

COMP 303

Project & Mini 5 Combo

Due: April 16, 2018 on myCourses at 23:30

Question 1 (20 points)

Please read the entire question before you start.

JavaFX is a library of classes that implement a prettier GUI than Swing. JavaFX does not replace Swing but upgrades it. At the heart of JavaFX 1 was Swing; Calling JavaFX methods resulted in Swing calls.

For this assignment you will create your own mini implementation of JavaFX called JavaCool303. You cannot use JavaFX in this assignment. You must use Swing. You must, like JavaFX, build an implementation that uses Swing components internally but abstracts it away in your own mini library called JavaCool303.

Create a package called JavaCool303.

Design and program using good OO techniques and Design Patterns. Hand in a README1.txt file that lists the well-designed techniques and design patterns you used in your program. Indicate where these techniques were implemented so that the TA can verify.

Your programming solution only needs to implement enough code to answer these three submission requirements:

1. Design your solution on paper using a UML Class diagram, use UMLet. Add notes to your diagram making it clear to the reader which design patterns you chose to include in your implementation. One way you can do this is to draw a dashed box around the objects that make up one design pattern, then label the dashed box with the name of the design pattern. Remember Goldilocks, not too much, not too little, develop something balanced, optimal and following the good designed principles we covered in class.
2. You will build two things: a simple application and the JavaCool303 library. The simple application uses the JavaCool303 library.
 - Design the JavaCool303 library as a standalone library that can be shared with other developers. Make sure to protect your library and provide an API that makes it easy to use. Create a JavaDoc for your library to help users who may want to use your library.
 - The simple application displays a window with a square area that can accept n objects (the application will populate the area with 20 buttons). The square area is contained within the window but is a different component. Each button is labeled by a unique integer number. When a button is pressed the label is output to the consol. The various components of this application are JavaCool303 items.

3. JavaCool303 is a Java Swing library upgrade that is based on themes. A developer wanting to use your JavaCool303 package must provide a Cool303Theme class that modifies the look of all the GUI elements contained within JavaCool303. Your package will come built in with two themes: Pastel and Summer. Pastel must use pastel colours with bubble-like shapes (as best as you can do) for buttons / boxes / windows / areas; Winter I will leave it to your imagination but the implementation must be different from Pastel.

Your solution must use what we have covered in class: abstract classes, interfaces, inheritance, and design patterns.

You will have to wrap the paint methods of Swing components with your own version for JavaCool303. Don't animate anything, unless you want a greater challenge (there are no extra points for animation).

Take time to think of the problem and how to implement it by wrapping or replacing a minimum number of Swing components with JavaCool303 components. Name your components like this: Cool303ZZZ. For example: Cool303Component, Cool303Container, Cool303Theme, Cool303Box and Cool303Button. Your solution must maximize Swing reuse under/within this new packaging. Your solution must maximize reuse within the JavaCool303 library elements themselves.

What makes this different from Swing? Once a theme is selected, then all the Cool303ZZZ classes are affected by that theme. For example, the application program will display all the components of the GUI according to the theme. A good test, by the TA, would be to keep your application as-is but change only the theme to your other theme. If that test works, then they will keep your application as-is but write their own theme, following the instructions from your JavaDoc. Your program must be able to work in all these cases to receive full points.

Note: If the user leaves the theme empty (NULL), your library must throw an error.

The basic classes will work this way:

- Cool 303 Components are individual GUI units that are used to populate Containers.
- Cool 303 Containers store and display components in a fixed bounded area. But the bounded area cannot be seen. It is an invisible area; however, the components can be seen and the containers can be given a background. Containers have an optional string. If this string is present, then the string is displayed as a title (in bold) appearing at the top left corner of the container area. Containers have an optional background color, if provided the invisible area is revealed.
- Class Cool303Root is a container populated with Components and Containers. A user can implement more than one Cool303Root in their application. The Root is fancy and follows the selected theme and applies the theme to all elements populated within it. Root does not have a title or any other text. Root will auto size itself to the minimum size needed to display all the components if the user does not specify a size. If the user of Root defines a size the class will attempt to follow that request if it is sufficient to display

all the components. If it is not big enough then the automatic sizing option is automatically initiated overriding the user's specified size.

WHAT TO HAND IN FOR QUESTION 1 AND GRADING

- 1 point: A readme.txt file explicitly stating the design techniques and the design patterns you used
- 5 points:
 - Proper use of techniques (for example, abstract classes, interfaces, inheritance, protected, contracts, analogies, design with reuse)
 - Properly used design patterns
 - Good selection of techniques and design patterns
 - Following the Goldilocks principal
 - Followed instructions
- 1 point: Easy for developer to create their own themes
- 1 point: Easy to incorporate JavaCool303 into a new project
- 2 point: JavaDoc and commenting and good variable names + easy to read code
- 5 points: UML diagram demonstrating a well-formed design
- 4 points: Sample program displaying window, box and 20 buttons (as described)
- 1 point: JavaCool303 as a package


Everything is graded proportionally.

Question 2 (10 points)

Simulators are programs that attempt to duplicate real life in software. For this question, you will use well-formed techniques and design patterns to construct a portion of a simulator. As in question 1, you will need to provide a readme2.txt file making explicit the well-formed techniques and the design patterns you used to answer this question. Specify where these techniques were implemented so that the TA can verify.

One of the most fundamental parts of a simulator is the World and those items that populate the World. Often a simple way to represent a world is using a 2D array. This array is imagined to be the ground. Items populate this array. Some items are immovable, others are moveable, and others move autonomously.

You will need to build the following elements of your simulator using proper object-oriented techniques, good software development techniques and using design patterns. You may add additional helper methods and objects if needed. Remember to be optimal.

- The World: 
 - A 2D array that can store one Moveable, Immoveable, or Autonomous object per cell of the array.
 - A method called `public void step()` that iterates through the cells of the array changing the state of the world by updating the position of all the Autonomous and Moveable objects (see below). It does this once for each call to the method.
 - The method `public void display()` to display the world on the screen using Swing or JavaFX. This must be a GUI grid displaying simple text tokens that represent the items in the world.
 - The method `public void add(item, x, y)` is used to populate the world by adding items to the array at cell x,y. The cell needs to be available (empty) or the add fails.
 - The constructor defines the size of the array. The array is empty.
- Immoveable:
 - Private string name, describing what it is.
 - Private char token, a character that stores the symbol that represents the item when printed to the screen.
 - Private int x, y, which specifies the location in the array this item exists.
 - The method `public char getToken()` returns the character symbol.
(If you want a greater challenge, you can replace the symbol by a graphic.)
- Moveable:
 - Implemented exactly as Immoveable, however it can be moved by one cell position if bumped into by an Autonomous object. It is displaced in the same direction as the bump. For example, if the item was bumped on its right side and the motion of the bump was towards the left, then the item will move to the left.
- Autonomous:
 - Implemented exactly like Moveable (bumped by a Moveable object or another Autonomous object causes it to shift one cell in the direction it was bumped).
 - A `public void step()` method that randomly picks a direction and updates the item to a new location by one cell.
- Construct a main method that builds a world and then runs a simulation for 100 iterations.
 - Your program will not prompt the user at the beginning.
 - Your main method will call `private static buildWorld()` method that will be hand coded to construct a world of a particular size and then populated with 5 immoveable, 3 moveable and 2 autonomous objects. Your implementation must be able to handle any number of items since the TAs will change these parameters when they test your program. Once the world is populated the method returns to the main.
 - From the main, a for-loop is used to iterate 100 times. For each iteration of the loop the objects of the world are updated and their new positions are displayed on the screen.
 - When the simulation is complete, the program prompts the user: "Would you like to run the simulation again?". If they say yes then the world iterates for an additional 100 loops, continuing with the world as it was before the prompt. If

they say no, then the program terminates. After this second 100 iteration the user is prompted again, etc.

WHAT TO HAND IN FOR QUESTION 2 AND GRADING

- 1 points: A readme.txt file making it explicit the well-formed techniques and design patterns you elected to use to answer this question
- 2 points: Well-formed techniques and design pattern choices
- 2 points: World
- 1 point: Immovable
- 1 point: Moveable
- 2 points: Autonomous
- 1 point: Main

Graded proportionally