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

Lecture 24

Yung-Hsiang Lu

yunglu@purdue.edu

const in C++

Use const **as much as possible** to protect attributes from accidentally being modified.

```
func(const int x) { x = 3; } // error, cannot change the parameter
class X {
private:
    int val;
public:
    void foo() const {
        val = 3; // error, cannot change any attribute
    }
}
```

```
class X {
private:
    int val;
public:
    X & operator = (const X & orig) {
            orig.val = 3; // error, cannot change the parameter's attribute
    }
    bool operator == (const X & second) const {
            val = 5; // error
            second.val = 7; // error
            return (val == second.val);
   }
}
```

Self Test

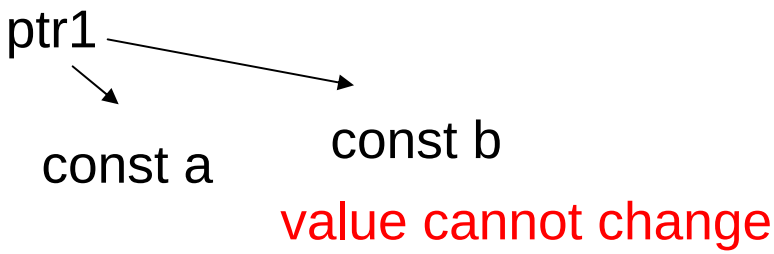
```
int a = 5;
const int * ptr1 = &a;
int const * ptr3 = &a;
```

```
int b = 9;
int * const ptr2 = &a;
```

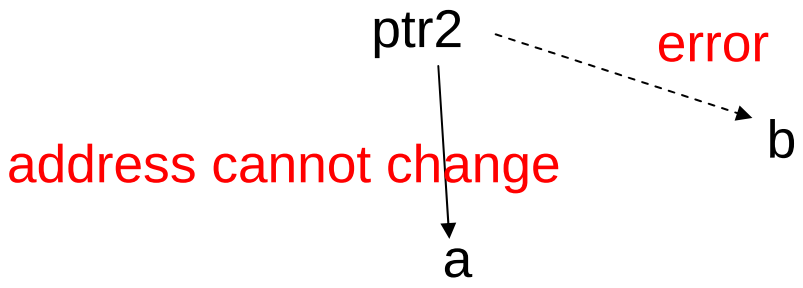
which one is valid?

```
ptr1 = &b;
ptr2 = &b;
ptr3 = &b;
*ptr1 = 17;
*ptr2 = 17;
*ptr3 = 17;
```

```
const int * ptr1 = & a;      // pointer to a constant integer
* ptr1 = 17;                 // error, the value (a) is a constant
ptr1 = & b;                  // no problem, ptr points to another constant
```




```
int const * ptr2 = &a;       // pointer to a constant location
ptr2 = &b;                   // error
*ptr2 = 17;                  // no problem, change a's value to 17
```



const function calls const

```
func(const Type x) {
      func2(x);       // func2 must be func2(const Type' x)
      // here Type' is a type compatible with Type, for example, base
      // class of Type
}

class X {
public:
     void foo(...) const {
            bar();
     }
     void bar() const { // bar must have const
     }
}
```

```
class X {
public:
    void foo(int x) {
        bar(x);          // no problem
    }
    void bar(const int x) const {
    }
}
```

```
class X {
public:
    void foo(const ...) const {
            ...
    }
}

class Y: public X {
public:
    void foo(const ...) const  { // need const?
            ...
    }
}
```

```
#include <iostream>
using namespace std;
void func(const int x) {
  // x = 3;
}
class X {
private:
  int xval;
public:
  X() {
    xval = 28;
  }
  void func1() {
    // notice that func1() does not actually change any attribute
  }
```

```
  void func2() const {
    // xval = 5; // error
    // func1();
  }
  void func3(const int * ptr) {
    // *ptr = 10;
  }
  void func4() {
    const int * ptr1 = & xval;
    int * const ptr2 = & xval;
    func3(ptr1);
    func3(ptr2);
  }
  void func5(const int x) const {
  }
  void func6(const int x) const {
    func5(x);
  }
```

```
  X & operator = (const X & orig) {
    // orig.xval = 3; // error, cannot change the parameter's attribute
    return * this;
  }
  bool operator == (const X & second) const {
    // xval = 5; // error
    // second.xval = 7; // error
    return (xval == second.xval);
  }
};
class Y: public X {
public:
  void func5(const int x) const {
  }
  void func6(const int x) const {
    func5(x);
  }
};
```

```
int main(int argc, char * argv[]) {
  int a = 5;
  int b = 9;
  const int c = 11;
  const int * ptr1 = &a;
  int * const ptr2 = &a;
  int const * ptr3 = &a;
  const int * const ptr4 = &c;
  ptr1 = &b;
  // ptr2 = &b;
  ptr3 = &b;
  // ptr4 = &b;
  // *ptr1 = 17;
  *ptr2 = 17;
  // *ptr3 = 17;
  // *ptr4 = 17;
```

```
  cout << a << " " << b << endl;
  cout << (* ptr1) << " " << (* ptr2) << " "
       << (* ptr3) << " " << (* ptr4) << endl;
  a = 26;
  b = 35;
  cout << a << " " << b << endl;
  cout << (* ptr1) << " " << (* ptr2) << " "
       << (* ptr3) << " " << (* ptr4) << endl;
  return 0;
}
```

```
#include <iostream>
using namespace std;
class MyString {
private:
  char * mstr;
public:
  MyString(const char *st) {
    mstr = new char[strlen(st) + 1];
    strcpy(mstr, st);
  }
  char & operator[] (unsigned int index) { // char &, not char
    cout << "char & operator[] (int index)" << endl;
    if (index < strlen(mstr)) {
      return mstr[index];
    } else {
      // should be an error, but don't worry about that now
```

```
    return mstr[0];
   }
  }
  const char & operator[] (unsigned int index) const {
   cout << "const char & operator[] (unsigned int index) const"
     << endl;
   if (index < strlen(mstr)) {
    return mstr[index];
   } else {
    return mstr[0];
   }
  }
  friend ostream & operator << (ostream & os, const MyString & ms);
};
ostream & operator << (ostream & os, const MyString & ms) {
  os << ms.mstr;
  return os;
}
```

```
int main(int argc, char * argv[]) {
  MyString s1 = "Hello";
  const MyString s2 = "World";
  cout << s1[2] << endl;
  cout << s2[2] << endl;
  s1[2] = 'x'; // error, if return char, not char &
  // s2[2] = 'y';
  cout << s1 << endl;
  cout << s2 << endl;
  return 0;
}
```

final in Java

- create constants, similar to the usage in C++

```
final int x;
x = 15; // can assign the first time
x = 22; // cannot assign again
```

- create global constants (usually as static members)

```
Math.E          // natural logarithm
Math.PI         // pi
Color.WHITE
```

- disallow modifying input parameters

```
public void func2(final int v) {
    v = -2;     // error
```

final ≠ const for objects

- Modifying the attributes of an object is allowed

```
class class1 {
   public int val;
...
class class2 {
   public void func5(final class1 obj1) {
       obj1 = new class1(); // error
       obj1.val = 91; // allowed
```

final ≠ const for function call

- Calling another function without final is allowed.

```
class class2 {
public void func1(int v) {
    v = 11;
}
public void func2(final int v) {
    v = -2; // error
    func1(v); // allowed
}
```

final ≠ const for protecting attributes

- Adding "final" after () is not allowed.

```
class class2 {
    public void func1(int v) final { // error
```

- Adding "final" before a function is allowed. It means the function cannot be overridden in a derived class.

```
class class2 {
final public void func3() {
...
class class3  extends class2 {
public void func3() { // error
```

final to prevent inheritance

```
final class class3  {
}
class class4 extends class3 { // error
}
```

```
// different usages of final in Java
class class1 {
    public static final double PI = 3.14159;
    public int val;
    public class1() {
     val = 3;
    }
    public String toString () {
     return String.valueOf(val);
    }
}
class class2 {
    public class2 () { }
    public void func1(int v) {
     v = 11;
    }
```

```
public void func2(final int v) {
 System.out.println(v);
 // v = -2; // error
 func1(v); // this cannot change the value of v
 System.out.println(v);
}
// final
public void func3() {
 // final ==> does not allow overriding
 // be aware that this disables polymorphism
 // you'd better know what you are doing
}
public void func4(class1 obj1)
 // final, cannot put final after ()
{
 obj1.val = 5;
}
```

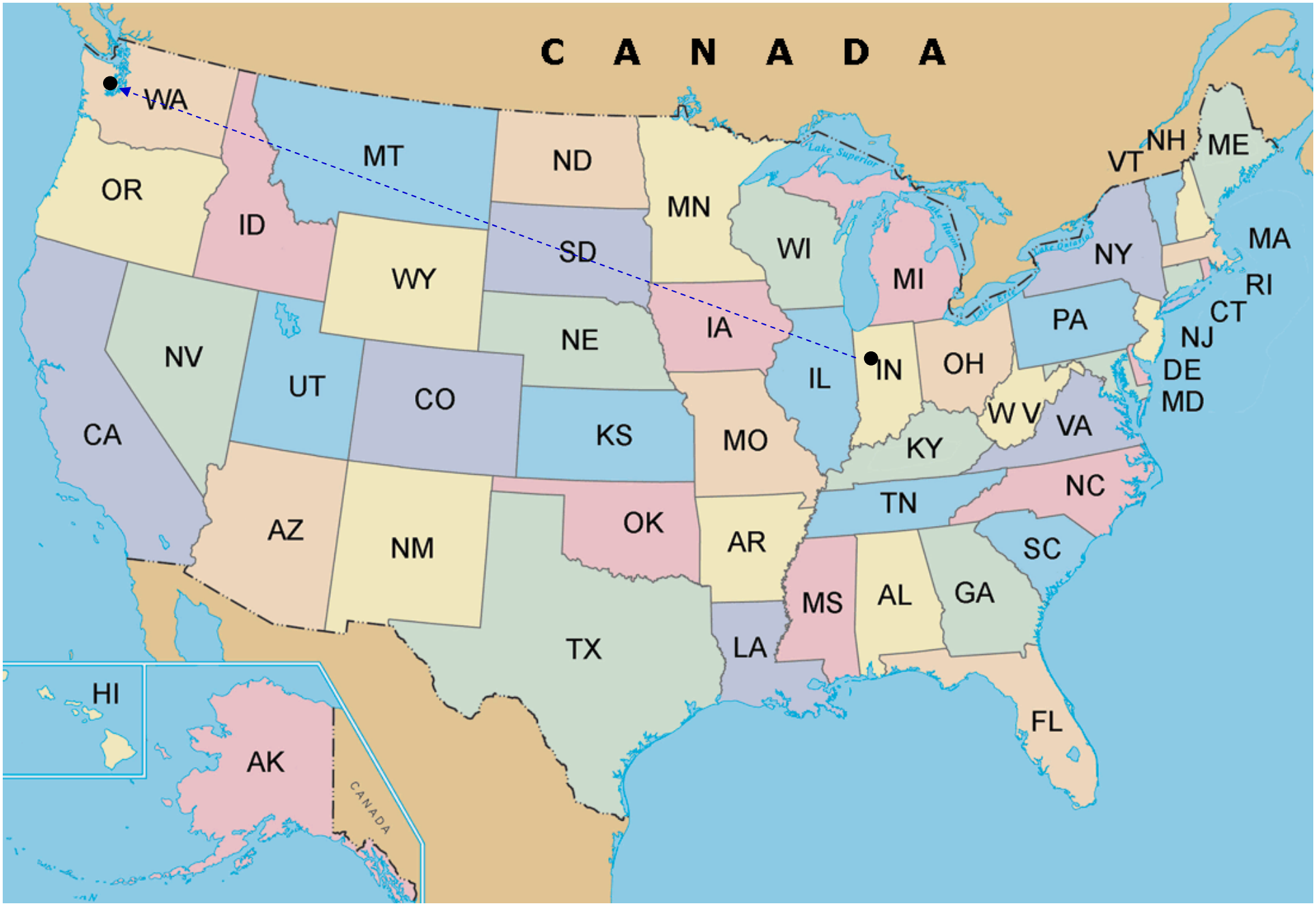
```
    public void func5(final class1 obj1) {
      obj1 = new class1(); // error
      obj1.val = 91; // allowed
      func4(obj1);
      obj1 = new class1(); // error
    }
}
// final
// why don't you allow derived classes?
// you must have a good reason
class class3  extends class2 {
    // final ==> does not allow derived class
    public void func3() {
      // final ==> does not allow overriding
    }
}
```

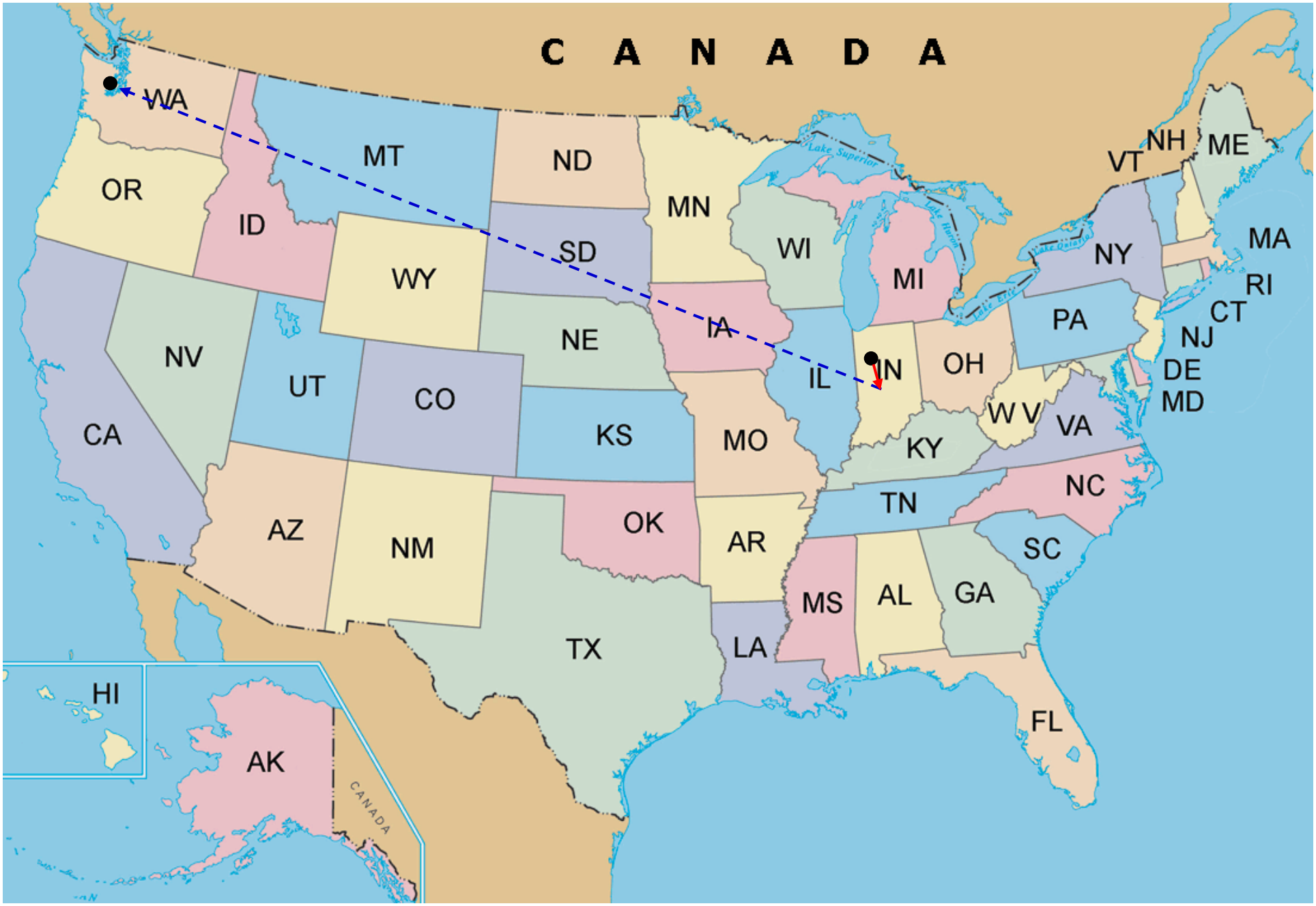
```
class class4 extends class3 {
}

public class finalusage {
    public static void main( String[] args ) {
     class1 obj1 = new class1();
     class2 obj2 = new class2();
     System.out.println(obj1);
     obj2.func5(obj1);
     System.out.println(obj1);
     System.out.println(class1.PI);
     final int x;
     x = 15; // can assign the first time
     // x = 22; // cannot assign again
    }
}
```

Observations from PA1

- Good programming styles produce good code.
- Good code is easier to write and debug.
- Learn tools and save time.
- Do not write code that needs to be "fixed later". You will **never** fix it. You waste a lot more time by buggy code.
- "Quick and dirty" code is always dirty and never quick.
- Think carefully and write code carefully. Make sure it works before doing something else.
- Use version control.
- Fix known problems first.


C A N A D A
WA
OR
CA
ID
NV
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MS
MI
IN
KY
TN
AL
OH
W V
GA
FL
SC
NC
VA
PA
NY
VT
NH
ME
MA
RI
CT
NJ
DE
MD
HI
AK
CANADA
Lake Superior
Lake Huron
Lake Erie
Lake Ontario


C A N A D A
WA
OR
CA
ID
NV
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MS
MI
IN
KY
TN
AL
OH
W V
GA
FL
SC
NC
VA
PA
NY
VT
NH
ME
MA
RI
CT
NJ
DE
MD
HI
AK
CANADA
Lake Superior
Lake Huron
Lake Erie
Lake Ontario

Programming Rules

- A program that is not meant to be improved cannot be debugged.
- Avoid "magic numbers". Never write code with constants. Always create symbolic constants.

```
for (int x = 0; x < 60; x ++) {
```

⇒

```
const int numBrick = 60;          // C++
final int numBrick = 60;          // Java
for (int x = 0; x < numBrick; x ++) {
```

```
const int numRow = 6;
const int numCol = 10;
const int numBrick = numRow * numCol;
```

Don't Write too Much Code

- If you copy-paste code and make a slight change, you are probably doing something wrong.

```
if (slider.getValue() < 10) {
        delay = 1;
} else if (slider.getValue() < 20) {
        delay = 2;
} else if (slider.getValue() < 30) {
        delay = 3;
```

$\Rightarrow$

```
delay = 1 + slider.getValue() / 10;
```

Logic

- Before you do a lot of testing, read your code carefully can save you a lot of time.

```
if (x < 0 && x > 300)
```

⇒

```
if ( (x < 0) || (x > 300))
```

⇒

```
if ( (x < 0) || (x > pgWidth))
```

if ... else if ...

- Before you write if ... else if ..., ask yourself whether the conditions include all possible cases. If so, add the final else.

```
if (x > 100) {
        ...
} else if (x > 200) {
        ...
} // what happens if x is smaller than 100?
```

Always Use { }

- When you have a condition, **always** use {}. Never try to save the few keystrokes.

```
long t1 = getCurrentTime();
while (getCurrentTime() < t1 + delay)
updateBall();
```

⇒

```
long t1 = getCurrentTime();
while (getCurrentTime() < t1 + delay)  {
}
updateBall();
```

Autoformat Code

- Most code development tools can automatically format code. Use it. You can easily find where you have mismatch { - }.
- In Netbeans, click Source- Reformat Code

Attribute vs. Local Variable

```
class PlayGround {
    Ball ballobj;
    public PlayGround() {
        ballobj = new Ball();
    }
    public void paintComponent
        (Graphics gfx) {
        ballobj.update();
    }
}
```

```
class PlayGround {
    public PlayGround() {
        Ball ballobj = new Ball();
        // local
    }
    public void paintComponent
        (Graphics gfx) {
        ballobj.update();
        // ballobj does not exist
    }
}
```

Attribute vs. Local Variable

```
class PlayGround {
    Ball ballobj;
    public PlayGround() {
        ballobj = new Ball();
    }
    public void paintComponent
        (Graphics gfx) {
        ballobj.update();
    }
}
```

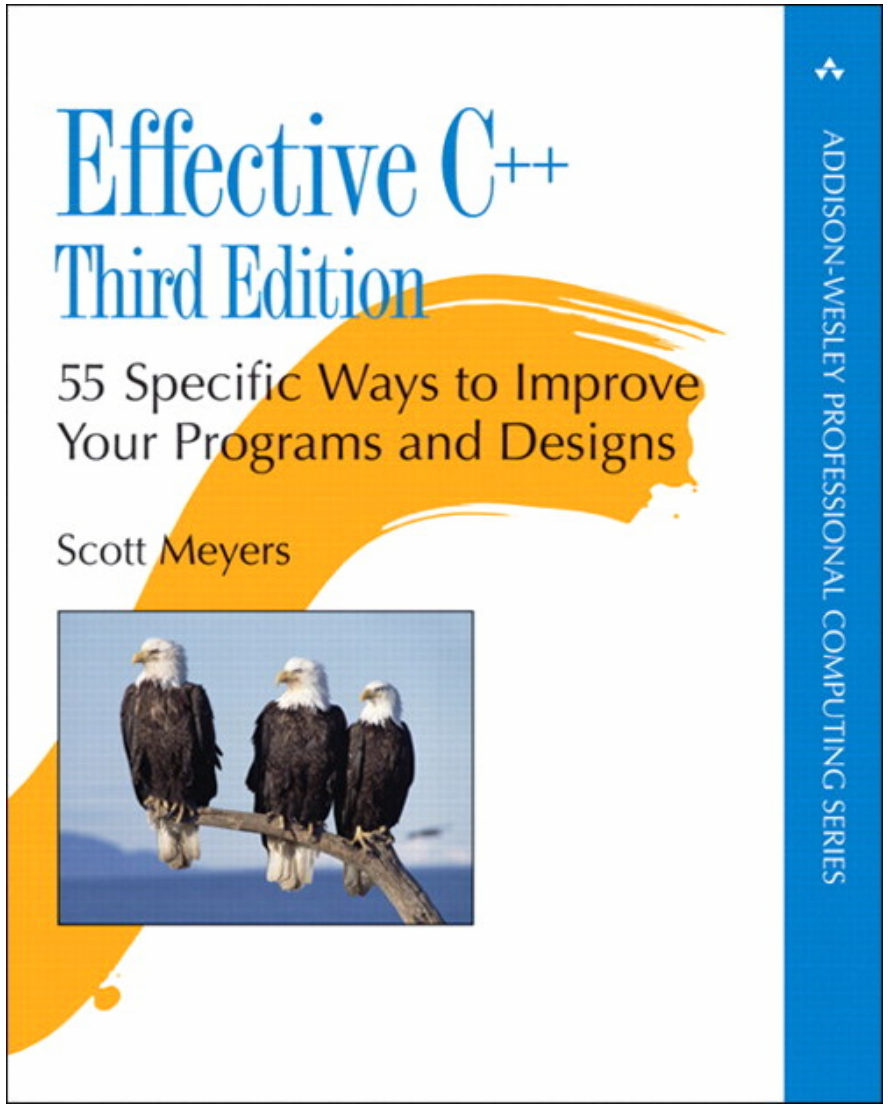
```
class PlayGround {
    Ball ballobj;
    public PlayGround() {
        Ball ballobj = new Ball();
        // local
    }
    public void paintComponent
        (Graphics gfx) {
        ballobj.update();
        // ballobj not created
    }
}
```

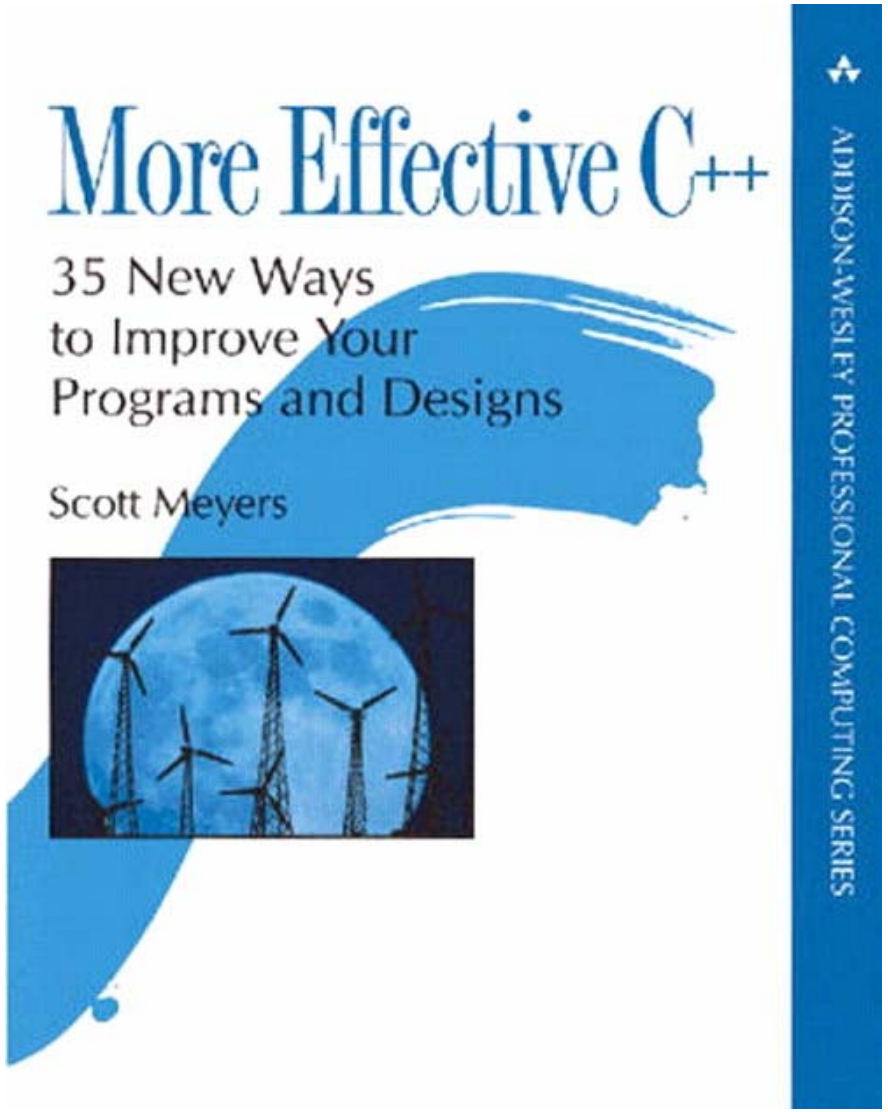
ECE 462
Object-Oriented Programming using C++ and Java

Lecture 25

Yung-Hsiang Lu

yunglu@purdue.edu


Effective C++
Third Edition
55 Specific Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


More Effective C++
35 New Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Do Not Break Encapsulation

```
// an example where encapsulate is
// broken
class X {
    public int val = 5;
}
class Y {
    private X xobj;
    public Y () {
        xobj = new X();
    }
    public X getX() {
        return xobj;
    }
public String toString()
    {  return String.valueOf(xobj.val);  }
}
public class breakencapsulate {
    public static void main( String[]
     args ) {
        Y yobj = new Y();
        System.out.println("jobj = " +
      yobj);
        X xobj = yobj.getX();
        xobj.val = 20;
        System.out.println("jobj = " +
      yobj);
    }
}
```

```
// encapsulate is broken
#include <iostream>
using namespace std;
class X {
public:
  int val;
  X() { val = 5; }
};
class Y {
private:
  X xobj;
public:
  Y () { }
  X & getX1() {
    return xobj;
  }
  X getX2() {
    return xobj;
  }
  void print() {
    cout << xobj.val << endl;
  }
};

int main(int argc, char* argv[]) {
  Y yobj;
  yobj.print();
  X & xobj1 = yobj.getX1();
  xobj1.val = 20;
  yobj.print();
  X xobj2 = yobj.getX2();
  xobj2.val = 35;
  yobj.print();
}
```

Virtual + Default Parameter Values

•Do not change the default value in a virtual function

```
#include <iostream>
using namespace std;
class BaseC {
public:
  virtual void foo(int val = 17) {
    cout << "BaseC::foo("
         << val << ")" << endl;
  }
};
class DerivedC: public BaseC {
public:
  virtual void foo(int val = 24) {
    cout << "DeriveC::foo("
         << val << ")" << endl;
  }
};
int main(int argc, char * argv[]) {
  BaseC * bobj = new DerivedC;
  bobj -> foo(3);
  bobj -> foo(); // 17 or 24?
}
```

Run-Time ⇒ Compiler Checking

```
class Date {
public:
     Date (int day, int month, int year);
}
```

- How to prevent an invalid object from being created?

```
Date today(-3, 19, 1999);
```

```
class Date {
public:
     Date (unsigned int day, unsigned int month, unsigned int year);
}
```

- Invalid objects are still possible.

```
Date today(67, 19, 10234);
```

```
class Date {
public:
    Date (unsigned int day, unsigned int month, unsigned int year) {
        if (month > 12) { /* handle error */ }
        if (year > 3000) { /* handle error */ }
        switch (month) {
        case 1:
        case 3:
        ...
          if (day > 31) { /* handle error */ }
        break;
        case 2:
          if ((year % 4) == 0) {
              if (day > 29) { /* handle error */ }
          } else {
              if (day > 28) { /* handle error */ }
          }
        }
```

Enumerate?

```
enum Month {Jan, Feb, Mar ...}
class Date {
public:
    Date (unsigned int d, Month m, unsigned int y);
}

Month m; // uninitialized
Date dobj (19, m, 2007); // m is uninitialized
```

- How do you force an attribute (with a small number of possible values) to be initialized at compile-time?

```
#include <iostream>
class Month {
public:
  static const int Jan = 1;
  static const int Feb = 2;
  static const int Mar = 3;
  // continue to create the 12 months
private:
  friend class Date;
  int m_month;
  Month(int m) { m_month = m; }
  // do not allow the creation of any
  // object, except Date
};
class Date {
private:
  unsigned int d_day;
  Month d_month;
  unsigned int d_year;
public:
  Date (unsigned int d, unsigned m,
     unsigned int y):
    d_day(d), d_month(m), d_year(y) {
  }
};

int main(int argc, char * argv[]) {
  // Month m1(Month::Jan); // error
  // Month m2(13); // error
  Date dobj (20, Month::Feb, 2008);
  Date dobj2(10, 50, 2008); // allowed
  return 0;
}
```

Restrict Object Creation

```
#include <iostream>
class Month {
public:
  static const int Jan = 1;
  static const int Feb = 2;
  static const int Mar = 3;
  // continue to create the 12 months
private:
  friend class Date;
  int m_month;
  Month(int m) { m_month = m; }
  // do not allow the creation of any
  // object, except Date
};
```

```
class Date {
private:
  unsigned int d_day;
  Month d_month;
  unsigned int d_year;
public:
  Date (unsigned int d, Month m,
     unsigned int y):
    d_day(d), d_month(m), d_year(y) {
  }
  static const Month Jan;
  static const Month Feb;
  static const Month Mar;
};
```

```
const Month Date::Jan(Month::Jan);
const Month Date::Feb(Month::Feb);
const Month Date::Mar(Month::Mar);

int main(int argc, char * argv[]) {
  Date dobj1 (20, Date::Feb, 2008);
  // Date dobj2 (10, 50, 2008); // error
  return 0;
}
```

Polymorphism + Array

Do not mix polymorphism and pointer arithmetics (you shouldn't use the latter anyway). Be careful about array of objects.

```
#include <iostream>
using namespace std;
class BaseC {
public:
  BaseC(int u = 3) { setU(u); }
  void setU(int u) { b_u = u; }
protected:
  int b_u;
  friend ostream & operator << (ostream & os, const BaseC & bobj);
};
```

```
ostream & operator << (ostream & os, const BaseC & bobj) {
  os << "B: " << bobj.b_u << endl;
  return os;
}
class DerivedC:public BaseC {
public:
  DerivedC(int u = 5, int v = 7) { setUV(u, v); }
  void setUV(int u, int v) { setU(u); d_v = v; }
private:
  int d_v;
  friend ostream & operator << (ostream & os, const DerivedC & dobj);
};
ostream & operator << (ostream & os, const DerivedC & dobj) {
  os << "D: " << dobj.b_u << " " << dobj.d_v << endl;
  return os;
}
```

```
void printArray(ostream & os, const BaseC array[], int numElem) {
  for (int ecnt = 0; ecnt < numElem; ecnt ++) {
    os << array[ecnt];
  }
  os << endl;
}
int main(int argc, char * argv[]) {
  BaseC bobj[3];
  printArray(cout, bobj, 3);
  DerivedC dobj[3];
  printArray(cout, dobj, 3);
  return 0;
}
```

What is the output?
Is it

B: 3
B: 3
B: 3

D: 5 7
D: 5 7
D: 5 7

Of course not, otherwise,
I wouldn't be asking you.

Array of Pointers?

```
#include <iostream>
using namespace std;
class BaseC {
public:
  BaseC(int u = 3) { setU(u); }
  void setU(int u) { b_u = u; }
  virtual void print() const {
    cout << "B: " << b_u << endl;
  }
protected:
  int b_u;
};
```

```
class DerivedC:public BaseC {
public:
  DerivedC(int u = 5, int v = 7) { setUV(u, v); }
  void setUV(int u, int v) { setU(u); d_v = v; }
  void print() const {
    cout << "D: " << b_u << " " << d_v << endl;
  }
private:
  int d_v;
};
void printArray(BaseC * array[], int numElem) {
  for (int ecnt = 0; ecnt < numElem; ecnt ++) {
    array[ecnt] -> print();
  }
  cout << endl;
}
```

```
int main(int argc, char * argv[]) {
  const int numElem = 3;
  BaseC * * bobj = new BaseC* [numElem];
  for (int ecnt = 0; ecnt < numElem; ecnt ++) {
    bobj[ecnt] = new BaseC;
  }
  printArray(bobj, numElem);
  DerivedC * * dobj = new DerivedC * [numElem];
  for (int ecnt = 0; ecnt < numElem; ecnt ++) {
    dobj[ecnt] = new DerivedC;
  }
  printArray(dobj, numElem);
  return 0;
}
```

```
#include <iostream>
#include <list>
using namespace std;
class BaseC {
public:
  BaseC(int u = 3) { setU(u); }
  void setU(int u) { b_u = u; }
  void print() const {
    cout << "B: " << b_u << endl;
  }
protected:
  int b_u;
};
class DerivedC:public BaseC {
public:
  DerivedC(int u = 5, int v = 7) { setUV(u, v); }
  void setUV(int u, int v) { setU(u); d_v = v; }
```

```
 void print() const {
    cout << "D: " << b_u << " " << d_v << endl;
  }
private:
  int d_v;
};
void printArray(list<BaseC*> bobj) {
  list<BaseC*>::iterator p = bobj.begin();
  while (p != bobj.end()) {
    (*p)->print();
    p ++;
  }
  cout << endl;
}
int main(int argc, char * argv[]) {
  const int numElem = 3;
```

```
 list<BaseC*> bobj;
  for (int ecnt = 0; ecnt < numElem; ecnt ++) {
    bobj.push_back(new BaseC);
  }
  printArray(bobj);
  list<BaseC*> dobj;
  for (int ecnt = 0; ecnt < numElem; ecnt ++) {
    dobj.push_back(new DerivedC);
  }
  printArray(dobj);
  return 0;
}
```

⇒ It is harder than you think.

How about Java?

```
import java.util.*;
class BaseC {
    public BaseC() { b_u = 3; }
    public void print()  {
     System.out.println("B: " + b_u);
    }
    protected int b_u = 3;
}
class DerivedC extends BaseC {
    public DerivedC() { b_u = 5; d_v = 7; }
    public void print() {
     System.out.println("D: " + b_u + " " + d_v);
    }
```

```
 private int d_v;
};
class listprog {
    private static void printArray(List<BaseC> blist) {
     ListIterator<BaseC> iter = blist.listIterator();
     while (iter.hasNext()) {
         BaseC bobj = iter.next();
         bobj.print();
     }
    }
    public static void main(String [] args) {
     final int numElem = 3;
     List<BaseC> blist = new ArrayList<BaseC>();
     for (int ecnt = 0; ecnt < numElem; ecnt ++) {
         blist.add(new BaseC());
     }
```

```
        printArray(blist);
        List<BaseC> dlist = new ArrayList<BaseC>();
        for (int ecnt = 0; ecnt < numElem; ecnt ++) {
            dlist.add(new DerivedC());
        }
        printArray(dlist);
    }
}
```

Combine C++ and C

- disable name mangling, add **extern "C"** before a C function
- initialize static variables, compile **main** in C++
- manage memory, follow **new-delete**, **malloc-free** rule
- minimize parameter passing, do not use virtual

```c
/* cppc1.c */
#include <stdio.h>
int func1(int v) {
  printf("C func1 %d\n", v);
  return (v + 4);
}
```

```cpp
// cppc2.cpp
#include <iostream>
using namespace std;
extern "C" {
  int func1(int v);
}

int main(int argc, char * argv[]) {
  cout << func1(15) << endl;
  return 0;
}
```

```c
/* cppc3.c */
#include <stdio.h>
int main(int argc, char * argv[]) {
  printf("main\n");
  return 0;
}
```

```cpp
// cppc4.cpp
#include <iostream>
using namespace std;
class X {
public:
  X() { cout << "X::X" << endl; }
};

X xobj;
```

```
/* main.c */
int main(int argc, char * argv[]) {
}
```

⇒

```
/* realmain.c */
int realmain(int argc, char * argv[])
    {
}
```

```
// main.cpp
int main(int argc, char * argv[]) {
    realmain(argc, argv);
}
```

Courses After 462

- Computer Graphics
- Embedded Systems
- Programming Languages
- Software Engineering: plan, design, manage, evaluate, and maintain non-trivial software projects

Software Engineering

- "Software Engineering" started in 1968
- a subject for constructing non-trivial software projects
  - satisfy the requirements
  - meet the schedule
  - not exceed the budget
  - be robust
  - (in many cases) possible to understand and modify by people that did not construct the original software

What is NOT Software Engineering

- programming skills
- programming languages
- computation theory (automata theory)
  - what problems can be solved by computers?
  - are there solutions to a problem with constraints?
  - what is the minimum complexity of a problem?
- design or analysis of algorithms
- building a large program

Software is not “Flexible”

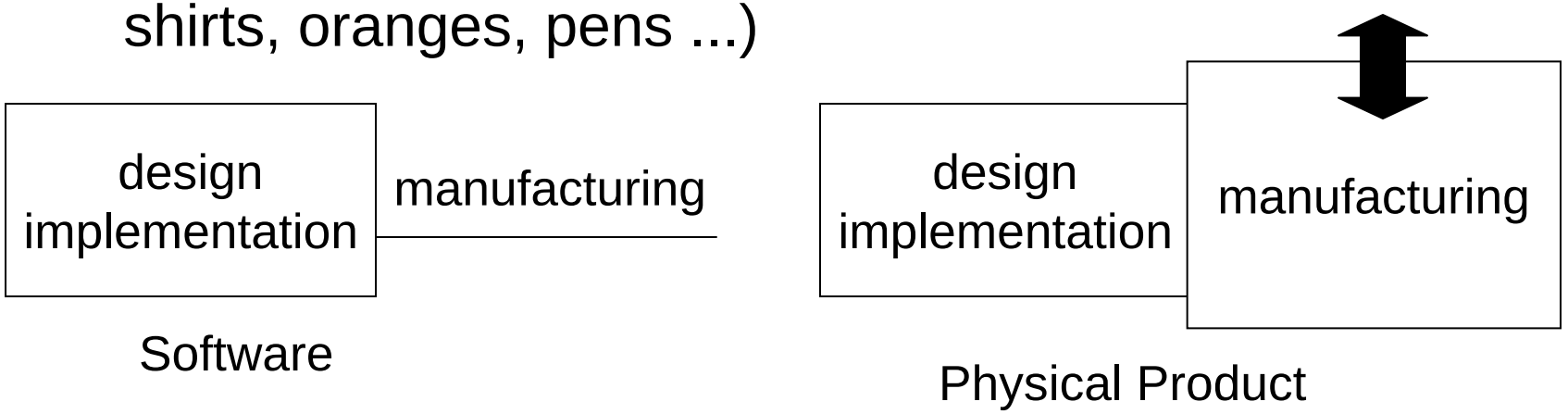
- When was the last time you change the operating system, editor, email, browser…?
- When was the last time you change the computer hardware?
- Today, software has so much “inertia” that software is not “flexible” any more. Changing hardware is easy. Changing software is hard.
- Your code may live much longer than you expect. Think of Y2K problem (next deadline: 2038)

Problems of Software

- unreliable, crash too often
- cost too much and take too long to develop
- Most developers do not know why the software works or fails. New developers cannot understand existing software and make fixes or improvement.
- People have wrong ideas about software. Some people demand substantial changes as "maintenance". (Can you ask for a sunroof and a new engine during the maintenance of your car?)

Software vs. Tangible "Things"

- Software has unique properties:
  - Software does not degrade, rust, wear out ...
  - "Software" is applied to practically every field now.
  - Reproduction (i.e. copy) cost is very low ⇒ **Every commercial software is unique**. You cannot compete by manufacturing capacity (like making cars, shirts, oranges, pens ...)



What does Software Engineering Teach You

- requirement analysis: what needs to be done (tasks)
- schedule management: order of tasks, what should be done earlier, how long does each task take
- testing and bug tracking
- version control and concurrent development
- team management: find the right person for the job
- documentation
- change management
- measurement and evaluation

Requirement Analysis

- How to write the requirements for a software project
  - feasible within the time, personnel, budget, technology limits
  - attractive to potential customers
  - measurable success in several stages
  - superior to competitors' products
  - satisfy specific constraints, such as time, weight, power, heat / temperature ...
- What would you do to create a project 20-25 times larger than PA1 (4-5 people for 15 weeks)?

Quality Assurance

- How to develop **systematic and automatic** procedures to ensure the quality of software at different stages of the projects?
- How to quickly discover problems, isolate and solve the problems, and prevent similar problems from happening?
- How to track bugs (who puts them in, when are they introduced, what are affected, when are they discovered, who fixes them ...) and develop software with different versions with bug fixes?

Evaluation

- How do you know one team if more likely to be successful in developing high-quality software?
- How to know you have
  - right amount of time is allocated to a project?
  - right number of personnel with the right sets of skills?
  - right tools and development environment?
- If you are a customer, which team will you hire?
- If you are a company, which project will you bid?

See You Next Semester in ECE 461 Software Engineering