PPA Assignment X

Overview

Do you think there is a problem with any of the content below? Let us know immediately at programming@kcl.ac.uk.

You are not advised to print this assignment, but instead to always access the latest version of this brief through KEATS, which will evolve with minor clarifications and corrections throughout the assessment period. Students will be notified of any major modifications to the brief by email.

The deadline for completing CWX, Requirement 0 has now passed, but the activities associated with this requirement will still need to be completed before you are able to proceed.

The deadline for completing the remaining requirements (R1+), listed in this brief, is Friday 31st March, at 7.00pm.

You are strongly advised to spread the work for this assignment throughout the assessment period.

The tasks in the brief will need to be appropriately delegated to different group members in order to complete the work in the time given. For certain tasks, this delegation is mandatory.

If you have any questions about this assignment, please follow the steps listed in the 'Support' section on KEATS.

Remember, we will be using the Team Feedback system to elicit feedback from each of your fellow group members about your contribution to the group project. If significantly negative, and your commits to Github are not sufficient, this feedback will directly affect your grade in this assignment (15% of your PPA grade). Therefore, be sure to make a good impression on those students in your group throughout the assignment.

Other tips and important information will be given in this column.

Aims

The aim of this assignment is to provide practice in:

1. Developing user-friendly interfaces;

2. Building self-contained applications with components, layout managers and event handlers;

3. Using a version control system for managing source code;

4. Interpreting application requirements that go beyond a Computer Science technical level and relate to application domain user needs;

5. Identifying difficulties in GUI design;

6. Working with pre-existing code, provided as a library;

7. Performing task analysis and UI design;

8. Documenting code.

Domain Description

The National UFO Reporting Center (NUFORC), located in Seattle, WA, was founded in 1974 by noted UFO investigator Robert J. Gribble. The Center's primary function over the past two decades has been to receive, record, and to the greatest degree possible, corroborate and document reports from individuals who have been witness to unusual, possibly UFO-related events.

NUFORC has catalogued almost 90,000 reported UFO sightings over its history, most of which were in the United States.

In addition to record keeping, the center has provided statistics and graphs to assist others looking for information. In 2014, the Economist used the statistics compiled by the organisation to analyse 'peak times' for UFO reports, drawing the conclusion that most reports came during the hours during which people were most likely to be intoxicated.

In this assignment, you will develop an application to work with this UFO sightings data.

Requirements Overview

The application that you build for this assignment should exhibit the following functionality:

1. The software should be able to connect to a server (using a provided API) in order to access UFO sighting incident data.

2. The application should present a main window, within which additional panels appear, and can be scrolled between.

3. When the main window loads, the user should be presented with an introductory panel, prompting them to selected a time frame from which to collect UFO sighting reports.

4. The user should be able to scroll to a pane presenting a map, upon which information about UFO sightings is marked.

5. Clicking a UFO sighting marker should yield an additional window giving more information about those sightings.

6. It should be possible to view a detailed description of a sighting by clicking on a summary of that sighting.

7. It should be possible to order the sightings in a given area, by different attributes.

8. The user should be able to scroll to a pane presenting four statistics (of a possible eight), which provide a summary of the data they have collected from the remote API by selecting a date range.

9. The user should be able to browse these statistics, but never see the same statistic on the panel at the same time.

10. The statistics that a user elects to see should remain persistent when the application closed, and then opened again.

11. The application should present an additional feature, not covered by these requirements.

The Ripley API

The term API (Application Programming Interface) has developed a reasonably flexible definition in recent years. In general though, an API provides you with the resources to develop an application, these resources 'plug in' to your application, and they become a fundamental part of its operation. Most typically you will be familiar with the Java API. In this API, the supplied resource is the Java library classes (e.g. Scanner, ArrayList etc.), which you use to produce your programs. When you import classes from this library you are plugging them in, and the operation of your program relies on their presence.

The resources provided by an API don't have to be purely function based. Instead, an API can provide you with data as a resource. A good example of this is the Twitter API, which

provides you with data from Twitter that can be used as a resource for building applications. Typically APIs of this type also provide you with the means to manipulate the data provided, at source. Unlike the Java library, which can be accessed locally, APIs like the one provided by Twitter are accessed remotely over the Internet using a network connection.

The Ripley API provides a remote data resource for the UFO sighting information mentioned in the previous section. In this piece of coursework, you will interact with the Ripley API via a local proxy (in the non-technical sense of the word) which takes the form of a custom Java library. As we know, custom Java libraries contain additional classes that do not exist in the standard Java library. In order to call the methods contained in a custom library, the associated jar file (in this case `ripley.jar`) must added to the build path of a project (or the classpath, when compiling from the terminal) so that classes from that library can be imported. The classes provided by the Ripley API Java library (the Ripley library) encapsulate calls to the Ripley API allowing you to obtain remote data via local method invocations. Thus, in this coursework, you will use an amalgam of a local API and a remote API.

Like all remote APIs, use of the Ripley API, and thus the Ripley library, is subject to the constraints of network communication. You may therefore experience latency issues when accessing the Ripley API, especially if you decide to request a large amount of information, but this is normal. If the Ripley API goes offline for any reason, details will be posted here:

https://nms.kcl.ac.uk/martin.chapman/ppa/ripley/

In addition, like all software, the Ripley library will be subject to revisions throughout the lifetime of the major project. So keep an eye on your repository for new versions. A version number is included and accessible through the library in order to monitor this.

Connecting to and testing the Ripley API

Your first preliminary task should be to ensure that you can use the Ripley library to communicate with the Ripley API. You will find the Ripley library jar file in a new branch of your repository. Acquire this library (by either merging the branch, or pulling the branch and manually extracting it from your repository), and include it in the build path of project, or save it in an accessible location locally if you intend to compile and run from the command line. You may need to conduct some additional research to find out how to do this from your chosen IDE or, if you so choose, how to compile and run a program written to work with a custom library from the command line.

Once added, import the library as follows:

```
import api.ripley.Ripley;
```

You can then make a Ripley object, and interact with it in order to experiment with various method calls. In order to make a Ripley object, you will need to pass both your group's public and private keys. All these details are confirmed in the Ripley library documentation, which is also available in your repository. You should use this documentation from now on to understand how to use the Ripley library.

The first method I recommend you call is the simplest: `getLastUpdated`. This method will tell you when the last set of UFO sighting data was received. Print the result of calling this method in order to confirm your connection with the API.

When adding the `ripley.jar` file to your group's project, make sure you don't include a path on your machine to the file, as this won't be accessible by the other members of your group who are cloning your repository, but instead always add this .jar file with a local reference.

The Ripley API documentation is packaged as a supporting jar file, which you will also find in your repo. You should associate this documentation with the Ripley library jar in your build path, so that you can view the documentation in the relevant tooltips. Within the jar, the documentation is available in a folder called `doc'.

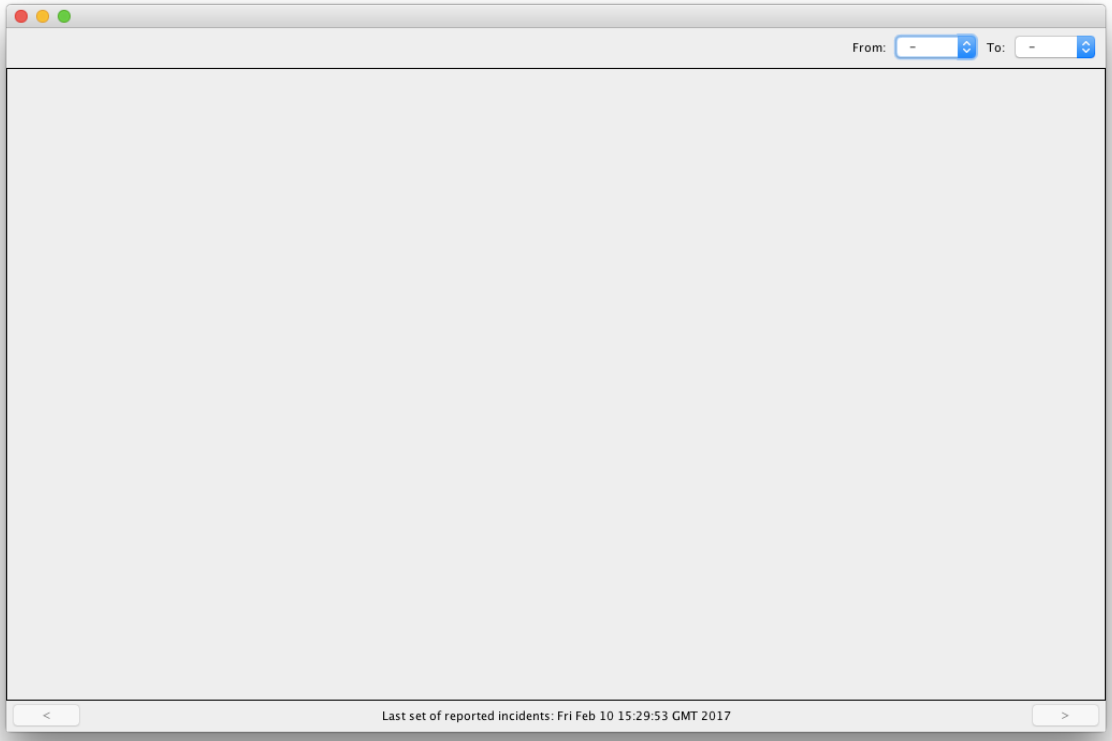
If you are having trouble using the Ripley library to access the remote Ripley API in the Informatics labs, then you may have to configure your IDE to use a HTTP proxy. Conduct the appropriate research in order to learn how to do this, and share your experiences with your peers via the PPA KEATS discussion forum. The common proxy settings required are proxy.inf.kcl.ac.uk as the host, and 3128 as the port.

Application Structure: Main Window

The application we are going to build for CWX is designed to process and display the data from the Ripley API, along with associated data, in a digestable form for a user.

To do this, we will create a multi-panel and multi-window application.

The first, and main, window of this application should look similar to the following:



This window is designed to hold a series of different panels, each of which contain data relating to the retrieval and display of data from the Ripley API. Details of what these panels should contain is given in some of the remaining sections of this brief.

Key behaviour that should be offered by this frame itself is as follows:

1. There should be the ability to move left and right through the panels contained in the centre of the frame using `back' and `forward' buttons in appropriate positions. Naturally each of these buttons will only work if there are still additional panels to access in the selected direction.

2. There should be a string, in an appropriate place on the frame, informing the user of when the data set, to be presented by the application, was last updated.

3. The top right of the frame should feature two drop-down boxes, appropriately labelled, allowing a user to select a date range from which to derive alien sighting data. The upper and lower bounds of this range can be retrieved from the API.

4. The user should be alerted if they have selected a range that is invalid, such as the start date they have selected is greater than the end. You may also wish to restrict the length of duration that the user can select tracking information for, if you believe this will be deterimental to their use of the program, in respect of the time taken to load data.

5. The `back' and `forward' buttons should be initially disabled, until the user has selected a date range from the ranges shown. This is because the other frames available are going to process and display the data loaded when a user selects a date range, and

are thus initially empty. The first panel, however, is unconnected to the data, and should thus be shown to the user when the frame first loads, as discussed in the next section.

Important: Acknowledgement String

Your program must print the acknowledgement string supplied by the Ripley Library to standard output at some point during the lifetime of your program (preferably at the start).

Students who do not display this acknowledgement string will receive a zero for the entirety of CWX.

You're welcome to use any Swing components to construct your UI for CWX, but always make sure they offer you the appropriate flexibility before employing them. A discussion about additional library classes appears at the end of this document.

Application structure does not just apply to the visual structure of your program. You should also aim to structure your classes according to the MVC paradigm.

Your program may at times try and pull a large amount of data. You will want to consider how you handle this both in the backend of the application (e.g. reducing calls to the API, caching mechanisms, etc.) and in the UI itself (e.g. informaing the user that the application is loading). Marks are awarded for efficient use of the API, as discussed later.

Panel 1: Welcome

Having a user land on a blank screen like the one shown in the image above isn't too good in respect of usability.

Instead, the first panel loaded into the Main window when a user loads the application should welcome them to the application, and given them instructions on its basic use.

As part of the welcome, you should display the version of the API that is currently being used, by accessing the appropriate method from the Ripley library. Your application should always use the latest version of the Ripley library, so check your repository regularly.

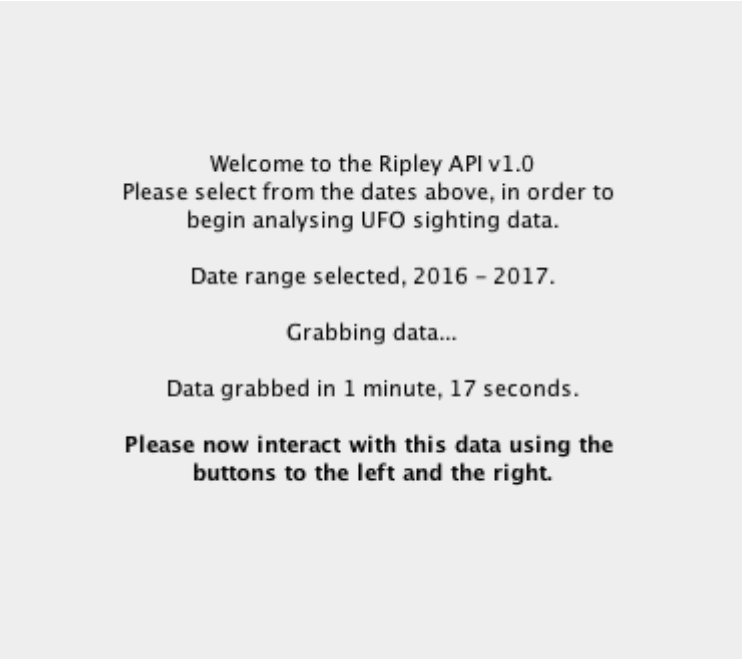
You should then use this window as a log, showing the user key events in the program as they begin to interact with it by selecting a date range from the first window.

Key events are as follows:

1. The date range the user has selected.

2. The total time taken to load the data.

3. Once the application is ready (i.e. the left and right control buttons are no longer disabled), the user should be informed, preferably in bold text, similar to the below:



Remember, we want a proper MVC separation in our code, so think carefully about how all the panels (views) communicate with each other, and are supporting by different classes unconnected to the visual appearance of the program (models).

Panel 2: Map

The first data related panel that should become available for access by the user once they have selected a date range, and the application has loaded this data, is a panel that visually demonstrates the content of the Ripley dataset to the user.

This should occur by showing the user a map of the United States (where a large number of alien sightings occur) on the panel, with an appropriately sized UFO or alien themed graphic placed over a state to indicate that both alien activity has taken place in that state during the selected time period, and the extent of this activity. We will refer to these as markers. Please find and select an appropriate UFO graphic, and the map of the US onto which copies of this graphic are to be placed.

An example section of a map of this nature is as follows:

If a state has at least one alien sighting, then a marker should appear within the bounds of that state on the image, in order to demonstrate this.

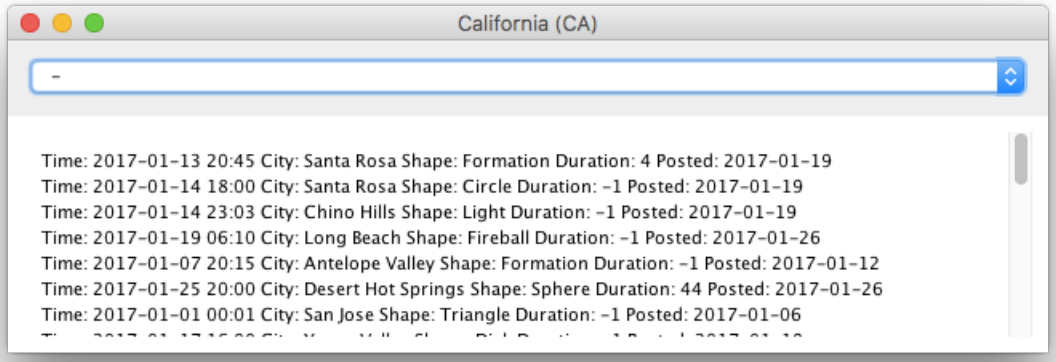
We will not be too concerned with where the marker appears within the state boundaries, but it should appear within them, such that it is clear to which state each graphic pertains.

The size of the marker should depend on the relative number of alien sightings in that state. For example, in the graphic above, California (CA) has a larger number of sightings reported in the period chosen by the user than Arizona (AZ), while Arizona has a larger number of reported sightings that Nevada (NV).

Lists of sightings

Upon a user selecting a date range, and the appropriate markers being appended to the map of the United States, it should be possible for a user to click any of these markers in order to learn more about the sightings that have occurred within this state.

This information should be presented in a new window, which pops up when each marker is clicked, and resembles the following:

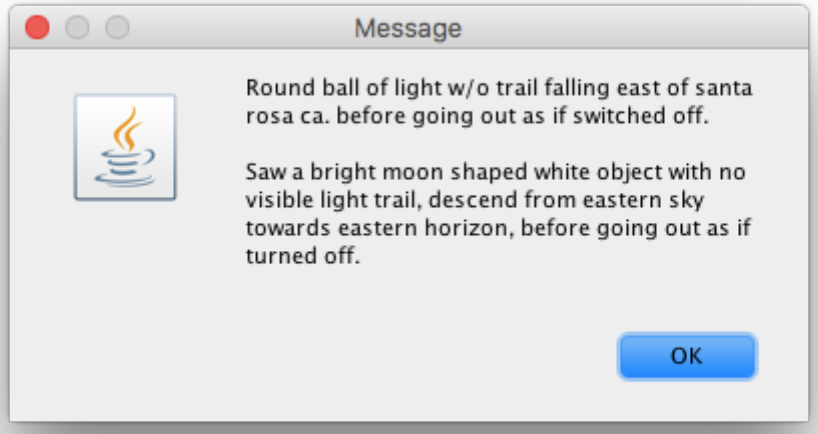


The title of the window should include the full name of the state in which the sighting occurred, along with the abbreviation of that state from the map.

This window should present a list of all the sightings, including the following data:

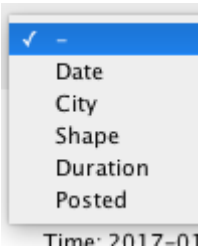
1. The time of the sighting (in a processable time format).
2. The city in which the sighting took place (in a string format).
3. The shape of the UFO observed during the sighting (in a string format).
4. The duration of the sighting in minutes (in an integer format).
5. The date on which the sighting was added to the dataset (in a processable date format).

When one of the sightings is clicked in the list, further information should appear about the details of the event, similar to the below:



Sorting sightings

Using an additional drop-down menu, it should be possible to sort the sightings by any of the listed data (date of sighting, city, shape, duration and added date):



Making this selection should reorder the list accordingly.

You'll likely have to position some things using absolute values here. Ideally we would like to strive to make every panel as responsive as possible, but here we will break this rule in order to practise other ideas.

You'll need to think carefully about how the graphics are sized over each state. We want some way to size each graphic relative to the others, using size to indicate when one state has a higher number of sightings than another. We also don't want our graphics to be too small, or too big.

Most of the data that you receive from the Ripley API will be in string format, so when it comes to sorting the data, you will have to parse some of this information.

The trickiest thing to parse will probably be the duration of each sighting. To do this, I would recommend looking at the natty library. For convenience, I have packaged a JAR file containing the natty library, and an associated logging library that it needs to run, for you here. Even given this library, it won't always be possible to parse a suitable time from the information given to you by the Ripley API, in which case you should list a suitable integer to indicate this, such as a -1, as in the screenshot shown.

Panel 3: Stastistics

In the next panel, we will present information about the Ripley dataset in a more traditional manner.

This panel will present a series of statistics, based on the information derived from the dataset, and should thus be available at the same time as the map to be observed by the

user, should they scroll to this panel.

We will derive eight statistics over the data available.

Panel behaviour

The panel should be separated into four distinct sections, which we will refer to as statistic boxes. Therefore, at any given time, only four of the available eight statistics are shown. Each statistic box should look similar to the following, with each box displaying a different statistic:

Using the buttons shown, the user should be able to click between the different available statistics, in a similar manner to the way in which they are able to click between the different panels of the main window.

At no point should the same statistic appear twice on the panel (as this would confuse the user), instead, when the panel is first shown, four different statistics should be selected. Moreover, when the user is clicking between the different statistics, it should not be possible for them to configure the panel such that it shows the same statistic in more than one box.

Statistics

The base statistics you are asked to implement are as follows:

1. **Number of Hoaxes**: The description component of each incident regularly receives annotations from the NUFORC administrators. Often, these annotations indicate whether NUFORC believe the entry to be a hoax or not. This statistic should count the number of suspected hoaxes in the selected time period, and display them to the user.

2. **Non-US Sightings**: At this risk of this project focussing too much on the United States, the second statistic should show the number of reported sightings in the dataset that did not occur in the US (which includes jolly old Great Britain).

3. **Likeliest State**: Based on the data available over the selected period, this statistic should show which state is likely to receive a UFO sighting next. The simplest way to derive this statistic is to look at the state with the most sightings over the period, and assume that this is therefore a likely place for the next sighting to occur. More sophisticated techniques for determining this statistic are welcome.

4. **Sightings via other platforms**: This statistic should aim to obtain a very high-level understanding of the current state of UFO sightings via another dataset, in addition to the one supplied by the Ripley API. The recommended way to do this is to use the Google API to search YouTube for recent videos that contain UFO sightings.

Additional Statistics

It is now your task to come up with four additional statistics that a user can scroll to in any of the statistic boxes.

These statistics should be significantly different to the four supplied above, and this will be marked accordingly.

Each group member should take responsibility for coordinating the invention and development of each statistic. Your Github branches will be checked to ensure that the majority of the development work behind each statistic is undertaken by one group member.

A prize will be awarded to the team that derive the most insightful additional four statistics from the dataset.

Saving preferences

When the user scrolls one of the statistic boxes to a particular statistic, they are expressing a preference for the information that they want to see on their interface (e.g. they are particulalry interesting in seeing hoaxes, and perhaps less interested in Non-US sightings).

Therefore, these preferences should be saved, such that when the program is closed and reopened again, the same four statistics that were visible in the four statistics boxes due to the user's choice are visible again.

This panel should be as responsive as possible.

Just like the Ripley API, when using an external API such as the one offered by Google, you will require a key to access this information. Make sure you conduct the appropriate research in good time, in order to learn how to derive such a key.

In addition, to contact this API, you will need to consider how your program can send HTTP(S) requests to a remote server.

Panel 4: Surprise me

The final panel is your blank canvas, designed for you to make your program do something additional and interesting, that is doesn't do already. This can include launching new windows, if you would like to.

Marks for this section will be awarded liberally, so be as creative or reserved as you like, but make sure your functional addition is evident.

Only one functional addition is required per group.

You can see a selection of the surprise me features offered by students last year here.

A prize will be awarded to the team that derives the most interesting, inventive or amusing surprise me feature.

This is your opportunity to set your own goals for a part of this task, and receive marks for it. Coming up with an idea that interests you and implementing it, given the syntax and understanding you have acquired over the duration of PPA, is your last step in becoming a true programmer!

Project Report

This requirement is to be completed individually, not as a group.

In this requirement you are asked to think of scenarios in which your application will be used and how it should be extended in order to be of better help potential users. You are not asked to actually implement anything. Rather you are asked to write a report about your extensions (maximum 6 pages). Your report should be written based on an analysis of the tasks users need to perform (through a hierarchical task analysis) and the related domain concepts (through a domain analysis).

Your report should include models of virtual windows and the global navigation structure of your extended application. Decisions or assumptions made in analysis and design need to be clearly identified. Details of all these design techniques will be given during the PPA lectures.

This documentation is similar in purpose to the documentation you submitted last term, in that it allow you to reflect on the code that you have written. However, it will contain different analysis techniques that you will not cover until later in the course.

Mark Overview

In addition to implementing all of the functionality specified in this brief so far, you must do the following in order to achieve a high mark for CWX:

1. Make efficient use of the API; minimising calls to the API and implement caching techniques, in order to improve user experience.

2. Use only the Ripley API to derive the information relating to UFO events.

3. Keep your private and public keys confidential to yourself and your group.

4. Only use those libraries cited in the lecture notes to date, or referenced in this brief. If you feel that an additional library may be relevant, you may ask one of the module leads for permission to use it.

5. The purpose of each public class, public method, and public field must be explained with Javadoc comments. Inline comments must be included where appropriate to explain the logic of your code.

6. Your application must be well-structured, aiming to avoid duplication of similar code by encapsulating it in appropriate methods or classes and using anonymous classes where appropraite. Use loops rather than repeating code, and so on.

7. The interaction between the components must be organised according to the structure dictated by the MVC paradigm.

A more detailed mark scheme will be released further on in the project lifecycle.