

JEquals

The template class JEquals resides in the name space JLANG and constitutes an auxiliary *base* class. It implements the (not-)equal operators == and != of a derived class.

A possible implementation of class JEquals is.

```
template<class T>
class JEquals {
    friend bool operator==(T first, T second) { return first.equals(second); }
    friend bool operator!=(T first, T second) { return !first.equals(second); }
};
```

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

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

For example.

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

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

    int value;
};

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

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

will produce

```
1
0
0
1
```

The class `JEquals` allows for a second template parameter. In that case, the (not-)equal operators are extended and also apply to a value corresponding to the second data type. In this case, a second corresponding policy method `bool equals(const T& const)` should be provided where `T` refers to the second template argument.

For example.

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

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

    bool equals(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 << (b2 == b2) << ' ' << (b2 == a2) << ' ' << (a2 == b2) << endl;
cout << (b2 != b2) << ' ' << (b2 != a2) << ' ' << (a2 != b2) << endl;
```

will produce

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