INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney

Lecture 14: More Classes and Objects

reference type, instance vs static, the `this` keyword

Define a WorldPoint class

Let WorldPoint represent a geographical coordinate on the surface of the earth[1]

Stores a name AND two floating point numbers to represent the coordinate

Default values are Greenwich (51.48, 0)

Can *optionally* be initialised with values

Values are read/write, but only through supported operation (methods)

WorldPoint has range restrictions

- latitude - South to North is [-90, 90]
- longitude - West to East is [-180, 180]

[1]biaxial ellipsoid

WorldPoint operations

Report the Latitude

Report the Longitude

Report the name

Report the name, Latitude and Longitude as a formatted String

Report both Latitude and Longitude as an array

Set Latitude

Set Longitude

Report the Euclidean distance of this WorldPoint to another WorldPoint

Using WorldPoint

Write a program to

- Construct 3 WorldPoint objects
- Initialise each WorldPoint from command line arguments
- Print the information of the WorldPoint

```
   ~> javac MainProgram.java WorldPoint.java
   ~> java MainProgram Sydney 33.87S 151.21E    Moscow 55.75N 37.62E
               Manitoba 49.54N 97.08W
Sydney::Latitude: -33.87 Longitude:151.21
Moscow::Latitude: 55.75 Longitude:37.62
Manitoba::Latitude: 49.54 Longitude:-97.08
```

Reference value

When calling a method, we *copy* the value to be used in the method

```java
public static void printPlusOne(int x) {
    x = x + 1;
    System.out.println(x);
}

public static void main(String[] args) {
    int num = 75;
    printPlusOne(num);
    System.out.println(num);
}
```

When creating an Object we have a variable that stores a reference, the value of memory address

```java
WorldPoint wp = new WorldPoint("Beijing", 39.92, 116.38);
```

Reference value

When calling a method, the reference value is copied.

```
public static void initialiseData(int [] data, int offset) {
    int i = 0;
    while (i < data.length) {
        data[i] = i + offset;
        i = i  + 1;
        offset = offset + 1;
    }
}
public static void main(String[] args) {
    int offset = 10;
    int [] numbers =  new int[10];
    initialiseData(numbers, offset);
    System.out.println("offset: " + offset + " 6th number: " + number
}
```

Good news: Methods that have a reference can make changes to the object

Bad news: Methods that have a reference can make changes to the object

Reference value

When calling a method, the reference value is copied.

```java
public static void initialiseData(int [] data, int offset) {
    data = new int[10];
    int i = 0;
    while (i < data.length) {
        data[i] = i;
        i = i  + 1;
    }
}
public static void main(String[] args) {
    int [] numbers;
    initialiseData(numbers, offset);
    System.out.println("6th number: " + numbers[5]);
}
```

What is the output here?

Storing multiple return values using reference

When calling a method, the reference value is copied

Previously seen, values can be stored in a reference type

```java
// returns roots of quadratic equation
// ax^2 + bx + c = 0
// roots has at least two elements
public static void quadraticRoots
   (double a, double b, double c, double [] roots)
   throws NullPointerException, ArithmeticException
```

The same can apply to any Object

```java
// converts a human version of latitude/longitude to
// numeric form and sets those values in the WorldPoint object
public static void setLocation(WorldPoint point,
   String hLatitude, String hLongitude)
   throws NumberFormatException, IndexOutOfBoundsException
```

The above was not a supported operation of WorldPoint. Does it belong in the class WorldPoint?

Return value with reference

Define a class MatryoshkaDoll (Russian Doll)

A Matryoshka Doll contains zero or one Matryoshka Doll

A Matryoshka Doll has a size, 0 being the smallest

The size of the most outer doll can be increased

The size of any inner doll cannot change

The information about the inner doll has read access

Return value with reference

```java
public class MatryoshkaDoll {
    private int size; // only changes for outer
    private MatryoshkaDoll innerDoll; // cannot change

    // constructor?

    public void increaseSize() {
        size = size + 1;
    }

    // return copy for read access
    public MatryoshkaDoll getInner() {
        return innerDoll;
    }
}
```

Define a suitable constructor

This class does not satisfy: *The size of any inner doll cannot change*

Copying the reference value shares memory area with other code

static variables

The `static` keyword refers to variables or methods that belong to the `class`.

Instance variables belong to *one* instance, whereas `static` variables are common to *all* instances.

What is common to all objects? Identifiers, global values, shared settings

```
public class Student {

   private static int studentID = 0; // global counter for ID

   private String name;
   private int id;
   public Student() {
      id = studentID;
      studentID = studentID + 1; // increment global counter
   }
}
```

static methods

`static` methods can always be called without any objects ever being created.

methods without `static` must be associated with the memory of an instance.

`static` methods can be used to operate on static variables, or they can perform operations related to the class similar to a function (input, process, output)

```java
public class WorldPoint {
    ...

    // Latitude/Longitude converted to read as xxx S/N  yyy W/E
    // returns a String array of 2 elements. Each is positive num
    // 1st element has S/N, 2nd element W/E
    public static String[] getHumanReadable(WorldPoint point);
    public static String[] getHumanReadable(
                                double latitude, double longitude);
}
```

Another example class

Describe a student

A student has Name, SID, degree, year of birth

Information is read only

SID never changes for the student

Values initialised when object is constructed

Another example class

```java
public class Student {
    private String name;
    private final long SID;
    private String degree;
    private int yearOfBirth;
    public Student(String name, long SID, String degree, int yob) {
        this.name = name;
        this.SID = SID;
        this.degree = degree;
        yearOfBirth = yob;
    }
}
```

```java
public class StudentTest {
    public static void main(String[] args) {
        Student stud = new Student(); // won't compile
        Student stu = new Student("Stu", 430000001, "BSc/BE", 1995);
    }

}
```

this

this refers to “this” instance of a class. It is the same reference value returned from the constructor when instantiating a new object. The reference is for the *same area of memory*

this can be used to distinguish between local variable and class/instance variable

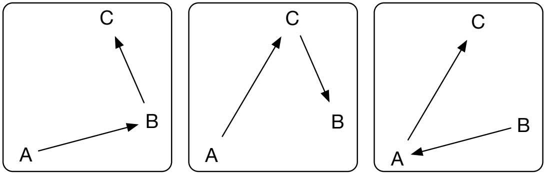
```java
public class Numbers {
    private int x, y;
    private double d;
    public Numbers(int x, int y, double d) {
        this.x = x;
        this.y = y;
        this.d = d;
        // "this" a reference to some area of memory for this object
    }
}
...
Numbers num = new Numbers(7, 3, 14.075);
// num is now a reference to some area of memory for this object
```

Review

class	the type of an object, e.g., the class `String`; to do with the type of an object, e.g. class variable or class method;
`Object`	the most basic class in `Java`;
instance	to do with a single copy or case of, e.g. "piano" is an *instance* of the type “MusicInstrument”
method	a separate block of code that can be called, e.g., `Integer.parseInt(String s)` or `length()` for a `String s`;
static	in the current context, applying to the whole class, as static variables, e.g. `System.out`), or static methods, e.g., `Integer.parseInt()`;

WorldPoint Shortest path

Calculate the shortest distance path of the three WorldPoint's and when found, print the name of the point and the distance from the previous point.



```
~> javac MainProgram.java WorldPoint.java
~> java MainProgram Sydney 33.52S 151.13E Perth 31.95S 115.86E
                                   Brisbane 27.47S 153.03E
Perth 0
Sydney 17542.643054323693
Brisbane 23145.74687879746
```

Can Euclidean distance measure with these values lead to false result?

Shortcuts: “for” loop

compact iteration [supp]

Loops revisited

`while` loop is simple and good

only need to consider the condition, when the loop keeps going or stops

other code can contribute to the setup and change in the loop condition

e.g. a counter variable initialised, checked, updated, checked, updated, checked, etc.

```
int counter = 0; // initialise
while ( counter < 10 ) // check condition
{
   counter = counter + 1;  // update something in each iteration
}
```

Most loops follow a general structure, can be shortened as a for loop

`for`

Syntax: `for` (*initialisation* ; *condition* ; *update*) *statement*

"First, do the *initialisation*; then, while the *condition* is true, execute the *statement*. After each execution of the *statement*, execute the *update*."

The four components are called (here, and in the course textbook) *initialisation*, *condition*, *update* and *statement*.

Inside `for`

Here's the order of execution of the `for` loop:

1. *Initialisation* always happens.
2. The *condition* is checked at the beginning of each iteration through the loop: if the condition is false, then the loop isn't executed, and won't be again.
3. The *statement* is executed.
4. The *update* is executed: usually this is incrementing or decrementing an index as you move through an array, but it can be anything you like.

Steps 2-4 are repeated until, when the *condition* is checked at Step 2, it is false.

Initialisation

This is executed *first*, even if the body of the loop is never entered: for instance, given the loop

```
    for (int x = 5; x < 5; x = x + 1) {
        System.out.println("Hah! Made it!");
    }
```

the message is never printed.

Variations on `for`

We can move things around so we can access the index variables outside, like this:

```java
    int i;
    for (i = 0; i < 10; i = i + 1) {
        System.out.println(i);
    }
    System.out.println(i); // still defined
```

```java
    int i;
    for (i = 10; i >= 0; i = i - 1) {
        System.out.println(i);
    }
    System.out.println(i); // still defined
```

for example

What will this print?

```java
public class ForExample {
    public static void main(String []  args) {
        int i = 10;
        int j = 0;
        for (j = 1; j < i; j = j + 1) {
            System.out.println("i.j = " + i + "." + j);
            i--;
        }
    }
}
```

for example

What will this print?

```java
public class ForExample {
    public static void main(String []  args) {
        int i = 10;
        int j = 0;
        for (j = 1; j < i; j = j + 1) {
            System.out.println("i.j = " + i + "." + j);
            i--;
        }
    }
}
```

```
~> javac ForExample.java
~> java ForExample
i.j = 10.1
i.j = 9.2
i.j = 8.3
i.j = 7.4
i.j = 6.5
```

More options with for

```
    for (int i = 0; ; i = i + 1) {
        System.out.println(i);
    }
```

If you miss out the *condition*, it will never be checked.

That means it can never be *false*, so you must have some other way of getting out of the loop or it will go on forever.

Another common(-ish) error with `for`

If you put a semicolon after the parentheses of the `for` loop like this:

```
1    int i = 0;
2    for ( ; i < 10; i = i + 1); {
3       System.out.println("i = " + i);
4    }
```

the code will compile fine.

What will be printed? It's a very subtle error that often occurs and can be hard to spot: the semicolon ';' near the end of line 2 above means *there is no statement to execute in the loop*, so the `System.out.println` command isn't *in* the loop.

The `println` statement will therefore be executed *after* the loop.

What will this loop do?

```
    for (int i = 0; ; ) {
        System.out.println(i);
    }
```

for — everything is optional!

You can leave out any or all of the ingredients of the `for` loop:

`while` loop

```
int i = 0;
for ( ; i < 10 ; )
{
    i = i + 1;
}
```

What will it do?

The empty for loop

If the “loop” is just `for ( ; ; ) { }` then this will actually compile

1. nothing is initialised
2. nothing gets checked (so it can’t every be false!)
3. the body is empty: nothing gets executed
4. there’s no update

It never finishes.

Not useful everywhere

```java
int i = 0;
while ( i < 36 ) {
    if ( someCondition(someFunctionX()) ) {
        i = i + 1;
        continue;
    }
    if ( someCondition( someFunctionY(), someFunctionZ()) )
        continue;
    i = i + 3;
    System.out.println(" i " + i);
}
```

```java
int i = 0;
for ( ; i < 36 ; ) {
    if ( someCondition(someFunctionX()) ) {
        i = i + 1;
        continue;
    }
    if ( someCondition( someFunctionY(), someFunctionZ()) )
        continue;
    i = i + 3;
    System.out.println(" i " + i);
}
```