JMath

The template class JMath resides in the name space JMATH and constitutes an auxiliary *base* class. It implements the affirm + and negate − operators, the modifying arithmetic operators +=, −=, *= and /= the non-modifying arithmetic operators +, −, * and / of a derived class.

To make this work, a derived class A should provide for a subset of the corresponding policy methods.

```
class A :
  public JMath<A>
{
  public:
  A& negate() { // implementation }
  A& add(const A&) { // implementation }
  A& sub(const A&) { // implementation }
  A& add(const double) { // implementation }
  A& div(const double) { // implementation }
};
```

method	operators
	+X
negate()	-X
add(..)	+=, X + X
sub(..)	-=, X - X
mul(..)	*=x, x * X, X * x
div(..)	/=x, X / x

where X and x correspond to a value of type A and double, respectively. Note that it is not necessary to implement all methods. For example.

```
  struct A :
    public JMath<A>
  {
    A() :
      x(0.0),
      y(0.0),
      z(0.0)
    {}

    A(const double x,
      const double y,
      const double z) :
      x(x),
      y(y),
      z(z)
    {}

    A& add(const A& object)
    {
      x += object.x;
      y += object.y;
      z += object.z;

      return *this;
```

```
  }

  A& mul(const double factor)
  {
    x *= factor;
    y *= factor;
    z *= factor;

    return *this;
  }

  friend inline std::ostream& operator<<(std::ostream& out, const A& object)
  {
    using namespace std;

    out << FIXED(5,2) << object.x << ' '
<< FIXED(5,2) << object.y << ' '
<< FIXED(5,2) << object.z;

    return out;
  }

  double x;
  double y;
  double z;
};
```

The following

```
A a1( 1.0,  2.0,  3.0);
A a2(-1.0, -2.0, -3.0);

cout << +a1 << endl;
cout << a1 + a2 << endl;
cout << a1 * 2.0 << endl;
cout << 2.0 * a1 << endl;

a1 += a2;

cout << a1 << endl;
```

will produce

```
1.00  2.00  3.00
0.00  0.00  0.00
2.00  4.00  6.00
2.00  4.00  6.00
0.00  0.00  0.00
```