Lesson 8 Configuring Payment Allocation

8.1 Configure a new payment allocation filter

Succeed Insurance wants to customize how the payment allocation filter behaves.

8.1.1 Requirements

Spec 1 When BillingCenter receives a payment targeted for a policy period, restrict the payment allocation to invoice items belonging to the policy only if the policy is delinquent.

Spec 2 Otherwise, the filter does nothing.

8.1.2 Tasks

1. Add a typecode to the *DistributionFilterType* typelist.
    a. Extend the *DistributionFilterType.tti* typelist.
    b. Suggested typecode name: PolicyPeriodIfDelinquent
    c. Do not restart the server until all configuration is done; otherwise, the system will throw an exception.
2. Create a new class based on PolicyPeriodDistributionFilterCriterion.gs.
    a. Suggested package name: si.bc.classes.paymentallocation
    b. Suggested class name: PolicyPeriodIfDelinquentFilterCriterion
3. Register the new class in LinkedImplementationLoaderImpl.gs
4. Restart the server.

8.1.3 Testing procedure

1. Create a new payment allocation plan by cloning the Default Payment Allocation Plan with these details: (Administration → Business Settings → Payment Allocation Plans)
    a. Name: PP Delinquent Allocation Plan
    b. Remove all existing filters
    c. Add new PolicyPeriodIfDelinquent filter
2. Move the *PP Delinquent AllocationPlan* to the top priority in the list of payment allocation plans.
3. Create a new account.
    a. *QuickJump: Run Account with1PolicyWithNoProducer*
4. Add a second policy to the new account with these details: *(Actions → Add Policy)*
    a. Policy Number: PPID2
    b. Payment Plan: PP02
    c. Premium: $1000
5. Make the first invoice *Billed* by advancing the clock forward to the bill date of the first invoice and running the invoice batch process.

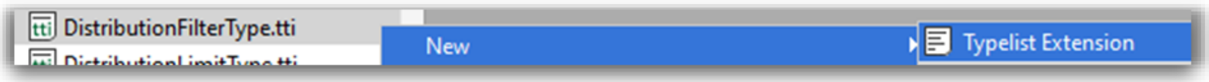
   a. *Open Server Tools (ALT-SHIFT-T)*
   b. *Advance the clock to the first invoice date (if needed) using Internal Tools → Testing System Clock*
   c. *Run the Invoice batch process from Server Tools → Batch Process Info*
6. From the Multiple Payment Entry screen, make a $300 payment on PPID2 policy.
   a. Desktop Tab → Actions → New Payment → Multiple Payment Entry
7. Run the New Payment batch process to allocate the payment.
   a. *Open Server Tools (ALT-SHIFT-T)*
   b. *Run the New Payment batch process from Server Tools → Batch Process Info*
8. Go to the *Charges* screen on the account and examine how the money was allocated.
   a. They should be allocated to both policies since the filter was ignored because neither policy is delinquent.
9. Advance the clock to one day past the due date of the second invoice.
   a. *Open Server Tools (ALT-SHIFT-T)*
   b. *Internal Tools → Testing System Clock*
10. Run the *Invoice* and *Invoice Due* batch processes to cause a past due delinquency on the policies.
   a. *Server Tools → Batch Process Info*
11. From the Multiple Payment Entry screen, make a $300 payment on PPID2 policy.
   a. Desktop Tab → Actions → New Payment → Multiple Payment Entry
12. Run the New Payment batch process to allocate the payment.
   a. *Server Tools → Batch Process Info*
13. Go to the *Charges* screen on the account and examine how the money was allocated.
14. The payment should only be allocated to PPID2 policy that is now delinquent. The other policy should be ignored since this was a targeted payment to PPID2 policy.

8.1.4 Solution

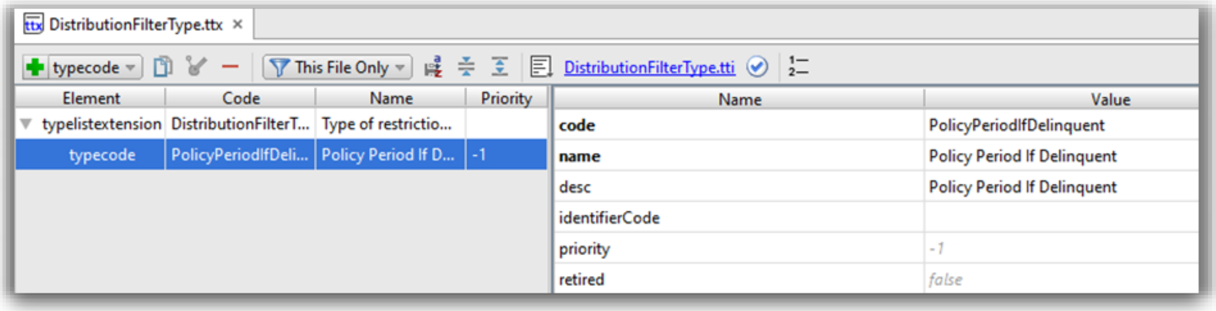
Solution

Exact details on how to complete the exercise.

1. Add a typecode to the *DistributionFilterType* typelist.
   a. Extend the *DistributionFilterType.tti* typelist.



   b. Suggested typecode name: PolicyPeriodIfDelinquent



DistributionFilterType.ttx

Element	Code	Name	Priority
typelistextension	DistributionFilterT...	Type of restrictio...	
typecode	PolicyPeriodIfDeli...	Policy Period If D...	-1

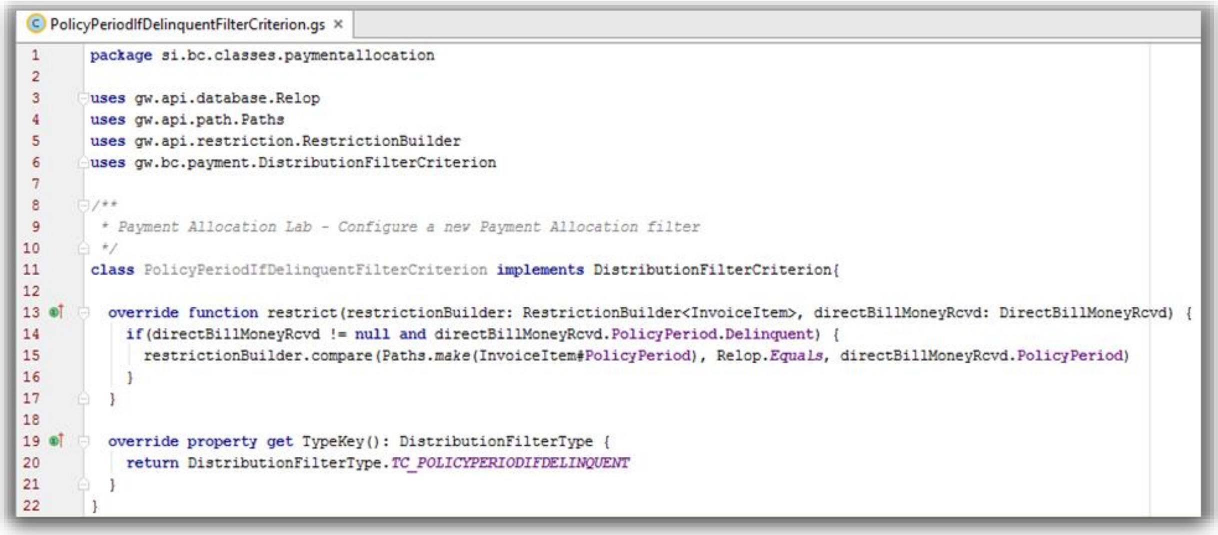
Name	Value
code	PolicyPeriodIfDelinquent
name	Policy Period If Delinquent
desc	Policy Period If Delinquent
identifierCode	
priority	-1
retired	false

c. Do not restart the server until all configuration is done, otherwise system will throw an exception.

2. Create a new class based on PolicyPeriodDistributionFilterCriterion.gs.

   a. Suggested package name: si.bc.classes.paymentallocation

   b. Suggested class name: PolicyPeriodIfDelinquentFilterCriterion



PolicyPeriodIfDelinquentFilterCriterion.gs

```
package si.bc.classes.paymentallocation

uses gw.api.database.Relop
uses gw.api.path.Paths
uses gw.api.restriction.RestrictionBuilder
uses gw.bc.payment.DistributionFilterCriterion

/**
 * Payment Allocation Lab - Configure a new Payment Allocation filter
 */
class PolicyPeriodIfDelinquentFilterCriterion implements DistributionFilterCriterion{

  override function restrict(restrictionBuilder: RestrictionBuilder<InvoiceItem>, directBillMoneyRcvd: DirectBillMoneyRcvd) {
    if(directBillMoneyRcvd != null and directBillMoneyRcvd.PolicyPeriod.Delinquent) {
      restrictionBuilder.compare(Paths.make(InvoiceItem#PolicyPeriod), Relop.Equals, directBillMoneyRcvd.PolicyPeriod)
    }
  }

  override property get TypeKey(): DistributionFilterType {
    return DistributionFilterType.TC_POLICYPERIODIFDELINQUENT
  }
}
```

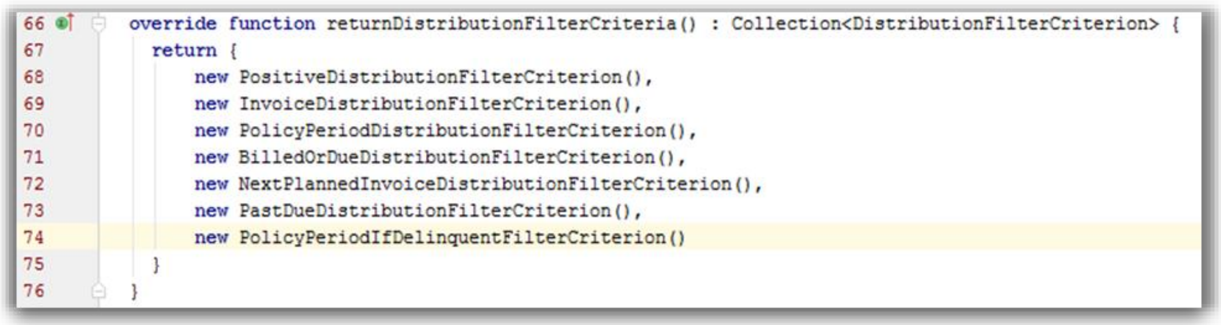
```
package si.bc.classes.paymentallocation

uses gw.api.database.Relop
uses gw.api.path.Paths
uses gw.api.restriction.RestrictionBuilder
uses gw.bc.payment.DistributionFilterCriterion

/**
 * Payment Allocation Lab - Configure a new Payment Allocation filter
 */
class PolicyPeriodIfDelinquentFilterCriterion implements DistributionFilterCriterion{

  override function restrict(restrictionBuilder: RestrictionBuilder<InvoiceItem>, directBillMoneyRcvd: 
DirectBillMoneyRcvd) {
    if(directBillMoneyRcvd != null and directBillMoneyRcvd.PolicyPeriod.Delinquent) {
      restrictionBuilder.compare(Paths.make(InvoiceItem#PolicyPeriod), Relop.Equals, 
directBillMoneyRcvd.PolicyPeriod)
    }
  }
  override property get TypeKey(): DistributionFilterType {
    return DistributionFilterType.TC_POLICYPERIODIFDELINQUENT
  }
}
```

3. Register the new class in LinkedImplementationLoaderImpl.gs



```
override function returnDistributionFilterCriteria() : Collection<DistributionFilterCriterion> {
  return {
      new PositiveDistributionFilterCriterion(),
      new InvoiceDistributionFilterCriterion(),
      new PolicyPeriodDistributionFilterCriterion(),
      new BilledOrDueDistributionFilterCriterion(),
      new NextPlannedInvoiceDistributionFilterCriterion(),
      new PastDueDistributionFilterCriterion(),
      new PolicyPeriodIfDelinquentFilterCriterion()
  }
}
```

4. Restart the server.

8.2 Customize default payment distribution

When a direct bill payment is made, Succeed Insurance wants billed items from workers' compensation policies to be paid before billed items for policies for other LOBs.

8.2.1 Requirements

Spec 1 Distribute direct bill payments to the workers' compensation distribution items first

Spec 2 Distribute the remaining amount (if any) to the other items.

8.2.2 Tasks

1. Customize the allocatePayment() method in the DirectBillPayment plugin.
2. Compile code changes.

8.2.3 Testing procedure

1. Verify that the *Default Payment Allocation Plan* is top priority.
   a. *Administration → Business Settings → Payment Allocation Plans*
2. Create a new account.
   a. *QuickJump: Run Account with1PolicyWithNoProducer*
3. Add a second policy to the new account with these details:
   a. *Actions → Add Policy*
   b. Policy Number: DPD1
   c. Product: Workers' Compensation

   d. Payment Plan: PP02
   e. Premium: $3500
   f. Taxes: $280

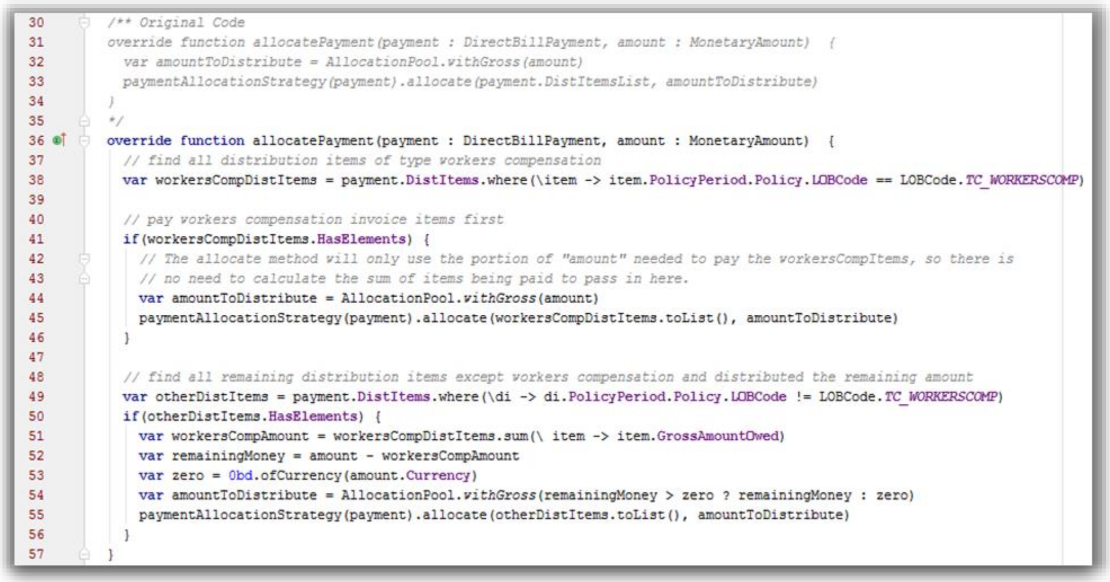
4. Make the first invoice *Billed* by advancing the clock forward and running the invoice batch process.
   a. *Open Server Tools (ALT-SHIFT-T)*
   b. *Advance the clock to the first invoice date (if needed) using Internal Tools → Testing System Clock*
   c. *Run the Invoice batch process from Server Tools → Batch Process Info*
5. Go to the Direct Bill Payment screen.
   a. Actions → New Payment → New Direct Bill Payment
6. Enter $650 as the amount and tab out of the field to observe the customized default allocation.
   a. The workers' compensation items should be paid first, and any remainder should be paid to the other policy.

8.2.4 Solution

Solution

Exact details on how to complete the exercise.

1. Customize the allocatePayment() method in the DirectBillPayment plugin.



```
/** Original Code
override function allocatePayment(payment : DirectBillPayment, amount : MonetaryAmount)  {
  var amountToDistribute = AllocationPool.withGross(amount)
  paymentAllocationStrategy(payment).allocate(payment.DistItemsList, amountToDistribute)
}
*/
override function allocatePayment(payment : DirectBillPayment, amount : MonetaryAmount)  {
  // find all distribution items of type workers compensation
  var workersCompDistItems = payment.DistItems.where(\item -> item.PolicyPeriod.Policy.LOBCode == LOBCode.TC_WORKERSCOMP)

  // pay workers compensation invoice items first
  if(workersCompDistItems.HasElements) {
    // The allocate method will only use the portion of "amount" needed to pay the workersCompItems, so there is
    // no need to calculate the sum of items being paid to pass in here.
    var amountToDistribute = AllocationPool.withGross(amount)
    paymentAllocationStrategy(payment).allocate(workersCompDistItems.toList(), amountToDistribute)
  }

  // find all remaining distribution items except workers compensation and distributed the remaining amount
  var otherDistItems = payment.DistItems.where(\di -> di.PolicyPeriod.Policy.LOBCode != LOBCode.TC_WORKERSCOMP)
  if(otherDistItems.HasElements) {
    var workersCompAmount = workersCompDistItems.sum(\ item -> item.GrossAmountOwed)
    var remainingMoney = amount - workersCompAmount
    var zero = 0bd.ofCurrency(amount.Currency)
    var amountToDistribute = AllocationPool.withGross(remainingMoney > zero ? remainingMoney : zero)
    paymentAllocationStrategy(payment).allocate(otherDistItems.toList(), amountToDistribute)
  }
}
```

```
override function allocatePayment(payment : DirectBillPayment, amount : MonetaryAmount)  {
  // find all distribution items of type workers compensation
  var workersCompDistItems = payment.DistItems.where(\item -> item.PolicyPeriod.Policy.LOBCode ==
LOBCode.TC_WORKERSCOMP)
```

```
  // pay workers compensation invoice items first
  if(workersCompDistItems.HasElements) {
    // The allocate method will only use the portion of "amount" needed to pay the workersCompItems, so
there is
    // no need to calculate the sum of items being paid to pass in here.
    var amountToDistribute = AllocationPool.withGross(amount)
    paymentAllocationStrategy(payment).allocate(workersCompDistItems.toList(), amountToDistribute)
  }

  // find all remaining distribution items except workers compensation and distributed the remaining
amount
  var otherDistItems = payment.DistItems.where(\di -> di.PolicyPeriod.Policy.LOBCode !=
LOBCode.TC_WORKERSCOMP)
  if(otherDistItems.HasElements) {
    var workersCompAmount = workersCompDistItems.sum(\ item -> item.GrossAmountOwed)
    var remainingMoney = amount - workersCompAmount
    var zero = 0bd.ofCurrency(amount.Currency)
    var amountToDistribute = AllocationPool.withGross(remainingMoney > zero ? remainingMoney : zero)
    paymentAllocationStrategy(payment).allocate(otherDistItems.toList(), amountToDistribute)
  }
}
```

2. Compile code changes.

    a. Run → Debugging Actions → Reload Changed Classes

8.3 Demo code: Creating a custom priority

The following code is from the instructor demo.

```
package demo.bc.classes.paymentallocation

uses com.google.common.collect.Ordering
uses gw.bc.payment.InvoiceItemAllocationOrdering

/**
 * Creating a Custom Priority Demo
 */
class ChargePatternCategoryOrdering  implements InvoiceItemAllocationOrdering {

  override property get TypeKey() : InvoiceItemOrderingType {
    return InvoiceItemOrderingType.TC_CHARGEPATTERNCATEGORY
  }

  override function getInvoiceItemOrdering(directBillMoneyRcvd: DirectBillMoneyRcvd):
Ordering<InvoiceItem> {
    return new Ordering<InvoiceItem>() {
      override function compare(item1: InvoiceItem, item2: InvoiceItem): int {
        var patternListByPriority = {
            ChargeCategory.TC_PREMIUM,
            ChargeCategory.TC_TAX,
            ChargeCategory.TC_FEE
        }.reverse()

        var leftPriority = patternListByPriority.indexOf(item1.Charge.ChargePattern.Category)
        var rightPriority = patternListByPriority.indexOf(item2.Charge.ChargePattern.Category)

        return Integer.compare(rightPriority, leftPriority)
      }
    }
  }
}
```