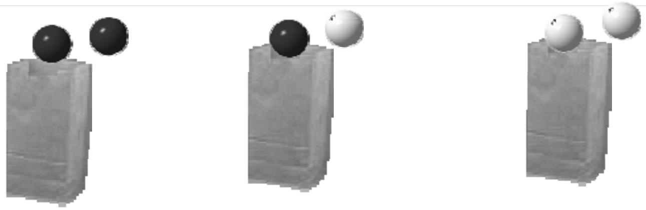
A C++ Program Example: Three Bags

C++ Object Oriented Programming
Pei-yih Ting
NTOU CSE

A Simple Probabilistic Experiment



- Three paper bags, each bag is given two balls with colors shown in the above figure
- We perform the following probabilistic experiment:
  - Step 1: put balls into each bags
  - Step 2: randomly choose a bag
  - Step 3: randomly draw one ball out of the bag
  - Step 4: if the color is red, then take the second ball out of the bag
    otherwise stop the experiment

we want to find out the probability that the **second ball is red** at step 4

蒙提霍爾 (Monty Hall) 問題

決勝 **21** 點

- 米奇：假設你正參加一個遊戲節目，要求你

從**三扇不同的門**裡選一扇，其中一扇門後面
有**一輛新車**，另外兩扇門後面各有**一頭山羊**

挑到什麼帶走什麼, 你要選擇哪一扇門？

- 班： 一號門。(1/3 的機會，隨便挑一扇門)
- 米奇：好！這時節目主持人 (他知道門後的秘密) 去打開另一扇門，比方說三號門，當然後面是一頭山羊。這時節目主持人問，你想要堅持選擇原來的一號門，還是換成二號門？
- 班： **換**，……，當一開始他讓我選一扇門時，我有 1/3的機率是選對的，但當他開其中一扇門時，此刻如果我選擇換一扇門，選對的機率是 2/3，……。

「**三門問題**」最初是美國電視節目 Let’s Make a Deal
中主持人 **Monty Hall** 在節目上玩的一個益智遊戲

一開始選到車子的機會是 1/3, 羊的機會是 2/3, 現在呢?

蒙提霍爾問題 (cont’d)

★ 如果主持人**換個方法說**:

現在製作單位大放送,
二號、三號門合起來算是一個選擇,
如果其中有一扇門後面有車子你就把車子開回家

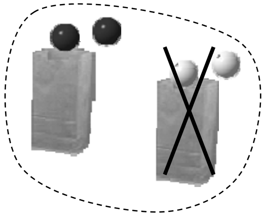
你要**堅持**選一號門還是要**換**二+三號門**?**

堅持的話把車子開回家的機率是 **1/3**, 換的話顯然是 **2/3**

- <u>回到原來的問題</u>, **你挑了一號門**, 主持人把**二+三號門**裡面是羊的那扇門打開, 然後問你要堅持選一號門還是要換? **你說呢?**
- 仔細分析這兩個問題還是有一點點差異, **原本問題裡製作單位多賺到一頭羊XD…**
- 大部分同學不喜歡機率課程, 尤其是不知道為什麼一定要積分積分的作法, **可是機率問題最有趣的就在於腦筋轉一轉有很多直觀的看法, 很多問題也都直接出現在你的日常生活之中**

蒙提霍爾問題 (cont'd)

回到 **3 bags** 問題



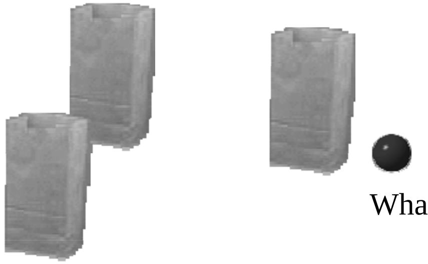
三個袋子任選一個選到不同色球袋子的機率是 **1/3**, 同色球袋子的機率是 **2/3**

袋子裡挑出第一顆球是紅球這件事告訴你: 你挑到的這一袋不可能是兩個白球

在此條件下, 你挑到的這一袋是兩個紅球的機率是 **2/3**

所以 **2/3** 也就是袋子裡剩下那一個球是紅球的機率

A Simple Probabilistic Experiment



Is the remaining ball red or white?

What is the probability of being red again?

$$\Pr\{\text{ 2nd is red} \mid \text{1st is red }\} = \frac{\Pr\{\text{ 1st is red and 2nd is red }\}}{\Pr\{\text{ 1st is red }\}}$$

$$= \frac{\Pr\{\text{ RR bag is picked }\}}{\Pr\{\text{ RR bag picked and 1st ball is red }\} + \Pr\{\text{ RW bag picked and 1st ball is red }\}}$$

$$= \frac{1/3}{1/3 + 1/3 \times 1/2} = 2/3$$

A Program Written in C (1/3)

- Let's try simulating this experiment and calculating the probability by the so called ***Monte Carlo*** method
- Converting the problem specification into C
  - Let's do the experiments 10000 times to estimate the probability → a **for** loop
  - Using a random variable in the range {0, 1, 2} to emulate the random choice of a bag at step 2 → variable **draw1**
  - Using another random variable in the range {0, 1} to emulate the random selection of a ball from the chosen bag at step 3 → variable **draw2**
  - At each run of experiment, keep the count of those experiments with the first selected ball being red → variable **totalCount**
  - At each run of experiment, keep the count of those experiments with both balls being red → variable **redCount**
  - Take the ratio of **redCount** and **totalCount** to be the result

A Program Written in C (2/3)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main()
{
    long i;
    int draw1, draw2, choice, tmp;
    long totalCount=0L,
          redCount=0L;

    srand(time(NULL));
    for (i=0; i<100000L; i++)
    {
       draw1 = rand() % 3; // pick a bag out of the three

       if (draw1 == 0) // (Red, Red)
       {
          totalCount++;
          redCount++;
       }
       else if (draw1 == 1) // (Red, White)
       {
          draw2 = rand() % 2;
          if (draw2 == 0) // the first is Red
             totalCount++;
          else // the first is White
             /* do nothing */;
       }
    }

    printf("Pr(2nd is red | 1st is red)=%lf\n",
      (double)redCount / (double)totalCount);
}
```

Output:
Pr(2nd is red | 1st is red)=**0.665299**

A Program Written in C (3/3)

- Is the conversion process from the problem specification to a C program direct and trivial? Not really
- If you just read the C program alone, can you reconstruct the problem easily and exactly? Not quite easy
- There are many missing pieces of the original problem specification in the above C program.
  - 100000 experiments mixed together (without my explanations, some might have a wrong picture of what the program actually does) Variables totalCount and redCount are something not in the original problem specification.
  - Meaning of variables draw1 and draw2 are a little bit intriguing.
  - There is no bag appearing in the program.
  - No code is associated with the case that the bag with two white balls is selected.

The Same Program Written in C++

- Model the problem *in the **application domain*** (***problem domain***) with minimal transformation to the computer technical domain
- Identify all objects, describe their functionalities and inter-relationships, categorize them, extract common characteristics
  - Experiment (Game)
    - contain three bags
    - random selection of a bag
  - Bag
    - contain zero, one, or two balls
    - random selection of a ball inside
  - Ball
    - color

The Same Program Written in C++

- Characterize the usages of the overall system: these usages would integrate the functionalities of the above designed set of objects (classes) (**Use cases**, **Scenarios**)
  - Perform an experiment: requires the participation of three bags, each bag has two balls with color as specified, select a bag, then select a ball, check its color, if red, check the color of the second ball
  - Perform the above experiment for 100000 times and keep the statistics

bottom-up programming methodology

- Use existing/common OO architecture or components to implement the designed architecture.
- Move on to customized OO programming.

OOA
OOD
OOP

Game Class

```
---------------- 2:Game.h ----------------


#ifndef game_h
#define game_h

#include "Bag.h"

class Game
{
public:
    Bag *getABag();
    Game();
    ~Game();
private:
    Bag *m_bags[3];
};

#endif
```

```
---------------- 3:Game.cpp -----------


#include "Game.h"
#include "Bag.h"
#include <stdlib.h> // rand()

Game::Game()
{
    m_bags[0] = new Bag(0,0);
    m_bags[1] = new Bag(0,1);
    m_bags[2] = new Bag(1,1);
}

Game::~Game()
{
    int i;
    for (i=0; i<3; i++)
        delete m_bags[i];
}

Bag *Game::getABag()
{
    return m_bags[rand()%3];
}
```

Bag Class

```
---------------- 4:Bag.h ----------------


#ifndef BAG_H
#define BAG_H

class Ball;

class Bag
{
public:
    Ball *getABall();
    void putBallsBack();
    Bag(int color1, int color2);
    ~Bag();
private:
    Ball *m_balls[2];
    int m_numberOfBalls;
};

#endif
```

```
---------------- 5:Bag.cpp ----------------


#include "Bag.h"
#include "Ball.h"
#include <stdlib.h> // rand()

Bag::Bag(int color1, int color2)
    : m_numberOfBalls(2)
{
    m_balls[0] = new Ball(color1);
    m_balls[1] = new Ball(color2);
}

Bag::~Bag()
{
    delete m_balls[0];
    delete m_balls[1];
}

```

Bag Class (cont’d)

```
Ball *Bag::getABall()
{
    if (m_numberOfBalls == 0)
        return 0;
    else if (m_numberOfBalls == 1)
    {
        m_numberOfBalls = 0;
        return m_balls[0];
    }
    else
    {
        int iPicked = rand()%2;
        Ball *pickedBall = m_balls[iPicked];
        if (iPicked == 0)
        {
            m_balls[0] = m_balls[1];
            m_balls[1] = pickedBall;
        }
        m_numberOfBalls = 1;
        return pickedBall;
    }
}
```

```

void Bag::putBallsBack()
{
    m_numberOfBalls = 2;
}
```

This design and implementation are problematic. When you get a ball from a bag, the ownership of the ball is better naturally transferred.

Ball Class

```
---------------- 6:Ball.h ----------------


#ifndef BALL_H
#define BALL_H

class Ball
{
public:
    bool IsRed();
    Ball(int color);
private:
    int m_redWhite;
};

#endif
```

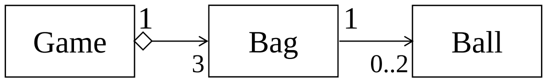
```
---------------- 7:Ball.cpp ----------------


#include "Ball.h"

Ball::Ball(int color)
: m_redWhite(color)
{
}

bool Ball::IsRed()
{
    if (m_redWhite == 0)
        return true;
    else
        return false;
}
```

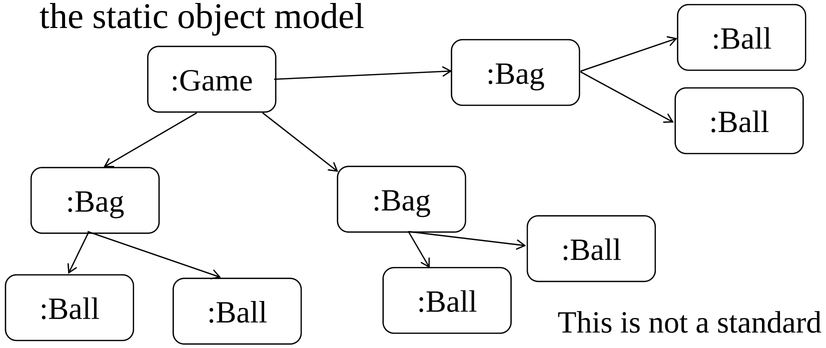
main()

```

------------- 1:main.cpp -------------


#include "Game.h"
#include "Bag.h"
#include "Ball.h"
#include <stdlib.h> // srand()
#include <time.h> // time()
#include <iostream.h>

void main()
{
    int  i;
    Game  theGame;
    Bag  *pickedBag;
    Ball *pickedBall;
    int   totalCount = 0;
    int   secondIsAlsoRed = 0;

    srand(time(0));

    for (i=0; i<100000; i++)
    {
        pickedBag = theGame.getABag();
        pickedBall = pickedBag–>getABall();
        if (pickedBall–>IsRed())
        {
            totalCount++;
            if (pickedBag–>getABall()–>IsRed())
                secondIsAlsoRed++;
        }
        pickedBag–>putBallsBack();
    }

    cout << "The probability that remaining
  ball is red = "
        << ((double)secondIsAlsoRed/totalCount)
  << "\n";
}


```

Some Observations

- Lengthier codes
- More functions
- Slower (maybe)
- There is a clear conceptual architecture for the program: the static object model





This is not a standard graph.

More Observations

- Bottom-up design: some of the functions of an object might not even be used in this particular application. Ex. the Complex class in the lab
- The functions and data of each class/object are self-contained.
- The *data coupling* and *control coupling* between an object and other objects are designed to be minimal. Objects interact with each other through constrained interface functions.
- Software operations mimic the physical functions of the original real world problem.
- The overall program functionalities are provided by a set of cooperating objects.

Even More

- Many consumer products are designed with cooperating parts: e.g.
  - Car: engine, fuel system, wheels, transmission, steeling, bucket seats, …
  - Computer: CPU, MB, RAM, HD, display interface, keyboard/mouse, screen
- ++ Just a little engineering common sense would tell you how to maintain or repair a car/computer when it breaks down – find out which part is not functioning well and replace it with a good one.
- ++ The quality control of manufacturing each part is much easier.
- — The design of such a product with many replaceable parts are not trivial. It certainly increases the design/manufacturing cost and thus the price/competitive capability of the product.
- ++ However, you can see that this is a cost efficient strategy to make a product work for a few years and your customers satisfied.
- Ask yourself a question: Is the technology not good to glue everything together as a whole? to make the product more monolithic, more tasteful, more handy, more style of future

Summary

- There are many OOA / OOD methodologies since ’80s.
- After a major unification of *Jacobson*, *Booch*, and *Rumbaugh* in the ’90s, we have the UML, **Unified Modeling Language** for describing the OO design artifacts and the design process (the methodology) associated with it.
- In this course, we will focus on OOP, especially on how C++ provides features for implementing your OO design.
- We will try to elaborate those OO concepts provided by the implementation language: namely, ***objects***, ***abstraction***, ***interface***, ***encapsulation***, ***inheritance***, ***polymorphism***, ***generic programming*** (the ***templates***), and ***exceptions***.
- You are encouraged to browse the OOA, OOD stuffs.