

# Polymorphism Lab 1

---

## Lab 1

This lab will focus on using polymorphism while interacting with a contact list. There will be a main class `Contacts` that controls the “view” the user sees and responds to user input. The contact information (personal and work) will be an instance of the `Information` class.

## The Contacts Class

This lab is built around the `Contacts` class, which has four attributes:

- \* `view` - This string attribute controls what the user sees. When the value for `view` changes, the information changes. There are four different views.
- \* `List view` - Shows the list of all of the contacts.
- \* `Information view` - Shows the work and personal information for a particular contact.
- \* `Add view` - Add information for a new contact.
- \* `Quit view` - Leave a message for the user and then end the program.
- \* `names` - `ArrayList` of strings that stores the names of each person in the contact list.
- \* `titles` - `ArrayList` of strings that stores the titles for each person in the contact list.
- \* `workPhoneNumbers` - `ArrayList` of strings that stores the work phone numbers for each person in the contact list.
- \* `workEmails` - `ArrayList` of strings that stores the work email addresses for each person in the contact list.
- \* `personalPhoneNumbers` - `ArrayList` of strings that stores the personal phone numbers for each person in the contact list.
- \* `personalEmails` - `ArrayList` of strings that stores the personal email addresses for each person in the contact list.
- \* `choice` - This string attribute represents input from the user and is used to change view.
- \* `index` - This integer attribute keeps track of the particular contact whose information is to be displayed.
- \* `length` - This integer attribute keeps track of the length of the above `ArrayLists`.

To put polymorphism into practice, we are going to create the abstract class `Information`. This class will have the abstract methods `displayInfo` and `addInfo`. The `Contacts` class inherits from `Information`, and therefore it must override the `displayInfo` and `addInfo` methods.

*//add class definitions below this line*

```
abstract class Information {  
    public abstract void displayInfo();  
    public abstract void addInfo();  
}
```

```
class Contacts extends Information {  
  
    public void displayInfo() {  
  
    }  
  
    public void addInfo() {  
  
    }  
}
```

*//add class definitions above this line*

Add the attributes and constructor for the Contacts class. For testing purposes, set view to "quit". We will change this to a more appropriate value later on. The other attributes do not need a value when instantiating the object. Instantiate each ArrayList attribute, set choice to null, and set index and length to 0.

```

//add class definitions below this line

abstract class Information {
    public abstract void displayInfo();
    public abstract void addInfo();
}

class Contacts extends Information {
    private String view;
    private ArrayList<String> names;
    private ArrayList<String> titles;
    private ArrayList<String> workPhoneNumbers;
    private ArrayList<String> workEmails;
    private ArrayList<String> personalPhoneNumbers;
    private ArrayList<String> personalEmails;
    private String choice;
    private int index;
    private int length;

    public Contacts() {
        view = "quit";
        names = new ArrayList<String>();
        titles = new ArrayList<String>();
        workPhoneNumbers = new ArrayList<String>();
        workEmails = new ArrayList<String>();
        personalPhoneNumbers = new ArrayList<String>();
        personalEmails = new ArrayList<String>();
        choice = null;
        index = 0;
        length = 0;
    }

    public void displayInfo() {

    }

    public void addInfo() {

    }
}

//add class definitions above this line

```

## The Display Method

The display method is designed to be a loop that runs until the user tells the program to end. The method checks the value of the view attribute and calls the appropriate method that displays the information for each view. Since the loop is while (true), be sure to include a break statement otherwise the loop would never stop (Java would eventually stop the program with an error message).

```
public void displayInfo() {  
  
}  
  
public void addInfo() {  
  
}  
  
public void display() {  
    while (true) {  
        if (view.equals("list")) {  
            showList();  
        } else if (view.equals("info")) {  
            displayInfo();  
        } else if (view.equals("add")) {  
            System.out.println();  
            addInfo();  
        } else if (view.equals("quit")) {  
            System.out.println("\nClosing the contact list...\n");  
            break;  
        }  
    }  
}
```

## Starting the Other Methods

The display method calls three other methods; displayInfo and addInfo have already been declared. Trying to test the code would cause your program to crash as the showList method has not yet been defined. Create the empty method showList just as was done with displayInfo and addInfo. We will come back later and add working code to each method.

```

public void displayInfo() {

}

public void addInfo() {

}

public void showList() {

}

```

## Testing Your Code

Before moving on to the next part of the script, we want to check that our code is working. To do that, instantiate a `Contacts` object and call the `display` method.

```

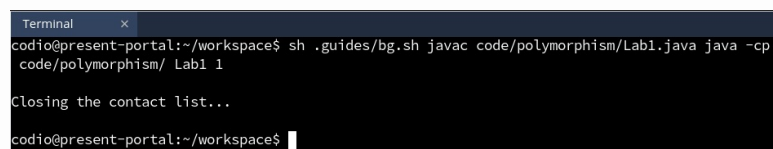
//add code below this line

Contacts contacts = new Contacts();
contacts.display();

//add code above this line

```

Run your program. Because the `view` attribute is "quit", the script should immediately display a message and then stop. Your output should look something like this.



```

Terminal
x
codio@present-portal:~/workspace$ sh .guides/bg.sh javac code/polymorphism/Lab1.java java -cp
code/polymorphism/ Lab1 1
Closing the contact list...
codio@present-portal:~/workspace$

```

Lab 1 Output

### ▼ Code

Your code should look like this:

```

import java.util.ArrayList;
import java.util.Scanner;

//add class definitions below this line

abstract class Information {

```

```

    public abstract void displayInfo();
    public abstract void addInfo();
}

class Contacts extends Information {
    private String view;
    private ArrayList<String> names;
    private ArrayList<String> titles;
    private ArrayList<String> workPhoneNumbers;
    private ArrayList<String> workEmails;
    private ArrayList<String> personalPhoneNumbers;
    private ArrayList<String> personalEmails;
    private String choice;
    private int index;
    private int length;

    public Contacts() {
        view = "quit";
        names = new ArrayList<String>();
        titles = new ArrayList<String>();
        workPhoneNumbers = new ArrayList<String>();
        workEmails = new ArrayList<String>();
        personalPhoneNumbers = new ArrayList<String>();
        personalEmails = new ArrayList<String>();
        choice = null;
        index = 0;
        length = 0;
    }

    public void displayInfo() {

    }

    public void addInfo() {

    }

    public void showList() {

    }

    public void display() {
        while (true) {
            if (view.equals("list")) {
                showList();
            } else if (view.equals("info")) {
                displayInfo();
            } else if (view.equals("add")) {

```

```

        System.out.println();
        addInfo();
    } else if (view.equals("quit")) {
        System.out.println("\nClosing the contact list...\n");
        break;
    }
}
}
}

//add class definitions above this line

public class Lab1 {
    public static void main(String[] args) {

        //add code below this line

        Contacts contacts = new Contacts();
        contacts.display();

        //add code above this line
    }
}

```

# Polymorphism Lab 2

---

## Adding a Contact

The first thing we need to do is change the default view when a `Contacts` object is instantiated. The list view should be the first view shown to the user. Change the value for `view` from `"quit"` to `"list"` in the constructor. The rest of the constructor should not be changed.

```
public Contacts() {  
    view = "list";  
    // rest of the constructor remains unchanged  
}
```

Next we want to modify the `showList` method to show the list of people in the contact list. There are two possible states for the list view: the list is empty or there are contacts in the list. When the list is empty, the user will be provided with the choice to add a contact or quit the program. Use a conditional to represent these two states. For now, set `view` to `"quit"` in the `else` branch.

The print statement is to add a blank line for legibility. We also need a scanner object to collect input from the user. If `length` is 0, then present the user with a choice. Store their input in `choice`. The `.toLowerCase()` will convert the user choice to a lowercase letter. This will make comparisons easier. Remember, Java is case sensitive; `q` and `Q` are not the same. By forcing all input to lowercase, we only need to test for the lowercase letter. The `showList` method ends by calling another method to handle the user's choice.

```
public void showList() {  
    System.out.println();  
    Scanner sc = new Scanner(System.in);  
    if (length == 0) {  
        System.out.print("(A)dd a new contact \n(Q)uit \n> ");  
        choice = sc.nextLine().toLowerCase();  
    } else {  
        view = "quit";  
    }  
    handleChoice();  
}
```



## Handling User Choices

Every time the user makes a choice, we want to evaluate that choice and perform the appropriate action. In this case, the user can choose between adding a contact or quitting the program. Notice that view only changes to "add" if "a" is entered **and** we are in list view. We only want to add new contacts from the list view.

```
public void handleChoice() {  
    if (choice.equals("q")) {  
        view = "quit";  
    } else if (choice.equals("a") && view.equals("list")) {  
        view = "add";  
    }  
}
```

## Adding a Contact

When the user enters "a", we need to create a new contact. To do this, we are going to modify the addInfo method. Create a scanner object and ask the user to enter the name, personal phone number, personal email, work title, work phone number, and work email. Each piece of information should go into the corresponding ArrayList attribute. Once the information has been added, increase the length attribute and revert back to the list view.

```

public void addInfo() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter their name: ");
    String name = sc.nextLine();
    names.add(name);

    System.out.print("Enter their personal phone number: ");
    String personalPhone = sc.nextLine();
    personalPhoneNumbers.add(personalPhone);

    System.out.print("Enter their personal email: ");
    String personalEmail = sc.nextLine();
    personalEmails.add(personalEmail);

    System.out.print("Enter their work title: ");
    String title = sc.nextLine();
    titles.add(title);

    System.out.print("Enter their work phone number: ");
    String workPhone = sc.nextLine();
    workPhoneNumbers.add(workPhone);

    System.out.print("Enter their work email: ");
    String workEmail = sc.nextLine();
    workEmails.add(workEmail);

    length++;
    view = "list";
}

```

## Testing Your Code

Before moving on to the next part of the program, we want to check that our code is adding a contact to the list. To do that, we need to create a getter for the length attribute.

```

public int getLength() {
    return length;
}

```

Now call print the result from the getLength method.

```
//add code below this line
```

```
Contacts contacts = new Contacts();  
contacts.display();  
System.out.println(contacts.getLength());
```

```
//add code above this line
```

Run the program and enter a when prompted, then add the following contact:

```
Rachel Kim  
555 123-4567  
rachel_k@personalMail.com  
Senior Software Engineer  
555 890-1234  
rkim@workMail.com
```

If everything worked properly, your program should print 1 in the terminal as there is one person in our contact list.

#### ▼ Code

Your code should look like this:

```
import java.util.ArrayList;  
import java.util.Scanner;  
  
//add class definitions below this line  
  
abstract class Information {  
    public abstract void displayInfo();  
    public abstract void addInfo();  
}  
  
class Contacts extends Information {  
    private String view;  
    private ArrayList<String> names;  
    private ArrayList<String> titles;  
    private ArrayList<String> workPhoneNumbers;  
    private ArrayList<String> workEmails;  
    private ArrayList<String> personalPhoneNumbers;  
    private ArrayList<String> personalEmails;  
    private String choice;  
    private int index;  
    private int length;
```

```

public Contacts() {
    view = "list";
    names = new ArrayList<String>();
    titles = new ArrayList<String>();
    workPhoneNumbers = new ArrayList<String>();
    workEmails = new ArrayList<String>();
    personalPhoneNumbers = new ArrayList<String>();
    personalEmails = new ArrayList<String>();
    choice = null;
    index = 0;
    length = 0;
}

public void displayInfo() {

}

public void addInfo() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter their name: ");
    String name = sc.nextLine();
    names.add(name);

    System.out.print("Enter their personal phone number: ");
    String personalPhone = sc.nextLine();
    personalPhoneNumbers.add(personalPhone);

    System.out.print("Enter their personal email: ");
    String personalEmail = sc.nextLine();
    personalEmails.add(personalEmail);

    System.out.print("Enter their work title: ");
    String title = sc.nextLine();
    titles.add(title);

    System.out.print("Enter their work phone number: ");
    String workPhone = sc.nextLine();
    workPhoneNumbers.add(workPhone);

    System.out.print("Enter their work email: ");
    String workEmail = sc.nextLine();
    workEmails.add(workEmail);

    length++;
    view = "list";
}

```

```

public void showList() {
    System.out.println();
    Scanner sc = new Scanner(System.in);
    if (length == 0) {
        System.out.print("(A)dd a new contact \n(Q)uit \n> ");
        choice = sc.nextLine().toLowerCase();
    } else {
        view = "quit";
    }
    handleChoice();
}

public void handleChoice() {
    switch(choice) {
        case "q":
            view = "quit";
            break;
        case "a":
            if (view.equals("list")) {
                view = "add";
                break;
            }
    }
}

public void display() {
    while (true) {
        if (view.equals("list")) {
            showList();
        } else if (view.equals("info")) {
            displayInfo();
        } else if (view.equals("add")) {
            System.out.println();
            addInfo();
        } else if (view.equals("quit")) {
            System.out.println("\nClosing the contact list...\n");
            break;
        }
    }
}

public int getLength() {
    return length;
}
}

```

*//add class definitions above this line*

```
public class Lab2 {  
    public static void main(String[] args) {  
  
        //add code below this line  
  
        Contacts contacts = new Contacts();  
        contacts.display();  
        System.out.println(contacts.getLength());  
  
        //add code above this line  
    }  
}
```

# Polymorphism Lab 3

---

## Displaying the List View

Now that we can add a contact to the list, we will want to show all of the contacts in the list. But before doing that, we need to remove the print statement at the end of the script. The final two lines of code should look like this:

```
//add code below this line

Contacts contacts = new Contacts();
contacts.display();

//add code above this line
```

We are going to iterate through the list of contacts and print the name. To help users select a contact, a number will appear before each name. This way, the user types the number, and the contact's information appears. In the else branch of the `showList` method, create a for loop to go from 0 to length. We want a number followed by the name. The numbers should start at 1, so print the loop index plus 1 followed by the element from the `names ArrayList`. After displaying the list of names, ask the user to make a selection. Entering a number will show all of the information about a contact.

```

public void showList() {
    System.out.println();
    Scanner sc = new Scanner(System.in);
    if (length == 0) {
        System.out.print("(A)dd a new contact \n(Q)uit \n> ");
        choice = sc.nextLine().toLowerCase();
    } else {
        for (int i = 0; i < length; i++) {
            System.out.println(i + 1 + " " + names.get(i));
        }
        System.out.print("\n(#) Select a name \n(A)dd a new
        contact\n(Q)uit \n> ");
        String input = sc.nextLine().toLowerCase();
        choice = input;
    }
    handleChoice();
}

```

## Handling Numeric Input

Add an `else if` branch to the `handleChoice` method that asks if the user input is numeric (remember, the `nextLine` method stores user input as a string) and if the user is in the list view. If yes, then convert the user input to an integer, subtract 1, and store it in the variable `num`. Remember, we added one to the loop index in the `showList` method. If the user made a mistake in entering the number, the script will crash if you try to access an index that is outside of the `ArrayList`. So we need to verify that the number is between 0 and `length`. Finally, set `index` to `num` and set `view` to 'info'.

```

public void handleChoice() {
    if (choice.equals("q")) {
        view = "quit";
    } else if (choice.equals("a") && view.equals("list")) {
        view = "add";
    } else if (isNumeric(choice) && view.equals("list")) {
        int num = Integer.parseInt(choice) - 1;
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    }
}

```



We need to create the boolean helper method `isNumeric` which takes the user input and determines if they entered a number. First see if the user input (a string) is null or empty. Return `false` if either condition is true. Then try to convert the string to an integer. If this works return `true`. If Java throws an exception because the string cannot be converted to an integer, then return `false`.

```
private boolean isNumeric(String s) {  
    int value;  
  
    if (s == null || s.equals("")) {  
        return false;  
    }  
  
    try {  
        value = Integer.parseInt(s);  
        return true;  
    } catch (NumberFormatException e) {  
        return false;  
    }  
}
```

## Testing Your Code

Before moving on to the next part of the program, we want to check that our code is displaying all of the contacts in the list. To do that, enter two different contacts. The first one is:

```
John Calvin  
555 111-2222  
john.calvin@email.net  
Philosopher  
555 333-4444  
jcalvin@work.org
```

You should see a list that looks like this:

```
1) John Calvin  
  
(#) Select a name  
(A)dd a new contact  
(Q)uit
```

Contact 1

Now add a second contact to the list:

Thomas Hobbes  
555 666-7777  
t\_hobbes@email.net  
Philosopher  
555 888-9999  
tom\_hobbes@work.org

Your program should now show the following output:

```
1) John Calvin
2) Thomas Hobbes

(#) Select a name
(A)dd a new contact
(Q)uit
```

Contact 2

#### ▼ Code

Your code should look like this:

```
import java.util.ArrayList;
import java.util.Scanner;

//add class definitions below this line

abstract class Information {
    public abstract void displayInfo();
    public abstract void addInfo();
}

class Contacts extends Information {
    private String view;
    private ArrayList<String> names;
    private ArrayList<String> titles;
    private ArrayList<String> workPhoneNumbers;
    private ArrayList<String> workEmails;
    private ArrayList<String> personalPhoneNumbers;
    private ArrayList<String> personalEmails;
    private String choice;
    private int index;
    private int length;
```

```
public Contacts() {
    view = "list";
    names = new ArrayList<String>();
    titles = new ArrayList<String>();
    workPhoneNumbers = new ArrayList<String>();
    workEmails = new ArrayList<String>();
    personalPhoneNumbers = new ArrayList<String>();
    personalEmails = new ArrayList<String>();
    choice = null;
    index = 0;
    length = 0;
}

public void displayInfo() {

}

public void addInfo() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter their name: ");
    String name = sc.nextLine();
    names.add(name);

    System.out.print("Enter their personal phone number: ");
    String personalPhone = sc.nextLine();
    personalPhoneNumbers.add(personalPhone);

    System.out.print("Enter their personal email: ");
    String personalEmail = sc.nextLine();
    personalEmails.add(personalEmail);

    System.out.print("Enter their work title: ");
    String title = sc.nextLine();
    titles.add(title);

    System.out.print("Enter their work phone number: ");
    String workPhone = sc.nextLine();
    workPhoneNumbers.add(workPhone);

    System.out.print("Enter their work email: ");
    String workEmail = sc.nextLine();
    workEmails.add(workEmail);

    length++;
    view = "list";
}
```

```

public void showList() {
    System.out.println();
    Scanner sc = new Scanner(System.in);
    if (length == 0) {
        System.out.print("(A)dd a new contact \n(Q)uit \n> ");
        choice = sc.nextLine().toLowerCase();
    } else {
        for (int i = 0; i < length; i++) {
            System.out.println(i + 1 + " " + names.get(i));
        }
        System.out.print("\n(#) Select a name \n(A)dd a new contact\n(Q)uit \n> ");
        String input = sc.nextLine().toLowerCase();
        choice = input;
    }
    handleChoice();
}

private boolean isNumeric(String s) {
    int value;

    if (s == null || s.equals("")) {
        return false;
    }

    try {
        value = Integer.parseInt(s);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}

public void handleChoice() {
    if (choice.equals("q")) {
        view = "quit";
    } else if (choice.equals("a") && view.equals("list")) {
        view = "add";
    } else if (isNumeric(choice) && view.equals("list")) {
        int num = Integer.parseInt(choice) - 1;
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    }
}

public void display() {

```

```

        while (true) {
            if (view.equals("list")) {
                showList();
            } else if (view.equals("info")) {
                displayInfo();
            } else if (view.equals("add")) {
                System.out.println();
                addInfo();
            } else if (view.equals("quit")) {
                System.out.println("\nClosing the contact list...\n");
                break;
            }
        }
    }

    public int getLength() {
        return length;
    }
}

//add class definitions above this line

public class Lab3 {
    public static void main(String[] args) {

        //add code below this line

        Contacts contacts = new Contacts();
        contacts.display();

        //add code above this line
    }
}

```

# Polymorphism Lab 4

---

## Displaying Contact Info

The next step is to display the contact information for a selected contact. On the last page, we already modified `handleChoice` to deal with numeric input. So let's update the `displayInfo` method to display the information. Print the elements from each `ArrayList` using the attribute `index`. Create a scanner object for user input. Then present the user with some options to return to the list view, go to the next contact, go to the previous contact, or quit. Call the `handleChoice` method to act on the user input.

```
public void displayInfo() {
    System.out.println();
    System.out.println(names.get(index));
    System.out.println("Personal email address: " +
        personalEmails.get(index));
    System.out.println("Personal phone number: " +
        personalPhoneNumbers.get(index));
    System.out.println("Work title: " + titles.get(index));
    System.out.println("Work email address: " +
        workEmails.get(index));
    System.out.println("Work phone number: " +
        workPhoneNumbers.get(index));
    Scanner sc = new Scanner(System.in);
    System.out.print("\n(C)ontact List \n(P)revious contact
        \n(N)ext contact \n(Q)uit \n> ");
    String input = sc.nextLine().toLowerCase();
    choice = input;
    handleChoice();
}
```

## Return to Contact List

We want the user to be able to see a list view of all contacts from the info view when the user enters `c`. Modify the `handleChoice` method to set view to `'list'` when choice is `c`.

```

public void handleChoice() {
    if (choice.equals("q")) {
        view = "quit";
    } else if (choice.equals("a") && view.equals("list")) {
        view = "add";
    } else if (isNumeric(choice) && view.equals("list")) {
        int num = Integer.parseInt(choice) - 1;
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    } else if (choice.equals("c") && view.equals("info")) {
        view = "list";
    }
}

```

## Next and Previous Contacts

The next feature to add is the ability to change contacts (next or previous) from the info view. Moving the next contact should increase index by 1, and moving to the previous contact should decrease index by 1. However, we need to make sure that the index is not out of the range of an ArrayList; this would cause the program to crash. To avoid this we are going to have a “wrapping” effect. If the index is at the end of the ArrayList, advancing to the next contact takes you back to the beginning. Similarly, going to the previous contact from the beginning of the ArrayList will take you to the end of the ArrayList.

To do this, we could write a traditional conditional statement. However, that is several lines of code. Instead, you can use a ternary operator (also called a conditional expression) to write the same logic in one line of code.

```

variable = (boolean_expression) ? value if true : value if false;

```

### Ternary Operator

So when the user enters n the index will increase by 1 as long as the index plus 1 is less than length. If not, the index becomes 0 (the first contact). Similarly, when the user enters p, the index will decrease by 1 as long as the index minus 1 is greater than or equal to 0. If not, the index becomes length minus 1 (the last contact). Add the following conditional branches to the handleChoice method.

```

// The end of the handleChoice method
} else if (choice.equals("c") && view.equals("info")) {
    view = "list";
} else if (choice.equals("n") && view.equals("info")) {
    index = (index + 1 < length) ? index + 1 : 0;
} else if (choice.equals("p") && view.equals("info")) {
    index = (index - 1 >= 0) ? index - 1 : length - 1;
}

```

## Testing Your Code

This program should be complete now. To test it, add at least two contacts. Select one of the contacts. Use n and p to cycle through the contact information. Enter c to return to the list view of all the contacts.

### ▼ Code

Your code should look like this:

```

import java.util.ArrayList;
import java.util.Scanner;

//add class definitions below this line

abstract class Information {
    public abstract void displayInfo();
    public abstract void addInfo();
}

class Contacts extends Information {
    private String view;
    private ArrayList<String> names;
    private ArrayList<String> titles;
    private ArrayList<String> workPhoneNumbers;
    private ArrayList<String> workEmails;
    private ArrayList<String> personalPhoneNumbers;
    private ArrayList<String> personalEmails;
    private String choice;
    private int index;
    private int length;

    public Contacts() {
        view = "list";
        names = new ArrayList<String>();
        titles = new ArrayList<String>();
        workPhoneNumbers = new ArrayList<String>();
    }
}

```



```

workEmails = new ArrayList<String>();
personalPhoneNumbers = new ArrayList<String>();
personalEmails = new ArrayList<String>();
choice = null;
index = 0;
length = 0;
}

public void displayInfo() {
    System.out.println();
    System.out.println(names.get(index));
    System.out.println("Personal email address: " +
        personalEmails.get(index));
    System.out.println("Personal phone number: " +
        personalPhoneNumbers.get(index));
    System.out.println("Work title: " + titles.get(index));
    System.out.println("Work email address: " +
        workEmails.get(index));
    System.out.println("Work phone number: " +
        workPhoneNumbers.get(index));
    Scanner sc = new Scanner(System.in);
    System.out.print("\n(C)ontact List \n(P)revious contact
        \n(N)ext contact \n(Q)uit \n> ");
    String input = sc.nextLine().toLowerCase();
    choice = input;
    handleChoice();
}

public void addInfo() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter their name: ");
    String name = sc.nextLine();
    names.add(name);

    System.out.print("Enter their personal phone number: ");
    String personalPhone = sc.nextLine();
    personalPhoneNumbers.add(personalPhone);

    System.out.print("Enter their personal email: ");
    String personalEmail = sc.nextLine();
    personalEmails.add(personalEmail);

    System.out.print("Enter their work title: ");
    String title = sc.nextLine();
    titles.add(title);

    System.out.print("Enter their work phone number: ");
    String workPhone = sc.nextLine();
    workPhoneNumbers.add(workPhone);
}

```

```

        System.out.print("Enter their work email: ");
        String workEmail = sc.nextLine();
        workEmails.add(workEmail);

        length++;
        view = "list";
    }

    public void showList() {
        System.out.println();
        Scanner sc = new Scanner(System.in);
        if (length == 0) {
            System.out.print("(A)dd a new contact \n(Q)uit \n> ");
            choice = sc.nextLine().toLowerCase();
        } else {
            for (int i = 0; i < length; i++) {
                System.out.println(i + 1 + " " + names.get(i));
            }
            System.out.print("\n(#) Select a name \n(A)dd a new contact\n(Q)uit \n> ");
            String input = sc.nextLine().toLowerCase();
            choice = input;
        }
        handleChoice();
    }

    private boolean isNumeric(String s) {
        int value;

        if (s == null || s.equals("")) {
            return false;
        }

        try {
            value = Integer.parseInt(s);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

    public void handleChoice() {
        if (choice.equals("q")) {
            view = "quit";
        } else if (choice.equals("a") && view.equals("list")) {
            view = "add";
        } else if (isNumeric(choice) && view.equals("list")) {

```

```

        int num = Integer.parseInt(choice) - 1;
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    } else if (choice.equals("c") && view.equals("info")) {
        view = "list";
    } else if (choice.equals("n") && view.equals("info")) {
        index = (index + 1 < length) ? index + 1 : 0;
    } else if (choice.equals("p") && view.equals("info")) {
        index = (index - 1 >= 0) ? index - 1 : length - 1;
    }
}

public void display() {
    while (true) {
        if (view.equals("list")) {
            showList();
        } else if (view.equals("info")) {
            displayInfo();
        } else if (view.equals("add")) {
            System.out.println();
            addInfo();
        } else if (view.equals("quit")) {
            System.out.println("\nClosing the contact list...\n");
            break;
        }
    }
}

public int getLength() {
    return length;
}
}

//add class definitions above this line

public class Lab4 {
    public static void main(String[] args) {

        //add code below this line

        Contacts contacts = new Contacts();
        contacts.display();

        //add code above this line
    }
}

```



# Polymorphism Lab Challenge

---

## Problem

In the IDE to the left, the class `Chef` is already defined, as is the `display` method. However, it does not have a constructor. Create three constructors that take one, two, and three parameters respectively.

## Expected Output

Instantiate three `Chef` objects each one using a different constructor.

```
//add code below this line

Chef c1 = new Chef("Marco Pierre White");
Chef c2 = new Chef("Rene Redzepi", "Nordic");
Chef c3 = new Chef("Thomas Keller", "French", 3);

//add code above this line
```

Calling the `display` method for each object should return the following text:

Method Call	Return Value
<code>c1.display()</code>	Marco Pierre White is known for null cuisine and has 0 Michelin stars.
<code>c2.display()</code>	Rene Redzepi is known for Nordic cuisine and has 0 Michelin stars.
<code>c3.display()</code>	Thomas Keller is known for French cuisine and has 3 Michelin stars.