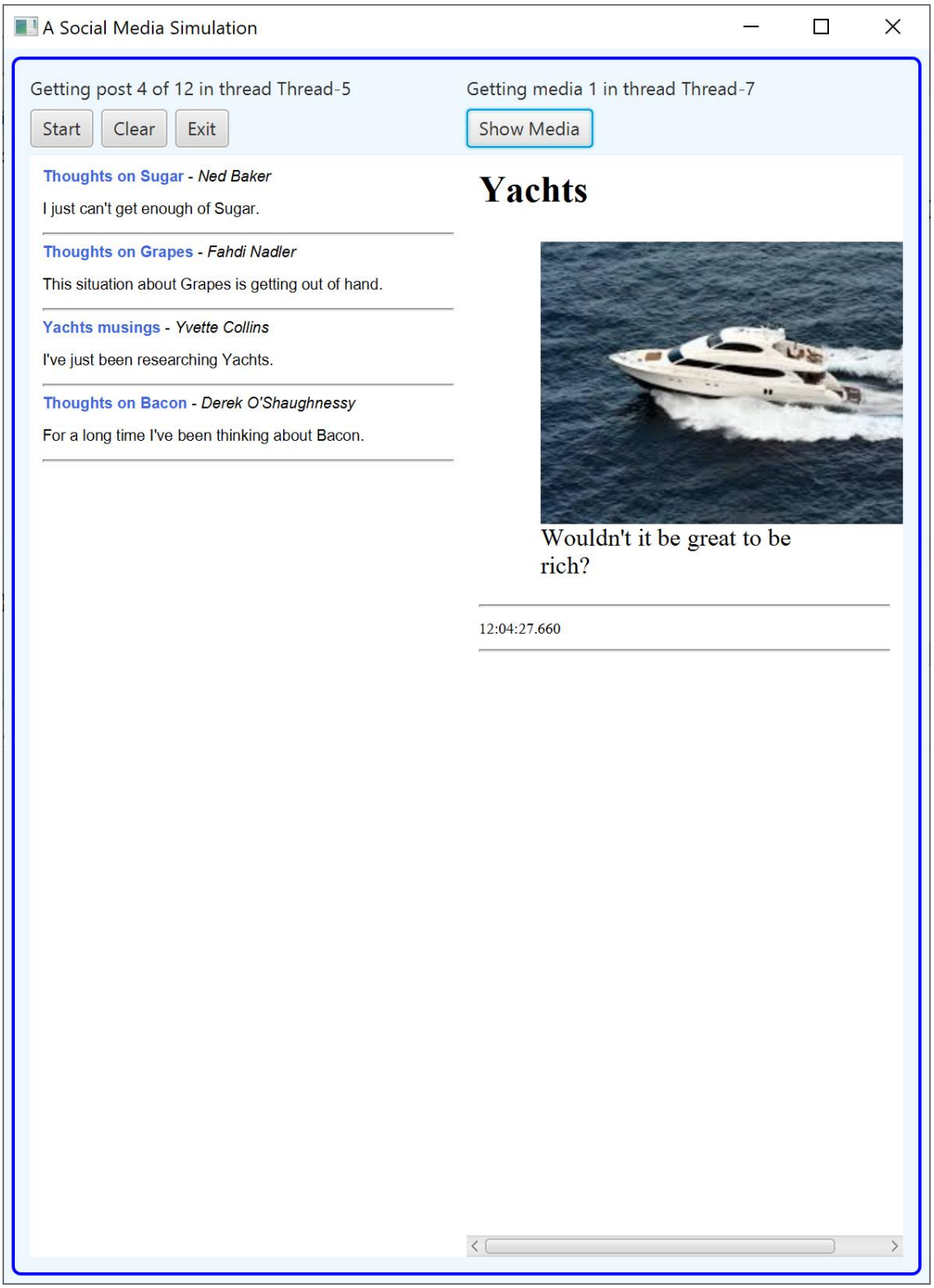
Social Media Lab



You are going to make a social media application like the one pictured on the left.

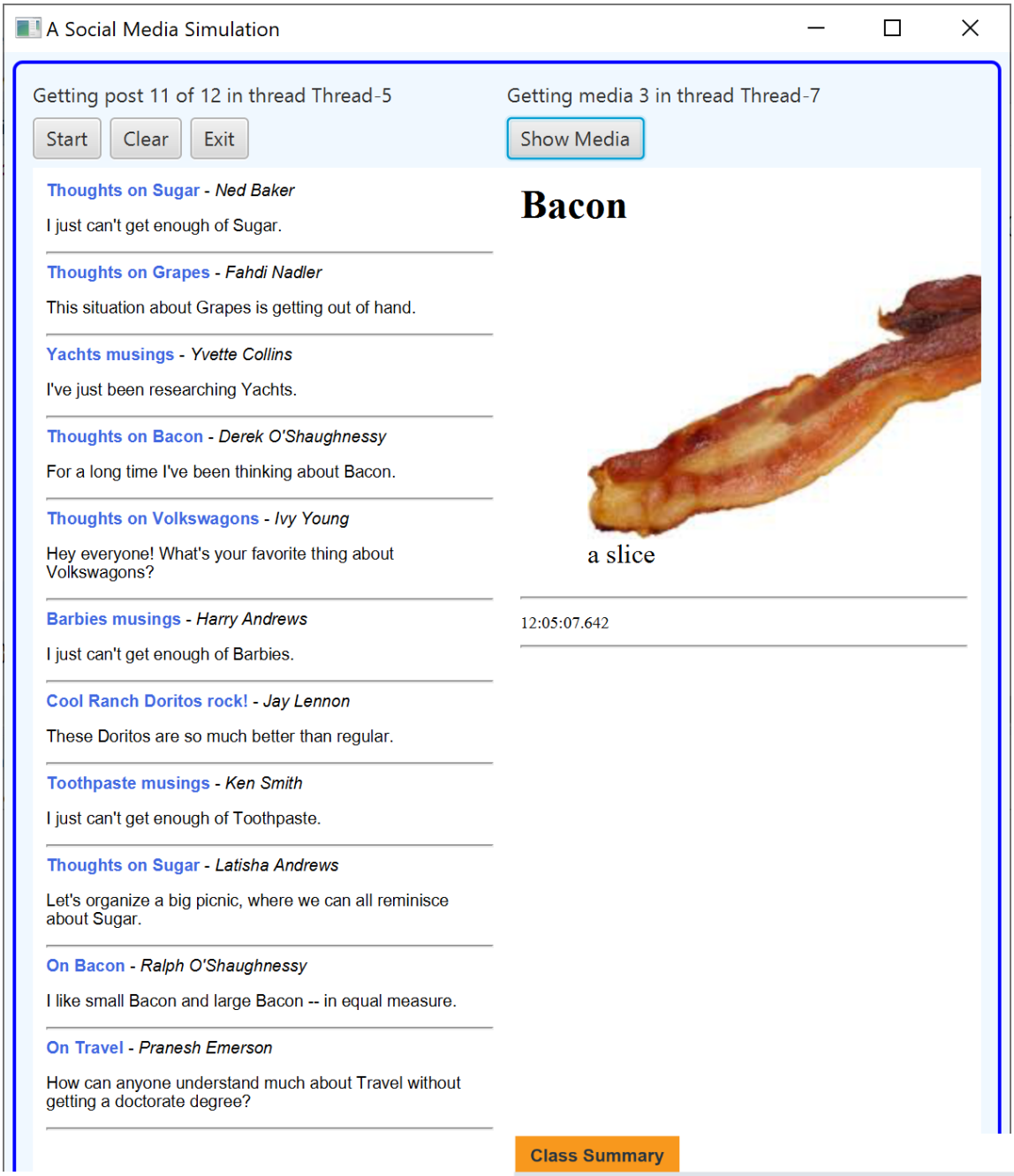
The screen is divided into a right half and a left half. The left half will be provided for you.

Pressing the **Start** button will launch a separate thread (process) that will add posts to the left-hand side. It is your job to create the right half.

The right half will analyze the posts looking for topics. It will identify the most frequent topic and present some media item in the right half. Notice that, after four posts, the topics **Sugar**, **Grapes**, **Yachts**, and **Bacon** are all discussed two times in the post feed. Thus any one of these qualifies as having the maximum number of references.

In this case, Yachts was selected, so the media feed on the right showed an image of Yachts.

The GUI is running on one Thread; there is a separate Thread adding posts, and yet another Tread analyzing posts and displaying media.



After the threads have been running a while, (we're up to post 11), here is how the topic count is going:

Topic	count
Sugar	4
Grapes	2
Yachts	2
Bacon	5
Volkswagons	2
Barbies	2
Doritos	2
Toothpaste	2
Travel	2

Now since **Bacon** is the most discussed topic, the media feed now shows a picture of bacon on its third fetch. (On its second fetch, it still showed the picture of Yachts.)

Notice the buttons on the left to **Start** the Thread of the post feed, to **Clear** the post feed and to **Exit** the application.

The button on the right starts the Thread of the media feed.

I. Elements provided to you

You will be provided with the following classes. The **Post** and **PostGenerator** classes can be used as is. **SocialMediaGUI** needs to have more functionality added. **Tokenizer** needs to have its *mostUsedTopic*() method finished.

These are the values of the **Topic** enum: Bacon, Barbies, Doritos, Grapes, Hairstyles, Hamsters, Investments, Music, Pancakes, Pizza, Popcorn, Romance, Squirrels, Sugar, Superman, Toothpaste, Travel, Volkswagons, Yachts

Class Summary

Class	Description
Post	A class to define a simple social media post
PostGenerator	A helper class that will generate a list of Posts using random generation of post authors, post titles, and post texts.
SocialMediaGUI	A GUI that will simulate Social Media posts to test an application to create ads.
Tokenizer	A helper class to deal with String analysis by either tokenizing a String or extracting tokens from a String.

Enum Summary

Enum	Description
Topic	An enum to list topics that people might discuss on social media.

II. Modify the Tokenizer

The Tokenizer code already creates a HashMap for you. You will need to find the Entry with the largest number of matches and return the value of its Topic.

mostUsedTopic

```
public static java.lang.String mostUsedTopic(java.lang.String inputString)
```

This method takes in a String (which could be a very long String) and builds a HashMap to store the analyzed results. The key of the HashMap will be a value of the enum Topic. The value of the HashMap will be the count of how many times the word appears in the input string.

Parameters:

`inputString - the String to analyze`

Returns:

`the value of the Topic enum with the highest count. (Note: in case of a tie, any value of the enum can be returned.)`

III. Create additional controls on the GUI

*The starting view of **SocialMediaGUI***

*The modified view of **SocialMediaGUI***

The button you create will have to start a new Thread.

IV. Background on Processes and Threads[1]

A process runs independently and isolated of other processes. It cannot directly access shared data in other processes. The resources of the process, e.g. memory and CPU time, are allocated to it via the operating system.

A thread is a so-called lightweight process. It has its own call stack but can access shared data of other threads in the same process. Every thread has its own memory cache. If a thread reads shared data, it stores this data in its own memory cache. A thread can re-read the shared data. A process runs independently and isolated of other processes. It cannot directly access shared data in other processes. The resources of the process, e.g. memory and CPU time, are allocated to it via the operating system.

A Java application runs by default in one process. Within a Java application you work with several threads to achieve parallel processing or asynchronous behavior.

The base means for concurrency are is the java.lang.Threads class. A **Thread** executes an object of type java.lang.Runnable. **Runnable** is an interface with defines the *run*() method. This method is called by the Thread object and contains the work which should be done. Therefore the "Runnable" is the task to perform. The Thread is the worker who is doing this task.

[1] https://www.vogella.com/tutorials/JavaConcurrency/article.html

startFeed

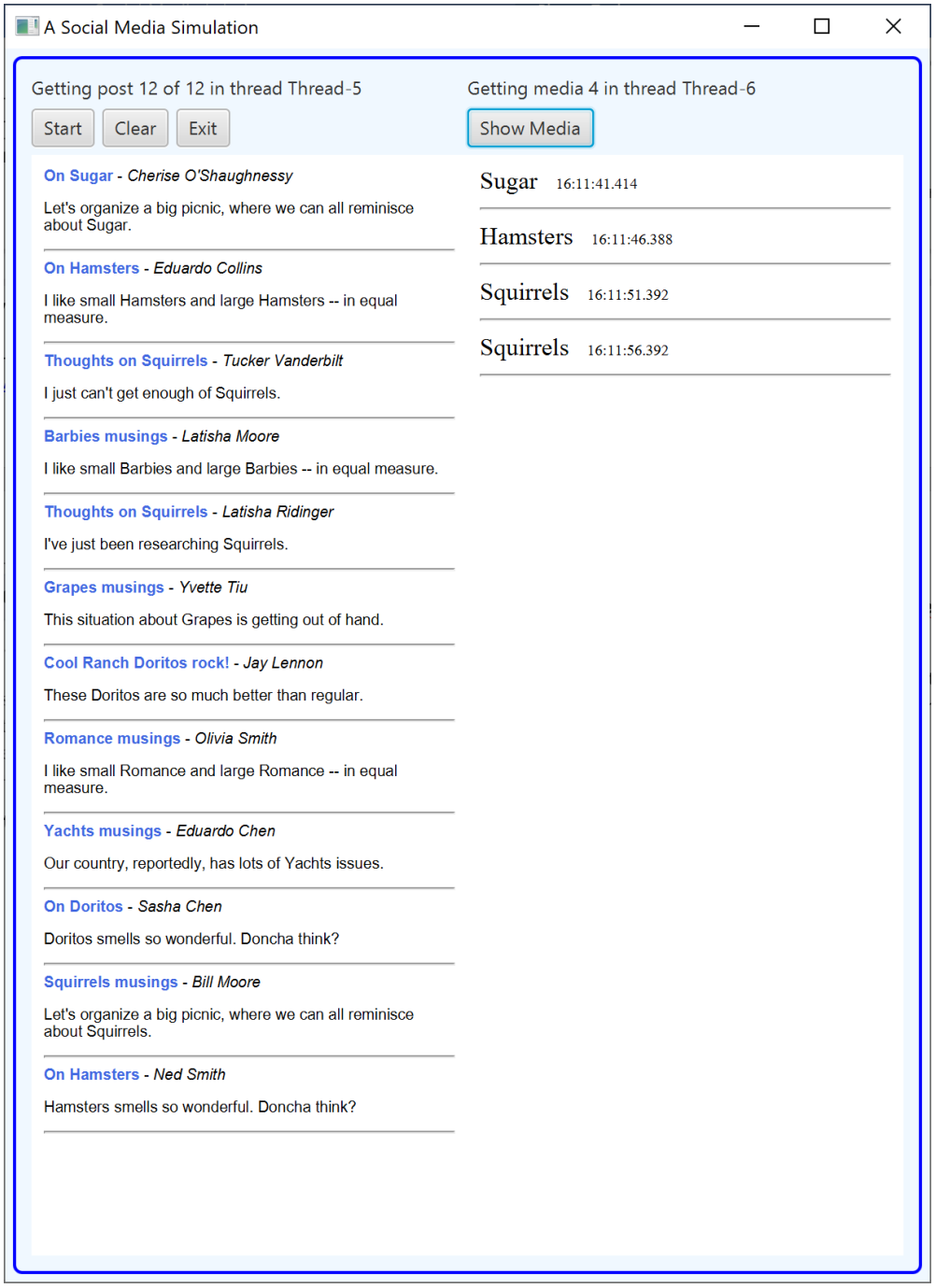
```
private void startFeed()
```

This is the method that will run as the action when the Start button is pressed. It will start a new Thread that will act independently of the Thread on which the GUI is running.

Examine the method *startFeed*() in **SocialMediaGUI**.

This shows how a new Thread is spawned to display the Sample posts. You will need to do something similar to start the feed of media items when the Show Media button is pressed.

V. A first attempt at the Media Feed



To test your Media Feed, you can get to this intermediate step.

Here four times during the post feed, our new Thread indicates that it recognized the most often mentioned topic.

At 16:11:41, it saw only post 1 on sugar, and it added the word **Sugar** to the media feed.

About five seconds later, at 16:11:46, it saw the first three posts, and it determined that **Hamsters**, which was mentioned three times, was now the most popular topic.

At 16:11:51, it saw Latisha Ridinger's post, which upped the count of **Squirrels** to four. Making it the new most popular topic – which was also the case at 16:11:56.

Note that the WebView accepts HTML markup. Each successive item in the media feed was tagged thusly:

```
"<span style=
'font-size: x-small;'>"
+ LocalTime.now() +
"</span><hr />"
```

The tag shrunk the font, and the <hr /> tag added a horizontal line.

VI. Adding in actual media items

The application will present different types of media items. All media items will have a title and a description. Some of the media items will be file-based. In these cases, they will also have a file object (java.io.File) that can be created when one passes in a String for the filename. Songs and videos are playable, although you do not have to provide a working implementation; i.e., the play method can have a body with the comment `// to be coded later`. Also, playable objects must all be able to return their length. Sponsorable media must be able to return a URL using a *url()* method. Please build these with good object-oriented principles.

Media Item	**Other information**	**Are the media file-based?**	**Are the media sponsorable?**	**Possible HTML renderings**

Ad	URL to direct customers to the advertiser website (java.net.URL)	No	Yes	`<h2>`*title*`</h2>`
				`<p>`*description*`</p>`
				
				hyperlinks can be coded as an HTML String (where url is the URL object)
				`"<a href='" + url.toURI() + "'>" + url + "</a>"`

Event	The location and date of the event.	No	Not yet	`<h2>`*title*`</h2>` `<p>`*description*`</p>`

Image		Yes	No	`<h2>`*title*`</h2>`
				
				Images can be coded as an HTML String (where file is the File object)
				`"<figure><img src='" + file.toURI() + "' />" +`
				`"<figcaption>" + description + "</figcaption></figure>"`

Song	The artist of the song.	Yes	Not yet	`<h2>`*title*`</h2>`
				`<p>`*description*`</p>`
				
				Audio can be coded as an HTML String (where file is the File object)
				`"<audio controls autoplay>`
				`<source src='" + file.toURI() +`
				`"' type='audio/mp3' />" +`
				`"Your browser does not support the audio`
				`element.</audio>" + "by " + artist`

Video		Yes	Not yet	`<h2>`*title*`</h2>`
				`<p>`*description*`</p>`
				
				Video can be coded as an HTML String (where file is the File object)
				`" <video width='300' autoplay>`
				`<source src='" + file.toURI() +`
				`"' type='video/mp4' />" +`
				`"Your browser does not support the video`
				`element.</video>`

You should make a media item for each topic in the enum. Set it up so that a new media item appears every 20 seconds. (You might want to slow your post feed down to show a new post every 5 to 6 seconds.) Have the media feed only produce four postings.

Be aware that this GUI will use event driven processes, so someone might press the Show Media button before they press the Start button. Then there will be no post feed to analyze. How will you manage this?

VII. Background on WebViews[2]

The JavaFX **WebView** (javafx.scene.web.WebView) component is capable of showing web pages (HTML, CSS, SVG, JavaScript) inside a JavaFX application. As such, the JavaFX WebView is a mini browser. The WebView component is very handy when you need to show documentation (e.g. Help texts), news, blog posts or other content which needs to be downloaded from a web server at runtime.

The JavaFX WebView uses the WebKit open source browser engine internally to render the web pages.

The JavaFX WebView **WebEngine** (javafx.scene.web.WebEngine) is an internal component used by the WebView to load the data that is to be displayed inside the WebView. To make the WebView WebEngine load data, you must first obtain the WebEngine instance from the WebView.

You can use the WebEngine (as in "*engine*" below) to retrieve the contents of the WebView, this way:

```
String content = (String) engine.executeScript("document.documentElement.outerHTML");
```

This will retrieve the content of the WebView; the *executeScript*() method will return an Object, which you can safely cast into a String. Of course, that String will contain the text with the HTML markup. (*Hint: Maybe the Tokenizer can help you strip out the HTML tags!*)

[2] http://tutorials.jenkov.com/javafx/webview.html

VIII. A MediaCollection class and Exceptions

You should make a **MediaCollection** class that has a HashMap<Topic, MediaItem> named *media.*
The constructor for this class should create the media collection that you need for this lab. There should be one entry in the HashMap for every Topic. As an example, here is how you might handle **Bacon**:

```
this.media.put(Topic.Bacon, new Image("Bacon", "a slice", "bacon.jpg"));
```

In the constructor of **Ad**, you will need to turn accept a String parameter and make a URL object with it. To do this, you can use the constructor from java.net.URL as shown on the right.

URL

```
public URL(String spec)
    throws MalformedURLException
```

Creates a URL object from the `String` representation.

This constructor is equivalent to a call to the two-argument constructor with a `null` first argument.

Parameters:

`spec` - the String to parse as a URL.

Throws:

`MalformedURLException` - if no protocol is specified, or an unknown protocol is found, or spec is null.

See Also:

`URL(java.net.URL, java.lang.String)`

However, this constructor throws a checked Exception that you will have to deal with.

For this lab, you don't want to deal with this exception in the **Ad** class, nor in your **MediaCollection** class. Where you want to deal with it is in your GUI class **SocialMediaGUI**.

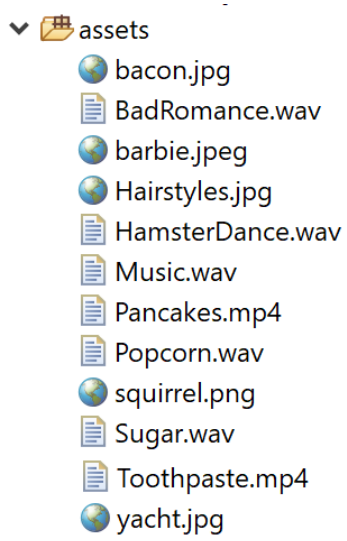
When you finally catch the MalformedURLException, you can just print something to the console.

IX.

If you want to work with media files in Eclipse, the easiest way is to make an "assets" folder. (Right click on the project and add a new Folder. Name it *assets*.)

To help you out, I have created some media files for you which you should put in your assets folder:

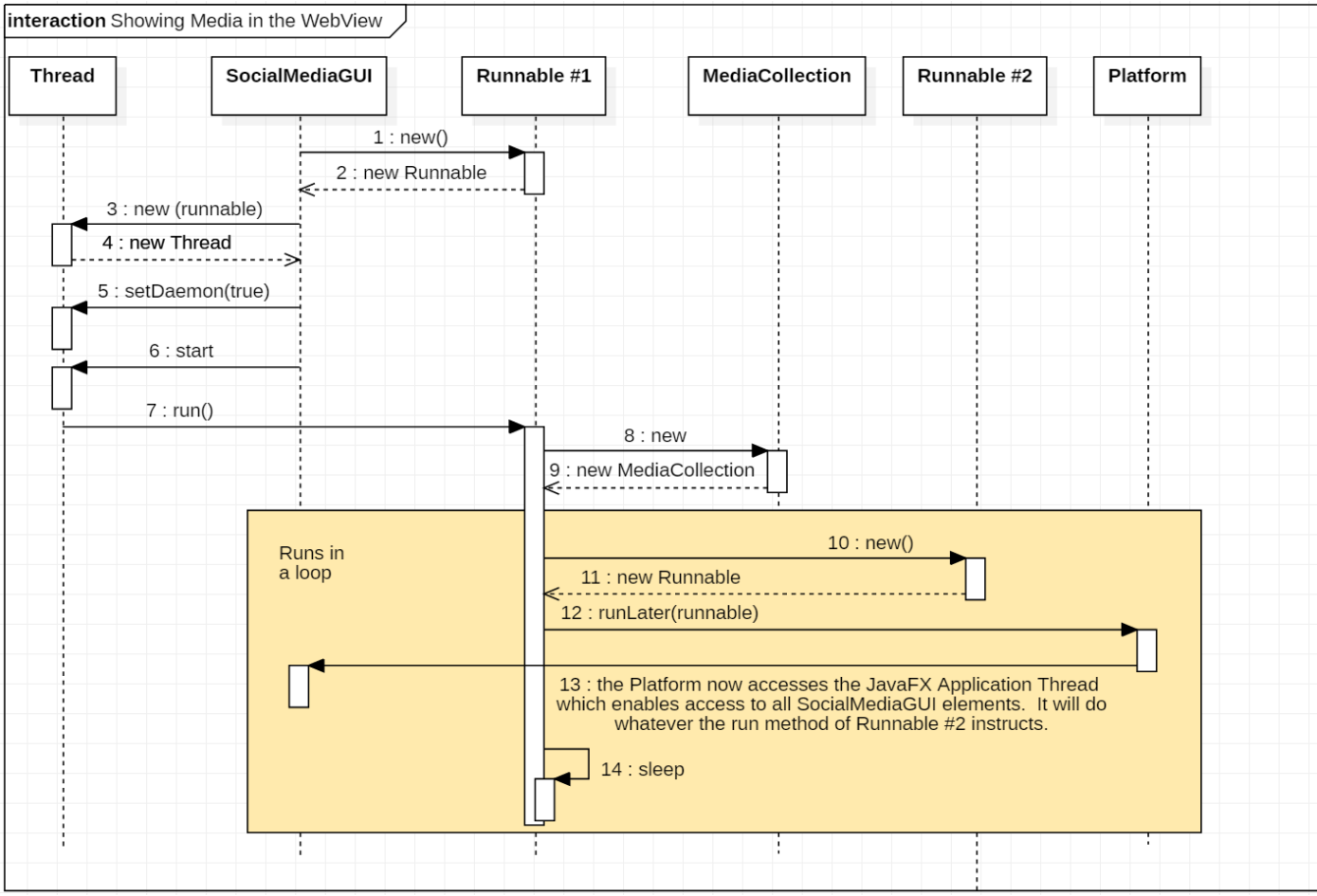
Topic	Type of Media Item	File (can be accessed in Eclipse by filename "assets/*filename*")
Bacon	Image	bacon.jog.
Barbies	Image	barbie.ppeg
Doritos		
Grapes		
Hairstyles	Image	Hairstyles.jpg
Hamsters	Song	HamsterDance.wav
Investments		
Music	Song	Music.wav
Pancakes	Video	Pancakes.mp4
Pizza		
Popcorn	Song	Popcorn.wav
Romance	Song	BadRomance.wav
Squirrels	Image	squirrel.png
Sugar	Song	Sugar.wav
Superman		
Toothpaste	Video	Toothpaste.mp4
Travel		
Volkswagons		
Yachts	Image	yacht.jpg



the Topics without a media file can be Ads or Events. Feel free to use these assets or substitute your own.

X. What to turn in

- A UML class diagram showing all the classes and interfaces. (Enums need not be shown.)
- Fully Javadocced source code. The Sequence Diagram below describes what is really going on at the Thread level for your completed GUI



Note that Threads close themselves after the run method has finished.