

Midterm One Practice Paper – Spring 2023

This practice exam will help you prepare for the first midterm. There is an appendix at the end with methods that may be helpful. You will not be allowed any cheat sheet or other extra materials for the exam.

Total [100 marks, 110 minutes]

Question 1: [10 marks]

Explain the following terms/concepts (The real paper will take about 5 from this list):

Encapsulation, Polymorphism, Inheritance, Overloading, Overriding, Dependency, Aggregation, Association, Composition, Unit Testing, Instance, Enumerated Type, static, Declared type vs Actual type, abstract

What does the protected visibility keyword do?

Question 2: [10 marks]

```
/**
 * This StringTools class contains some useful methods that work with Strings.
 */
public class StringTools {

    /**
     * Return the count of consonants minus the count of vowels (a, e, i, o, u)
     * @param s The string to analyze. If it is null, an exception will be thrown
     * @return The calculated number. If the parameter does not contain a string with
     characters, returns 0.
     */
    public static int consMinusVowels(String s) {...}

}
```

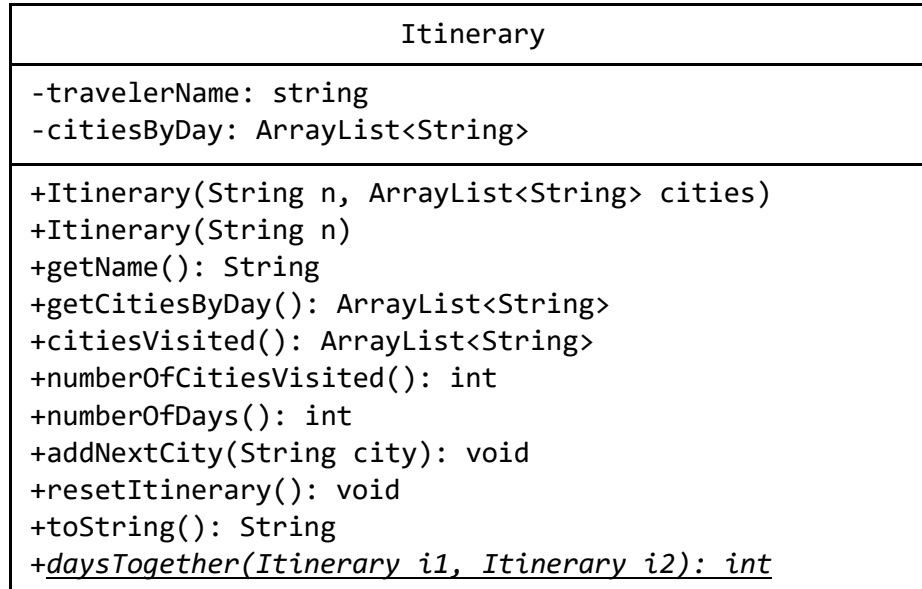
What test cases should be run to test this method? DO NOT WRITE A FULL JUNIT TEST METHOD OR ASSERTIONS.

Write only the argument values to pass in for the test cases. Write a few words about each test case explaining why it is important.

INPUT	EXPLANATION
"bed"	

Question 3: [50 marks]

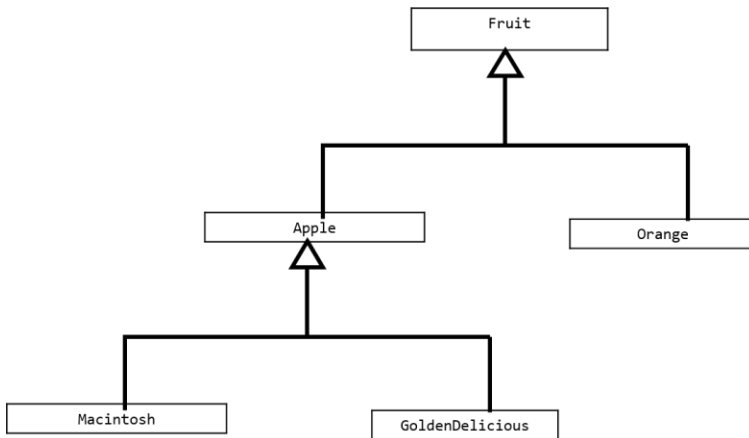
Implement the class specified by the following UML diagram with the specifications that follow. It represents a road trip itinerary for a traveler. It will contain two data members, the traveler's name and the list of cities they will be in. Each element in citiesByDay represents one day. A city may appear multiple times in the list.



- The class contains two data members
- Any time a parameter value is not usable, throw an exception
- Create two constructors. One sets the name, the other sets the name and provides an arraylist of cities.
- `getName`, `getCitiesByDay` return the appropriate data
- `citiesVisited` returns a list of unique cities visited from the `citiesByDay` list, but with duplicates removed
- `numberOfCitiesVisited` returns the number of unique cities visited
- `numberOfDays()`: returns how many days long the itinerary is
- `+addNextCity(String city)`: adds the city to the end of the `citiesByDay` list
- `+resetItinerary()`: empties the `citiesByDay` list
- `+toString()`: output the name and `citiesByDay` list in a reasonable format
- `+daysTogether(Itinerary i1, Itinerary i2)`: A static method that given two itineraries, calculates on how many days both travelers will be in the same city on the same day. {"Vancouver", "Vancouver", "Victoria"} and {"Nanaimo", "Vancouver", "Vancouver", "Victoria"} would be 1 because they only match on the second day.

Question 4: [20 marks]

Given the following diagram, answer the following questions:



Given the declaration:

```
Fruit fruit = new GoldenDelicious();
```

```
Orange orange = new Orange();
```

- a. Is fruit instanceof Fruit true?
- b. Is fruit instanceof Orange true?
- c. Is fruit instanceof Apple true?
- d. Is fruit instanceof GoldenDelicious true?
- e. Is fruit instanceof Macintosh true?
- f. Is orange instanceof Orange true?
- g. Is orange instance of Fruit true?
- h. Is orange instance of Apple true?
- i. Suppose Apple contains a method called makeApple(). Can fruit invoke makeApple()? Can orange invoke makeApple()?
- j. Suppose makeOrangeJuice() is defined in Orange. Can orange invoke this method? Can fruit invoke this method?
- k. Is the statement Orange p = new Apple() legal?
- l. Is the statement Macintosh p = new Apple() legal?
- m. Is the statement Apple p = new Macintosh() legal?

Question 5: [10 marks]

```
public abstract class Animal {  
    public String action() {  
        return "lives";  
    }  
  
    public String toString() {  
        return "This animal " + this.action();  
    }  
}  
  
public class Bird extends Animal {  
    private boolean flies;
```

```

    public Bird(boolean canFly) {
        super();
        this.flies = canFly;
    }

    public boolean canFly() {
        return this.flies;
    }

    public String action() {
        String result = super.action();
        if (this.flies) {
            result = result + " and flies";
        }
        return result;
    }
}

public class Parrot extends Bird {
    public Parrot() {
        super(true);
    }

    public String action() {
        return super.action() + " and talks";
    }
}

public class Coral extends Animal {
    public Coral() {
        super();
    }

    public String toString() {
        return "This can't be an animal";
    }
}

```

For each of the following pieces of code, either write what will be displayed to the console **or** if the code would create an error, **clearly** describe the cause of the error. [2 marks each]

```

Animal s = new Bird(true);
System.out.println(s.canFly());

```

↑ Error, Animal doesn't have the canFly() method

```

Parrot p = new Bird(true);
System.out.println(p.action());

```

← Error, cannot typecast

```
Animal c = new Coral();
System.out.println(c);
```

This can't be an animal

```
Bird m = new Parrot();
System.out.println(m);
```

This animal lives, and flies and talks

```
Animal d = new Bird(false);
System.out.println(d);
```

This animal lives

Appendix

An abbreviated list of the methods of different classes we have used in this course.

The ArrayList class

Constructors	
	ArrayList<E>() Creates an empty ArrayList with initial capacity of ten.
	ArrayList<E>(ArrayList<E> a) Creates a copy of the parameter ArrayList.
Modifier and Type	Method and Description
boolean	add(E e) Appends the specified element to the end of this list.
void	add(int index, E element) Inserts the specified element at the specified position in this list.
boolean	contains(Object o) Returns true if this list contains the specified element.
E	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	isEmpty() Returns true if this list contains no elements.
int	lastIndexOf(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
E	remove(int index)

	Removes the element at the specified position in this list.
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present.
E	set(int index, E element) Replaces the element at the specified position in this list with the specified element.
int	size() Returns the number of elements in this list.
String	toString() Returns a string representation of this collection. The string representation consists of the collection's elements in order, enclosed in square brackets () "[]". Adjacent elements are separated by the characters " , ".

The String Class

char	<code>charAt(int index)</code> Returns the character at the specified index.
int	<code>compareTo(String anotherString)</code> Compares two strings lexicographically.
String	<code>concat(String str)</code> Concatenates the specified string to the end of this string.
boolean	<code>equals(String str)</code> Compares this String to another string.
boolean	<code>equalsIgnoreCase(String str)</code> Compares this String to another string ignoring case considerations.
int	<code>indexOf(int ch)</code> Returns the index within this string of the first occurrence of the specified character.
int	<code>indexOf(int ch, int fromIndex)</code> Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	<code>indexOf(String str)</code> Returns the index within this string of the first occurrence of the specified substring.
int	<code>indexOf(String str, int fromIndex)</code> Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
int	<code>lastIndexOf(int ch)</code> Returns the index within this string of the last occurrence of the specified character.
int	<code>lastIndexOf(int ch, int fromIndex)</code> Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	<code>lastIndexOf(String str)</code> Returns the index within this string of the rightmost occurrence of the specified substring.
int	<code>lastIndexOf(String str, int fromIndex)</code> Returns the index within this string of the last occurrence of the specified substring.

int	<code>length()</code> Returns the length of this string.
String	<code>replace(char oldChar, char newChar)</code> Returns a new string resulting from replacing all occurrences of <code>oldChar</code> in this string with <code>newChar</code> .
String	<code>substring(int beginIndex)</code> Returns a new string that is a substring of this string.
String	<code>substring(int beginIndex, int endIndex)</code> Returns a new string that is a substring of this string.
String	<code>toLowerCase()</code> Converts all of the characters in this <code>String</code> to lower case using the rules of the default locale, which is returned by <code>Locale.getDefault</code> .
String	<code>toUpperCase()</code> Converts all of the characters in this <code>String</code> to upper case using the rules of the default locale, which is returned by <code>Locale.getDefault</code> .