ECE 462 Fall 2011, Third Exam

DO NOT START WORKING ON THIS UNTIL TOLD TO DO SO.

You have until 9:20 to take this exam.

Your exam should have 12 pages total (including this cover sheet). *Please let Prof. Midkiff know immediately if it does not.*

This exam is open book, open notes, but no electronics. If you have a question, please ask for clarification. If the question is not resolved, state on the test whatever assumptions you need to make to answer the question, and answer it under those assumptions. *Check the front board occasionally for corrections.*

Name:

Student ID:

Q1 (7 pts): Circle the most true of the following. **The answer in bold is the correct one.**

- In Java, a function may throw at most one type of exception.
- In Java, an integer can be thrown as an exception
- In Java, exceptions are asynchronous, and classes for objects that are thrown must extend *Thread* or implement *Runnable*
- In Java, any object can be thrown as an exception
- **In Java, classes for objects that are thrown by exceptions must extend `Exception`**
- In Java exception must be caught by the immediately surrounding try-catch block, or by the immediate caller, otherwise the program terminates.

Q2 (7 pts): Circle the most true of the following. **The ansswer in bold is the correct one.**

- C++, the code inside finally is executed only when an exception has *not* occurred.
- **The same C++ function may throw different types of exceptions.**
- In C++, the code inside a catch clause cannot throw any exception.
- In C++, each try has one and only one corresponding catch.
- In C++, a function called inside a `try` block must throw an exception.

Q3 (7 pts): What is printed by the following program:

1
Exception caught in f1
in Finally

```
class Q3 {

static void f(int j) throws Exception {
   System.out.println( j );
   if (j > 0) throw new Exception();
   f( ++j );
}

static void f1(int j) throws Exception {
   try {
      f(1);
   } catch(Exception e ) {
      System.out.println("Exception caught in f1");
   } finally {
      System.out.println("in Finally");
   }
}

public static void main( String[] args ) {
   try {
      f1(0);
   } catch(Exception e ) {
      System.out.println("Exception caught in main");
   }
}

}
```

Q4 (7 pts): Either say what is printed by the program below, or write "bad program" if it receives an error because the type of the thrown exception, and the type in the `catch` clause, are not the same.

1
Exception caught
in Finally

```
class Bad extends Exception {
   private int val;
   public Bad( ) {val = 0;}

   public Bad(int v) {
      val = v;
   }
   public void printError( ) {
      System.out.println("Error code is: "+val);
   }
}

class Q4 {

   static void f(int j) throws Bad {
      System.out.println( j );
      if (j > 0) throw new Bad();
      f( ++j );
   }

   public static void main( String[] args ) {
      try {
         f(1);
      } catch(Exception e ) {
         System.out.println("Exception caught");
      } finally {
         System.out.println("in Finally");
      }
   }
}


}
```

Q5 (7 pts): What is printed by the program below?

type 1

```
#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() {
   try {
      f (1);
   } catch (ExceptionType1 et1 ) {
     cout << et1.getMessage() << endl;
   } catch (ExceptionType2 et2 ) {
     cout << et2.getValue() << endl;
   }

   return 0;
}
```

Q6 (7 pts): Is it ever possible for the `if` expression `(f1 != f2)` to be true? Why or why not?
Yes. Consider the following sequence of statement executions, with the value of i initially *0*, and $f1 = 1$ and $f2 = 2$ in both threads *T0* and *T0*:

T0: f1 = I.i; // T0's f1 = 0
T1: f1 = I.i; // T1's f1 = 0
T1: f2 = I.i; // T1's f2 = 0
T1: I.i++; // shared I.i = 1
T0: f2 = I.i // T0's f2 = 1

This is possible because there is no synchronization and the operations in T0 and T1 can be *interleaved*.

```
class T0 extends Thread {
   int f1;
   int f2;
   T0() {
      f1 = 1;
      f2 = 2;
   }

   public void run( ) {
      System.out.println("starting a T0");
      f1 = I.i;
      f2 = I.i;
      I.i++;
      if (f1 != f2) {System.out.println("what's going on???");}
   }
}

class I {
   public static int i;
   I(int a) {i = a;}

   public static void main( String[] args ) {
      T0 t0 = new T0( );
      T0 t1 = new T0( );
      t0.start( ); // run CHANGED to start
      t1.start( ); // run CHANGED to start
   }
}
```

Q7 (7 pts): Is it ever possible for the `if` expression `(f1 != f2)` to be true? Why or why not?
Yes. Even though the `run` methods are `synchronized`, they are not synchronized on the same object. The `run` method in thread `T0` is syncrhonized on the *T0* object, and the run mehtods on `T1` is synchronized on the `T1` object. Thus the statements in the `run` methods on the two threads can still be interleaved as shown in **Q6**.

```
class T0 extends Thread {
   int f1;
   int f2;
   T0() {
      f1 = 1;
      f2 = 2;
   }

   public synchronized void run( ) {
      System.out.println("starting a T0");
      f1 = I.i;
      f2 = I.i;
      I.i++;
      if (f1 != f2) {System.out.println("what's going on???");}
   }
}

class T1 extends Thread {
   int f1;
   int f2;
   T1() {
      f1 = 1;
      f2 = 2;
   }

   public synchronized void run( ) {
      System.out.println("starting a T1");
      f1 = I.i;
      f2 = I.i;
      I.i++;
      if (f1 != f2) {System.out.println("what's going on???");}
   }
}

class I {
   public static int i;
   I(int a) {i = a;}

   public static void main( String[] args ) {
      T0 t0 = new T0( );
      T1 t1 = new T1( );
      t0.start( ); // run CHANGED to start
      t1.start( ); // run CHANGED to start
   }
}
```

Q8 (7 pts): Is it ever possible for the `if` expression `(f1 != f2)` to be true? Why or why not?
No. The operations in the run methods are both synchronized on a single object, the object referenced by the static reference `I.s`. Thus the statements of the two `run` methods cannot be interleaved, and once a `run` method begins executing the synchronized block containing the reads of `I.i` it is not possible for another thread to change the value of `I.i` until the synchronized block finishes. Thus both `f1` and `f2` in the synchronized block get the same value.

```
class T0 extends Thread {
   int f1;
   int f2;
   T0() {
      f1 = 1;
      f2 = 2;
   }
   public void run( ) {
      System.out.println("starting a T0");
      synchronized(I.s) {
         f1 = I.i;
         f2 = I.i;
         I.i++;
         if (f1 != f2) {System.out.println("what's going on???");}
      }
   }
}
class T1 extends Thread {
   int f1;
   int f2;
   T1() {
      f1 = 1;
      f2 = 2;
   }
   public void run( ) {
      System.out.println("starting a T1");
      synchronized(I.s) {
         f1 = I.i;
         f2 = I.i;
         I.i++;
         if (f1 != f2) {System.out.println("what's going on???");}
      }
   }
}
class I {
   public static int i;
   public static Object s = new Object( );
   I(int a) {i = a;}

   public static void main( String[] args ) {
      T0 t0 = new T0( );
      T1 t1 = new T1( );
      t0.start( ); // run CHANGED to start
      t1.start( ); // run CHANGED to start
   }
}
```

Q9 (7 pts): Circle the most correct statement about the program below. **The correct answer is shown in bold.**

- In `main`, the statements `d->i = 4;` and `b->i = 4;` are legal because `i` is declared public in `B`
- **In `main`, the statement `d->i = 4;` is illegal because `D` inherits privately from `B`, and therefore the `i` field is private when accessed through the `D` object. The statement `b->i = 4;` is legal, however.**
- In `main`, both the statements `d->i = 4;` and `b->i = 4;` are illegal because the `private` inheritance of `B` by `D` makes `i` private regardless of how it is accessed.
- In `main`, both the statements `d->i = 4;` and `b->i = 4;` are legal because the `private` inheritance of `B` by `D` makes `i` private only to accesses within the declaration and definition of `D`

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

class B {
public:
   int i;
   B( ) {i=0;}
};

class D : private B {
public:
   int j;
   D( ) : B( ) {j=1;}
};

int main(int argc, char * argv[ ]) {
   D* d = new D( );
   B* b = new B( );
   b->i = 4;
   d->i = 4;
}
```

Q10 (7 pts): Circle the correct answer about the program below, given the `protected` inheritance of `B` by `D1`. **Again, the correct answer is shown in bold.**

- The access of `i` in `D2::foo` is legal, and the access of `i` in `main` is legal
- **The access of `i` in `D2::foo` is legal, and the access of `i` in `main` is illegal**
- The access of `i` in `D2::foo` is illegal, and the access of `i` in `main` is legal
- The access of `i` in `D2::foo` is illegal, and the access of `i` in `main` is illegal

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

class B {
public:
   int i;
   B( ) {i=0;}
};

class D1 : protected B {
public:
   int j;
   D1( ) : B( ) {j=1;}
};

class D2 : public D1 {
public:
   D2( ) : D1( ) { }
   void foo( ) {i = 20;}
};

int main(int argc, char * argv[ ]) {
   D2* d2 = new D2( );
   d2->i = 4;
}
```

Q11 (7 pts): Which of the following is *not* a reason to do threading (circle the correct response.) **As before, the correct answer is shown in bold.**

- Even on a single core threads allow the appearance of progress across different pieces of work
- **Threads allow encapsulation of data**
- Threads allow better performance when multiple cores are available
- Threads allow a convenient way for a program to handle asynchronous events like mouse clicks

Q12 (7 pts): The declaration of `foo` in class `D` below gives an error.Why?
The declaration in `D` attempts to *override*, that is redefine, the definition in `B`, which is prohibited by the final declaration.

```
class B {
   public B( ) { }
   final public int foo(int i) {
     return i++;
   }
}
class D extends B {
   public D( ) { }
   final public int foo(int i) {
      return i--;
   }
   public static void main( String[] args ) {
      D d = new D( );
      d.foo(57);
   }
}
```

Q13 (7 pts): The declaration of `foo` in class `D` below does not give an error, unlike the declaration in **Q12**. Why is this?
The declaration in `D` *overloads* the definition, that is it declares a function with a different function for te name `foo`. Because this declares a different function, it is not prohibited by the `final` declaration.

```
import java.io.*;

class B {
   public B( ) { }
   final public int foo(int i) {
     return i++;
   }
}

class D extends B {
   public D( ) { }
   final public int foo(float i) {
      return (int) i-1;
   }

   public static void main( String[] args ) {
      D d = new D( );
      d.foo(57);
   }
}
```

Q14 (1 pt): What is Prof. Midkiff's favorite color?

Most days is it blue, but 1 point was given for any answer.
Q15 (8 pts): What object oriented concepts do you still not understand? (this will not affect future test questions, and as long as you answer something it will not affect your grade. This is just to motivate you to let me know what isn't clear. *Any answer received 8 points.*