Linked List Lab

After finishing each part of the lab, copy your entire project and work on the copy for the next part!

Part 1: Create an `Actor` class with a link.

- Make a simple `Actor` class containing a name & a *self-referential* field to the next `Actor`.

```
public class Actor
--------------------------------------------------------------------------
         Actor()                 // be sure to set the next field to null!
         Actor(String name)      // be sure to set the next field to null!
  String  getName()
    void  setName(String name)
   Actor  getNextPtr()           // get a reference to the next Actor in the linked list
    void  setNextPtr(Actor next) // set the reference to the next Actor in the linked list
  String  toString()             // return a formatted string suitable for printing
```

- Make a class `NodeTester`, containing a *main* method.
- The 1st part of *main* loads data into a linked list:
  - Hard code building an `Actor` object with an input string for the name.
  - Copy the *head* pointer into the new actor's *next* pointer
  - Change the *head* pointer to point to the new actor
  - Repeat these steps 4 times, giving you a list of 5 actors (in reverse order, since you inserted each new object at the head of the list).
- The 2nd part of *main* prints the data from the linked list:
  - Set a *current* pointer to point to the *head*
  - Use a while loop to traverse the list (while *current* is not `null`)
    - Get the actor object & print it
    - Set the current pointer to current's *next* value
- The last part of *main* prints the name from the first node on linked list, using the *toString* method of the `Actor` class

- Test your program by running the *main* method.

Part 2: Create an `ActorLinkedList` class to manage the linked actor objects.

- Leave the `Actor` class unchanged.
- Make an `ActorLinkedList` class
  - Make a field *head* (type `Actor`)
  - Make a field *count* (type `int`)
- Constructor: set fields to `null` and 0
- The *add* method adds a new `Actor` to the beginning of the list (be sure to increment count)
- The *get* method returns a pointer to the actor object at the index requested

```
public class ActorLinkedList
-------------------------------------------------------------------------
         ActorLinkedList()       // be sure to set the head to null!
   void  add(Actor actor)        // adds actor to the beginning of the list
  Actor  get(int index)          // get a reference to the Actor at the position given,
                                 //    returns null if the index is invalid
    int  size()                  // returns number of actors on list
```

- Make a class `ListTester`, containing a *main* method.
- The 1st part of *main* loads hard coded data into a linked list:
  - Make an `ActorLinkedList` object.
  - Add 5 actors to the list, using the *add* method in your `ActorLinkedList` object.
- The 2nd part of *main* prints the data from the linked list:
  - Use a loop to traverse the list (using the *size* method to limit repetitions)
    - Get the actor object & print it
- The last part of *main* prints the name from the first node on the linked list.
  - Use the *get(0)* method of the `ActorLinkedList` object to get the Actor object saved as the head node
  - Use the `Actor`'s *getName* method to retrieve the name for printing

- Test your program by running the *main* method in `ListTester`.

Part 3: Refactor the `Actor` class into a simplified `Actor` and a `Node` class.

- Make a `Node` class
  - Copy the code from the `Actor` class into the new `Node` class
  - Delete the *name* field and replace it with a *data* field (type `Object`)
  - Change the *next* field so it points to a `Node` object
  - The no argument constructor should `null` both field values
  - Change the single argument constructor so it takes a single `Object` parameter & sets the *data* field (and nulls the *next* pointer).
  - Change *getName* & *setName* to *get* & *set*. These methods should now operate on `Object` types.
  - Change the *getNextPtr* & *setNextPtr* methods to operate on `Node` types.
  - Change *toString* so that it returns the *toString* result of the *data* object.

```
public class Node
-------------------------------------------------------------------------
         Node()                  // be sure to set the next field to null!
         Node(Object data)       // be sure to set the next field to null!
   void  set(Object data)        // sets the data field to the input object
 Object  get()                   // get a reference to the data object
   void  setNextPtr(Node next)   // sets the next field to the input value
   Node  getNextPtr()            // get a reference to the next node
 String  toString()              // returns the toString value of the enclosed data's toString
```

- Modify the `Actor` class
  - Delete the *next* field, and references to it in the constructors.
  - The no argument constructor should `null` the name field.
  - Delete the *getNextPtr* & *setNextPtr* methods.

```
public class Actor
-------------------------------------------------------------------------
         Actor()                 // set the name field to null
         Actor(String name)
 String  getName()
   void  setName(String name)
 String  toString()              // return a formatted string suitable for printing
```

- **Now, `Node` objects can refer to any type as *data*, without changing the `Node` class.**

- Modify the `NodeTester` class:
- The 1st part of *main* loads data into a linked list:
  - Hard code building an `Actor` object with an input string for the name.
  - Hard code building a `Node` object using the `Actor` object for the data.
  - Copy the *head* pointer into the new node's *next* pointer
  - Change the *head* pointer to point to the new node
  - Repeat these steps 4 times, giving you a list of 5 `Node` objects, each containing an `Actor` object (in reverse order, since you inserted each new object at the head of the list).

- The 2nd part of *main* prints the data from the linked list:
    - Set a *current* pointer to point to the *head*
    - Use a while loop to traverse the list (while *current* is not `null`)
        - Get the actor object & print it
        - Set the *current* pointer to current's *next* pointer
- The last part of *main* prints the name from the first node on the linked list.
    - Use the `Node`'s *get* method to get the data from the head node
    - Use the `Actor`'s *getName* method to retrieve the name for printing (see NOTE)

- Test your program by running the *main* method.

NOTE: Because your `Node` object now holds an `Object` (an `Actor` underneath, but `Node` thinks it's an `Object`), you will have to cast the returned *data* object to an `Actor` before running `Actor` methods.

Part 4: Refactor the `ActorLinkedList` class into a `LinkedList` class.

- Open `ActorLinkedList`
  - Change the class name and constructor to `LinkedList`
  - The *head* should reference a `Node` instead of `Actor`
  - Your *add* method will take an `Object` parameter instead of an `Actor`. You will also have to make a `Node` object and use the parameter to set the `Node`'s *data* field.
  - In your *get* method, change pointer references to `Node` objects. Remember to return the `Actor` object referenced by the *data* field (don't return the `Node` object).

```
public class LinkedList
--------------------------------------------------------------------------------
          LinkedList()            // be sure to set the head to null!
    void  add(Object data)        // adds an object to the beginning of the list
                                        // for now, this is an Actor object
  Object  get(int index)          // get a reference to the data object at the position given,
                                        // NOT the Node object!
                                        // Returns null if the index is invalid
     int  size()                  // returns number of objects on the list
```

- Modify the class `ListTester`, containing a *main* method.
- The 1st part of *main* loads hard coded data into a linked list:
  - Make a `LinkedList` object.
  - Add 5 actors to the list, using the *add* method in your `LinkedList` object.
    - The *add* method expects an `Object` – be sure it's an `Actor` object, not a `String`!
- The 2nd part of *main* prints the data from the linked list:
  - Use a loop to traverse the list (using the *size* method to control repetitions)
    - Get the actor object & print it
- The last part of *main* prints the name from the first node on the linked list.
  - Use the *get(0)* method of the `LinkedList` object to get the `Actor` object saved as the head node
  - Use the `Actor`'s *getName* method to retrieve the name for printing (see NOTE)

- Test your program by running the *main* method in `ListTester`.

NOTE: Because your `Node` object now holds an `Object` (an `Actor` underneath, but `Node` thinks it's an `Object`), you will have to cast the returned *data* object to an `Actor` before running `Actor` methods.

Part 5: Read Actor data from a file.

- Modify the test code in `ListTester`'s *main* method to read file data…
- The 1st part of *main* loads file data into a linked list:
  - Make a `LinkedList` object.
  - Add the actors from the file to the list, using the *add* method in your `LinkedList` object.
    - The *add* method expects an `Object` – be sure it's an `Actor` object, not a `String`!

The remaining parts of *main* are unchanged…

- The 2nd part of *main* prints the data from the linked list
  - Use a loop to traverse the list (using the *size* method to control repetitions)
    - Get the actor object & print it
- The last part of *main* prints the name from the first node on the linked list.
  - Use the *get(0)* method of the `LinkedList` object to get the `Actor` object saved as the head node
  - Use the `Actor`'s *getName* method to retrieve the name for printing (see NOTE)

- Test your program by running the *main* method in `ListTester`.

NOTE: Because your `Node` object now holds an `Object` (an `Actor` underneath, but `Node` thinks it's an `Object`), you will have to cast the returned *data* object to an `Actor` before running `Actor` methods.

Part 6: Create a `Movie` class.

- Make a simple `Movie` class containing fields representing:
  - Date
  - Title
  - A list of actors (use your linked list)
  - A list of directors (use your linked list)

```
public class Actor
--------------------------------------------------------------------------
            Movie()
            Movie(int date, String title, LinkedList actors, LinkedList directors)
       int  getDate()
      void  setDate(int date)
    String  getTitle()
      void  setTitle(String title)
LinkedList  getActors()               // returns a reference to the list
      void  setActors(LinkedList actors)   // sets the field to a list
LinkedList  getDirectors()
      void  setDirectors(LinkedList actors)
    String  toString()            // return a formatted string suitable for printing
```

- Add test code in `ListTester`'s *main* method to test your `Movie` class…
- The 4th part of *main* (after the actor tests) loads hard coded data into a linked list:
  - Make a `LinkedList` object called *movie*s.
  - Add 5 hard coded movies to the list, using the *add* method in your `LinkedList` object.
    - The *add* method expects an `Object` – be sure it's a `Movie` object!
- The 5th part of *main* prints the data from the linked list:
  - Use a loop to traverse the list (using the *size* method to control repetitions)
    - Get the movie object & print it
- The last part of *main* prints the movie name from the first node on the linked list.
  - Use the *get(0)* method of the `LinkedList` object to get the `Movie` object saved as the head node
  - Use the `Movie`'s *getTitle* method to retrieve the name for printing (see NOTE)

- Test your program by running the *main* method in `ListTester`.

NOTE: Because your `Node` object now holds an `Object` (a `Movie` underneath, but `Node` thinks it's an `Object`), you will have to cast the returned *data* object to an `Movie` before running `Movie` methods.

Part 7: Read Movie data from a file.

- Modify the test code in `ListTester`'s *main* method to read file data…
- The 4th part of *main* (after the actor tests) should load file data into a linked list:
  - Make a `LinkedList` object called *movie*s.
  - For each line in the movie data file:
    - Parse each data section & build a movie object
    - Use your `LinkedList` *add* method to put the movie into the list.
    - The *add* method expects an `Object` – be sure it's a `Movie` object!

The remaining parts of *main* are unchanged…

- The 5th part of *main* prints the data from the linked list:
  - Use a loop to traverse the list (using the *size* method to control repetitions)
    - Get the movie object & print it
- The last part of *main* prints the movie name from the first node on the linked list.
  - Use the *get(0)* method of the `LinkedList` object to get the `Movie` object saved as the head node
  - Use the `Movie`'s *getTitle* method to retrieve the name for printing (see NOTE)

- Test your program by running the *main* method in `ListTester`.

NOTE: Because your `Node` object now holds an `Object` (a `Movie` underneath, but `Node` thinks it's an `Object`), you will have to cast the returned *data* object to an `Movie` before running `Movie` methods.

Part 8: Create an `IMDb` class.

- Read actor file data & create a linked list of actors.
- Read movie file data & create a linked list of movies.
- Traverse the actor list; for each actor
    - Print the actor name
    - Traverse the movie list
        - If the actor appears in the movie
            - Print the date & title of the movie
    - Be sure to print the movies most recent first