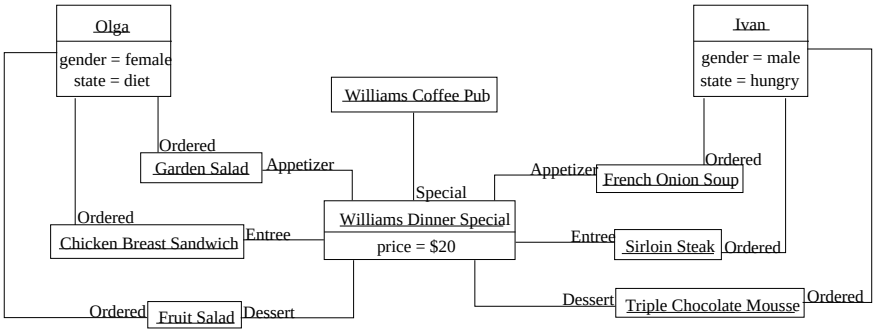
Exercise 1

Christie Chan ID:20216612 UserID:c45chan

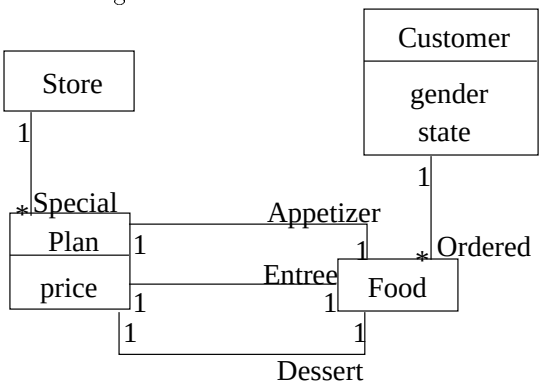
January 17, 2007

Part 1: Instance/Class Diagrams

1. Instance Diagram



2. Class Diagram



Part 2: C++ Basics

1.
```
#include <iostream>

int main() {
    int **array;        /* declare array */
```

```
    int sizex, sizey;   /* size of array */
    int i, j;           /* counters */

    /* suppose a 10x10 matrix */
    sizex = 10;
    sizey = 10;

    /* allocate storage for array of pointers */
    array = (int **)malloc(sizex * sizeof(int *));
    /* allocate storage for array of ints for each pointer */
    for (i = 0; i < sizex; i++) {
        array[i] = (int *)malloc(sizey * sizeof(int));
    }

    /* forming an Identity Matrix */
    for (i = 0; i < sizex; i++) {
        for (j = 0; j < sizey; j++) {
            if (i == j) array[i][j] = 1;
            else array[i][j] = 0;
        }
    }

    /* prints contents of matrix to standard output */
    for (i = 0; i < sizex; i++) {
        for (j = 0; j < sizey; j++) {
            std::cerr << array[i][j];
        }
        std::cerr << std::endl;
    }

    /* free array of ints for each pointer */
    for (i = 0; i < sizey; i++) {
        free(array[i]);
    }
    /* free array of pointers */
    free(array);

    return 0;
}
```

2. The output is "All Conditions Fail".
   "if((int)a == b)" fails because a is a pointer, and "int* a = new int(100);" gives the value 100 at pointer a, so *a = b, (int)a does not.
   "if((int)&b == c)" fails because (int)&b = the address of b, and c is the reference of b, which c=100, so (int)&b = (int)&c and b=c.
   "if((int)a != b && (int)&b != c)" passes since (int)a not = b and (int)&b

not = c as mentioned above.

3. Pass by reference passes the locations of the original values, and so changes to the variables passed by reference will affect the originals. On the other hand, pass by value passes the copies of the original values, and so changes made to these variables don't affect the originals. Java parameter passing mechanism doesn't allow programmers to specify passes by value or passes by reference, they are internally set. Java passes the copies of the values when parameters are type int, char, string, etc. Thou, it's different from passes by reference in C++. Parameters may passes by reference in some other object types, so it's not the same as passes by value neither.

4.
```
#include <iostream>

    int original = 10;          // initialize original value
    int byVal = original;       // pass by value
    int& byRef = original;      // pass by reference

void output() {
    std::cout << "Original value: " << original << std::endl;
    std::cout << "Variable passed by value: " << byVal <<std::endl;
    std::cout << "Variable passed by reference: " << byRef << std::endl;
    std::cout << std::endl;
}

int main() {
    std::cout << "Initially," << std::endl;
    output();

    original = 5;               // change original value
    std::cout << "When original value is changed to 5," << std::endl;
    output();

    original = 10;              // reset back to initial value
    byVal = 5;                  // change value of the variable passed by value
    std::cout << "When the variable passed by value is changed to 5," << std::endl;
    output();

    byVal = 10;                 // reset back to initial value
    byRef = 5;                  // change value of the variable passed by reference
    std::cout << "When the variable passed by reference is changed to 5," << std::endl;
    output();

    return 0;
}
```

Output:

Initially,
Original value: 10
Variable passed by value: 10
Variable passed by reference: 10

When original value is changed to 5,
Original value: 5
Variable passed by value: 10
Variable passed by reference: 5

When the variable passed by value is changed to 5,
Original value: 10
Variable passed by value: 5
Variable passed by reference: 10

When the variable passed by reference is changed to 5,
Original value: 5
Variable passed by value: 10
Variable passed by reference: 5

5. The code is incorrect because the "return &myObject;" doesn't necessary return the address of myObject. When MakeSomeObject() is executed, myObject is created in the stack and address of it is returned; however, if any other codes are executed that changes the stack, then the address may no longer points to myObject. Therefore, the code needs allocation and deallocation to be correct.