# ECE 462 Midterm Exam 2

# 10:30-11:20AM, October 19, 2007

## 1 Overloading and Overriding

### 1.1 Overloading in C++ and Java

Which statement is correct?

*Answer: E*

A. In C++, if a function is overloaded, it cannot be overridden in a derived class.

B. In Java, if a function is overloaded, it cannot be overridden in a derived class.

C. Overloaded functions in C++ must use primitive types; objects cannot be used as parameters in overloaded functions.

D. In Java, overloaded functions can be distinguished by both the argument types and the return types.

E. In Java, overloaded functions can use objects as parameters.

### 1.2 Overloading in C++

Which statement is correct?

*Answer: A*

A. In C++, an integer cannot be promoted to string automatically.

B. In C++, only primitive types can distinguish which version of the overloaded function. Objects cannot decide the overloaded function.

C. In C++, a function can be overloaded only if it is a method in a class.

D. In C++, if a function is overloaded, a derived class must override all versions of the overloaded function.

E. In C++, a string argument may be automatically converted to double to match an overloaded function.

## 1.3 Overloading in Java

Which statement is correct?

*Answer: A*

A. In Java, a program cannot compile if ambiguity occurs in deciding which version of an overloaded function to choose.

B. In Java, a user can give "hint" during program execution to decide which version of an overloaded function to choose by specifying input.

C. In Java, if an exact match is unavailable, an object of class X may match to an object of class Y, here Y is derived from X (`class Y extends X`).

D. In Java, overloading must be distinguished using primitive types; objects are not allowed.

E. In Java, all versions of an overloaded function must have the same number of arguments; these arguments have different types.

## 1.4 Overloading in Java

What is the output of this program?

*Answer: E*

```
class X
{
    public void print() { System.out.println("()"); }
    public void print(int v1) { System.out.println("(int)"); }
    public void print(int v1, int v2) { System.out.println("(int, int)"); }
}
class prog
{
    public static void main (String[] args)
    {
        X obj;
        obj = new X();
        obj.print();
        obj.print(1);
        obj.print(1, 2);
    }
}
```

```
A. This program does not compile.
```

```
B. ()
   ()
   ()
```

```
C. (int)
   (int)
   (int)

D. (int, int)
   (int, int)
   (int, int)

E. ()
   (int)
   (int, int)
```

## 1.5 Overloading and Overriding and Class Hierarchy in Java

What is the output of this program? If the program cannot compile, write "cannot compile".

*Answer:* `X() X(X, X) X(X, X) X(X, X)`

```
class X
{
    public void print() { System.out.println("X()"); }
    public void print(X v1) { System.out.println("X(X)"); }
    public void print(X v1, X v2) { System.out.println("X(X, X)"); }
}
class Y extends X
{
    public void print(Y v1, Y v2)
    { System.out.println("Y(Y, Y)"); }
}
class prog
{
    public static void main (String[] args)
    {
        X obj0, obj1;
        obj0 = new X();
        obj1 = new Y();
        obj0.print();
        obj0.print(obj0, obj0);
        obj0.print(obj0, obj1);
        obj1.print(obj1, obj0);
    }
}
```

## 1.6 Overloading and Overriding

What is the output of this program? You can write the steps how `val` is changed without doing the calculation. For example, you can write `1 + 2` instead of `3`. If the program cannot compile, write "cannot compile".

*Answer: cannot compile.*

*cannot find symbol symbol : method foo(java.lang.String) location: class MyBaseClass mobj.foo(str);*

```
class MyBaseClass {
    protected int val;
    public MyBaseClass() { val = 1; }
    public void foo() { val += 2; }
    public void foo(int i) { val += 3; }
    public int getVal() { return val; }
}

class MyDerivedClass extends MyBaseClass {
    public MyDerivedClass () { val = 4; }
    public void foo() { val += 5; }
    public void foo(String str) { val += 6; }
}

class Test {
    public static void main(String[] args)
    {
        MyBaseClass mobj = new MyDerivedClass();
        String str = new String("hello");
        mobj.foo();
        mobj.foo(str);
        mobj.foo(4);
        System.out.println("val = " + mobj.getVal());
    }
}
```

# 2 Operator Overloading

## 2.1 Operator Overloading and Object-Oriented Programming

Which statement is correct?

*Answer: E*

A. Operator overloading is supported in C++ and Java.

B. If operator overloading is not supported, inheritance cannot be achieved.

C. C++ allows redefining the precedence of operators.

D. In C++, an operator can be overloaded for primitive types. In Java, an operator can be overloaded only for user-defined classes.

E. Operator overloading is not essential for object-oriented programming.

## 2.2 Operator Overloading in C++

Which statement is correct?

*Answer: C*

A. If operators < and == are overloaded, it is unnecessary to overload <=. Calling <= will be automatically converted to calling < and ==.

B. Operator "-" can be overloaded only for binary operation.

C. A binary operator can be implemented as a member function.

D. A programmer can change the precedence of operators in C++.

E. Overloaded operators must be member functions.

## 2.3 Operator Overloading as Global Functions

Which statement is correct, assuming correct code segments are filled into the appropriate locations.

*Answer: C*

```
#include <iostream>
using namespace std;
class MyComplex {
    double re, im;
public:
    MyComplex( double r, double i ) : re(r), im(i) {}
    double getReal() const { return re; }
    double getImag() const { return im; }
};
MyComplex operator+ /* A */ {
    double d1 = arg1.getReal() + arg2.getReal();
    double d2 = arg1.getImag() + arg2.getImag();
    return MyComplex( d1, d2 );
}
MyComplex operator- (MyComplex arg1, MyComplex arg2 ) {
    double d1 = arg1.getReal() - arg2.getReal();
    double d2 = arg1.getImag() - arg2.getImag();
    return MyComplex( d1, d2 );
}
ostream& operator << /* B */ {
    double d1 = arg.getReal();
    double d2 = arg.getImag();
    os << "(" << d1 << ", " << d2 << ")" << endl;
    /* C */
}
int main() {
    MyComplex first(3, 4);
    MyComplex second(2, 9);
    cout << first;
    cout << second;
    cout << first + second;
    cout << first - second; /* D */
    return 0;
}
```

A. This program has compile-time error.

B. /* A */ should be replaced by `(const MyComplex arg2 )`.

C. /* B */ should be replaced by `( ostream & os, const MyComplex & arg )`.

D. /* C */ should be replaced by `return MyComplex( d1, d2 );`.

E. /* D */ can be replaced by `cout << - second + first;` .

## 2.4 Operator Overloading as Global and Member Functions

Which statement is correct, assuming correct code segments are filled into the appropriate locations.

*Answer: D*

```
#include <iostream>
using namespace std;
class MyComplex {
  double re, im;
public:
  MyComplex( double r, double i ) : re(r), im(i) {}
  MyComplex operator-() const; /* A */
  double getReal() const { return re; }
  double getImag() const { return im; }
  friend ostream& operator<< ( ostream&, const MyComplex& );
};
MyComplex MyComplex::operator-() const {
  return MyComplex( -re, -im ); /* B */
}
MyComplex operator+( const MyComplex arg1, const MyComplex arg2 ) {
  double d1 = arg1.getReal() + arg2.getReal();
  double d2 = arg1.getImag() + arg2.getImag();
  return MyComplex( d1, d2 );
}
MyComplex operator-( MyComplex arg1, MyComplex arg2 ) {
  double d1 = arg1.getReal() - arg2.getReal(); /* C */
  double d2 = arg1.getImag() - arg2.getImag();
  return MyComplex( d1, d2 );
}

ostream& operator<< ( ostream& os, const MyComplex& c ) {
  os << "(" << c.re << ", " << c.im << ")" << endl;
  return os;
}

int main()
{
  MyComplex c(3, 4);
  MyComplex z = -c;       /* D */
  cout << z << endl;
  MyComplex first(3, 4);
  MyComplex second(2, 9);
  cout << first;
  cout << second;
  cout << first + second;
  cout << first - second;
  cout << - second + first; /* E */
  return 0;
}
```

A. /* A */ causes compile-time error because there is no argument.

B. /* B */ causes compile-time error because `re` and `im` are private.

C. /* C */ can be replaced by `double d1 = arg1.re - arg2.re;` .

D. /* D */ calls `MyComplex MyComplex::operator-() const`.

E. /* E */ calls `MyComplex operator-( MyComplex arg1, MyComplex arg2)` and `MyComplex operator+( const MyComplex arg1, const MyComplex arg2 )`.

## 2.5 Overloading Operator <<

Please fill in the place marked by /* here */ so that the program can compile. If nothing is needed, please write *nothing*.

*Answer:* `friend`

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

class MyComplex {
    double re, im;
public:
    MyComplex( double r, double i ) : re(r), im(i) {}
    MyComplex operator+( MyComplex) const;
    MyComplex operator-( MyComplex) const;
    /* here */ ostream& operator<< ( ostream&, const MyComplex& );
};
MyComplex MyComplex::operator+( const MyComplex arg ) const {
    double d1 = re + arg.re;
    double d2 = im + arg.im;
    return MyComplex( d1, d2 );
}
MyComplex MyComplex::operator-( const MyComplex arg ) const {
    double d1 = re - arg.re;
    double d2 = im - arg.im;
    return MyComplex( d1, d2 );
}
ostream& operator<< ( ostream& os, const MyComplex& c ) {
    os << "(" << c.re << ", " << c.im << ")" << endl;
    return os;
}
int main()
{
    MyComplex first(3, 4);
    MyComplex second(2, 9);

    cout << first;                      // (3, 4)
    cout << second;                     // (2, 9)
    cout << first + second;             // (5, 13)
    cout << first - second;             // (1, -5)
    return 0;
}
```

## 2.6 Overloading Operator <<

Please fill in the place marked by /* here */. If nothing is needed, please write *nothing*.

*Answer:* `ostream& operator<< ( ostream& os, const MyComplex& arg )`

```
#include <iostream>
using namespace std;
class MyComplex {
    double re, im;
public:
    MyComplex( double r, double i ) : re(r), im(i) {}
    double getReal() const { return re; }
    double getImag() const { return im; }
};
MyComplex operator-( const MyComplex arg ) {
    return MyComplex( -arg.getReal(), -arg.getImag() );
}

/* here */
{
    double d1 = arg.getReal();
    double d2 = arg.getImag();
    os << "(" << arg.getReal() << ", " << arg.getImag() << ")" << endl;
    return os;
}
int main()
{
    MyComplex c(3, 4);
    cout << c;               // (3, 4)
    cout << -c;              // (-3, -4)
    return 0;
}
```

# 3 Exception Handling

## 3.1 Exception in C++

Which statement is correct?

*Answer: B*

A. In C++, a function may throw at most one type of exception.

B. In C++, an integer can be passed when an exception is thrown.

C. In C++, a function that may throw an exception must not call another function that may throw a different type of exception.

D. An exception must be caught by the immediate caller; otherwise the program terminates.

E. In C++, if a function may throw an exception, the function must be called within a try-catch block.

## 3.2 Exception in Java

Which statement is correct?

*Answer: E*

A. In Java, an integer can be passed when an exception is thrown.

B. In Java, a function may throw at most one type of exception.

C. If an exception is not caught, the program causes memory leak.

D. In Java, an exception can pass only a primitive type (such as integer, char, or float).

E. In Java, if a function may throw an exception, this must be declared at the beginning of the function.

## 3.3 Exception in Java

Which statement is correct?

*Answer: A*

```
class MyException /* A */
{
    String message;
    public MyException( String mess ) { message = mess; }
}
```

```
class Test
{
    static void f() /* B */
    {
        throw new MyException( "Hello from f()" );
    }
    public static void main( String[] args )
    {
        /* C */
        try {
            f();
        } catch( MyException e ) { /* D */
            System.out.println( e.message );
        }
    }
}
```

A. /* A */ can be replaced by `extends Exception`.

B. /* A */ can be replaced by `extends Exception` or left empty; either can work.

C. /* B */ can be replaced by `throws String;`.

D. /* C */ can call `f();`

E. /* D */ anything between `catch ...  { ...  }` can be removed.

### 3.4 Exception in C++

Which statement is correct?

*Answer: B*

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Err {};
void f( int ); /* A */
int main()
{
  try
    {
      f(0); /* B */
    }
  catch ( Err )
    {
      cout << "caught Err" << endl;
      exit(0);
    }
  return 0;
```

```
}
void f(int j) /* C */
{
  cout << "function f invoked with j = " << j << endl;
  if (j == 3) throw Err();
  f( ++j ); /* D */
}
```

A. /* A */ must be replaced by `throw(int);`.

B. If /* B */ `f(0);` is replaced by `f(5);` no exception will be thrown but the program will eventually run out of call stack.

C. /* A */ must be replaced by `throw(Err);`.

D. /* C */ must be replaced by `throw(Err);`.

E. /* D */ must put `f(++j);` inside a `try-catch` block.

## 3.5 Exception in C++

What should be /* here */ to handle an exception created by `f`? If nothing is needed, please write *nothing*.

*Answer:* `catch( MyException ex )`

```
#include <iostream>
#include <string>
using namespace std;
class MyException {
    string message;
public:
    MyException( string m ) { message = m; }
    string getMessage() { return message; }
};
void f() throw( MyException );
int main()
{
    try {
        f();
    } /* here */
    { cout << ex.getMessage(); }
    return 0;
}

void f() throw( MyException ) {
    throw MyException( "hello there" );
}
```

### 3.6 Exception in Java

What should be /* here */ so that the `catch` block prints "Hi"? If nothing is needed, please write *nothing*.

*Answer:* `throw new MyException( "Hi" );`

```
class MyException extends Exception {
    String message;
    public MyException( String mess ) { message = mess; }
}

class Test {
    static void f() throws MyException {
        /* here */
    }

    public static void main( String[] args )
    {
        try {
            f();
        } catch( MyException e ) {
            System.out.println( e.message ); // Hi
        }
    }
}
```

# 4 Multiple Thread and Synchronization

## 4.1 Thread States in Java

A Java thread may be in one of several states. Which one is **not** included?

*Answer: B*

A. running

B. Indiana

C. waiting

D. sleeping

E. born

## 4.2 Thread in Java

Which statement is correct?

*Answer: D*

A. Calling the "run" method of a thread object (for example `obj.run()`) will invoke the object's "start" method.

B. Creating a thread (calling `new`) makes the thread execute immediately.

C. A thread can be created by instantiating a class that is derived (`extends`) from `Runnable`.

D. After creating a thread, call "start" to start the thread's execution.

E. If a thread is created (`new`) first, the thread always finishes first.

## 4.3 Thread Execution Order

Which statement is correct?

*Answer: C*

```
class HelloThread extends Thread {
    String message;

    HelloThread( String message ) { this.message = message; }

    public void run() {
        int sleeptime = (int) ( Math.random() * 3000 );
        try {
            sleep( sleeptime );
        } catch( InterruptedException e ){}
        System.out.print( message );
    }
    public static void main( String[] args )
    {
        HelloThread ht1 = new HelloThread( " Good" );
        HelloThread ht2 = new HelloThread( " morning" );
        HelloThread ht3 = new HelloThread( " to" );
        ht1.start();
        ht2.start();
        ht3.start();
        int sleeptime = (int) ( Math.random() * 3000 );
        try {
            sleep( sleeptime );
        } catch( InterruptedException e ){}
        System.out.println( " you!" );
    }
}
```

A. The output is always "Good morning to you!".

B. The word "you!" appears last but the order of "Good", "morning", and "to" may change in each execution.

C. The order of "Good", "morning", "to", and "you!" may change in each execution.

D. This program may cause deadlock.

E. The statements `ht1.start(); ht2.start(); ht3.start();` are unnecessary for printing the words.

## 4.4 Java `wait-notify`

Which statement is correct?

*Answer: E*

A. If a `synchronized` method calls `wait`, the effect is equivalent to calling `abort`.

B. If a `synchronized` method calls `wait`, the effect is equivalent to calling `return`.

C. A `synchronized` method must not call `wait`; otherwise, the program cannot compile.

D. After calling `wait`, the method should immediately calls `notifyAll`.

E. Calling `wait` is usually enclosed within a condition that is checked by a `while` block.

## 4.5 Thread in C++ using Qt

Rewrite the following Java code to equivalent C++ code using Qt. Your code will replace /* here */. Please use Qt 4.0 or later.

```
synchronized void deposit( int dep ) {
    balance += dep;
    notifyAll();
}
```

*Answer:*

```
mutex.lock();
balance += dep;
cond.wakeAll();
mutex.unlock();
```

```
#include <QThread>
#include <QMutex>
#include <QWaitCondition>
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;

QMutex mutex;
QWaitCondition cond;

class Account : public QThread {
public:
    int balance;
    Account() { balance = 0; }
    void deposit( int dep ) {
        /* here */
    }
    void withdraw( int draw ) {
        mutex.lock();
        while ( balance < draw ) {
            cond.wait( &mutex );
        }
        balance -= draw;
        mutex.unlock();
    }
    void run(){}
};

Account sacct;
class Depositor : public QThread {
```

```
public:
    void run() {
        int i = 0;
        while ( true ) {
            int x = (int) ( rand() % 10 );
            sacct.deposit( x );
            if ( i++ % 100 == 0 )
                cerr << "balance after deposits: "
                     <<  sacct.balance << endl;
        }
    }
};

class Withdrawer : public QThread {
public:

    void run() {
        int i = 0;
        while ( true ) {
            int x = (int) ( rand() % 10 );
            sacct.withdraw( x );
            if ( i++ % 100 == 0 )
                cerr << "balance after withdrawals:  "
                     << sacct.balance << endl;
        }
    }
};
```

## 4.6 Threads and Object

What is the output of this program?

*Answer: always 100 (the threads do not share objects)*

```
class DataObject {
  int dataItem1;
  int dataItem2;

  DataObject() {
    dataItem1 = 50;
    dataItem2 = 50;
  }
  void itemSwap() {
    int x = (int) ( -4.999999 + Math.random() * 10 );
    dataItem1 -= x;
    keepBusy(10);
    dataItem2 += x;
  }
  void test() {
    int sum = dataItem1 + dataItem2;
    System.out.println( sum );
  }
  public void keepBusy( int howLong ) {
    long curr = System.currentTimeMillis();
    while ( System.currentTimeMillis() < curr + howLong ) ;
  }
}

class RepeatedSwaps extends Thread {
  DataObject dobj;

  RepeatedSwaps( DataObject d ) {
    dobj = new DataObject();
    start();
  }
  public void run( ) {
    int i = 0;
    while ( i < 200 ) {
      dobj.itemSwap();
      dobj.test();
      i++;
    }
  }
}

public class Swap {
  public static void main( String[] args ) {
    DataObject d = new DataObject();
    new RepeatedSwaps( d );
```

```
    new RepeatedSwaps( d );
    new RepeatedSwaps( d );
    new RepeatedSwaps( d );
  }
}
```

# Contents