

Step 1: Add a JButton to CreatorState

```
48 public class CreatorState {
49
50     //global variables
51     int exitCode; //run method returns exit code when returning to application loop to determine what state to enter next
52     JButton btnMatrix=new JButton("Add Matrix Question");
53     JButton btnShortAnswer = new JButton("Add Short Answer Question");
54     JButton btnMultipleChoice = new JButton("Add Multiple Choice Question");
```

At the top of the CreatorState class add your button in the style of btnMatrix or btnShortAnswer

```
JButton btnXXXX=new JButton("Add XXXX Question");
```

Step 2: Add a button event handler

```
102 //Btn that creates new Matrix Question and adds it to the test
103 btnMatrix.addActionListener(new ActionListener()
104 {
105     @Override
106     public void actionPerformed(ActionEvent e)
107     {
108         Question question = new MatrixQuestion("Solve for the unknown matrix", numQuestions, "Matrix");
109         addQuestion(question);
110     }
111 });
112
113 btnShortAnswer.addActionListener(new ActionListener()
114 {
115     @Override
116     public void actionPerformed(ActionEvent e)
117     {
118         Question question = new ShortAnswerQuestion("Fill in the short answer", numQuestions, "ShortAnswer");
119         addQuestion(question);
120     }
121 });
122
123
124
```

There will be a section a little bit below where there will be the different listeners for the buttons. These will all follow the same format. You will get some errors as your question type does not yet exist.

```
btnXXXX.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        Question question = new XXXXQuestion("description of question", numQuestions, "question name");
        addQuestion(question);
    }
});
```

Step 3: Pass button to QuestionPanelManager

Within the CreatorState class, search near the bottom for the function named “createPanel”. Near the top of that function you will see this line:

```
399
400     questionPanelManager = new QuestionPanelManager(relUnitX, relUnitY, btnShortAnswer, btnMultipleChoice, btnMatrix);
401
```

Simply add your button to the end of the list of arguments.

Step 4: Add button to QuestionPanelManager

Go to the QuestionPanelManager Class. At the top, declare but do not instantiate the button. We will be getting that button in the constructor.

```
17 public class QuestionPanelManager {
18
19     int relUnitX;
20     int relUnitY;
21     int type = 0; //General = 0, Math = 1, Physics = 2, Chemistry = 3
22     JButton btnShortAnswer;
23     JButton btnMultipleChoice;
24
25     JButton btnMatrix;
```

Within the constructor, add your button argument but rename it as bXXXXX instead of btnXXXXX. Then set btnXXXXX to bXXXXX.

```
27 public QuestionPanelManager(int relX, int relY, JButton bShortAnswer, JButton bMultipleChoice, JButton bMatrix)
28 {
29     relUnitX = relX;
30     relUnitY = relY;
31     btnMultipleChoice = bMultipleChoice;
32     btnShortAnswer = bShortAnswer;
33     btnMatrix = bMatrix;
34 }
35
```

Step 5: Assign button to show up based on type

A little bit lower and you will see the addQuestions functions. This function runs whenever we click on a question type button. For example math or physics. Based on the question type (you can look in the pictures above at line 21 for what the numbers correspond to).

```

66     if (type == 0)
67     {
68         questionsPanel.add(Box.createVerticalStrut(relUnitY));
69
70         btnShortAnswer.setFont(font1);
71         btnShortAnswer.setAlignmentX(Component.CENTER_ALIGNMENT);
72         btnShortAnswer.setVisible(true);
73         questionsPanel.add(btnShortAnswer);
74
75         questionsPanel.add(Box.createVerticalStrut(relUnitY));
76
77         btnMultipleChoice.setFont(font1);
78         btnMultipleChoice.setAlignmentX(Component.CENTER_ALIGNMENT);
79         btnMultipleChoice.setVisible(true);
80         questionsPanel.add(btnMultipleChoice);
81     }
82     else if (type == 1)
83     {
84         questionsPanel.add(Box.createVerticalStrut(relUnitY));
85
86         btnMatrix.setFont(font1);
87         btnMatrix.setAlignmentX(Component.CENTER_ALIGNMENT);
88         btnMatrix.setVisible(true);
89         questionsPanel.add(btnMatrix);
90     }
91
92

```

So basically you are going to find your type and then add this:

```

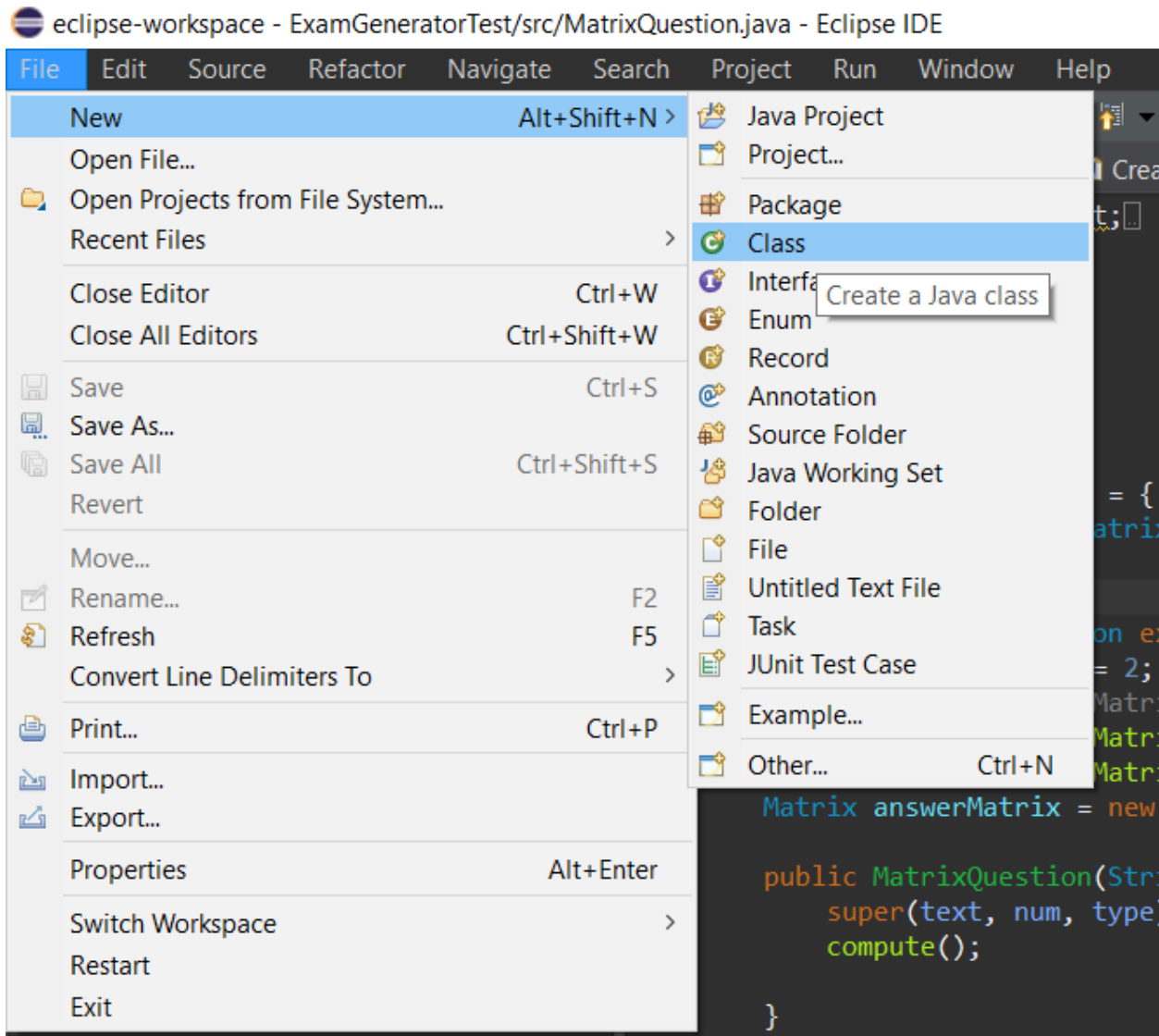
questionsPanel.add(Box.createVerticalStrut(relUnitY));

btnXXXX.setFont(font1);
btnXXXX.setAlignmentX(Component.CENTER_ALIGNMENT);
btnXXXX.setVisible(true);
questionsPanel.add(btnXXXX);

```


Step 6: Creating a class for your question

Now it is time to get into the specifics of your question and create a class for it. First, while in the project, go to File, then New, then Class. You will then get a pop-up which you should fill out in the way as the picture below.



New Java Class

Java Class

 The use of the default package is discouraged.

Source folder: ExamGeneratorTest/src Browse...

Package: (default) Browse...

☐ Enclosing type: MatrixQuestion Browse...

Name: ShortAnswerQuestion

Modifiers: ☐ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

? Finish Cancel

Hit finish and you will have a new class. Make sure to have your class extend the Question class and to add a constructor that takes in the text, num, and type.

```

1
2 public class ShortAnswerQuestion extends Question {
3
4     public ShortAnswerQuestion(String text, int num, String type) {
5         super(text, num, type);
6         // TODO Auto-generated constructor stub
7     }
8
9 }
10 |

```

What you do next depends entirely on your question and how you decide to handle it. Here is a snippet of code from the MatrixQuestion.

```

9 class Matrix
10 {
11     int sizeX = 2;
12     int sizeY = 2;
13
14     double[][] matrixData = { {1,2,3}, {4,5,6}, {7,8,9}};
15     RealMatrix matrix = MatrixUtils.createRealMatrix(matrixData);
16 }
17
18 public class MatrixQuestion extends Question {
19     int numberOfMatrices = 2;
20     //List<Matrix> listOfMatrices=new ArrayList<Matrix>();
21     Matrix matrix1 = new Matrix();
22     Matrix matrix2 = new Matrix();
23     Matrix answerMatrix = new Matrix();
24
25     public MatrixQuestion(String text, int num, String type) {
26         super(text, num, type);
27         compute();
28     }
29
30
31     public Matrix compute()
32     {
33         answerMatrix.matrix = matrix1.matrix.multiply(matrix2.matrix);
34         answerMatrix.sizeX = answerMatrix.matrix.getRowDimension();
35         answerMatrix.sizeY = answerMatrix.matrix.getColumnDimension();
36         answerMatrix.matrixData = answerMatrix.matrix.getData();
37
38         return answerMatrix;
39     }
40

```

Step 7: Adding the question to the Test Panel

Now that your question actually exists. We want it to show up on the test panel when your question is clicked on. To do this, go to the TestPanelManager class. Then go to the function “updateTestPanel”. There is an if statement that creates different JSwing elements depending on the question. Go to the bottom of the if statement and add a new else if statement where updateCode.equals("name of your question").

```
else if (updateCode.equals("ShortAnswer"))
{
    cons.gridx = 0;
    cons.gridy = 1;
    cons.weighty = 0.5;
    cons.weightx = 0.5;
    cons.fill = cons.BOTH;
    JPanel newPanel = new JPanel();
    newPanel.setLayout(new BorderLayout(newPanel, BorderLayout.Y_AXIS));

    newPanel.setBorder(new EmptyBorder(10, 20, 20, 20));
    JLabel name = new JLabel("Question " + numQuestions + ": Short
Answer Text");
    newPanel.add(name, BorderLayout.CENTER);
    JTextArea answerBox = new JTextArea(50, 5);
    newPanel.add(answerBox);

    basicPanel.add(newPanel, cons);
}
```

This example is for the “shortAnswer” question and is relatively simple compared to the Matrix question. However there are many similar elements. Because the “basicPanel” is a gridBagLayout, you must have the gridX = 0, and gridY = 1. The rest of the gridBag settings are up to you. Next create panels that will go inside the basicPanel. Make sure to add a JLabel with the Question number and text. Finally make sure to add everything back to the basicPanel. Now, when you click on your question, it should show up.

Step 8: Adding Question Settings

Now that we have your question, and it shows up when clicked, we have to add functionality to the settings panel so the user can make changes to the question. For a matrix question this could be the size of the matrices and for a multiple choice question this could be the number of answers.

First, go to the Settings class. There you will see the settings superclass as well as the many setting subclasses.

The settings class takes in the question and the question type.

```
16 public class Settings {
17
18     String type;
19     Question question;
20     public Settings(String type, Question question)
21     {
22         type= this.type;
23         question = this.question;
24     }
25
26 }
27
```

Above this, you will make a new class for your questions settings. Here is the MatrixSettings class as an example:

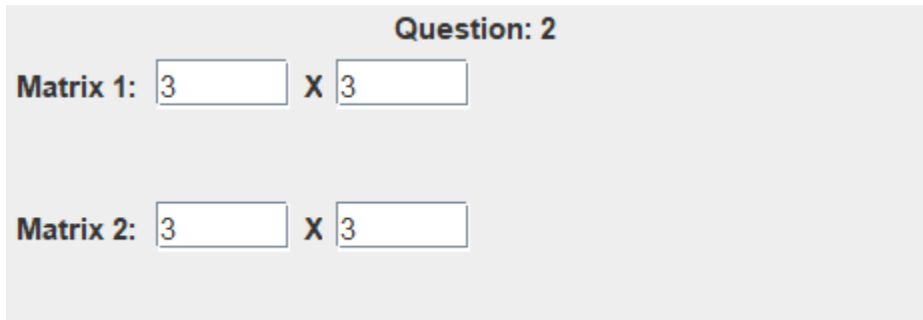
```
class MatrixSettings extends Settings
{
    //m1d1 = matrix 1 dimension 1, etc
    int m1d1;
    int m1d2;
    int m2d1;
    int m2d2;
    public MatrixSettings(String type, Question question) {
        super(type, question);
    }
}
```


Your settings variables should correspond to things that you will be able to change in the settings panel which we will cover next.

Step 9: Adding settings to the settings panel

Go to SettingsPanel class and go to the addSettings function. There will be a series of if statements. You will need to add an “else if type.equals("question name/type")” to the end for your question. There you will add whatever JSwing elements are necessary to alter the settings of your question.

In the MatrixQuestion example, see how there are 4 JTextFields that correspond to the 4 integers in the MatrixSettings class:



You will be adding the JSwing elements to newPanel which is returned at the end of the function:

```
JPanel newPanel = new JPanel();
newPanel.setLayout(new BoxLayout(newPanel, BoxLayout.Y_AXIS));
newPanel.setBorder(BorderFactory.createLineBorder(Color.green));
newPanel.add(new JLabel("Question: " + Integer.toString(question.questionNum + 1)));
```

You may also want to have a check for newQuestion if this question is new (as oppose to returning to this already created question at a later point) to set the settings to a default value and add the settings to the settingsList:

```
if (newQuestion)
{
    newSettings.m1d1 = Integer.valueOf(dim1.getText());
    newSettings.m1d2 = Integer.valueOf(dim2.getText());
    newSettings.m2d1 = Integer.valueOf(dim3.getText());
    newSettings.m2d2 = Integer.valueOf(dim4.getText());

    settingsList.add(newSettings);
}
```

Step 10: Committing Changes

Next you will want something to actually happen when you commit your changes. For example, when you commit changes for Matrix Question it completely redoes all the calculations and also changes the size of the matrix.

First of all you will want to go to the CreatorState Class. Go to the ActionListener for btnCommit and you will find an if else statement. Add an “else if” like to the end like this:

```
else if (questionsList.get(curQuestionNum).questionType.equals("questionName"))
```

Now in there, you will need to get everything that will change and that you need to make those changes and pass it to the commitHandler.

```
if (questionsList.get(curQuestionNum).questionType.equals("Matrix"))
{
    MatrixSettings curSettings = (MatrixSettings) settingsPanelManager.settingsList.get(curQuestionNum);
    JPanel curPanel = (JPanel) view.getComponent(curQuestionNum * 2);
    curPanel = (JPanel) curPanel.getComponent(1);
    MatrixQuestion question = (MatrixQuestion) questionsList.get(curQuestionNum);

    question = commitHandler.commitMatrixChanges(view, mainPanel, curSettings, curPanel, question);
}
```

This will most likely include the settings for the question, the question panel in the testPanel, and the question itself.

You will then make a line that goes:

```
question = commitHandler.commitXXXXChanges(parameters);
```

Where XXXX is the name of your question. You will get errors as this function doesn't exist yet, but that's okay because we are going to create it next.

Go to the CommitHandler class and create a function with that name:

```
public MatrixQuestion commitMatrixChanges(JPanel view, JPanel mainPanel, MatrixSettings curSettings, JPanel curPanel, MatrixQuestion question)
{
```

You will then need to go and make all the changes that need to be made when a user commits their changes. Look at the code already there for examples.

Step 11: HTML Parsing

The final step in the process is to convert your question to HTML. First go to the HTMLParser class. There you will make a new function that will return an HTML string. Name it parseXXXX:

```
public String parseMatrix(int id, String title, int points, double[][] matrice1, double[][] matrice2, double[][] result) {
```

Within this function you will take your question and all relevant information and convert it to html. This will vary wildly from question to question.

Then within the convert function, add your question to the if else statement:

```
for (int i = 0; i < questionsList.size(); i++)
{
    //Question curQ;
    if (questionsList.get(i).questionType.equals("Matrix"));
    {
        MatrixQuestion curQ = (MatrixQuestion) questionsList.get(i);
        parsedInput += parseMatrix(i + 1, curQ.questionText, 10, curQ.matrix1.matrixData, curQ.matrix2.matrixData, curQ.answerMatrix.m
    }
}
System.out.println(parsedInput);
```

```
else if (questionsList.get(i).questionType.equals("Name"))
```

Follow the formula of the Matrix Question and just add parseXXXX instead to the parsedInput. Now when the test is exported, it will take your question and all its information and convert it to HTML.