1 Inheritance

[10 pts] In several OO languages, including C++ and Eiffel, a subclass can hide members of the superclass. In C++, for example, we can declare a base class to be public, protected, or private:

```cpp
class B : public A {
    // public members of A are public members of B
    // protected members of A are protected members of B
}
class C : protected A {
    // public and protected members of A are protected members of C
}
class D : private A {
    // public and protected members of A are private members of D
}
```

In all cases, private members of A are inaccessible to methods of B, C, or D.

Consider the impact of protected and private superclasses on dynamic method binding. Under what circumstances can a reference to an object of class B, C, or D be assigned into a variable of type A*. Should B*, C*, or D* be considered subtypes of A*?
2 Object implementation

[10 pts] Consider the Java program below:

```java
interface Pingable {
    public void ping();
}

class Counter implements Pingable {
    int count = 0;

    public void ping() {
        ++count;
    }

    public int val() {
        return count;
    }
}

public class Ping {
    public static void main(String[] args) {
        Counter c = new Counter(); // (a)
        c.ping();
        c.ping();
        int v = c.val();
        System.out.println(v);
    }
}
```

Assume this is compiled onto a machine with 4-byte addresses.

(a) Draw a picture of the layout in memory of the `Counter` object created in `main` and labeled (a).

(b) Give assembly-level pseudocode for the call to `c.val` in `main`. Assume the address of `c` is in register `r1` immediately before the call and the same register is used to pass the hidden `this` parameter. Ignore details like saving and restoring registers or where to put the return value. Assume there are instructions for loading and storing from an address with a given offset, for arithmetic, for calling procedures.

(c) Give assembly-level pseudocode for the body of the method `Counter.ping` (again ignoring register saves and restores).
3 Reflection

[10 pts] Write a Java class Reflect that uses the reflection API (i.e., java.lang.Class, java.lang.reflect.*) as follows.

(a) Declare a non-static method make that takes the name of a class (a String) as parameter, loads the class, creates an instance of the class, and returns the new instance. Assume the default (zero-argument) constructor is used.

(b) Declare a non-static method invoke that takes the name of a method, the receiver (an Object), an Object[] of arguments to pass to the method, and then invokes it, returning the result as an Object. If an error occurs, exit or throw a RuntimeException.

(c) There should be no other members of Reflect.

For example, one should be able to use your class as follows:

```java
Reflect r = new Reflect();
Object r0 = r.make("java.lang.String");
Object r1 = r.invoke("concat", r0, new Object[] { "hello" });
Object r2 = r.invoke("concat", r1, new Object[] { ", world" });
System.out.println(r2);
```