Seminář Java
IV

Radek Kočí

Fakulta informačních technologií VUT

Únor 2012

Téma přednášky

- Abstraktní třída
- Rozhraní
- Zaměnitelnost objektů (typová kompatibilita)
- Pole

Abstraktní třída

Abstraktní třída

- třída, která danou specifikaci implementuje jen částečně
- nemůže mít instance
- klíčové slovo `abstract`

Abstraktní třída = částečná implementace
Třída = úplná implementace

Abstraktní třída – Příklad

```
public class Vehicle {
    ...
    public int price(int km) { ??? }
}

public class Car extends Vehicle {
    protected int price(int km) {
        // podle osobniho auta ...
    }
}

public class Bus extends Vehicle {
    protected int price(int km) {
        // podle autobusu ...
    }
}
```

Abstraktní třída – Příklad

```
public abstract class Vehicle {
    ...
    public Vehicle(int weight, int capacity) {
        this.weight = weight;
        this.capacity = capacity;
    }
    protected abstract int price(int km);
}

public class Car extends Vehicle {
    public Car(int weight, int capacity) {
        super(weight, capacity);
    }
    protected int price(int km) { ... }
}
```

Rozhraní, třída a objekt v Javě

Rozhraní

- specifikuje množinu vlastností, ale *neimplementuje je*
- definuje *typ* objektu

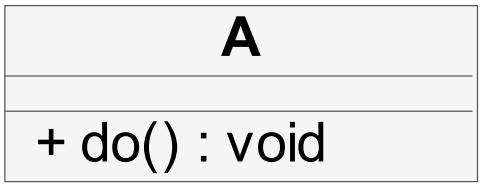
Třída (také poněkud nepřesně zvaná objektový typ)

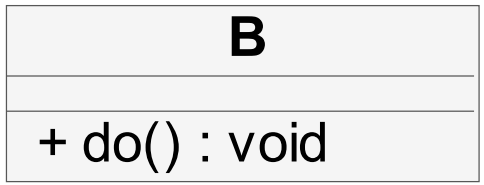
- implementuje rozhraní (tj. všechny metody rozhraní)
- *pozn.: třída sama o sobě deklaruje rozhraní* $\Rightarrow$ *třída také definuje typ objektu*
- *Abstraktní třída* = částečná implementace rozhraní

Objekt

- objekt je *instancí* třídy

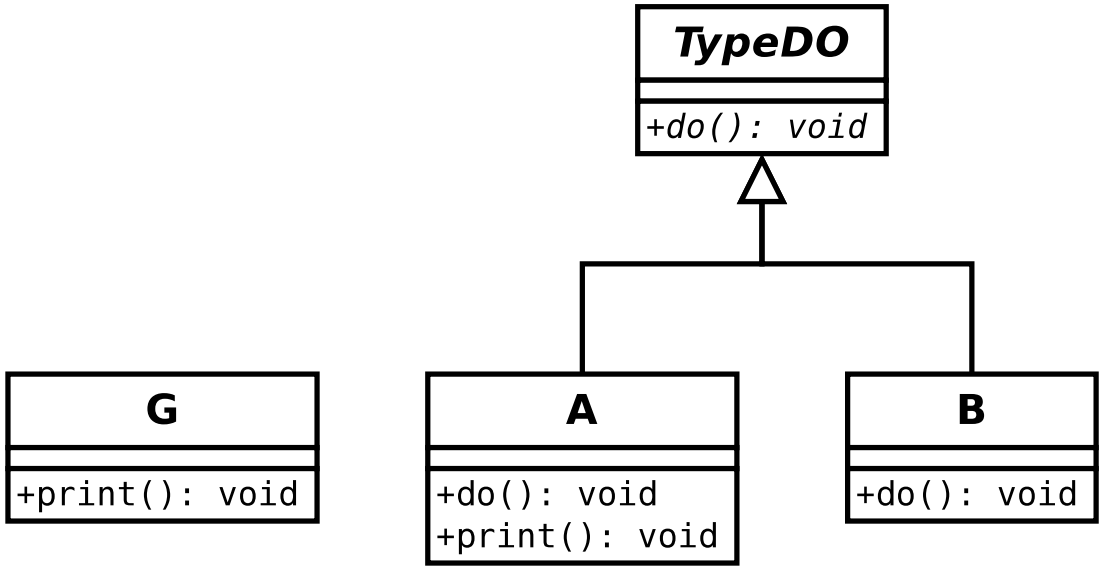
Rozhraní






```
public void m1(A obj) { obj.do(); }

m1(new A());
m1(new B()); <- !
```

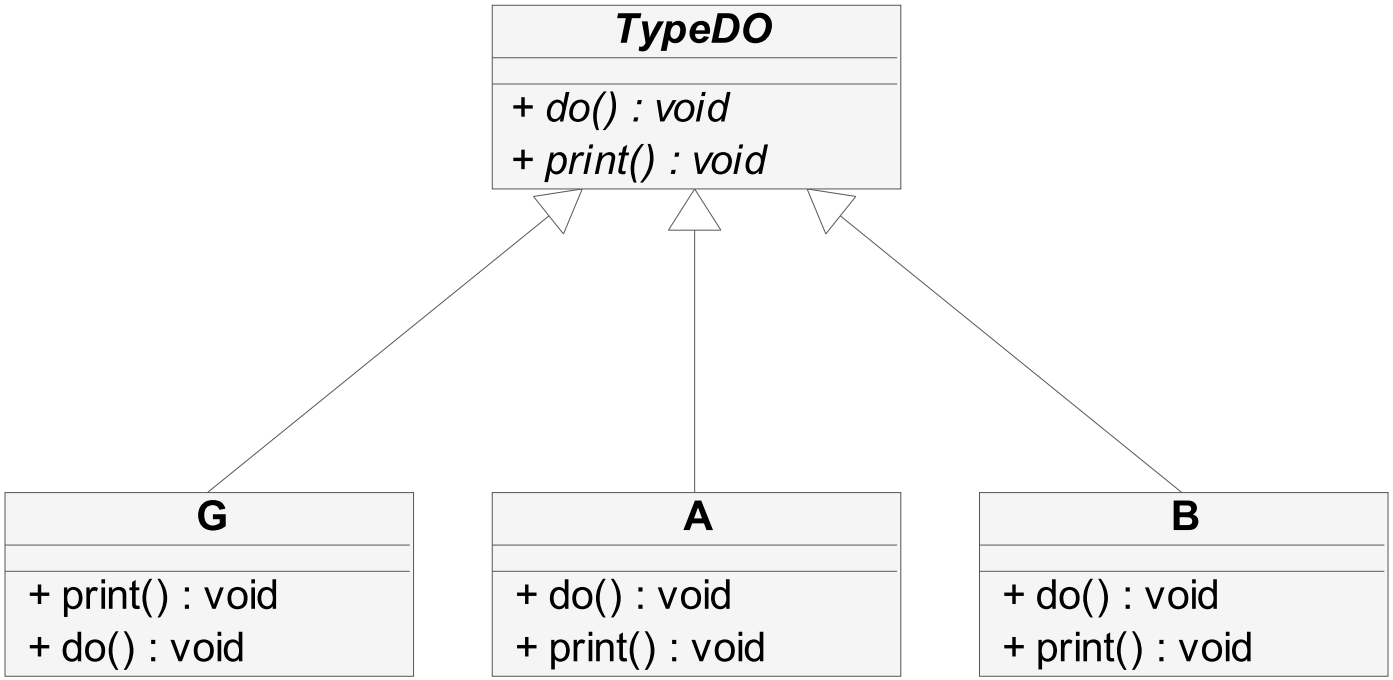
Rozhraní




```
public void m1(TypeDO obj) { obj.do(); }
m1(new A());
m1(new B());
```

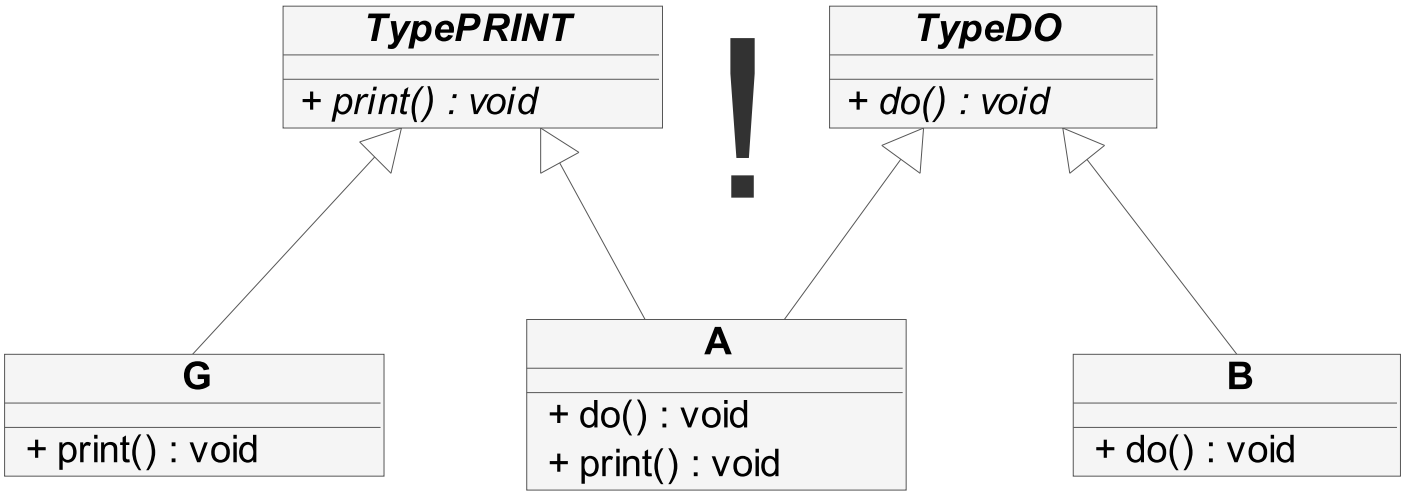
```
public void m2(A obj) { obj.print(); }
m2(new A());
m2(new G()); <-!
```

Rozhraní




```
public void m1(TypeDO obj) { obj.do(); }
public void m2(TypeDO obj) { obj.print(); }

m1(new A()); <-- B, G
m2(new A()); <-- B, G
```

Rozhraní




```
public void m1(TypeDO obj) { obj.do(); }
public void m2(TypePRINT obj) { obj.print(); }

m1(new A()); <-- B
m2(new A()); <-- G
```

Rozhraní

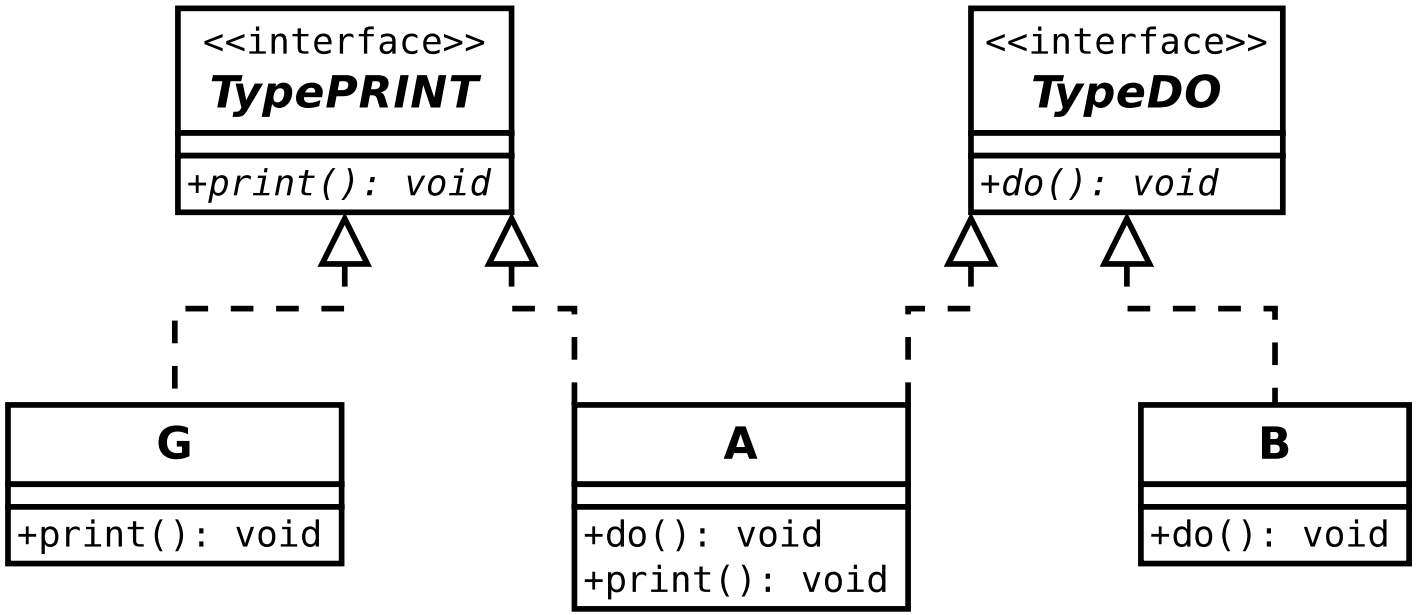
V Javě, na rozdíl od C++ neexistuje vícenásobná dědičnost

- to nám ušetří řadu komplikací (problém nejednoznačnosti)
- ale je třeba to něčím nahradit

Pokud po třídě chceme, aby disponovala vlastnostmi z několika různých množin (skupin), můžeme ji deklarovat tak, že implementuje více rozhraní

- objekt je typu *A*, pokud její třída implementuje rozhraní *A*
- implementace více rozhraní $\Rightarrow$ objekt může mít více typů

Rozhraní




```
public void m1(TypeDO obj) { obj.do(); }
public void m2(TypePRINT obj) { obj.print(); }

m1(new A()); <-- B
m2(new A()); <-- G
```

Třída, rozhraní a typová zaměnitelnost

Typová zaměnitelnost

- Do proměnné, jejíž typ je deklarován jako třída `A`, lze dosadit všechny instance třídy `A` a všechny instance tříd odvozených od třídy `A`.
- Do proměnné, jejíž typ je deklarován jako rozhraní `I`, lze dosadit všechny instance tříd, které implementují rozhraní `I`, příp. jsou odvozeny od těchto tříd.

Rozhraní

Co je rozhraní

- popis (specifikace) množiny vlastností (metod), aniž bychom tyto vlastnosti ihned implementovali.
- určitá třída implementuje rozhraní, pokud implementuje všechny metody, které jsou daným rozhraním předepsány.

Rozhraní v Javě je specifikováno

- množinou hlaviček metod označenou identifikátorem – názvem rozhraní
- ucelenou specifikací – tj. popisem, co přesně má metoda dělat (vstupy/výstupy metody, její vedlejší efekty . . .)

Deklarace rozhraní

- Vypadá i umisťuje se do souborů podobně jako deklarace třídy
- Všechny metody v rozhraní musí být public a v hlavičce se to ani nemusí uvádět.
- Všechny metody v rozhraní jsou zároveň automaticky abstraktní $\Rightarrow$ těla metod se neuvádějí.
- Rozhraní může obsahovat proměnné – jedná se vždy o konstantu (modifikátor `final` se uvádět nemusí)

Příklad deklarace rozhraní

```
public interface Informator {
    public void vypisInfo();
}
```

Implementace rozhraní

```
public class Ucet implements Informator {
   ...
   public void vypisInfo() {
      ...
   }
}
```

- Třída implementuje všechny metody předepsané rozhraním.
- Třída může implementovat více rozhraní současně.

```
public class Name implements Interface1,
                             Interface2
{ ... }
```

Rozšiřování rozhraní

- Podobně jako u tříd i rozhraní může být *děděno*.
- Třída dědí maximálně z jednoho předka.
- Rozhraní může dědit z více předků (*vícenásobná dědičnost*).

```
public interface DobryInformator
                 extends Informator
{
  public void vypisViceInfo();
}
```

Rozšiřování rozhraní

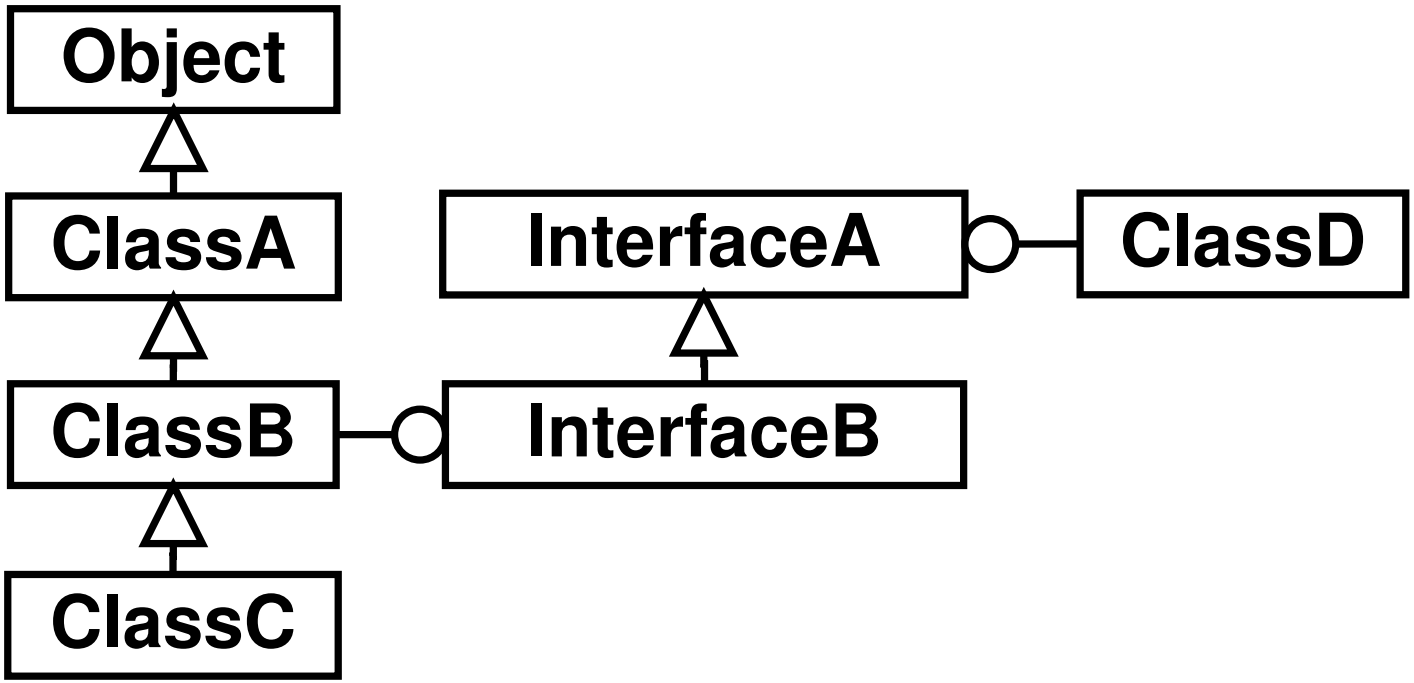
Třída, která implementuje rozhraní *DobryInformator* musí implementovat *obě* metody:

```
public class Ucet implements DobryInformator {
   ...
   public void vypisInfo() {
      ...
   }
   public void vypisViceInfo() {
      ...
   }
}
```

Typová zaměnitelnost

- Do proměnné, jejíž typ je deklarován jako třída `A`, lze dosadit všechny instance třídy `A` a všechny instance tříd odvozených od třídy `A`.
- Do proměnné, jejíž typ je deklarován jako rozhraní `I`, lze dosadit všechny instance tříd, které implementují rozhraní `I`, příp. jsou odvozeny od těchto tříd.
- Do proměnné, jejíž typ je deklarován jako rozhraní `I`, lze dosadit všechny instance tříd (a odvozených tříd), které implementují rozhraní `I` nebo rozhraní odvozené od rozhraní `I`.

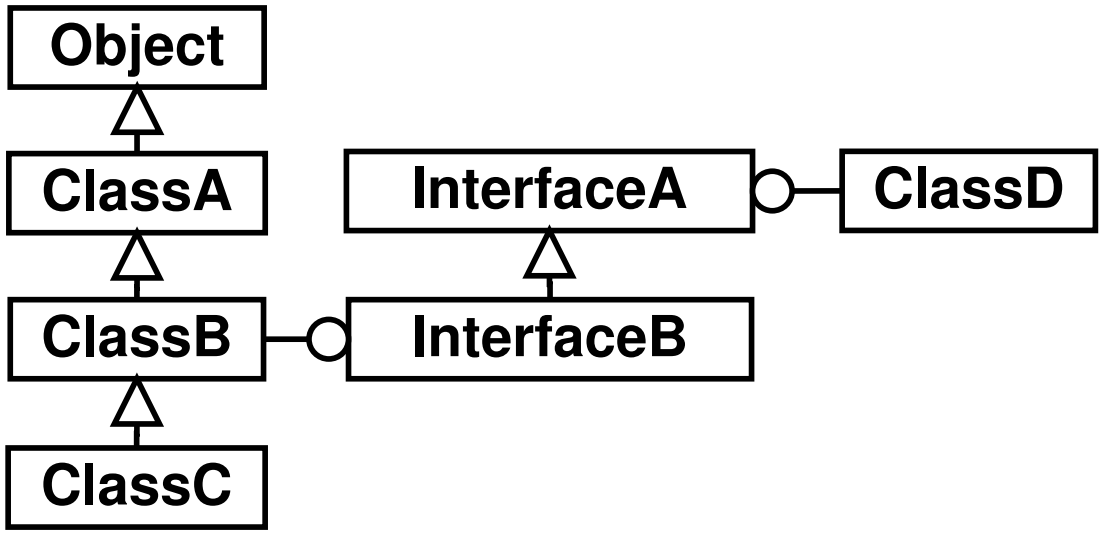
Dosazení objektu do proměnné – I




```
void method(ClassA o) { ... }
```

- `o` ⇒ `ClassA`, `ClassB`, `ClassC`
- `o == ClassB` ⇒ `(ClassB) o`

Dosazení objektu do proměnné – II




```
void method(InterfaceB o) { ... }
```

- `o` ⇒ `ClassB`, `ClassC`

```
void method(InterfaceA o) { ... }
```

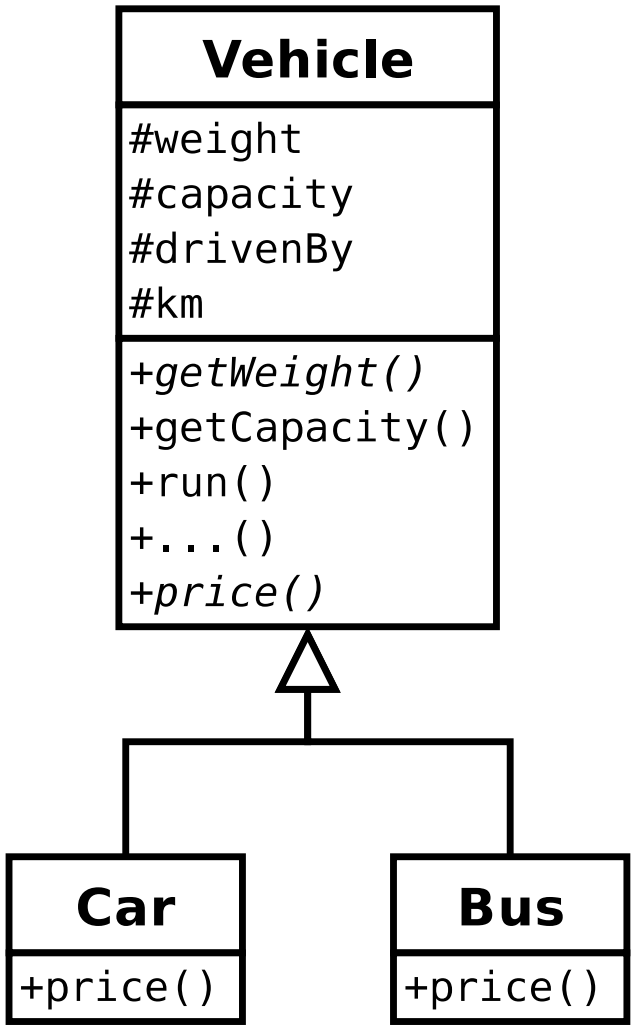
- `o` ⇒ `ClassB`, `ClassC`, `ClassD`
- `o == ClassC` ⇒ `(InterfaceB) o`
- `o == ClassC` ⇒ `(ClassC) o`

Dosazení objektu do proměnné – příklad

```
public abstract class Vehicle {
    public abstract int price(int km);
}
public class Car extends Vehicle {
    public int price(int km) { ... }
}
public class Bus extends Vehicle {
    public int price(int km) { ... }
}
```

```
method(new Bus());
public void method(Vehicle v) {
    Car c = (Car) v;    ⇐ (ClassCastException)
    System.out.println(c.price(200);
}
```

Dosazení objektu do proměnné – příklad



Dosazení objektu do proměnné – příklad

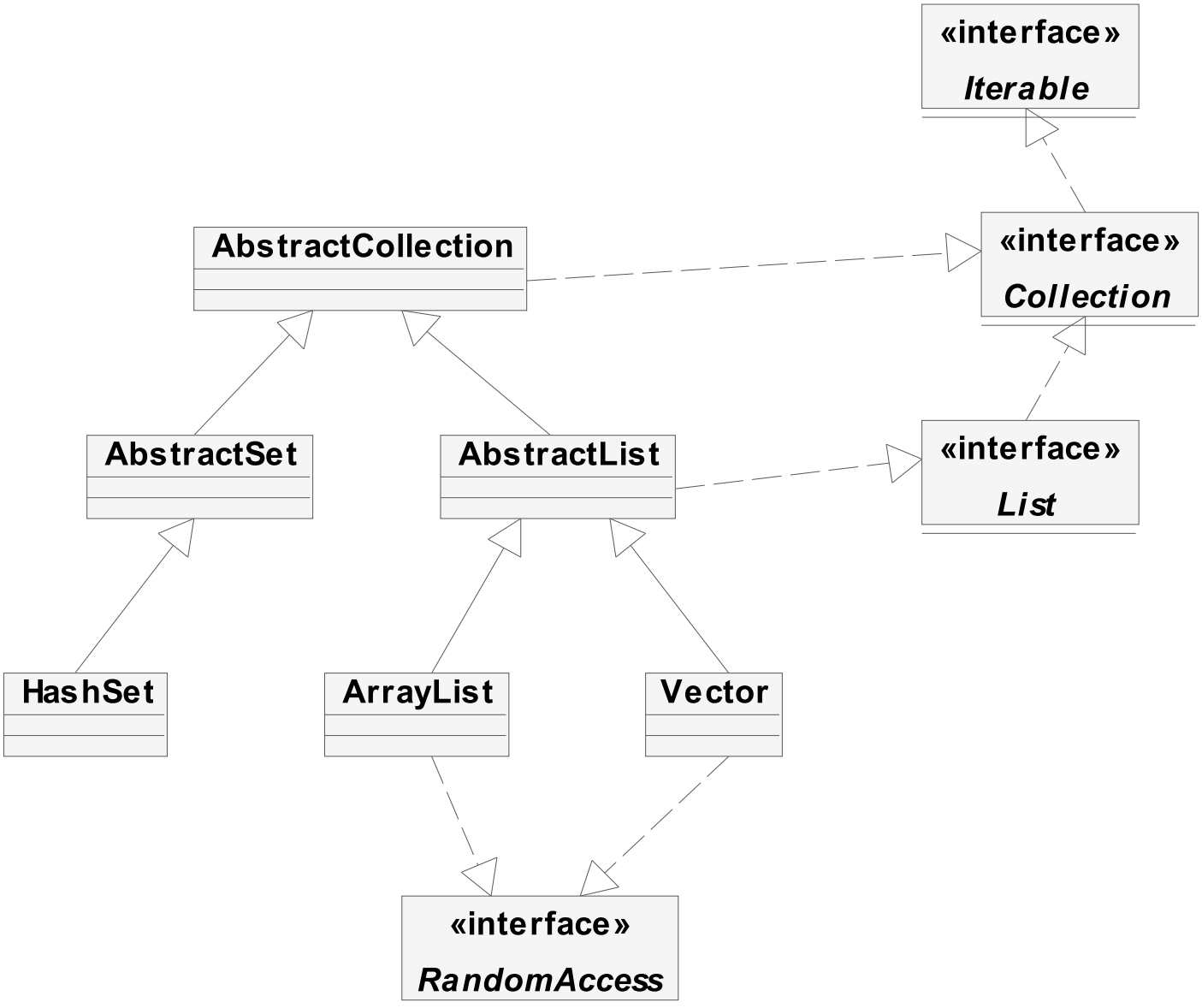
```
(1) Car c = new Car();

(2) Bus b = new Bus();

(3) Vehicle vc = c;

(4) Object ob = b;

(5) Car cc = (Car) ob;

(6) Bus bb = (Bus) ob;

(7) Car ccc = (Car) vc;
```

Použití rozhraní

- Tam, kde stačí funkcionalita definovaná rozhraním (lze používat pouze metody deklarované rozhraním).
- Typ proměnné definujeme jako rozhraní (ne třídu, která rozhraní implementuje).
- Do proměnné lze přiřadit libovolný objekt, který implementuje uvedené rozhraní.

- **Umožňuje větší flexibilitu kódu při zachování statické typové kontroly.**

Použití rozhraní



Použití rozhraní

Rozhraní (**java.util**):

```
public interface Collection ...
```

Implementující třídy:

AbstractCollection, AbstractList, AbstractSet, ArrayList, BeanContextServicesSupport, BeanContextSupport, HashSet, LinkedHashSet, LinkedList, TreeSet, Vector

---

Třída **Vector**:

```
public class Vector ...  {
   ...
   public Vector(Collection c) ...
   ...
}
```

Rozhraní, třída a objekt v Javě

Rozhraní

- specifikuje množinu vlastností, ale *neimplementuje je*

Abstraktní třída

- částečně implementuje rozhraní

Třída

- implementuje rozhraní (tj. všechny metody rozhraní)

Objekt

- objekt je *instancí* třídy

Pole

Pole

- Pole v Javě je speciálním objektem.
- Můžeme mít pole jak primitivních, tak objektových hodnot
  - pole primitivních hodnot tyto hodnoty obsahuje
  - pole objektů obsahuje odkazy na objekty
- Kromě pole v Javě existují i jiné objekty na ukládání více hodnot – *kontejnery* (bude později ...)

Pole – I

Před použitím je nutné pole

- deklarovat
- vytvořit
- inicializovat (naplnit)

Syntaxe deklarace

- **`typ [] identifikator`**
- na rozdíl od C/C++ nikdy neuvádíme při deklaraci počet prvků pole – ten je podstatný až při vytvoření objektu pole

Pole – II

Vytvoření pole

- jako u jiného objektu – voláním konstruktoru:
  - `nazevPole = new typ[velikost];`
  - `int[] pole = new int[10];`
- nebo inicializací při deklaraci:
  - `int[] nazevPole = { 1, 2, 3 };`

Syntaxe přístupu k prvkům

- `pole[i]`
- `pole[i] = 20;`
- `int j = pole[2];`

Pole – III

```
Ucet [] ucty;                    // deklarace pole
ucty = new Ucet[5];              // vytvoření pole

// vytvoření objektu
// a inicializace 1.  prvku pole
ucty[0] = new Ucet("Franta");

ucty[0].vypisInfo();  // přístup k prvku pole
```

- V poli **ucty** je naplněn 1. prvek odkazem na objekt
- Ostatní prvky zůstaly naplněny prázdnými odkazy **null**.

Pole – IV

Co když vynecháme vytvoření pole?

```
Ucet [] ucty;
ucty[0] = new Ucet("Franta");
   // chyba, pole neexistuje
```

Co když vynecháme inicializaci pole?

```
Ucet [] ucty;
ucty = new Ucet[5];
ucty[0].vypisInfo();
   // chyba, prvek neexistuje
```

Kopírování pole

Přiřazení proměnné objektového typu (a tedy i polí) vede pouze k duplikaci odkazu, nikoli celého odkazovaného objektu.

```
Ucet [] ucty = new Ucet[5];
Ucet [] ucty2;
ucty2 = ucty;
```

Proměnná **ucty2** obsahuje odkaz na stejné pole jako **ucty**.

Kopírování pole

```
Ucet [] ucty2 = new Ucet[5];
System.arraycopy(ucty, 0, ucty2, 0,
                 ucty.length);
```

- Proměnná **`ucty2`** obsahuje kopii původního pole.
- Také **`arraycopy`** však do cílového pole zduplikuje jen odkazy na objekty, nevytvoří kopie objektů!

Výpis argumentů programu

```
public class Pole {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++)
            System.out.println(args[i]);
    }
}
```

Objektová verze primitivních datových typů

- int $\longrightarrow$ Integer
- long $\longrightarrow$ Long
- short $\longrightarrow$ Short
- byte $\longrightarrow$ Byte
- char $\longrightarrow$ Character
- float $\longrightarrow$ Float
- double $\longrightarrow$ Double
- boolean $\longrightarrow$ Boolean
- void $\longrightarrow$ Void

Double.MAX_VALUE
float f = Float.parseFloat(řetězec)

Ukázka

```
int x;
String s = "28";

x = Integer.parseInt(s);

// zbytecne vytvari objekt
x = (new Integer(s)).intValue();

// zbytecne vytvari objekt
// Autoboxing (Java 5.0)
x = new Integer(s);

s = String.valueOf(x);
s = Integer.toString(x);
```

Další ...

- informace k projektu a týmovým úkolům
- práce s svn