SECURITYBOX™

COMPREHENSIVE CYBER SECURITY SERVICES

INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Agenda

1. Introduction:
   a. Why OOP?
   b. Objects and Classes
2. OOP concepts
3. OOP in C++
4. Homework

Agenda

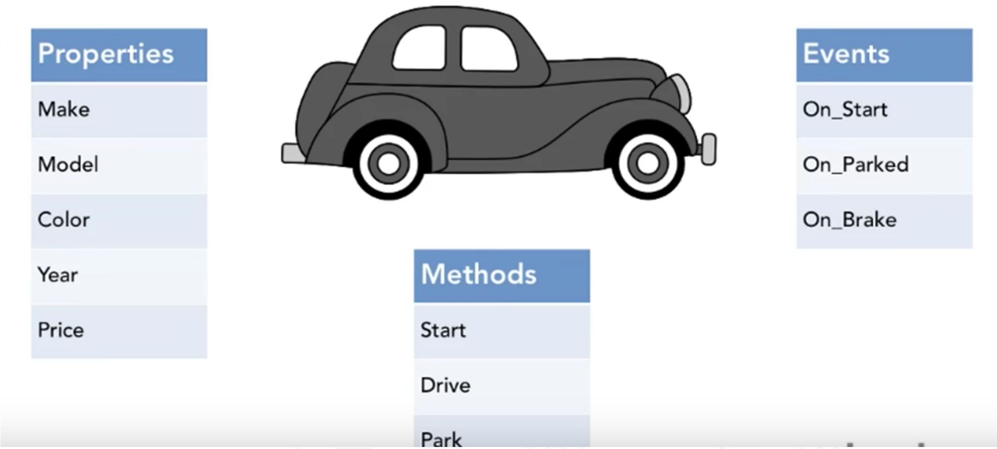
1. Introduction:
   a. **Why OOP?**
   b. Objects and Classes
2. OOP concepts
3. OOP in C++
4. Homework

Procedural Approach

- Focus on procedures
- All data is shared: no protection
- More difficult to modify
- Hard to manage complexity
- Examples: Perl, C, VBScript ...

Object-oriented programming (OOP)

- A programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.



Properties
Make
Model
Color
Year
Price

Events
On_Start
On_Parked
On_Brake

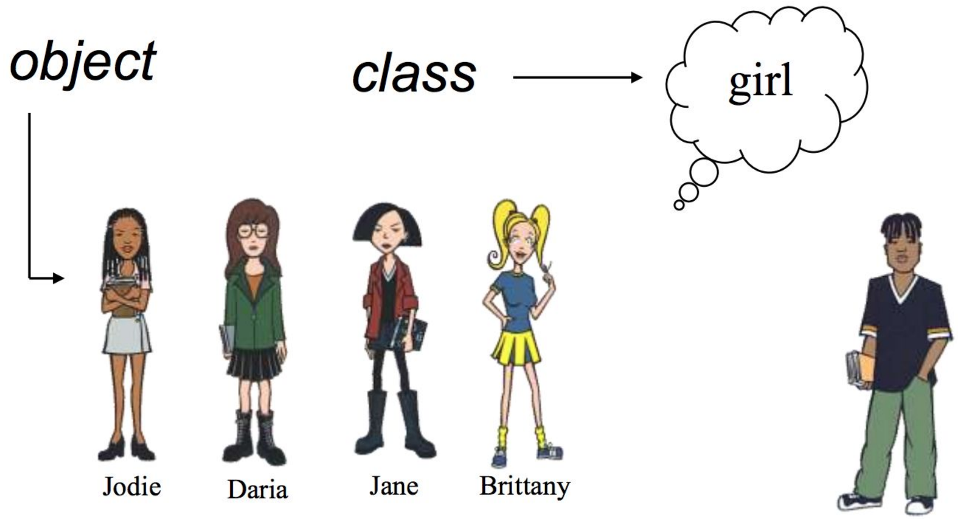
Methods
Start
Drive
Park

Agenda

1. Introduction:
   a. Why OOP?
   b. Classes and Objects
2. OOP concepts
3. OOP in C++
4. Homework

Classes and Objects

- Class: Prototype, idea, and blueprint for creating objects
- Object: instance of class

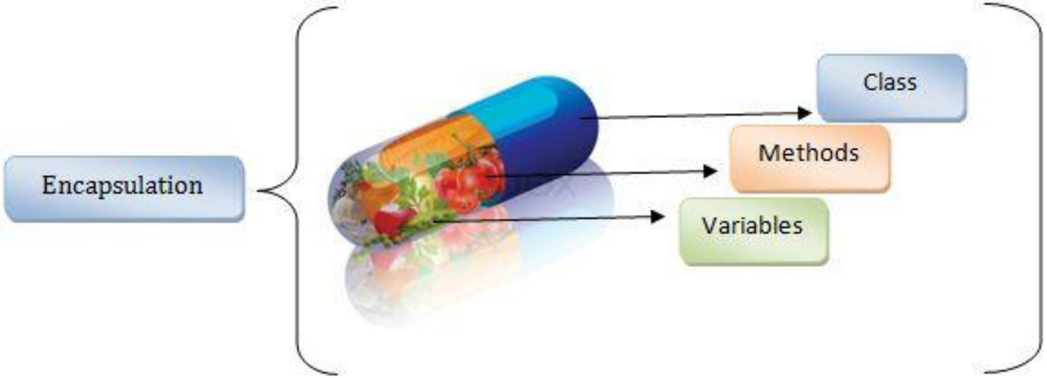


Agenda

1. Introduction:
   a. Why OOP?
   b. Classes and Objects
2. **OOP concepts**
3. OOP in C++
4. Homework

Encapsulation

- Binds together data and functions.
- Enables reusability.
- Hiding and protecting methods and properties from the client classes.



Encapsulation - Benefits

- Ensures that structural changes remain local.
- Hiding implementation details, reduces complexity -> easier maintenance.

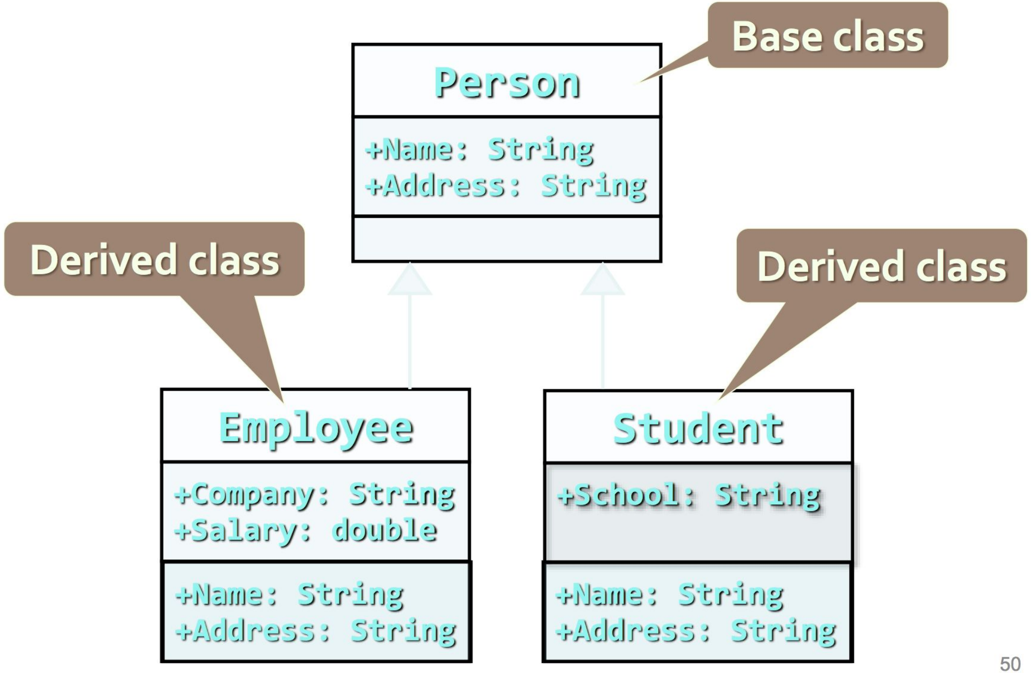
Inheritance

- A way of organizing classes.
- Classes with properties in common can be grouped so that their common properties are only defined once in parent class.
- Superclass – inherit its attributes & methods to the subclass(es).
- Subclass – inherit all its superclass attributes & methods besides having its own unique attributes & methods.

Inheritance - Benefit

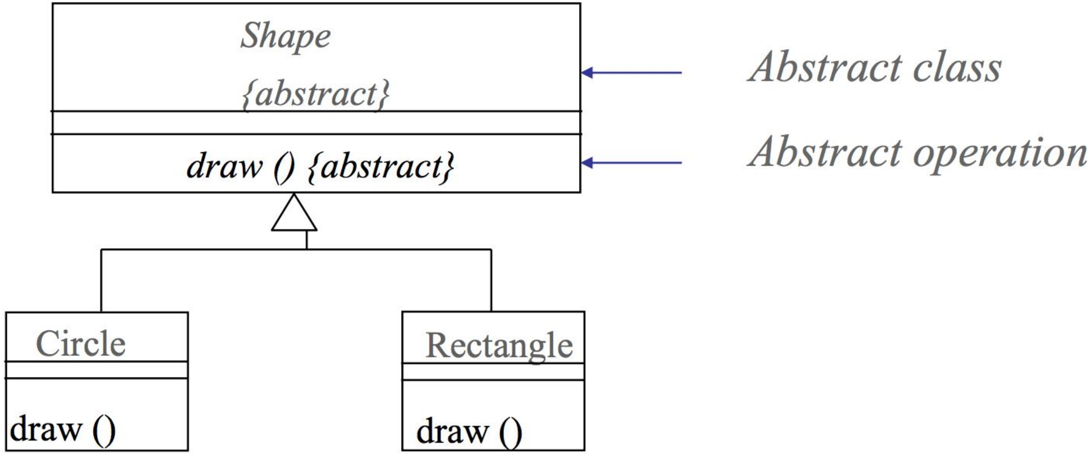
- Expresses commonality among classes/objects.
- Allows code reusability.
- Highlights relationships.
- Helps in code organization.

Inheritance - Example



Abstraction

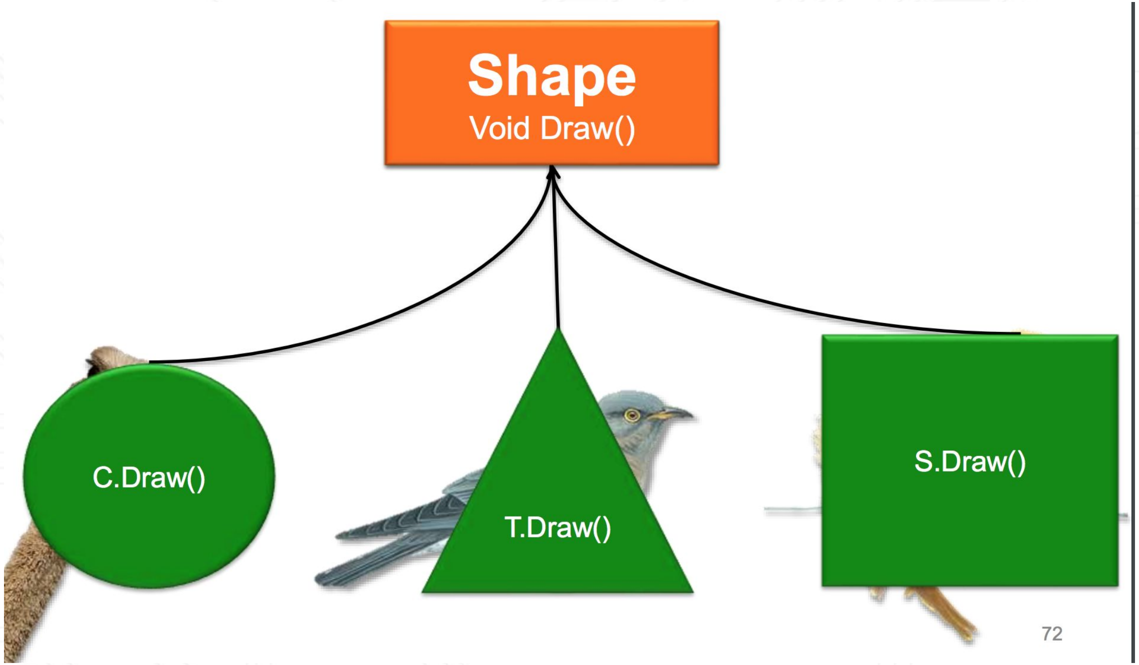
- A design principle.
- Providing only essential information to the outside world and hiding their background details.
- **abstract class** is a class that may not have any direct instances.
- **abstract operation** is an operation that it is incomplete and requires a child to supply an implementation of the operation.



Abstraction vs Encapsulation

Polymorphism

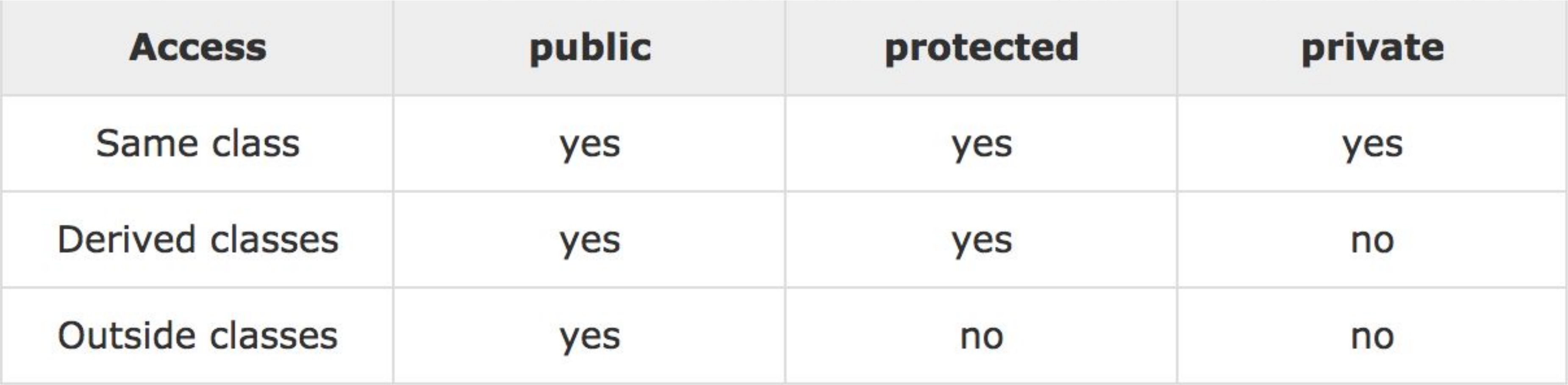
- Ability to request that the same methods be performed by a wide range of different types of things.
- achieved by using many techniques named method overloading, operator overloading, and method overriding.



Agenda

1. Introduction:
    a. Why OOP?
    b. Classes and Objects
2. OOP concepts
3. **OOP in C++**
4. Homework

Access control in C++



Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Class in C++

```
#include <iostream>
#include <string>

class Employee {
public:
    std::string m_name;
    int m_id;
    double m_wage;

    // Print employee information to the screen
    void print()
    {
        std::cout << "Name: " << m_name << "  Id: " << m_id << "  Wage: $" << m_wage << '\n';
    }
};

int main()
{
    // Declare two employees
    Employee alex{ "Alex", 1, 25.00 };
    Employee joe{ "Joe", 2, 22.25 };

    // Print out the employee information
    alex.print();
    joe.print();

    return 0;
}
```

Encapsulation example

```
#include <iostream>
using namespace std;
class Adder {
  public:
    // constructor
    Adder(int i = 0) {
      total = i;
    }
    // interface to outside world
    void addNum(int number) {
      total += number;
    }
    // interface to outside world
    int getTotal() {
      return total;
    };

  private:
    // hidden data from outside world
    int total;
};
int main() {
  Adder a;
  a.addNum(10);
  a.addNum(20);
  a.addNum(30);
  cout << "Total " << a.getTotal() <<endl;
  return 0;
}
```

Inheritance example

```
#include <iostream>
using namespace std;
// Base class
class Shape {
  public:
    void setWidth(int w) {
      width = w;
    }
    void setHeight(int h) {
      height = h;
    }
  protected:
    int width;
    int height;
};
// Derived class
class Rectangle: public Shape {
  public:
    int getArea() {
      return (width * height);
    }
};
int main(void) {
  Rectangle Rect;
  Rect.setWidth(5);
  Rect.setHeight(7);
  // Print the area of the object.
  cout << "Total area: " << Rect.getArea() << endl;
  return 0;
}
```

Abstraction example

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

class implementAbstraction
{
    private:
        int a, b;
    public:
        // method to set values of
        // private members
        void set(int x, int y)
        {
            a = x;
            b = y;
        }
        void display()
        {
            cout<<"a = " <<a << endl;
            cout<<"b = " << b << endl;
        }
};

int main()
{
    implementAbstraction obj;
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

Polymorphism example

```
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
};
class Rectangle: public Polygon {
  public:
    int area()
      { return width*height; }
};
class Triangle: public Polygon {
  public:
    int area()
      { return width*height/2; }
};
int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

Homework Week 3

https://drive.google.com/file/d/1lEwdIDs5Z2asghMpDrYM5L3dnAEbqc_q/view?usp=sharing

References:

1. Leancpp.com
2. https://en.wikipedia.org/wiki/Object-oriented_programming
3. http://www.cplusplus.com/
4. https://www.tutorialspoint.com/cplusplus/cpp_object_oriented.htm

THANK YOU FOR LISTENING