Due Feb 7, 2024 by 11:59pm **Points** 2 **Submitting** an external tool

Submission

Submission Details

Grade: 2 (2 pts possible)
Graded Anonymously: no

Comments:
No Comments

Lab 4 — More Interfaces and Inheritance

Home | Modules

Lab 4: Interfaces (More) & Inheritance

Artists

1.1 Introduction

The purpose of this lab is to give you more practice with the concepts of interfaces, inheritance, typing and subclassing. We'll also continue to use polymorphism as you build out more intricate class hierarchies.

1.2 What to do

A non-profit organization is creating a digital index of all artists who have some connection to the area. You'll help develop part of the solution. Here's the outline: An `Artist` can be any of these:

- An Actor
- A Musician
- A Poet

All Artists maintain the following information:

- name, a String containing information about an Artist's first and last name (throw an IllegalArgumentException if the name is null or an empty string)
- age, which is an integer in the range [0..128], containing information an Artist's age (throw an IllegalArgument exception if the age is outside of this range)
- genres, which is an array of Strings, representing an Artist's genres
- awards, which is an array of Strings, representing all awards that an Artist received

Additionally,

- All Actors keep track of the movies they've acted in (a String array)
  - Aside from the shared attributes, this should be another parameter for the Actor constructor.
  - Also the toString method for Actor should follow the format provided below:
    My name is Samuel L. Jackson
    My age is 73
    I am an ACTOR
    I make these types of movies: [Action, SciFi, Drama]
    I have acted in these movies: [Star Wars, Captain America: Winter Soldier, Pulp Fiction]

- All Musicians keep track of their Recording company (a String) and the title of their current album (a String)
  - Aside from the shared attributes, these should be added parameters to the Musician constructor, where the current album must precede the record company in the constructor.
  - Also the toString method for Musician should follow the format provided below:
    My name is The Weeknd
    My age is 32
    I am an MUSICIAN
    I make these types of music: [R&B, Pop]
    My current album is: Dawn FM
    My recording company is: XO

- All Poets keep track of the publisher (a String) that publishes their books/work
  - Aside from the shared attributes, this should be the another parameter for the Poet constructor.
  - Also the toString method for Poet should follow the format provided below:
    My name is Maya Angelou
    My age is 86
    I am an POET
    I make these types of poems: [Autobiographical Fiction]
    My publishing company is: Random House

Finally, all Artists can receive an award, which gets added to their current array of awards, and the method signature for that behavior is: `receiveAward(String award)`.

All of your artists should implement the `IArtist` interface (given below). We're also giving you some "starter code" in the form of an `AbstractArtist`.

Be sure to place all of your **solution assets** in a **package** named `artists`

All of your **test assets** should be in the **default package**

```
package artists;

public interface IArtist {
  void receiveAward(String award);
  String [] getAwards();
}
```

AbstractArtist code:

```
package artists;

import java.util.Arrays;

/**
 *  AbstractArtist class.
 *  This is the abstract superclass of all Artist concrete classes. It implemen
ts the
 *  IArtist interface and serves as the "repository" of reuse code for certain
common services.
 */
public abstract class AbstractArtist implements IArtist {
  private String name;
  private int age;
  private String [] genres;
  private String [] awards;

  /**
   * AbstractArtist constructor.
   * This class cannot be instantiated, but this creation mechanism ensures the
superclass
   * portion of all subclasses is appropriately initialized.
   * @param name : name of the artist.
   * @param age : age of the artist.
   * @param genres : genres of art the artist develops.
   * @param awards : awards earned by the artist.
   */
  public AbstractArtist(String name, int age, String [] genres, String [] award
s) throws IllegalArgumentException {
    if(age > 128 || age < 0 || name == null || name.length() == 0) {
      throw new IllegalArgumentException();
    }
    this.name = name;
    this.age = age;
    this.genres = genres;
    this.awards = awards;
  }

  /**
   * This method answers the age of the artist. This is a protected method and
not intended to be
   * used by clients outside of the hierarchy.
   * @return (int)
   */
  protected int getAge() {
    return this.age;
  }

  /**
   * This method answers the full name of the artist. It is a protected getter
for subclass use
   * @return (String)
   */
  protected String getName() { return this.name; }

  /**
   * This method answers an array representing the genres the artist creates fo
r.
   * @return (String [])
   */
  protected String [] getGenres() { return this.genres;}

  /**
   * This method answers the genres as a single string that is aggregated from
the array.
   * @return (String)
   */
  protected String getGenresAsSingleString() {
    if(this.genres == null || this.genres.length == 0) {
      return "";
    }
    return Arrays.toString(this.genres); // convert the array to a string
  }

  /**
   * Answers the awards earned by the artist.
   * @return (String [])
   */
  public String [] getAwards() { return this.awards;}

  /**
   * This method adds another award to the list of awards won by the artist.
   * @param award (String)
   */
  public void receiveAward(String award) {
    int size = this.awards.length; // get the current size of the array

    // Iterate through the current array, copy the values, then add the new val
ue
    String [] updatedAwards = new String[size + 1];
    for (int i=0; i<size; i++) {
      updatedAwards[i] = this.awards[i];
    }
    updatedAwards[size] = award;
    this.awards = updatedAwards;
  }

  /**
   * Override of the toString() method that answers the basic information held
by the AbstractArtist.
   * @return (String)
   */
  public String toString() {
    return "My name is " + this.name + "\n" + "My age is " + this.age + "\n";
  }
}
```

Here are some local tests you can use to validate your code. Write your own JUnit tests as well, since these are not exhaustive.

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.*;

import artists.IArtist;
import artists.Actor;
import artists.Musician;
import artists.Poet;

public class IArtistTest {

  String[] genre1 = {"Action", "SciFi", "Drama"};
  String[] genre2 = {"Rock", "Rock-Soul"};
  String[] genre3 = {"Comedy", "Romantic Comedy"};
  String[] genre4 = {"R&B", "Pop", "Rap"};
  String[] genre5 = {"Autobiographical Fiction"};

  String[] awards = {"Academy Award", "Golden Globe"};
  String[] awards2 = {"Pulitzer"};
  String[] awards3 = {"Emmy", "People's Choice"};
  String[] awards4 = {"Grammy", "American Music Award"};
  String[] awards5 = {"Grammy", "Billboard"};

  String[] movies = {"Glory", "Flight", "Training Day", "Book of Eli", "Fence
s"};
  String[] movies3 = {"Bridesmaids", "Tammy", "Life of the Party", "Ghostbuster
s"};

  IArtist denzel;
  IArtist melissa;
  IArtist musician;
  IArtist musician2;
  IArtist poet;

  @BeforeEach
  public void setUp() throws IllegalArgumentException {
    denzel = new Actor("Denzel Washington", 67, genre1, awards, movies);
    melissa = new Actor("Melissa McCarthy", 52, genre3, awards3, movies3);

    musician = new Musician("Bruce Springsteen", 73, genre2, awards4,
        "Only the Strong Survive", "Columbia Records");

    musician2 = new Musician("Lizzo", 34, genre4, awards5,
        "Special", "Atlantic Records");
    poet = new Poet("Maya Angelou", 86, genre5, awards2, "Random House");
  }

  @Test
  public void testCreatedInstances() {
    String test = """
        My name is Denzel Washington
        My age is 67
        I am an ACTOR
        I make these types of movies: [Action, SciFi, Drama]
        I have acted in these movies: [Glory, Flight, Training Day, Book of El
i, Fences]""";
    assertTrue(denzel.toString().equalsIgnoreCase(test));

    test = """
        My name is Melissa McCarthy
        My age is 52
        I am an ACTOR
        I make these types of movies: [Comedy, Romantic Comedy]
        I have acted in these movies: [Bridesmaids, Tammy, Life of the Party, G
hostbusters]""";
    assertTrue(melissa.toString().equalsIgnoreCase(test));

    test = """
        My name is Bruce Springsteen
        My age is 73
        I am an MUSICIAN
        I make these types of music: [Rock, Rock-Soul]
        My current album is: Only the Strong Survive
        My recording company is: Columbia Records""";
    assertTrue(musician.toString().equalsIgnoreCase(test));

    test = """
        My name is Lizzo
        My age is 34
        I am an MUSICIAN
        I make these types of music: [R&B, Pop, Rap]
        My current album is: Special
        My recording company is: Atlantic Records""";
    assertTrue(musician2.toString().equalsIgnoreCase(test));

    test = """
        My name is Maya Angelou
        My age is 86
        I am an POET
        I make these types of poems: [Autobiographical Fiction]
        My publishing company is: Random House""";
    assertTrue(poet.toString().equalsIgnoreCase(test));
  }

  @Test
  public void testReceiveAward() {
    String[] testAwards = {"Pulitzer", "Tony"};
    poet.receiveAward("Tony");
    assertArrayEquals(poet.getAwards(), testAwards);
  }

  @Test
  public void testGetAwards() {
    String[] testAwards = {"Academy Award", "Golden Globe"};
    assertArrayEquals(denzel.getAwards(), testAwards);
  }

  @Test
  public void testBadAge() {
    assertThrows(IllegalArgumentException.class,
        () -> new Musician("Bruce Springsteen", 129, genre2, awards4,
            "Only the Strong Survive", "Columbia Records"));
  }

  @Test
  public void testBadAge2() {
    assertThrows(IllegalArgumentException.class,
        () -> new Musician("Bruce Springsteen", -1, genre2, awards4,
            "Only the Strong Survive", "Columbia Records"));
  }

  @Test
  public void testBadName() {
    assertThrows(IllegalArgumentException.class,
        () -> new Musician(null, 10, genre2, awards4,
            "Only the Strong Survive", "Columbia Records"));
  }

  @Test
  public void testBadName2() {
    assertThrows(IllegalArgumentException.class,
        () -> new Musician("", 10, genre2, awards4,
            "Only the Strong Survive", "Columbia Records"));
  }
}
```

Notes

For each method you write:

- Design the signature of the method.
- Write Javadoc-style comments for that method.
- Write the body for the method.
- Write one or more tests that check that the method works as specified in all cases.

Feel free to create private “helper” methods if you need to do so.

Be sure to use access modifiers, private, *default* (no keyword), protected, and public appropriately.

Include JavaDoc for your classes and constructors as appropriate. You do not need to repeat JavaDoc already existing in a superclass or interface when you override a method. (This is true for the course in general.)

No UML class diagram is required for this lab, but we encourage you to create one for yourself to help with your forward-design, and then (if you wish) use IntelliJ to "reverse-engineer" your code for a system-generated UML diagram.

This tool needs to be loaded in a new browser window

Load Lab 4: Interfaces & Inheritance in a new window