Методика наставе рачунарства

Владимир Филиповић
vladaf@matf.bg.ac.rs

Апстрактне класе и интерфејси

Владимир Филиповић
vladaf@matf.bg.ac.rs

Апстрактне класе

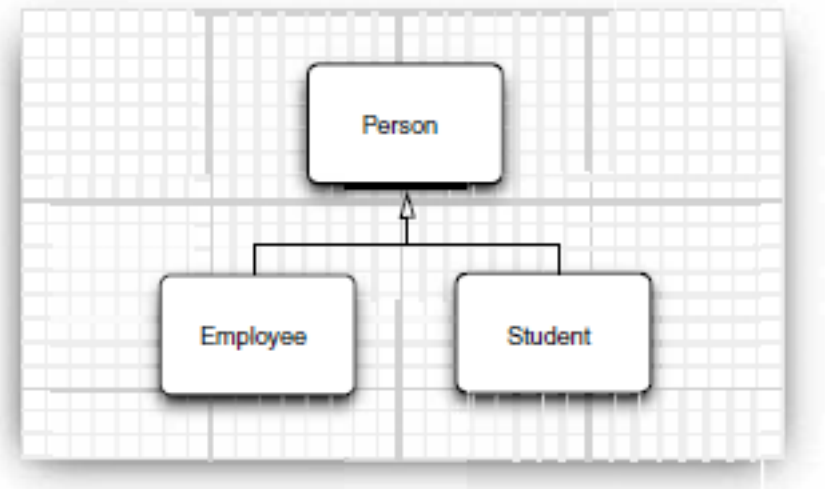
Како се крећемо уз хијерархију наслеђивања, класе постају све општије и све апстрактније. У неком тренутку наткласа постаје у тој мери општа да више представља основу за друге класе него класу чије конкретне примерке желимо да користимо.

Пример:

Проширење хијерархије класа за запослене и студенте.

Зашто правити више нивое апстракције?

Мада свака особа има опис, ипак тај опис зависи од тога шта и како дата особа ради.



Природно математички факултет

vladaf@matf.bg.ac.rs

Апстрактне класе (2)

Апстрактне класе и методе карактерише кључна реч abstract.

Класа мора бити апстрактна ако садржи бар један апстрактни метод.

Класа може бити апстрактна чак иако не садржи ни један апстрактни метод.

Апстрактне класе служе за обједињавање својства неких конкретних класа.

Примерак конкретне класе може да користи неапстрактне методе апстрактне надкласе, који нису предефинисани у конкретним класама. Тај примерак може бити декларисан као инстанца апстрактне класе.

Он може бити креиран само помоћу конструктора конкретне класе. Међутим, и апстрактна класа може да има конструктор којим дефинише сопстевена поља, а тај контруктор се потом позива од стране конструктора конкретне надкласе.

Природно математички факултет

vladaf@matf.bg.ac.rs

Интерфејси

Постоји велики број ситуација у развоју софтвера у којим је важно да се различите групе програмера договоре око „уговора“ који ће исказати на који се начин одвија интеракција у софтверу. Свака од тих група треба да буде у могућности да напише свој део кода, а да при томе нема информације како је писан код друге стране. Уопштено говорећи, интерфејси представљају такву врсту уговора.

У језику Јава, интерфејс је референтни тип, сличан класи, али може садржати само константе и потписе метода. Интерфејс не може да садржи тела метода.

Није могуће директно правити примерак интерфејса – он само може да буде имплементиран од стране класе или наслеђен од стране другог интерфејса.

Интерфејси (2)

Интерфејси (као апстрактне класе и методи) обезбеђују шаблоне за неко понашање, а које ће друге класе користити. Обезбеђују већу функционалност у односу на апстрактне класе и методе. Преко интерфејса уводи се неки вид ограниченог вишеструког наслеђивања.

Интерфејс обезбеђује апстрактно понашање које се додаје било којој класи, а које није обезбеђено преко њених надкласа.

Они представљају неку врсту протокола за комуникацију између класа, тј. дефинишу шаблоне за понашање класа.

Интерфејси (3)

Интерфејс се понаша свуда као класа (и преводи се као класа), али не може имати инстанце (не може се на њега применити оператор new).

Интерфејс садржи апстрактне методе (што се не мора посебно нагласити јер се подразумева) и константе. Дакле, интерфејс не може садржавати променљиве.

Novi интерфејс се креира са:

```
public interface MojInterfejs
{
        .......
}
```

Интерфејси (4)

Код интерфејса нема хијерархијске организације. Како би нагласили да један интерфејс наслеђује више других, иза кључне речи extends наводимо све интерфејсе које овај наслеђује.

```
public interface DrugiInterfejs extends Prvi, Primarni
{
    …// svi metodi su public i abstract
    …//sve promenljive su: public, static i final
}
```

Интерфејси се, као и класе, смештају у пакете.

Ако се за методе и променљиве у интерфејсу не нагласимо да су abstract, public и final, подразумеваће се да је тако.

Природно математички факултет

vladaf@matf.bg.ac.rs

Интерфејси (5)

Дефинисани интерфејси се имплементирају од стране Јава класа. Можемо користити већ раније дефинисане интерфејсе (који већ постоје у Јава-библиотеци) или направити своје.

Овај концепт имплементације (било постојећих, било нових) интерфејса је суштински исти као концепт наслеђивања (било постојећих, било нових) класа. У оба случаја проширују се могућности класа имплеметирањем интерфејса.

Пример:

```
class MojAplet extends java.applet.Applet
 implements  Runnable
{
      ……..
}
```

Интерфејси (6)

Пример:

```
class Druga extends Prva implements  MojInterfejs
{
      ……..
}
```

Када класа имплементира интерфејс, тада се морају имплементирати сви методи интерфејса (не могу се бирати само они који нам одговарају).

Интерфејси (7)

Када се интерфејс имплементира у некој класи, њена поткласа наслеђује све методе и може их превазићи (предефинисати). Ако је у класи имплеметиран интерфејс, није неопходно да се реч implements јави и у дефиницији поткласе.

Пример:

```
interface Radoznao{
        void pita();
        void interesuje_se();
      //…
 }
class Naucnik implemets Radozano{
       String ime;
       // …
 }
class Istrazivac extends Naucnik {
       // Ovde se mogu koristiti metodi pita() i Interesuje_se()
}
```

Интерфејси (8)

Једна класа може имплементирати више интерфејса.

Можемо писати:

```
public class Moja implements Prvi, Drugi, Treci
{
    // ...
}
```

Овде се могу појавити иста имена метода (са истим потписом!) у различитим интерфејсима.

Тада се коришћењем кратког имена може имплементирати само један од два таква метода (ако се редефинишу оба метода унутар класе, неопходно је користити пуна имена).

Интерфејси (9)

Могу се декларитати променљиве које ће бити типа интерфејс (јер скоро свуда где користимо класе, можемо користити и интерфејсе!)

Можемо писати:

```
Runnable  trceci = new MojObjekat();
```

Од објекта trceci се очекује да извршава метод run() интерфејса Runnable.

Интерфејси (10)

Како се могу користити параметри у методима интерфејса ако ће их имплементирати различите класе? Једна од могућности је да се параметри декларишу тако да буду типа интерфејса.

Пример:

```
public interface Radoznao
{
      public abstract pita(Radoznao neko);
    //…
}
public class Naucnik implemets Radozano
{
   public pita(Radoznao neko){
     Naucnik pravi = (Naucnik)neko;
     // …
   }
}
```

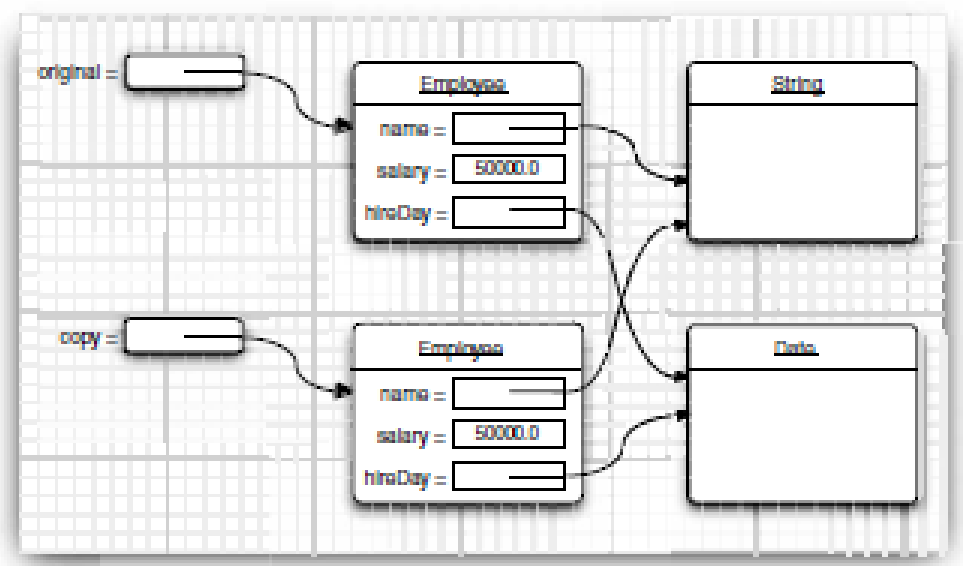
Природно математички факултет

vladaf@matf.bg.ac.rs

Интерфејси у ЈДК-у

Интерфејс омогућује прецизно дефинисање како се објекат клонира.

Клонирање, онако како је реализовано у класи Object представља тзв. „плитко“ копирање.



У неким ситуацијама, то је баш оно што треба програмеру. У другим ситуацијама, када је оригинал мутабилан, то може довести до озбиљних проблема.

Природно математички факултет

vladaf@matf.bg.ac.rs

Интерфејси у ЈДК-у (2)

Што се клонирања тиче, за дату класу треба изабрати шта је најбоље:

1. Подразумевано плитко клонирање је довољно добро;
2. Подразумевани метод clone треба надоградити позивима метода clone над подобјектима тј. пољима која су мутабилна.
3. уопште не треба позивати метод clone.

Трећа могућност је подразумевана. Ако програмер изабере прву или другу могућност, тада његова класа мора да:

1. Имплементира интерфејс Cloneable;
2. Редефинише метод clone са модификатором приступа public.

Интерфејси у ЈДК-у (3)

Коришћење интерфејса Cloneable нема никакве везе са уобичајеним коришћењем интерфејса.

Конкретно, интерфејс Cloneable не садржи метод clone – тај метод је наслеђен од класе Object. У овом случају интерфејс служи као маркер, који указује да дизајнер класе разуме процес клонирања и да жели да га подржи.

Објекти су у тој мери „параноични“ што се тиче клонирања, да генеришу изузетак уколико неки објекат захтева да буде клониран а да при том не имплементира овај интерфејс.

Чак и у случају када одговара подразумевана имплементација метода clone (плитко копирање), ипак је потребно да се имплементира интерфејс Cloneable, да се редефинише метод clone тако да буде public, и да се у телу тог метода позове super.clone().

Интерфејси у ЈДК-у (4)

Обично испоручилац сервиса тврди: “Ако ваша класа испуњава конкретни интерфејс, ја ћу онда пружити услугу.”

Размотримо конкретан пример. Метод sort у класи Arrays обећава да ће сортирати низ објеката, али под једним условом – објекти у низу морају сами знати како да се упореде тј. морају припадати класи која имплементира интерфејс Comparable.

```
public interface Comparable
{
   int compareTo(Object other);
}
```

Да би класа имплементирала интерфејс Comparable она мора да садржи метод compareTo, и тај метод мора да има параметар типа Object и мора да врати вредност типа integer.

Интерфејси у ЈДК-у (5)

java.lang.Comparable

- int compareTo(Object other)
  compares this object with other and returns a negative integer if this object is less than other, zero if they are equal, and a positive integer otherwise.

java.util.Arrays

- static void sort(Object[] a)
  sorts the elements in the array a, using a tuned mergesort algorithm. All elements in the array must belong to classes that implement the Comparable interface, and they must all be comparable to each other.
- static void sort(Object[] a, Comparator c)

java.util.Comparator

- int compareTo(Object o1, Object o2)
  compares its two arguments o1 and o2 for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. It is generally the case, but not strictly required that (compare(x, y)==0) if and only if (x.equals(y)).

Препоруке за наслеђивање

1. Заједничке операције и поља сместити у надкласе.

Тако је оформљена класа Person као надкласа Employee и Student.

2. Не користити заштићена поља.

Модификатор protected не пружа много заштите, из два разлога:
- Скуп подкласа је неограничен и свако може да прави подкласу наше класе и тако да пише код који директно приступа protected пољима и на тај начин разбија енкапсулацију.
- У програмском језику Јава све класе у истом пакету имају приступ protected пољима, без обзира на то да ли се ради о поткласама или не.

Међутим, protected методи могу бити корисни за назначавање да дати метод није спреман за општу употребу и да треба да буде редефинисан у поткласама. Добар пример за то је метод clone.

Препоруке за наслеђивање (2)

3. Користити наслеђивање за моделирање односа “јесте”.

Понекад програмери претерују у коришћењу наслеђивања.

На пример, претпоставимо да нам требају радници по уговору, тј. класа Contractor. Радници под уговором садрже имена и датум запослења, али не садрже плату, већ се плаћају по сату.

Иако постоји изазов да се класа Contractor направи као подкласа класе Employee којој је додато поље hourlyWage, то не би била добра идеја јер би тада примерак класе Contractor садржао и поље за плату и поље за сатницу, а то би водило у проблеме.

Наиме однос између ентитета радник по уговору и запослени не пролази тест “јесте”. Радник по уговору није специјалан случај запосленог.

Препоруке за наслеђивање (3)

4. Не користити наслеђивање сем уколико оно има смисла за све методе класе из које се наслеђује.

Претпоставимо да желимо да направимо класу за празнике Holiday. Будући да је сваки празник дан, а да су дани примерци класе GregorianCalendar, то можемо користити наслеђивање.

```
class Holiday extends GregorianCalendar { . . . }
```

На несрећу, скуп празника није затворен у односу на наслеђене операције. Један од јавних метода класе GregorianCalendar је метод add. Међутим, овај метод може да претвори празник у нерадни дан:

```
Holiday christmas;
christmas.add(Calendar.DAY_OF_MONTH, 12);
```

Стога у овом примеру наслеђивање није адекватно.

Препоруке за наслеђивање (4)

5. Приликом превазилажења метода не мењати очекивано понашање тј. поштовати принцип замене.

Принцип замене се примењује и на синтаксу и на понашање.

При превазилажењу метода се не сме неразумно мењати његово понашање. На пример, ако се „поправи“ проблем са методом add у класи Holiday тако да сада add или не ради ништа или диже изузетак или пребацује на следећи празник, тада бива нарушен принцип замене. Наиме, секвенца нареби:

```
int d1 = x.get(Calendar.DAY_OF_MONTH);
x.add(Calendar.DAY_OF_MONTH, 1);
int d2 = x.get(Calendar.DAY_OF_MONTH);
System.out.println(d2 - d1);
```

треба да има очекивано понашање, тј. да врати 1, без обзира да ли је променљива x типа GregorianCalendar или Holiday.

Препоруке за наслеђивање (5)

6. Користити полиморфизам, а не информације о типу.

Кад год се наиђе на код облика

```
if (x is of type 1)
  action1(x);
else if (x is of type 2)
  action2(x);
```

треба размотрити могућност полиморфизма.

Да ли action1 и action2 представљају заједничке концепте? Ако је одговор да, тај заједнички концепт треба да буде метод заједничке надкласе или интерфејса. Потом се једноставно треба позвати x.action(); па да механизам динамичког активирања који је инхерентан полиморфизму покреће одговарајућу акцију.

Код који користи полиморфне методе или интерфејсе се много лакше одржава и надограђује, него што је то случај са кодом који користи вишестуко тестирање типова.

Природно математички факултет

vladaf@matf.bg.ac.rs

Препоруке за наслеђивање (6)

7. Не претеривати са рефлексијом.

Механизам рефлексије омогућава програмерима да пишу програме са запањујућом уопштеношћу, тако што се при извршавању детектују поља и методи.

Ова могућност може бити изузетно корисна за системско програмирање, али обично није од користи за програмирање апликација.

Рефлексија је ломљива, зато што преводилац не може да помогне у проналажењу грешака у програму. Стога се грешке при овој врсти програмирања откривају приликом извршавања и доводе до дизања изузетака.

Природно математички факултет

vladaf@matf.bg.ac.rs

Захвалница

Велики део материјала који је укључен у ову презентацију је преузет из презентације коју је раније (у време када је он држао курс Објектно орјентисано програмирање) направио проф. др Душан Тошић.

Хвала проф. Тошићу што се сагласио са укључивањем тог материјала у садашњу презентацији, као и на помоћи коју ми је пружио током конципцирања и реализације курса.