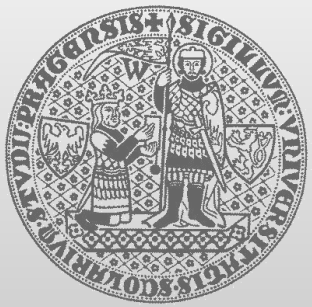
Advanced C# Programming
1st Labs
(counting from 0)

http://d3s.mff.cuni.cz/~jezek

Pavel Ježek

pavel.jezek@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Conversion Operators

Implicit conversion

- If the conversion is always possible without loss of precision
- e.g. long = int;

Explicit conversion

- If a run time check is necessary or truncation is possible
- e.g. int = (int) long;

Conversion operators for user-defined types

```
class Fraction {
    int x, y;
    ...
    public static implicit operator Fraction (int x) { return new Fraction(x, 1); }
    public static explicit operator int (Fraction f) { return f.x / f.y; }
}
```

Usage

```
Fraction f = 3;        // implicit conversion, f.x == 3, f.y == 1
int i = (int) f;       // explicit conversion, i == 3
```

Special Operator Methods

Operators, indexers, properties and events are compiled into “normal” methods

- `get_*` (property getter)
- `set_*` (property setter)
- `op_Implicit` (implicit cast)
- `op_Explicit` (explicit cast)

Operator Overloading

Static method for implementing a certain operator

```
struct Fraction {
    int x, y;
    public Fraction (int x, int y) {this.x = x; this.y = y; }

    public static Fraction operator + (Fraction a, Fraction b) {
        return new Fraction(a.x * b.y + b.x * a.y, a.y * b.y);
    }
}
```

Usage

```
Fraction a = new Fraction(1, 2);
Fraction b = new Fraction(3, 4);
Fraction c = a + b;    // c.x == 10, c.y == 8
```

- The following operators can be overloaded:
  - arithmetic: +, - (unary and binary), *, /, %, ++, --
  - relational: ==, !=, <, >, <=, >=
  - bit operators: &, |, ^
  - others: !, ~, >>, <<, true, false
- Must always return a function result
- If == (<, <=, true) is overloaded,!= (>=, >, false) must be overloaded as well.

Special Operator Methods

Operators, indexers, properties and events are compiled into “normal” methods

- `get_*` (property getter)
- `set_*` (property setter)
- `add_*` (event handler addition)
- `remove_*` (event handler removal)
- `op_Addition` (binary +)
- `op_Subtraction` (binary -)
- `op_Implicit` (implicit cast)
- `op_Explicit` (explicit cast)

  etc.

BONUS

Overloading of && and ||

In order to overload && and ||, one must overload &, |, true and false

```
class TriState {
    int state;  // -1 == false, +1 == true, 0 == undecided
    public TriState(int s) { state = s; }

    public static bool operator true (TriState x) { return x.state > 0; }
    public static bool operator false (TriState x) { return x.state < 0; }

    public static TriState operator & (TriState x, TriState y) {
        if (x.state > 0 && y.state > 0) return new TriState(1);
        else if (x.state < 0 || y.state < 0) return new TriState(-1);
        else return new TriState(0);
    }
    public static TriState operator | (TriState x, TriState y) {
        if (x.state > 0 || y.state > 0) return new TriState(1);
        else if (x.state < 0 && y.state < 0) return new TriState(-1);
        else return new TriState(0);
    }
}
```

true and false are called implicitly

```
TriState x, y;
if (x) ...          => if (TriState.true(x)) ...
x = x && y;         => x = TriState.false(x) ?  x : TriState.&(x, y);
x = x || y;         => x = TriState.true(x) ? x : TriState.|(x, y)
```