Seminář Java

III

Radek Kočí

Fakulta informačních technologií VUT

21. února 2007

Téma přednášky

- Ladění programu
- Základy OO – abstrakce, zapouzdření, dědičnost, polymorfismus, objektové rozhraní, identita objektu
- Typová kontrola

Ladění programu

Pro ladění programů v Javě lze využít

- kontrolní tisky: `System.err.println(...)`
- řádkový debugger `jdb`
- integrovaný debugger v IDE
- speciální nástroje na záznam běhu balíků

Uvědomte si, že žádný nástroj za nás nevymyslí, JAK máme své třídy testovat. Pouze nám pomůže ke snadnějšímu sestavení a spuštění testu.

Ladění programu

- standardní klíčové slovo (od JDK1.4) `assert`
  - `assert booleovský_výraz`
- testovací nástroje typu JUnit (a varianty – `HttpUnit`, ...)
  - metoda `assertEquals()`
  - metoda `assertTrue()`
  - ...
  - `http://junit.org/`
- pokročilé nástroje na běhovou kontrolu platnosti invariantů, vstupních, výstupních a dalších podmínek
  - např. jass (Java with ASSertions),
  - `http://csd.informatik.uni-oldenburg.de/~jass/`

Ladění programu – assert

```
public class AssertDemo {
    public static void main(String args[]) {
        int x = 10;
        boolean enabled = false;

        assert enabled = true;

        System.out.println("Assertions are " +
            (enabled ? "enabled" : "disabled");

        assert x < 0 : "x is not < 0";
    }
}
```

Ladění programu – assert

- přeložit s volbou `-source 1.4`
- spustit s volbou `-ea` (`-enableassertions`)
- dojde-li za běhu programu k porušení podmínky stanovené za `assert`, vznikne běhová chyba (`AssertionError`) a program skončí

Ladění programu – JUnit

Postup

- stáhnout si distribuci testovacího prostředí (stačí binární) `http://junit.org`
- nainstalovat `JUnit` (tj. rozbalit do adresáře)
- napsat testovací třídu (třídy) – obvykle rozšiřují (dědí) třídu `junit.framework.TestCase`
- testovací třída obsahuje metody
  - metodu pro nastavení testu – `setUp()`
  - testovací metody – `test`*`Neco`*`()`
  - úklidovou metodu – `tearDown()`
- testovací třídu spustit v textovém nebo grafickém prostředí
  - `junit.textui.TestRunner`
  - `junit.swingui.TestRunner`
- testování zobrazí, které testovací metody případně selhaly

Ladění programu – JUnit

```
public class JUnitDemo extends TestCase {
   Zlomek x, y, z;

   public void setUp() {
      x = new Zlomek(2,3);
      y = new Zlomek(4,6);
      z = new Zlomek(4,3);
   }
   public void testRovna() {
      assertEquals("2/3 = 4/6.", x, y);
   }
   public void testSoucet() {
      Zlomek z = x.plus(y);
      assertEquals("2/3 + 4/6 = 4/3.", z, soucet);
   }
}
```

Základy objektové orientace

Objektově orientovaný přístup k modelování a vývoji systémů

- kolekce vzájemně komunikujících objektů
- objekt = abstrakce doménově specifických entit
- objekt = sloučení dat a funkcionality do uzavřené jednotky
- vykazuje vyšší stabilitu navrhovaných prvků z pohledu měnících se požadavků
- soubor objektově orientovaných prostředků (objekty, třídy, UML, . . .) a metodologie (např. RUP)
- *Objektový návrh nutně neimplikuje objektovou implementaci!*

Objektově orientované jazyky

Rozlišujeme dva typy jazyků

- Prototypově založené (prototype-based)
  - *Self, JavaScript, . . .*
- Třídně založené (class-based)
  - *Smalltalk, Java, C++, C#, . . .*

Prototypově založené jazyky

Představují takový styl OO přístupu, který pracuje *pouze* s objekty

- nové objekty se vytvářejí klonováním již existujících objektů
- vždy existuje alespoň jeden počáteční objekt (prototyp)

Třídně založené jazyky

Představují takový styl OO přístupu, který definuje *třídy* objektů

- nalezené objekty jsou klasifikovány do tříd
- třída je generická definice pro množinu podobných objektů (šablona)
- třída je množina objektů, které mají stejné chování a stejnou množinu atributů

Třídně založené jazyky

Třída

- třída definuje atributy a chování objektu (metody)
- objekt je instance třídy
- objekty stejné třídy sdílejí chování (metody), atributy má každý objekt vlastní
- třída může definovat *třídní atributy* $\Rightarrow$ jsou sdíleny všemi instancemi třídy

Proměnné a metody třídy – statické

Dosud jsme zmiňovali proměnné a metody (tj. souhrnně prvky – members) objektu.

- Lze deklarovat také metody a proměnné patřící celé třídě, tj. skupině všech objektů daného typu.
- Takové metody a proměnné nazýváme statické a označujeme v deklaraci modifikátorem `static`

Příklad – počítání účtů

```
public class Ucet {
    protected Klient majitel;
    protected double zustatek = 0;
    protected static int pocet = 0;

    public Ucet(Klient k) {
        majitel = k;
        pocet++;
    }
    public static int pocetUctu() {
        return pocet;
    }
}
```

Příklad

```
public final class System {
    public static final PrintStream out;
    public static final InputStream in;
    ...

    public static long currentTimeMillis();
    ...
}
```

```
public static void main(String[] argv) {
    ...
}
```

Základní pojmy – Objektové rozhraní

Operace vs. metoda

- množina operací reprezentuje chování objektu
- metoda implementuje operaci

Rozhraní objektu

- množina operací, které objekt nabízí
- pouze definuje *co* objekt umí (nabízí)

Příklad

```
public class Car {
    protected int weight;
    protected int capacity;
    protected Driver drivenBy;
    protected int km;

    public Car(int weight, int capacity) { ... }
    public int getWeight() { ... }
    public int getCapacity() { ... }
    public void drivenBy(Driver d) { ... }
    public Driver getDriver() { ... }
    public int run(int km) { ... }
    protected int price(int km) { ... }
}
```

Základní pojmy – Stav a identita objektu

Stav objektu

- stavová množina je reprezentována množinou hodnot atributů objektu
- aktuální hodnoty všech atributů představují aktuální stav
- v každém okamžiku je objekt v definovatelném stavu

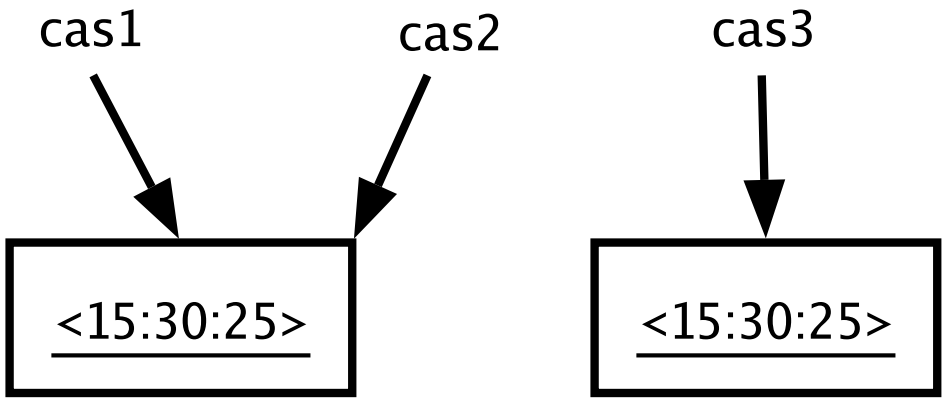
Identita objektu

- každý objekt je jedinečný bez ohledu na stav

Shodnost objektů

- shodnost je vázána na stavy objektů
- objekty, které nejsou identické, mohou být shodné

Základní pojmy – Identita objektu



	Java	Smalltalk	výsledek
shodnost	`cas1.equals(cas2)` `cas2.equals(cas3)`	`cas1 = cas2` `cas2 = cas3`	`true` `true`
identita	`cas1 == cas2` `cas2 == cas3`	`cas1 == cas2` `cas2 == cas3`	`true` `false`

Základní pojmy – Komunikace objektů

Komunikace objektů

- objekty spolu komunikují zasíláním zpráv
- příjemce chápe zprávu jako požadavek na provedení služby (operace)
- zpráva obsahuje
  - identifikátor příjemce
  - název operace
  - argumenty
- obsluha zprávy (vykonání metody) reaguje podle stavu / modifikuje stav objektu
- po ukončení obsluhy může metoda vracet výsledek

Příklad

```
Car car = new Car(2000, 9);
Driver pepa = new Driver("Pepa");

pepa.driveCar(car);
int cena = car.run(157);
```

Základy objektové orientace

Vlastnosti objektové orientace

- Abstrakce (abstraction)
- Zapouzdření (encapsulation)
- Polymorfismus (polymorphism)
- Dědičnost (inheritance)

Vlastnosti objektové orientace – Abstrakce

```
public class Car {
    protected int weight;
    protected int capacity;
    protected Driver drivenBy;
    protected int km;

    public Car(int weight, int capacity) { ... }
    public int getWeight() { ... }
    public int getCapacity() { ... }
    public void drivenBy(Driver d) { ... }
    public Driver getDriver() { ... }
    public int run(int km) { ... }
    protected int price(int km) { ... }
}
```

Vlastnosti objektové orientace – Abstrakce

Abstrakce

- vytvářený systém objektů je abstrakcí řešeného problému
- analýza problému $\Rightarrow$ klasifikace do abstraktních struktur $\Rightarrow$ objekty
- klasifikace je založena na rozpoznávání podobností v řešené problematice
- zjednodušený pohled na systém bez ztráty jeho významu
- objekt je abstrakcí části řešené domény, má definovanou zodpovědnost za řešení části problému

Vlastnosti objektové orientace – Zapouzdření

```
int obsah(int x, int y) {
    return x * y;
}
```

```
struct Obdelnik {
    int x, y;
}
int obsah(struct Obdelnik o) {
    return o.x * o.y;
}
```

```
struct Obdelnik {
    int x, y;
    int obsah() { return x * y; }
}
```

Vlastnosti objektové orientace – Zapouzdření

```
int obsah(int x, int y) {
    return x * y;
}
```

```
struct Obdelnik {
    int x, y;
}
int obsah(struct Obdelnik o) {
    return o.x * o.y;
}
```

```
struct Obdelnik {
    int x, y;
    int obsah() { return x * y; }
}
```

Vlastnosti objektové orientace – Zapouzdření

```
int obsah(int x, int y) {
    return x * y;
}
```

```
struct Obdelnik {
    int x, y;
}
int obsah(struct Obdelnik o) {
    return o.x * o.y;
}
```

```
struct Obdelnik {
    int x, y;
    int obsah() { return x * y; }
}
```

Vlastnosti objektové orientace – Zapouzdření

Zapouzdření

- Seskupení souvisejících idejí do jedné jednotky, na kterou se lze následně odkazovat jediným názvem (objekt).
- Objektově orientované zapouzdření je seskupení operací a atributů (reprezentujících stav) do jednoho typu objektu. Stav je pak dostupný či modifikovatelný pouze prostřednictvím rozhraní (operace, metody).
- Omezení externí viditelnosti informací nebo implementačních detailů.
- Zaručené rozhraní.

Příklad – Zapouzdření

```
public class Car {
    protected int weight;
    protected int capacity;
    protected Driver drivenBy;
    protected int km;

    public Car(int weight, int capacity) { ... }
    public int getWeight() { ... }
    public int getCapacity() { ... }
    public void drivenBy(Driver d) { ... }
    public Driver getDriver() { ... }
    public int run(int km) { ... }
    protected int price(int km) { ... }
}
```

Příklad – Zapouzdření a použití objektu

```
Car car = new Car(2000, 9);
Driver pepa = new Driver("Pepa");

pepa.driveCar(car);
int cena = car.run(157);
```

Základní pojmy – Dědičnost

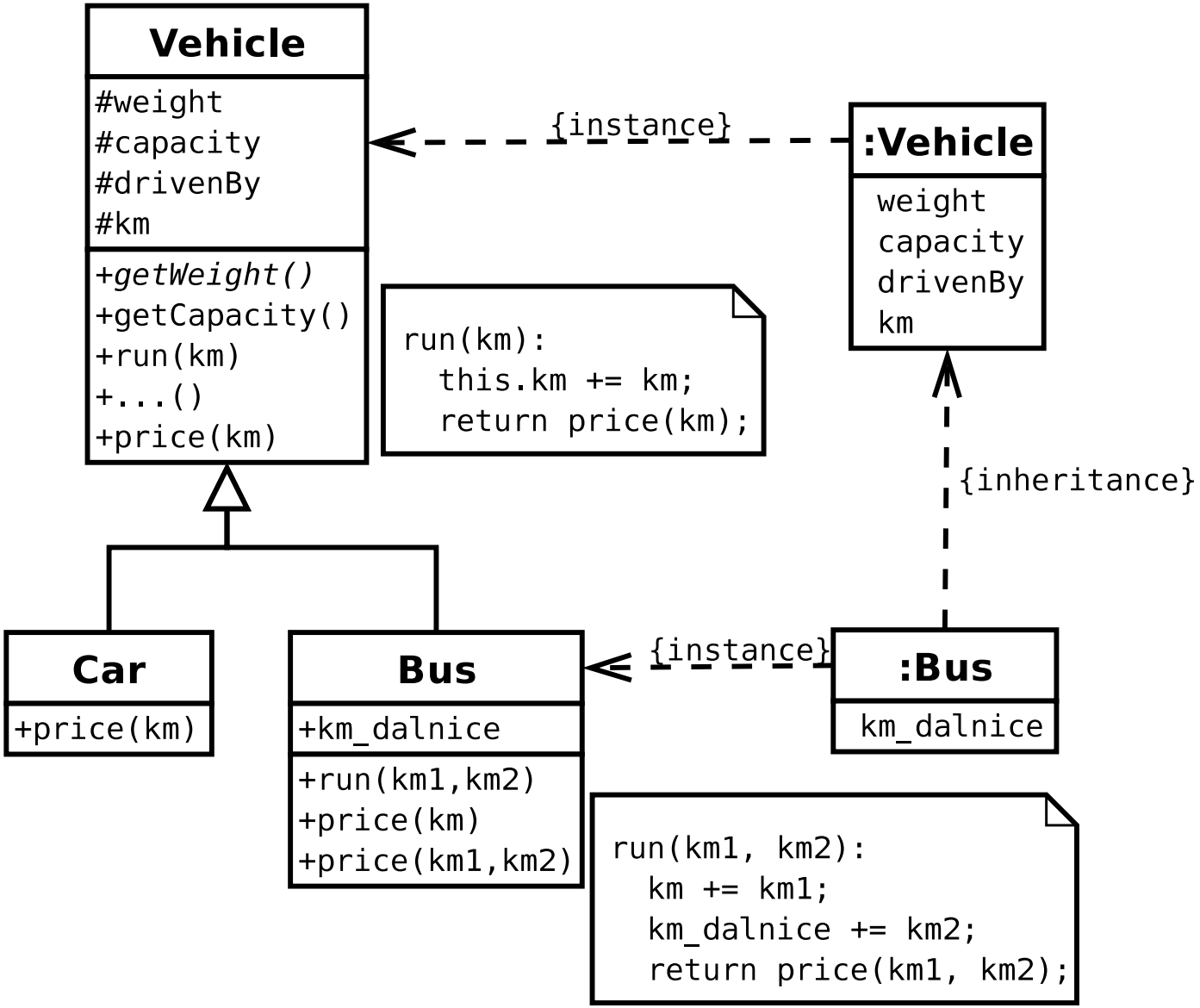
Dědičnost

- vyjadřuje hierarchický vztah mezi objekty
- definuje a vytváří objekty na základě již existujících objektů
  - možnost sdílení chování bez nutnosti reimplementace
  - možnost rozšíření chování

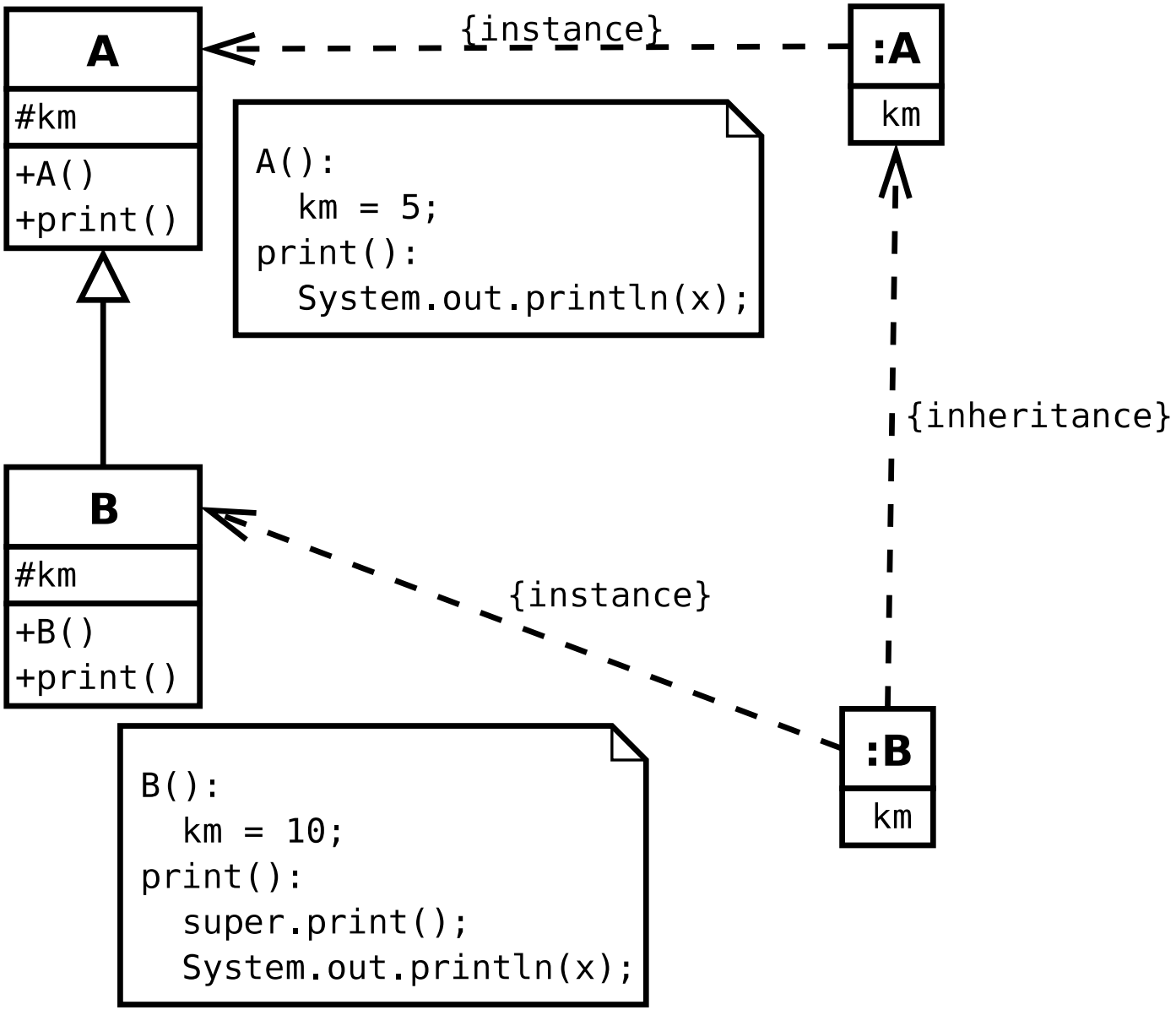
Způsob vyjádření dědičnosti závisí na typu jazyku

- třídně orientované jazyky
- prototypově orientované jazyky

Třídně orientované jazyky – dědičnost



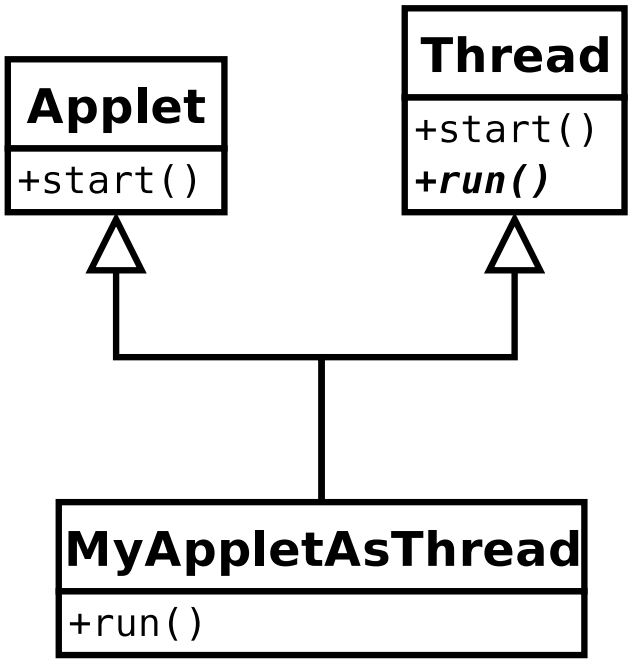
Třídně orientované jazyky – dědičnost



Třídně orientované jazyky – dědičnost

Vícenásobná dědičnost

- komplikuje návrh (čitelnost)
- problém nejednoznačnosti
- dá se obejít (skládání objektů)
- existují případy, kdy má vícenásobná dědičnost význam



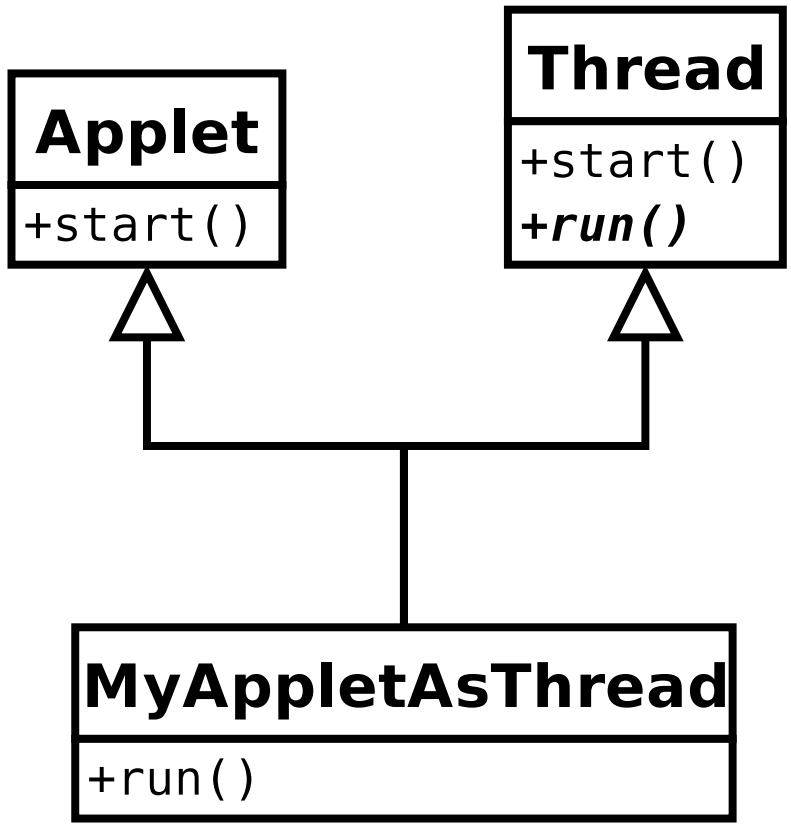
Prototypově založené jazyky – dědičnost

Dědičnost (delegování)

- dědičnost objektů je vyjádřena delegováním
- objekt může určit množinu jiných objektů, na které deleguje zprávy, kterým sám nerozumí $\Rightarrow$ sdílení chování s jinými objekty
- více ”nadřazených” objektů $\Rightarrow$ problém nejednoznačnosti $\Rightarrow$ priorita ”nadřazených” objektů

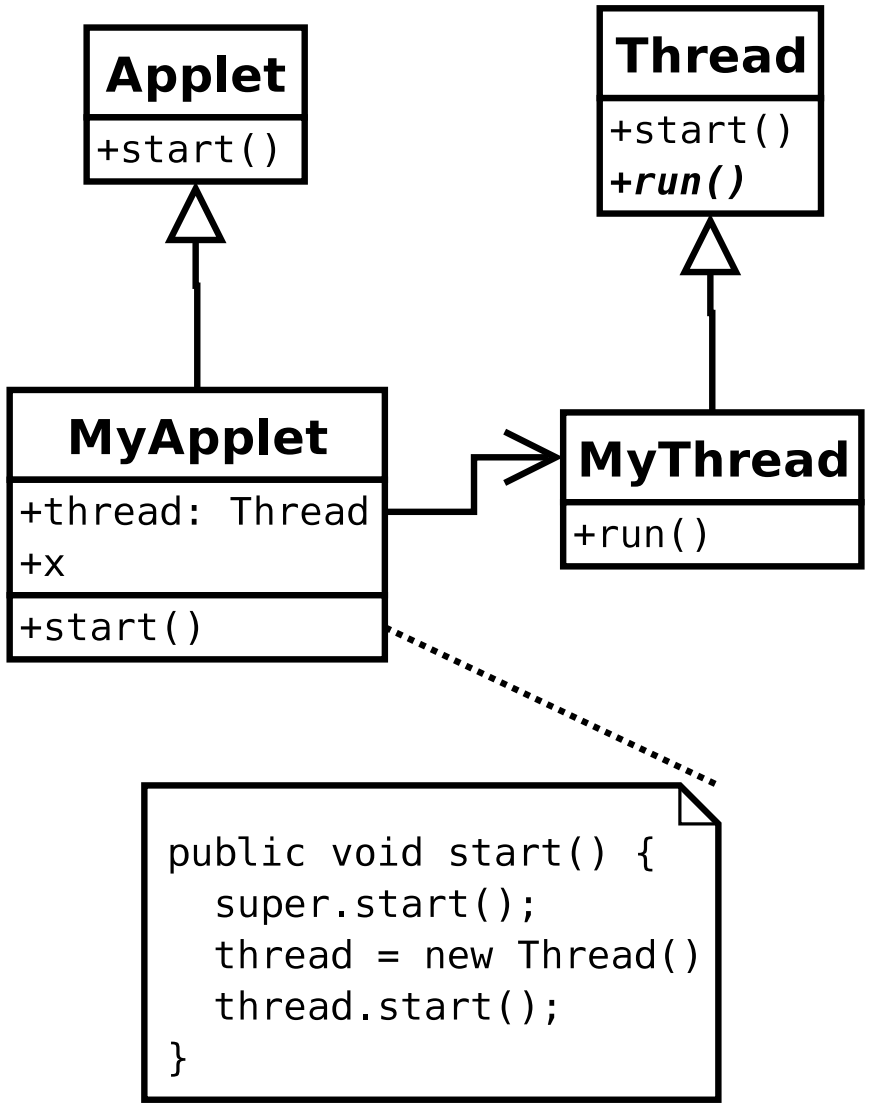
Vícenásobná dědičnost

Java nemá vícenásobnou dědičnost!



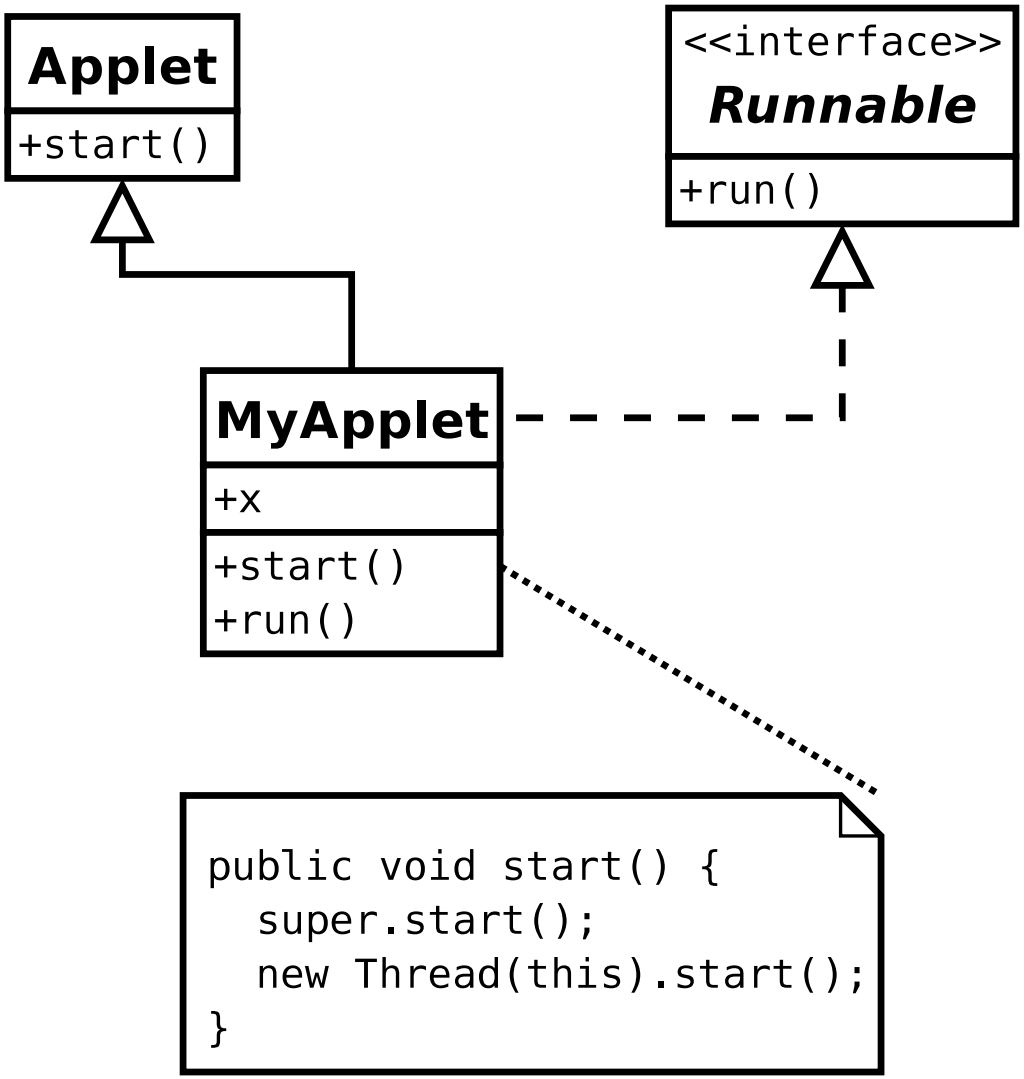
Vícenásobná dědičnost

Jak se obejít bez vícenásobné dědičnosti – I



Vícenásobná dědičnost

Jak se obejít bez vícenásobné dědičnosti – II



Rozhraní objektu

Rozhraní objektu

- množina operací, které objekt nabízí
- pouze definuje *co* objekt umí (nabízí), nedefinuje *jak*
- způsob provedení operace závisí na implementaci (metoda)
  - stejné rozhraní může být implementováno různými objekty
  - stejné operace mohou mít různé implementace

Vlastnosti objektové orientace – Polymorfismus

Polymorfismus

- mnohotvarost, schopnost výskytu v mnoha formách
- výskyt různých typů chování na základě stejné zprávy
  - možnost vícenásobné definice operace s jedním názvem, která tak může nabývat více implementací (implementuje různé chování).
  - logický vztah podobných operací (aplikace operací na podobné, ale technicky různé situace)

```
obj.start();
```

Rozhraní, třída a objekt v Javě

Rozhraní

- specifikuje množinu vlastností, ale *neimplementuje je*
- vlastnostmi se myslí především operace
- definuje *typ* objektu

Rozhraní, třída a objekt v Javě

Třída (také poněkud nepřesně zvaná objektový typ)

- implementuje rozhraní (tj. všechny metody rozhraní)
- třída představuje vzor pro vytváření objektů
- třída představuje skupinu objektů, které nesou stejné vlastnosti (kvalitativně)
  - např. všechny objekty třídy `Clovek` mají vlastnost `jmeno`,
  - tato vlastnost má však obecně pro různé lidi různé hodnoty – lidi mají různá jména
- *pozn.: třída sama o sobě deklaruje rozhraní $\Rightarrow$ třída také definuje typ objektu*

Rozhraní, třída a objekt v Javě

Objekt

- objekt je jeden konkrétní jedinec (reprezentant, entita) příslušné třídy
- pro konkrétní objekt nabývají vlastnosti deklarované třídou konkrétních hodnot
- objekt je *instancí* třídy

Typy, kontrola typů

Význam typování

- určit sémantický význam elementům (hodnoty v paměti)
- pokud má paměťová hodnota přiřazený typ, můžeme s ní pracovat na vyšší úrovni – víme jaké operace je možné provést, můžeme provádět kontrolu typové konzistence atp.

Staticky typované jazyky

- k typové kontrole dochází v době kompilace
- *C++, Java, . . .*

Dynamicky typované jazyky

- k typové kontrole dochází v době běhu programu
- *Smalltalk, Self, Python, Lisp . . .*

Typy, kontrola typů

Ukázka chování staticky a dynamicky typovaných systémů

- staticky typované: řádek č. 3 je ilegální
- dynamicky typované: řádek č. 3 je OK (není požadovaná typová konzistence pro proměnnou *x*)

```
var x;          // (1)
x := 5;         // (2)
x := "hi";      // (3)
```

Typy, kontrola typů

Ukázka chování staticky a dynamicky typovaných systémů

- staticky typované: řádek č. 3 je ilegální
- dynamicky typované: řádek č. 3 vyvolá chybu za běhu programu

---

```
var x;          // (1)
x := 5;         // (2)
5 / "hi";       // (3)
```

Typy, kontrola typů

```
| x |
x := self get.
x message.          "překlad:  neznámý typ"
```

```
Type x;
x = this.get();
x.message();   //překlad:  Type a odvozené
```

Typy, kontrola typů

Dynamická kontrola

- probíhá u všech jazyků
- jako dynamicky typované se označují ty, které nemají statickou kontrolu
- některé staticky typované jazyky (*C* + +, *Java*) umožňují dynamické přetypování, čímž částečně obcházejí statickou typovou kontrolu

Silně a slabě typované jazyky

- lze se setkat s tímto rozdělením
- avšak význam těchto pojmů není jednoznačný
- viz např. `http://en.wikipedia.org`