Final Exam

1. Multiple Choice. Please circle the one correct completion of each sentence.

(a) Java, in common lingo, refers to
   i. chocolate.
   ii. coffee.
   iii. tea.
   iv. sushi.

(b) Java doesn’t need the delete operator because
   i. the user can write their own destructors.
   ii. there’s no dynamic allocation of objects via the new operator.
   iii. it has garbage collection.
   iv. all of the above.

(c) Given the C++ function definition void f(A x) { x.g() }, and the function call f(y) where y is an object of class A,
   i. f() must be a member function of class A.
   ii. a copy of y is made when f() is called.
   iii. y is also the implicit this object in f().
   iv. none of the above.

(d) In C++, even though operator<< can be overloaded to print an object of class A, operator<< is not usually made a member function of A because
   i. operator<< should not access any of A’s private members.
   ii. operator<< is already a public member of the ostream class.
   iii. the ostream object usually appears to the left of << in a statement that prints something.
   iv. operator<< is already a private member of the ostream class.

(e) The following statement is false:
   i. A Java function that explicitly throws an exception must use the throws ... declaration, but a C++ function does not have to.
   ii. A C++ function can throw an object of any type, but a Java function can only throw objects of a subclass of Exception.
   iii. C++ has a special catch-all clause, catch(...), to catch all exceptions, but Java does not need a special version of catch to accomplish this.
   iv. Java programs will not abort if an exception is not caught, but C++ programs will.

(f) Subtyping refers to
   i. The ability in C++ to treat an object of a derived class as if it were an object of the base class.
   ii. The ability in C++ to instantiate a class template in many different ways.
   iii. The ability in Java to pass an object of any class implementing the runnable() interface to a constructor of the Thread class.
   iv. None of the above.

(g) A Java interface, declared by interface I { ... }, is used to
   i. export the components of a Java package.
   ii. specify of the methods that a class must provide.
   iii. describe the interaction between Java and the host machine’s window system.
   iv. describe all the packages and classes that are provided with the JDK1.2 implementation.
Double-buffering of animated images in Java graphics programs is accomplished by
- creating doubly-linked lists in order to traverse lists of images in both directions
- having a thread that repeatedly writes an image to the screen and then sleeps.
- using `paint()` to change an image in memory, then copying the changed image to the frame or applet.
- All of the above.

C++ function templates
- must be explicitly instantiated.
- may be explicitly instantiated.
- cannot be explicitly instantiated.
- are not part of the C++ language.

Reference types (i.e. those using `&` in C++) are not used in Java because
- variables that refer to objects are implicitly pointers in Java.
- the Java designers decided that aliasing between variables (i.e. two variables referring to the same object) makes programs too complicated.
- the Java designers decided that explicit C-style pointers should be used instead in Java.
- there are other ways in Java to specify whether the parameters to a function are pass-by-value or pass-by-reference.

In the object oriented programming community, there is disagreement over whether multiple inheritance is truly necessary. Java proponents claim that Java interfaces provide the benefits of C++’s multiple inheritance without the complexity.

- Give an example, showing both C++ and Java code, that demonstrates the similarity between multiple inheritance and interfaces.

Multiple inheritance in C++ allows the type of an object to be considered the subtype to two or more classes. For example, given

```java
class C: public A, B { ... }
```

an object of class C can be passed as a parameter to both of the following functions:

```java
void f(A &x) { ... }
void g(B &y) { ... }
```

The use of interfaces in Java provides a similar facility. If a class is defined as

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

then object of class C can be passed as a parameter to both of the following functions:

```java
void f(A x) { ... }
void g(B y) { ... }
```

- There is also substantial dissimilarity between multiple inheritance and interfaces. Give an example, again showing both C++ and Java code, that demonstrates this dissimilarity.

Although Java interfaces provide a similar subtyping behavior to C++’s multiple inheritance, as described above, interfaces do not support the inheritance of data members and of the code for methods from multiple parent classes. In C++, writing

```java
class C: public A, B { ... }
```

means that C has inherited the code and data members from both class A and class B. However, by writing
class C extends A implements B { ... }

In Java, one must still explicitly provide the code for the methods required by interface B within the body of class C.

3. To build a Graphical User Interface (GUI) in Java, the following items are used:
   - Listener interfaces
   - Event objects
   - Addlistener methods
   - Adapter classes

Briefly describe what each of these things are and how they interact in a GUI.

4. I’m having trouble getting my C++ program, below, to compile and run correctly.

```cpp
#include <iostream.h>

class A {
    A(): y(7) {}
    void f(int x) { cout << "x = " << x <<", y = " << y << endl; }
    int y;
};

class B: public A {
    B(int w): z(w), A(4) {}  
    void f(int x) { cout << "x = " << x <<", y = " << y <<", z = " << z << endl; }
    int z;
};

void g(A c)
{ c.f(7); }

int main()
{ B b(13);
  g(b);
}
```

I want my main() procedure,

```cpp
int main()
{ B b(13);
  g(b);
}
```

to print out

\[
    x = 7, \ y = 4, \ z = 13
\]

Without modifying main(), fix the code above (on the exam sheet, if you like) so that the program will compile and run as I expect. Be sure that, in good object oriented style, no class member is more widely visible than it needs to be.

The problems with the code are:

(a) In class A, the constructor and the f() method are private (since private is the default visibility), thus no object of class A can be created nor can the f() method be called outside of the class. Similar, the B constructor and definition of f() should be made public. Furthermore, since A::y is accessed in B, it should be made protected.

(b) In A, no constructor taking an integer parameter is defined. Thus, the initialization “A(4)” in B::B() will not work. Such a constructor for A needs to be defined.

(c) In order to get the dynamic dispatch (i.e. overriding) behavior desired, method A::f() should be made virtual and the parameter to procedure g should be a reference parameter.
Thus, the resulting code should be:

```cpp
#include <iostream.h>

class A {
public:
    A(): y(7) {}
    A(int x): y(x) {}
    virtual void f(int x) { cout << "x = " << x << ", y = " << y << endl; }
protected:
    int y;
};

class B: public A {
public:
    B(int w): z(w), A(4) {}
    void f(int x) { cout << "x = " << x << ", y = " << y << ", z = " << z << endl ; }
protected:
    int z;
};

void g(A &c)
{ c.f(7);
}

int main()
{ B b13);
    g(b);
}

5. (a) Write a simple C++ class `A` that overloads its `operator==( )` method such that the following procedure will work:

```cpp
void f(A &p, A &q)
{
    if (p==q)
        cout << "yes\n";
    else
        cout << "No\n";
}
```

**Answer:**

```cpp
class A {
public:
    int operator==(const A &other) { return 1; }
};
```

(b) Suppose class `B` was derived from class `A` such that `B` defined an `operator==( )` to compare two objects of class `B`, as in

```cpp
class B: public A {
    bool operator==(B &x);
}
```

Note, though, that in,

```cpp
int main()
{ A a1;
    A a2;
    B b1;
```
B b2;
f(a1,a2); //uses A's == operator
f(b1,b2); //also uses A's == operator
}

It would not be the case that B's operator==() would be used to compare b1 and b2 within f(). Explain why not.

In a child class, in order to override a method from the parent class, the signature of the method (i.e. parameter and return types) must be identical in the child and the parent classes. In this case, B's operator== is defined as taking a B parameter, but A's operator== is defined as taking an A parameter. Thus, this is not an overriding and so the occurrence of == in procedure f() is always A's operator==.

(c) How would the program have to be modified, without changing main(), so that the call to f(b1,b2) in main() would use B's operator==()? Be sure to show the code.

This can be accomplished by writing f() as a function template, as follows:

```cpp
template<class T>
void f(T &p, T &q)
{
    if (p==q)
        cout << "yes\n";
    else
        cout << "No\n";
}
```

(d) The same situation described in part (b) is true in Java, except that the equality method is named equals() instead of operator==(). Thus, the original f() would look like

```java
void f(A p, A q)
{
    if (p.equals(q))
        System.out.println("yes");
    else
        System.out.println("No");
}
```

Answer the same question as in part (c), but for Java.

In Java, this can be accomplished by defining the equals() method in B to take a parameter of class A, then cast the parameter, if appropriate, to a B:

```java
class B extends A {
    
    bool equals(A x) {
        if (x instanceof B) {
            B y = (B) x;
            ... // code for computing equals between two B's
        }
        else
            super.equals(x);
    }
}
```