Object Oriented Programming

Objectives

- Read the concepts and terminology of object-oriented programming
- Discuss the principles and features of object-oriented design
- Implement a new class with instance variables and methods

Objects

- In Java and other Object-Oriented Programming (OOP) languages, the focus is on ***objects***
- ***Objects*** are program modules that can do actions or be acted upon by other objects
- 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***)

Objects and Classes

- **Every object belongs to a specific *class***
  - **Objects that belong to the same class have the same properties and can perform the same actions**
- **We can think of a class as being a *template* or *pattern* or *model* or *definition* for objects of that class**

Class 1 { Obj 1
Obj 2
Obj 3
⋮

Object-Oriented Programming

- ***Object-oriented programs*** **consist of** ***interacting objects***
  - **Objects are *defined by* classes**
  - **Objects can be *created by* objects of other classes (*client classes*) which *use* them in implementing a programming solution to a problem**

Example: Social Networking

- Suppose we want to keep track of social contact information for our friends / relatives
- We wish to write a program that allows us to add contact information of a friend to our list of friends, remove a contact from the list, and print information about all our contacts.

Example: Social Networking

- 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

Class Definition

- A *class definition* consists of
  - Attribute declarations (also known as fields or 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

The name of the class have to be the same as the name of 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 (first and last)
    - Email address
  - What type should each of these be?
    - A name can be a string
    - An email address can be a string

Example Python: Person Class

```
class Person:

        def __init__(self, firstName="", lastName,="" email=""):
            self.firstName = firstName
            self.lastName = lastName
            self.email = email
```

- Note in Python we can assign default values to the attributes in this case we used an empty string

Example Java: Person Class

```
public class Person{
/* 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)

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
  - In Python we use the special method __init__ to do the job of a constructor
  - In Java 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;
}
```

Because the input and the parameter has the same name.

public Person(String A, String B, String C).
lastName = A
* No need to add this.
Because the names are different.

Compared to Python, in Java one must EXPLICITLY give types to the attributes. Also note the difference between the keyword *this* vs Python's *self*.

Terminology

- Keyword *this* similar to *self* in Python
- Scope of variables
  - *Scope* refers to the parts of the code in which those variables can be used
    the function declared.
  - Scope of instance variables?
- Formal parameters – input
  - What is their scope?

Example: Person Class

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

Comments:
① //
② /* */

③ /** ← Javadoc.
@param
@return
*/

Example: Person class

Java

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

Python

```
"""
setEmail method sets the person's
    email address.
:param email: email address to set
"""
def setEmail(self,email):
        self.email=email
```

Note that Python uses WHITESPACE to tie blocks of code together Java uses BRACES and SEMICOLONS (you should still code with whitespace as well)

What is this @param?

- Javadoc documentation (we will do it in Lab 1)

Example: Person class

Python

```
def __repr__(self):
        s = self.firstName +" " self.lastName +"\t" + self.email
        return s
```

Java

custom how to printout object.

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

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

do no directly print out the information itself.

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*

Python

```
def equals(self, other):
        if self.firstName == other.getFirstName() and self.lastName == other.getLastName() :
            return True
        else :
            return False
```

Java

```
/**
* 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;
        }
```

this person another person.

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

Example: SocialNetwork Class

- We are 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 (this is similar to the notation of a list in Python)*
  - 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

Python:

```
from Person import Person
class SocialNetwork:
    def __init__(self,num=0):
        self.friends = []
        self.numFriends = num
```

Java:

```
/* Attribute declarations */
// array of persons (list of friends)
private Person[] friendList;

//current number of friends in list
private int numFriends;
/* Constant definition */
private final int DEFAULT_MAX_FRIENDS = 10;
```

a default umber of friendlist

Notice in Python we declare the attributes in the constructor itself

Terminology

- **Keyword *final* (no such thing in Python, by convention we used all capitalized words to represent a constant)**
- **Array declaration [] (array's and python lists do NOT always act the same)**

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*
  - *In Python we do this by setting defaults in the method*

Python:

```
from Person import Person
class SocialNetwork:
        def __init__(self,num=0):
                self.friends =[]
                self.numFriends =num
```

Notice how there is only one constructor for Python but it uses default values to allow for different uses of it.

Also note than in Java arrays must MUST have an specified size; lists can grow dynamically in Python.

Java:

```
/**
* 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?**

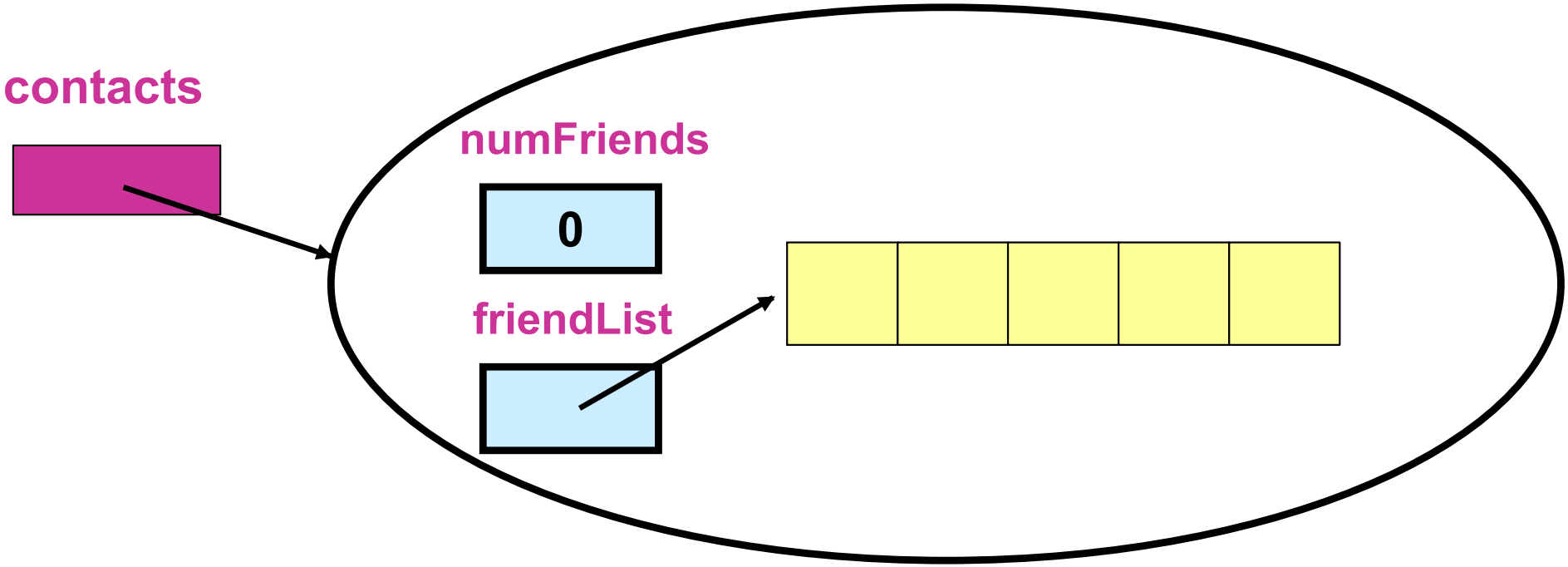
```
friendList = new
Person[DEFAULT_MAX_FRIENDS];
```

- **How does this differ from Python?**

```
self.friends = []
```

Example: SocialNetwork Object

contacts = new SocialNetwork(5);



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)***

Add method

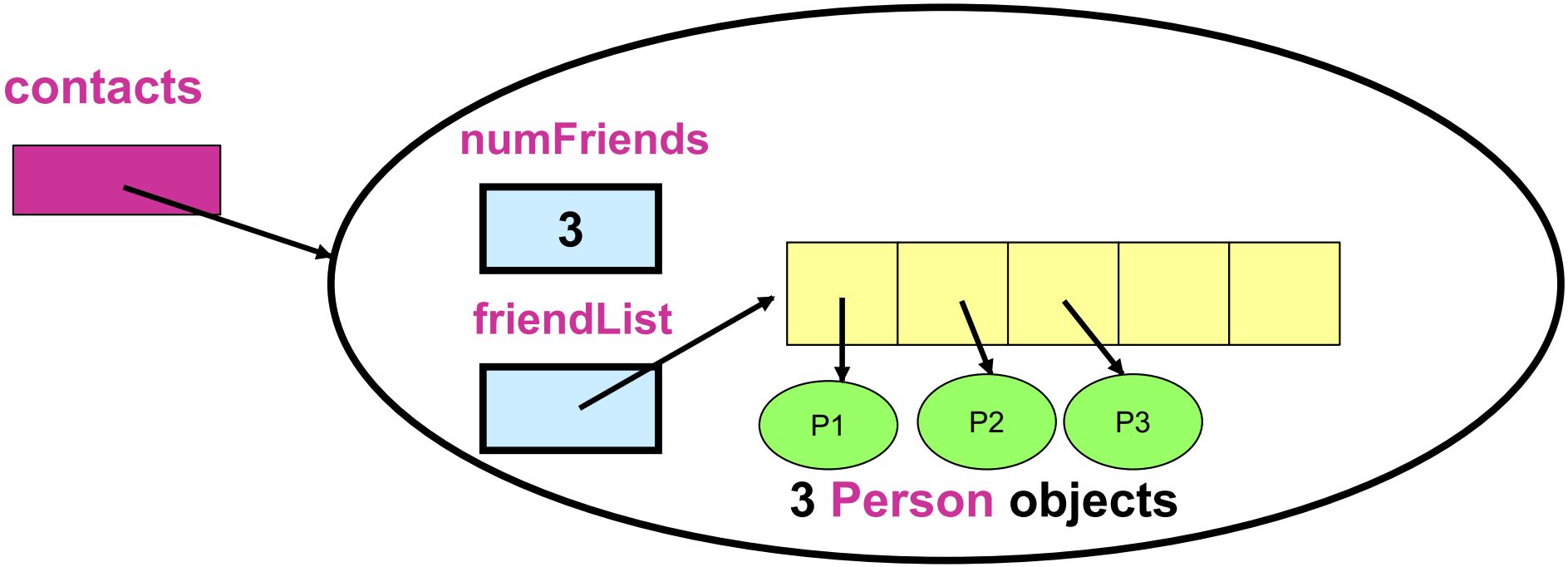
```
Python:
def add(aFriend):
    self.friends.append(aFriend)
    self.numFriends = len(self.friends)
```

```
Java:
/**
* add method adds a person to the list
* @param firstName
* @param lastName
* @param email
*/
public void add (Person friend) {

        // 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

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!

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!

```
/**
* 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;
}
```

Note in Python we did not have to do this as lists can grow dynamically.

Identify the scope of the variables:

- **friendList**
- **largerList**
- **i**

Python

```
def __repr__(self):
        s = ""
        for element in self.friends:
            s = s + "\n" + element.getFriend()
        return s
```

toString

Java

```
/**
* 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 < this.numFriends; i++){
                s = s + friendList[i].toString() + "\n" ;
        }
        return s;
}
```

- What is "" ? "\n" ?

Class SocialNetwork contains a method for removing a data item from the array. To remove a data item, say *target*, from the array we first need to find the position of such an item in the array. A simple way of looking for *target* in array *friendList* is to take the data items stored in the array one by one starting at the data item stored in index 0 and compare each one of them with *target* until either

- *target* is found, or
- all data items have been examined and *target* is not found

The above algorithm for looking for a data item in a list is called *linear search*.

Once item *target* has been found in the array we can remove it by replacing it with the last item in the array. Pseudocode for removing a data item from the array follows.

Algorithm remove(*target*)
Input: data item to be removed
Output: true if *target* was removed from the array; false
if *target* was not found in the array

```
i = 0
while (i < numFriends) and (friendList[i] not equal target) do
    i = i+1
if i = numFriends then return false
else {
    friendList[i] = friendList[numFriends-1]
    friendList[numFriends-1] = null
    numFriends = numFriends -1
    return true
}
```

The advantage of writing an algorithm in pseudocode is that we can concentrate on designing the steps that the algorithm needs to perform to achieve the desired task without having to think about how to express the algorithm in correct java syntax.

Once we have designed a correct algorithm for a problem in pseudocode, translating it into Java is a somewhat mechanical process.

Writing algorithms in pseudocode and then translating them into Java makes it easier to design programs.

The beauty of pseudocode is that there is no fixed syntax or rigid rules for it. Pseudocode is a mixture of English and programming-like statements.

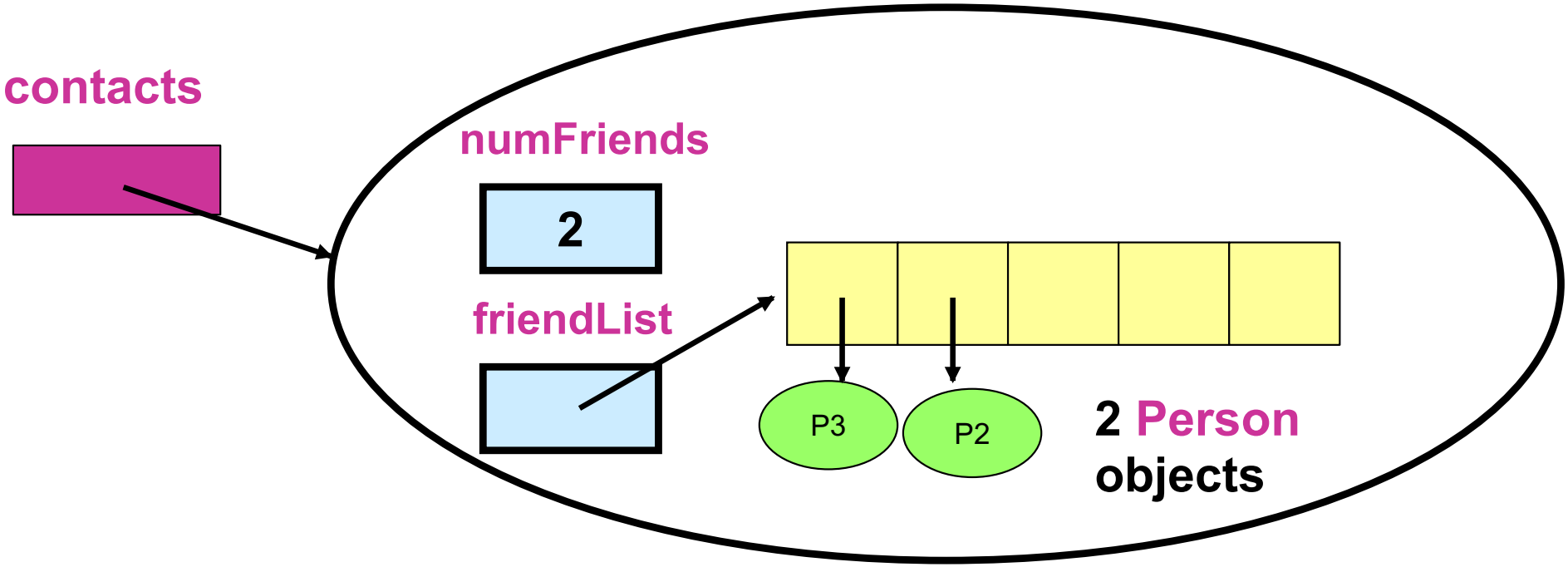
Each programmer comes up with their own version of pseudocode. The programmer just needs to ensure that pseudocode is understandable to other people and that it is detailed enough that translation into java or other programming language is simple. There should be an (almost) one-to-one correspondence between lines of pseudocode and lines of Java.

The java version for the remove algorithm follows.

```
public boolean remove(Person target) {
    // search the list for the specified friend
    int i  = 0;
    while ((i < numFriends) && !friendList[i].equals(target))
        i++;

    if (i == numFriends) return false;
    else {
        // person found, remove by replacing with last one
        friendList[i] = friendList[numFriends - 1];
        friendList[numFriends - 1] = null;
        numFriends --;
        return true;
         }
}
```

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
- Where is the equals method of the line
  `friendList[i].equals(target)` defined?

Example: Using the SocialNetwork Class

Python:

```
def main():
        from SocialNetwork import SocialNetwork
        from Person import Person

        contacts = SocialNetwork();
        contacts.add("Snoopy","Dog","snoopy@uwo.ca");
        contacts.add("Felix","Cat","felix@uwo.ca");
        contacts.add("Mickey","Mouse","mickey@uwo.ca");
        print(contacts)
          print("I have " , contacts.getNumFriends() , " friends in my contact
    list")

    main()
```

Java:

```
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.

- In other words, the following two lines are equivalent:

  System.out.println(contacts)
  System.out.println(contacts.toString(
  ));

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**

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 ***passed by value*** 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**
- **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**