Математички факултет, Универзитет у Београду

Катедра за рачунарство и информатику

Објектно оријентисано програмирање

вежбе

школска 2016/2017

Биљана Стојановић

Немања Мићовић

Никола Милев

1 Полиморфизам *

До сада смо увек користили промењљиву типа неке класе како бисмо реферисали објекте типа те исте класе: у промењљивој типа `Sfera`, чували смо референцу на објекат типа `Sfera`, у промењљивој типа `Tacka` референцу на објекат класе `Tacka`, итд.

Међутим, када имамо промењљиву типа базне класе, нпр. `Povrs`, осим што у њој можемо чувати референцу на објекат те класе, `Povrs`, на шта смо навикли, можемо такође чувати и референцу на објекат произвољне потклассе (било директне или индиректне), нпр. `Krug`, `Pravougaonik`, `Kvadrat`. Штавише, то је један од услова који морају бити испуњени да би полиморфизам функционисао.

Полиморфизам је фундаментални концепт објектно оријентисаног програмирања, који представља могућност да се један исти позив метода понаша различито у зависности од типа објекта над којим се метод примењује.

Другим речима, полиморфизам је могућност да промењљива класног (или интерфејсног) типа реферише на објекте различитих класа и да се користи за аутоматско позивање метода оне класе која представља тип реферисаног објекта, а не тип промењљиве.
Тиме се позив метода понаша различито у зависности од типа објекта на који је примењен.

Дакле, полиморфно својство могу имати позиви метода над објектима изведених класа, али само уколико се као референца на објекат изведене класе користи промењљива базне класе (директне или индиректне).

Напомена! Полиморфизам се примењује искључиво на методе, не и на атрибуте класе. На пример, за приступ атрибутима класе `Pravougaonik` може да се користи искључиво промењљива типа `Pravougaonik`. Чак и када реферише на објекат класе `Kvadrat`, можемо је користити само за приступ атрибутима који се односе на базни део објекта класе `Kvadrat` (што су у овом случају сви атрибути класе `Kvadrat`).

Услови који морају бити испуњени:
- у промењљивој типа базне класе налази се референца на објекат изведене
- позвани метод мора бити дефинисан у изведеној класи
- позвани метод мора такође бити декларисан у базној класи
- потписи метода у базној и изведеној класи морају бити исти
- повратни типови метода базне и изведене класе морају бити исти или коваријантни (повратни тип метода изведене је потклаcа повратног типа метода базне класе)
- приступни атрибут метода изведене класе не сме бити рестриктивнији од одговарајућег приступног атрибута базне (код нас ће оба бити `public`, тако да то важи)

Када се метод позове, биће извршена верзија метода из оне изведене класе на чији се објекат чува референца у промењљивој базног типа.

То који метод ће бити позван одређује се динамички, у време извршавања програма, а не у време његовог компајлирања. Позива се метод који одговара типу објекта за који је позван, а не типу промењљиве која реферише на тај објекат. Како промењљива базног типа може реферисати на објекат произвољне потклассе, тачан тип реферисаног објекта не може бити одређен у време компајлирања.

То је илустровано у примеру са вежби тако што се случајно бира елемент низа (елементи су типа разних изведених класа) и за њега врши полиморфни позив метода. Пре него што програм започне извршавање, не можемо знати који елемент низа ће бити случајно изабран, а ипак се увек изврши полиморфни метод класе изабраног објекта. Ако се

* *Укључује и садржај из материјала Марије Милановић*

изабере круг, површина се израчуна по формули $P=r^2\pi$, док ако се изабере правоугаоник, за рачунање површине примени се формула `P=a*b`.

Полиморфан метод са коваријантним повратним типом у изведеним класама јесте метод `izvedenaPovrs()`, чији је повратни тип у базној класи `Povrs`, а у изведеним класама тип саме изведене класе.

2 Кастовање

Кастовање представља објекат као да је неког другог типа.
Могуће је кастовати објекат у тип (директне или индиректне) натклase или потклase, дакле само навише и наниже кроз неку хијерархију класа.

Кастовање навише врши се имплицитно (компајлер га убацује сâм, када је потребно).
То је управо оно што имамо код полиморфизма, када референцу на објекат типа изведене класе чувамо у променљивој типа базне:

```
Povrs p = new Kvadrat(new Tacka(4,6), 2);
```

Други разлог за кастовање навише кроз хијерархију је да би се методу, као аргумент, проследила референца на објекат неке од могућих потклаcа. Задавањем да је параметар базног типа имамо флексибилност да методу, приликом позива, проследимо референцу на објекат произвољне изведене класе.

Нпр. ако је параметар типа `Povrs`, можемо проследити референцу на објекат типа `Krug`, `Pravougaonik` или `Kvadrat`. То нпр. радимо када у копи-конструктору изведене класе позивамо копи-конструктор базне да изврши иницијализацију базног дела објекта изведене класе.

Кастовање наниже кроз хијерархију класа, тј. кастовање у тип изведене класе, мора се извршити експлицитно. Да би кастовање радило, класа у коју кастујемо мора бити стварна класа објекта или суперкласа објекта!

Дакле, након горње наредбе, није добро (биће избачен изузетак типа `ClassCastException`):

```
(Krug)p // pokusaj kastovanja kvadrata u krug!!! kvadrat nije krug!
```

а добро је

```
(Pravougaonik)p // kastovanje kvadrata u pravougaonik je OK. kvadrat JE pravougaonik
```

као и

```
(Kvadrat)p // kastovanje kvadrata u njegov stvarni tip je OK.
```

Када кастујемо наниже кроз хијерархију класа?

Кастујемо онда када смо нпр. због полиморфизма референцу на објекат морали да чувамо у променљивој типа базне класе, а желимо да извршимо метод специфичан за класу објекта (дакле, метод који нисмо наследили од базне класе).

То илуструје пример са вежби у коме имамо метод `opisaniKrug()` у класи `Pravougaonik`. Метод не задовољава услове за полиморфно позивање (није дефинисан у базној класи `Povrs`).

Класа `Povrs` је базна за класу `Pravougaonik` и ако се у промењивој `p`, типа базне класе, налази референца на објекат типа `Pravougaonik`, да бисмо за тај објекат позвали метод `opisaniKrug()`, неопходно је прво извршити кастовање у тип `Pravougaonik`:

```
Povrs p = new Pravougaonik(new Tacka(),3,4);
// pre kastovanja uvek u programu treba proveriti
// da li je to kastovanje legitimno!!!

// ovde bi trebalo da dodje ta provera
Krug oKrug = ((Pravougaonik)p).opisaniKrug();
```

3 Оператор instanceof

Оператором **`instanceof`** могуће је проверити да ли је легитимно кастовање неке промењиве у тип неке класе. Нпр.

```
if(p instanceof Pravougaonik){ // ovo je provera koja gore nedostaje
       Pravougaonik pravougaonik = (Pravougaonik)p;
       Krug oKrug = pravougaonik.opisaniKrug();

       // ili, umesto dve gornje naredbe, naredba
       Krug oKrug = ((Pravougaonik)p).opisaniKrug();
}
```

`instanceof` враћа `true` ако је објекат референсан левим операндом истог типа као и десни операнд или ако је типа произвољне његове потклассе.
У том случају, допуштено је кастовање објекта референсаног левим операндом (промењивом `p`) у тип десног операнда (`Pravougaonik`).

4 Одређивање класе објекта – getClass(), Class

За сваку класу, интерфејс, тип низа и примитивни тип који користимо у програму постоји тачно по један `Class` објекат који можемо користити за проверу да ли се у датој промењивој чува вредност неког конкретног референтног или примитивног типа.
(Подсетимо се: референтни типови су класни, интерфејсни и низовски).

`Class` објекте генерише Јава Виртуелна Машина (JBM) када се учитава наш програм. Пошто је примарна сврха за `Class` да га користи JBM, нема `public` конструктора, па не можемо сами креирати `Class` објекте.

У програму, референца на одговарајући `Class` објекат добија се директно тако што се `.class` надовеже на име класе, интерфејса, примитивног типа или на тип низа. Примери:

```
String.class (java.lang.String.class) Pravougaonik.class    // klase
Runnable.class      // interfejs
int.class           // primitivni tip
int[].class         // niz
```

`Class` је пример генеричког типа (заправо дефинише скуп класа).

Метод `getClass()`, који све наше класе наслеђују од универзалне суперкласе `Object`, враћа референцу на објекат типа `Class` који одређује класу објекта.

`getClass()` враћа стварни тип (класу) објекта. Чињеница да је референца на објекат можда смештена у променљиву типа његове базне класе не утиче на тип самог објекта.

Нпр. у случају:

```
Object str = "Neki string";    // referencu na String objekat cuvamo kao tip Object
```

`str.getClass()` вратиће `java.lang.String.class`.
Метод је у класи `Object` дефинисан као `final` (не може се предефинисати у изведеним класама).

Тестирање да ли дата променљива `p` реферише (баш) на објекат класе `Pravougaonik`:

```
if(p.getClass() == Pravougaonik.class)
        System.out.println("Pravougaonik!");
```

("баш" се односи на:) Ако `p` садржи референцу на објекат поткласе класе `Pravougaonik`, нпр. `Kvadrat`, резултат поређења у `if`-у је `false`.
Провера помоћу == је могућа јер постоји тачно по 1 примерак `Class` објекта за сваки (примитивни и референтни) тип коришћен у програму.

Разлика између `instanceof` *и* `getClass()`

```
if(p instanceof Pravougaonik)
        System.out.println("Pravougaonik!");
else
        System.out.println("Definitivno nije pravougaonik!");
```

Добићемо потврду да имамо правоугаоник чак и ако је у `p` референца на квадрат. Ово је добро за потребе кастовања!

Међутим, ако напишемо:

```
if(p.getClass() == Pravougaonik.class)
        System.out.println("Pravougaonik!");
else
        System.out.println("Definitivno nije pravougaonik!");
```

услов у `if` ће имати вредност `false` ако је класа објекта на који реферише променљива `p` `Kvadrat`, јер је њен `Class` објекат различит од `Pravougaonik.class` – морали бисмо да напишемо `Kvadrat.class` да би услов у `if` имао вредност `true`.

`instanceof` или `Class`?

За сврхе кастовања увек треба користити `instanceof`,

док уколико је потребно потврдити тачан тип референце, треба користити одговарајући `Class` објекат.