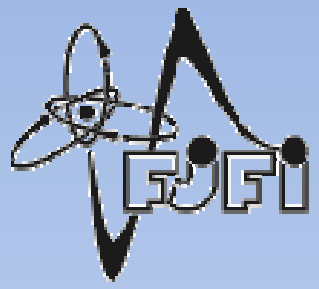
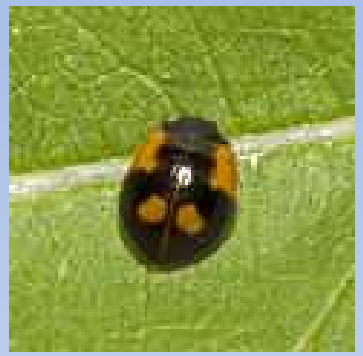


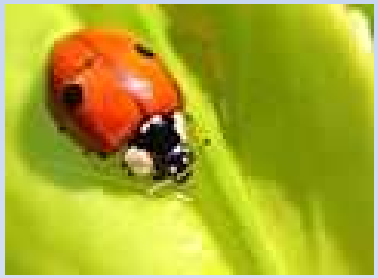
Polymorfismus a jeho historie

Pastirčáková, Pohanka

Polymorfismus



- Co je to polymorfismus?
- Kde se s ním setkáme?

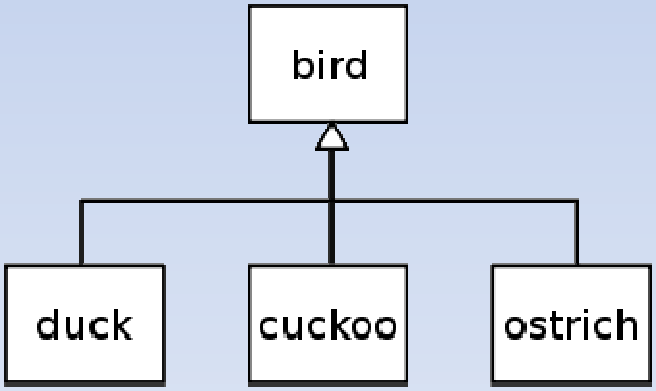


Polymorfismus v programování

- **Subtypový polymorfismus -** parametrem funkce je typ T, ale funkce může přijímat také podtřídy T
- **Parametrický polymorfismus -** funkce nebo datový typ jsou napsány obecně, bez závislosti na typu parametru
- **Operátorový polymorfismus -** provedení operace v závislosti na typu operandů (přetěžování operátorů)

Subtypový polymorfismus

- Různé třídy odvozené od jedné nadtřídy – funkce napsaná pro typ nadtřídy může přijímat i jednotlivé třídy



Subtypový polymorfismus

```php
interface IShape
{
	public function getArea();
}

class Triangle implements IShape
{
	private $a, $b, $c;

	public function __construct($a, $b, $c)
	{
		$this->a = $a;
		$this->b = $b;
		$this->c = $c;
	}

	public function getArea()
	{
		//výpočet obsahu pomocí Heronova vzorce
		$o = ($this->a + $this->b + $this->c) / 2;
		return sqrt($o * ($o - $this->a) * ($o - $this->b) * ($o - $this->c));
	}
}

class Circle implements IShape
{
	private $radius;

	public function __construct($radius)
	{
		$this->radius = $radius;
	}

	public function getArea()
	{
		return M_PI * $this->radius * $this->radius;
	}
}

function printShapeInfo(IShape $shape)
{
	echo 'Instance třídy ', get_class($shape), ', obsah: ', round($shape->getArea(), 2);
}

printShapeInfo(new Triangle(5, 10, 12)); //Instance třídy Triangle, obsah: 24.54
echo PHP_EOL;
printShapeInfo(new Circle(10)); //Instance třídy Circle, obsah: 314.16
```

Parametrický

- V kódu nejsou napsané typy, tyto doplníme až při specifické implementaci
- Výhody – méně kódu, šablony obvykle implementovány v knihovnách
- Nevýhody – není optimalizováno pro daný typ, musíme mít správně přetížené operátory

```
class List<T> {
    class Node<T> {
        T elem;
        Node<T> next;
    }
    Node<T> head;
    int length() { ... }
}
```

Operátorový polymorfismus

```
program Adhoc;

function Add( x, y : Integer ) : Integer;
begin
    Add := x + y
end;

function Add( s, t : String ) : String;
begin
    Add := Concat( s, t )
end;

begin
    Writeln(Add(1, 2));
    Writeln(Add('Hello, ', 'World!'));
end.
```

Operátorový polymorfismus – přetěžování operátorů

```
a := 1 + 2;
b := 1.0 + 2.0;
c := "hello " + "there";
```

Existuje také možnost dodefinovat si (přetížit) operátor po svém:

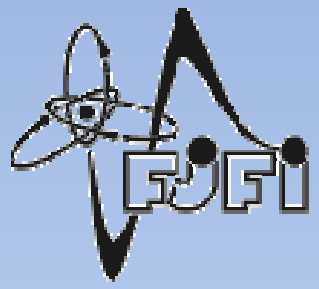
```
class complex {
  double rp, ip; // real part, imaginary part
public:
  complex(double r, double i) {rp=r; ip=i;}
  friend complex operator+(complex, complex);
  friend complex operator*(complex, complex);
};
```

Historie polymorfismu

- 1966 – poprvé použit subtypový polymorfismus (jazyk Simula)
- 1967 – polymorfismus představen na informatické konferenci
- 1976 – poprvé uveden na trh parametrický polymorfismus (jazyk ML)
- 1983 – první šablony (generické p., jazyk Ada)
- Java, C++, C#, Visual Basic, Delphi, Standard ML, OCaml, Ada, Haskell, Visual Prolog, Scala

Zdroje

- http://en.wikipedia.org/wiki/Simula
- http://en.wikipedia.org/wiki/Parametric_polymorphism#History
- http://cs.wikipedia.org/wiki/Objektove-orientovane-programovani
- http://www.builder.cz/rubriky/polymorfismus-dokonceni
- http://www.javaworld.com/jw-04-2001/jw-0413-polymorph.html
- http://web.cecs.pdx.edu/~antoy/Courses/TPFLP/lectures/TYPE/BasicTypechecking.pdf
- http://www.webber-labs.com/mpl



Děkuji za pozornost

A předávám slovo kolegovi ;-)