# ECE 462
# Object-Oriented Programming using C++ and Java

## Lecture 10

**Yung-Hsiang Lu**

**yunglu@purdue.edu**

# Lab 5: Comparing the Performance of C++ and Java Container Classes

**measure the time**

- C++ and Java (linked) lists of strings
  - push_back 1 million strings
  - push_front 1 million strings
- C++ and Java set of Student objects
  - insert 1 million distinct objects
  - insert 1 million duplicate objects

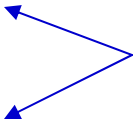
# Container Classes & Methods

- Some container classes (such as set) need special methods.
- C++ set needs
  - operator == to determine whether a new element is already in the set
  - operator < to order the elements in the set
- Java Set needs
  - class to implement Comparable interface
  - override compareTo method

# Self Test
## (Do Java containers copy objects?)

```
import java.io.*;
import java.util.*;
// similar to VectorForClassType.cc
class X
{
    private int x_val;
    public X(int val)
            { x_val = val; }
    public int getVal()
            { return x_val; }
    public void setVal(int val)
            { x_val = val; }
    public String toString()
    {
        String rtv = "val = " + x_val;
        return rtv;
    }
}
```

```
class VectorForClass {
    public static void main( String[] args )
    {
        Vector<X> XVec = new Vector<X>();
        X x1 = new X(2);
        X x2 = new X(3);
        X x3 = new X(5);
        XVec.addElement(x1);
        XVec.addElement(x2);
        XVec.addElement(x3);
        System.out.println(XVec);
        x2.setVal(1000);
        System.out.println(XVec);
    }
}
```



**same output?**

# Self Test

```
#include <iostream>
#include <vector>
using namespace std;
int callcount = 0;
class X {
  int x_val;
public:
  X(int val = 74)
  {
    x_val = val;
    cout << "X(int) " << x_val << endl;
    callcount ++;
  }
```

```
X (const X & xin)
{
  x_val = xin.x_val;
  cout << "X(const X&) " << x_val <<
    endl;
  callcount += 2;
}
X & operator = (const X & xin)
{
  if ( this != &xin )
    { x_val = xin.x_val; }
  cout << "operator = (const X&) " <<
    x_val << endl;
  callcount += 3;
  return *this;
}
```

```
  int getVal(void) const { return x_val; }
  friend ostream & operator << (ostream &
      os, const X & xin);
};
ostream & operator << (ostream & os,
      const X & xin)
{
  os << xin.x_val;
  return os;
}
int sum(vector<X> v )
{
  cout << "\nvector sum is: ";
  int sum = 0;
  vector<X>::iterator p = v.begin();
  while ( p != v.end() )
    {
      sum += p->getVal();
      p ++;
    }
  cout << sum << endl;
  return sum;
}
void print( vector<X> v )
{
  cout << "\nvector size is: " << v.size() <<
      endl;
  vector<X>::iterator p = v.begin();
  while ( p != v.end() )
    {
      cout << (*p) << "  ";
      p ++;
      cout << " ";
    }
  cout << endl;
  sum(v);
}
```

?

```
int main()
{
  vector<X> vec;
  X x1( 2 );
  X x2( 3 );
  X x3( 5 );
  vec.push_back( x1 );
  vec.push_back( x2 );
  vec.push_back( x3 );
  vec[2] = x1;
  print( vec );
  cout << "callcount = " << callcount << endl;
  return 0;
}
```

**?**

# How to Improve the Speed?
## (avoid copying objects)

```
void print(vector<X> & );     // object reference, chapter 9
void print(vector<X> & v ) {
  cout << "\nvector size is: " << v.size() << endl;
  vector<X>::iterator p = v.begin();
  while ( p != v.end() )
    cout << (*p++).getp() << "  ";
  cout << endl << endl;
}
```

# Java IO Stream

- Java has a wide selection of IO streams, for different data types, buffered or not, whether the file is readable from a text editor, and file size.
- example: 98
  - 4-byte integer (hexadecimal): 00 00 00 62
  - 62
  - ASCII: 39 38
- Java IO streams do not provide buffering, unless BufferOutputStream is used.
- Java streams can be combined.

# File IO for Class

```
import java.io.*;
class ClassWithFileIO
{
   private int c_iVal;
   private double c_dVal;
   private String c_sVal;
   public ClassWithFileIO(String fileName)
     throws Exception
   {
      DataInputStream dis =
         new DataInputStream(
         new FileInputStream(fileName));
      c_iVal = dis.readInt();
      c_dVal = dis.readDouble();
      c_sVal = dis.readUTF();
      dis.close();
      System.out.println("c_iVal = " +
     c_iVal +
                    "\nc_dVal = " + c_dVal +
                    "\nc_sVal = " + c_sVal);
}
public ClassWithFileIO(int ival, double
  dval, String sval)
{
   c_iVal = ival;
   c_dVal = dval;
   c_sVal = sval;
}
```

```
    public void writeFile(String fileName)
      throws Exception
    {
        DataOutputStream dos =
            new DataOutputStream(
                new FileOutputStream(fileName)
      );
        dos.writeInt(c_iVal);
        dos.writeDouble(c_dVal);
        dos.writeUTF(c_sVal);
        dos.close();
    }
}
```

```
class WriteReadClass {
    public static void main( String[] args )
      throws Exception
    {
        ClassWithFileIO cfio1 =
            new ClassWithFileIO(-15, 62.347,
      "Purdue Boiler");
        cfio1.writeFile("classdata");
        ClassWithFileIO cfio2 =
            new ClassWithFileIO("classdata");
    }
}
```

# ECE 462
# Object-Oriented Programming using C++ and Java

## Lecture 11

**Yung-Hsiang Lu**

**yunglu@purdue.edu**

# C++ IO Stream

- open file for output:

  ofstream fileout(“filename”);

  fileout << 123 << “a string” << endl;

  ofstream fileout(“filename”, **ios::app**); // append mode

- open file for input:

  ifstream filein(“filename”);

  int x;

  filein >> x;

- open file for both input and output:

  fstream finout(“filename”, **ios::in | ios::out**);

**ios::binary** for binary file

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
class ClassWithFileIO
{
private:
  int c_iVal;
  double c_dVal;
  string c_sVal;
public:
  ClassWithFileIO(string fileName)
  {
    ifstream fin(fileName.c_str(),
      ios::binary);
    fin >> c_iVal;
    fin >> c_dVal;
    fin >> c_sVal;
    fin.close();
    cout << "c_iVal = " << c_iVal
         << "\nc_dVal = " << c_dVal
         << "\nc_sVal = " << c_sVal << endl;
  }
  void writeFile(string fileName)
  {
    ofstream fout(fileName.c_str(),
      ios::binary);
    fout << c_iVal;
    fout << c_dVal;
    fout << c_sVal;
    fout.close();
  }
  ClassWithFileIO(int ival, double dval,
     string sval)
  {
    c_iVal = ival;
    c_dVal = dval;
    c_sVal = sval;
```

```
  }
};

int main(int argc, char * argv[])
{
  ClassWithFileIO cfio1(-217, 64.35,
      "Purdue Boiler");
  cfio1.writeFile("classdata");
  ClassWithFileIO cfio2("classdata");
  return 0;
}
```

# Declaration, Definition, Initialization

- declaration: inform the compiler the existence of an identifier (variable name, function name, class name ...)
- definition: for variable, memory is allocated; for function, code is written
- initialization: assign (default) value to variable
- You should **always** initialize all variables and objects. Uninitialized values are a common source for errors.
- Never leave anything uninitialized.

# Declaration and Definition

```
int xvar; // definition
int xvar = 4; // definition and initialization
extern int xvar;   // declaration only
                                  // defined somewhere else
class Student;    // declaration
class Student {  // definition
    int s_id;
    int s_year;
};
double func(int, float); // declaration
double func(int xval, float yval)    // definition
{ return (exp(yval) * xval); }
```

# Declaration before Definition

```
class Student;    // declaration
class Teacher {
    list<Student*> t_stud;          // list<Student> causes error in C++
};
class Student {
    int s_id;
};
```

C ++ initialization:

- primitive types: default value
- objects: if **no-arg constructor** provided

# Java and C++

| Java | C++ |
|---|---|
| int x; // declare | declare and define |
| int x = 6; // declare, define, and // initialize to 6 | same |
| AClass anobj; // declare // null | declare, define, and initialize using **no-arg constructor** |
| syntax error | AClass * objptr; // declared |
| AClass anobj = new AClass(...) // declare, define, initialized | AClass * objptr = new AClass(...); // same |
| unnecessary | class AClass; // declaration |

```
import java.io.*;
class ClassA
{
    private ClassB a_bobj;
    private int a_int;
    public ClassA()
    {
        System.out.println("ClassA()");
        a_bobj = new ClassB();
        a_int = 462;
    }
    public String toString()
    {
        String str = "a_int = " + a_int +
            " " + a_bobj.toString();
        return str;
    }
}
```

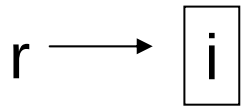
```
class ClassB
{
    private int b_int;
    public ClassB()
    {
        System.out.println("ClassB()");
        b_int = 2007;
    }
    public String toString()
    {
        String str = "b_int = " + b_int;
        return str;
    }
}
class ClassDeclare {
    public static void main( String[] args )
    {
        ClassA aobj;
```

```
        System.out.println("aobj declared but
      not defined");
        /*
        int x;
        System.out.println(x);
        */
        aobj = new ClassA();
        System.out.println(aobj);
    }
}
```

# Lab 6: Order of Object Initialization in C++ and Java

# Reference (alias) in C++

- Reference is alias.




  - int i = 2;
  - int & r = i;
  - r = 3; ⇒ i is 3 now.
- Reference is similar to a pointer but reference, once assigned, cannot be reassigned.
  - r ++; ⇒ i is 4 now. // different from pointer arithmetics
  - int j = 100;
  - r = j; ⇒ **i** is 100

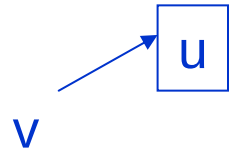
# Parameter Passing in C++ and Java

| Java | C++ |
|---|---|
| pass by value, primitive types (int, char, double ...)<br>func(int x) { x = new_value; }<br>// change not seen by caller | same<br>func(int x) { x = 0; }<br>x = 6;<br>func(x);<br>x is still 6 |
| none | pass by pointer, primitive type<br>func(int * x) { *x = new_value; }<br>// change seen by caller |
| none | pass by reference, primitive type<br>func(int & x) { x = new_value; }<br>// change seen by caller |

| Java | C++ |
| --- | --- |
| same syntax<br>but change seen by caller<br>the behavior, however, is similar to “pointer” | pass by value, object<br>func(AClass obj)<br>{ obj.att = new_value; }<br>// object copied before passing<br>// change not seen by caller |
| none | pass by pointer, object<br>func(AClass * obj)<br>{ obj->att = new_value; }<br>// change seen by caller |
| none | pass by reference, object<br>func(AClass & obj)<br>{ obj.att = new_value; }<br>// change seen by caller<br>// reference = alias |

# Pass Object in Java
## (same object inside the callee)

```
//PassClassTypeByValue.java
class User {
    String name;     int age;
    User( String nam, int yy ) { name = nam; age = yy; }
};
class Test {
    public static void main( String[] args )   {
        User u = new User( "Xeno", 89 );
        g(u);
        System.out.println( u.name + "  " + u.age );    // Yuki 200 (changed)
    }
    static void g( User v ) {  v.name = "Yuki";  v.age = 200; }
}
```



```
//PassClassTypeByValue.cc
#include <iostream>
#include <string>
using namespace std;
class User {
public:
    string name;
    int age;
    User( string nam, int yy )
    { name = nam; age = yy; }
};
void g( User v ) {  v.name = "Yukon";      v.age = 200;  }
int main()
{
    User u( "Xenon", 89 );
    g(u);
    cout << u.name << "  " << u.age << endl;  // Xenon 89
    return 0;
}
```

```
//Swap.java
class User {
    String name;     int age;
    User(String nm, int a) {name=nm; age=a;}
}
class Test {
    public static void main(String[] args)
    {
        User u1 = new User("Xeno", 95);
        User u2 = new User("Yuki", 98);
        swap( u1, u2 );
        System.out.println( u1.name );     // Xeno (not changed)
        System.out.println( u2.name );     // Yuki
    }
    static void swap(User s, User t) { User temp = s;  s = t;   t = temp;    }
}
```

s ⟶ u1

t ⟶ u2

temp ⟶ s ⟶ u1

s ⟶ t ⟶ u2

t ⟶ temp ⟶ u1

```
String temps = s.name;
   s.name = t.name;
   t.name = temps;    // handle age in the same way
```

```
//Swap.cc
#include <iostream>
#include <string>
using namespace std;
class User {
public:
    string name;     int age;
    User(string nm, int a) {name=nm; age=a;}
};
void swap(User& s, User& t) {
    User temp = s;
    s = t;
    t = temp;
}
```

C++ pass-by-reference different

```
int main()
{
    User u1("Xeno", 95);
    User u2("Yuki", 98);
    swap( u1, u2 );
    cout << u1.name  << endl;                // Yuki (changed)
    cout << u2.name  << endl;                // Xeno
    return 0;
}
```

```
//CopyOnReturn.cc (modified)
#include <iostream>
#include <string>
using namespace std;
class User {
public:
  string name;                int age;
  User( string nam, int yy ) { name = nam; age = yy; }
  User(const User & orig) {          // copy constructor
    cout << "copy constructor " << orig.name << endl;
    name = orig.name; age = orig.age; }
};
User f( User usr ) { cout << "before return" << endl; return usr; }
```

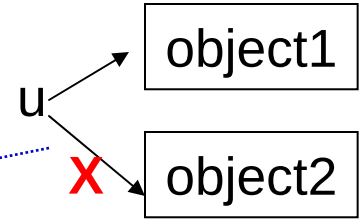
```
int main()
{
    User u( "Xino", 120 );
    User y = f( u );
    cout << y.name << endl;          // Xino
    return 0;
}
```

```
copy constructor Xino (copy u to usr in f)
before return
copy constructor Xino (copy the return object to y)
Xino
```

```
import java.io.*;
class User {
    private String name;
    private int age;
    public User( String nam, int yy ) { name = nam; age = yy; }
    public String toString()
    {
        String str = name + " " + age;
        return str;
    }
}
class ReturnObject
{
    public static void main(String [] args)
    {
        User u1 = new User("David", 36);
        User u2 = f(u1);
```

```
        System.out.println(u1);
        System.out.println(u2);
    }
    public static User f(User usr)
    {
        return usr;
    }
}
```

# Constant Parameters




Java:

```
    g (final int x) { x = 100; } // error
    f (final User u) { u = new User(); }  // error
    h (final User u) { u.name = "John"; } // acceptable
    // in Java, parameters are equivalent to "pointers" in C++
```

C++:

```
    g (const User & u) { /* u cannot be changed */ }
    f (const User & u) { u.name = "John"; } // error
   h (...) const { /* h cannot change any attribute */ }
      /* if a function's parameter is const, the function can
         only call functions whose parameters are also const */
      /* if a function's parameter is not const, the function can call
         another function whose parameter is const */
```