University of Pittsburgh

Final Project Report

INFSCI 2470 Interactive Systems Design

Interactive Programming Problems in Java

Team:
Sai Supreetha Varanasi (sav52)
Preksha Behede (pnb16)
Zhenqiang Xu (zhx31)

Clients:
Roya Hosseini
PhD Student (Intelligent Systems Program)

Prof. Peter Brusilovsky
School of Information Sciences
University of Pittsburgh

Abstract

Interactive programming problems is a platform for presenting examples designed to help users learn concepts in Java programming language. The innovative idea is to create "rich examples" that allow students to explore the programming constructs in three different ways. *Examples* comprises of multiple "goals" that gives the user a problem statement. The user can change the "goal" and play with different versions of the programming concept. The features are implemented in a sequence of short fragments of code to illustrate how the program is constructed. *Faded Examples* is fixed in terms of its goal statement. It is similar to multiple choice question answers where user has to understand the goal statement and choose the correct answer to it. *Problem* is also single goal oriented exercise designed to be more challenging than the former two types. Here the lines of code are jumbled and the user needs to rearrange them into a correct version of java code to make it work. The user get to think in terms of the control flow that will solve the goal.

The interface of this application tries to encompass elements that are perceived to be engaging to a user and more interactive in nature. The design of the interface allows for challenging the user to be more explorative in terms of learning a new programming language constructs. The interactive elements include the draggable tiles containing the lines of code, designated button for availing hints/help, visually easy on the eye palette with some colors specifically chosen for implicit feedback, instructional explanations for a subset of example lines that play an important role in understanding the goal statement, explorability function in making the user change interface elements to obtain a specific result rather than simply selecting a pre-defined modification which in turn is allows user to modify and change the goal and the output of the programs.

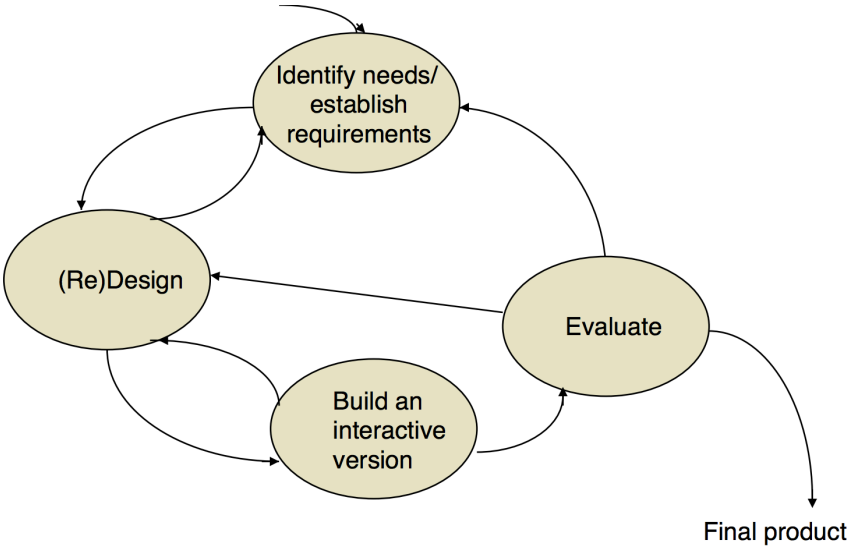
The outcome we wish to see is to make the task of learning Java programming language more fun. Our hope is to implement these interactive programming so learning in explorative way and engaging with examples is possible and useful to all kinds of users from beginner level to advanced.

Contents

Table of Contents

Development lifecycle

In this project we followed an iterative design and development lifecycle that is of the form portrayed in the below figure. The process involved constant communication with the client, Roya Hosseini, to meet her requirements and to adapt to the changes required by her. Our team started with the first step of Requirement gathering, then designed an initial version of the mockup that had improved over a number of iterations for incorporating clarifications and interface elements specifications. Once the initial version was presented to the class, the feedback from peers as well as the instructor was taken into consideration for a major change in the interface design. This new design was also refined over some more for consistency and was established as the prototype for the implementation of the project. The next phase involved the Task analysis of this prototype. This phase involved the participation in user studies to garner their perspective for the usability and for prototype evaluation. The user studies culminated with the users taking and online questionnaire to provide feedback on record about the prototype. The responses were consolidated for data analysis.



The next phase is the implementation of the prototype. This again involved the iterative approach where the team consulted with the client to propose the schema for the data to be used in the interface. For the front-end part, the project follows a Single Page Web Application format with the design pattern of model-view-viewmodel. The team implemented the parts of the interface and then integrated in those parts periodically for maintaining consistency. For implementation latest web technologies were used because many of the interactive elements needed were readily available for incorporation. The project currently has only one class of problem implemented completely which can be further extended to other concepts of Java with the help of the master data file being used.

Requirements Gathering

Mockup from Client:

The requirements from client initially had the below specifications for the interface. These mockups provided our team a head-start into design ideas and gave a reference for the expectation

from the client. These mockups describe the user tasks for the python programming language which needed to be emulated for the current project which deals with Java programming problem.

Our team also had a reference for design from the Parson-js examples, also based on python programming language. The brief overview of the tasks in the interface are given in these below figures which the details given in figure description.

1. An instance of the program construction example with an if-else condition

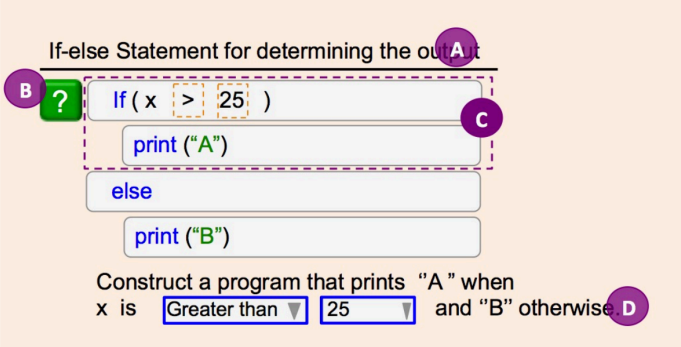


Fig1: (A) The subgoal label for the if-else block (B) The link to instructional explanation for the first fragment in the example (C) The code fragments in the example (D) The example's goal

2. An instance of program construction example with an if-else condition

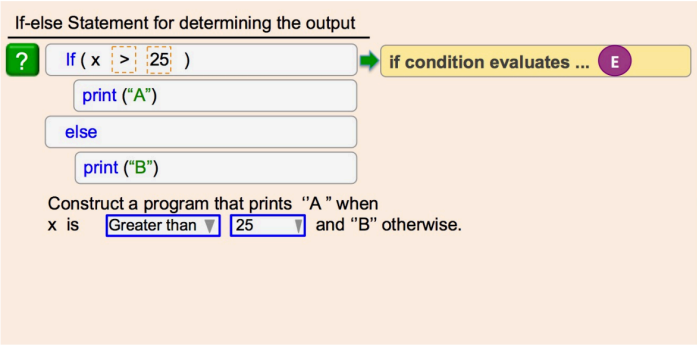


Fig2: (E) Clicking on the question mark icon for explanation

3. Exportability function to let user change example's goal

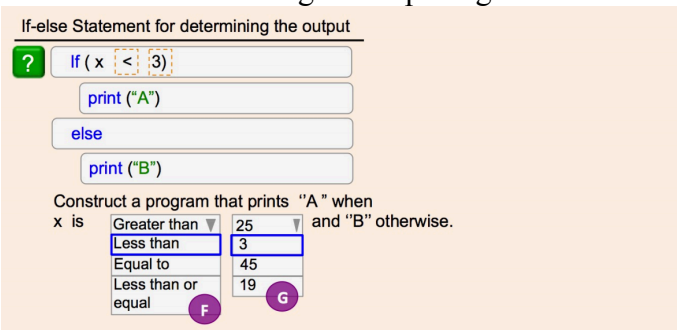


Fig3: (F) Modifying operations and constants (G) Updating of explanation with selected modification

4. Exportability function to with changing results

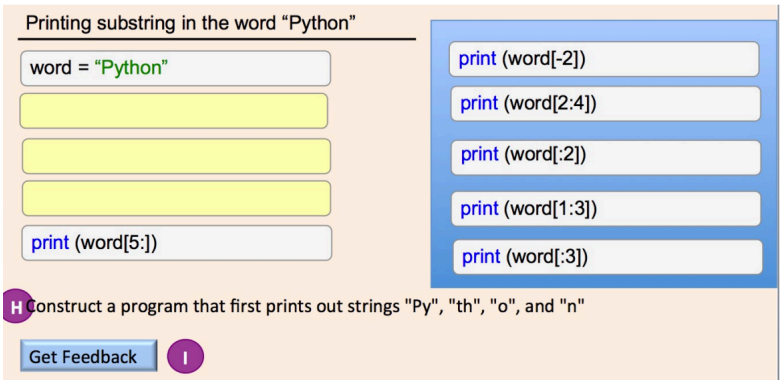


Fig4: (H) Drag and drop function of code fragments (I) Obtain feedback regarding the correctness of the fragment placement

5. Evaluation of fragment placement

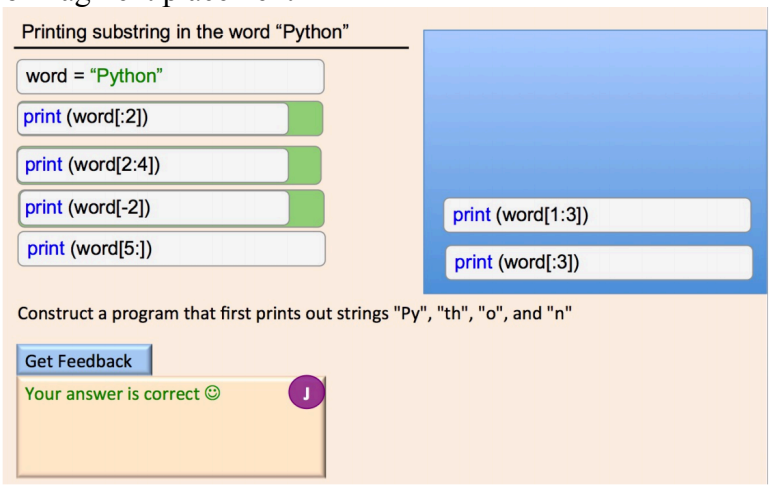


Fig5: (J) Result box for displaying the feedback of evaluation

6. Different version of evaluation

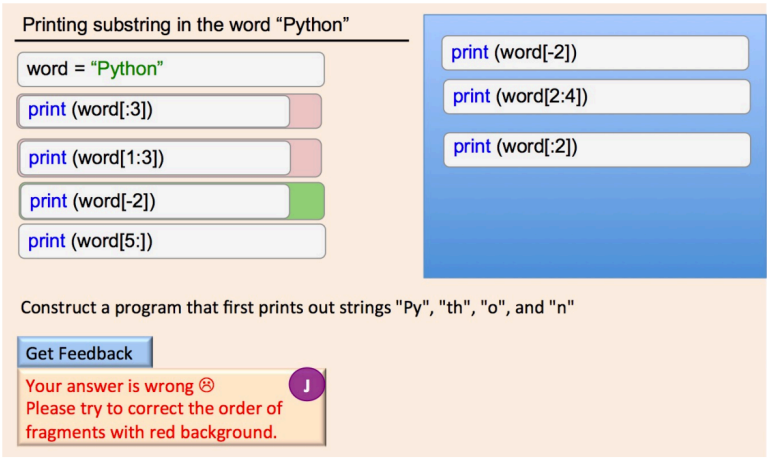


Fig6: (J) Result box with different feedback message

Timeline corresponding to requirements gathering:

Week 1:

On Mar 7th 2017, we had our first meeting with the client, Roya Hosseini, to know more about the project. The salient features of our project are:

- This will be a stand-alone compilation of Interactive Java programming problems.
- Problems are designed to help to reinforce the concepts of the language to the user.
- User gets to play with the elements of the program and see or change the output.
- Initially we decided on the three types of examples for the user to play with: 'Static example', 'Construct your program' and 'Goal driven example' based on the mockup provided to us.
- 'Static example' is fixed in its goal and the user can only change the inputs - Designed to make user see different output based on changing inputs for a given goal.
- 'Construct your program' is a jumbled, fully written program that are user needs to arrange to make it produce output without any error -Designed to make user understand the syntax of the language.
- 'Goal driven example' is one where the user not only gets to change the input but also part of the logic such so as to change the goal of the problem - Designed to make user understand different logic snippets in the program.
- All the problems will have a help button that a user can use to click and move forward with the example when stuck.

Week 2:

- We spent the next week on designing the initial mockup for the project. We took help of the references our client provided as the basis for our design.
- We used the Balsamiq wireframing tool to help us design the mockup and tried to portray all the tasks involved with its many different elements.
- Since we wanted to follow an iterative model for the design and implementation, we set up a meeting with the client again to show her our initial project mockup.

Design and Task Analysis of Mockup 1

The target users for this platform was established as beginner Java learners or students who are expected to have had some basic knowledge about this programming language and are looking for playing with some interactive examples.

An interactive example in this project can be one of three kinds: 'Examples', 'Faded Examples' and 'Problems'. We took help of the references our client provided as the basis for our design. We used the Balsamiq wireframing tool to help us design the mockup and tried to portray all the tasks involved with its many different elements.

There are supporting elements like the help button. Each help button will be 'hover over it to see help' and/or 'click it to see help'. This design is to minimize clicks and to help with recollection problem a user might face while accessing help.

Interface elements:

We compiled the list of various elements in the interface that need to be designed. The type of interface is established as tabbed-pane where the three tabs will hold their corresponding elements. Compilation of mockup implementation decision we took is given below:

- Interface design for example tab, faded example tab, and problem tab:
  - Design goal material on the top
  - Design the program view on the left side
  - Design the fragment code or goal that need to drag on the right side
  - Design the output view on the bottom
- Combine these three tabs into the same page
  - Organize the view consistently
  - Implement help feature
  - Implement the elements to be dragged
  - The dragged elements are able to fill in program or goal suitable
  - Size of tiles should be consistency
  - User can change sequence of fragment code in problem tab
- Display output or error
  - Red color border means the program has error
  - Green color border means the program is correct
  - Highlight selected element and related place where needs to fill

Task Analysis:

'Example' is making user to know how program changes when user selects different inputs or goals in the program. The changes and goal would be highlighted, so **user is able to see the condition changes in program** clearly. We have decided upon three versions of implementing this tab because even after discussing with the client, we wanted a mass opinion from our peers after our in-class presentation regarding the suitable version for the main task of user being able to change the input as well as goals of the example. The task analysis of this main action in all the three versions is given below.

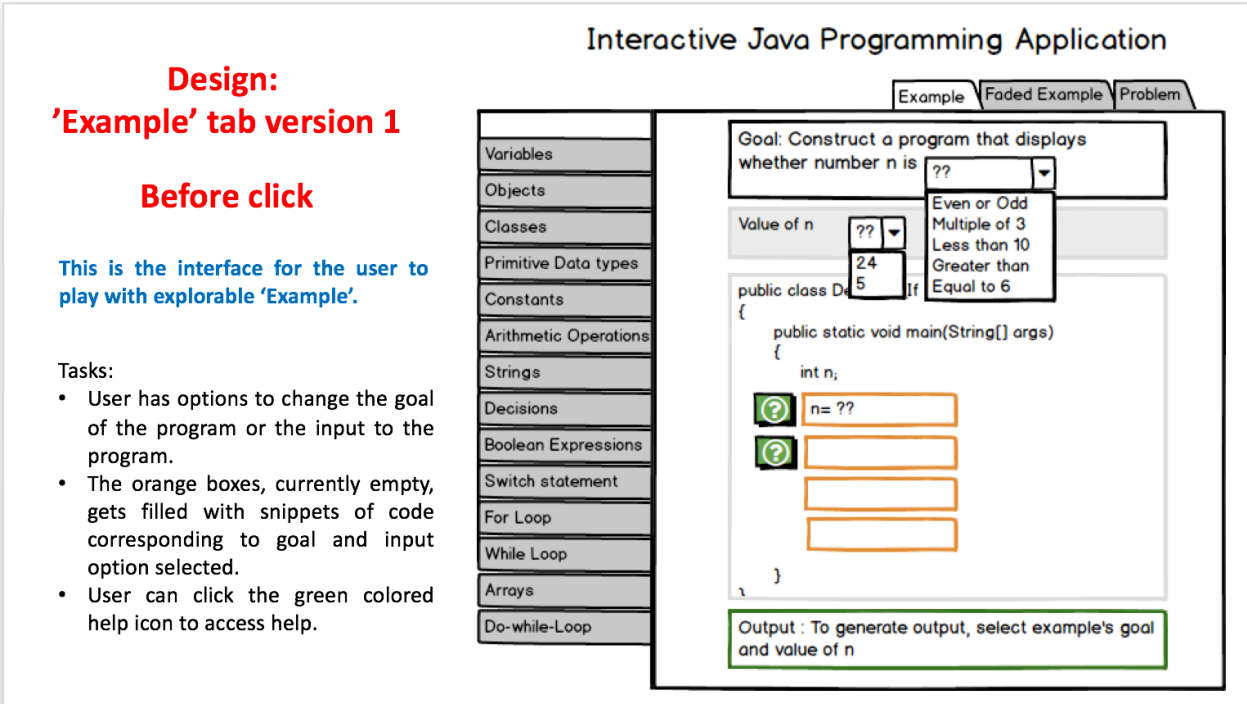


Fig7: Before click – Example version 1

Design:
'Example' tab version 1

After click

Analysis:

- Here all the available options are visible from the drop-down menu design.
- User may feel overloaded with these options if they are all visible at once.
- This is because in this 'Example' tab the user is changing both goal as well as input in any order.

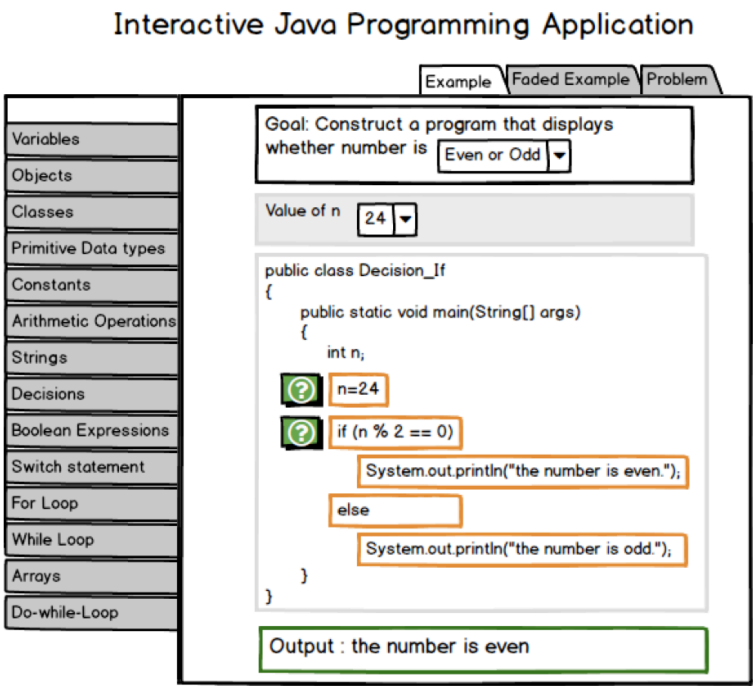


Fig8: After click – Example version 1

Design:
'Example' tab version 2

Before click

Tasks:

- User has to change the goal or input of the program by clicking the ?? Buttons.
- User then gets to see that orange colored boxes automatically are filled based on the current option clicked.

Analysis:

- Here all the available options are hidden because the chosen design for options is a button.

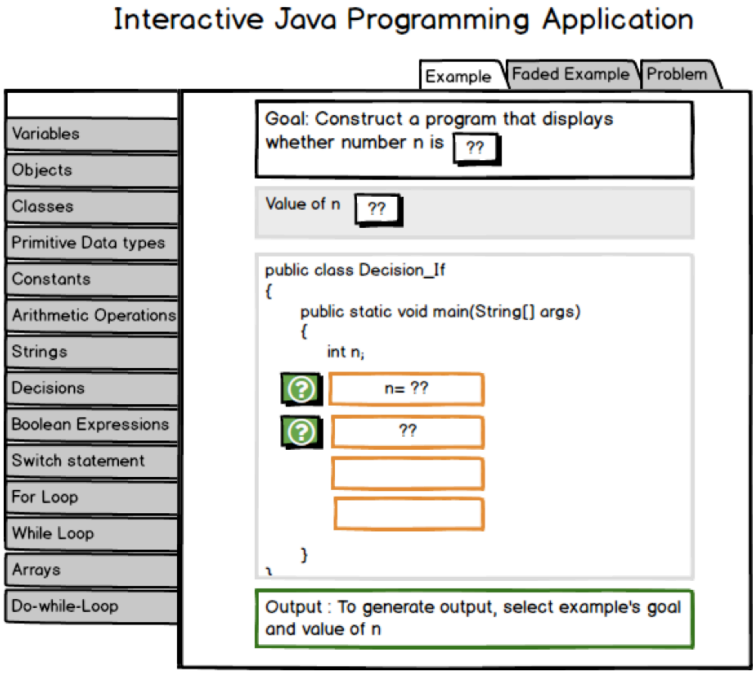


Fig9: Before click – Example version 2

Design: 'Example' tab version 2

After click

Analysis (cont.):

- User clicks the button and sees the option but may not know what are all the other options.
- This, on one hand reduces the information overload and on the other hand increases the number of clicks the user has to do to arrive at a desired option or to know all the available options.
- Another problem is the recall problem – user may not remember the previous option unless going through the cycle of options again, increasing the clicks even more.

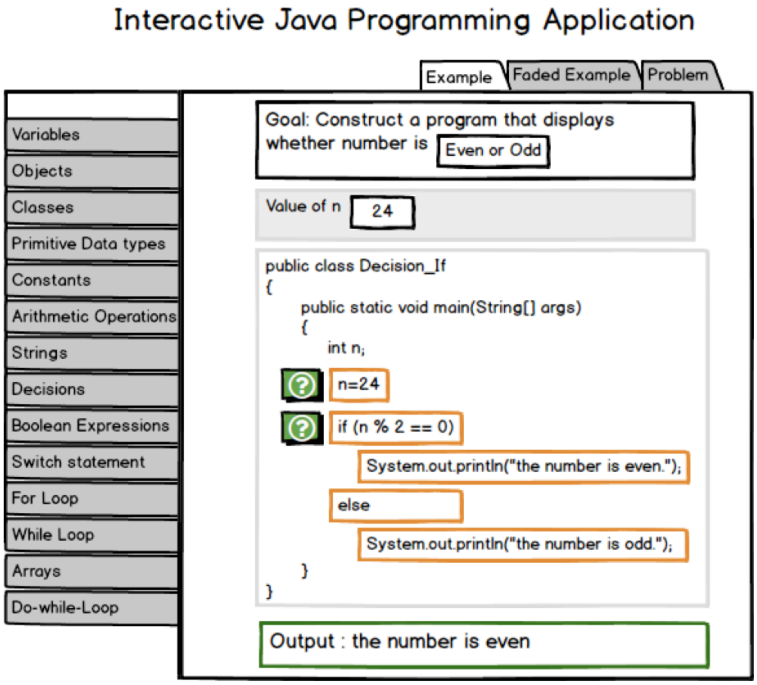


Fig10: After click – Example version 2

Design: 'Example' tab version 3

Before click

Tasks:

- User can change the goal of the program by dragging the tiles from the options on right-hand side to the red-outlined box in the interface. This updates the empty box in the goal portion of the interface.
- However, the input selection is a drop-down menu element.
- User can know what he needs to do by clicking the green help icon if he/she feels the interface is confusing.

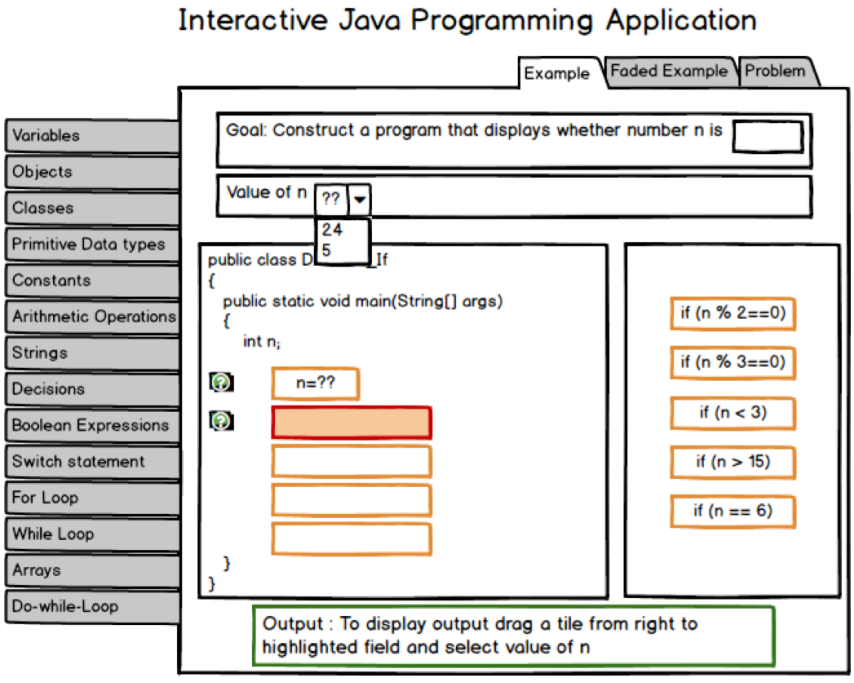


Fig11: Before click – Example version 3

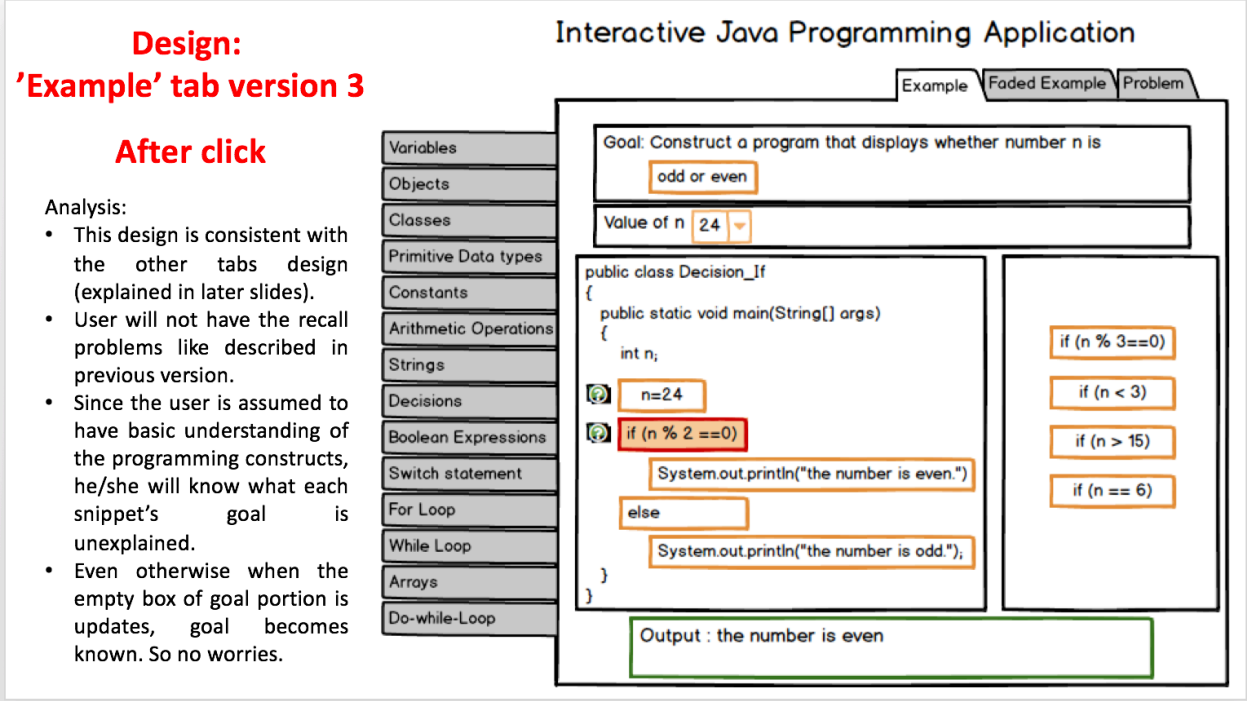


Fig12: After click – Example version 3

'Faded example' is the next tab in our design of the mockup. Here making user to fix the part of fragment code depending on the specific goal by dragging the fragment code is the main task. This time, user has more exploration because they need to **construct a part of program depends on the goal** rather than observing related goals and conditions.

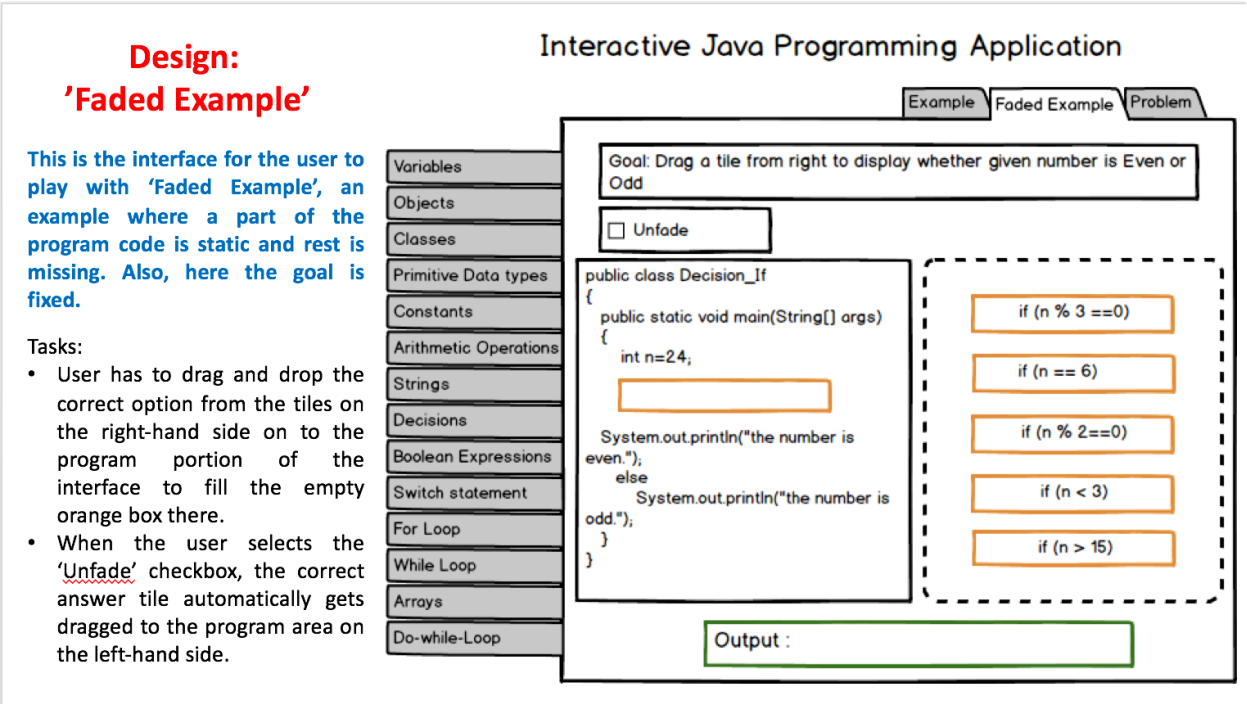


Fig13: Before dragging – Faded example

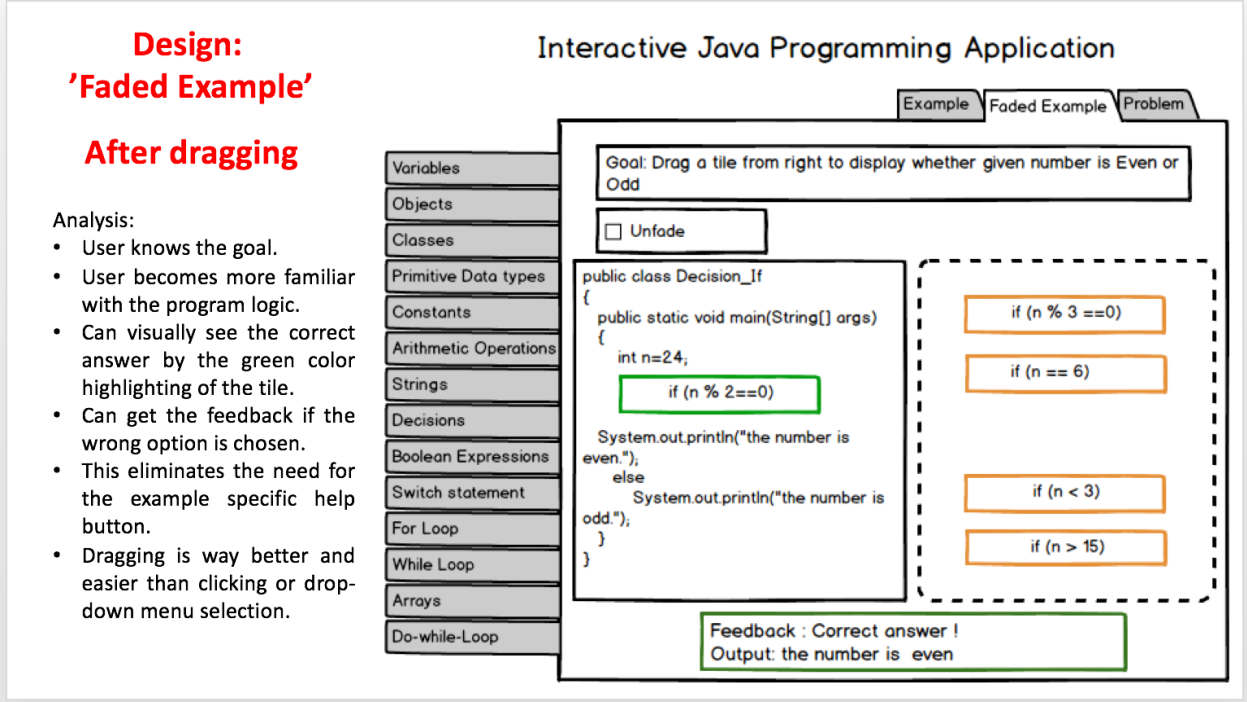


Fig14: After dragging – Faded example

'Problem' tab is next in the interface design. It challenges the user to fill in the multiple line fragment code to construct a completed program depends on a specific goal. Due to previous familiar with example and faded example, the problem can **let user to think more about sequence and logic in the program**, so they are able to write their own code after using this Java program construction.

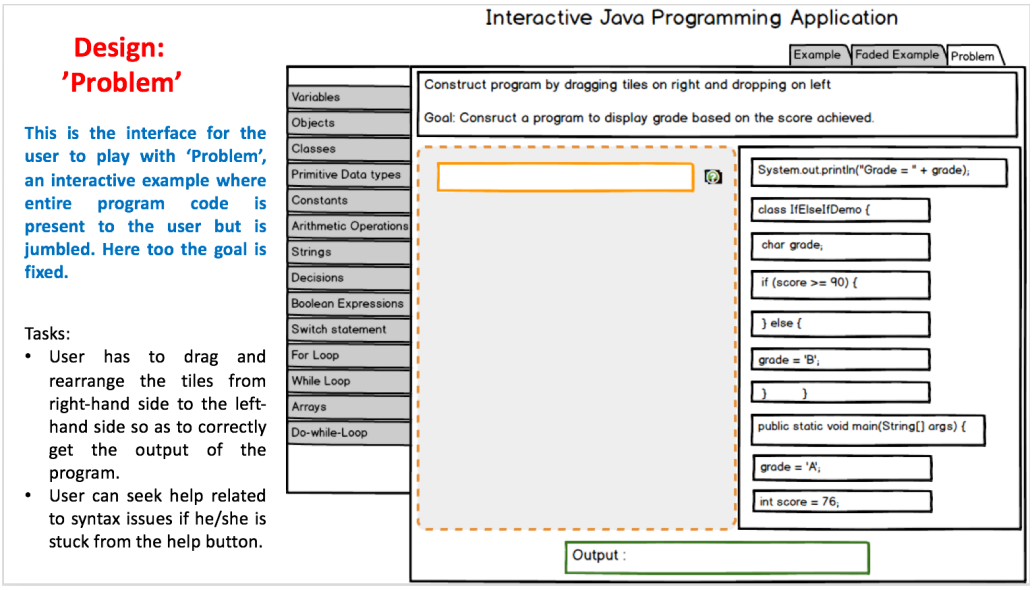


Fig15: Problem

Evaluation of Mockup 1

Our team meetings with client and in-class presentation served the purpose of evaluation of this first version of the mockup. The problems with the Mockup 1 is listed below:

- No clear difference between the tabs and their functions:
- The example tab had three versions but they didn't seem to represent the main task of the tab rather they are doing different things.
- The interface was not consistent in that two of the tabs had draggable tiles portrayed while the only one version of the example tab had the draggable elements.
- There was no help feature displayed in the mockup when is one of the most important design heuristic.

Timeline corresponding to design and analysis of mockup 1:

Week 3:

On Mar 20th 2017, we had our second meeting with the client where Prof. Brusilovsky also joined us. We presented the first version of mock up that we prepared.
Based on the discussions we had on that day, we decided to change the features of the project in such a way that the new version has the programming examples defined as below:

- User now will have three options in terms of the kinds of problems: Explorable examples, Faded examples and Full Problems.
- Explorable examples are the goal driven problems where the user can change the goal statement and consequently the output of the example - Designed to make the user change the goal and the inputs for the program.
- Faded examples and Full problems are fixed in terms of the end goal.
- Faded examples are those where the user gets to change the logic snippets in the program while the rest of the program is static- Designed to make the user study the program and select the best possible snippet that satisfies the logic and produces the correct output.
- Full Problems are the jumbled lines of code that the user needs to rearrange to make it work - Designed to reinforce the syntax of the language to the user.
- Each help button will be 'hover over it to see help' and/or 'click it to see help'. This design is to minimize clicks and to help with recollection problem a user might face while accessing help.
- We spent the weekend on refining the initial version to incorporate the changes discussed in the meeting. We also kept constantly in touch with the client through email to take in the comments for each version of the mockup and iteratively improve our design.
- After a number of email exchanges we finally arrived at our 5th revision of the mockup that we presented in the following week's class.

Week 4:

On Mar 25th 2017, we presented the fifth version of our mockup to the class. This one is different from the first in many ways:

- This contains three designs for the implementation of 'Explorable problems'. We decided to do this to get a feedback from our peers as prospective users of this system to know which one is best to go with.

- We also tried to present the pros and cons associated with each of the three designs for 'Explorable example'.
- Based on the comments we received to our presentation, we understood that our project needs to follow the 'Be consistent' heuristic.
- We will incorporate the necessary changes where applicable during the implementation of the project.

Design and Task Analysis of Mockup 2

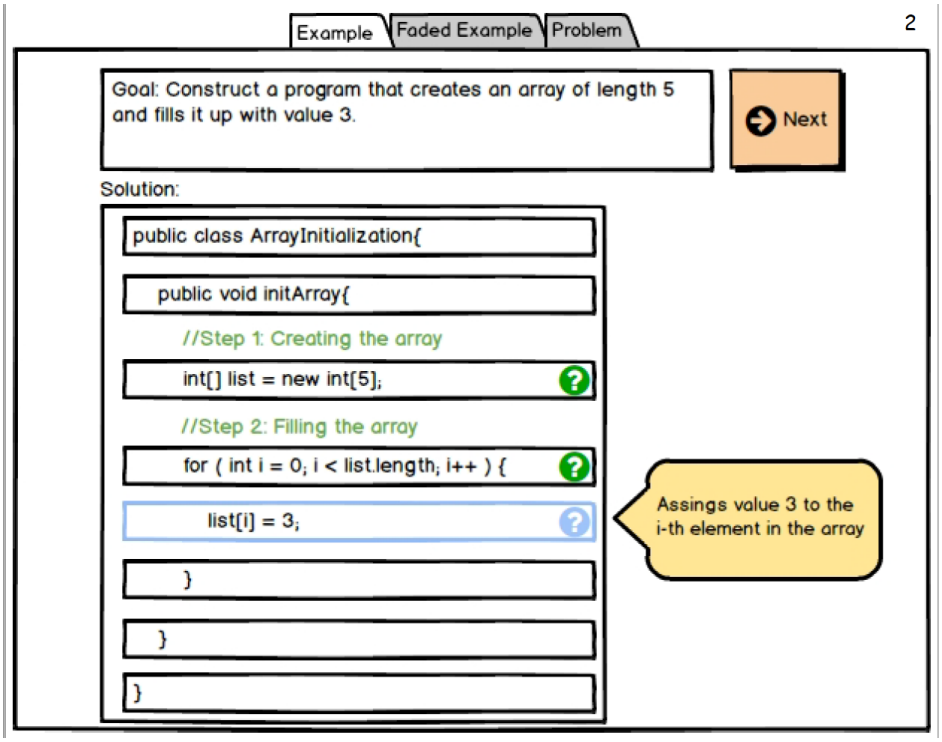
About the current Mockup:

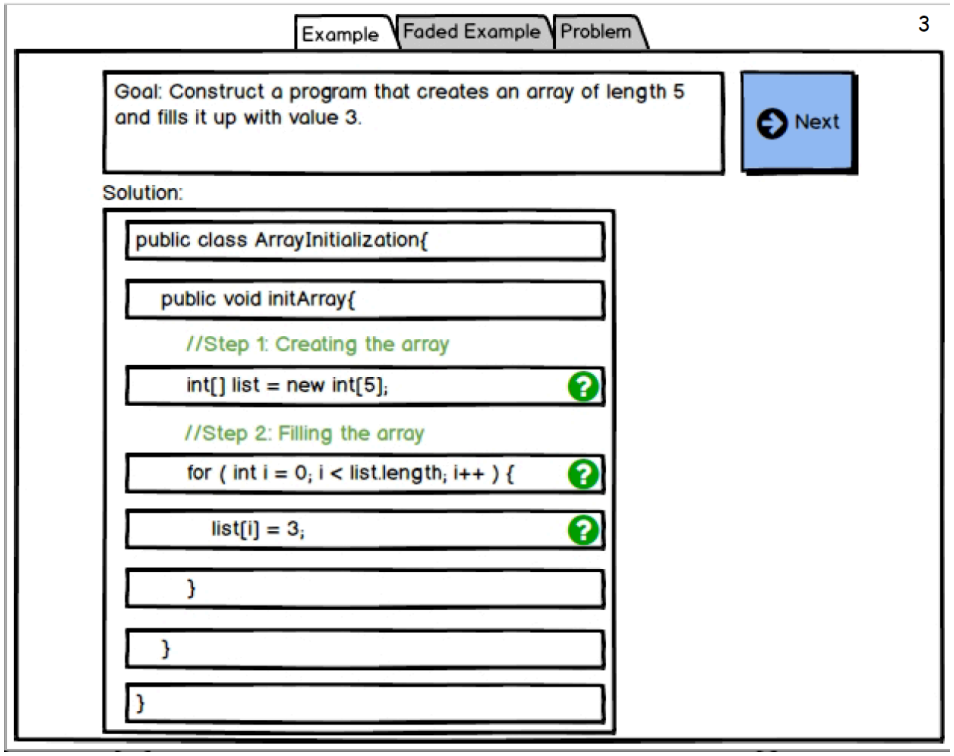
The main motivation for major change in the design prototype is for following the design heuristic to be consistent. Lack of it was the major flaw in the previous mockup. We wanted the design to be self explanatory but the previous one was more confusing as per the feedback received from peers after being introduced to it. The changes and differences from the previous version to this one is given below:

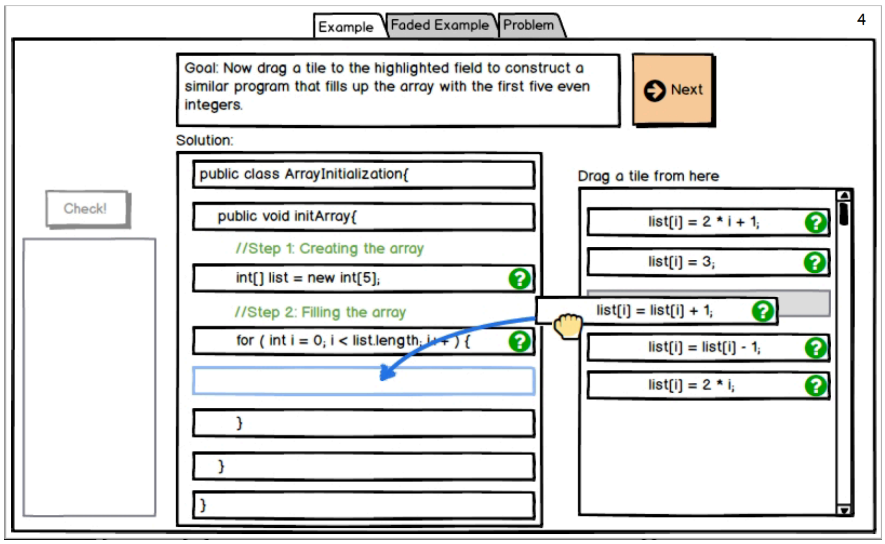
- The mockup no longer has the concepts in Java listed on the left side.
- All the three tabs have the functionality in terms of draggable elements only.
- The purpose of the tabs are also changed for the sake of clarity.
    - The Example tab has multiple goals which can be changed by clicking Next
    - The Faded example and Problem tabs have just one goal
- Help is implemented in different ways on all the tabs
    - The help is in the form of explanations on the code fragment tiles themselves in Example tab.
    - There is separate Help and Hint button on the other two tabs due to their nature of functionality (These buttons are in the place of Next button for consistency in look).
- Placement of output box is changed.
    - It is now on the left side of the interface on all three tabs.
    - It shows the whether the tiles dragged in place are the correct solution for the corresponding goals or if they are wrong, it gives the output of corresponding to the tile placed so the user will analyze and switch the tile with the correct one.
    - Also, the output is displayed as it is but only when the button called 'Check' is clicked. This button is not supposed to be shown when the user freshly opens the interface where it by defaults the Example tab. Only when he clicks the button Next in the Example tab will it be shown as it is when the user actually sees the option tiles on the right side to be dragged on to the code area.
    - Check not enabled unless the user first drags and drops the tile for answering the goal.
- Visual affordances:
    - The buttons gets highlighted when hovered over them.
    - The green colored help is also hover enabled to display the corresponding help text.
    - Color highlighting for empty slot in code area to place the drop the tile from options area.
    - Correct and wrong selections are displayed with green and red colors respectively on the tiles and in the output area when Check is clicked.
- Has sub goals in the form of comments to help user understand the part of the code and what it does.

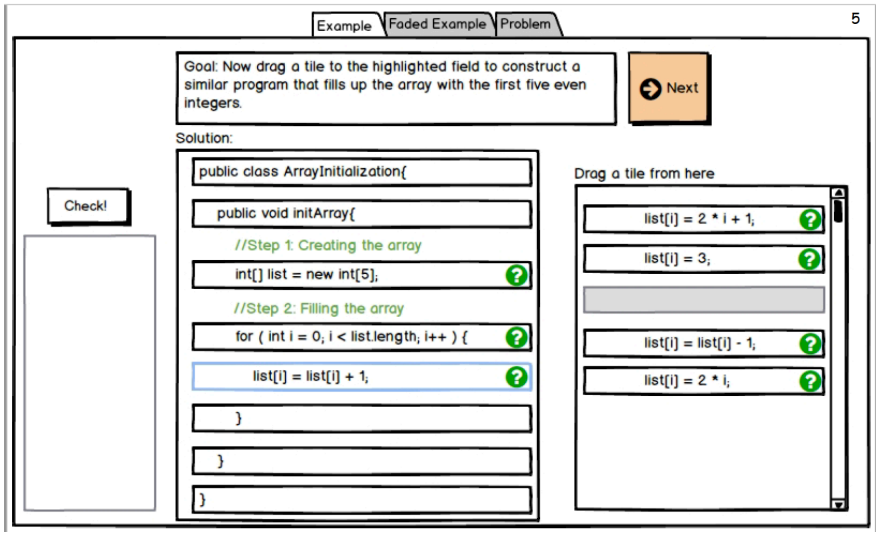
Task Analysis:

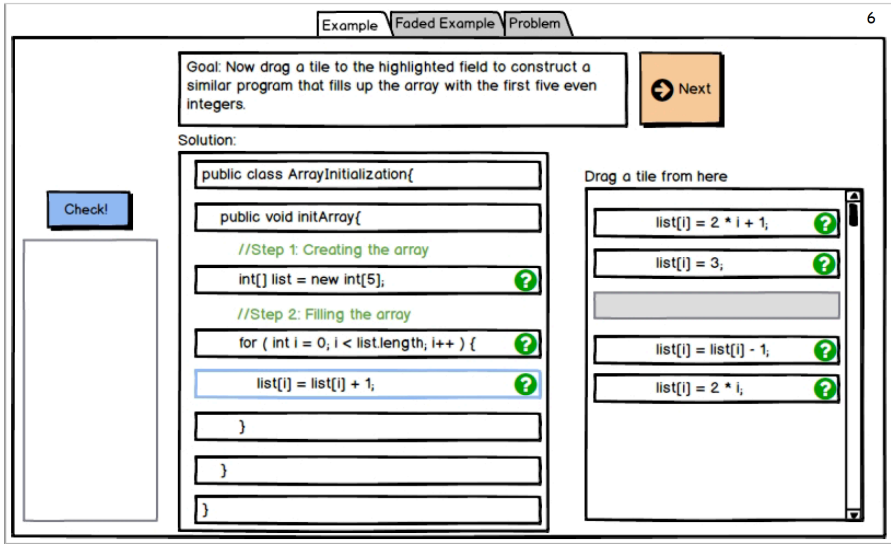
The screenshots below provide the 'labeled' actions and the results of those actions in a sequential manner.

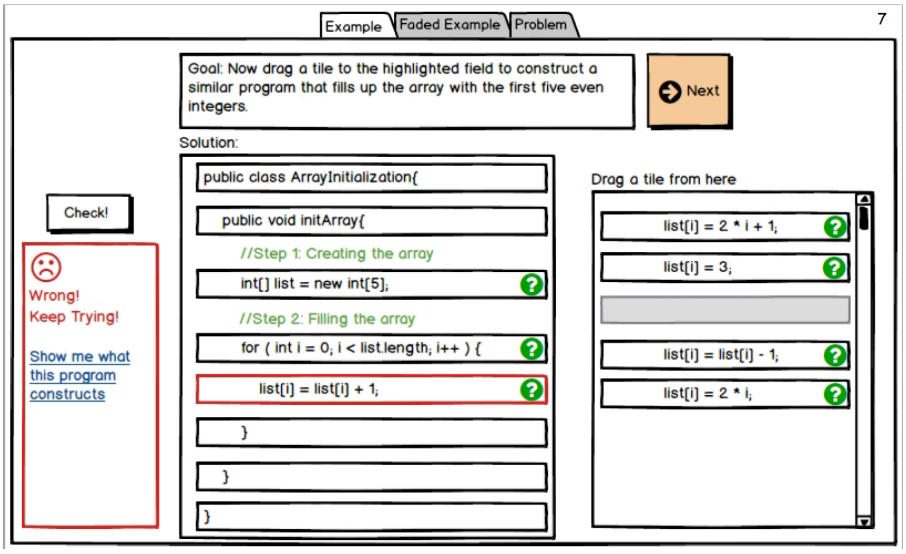


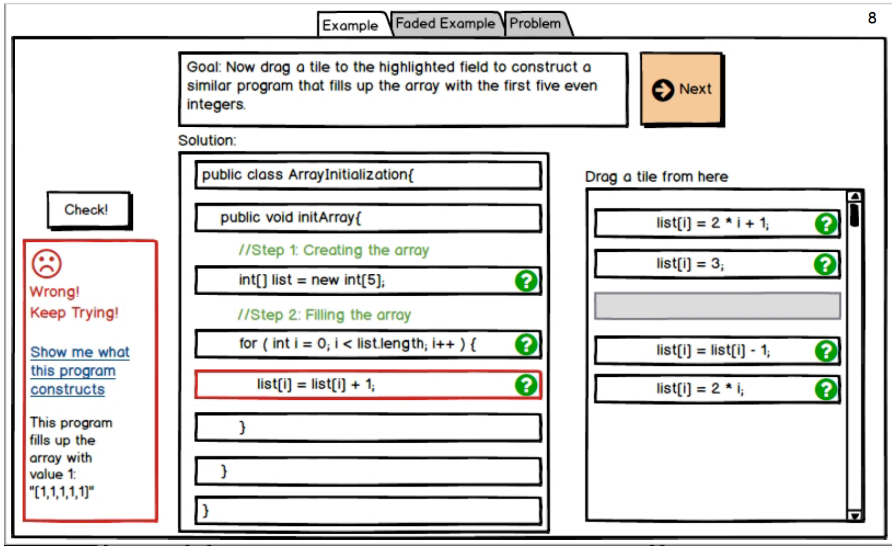


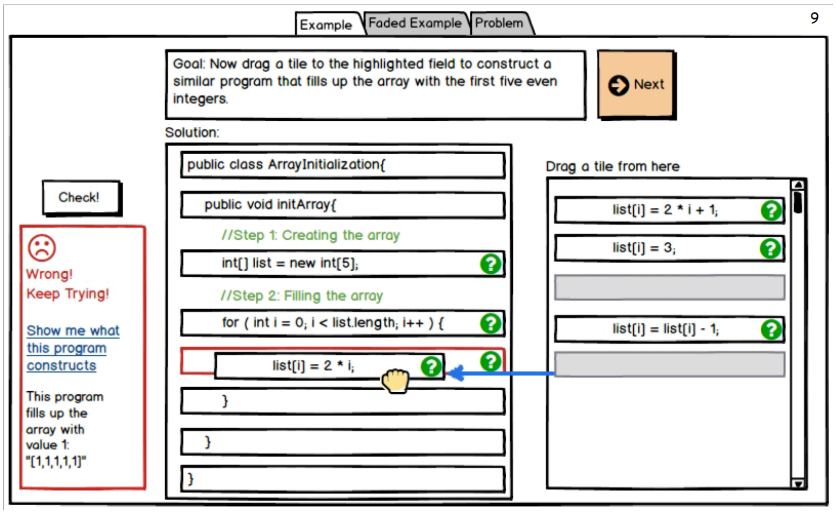

Example
Faded Example
Problem
4
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

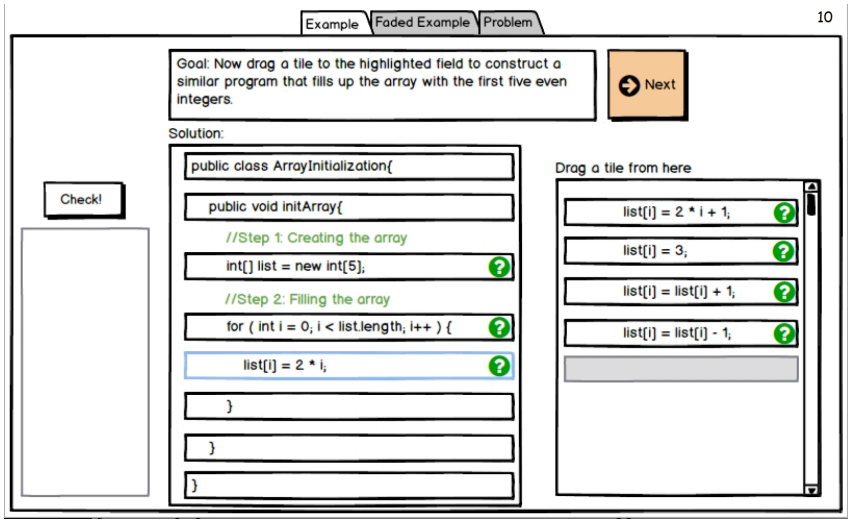

Example
Faded Example
Problem
5
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

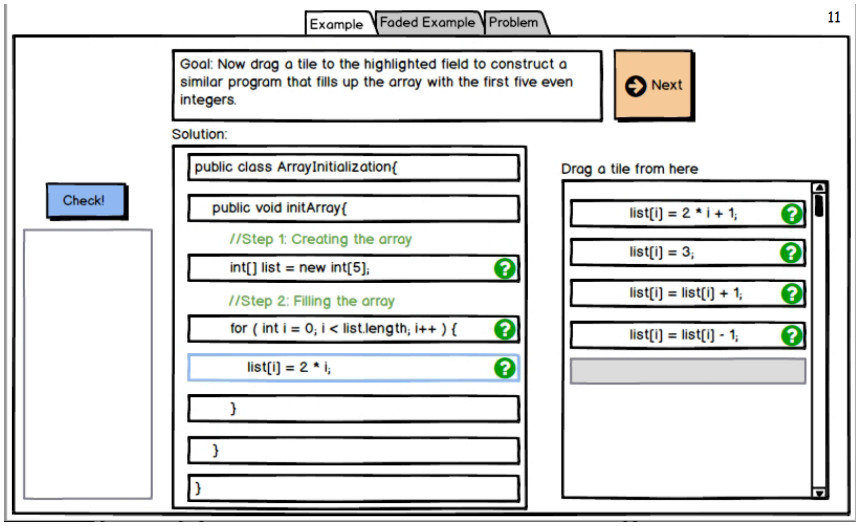

Example
Faded Example
Problem
6
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

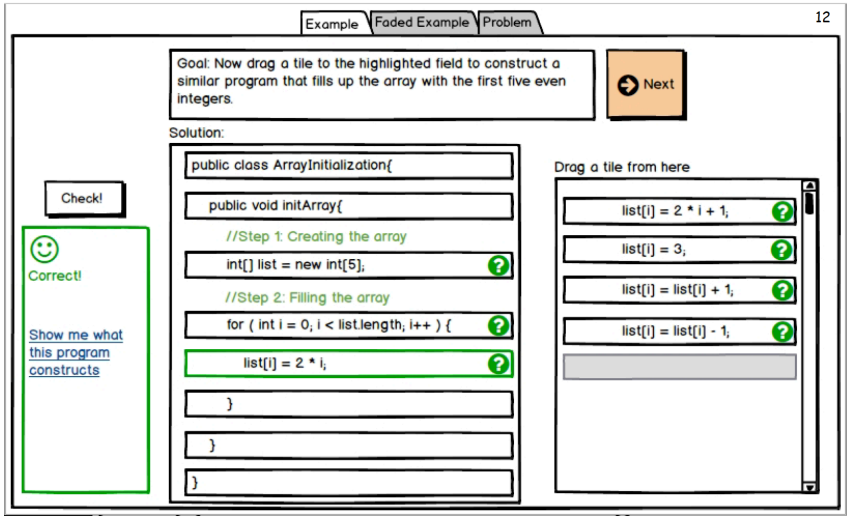

7
Example
Faded Example
Problem
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Check!
Wrong!
Keep Trying!
Show me what this program constructs
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

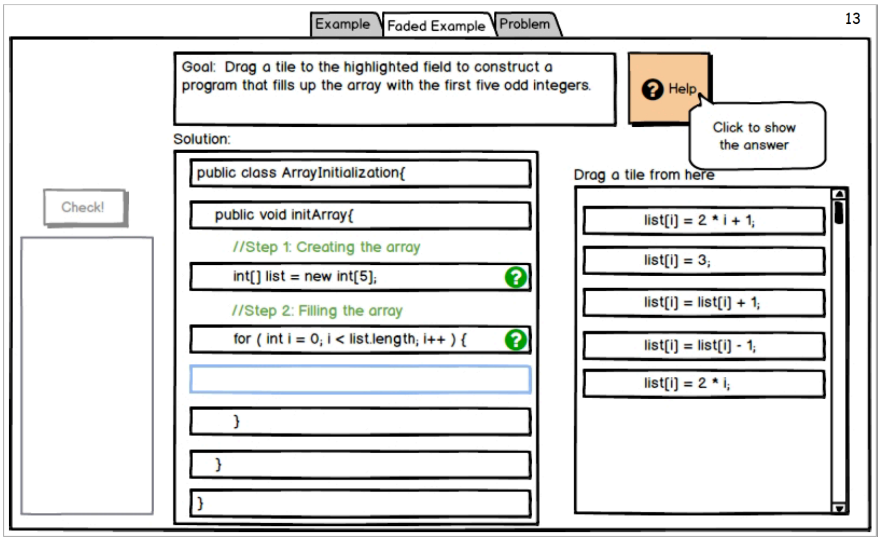

8
Example
Faded Example
Problem
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Check!
Wrong!
Keep Trying!
Show me what this program constructs
This program fills up the array with value 1: "[1,1,1,1,1]"
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

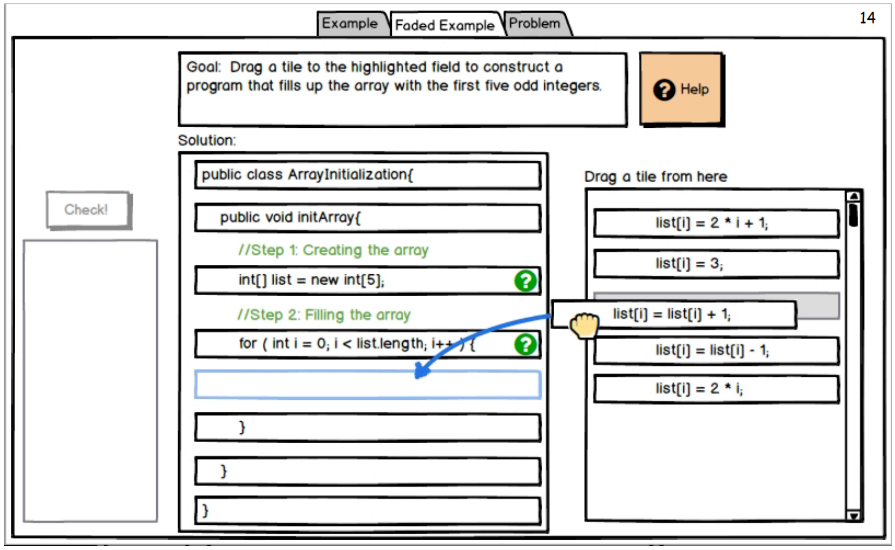

9
Example
Faded Example
Problem
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Check!
Wrong!
Keep Trying!
Show me what this program constructs
This program fills up the array with value 1: "[1,1,1,1,1]"
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;

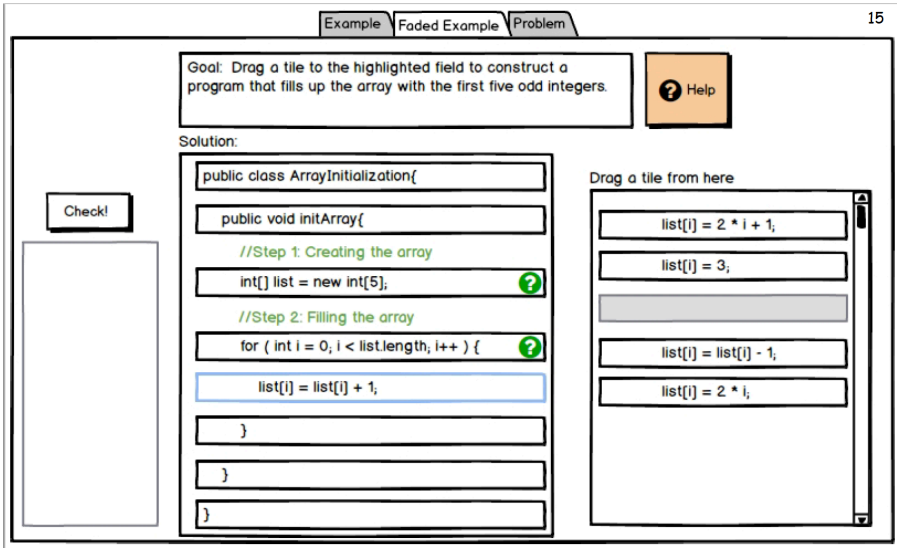

10
Example
Faded Example
Problem
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;

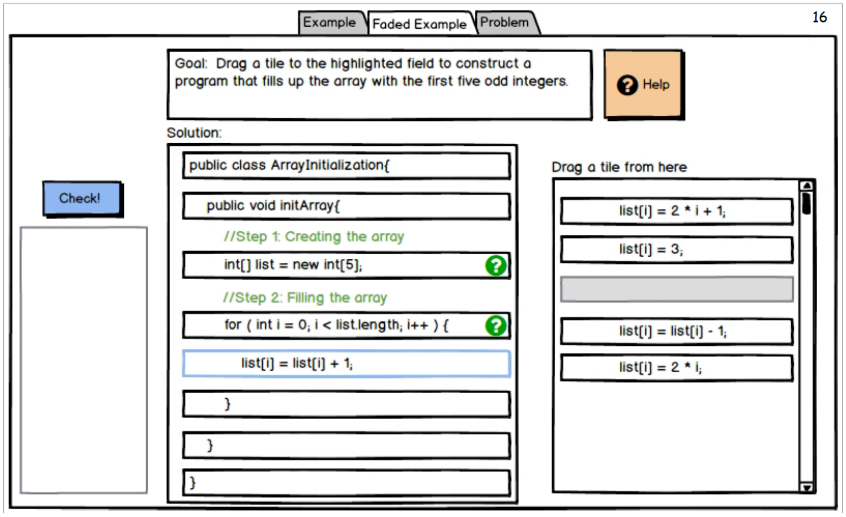

11
Example
Faded Example
Problem
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;

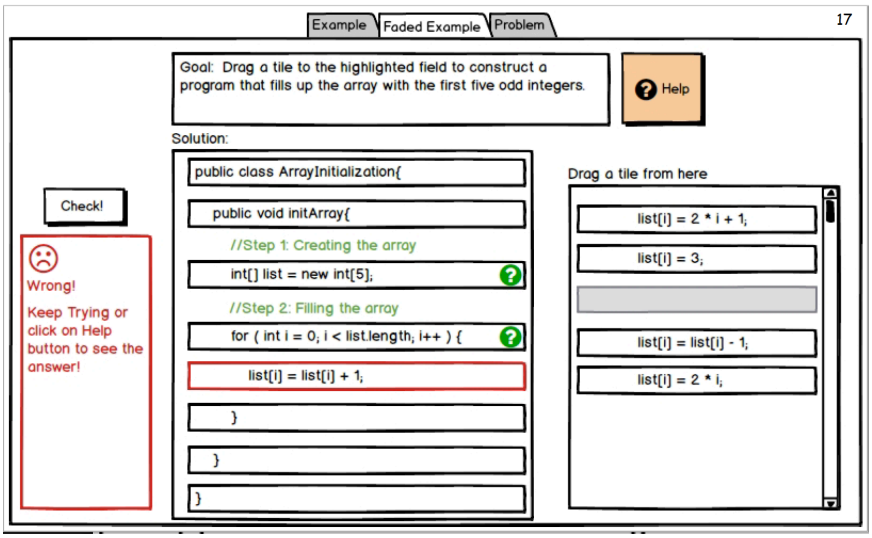

12
Example
Faded Example
Problem
Goal: Now drag a tile to the highlighted field to construct a similar program that fills up the array with the first five even integers.
Next
Solution:
Check!
Correct!
Show me what this program constructs
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;

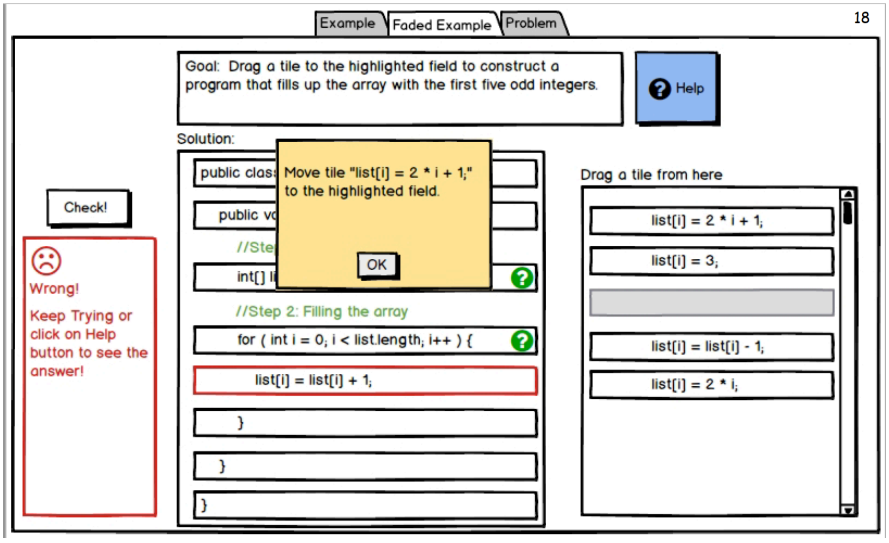

Example
Faded Example
Problem
13
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Click to show the answer
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

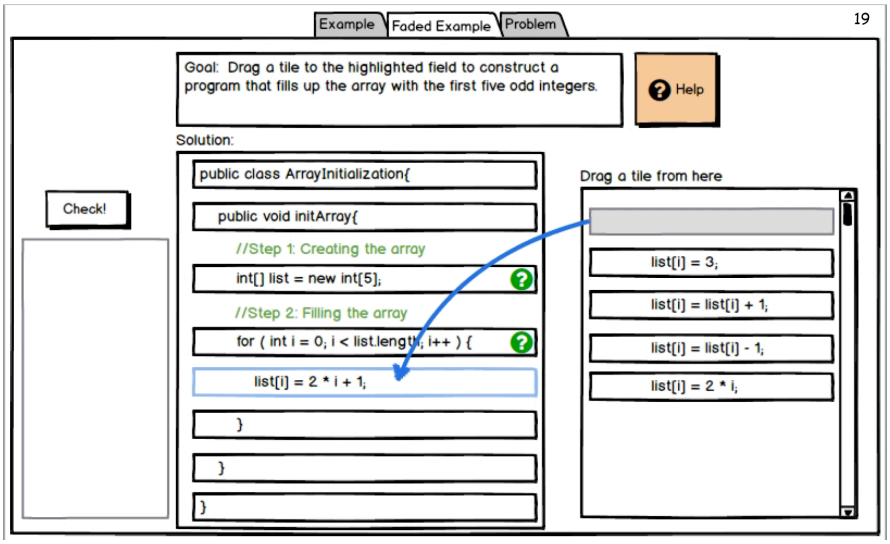

Example
Faded Example
Problem
14
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

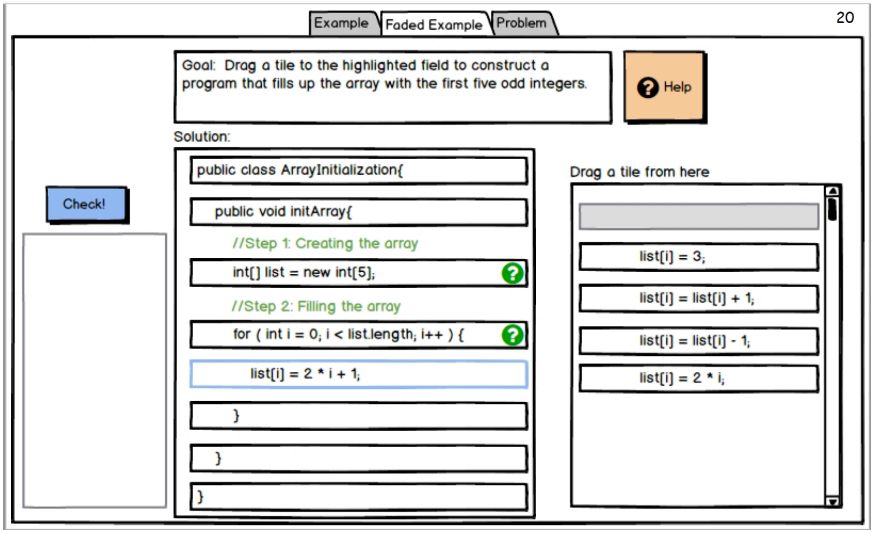

Example
Faded Example
Problem
15
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

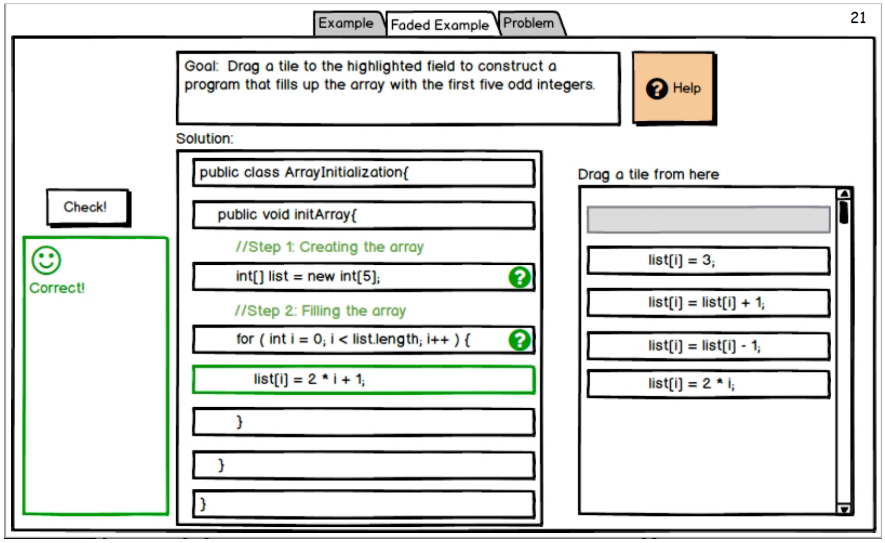

16
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

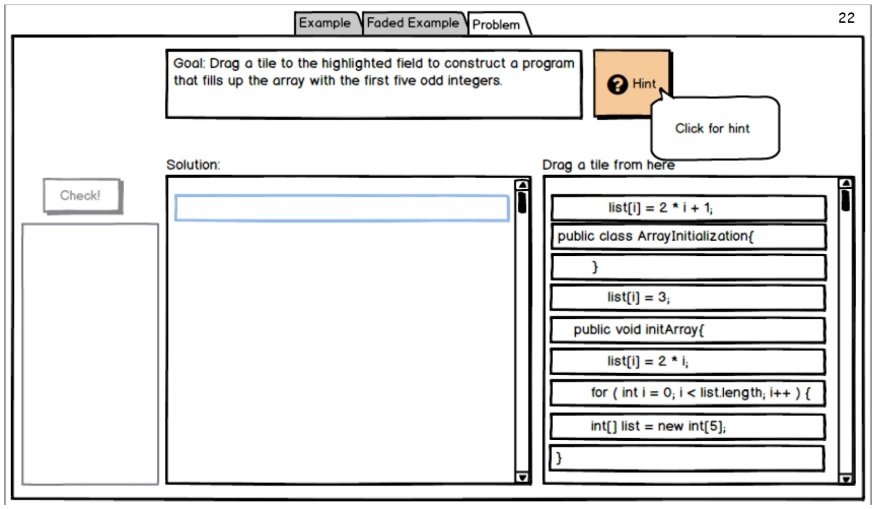

17
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
Wrong!
Keep Trying or click on Help button to see the answer!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

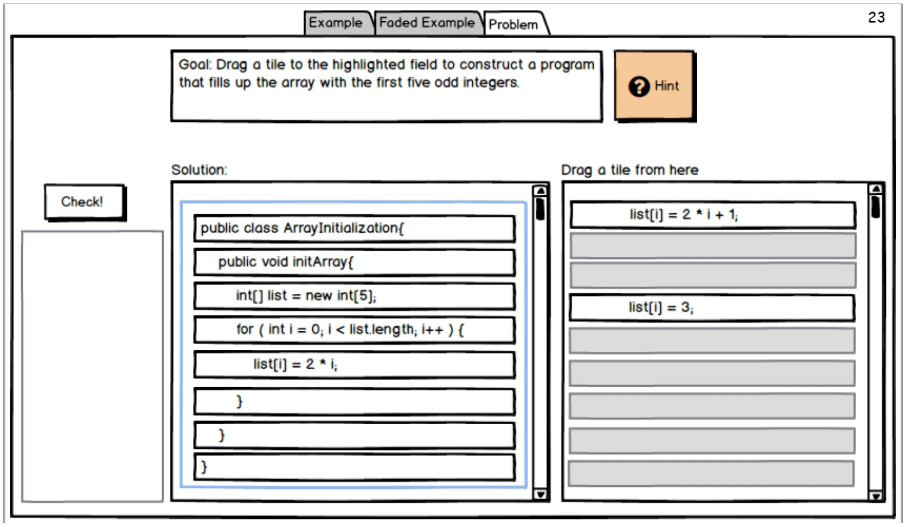

18
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Move tile "list[i] = 2 * i + 1;" to the highlighted field.
OK
Check!
Wrong!
Keep Trying or click on Help button to see the answer!
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = list[i] + 1;
}
}
}
Drag a tile from here
list[i] = 2 * i + 1;
list[i] = 3;
list[i] = list[i] - 1;
list[i] = 2 * i;

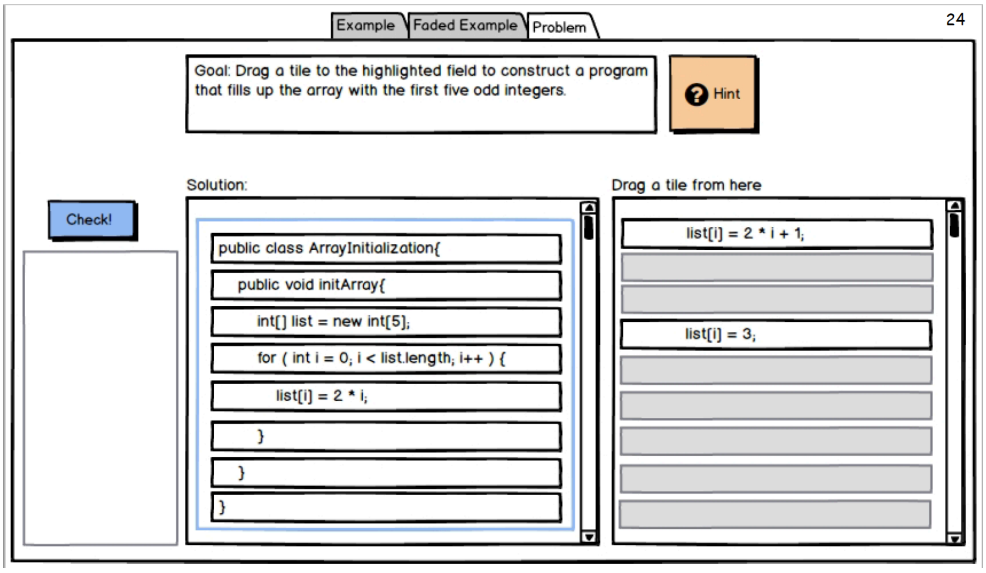

19
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
Drag a tile from here
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

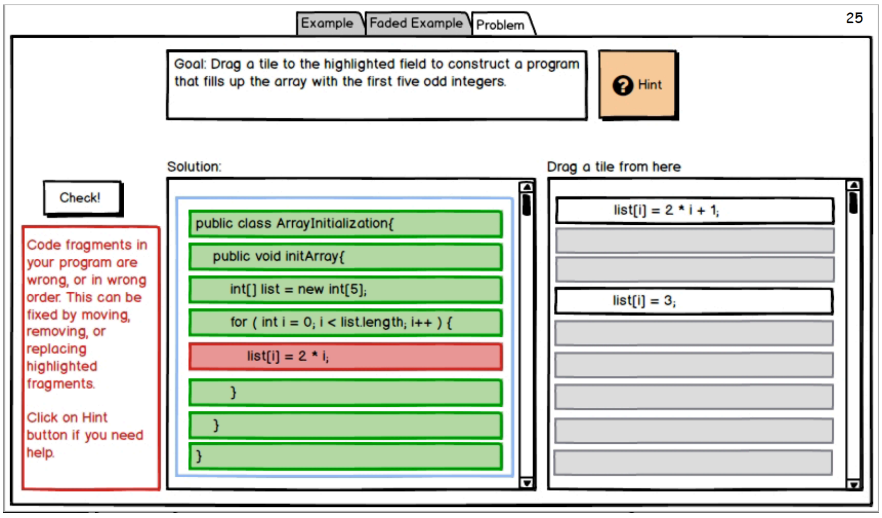

20
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
Drag a tile from here
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

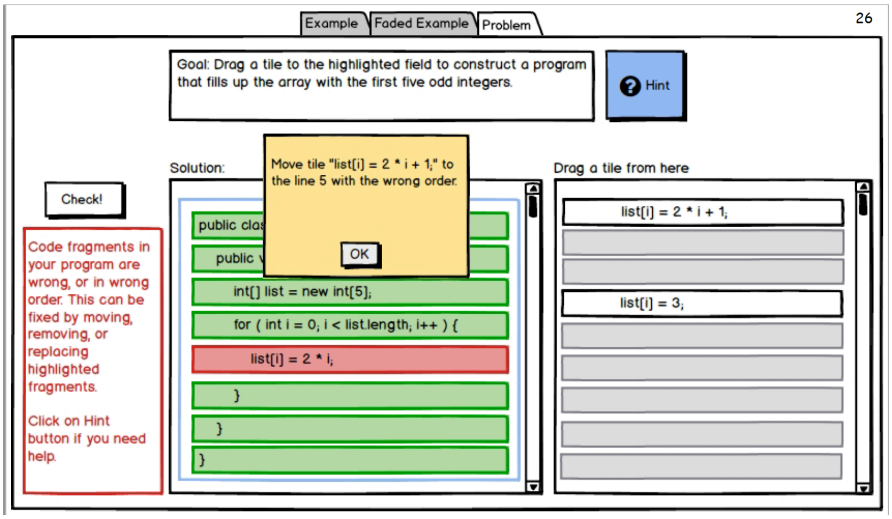

21
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Help
Solution:
Check!
Correct!
public class ArrayInitialization{
public void initArray{
//Step 1: Creating the array
int[] list = new int[5];
//Step 2: Filling the array
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
Drag a tile from here
list[i] = 3;
list[i] = list[i] + 1;
list[i] = list[i] - 1;
list[i] = 2 * i;

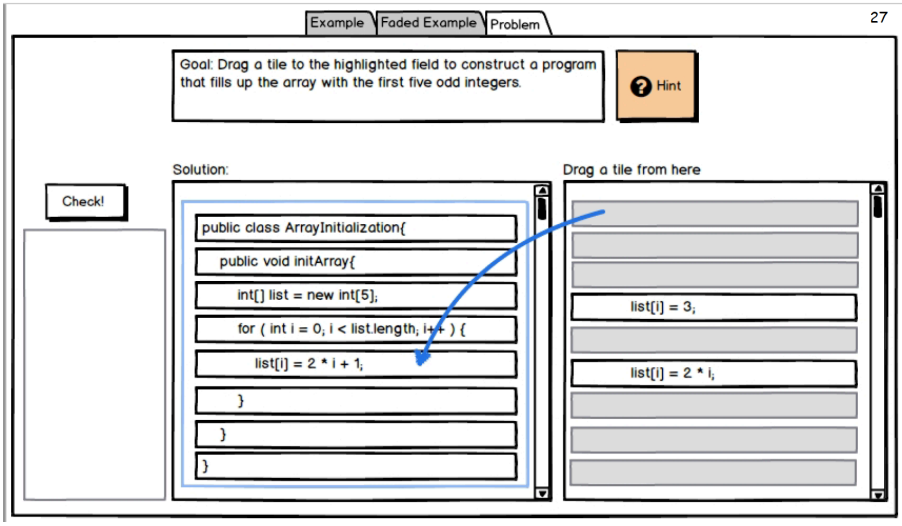

22
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Click for hint
Solution:
Drag a tile from here
Check!
list[i] = 2 * i + 1;
public class ArrayInitialization{
}
list[i] = 3;
public void initArray{
list[i] = 2 * i;
for (int i = 0; i < list.length; i++) {
int[] list = new int[5];
}

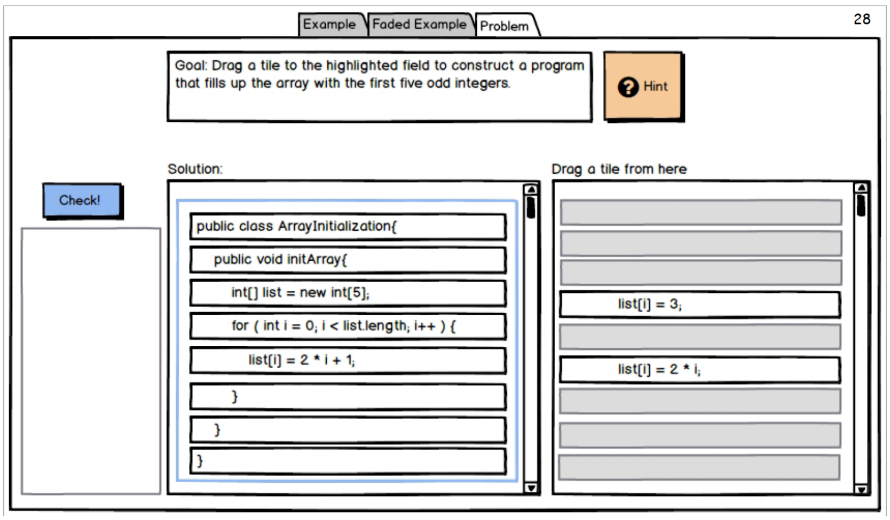

23
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;

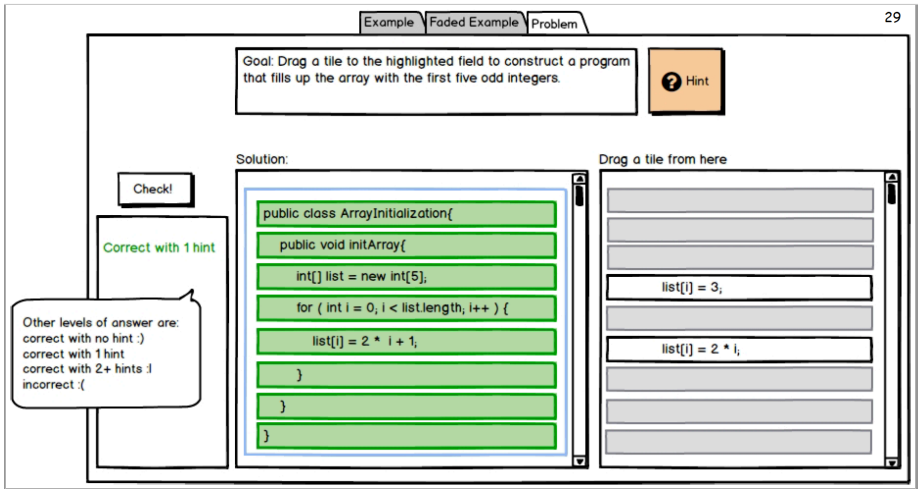

24
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;


25
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
Code fragments in your program are wrong, or in wrong order. This can be fixed by moving, removing, or replacing highlighted fragments.
Click on Hint button if you need help.
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
}
list[i] = 2 * i + 1;
list[i] = 3;


26
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Move tile "list[i] = 2 * i + 1;" to the line 5 with the wrong order.
OK
Drag a tile from here
Check!
Code fragments in your program are wrong, or in wrong order. This can be fixed by moving, removing, or replacing highlighted fragments.
Click on Hint button if you need help.
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i;
}
}
list[i] = 2 * i + 1;
list[i] = 3;


27
Example
Faded Example
Problem
Goal: Drag a tile to the highlighted field to construct a program that fills up the array with the first five odd integers.
Hint
Solution:
Drag a tile from here
Check!
public class ArrayInitialization{
public void initArray{
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
list[i] = 2 * i + 1;
}
}
}
list[i] = 3;
list[i] = 2 * i;





Summary of the task analysis of this new mockup:

Example tab:

This tab now has the following user tasks:

- To see the goal at the top and the solution at a fresh load
- To click Next and make the interface quiz-like with feedback, code and option areas.
- To answer a given goal by dragging and dropping the tile in code area
- To be able to change the answer by dragging and dropping a different tile
- To seek help at code fragment level of both code area and fragment area
- To get feedback clicking Check
- To get the currently place tile's output in the feedback area
- To change goals by clicking Next and do all the tasks listed above

Faded Example Tab:

This tab now has the following user tasks:

- To see the goal at the top and quiz-like interface with feedback, code and option areas.

- To answer a given goal by dragging and dropping the tile in code area
- To be able to change the answer by dragging and dropping a different tile
- To seek help at code fragment level at the code area but not in the options area
- The presence of global Help to help user choose right option
- To get feedback clicking Check
- To get the currently place tile's output in the feedback area

Problem Tab:

This tab now has the following user tasks:

- To see the goal at the top and quiz-like interface with feedback, code and option areas.
- To answer a given goal by dragging and dropping multiple tiles in code area which is empty on fresh load
- To get feedback clicking Check
- To get the currently place tile's output in the feedback area
- The presence of global Hint to help arrange the tiles in correct order to answer the goal
- To be able to put back multiple tiles one by one by drag and drop
- To be able to change the tiles in code area by drag and drop
- To be able to different hints after different tries
- To be able to change the feedback level based on number of tries

It can be seen that the level of difficulty in the tasks and the kind of help a user receives differs with each tab. This is done to make the user feel challenged to try to solve the goals posed to him in different ways.

Timeline corresponding to the design and task analysis of mockup 2:

Week 5:

We met the client on 1st April 2017 and made the major changes to the prototype design. Summary of them is given below:

- Example Tab:
    - Addition of a Next button instead of 'dropdown' or 'press to see next goal' button.
    - Addition of a 'Show me what this program does' button that will show user what output is expected.
    - Addition of a 'Check' button, user can now check his the outcome of the changes he made at anytime by clicking the button.
- Faded Example Tab:
    - Replacing the unfade button with Help button.
    - Addition of 'check' button
- Problem Tab:
    - Addition of Hint button
    - Addition of a check button
    - Provision of an interactive feedback and points system where user gets feedback like correct with no hint, correct with 1 hint, correct with 2+ hints and incorrect answer.

Evaluation of Mockup 2

User study 1 [04/04/2017]:

User Interview:

1. **How long has the user been programming?**

Three courses long

2. **What programming languages is the user familiar with?**

C++, Java and SQL

3. **What does user like in the KT mastery grid platform?**

User likes the explanation part where if she presses the green checkbox, an explanation gets displayed

4. **What does the user like better a) Content related to area of interest like application of programming examples in business or finance areas, or b) Generic programming examples?**

User likes the idea of application of area of interest examples.

5. **What does the user like better a) Stand-alone examples, or b) Complex examples that are part of a bigger problem?**

User likes stand-alone examples as she a beginner.

6. **How many times has the user used this KT mastery grid platform?**

User has used over the semester and has noticed an increase in her quiz score after using this platform to supplement her learning endeavors.

7. **Does the user want to modify the program if given a chance and does the have any ideas?**

The user wants to be able to modify the programming examples given a chance and suggests that the current mastery grid's green checkbox explanations and animated examples be combined.

User feedback on mockup:

Example:

The question button follows the feature in the Knowledge Tree, which the user thinks it is good. Meanwhile, it has general code comment with green color, user thinks this is good because it can make user easy to understand the program structure directly.

- User also thinks the idea of enable check button after drag the tiles is reasonable, but for another option of check the program with dragged tile automatically without clicking button to check would be better because of reducing the time of clicking.
- This tab has specific goal and multiple options that promotes user to think about correct solution.

Faded Example:

The user thinks the Help button should be improved because the this tab is more challenge to user as a quiz. Therefore, the Help button is better not showing the answer to the user directly after choosing wrong answer. The Help button may can provide little or high level hint to the user, so user can consider the correct answer more. At the same time, the application may can have an option explanation to show user how to use dragging and checking.

User study 1 [04/06/2017]:

User Interview:

1. **For your recent java class, what type of example to learn is better? The good feature of good example should have? The dislike of example?**

YouTube Example, because the code in the YouTube can be explain by people. So the example should be run through with explanation in the video with more interactive. The suggestion to the KT platform is to add audio explanation. The dislike feature for the KT platform is different color because user does not know each color's meaning that will make confusing. Therefore, there needs some keys or helps.

2. If you are a designer to make a Java teaching program, how would you implement it?

I would make standard codes with basic program and multiple choices for user plugging in the choice to the existing code, and check answer.

3. How do you think the concept of example? Such as math or business concept?

The concept of example is not really important in java program learning.

4. Do you like the size of learning program is small or large?

Start from small java learning program as even or odd to go to difficult program.

User Feedback:

Example:

The explanation of multiple options should show up when user gets incorrect answer, so user can think more of the code.

When clicking question button, user is able to open another question buttons and the previous explanation will keep displaying unless user clicks button again to close it.

Comments can be more, so they can describe each step what to do.

Faded Example:

The HELP button is good that do not need to change.

Check button should not be changed to check automatically, because user may want to change the answer.

Problem:

Problem could be like quiz to test the score, making wrong answer will lose points.

- In 'faded example' tab, the current mockup is designed to show help, which is the correct answer, whenever it is clicked. User feels that this specific help shouldn't be displayed unless the user has tried to answer the goal at least once.
- Regarding the interface level help, user feels that there needs to be a generic help across tabs that says how the interface works and what is expected from a general user.
- User feels that the first time clicking the 'Help' button should display a 'hint' about the problem and not the answer itself.
- Also, in the faded-example tab, when presented with an alternative design idea of automatically displaying the correct vs wrong tiles through color coding instead of making the user click 'Check' button, the user feels that this is better than the current design of having to click 'Check' button.
- In 'Problem' tab, clicking check button after placing every single tile on the left-side is not desirable.

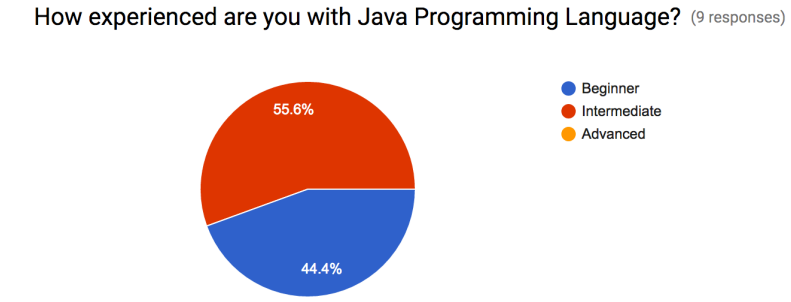
Questionnaire

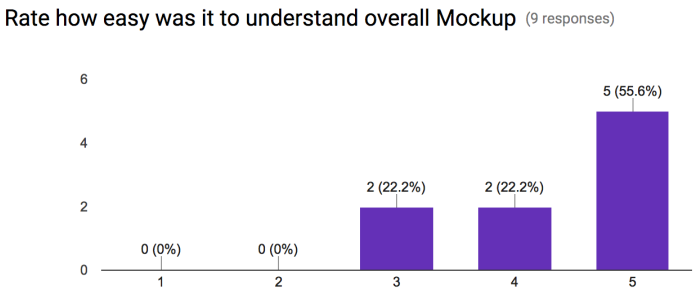
To garner the feedback of users who took part in the study as well as from peers and other general evaluators, we created a google form at the link tinyurl.com/ISD-ProtoEval. We created a set of

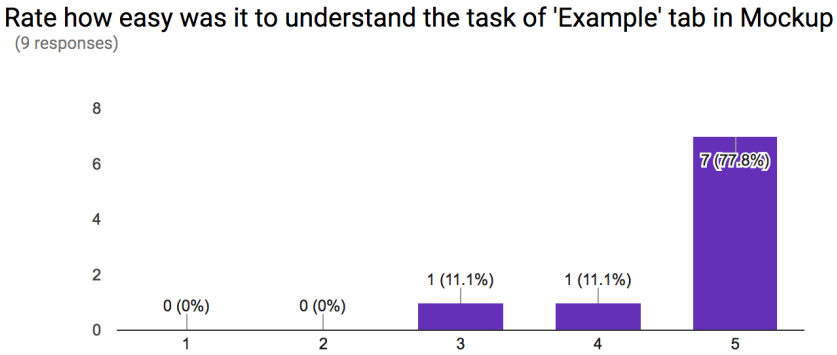
questions that we thought are suitable for gathering needed feedback from the user. The questions are:

- How experienced are you with Java Programming Language?
- Rate how easy was it to understand overall Mockup .
- Rate how easy was it to understand the task of 'Example' tab in Mockup/Mockup2
- Rate how easy was it to understand the task of 'Faded Example' tab in Mockup/Mockup2
- Rate how easy was it to understand the task of 'Problem' tab in Mockup/Mockup2
- What did you not like about the mockup ?
- How useful you feel is a platform like this to help you get better in Java Programming ?
- What are your suggestions for improvement ?

The below screenshots give the Data Analysis summary of the responses.

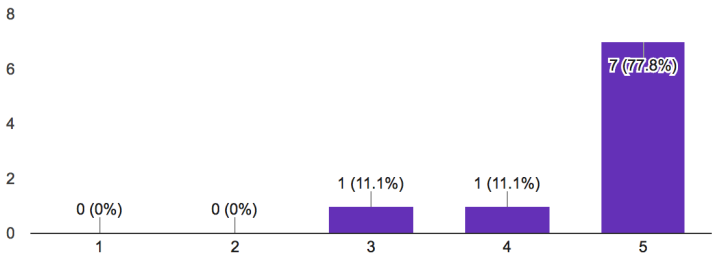




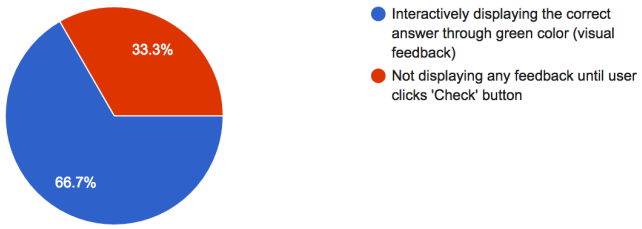


Rate how easy was it to understand the task of 'Faded Example' tab in Mockup

(9 responses)

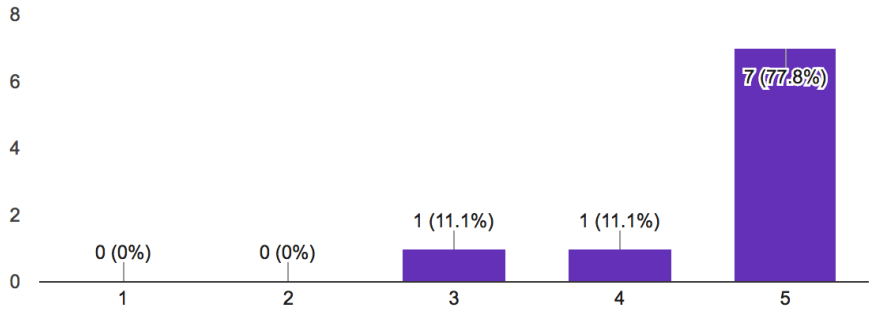


What do you prefer? (9 responses)



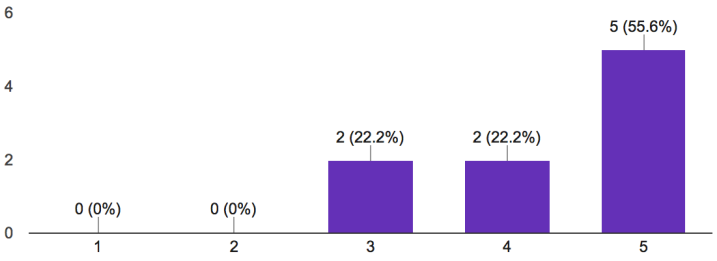
Rate how easy was it to understand the task of 'Problem' tab in Mockup

(9 responses)



How useful you feel is a platform like this to help you get better in Java Programming

(9 responses)



What did you not like about the mockup? (7 responses)

Help button
needs better instructions and/or make instructions more prominent. sometimes unclear on what to do.
Hints on the problem part.
Hint button was placed in an area where it might make me click it before I have really tried to solve the problem on my own.
Maybe this mockup is not good for users to remember the details of java code.
it is very helpful to have help and hits
I found the terminology - "Example", "Faded Example", and "Problem" - to be confusing and found it difficult to understand the distinction between the tasks involved with each.

What are your suggestions for improvement? (9 responses)

Provides little hint rather than the answer
none, everything looks good
This may benefit from an animation tutorial to show what to do for the problems. I didn't check, but does your program show compiling errors? Also, when a problem is "wrong" but still compiles without errors, does your program show this?
Eliminate the hint for the problem part and add descriptions for all the code in the examples part.
The hint counter could have a score for each exercise and reward users that use fewer hints with a higher score. Scores could also be cumulative across all topics and be used as part of a ranking system among students/classes.
I think there could be some fields that can let users type some codes.
to be available in phone devices other systems
It might be better to find terms for the sections that better explain what the user does in each section. For example, "Fill-in-the-blank" or "Class Construction".
1. I think "drag" action is not necessary. It requires more effort than clicking. 2. Your app seems to be mostly used in desktops. So I would suggest to resize the prototype to a desktop size. You can even go further to consider multi-platform performance

Timeline corresponding to evaluation of mockup 2:

Week 6:

In this week, we have worked on user studies with the client on 4th and 6th April. We get many helpful feedbacks from user studies, most of them give good comment for Interactive Problem Design. We also did an evaluation survey online, to statistic more feedback from different users.

Week 7:

In this week, we have considered how to make JSON file, and interface of the application. And we have built JSON file that includes the goal sequence of tiles. After building JSON file, we have met the client on 12th April to get suggestion and discuss our plan of application functions. Meanwhile, we used jQuery to achieve the drag and drop tiles function, and make a basic user interface of three tabs: Example, Faded Example, and Problem.

JSON File:
In our JSON file, we used Goal Option to distinguish the different goal, then depend on the different goals, we used array sequence to check the answer whether correct or incorrect. Inside the array, the sequence number is the number of each tile. Therefore, when user drag the tiles and check answer, the array of user selected tiles' sequence would compare with the correct sequence in JSON file. For our teaching code, we made all of them as tiles show in the web page, so the content of each tile also will store at JSON file. In Example and Faded Example tabs, filled tiles should not be able to drag by user, so we implement is_draggable option to define the tile is able to drag or not. Also, each tile in the Example tab has help button to comment the meaning of code, these contents are added in the JSON file marked by fragment id.

User Interface:
For our user interface, we build the form by HTML based on the Mockup design, and implement drag and drop tiles by jQuery. And we combined three tabs to generate as the whole application.

Implementation

Front-end:

For the implementation of the interface, we used HTML based on the Mockup design, and implement drag and drop tiles by jQuery. And we combined three tabs to generate as the whole Single Page Web application. The below screenshots are from after implementation.

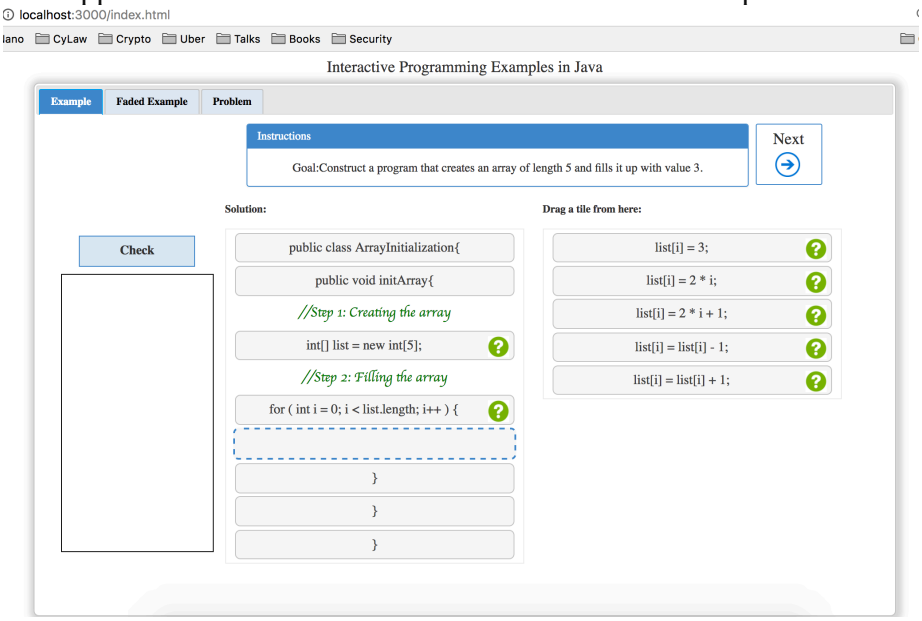


Fig: Example tab

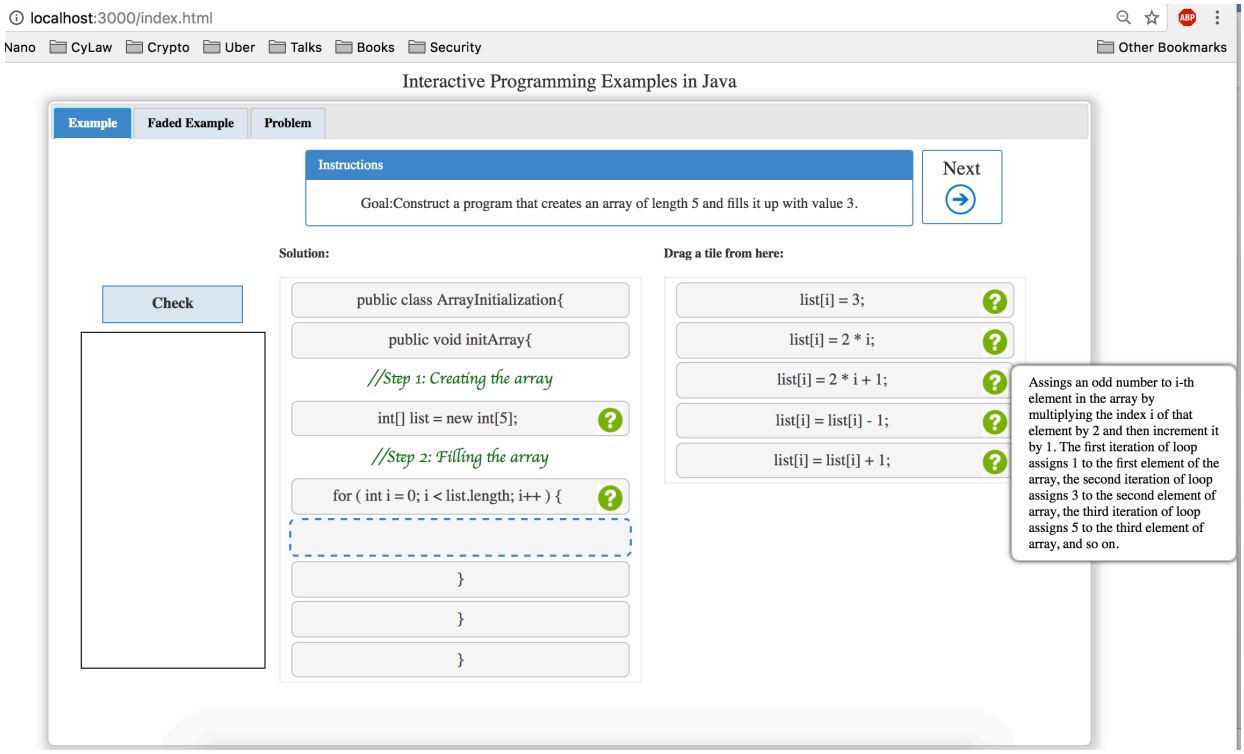


Fig: Example tab hover on help

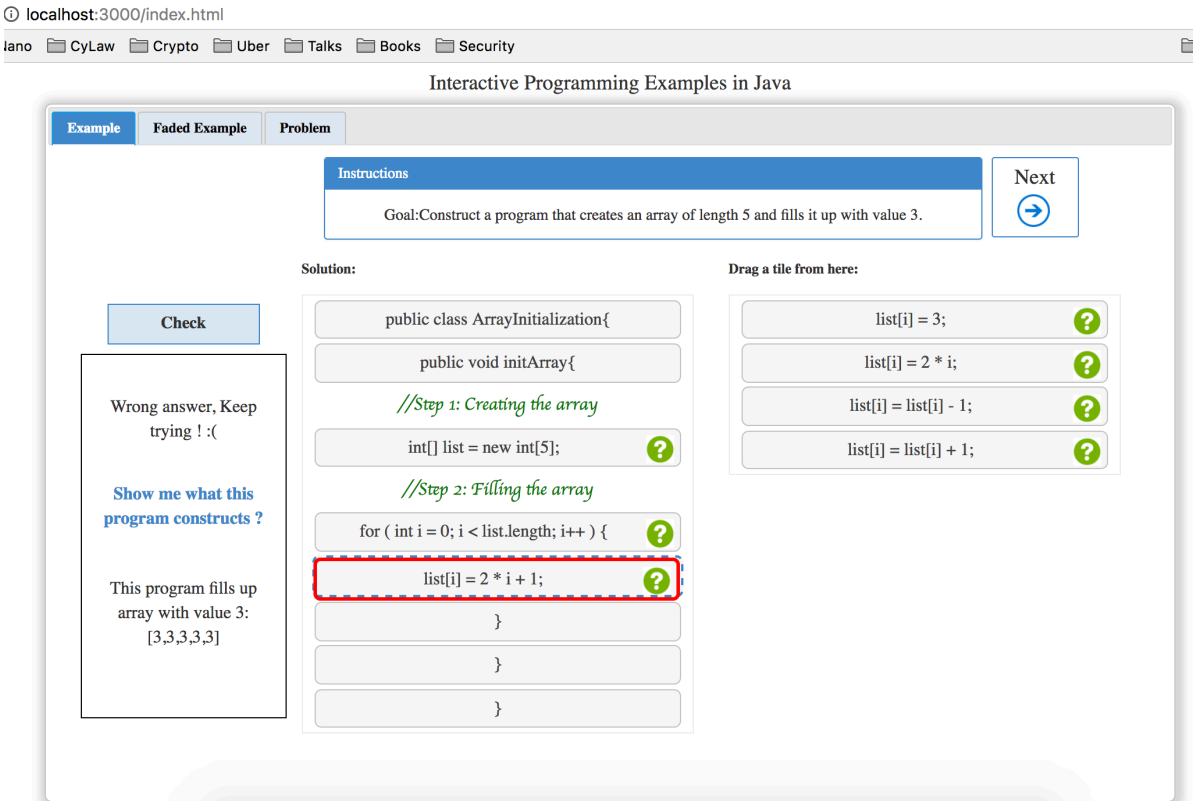


Fig: Example tab wrong answer selection

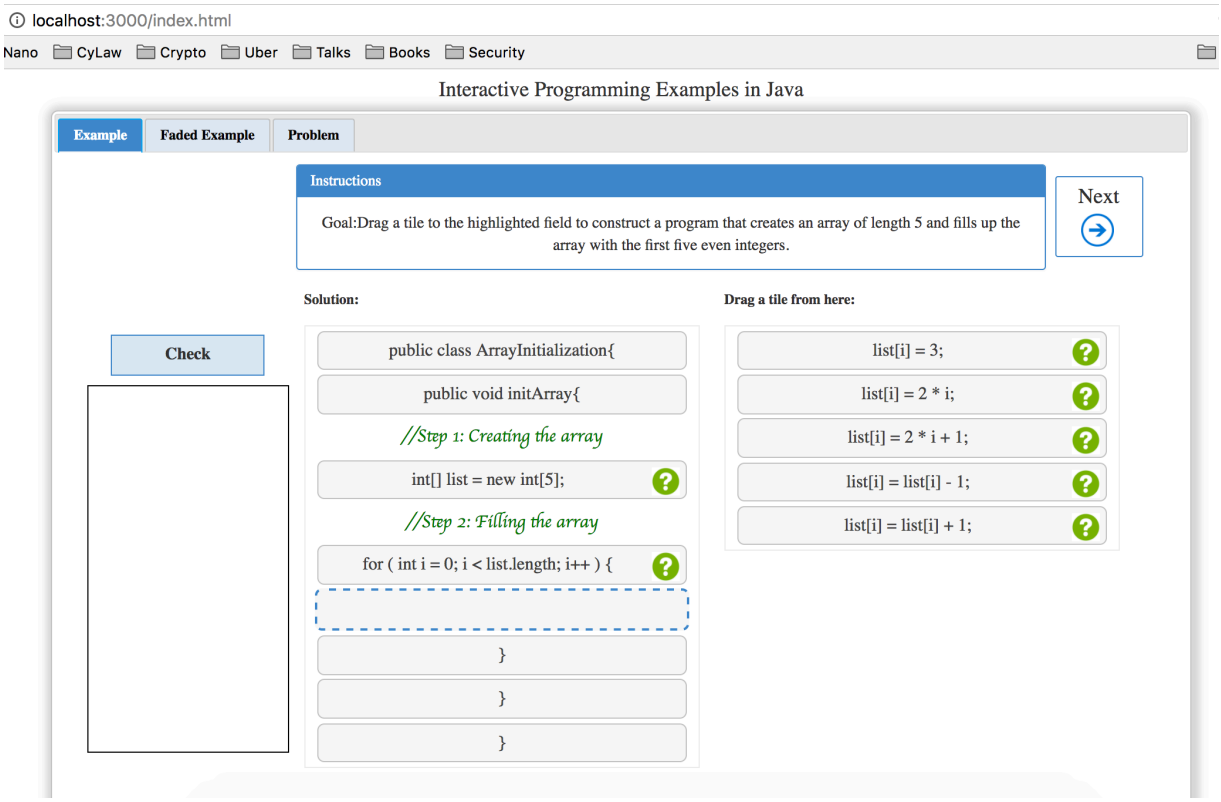


Fig: Example tab goal change

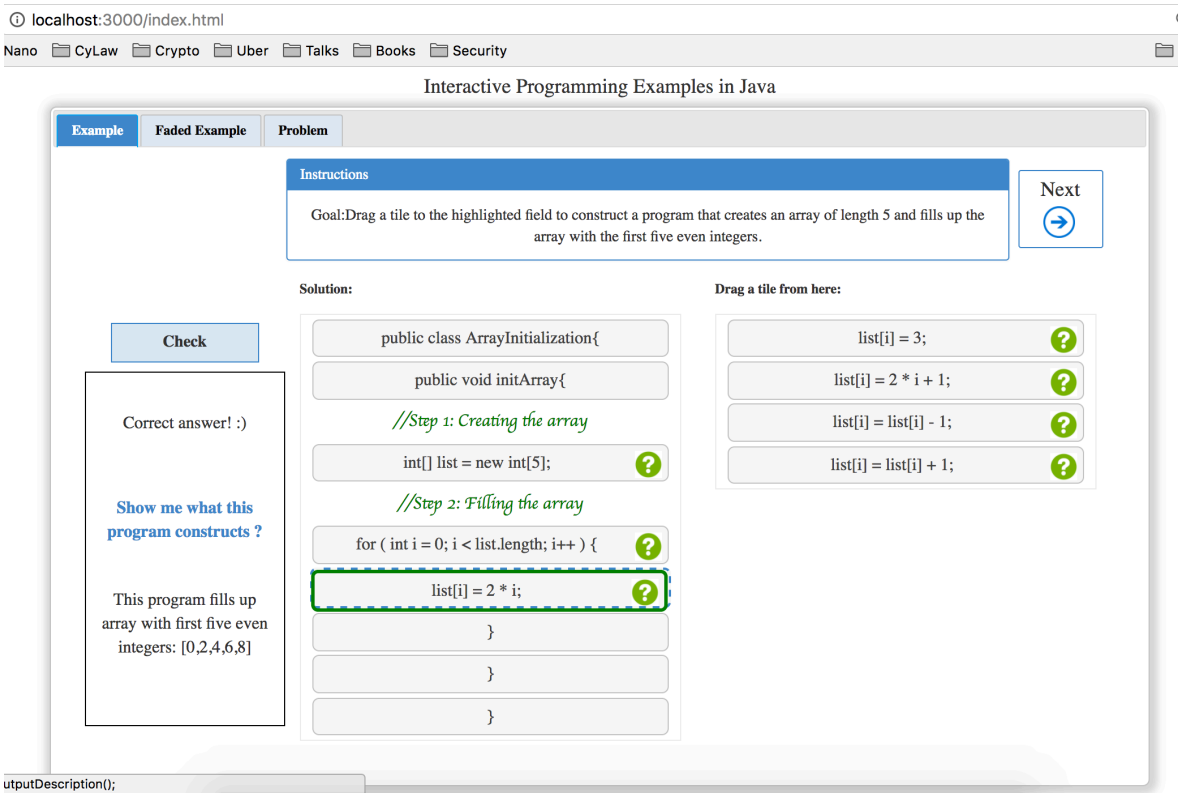


Fig: Example tab correct answer selection

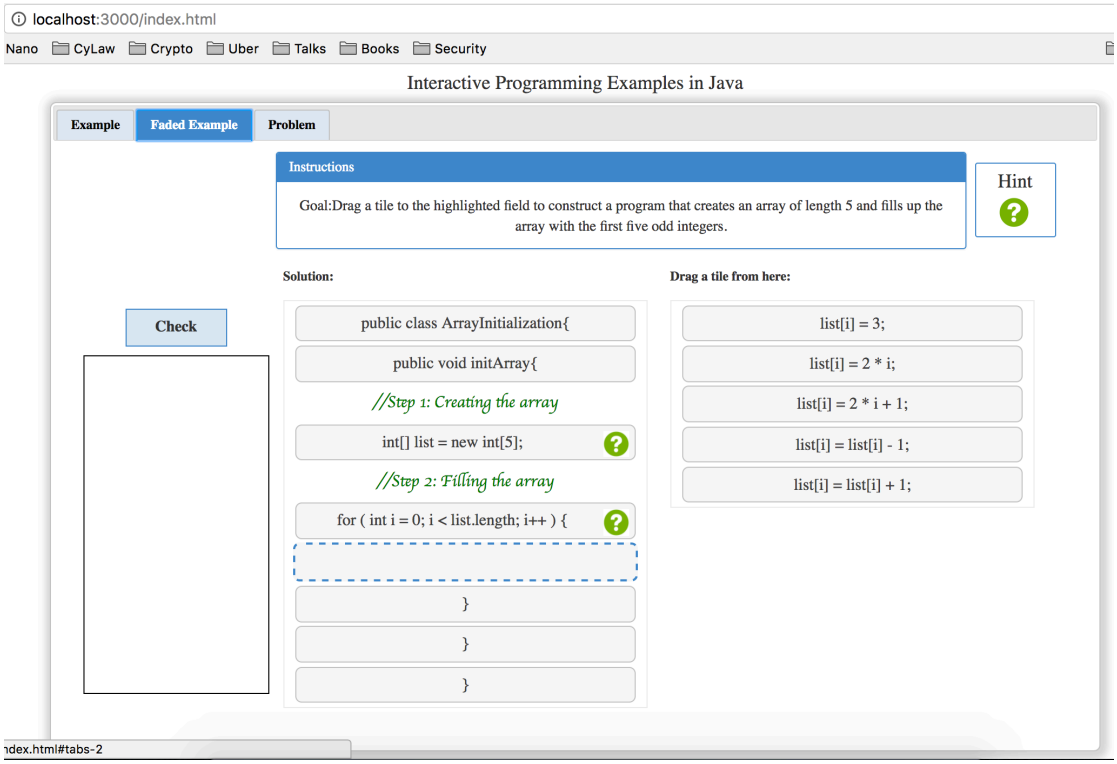


Fig: Faded example

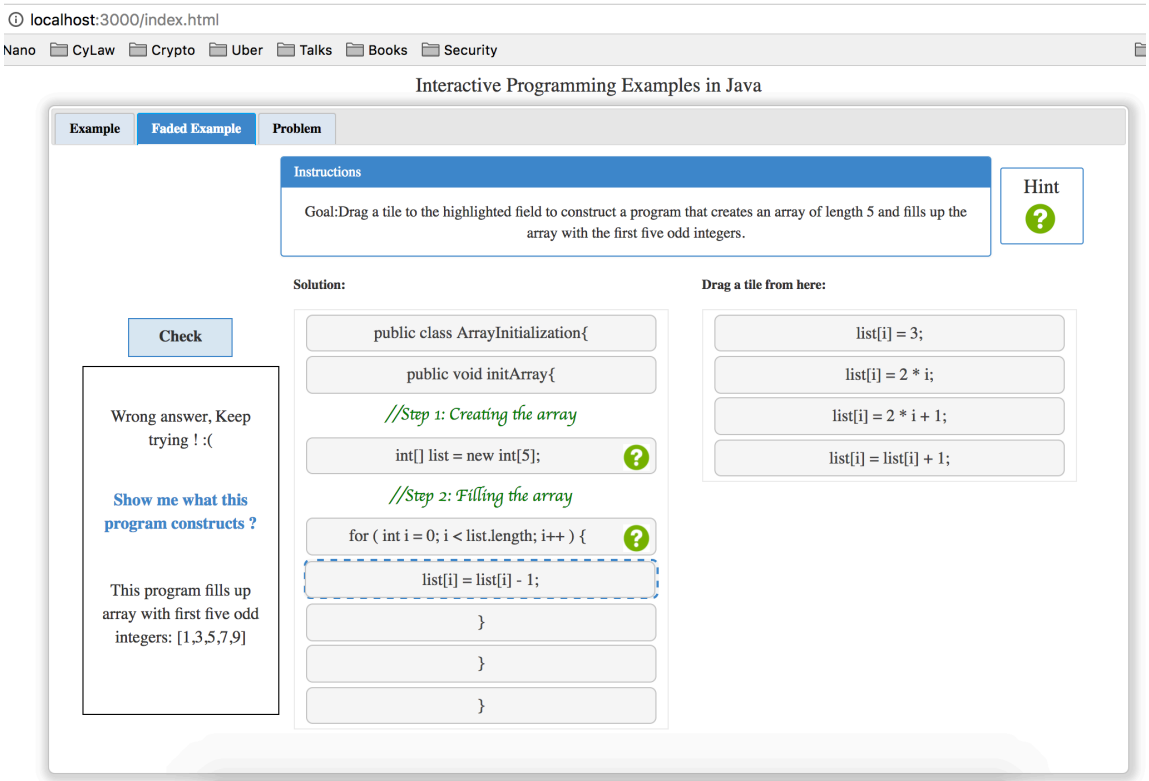


Fig: Faded example wrong tile selection

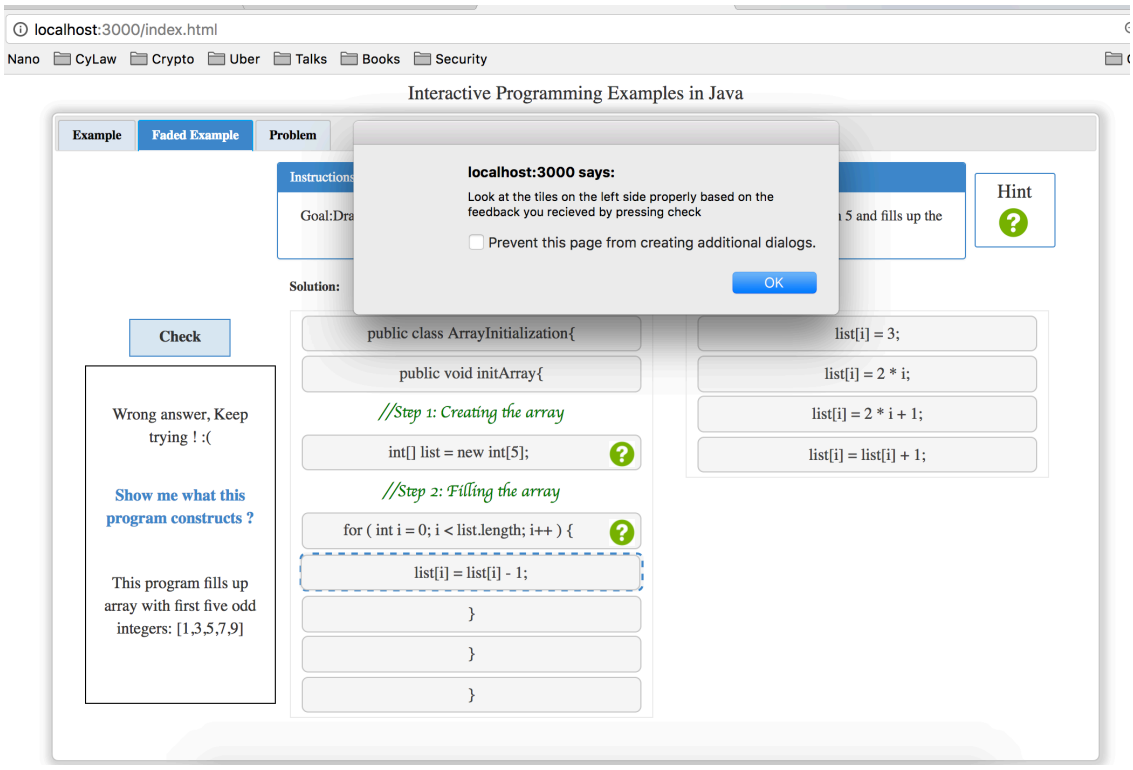


Fig: Hint for faded example

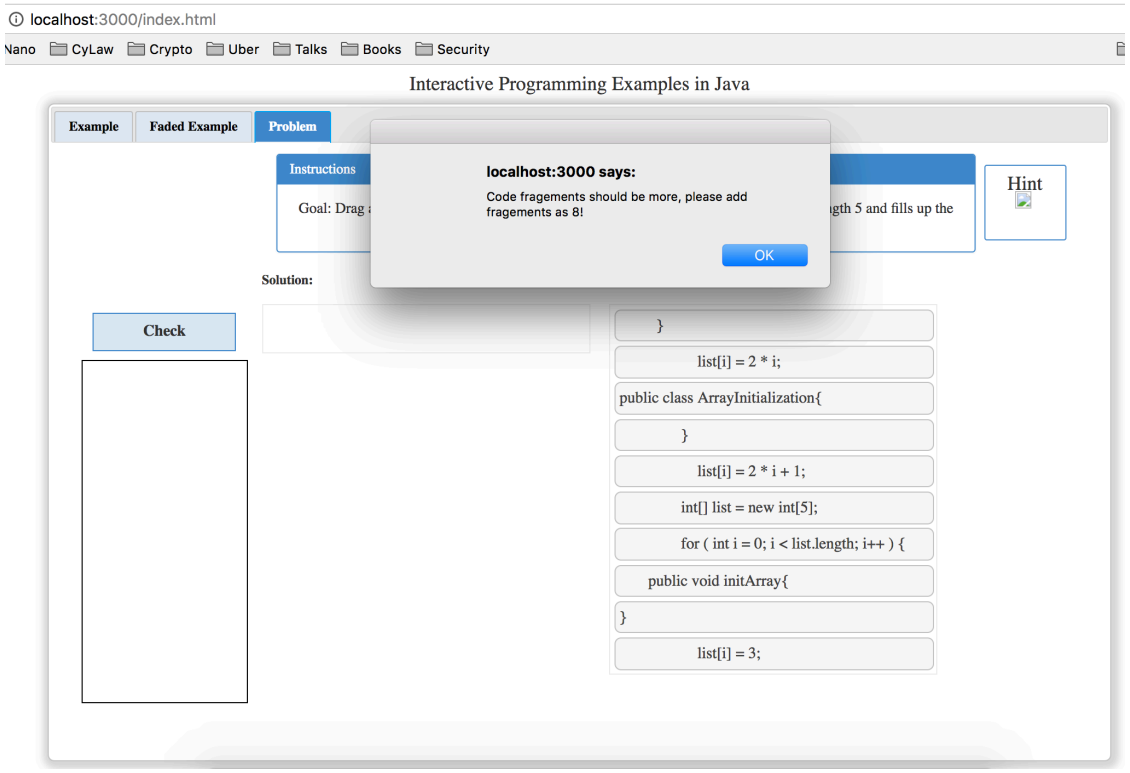


Fig: Problem tab hint

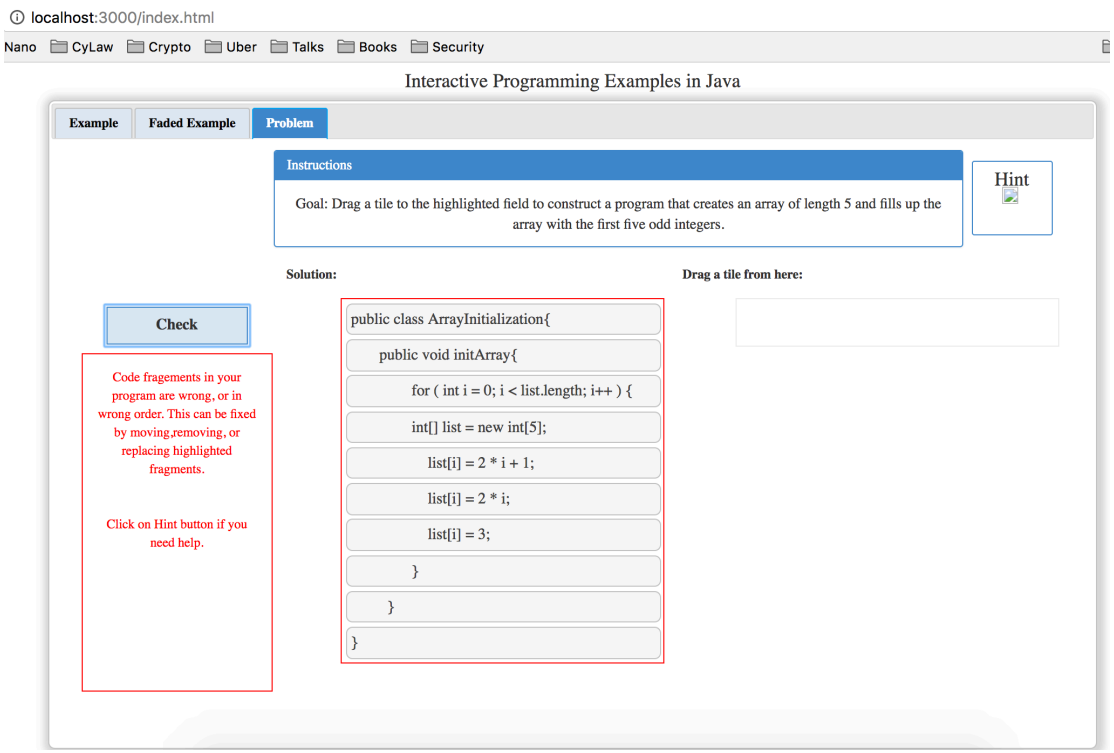


Fig: Problem tiles placement order wrong

Data format

We decide to create the elements dynamically in the front-end based on data from a JSON file rather than a separate backend. We created this below JSON structure as the master template for all the data related to the code logic and the interface elements. We have defined a correct answer sequence array that holds the sequence of fragments that correspond to the correct code. The sequence number is the id number of each tile. Therefore, when user drag the tiles and check answer, the array of user selected tiles' sequence would compare with the correct sequence in JSON file. In Example and Faded Example tabs, filled tiles should not be able to drag by user, so we implement is_draggable option to define the tile is able to drag or not. Also, each tile in the Example tab has help button to comment the meaning of code, these contents are added in the JSON file marked by fragment id.

Structure of each entity:

```
{
"Entity _id": 1,
"Entity_name": "array",
"Clusters": {
        "Cluster_id": 1,
        "Cluster_name": "Array initialization",
        "Cluster_goals": [list of goals whose structure is given belwo]
        },
"Fragment":{ [list of fragments whose structure is given below]
        }
```

}

Structure of each fragment:


```
{
"f_id": 6,
"Is_draggable": "false",
"help": {
        "has_help": "true",
        "help_desc": "A for loop for traversing the array and initializing each of its elements. The number of loop iteration is the same as the length of the list (i.e., 5)."
        },
"Goal_id": 1,
"Fragment_text": "       for ( int i = 0; i < list.length; i++ ) {",
"Is_subgoal": "false"
}
```


Structure of each goal:


```
{
"Goal_id": 1,
"Goal_name": "Construct a program that creates an array of length 5 and fills it up with value 3.",
"Is_problem": "false",
"Is_faded": "false",
"correct_ans_set": [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
"Output_desc": "This program fills up array with value 3: [3,3,3,3,3]"
}
```


Future Scope

- Levels of hint instead of just one hint in faded example tab
- Maintain attempts so we can visually show a bar or something that gets updated with less max score for each wrong attempt
- Implement the suggestions that we received on the google form
- To dynamically generate json given a java code file with extra things as comments
- To dynamically generate the hints based on the current tile order or maybe browser-level compilation and testing of code correctness.