Exercise 5

Multiple Inheritance and Traits

1) Consider the following C++ code:

```
class Person
{
     Person *spouse;
     string name;

public:
     Person (string n) { name = n; spouse = NULL; }

     bool marry (Person *p)
     {
          if (p == this) return false;
          spouse = p;
          if (p) p->spouse = this;
          return true;
     }

     Person *getSpouse () { return spouse; }
     string getName () { return name; }
 };
```

The method `marry` is supposed to ensure that a person cannot marry itself. Without changing the code above, create a new object that belongs to a subclass of `Person` and marry it with itself. Hint: use multiple inheritance. Explain exactly what happens.

2) Consider the following C++ code:

```
class Person
{
     bool likesDiamonds;

public:
     Person (bool l) { likesDiamonds = l; }
};

class Programmer : virtual public Person
{
public: Programmer () : Person (false) {}
     // diamonds are a programmer's worst enemy
};
```

It is expected that `!likesDiamonds` is an invariant in class `Programmer`. Use virtual inheritance to break this invariant, without altering the above code.

3) Why doesn't C++ allow the following?

```
class C : public D, public D {…}
```

4) Write three classes

- A normal queue class `Queue`
- A subclass of `Queue` that maintains the sum of all items in the queue, using the `enqueue` and `dequeue` methods
- A subclass of `Queue` that maintains the size of the queue, using the `enqueue` and `dequeue` methods

We now want a class that supports both functionalities.

- Suppose that we want to use multiple inheritance to do that. We want to override the enqueue and dequeue methods of the new class, such that the new methods call the enqueue and dequeue methods of both the old classes. Are there any problems with this approach?
- How do we attack the problem using traits? Does this fix the above-mentioned problems?

5) Find an example of a class `C` and two traits `A` and `B`, such that `C with A with B` behaves differently from `C with B with A`. You may adapt your solution for Q.2 to traits.

6) Consider the following declarations in Scala:

```
class C
trait T extends C
trait U extends C
class D extends C
```

Find all the classes that can be created with or without traits, as well as their subtype relations between them.

7) Suppose that trait `U` extends trait `T`. Is it reasonable to expect that `C with U` is a subtype of `C with T`? What happens in Scala?

8) Consider our favorite classes `Matrix` and `SparseMatrix` and suppose that they are implemented somehow in Scala using single dispatch. You may adopt any reasonable configuration. You may assume the existence of methods `add`, `multiply` etc. Write a trait to implement a thick interface on top of matrices. For example, the trait may support a method that raises a matrix to a power of n, or a method that pretty-prints the matrix etc. What are the merits of such a design?