

INFSCI 2930 - INDEPENDENT STUDY

Enhance features of Authoring tool for Java and Python

Saurabh Dhamnaskar

sad148@pitt.edu

Raghav Raman

rar155@pitt.edu

1.Overview:

Inspired by Professor Peter Brusilovsky, Kamil Akhuseyinoglu, Roya Hosseini, our topic of independent study in 2018 Spring is to enhance features in existing authoring tool and pc ex compiler to enable professors create challenges and examples for students in Java and Python programming language courses.

In this enhancement, we focused mainly towards easing the process of creating exercises for the professors which can be given to students to complete.

2. Used Technologies:

JSP, HTML, CSS, nodejs

3.Working:

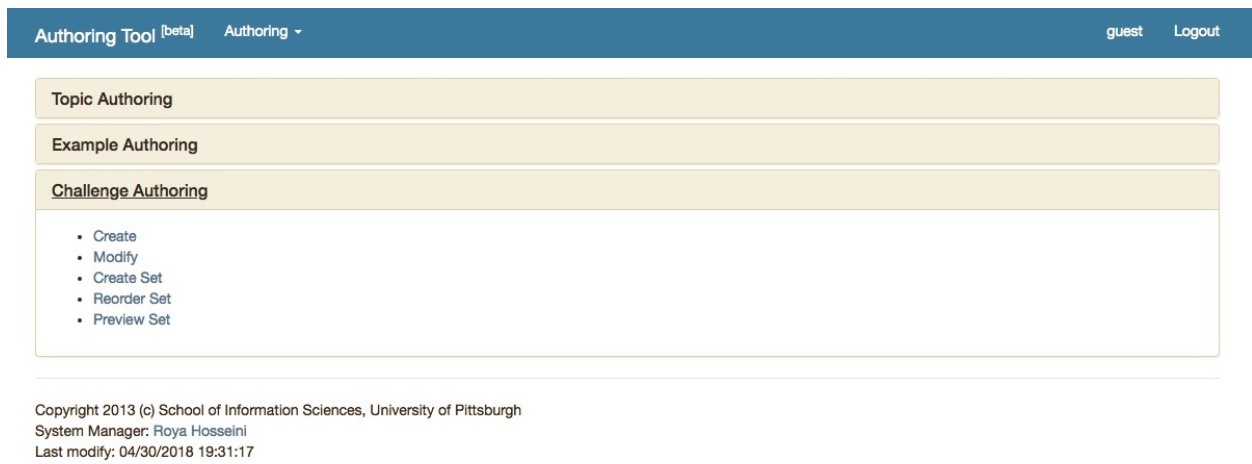


Figure 1 - First page

There were tabs to create a topic and example for a various programming language. But there was no option to create a combination of challenges and examples. Hence, we designed the part “Challenge authoring” and everything it has to offer. Now, the professors can select various options like create, modify, edit challenge/example/set.

Create:

Type: * Example

Language: * Java

Scope: * Java examples for IS2470

Topic: * Interfaces

Title: * Example 1 Interfaces

Description: This example is for interfaces in JAVA

Privacy: * Private Public

User Input: 12
34|

Code: *

```
2      A cash register totals up sales and computes change due.
3      */
4      public class CashRegister
5      {
6          /**
7           Constructs a cash register with no money in it.
8          */
9          public CashRegister()
10         {
11             purchase = 0;
12             payment = 0;
13         }
14
15         /**
16         Records the sale of an item.
17         @param amount the price of the item
18         */
```

Next

Figure 2 - Create example 1

There are 2 options – Challenge and Example. As shown above, we will first create an example. After selecting example, we get options to select language. Since, we focused only on Java and Python, there are only 2 options to choose from. After selecting language, scope of the activity(challenge/example) is selected, upon which we get list of topics like arrays, classes etc depending on the combination of language and scope. There are 2 fields to enter activity title and description. The professor also needs to select privacy level. Privacy level public defines that the activity is open to students to complete whereas privacy level private defines that the activity is not complete and is under progress.

There can be activities in which user input might be required. Hence, there is a field which is not mandatory to enter user inputs. There is an area given to upload code. After clicking on submit, an API is called which indents the code and sends back the indented response.

Code:	Comment:
<code>/** A cash register totals up sales and computes change</code>	
<code>public class CashRegister {</code>	
<code>/** Constructs a cash register with no money in it. */</code>	
<code>public CashRegister() {</code>	
<code>purchase = 0;</code>	
<code>payment = 0;</code>	
<code>}</code>	

Figure 3 - Code splitting

As shown in above figure, the code is split on every new line. There is an option to indent or outdent the line. The professors can also add comments against every line which will be visible to students when they go through the activity. To delete a line, a “trash” icon is present and “add icon” is used to add a new line to the existing code.

Type: *	Challenge
Language: *	Java
Scope: *	Java examples for IS2470
Topic: *	Interfaces
Example: *	Example Interfaces
Title: *	Challenge Intefaces
Description:	Challenge for Interfaces
Privacy: *	<input checked="" type="radio"/> Private <input type="radio"/> Public
User Input:	45 67

Please uncheck the checkboxes to create blank lines in the challenge

<input checked="" type="checkbox"/>	Code:	Comment:	
<input checked="" type="checkbox"/>	<code>/** A cash register totals up sales and computes change</code>		+
<input checked="" type="checkbox"/>	<code>public class CashRegister {</code>		🗑️ +
<input checked="" type="checkbox"/>	<code>/** Constructs a cash register with no money in it. */</code>		🗑️ +
<input checked="" type="checkbox"/>	<code>public CashRegister() {</code>		🗑️ +
<input checked="" type="checkbox"/>			🗑️ +
<input checked="" type="checkbox"/>	<code>private double purchase;</code>		🗑️ +
<input checked="" type="checkbox"/>	<code>private double payment;</code>		🗑️ +
<input checked="" type="checkbox"/>	<code>}</code>		🗑️ +
<input checked="" type="checkbox"/>			🗑️ +

Figure 4 - Create challenge

To maintain consistency in the system, the process to create challenge and example are same. For every example, there are challenges created. Hence, while creating challenge, the professor needs to select the example title from Example dropdown. After selecting the appropriate example, the specific challenge is mapped. Additionally, after indenting the code there are few more operations need to be performed by the professor for a challenge creation. After code is split into separate lines, each line has a checkbox assigned against it. These checkboxes help to eliminate those specific lines in the challenge. The lines which are unchecked are shown as blank lines to the students. To fill these blank lines, options are given to students to choose from. These options are called distractor lines and can be added by clicking on Add distractor lines button.

Add Distractor Lines ✕

Code:

int i;

Comment

+ +

Close Save changes

Figure 5 - Add distractors

The professors can add distractors using add icon aligned against comment textbox. To explain the meaning of a specific distractor line, additional comments can be added in the comment box. These distractors are not mapped to blank lines. It means that for 2 blank lines there can be many distractor lines making it difficult for students to choose the correct ones. The options remain same for every blank line.

Once all data is filled for challenge or example, save changes button need to be clicked to create the activity.

Modify your challenge or Example

Type: *

Scope: *

Topic: *

Example: *

Get Code

	<code>public static final double PENNY_VALUE = 0.01;</code>	<input type="text"/>		
	<input type="text"/>	<input type="text"/>		
	<code>private double purchase;</code>	<input type="text"/>		
	<code>private double payment;</code>	<input type="text"/>		
	<code>}</code>	<input type="text"/>		
	<input type="text"/>	<input type="text"/>		

Copyright 2013 (c) School of Information Sciences, University of Pittsburgh
 System Manager: Roya Hosseini
 Last modify: 04/30/2018 19:43:08

Save

Figure 6 - Edit example

The already created example can be edited by selecting edit option on homepage(Fig 1). The process of editing example is similar to creating one. The options to select a specific example is few to filter out the appropriate example.

Modify your challenge or Example













Type:

Scope:

Topic:

Challenge:

Get Code

<input checked="" type="checkbox"/>		<input type="text" value="private double purchase;"/>	<input type="text"/>		
<input checked="" type="checkbox"/>		<input type="text" value="private double payment;"/>	<input type="text"/>		
<input checked="" type="checkbox"/>		<input type="text" value="}"/>	<input type="text"/>		
<input checked="" type="checkbox"/>		<input type="text"/>	<input type="text"/>		

Distractor Lines

Code: Help List:

Copyright 2013 (c) School of Information Sciences, University of Pittsburgh
 System Manager: Roya Hosseini
 Last modify: 04/30/2018 19:43:08

Save

Figure 7 - Edit challenge

The already created challenge can be edited by selecting edit option on homepage(Fig 1). The process of editing challenge is similar to creating one. The options to select a specific challenge is few to filter out the appropriate challenge.

Create A Set:

1
2
3

Select Example Select Challenge Order the Set

Select the Example

Scope: *

Topic: *

Example: *

Next

Figure 8 - Create Set (Select example)

To create a set of challenge and example, the professor needs to select an example first. Depending on the selected example, the challenges associated with that example are fetched and displayed on next page (Fig 9).

Create A Set:

A horizontal progress bar at the top shows three steps: 1. Select Example, 2. Select Challenge (highlighted in blue), and 3. Order the Set. Below the progress bar, the heading 'Add Challenges' is centered. Underneath, the text 'Please select the list of Challenges for the Activity' is followed by a single checkbox labeled 'Challenge Interfaces' which is checked. A blue 'Next' button is positioned to the right of the checkbox.

Figure 9 - Create Set (Select challenge)

Here, the professor can select challenges which need to be included in the set.

Create A Set:

A horizontal progress bar at the top shows three steps: 1. Select Example, 2. Select Challenge, and 3. Order the Set (highlighted in blue). Below the progress bar, the heading 'Order the Set' is centered. Underneath, there are two list boxes. The first list box contains 'Example Interfaces' and the second list box contains 'Challenge Interfaces'. A green 'Submit' button is positioned to the right of the list boxes.

Figure 10 - Create Set (Re-order set)

In the final step of creating a set, the professor needs to re-order the activities inside the set. By default, the example is kept at first position but it can be re-ordered. On clicking “Submit”, the set is created and ready to be shown on Pc-Ex.

After the exercises are visible to students, the students can see the sequence as was defined by the professor while creating the set. Earlier, all possibilities of options were calculated using permutation and combination and the output of that specific combination was saved. This helped in quicker response when any of the option was chosen by students irrespective of whether it is correct or incorrect. But the drawback of this approach was seen when the options increased for an exercise. The CPU intensive task gradually increased making the whole application slow.

To solve this, we came up with the approach of dynamic execution of java and python code. We implemented caching using memcache npm module. Whenever a student chooses an option, the code is sent to the API to execute. The execution is processed in following steps –

- a. Check if code is already present in cache
- b. If present in cache, return the output of the code from the cache
- c. If it is not present in cache, compile and execute the code. Store the output in cache
- d. Send the output to the front-end

This approach helped in increasing efficiency. For every new option, a cache was hit, and empty was returned. This ensured that the combination is not present in cache and hence, the code is executed, and output is stored in cache. For every request after the first request for that combination, the response was sent directly from cache. But this is not the case for previewing the code. If a professor wants to preview a code, then the combination is stored in the cache. The code is compiled and executed and returned to the front-end. This is because, the professor just checks to see whether the code is running or not.

4. Future Scope

1. We intend to include other programming languages in the application
2. Currently only one set of 1 example and many challenges can be created. We plan to make it flexible by allowing creation of set of any combination of example and challenge.

5. Codes

1. Authoring tool - https://github.com/sad148/authoring_tool
2. PcEx Compiler - https://github.com/sad148/pcex_compiler