Write the answers to question 1 and 2 on this sheet. Write the other answers in the exam booklet. There are four pages to this exam!

1. True/False. Please circle the correct answer on this sheet.

(a) F Two instances of the same C++ template class are subclasses of each other.
(b) T An object whose class implements the MouseListener interface in Java can be used to handle an event generated by clicking a mouse button when the cursor is on a Frame.
(c) ? A pregnant chad and a dimpled chad are the same thing.
(d) F If there is a finally clause in a Java try block, no other catch clause will be executed.
(e) T The output of the C++ code
   ```
   int x = 7; int &y = x; x++; cout << y;
   ```
   will be 8.
(f) F In Java, if you want to throw an int as an exception, you can simply declare
   ```
   class int extends Exception;
   ```
   and then say, for example, throw(6).
(g) T Public derivation is the only form of derivation in C++ that provides inheritance with subtyping.
(h) T In Java, a parent class is referred to in the child class using super. In C++, this is not the case.
(i) F In C++, a declaration of the form
   ```
   A s1(s2);
   ```
   where A is a user-defined class and s2 is an object of that class, always creates a new object s1 that is identical to s2.
(j) T The closest thing that Java has to a C++ namespace is a package.

2. Multiple Choice. Circle the one correct answer.

(a) Given the C++ declaration
   ```
   class A {public: int x; protected: int y;};
   ```
   Which of the following would be rejected by the compiler?
   i. class B: public A { void f() { x = y; }};
   ii. class B { void f() { A a; a.x = a.y; }};
   iii. class B: public A{}; class C: public B { void f() { x = y; }};
   iv. None of the above
   (b) For the Java code
   ```
   f.addActionListener(new A() {
       void actionPerformed(ActionEvent e) { System.out.println("yup"); 
   }});
   ```
   to be correct, where f is a Frame, which of the following must be true?
   i. A is a class implementing the ActionListener interface.
   ii. A is a class implementing the MouseListener interface.
   iii. A is an interface that extends the ActionListener interface.
   iv. None of the above
(c) The output of the C++ code,

```cpp
class A { public: int f(int x) { cout << x << " "; }};
class B: public A { public: int f(int y) { A::f(y+1); }};
void g(A a, B b) { a.f(3); b.f(3); }
int main() { B p; B q; g(p,q); }
```

would be

i. 3 3
ii. 3 4
iii. 4 4
iv. None of the above

(d) Given the C++ code

```cpp
class A { friend class B; ... }
```

which of the following statements is true?

i. The friendship declaration, above, has the same effect as making B a subclass of A, since B now can access A’s protected members.
ii. If B is already a subclass of A, the friendship declaration has no effect.
iii. The friend declaration gives B greater access to A’s members than if B were a subclass of A.
iv. None of the above

(e) Which of the following is not a correct creation of a Java thread?

i. new Thread();
ii. new Thread(new Runnable() { public void run() {} });
iii. new Thread() { public void run() {} };
iv. new Runnable() { public void run() {} };

(f) The signature of the standard << operator for printing an integer is

i. `ostream & operator<<(ostream &, const int)`
ii. `ostream & ostream::operator<<(ostream &, const int)`
iii. `int & operator<<(ostream &, const int)`
iv. `int int:: & operator<<(ostream &, const int)`

(g) Given the Java declarations

```java
interface I { void foo(); }
```

and

```java
class B extends A implements I { ... }
```

the following statement is true:

i. Class B must provide a definition for foo(), no matter how class A is defined.
ii. Class B need only provide a definition of foo() if A does not.
iii. Class B need only provide a definition of foo() if A does not implement I.
iv. Class B inherits foo() from I, thus B does not have to provide a definition of foo().

(h) Given the C++ declaration

```cpp
template<class T>
class set { ... }
```

which of the following declarations (outside of the template) could not be correct?

i. set s;
ii. set<int> s;
iii. set<float> s;
iv. set< set<int> > s;
(i) Which of the following statements is true?
   i. Neither C++ or Java arrays are objects.
   ii. Both C++ and Java arrays are objects.
   iii. C++ arrays are objects but Java arrays are not.
   iv. Java arrays are objects but C++ arrays are not.

(j) Which of the following declarations could be correct in Java (where ... represents some code)?
   i. class C extends A, B { ... }
   ii. interface C implements A, B { ... }
   iii. interface C extends A, B { ... }
   iv. class A {...} class B {...} class C implements A, B {...}

3. (a) Novice Java programmers often write code similar to,
   ```java
   class C { public int x; ... }
   ...
   C[] a = new C[10];
   for(int i = 0; i < a.length; i++)
     a[i].x = i;
   ```
   What is the problem with this code? Fix it.

   The problem is that although the array object was created, none of the elements of the array have been initialized to refer to objects of class C. To fix it, the loop should be:
   ```java
   for(int i = 0; i < a.length; i++) {
     a[i] = new C();
     a[i].x = i;
   }
   ```

(b) Other novice Java programmers might write,
   ```java
   class C {
     int f() {
       return System.in.read();
     }
   }
   ```
   and get a compiler error that says something about System.in.read() throwing an exception. What is the problem? Fix it.

   Since System.in.read() might throw an exception, either a try block is required to catch the exception or f() has to be declared as throwing an exception. Thus, either the body of f() should be
   ```java
   try {
     return System.in.read();
   } catch (Exception e) { return 0; } //some int value must be returned
   ```
   or the declaration of f() should be
   ```java
   int f() throws Exception {
   }
   ```

(c) Given the C++ code
   ```cpp
   class A {
   public:
     A() { a = new int[3]; for(int i=0; i<3; i++) a[i] = i; }
     ~A() { delete[] a; }
   private:
     int *a;
   }

   void init(A &x) //sets elements of x.a to to 0,1,2
   ```
you might notice that as the program runs following the call init(p), the values stored in p.a start changing unpredictably. What might be happening? Fix this problem.

The problem is that the assignment in init(), namely x = y, uses the default copy assignment which simply copies each field from y into the corresponding field of x. This means that the address stored in y.a is simply copied into x.a, without the array being copied. At the end of init(), the destructor is called for the local variable y, which causes the array to be deleted. Thus x is left pointing to an array that is being reclaimed for other use by the program, and those locations in memory may be written to during the computation. Since p is passed to init() by reference, p is modified by init() so that p.a refers to the deleted array.

To fix it, the copy assignment operator should be redefined in A so that the elements of the array are copied, not the array pointer.

```cpp
class A {
public:
    ...
    A &operator=(const A &other) {
        for(i=0; i<3; i++) a[i] = other.a[i];
    }
    ...
}
```

4. Brief answers, please.

(a) What is a virtual member function in C++?
It is a member function whose calls are dynamically dispatched. That is, given the declaration of the form

```cpp
class C {
    ...
    virtual .. f(..) ..
    ...
}
```

and a call of the form

```cpp
... x.f(...) ...
```

where the variable x is declared to be of type C, the version of f that is executed depends on the actual type of the object that x refers to, not necessarily the declared type of x.

(b) What happens if you leave the virtual keyword off the declaration of a member function? Is there an equivalent of a non-virtual member function in Java?
If you leave the keyword virtual off, then no dynamic dispatching of the member function occurs. In the example above, no matter what the actual type of the object that x refers to is, the version of f that is called is the one defined in the class C.

(c) Write the simplest code you can think of in C++ that fully demonstrates the difference between virtual and non-virtual member functions.
**Answer:** The program,

```cpp
class A {
public:
    void f() { cout << "A's f\n"; }
    virtual void g() { cout << "A's g\n"; }
}

class B: public A {
public:
    void f() { cout << "B's f\n"; }
    virtual void g() { cout << "B's g\n"; }
}

void h(A &x)
{x.f();
 x.g();
}

typedef int main()
{ B b;
  h(a);
}
```

would print

```
A's f
B's g
```

5. In Java, write a single class called `Ouch` which, when an `Ouch` object is created, draws a frame on the screen that contains a button. When the button is clicked upon, “ouch” is printed on the standard output (i.e. using `System.out.println`). Be sure not to define any other class (either named or anonymous).

If you need to know about particular item in the Java API, feel free to ask.

There are many ways to do this. Here’s one way.

```java
class Ouch extends Frame implements ActionListener {
    Ouch() {
        super("Ouch Frame");
        setSize(200,200);
        Button b = new Button("ouch");
        b.addActionListener(this);
        add(b);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Ouch");
    }
}
```

6. (a) Turn the C++ definition

```cpp
int sum(int a, int b, int c)
{ return a+b+c;
}
```

into a function template that can be used to work on any type that supports `+`, instead of just `int`. Be sure that the template will work for objects that implement the `+` operator, so that dynamic dispatching of `+` will occur where possible.
Answer:

```cpp
template<class T>
T sum(T &a, T &b, T &c)
{ return a+b+c;
}
```

Note the importance of the reference parameters, otherwise dynamic dispatching will not occur.

(b) Define a simple C++ class template `myobj`, parameterized by a single type, such that the following code will work:

```cpp
myobj<int> s1, s2;
myobj<int> s3 = sum(s1,s2,s2);
```

Make the `myobj` template as simple as you like.

Answer:

```cpp
template<class T>
class myobj {
public:
    myobj() { cin >> val; } //not really necessary

    myobj operator+(myobj &m) {
        myobj r(*this);
        r.val = r.val + m.val;
        return r;
    }

protected:
    T val;
}
```

7. The Java API defines the `Comparable` interface as

```java
interface Comparable {
    int compareTo(Object);
}
```

where `x.compareTo(y)` should return -1 if `x` is less than `y`, 0 if they are equal, and 1 otherwise.

Define a class `MyArray` that implements `Comparable` and whose objects behave like integer arrays. `MyArray`s should be compared based on the sum of their elements. For example,

```java
int[] a = new int[] {1,2,3,4}; //create an array and initialize the elements
int[] b = new int[] {-1,2,-3,4,-5};
MyArray m1 = new MyArray(a); //the elements of m1 are those of a
MyArray m2 = new MyArray(b); //the elements of m1 are those of b
System.out.println(m1.compareTo(m2)); //prints 1, since 1+2+3+4 > -1+2-3+4-5
```

Define just enough of the `MyArray` class for the above code to work.
class MyArray implements Comparable {
    MyArray(int[] b) { a = b; }

    public int compareTo(Object o) {
        MyArray other = (MyArray) o;
        int mysum = 0;
        int othersum = 0;
        for(int i=0; i<a.length; i++) mysum = mysum + a[i];
        for(int i=0; i<other.a.length; i++) othersum = othersum + other.a[i];
        if (mysum < othersum) return -1;
        else if (mysum == othersum) return 0;
        else return 1;
    }

    private int[] a;
}