Object Oriented Programming

Programming Languages

Declarative languages (Haskell, ML, Prolog...)

OO languages (C++, Java, Python...)

procedural languages (C, FORTRAN)

assembly languages

Machine languages

Some Features of OOP languages

- An OOP language should support
  - Easy Representation of
    - Real-world objects
    - Their States and Abilities
  - Interaction with objects of same type
  - Relations with objects of other type
  - Polymorphism and Overloading
- Reusability of code
- Convenient type definitions

What are objects?

- Objects model elements of the problem context

Each object has:

- characteristics
- responsibilities (or behaviours)

An Example

Problem Design and build a computer hockey game

Object Hockey player

Characteristics Position, height, weight, salary, number of goals

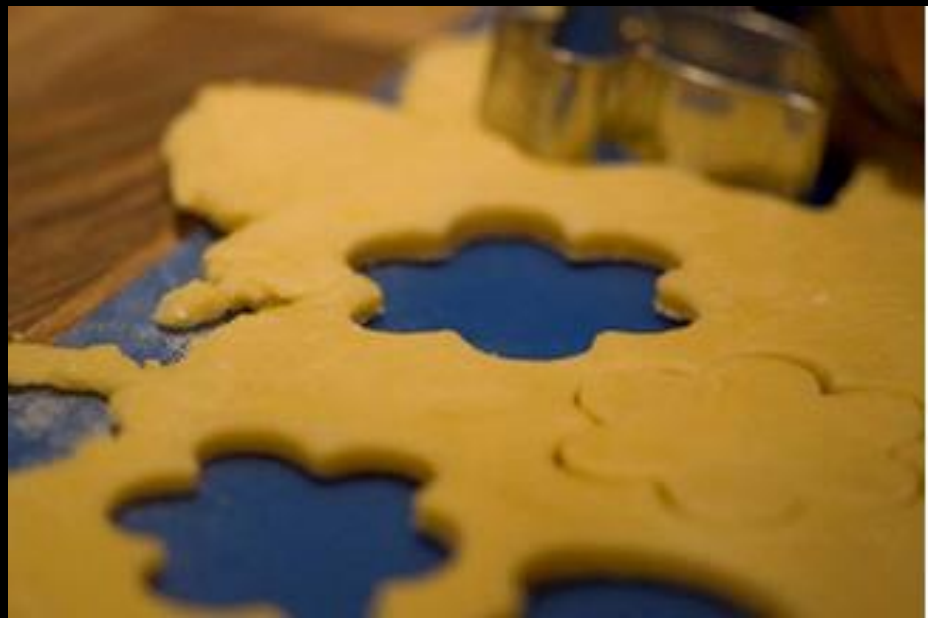
Responsibilities Pass the puck, shoot, skate forward, skate backward, punch another player, etc.

Basic OOP in C++

Classes

A **class** is like a cookie cutter; it defines the shape of objects

Objects are like cookies; they are instances of the class



Often the objects are modeled after real-world entities.

Class and Object

- Object
  - An entity with unique identity that encapsulate state
  - state can be accessed in a controlled way from outside
  - The access is provided by means of methods (procedures that can directly access the internal state)
- Class
  - A specification of objects in an incremental way
  - By inheriting from other classes
  - And specifying how its objects (instances) differ from the objects of the inherited classes

Classes

- **A class definition begins with the keyword *class*.**
- **The body of the class is contained within a set of braces, { } ; (notice the semi-colon).**

```
class class_name
{
        ....
....
....
};
```

Any valid identifier

Class body (data member + methods)

Classes

- Within the body, the keywords *private:* and *public:* specify the access level of the members of the class.
  - the default is private.

- Usually, the data members of a class are declared in the *private:* section of the class and the member functions are in *public:* section.

Classes

```
class class_name
{
    private:
        …
        …
        …
    public:
        …
        …
        …
};
```

private members or methods

Public members or methods

Classes

- Member access specifiers
  - public:
    - can be accessed outside the class directly.
      - The public stuff is *the interface*.
  - private:
    - Accessible only to member functions of class
    - Private members and methods are for internal use only.

```
class Circle
{
    private:
        double radius;
    public:
        void setRadius(double r);
        double getDiameter();
        double getArea();
        double getCircumference();
};
```

No need for others classes to access and retrieve its value directly. The class methods are responsible for that.

They are accessible from outside the class, and they can access the member (radius)

Creating an object of a Class

- Declaring a variable of a class type creates an object. You can have many variables of the same type (class).
  - *Instantiation*
- Once an object of a certain class is instantiated, a new memory location is created for it to store its data members and code
- You can instantiate many objects from a class type.
  - Ex) Circle c; Circle *c;

Implementing class methods

Class implementation: writing the code of class methods.

There are two ways:

1. Member functions defined outside class
   - Using Binary scope resolution operator (`::`)
   - “Ties” member name to class name
   - Uniquely identify functions of particular class
   - Different classes can have member functions with same name

– Format for defining member functions

```
ReturnType ClassName::MemberFunctionName( )
    {

    }
```

Implementing class methods

2. Member functions defined inside class
   - Do not need scope resolution operator, class name;

```
class Circle
{
    private:
        double radius;
    public:
        Circle() { radius = 0.0;}
        Circle(int r);
        void setRadius(double r){radius = r;}
        double getDiameter(){ return radius *2;}
        double getArea();
        double getCircumference();
};
```

Defined inside class

Example

Accessing Class Members

- Operators to access class members
  - Identical to those for **`struct`**s
  - Dot member selection operator (`.`)
    - Object
    - Reference to object
  - Arrow member selection operator (`->`)
    - Pointers

Special Member Functions

- Constructor:
  - Public function member
  - called when a new object is created (instantiated).
  - Initialize data members.
  - Same name as class
  - No return type
  - Several constructors
    - Function overloading

Special Member Functions

```
class Circle
{
    private:
        double radius;
    public:
        Circle();
        Circle(int r);
         void setRadius(double r);
        double getDiameter();
        double getArea();
        double getCircumference();
};
```

Constructor with no argument

Constructor with one argument

```
    public:
        Circle() { radius = 0.0;}
        Circle(int r);
        void setRadius(double r){radius =
r;}
        double getDiameter(){ return radius
*2;}
        double getArea();
        double getCircumferen
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22
}
```

The second constructor is called

Since radius is a private class data member

```
void main()
{
    Circle c1,c2(7);

    cout<<"The area of c1
        <<c1.getArea()<<"\n";

    //c1.raduis = 5;//syntax error
    c1.setRadius(5);

    cout<<"The circumference of c1:"
        << c1.getCircumference()<<"\n";

    cout<<"The Diameter of c2:"
        <<c2.getDiameter()<<"\n";
}
```

```
    public:
        Circle() { radius = 0.0;}
        Circle(int r);
        void setRadius(double r){radius =
r;}
        double getDiameter(){ return radius
*2;}
        double getArea();
        double getCircumference();
};
Circle::Circle(int r)
{
    radius = r;
}
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
```

```
void main()
{
    Circle c(7);
    Circle *cp1 = &c;
    Circle *cp2 = new Circle(7);

    cout<<"The are of cp2:"
              <<cp2->getArea();

}
```

Destructors

- Destructors
  - Special member function
  - Same name as class
    - Preceded with tilde (~)
  - No arguments
  - No return value
  - Cannot be overloaded
  - Before system reclaims object's memory
    - Reuse memory for new objects
    - Mainly used to de-allocate dynamic memory locations

```
void Time::printTime()
}
     cout<<"The time is : ("<<*hour<<":"<<*minute<<":"<<*second<<")"
                >>endl;
{

Time::~Time()
}
       delete hour; delete minute;delete second;
{

void main()
}
       Time *t;
       t= new Time(3,55,54);
       t->printTime();

       t->setHour(7);
       t->setMinute(17);
       t->setSecond(43);

       t->printTime();

       delete t;
{
```

Destructor: used here to de-allocate memory locations

Output:
The time is : (3:55:54)
The time is : (7:17:43)
Press any key to continue

When executed, the destructor is called

Access control: public vs. private

```
class Virus {

 float reproductionRate;    // rate of reproduction, in %
  float resistance;         // resistance against drugs, in %
  static const float defaultReproductionRate = 0.1;

 public:

  Virus(float newResistance);
  Virus(float newReproductionRate, float newResistance);
  Virus* reproduce(float immunity);
  bool survive(float immunity);

};
```

private

public

In general,

- keep member fields as private
- minimize the amount of public parts