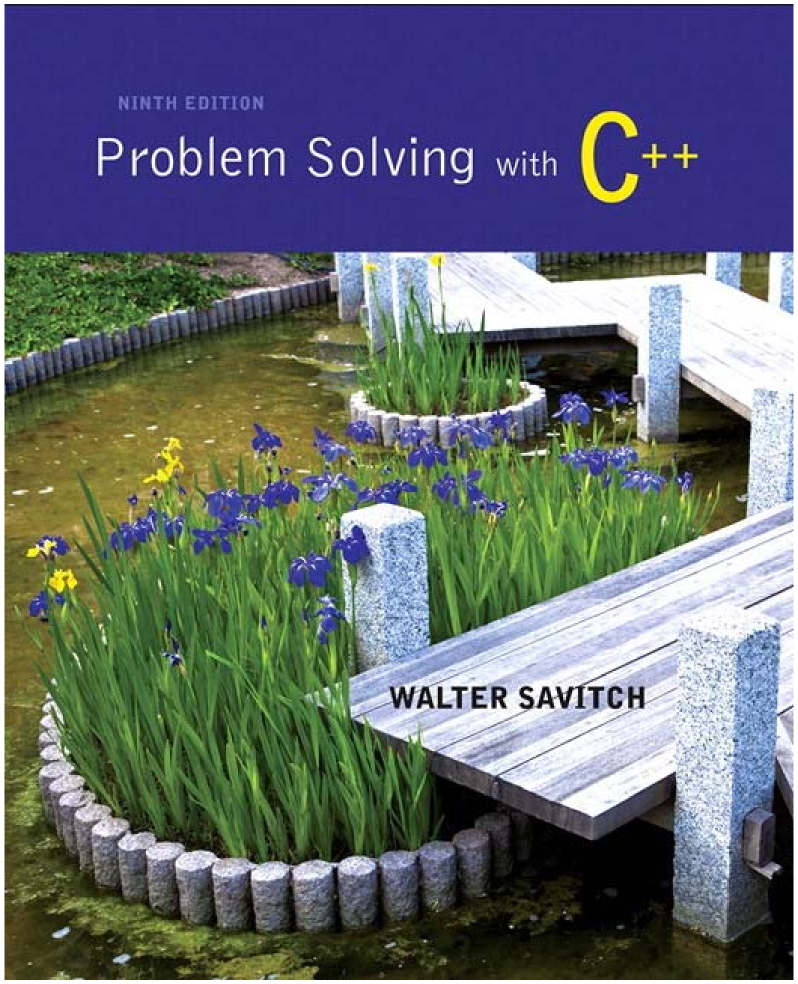
SFU SIMON FRASER UNIVERSITY BURNABY

# Review for Midterm



**Instructor: Scott Kristjanson**

CMPT 135

SFU Surrey, Spring 2016

## What will be covered by the Midterm?

**Selected material from these topics:**

Assignment 2 - Parts A, B, C1-C2

Assignment 1

UML Diagrams, Data Flow Diagrams, Testing

Chapter 7 - Arrays, Sections 7.1-7.2 only

Chapter 10 - Structs and Classes

Chapter 4 - Functions

Chapter 3 - Loops and Flow of Control

Chapter 2 - C++ Basics and Expressions

## Assignment 2 - Parts A, B, C1-C2

A1 - Chapter 10 - Classes (Reviewed in Lab08)

B1 - Class Definition Questions

B2 - Chapter 7 Arrays

B3 - Intro to Inheritance

B4 - Testing

B5 - Data Flow Diagrams

B6 - UML Diagrams

Coding

- C1-C4 - Arrays as Parameters to Functions

## Assignment 1 Review

A1 - Operator Precedence

A2 - Random Numbers

A3 - Definite Loops   (for, for-each)

A4 - Indefinite Loops (while, do-while)

Coding

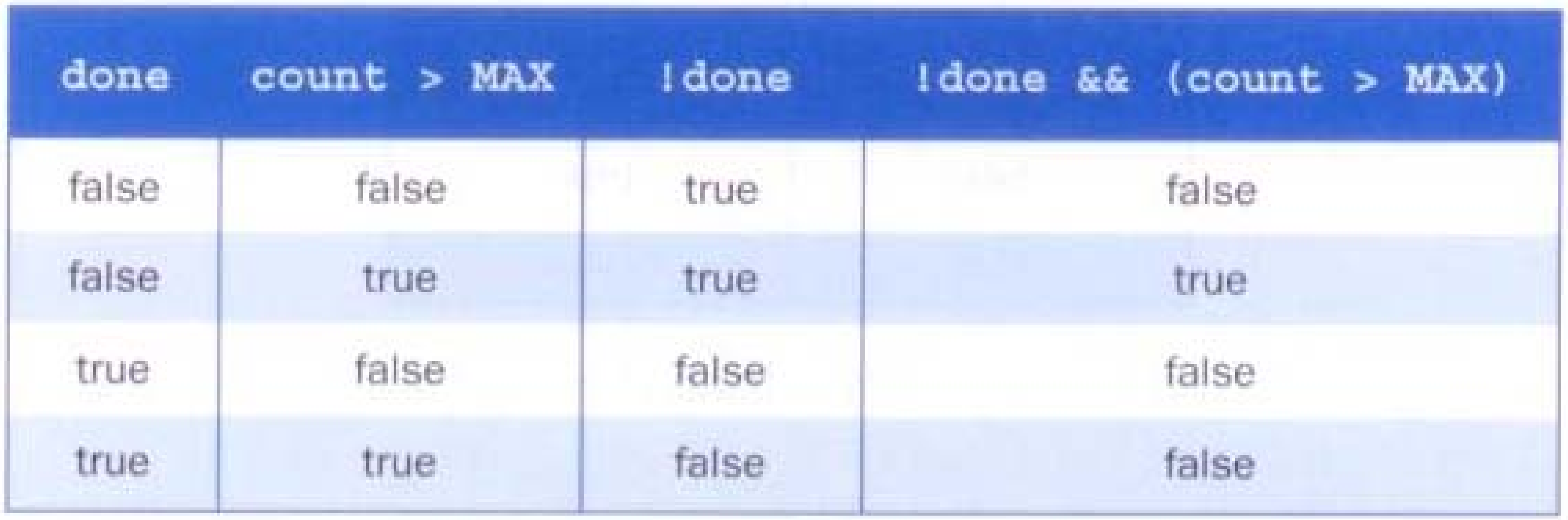
- B1 - Random Numbers
- B2 - CMath functions
- B3 - Stacks and Expression Processing
- B4 - Defining Classes

# Logical Operators and Truth Tables

Expressions can be evaluated using truth tables

For example, let’s evaluate the following using a truth table:

`!done && (count > MAX)`



| `done` | `count > MAX` | `!done` | `!done && (count > MAX)` |
|---|---|---|---|
| false | false | true | false |
| false | true | true | true |
| true | false | false | false |
| true | true | false | false |

## Midterm will test your Basic Knowledge from Cmpt130

From Cmpt 130, it's assumed that you will be familiar with the basic concepts of C programming. (Section numbers refer to the course text.)

- Data types and Variables (§2.1, 2.3)
- Expressions (§2.3)
- Strings (§2.3, 8.3)
- Conditionals (if-then-else) (§2.4)
- Definite (for) and indefinite (while) loops (§2.4-3.4)
- Functions and Procedures (§4.1-4.5, 5.1).
- Basic terminal input/output (§2.2)

# Know the definition of an Object

An object has

- *state* - time-varying data describes characteristics
- *behaviors* - what it can do (or what can be done to it)

The state of a bank account includes its account number and its current balance

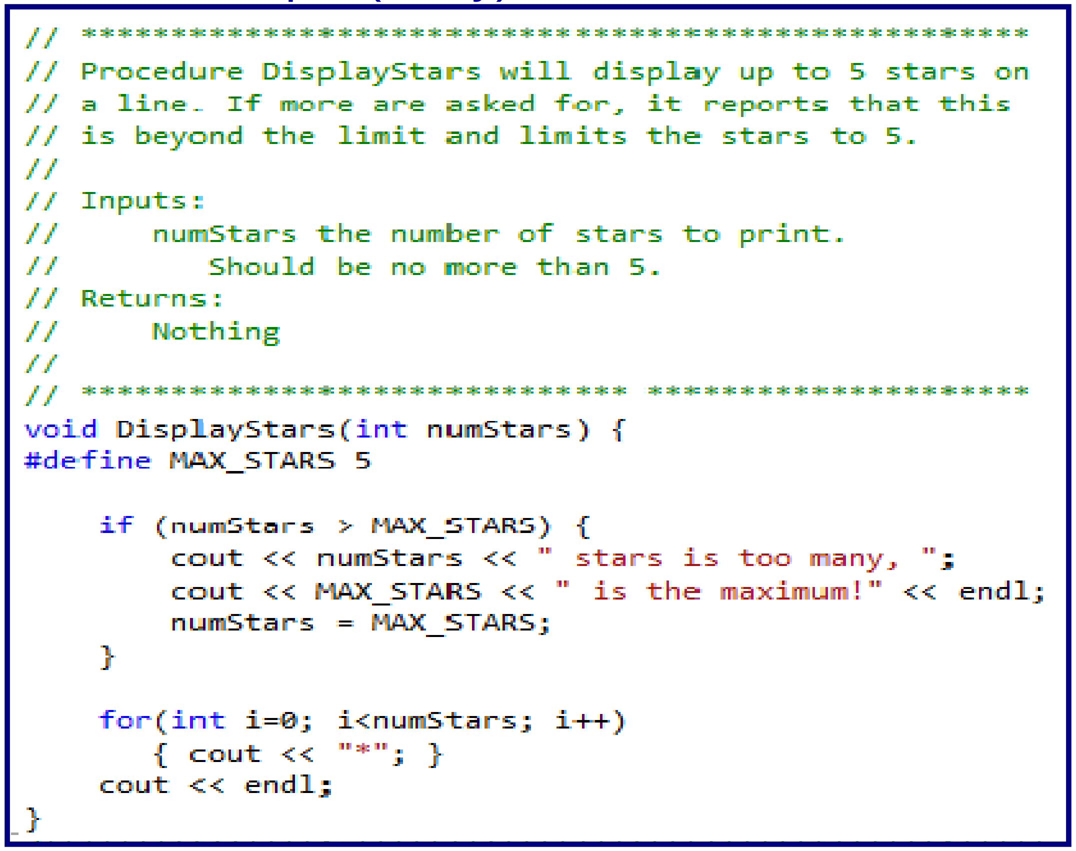
The behaviors associated with a bank account include the ability to make deposits and withdrawals

Note that the behavior of an object often changes its state

# Be able to Write a Helper Function

Block comment describes: (Not Required on Quiz/Exams!)

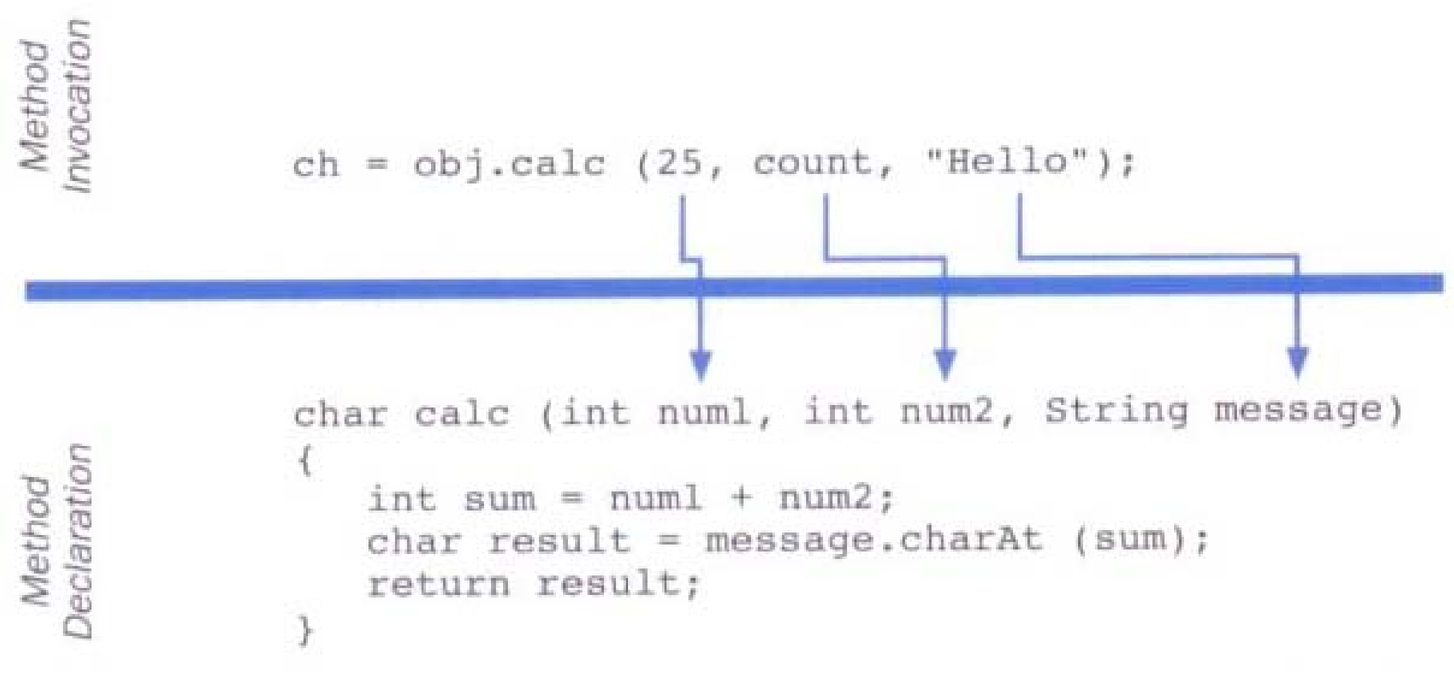
- High level description of what the function does, including side-effects
- Describe each input (if any)
- Describe each output (if any) and return value



```
// ****************************************************
// Procedure DisplayStars will display up to 5 stars on
// a line. If more are asked for, it reports that this
// is beyond the limit and limits the stars to 5.
//
// Inputs:
//     numStars the number of stars to print.
//        Should be no more than 5.
// Returns:
//     Nothing
//
// ****************************** *********************
void DisplayStars(int numStars) {
#define MAX_STARS 5

    if (numStars > MAX_STARS) {
        cout << numStars << " stars is too many, ";
        cout << MAX_STARS << " is the maximum!" << endl;
        numStars = MAX_STARS;
    }

    for(int i=0; i<numStars; i++)
       { cout << "*"; }
    cout << endl;
}
```

## Parameters: by Value, by Ref, by Array

When a method is called, the *actual parameters* in the invocation are copied into the *formal parameters* in the method header

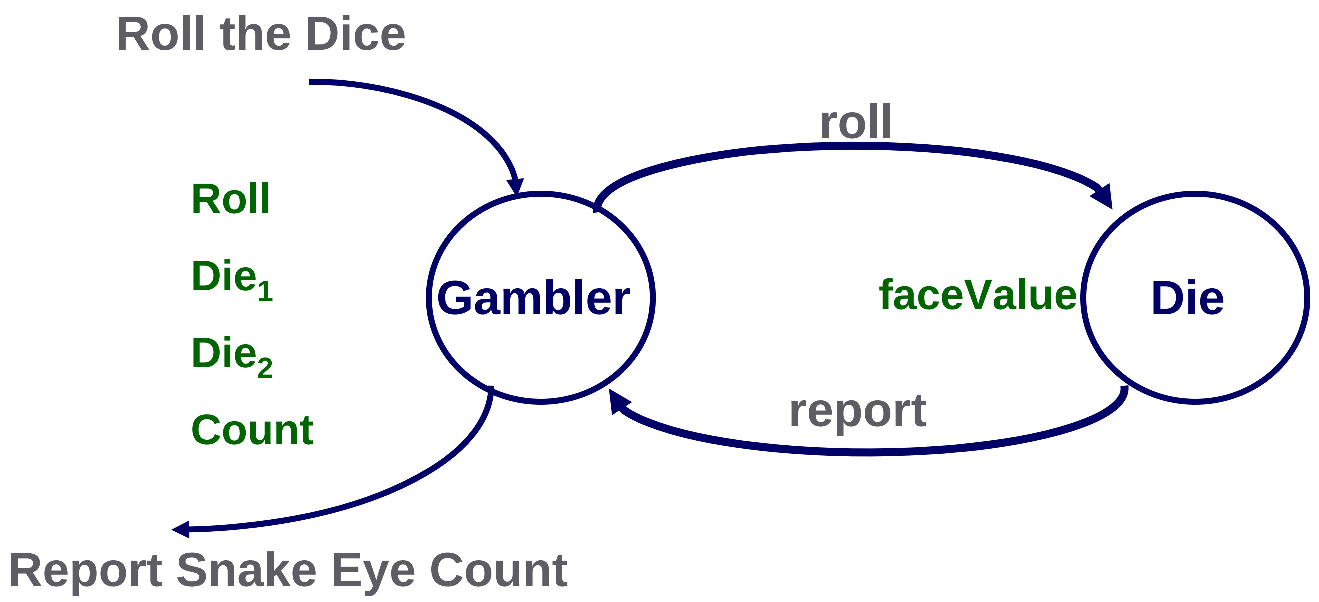


```
ch = obj.calc (25, count, "Hello");

char calc (int num1, int num2, String message)
{
   int sum = num1 + num2;
   char result = message.charAt (sum);
   return result;
}
```

# Be Able to Code a Class

Code a Class given a specification using:

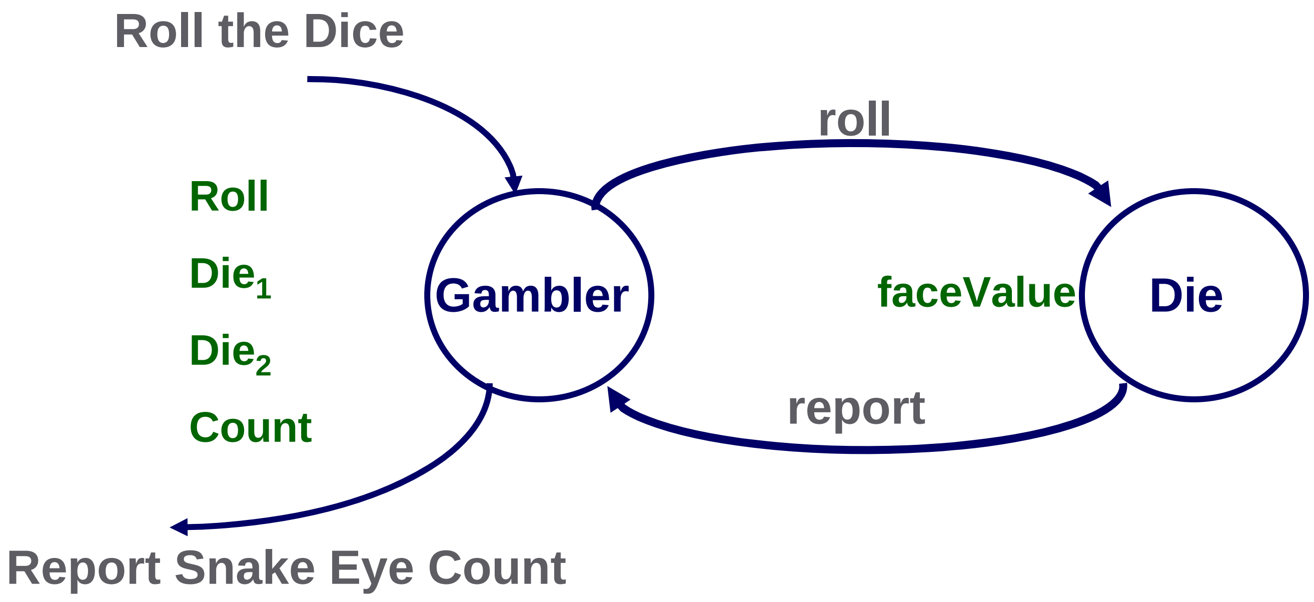
- English
- Data Flow Diagram
- UML Graph



# Declaring the Class, Methods, and Members

```
class Gambler
{
 public:
 Gambler(); // Constructor
 int rollTheDice(int MaxRolls);
 /* Returns Count of # SnakeEyes */

 private:
    int roll, count;
    Die die1, die2;
};
```

```
class Die
{
 public:
   Die();     // Constructor
   int roll();
   /* Returns new faceValue */

 private:
   int faceValue;
};
```



# Fill in the code for the Methods

```
Gambler::Gambler() // Constuctor
{
 roll  = 0;
 count = 0;
}

int Gambler::rollTheDice(int MaxRolls)
{
 /* Returns Count of # SnakeEyes */
 count = 0;
 for (roll=1;roll<=MaxRolls;roll++)
   if ((die1.roll()+die2.roll())==2)
    count++; // SnakeEyes!

 return count;
}
```

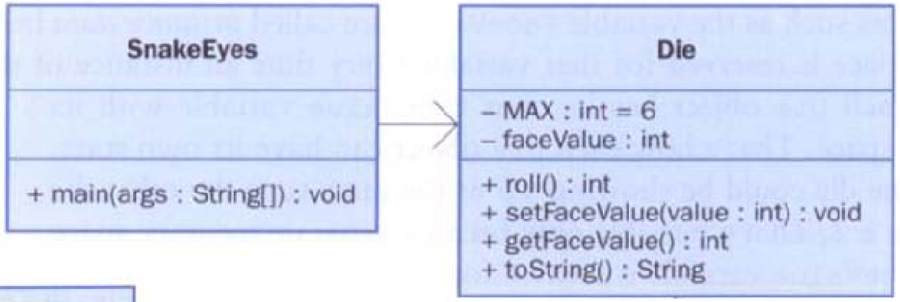
```
Die::Die() // Constructor
 {
  srand(time(0));
  faceValue = (rand()%6)+1;
 }

 int Die::roll()
 {
  // Roll the dice and return faceValue!
  faceValue = (rand()%6)+1;
  return faceValue;
 }
```

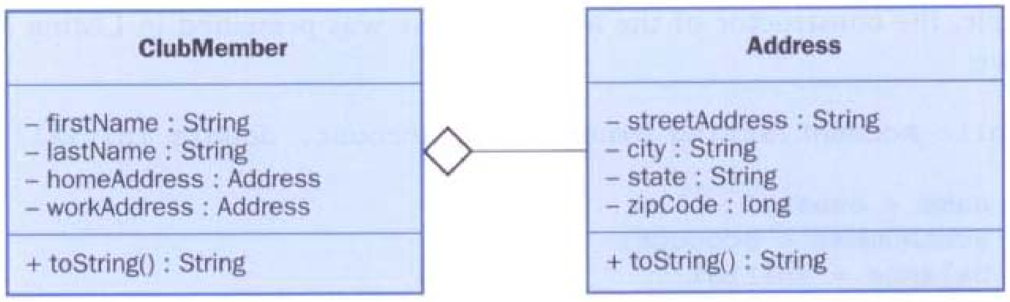
# UML Diagrams and Class Relationships

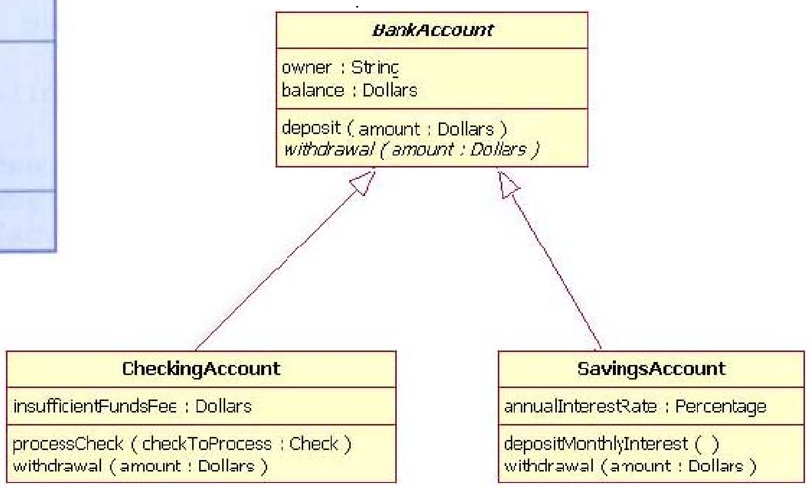
Classes in a software system can have various types of relationships:

Three of the most common relationships:

- Dependency: A *uses* B
- Aggregation: A *has-a* B
- Inheritance : A *is-a* B



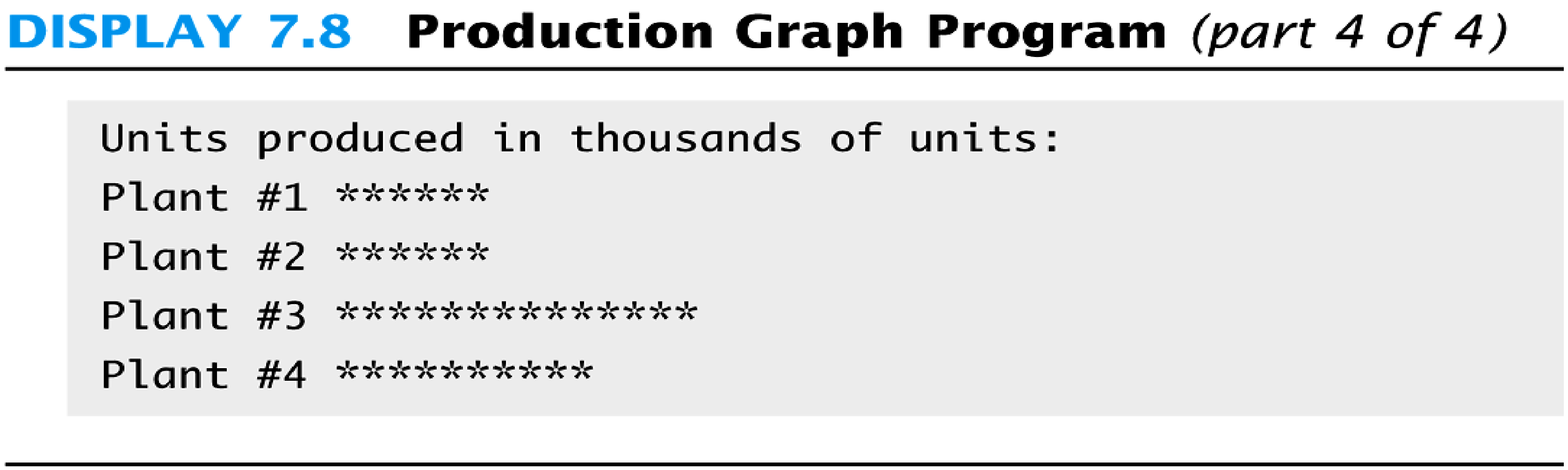



BankAccount
owner : String
balance : Dollars
deposit ( amount : Dollars )
withdrawal ( amount : Dollars )
CheckingAccount
insufficientFundsFee : Dollars
processCheck ( checkToProcess : Check )
withdrawal ( amount : Dollars )
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal ( amount : Dollars )

## Case Study: Production Graph

Problem Definition:

- We are writing a program for the Apex Plastic Spoon Company
- The program will display a bar graph showing the production of each of four plants for a week
- Each plant has separate records for each department
- Input is entered plant by plant
- Output shows one asterisk for each 1000 units, and production is rounded to the nearest 1000 units



**DISPLAY 7.8 Production Graph Program** *(part 4 of 4)*

```
Units produced in thousands of units:
Plant #1 ******
Plant #2 ******
Plant #3 **************
Plant #4 **********
```

## (a) What is the benefit of encapsulation?

Main benefit is reducing system complexity

- Treat data or object as a black box
- hides implementation details
- hides data details
- Reduces inter-module coupling
- Increases intra-module cohesion

## (c) What is the difference between an Object and a Class

A Class is a blue print for an object. It defines a set of variables, methods, and interfaces but has no state and reserves no memory space for those variables.

An Object is an instantiation of a Class and is assigned a new copy of any instance data associated with that class. It has both state and methods that operate on that state.

## (i) Describe what an Abstract Data Type is and why we use them?

An Abstract Data Type (ADT) hides the details about how the data is implemented, so the user can focus on what it can be used for.

An ADT allows the implementation to be changed without impacting users since their interface to the ADT is perserved.

ADTs reduce system complexity by hiding details from users.

## (j) What is Inheritance and what is its key benefit?

Inheritance allows derived classes to be created that re-use the existing methods and data of the base class.

It's key benefit is software re-use. It allows new services to quickly benefit from existing software features and already tested designs.

Re-Use results in more reliable software systems that are produced quicker with less effort.

# Questions?

Questions about the Assignment?

Questions about the Midterm?

## References:

1. Walter Savitch, Problem Solving with C++. Pearson, 9th Edition, 2014, ISBN 978-0-13-359174-3
2. T. DeMarco, *Structured Analysis and System Specification*, 1979, ISBN 978-0-13-8543808
3. T DeMarco, Structured Analysis, Structural Design and Materials Conference 2001, Software Pioneers, Eds.: M. Broy, E. Denert, Springer 2002 http://cs.txstate.edu/~rp31/papersSP/TDMSpringer2002.pdf
4. Stevens, W., G. Meyers, and L. Constantine, *Structured Design*, IBM Systems Journal, Vol 13, No 2. 1974
5. Fairley, Richard E., *Software Engineering Concepts*, McGraw-Hill, 1985, ISBN 0-07-019902-7