ECE 462 Written Assignment 3

Due: 12:20PM, November 15, 2010

Name:

PUID:

1 Exception Handling (outcome 7)

Which statement is correct?

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

B. In Java, 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 Java, if an exception is not handled, the program terminates.

2 Java Exception (outcome 7)

What is the output of this program?

```
import java.io.*;

class Err extends Exception { }
class outcome621 {
    public static void main( String[] args )
    {
        try {
            f(0);
        } catch( Err e ) {
            System.out.println( "Exception caught" );
        }
    }
    static void f(int j) throws Err {
        System.out.println( j );
        if (j == 3) throw new Err();
        f( ++j );
    }
}
```

3 `try - catch` (outcome 7)

Which statement is correct?

A. In C++, the code inside `finally` is always executed regardless whether an exception has occurred.

B. The same function may throw different types of exceptions.

C. The code inside `catch` **cannot** throw any exception.

D. In Java, each `try` has one and only one correspoding `catch`.

E. If a function does **not** throw any exception, this function **cannot** be called inside a `try` block.

4 Java Exception Class (outcome 7)

Please write the class of `Exception74`.

```
class Exception641 // >>>>>
// fill the code for this class
// <<<<<
class outcome641 {
    static void f( ) throws Exception641 {
        throw new Exception641("thrown by f" );
    }
    public static void main( String[] args )
    {
        try {
            f();
        } catch( Exception641 exp ) {
            System.out.println( exp.getMessage() );
            // output:
            // thrown by f
        }
    }
}
```

5 Catch C++ Exception (outcome 7)

Please write the code so that the caller can `catch` the two types of exceptions.

```
#include <iostream>
#include <string>
using namespace std;
class ExceptionType1
{
private:
  string et1_message;
public:
  ExceptionType1(string m): et1_message(m) { }
  string getMessage() const { return et1_message; }
};

class ExceptionType2
{
private:
  int et2_value;
public:
  ExceptionType2(int v) { et2_value = v; }
  int getValue() const { return et2_value; }
};

void f (int i) throw (ExceptionType1, ExceptionType2)
{
  switch (i)
    {
    case 1:
      throw ExceptionType1("type 1");
      break;
    case 2:
      throw ExceptionType2(2);
      break;
    default:
      cout << "no exception" << endl;
    }
}

int main()
{
  srand(time(NULL)); // initialize the random number
  try {
    f (rand() % 3);
  }
  // >>>>>
  // catch exception
  // if it is type 1, print the message
  // if it is type 2, print the value
  // <<<<<
  return 0;
}
```

6 C++ Exception Objects (outcome 7)

What is the output of this program?

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

class ExceptionType1
{
protected:
  static int counter;
public:
  ExceptionType1() { counter ++; }
  virtual ~ExceptionType1() { counter --; }
  int getCounter()
  { return counter; }
};
int ExceptionType1::counter = 0;

class ExceptionType2: public ExceptionType1
{
public:
  ExceptionType2() { counter ++; }
};

void f (int i) throw (ExceptionType1, ExceptionType2)
{
  switch (i)
    {
    case 1:
      throw ExceptionType1();
      break;
    case 2:
      throw ExceptionType2();
      break;
    default:
      cout << "no exception" << endl;
    }
}

void g (int i) throw (ExceptionType1, ExceptionType2)
{
  try {
    f(i);
  } catch (ExceptionType1 et1) {
    throw ExceptionType2();
  }
}

int main()
{
  try {
    g(2);
    g(1);
    g(0);
  } catch (ExceptionType2 et2 ) {
    cout << "caught Type 2 " << et2.getCounter() << endl;
  } catch (ExceptionType1 et1 ) {
    cout << "caught Type 1 " << et1.getCounter() << endl;
  }
  return 0;
}
```

7 Multiple Thread (outcome 8)

Which statement is correct?

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 Timer using Thread (outcome 8)

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

```
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");
            }
    }
```

```
}
//
```

9 Interleaving (outcome 8)

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.

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

10 C++ Thread (outcome 8)

Create 4 threads; each thread adds two elements.

```
#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;
}
//
```

11 Java Thread (outcome 8)

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).

```
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( );
    }
}
//
```

12 C++ Thread Conditions (outcome 8)

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

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

class Account {
private:
  // >>>>>
  // add needed attributes
  // <<<<<


  int balance;
public:
  Account() { balance = 0; }
  void deposit( int dep ) {
    // >>>>>
    // add needed code
    // <<<<<


    balance += dep;



  }
  void withdraw( int draw ) {
    // >>>>>
    // add needed code
    // <<<<<



    while ( balance < draw ) {



    }



    balance -= draw;




  }
// ----------
// Do not modify anything below this line
// ----------
  void getBalance() {
    mutex.lock();
    cout << "balance: " << balance << endl;
    mutex.unlock();
  }
};

class Depositor : public QThread {
  Account * act;
public:
  Depositor (Account * a) { act = a; }
  void run() {
    int i = 0;
    while ( true ) {
      int x = (int) ( rand() % 10 );
      act -> deposit( x );
      if ( i++ % 100 == 0 )
        { act -> getBalance(); }
    }
  }
};

class Withdrawer : public QThread {
  Account * act;
public:
  Withdrawer(Account * a) { act = a; }
  void run() {
    int i = 0;
    while ( true ) {
      int x = (int) ( rand() % 100 );
      act -> withdraw( x );
      if ( i++ % 100 == 0 )
        { act -> getBalance(); }
    }
  }
};

int main()
{
  Account act;
  Depositor* depositors[5];
  Withdrawer* withdrawers[5];

  for ( int i=0; i < 5; i++ ) {
    depositors[ i ] = new Depositor(& act);
    withdrawers[ i ] = new Withdrawer(& act);
    depositors[ i ]->start();
    withdrawers[ i ]->start();
  }
  for ( int i=0; i < 5; i++ ) {
    depositors[ i ]->wait();
    withdrawers[ i ]->wait();
  }
}
//
```