20.Projekt Domácí mediotéka

20.1. Základní popis, zadání úkolu

V projektu Domácí mediotéka (Dome) se jednoduchým způsobem evidují CD a videa. Projekt je velmi jednoduchý (tj. v praxi nepoužitelný), bude použit pro vysvětlení dědičnosti a polymorfismu. V tomto projektu budeme řešit tyto úkoly:

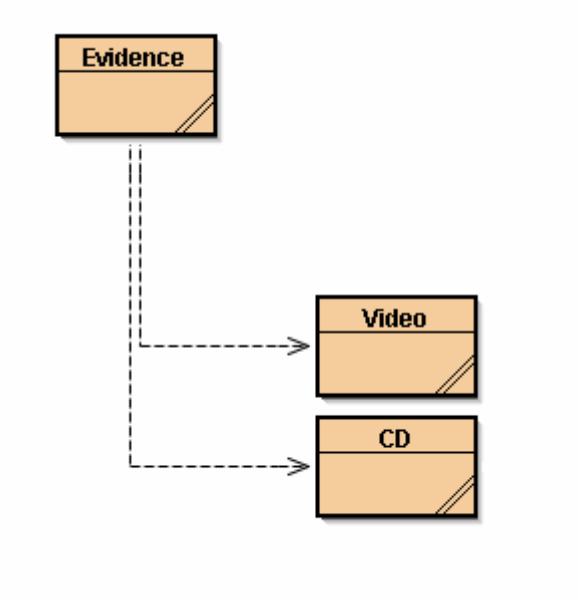
- omezit duplicity ve třídách `Video` a `CD` – vytvoří se předek, který bude obsahovat společné datové atributy a metody,
- omezit duplicity ve třídě `Evidence` – pouze jeden seznam, který bude obsahovat videa i CD,
- doplnit evidenci o knihy,
- použití rozhraní (interface), které budou implementovat třídy `Video` a `CD`.

Tento projekt má následující cíle:

- ukázat základní použití dědičnosti a polymorfismu.

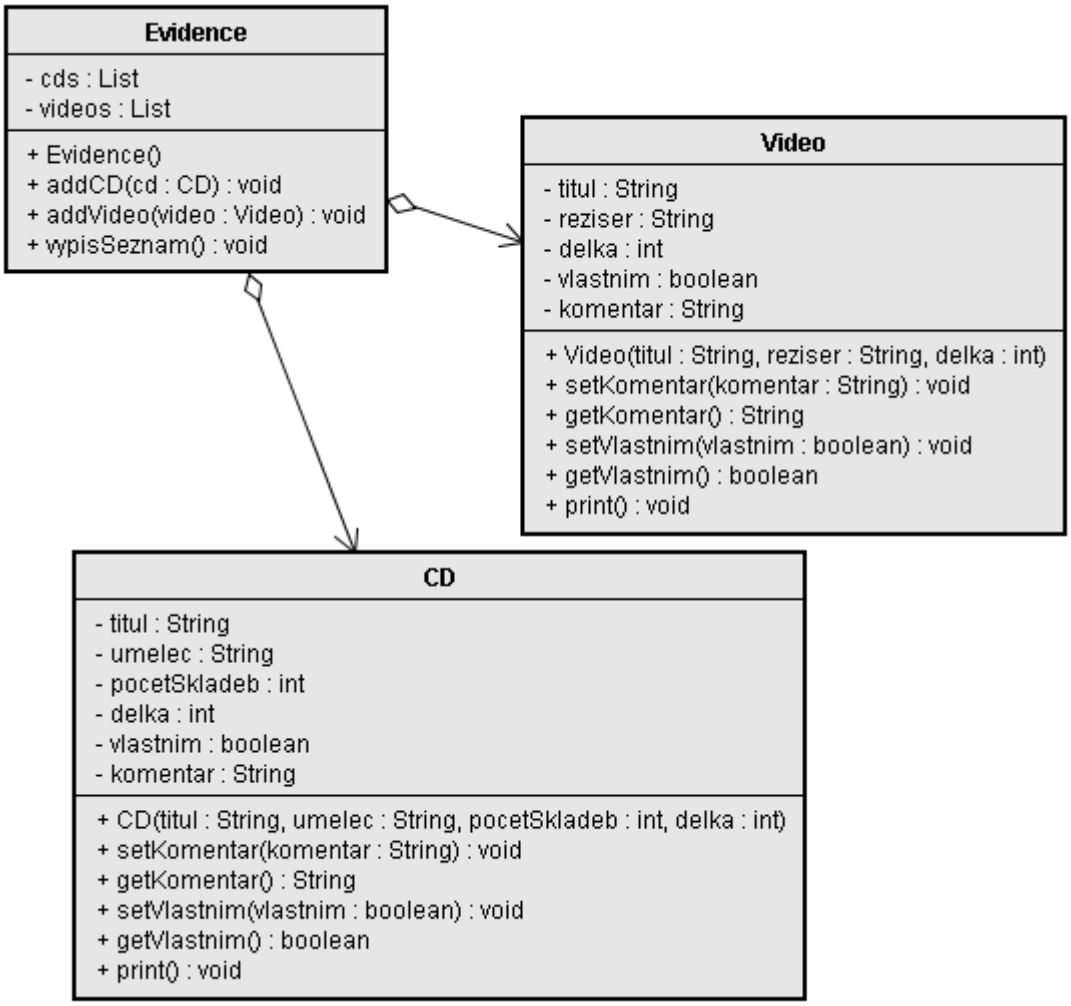
20.2. Struktura tříd

Projekt Dome se na začátku skládá z následujících tříd (obrázek z BlueJ):



Obrázek 20.1 Diagram tříd projektu Dome převzatý z BlueJ

Instance třídy `Evidence` obsahuje seznamy evidovaných videí a CD, má metody pro přidání videa, pro přidání CD a pro vypsání seznamu evidovaných videí a CD.
O každém videu se eviduje titul, režisér, délka videa v minutách. O každém CD se eviduje titul, autor (umělec), počet skladeb, délka v minutách. U obou typů lze ještě zadávat komentář (metody `setKomentar()` a `getKomentar()`) a údaj o vlastnictví konkrétního titulu (metody `setVlastnim()` a `getVlastnim()`).



Obrázek 20.2 Diagram tříd projektu Dome včetně datových atributů a metod

20.3. Výpis kódu třídy Evidence

```
 1 import java.util.ArrayList;
 2 import java.util.List;
 3 /**
 4  * Třída slouží pro evidenci CD a videa. Seznam všech uložených
 5  * CD a videí může být vypsán do textového okna.
 6  * Tato verze neukládá data na disk ani neposkytuje funkce pro
 7  * vyhledání nějakého titulu.
 8  *
 9  * @author Michael Kolling and David J. Barnes
10  * @author Luboš Pavlíček
11  * @version 2005-jul-31
12  */
13 public class Evidence {
14   private List <CD>cds;
15   private List <Video>videos;
16
```

```
17    /**
18     * Vytvoří prázdnou evidenci.
19     */
20    public Evidence() {
21    cds = new ArrayList<CD>();
22    videos = new ArrayList<Video>();
23    }
24
25    /**
26     * Přídá CD do evidence.
27     */
28    public void addCD(CD cd) {
29       cds.add(cd);
30    }
31
32    /**
33     * Přidá video do evidence.
34     */
35    public void addVideo(Video video) {
36       videos.add(video);
37    }
38
39    /**
40     * Vypíše seznam všech aktuálně uložených CD a videi do
41     * textového okna.
42     */
43    public void vypisSeznam() {
44       // vypíše seznam CD
45       for (CD cd : cds) {
46            cd.print();
47            System.out.println();   // prázdný řádek mezi položkami
48       }
49
50       // vypíše seznam videí
51       for (Video video : videos) {
52            video.print();
53            System.out.println();   // prázdný řádek mezi položkami
54       }
55    }
56 }
```

20.4. Výpis kódu tříd Video a CD

```
 1 /**
 2  * Třída Video představuje jedno video (jeden pořad/nahrávku).
 3  * Informace o videu je možné ukládat a získávat. K nahrávce lze
 4  * nastavit komentář a vlastnictví kopie videa.
 5  * @author Michael Kolling and David J. Barnes
 6  * @author Luboš Pavlíček
 7  * @version 2005-jul-31
 8  */
 9 public class Video {
10   private String  titul;
11   private String  reziser;
12   private int     delka;
13   private boolean vlastnim;
14   private String  komentar;
```

```
15
16   /**
17    * Konstruktor pro vytvoření instance třídy Video. Vlastnictví
18    * nahrávky se nastavuje samostatně pomocí metody setVlastnim
19    *
20    * @param   titul   titul videokazety/nahrávky
21    * @param   reziser režisér nahrávky
22    * @param   delka   délka nahrávky v minutách
23    */
24   public Video(String titul, String reziser, int delka) {
25      this.titul = titul;
26      this.reziser = reziser;
27      this.delka = delka;
28      vlastnim = false;
29      komentar = "<bez komentáře>";
30   }
31
32   /**
33    * Vloží komentář k nahrávce.
34    * @param   komentar   komentář k nahrávce
35    */
36   public void setKomentar(String komentar) {
37      this.komentar = komentar;
38   }
39   /**
40    * Vrací komentář k této nahrávce.
41    *
42    * @return       komentář k nahrávce
43    */
44   public String getKomentar() {
45      return komentar;
46   }
47
48   /**
49    * nastavení příznaku, zda vlastníme kopii tohoto videa.
50    *
51    * @param   vlastnim    true, pokud vlastním nahrávku videa
52    */
53   public void setVlastnim(boolean vlastnim) {
54      this.vlastnim = vlastnim;
55   }
56
57   /**
58    * vrací informaci, zda vlastníme kopii tohoto videa.
59    *
60    * @return      true, pokud vlastníme kopii tohoto videa
61    */
62   public boolean getVlastnim() {
63   return vlastnim;
64   }
65
66   /**
67    * Vypíše informace o videu do textového okna.
68    */
```

```
69   public void print() {
70   System.out.printf("video: %s (%d min)", titul, delka);
71   if(vlastnim) {
72      System.out.println("*");
73   } else {
74      System.out.println();
75   }
76   System.out.println("    " + reziser);
77   System.out.println("    " + komentar);
78   }
79 }
```

Třída CD je velmi podobná:

```
 1 /**
 2  * Třída CD představuje jedno cédéčko. Informace o cédéčku je
 3  * možné ukládat a získávat. K cédečku lze nastavit komentář a
 4  * příznak o vlastnictví kopie CD.
 5  *
 6  * @author Michael Kolling and David J. Barnes
 7  * @author Luboš Pavlíček
 8  * @version 2005-jul-31
 9  */
10 public class CD{
11   private String titul;
12   private String umelec;
13   private int pocetSkladeb;
14   private int delka;
15   private boolean vlastnim;
16   private String komentar;
17
18   /**
19    * Inicializace CD. Vlastnictví kopie CD je potřeba nastavit
20    * pomocí metody setVlastnim.
21    *
22    * @param titul         titul CD
23    * @param umelec        umělec (zpěvák apod.)
24    * @param pocetSkladeb  počet skladeb
25    * @param delka         délka nahrávky v minutách
26    */
27   public CD(String titul, String umelec, int pocetSkladeb,
28                     int delka) {
29   this.titul = titul;
30   this.umelec = umelec;
31   this.pocetSkladeb = pocetSkladeb;
32   this.delka = delka;
33   vlastnim = false;
34   komentar = "<bez komentáře>";
35   }
36
37   /**
38    * Vlož komentář k nahrávce.
39    *
40    * @param komentar   komentář k nahrávce
41    */
42   public void setKomentar(String komentar) {
43      this.komentar = komentar;
44   }
```

```
45
46    /**
47     * Vrací komentář k tomuto CD.
48     *
49     * @return      komentář k CD
50     */
51    public String getKomentar() {
52       return komentar;
53    }
54
55    /**
56     * nastavení příznaku, zda vlastníme kopii CD.
57     *
58     * @param    vlastnim   true, pokud vlastním nahrávku tohoto CD
59     */
60    public void setVlastnim(boolean vlastnim) {
61       this.vlastnim = vlastnim;
62    }
63
64    /**
65     * Vrací informaci, zda vlastníme kopii tohoto CD.
66     *
67     * @return      true, pokud vlastníme kopii tohoto CD
68     */
69    public boolean getVlastnim() {
70       return vlastnim;
71    }
72
73    /**
74     * Vypíše informace o CD do textového okna.
75     */
76    public void print() {
77    System.out.printf("CD: %s (%d min)", titul, delka);
78    if(vlastnim) {
79       System.out.println("*");
80    } else {
81       System.out.println();
82    }
83    System.out.println("    " + umelec);
84    System.out.println("    počet skladeb: " + pocetSkladeb);
85    System.out.println("    " + komentar);
86    }
87 }
```

20.5. Postup řešení

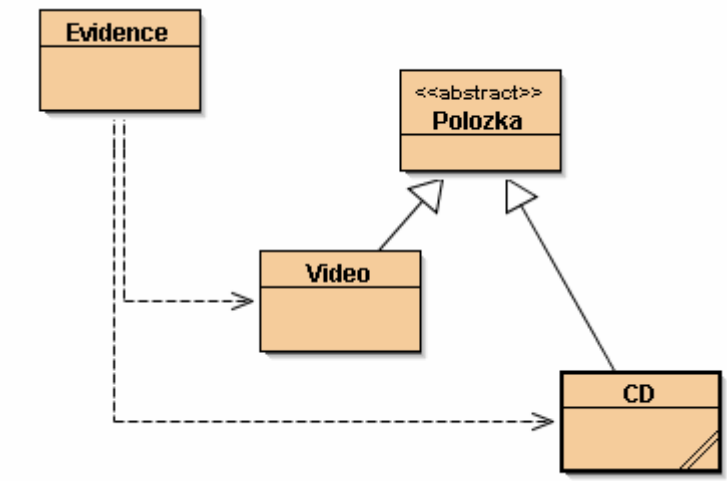
20.5.1. Vytvoření předka tříd Video a CD

Nejdříve zkusíme omezit duplicity u tříd Video a CD – pokud se podíváte do kódu těchto tříd, tak zjistíte, že mají několik stejných datových atributů a několik stejných metod. Tyto datové atributy a metody lze přesunout do společného předka a z něho je dědit. Při přesunu do společného předka je potřeba rozhodnout:

- Zda předkem bude konkrétní či abstraktní třída – rozhoduje se na základě toho, zda má význam vytvářet samostatné instance předka. V našem příkladu instance předka smysl nemá, tudíž předek bude abstraktní.

- Jak se třída předka bude jmenovat – toto je často největší problém, my použijeme nenápadité jméno `Polozka`,
- Které datové atributy se do předka přesunou – zběžným nahlédnutím do tříd `Video` a `CD` zjistíme, že mají jménem, typem a významem čtyři shodné datové atributy: `titul`, `delka`, `vlastnim` a `komentar`. Atributy `vlastnim` a `komentar` získávají počáteční obsah v konstruktoru, atributy `titul` a `delka` ale získávají počáteční obsah přes parametr konstruktoru. To komplikuje jejich přesun do předka, proto si problematiku jejich přesunu necháme do další části kapitoly.
- Které metody přesunout do předka – v obou třídách jsou stejné metody `setVlastnim()`, `getVlastnim()`, `setKomentar()` a `getKomentar()`. Přesun těchto metod též usnadňuje to, že používají pouze datové atributy `vlastnim` a `komentar`, které se též přesouvají do předka. Metoda `print()` je v každé metodě odlišná, proto ji nebudeme přesouvat. Konstruktory se nejen odlišují, ale konstruktory se na rozdíl od metod nedědí, každá třída musí mít aspoň jeden vlastní konstruktor. Z konstruktorů do konstruktoru předka přesuneme inicializaci datových atributů `vlastnim` a `komentar`.
- Jak upravit odkazy na přesunuté datové atributy – v nepřesunutých metodách mohou být odkazy na datové atributy přesunuté do předka. Tyto odkazy je potřeba upravit na metody get a set pro příslušný datový atribut (i u předka by měly být datové atributy `private`). V našem případě je potřeba upravit metody `print()` v obou třídách. Pokud by tyto úpravy byly náročné, doporučujeme znovu zvážit přesun těchto datových atributů či přesun metod.

Po úpravách diagram tříd v BlueJ obsahuje abstraktní třídu `Polozka`. Pomocí šipek s trojúhelníkem na konci je vyjádřena dědičnost mezi třídami `Polozka`, `Video` a `CD`.



Obrázek 20.3 Diagram tříd po vytvoření abstraktní třídy Polozka

Následuje kód třídy `Polozka`.

```
 1 /**
 2  * Abstraktní třída Polozka je předkem pro třídy Video a CD a
 3  * obsahuje společné prvky těchto dvou tříd.
 4  *
 5  * @author     Luboš Pavlíček
 6  * @version    2005-jul-31
 7  */
 8 public abstract class Polozka {
 9   private boolean vlastnim;
10   private String komentar;
11
12   /**
```

```
13     * Konstruktor – inicializace datových atributů
14     */
15    protected Polozka() {
16       vlastnim = false;
17       komentar = "<bez komentáře>";
18    }
19    /**
20     * Vlož komentář k nahrávce.
21     * @param   komentar   komentář k nahrávce
22     */
23    public void setKomentar(String komentar) {
24       this.komentar = komentar;
25    }
26    /**
27     * Vrací komentář k této nahrávce.
28     *
29     * @return       komentář k nahrávce
30     */
31    public String getKomentar() {
32       return komentar;
33    }
34    /**
35     * nastavení příznaku, zda vlastníme kopii tohoto videa.
36     *
37     * @param   vlastnim   true, pokud mám nahrávku tohoto videa
38     */
39    public void setVlastnim(boolean vlastnim) {
40       this.vlastnim = vlastnim;
41    }
42    /**
43     * vrací informaci, zda vlastníme kopii tohoto videa.
44     *
45     * @return      true, pokud vlastníme kopii tohoto videa
46     */
47    public boolean getVlastnim() {
48       return vlastnim;
49    }
50 }
```

Úpravy kódu tříd *Video* a *CD* jsou triviální:

- doplní se deklarace třídy o odkaz na předka:

```
public class Video extends Polozka {
```

- vymažou se datové atributy *vlastnim* a *komentar*,
- vymažou se metody *setKomentar()*, *getKomentar()*, *setVlastnim()* a *getVlastnim()*,
- v metodě *print()* se musí upravit odkazy na datové atributy *komentar* a *vlastnim* – tyto datové atributy nejsou přístupné (jsou *private* v předkovi), proto se místo nich musí zavolat příslušné metody *getVlastnim()* a *getKomentar()*.

Celou upravenou třídu *Video* uvedeme později v této kapitole.

20.5.2. Přesun titulu a délky do předka

Při přesunu datových atributů, které se inicializují přes parametr konstruktoru (v našem případě se jedná o položky *titul* a *delka*), je potřeba zajistit jejich správnou inicializaci. I abstraktní třída má konstruktor, přestože nelze vytvořit instanci abstraktní třídy. Tento konstruktor se volá jako první příkaz v konstruktoru potomka – pro volání se používá klíčové slovo *super*:

```
        super(titul, delka);
```

Pro přístup k datovým atributům *titul* a *delka* je potřeba vytvořit přístupové metody – metody *getTitul()* a *getDelka()*.
Následuje část kódu třídy *Polozka* – deklarace datových atributů a konstruktor.

```
    private boolean vlastnim;
    private String komentar;
    private String titul;
    private int delka;

    protected Polozka(String titul, int delka) {
        this.titul = titul;
        this.delka = delka;
        vlastnim = false;
        komentar = "<bez komentáře>";
    }
```

Dále je uveden kód upravené třídy *Video*. Všimněte si volání konstruktoru předka a odkazů na přesunuté datové atributy.

```
 1 /**
 2  * Třída Video představuje jedno video (jeden pořad/nahrávku).
 3  * Informace o videu je možné ukládat a získávat. K nahrávce lze
 4  * nastavit komentář a vlastnictví kopie videa.
 5  * @author Michael Kolling and David J. Barnes
 6  * @author Luboš Pavlíček
 7  * @version 2005-jul-31 – potomek třídy Polozka
 8  */
 9 public class Video extends Polozka {
10  private String reziser;
11
12  /**
13   * Konstruktor pro vytvoření instance třídy Video.
14   * @param   titul   titul videokazety/nahrávky
15   * @param   reziser režisér nahrávky
16   * @param   delka     délka nahrávky v minutách
17   */
18  public Video(String titul, String reziser, int delka) {
19      super(titul, delka);
20      this.reziser = reziser;
21  }
22
23  /**
24   * Vypíše informace o videu do textového okna.
25   */
26  public void print() {
27  System.out.printf("video: %s (%d min)", getTitul(),
28                    getDelka());
29  if(getVlastnim()) {
30      System.out.println("*");
31  } else {
32      System.out.println();
33  }
34  System.out.println("    " + reziser);
35  System.out.println("    " + getKomentar());
36  }
37 }
```

20.5.3. Jeden seznam ve třídě Evidence

V této části zjednodušíme třídu `Evidence` – místo dvou seznamů bude jeden, bude též jenom jedna metoda pro vkládání prvků.
Využijeme toho, že třídy `Video` a `CD` mají společného předka[39] – do evidence je možné vkládat instance potomků třídy `Polozka`. Tudíž místo dvou seznamů ve třídě `Evidence` vytvoříme jeden, který bude mít prvky typu `Polozka`:

```
    private List<Polozka> seznam

    public void add(Polozka pol) {
        seznam.add(pol);
    }
```

Nyní nám již zbývá upravit metodu `vypisSeznam()` – místo procházení dvou seznamů budeme procházet jen jeden. Narazíme však na problém s metodou `print()` – ta je definována ve třídách `Video` a `CD`, není v abstraktní třídě `Polozka`. Nemá smysl ji rušit ve třídách `Video` a `CD` a přesunout ji do třídy `Polozka`, neboť potřebujeme, aby pro videa a pro CD vypisovala něco jiného. Špatným řešením je přetypovávat položky v seznamu na příslušný typ a poté volat metodu `print()`. Kód by vypadal následovně:

```
        for (Polozka pol : seznam) {
            if (pol instanceof Video) {
                ((Video)pol).print();
            }
            else {
                ((CD)pol).print();
            }
        }
```

Elegantnějším řešením je, pokud do abstraktní třídy `Polozka` doplníme předpis, že každý potomek musí mít implementovanou metodu `print()`. Povinnost mít nějakou konkrétní metodu se předepisuje pomocí tzv. abstraktních metod. V našem případě se do třídy `Polozka` doplní:

```
    abstract protected void print();
```

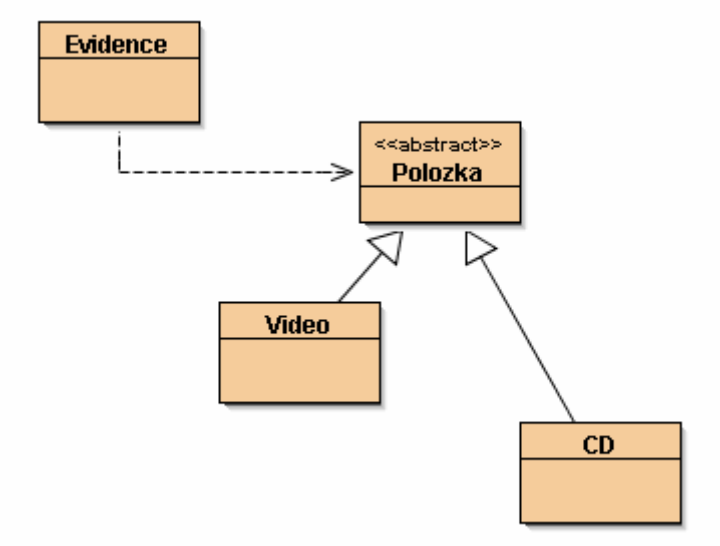
Uvedený modifikátor přístupu `protected` určuje, že i v potomcích musí mít metoda `print()` úroveň přístupnosti `protected` či širší `public`.
Kód pro výpis seznamu položek poté bude vypadat následovně:

```
        for (Polozka pol : seznam) {
            pol.print();
        }
```

Při procházení seznamu se zavolá příslušná metoda `print()` v závislosti na tom, instance které třídy se zrovna vybírá se seznamu. Je to jedna ze dvou základních variant **polymorfismu** – v kódu třídy `Evidence` je zapsaná jednou metoda `print()`, ale provádí se různé metody `print()` v závislosti na tom, zda pod typem `Polozka` se v seznamu schovává instance třídy `Video` či instance třídy `CD`. Přiřazení metody `print()` se provádí v okamžiku vytvoření jednotlivých instancí, ne v době překladu či až v době použití metody `print()`.
Použití jednoho seznamu a polymorfismu se projeví i v diagramu tříd – od třídy `Evidence` nevede šipka ke třídám `Video` a `CD`, ale pouze k abstraktní třídě `Polozka`.

---

[39] Již na začátku měly třídy Video a CD společného předka – třídu `Object`. Pokud bychom se ve třídě Evidence odkazovali na třídu `Object` jako na předka, umožnili bychom vkládání libovolných tříd do evidence, což by bylo v rozporu se smyslem aplikace.



Obrázek 20.4 Diagram tříd aplikace po úpravě třídy Evidence

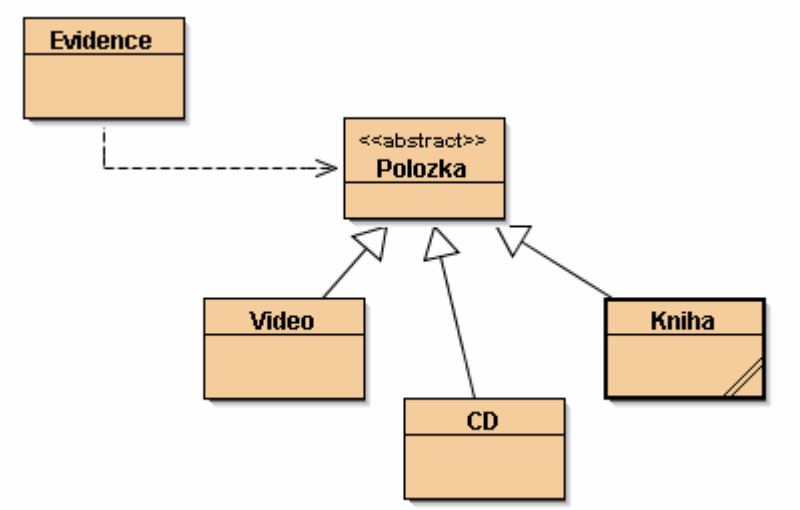
Celá upravená třída *Evidence* následuje:

```
 1 import java.util.ArrayList;
 2 import java.util.List;
 3 /**
 4  * Třída slouží pro evidenci CD a videa. Seznam všech uložených
 5  * CD a videí může být vypsán do textového okna.
 6  * Tato verze neukládá data na disk ani neposkytuje funkce pro
 7  * vyhledání nějakého titulu.
 8  *
 9  * @author Michael Kolling and David J. Barnes
10  * @author Luboš Pavlíček
11  * @version 2005-jul-31 – jeden seznam
12  */
13 public class Evidence {
14   private List<Polozka> seznam;
15
16   /**
17    * Vytvoří prázdnou evidenci.
18    */
19   public Evidence(){
20       seznam = new ArrayList<Polozka>();
21   }
22   /**
23    * Přídá polozku do evidence.
24    */
25   public void add(Polozka pol) {
26       seznam.add(pol);
27   }
```

```
28    /**
29     * Vypíše seznam všech aktuálně uložených CD a videí.
30     */
31    public void vypisSeznam() {
32    for (Polozka pol : seznam) {
33       pol.print();
34       System.out.println();   // prázdný řádek mezi položkami
35    }
36    }
37  }
```

20.5.4. Přidání knih

Nyní se pokusíme do evidence přidat knihy. U každé knihy budeme evidovat titul, autora, počet stran a stejně jako u ostatních vlastnictví a komentář. Pro řešení je výhodné využít abstraktní třídu `Polozka` obdobně jako u tříd `Video` a `CD`. Je pouze potřeba vyřešit rozpor mezi datovým atributem `delka` u abstraktní třídy a počtem stran, které chceme evidovat u knih. Nyní budeme vycházet z předpokladu, že počet stran je v podstatě stejný jako délka např. videa a budeme evidovat počet stran v datovém atributu `delka` u abstraktní třídy.
Diagram tříd bude po přidání třídy `Kniha` vypadat následovně:



Obrázek 20.5 Diagram tříd s třídou Kniha

Zdrojový kód třídy `Kniha` je následující:

```
1  /**
2   * Třída Kniha představuje jednu knihu – eviduje se titul,
3   * autor a počet stran. Ke knize lze nastavit komentář a
4   * vlastnictví výtisku.
5   * @author Luboš Pavlíček
6   * @version 2005-jul-31 – potomek třídy Polozka
7   */
8  public class Kniha extends Polozka {
9   private String autor;
10
```

```
11   /**
12    * Konstruktor pro vytvoření instance třídy Kniha.
13    *
14    * @param   titul   titul knihy
15    * @param   autor   autor knihy
16    * @param   pocetStran  počet stran knihy
17    */
18   public Kniha(String titul, String autor, int pocetStran) {
19      super(titul, pocetStran);
20      this.autor = autor;
21   }
22
23   /**
24    * Vypíše informace o knize do textového okna.
25    */
26   public void print() {
27      System.out.printf("kniha: %s (%d min)", getTitul(),
28                            getDelka());
29      if(getVlastnim()) {
30          System.out.println("*");
31      } else {
32          System.out.println();
33      }
34      System.out.println("    " + autor);
35      System.out.println("    " + getKomentar());
36   }
37 }
```

Nyní by již celá aplikace měla fungovat, není potřeba měnit další třídy. Vytvořili jsme takovou strukturu aplikace, že pro přidání dalšího druhu média není nutné měnit ani třídu *Evidence*, ani žádné jiné třídy. Dosáhli jsme větší nezávislosti a většího zapouzdření tříd, než byl původní návrh aplikace. Úspora řádek kódu je jenom vedlejším příjemným efektem těchto úprav[40].

V Javě se často používá pro specifikaci přípustných prvků nějakého seznamu **rozhraní**, které určuje dostupné metody a neomezuje dědičnou hierarchii – jedna třída může implementovat více rozhraní.

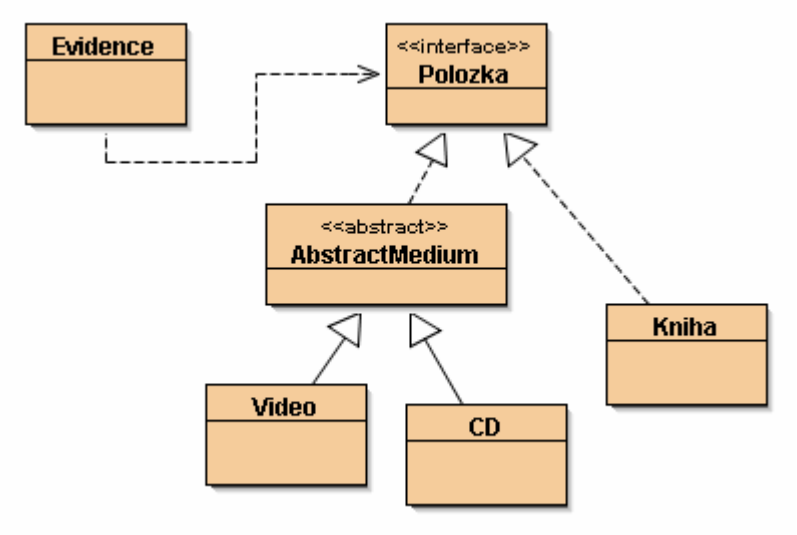
V našem případě by diagram tříd mohl vypadat následovně (třída *Kniha* není potomkem *AbstractMedium*, neboť počet stran je něco jiného než délka videa v minutách), viz diagram na obrázku 20.6.:

Do evidence je možné vkládat instance všech tříd, které implementují rozhraní *Polozka*. Některé z nich mohou využívat abstraktní třídu *AbstractMedium*, některé ne. V této struktuře má abstraktní třída význam hlavně z hlediska úspory kódu a sdílení implementace některých částí.

Deklarace rozhraní *Polozka* (soubor *Polozka.java*) vypadá v této variantě následovně:

```
public interface Polozka {
    void print();
    void setVlastnim(boolean vlastnim);
    boolean getVlastnim();
    void setKomentar(String komentar);
    String getKomentar();
}
```

[40] Zkušení programátoři se na změny, jejichž cílem je pouhá úspora kódu, dívají podezíravě – mnohdy je to za cenu snížení přehlednosti a srozumitelnosti kódu. To platí i v našem případě – autoři i čtenáři kódu musí pochopit problematiku dědičnosti a polymorfismu.



Obrázek 20.6 Diagram tříd s rozhraním a abstraktní třídou

20.6. Domácí úkoly

1. Setřiďte výpis položek dle titulu. Použijte variantu, ve které je abstraktní třída `Polozka` (tj. před použitím `interface`). Asi nejvhodnější postup je implementovat rozhraní `Comparable` ve třídě `Polozka` a v metodě `vypisSeznamu()` ve třídě `Evidence` seznam setřídit pomocí metody `Collections.sort()`.

2. Setřiďte výpis položek dle typu a v rámci typu dle titulu. Proti předchozí variantě potřebujete ve třídě `Polozka` znát i typ příslušných instancí. Nejjednodušší řešením je, pokud všichni potomci implementují metodu `getTyp()`, která bude vracet `String` s typem.

3. Setřiďte výpis položek dle titulu ve variantě s rozhraním `Polozka`. Jak se bude lišit od třídění ve variantě s abstraktní třídou `Polozka`?