COMP 202 Review

Instructors: Emily Sager, Valerie Saunders Duncan

Basic Plan

~ 90 min -- Review of Concepts (focusing on the harder material)
~ 10 min -- break
~ 50 min -- Selected practice problems from the 2013 practice final
~ 30 min -- Questions

A few things to note:

- These slides will be made available after the review
- All example code will be made available after the review

Types

- The following are primitive types:
  - byte, short, long, float (not that important for this course)
  - int, boolean, char, double
- The following are reference types:
  - String, Array, Integer
  - Objects are reference types!
    - e.g. a Country class with attributes name, cities

More On Primitive Types

- int → represents an integer number in memory
- double → represents a decimal place number in memory

- boolean → two values: True and False
  - All if statements must evaluate to a boolean value
  - Incorrect:
    - `if (x = 7) { some code }` //x is being assigned value 7
    - `if (y = true) {some code }` //this is dangerous, y is being assigned the value of true! so the statement always evaluates to true!
  - Correct:
    - `if (x == 7) { some code }`
    - `if (y) { some code }`
- char → represent one character in memory
  - e.g. ‘t’ with ASCII value 116
  - can do comparison operations on chars (e.g. `if (‘t’ < ‘y’) { some code}`)

Example on Casting

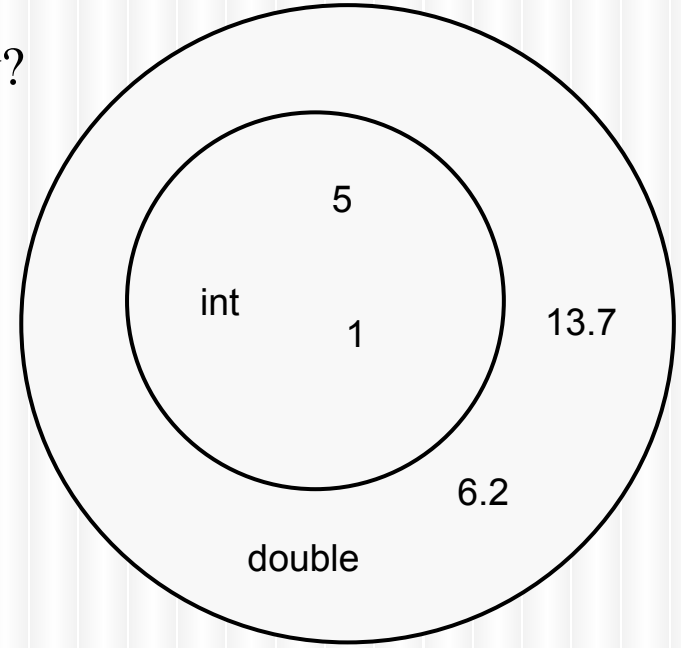
- Example: Which of the following lines will NOT compile?

```
public static void main(String[] args)
{
    int myInt = 5;
    double myDouble = 5.0;
    fooDouble(myInt);  // Line 1
    fooInt(myInt);  // Line 2
    fooDouble(myDouble); //Line 3
    fooInt(myDouble);  //Line 4
}

public static void fooDouble(double x)
{
    System.out.println(x);
}

public static void fooInt(int x)
{
    System.out.println(x);
}
```

Casting continued

- Any method that is expecting a double will happily accept an int
- Any method that is expecting an int will NEVER accept a double

Why?



Methods

- Basic method structure:
  - visibility staticOrNot returnType name(parameters) { code }
  - public static String foo(int x){does something}

- Think of a method as a Black Box

For Loop

```
public static void main(String[] args)
    {
        for (int i =1; i<25; i*5)
        {
            for (int j = 0; j< 10; j++)
            {
                System.out.println("x");          }
        }
    }
```

Loops and Search

- The exam is mostly going to ask you about the number of comparisons a search or sort algorithm does.
- Very important to understand when a loop or a method terminates

Loops and Search

```
public static int BinarySearchAlgorithm(int [] array, int minIndex, int
maxIndex, int target)
    {
        while (maxIndex>=minIndex)
        {
            int midpoint = (maxIndex+minIndex)/2;

            if (array[midpoint] == target )
            {
                return midpoint;
            }
            else if (array[midpoint] < target)  //search the upper half of
the array
            {
                minIndex = midpoint + 1;
            }
            else //search the lower half of the array
            {
                maxIndex = midpoint - 1;
            }
        }
        return -1;
    }
```

Loops and Binary Search

- Suppose we have an array of size 16, and we are searching for an element that is NOT in the array
- How many iterations of the while loop would occur?

Scope and Methods

- Variables only exist inside their block of code { }
- What does the following code print?

```
public static void main(String [] args)
    {
        int x = 5;
        int y = 10;
        foo(x, y);
        foo(x, y);
    }

    private static int foo(int x, int y)
    {
        x = x + 10;
        y = x;
        System.out.println(y);
        return y;
    }
```

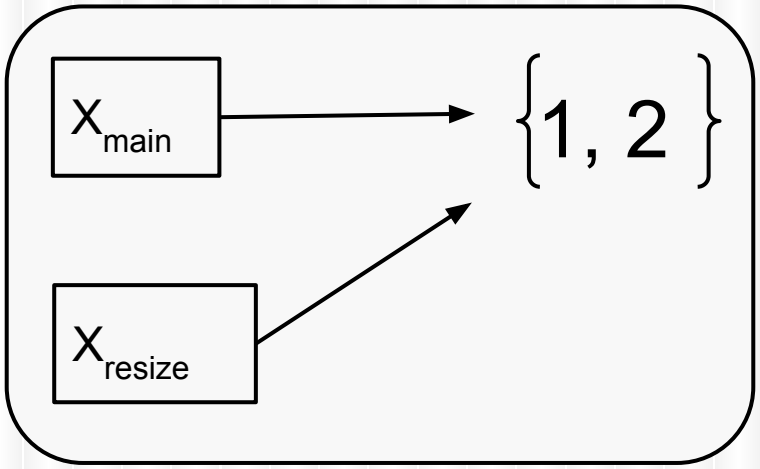
Scope and Methods

- Variables only exist inside their block of code { }
- This is even true for reference types!
- What does this code print?

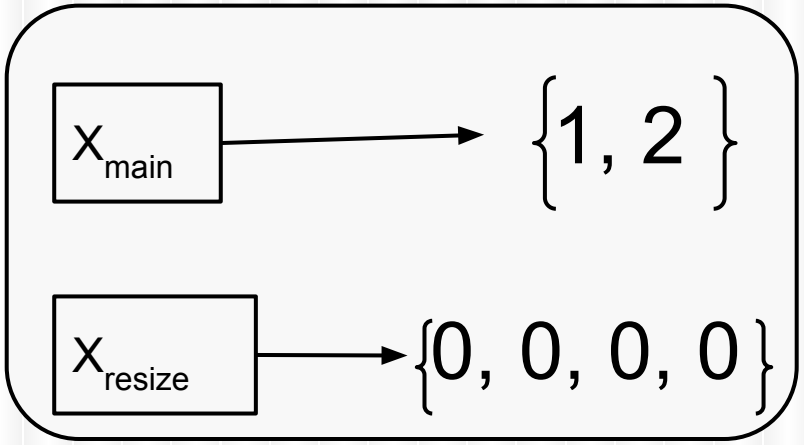
```
public static void resize(int[] y)
{
        y = new int[y.length * 2];
        System.out.print(y.length + " ");
}

public static void main(String[] args)
{
    int[] x = {1,2};
    resize(x);
    System.out.println(x.length);
}
```

Scope and Methods

Before any line of resize occurs, the variables look like this:



After resize performs the following line of code `x = new int[x.length * 2];`



Scope and Loops

- Variables must be declared in the scope they will be used.
- What (if anything) is wrong with the following code?

```
public static void foo()
{
    for (int i = 0; i < 10; i++)
    {
        int y = 7;
    }
    System.out.println(y);
}
```

Scope and Loops

- Variables must be declared in the scope they will be used.
- What (if anything) is wrong with the following code?

```
public static void foo()
{
    int y = 5;
    for (int i = 0; i < 10; i++)
    {
        int y = 7;
    }
    System.out.println(y);
}
```

Scope and Loops

- Variables must be declared in the scope they will be used.
- What (if anything) is wrong with the following code?

```
public static void foo()
{
    int y = 5;
    for (int i = 0; i < 10; i++)
    {
        int y = 7;
    }
    int y = 9;
    System.out.println(y);
}
```

Classes and Objects -- The basics

- Imagine that you and all your friends are separately cooking a fancy meal
- You will all follow a similar recipe, but will ultimately end up with different meals
- In coding, a class is the recipe and an Object is the meal.
- All the different meals made by you and your friends are *instances* of the class meal.
- Instances of a class are objects
- Objects are Reference Types

Objects

- They have attributes
- Attributes get assigned in the object constructor
- They have methods -- think about what functionality an object should be providing
- Some of these methods simply tell us information about the object
  - These are called getters
- Some of these methods manipulate attributes of the object
  - These are called setters

Static vs. non-static

- Non-Static methods are simply methods called on an instance of a class.
- An example you've used all the time is the .equals() method.
  - String myString = "HelloWorld!";
  - String yourString = "HelloThere!";
  - System.out.println(myString.equals(yourString));

- Another Example:
  - Car myCar = new Car ("Audi");
    myCar.drive();

- The "this" keyword is sometimes used in non-static methods to refer to the instance that the method was called on.

Non-Static and “this”

- The “this” keyword is sometimes used in non-static methods to refer to the instance that the method was called on.
    - Consider two car objects, JanesCar and JohnsCar. Let the Car class have the following method:

```
public void stop()
{
     if (this.getSpeed()!= 0)
     {
               this.setSpeed(0);
     }

     System.out.println(this.getCarOwnerName() +
     “‘s car has stopped.”);
}
```

```
JanesCar.stop();
JohnsCar.stop();
```

Static vs. non-static

- Sometimes it makes more sense to have a Static method.

- For example, many utility functions belong in a static method.
  - Eg. Translating a sentence into pig latin
  - Eg. Adding two numbers
- Stat

Good resources: http://www.javatpoint.com/static-keyword-in-java

ArrayLists

- ArrayLists are objects
- They contain other objects, but not primitive types → Can package up a primitive type into a wrapper type
- How are they different from arrays?
  - Don't need to know the size beforehand
  - Have different methods defined on them
  - Can use an enhanced for loop with ArrayLists
  - both of the following iterate over an ArrayList<City>
    - `for (City lCity: citiesArrayList)`
    - `for (int i = 0; i<citiesArrayList.size(); i++)`

ArrayList

But they are really just a better array!

You only need to know how to interact with an ArrayList, not how it fully works. Specifically, you need to know the following methods:

add(E e) -- adds an element to your arraylist
add(int index, E element) -- adds an element to your arraylist at the specified position
clear() -- deletes all the elements from the arraylist
get(int index) -- returns the object at index
indexOf(Object o) -- returns the index of the the item you're looking for
remove(Object o) -- removes this item
remove(int index) -- removes the item at index
size() -- gives you the length of the arraylist

ArrayList Examples

Consider the following chunk of code:

```
ArrayList<Double> myList = new ArrayList<Double>();
Double y = new Double(5);
Double x = new Double(7);
myList.add(y);
myList.add(x);
```

Which of the following statements are correct (assume this occurs in the same scope as the above code)?

a. `myList.add(10);`

b. `myList.add(10.0);`

c. `myList.remove(3);`

d. `myList.indexOf(x);`

e.
```
Integer z = new Integer(10);
myList.add(z);
```

ArrayList Examples

Consider the following chunk of code:

```
ArrayList<Double> myList = new ArrayList<Double>();
Double y = new Double(5);
Double x = new Double(7);
myList.add(y);
myList.add(x);
```

Which of the following statements are correct (assume this occurs in the same scope as the above code)?

a. `myList.add(10);  //Compiler Error`

b. **`myList.add(10.0);`**

c. `myList.remove(3);  //Runtime Error`

d. **`myList.indexOf(x);`**

e.
```
Integer z = new Integer(10); //Compiler Error
myList.add(z);
```

Exceptions

- Sometimes things can go wrong in our code, and we need a safety net to “catch” what went wrong.

- Exceptions at their core are just “alerts” telling the programmer, user, or other parts of code that something out of the ordinary occurred.

- Example:
  - You are writing a game that allows four players
  - a Fifth player **tries** to join the game
  - your code throws a TooManyPlayersException when the fifth player tries to join
  - your code catches this exception, and lets the fifth player know that they can’t join the game

Exceptions

```
public class Player
{

\\... some code

public void joinGame(Game pGame)
{

try{
     pGame.addPlayer(this);
}

catch (TooManyPlayersException e)
{
     System.out.println("There are
     already four players in this game.
     Please create a new game");
}
}
}
```

```
public class Game
{
     private ArrayList<Player> aPlayers;
\\... some code

public void addPlayer(Player p) throws
TooManyPlayersException
{
     if (aPlayers.size()==4){
     throw new TooManyPlayersException()
     }
     else{
          aPlayers.add(p);
     }
}
```

Two Examples in Eclipse

- File Reading
- Meal

Reference Types and Objects

"Canada"

Cities array:
[Montreal, Toronto, Vancouver]

Montreal, Toronto, and Vancouver are *instances* of a class city. They have type City!

Reference Types and Objects

"Canada"

Cities array:
[Montreal, Toronto, Vancouver]

[Toronto, Vancouver]

[Montreal, Vancouver]

[Montreal, Toronto]

Example

- Let's say in the Country class we have the following method:

```
public City[] getCities()
{
    return aCities;
}
```

- The main method looks like this:

```
public static void main(String [] args)
{
    City Montreal = new City("Montreal");
    City Toronto = new City("Toronto");
    City Vancouver = new City("Vancouver");
    City[] lCities = {Montreal, Toronto, Vancouver};
    Country Canada = new Country("Canada", lCities);
    City [] canadasCities = Canada.getCities();
    canadasCities[0] = new City("Boston");
}
```

At first Canada looks like this:

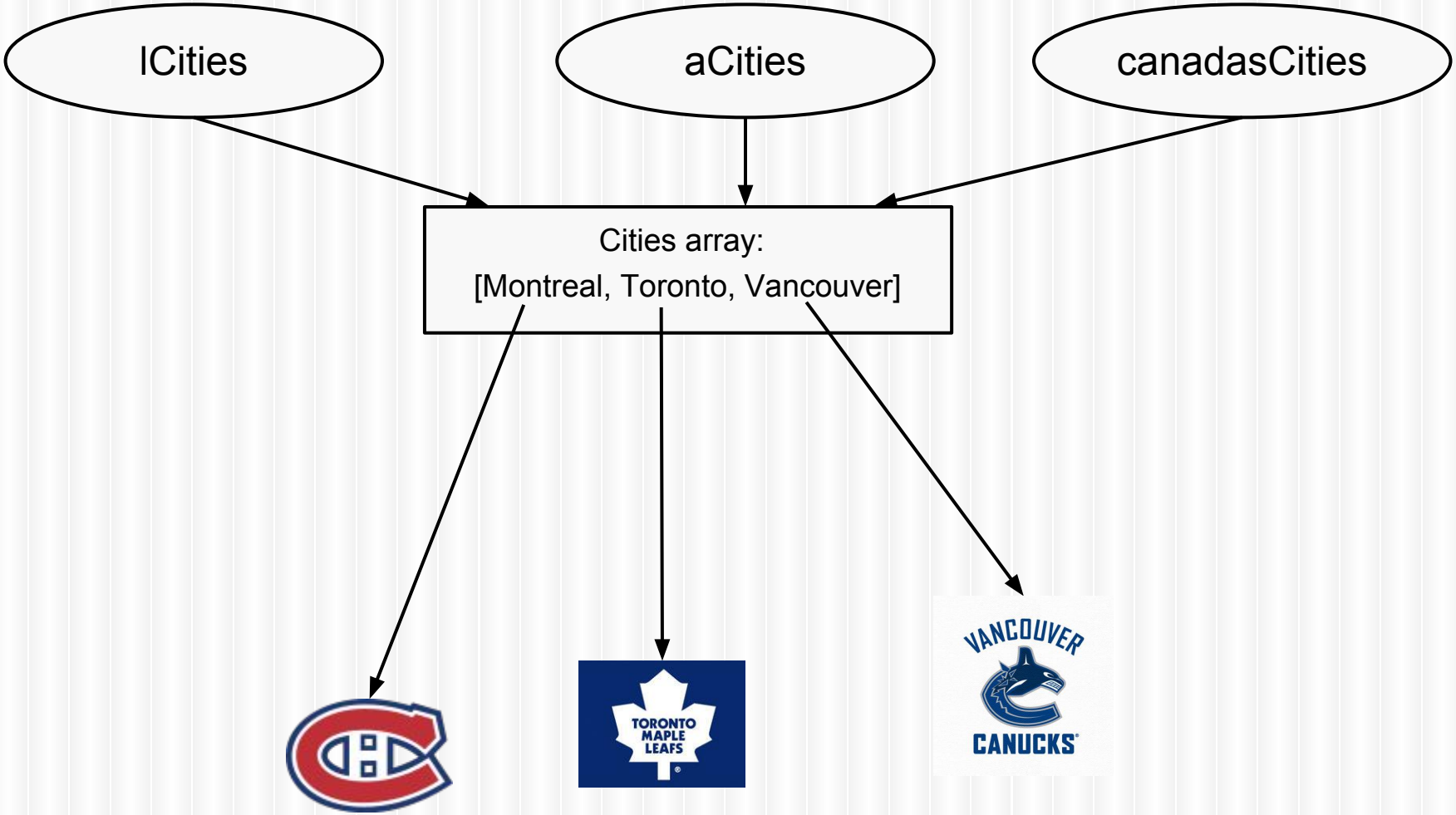
aName

“Canada”

aCities

Cities array:
[Montreal, Toronto, Vancouver]

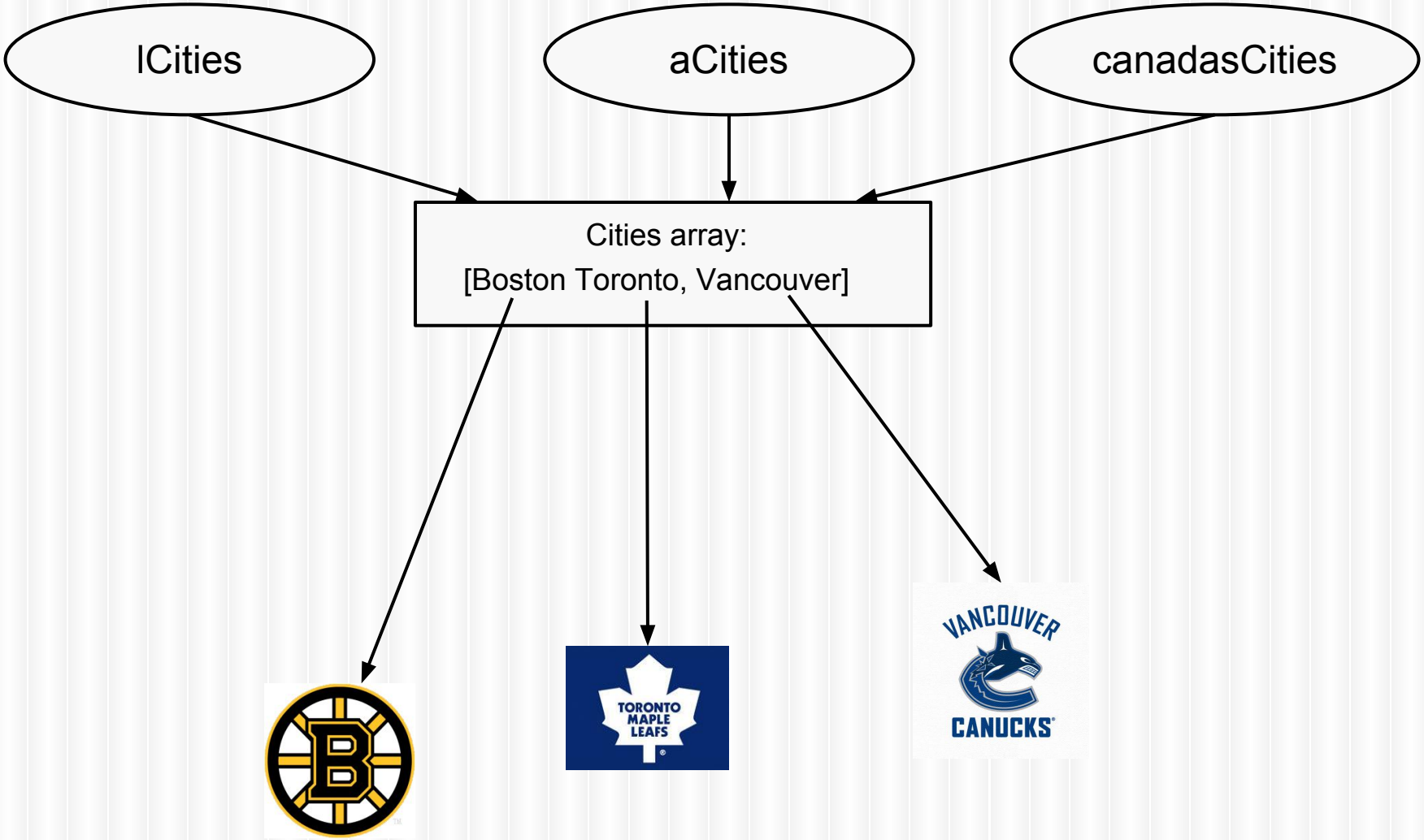
The arrays lCities, aCities, and canadasCities end up all referencing the same place in memory!



So when this line of code occurs:

```
canadasCities [0] = new City("Boston");
```

all the variables referencing the array get updated



Now Canada looks like this:

aName

“Canada”

aCities

Cities array:
[Montreal, Toronto, Vancouver]

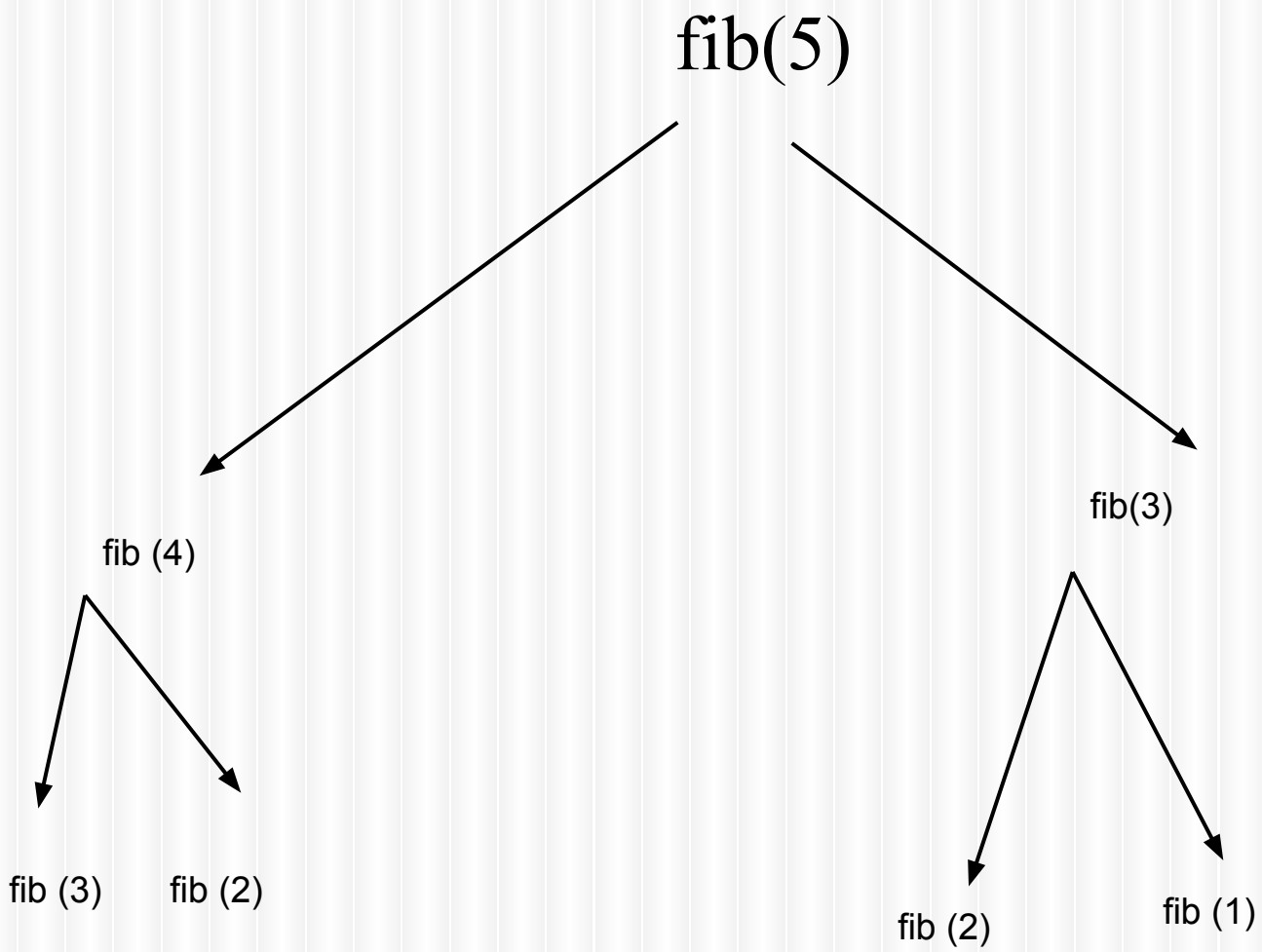
Recursion

- A Fancy way to say a method that calls itself

Example:

```
public static int fibonacciRecusion(int number){
        if(number == 1 || number == 2){
            return 1;
        }

return fibonacciRecusion(number-1) + fibonacciRecusion(number -2);
    }
```

Unwinding the recursion



Wrapper Classes

There are 8 wrapper classes for each of the 8 primitive types.

Wrapper classes make Objects out of primitive types!

```
int x = 10;
Integer i= new Integer(x);
```

Why is this important?

Wrapper Classes

There are 8 wrapper classes for each of the 8 primitive types.

Wrapper classes make Objects out of primitive types!

```
int x = 10;
Integer i= new Integer(x);
```

Why is this important?

Consider an ArrayList class with the following method signature:

```
public void add(Object o){
    //adds object to ArrayList
}
```

Wrapper Classes

Consider an ArrayList class with the following method signature:

```
public void add(Object o){
    //adds object to ArrayList
}
```

Let's say we want to store an ArrayList of ints! Well, technically we can't do this, because an int isn't a reference type, and an object is… wrapper types are the workaround!

HashSet

add(E e) -- Add an element to the set
contains(Object o)-- returns true if element O is in the set
remove(Object o) -- remove element
 size()-- return number of elements in the set

HashMap

This is a map! Let's say we want to **associate** a string with an integer (ie. we want to enter social insurance number and return a name), we can do so like this:

```
public static void main(String [] args){
    HashMap<Integer, String> SINmap = new HashMap<Integer, String>();
    SINmap.put(123456789, “Aubrey Graham”);
    SINmap.put(928435393, “Justin Bieber”);
    SINmap.put(712897365, “Dwayne Johnson”);
}
```

HashMap

We can update the value of an entry by simply inputting the same number, and a different name:

Note that a new entry is NOT created, the old one is overwritten!

```
public static void main(String [] args){
    HashMap<Integer, String> SINmap = new HashMap<Integer, String>();
    SINmap.put(123456789, “Aubrey Graham”);
    SINmap.put(928435393, “Justin Bieber”);
    SINmap.put(712897365, “Dwayne Johnson”);
}
```

```
public static void updateNames(HashMap<Integer, String> SINmap){
    HashMap<Integer, String> SINmap = new HashMap<Integer, String>();
    SINmap.put(123456789, “Drake”);
    SINmap.put(712897365, “The Rock”);
}
```

Enhanced-For-Loop

ArrayList, HashSet, HashMap can all be looped over in the following way:

```
ArrayList<String> myArrayList = new ArrayList<String>();
        :
        :
    add some stuff to myArrayList
        :
        :
for(String s: myArrayList){
    System.out.print(s);
}
```

Coding Question

You've been hired by YellowPages to program a phone book in Java. All they told you was that there was a file called phone_numbers.txt, which is a .csv containing all the phone numbers for the montreal area. Some examples from the file are below:

John Smith, 514-227-1919
Juliana Brown, 438-229-1717

In order to make a successful phone book, you need to write three methods in the class PhoneBook.

PhoneBook has two private properties:
A String, **cityNam**e, which stores the name of the city of the phonebook (in our case montreal), and a HashMap<String, Integer> **aPhoneBook**, which stores the names and numbers of the inhabitants of the city. To complete this class you'll need to write three methods:

HashMap<String, Integer> readFile(String filename): which takes as input a filename, and outputs a HashMap<String, Integer> representing the phonebook.

A constructor, PhoneBook, which takes as input two Strings representing the filename and the city name. The constructor should set the city name, and call readFile to set the HashMap.

The final method is a method called getNumber, which takes as input a person's name and outputs their phone number as a String.
All methods are public and non-static.

Coding Questions

Cypher (taken from 2009 exam):

http://s3.amazonaws.com/docuum/attachments/5649/Midterm%20Fall%2009.pdf?1284355943