3. Rozhraní

Obsah

V této kapitole se seznámíme s novou vlastností tříd a jejich instancí. Označujeme ji jako **polymorfizmus** (mnohotvárnost) a vyjadřujeme jí skutečnost, že objekty se mohou v různých situacích vydávat za instance různých typů.

Ukažme si to na příkladu ze života: jdete-li do restaurace, obsluhuje vás číšník. Kdybychom takovouto situaci programovali, pak by v našem programu vystupovali (mimo jiné) dva objekty: vy jako instance třídy **Zákazník** a obsluhující jako instance třídy **Číšník**. I číšníci však mají občas volno a mohou si pak dojít do restaurace. Pak ale vystupují jako instance třídy **Zákazník** a obsluhuje je jiná instance třídy **Číšník**.

Obdobných situací bychom v životě našli celou řadu. V této kapitole si ukážeme její ekvivalent ve světě grafických objektů a pláten, na která se tyto objekty kreslí.

S novým tématem otevřeme i nový projekt, kde některé třídy přibudou, jiné budou chybět a definice ostatních budou alespoň upravené. Abyste mohli sledovat výklad, stáhněte si z adresy http://vyuka.pecinovsky.cz/ projekt **03_Tvary_1**, kolem kterého se bude výklad na počátku této kapitoly točit. V dalším textu pak postupně použijeme ještě projekty **03_Tvary_2** a **03_Tvary_3**, které najdete na stejné stránce.

3.1 Kreslíme jinak

V našich dosavadních příkladech jsme se smiřovali s tím, že pohybující se tvary při svém pohybu mazaly vše, co jim přišlo do cesty, a pokud jsme chtěli nějaký umazaný nebo dokonce smazaný tvar vidět, museli jsme jej explicitně požádat o to, aby se překreslil. Kreslené objekty jeden o druhém nevěděly a ani plátno nevědělo nic o tom, co je na něm nakresleno.

V této kapitole naše kreslící možnosti trochu vylepšíme. Opustíme třídu **Plátno**, kterou jsme používali doposud, a přestaneme říkat našim obrázkům, aby se nakreslily. Místo toho začneme používat třídu **AnimPlatno**, která se o nakreslení našich obrázků na plátno postará sama. tj. sama je ve vhodnou chvíli požádá, aby se nakreslili..

Třída **AnimPlatno** obrací celou naši dosavadní filozofii kreslení obrázků naruby. Doposud jsme to dělali tak, že když jsme chtěli objekt nakreslit, požádali jsme třídu **Platno** o nějaké plátno, tomu jsme nastavili kreslící barvu a nařídili mu, aby náš objekt touto barvou nakreslilo (podívejte se, jak byly definovány metody **kresli()**). Třída **AnimPlatno** na to však jde úplně jinak. Její instance není pouze pasivním objektem, jenž nám umožňuje kreslit na obrazovku, ale chová se spíše jako manažer, který dostane nějaké objekty do správy a dohlíží na to, aby byly všechny správně nakresleny. Ví, které objekty jsou na plátně, a vždy, když se dozví, že se něco změnilo, tak zařídí, aby obrázek na plátně odpovídal skutečnosti.

Aby se třída mohla takto chovat, potřebuje vědět dvě věci:

- ☞ které objekty se mají na plátně zobrazovat,
- ☞ kdy se stav některého z nakreslených objektů změnil natolik, že je třeba plátno překreslit.

Jak jsem již řekl, nyní již nebudeme našim objektům říkat, aby se nakreslily. Budeme-li chtít, aby byl náš objekt na plátně nakreslen, požádáme třídu **AnimPlatno**, aby jej zařadila mezi objekty, o jejichž vykreslení se stará. Musíme jí přitom slíbit, že náš objekt se bude schopen na požádání nakreslit – bude mít ve své „sbírce“ metodu, jež bude schopna převzít od třídy **AnimPlatno** kreslítko (objekt třídy **Graphics2D**) a s jeho pomocí se nakreslit. Kdykoliv se nyní na plátně cokoliv změní, třída bude schopna zařídit překreslení všech objektů včetně těch trochu (nebo úplně) odmazaných.

3.2 Rozhraní

Jistě vás zajímá, jak takový slib realizujeme. Pomůžeme si zvláštním druhem třídy označované jako **rozhraní** (**interface**). Tento termín jsme již zavedli, když jsme hovořili o zapouzdření a o dokumentaci. Tehdy jsme si řekli, že „rozhraní třídy budeme chápat jako množinu informací, které o sobě třída zveřejní“. Způsob, jakým je rozhraní naprogramováno jsme označili jako **implementaci**. Tato terminologie zůstane v platnosti i v okamžiku, kdy budeme hovořit o rozhraní jako o zvláštním druhu třídy.

Rozhraní totiž na rozdíl od obyčejných tříd neříká vůbec nic o tom, jak budou jednotlivé metody implementovány. Pouze deklaruje seznam veřejných metod, které budou jeho „instance“ implementovat. V našem novém projektu je např. definováno rozhraní **IKresleny**. Jeho definice vypadá (bez dokumentačních komentářů) následovně:

```
public interface IKresleny
{
    void kresli( Graphics2D g );

}//public interface IKresleny
```

Jinými slovy: všechny objekty, které se budou chtít vydávat za „instance“ rozhraní **IKrelseny**, budou muset implementovat metodu `kresli(Graphics2D)`.

> **Terminologická poznámka:**
> V tomto kurzu budu všechna rozhraní označovat identifikátory začínajícími na **I**. Přiznám se, že jsem tento zvyk převzal z dob, kdy jsem ještě programoval v C++, kde se nejrůznější předpony používaly poměrně často. V javové komunitě sice zdobení identifikátorů prefixy nebývá zvykem, ale pro začátečníky je výhodné, když na první pohled poznají, zda se jedná o identifikátor třídy nebo rozhraní. Proto se v tomto textu s podobnými „návodnými značeními“ setkáte ještě několikrát.

V definici rozhraní si všimněte dvou odchylek od definice standardní třídy:

- ☞ v hlavičce je místo klíčového slova **class** použito klíčové slovo **interface**, jímž překladači oznamujeme, že nepřekládá standardní třídu ale rozhraní.
- ☞ v těle rozhraní je zapsána pouze hlavička metody ukončená středníkem. Tělo metody je totiž záležitostí implementace, a o tu se rozhraní nestará.

Ti bystřejší z vás si možná ještě všimli toho, že u metody není uveden modifikátor **public**. Je to proto, že všechny metody deklarované v rozhraní jsou automaticky veřejné. Překladač proto na explicitním zapsání tohoto modifikátoru netrvá.

Opakování: třídy a jejich instance

> **Poznámka:**
> Tato a následující část je určena pro ty, kteří ke každému **jak** chtějí znát také **proč**. Pokusím se vám v ní naznačit trochu z pozadí mechanizmu rozhraní a jejich tříd. Kdyby vám připadala moc teoretická, tak ji přeběhněte a vraťte se k ní, až si budete myslet, že vám bude užitečná.

Ti bystřejší si možná všimli, že jsem na počátku této podkapitoly uvedl dvakrát slovo instance v uvozovkách. Víte proč? Rozhraní totiž nemůže mít žádné instance, přesněji řečeno žádné vlastní instance.

Než se pustím do zdůvodňování, tak pro stručně zopakuji, co jsme si v první kapitole vyprávěli o instancích a vlastních instancích. Říkali jsme si, že všechny objekty se dělí do tříd. Naše dosavadní zkušenosti s objektovým programováním nám navíc napovídají, že třída (mimo jiné) definuje předpis, podle nějž se vytvářejí objekty – její instance, popisuje vlastnosti těchto objektů a definuje jejich chování.

Vysvětlili jsem si, že pro skupiny objektů, které mají nějaké speciální vlastnosti, můžeme definovat speciální třídu, která bude popisovat zvláštnosti oné skupiny objektů. Tato třída bude podtřídou (dceřinnou třídou) oné obecnější (rodičovské) třídy. Dceřinná třída pouze přidá (případně upraví) některé rysy, ale nic z toho, co definovala rodičovská třída, nezruší. Všechny instance dce-

řinné třídy proto budeme moci kdykoliv vydávat za instance rodičovské třídy, protože mají všechny potřebné vlastnosti jejích instancí.

Zavedli jsme ještě pojem **vlastní instance**, jímž jsme označovali instance, které daná třída přímo „porodila“. Řekli jsme si, že objekt může být vlastní instancí pouze jediné třídy, avšak můžeme jej považovat za instanci kteréhokoliv z jejích rodičů.

Instance rozhraní

Vraťme se nyní k rozhraní. Před chvílí jsem říkal, že rozhraní nemůže mít vlastní instance. Nemá totiž žádnou implementační část, která by mu je umožnila vytvořit. To mohou až třídy, které toto rozhraní implementují, tj. definují (=implementují) těla všech deklarovaných metod a definují také konstruktor, pomocí nějž je možné požadovanou instanci vytvořit. Třídy, které budou implementovat dané rozhraní, můžeme považovat za jeho dceřinné třídy. Proto jejich vlastní instance mohou vystupovat jako instance implementovaného rozhraní.

Abychom přestali teoretizovat, vraťme se k našemu příkladu s animačním plátnem. Řekli jsme si, že k tomu, aby plátno mohlo správně plnit svoji manažerskou funkci, potřebuje, aby se obrazce, které se hlásí do jeho správy, uměly nakreslit dodaným kreslítkem. Definovali jsme proto rozhraní **IKresleny**, jehož instance to umějí.

Vzápětí jsme si řekli, že rozhraní nemůže mít vlastní instance, takže se musí jednat o instance některé z jeho dceřinných tříd, které toto rozhraní implementují. Po třídách, které implementují dané rozhraní, chceme jediné: aby definovali (=implementovali) všechny jeho metody. Dál ať si dělají, co chtějí. Budou-li tedy třídy **Elipsa**, **Obdelnik**, **Trojuhelnik** a **Smrk** definovat metodu **void kresli(Graphics2D)**, budeme je moci vydávat za třídy implementující rozhraní **IKresleny** a plátno bude ochotno přidat jejich instance mezi spravované.

Implementace rozhraní

Třída se musí k implementaci rozhraní veřejně přihlásit, aby překladač mohl zkontrolovat, že všechny metody rozhraní opravdu implementovala. Učiní to tak, že na konec své hlavičky přidá klíčové slovo **implements** následované názvem rozhraní, které implementuje.

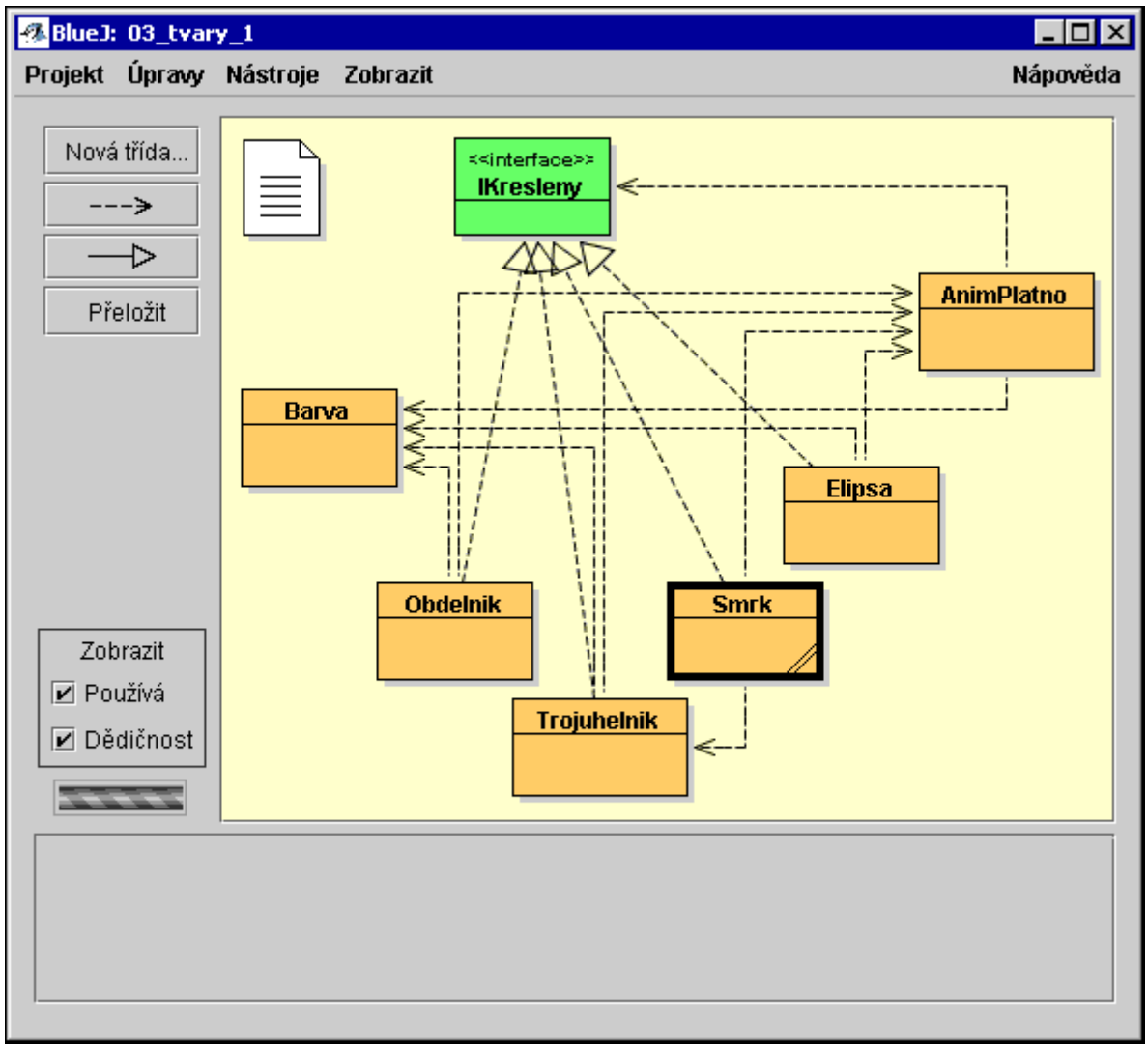
Pokud bychom tedy chtěli, aby třída **AnimPlatno** byla ochotna zařadit instance třídy **Smrk** mezi ty, které zobrazuje, museli bychom upravit definici třídy **Smrk** tak, aby implementovala rozhraní **IKresleny**. Její hlavička by pak vypadala následovně:

```
public class Smrk implements IKresleny
```

V těle metody bychom pak samozřejmě museli definovat všechny metody, které dané rozhraní požaduje – v našem případě metodu **void kresli(Graphics2D)**.

3.3 Diagram tříd a vztah Používá

Konec řečí – jdeme programovat. Otevřete si projekt **03_tvary_1** a prohlédněte si jeho diagram tříd.

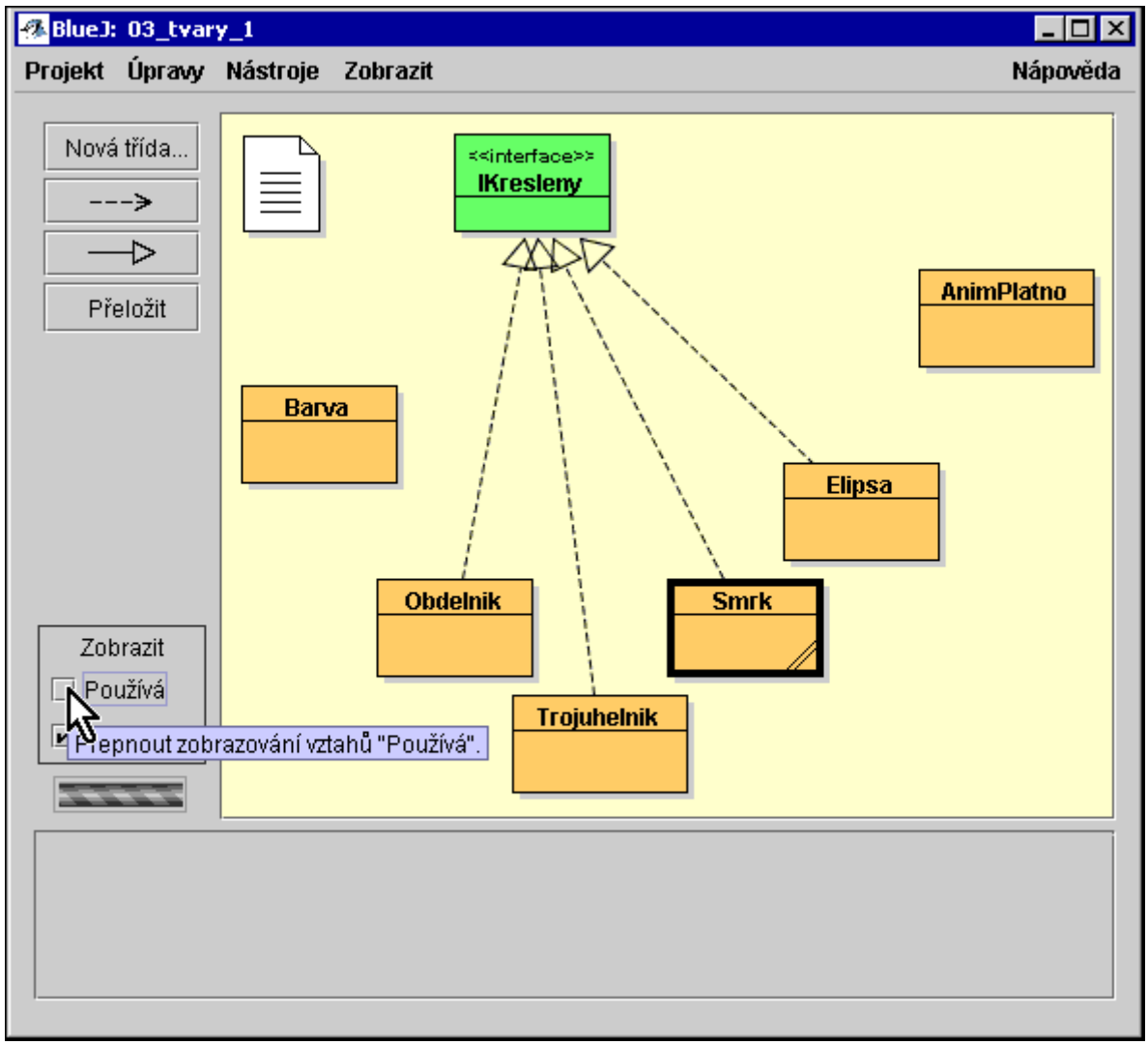


Obrázek 3.1: *Diagram tříd projektu 03_tvary_1*

Jak vidíte, tento diagram tříd je trochu složitější než diagramy, s nimiž jsme se setkávali doposud, protože v něm přibylo nejenom rozhraní **IKresleny**, ale také spojnice od tříd k němu. Spojnic již začíná být tolik, že je těžké se v nich orientovat.

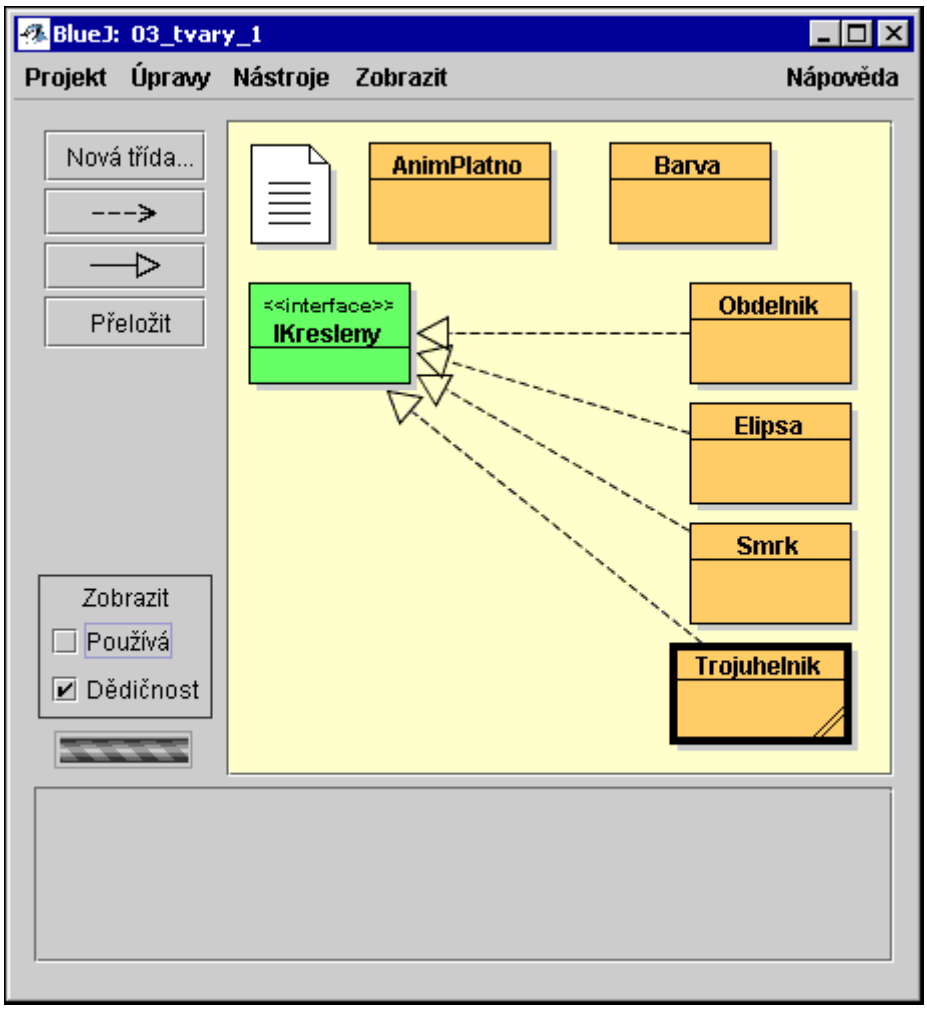
Využijeme možnosti nezobrazovat šipky závislostí a zobrazení zjednodušíme. Podívejte se do levého panelu aplikačního okna. V bloku **Zobrazit** jsou dvě zaškrtávací políčka ovlivňující zobrazování šipek. Zaškrtnutí políčka by mělo symbolizovat, zda je uvedený druh šipek zobrazován či nikoliv. (Mělo by, ale občas se mu to nedaří – nedejte se tím vyvést z míry.)

Klepněte na políčko **Používá** a BlueJ smaže všechny šipky označující použití jedné třídy v definicích atributů a metod jiné třídy.



Obrázek 3.2: *Diagram tříd projektu 03_tvary_1 s vypnutým zobrazováním vztahu Používá*

Využijeme zjednodušení a trochu ikony tříd přesuneme, abychom mohli v budoucnu jednoduše přidávat další rozhraní. Po těchto úpravách bude diagram tříd vypadat následovně:



Obrázek 3.3: *Zjednodušený diagram tříd*

Jak jste se sami přesvědčili, při větším počtu tříd a vazeb mezi nimi je diagram s vypnutou vazbou **Používá** mnohem přehlednější. Informace o tom, která třída kterou používá přitom nejsou většinou tím, co nás na diagramu nejvíce zajímá. Proto v dalších diagramech již nebudu nechávat zobrazovat šipky naznačující používání jedněch tříd jinými třídami.

Úkol:
Otevřete zdrojový soubor třídy **Smrk** a ověřte, že se oproti verzi vytvořené v minulé kapitole změnila pouze hlavička třídy a definice metody **kresli**.

3.4 Zobrazení rozhraní v diagramu tříd

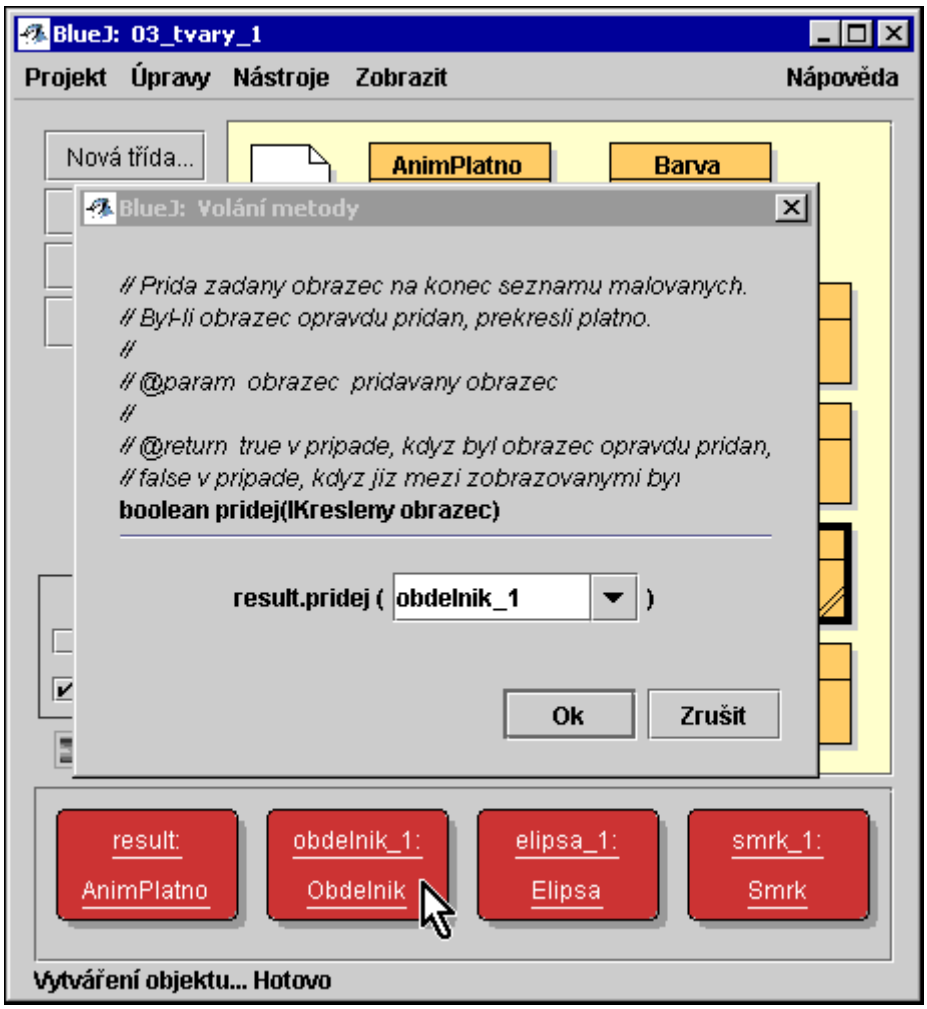
V horní části diagramu vidíte zelenou ikonu rozhraní, v níž je kromě názvu rozhraní zobrazen i text (tzv. **stereotyp**) «interface». Od tříd, které toto rozhraní implementují, vedou k rozhraní čárkované šipky s trojúhelníkovou hlavičkou.

> **Terminologická poznámka:**
> Termín **stereotyp** je v jazyku UML zaveden pro pomocné texty, které se píší ve «francouzských uvozovkách» a které blíže specifikují druh daného objektu (v našem případě třídy).

Zobrazovací poznámka:
Ikona rozhraní se zobrazí zeleně pouze těm, kdo mají staženou moji lokalizaci s „rozumně novým“ datem. Původní BlueJ používá pro ikony všech druhů tříd stejnou barvu (rozlišují rozhraní pouze stereotypem). Protože jsem se domníval, že odlišné barvy zvýší přehlednost a informační obsah diagramu, přidal jsem do lokalizace barevné odlišení rozhraní.

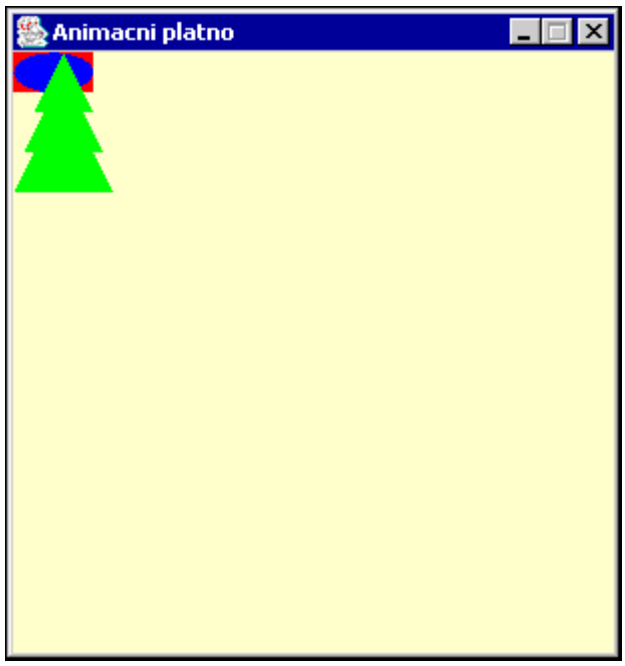
Z kteréhokoliv z obrázků 3.1 až 3.3 můžeme vyčíst, že jak všechny naše základní geometrické třídy, tak i třída **Smrk** implementují rozhraní **IKresleny** a můžeme je proto zařadit mezi objekty spravované třídou **AnimPlatno.** Vyzkoušejme si to – postupujte podle následujícího návodu:

1. V místní nabídce třídy **AnimPlatno** zadejte příkaz **AnimPlatno getPlatno()**.
2. V následně otevřeném dialogovém okně klepněte na text
   Objekct result = <object reference>
   a stiskněte pak tlačítko **Získat odkaz**. V seznamu odkazů se objeví odkaz **result** na instanci třídy **AnimPlatno**.Dialogové okno pak zavřete.
3. Za použití bezparametrických konstruktorů vytvořte postupně instance tříd **Obdelnik**, **Elipsa** a **Smrk**.
4. Požádejte instanci třídy **AnimPlátno** o zařazení obdélníku, elipsy a stromku mezi zobrazované tvary. Dosáhnete toho tak, že v místní nabídce instance třídy **AnimPlatno** zadáte povel **boolean pridej(IKresleny).** Tím otevřete dialogové okno pro zadávání parametrů, které po vás bude chtít parametr typu **IKresleny**. Protože všechny naše třídy toto rozhraní implementují, můžeme jejich instance vdávat za jeho instance. Klepněte proto na odkaz obdélníka (tím jej zadáte jako parametr – viz obr. 3.4) a své zadání potvrďte.



Obrázek 3.4: *Přidání obdélníku mezi zobrazované obrazce.*

5. Totéž pak udělejte s instancí elipsy a smrku. Na závěr by mělo animační plátno vypadat následovně:



Obrázek 3.5: *Animační plátno po přidání obdélníku, elipsy a smrku.*

Úkol:
Nyní zkuste s jednotlivými instancemi manipulovat. Posouvejte je (využijte jejich metod **posunXXX** a **pomaluXXX**), měňte jejich velikost (**setSirka**, **setVyska** a **setRozmer**) a ověřte, že budou na plátně neustále správně zobrazeny a nebudou se vzájemně mazat.

3.5 Definice nového rozhraní

Naše dosavadní tvorba dalších tvarů nebyla příliš efektivní. Když jsme navrhli nějaký nový tvar (smrk, sněhuláka, panáka, domek, …), tak jsme v příslušné třídě definovali také všechny metody pro manipulaci s tímto objektem. Přitom tyto metody byly pro většinu objektů téměř stejné – často stačilo vzít jejich definice z jiné třídy a do právě definované třídy je zkopírovat.

Mezi důležité programátorské zásady patří **vyvarovat se kopírování kódu**. Kopie kódu totiž přinášejí obdobné problémy jako magické hodnoty (literály), o nichž jsem hovořil v kapitole 2.16 Konstanty. Když totiž později zjistíte, že jste něco naprogramovali špatně a nebo se změní zadání a je potřeba kód změnit, musíte projít všechna místa s tímto kódem a všude zanést stejnou opravu, což je potenciální zdroj nepříjemných chyb.

Opakování částí kódu se dá zamezit různými metodami. Nyní si ukážeme jednu z nich. Ukážeme si, jak lze zařídit, aby se všechny naše dále vytvořené objekty mohly pohybovat po plátně aniž bychom museli v jejich třídách definovat metody, které tento pohyb realizují.

Představte si, že bychom měli k dispozici třídu **Přesouvač**, které bychom vždy předali objekt a ona by jej plynule přesunula. Aby to mohla učinit, museli bychom jí ale předat objekt, který je takového přesunu schopen.

Požadavek na přemístění objektu můžeme obecně zadat dvěma způsoby – záleží na tom, zda jej potřebujeme přesunout na zadané souřadnice, nebo zda jej potřebujeme odsunout o požadovanou vzdálenost. V podstatě však jde o stejný problém, protože kdybychom uměli zjistit současné sou-

řadnice objektu, mohli bychom jednu metodu realizovat pomocí druhé – např. posun o zadanou vzdálenost řešit tak, že bychom ze současných souřadnic a požadovaného posunu spočetli nové souřadnice a na ty bychom objekt přemístili nebo naopak posun na zadané souřadnice realizovat tak, že bychom spočetli vzdálenost současných a cílových souřadnic a o tu pak objekt posunuli.

Z předchozích úvah nám tedy vyplývá, že k tomu, abychom mohli objekty přesouvat, bychom potřebovali, aby tyto objekty měly implementované následující metody (jinými slovy: aby uměly správně reagovat na následující zprávy):

- **int getX()**, která vrátí aktuální x-ovou souřadnici objektu,
- **int getY()**, která vrátí aktuální y-ovou souřadnici objektu,
- **int setXY(int, int)**, která přesune objekt na zadanou pozici.

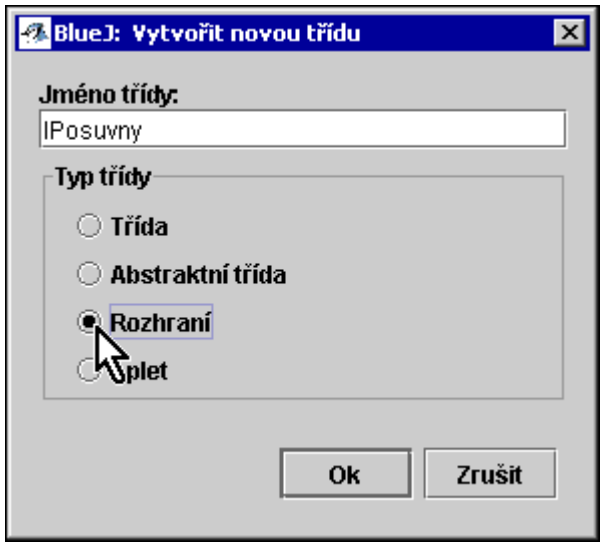
Definujme proto rozhraní **IPosuvny**, v němž naše požadavky převedeme do řeči, jíž je počítač schopen rozumět:

```
/**
 * Rozhrani IPosuvny popisuje metody, jez musi implementovat objekty,
 * ktere maji byt schopny posunu po animovanem platne.
 *
 * @author     Rudolf Pecinovsky
 * @version    1.01, 28.1.2003
 */
public interface IPosuvny
{
    /**
     * Vrati x-ovou souradnici posuvneho objektu,
     * tj. x-ovou souradnici leveho horniho rohu opsaneho obdelniku.
     *
     * @return x-ova souradnice leveho horniho rohu opsaneho obdelniku
     */
     public int getX();


    /**
     * Vrati y-ovou souradnici posuvneho objektu,
     * tj. y-ovou souradnici leveho horniho rohu opsaneho obdelniku.
     *
     * @return y-ova souradnice leveho horniho rohu opsaneho obdelniku
     */
     public int getY();


    /**
     * Nastavi novou polohu posuvneho objektu, tj. presune jej tak,
     * aby levy horni roh opsaneho obdelniku byl na zadanych souradnicich.
     *
     * @param x  nova x-ova pozice leveho horniho rohu opsaneho obdelniku
     * @param y  nova y-ova pozice leveho horniho rohu opsaneho obdelniku
     */
    public void setXY(int x, int y);


}//public interface IPosuvny
```

Jak vidíte, definice samotného rozhraní je nesmírně jednoduchá. Pokud bychom odstranili dokumentační komentáře, zbylo by nám pouhé:

```
public interface IPosuvny
{
    public int getX();
    public int getY();
    public void setXY(int x, int y);

}//public interface Iposuvny
```

Jednoduché je i zanesení definovaného rozhraní do diagramu tříd. Stačí na levém panelu stisknout tlačítko **Nová třída** a v následně otevřeném dialogovém okně **Vytvořit novou třídu** zadat název definovaného rozhraní, nastavit přepínač **Typ třídy** do polohy **Rozhraní** a své zadání potvrdit.



Obrázek 3.6: *Vytváříme nové rozhraní*

Nově vytvořené rozhraní se pak objeví v diagramu tříd a v jeho ikoně bude nad jeho názvem zapsán ještě **stereotyp** «interface» obdobně, jako tomu bylo u rozhraní **IKresleny** na obrázcích 3.1 až 3.3.

Nyní již do něj stačí doplnit potřebné deklarace a přidat příslušné komentáře (nevynechávejte je!) a jste hotovi.

> **Poznámka:**
> Originální BlueJ ikony obyčejných tříd a rozhraní barevně neodlišuje. Chcete-li rozhraní odlišit nejenom stereotypem, ale také barevně, stáhněte a instalujte si moji lokalizaci.

3.6 Začlenění hotových tříd do projektu

Rozhraní tedy máme připravené – zbývá nám třída **Presouvac**. To jak ji naprogramovat, vám nyní ještě vysvětlovat nebudu (dozvíte se to v příštích lekcích) – prozatím vám ji ještě dám hotovou. Najdete ji spolu s rozhraním **IPosuvny** v druhém balíku programů, který se můžete stáhnout na adrese http://vyuka.pecinovsky.cz/.

Předpokládám, že si chcete vyzkoušet, jak se dodatečně implementuje rozhraní stávajícími třídami. Je-li tomu tak, neměňte projekt, ale vezměte ze staženého projektu pouze zdrojový kód třídy **Presouvac** a zkopírujte jej do složky se stávajícím projektem. Ukážeme si dva možné postupy.

Poznámka:
Nechce-li se vám provádět vše sami a dáváte-li přednost pouhému prohlížení hotového kódu, můžete si někam rozbalit celý projekt a rovnou se do něj přesunout. Doporučuji však dát přednost vlastním pokusům.

1. Zkopírování zdrojového souboru v externím správci souborů

První možností je využít Průzkumníka nebo jiného správce souborů (např. Total Commander) a zkopírovat zdrojové soubory v něm. Výhodou tohoto postupu je, že můžete zkopírovat několik souborů najednou, jeho nevýhodou je naopak to, že BlueJ nebude o nově se objevivších souborech vědět. Budete proto muset zavřít projekt a znovu jej otevřít.

Potlačení šipek Používá:
Ve verzi 1.2.2 je bohužel drobná chyba: po opětném otevření projektu se znovu zobrazí všechny šipky závislostí, přestože je toto zobrazování podle stavu zaškrtávacího políčka vypnuto. Musíte proto znovu klepnout v poli **Zobrazit** na zaškrtávací políčko **Používá**. Políčko se pak sice zaškrtne, ale šipky závislostí zmizí.

2. Přímé vložení třídy z externího zdroje

Nebude-li vám vadit, že budete muset soubory přenášet po jednom, můžete využít i možnost zkopírovat soubory přímo v prostředí BlueJ. Stačí nabídce **Úpravy** zadat příkaz **Nová třída ze souboru**... a v následně otevřeném dialogovém okně pak příslušný soubor nalézt a zadat.

V tomto případě BlueJ nově dodaný soubor ihned zanese do diagramu tříd. Pravda, místo které pro ikonu vybere, se vám asi líbit nebude, ale není problém přesunout ikonu do pozice, která vám bude vyhovovat.

Úkol:
Zkuste do projektu přidat i třídu **Snehulak**, kterou jste vytvořili v minulé kapitole. Upravte její zdrojový kód podle třídy **Smrk** (tj. upravte její hlavičku a definici metoda **kresli**) a ověřte, že její instance mohou být zařazeny mezi instance spravované animačním plátnem.

3.7 Přesouvač

Třídu **Presouvac** jsme tedy začlenili do projektu. Pojďme se s ní trochu seznámit. Třída má dva veřejné konstruktory a dvě veřejné metody. Její rozhraní (tj. to, co o sobě třída zveřejňuje) bychom mohli ve stručnosti popsat následovně:

```
/** Trida slouzi k pohybu s instancemi trid implementujicich rozhrani IPosuvny. */
public class Presouvac
{
    /** Vytvori presouvac, ktery bude presouvat objekty rychlosti 1. */
    public Presouvac()

    /** Vytvori presouvac, ktery bude presouvat objekty zadanou rychlosti. */
```

```
    public Presouvac( int rychlost )

    /** Presune zadany objekt o pozadovany pocet bodu. */
    public void presunO( IPosuvny objekt, int doprava, int dolu )

    /** Presune zadany objekt do pozadovane pozice. */
    public void presunNa( IPosuvny objekt, int x, int y )
}//public class Presouvac
```

Všimněte si, že prvním parametrem obou přesunových metod je objekt typu **IPosuvny**. Podle toho, co jsme si říkali před chvílí, může tedy těmto metodám předat jako parametr instanci kterékoliv z tříd, jež implementují rozhraní **IPosuvny**. Prozatím však žádnou takovou třídu nemáme – nejvyšší čas, abychom si nějakou „opatřili“.

3.8 Implementace rozhraní stávajícími třídami

Víme, že třídy, jejichž instance jsme doposud „prohánělí“ po plátně, tj. třídy **Obdelnik**, **Elipsa**, **Trojuhelnik** a **Smrk** již rozhraní **IPosuvny** de facto implementují (mají požadované metody ve svém repertoáru). Nyní bychom měli stvrdit tuto implementaci i de iure, tj. přesně podle syntaktických pravidel tuto skutečnost veřejně vyhlásit.

Postup je velice jednoduchý: stačí na levém panelu stisknout tlačítko s trojúhelníkovou šipkou. BlueJ vás pak požádá, abyste klepli na ikonu třídy, která bude dědit (v našem případě implementovat), tj. jedné z tříd **Obdelnik**, **Elipsa**, **Trojuhelnik** a **Smrk**, a poté na ikonu její rodičovské třídy – v našem případě rozhraní **IPosuvny**. Po těchto dvou kliknutích BlueJ natáhne mezi třídami šipku od implementující třídy k čerstvě definovanému rozhraní.

Podíváte-li se na zdrojový kód upravených tříd, uvidíte, že se hlavičky tříd rozrostly o klauzuli **implements IKresleny**. Např. hlavička třídy **Obdelnik** má nyní tvar:

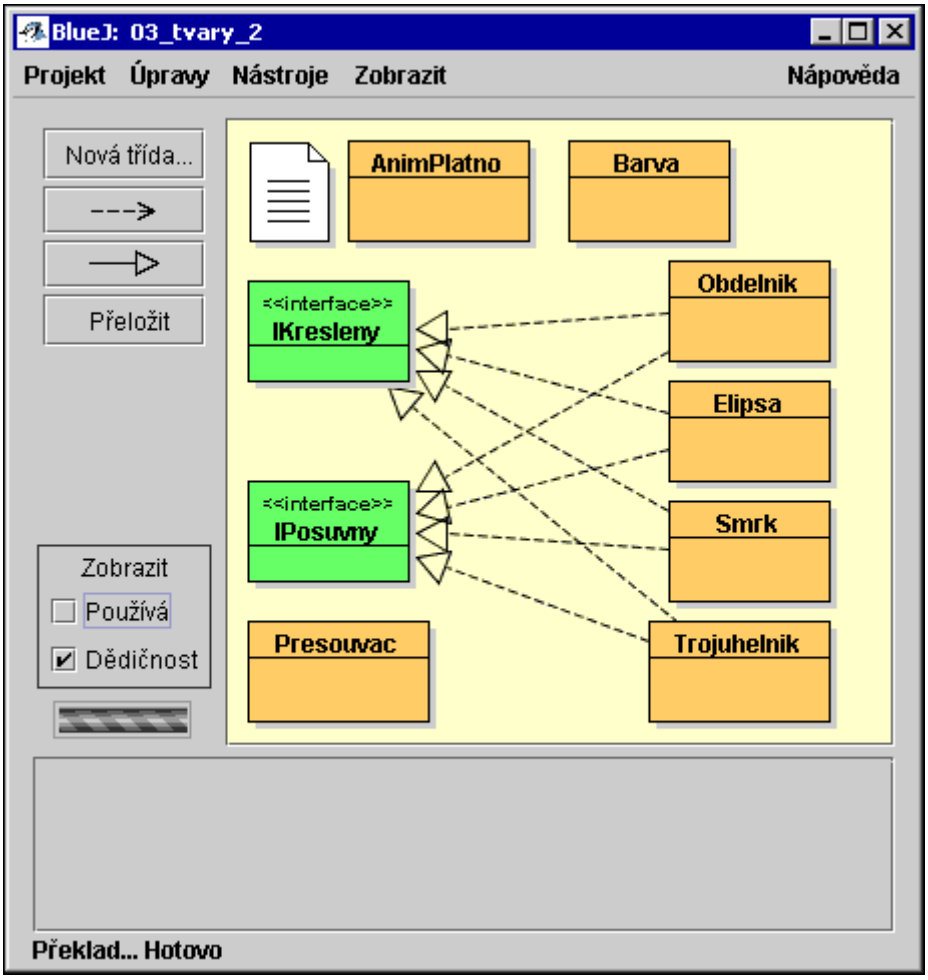
```
public class Obdelnik implements IKresleny, IPosuvny
```

Z ní poznáte (a překladač samozřejmě také), že třída **Obdelnik** nyní implementuje dvě rozhraní: již dříve implementované **IKresleny** a nove **IPosuvny**. Pamatujte si, že třídy mohou implementovat libovolný počet rozhraní. Jediné, na co musíme dát pozor, je to, aby byly metody požadované rozhraními, ve třídách skutečně implementovány. (Pokud však něco přehlédneme, překladač náš na to upozorní.)

> **Pozor na češtinu:**
> Ve verzi 1.2.2 je bohužel drobná chyba: Jak jsme si již řekli, BlueJ se občas nekamarádí s češtinou. Jedním z nepříjemných důsledků je to, že u souborů obsahujících znaky s diakritikou BlueJ občas neopraví po natažení implementační šipky jejich zdrojový kód, takže musíte příslušnou klauzuli **implements xxx** vložit do zdrojového kódu ručně.
>
> Druhou možností je využít služeb některého editoru a nechat si veškerou češtinu ze zdrojového kódu odstranit. Již jednou jsem pro tento účel doporučoval vynikající český freeware editor PSPad, který si můžete stáhnout na adrese http://pspad.zde.cz.

Máte-li dostatečně velkou obrazovku a můžete-li si zobrazit současně diagram tříd a zdrojový kód některé z nich, můžete se přesvědčit, že po natažení šipky se zdrojový kód okamžitě upraví, a to i v případě, kdy je zdrojový soubor zrovna otevřen v editoru. Vyzkoušejte si vše na instancích tříd, které jsou již definované a o nichž víte, že rozhraní **IPresuvny** implementují, tj. na instancích tříd **Obdelnik**, **Elipsa**, **Trojuhelnik** a **Smrk**.

Když si ještě ikony tříd trochu přeskupíme, abychom se s obrázkem vešli do vyhrazeného prostoru, mohl by po nastavení všech implementačních závislostí vypadat diagram tříd např. následovně:



Obrázek 3.7: *Implementace druhého rozhraní*

Úkol 1:

Nechte na plátně znovu zobrazit několik geometrických objektů (pokud jste zapomněli, jak se to dělá, zopakujte si postup z kapitoly *3.4 Zobrazení rozhraní v diagramu tříd*). Vytvořte pak instanci přesouvače (použijte bezparametrický konstruktor) a požádejte ji, ať některou z vytvořených posuvných instancí přesune např. na pozici [100;100], tj. zavolejte její metodu **presunNa(IPosuvny,int,int)** a předejte ji jako parametr některou posuvnou instanci a souřadnice cílové pozice. Nezapomeňte přitom, že objektový parametr můžete zadat i tak, že místo psaní názvu jednoduše klepnete v zásobníku odkazů na odkaz na daný objekt.

Požádejte instanci přesouvače o totéž pro jinou instanci a nechte jej s jejím obrazem přejet přes obraz prve přesunuté instance.

Vytvořte instanci přesouvače přesouvající objekty jinou rychlostí a porovnejte její akce a akcemi prvé instance.

Úkol 2:
Zřiďte, aby rozhraní **IPosuvny** implementovaly i vaše třídy **Snehulak**, **Domek** a **Panak**. Vytvořte jejich instance, zobrazte je na plátně a požádejte některého posunovače, aby s nimi zahýbal.

3.9 Další možnosti třídy AnimPlatno

Hierarchie kreslených objektů

Přestaneme na chvíli pracovat a budeme si hrát. Nejprve si připomeneme, že metoda **pridej(IKresleny)** vrací informaci o tom, jestli jsme objekt mezi spravované opravdu přidali a nebo jestli mezi nimi již byl. Bude-li se vám tato informace v programu hodit, můžete ji použít, nebudete-li ji potřebovat, můžete se tvářit, že vám metoda žádnou informaci nevrátila.

Třída **AnimPlatno** nabízí vedle možnosti přidání objektu mezi spravované také možnost jeho přesného zařazení do hierarchie kreslených objektů. Objekty jsou na plátně „naskládány“ jako papíry na hromádce – některé jsou na vrchu, jiné jsou pod nimi. Při přidávání objektu mezi spravované si můžeme vybrat, kam do této hierarchie spravovaný objekt umístíme.

Použijeme-li místo metody **pridej(IKresleny)** metodu **pridejNad(IKresleny,IKresleny**), přidá se objekt na nějž odkazuje druhý parametr zařazen těsně nad objekt, na nějž odkazuje první parametr. Použijeme-li naopak metodu **pridejPod(IKresleny,IKresleny**), bude objekt, na nějž odkazuje druhý parametr zařazen těsně pod objekt, na nějž odkazuje první parametr.

V obou případech je nutné, aby objekt zadávaný v prvém parametru (tj. objekt, vůči němuž druhý objekt umísťujete) již patřil mezi spravované. Nebude-li tomu tak, vyvolá program výjimečnou situaci a jeho chod se zastaví.

Druhý parametr mezi spravovanými objekty být může a nemusí. Pokud mezi nimi nebude, přidá se mezi ně na požadovanou pozici. Bude-li již mezi spravovanými, tak se pouze přemístí v hierarchii na požadovanou pozici.

Úkol:
Přidejte instance kreslitelných tříd na různé pozice v hierarchii spravovaných objektů a potom s nimi posouvejte pomocí nějakého přesouvače. Ověřte, že při pohybu jejich obrazy opravdu zajíždějí pod obrazy instancí, které jsou v hierarchii nad nimi a naopak přejíždějí přes obrazy instancí, které jsou v hierarchii pod nimi.

Změna rozměrů plátna

Když jsme v minulých kapitolách změnili rozměr plátna, přišli jsme o všechny nakreslené obrázky. Po každé změně rozměru se totiž plátno přemazalo obdobně, jako kdybychom je vyměnili za nové. Ukažme si, že nové, animační plátno nám již obrázky našich instancí při změně rozměru nesmaže.

Naopak: zajedem-li s obrázkem instance za okraj plátna a plátno správně zvětšíme, najdeme instanci nakreslenou na tom místě, kam jsem s ní posledním posunovým příkazem zajeli. Je přitom jedno, jestli jsme ji tam dovedli některou z jejích posunových metod nebo jestli ji tam „dovezl“ některý posunovač.

Zmenšíme-li naopak plátno tak, že některé instance zmizí, po opětném zvětšení plátna je opět najdeme na svém místě (pokud jsme s nimi mezit tím samozřejmě nehýbali).

Úkol:
Zkoušejte měnit rozměry plátna a odjíždějte obrazy instancí za jeho okraje. Ověřte, že plátno o všech instancích stále ví a že je při vhodném nastavení rozměrů vždy všechny zobrazí.

3.10 Příklad: Kompresor a nafukovací instance

Zkuste si na závěr této kapitoly definovat vlastní rozhraní a třídu, která bude obsluhovat jeho instance. Vyřešte následující úlohu:

Definujte rozhraní **INafukovaci**, v němž definujete metody potřebné k tomu, abyste mohli změnit rozměr obrázku instance.

Pak definujte třídu **Kompresor** s konstruktorem, jehož jediný parametr bude celočíselný a bude definovat sílu vytvářeného kompresoru. Definujte zároveň bezparametrický konstruktor, který bude mít vámi předem definovanou sílu (např. 3).

Instance třídy **Kompresor** budou mít dvě metody: metoda **prifoukni(INafukovaci)** pětkrát za sebou zvětší oba rozměry nafukovaného objektu o velikost rovnu síle kompresoru. Naopak metoda **ufoukni(INafukovaci)** obdobným způsobem obrázek svého parametru zmenší.

Aby nebyla animace příliš rychlá, vložte mezi jednotlivá překreslení objektu volání metody **cekej(int)**, jež je metodou instance plátna a pozastaví další činnost programu na zadaný počet milisekund.

Zkuste úlohu vyřešit nejprve sami. Do ukázkového řešení, které uvedu za chvíli, se podívejte až budete se svoji verzí hotovi anebo když si náhodou nebudete vědět rady. Projekt, v němž je úloha vyřešena, nese označení **03_Tvary_3**.

Ukázkové řešení

Doufám, že jste před čtením této sekce dokončili vlastní verzi řešení předchozí úlohy. Zkusíme si přiblížit, jak jste měli postupovat.

1. První věcí, na níž bychom se měli soustředit, je rozhraní. Nemá smysl vytvářet konstruktor, když by neměl co nafukovat. Definujeme proto nejprve třídu (rozhraní) **INafukovací**. To by nám jistě nemělo dělat problémy. Požádáme BlueJ o vytvoření nové třídy, v následně otevřeném dialogovém okně zadáme jeho název a nastavíte přepínač do polohy **Rozhraní**.
2. V dalším kroku doplníme tělo rozhraní. Mohlo by vypadat např. následovně:

```
/**
 * Rozhrani INafukovaci definuje povinou sadu metod, jez musi mit objekty,
```

```
 * ktere je schopne instance tridy Kompresor "nafouknout".
 *
 * @author     Rudolf Pecinovsky
 * @version    1.00, 3.2.2003
 */
public interface INafukovaci
{
    /**
     * Vrati aktuální sirku nafukovacího objektu,
     *
     * @return sirka objektu
     */
     public int getSirka();


    /**
     * Vrati aktuální vysku nafukovacího objektu,
     *
     * @return vyska objektu
     */
     public int getVyska();


    /**
     * Nastavi novou velikost navukovaciho objektu.
     * Levy horni roh opsaneho obdelniku pritom zustane na stejnem miste.
     *
     * @param vyska  nova sirka objektu
     * @param sirka  nova vyska objektu
     */
    public void setRozmer(int sirka, int vyska);

}//public interface INafukovaci
```

3. Jakmile máme definované rozhraní, můžeme upravit definice tříd, o nichž beztak víme, že je implementují – jen to doposud o sobě veřejně neříkají. Tak to napravíme. Postupným klepáním na šipku dědičnosti a na ikony implementační třídy a implementovaného rozhraní natahámeme mezi ikonami potřebné šipky dědičnosti a tím zároveň upravíme zdrojové kódy příslušných tříd.

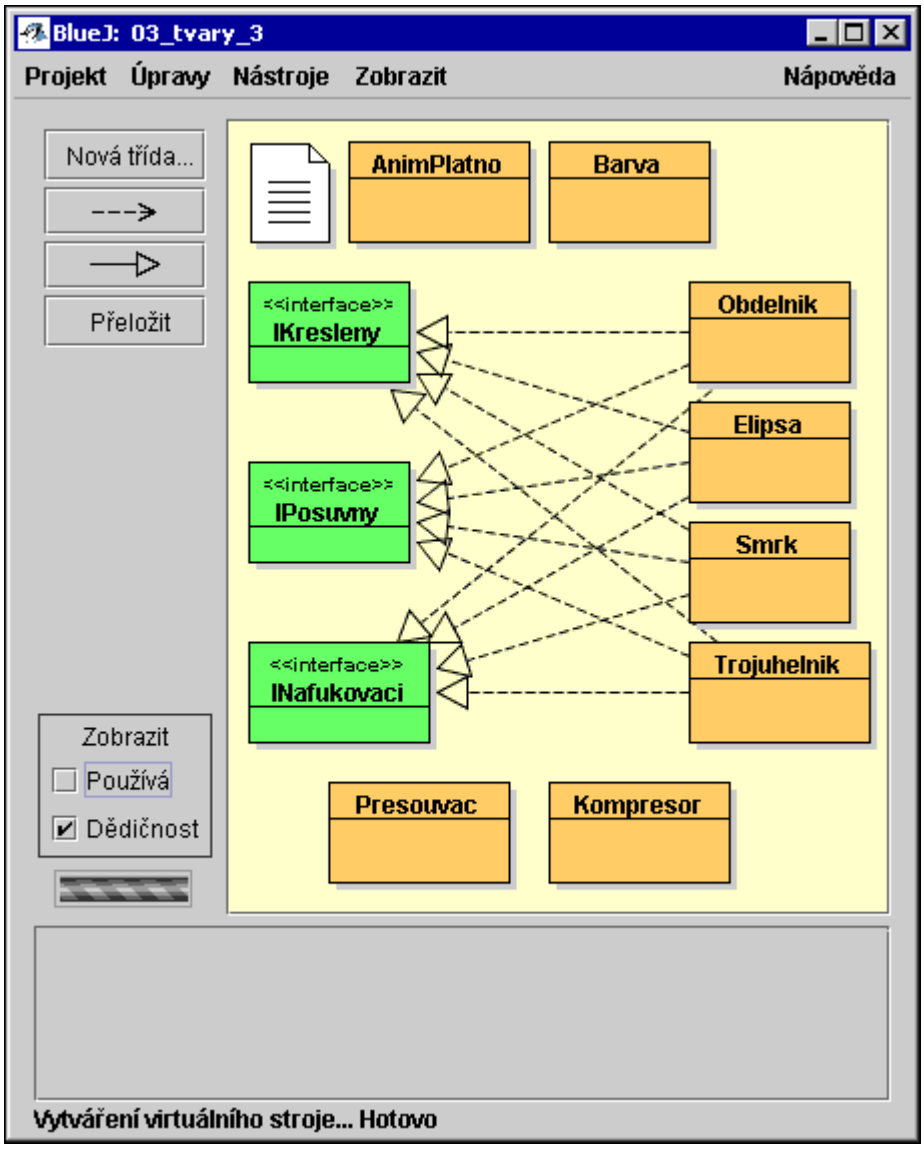
4. Zbývá nám už pouze definovat kompresor. Vytvoříme novou (obyčejnou) třídu, a definujeme požadované konstruktory a metody. Výsledná definice by mohla vypadat např. následovně:

```
/**
 * Trida Kompresor slouzi ke zmene velikosti objektu
 * implementujicich rozhrani IPosuvny.
 *
 * @author     Rudolf Pecinovsky
 * @version    1.00, 3.2.2003
 */
public class Kompresor
{
//== VEREJNE KONSTANTY ================================================
//== SOUKROME KONSTANTY ===============================================
//== ATRIBUTY TRIDY ===================================================

    /** Tento atribut je tu pouze pro zjednodušení psaní. */
    private static AnimPlatno platno = AnimPlatno.getPlatno();

//== ATRIBUTY INSTANCI ================================================

    /** Specifikuje silu "nafukovani" objektu danou instanci kompresoru,
```

```
     * tj. miru jeho prifouknuti. */
    private int sila;


//== PRISTUPOVE METODY ATRIBUTU TRIDY ==========================================
//== OSTATNI METODY TRIDY ======================================================

//== KONSTRUKTORY ==============================================================

    /**
     * Konstruktor pro objekty tridy Kompresor se silou nafukovani 1.
     */
    public Kompresor()
    {
        this( 1 );
    }//public Kompresor()


    /**
     * Konstruktor pro objekty tridy Kompresor se zadanou silou nafukovani.
     *
     * @param sila  sila nafukovani vytvareneho kompresoru
     */
    public Kompresor( int sila )
    {
        this.sila = sila;
    }//public Kompresor()


//== PRISTUPOVE METODY ATRIBUTU INSTANCI ========================================
//== OSTATNI METODY INSTANCI ===================================================

    /**
     * Metoda zvetsi instanci, na niz ukazuje parametr,
     * o petinasobek sily sveho kompresoru.
     *
     * @objekt  zvetsovany objekt
     */
    public void prifoukni(INafukovaci o)
    {   //Prozatím ještě neumíme zapsat cyklus, tak zadáme 5 krát stejný příkaz
        int s = o.getSirka();
        int v = o.getVyska();
        o.setRozmer(s+1*sila, v+1*sila);  platno.cekej(50);
        o.setRozmer(s+2*sila, v+2*sila);  platno.cekej(50);
        o.setRozmer(s+3*sila, v+3*sila);  platno.cekej(50);
        o.setRozmer(s+4*sila, v+4*sila);  platno.cekej(50);
        o.setRozmer(s+5*sila, v+5*sila);  platno.cekej(50);
    }//public void prifoukni(INafukovaci o)


    /**
     * Metoda zmensi instanci, na niz ukazuje parametr,
     * o petinasobek sily sveho kompresoru.
     *
     * @objekt  zmensovany objekt
     */
    public void ufoukni(INafukovaci o)
    {   //Prozatím ještě neumíme zapsat cyklus, tak zadáme 5 krát stejný příkaz
        int s = o.getSirka();
        int v = o.getVyska();
        o.setRozmer(s-1*sila, v-1*sila);  platno.cekej(50);
        o.setRozmer(s-2*sila, v-2*sila);  platno.cekej(50);
        o.setRozmer(s-3*sila, v-3*sila);  platno.cekej(50);
        o.setRozmer(s-4*sila, v-4*sila);  platno.cekej(50);
```

```
            o.setRozmer(s-5*sila, v-5*sila);  platno.cekej(50);
        }//public void ufoukni(INafukovaci o)


    //== SOUKROME A POMOCNE METODY ================================================
    //== VNORENE A VNITRNI TRIDY ===================================================

    }//public class Kompresor
```

Výsledný diagram by mohl vypadat např. tak jako na následujícím obrázku.



Obrázek 3.8: *Možná podoba diagramu tříd výsledného řešení*

5. Posledním krokem celé operace je přezkoušení správnosti naprogramování všech metod.

3.11 Shrnutí – co jsme se v kapitole naučili

V této kapitole jsem se seznámili s rozhraními a některými aspekty jejich použití. Nyní byste již měli vědět, že:

- ☞ Prostředí BlueJ umožňuje potlačit zobrazování šipek závislostí a/nebo šipek dědičnosti.
- ☞ Rozhraní je zvláštní druh třídy, která metody svých instancí pouze deklaruje, ale neimplementuje.
- ☞ Proměnné odkazující na instance nějakého rozhraní odkazují ve skutečnosti na instance tříd, které dané rozhraní implementují.
- ☞ Třída, která implementuje rozhraní, se musí k této implementaci explicitně (zjevně) přihlásit ve své hlavičce.
- ☞ Třídy implementující dané rozhraní musí implementovat všechny v něm deklarované metody. Tyto metody musí být veřejné a musí to být metody instancí.
- ☞ Implementaci rozhraní třídou můžeme zadat v diagramu tříd natažením šipky dědičnosti od implementující třídy k implementovanému rozhraní.
- ☞ Třídu s předem připraveným zdrojovým kódem vložíme do projektu nejlépe prostřednictvím příkazu **Úpravy → Nová třída ze souboru**....
- ☞ Potřebujeme-li vložit do projektu větší množství tříd najednou, můžeme využít služeb libovolného správce souborů. Nově přidané třídy však BlueJ zahrne do repertoáru až po novém zapnutí.

Autor pracuje jako EDU expert ve firmě Amaio Technologies, Inc.