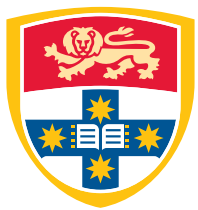
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Week 10: Programming Idioms, Inheritance

We will cover: Solving common problems, Refining class definitions

You should read: §§7.4, 8.1, 9.2

Lecture 19: Programming idioms

Identifying common problems and their solutions

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.

```
1 int counter = 0; // initialise
2 while ( counter < 10 ) // check condition
3 {
4     counter = counter + 1;  // update something in each iteration
5 }
```

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

```
1    for (int x = 5; x < 5; x = x + 1) {
2        System.out.println("Hah! Made it!");
3    }
```

the message is never printed.

Variations on `for`

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

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

```
1    int i;
2    for (i = 10; i >= 0; i = i - 1) {
3        System.out.println(i);
4    }
5    System.out.println(i); // still defined
```

for example

What will this print?

```
1  public class ForExample {
2      public static void main(String []  args) {
3          int i = 10;
4          int j = 0;
5          for (j = 1; j < i; j = j + 1) {
6              System.out.println("i.j = " + i + "." + j);
7              i--;
8          }
9      }
10 }
```

for example

What will this print?

```
1  public class ForExample {
2      public static void main(String []  args) {
3          int i = 10;
4          int j = 0;
5          for (j = 1; j < i; j = j + 1) {
6              System.out.println("i.j = " + i + "." + j);
7              i--;
8          }
9      }
10 }
```

```
~> 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

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

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?

```
1    for (int i = 0; ; ) {
2        System.out.println(i);
3    }
```

`for` — everything is optional!

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

`while` loop

```
1  int i = 0;
2  for ( ; i < 10 ; )
3  {
4      i = i + 1;
5  }
```

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

```
1  int i = 0;
2  while ( i < 36 ) {
3     if ( someCondition(someFunctionX()) ) {
4        i = i + 1;
5        continue;
6     }
7     if ( someCondition( someFunctionY(), someFunctionZ()) )
8        continue;
9     i = i + 3;
10    System.out.println(" i " + i);
11 }
```

```
1  int i = 0;
2  for ( ; i < 36 ; ) {
3     if ( someCondition(someFunctionX()) ) {
4        i = i + 1;
5        continue;
6     }
7     if ( someCondition( someFunctionY(), someFunctionZ()) )
8        continue;
9     i = i + 3;
10    System.out.println(" i " + i);
11 }
```

Programming Idioms

Common tasks

Programming Idioms

There are many small problems to solve in programming

A programming idiom is a commonly used method to solve a small problem

These are all related to search

Find the smallest of two numbers

```
1 if (x < y)
2     // x is the smallest
3 else
4     // y is the smallest
```

Programming Idioms

Loop exactly N times

```
1  for (int i = 0; i < 10; i++) {
2
3  }
```

STOP when you have reached the goal

```
1  for (int i = 0; i < 10000000; i++) {
2      ...
3      if ( goal is true )
4          break;
5      ...
6  }
```

Sequence control flow

Not so common. Skip every *n*th number

Even and odd numbers

```
1  for (int i = 0; i < 10; i++) {
2      if ( i % 2 == 0 ) {
3          // do something when even sequence
4      }
5  }
```

Special sequence: every 7th number OR every 3rd number

```
1  for (int i = 0; i < 100; i++) {
2      if ( (i % 7 == 0) || (i % 3 == 0) ) {
3          // do something when 0, 3, 6, 7, 9, 12, 14, 15, 18, 21
4          //                   ^          ^              ^
5      }
6  }
```

Sequence control flow

one thing every 7th number, otherwise a different thing every 3rd number

```
1  for (int i = 0; i < 100; i++) {
2      if ( i % 7 == 0 ) {
3          // do something when 0, 7, 14, 21
4      } else if ( i % 3 == 0) {
5          // do something when 3, 6, 9, 12
6          // WILL NOT HAPPEN when i == 0
7      }
8  }
```

What about this?

```
1  for (int i = 0; i < 100; i++) {
2      if ( i == 0 ) {
3          ...statement block X...
4          continue;
5      }
6      if ( i >= 54 && i < 60 ) {
7          ...statement block Y...
8      } else {
9          ...statement block Z...
10     }
11 }
```

Idioms with arrays

Programs tend to work with same data but different size. e.g. sum of N numbers, where N could be small, or really big

The data is stored in some kind of memory, could be on disk as a file, or could be in main memory

To solve the problem, we have to search. We need to **visit** each number that is somewhere in memory/file

An array is the most typical way to represent N pieces of data, where each piece is the same data type (e.g. Integer)

The idiom is to scan the entire array and *do something* with each value

Idiom: computation over all values

Search for: The sum of N numbers

```
1  int[] numbers = ....
2  int N = numbers.length;
3
4  int sum = 0;
5  for (int i = 0; i < N; i++)
6  {
7      sum = sum + numbers[i];
8  }
```

Idiom: search for a specific value

Search for: The number of occurrences of a value

```
1  int[] numbers = ....
2  int N = numbers.length;
3  int value = 2; // searching for this
4
5  int count = 0;
6  for (int i = 0; i < N; i++)
7  {
8      if (numbers[i] == value)
9      {
10         count++;
11     }
12 }
```

Search failed?

```
13 if (count == 0) {
14     // did not find any occurrences of value
15 }
```

Idiom: search for a specific value

Search for: The minimum of N numbers

```
1  int[] numbers = ....
2  int N = numbers.length;
3
4  int min = Integer.MAX_VALUE; // largest possible
5  for (int i = 0; i < N; i++)
6     if (min > numbers[i])
7        min = numbers[i];
```

Search failed? can we say this?

```
8  if (min == Integer.MAX_VALUE) {
9      // no minimum found
10 }
```

Idiom: search for a specific value

Search for: The minimum of N Objects

What does minimum mean? e.g. for a Person Object

What if we don't know the largest possible value in advance?

```
1  Person[] person = ....
2  int N = person.length;
3
4  if (N > 0)
5  {
6     int min = person[0];
7     for (int i = 1; i < N; i++) // start from 1
8     {
9        // we have some way of asking is min > p[i]
10       if ( min.compareTo(persons[i]) > 0 ) {
11          min = persons[i];
12    }
13 }
```

Idiom: general idea with arrays

- A search task
- input is some array
- there is some criteria for the search
- there is a failed search outcome that needs to be considered

```
1  input: array[N]
2  output: result
3
4  LOOP over each element of the array
5  {
6        if ( search criteria ) {
7           update result
8        }
9  }
10
11 if (search failed criteria)
12    error handling/reporting
```

Idiom with Person Objects

Search for: elements in the array to be visited

A Person object has a date for their birthday

```
1  public class Person {
2      private SimpleDate birthday;
3      public SimpleDate getBirthday() {
4          return birthday.copy();
5      }
6  }
```

```
1  public class SimpleDate {
2      ...
3      public static SimpleDate copy() { ... }
4      public static boolean isEqual( SimpleDate a, SimpleDate b )
5      { ... }
6  }
```

Given an array of Persons and `SimpleDate today`. Print a message if their birthday is `today`.

Solve these small problems

Each problem can be solve using the general form of an idiom.

Given an array of integers. Increment each value by one.

Given an array of integers. For any value that is less than 5, set the value to 5.

Given an array of integers. Create a new array of integers that contains no negative values.

Given an array of Strings. Create a single new String that is the concatenation of each String in the array.

Do the previous one, in reverse concatenation order.