JComparable

The template class `JComparable` resides in the name space `JLANG` and constitutes an auxiliary *base* class. In analogy with class `JEquals`, it implements the (not-)equal operators == and != as well as the comparison operators <, <=, > and >= of a derived class.

To make this work, a derived class `X` should provide for the policy method `bool less(const X&) const`.

```
class X :
  public JComparable<X>
{
  public:
  bool less(const X&) const { // implementation }
};
```

For example.

```
  struct A :
    public JComparable<A>
  {
    A(int value) :
      value(value)
    {}

    bool less(const A& object) const
    {
      return this->value < object.value;
    }

    int value;
  };

  A a1(0);
  A a2(1);

  cout << (a1 == a2) << endl;
  cout << (a1 != a2) << endl;
  cout << (a1 <  a2) << endl;
  cout << (a1 <= a2) << endl;
  cout << (a1 >  a2) << endl;
  cout << (a1 >= a2) << endl;
```

will produce

```
0
1
1
1
0
0
```

The class `JComparable` allows for a second template parameter. In that case, the (not-)equal and comparison operators are extended and also apply to a value corresponding to the second data type. In this

case, two additional policy methods should be provided, namely `bool less(const T&) const` and `bool more(const T&) const`, where `T` refers to the second template argument.

For example.

```
struct B :
  public JComparable<B>,
  public JComparable<B, A>
{
  B(const int value) :
    value(value)
  {}

  bool less(const B& object) const
  {
    return this->value < object.value;
  }

  bool less(const A& object) const
  {
    return this->value < object.value;
  }

  bool more(const A& object) const
  {
    return this->value > object.value;
  }

  int value;
};

B b1(0);
B b2(1);

cout << (b1 == b2) << ' ' << (b1 == a2) << ' ' << (a2 == b1) << endl;
cout << (b1 != b2) << ' ' << (b1 != a2) << ' ' << (a2 != b1) << endl;
cout << (b1 <  b2) << ' ' << (b1 <  a2) << ' ' << (a2 >  b1) << endl;
cout << (b1 <= b2) << ' ' << (b1 <= a2) << ' ' << (a2 >= b1) << endl;
cout << (b1 >  b2) << ' ' << (b1 >  a2) << ' ' << (a2 <= b1) << endl;
cout << (b1 >= b2) << ' ' << (b1 >= a2) << ' ' << (a2 <  b1) << endl;
```

will produce

```
0 0 0
1 1 1
1 1 1
1 1 1
0 0 0
0 0 0
```