Topic 1

Object Oriented Programming

Objectives

- To review the concepts and terminology of object-oriented programming
- To discuss some features of object-oriented design

Review: Objects

- In Java and other Object-Oriented Programming (OOP) languages, the focus is on ***objects***
- ***Objects*** are entities that can do actions or be acted upon in a Java program
- All objects have
  - ***Properties***
    - These are the *data* about an object
    - In Java we call them ***attributes*** or ***fields*** or ***instance variables***
  - ***Behaviours*** (***actions***)
    - In Java they are implemented as ***methods*** (more specifically, ***instance methods***)

Review: Objects and Classes

- Every object belongs to a specific ***class***
  - Objects that belong to the same class have the same properties and can have the same actions performed
    - Example from CS1026: we can invoke the methods of the Picture class on *any* Picture object
- We can think of a class as being a ***template*** or ***pattern*** or ***model*** for objects of that class

Review: Object-Oriented Programming

- ***Object-oriented programs*** consist of ***interacting objects***
  - Objects are ***defined by*** classes
  - Objects are ***created by*** other classes (***client classes***) which ***use*** them in implementing a programming solution to a problem
- Example from CS1026:
  - Picture class ***defines*** the attributes of a Picture object and the methods that can be invoked on a Picture object
  - We can write application programs that ***create and use*** objects of the Picture class
    - Example: lab and assignment classes that have a main method, e.g. MakeCollage

Example: Collage

by CS1026a student using objects and methods of the Picture class

Example: Social Networking

- Suppose we want to keep track of social contact information for our friends / relatives
- Part of OOP design is deciding on what classes we will need for our problem
- Let's start with a class called Person, that will model the information about one person in our social network …

Review: Class Definition

- A ***class definition*** consists of
  - Attribute declarations (aka fields, instance variables)
  - Constructor definitions
  - Method definitions
- A class definition is stored in a file
  - With the same name as the class
  - With a .java extension on the file

Example: Person Class

- ***Attributes (instance variables, fields)***
  - What kind of information do we want to have about a person? Let's keep it short for now …
    - Person's name
    - Email address
  - What type should each of these be?
    - A name can be a string
    - An email address can be a string
  - What other attributes could we add in the future?

Example: Person Class

```
/* Attribute declarations */
private String lastName;
private String firstName;
private String email;
```

- Why are the attributes private?
- Note that the instance variables are just being ***declared*** here (not explicitly assigned values)

Review: Constructors

- A ***constructor*** is a special *method* that is called automatically when an object is created with the new operator
  - Its purpose is to *initialize the attributes* of an object when the object is created
  - A constructor has the same name as the class name

Example: Person class

```
/**
*  Constructor initializes the person's name
*                                    and email address
*/
public Person(String firstName, String lastName,
                                       String email) {
   this.lastName = lastName;
   this.firstName = firstName;
   this.email = email;
}
```

Review: Terminology

- Keyword this
- Scope of variables
  - ***Scope*** refers to the parts of the code in which those variables are known
  - Scope of instance variables?
- Formal parameters
  - What is their scope?

Example: Person Class

- What ***methods*** might we want to have?
  - ***accessor methods*** (aka ***getters***)
  - ***modifier methods*** (aka ***setters***)
  - toString method
  - equals method
    - two Person objects are the same if they have the same first name and same last name

Example: Person class

```
/**
* setEmail method sets the person's email address
* @param email
*/
public void setEmail (String email) {
   this.email = email;
}
```

What is this @param?

- Javadoc documentation (we will do it in Lab 2)

Example: Person class

```
/**
* toString method returns a string representation of the person
* @return string with first name and last name, email address
*/
public String toString() {

   String s = firstName + " " + lastName + "\t" +
                                             email ;
   return s;
}
```


Discussion

- What is the return type of this method?
- What is \t ?
- What kind of variable is s?
  - A ***reference variable*** of type String
- What is its scope?
  - It is a ***local variable***


```
/**
* equals determines whether two persons have the same name
* @param other      other Person object that this is compared to
* @return true if they have the same first and last name, false
   otherwise
*/
public boolean equals(Person other) {
   if (this.firstName.equals(other.firstName) &&

   this.lastName.equals(other.lastName))
        return true;
   else
        return false;
   }
```

- What is this.firstName? other.firstName?
- Where is the equals method that is used in the code?

Example: SocialNetwork Class

- We're now ready to provide a class that allows us to keep track of our social contacts
- What attributes might it have?
  - A list of Person objects
    - We'll use an ***array*** as our ***data structure***
  - A count of the number of friends currently in the list
    - Why is this not necessarily the same as the size of the array?

Example: SocialNetwork Class

```
/* Attribute declarations */
private Person[] friendList;
        // array of persons (list of friends)
private int numFriends;
        //current number of friends in list

/* Constant definition */
private final int DEFAULT_MAX_FRIENDS = 10;
        //default size of array (max. no. of persons)
```


Review: Terminology

- Keyword final
- Array declaration []

Example: SocialNetwork Class

- Constructors:
  - One that creates an array of default size
  - One that takes the size of the array as a parameter
- What do we call it when there is more than one constructor?
  - ***overloading***


```
/**
* Constructor creates Person array of default size
*/
public SocialNetwork () {
   friendList = new Person[DEFAULT_MAX_FRIENDS];
   numFriends = 0;
}

/**
* Constructor creates Person array of specified size
* @param max       maximum size of array
*/
public SocialNetwork(int max) {
   friendList = new Person[max];
   numFriends = 0;
}
```

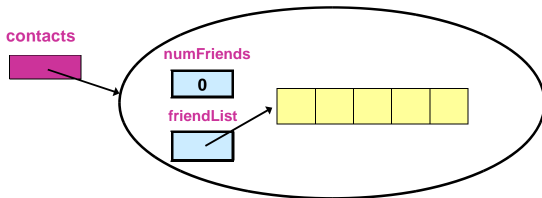
Discussion

- What is stored in the friendList array after the following is executed?

```
friendList = new Person[DEFAULT_MAX_FRIENDS];
```

Example: SocialNetwork Object

SocialNetwork contacts = new SocialNetwork(5);



08/28/11 1-25

Example: SocialNetwork Class

- Instance methods: let's start with methods to
  - add a person to the list
  - remove a specified person from the list
  - clear the list, i.e. remove all persons
  - return how many persons are in the list
  - toString
- *(we will add other methods later)*

08/28/11 1-26

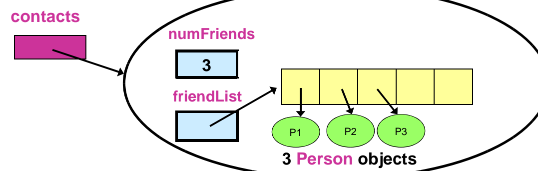
add method

```
/**
* add method adds a person to the list
* @param firstName
* @param lastName
* @param email
*/
public void add (String firstName, String lastName, String email) {

    // create a new Person object
    Person friend = new Person (firstName, lastName, email);

    // add it to the array of friends
        // but, what if array is not big enough?
        // double its capacity automatically

    if (numFriends == friendList.length)
        expandCapacity();

    // add reference to friend at first free spot in array

    friendList [numFriends] = friend;
    numFriends++;
}
```

Example: SocialNetwork Object

SocialNetwork contacts = new SocialNetwork(5);

After 3 friends are added it will look like this:



Note that numFriends also acts as the index of the first free spot in the array!

08/28/11 1-28

Review: Arrays

- An array has a particular number of cells when it is created (its ***capacity***)
- What happens when an array is full and we try to store past the last element in the array?
  - An ***exception*** is thrown
  - What happens then?
- We can instead *automatically expand the capacity* of the array in our code!

08/28/11 1-29

expandCapacity helper method

```
/**
* expandCapacity method is a helper method
* that creates a new array to store friends, with twice
* the capacity of the existing one
*/
private void expandCapacity() {

    Person[] largerList = new Person[friendList.length * 2];

    for (int i = 0; i < friendList.length; i++)
        largerList [i] = friendList [i];

    friendList = largerList;
}
```

08/28/11 1-30

Review: Terminology

- Helper method
- Array length
- Scope of variables: what is the scope of each of the following variables?
  - friendList
  - largerList
  - i


```
/**
* toString method returns a string representation of all
   persons in the list
* @return string representation of list
*/
public String toString() {
   String s = "";
   for (int i = 0; i < numFriends; i++){
       s = s + friendList[i].toString() + "\n" ;
   }
   return s;
}
```

toString method

- What is "" ? "\n" ?


```
/**
* remove method removes a specified friend from the list
* @param firstName      first name of person to be removed
* @param lastName       last name of person to be removed
* @return true if friend was removed successfully, false otherwise
*/
public boolean remove (String firstName, String lastName) {
    final int NOT_FOUND = -1;
    int search = NOT_FOUND;
    Person target = new Person(firstName, lastName, "");

    // if list is empty, can't remove
    if (numFriends == 0)
        return false;
    // search the list for the specified friend
    for (int i = 0; i < numFriends &&
                        search == NOT_FOUND; i ++)
        if (friendList [i].equals(target))
            search = i;
```

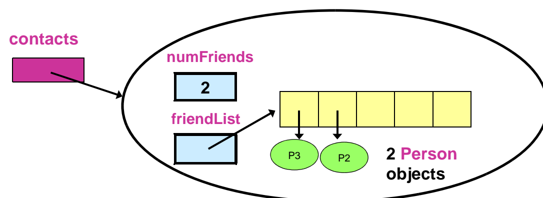
remove method

```
    // if not found, can't remove
    if (search == NOT_FOUND)
       return false;

    // target person found, remove by replacing with
    // last one in list

       friendList[search] = friendList[numFriends - 1];
       friendList[numFriends - 1] = null;
       numFriends -- ;

       return true;
}
```

remove method (cont'd)

Example: SocialNetwork Object

Suppose the target person to be removed was the first one (P1) ; after it is removed, we will have:



Discussion

- The search in the remove method is called a ***linear search***
  - It starts at the beginning and continues in a sequential manner
- Why have we used a constant definition here, and why is it -1?
  - final int NOT_FOUND = -1;
- Where is the equals method of the line
  - if (friendList [i].equals(target))

Discussion

- Why is it OK to replace the reference to the found Person with the last one in the list?

Exercises

- Write getNumFriends
- Write clearFriends

Example: Using the SocialNetwork Class

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

    SocialNetwork contacts = new SocialNetwork();

    contacts.add("Snoopy", "Dog", "snoopy@uwo.ca");
    contacts.add("Felix", "Cat", "felix@uwo.ca");
    contacts.add("Mickey", "Mouse", mickey@uwo.ca);

    System.out.println(contacts.toString());
    System.out.println("I have " + contacts.getNumFriends() +
                                          "  friends in my contact list.");
    }
}
```


Discussion

- Note that if we had
  System.out.println(contacts);
  then Java would automatically invoke the toString method of the class that contacts belongs to
- How many friends could you add to your list of friends, in an application program that uses our SocialNetwork class?

Exercise: Expand the SocialNetwork Class

- The SocialNetwork class could use some more methods in order to be useful!
  - A method that writes the list to a file
  - A method that reads the list from a file
  - A method that searches for a particular friend in the list, and returns the email address
  - Others?

Review: Passing Parameters

- Why are methods written with parameter lists?
  - So that the methods can be more general
    - We can use methods with *different values* passed in as parameters

Review: Passing Parameters

- How are parameters actually passed?
- The variable in the parameter list in the *method definition* is known as a ***formal parameter***
- When we *invoke a method* with a parameter, that is known as an ***actual parameter***

Passing Parameters: How it Works

```
public class MyFriends {
{
  public static void main(String[] args)
  { ...
   contacts.add("Felix", "Cat",
                "felix@uwo.ca");
      ....
```

```
public class SocialNetwork {
   ...

public void add (String firstName, String
                 lastName, String email) {
    ...
```

actual parameters are provided by the calling program when it **invokes** the method

formal parameters are part of the **method definition**

When the add method is executed, the value of each actual parameter is ***copied to*** the corresponding formal parameter variable

Aspects of Object-Oriented Design

- Modularity
- Information Hiding
- Encapsulation

Aspects of Program Design: Modularity

- ***Modularity*** refers to subdividing a large problem into smaller components, or ***modules***, to make the design of a solution easier
  - Modules should be as independent from each other as possible
  - Each module should perform one well-defined task

Aspects of Program Design: Information Hiding

- ***Information hiding*** refers to making implementation details inaccessible
  - To users of a program (they do not need to know about implementation details)
  - To other modules in a program (they cannot see nor change the *hidden* details)
  - Example: attributes (instance variables) in a class definition are ***private***
    - What parts of a program can access instance variables directly?

Aspects of OOP Design: Encapsulation

- ***Object-oriented Design*** produces modular solutions for problems that primarily involve ***data***
- We identify the components involved within the problem: the ***objects***
  - An object has data: ***characteristics (attributes)***
  - And ***behaviours (operations)***
- Combining the *data* and the *operations on the data* is called ***encapsulation***
  - They are combined in the class definition