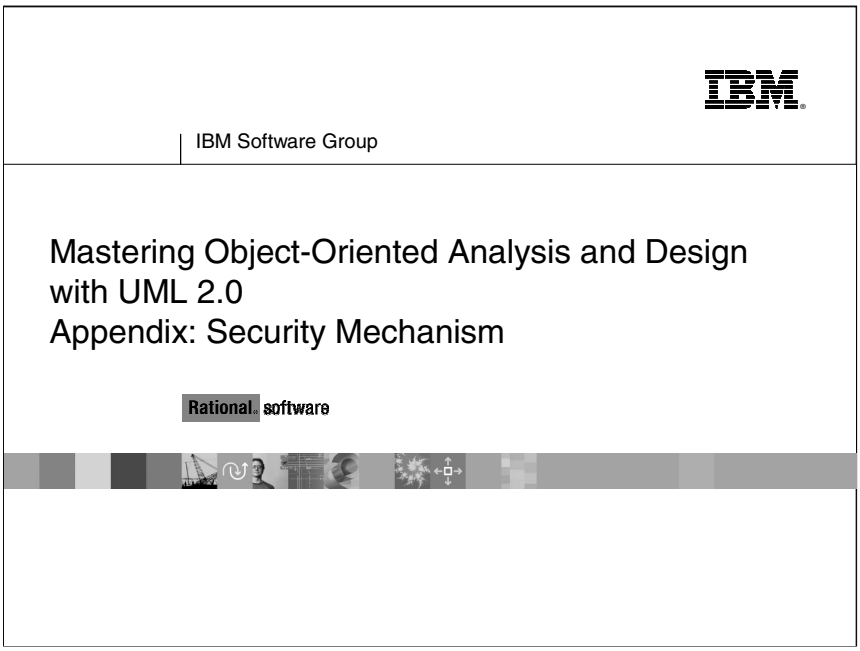
Instructor Notes:



Instructor Notes:

Note: If your students are interested in the Security mechanism, these slides should be inserted after presenting the reuse section in the Identify Design Mechanisms module.

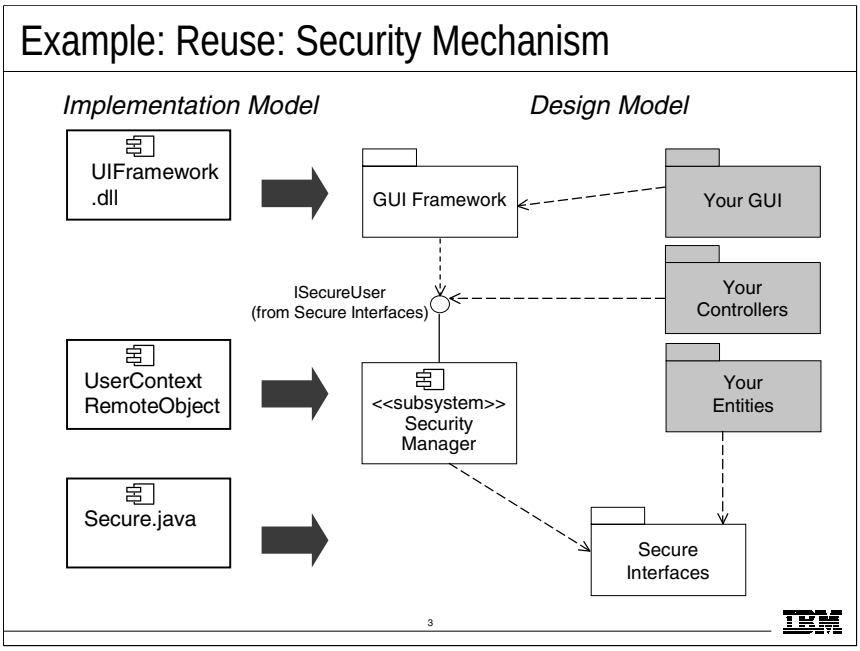
Identify Design Mechanisms Slides

The following slides can be inserted during the Identify Design Mechanisms module

2

Instructor Notes:

This slide demonstrates Reuse by applying reverse engineering. Reverse-engineering existing components can give you a bag of tricks to pull from. Components were defined in the Introduction to Object Orientation module. A subsystem is the design representation of a component. They both encapsulate a set of replaceable behaviors behind one or more interfaces.
In this course, we will concentrate on the impact on the Design Model.The Implementation Model is out of the scope of the OOAD course.
This slide, becomes more clear once the remaining security slides have been presented. "Secure" entities need to know how to implement the ISecureData interface; therefore they depend on SecureInterfaces.
Controllers will work with ISecureUser (residing down in SecureInterfaces) to set the security attributes of new objects and check the security of objects that are being manipulated; therefore, their package is shown as depending on ISecureUser.
The GUI framework includes the LoginForm and MainApplicationForm that depend on the ISecureUser interface, so must depend on the SecureInterfaces package.



The above example demonstrates the results of reverse engineering existing components into the Implementation and the Design Model. There are three components that we can reuse from a previous project. One is a GUI framework. The other two support security on the server. The components being reused are shown down the left-hand side of the example (Implementation Model). The representation of these components in the Design Model are shown down the middle, with their associations shown using the block arrows.

The GUI Framework provides a standard set of user interface classes. This GUI Framework is "security-aware" (note the dependency on the Secure User interface). For security, the UI provides a login screen that works with a server-side business object to create a user context. This server object will remain available to all server-based controllers for the duration of the session.

The Security Manager includes the classes that implement the security behavior (for example create the secure user context).

Secure Interfaces supplies the security interfaces. Entities that are secure will realize an interface and provide a small set of behavior.

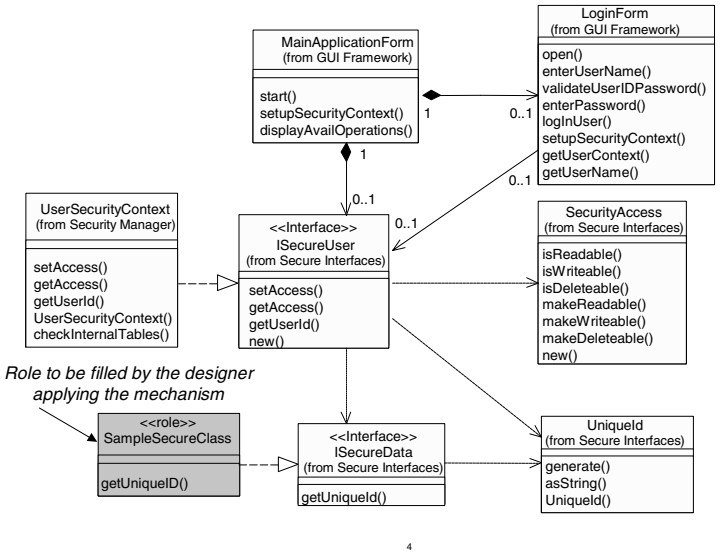
The three example packages ("Your GUI", Your Controllers", and "Your Entities") represent packages in the system being developed that might depend upon the security packages.

Instructor Notes:

Do not spend too much time on this slide. Just walk through the basics (as described in the Student Notes). Stay at a fairly high level. Some of this may be more clear to the students when they see the interaction diagrams provided on the next few slides.

The classes to be replaced by concrete classes by the designer applying the mechanism are shown in blue (but this does not show up in the black and white manuals).
There is no need for a separate ISecureData.SetUniqueID operation, as the ISecureData.GetUniqueID creates a unique id if one does not exist.
The reverse engineered security mechanism is actually a lightweight framework. You can make a big production out of the power of interfaces in supporting a framework view wherein a set of security classes written for a previous project end up working with your classes to provide this service. "So now our controllers are calling this security class asking if the current user has read access to the timecard, and they are passing the timecard in as a parameter. As long as the timecard implements the ISecureData interface, the security class is perfectly happy."
Note: Composition is shown for the LoginForm because that was reverse-engineered. Whether the other aggregation relationships should be composition or not will be addressed in the Class Design module.

Example: Reuse: Security Mechanism (continued)



This shows how the security-related classes fit together. Again, the <<role>> stereotype is used to which classes are roles that should be filled by the designer as they apply the mechanism.

For each application session, there must exist an object whose class realizes the ISecureUser interface (in our example, the UserSecurityContext class). This object manages information about the current user's access to secure data without directly depending on the classes (ISecureUser classes depend on the ISecureData interface, not on the actual secure data classes). The security system, via the operations exposed in ISecureUser, allows clients to set the access of the objects when they are created and check the access when they are manipulated at some later date. There is one SecurityAccess instance per user login/session per secure object.

The MainApplicationForm keeps around a reference to a ISecureUser object until the form closes (this is represented by the composition relationship). The reference to the ISecureUser object actually comes back from the LoginForm, which creates the ISecureUser object upon successful logon (this is represented by the dependency relationship).

Classes can be made "security-aware" by realizing the ISecureData interface and defining an attribute to hold a UniqueId. The UniqueId class makes sure that all users and all pieces of data from all different applications get their own unique ids. All classes that have been mapped to the Security analysis mechanism should realize the ISecureData interface.

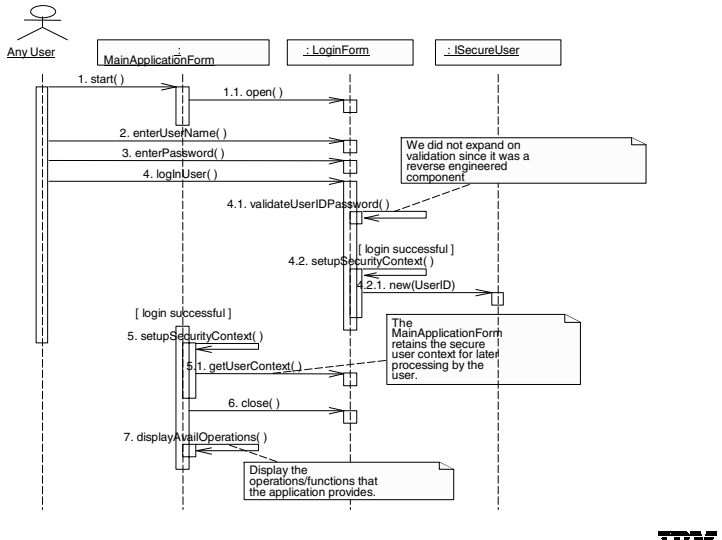
Instructor Notes:

Message 5 may seem like It comes out of nowhere because the SomeUser actor is independently active while the windows go through their flow of control. If you disregard steps 2, 3, 4, then nothing looks strange; it is obvious that step 5 is just the thing the MainApplicationForm does after step 1.1.

Message 4.2.1 is an oversimplification. You can't create an ISecureUser object just be sending a new() message. The creation of a new instance involves some sort of a factory that creates and instance of the class that realizes the interface. In the case of the ISecureUser interface, such an object exists on the server, so the distribution mechanism would need to be involved. Distribution is discussed in more detail in the Describe Distribution module.

LoginForm retains control until it reaches some concept of completion of the task which does not yet remove the window from memory.

Example: Security Mechanism: Secure User Set-Up at Login



As mentioned earlier, for each application session, there must exist an object whose class realizes the ISecureUser interface. This object manages information about the current user's access to secure data.

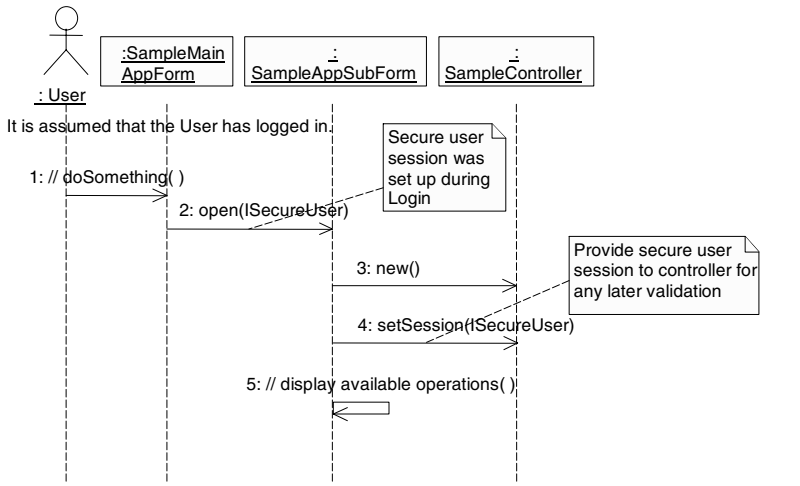
The LoginForm will—upon successful login—create an instance of ISecureUser (physically an instance of UserSecurityContext). The ISecureUser is passed to each form that is opened. The form in turn passes it into each controller that starts up. This is propagated to the remote controllers that use the ISecureUser to set and check privileges.

The setting up of a user context is described in more detail for the Course Registration System example in the Use-Case Design module where the Login use-case realization is discussed.

Instructor Notes:

The items in blue were added/utilized as part of incorporating the security mechanism (but this does not show up in the black and white manuals).

Example: Security Mechanism: Secure User Propagation



6

All entities that need to perform access checks must have access to the current user's session (i.e., have access to ISecureUser).

The above example shows how a sub-form and the associated controller are provided access to the secure user session. The secure user session is established during login and maintained by the main application form. The secure user session is then propagated from the main application form to any sub-forms and supporting controllers (note the use of ISecureUser as a parameter to the open command, as well as the new setSession() operation).

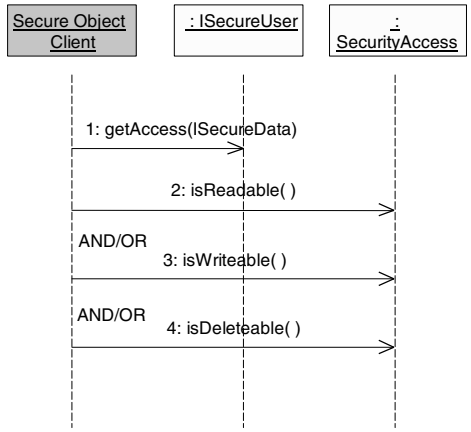
Instructor Notes:

When a secure object is created, access for it must be set for the current user. Usually, this is full access for the current user, read-only for others.

Only a retrieve example was provided for scoping reasons.

The classes to be replaced by concrete classes by the designer applying the mechanism are shown in blue (but this does not show up in the black and white manuals).

Example: Security Mechanism: Secure Object Access



7

All classes that have been mapped to the Security analysis mechanism should realize the ISecureData interface. All classes that must verify the access of some secure data should do so via the ISecureUser interface. There will be one object whose class realizes the ISecureUser interface per Student session.

When retrieving secure data, the client must verify the current user's ability to access the secure data, using the security access information for the current user.

Specifically, as described in the above example, after retrieving an object that realizes the ISecureData interface, the Secure Object Client must retrieve the security access information for the secure object for the current user and compare them to make sure that the current user can view/edit/delete the object.

Instructor Notes:

Note: On the presented slide, the italicized text is also blue, but this does not show up in the back-and-white manuals.

Remind the students that the actual design of this mechanism is deferred until detailed design. At this point, we just want to make sure that the architecture can handle the incorporation of the security mechanism (i.e., that the infrastructure is in place).

Incorporating the Security Mechanism: Steps

- Provide access to the class libraries needed to implement the Security mechanism
  - *New Security package*
- Establish main application forms, with associated login forms
  - *Package containing forms dependent on Security GUI Framework package*
  - *Create main application forms* — *Deferred*
- Have all secure classes realize the ISecureData interface
  - *Package containing core data types dependent on Security Secure Interfaces package*
  - *Add realization relationships* — *Deferred*

8

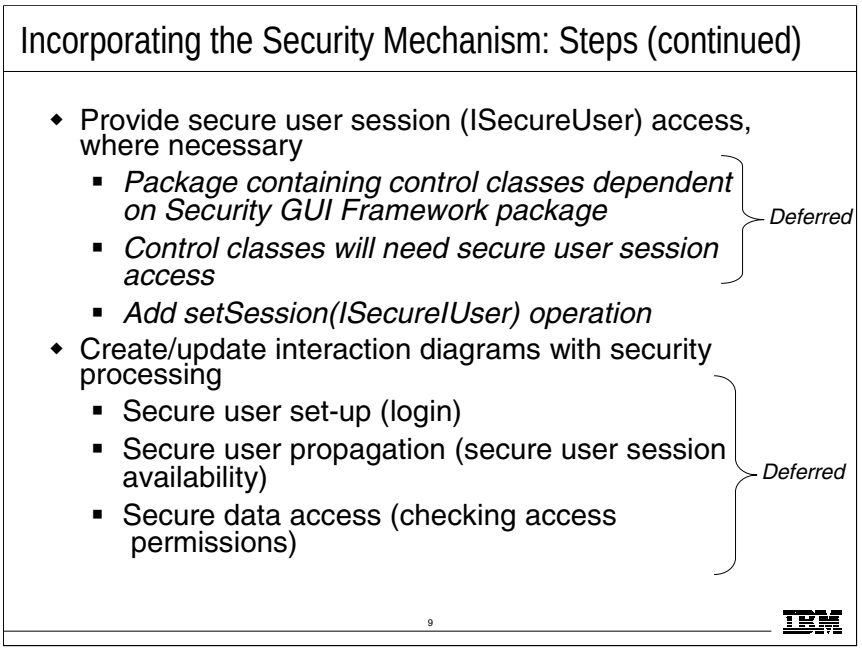
The above is a summary of the steps that can be used to implement the security mechanism described in this module. The italicized text describes the architectural decisions made with regards to JDBC for our Course Registration example:

- A new Security package will be created to contain the classes that implement the security mechanism.
- For each application that is to be developed, a main application form needs to be defined. In some cases, such a main form may have already been identified. If not, one needs to be defined now. In any case, the main application form must inherit from the MainApplicationForm provided in the GUI Frameworks package. You may need to create subforms to support the individual functions/operations provided by the application. If any existing Login forms were identified in analysis, these need to be replaced with the LoginForm provided in the GUI Frameworks package. This LoginForm will be associated and "driven by" the MainApplicationForm defined above via mechanisms provided in the GUI Frameworks package. In Identify Design Mechanisms, we make sure that the necessary package relationships exist, but the creation of the main application forms has been deferred until detailed design (for example, Use-Case and Subsystem Design).
- All classes that need to be secure must realize the ISecureData interface in the Secure Interfaces package.
  The necessary package relationships will be added here, but the introduction of all of the individual realization relationships (for all data that is to be secure) will be deferred until detailed design (for example, Use-Case and Subsystem Design).

Instructor Notes:

In many cases, the interaction diagrams developed in analysis can be "matured" to include the new classes and interactions required to support security. However, try to maintain the readability of the diagrams. Separate diagrams may need to be created.

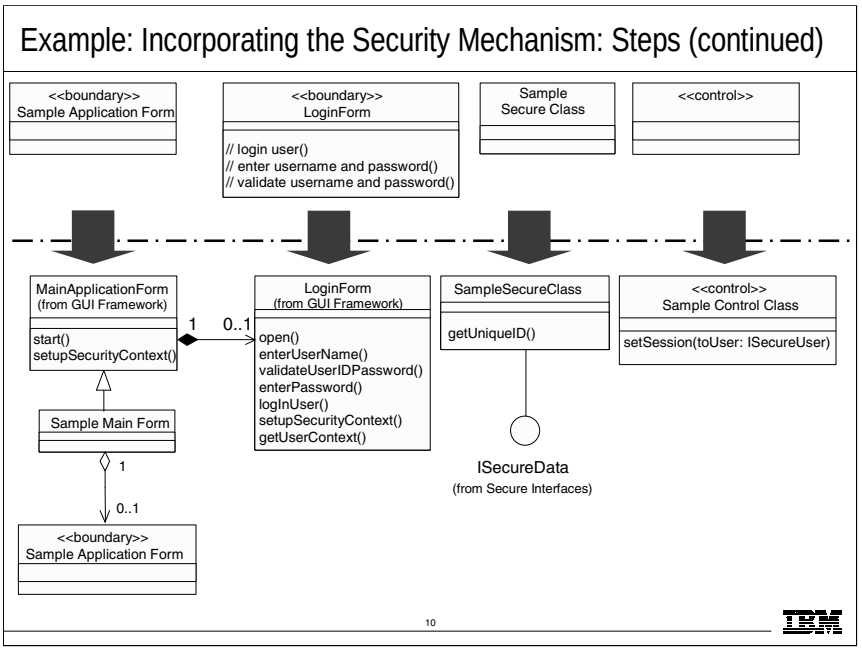
Note: On the presented slide, the italicized text is also blue, but this does not show up in the back-and-white manuals.



- All entities that need to perform access checks must have access to the current user's session (i.e., have access to ISecureUser). This means that you may need to add ISecureUser as a parameter to some operations. For the Course Registration System, the control classes will need access to the secure user session, so a setSession(ISecureUser) operation needs to be added to each control class.
  The modification of the individual control classes has been deferred until detailed design (for example Use-Case and Subsystem Design).
- In order to make sure that everything has been accounted for in the system, interaction diagrams should be defined which model the added security functionality. Specifically, secure user set-up (including Login, as well as making sure that the secure user session is made available to all entities that will need to perform security checks) and secure data access (includes checking a user's permission before providing access to secure data).
  The development of these interaction diagrams has been deferred until detailed design (for example Use-Case and Subsystem Design).

The actual incorporation of the mechanism is deferred until detailed design (for example Use-Case Design and Subsystem Design). At this point, the architect provides guidance to the designers and makes sure that the architecture has the necessary infrastructure to support the mechanism (i.e., has the necessary packages and package relationships).

Instructor Notes:

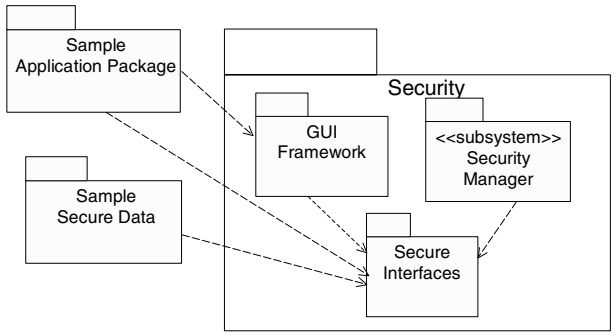


The above is an example of some of the changes described on the previous slide that must be made to the Course Registration Model to incorporate the Security mechanism:

- The LoginForm from the security GUI framework will replace the LoginForm identified in Use-Case Analysis
- The MainApplicationForm from the security GUI framework will be used as the basis for new application main forms that will serve as the context for the forms identified in Use-Case Analysis. Application Main forms will be created for each of the major system actors. This is because the functionality in each of these applications is disjoint. The actual definition of the main application forms has been deferred until detailed design (for example Use-Case and Subsystem Design).
- Any classes that must be secure will need to realize the ISecureData interface. This involves implementing the getUniqueID() operation. The actual definition of the individual realization relationships (one for each class that is to be secure) has been deferred until detailed design (for example Use-Case and Subsystem Design).
- The control classes will need access to the secure user session, so a setSession(ISecureUser) operation needs to be added to each control class. The modification of the individual control classes has been deferred until detailed design (for example Use-Case and Subsystem Design).

Instructor Notes:

Example: Incorporating the Security Mechanism: Steps (continued)



11

The above demonstrates the package dependencies needed to support the changes described on the previous slides to incorporate the security mechanism:

The design elements that support the security mechanism have been placed in a single package, Security.

The package(s) that contains the application main forms (in the above example, the Sample Application package) will have a dependency on the Security GUI Framework package.

The package(s) that contains the classes that need to be secure (in the above example, the Sample Secure Data package) will have a dependency on the Security Secure Interfaces package.

The package(s) that contains the classes that will need to access the secure classes (in the above example, the Sample Application package) will have a dependency on the Security Secure Interfaces package.

Specific application and core abstraction packages are defined in the next section.

Instructor Notes:

Note: If your students are interested in the Security mechanism, these slides should be inserted while presenting incorporation of the the security mechanism in the Use-Case Design module.

Use-Case Design Slides

The following slides can be inserted during the Use-Case Design module

12

Instructor Notes:

Note: On the presented slide, the italicized text is also blue, but this does not show up in the back-and-white manuals.

The check marks indicate what steps have been completed.

Remind the students that the actual design of this mechanism is deferred until detailed design. At this point, we just want to make sure that the architecture can handle the incorporation of the security mechanism (i.e., that the infrastructure is in place).

Review: Incorporating the Security Mechanism: Steps

- Provide access to the class libraries needed to implement the Security mechanism
  - √ *New Security package*
- Establish main application forms, with associated login forms
  - √ *Package containing forms dependent on Security GUI Framework package*
  - √ *Create main application forms*
- Have all secure classes realize the ISecureData interface
  - *Package containing core data types dependent on Security Secure Interfaces package*
  - *Add realization relationships*

√ = Done

13

IBM

The above is a summary of the steps first discussed in Identify Design Mechanisms that can be used to implement the security mechanism described in this module. The italicized text describes the architectural decisions made with regards to JDBC for our Course Registration example. The check marks indicate what steps have been completed. Now we will continue incorporate this mechanism.

- In Architecture Design, a new Security package was created to contain the classes that implement the security mechanism.
- For each application that is to be developed, a main application form needs to be defined. The main application form must inherit from the MainApplicationForm provided in the GUI Frameworks package. Subforms may need to be created to support the individual functions/operations provided by the application. If any existing Login forms were identified in analysis, these need to be replaced with the LoginForm provided in the GUI Frameworks package. This LoginForm will be associated and "driven by" the MainApplicationForm defined above via mechanisms provided in the GUI Frameworks package. In Identify Design Mechanisms, we put the infrastructure in place, now the actual main application forms will need to be created.
- All classes that need to be secure must realize the ISecureData interface in the Secure Interfaces package. In Identify Design Mechanisms, access to the interface was provided, now we must add the actual realizes relationships.

Instructor Notes:

In many cases, the interaction diagrams developed in analysis can be "matured" to include the new classes and interactions required to support security. However, try to maintain the readability of the diagrams. Separate diagrams may need to be created.

Note: On the presented slide, the italicized text is also blue, but this does not show up in the back-and-white manuals.

Review: Incorporating the Security Mechanism: Steps (continued)

- Provide secure user session (ISecureUser) access, where necessary
  - √ *Package containing control classes dependent on Security GUI Framework package*
  - *Control classes will need secure user session access*
  - *Add setSession(ISecureIUser) operation*
- Create/update interaction diagrams with security processing
  - Secure user set-up (login)
  - Secure user propagation (secure user session availability)
  - Secure data access (checking access permissions)

√ = Done

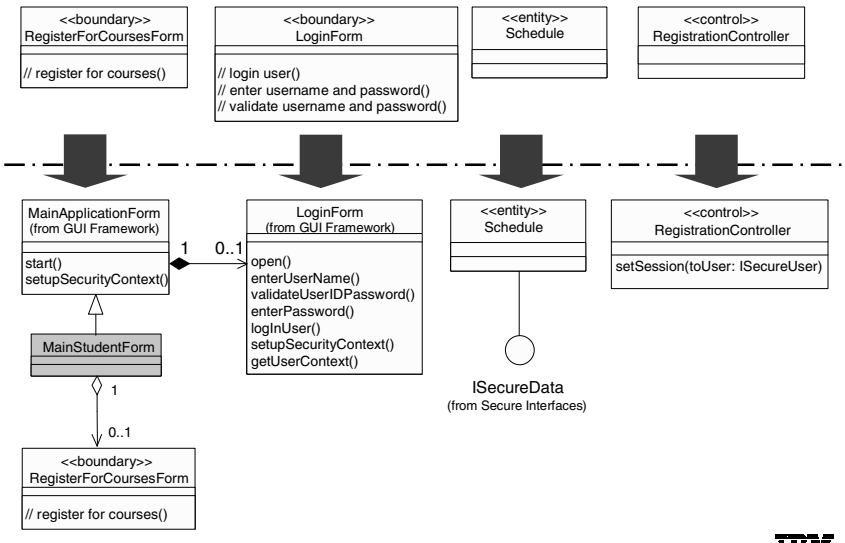
14

- All entities that need to perform access checks must have access to the current user's session (i.e., have access to ISecureUser). This means that you may need to add ISecureUser as a parameter to some operations.
  For the Course Registration System, the control classes will need access to the secure user session, so a setSession(ISecureUser) operation needs to be added to each control class.
  In Identify Design Mechanisms, the package and package dependencies were put into place. Now, we must update the control classes so that they have access to the secure user session.
- In order to make sure that everything has been accounted for in the system, interaction diagrams should be defined which model the added security functionality. Specifically, secure user set-up (including Login, as well as making sure that the secure user session is made available to all entities that will need to perform security checks) and secure data access (includes checking a user's permission before providing access to secure data).
  These interaction diagrams will now be developed.

Instructor Notes:

The items in blue were added/utilized as part of incorporating the security mechanism (but this does not show up in the black and white manuals).

Example: Incorporating the Security Mechanism



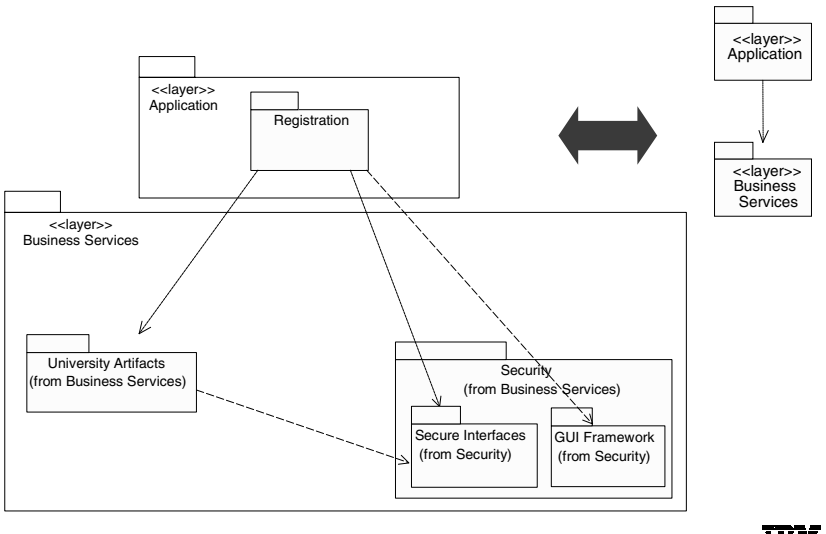
The above is an example of some of the changes described on the previous slide that were made to the Course Registration Model to incorporate the Security mechanism:

- The LoginForm from the security GUI framework will replace the LoginForm identified in Use-Case Analysis
- The MainApplicationForm from the security GUI framework will be used as the basis for the new MainStudentForm. TheMainStudentForm will inherit from the MainApplicationForm). Thus, the MainStudentForm will be "security-aware". The original RegisterForCoursesForm will now be a subform of the new MainStudentForm.
- The Schedule class must be secure, so it will realize the ISecureData interface.
- The RegistrationController control classes will need access to the secure user session, so a setSession(ISecureUser) operation will be added.

Instructor Notes:

The items in blue were added/utilized as part of incorporating the security mechanism (but this does not show up in the black and white manuals).
It is also possible to have security mechanisms in both the Application layer and the Business Services layer, with the Application layer security mechanisms dependent on the Business Services layer security mechanisms. The Application layer mechanisms would provide security that was aware of application-specific access rules (e.g. a person requires security level X to perform a specific transaction) while the Business Services layer mechanism would simply provide a generic way to define transactions and security access restrictions and to enforce those restrictions. The Business Services layer stuff is highly re-usable across applications, but there may be some application-specific aspects to handling security. Thus, there can actually be mechanisms at both levels. This information is provided FYI only. This is not the approach we took in this course.

Example: Incorporating the Security Mechanism (continued)



The above demonstrates the package dependencies needed to support the changes described on the previous slides to incorporate the security mechanism.

The packages and associated relationships were defined in Identify Design Mechanisms. This slide is included here for review purposes.

The design elements that support the security mechanism will be placed in a single package, Security.

The Application package(s) that contains the application main forms will have a dependency on the Security GUI Framework package (for the main and form definitions). For the course registration application, that package is the Application package.

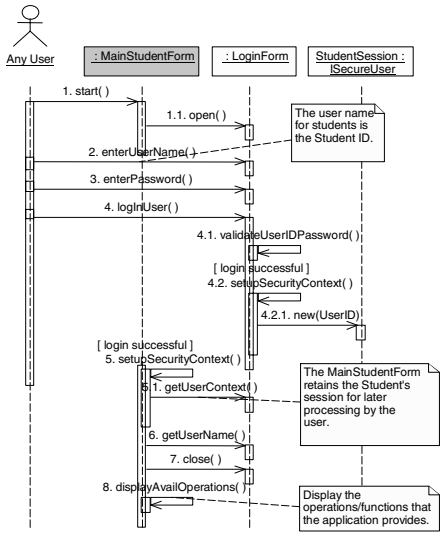
The Application package(s) that contains the control classes will have a dependency on the Security Secure Interfaces package (for the ISecureUser definition). For the course registration application, that package is the Application package.

The University Artifacts package contains the classes that need to be secure, so it will have a dependency on the Security Secure Interfaces package.

Instructor Notes:

The items in blue were added/utilized as part of incorporating the security mechanism (but this does not show up in the black and white manuals).

Example: Incorporating the Security Mechanism: Secure User Set-Up



In order to use the security features of the system, the secure user session must be established. This occurs during login.

The security mechanism provides a main application form and a login form, which will replace the original login form.

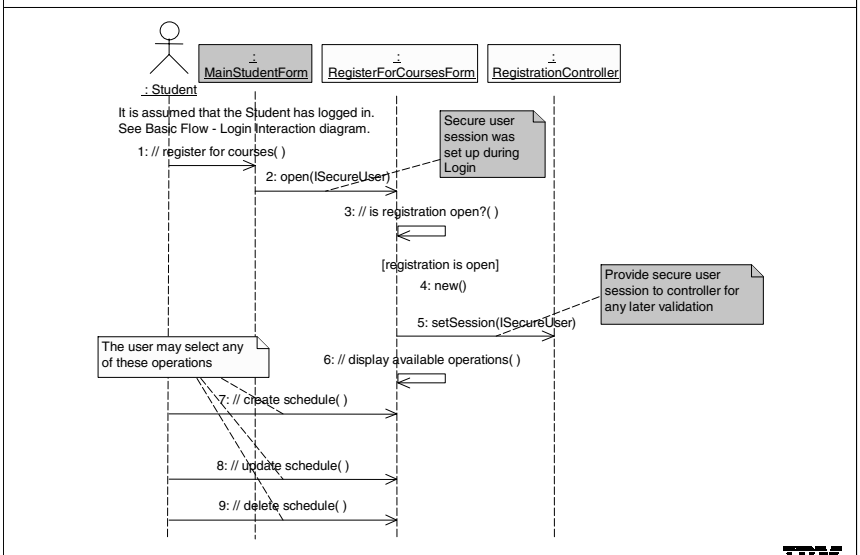
The above example is the design use-case realization for the Login use case.

It is almost completely re-written from Use-Case Analysis.

Instructor Notes:

Note: On the presented slide, the items that were added/utilized as part of incorporating the mechanism are shown in blue, but this does not show up in the back-and-white manuals.

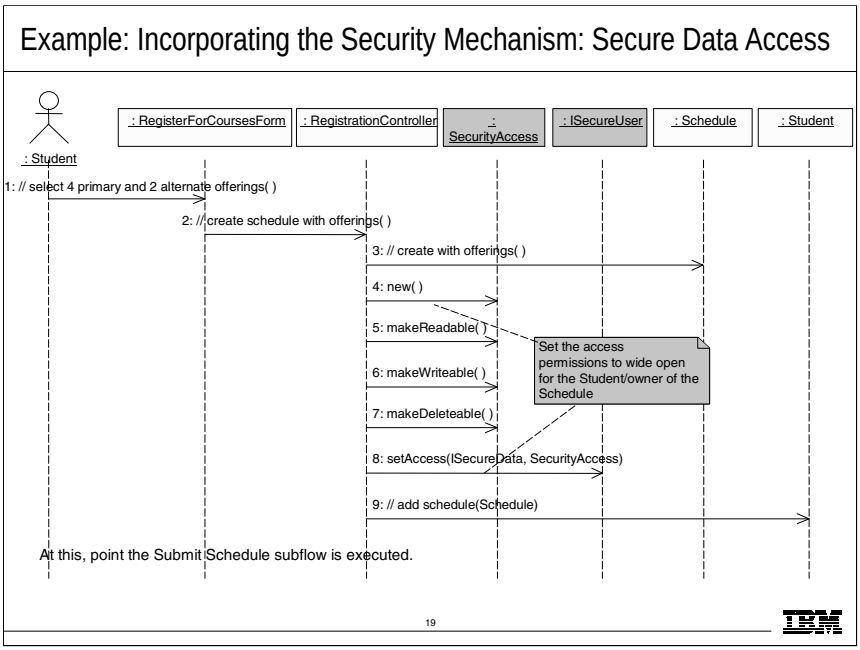
Example: Incorporating the Security Mechanism: Secure User Propagation



The above is a fragment of the Register for Courses use-case realization interaction diagram. It demonstrates how the created secure user session is provided to the RegisterForCoursesForm and the RegistrationController. This is necessary because in our system, the controllers will be performing the security checks.

Instructor Notes:

Note: On the presented slide, the items that were added/utilized as part of incorporating the mechanism are shown in blue, but this does not show up in the back-and-white manuals.



The above is a fragment of the sequence diagram from the Register for Courses use-case realization. It demonstrates the interactions that need to be added when the application creates secure data (specifically, a Student's Schedule). After creating the Student's Schedule, the RegistrationController sets the security access information for the Schedule for the current Student. Since the Student is the "owner" of the schedule, he/she is given full access.

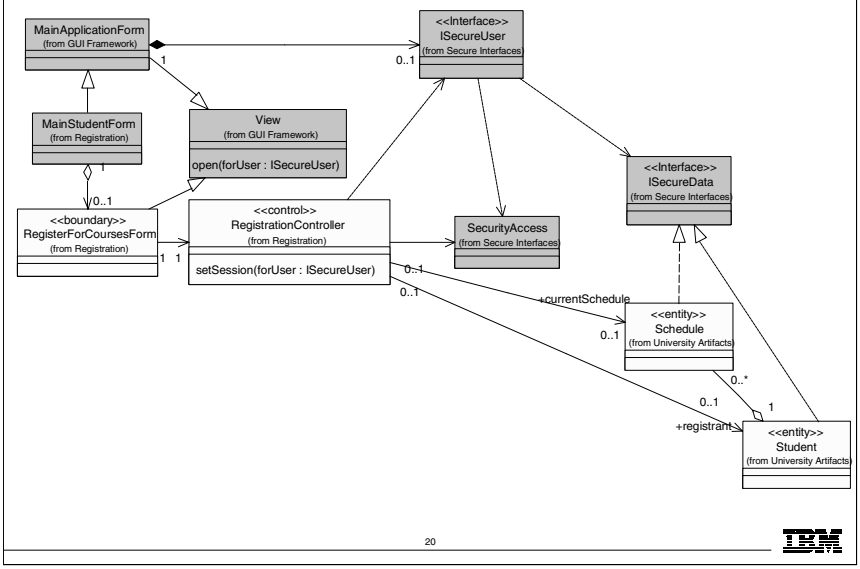
The access permissions are maintained in the SecureUser that is managed by the MainApplicationForm (the MainStudentForm, in the case of Register for Courses). See the Login sequence diagram on an earlier slide. The RegisterForCoursesForm is provided with the SecureUser when it is opened by the MainStudentForm.

As stated earlier, the Schedule class was mapped to the Security analysis mechanism, so it realizes the SecureData interface.

Instructor Notes:

Note: On the presented slide, the items that were added/utilized as part of incorporating the mechanism are shown in blue, but this does not show up in the back-and-white manuals.

Example: Incorporating the Security Mechanism: VOPC



The above is a subset of the View of Participating Classes (VOPC) for the Register for Courses use-case realization. It contains the classes for the instances in the previous interaction diagrams (i.e., the classes affected by the incorporation of the security mechanism).

Notice the addition of SecureData, SecurityAccess and ISecureUser and their relationships with the application classes.

The incorporation of the security processing has been localized to the controllers, in addition to the secure classes realizing the secure interface.

Any forms "get security for free" if they inherit from the View class provided with the GUI Frameworks.

Note: the display of most of the operations and attributes have been suppressed for clarity of the diagram. However, a few selected operations that demonstrate the security mechanism have been shown.