Exercise 6.1: Method Calls

Given are the following declarations and definitions in the object-oriented programming language Xava:

```java
class C {
    void m (void) { ... };
    void m1 (int p) { ... };
    void m2 (void) { ... };
};

class C1 extends C {
    void m (int p) { ... };
    void m1 (int p) { ... };
    int m2 (void) { ... };
};

class C2 extends C {
    void m (void) { ... };
    void m1 (void) { ... };
    void m3 (int p) { ... };
};

class C12 extends C1 {
    void m2 (void) { ... };
    void m3 (int p) { ... };
};
```

a. What notation do you expect for invoking a method in the language Xava? Is any other notation possible?

b. Suppose, the language uses prefix notation for method invocation (i.e., `o.m()` and not `m(o)`). Discuss the circumstances under which method calls are legal and what methods are in fact invoked (assume static name-binding as in Java or C++ with dispatching calls).

```java
C oc;
C12 oc12;

oc = oc12;
oc.m3(1);
oc.m1(1);
int x = oc.m2();
```
c. Which methods are redefined in classes? Which methods do you expect to be overloaded? Explain why?

Exercise 6.2: Classes in C++

The following C++ program computes prime numbers. What algorithm is used in the program? Describe the class hierarchy of the program. Explain the rules for accessing class components in C++.

```cpp
#include <iostream>

class Node {
  public:
    Node *input;
    Node (Node *s) { input = s; }
    virtual int displayValue() { return 0; }
};

class Generator : public Node {
  int value;
  public:
    int displayValue(void) { return value++; }
    Generator(int initialVal) : Node(0) { value = initialVal; }
};

class Filter : public Node {
  int factor;
  public:
    int displayValue();
    Filter (Node *s, int f) : Node(s) { factor = f; }
};

int Filter::displayValue() {
  int n;
  do {
    n = input->displayValue();
  } while (!(n % factor));
  return n;
}

int main(void) {
  Generator gen(2);
  Node *sieve = &gen;
  int prime, max;
  std::cout << "Prime number in C++\n\nHow many prime numbers?";
  std::cin >> max;
  for (int i = 0; i < max; i++) {
    prime = sieve->displayValue();
    std::cout << prime << " \n";
    sieve = new Filter(sieve, prime);
  }
  std::cout << " \n";
  return 0;
}
```
Exercise 6.3: Multiple Inheritance in C++

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

class A {
    int w;
    public: virtual void m1() { cout << "A::m1" << endl; m2(); }
    virtual void m2() { cout << "A::m2" << endl; }
};

class B {
    protected: int x;
    public: virtual void m3() = 0;
    virtual void m4() { cout << "B::m4" << endl; m3(); x=5; }
};

class C : public A, public B {
    int y;
    public: virtual void m2() { cout << "C::m2" << endl; y=2; }
    virtual void m3() { cout << "C::m3" << endl; x=3; }
};

class D : public C {
    int z;
    public: virtual void m3() { cout << "D::m3" << endl; m1(); }
    virtual void m5() { cout << "D::m5" << endl; z=4; }
};

int main() {
    A* a = new A;
    B* b = new C;
    C* c = new C;
    D* d = new D;

    ...
}
```

(i) Variables of which of the above types can be assigned to each other?
(ii) Which of the method calls are allowed in case of static name binding?
(iii) What is the output of the legal method calls if a, b, c and d are left unchanged?

Exercise 6.4: Multiple Inheritance and Abstract Classes

(i) What are the possible problems when using multiple inheritance? How can they be solved?
(ii) What is the advantage of abstract interfaces in the form offered by Java?
(iii) Why do many programming languages often not allow instantiation of abstract classes? What is the possible solution if it is allowed?