

Introduction to Object Oriented Programming (OOP)

Oct. 6, 2008

by

Dr. Lyle N. Long

Distinguished Professor of Aerospace Engineering
The Pennsylvania State University, University Park, PA 16802

<http://www.personal.psu.edu/lnl/>

Outline

- Computer Languages
- Characteristics of OOP:
 - Encapsulation
 - Inheritance
 - Polymorphism
- C++
- Java
- Conclusions

2 L.N.Long

Programming Languages

- C++ will remain the pre-eminent language for very large software projects. Extremely difficult language. Lot of room for errors. Not fully object oriented (has to carry the C legacy).
- Java's importance grew rapidly. Java has many features not available in C++, and does not have some of the problems of C++.
- C will be used for smaller programs or when speed is of paramount importance
- Fortran, Pascal, Cobol, Ada and other languages will be niche markets. They will remain for some time, due to the huge installed base of programs, but new programs will (most likely) be written in C++, C#, or Java.

3 L.N.Long

Programming

- Free {
- C++
 - Java or C#
 - C
 - Fortran
 - Basic
 - Perl
 - Matlab / Mathematica
 - Spreadsheet

Increasing
Complexity
and
Capability



For most small programming jobs, Matlab is great (builtin graphics, linear algebra, math functions,...), for most large projects C++ (or Java) might be best. Fortran is "best" for almost no job.

4 L.N.Long

Key OOP Languages

- C++
- Java
- C# (windows only)
- Ada95
- Smalltalk
- Eiffel
- Simula

5 L.N.Long

Introduction

- C++ is an ANSI standard now (1998)
- <http://www.ansi.org/>
- C++ is built for speed and power. As a programmer you can usually do whatever you want, and the compiler will not second guess you. So you can get into trouble very easily. For example, array indices are often not checked at compile or run-time. So if you want to do:
 - `int x[100];`
 - `x[-10] = 900;`
 - `x[200] = 9;`

the compiler will sometimes let you !! You can easily corrupt computer memory. Likewise with pointers and dynamic memory you can get into trouble (eg memory leaks).

6 L.N.Long

Introduction (cont.)

- The main elements of C++ that are different than Java are:
 - Compilers
 - Pointers
 - Memory handling
 - Structures
 - Classes
 - Operator Overloading
 - Input / Output
 - Libraries
 - Legacy with C

7 L.N.Long

Object Oriented Programming

- OOP allows programmers to more closely model the real world.
- Rapid prototyping. Object-Oriented programs can be built and modified very quickly because it provides the programmer with excellent tools for abstraction.
- OOP produces reusable code. Once objects are built, it is very easy to use them in future applications so you need not ever reinvent the wheel.
- OOP helps programmers work in dynamic environments. Object-Oriented programs can be modified quickly and easily as real-world requirements change.
- OOP lets you build large complex codes out of small manageable pieces of code
- All of this is only true, however, if the code is well written

8 L.N.Long

C++ and Java

- Don't forget that it is very easy to write bad code in any language.
- Writing good code is an art, and requires lots of experience.
- I would actually rather have a really well written code in Fortran or C than a bad C++ code
- Also, C++ gives you the capability to program "with reckless abandon". With pointers, you can sometimes change the value of any memory location in the machine! Even if the OS really needs that data!
- C++ is "better" than Fortran, but it is hard to learn to do it well
- If you are going to use C++ you have to be serious about programming. Java is more forgiving.

9 L.N.Lang

A Few Facts About Java

(from Scott McNealy, CEO Sun)

- Java was announced in 1995 (Fortran was created in 1957)
- 25,000 people attended the JavaOne Conference in 2000
- 2.5 million Java programmers (expect 4 million by 2003)
- 5 million downloads of Java Development Kit (JDK)
- 1.1 million visitors per month at java.sun.com
- 80% of universities teach Java (50% of these have it required)
- 2000 Java Books published
- 50% of programmers use Java

10 L.N.Lang

Java Features

- Object oriented
- Graphical user interfaces
- Remote method invocation
- Graphics
- Serial port drivers
- Fairly machine independent
- Multi-threaded
- Java docs

11 L.N.Lang

Java Features

But wait, C++ can do this with:

- Object oriented ← C++
- Graphical user interfaces ← FLTK
- Remote method invocation ← CORBA or Sockets
- Graphics ← OpenGL
- Serial port drivers ← Open source software
- Fairly machine independent ← Compilers
- Multi-threaded ← POSIX threads
- Java docs ← Doxygen

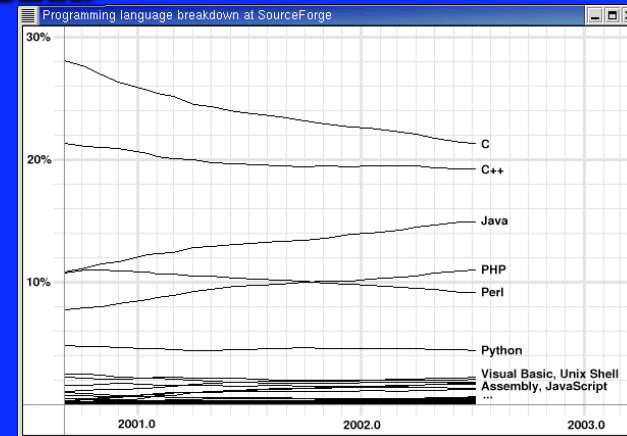
12 L.N.Lang

Java

- Java has definitely been oversold
- C, C++ and Ada all have their place
 - Easy to generate bad C++ code
 - Ada is good, but never caught on (outside of DOD)
- Java suffered early on with performance and scalability problems
- mySQL, javascript, and PHP might be better for many web applications
- But new Java processors and embedded computing apps are well suited to Java
- And it is easier to use Java instead of "C++ + FLTK + CORBA + POSIX + OpenGL + ... "

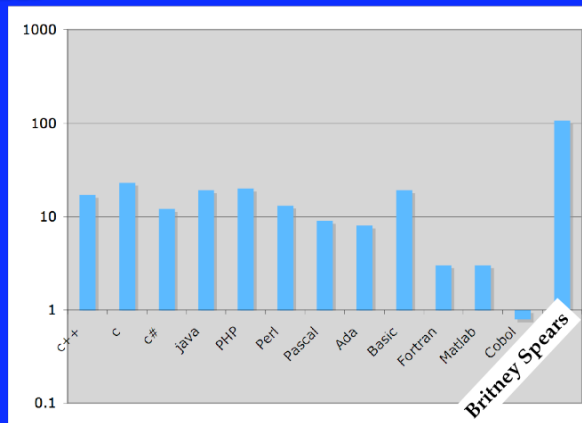
13 L.N.Lang

Measuring language popularity using SourceForge



14 L.N.Lang

Measuring language popularity using Altavista "hits"



15 L.N.Lang

Object Oriented Programming

- OOP: Inheritance, polymorphism, and encapsulation
 - Inheritance: Objects can inherit the properties of other objects. We can create "mammal" objects, and dogs and cats can be new objects that simply inherit the properties of the mammals.
 - Polymorphism: You can write functions that behave differently depending on what kind of data is passed to them. E.g. you could make a sqrt function that would work with integers, floats, or even complex numbers. You could also create a run() method for mammals that would work fine for cats or dogs.
 - Encapsulation: Data and methods (ie functions) are stored together in a "class". Code becomes modifiable, understandable, and portable.

16 L.N.Lang

Encapsulation

- In OOP the data and methods associated with objects are normally hidden from the users
- Objects communicate with each other thru well-defined interfaces
- The details of the code are not available outside the class, the author decides what others can see, modify, and use.
- Variables (data) can be Public, Protected, Private, ...

17 L.N.Long

Inheritance

- Inheritance permits software reusability
- New classes can be created from existing classes
- The attributes (data) and behavior (methods) of the old class are available to the new class
- New data and methods can be added to the new class also
- Inheritance allows you to write key pieces of code once and use it over and over
- This can get kind of confusing, just remember you can write lots of simple little C++ or Java programs without using inheritance and polymorphism (and you will often be using them without knowing it !)

18 L.N.Long

Polymorphism

- “An object’s ability to decide what method to apply to itself, depending on where it is in the inheritance heirarchy, is usually called *polymorphism*. The idea behind polymorphism is that while the message may be the same, objects may respond differently.” (From “Core Java book”)
- This is why I referred to it as *method overloading on steroids*. (i.e. it is more powerful than simply method overloading)
- The key to making polymorphism work is *dynamic binding*. When you apply a method to an object it decides at run time what to do, it is not decided when the code is compiled (*static binding*).

19 L.N.Long

Polymorphism (cont.)

- Polymorphism allows you to avoid very complicated **switch** or **if/else** constructs that are required in other languages to treat special cases
- The same method name can be used in many different ways by different objects, this is especially interesting for inherited objects
- Method overloading
- Ideally we would like to have the power to loop thru our objects as though they were all the same, but have the right data and methods used in all subclasses
- For example, what if I had lots of shapes (circles, squares, ..) and I wanted to animate them on the screen. I can simply loop thru all “Shape” objects and do that ! Without the need for any “switch” statements.

20 L.N.Long

A Few Terms

- **class:**
 - A class is a blueprint for how to make an object
 - **method:**
 - A function (defined in a class)
 - **data**
 - numbers, arrays, objects,... (defined in a class)
- **object:**
 - An instance of a class
- **dot notation:**
 - myCat.size (this could return the size of a cat)
 - myCat.run() (this could make the cat run)
 - System.out.println() (can have more than one dot)
- **Arrays of Objects**
 - myCats[i][j]

21 L.N.Lang

HelloWorld Java Program

```
public class HelloWorld {  
  
    public static void main ( String args[] ){  
  
        System.out.println( "Hello World !");  
  
    }  
  
}
```

Store the above in a file called HelloWorld.java
then do:
javac HelloWorld.java
then do:
java HelloWorld

22 L.N.Lang

C++ Hello World

preprocessor
command

```
#include <iostream>  
using namespace std;  
  
int main(void) {  
  
    cout << "Hello World!" << endl;  
    return 0;  
}
```

23 L.N.Lang

Object Oriented Programming (OOP)

- OOP encapsulates data (e.g. numbers and objects) and methods (functions)
- Information hiding is a key aspect of OOP
- Implementation details are hidden from the user ("we can drive a car without knowing how it is built")
- C and Fortran are "action oriented" (verbs)
- Java is "object oriented" (nouns)
- C++ can be in between....pure C code will compile in a C++ compiler

24 L.N.Lang

Benefits of OOP

- Modifiability
- Readability
- Maintainability
- Reliability
- Reusability
-

the “ilities”

Languages such as Fortran are fine for relatively small programs when you are only interested in number crunching (FORmula TRANslation). But for large programming tasks, it is a clunky old language (without graphics, GUI's, threads, OOP, and networking, ...).

25 L.N.Long

OOP Class

```
public class Molecule {
    public double u, v, w ;
    public double maxVelocity () {
        double max;
        max = Math.max ( u, v );
        max = Math.max ( max, w ) ;
        return max;
    }
}
```

To create an instance of this class (an object), do:
Molecule x = new Molecule();

To use “u” you could do:
newu = x.u

To use the method:
vmax = x.maxVelocity()

26 L.N.Long

C++ Class Example

```
class Molecule {
private:
    double x,y,z;           // molecule location
    double u,v,w;          // molecule velocity
    char * name;           // molecule name (e.g. H2O)
    int atoms;             // number of atoms
    static double dt;      // simulation time step
    static int total;      // number of molecules

public:
    void move() {
        x = x + dt * u;
        y = y + dt * v;
        z = z + dt * w;
    }
};
```

optional. these will be private even if the keyword private is not there.

a common mistake is to omit the semicolon

27 L.N.Long

Private and Public

- We usually use “private” class variables, which means only class methods can use them (private variables are recommended, otherwise encapsulation is broken)
- If you use public variables, then users can simply do:
 - myObject.x = 2.0;
- this is bad. We use set and get methods instead.
 - myObject.setX(2.0);
- It can seem like a lot of additional work, but will help you make better code, and it will give you more freedom to modify the code in the future

28 L.N.Long

Get and Set Methods

Instead of giving people access to your class data, you should supply get and set methods.

```
class Molecule {
    private:
        double u,v,w;
    public:
        void setU ( double Veloc ) { u = Veloc;}
        double getU () {return u;}
        void setV ( double Veloc ) { v = Veloc;}
        double getV () {return v;}
        void setW ( double Veloc ) { w = Veloc;}
        double getW () {return w;}
}
```

29 L.N.Lang

Constructors and Destructors

- To use C++ classes, you also need to understand “constructors” and “destructors”
- Java has constructors, but not destructors
- Constructors are simply methods that tell C++ how to create instances of objects, and these can be overloaded as in Java
- Destructors are needed because memory management is left to the programmer in C++. A destructor can be used to delete the memory associated with an object that is being deleted. If your objects don't use dynamic memory (ie pointers) then you don't have to worry about this too much.
- If you do not provide a constructor or destructor, the system will use the default ones. (which do little or nothing) The data values will have random values (not zeroes and NULL's as Java does)

30 L.N.Lang

Constructors

- Methods with the same name as the class are “constructors”
- There are often several different constructors (i.e. they can be over-loaded)
- You don't have to have constructors, but they are usually a good idea. A default constructor is automatically included.
- If you have any constructors, then there is no default constructor
- Constructors cannot return a value
- If the arguments in a “new” statement do not match any constructor this is an error
- If a default constructor is used, then you can't be sure how the data is initialized.
- Null Pointer Exception errors are often due to errors in constructors

31 L.N.Lang

OVERLOADED CONSTRUCTORS

```
class Molecule {
    private:
        double x[3];
        double u[3];
    public:
        Molecule (){
            x = { 0.0, 0.0, 0.0 };
            u = { 0.0, 0.0, 0.0 };
        }
        Molecule (double xx, double yy, double zz){
            x[0]= xx; x[1] = yy; x[2] = zz;
            u = { 0.0, 0.0, 0.0 };
        }
        ...
}
```

32 L.N.Lang

Class Destructor Example

```
class Molecule {
private:
    double x,y,z;           // molecule location
    double u,v,w;           // molecule velocity
    char * name;            // molecule name (e.g. H2O)
    int atoms;              // number of atoms
    static double dt;
    static int total;

public:
    void move();            // this is now a prototype
    Molecule();            // prototype for constructor
    Molecule(double x, double y, double z); //prototype
    ~Molecule();           // prototype for destructor
};
```

Notice the tilde ~

33 L.N.Long

Class Destructor Example

```
inline Molecule::~Molecule() {
    total = total - 1;
    delete name;
}
```

This lets you keep an accurate count of how many molecules you have.

Since "name" is a pointer to a char or an array of char's (ie a string), you need to make sure you remove that memory if you delete a Molecule object. If you do not do this, then you will have a memory leak.

If you write a destructor, then you usually should also write a copy constructor and an assignment operator.

destructors can take no arguments and can return no values.

34 L.N.Long

METHODS

- Functions. You have already seen some:
 - System.out.println (" Hello ");
 - Math.sqrt (12.0);
 - g.drawString (" Hello ", 20, 20);
- Methods are key pieces of Objects (the other is data) that you can create
- Divide and conquer !! Break your program up into well-defined and easy to debug pieces ! This also makes it easier to reuse software.

35 L.N.Long

Method Overloading

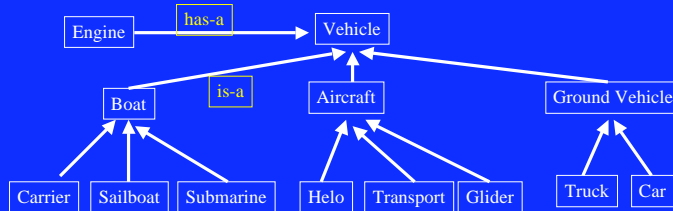
- Java does not allow operator overloading (C++ does), but it does allow method overloading
- For example:

```
public class MethodExample extends JApplet {
    public void init () {
        put code here....
    }
    public double square ( double y ) {
        return y * y; }
    public int square ( int y ) {
        return y * y; }
}
```

You can now call the method "square" and pass it an int or a double and it will use the correct version of the method. It will not work if you pass it a float.

36 L.N.Long

"Has-a" and "Is-a" in OOP



Vehicle "has-a" engine (containment)
Aircraft "is-a" vehicle (inheritance)

Don't confuse "containment" with "inheritance"

37 L.N.Long

"Has-a" and "Is-a"

- We could define a Molecule class such as:

```

class Atom {
private:
    double x, y, z, xold, yold, zold, xnew, ynew, znew; // position
    double u, v, w; // velocity
    double ax, ay, az; // acceleration
    double mass, invmass; // particle mass & 1/mass
    static double timestep; // global time step
    static double totalNumber; // total number of particles
    char * name; // string for name

public:
    ....
}

class Molecule {
private:
    Atom * atoms;

public:
    ....
}
  
```

So, for example, we could have a water molecule made up of two oxygen and one hydrogen atoms. Since atoms is a pointer, we could dynamically assign the atoms to each molecule.

When it appears in the data, we would say:
Molecule "has-a" Atom

38 L.N.Long

"Has-a" and "Is-a"

- We could also define a Particle class such as:

```

class Particle {
protected:
    double x, y, z, xold, yold, zold, xnew, ynew, znew; // position
    double u, v, w; // velocity
    double ax, ay, az; // acceleration
    double mass, invmass; // particle mass & 1/mass
    static double timestep; // global time step
    static double totalNumber; // total number of particles
    char * name; // string for name

public:
    ....
}

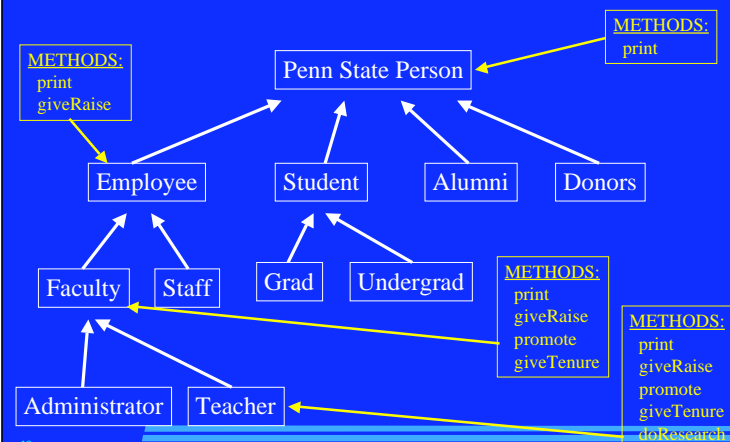
class Atom : public Particle {
}
  
```

Inheritance!

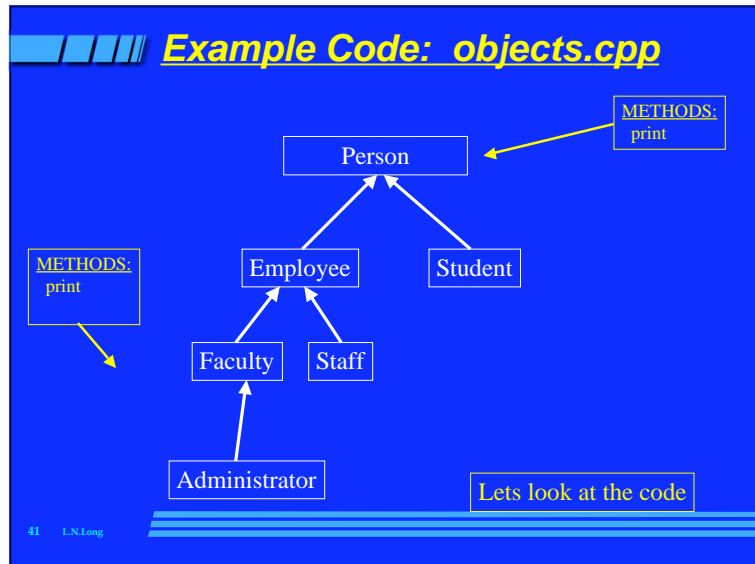
C++ Inheritance. We would say:
Atom "is-a" Particle

39 L.N.Long

Example



40 L.N.Long



Using Polymorphism

```

int main() {
  int i;
  Student nilay(25,66.0,"nilay",13);
  Faculty lyle(48,68.0,"lyle",15000.0,"Professor");
  Staff tom(35,69.0,"tom",10000.0,200);
  Administrator graham(53,69.0,"graham",20000.0,"Professor","President");

  Person * people;

  people[0] = new Student;
  people[0] = &nilay;
  people[1] = new Faculty;
  people[1] = &lyle;
  people[2] = new Staff;
  people[2] = &tom;
  people[3] = new Administrator;
  people[3] = &graham;

  for ( i = 0 ; i < 4 ; i++ ) {
    people[i]->print();
    cout << endl;
  }
}
  
```

In C++ the print() method will need to be defined as "virtual."

This will call the proper print method NOT the "Person" print method

very powerful, we now have an array full of all kinds of object objects, but yet we can treat them as though they were all similar!

42 L.N.Lang

- ## Private, Protected, and Public
- If you were really paying attention, you will have noticed the keyword "protected" used for data members (instead of private)
 - Private:
 - Data or methods are not visible to instances of objects of this class
 - Data or methods are visible to member functions of the class (eg Get and Set methods)
 - Protected:
 - Data or methods are visible to this class and to all classes derived from this class
 - Public:
 - If a function has an object of a certain class, then it can access all the public data or methods of that class
- 43 L.N.Lang

- ## C++ Keywords
- Storage:

| | | | | | | | |
|----------|----------|-------|---------|-------|--------|--------|----------|
| char | short | int | long | float | double | bool | signed |
| unsigned | struct | union | class | enum | auto | static | register |
| extern | volatile | const | typedef | void | sizeof | | |
 - Control:

| | | | | | | |
|-------|-------|---------|----------|------|-----|------|
| if | while | switch | continue | else | for | case |
| break | goto | default | return | | do | |
 - Other:

| | | | | | |
|-------------|--------|----------|----------|--------|---------|
| • true | false | asm | friend | public | private |
| • protected | try | catch | throw | new | delete |
| • virtual | inline | operator | template | this | |
 - More:
 - auto, register, volatile, sizeof, asm, virtual, template
- 44 L.N.Lang

Java Keywords

| | | | |
|----------|------------|-----------|--------------|
| abstract | else | Interface | super |
| boolean | extends | long | switch |
| break | false | native | synchronized |
| byte | final | new | this |
| case | finally | null | throw |
| catch | float | package | throws |
| char | for | private | transient |
| class | if | protected | true |
| continue | implements | public | try |
| default | import | return | void |
| do | instanceof | short | volatile |
| double | int | static | while |

45 L.N.Lang

ARRAYS OF OBJECTS

- If we have a class such as:

```
public Person {
    int age;
    double height;
    String job;
}
```

- We can create an array of these via:

```
Person[] group = new Person[20];
```

With just this you will get a "null pointer exception"

- But the above is not enough to allow us to use this array, we need to initialize this array, maybe via:

```
for ( int i = 0 ; i < 20 ; i++ )
    group[i] = new Person();
```

A way to remember this is that this one has () which indicates a method (a constructor) is used.

46 L.N.Lang

CONCLUSIONS

- Object oriented programming is extremely common now
- There are hundreds of books on OOP
- If done well, it can lead to much better code in a shorter period of time
- As discussed in Core Java, companies like Dell and Compaq became very successful by buying large quantities of PC components and building PC's. They do not build all the pieces. Likewise when you write a Java program you ought to use well-built pieces that exist.
- Since the programmer has to manage all the dynamic memory in C++, using objects can be quite a bit more challenging
- But keep in mind that you do not have to use all the features of C++ right away, you can slowly start using dynamic memory in your codes (and in some cases you may not need it....Fortran did without it for forty years...)

47 L.N.Lang

References

- "C++ and Object-Oriented Numeric Computing," D. Yang
- "Core C++," by Shtern, Prentice Hall
- "Practical C++ Programming," by Oualline, O'Reilly Press
- "C++ How to Program," by Deitel and Deitel, Prentice Hall
- "C++ Gotchas," by Dewhurst

- "Java for Scientists and Engineers", 2nd Edition, by S. Chapman
- "Core Java," Volume I, by Horstmann and Cornell, Prentice Hall
- "Java How to Program," by Deitel and Deitel, Prentice Hall
- "Core Web Programming," by Hall, Prentice Hall
- java.sun.com

48 L.N.Lang



lnl@psu.edu

<http://www.personal.psu.edu/lnl/>