Filtering the File Content MD18

You will now implement a search mechanism to search the product catalog file **catalog.xml**.

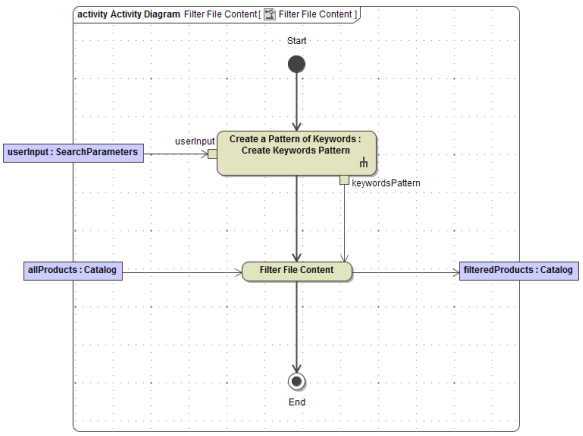
Components

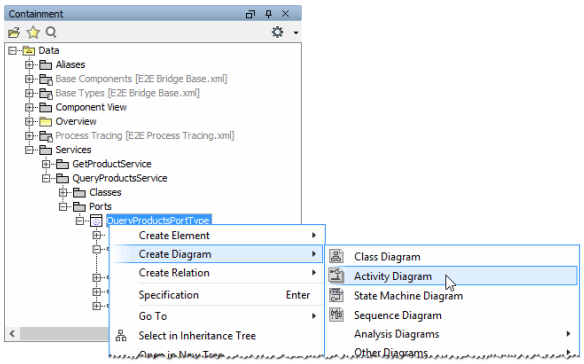
On this Page:

Drawing the Activity Diagram Filter File Content

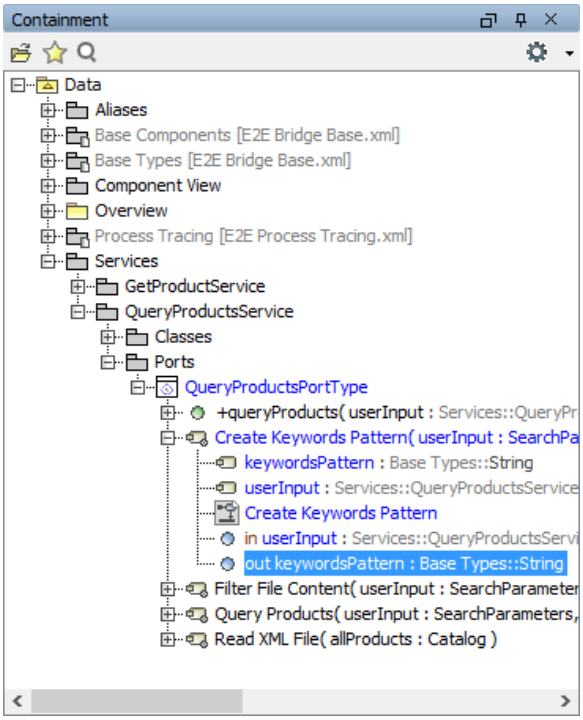
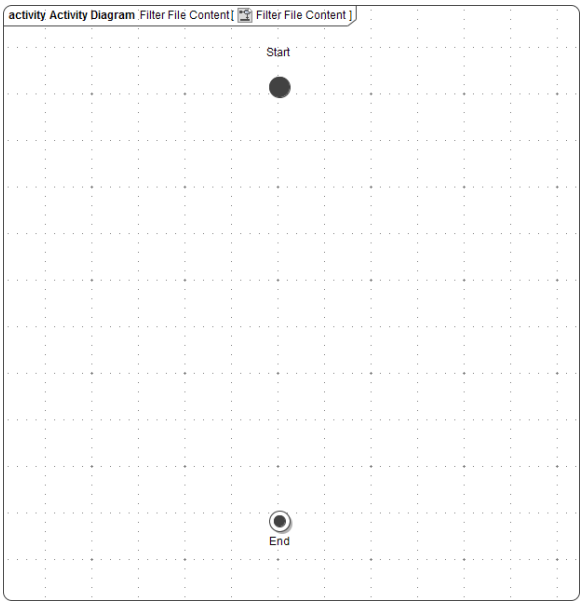
Open the activity diagram **Filter File Content**. It will contain all activities to filter the content of the XML file according to the search parameters.

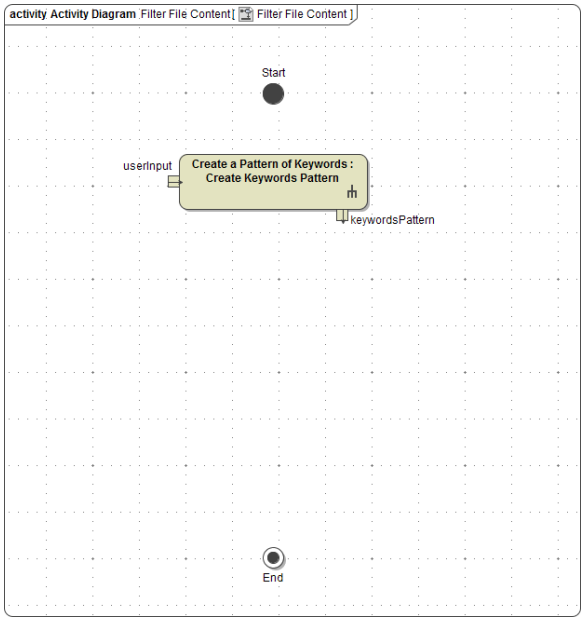
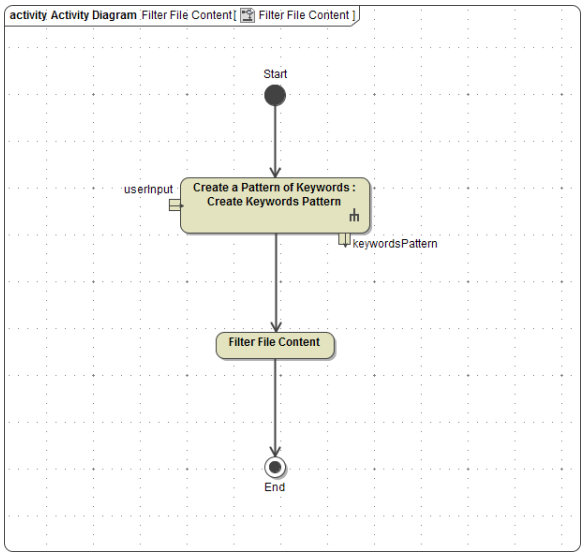
The finalized activity diagram will look as shown below.

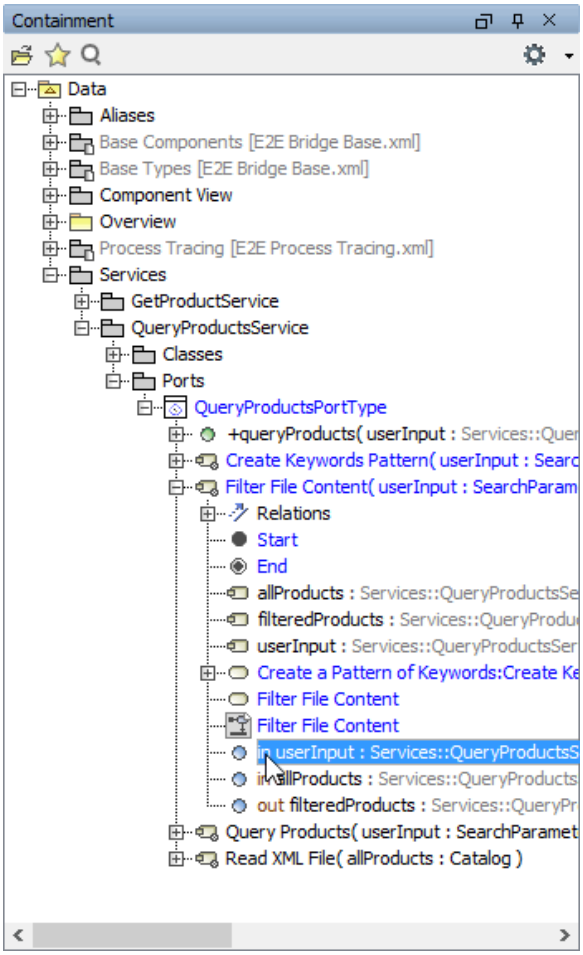




In the containment tree, select **QueryProductsPortType** and create the new activity diagram **Create Keywords Pattern**.

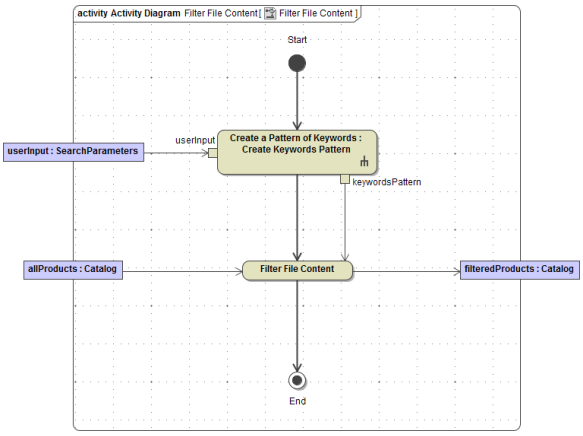
	Add the necessary parameters to the activity: • **userInput : SearchParameters** (direction **in**) • **keywordsPattern : String** (**out**)
	Return to the activity diagram **Filter File Content**: Add an inital node **Start** and an activity final node **End**.

	Drag and drop the activity **Create Keywords Pattern** on the diagram. Open the specification dialog of the action node and assign the name **Create a Pattern of Keywords**. Move the input pin **userInput** on the left side of the action node and the output pin **keywordsPattern** on the right side.
	Add another action node to the diagram and name it **Filter File Content**. Draw the control flow.



To finish the diagram, the necessary parameters have to be added to the diagram.

Select them in the containment tree and...



... drag and drop the input parameters on the left side of the diagram border, the output parameter on the right side and finish the diagram by drawing the necessary object flows.

Save the UML model.

Creating Keyword Patterns

In the next step, you will draw the activity diagram that addresses the keywords search. Requesting the service, the user may fill in no, some, or all keywords in the search mask. Upon this input you have to decide what has to be done next. This decision will be modeled with a decision node. If no keyword has been entered, all rows will be selected. If one or more keywords have been entered, a row will be selected if at least one of the keywords matches the title.

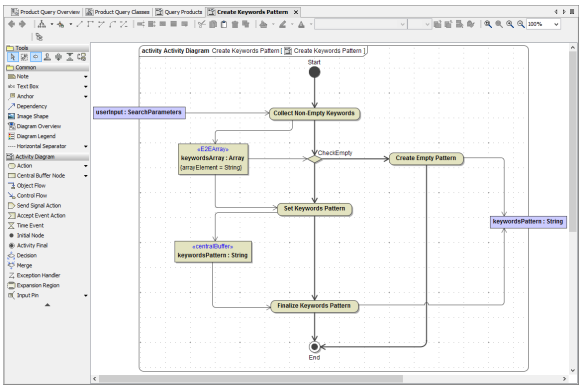
You will create an internal array object **keywordsArray**, which will contain all keywords the user has entered in the search mask. If the number of array elements is zero, you will create a string object called **keywordsPattern** that does not contain any keywords. Otherwise, you will assign a value to this string object that contains all given keywords within regular expressions. The string **keywordsPattern** will be used for filtering the rows later.

Regular expressions are used to find search patterns in a string. You will use this technique in this example to find keywords in the field **title** of each product record. This will be all implemented in the activity **Create a Pattern of Keywords**.

Create a new behavior activity diagram **Create Keywords Pattern** in the package **QueryProductsPortType** and assign it to the action node **Create a Pattern of Keywords**. Double-click the action node to open the empty activity diagram.

First, you will outline all actions of the activity diagram.

1. First, you filter out all empty keywords. The result is stored in an array.
2. In the decision node, the number of keywords in the array is evaluated.
3. If the array is empty, an empty pattern is created. No further action is executed.
4. If the array is not empty, another pattern is defined using the keywords in the array in the following two actions.

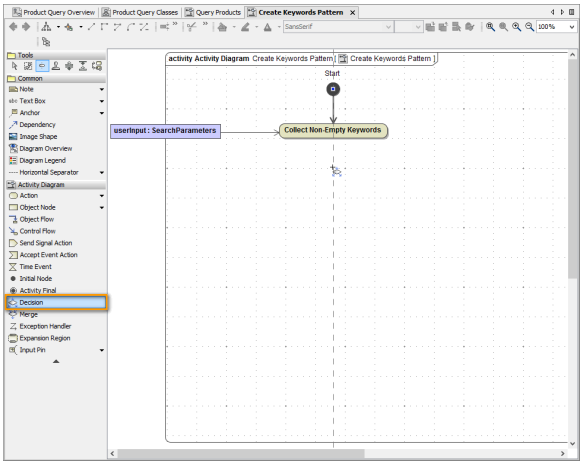
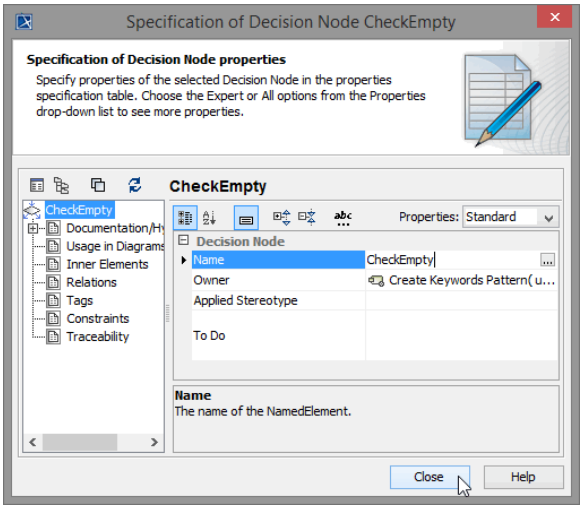


Create the outline of the activity diagram as shown in the picture above.

Element	Name	Type	Direction
Initial and final activity node	**Start** and **End**		
Action nodes	**Collect Non-Empty Keywords**		
	Set Keywords Pattern		
	Finalize Keywords Pattern		
	Create Empty Pattern		
Decision node	**CheckEmpty** (see below on how to draw a decision node)		
Input parameter (stores the search parameters passed from the client)	**userInput**	**SearchParameters**	**in**
Output parameter	**keywordsPattern**	**String** (base type)	**out**
Central buffer nodes	**keywordsArray** (see section below on how to define an array)	**Array** (base type)	
	keywordsPattern	**String** (base type)	

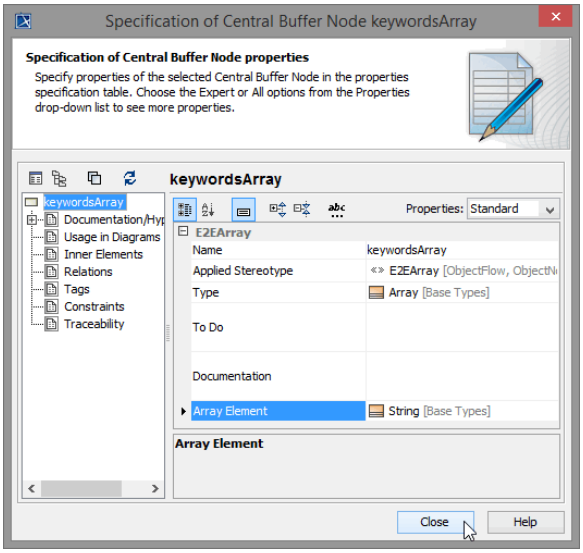
Draw all control and object flows as shown in the screenshot.

Decision Nodes

	In the decision node, you need to decide whether the keywords array from parameter **useRInput** is empty. Select a **Decision** icon from the diagram toolbar and place it on the activity diagram.
	Double-click it to open the Decision Node specification dialog and assign the name **CheckEmpty**.

Arrays

The object **keywordsArray** is an array, which can contain an unlimited number of **String** elements. If an array is drawn in the activity diagram, you always need to define the type of its array elements.

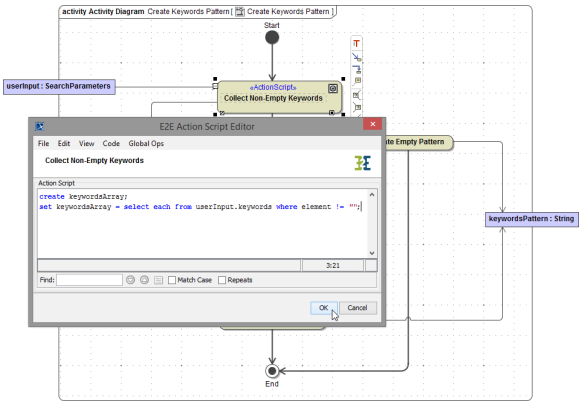
	Double-click the object **keywordsArray** to open its specification dialog. Set the **Type** to **Array**, which is a base type. Each array must be stereotyped as an array. Select the stereotype **E2EArray** in the **Applied Stereotype** field.

Setting this stereotype, the new field **Array Element** is displayed. Define the base type **String** as type of the array elements.

«E2EArray» **keywordsArray : Array** {arrayElement = String}	The type of the array elements is visible now in the second line of the object node.

Collecting Non-Empty Keywords

Open the Action Script Editor for action **Collect Non-Empty Keywords** and define the action script to process the five keywords passed from the client.



Action Script of Collect Non-Empty Keywords

```
create keywordsArray;
set keywordsArray = select each from userInput.keywords where element !=
"";
```

> Arrays always need to be created with the create statement.

> When typing the select keyword followed by a blank, the statement will be completed except the object and the condition in the where clause.

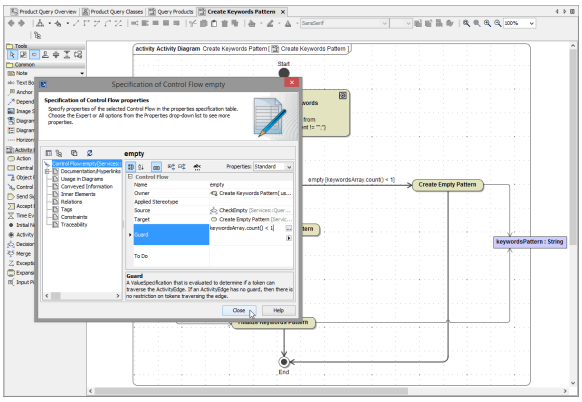
The operation `select each from` processes every element of array **keywords**, which is an attribute of input object **userInput**.

By defining the condition `where element != ""`, the operation only considers keyword fields, which the user did not leave empty in the search mask (read: array element that does not equal an empty string). All keywords that are found in the array **keywords** and match the condition will be copied to the array **keywordsArray**.

Counting Elements of the Array

If no keywords are passed, the flow branches from the decision node **CheckEmpty** to the right. As you copied all keyword fields which have not been empty from the input object **userInput** to the array object **k eywordsArray**, you can now count all elements. If the array has no elements the flow will branch to the right.

Double-click the control flow that branches to the right. Name the control flow **empty** and enter the following condition in the **Guard** field.

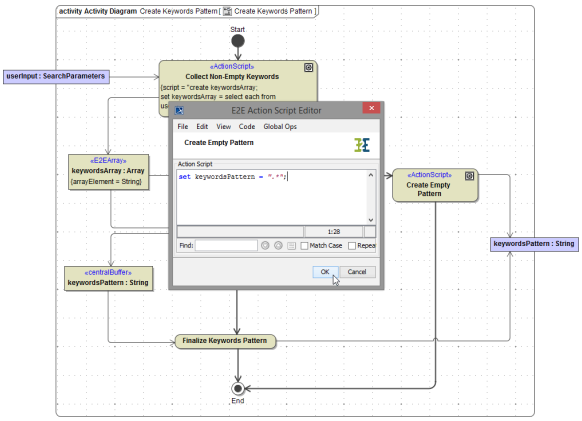


Statement	Description
`keywordsArray.count() < 1`	The count operation counts all elements in the array object **keywordsArray**. The condition is true, if the array has no elements (less then 1).

As an alternative, you can edit the guard statement with the Action Script Editor which gives you more control of the syntactical correctness. Select the control flow and press **Ctrl** - **Enter** to start the Action Script Editor.

Creating an Empty Pattern

Open the Action Script Editor for action **Create Empty Pattern**. Add the action script that assigns a regular expression to the string object **keywordsPattern**. Later on, this regular expression stored in the string will be used to select all rows, because no keywords were entered. See detailed information about regular expressions in the xUML Services Reference Guide.



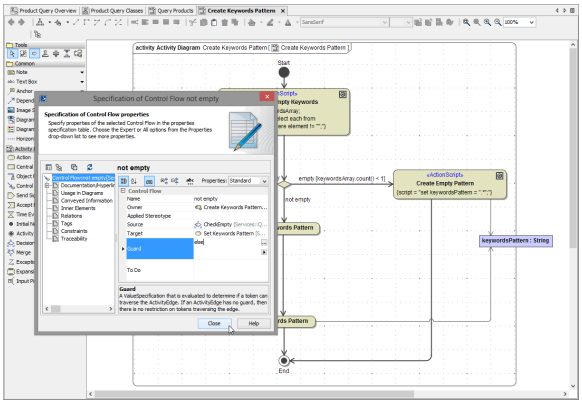
Operators of Regular Expressions	Description
.	Match any character.
*	Match 0 or more times. Match as many times as possible.

Statement	Description
`set keywordsPattern = ".*";`	Assigns the regular expression `.*` to the output string object **keywordsPattern**. The search pattern `.*` will find all characters of a string, which means that in the search every **title** will match (all records will be returned).

Second Branch of the Decision Node

If at least one keyword is entered, the flow branches from the decision node **CheckEmpty** downwards. Remember, that the flow branches to the right, if the number of elements of array **keywordsArray** is less than 1. If this condition is not true, the flow branches downwards.

Double-click the control flow, name it **not empty** , and enter else in the **Guard** field.

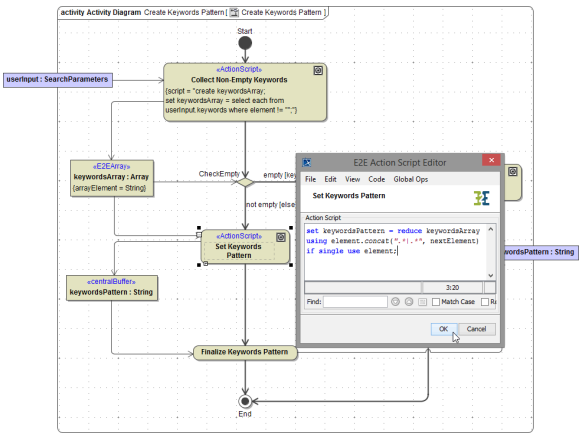


Statement	Description
`else`	else represents all other cases that were not specified explicitly.

In the next step, you will construct a search statement with regular expressions in the action state **Set Keywords Pattern**. The search pattern will also be stored in the string object **keywordsPattern** and contains all keywords found in the array **keywordsArray**.

Setting the Keyword Pattern

Open the Action Script Editor for action **Set Keywords Pattern**. Add the action script below that assigns a regular expression to the string **keywordsPattern**. Later on, the search pattern will be used to select all matching rows according to the entered keywords.



Operators of Regular Expressions	Description
.	Match any character.
*	Match 0 or more times. Match as many times as possible.
|	Alternation. A | B matches either A or B.

Action Script of Set Keywords Pattern

```
set keywordsPattern = reduce keywordsArray using element.concat(".*|.*",
nextElement) if single use element;
```

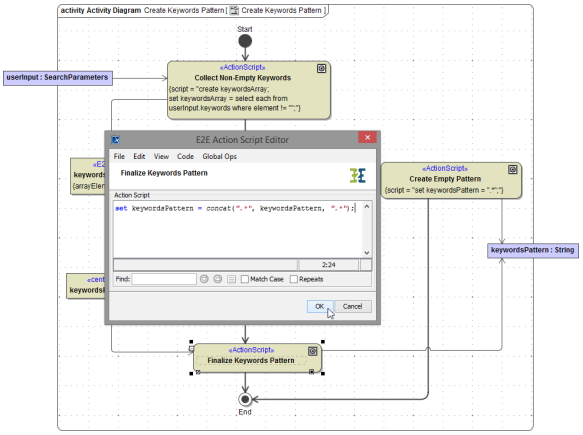
The statement `reduce ... using concat` collects all elements of an array in a simple string. When iterating over the array **keywordsArray**, element and nextElement are placeholders for the current and the next array element of each iteration.

`element, ".*|.*", nextElement` builds a list composed of a combination of **keywordsArray** elements and the regular expression `.*|.*`. If the array contained only one element, the resulting string would only contain the value of this element (defined by if single use element). Below, you will find an example on how this regular expression works.

Finalizing the Keywords Pattern

To finish the activity diagram, complete the search statement by concatenating another regular expression at the beginning and at the end of the string object **keywordsPattern**. In this activity diagram, this is the last step in the action flow.

Open the Action Script Editor for action **Finalize Keywords Pattern** and enter the statement below.



Action Script of Finalize Keywords Pattern
`set keywordsPattern = concat(".*", keywordsPattern, ".*");`

The operation `concat` concatenates the regular expression **.*, the current value of *keywordsPattern**, and the search pattern `.*` a second time. The new value will be re-assigned to the string **keywordsPattern**.

The following shows an example of a resulting value in the string object **keywordsPattern**.

Three keywords are passed as input from the client (**userInput.keywords**):

- **nemo**
- **lord**
- **pirates**

Value of string **keywordsPattern**, which will be used to search in each element **title** (compare action scripts in action node **Set Keywords Pattern** and **Finalize Keywords Pattern**):

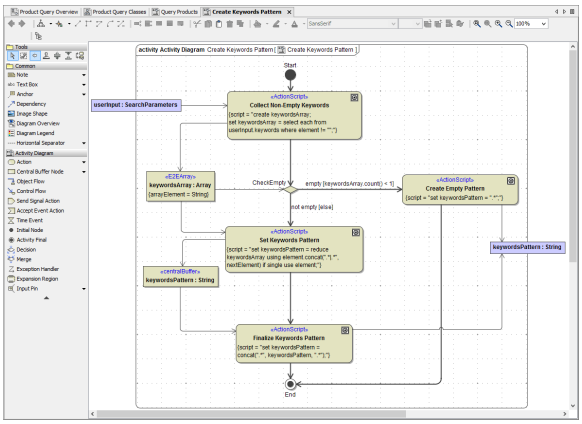
```
.*nemo.*|.*lord.*|.*pirates.*
```

How to read the regular expression:

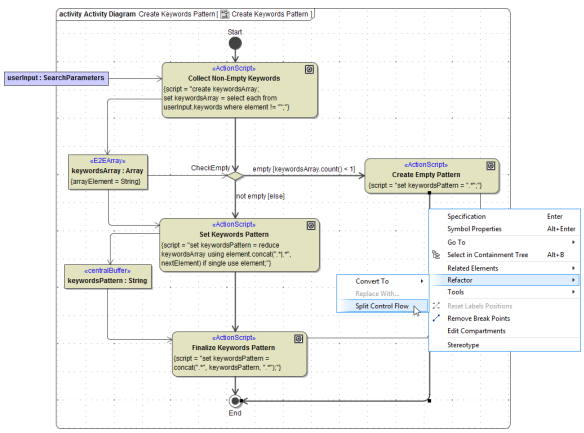
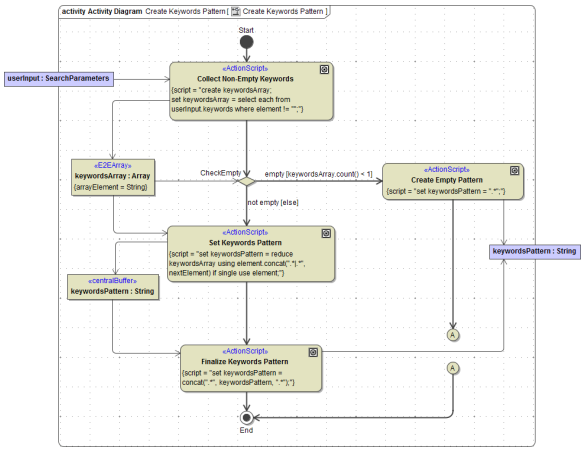
Match any character as many times before and behind **nemo**, or any character as many times before and behind **lord**, or any character as many times before and behind **pirates**.

In other words:
If any of the three keywords will be found anywhere in the title, then the search pattern matches, and the row has to be selected.

The activity has been completed.



In order to make the model more readable and to prevent crossing object or control flows, MagicDraw offers a functionality to split flows. Follow the following steps to split the control flow from the action node **Create Empty Pattern** to the activity final node.

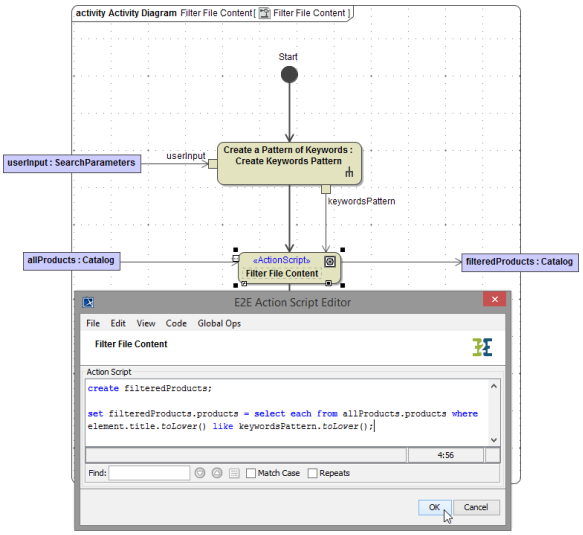
	Right-click the control flow and select the menu item **Refactor > Split Control Flow** from the context menu.
	The control flow is now split and has two new ends. The round symbol (A) marks the points where the control flow is bridged. Rearrange the control flow ends according to the picture on the left.

Save the UML model.

Filtering the Content

Return to the activity diagram **Filter File Content**.

Now, you will use the search term, which you have stored in the object **keywordsPattern** in the previous steps. In the action node **Filter File Content**, the actual search will be performed. Enter the statements below to the action script of the action node **Filter File Content**.



Action Script of Filter File Content

```
create filteredProducts;
set filteredProducts.products = select each from allProducts.products
where element.title.toLower() like keywordsPattern.toLower();
```

The operation `select each from` processes every element of array **products**, which is an attribute of input object **allProducts**. The search terms are applied to element.title, which is explained as follows. **element** temporarily stores array elements of array **products**, which are of type **Product**. The class **Product** has a string attribute **title**, which is considered in this search (compare also to the data structure, which you defined in the class diagram).

The search term is stored in the string **keywordsPattern**, which you prepared before. `toLower()` will set all characters of the search term and **title** elements to lower case as the search should not be case sensitive. Array elements that match the search pattern according to the where clause (`where element.title.toLower() like keywordsPattern.toLower()`) will be stored in the attribute **products** (base type array) of object **filteredProducts** (`set filteredProducts.products =`).

Save the UML model.