1
0..*

Session: The class that represents a database session. A session must be created in order to access the database and any persistent data.

A session is the context in which PSE/PSE Pro databases are created or opened, and transactions can be executed. Only one transaction at a time can exist in a session.

Map: A persistent map container classes that stores key/value pairs.

Database: The Database class represents an ObjectStore database.

Before you begin creating persistent objects, you must create a database to hold the objects. In subsequent processes, you open the database to allow the process to read or modify the objects. To create a database, you call the static create() method on the Database class and specify the database name and an access mode.

Transaction: An ObjectStore transaction. Manages a logical unit of work. All persistent objects must be accessed within a transaction.

ObjectStore: Defines system-level operations that are not specific to any database.

PayrollDBManager: For the Payroll System, there is one ObjectStore database, the Payroll Database, that contains employee, timecard, and purchase order information for the company. There is one PayrollDBManager (i.e., this class is a singleton).

This class is responsible for providing access to the persistent objects in the Payroll Database. It provides a single entry point into the Payroll Database. It contains operations to access entities in the database.

The PayrollDBManager class contains most of the database-specific code, such as starting and ending transactions. There are noPayrollDBManager objects stored in the database, which means that the PayrollDBManager class is not required to be persistence-capable.

The PayrollDBManager class has a static members that keep track of the database that is open. It also has a number of static methods, each of which executes a transaction in the ObjectStore database.

HourlyEmployee: An employee that is paid by the hour.

SalariedEmployee: An employee that receives a salary.

CommissionedEmployee: An employee that receives a commission.

PurchaseOrder: A record of a sale made by an employee.

Timecard: The timecard contains information regarding the hours worked by an employee for a given time period.

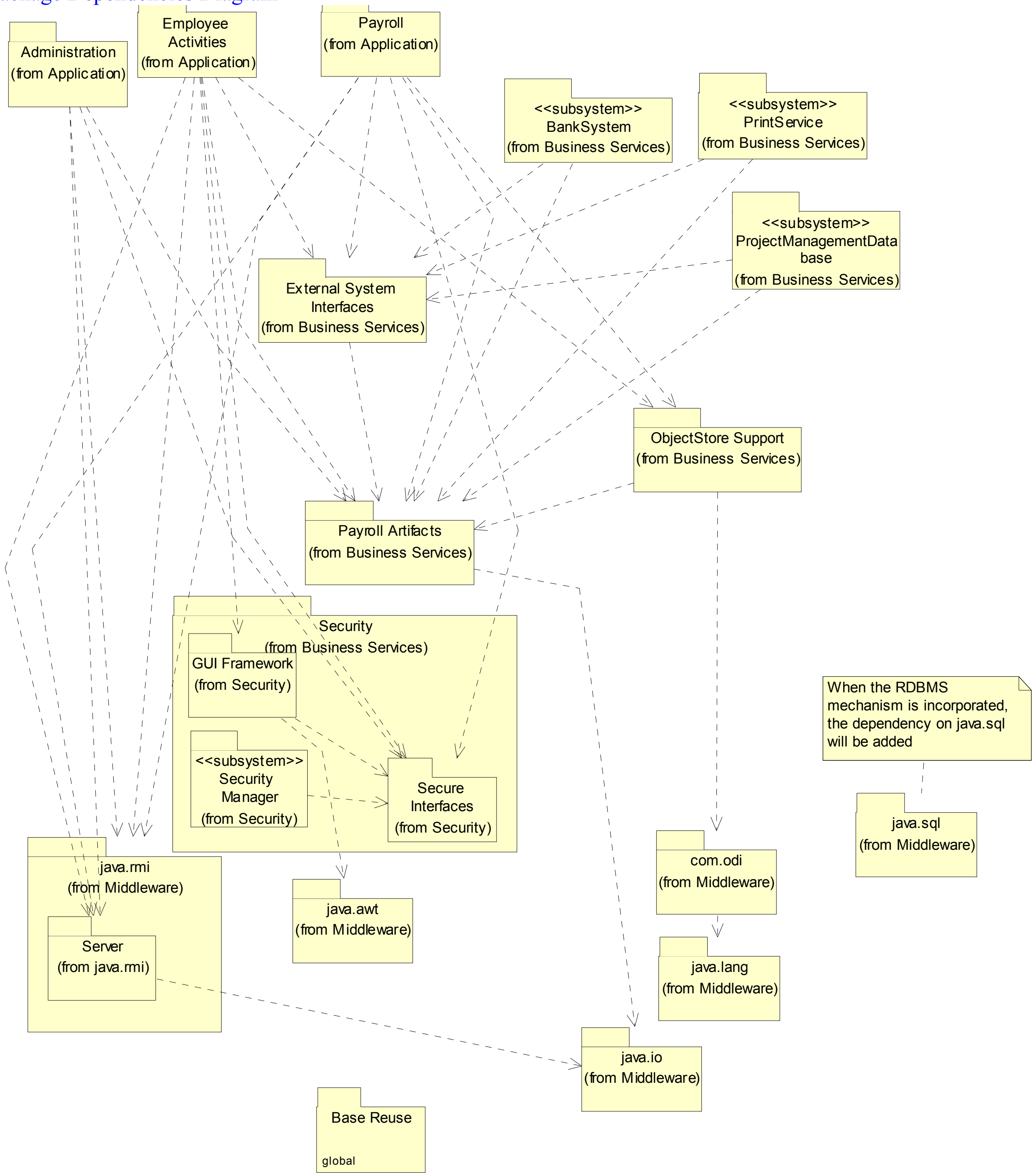
Employee: A person that works for the company.

Paycheck: A record of how much an employee was paid for a given pay period.

2. Exercise: Use-Case Design, Part 2

2.1 Packages and Their Dependencies

Package Dependencies Diagram



2.1.1 *Package Descriptions*

Administration : Contains the design elements that support the Payroll Administrator's applications.

BankSystem Subsystem: Encapsulates the details involved in communicating with external bank systems.

Base Reuse : Basic reusable design elements.

com.odi : The com.odi package contains the design elements that support the OODBMS persistency mechanism. The name of the package in the model reflects the naming convention for 3rd party Java software. The convention is to use the reverse of the domain name, so if Rational had a Java package called "util" they'd call it" com.rational.util". This com.odi has nothing to do with Microsoft COM/DCOM, they are totally separate. There is nothing COM/DCOM related when using CORBA, RMI, or ObjectStore.

Employee Activities : Contains the design elements that support the Employee's applications.

External System Interfaces : Contains the interfaces that support access to external systems. This is so that the external system interface classes can be version controlled independently from the subsystems that realize them.

GUI Framework : This package comprises a whole framework for user interface management.

It has a ViewHandler that manages the opening and closing of windows, plus window-to-window communication so that windows do not need to depend directly upon each other.

This framework is security-aware, it has a login window that will create a server-resident user context object. The ViewHandler class manages a handle to the user context object.

The ViewHandler also starts up the controller classes for each use case manager.
java.awt : The java.awt package contains the basic GUI design elements for java.

java.io :

java.lang : The package contains some basic java design elements.

java.rmi : The java.rmi package contains the classes that implement the RMI distribution mechanism. This package is commercially available with most standard JAVA IDEs.

java.sql : The package that contains the design elements that support RDBMS persistency.

ObjectStore Support : Contains the business-specific design elements that support the OODBMS persistency mechanism. This includes the DBManager. The DBManager class must contain operations for every OODBMS persistent class.

Payroll : Contains the design elements that support the execution of the payroll processing.

Payroll Artifacts : Contains the core payroll abstractions.

PrintService Subsystem: Provides utilities to produce hard-copy.

ProjectManagementDatabase Subsystem: Encapsulates the interface to the legacy database containing information regarding projects and charge numbers.

Secure Interfaces : Contains the interfaces that provide clients access to security services.

Security : Contains design elements that implement the security mechanism.

Security Manager Subsystem: Provides the implementation for the core security services.

Server :