Brief overview of object-oriented programming

Prof. Michel A. Kinsy & Dr. Christopher Dudley
Sena E. Koehnen & Shreeya Khadka
Dr. Linda Hansen & Rebecca Rodas
Abstraction

• What is abstraction?
  ‣ Answer

• Why is it so important?
  ‣ It is the way we humans manage our complex world
Kings and Queens of Abstraction

• **Humans think in object-Orientated Way**
  ‣ Everything is an object
  ‣ A system is an object composed of other objects
  ‣ The evolution and development of a system are caused by the interactions of the objects inside/outside a system

• **Examples**
  ‣ A book, a student, a teacher
  ‣ A house, a chair, a classroom, a building
  ‣ A dog, the world, shoe
From us to the computer

• Why not make the computer do what we do?
  ‣ Object-Oriented programming is a design approach
  ‣ Objects are the building blocks of a program
  ‣ Objects represent real-world abstractions within an application

• It allows us to communicate complex systems and their interactions more easily with computers
Abstraction Data Type

- Data structures
- A set of operations
- The interface is the only access mechanism to the data structures.
Abstraction Data Type

Data Structures
Operations
Interface

Data Structures
Operations
Interface

Data Structures
Operations
Interface

Data Structures
Operations
Interface

Data Structures
Operations
Interface

Data Structures
Operations
Interface
In Class Exercise

- Let us build our own abstract data type

We want the robot to perform a rectangular movement.

We need to first tell the robot what is a rectangle.

What are the properties of a rectangle:
In Class Exercise

• Let us build our own abstract data structure

```c++
#include <iostream>
using namespace std;

class Robot_Rectangle
{
    private:
        int width;
        int length;
```
In Class Exercise

• Let us build our own abstract data structure

```cpp
#include <iostream>
using namespace std;

class Robot_Rectangle {
    private:
        int width;
        int length;
    
    Make them less visible from the outside
```
Classes in C++

• Let us build our first class

```cpp
#include <iostream>
using namespace std;

class Robot_Rectangle
{
    private:
        int width;
        int length;

    public:
        Robot_Rectangle (void) {width = 0; length = 0;}
        void set(int w, int l){width = w; length = l;}
        int get_width(void) {return width;}
        int get_length(void) {return length;}
        int area(void) { int calculated_area = 0; calculated_area = width*length; return calculated_area; }
};
```
Classes in C++

- Let us build our first class

```cpp
#include <iostream>
using namespace std;

class Robot_Rectangle {
private:
    int width;
    int length;

public:
    Robot_Rectangle (void) {width = 0; length = 0;}
    void set(int w, int l){width = w; length = l;}
    int get_width( void) {return width;}
    int get_length( void) {return length;}
    int area( void) {
        int calculated_area = 0;
        calculated_area = width*length;
        return calculated_area;
    }
};
```
Let us build our first class

```cpp
#include <iostream>
using namespace std;

class Robot_Rectangle
{
    private:
        int width;
        int length;

    public:
        Robot_Rectangle (void) {width = 0; length = 0;}
        void set(int w, int l){width = w; length = l;}
        int get_width(void) {return width;}
        int get_length(void) {return length;}
        int area(void) { int calculated_area = 0; calculated_area = width*length; return calculated_area; }
};
```

Access and modifier interfaces
In Class Exercise

```cpp
#include <iostream>
using namespace std;

class Robot_Rectangle
{
    private:
        int width;
        int length;
    
    public:
        Robot_Rectangle (void) {
            width = 0; length = 0;
        }
        void set(int w, int l){width = w; length = l;}
        int get_width(void) {return width;}
        int get_length(void) {return length;}
        int area(void) {
            int calculated_area = 0;
            calculated_area = width*length;
            return calculated_area;
        }
};

int main (void) {
    Robot_Rectangle Rectangle1;
    Robot_Rectangle Rectangle2;

    Rectangle1.set (3,4);
    Rectangle2.set (5,6);

    cout << "Rectangle1 area: " << Rectangle1.area() << endl;
    cout << "Rectangle2 area: " << Rectangle2.area() << endl;

    return 0;
}
```
Class Inheritance

Student
- name
- id
- major

Junior
- year
- Placement Tests
- etc.

Senior
- AP Classes
- Graduating Exam
- etc...
#include <iostream>
using namespace std;

class Robot_Path {
    protected:
        int width, height;
    public:
        void set_values(int a, int b)
        {
            width = a; height = b;
        }
};

class Rectangle: public Robot_Path {
    public:
        int area()
        {
            return width * height;
        }
};

class Triangle: public Robot_Path {
    public:
        int area()
        {
            return width * height / 2;
        }
};

int main (void) {
    Rectangle Rectangle1;
    Triangle Rectangle2;

    Rectangle1.set (4,5);
    Rectangle2.set (4,5);

    cout << "Rectangle1 area: " <<
    Rectangle1.area() << endl;
    cout << "Rectangle2 area: " <<
    Rectangle2.area() << endl;

    return 0;
}
Data Structures & Information Processing

- Lists
- Stacks
- Queues
- Trees
- Graphs
Lists

• **Arrays**
  - We have already covered them

• **Linked List**
  - A linked list is a series of connected nodes
  - Each node contains at least
    - A piece of data (any type)
    - Pointer to the next node in the list
Stacks

- Stacks are lists
- A stack is a data structure that stores and retrieves items in a last-in-first-out (LIFO) manner
- Operations
  - Push
    - causes a value to be stored in (pushed onto) the stack
  - Pop
    - retrieves and removes a value from the stack
In Class Exercise

class Stack
{
private:
    int array[50];       // type of data in stack
    int n;               // number of elements in stack

public:
    Stack();
    void push(int);      // number of elements in stack
    int pop();
};

// What will happen when we take these actions?
push(5);
push(10);
push(15);

push(5);

Push(10);
Queues

- Queues are lists
- A queue is a data structure that stores and retrieves items in a first-in-first-out (FIFO) manner
- Operations
  - **enqueue**
    - causes a value to be stored in (pushed onto) the queue
  - **dequeue**
    - retrieves and removes a value from the queue
class Queue
{
private:
    int array[50];    // type of data in queue
    int n;            // number of elements in queue

public:
    Queue();
    void enqueue(int);
    int dequeue();
};

// What will happen when we take these actions?
enqueue(5);
enqueue(10);
enqueue(15);
enqueue(5);
enqueue(10);
Trees

- A tree is a data structure that stores and retrieves items in branches
• A graph is a data structure where items are organized in a less structured form
That's all Folks!