COMP1008 Recursion and More UML Examples

Recursion

- A method can call itself - recursion!

```
int factorial(int i)
{
  if (i > 0)
    return i * factorial(i-1) ;
  else
    return 1 ;
}
// See text book p199
```


Or

```
public int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}
```

Testing for *base case* explicitly

Tracing recursive calls

```
6!
(6 * 5!)
(6 * (5 * 4!))
(6 * (5 * (4 * 3!)))
(6 * (5 * (4 * (3 * 2!))))
(6 * (5 * (4 * (3 * (2 * 1!)))))
(6 * (5 * (4 * (3 * (2 * (1 * 0!))))))
(6 * (5 * (4 * (3 * (2 * (1 * 1))))))
(6 * (5 * (4 * (3 * (2 * 1)))))
(6 * (5 * (4 * (3 * 2))))
(6 * (5 * (4 * 6)))
(6 * (5 * 24))
(6 * 120)
720
```

A nice demonstration

- Found this at:
- http://www.cs.princeton.edu/introcs/lectures/

Recursion & iteration

- Recursion provides another way of doing iteration.
- Most use of recursion can be re-written using loops and vice-versa.
- Some algorithms are much easier to express using recursion.
- Remember all the Prolog...

Another example

```
int sum(final int n)
{
  if (n == 0)
  {
    return 0 ;
  }
  else
  {
    return sum(n-1) + n ;
  }
}
```

Sum the numbers from 1 to n.

Recursive calls made by sum

```
call sum(5)
sum(5):  n != 0, call sum(4)
sum(4):     n != 0, call sum(3)
sum(3):       n != 0, call sum(2)
sum(2):         n != 0, call sum(1)
sum(1):           n != 0, call sum(0)
sum(0):             n == 0, return 0
sum(1):           return 0 + 1 (1)
sum(2):         return 1 + 2 (3)
sum(3):       return 3 + 3 (6)
sum(4):     return 6 + 4 (10)
sum(5):  return 10 + 5 (15)
```

Linear Search

```
int linearSearch(int[] a, int value, int position) {
  if (position >= a.length)
    { return -1; }
  else {
    if (value == (a[position]))
      { return position; }
    }
  else
    { return linearSearch(a, value, position + 1); }
}
```

Base case - value not found

Value found

Recursive case

Called with
int n = linearSearch(array, 10, 0);

Remove else's

```
int linearSearch(int[] a, int value, int position) {
  if (position >= a.length)
    { return -1; }
  if (value == (a[position]))
    { return position; }
  return linearSearch(a, value, position + 1);
}
```

Shorter but possibly less readable. Not as clear that there are 3 options?

Binary Search

Array must be sorted

```
int binarySearch(int[] a, int low, int high, int value) {
    if (low > high) {
      return -1;
    }
    else {
      int middle = (low + high) / 2;
      if (value == a[middle])
        { return middle; }
      else {
        if (value < a[middle])
          { return binarySearch(a, low, middle - 1, value); }
        else
          {return binarySearch(a, middle + 1, high, value); }
      }
    }
}
```

Base case - value not found

Value found

Recursive cases

Quicksort

```
void quicksort(double[] a, int left, int right) {
  if (right <= left) return;
  int i = partition(a, left, right);
  quicksort(a, left, i-1);
  quicksort(a, i+1, right);
}
```

- Algorithm:
  - Split array into 2 partitions
    - such that all values in left are less than those in right.
  - recursively apply to partitions.
    - when partition is size one, it is sorted.

Partition

```
public static int partition(double[] a, int left, int right) {
  int i = left - 1;
  int j = right;
  while(true) {
    while (less(a[++i], a[right])) ;
    while (less(a[right], a[--j]))
      if (j == left) break;
    if (i >= j) break;
    swap(a, i, j);
  }
  swap(a, i, right);
  return i;
}
```

Take two partitions and swap values such that all values in left partition are less than all values in right.

Here is a demo...

What does this do?

```
void f(int a, int b) {
  if (a <= b) {
    int m = (a + b) / 2;
    System.out.print(m + " ");
    f(a, m-1);
    f(m+1, b);
  }
}
```


Cost of recursion

- Each recursive call has the 'cost' of a method call.
  - Recursion is potentially slower than iteration.
  - But a recursive algorithm may be more efficient.
- Each recursive call uses up a bit of memory.
- Too many recursive calls will use up all the stack space and the program will fail.
  - But not a reason to ignore recursion.

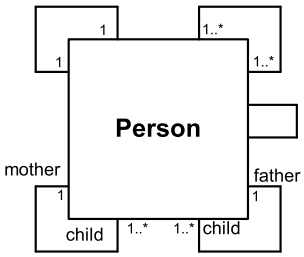
Recursion Pitfalls

- Infinite recursion.
  - fail to test for base cases, so recursion never terminates.
    - Program crashes with out of memory error.
- Mutual recursion.
  - Two method call each other recursively.
    - Directly or indirectly.
    - Usually a sign of poor design and can be difficult to spot.

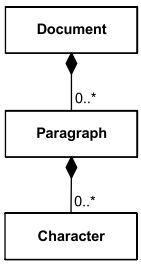
Questions?

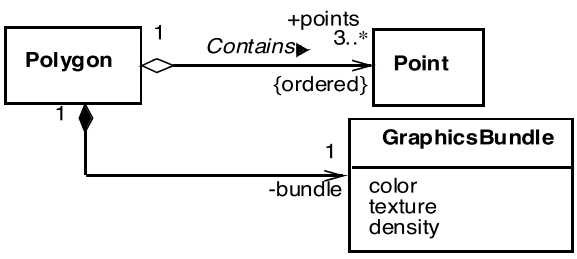
More UML Examples

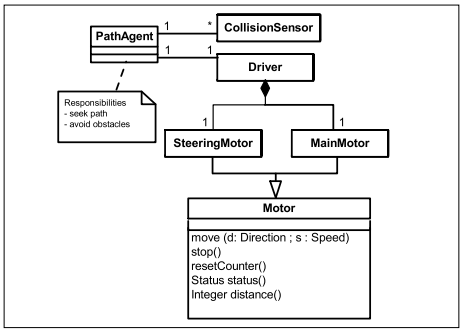
Family Trees

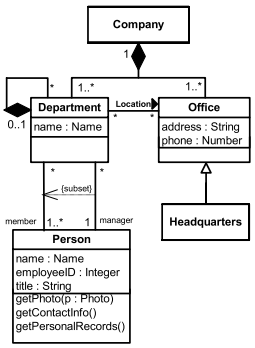


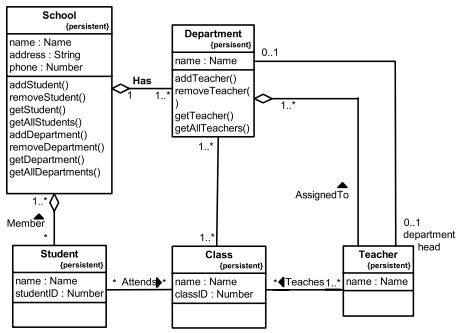
Composition - Strong Ownership



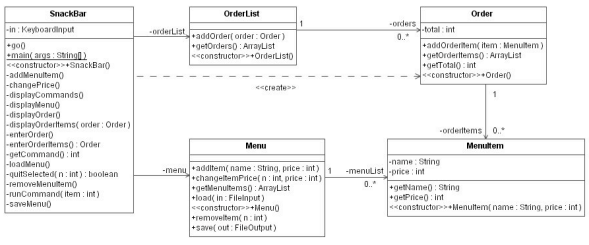




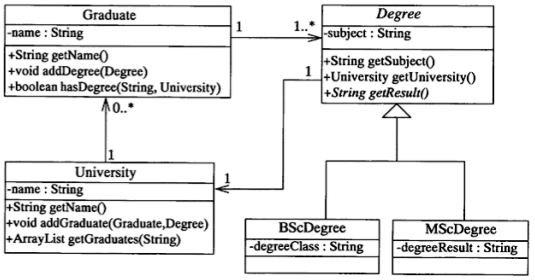




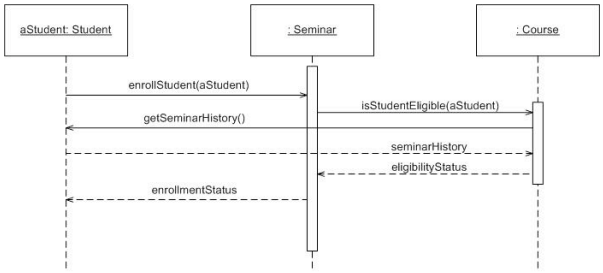
Snackbar example

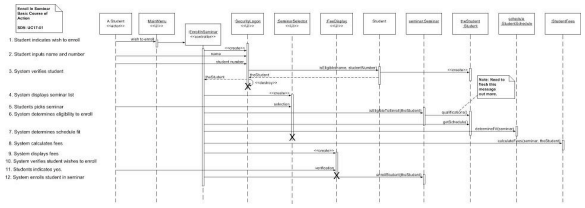


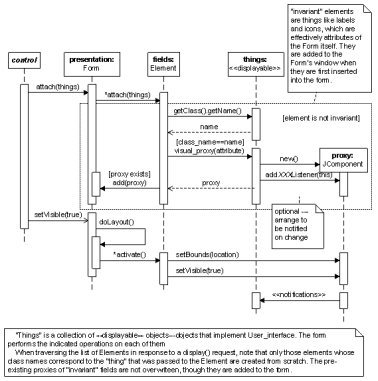
From a past exam paper...



UML Sequence Diagrams







Summary

- Recursion
  - another form of repetition
- More UML diagram examples