One of the points in [Snyder86] is that instance variables of a class should not all be visible to its subclasses (i.e., not "child accesses all"). Give two examples, one in Modula-3 and one in Java of classes where this is enforced (i.e., the child cannot access all the instance variables of its superclass). (10 points per example. Total: 20)

Java:
```java
class bag {
    private int contents[];
    protected void add_to_contents(int toadd);
    public void insert(int toinsert) {
        add_to_contents(toinsert);
    }
    public boolean lookup(int tofind) {...}
}

class set: bag {
    public void insert(int toinsert) {
        if (!lookup(toinsert))
            add_to_contents(toinsert);
    }
}

class set cannot access contents except through the exported interface of bag (namely add_to_contents).
```

Modula-3
```modula-3
TYPE ForPublic = OBJECT
  METHODS
    insert(toadd: INTEGER);
    lookup(tofind: INTEGER): BOOLEAN;
END;

TYPE Bag <: ForPublic;

To the public, only insert and lookup are available.

Now here is the interface to be exported to the children:
TYPE ForChildren = ForPublic OBJECT
  METHODS
    add_to_contents(toadd: INTEGER);
END;
REVEAL Bag <: ForChildren;

Children of Bag can inherit as follows:

TYPE Set = Bag OBJECT
  OVERRIDES
    insert(toadd: INTEGER) ...
END;

In some private place, there must be a full revelation of Bag:
REVEAL Bag = ForPublic BRANDED OBJECT
  contents: ARRAY OF INTEGER;
END;

Don't worry about "BRANDED"...
2. [Snyder86] distinguishes two kinds of inheritance: inheritance for subtypes and inheritance for reuse. Indicate which category the following languages fall into: (a) Smalltalk (b) Java (c) Modula-3. If a language does not cleanly fit into a single category, indicate that. (5 points per language. Total: 15)

Smalltalk: Inheritance for reuse
Java: interface inheritance is inheritance for subtyping. class inheritance is inheritance for subtyping and reuse
Modula-3: inheritance for subtyping and reuse

3. Consider the following class hierarchy in a single inheritance language. Assume all methods are virtual: (total points: 20)

```c
class T { f() {...} 
    g() {...} 
};
class S: public T {g() {...} }
```

Answer the following questions:
(a) Give the virtual function table for each of the two classes. (10 points)
(b) Give the steps involved in invoking the "g" method on a variable "o" declared to point to S (i.e., S *o) (10 points)

I have an almost identical example to this in my lecture notes.

4. In a multiple inheritance language (such as C++) we noted in class (and in the Stroustrup paper) that one needs to put "deltas" in the virtual function table along with pointers to functions. Explain why the "deltas" are needed in the virtual function table. Illustrate with an example. (20 points)

Consider the following classes:

```c
class C: A, B {... }
and
class D: E, C {...}
```

Assume that class B has a virtual m that is not overridden in C or D.
Let's say we have a variable C *c;
If we invoke the "m" method on c:
c->m();

The problem is that the "m" method expects the "this" pointer to be a reference to a B object. Now, D is a subclass of C, so 'c' could point to either a C object or a D object. In a C object, the offset of the B part is different from the offset of the B part in a D object. Thus, to do the above method invocation, we need to adjust the "this" pointer by different amounts depending on whether c points to a C object or a D object. Since this amount is potentially varying and not known at compile time, Stroustrup proposes putting this amount in the virtual function table in the entry corresponding to "m".