# ECE 462 Exam 3

# 09:30-10:20AM, November 10, 2008

## 1 Java Synchronized Methods (outcome 8, 5 points)

What is the output of this program? Briefly explain whether a deadlock is **possible**. Your need to answer **both** questions to obtain the full score.

*Answer:*

```
f1: 3
f2: 3
f2: 19
f1: 3
f2: 3
f2: 19
done

f1: 3 must appear before f2:3
3 must appear before 19

Deadlock is impossible because there is no cyclic waiting.
```

```
class Account {
    final int a_id;
    Account(int i) {
        a_id = i;
    }
    synchronized void func1 () {
        System.out.println("f1: " + a_id);
        func2();
    }
    synchronized void func2 () {
        System.out.println("f2: " + a_id);
    }
}

class TransThread extends Thread {
    Account tt_act1;
    Account tt_act2;
    TransThread(Account a1, Account a2) {
        tt_act1 = a1;
        tt_act2 = a2;
    }
    public void run() {
```

```
        tt_act1.func1();
        tt_act2.func2();
    }
}

public class outcome871 {
    public static void main( String[] args ) {
        Account act1 = new Account(3);
        Account act2 = new Account(19);
        TransThread tt1 = new TransThread(act1, act2);
        TransThread tt2 = new TransThread(act1, act2);
        tt1.start();
        tt2.start();
        try {
            tt1.join();
            tt2.join();
        } catch (Exception je) {
            System.out.println("join exception " + je);
        }
        System.out.println("done");
    }
}
//
```

## 2 Virtual Function in C++ (outcome 7, 5 points)

What is the output of this program?

*Answer:*

```
B()
D1(int)
B(string)
B(double)
D1(int)
```

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

class Base
{
public:
  void f1()                 { cout << "B()" << endl; } // <--- not virtual
  void f1(string s)         { cout << "B(string)" << endl; }
  virtual void f1(int v)    { cout << "B(int)" << endl; }
  virtual void f1(double v) { cout << "B(double)" << endl; }
};
```

```
class Derived1: public Base
{
public:
  void f1()                 { cout << "D1()" << endl; }
  virtual void f1(int v)    { cout << "D1(int)" << endl; }
};

class Derived2: public Derived1
{
public:
  void f1(string s)         { cout << "D2(string)" << endl; }
  virtual void f1(int v)    { cout << "D2(int)" << endl; }
};

int main(int argc, char * argv[])
{

  Base * obb = new Derived1;
  Base * obd1 = new Derived1;
  Base * obd2 = new Derived2;

  obb  -> f1();
  obd1 -> f1(3);
  obd2 -> f1("ece462");
  obd1 -> f1(46.2);
  obb  -> f1('c');
  // do not worry about delete here
  return 0;
}
//
```

## 3 Overload using Return Types (outcome 7, 5 points)

What is the output of this program?

*Answer:*

```
outcome741.java:9: f1() in Derived1 cannot override f1() in Base;
attempting to use incompatible return type

found   : java.lang.String
required: int
    public String f1() { return "Derived1"; }
                  ^
outcome741.java:10: f2() in Derived1 cannot override f2() in Base;
attempting to use incompatible return type
found   : int
required: java.lang.String
    public int f2() { return 4; }
               ^
outcome741.java:15: f1() in Derived2 cannot override f1() in Derived1;
attempting to use incompatible return type
found   : void
required: java.lang.String
    public void f1() { System.out.println("Derived2.f1"); }
                ^
outcome741.java:16: f2() in Derived2 cannot override f2() in Derived1;
attempting to use incompatible return type
found   : void
required: int
    public void f2() { System.out.println("Derived2.f2"); }
                ^
outcome741.java:22: 'void' type not allowed here
        System.out.println(dobj.f1());
                                  ^
outcome741.java:23: 'void' type not allowed here
        System.out.println(dobj.f2());
                                  ^
6 errors
```

```
class Base
{
    public int f1() { return 1; }
    public String f2() { return "Base"; }
}

class Derived1 extends Base
{
    public String f1() { return "Derived1"; }
    public int f2() { return 4; }
}

class Derived2 extends Derived1
{
    public void f1() { System.out.println("Derived2.f1"); }
    public void f2() { System.out.println("Derived2.f2"); }
}
```

```
class outcome731 {
    public static void main( String[] args ) {
        Derived2 dobj = new Derived2();
        System.out.println(dobj.f1());
        System.out.println(dobj.f2());
    }
}
//
```

# 4 Timer using Thread (outcome 8, 10 points)

Create a timer class so that it periodically calls the `update` function of an `Actuator` object.

*Answer:*

```
    public Outcome8Timer (Actuator a, int d, int r)
    {
        t_act = a;
        t_delay = d;
        t_repeat = r;
    }
    public void run()
    {
        for (int r = 0; r < t_repeat; r++)
            {
                try
                    {
                        t_act.update();
                        sleep(t_delay);
                    }
                catch (Exception ept)
                    {
                        System.out.println("caught exception");
                    }
            }
    }
```

```
class Actuator {
    public void update()
    {
        System.out.println("update called at " +
                           System.currentTimeMillis());
    }
}

class Outcome8Timer extends Thread {
    private Actuator t_act;
    private int t_delay;
```

```
    private int t_repeat;
    // >>>>>
    // The constructor initializes the three attributes
    // <<<<<









    // >>>>>
    // when start is called, call t_act.update() once every
    // t_delay millisecond.  This repeats t_repeat times.
    // If you use "sleep", please remember to enclose it inside
    // a try block.
    // <<<<<









}

public class outcome821 {
    public static void main( String[] args ) {
        Actuator act = new Actuator();
        Outcome8Timer t = new Outcome8Timer(act, 1500, 20);
        // call act.update every 1500 millisecond for 20 times
        t.start();
        try
            {
                t.join();
            }
        catch (Exception ept)
            {
                System.out.println("caught exception");
            }
    }
}
```

//

## 5 Java Thread (outcome 8, 5 points)

What are the **possible** outputs of the following program? If there are many possible outputs, describe their **common properties** (for example, positive integers between 50 and 200).

*Answer:*

*always 100 since the threads do not share objects.*

```
class DataObject {
    int do_x1;
    int do_x2;
    DataObject() {
        do_x1 = 50;
        do_x2 = 50;
    }
    void swap() { // <--- not "synchronized"
        int x = (int) ( -4.999999 + Math.random() * 10 );
        do_x1 -= x;
        do_x2 += x;
    }
    void print() {
        int sum = do_x1 + do_x2;
        System.out.println( sum );
    }
}

class SwappingThread extends Thread  {
    DataObject dobj;

    SwappingThread() {
        dobj = new DataObject();
        start();
    }
    public void run( ) {
        int i = 0;
        while ( i++ < 20000 ) {
            dobj.swap();
            yield(); // Causes the currently executing thread object
                     // to temporarily pause and allow other threads
                     // to execute.
            if ( i % 4000 == 0 ) dobj.print();
            try { sleep( 1 ); } catch( InterruptedException e ) {}
        }
    }
}
```

```
public class outcome851 {
    public static void main( String[] args ) {
        new SwappingThread( );
        new SwappingThread( );
        new SwappingThread( );
        new SwappingThread( );
    }
}
//
```

# 6 C++ Thread (outcome 8, 5 points)

Create 4 threads; each thread adds two elements.

*Answer:*

```
  AdderThread t1(a1, a2, a3);
  AdderThread t2(b1, b2, b3);
  AdderThread t3(c1, c2, c3);
  AdderThread t4(d1, d2, d3);
  t1.start();
  t2.start();
  t3.start();
  t4.start();
```

```
#include <iostream>
#include <QtCore>
using namespace std;
class AdderThread: public QThread
{
public:
  AdderThread(int & a, int & b, int & c):
  at_a(a), at_b(b), at_c(c)
  { }
  void run()
  { at_c = at_a + at_b; }
  int getC() { return at_c; }
private:
  int & at_a; // reference
  int & at_b;
  int & at_c;
};

int main(int argc, char * argv[])
{
  int a1 = 1;
  int a2 = 2;
  int a3 = 0;
```

```
  int b1 = 4;
  int b2 = 5;
  int b3 = -70;
  int c1 = 9;
  int c2 = 8;
  int c3 = -4;
  int d1 = 7;
  int d2 = 15;
  int d3 = -11;
  // >>>>>
  // create four threads called t1, t2, t3, and t4
  // t1 adds a1 and a2, stores the result in a3
  // t2 adds b1 and b2, stores the result in b3
  // t3 adds c1 and c2, stores the result in c3
  // t4 adds d1 and d2, stores the result in d3
  // <<<<<
  t1.wait();
  t2.wait();
  t3.wait();
  t4.wait();
  cout << a3 + b3 + c3 + d3 << endl;
  // output:
  // 51
  // (for your information) 51 = 1 + 2 + 4 + 5 + 9 + 8 + 7 + 15
  return 0;
}
//
```

# 7 Multiple Thread (outcome 8, 5 points)

Which statement is correct?

*Answer:* $\boxed{E}$

A. A multithread program always produces the same result if the program executes on the same computer.

B. In Java, `x += y;` is an atomic operation.

C. In C++, if a class implements the `Runnable` interface, the class must override the `run` method.

D. If a program **may** cause a deadlock, it **always** causes a deadlock.

E. In Java, "circular wait" is a necessary condition of deadlocks but it is **not** a sufficient condition.

## 8 Virtual and Overloaded Function in C++ (outcome 7, 5 points)

What is the output of this program?

*Answer:*

*16 = 10 Derived2::f1(int) + 1 Base::f1 + 5 Derived2::f2*

```
#include <iostream>
using namespace std;
int gvalue = 0;
class Base
{
public:
  void f1() { gvalue += 1; } // <--- not virtual
  virtual void f2() { gvalue += 2; }
};

class Derived1: public Base
{
public:
  void f1() { gvalue += 3; }
  void f2() { gvalue += 4; }
};

class Derived2: public Derived1
{
public:
  void f1(int v) { gvalue += v; }
  void f2() { gvalue += 5; }
};

int main(int argc, char * argv[])
{
  Derived2 dobj;
  dobj.f1(10);
  Base * bobj = new Derived2;
  bobj -> f1();
  bobj -> f2();
  cout << gvalue << endl;
  return 0;
}
//
```

## 9 Overloading and Overriding (outcome 7, 5 points)

Which statement is correct?

*Answer:* C

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

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

C. In C++, if a function is overloaded, it can still be overridden in a derived class.

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

E. In Java, overloaded functions cannot use objects as parameters.

## 10 Overloading with Promotion (outcome 7, 5 points)

What is the output of this program?

*Answer:*

```
Bd
D2s
D1i
D1i
D2s
```

```
class Base
{
    public void f1(double v) { System.out.println("Bd"); }
}

class Derived1 extends Base
{
    public void f1(String v) { System.out.println("D1s"); }
    public void f1(char v)   { System.out.println("D1c"); }
    public void f1(int v)    { System.out.println("D1i"); }
}

class Derived2 extends Derived1
{
    public void f1(double v) { System.out.println("D2d"); }
    public void f1(String v) { System.out.println("D2s"); }
}

class outcome761 {
    public static void main( String[] args ) {
        Base bd1obj = new Derived1();
        bd1obj.f1(3.14159);

        Derived2 d2d2obj = new Derived2();
        d2d2obj.f1("ece462");
        d2d2obj.f1(10);
```

```
        Derived1 d1d2obj = new Derived2();

        d1d2obj.f1(1110);
        d1d2obj.f1("ece462");
    }
}
//
```

## 11 Overload and Override in Java (outcome 7, 5 points)

What is the output of this program?

*Answer:*

*18 = 10 Derived2.f1(int) + 3 Derived1.f1 + 5 Derived2.f2*

```
class Base
{
    public static int gvalue = 0;
    public void f1() { gvalue += 1; }
    public void f2() { gvalue += 2; }
}

class Derived1 extends Base
{
    public void f1() { gvalue += 3; }
    public void f2() { gvalue += 4; }
}

class Derived2 extends Derived1
{
    public void f1(int v) { gvalue += v; }
    public void f2() { gvalue += 5; }
}

class outcome731 {
    public static void main( String[] args ) {
        Derived2 dobj = new Derived2();
        dobj.f1(10);
        Base bobj = new Derived2();
        bobj.f1();
        bobj.f2();
        System.out.println(bobj.gvalue);
    }
}
//
```

## 12 Java Producer - Consumer (outcome 8, 5 points)

In the following producer - consumer program, which functions **must** be `synchronized` so that (1) every produced item is consumed exactly once, (2) the elements are removed in the order as they are added (first-in-first-out), and (3) only produced items are consumed. The shared buffer does neither overflow nor underflow.

*Mark only the methods that* **must** *be* `synchronized`. *You will lose 1 point for each method that does not have to be* `synchronized`.

*Answer:*

`SharedBuffer.put` *and* `SharedBuffer.get`

```
class SharedBuffer
{
    private int sb_head;
    private int sb_tail;
    private int sb_numElem; // number of valid elements
    private int [] sb_element; // circular buffer
    public SharedBuffer(int size)
    {
        sb_element = new int [size];
        sb_head = 0;
        sb_tail = 0;
        sb_numElem = 0;
    }
    public void put(int v)
    {
        while (sb_numElem >= sb_element.length)
            {
                try
                    { wait(); }
                catch (InterruptedException e )
                    { System.out.println("caught exception in put"); }
            }
        sb_numElem ++;
        sb_element [sb_tail] = v;
        sb_tail = (sb_tail + 1) % sb_element.length;
        notifyAll();
    }
    public int get()
    {
        while (sb_numElem < 0)
            {
                try
                    { wait(); }
                catch (InterruptedException e )
                    { System.out.println("caught exception in get"); }
            }
        int v = sb_element [sb_head];
```

```
        sb_numElem --;
        sb_head = (sb_head + 1) % sb_element.length;
        notifyAll();
        return v;
    }
}

class Producer extends Thread
{
    private SharedBuffer p_sbuf;
    private final int p_numItem = 300;
    public Producer(SharedBuffer buf)
    { p_sbuf = buf; }
    public void run()
    {
        for (int iter = 0; iter < p_numItem; iter ++)
            { p_sbuf.put(iter); }
    }
}

class Consumer extends Thread
{
    private SharedBuffer c_sbuf;
    private final int c_numItem = 300;
    public Consumer(SharedBuffer buf)
    { c_sbuf = buf; }
    public void run()
    {
        int val;
        for (int iter = 0; iter < c_numItem; iter ++)
            { val = c_sbuf.get(); System.out.println(val); }
    }
}

public class outcome881 {
    public static void main (String[] args ) {
        // create a shared buffer
        int bufSize = 20;
        SharedBuffer sbuf = new SharedBuffer(bufSize);

        // create threads
        int numThread = 4;
        Producer [] prod = new Producer [numThread];
        Consumer [] cons = new Consumer [numThread];
        for (int index = 0; index < numThread; index ++)
            {
                prod[index] = new Producer (sbuf);
                cons[index] = new Consumer (sbuf);
            }

        // start producing and consuming
```

```
        for (int index = 0; index < numThread; index ++)
            {
                prod[index].start();
                cons[index].start();
            }

        // wait until all threads complete
        try
            {
                for (int index = 0; index < numThread; index ++)
                    {
                        prod[index].join();
                        cons[index].join();
                    }
            }
        catch (Exception je)
            { System.out.println("join exception " + je); }

    }
}
//
```

## 13 Overload Resolution (outcome 7, 10 points)

What is the output of this program?

*Answer:*

```
Y1(XB)
Y1(XB)
YB(XD2)
Y1(XB)
Y1(XB)
YB(XD2)
Y2(XB)
Y2(XB)
Y2(XD1)
Y1(int)
```

```
#include <iostream>
using namespace std;
class XBase
{
};

class XDerived1: public XBase
{
};
```

```
class XDerived2: public XDerived1
{
};

class YBase
{
public:
  virtual void f1()                       { cout << "YB()" << endl; }
  virtual void f1(double v)               { cout << "YB(double)" << endl; }
  virtual void f1(string s)               { cout << "YB(string)" << endl; }
  virtual void f1(const XBase * xo)       { cout << "YB(XB)" << endl; }
  virtual void f1(const XDerived2 * xd2) { cout << "YB(XD2)" << endl; }
};

class YDerived1: public YBase
{
public:
  virtual void f1(int v)                  { cout << "Y1(int)" << endl; }
  virtual void f1(const XBase * xo)       { cout << "Y1(XB)" << endl; }
  virtual void f1(const XDerived1 * xd1) { cout << "Y1(XD1)" << endl; }
};

class YDerived2: public YDerived1
{
public:
  virtual void f1(const XBase * xo)       { cout << "Y2(XB)" << endl; }
  virtual void f1(const XDerived1 * xd1) { cout << "Y2(XD1)" << endl; }
  virtual void f1(const XDerived2 * xd2) { cout << "Y2(XD2)" << endl; }
};

int main(int argc, char * argv[])
{
  XBase * xobb = new XDerived2;
  XBase * xobd1 = new XBase;
  XDerived2 * xobd2 = new XDerived2;

  YBase * yobb = new YDerived1;
  YBase * yobd1 = new YDerived1;
  YDerived1 * yobd2 = new YDerived2;

  yobb -> f1(xobb);
  yobb -> f1(xobd1);
  yobb -> f1(xobd2);

  delete xobd1;
  xobd1 = new XDerived1;
  yobd1 -> f1(xobb);
  yobd1 -> f1(xobd1);
  yobd1 -> f1(xobd2);
```

```
  delete xobb;
  xobb = new XBase;

  yobd2 -> f1(xobb);
  yobd2 -> f1(xobd1);
  yobd2 -> f1(xobd2);

  yobd2 -> f1(2);
  // do not worry about delete here
  return 0;
}
//
```

## 14 C++ Thread Conditions (outcome 8, 10 points)

Please add code in appropriate locations so that the account balance is **never** negative. You need to modify **multiple** locations.

*Answer:*

```
class Account {
private:
  QMutex mutex;
  QWaitCondition cond;
  int balance;
public:
  Account() { balance = 0; }
  void deposit( int dep ) {
    mutex.lock();
    balance += dep;
    cond.wakeAll();
    mutex.unlock();
  }
  void withdraw( int draw ) {
    mutex.lock();
    while ( balance < draw ) {
      cond.wait( & mutex );
    }
    balance -= draw;
    mutex.unlock();
  }
  void getBalance() {
    mutex.lock();
    cout << "balance: " << balance << endl;
    mutex.unlock();
  }
};
```

```
 1  #include <QtCore>
 2  #include <cstdlib>
 3  #include <iostream>
 4  using namespace std;
 5
 6  class Account {
 7  private:
 8    // >>>>>
 9    // add needed attributes
10    // <<<<<
11
12
13    int balance;
14  public:
15    Account() { balance = 0; }
16    void deposit( int dep ) {
17      // >>>>>
18      // add needed code
19      // <<<<<
20
21
22      balance += dep;
23
24
25
26    }
27    void withdraw( int draw ) {
28      // >>>>>
29      // add needed code
30      // <<<<<
31
32
33
34      while ( balance < draw ) {
35
36
37
38      }
39
40
41
42      balance -= draw;
43
44
45
46
47    }
48  // ----------
49  // Do not modify anything below this line
50  // ----------
51    void getBalance() {
```

```
52      mutex.lock();
53      cout << "balance: " << balance << endl;
54      mutex.unlock();
55    }
56  };
57
58  class Depositor : public QThread {
59    Account * act;
60  public:
61    Depositor (Account * a) { act = a; }
62    void run() {
63      int i = 0;
64      while ( true ) {
65        int x = (int) ( rand() % 10 );
66        act -> deposit( x );
67        if ( i++ % 100 == 0 )
68          { act -> getBalance(); }
69      }
70    }
71  };
72
73  class Withdrawer : public QThread {
74    Account * act;
75  public:
76    Withdrawer(Account * a) { act = a; }
77    void run() {
78      int i = 0;
79      while ( true ) {
80        int x = (int) ( rand() % 100 );
81        act -> withdraw( x );
82        if ( i++ % 100 == 0 )
83          { act -> getBalance(); }
84      }
85    }
86  };
87
88  int main()
89  {
90    Account act;
91    Depositor* depositors[5];
92    Withdrawer* withdrawers[5];
93
94    for ( int i=0; i < 5; i++ ) {
95      depositors[ i ] = new Depositor(& act);
96      withdrawers[ i ] = new Withdrawer(& act);
97      depositors[ i ]->start();
98      withdrawers[ i ]->start();
99    }
100   for ( int i=0; i < 5; i++ ) {
101     depositors[ i ]->wait();
102     withdrawers[ i ]->wait();
```

```
103    }
104  }
105  //
```

## 15 Overloading in Java (outcome 7, 10 points)

What is the output of this program?

*Answer:*

```
473 = 1 + 16 + 64 + 128 + 8 + 32 + 64 + 128 + 32
```

```
class Base
{
    protected static int val = 0; // <--- static
    public void f1()         { val += 1; }
    public void f1(int v)    { val += 2; }
    public void f1(double v) { val += 4; }
    public void f1(String s) { val += 8; }
    public int getVal() { return val; }
}

class Derived1 extends Base
{
    public void f1(int v)    { val += 16; }
    public void f1(double v) { f1("fall-2008"); val += 32; }
}

class Derived2 extends Derived1
{
    public void f1()         { val += 64; }
    public void f1(String s) { f1(); val += 128; }
}

class outcome781 {
    public static void main( String[] args ) {
        Base obb = new Base();
        Base obd1 = new Derived1();
        Base obd2 = new Derived2();

        obb.f1();
        obd1.f1(3);
        obd2.f1("ece462");
        obd1.f1(26.4);
        obd2.f1(12.9);
        System.out.println(obb.getVal());
    }
}
//
```

## 16 Interleaving (outcome 8, 5 points)

Consider the following two threads. Suppose each line is an *atomic* operation. What are the **possible** values of $z$? The three variables $x$, $y$, and $z$ are shared by the two threads.

*Answer:* *7, 8, 9, 10, 11*

| Thread 1 | Thread 2 |
|---|---|
| $x = 3;$ | $x = 5;$ |
| $y = x + 1;$ | $y = x + 2;$ |
| $z = y + 3;$ | $z = y + 4;$ |