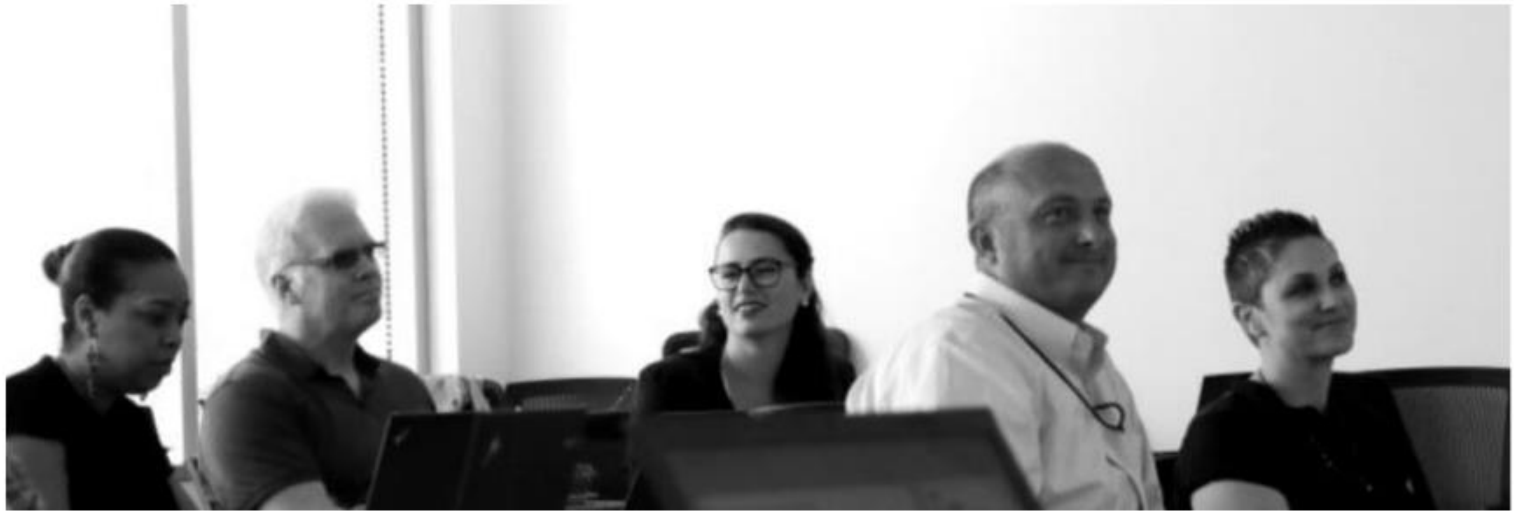
VALIDATION CLASSES

Source : Guidewire Educational Platform

Validation Classes



Lesson objectives

By the end of this lesson, you will be able to:

- Describe class-based validation
- Identify which objects should use validation classes
- Explain how validation levels are used
- Invoke class-based validation from different locations in PolicyCenter
- Write validation checks needed to validate objects on a policy

Class-based validation overview

Class-based validation

- Is validation used on-demand
- Allows users to:
  - choose at what time to validate a data object
  - decide how to validate data objects
  - write complex validation logic in Gosu classes

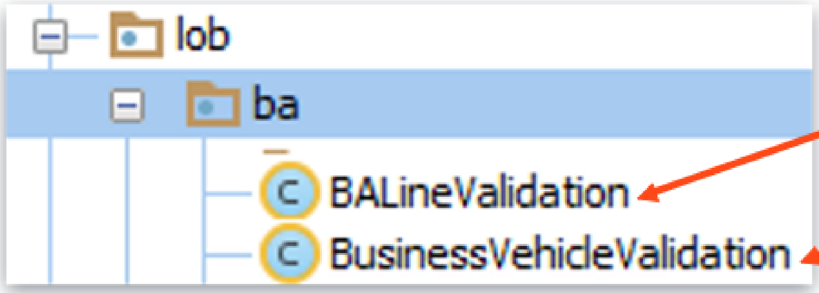
What can be validated using class-based validation?

- Class-based validation may be performed on policy-related objects only such as:
  - PolicyContactRole
  - PolicyLocation
  - PolicyPeriod
  - Line of business entities such as PALine or BALine
  - Policy-specific entities such as PersonalVehicle or BusinessVehicle
- Everything on the policy graph can be validated using class-based validation
  - Entities on a policy graph need not be defined specially or as validatable
  - Validation class can perform any number of checks against any entities on a policy graph

Source Guidewire Educational Platform

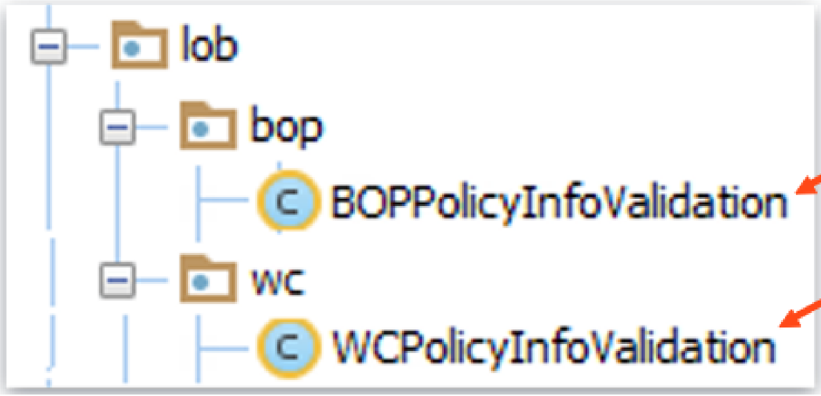
PolicyCenter validation classes

- Validation classes validate an entity on the policy graph



Validate Commerical Auto LOB and Business Vehicle

- Or potentially a wizard step

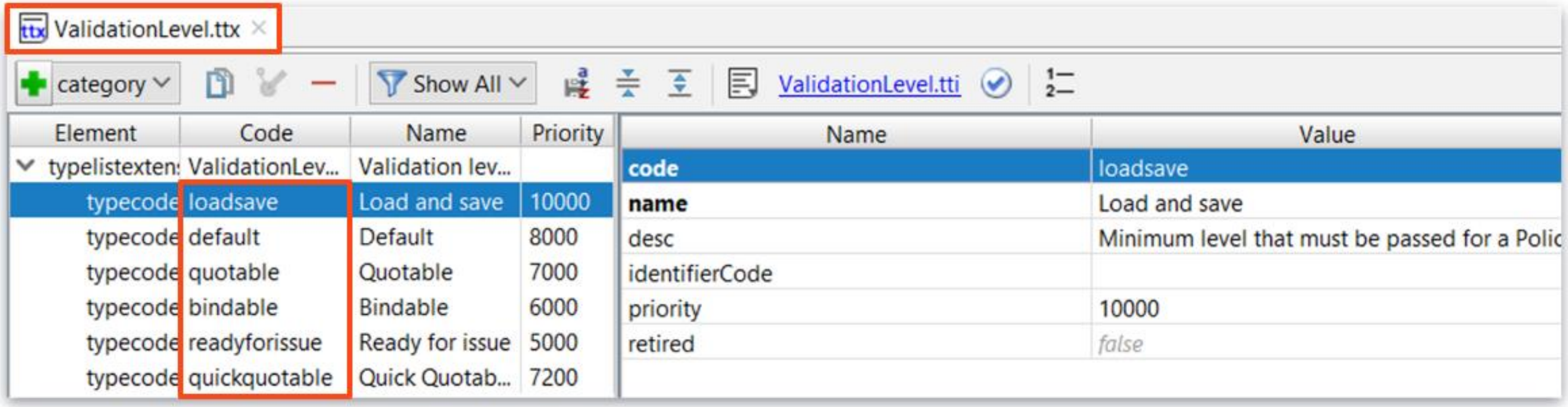


Validate the Policy Info wizard step in a submission

Validation levels

Validation levels

- Defined in ValidationLevel typelist
- New levels can be added

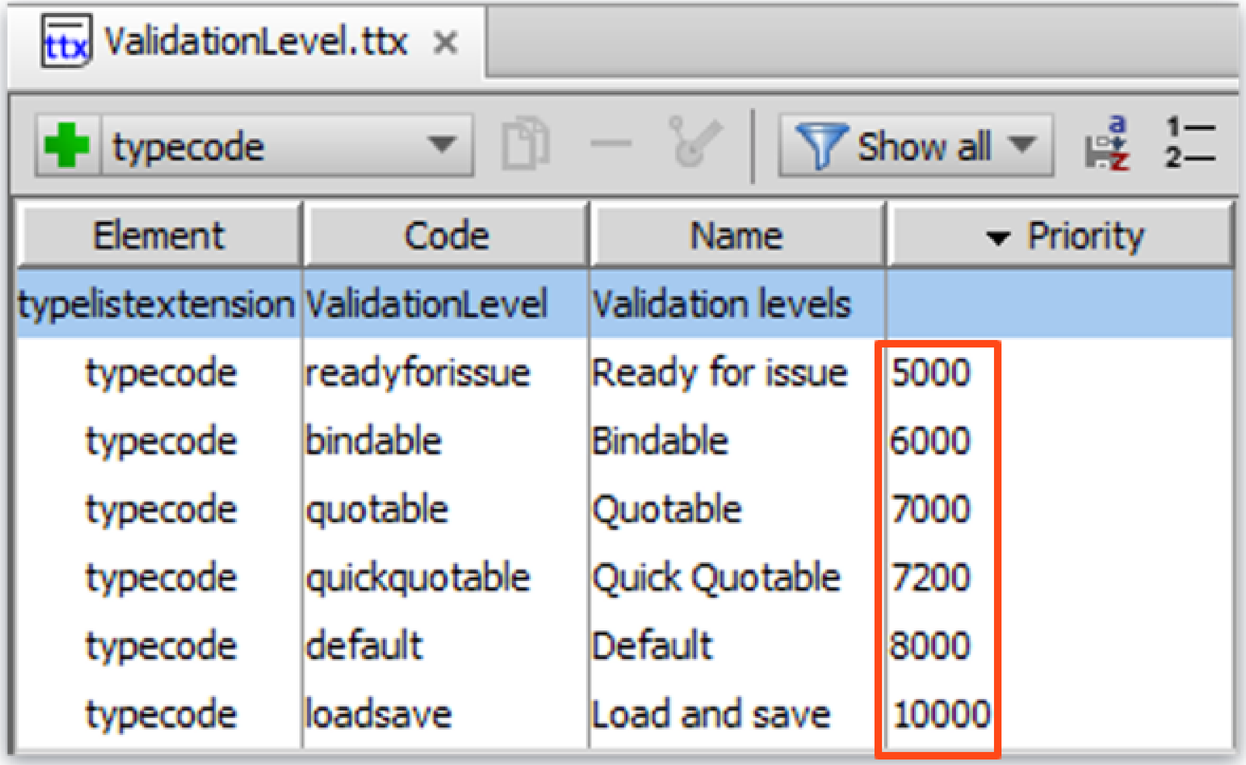

ValidationLevel.ttx

Element	Code	Name	Priority
typelistexten:	ValidationLev...	Validation lev...	
typecode	loadsave	Load and save	10000
typecode	default	Default	8000
typecode	quotable	Quotable	7000
typecode	bindable	Bindable	6000
typecode	readyforissue	Ready for issue	5000
typecode	quickquotable	Quick Quotab...	7200

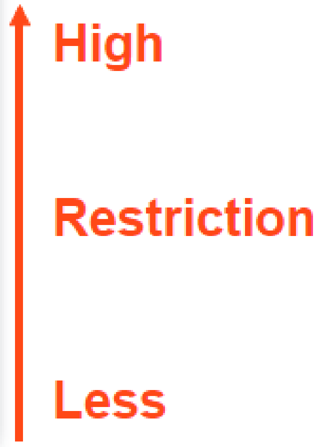
Name	Value
code	loadsave
name	Load and save
desc	Minimum level that must be passed for a Polic
identifierCode	
priority	10000
retired	false

Validation level priority

The higher the priority value, the less restrictive the validation at that level is



Element	Code	Name	Priority
typelistextension	ValidationLevel	Validation levels	
typecode	readyforissue	Ready for issue	5000
typecode	bindable	Bindable	6000
typecode	quotable	Quotable	7000
typecode	quickquotable	Quick Quotable	7200
typecode	default	Default	8000
typecode	loadsave	Load and save	10000



Validation levels and class-based validation

- Run validation at a certain level on an entity or a wizard step
- See if all the validation checks required at the level are passed
  - Add errors or warnings as needed for checks not passed
- PolicyCenter provides methods to determine what validation level is currently being tested

Validation class implementations and methods

Validation-related interfaces and classes

PolicyCenter provides the following in the package *gw.validation*:

- **PCValidation:** Interface that all validation classes must implement
- **PCValidationBase**: Abstract convenience class that takes a PCValidationContext instance
- **PCValidationContext**: Class that takes a ValidationLevel and creates a new PCValidationResult object during initialization
- **PCValidationResult:** Class that contains the methods to use in generating warnings and errors

Validation classes in the base configuration

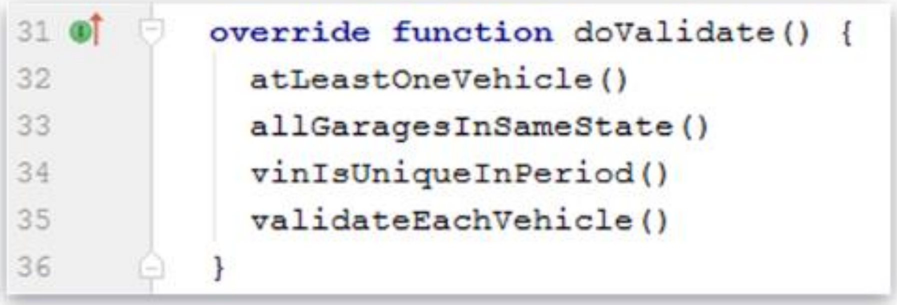
- Classes shared across LOB reside under
  - configuration → gsrc → gw → policy:
    - PolicyPeriodValidation
    - PolicyLineValidation
    - PolicyLocationValidation
    - PolicyContactRoleValidation
- LOB specific validation classes reside under
  - configuration → gsrc → gw → lob → [LOBName]:
  - For example, for Commercial Auto (BA)
    - BALineValidation
    - BusinessVehicleValidation

PCValidation implementations

- All validation classes must implement PCValidation or extend its implementation classes
  - Entity validation classes extend **PCValidationBase**
    - Such as PersonalVehicleValidation or BOPBuildingValidation classes
    - The subclasses have a **validateImpl**() method
  - Policy Line validation classes extend **PolicyLineValidation**
    - such as PALineValidation or BOPLineValidation classes
    - The subclasses have a **doValidate**() method
- Each validation class has a validateImpl or a doValidate method
  - calls other validation methods that check a single problem

Define validation methods that test for a single issue

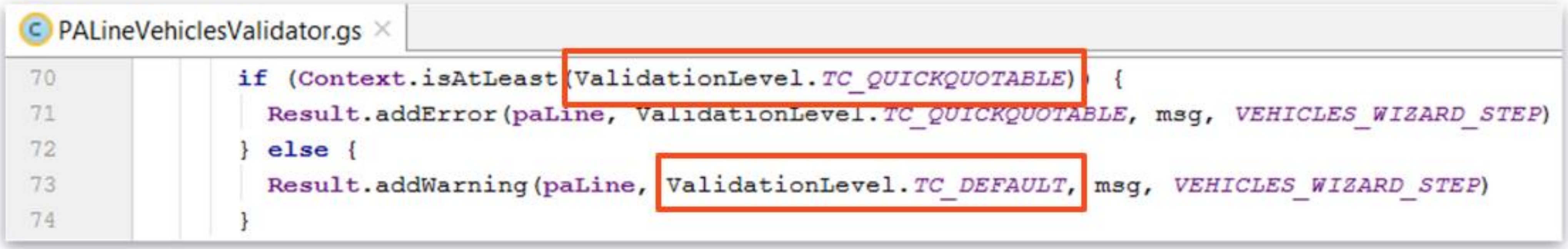
- Call them in the **validateImpl**() *or* **doValidate**() method
- For example, methods to validate issues on personal auto objects are defined in **PALineVehiclesValidator.gs** and called using the *doValidate* method



```
override function doValidate() {
  atLeastOneVehicle()
  allGaragesInSameState()
  vinIsUniqueInPeriod()
  validateEachVehicle()
}
```

PCValidationContext

- PCValidationBase constructor defines validation context PCValidationContext
- PCValidationContext:
  - Takes a ValidationLevel at which to perform validation
  - Creates a new PCValidationResult to store validation results (errors and warnings)



PALineVehiclesValidator.gs

```
if (Context.isAtLeast(ValidationLevel.TC_QUICKQUOTABLE)) {
  Result.addError(paLine, ValidationLevel.TC_QUICKQUOTABLE, msg, VEHICLES_WIZARD_STEP)
} else {
  Result.addWarning(paLine, ValidationLevel.TC_DEFAULT, msg, VEHICLES_WIZARD_STEP)
}
```

PCValidationContext methods

- **isAtLeast**(valLevel)
  - Tests if the level specified by ValidationContext is at least at **valLevel**
- **addToVisited**(validation, methodName)
  - Provides opportunity to trace validation logic
- **hasVisited**(className, methodName)
  - Indicates whether the combination of validation object and method have been seen before
  - returns true if the combination has been visited before
- **showVisited**()
  - Produces a string that lists the validation methods that were visited as validation was performed with the provided Context

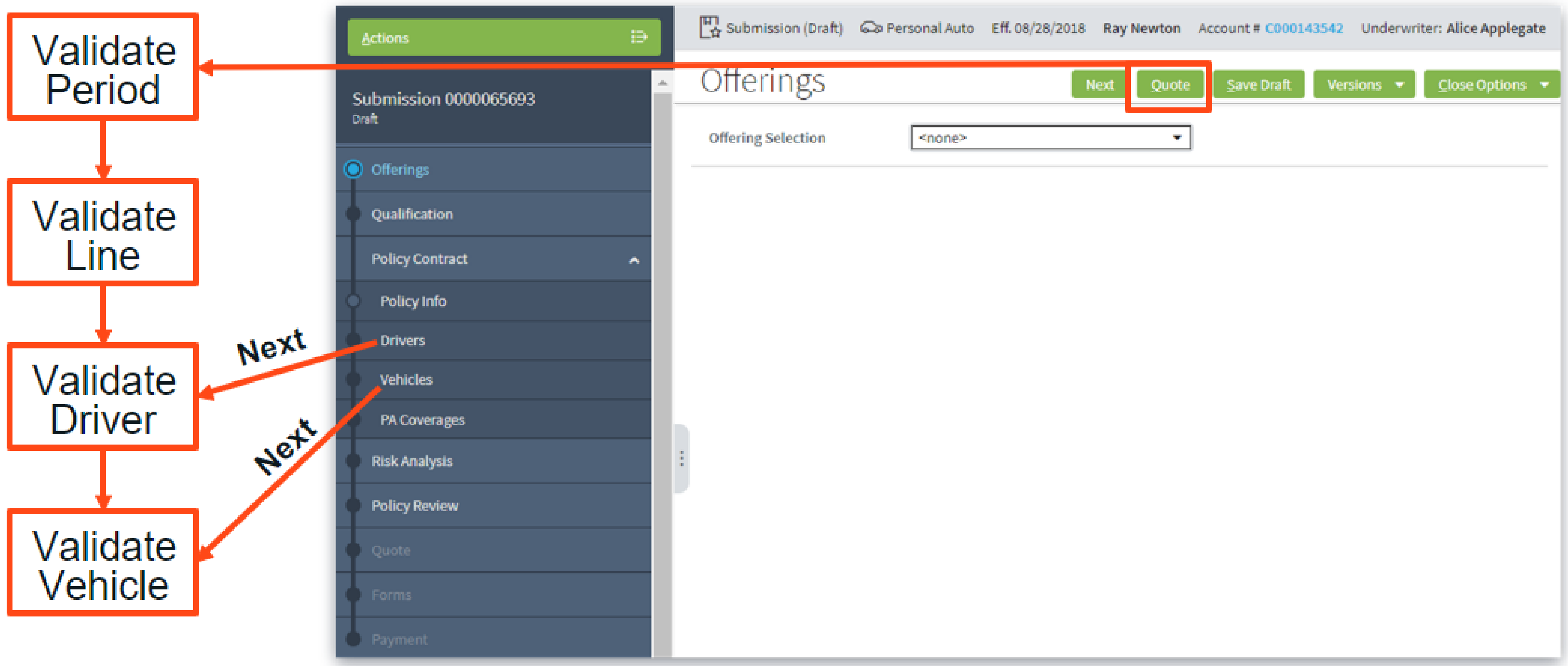
PCValidationResult holds validation results

- PCValidationResult is created by Context methods
- PCValidationResult class provides methods that hold the resulting warnings and errors:
  - **addError**(object, valLevel, errorMessage, wizardStepId)
  - **addFieldError**(object, strRelativeFieldPath, valLevel, errorMessage, wizardStepId)
  - **addWarning**(object, valLevel, warningMessage, wizardStepId)
  - **addFieldWarning**(object, strRelativeFieldPath, valLevel, warningMessage, wizardStepId)
  - where: **valLevel** means that the error or warning will be added at that level or any level more restrictive

Validation chaining

Validation chaining

Validation chaining is the process of one validation class calling another validation class to perform additional validation checks



Validation chain flow

1. Call validate() method which in turn calls other methods

2. Invoke validate() method on another validation class (may have to loop through a set of objects)

3. Classes chain to validations of entities they hold

- A class must be specifically set to perform validation chaining
- In the base configuration, PolicyPeriodValidation class performs validation chaining

Invoking class-based validation

Invoking class-based validation

- Class-based validation must be explicitly invoked through:
  - Code, job processes, wizard steps, workflow steps, integration plug-ins, or from any Gosu code
- To invoke validation from a particular PolicyCenter location:
  - Wizard steps - use beforeSave attribute for that step
  - Pop-ups - use beforeCommit attribute for that pop-up

Creating a new validation class and adding a validation check

Creating a new validation class

- For new entities:
  - Create new validation class for entity you want to validate or
  - Create new validation class when you create a new LOB
- For existing entities:
  - If a validation class for the entity exists add validation check methods to that class
  - If the validation class does not exist, then create new validation class
- New class should extend PCValidationBase
- Use an existing validation class as a reference

Adding a validation check

1. Edit/add appropriate validation class
2. Add the validation check method
3. Call method from validateImpl() or doValidate()
4. Invoke the method
5. Verify the result in the UI

Lesson objectives review

You are now able to:

- Describe class-based validation
- Identify which objects should use validation classes
- Explain how validation levels are used
- Invoke class-based validation from different locations in PolicyCenter
- Write validation checks needed to validate objects on a policy