

Osnove izrade PHP aplikacija

Nasljeđivanje i iznimke

Izradio: Davor Cihlar

Uredio: Marko Čupić; Savjetovali: demosi

Plan

- ▶ Što ćemo naučiti
 - ▶ Nasljeđivanje
 - ▶ Nadjačavanje metoda
 - ▶ Konačne metode
 - ▶ Operator `instanceof`
 - ▶ Sučelja i apstraktne klase
 - ▶ Automatsko učitavanje klasa
 - ▶ Iznimke
- ▶ <http://www.php.net/manual/en/language.oop5.php>

Nasljeđivanje

- ▶ Proširivanje mogućnosti osnovne klase (klase roditelja)
- ▶ Moguće je naslijediti samo jednu klasu
 - ▶ Ali moguće je više uzastopnih nasljeđivanja

Nasljeđivanje – primjer



```
▶ class Vozilo {  
    var $masa;  
}  
  
class Kamion extends Vozilo {  
    var $nosivost;  
}  
  
class Auto extends Vozilo {  
    var $brojVrata;  
}  
  
var_dump(new Kamion());  
var_dump(new Auto());
```

Nadjačavanje metoda i varijabli

- ▶ Istoimena metoda u naslijeđenoj klasi zamjenjuje metodu roditelja
 - ▶ Isto vrijedi i za varijable
- ▶ Primjer:

```
class Roditelj {
    function posao() {
        echo "posao @ roditelj\n";
    }
}
class Naslijedjena extends Roditelj {
    function posao() {
        echo "posao @ naslijedjena klasa\n";
    }
}

$a = new Naslijedjena();
$a->posao();
```

Prosljeđivanje nadjačanih metoda

- ▶ Nekad je u metodi koja nadjačava potrebno pozvati nadjačanu metodu roditelja
 - ▶ Koristi se sintaksa `parent::nazivMetode`

- ▶ Primjer:

```
class Roditelj {
    function posao() {
        echo "posao @ roditelj\n";
    }
}

class Naslijedjena extends Roditelj {
    function posao() {
        parent::posao();
        echo "posao @ naslijedjena klasa\n";
    }
}
```

Prosljeđivanje konstruktora

- ▶ Instanciranjem naslijeđene klase ne poziva se konstruktor roditelja!

- ▶ Potrebno ručno pozivati konstruktor roditelja
 - ▶ Nije kao u C++-u

- ▶ Primjer:

```
▶ class Vozilo {  
    function __construct($a, $b) { ... }  
}  
▶ class Pila extends Vozilo {  
    function __construct($a) {  
        parent::__construct($a, 2);  
    }  
}
```

Operator ::

- ▶ Omogućuje pristup:
 - ▶ Statičkim metodama
 - ▶ Konstantama
 - ▶ Roditelju
 - ▶ `parent::`
 - ▶ Svojim metodama/varijablama umjesto nadjačanim
 - ▶ `self::`

Operator :: – primjer 1



```
▶ class Vozilo {  
    function vozi() { echo "Vozilo ide\n"; }  
    function pomakni() { $this->vozi(); }  
}
```

```
class Pila extends Vozilo {  
    function vozi() { echo "Pila pili\n"; }  
}
```

```
$t = new Pila();  
$t->pomakni();
```

- ▶ Koja metoda vozi će se pozvati?

Operator :: – primjer 2



```
▶ class Vozilo {  
    function vozi() { echo "Vozilo ide\n"; }  
    function pomakni() { self::vozi(); }  
}
```

```
class Pila extends Vozilo {  
    function vozi() { echo "Pila pili\n"; }  
}
```

```
$t = new Pila();  
$t->pomakni();
```

- ▶ Koja će se sad metoda vozi pozvati?

Konačne (final) metode

- ▶ Moguće je zabraniti nadjačavanje pojedinih metoda
- ▶ Primjer:

```
▶ class Roditelj {  
    final function posao() {  
        echo "posao @ roditelj\n";  
    }  
}  
class Naslijedjena extends Roditelj {  
    function posao() { // greska!  
    }  
}
```

Zaštićene (protected) metode i varijable

- ▶ Prisjetimo se vidljivosti
 - ▶ `public`, `private` i `protected`
- ▶ `protected` metodama i varijablama moguće je pristupiti samo unutar naslijeđenih klasa
- ▶ `private` metodama i varijablama moguće je pristupiti samo unutar klase koja definira tu metodu, odnosno varijablu

Operator instanceof

- ▶ `instanceof` omogućuje provjeru je li neka varijabla instanca neke klase
- ▶ Provjerava i za roditelje
 - ▶ `$k` je instanca od `x` iako je `$k` instanca klase koja nasljeđuje `x` (`x` je roditelj)
- ▶ Primjer:
 - ▶

```
$a = new Klasa();  
$b = "Klasa";  
var_dump($a instanceof Klasa); // true  
var_dump($a instanceof $b); // true
```

Sučelja

- ▶ Sučelja su klase koje nemaju varijable i sadrže samo nazive metoda
 - ▶ Da bi smo dobili klasu koja implementira sučelje i koju je moguće instancirati, ta klasa mora implementirati sve metode sučelja
 - ▶ U suprotnom je apstraktna i nije ju moguće instancirati
- ▶ Sučelja mogu imati konstante
 - ▶ Te konstante nije moguće nadjačati
- ▶ Implementiraju se, a ne nasljeđuju
 - ▶ `implements` umjesto `extends`
- ▶ Nije ih moguće instancirati

Sučelja – primjer

```
▶ interface Telefon {  
    function zovi($broj);  
    function poklopi();  
}  
  
class Mobitel implements Telefon {  
    function zovi($broj) {  
        echo "Zovem $broj!\n";  
    }  
    function poklopi() {  
        echo "Bye!\n";  
    }  
}  
  
$x = new Telefon(); // Greška!  
$y = new Mobitel(); // OK!
```

Apstraktne klase

- ▶ Klase koje nemaju barem jednu implementiranu metodu
- ▶ Nije ih moguće instancirati
 - ▶ Moguće je napraviti apstraktnu klasu sa svim implementiranim metodama, ali ju i dalje nije moguće instancirati
 - ▶ Takve apstraktne klase nemaju smisla
- ▶ Nasljeđuju se

Apstraktne klase – primjer

```
▶ abstract class Tipka {  
    var $naslov = 'A';  
  
    abstract function klik();  
  
    function getNaslov() {  
        return $this->naslov;  
    }  
}  
  
class ZelenaTipka extends Tipka {  
    function klik() { echo "Zeleni klik!\n"; }  
}  
  
$t = new ZelenaTipka();
```

Automatsko učitavanje klasa



- ▶ Moguće je definirati funkciju koja pri korištenju nedefinirane klase učitava datoteku s opisom klase

- ▶ Za napredne korisnike

- ▶ Primjer:

- ▶

```
function __autoload($imeKlase) {  
    $fn = "$imeKlase.inc.php";  
    echo "Ucitavam: $fn\n";  
    require $fn;  
}
```

```
$k = new NepostojecaKlasa();
```

Dobra praksa

- ▶ Uporaba oblikovnih obrazaca
 - ▶ Korišćenje "tvornica" (engl. *factory method*)
 - ▶ Jedinstvene instance (engl. *singleton*)
- ▶ <http://www.php.net/manual/en/language.oop5.patterns.php>

Iznimke

- ▶ Jednostavna obrada grešaka
 - ▶ Funkcije ne vraćaju grešku, već prekidaju tok izvođenja izbacivanjem iznimke
- ▶ Iznimke su instance klase
 - ▶ Nije ih moguće klonirati
 - ▶ Specifične iznimke moraju nasljeđivati klasu `Exception`
- ▶ Primjer:
 - ▶ **try** {
 // blok unutar kojeg se hvataju greške
 ...
} **catch** (Exception \$e) {
 // blok unutar kojeg se obrađuju greške
 echo "Greska: ", \$e->getMessage(), "\n";
}

Klasa Exception

```
▶ class Exception {
    protected $message;           // poruka o iznimci
    protected $code;              // kod iznimke
    protected $file;              // mjesto bacanja iznimke
    protected $line;              // linija u kodu

    public __construct($message = "", $code = 0,
                       $previous = NULL);

    final public getMessage();
    // getCode, getFile, ...
    final public getTrace(); // stack trace
    final public getTraceAsString();
    public __toString();
}
```

Bacanje iznimaka

- ▶ Prije izbacivanja je potrebno instancirati iznimku
- ▶ Primjer:

```
▶ function dijeli($a, $b) {  
    if ($b == 0)  
        throw new Exception('Dijeljenje s 0!');  
    return $a / $b;  
}
```

```
try {  
    dijeli(42, 0);  
} catch (Exception $e) {  
    echo "Greska: ", $e->getMessage(), "\n";  
}
```

Nasljeđivanje iznimaka

▶ Ponekad su potrebne specifične iznimke

▶ Primjer:

```
▶ class NemaDatoteke extends Exception {  
    var $file;  
    function __construct($file) {  
        parent::__construct("Datoteka nije nadjena");  
        $this->file = $file;  
    }  
}
```

// metoda ili funkcija:

```
function otvori($file) {  
    $f = fopen($file, "r");  
    if (!$f) throw new NemaDatoteke($file);  
    ...  
}
```

Hvatanje različitih iznimaka

- ▶ Ponekad je potrebna obrada greške koja ovisi o tipu iznimke
 - ▶ Iznimka koju se ne uhvati se automatski prosljeđuje
 - ▶ Paziti na redoslijed `catch` blokova!
 - ▶ Kad bi se prvo hvatao tip `Exception`, taj `catch` bi hvatao SVE iznimke jer je `Exception` bazna klasa od svih iznimaka

- ▶ Primjer:

```
▶ try {  
    ...  
} catch (NemaDatoteke $fnf) {  
    // hvata samo NemaDatoteke  
    ...  
} catch (Exception $e) {  
    // hvata bilo koju iznimku osim gornjih  
    ...  
}
```

Prosljeđivanje iznimaka

- ▶ Ponekad je potrebno obraditi pogrešku i javiti vanjskom `try` bloku da je bila greška

- ▶ Primjer:

```
▶ try {  
    ...  
} catch (Exception $e) {  
    ...           // obradi gresku  
    throw $e; // proslijedi ju dalje  
}
```

Zadatak (1)



- ▶ Izmijeniti zadatak s kartama (kartas.php)
 - ▶ `Spil` treba biti apstraktna klasa s apstraktnom metodom `podijeli`
 - ▶ Dodati klasu `ObicnaIgra` koja nasljeđuje `Spil`
 - ▶ Implementirati metodu `podijeli` koja dijeli 6 karata svakom od četiri igrača
 - rezultat je isti kao u zadatku s prošlog predavanja
 - ▶ Dodati klasu `NeobicnaIgra` koja nasljeđuje `Spil`
 - ▶ Implementirati metodu `podijeli` koja dijeli karte četvorici igrača:
 - Raspored dijeljenja je 3+3, tj. svaki igrač dobije dva seta s ukupno 6 karata
- ▶ Omogućiti korisniku da odabere igru, te dobije raspored karata

Zadatak (2)



- ▶ Proširiti prošli zadatak tako da je moguće zadati broj igrača preko URL-a
- ▶ Klasa `Spil` mora bacati iznimku ako nema više karata u špilul
 - ▶ Uхватiti i prikazati iznimku!
 - ▶ Ne hvatati iznimku unutar `podijeli`, već oko koda koji poziva tu metodu