Generické implementace

Main Entry: [1]**ge·ner·ic**
Pronunciation: j&-'ner-ik
Function: *adjective*
Etymology: French *générique*, from Latin *gener-*, *genus* birth, kind, class
1 a : relating to or characteristic of a whole group or class **:** **GENERAL** **b :** being or having a nonproprietary name **c :** having no particularly distinctive quality or application
2 : relating to or having the rank of a biological genus
- **ge·ner·i·cal·ly** /-i-k(&-)lE/ *adverb*
- **ge·ner·ic·ness** *noun*

Meriam-Webster Online Dictionary

Entry Word: **generic**
Function: *adjective*
Text: **Synonyms** GENERAL 2, common, universal
Antonyms specific

Meriam-Webster Online Thesaurus

In metaphysics, epistemology, and logic, a general category, such as a property or relation, considered as distinct from the particular things that instantiate or exemplify it.

The problem of universals concerns the question of what sort of being should be ascribed to such categories (e.g., is there any such thing as redness apart from particular red things?). The debate over the status of universals stems from Plato's theory of forms. Whereas Plato held that forms (universals) exist independently of particulars, Aristotle argued that forms exist only in the particulars in which they are exemplified. *See also* realism.

"Universal." *Britannica Concise Encyclopedia. 2004. Encyclopædia Britannica Premium Service.*
10 May 2004
<**http://www.britannica.com/ebc/article?eu=406883**>.

Naše téma:

Jak vytvořit generickou implementaci datové struktury nebo algoritmu?

Například:

Jak vytvořit generickou implementaci zásobníku nebo řazení?

Dědičnost

specifický pojem vs. universální pojem

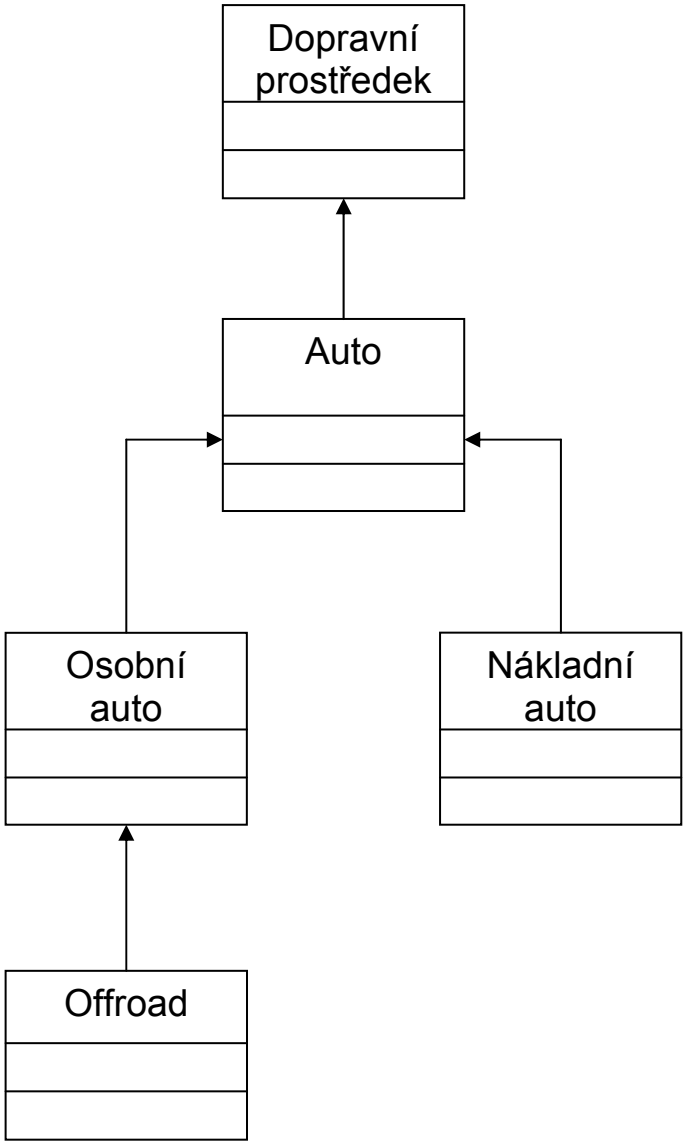
osobní auto je speciální druh *auta*
nákladní auto je speciální druh *auta*

vztah je relativní:

auto je speciálním druhem *dopravního prostředku*

offroad je speciální druh *osobního auta*

speciálnější druh má *další* nebo *změněné* vlastnosti



Poznámka: V UML diagramu jsou šipky „pázdné“.

osobní auto *nákladní auto*	*auto*
podtřída (subclass) **třída** **odvozená třída** **potomek**	**třída** **nadtřída** (superclass) **základní třída** **rodič, předek**

dědění
- realizováno již v jazyce Simula 67 (40 let)

potomek

- převezme všechny metody a proměnné rodiče

- může přidat nové metody a proměnné

- může předefinovat metody a proměnné rodiče (překrýt), nejde o přetížení

Java

- **`extends`** vytváří podtřídu

```
class B extends A {
...
}
```

- více tříd může být deklarováno jako podtřídy nějaké třídy

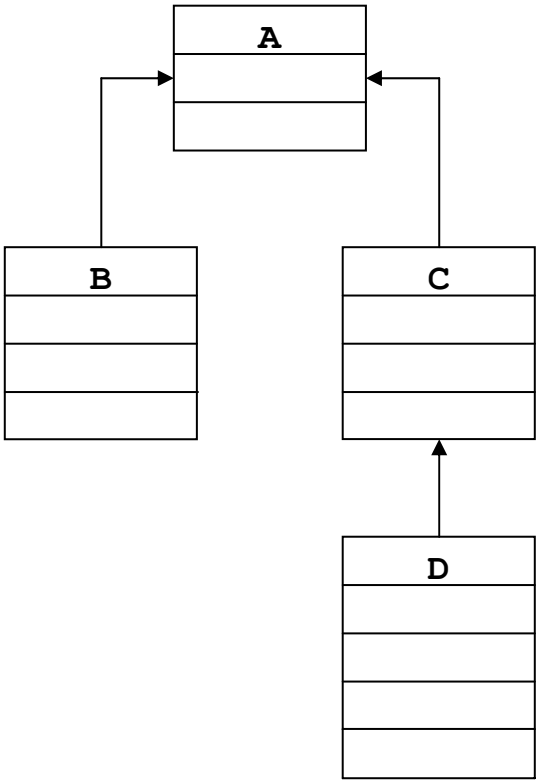
```
class C extends A {
...
}
```

- vztah rodič potomek může dále pokračovat vytvořením podtřídy D například třídy C

```
class D extends C {
...
}
```

- třída D je také podtřídou třídy A

- vznik hierarchie tříd


A
B
C
D

```
class Auto {
  final float kapacitaNadrze;
  float obsahNadrze;
  float spotreba;

  Auto(float kapacitaNadrze, float spotreba) {
    this.kapacitaNadrze = kapacitaNadrze;
    this.spotreba = spotreba;
  }
  Auto(Auto a) {
    kapacitaNadrze = a.kapacitaNadrze;
    spotreba = a.spotreba;
  }
  Auto() {
    kapacitaNadrze = 24.0F;
    spotreba = 8.0F;
  }
  float doplnNadrz() {
    float doplneni;
    doplneni = kapacitaNadrze - obsahNadrze;
    obsahNadrze = kapacitaNadrze;
    return doplneni;
  }
  float dojezd() {
    float dojede = obsahNadrze / spotreba *
                                        100.0F;
    return dojede;
  }
  void ujelo(float vzdalenost) {
    obsahNadrze = obsahNadrze - vzdalenost /
                              100.0F * spotreba;
  }
}
```

```
class OsobniAuto extends Auto {
  int maxOsob;
  int osoby;
  float aktualniSpotreba;

  OsobniAuto(float kapacitaNadrze,
                 float spotreba, int maxOsob) {
    super(kapacitaNadrze,spotreba); //konstruktor
    // super(...) musí být první příkaz
    this.maxOsob = maxOsob;
    aktualniSpotreba = spotreba;
  }
  private void zmenaSpotreby() {
    aktualniSpotreba = spotreba *
                          (1 + 0.02F * osoby);
  }
  void nastoupili(int n) {
    osoby += n;
    zmenaSpotreby();
  }
  void vystoupili(int n) {
    osoby -= n;
    zmenaSpotreby();
  }
  float dojezd() {
    float dojede = obsahNadrze /
                      aktualniSpotreba * 100.0F;
    return dojede;
  }
  void ujelo(float vzdalenost) {
    obsahNadrze = obsahNadrze - vzdalenost /
                      100.0F * aktualniSpotreba;
  }
}
```

Auto

```
kapacitaNadrze;
obsahNadrze;
spotreba;
Auto(...);
doplnNadrz();
dojezd();
ujelo();
```

OsobniAuto extends Auto

```
// převezme
// převezme
// převezme
    super();
// převezme
dojezd();      // překrytí
ujelo();       // překrytí
// přidáno
maxOsob;
osoby;
aktualniSpotreba;
OsobniAuto();
private zmenaSpotreby();
nastoupili();
vystoupili();
```

Aplikace

```
a = new OsobniAuto ...

a.doplnNadrz()
a.spotreba
                    a.dojezd()
                    a.nastoupili()
                    a.aktualniSpotreba
```

```
OsobniAuto a = new OsobniAuto(55.0F, 5.0F, 5);

System.out.println("Nacerpali jsme " +
                        a.doplnNadrz() + " l");

System.out.println("Spotreba je "+ a.spotreba
                                    +" l/100km");

System.out.println("Aktualni spotreba je " +
              a.aktualniSpotreba + " l/100km");

System.out.println("Dojezd je " + a.dojezd() +
                                          " km");

a.nastoupili(5);

System.out.println("Aktualni spotreba je " +
              a.aktualniSpotreba + " l/100km");

System.out.println("Dojezd je " + a.dojezd() +
                                          "km");
```

Nacerpali jsme 55.0 l
Spotreba je 5.0 l/100km
Aktualni spotreba je 5.0 l/100km
Dojezd je 1100.0 km
Aktualni spotreba je 5.5 l/100km
Dojezd je 1000.0 km

Auto	**OsobniAuto extends Auto**
`kapacitaNadrze;`	`// převezme`
`obsahNadrze;`	`// převezme`
`spotreba;`	`// převezme`
`Auto();`	`super();`
`doplnNadrz();`	`// převezme`
`dojezd();`	`dojezd();  // překrytí`
`ujelo();`	`ujelo();  // překrytí`
	`// přidáno`
	`maxOsob;`
	`osoby;`
	`aktualniSpotreba;`
	`OsobniAuto();`
	`private zmenaSpotreby();`
	`nastoupili();`
	`vystoupili();`

- je v objektech třídy **`OsobniAuto`** přístupná metoda **`dojezd()`** třídy **`Auto`** ?

- přístup k překrytým i nepřekrytým instančním metodám i proměnným **rodičovské** třídy umožňuje operátor **`super`**

```
super.jmeno;

super.dojezd();
```

```
class OsobniAuto extends Auto {
  int maxOsob;
  int osoby;
  float aktualniSpotreba;

  OsobniAuto(float kapacitaNadrze,
                 float spotreba, int maxOsob) {
    super(kapacitaNadrze,spotreba); //konstruktor
    this.maxOsob = maxOsob;
    aktualniSpotreba = spotreba;
  }
  private void zmenaSpotreby() {
    aktualniSpotreba = spotreba *
                          (1 + 0.02F * osoby);
  }
  void nastoupili(int n) {
    osoby += n;
    zmenaSpotreby();
  }
  void vystoupili(int n) {
    osoby -= n;
    zmenaSpotreby();
  }
  float dojezd() {
    float dojede = obsahNadrze /
                      aktualniSpotreba * 100.0F;
    return dojede;
  }
  float dojezdBezOsob() {
    float dojede = super.dojezd(); //metoda
    return dojede;                 //třídy Auto
  }
  void ujelo(float vzdalenost) {
    obsahNadrze = obsahNadrze - vzdalenost /
                      100.0F * aktualniSpotreba;
  }
}
```

```
OsobniAuto a = new OsobniAuto(55.0F, 5.0F, 5);

System.out.println("Spotreba je "+
              a.spotreba +" l/100km");

System.out.println("Aktualni spotreba je " +
              a.aktualniSpotreba + " l/100km");

System.out.println("Nacerpali jsme " +
              a.doplnNadrz() + " l");

a.nastoupili(5);

System.out.println("Aktualni spotreba je " +
              a.aktualniSpotreba + " l/100km");

System.out.println("Dojezd je " +
              a.dojezd() + " km");

System.out.println("Dojezd bez osob je " +
              a.dojezdBezOsob() + " km");
```

Spotreba je 5.0 l/100km
Aktualni spotreba je 5.0 l/100km
Nacerpali jsme 55.0 l
Aktualni spotreba je 5.5 l/100km
Dojezd je 1000.0 km
Dojezd bez osob je 1100.0 km

Implicitní konstruktor / konstruktor bez parametrů

- je volán automaticky

```
class Auto {
  final float kapacitaNadrze;
  float obsahNadrze;
  float spotreba;
...
  Auto() {
    System.out.println("Ja, bez parametru");
    kapacitaNadrze = 24.0F;
    spotreba = 8.0F;
  }
...
}

class OsobniAuto extends Auto {
  int maxOsob;
  int osoby;
  float aktualniSpotreba;

  OsobniAuto(int maxOsob) {
    // konstruktor Auto()je volán automaticky
    System.out.println("Ted ja, dodam
                                    maxOsob");
    this.maxOsob = maxOsob;
    aktualniSpotreba = spotreba;
  }
...

}
```

```
OsobniAuto a = new OsobniAuto(5);

System.out.println("Nacerpali jsme " +
              a.doplnNadrz() + " l");

System.out.println("Spotreba je "+
              a.spotreba +" l/100km");

System.out.println("Aktualni spotreba je " +
              a.aktualniSpotreba + " l/100km");

System.out.println("Dojezd je " +
              a.dojezd() + " km");

a.nastoupili(5);

System.out.println("Aktualni spotreba je " +
              a.aktualniSpotreba + " l/100km");

System.out.println("Dojezd je " +
              a.dojezd() + " km");
```

Ja, bez parametru
Ted ja, dodam maxOsob
Nacerpali jsme 24.0 l
Spotreba je 8.0 l/100km
Aktualni spotreba je 8.0 l/100km
Dojezd je 300.0 km
Aktualni spotreba je 8.8 l/100km
Dojezd je 272.72726 km

SOUTĚŽ

```
class NakladniAuto extends Auto {
  float maxNaklad;
  float naklad;
  float aktualniSpotreba;

  NakladniAuto(float kapacitaNadrze,
               float spotreba, float maxNaklad) {
    super(kapacitaNadrze, spotreba);
    this.maxNaklad = maxNaklad;
    aktualniSpotreba = spotreba;
  }
  private void zmenaSpotreby() {
    aktualniSpotreba = spotreba * (1 +
                                naklad/maxNaklad);
  }
  void naloz(float n) {
    naklad += n;
    zmenaSpotreby();
  }
  void vyloz(float n) {
    naklad -= n;
    zmenaSpotreby();
  }
  float dojezd() {
    float dojede = obsahNadrze /
                       aktualniSpotreba * 100.0F;
    return dojede;
  }
  void ujelo(float vzdalenost) {
    obsahNadrze = obsahNadrze - vzdalenost /
                       100.0F * aktualniSpotreba;
  }
}
```

```
class OsobniAuto extends Auto …
class NakladniAuto extends Auto …

OsobniAuto a = new OsobniAuto(55.0F, 5.0F, 5);
NakladniAuto b = new NakladniAuto(90.0F, 15.0F, 3.0F);
```

objekty **a** i **b** jsou také objektami třídy **Auto**

```
Auto x = a;                    Auto y = b;

x instanceof                   y instanceof
OsobniAuto                     NakladniAuto


(OsobniAuto)x                  (NakladniAuto)y
```

v poli **auta** jsou osobní i nákladní auta

```
Auto[] auta;
...
for (int i=0; i<pocetAut; i++) {
  if (auta[i] instanceof OsobniAuto) {
    System.out.println("Osobni - max osob: " +
                ((OsobniAuto)auta[i]).maxOsob);
  }
  if (auta[i] instanceof NakladniAuto) {
    System.out.println("Nakladni - max naklad:" +
                ((NakladniAuto)auta[i]).maxNaklad);
  }
}
```

- v Javě má každá třída jedinou rodičovskou třídu

- jediná třída, která nemá rodičovskou třídu a současně je tak nadtřídou všech tříd je třída **`Object`**

-objekty každé třídy jsou také objekty třídy **`Object`**

```
class Zasobnik {
  private Object[] z;
  private int vrchol;
  final int maxN=10;

  Zasobnik() {
    z = new Object[maxN];
    vrchol = 0;
  }

  boolean jePrazdny() {
    return (vrchol == 0);
  }

  void push(Object klic) {
    z[vrchol++] = klic;
  }

  Object pop() {
    Object t = z[--vrchol];
    z[vrchol] = null;
    return t;
  }
}
```

```
Zasobnik z = new Zasobnik();

A a = new A();
B b = new B();

z.push(a);
z.push(b);
...
b = (B)z.pop(); // správně
a = (A)z.pop();


z.push(a);
z.push(b);
...
a = (A)z.pop(); //Exception in thread "main"
                //java.lang.ClassCastException: B...
b = (B)z.pop();
```

IntZasobnik	**Zasobnik**
int	**Object**

pro základní datové typy musíme použít obalující třídy
(do verze 1.5)

pro **int** třída **Integer**

```
int x;
z.push(new Integer(x))

((Integer) z.pop()).intValue()
```

```
class IntZasobnik {
  // ADT rozhrani
  IntZasobnik()          // vytvoření prázdného zásobníku
  boolean jePrazdny()    // test je-li prázdný
  void push(int)         // vložení prvku
  int pop()              // výběr prvku
}
```

požadované ADT rozhraní implementuje adaptér

```
class IntZasobnik {
  private Zasobnik z;
  IntZasobnik() {
    z = new Zasobnik();
  }

  boolean jePrazdny() {
    return z.jePrazdny();
  }

  void push(int klic) {
    z.push(new Integer(klic));
  }

  int pop() {
    return ((Integer)z.pop()).intValue();
  }
}
```

Java 1.5

```
class IntZasobnikApl {

  public static void main(String[] arg) {
    Zasobnik z = new Zasobnik();

    if (z.jePrazdny())
      System.out.println("zasobnik je prazdny");

    z.push(new Integer(1));
    // ver. 1.5
    z.push(2);

    int x = ((Integer) z.pop()).intValue();
    System.out.println(x);
    // ver. 1.5
    System.out.println(z.pop());
  }
}
```

Třída `Stack`

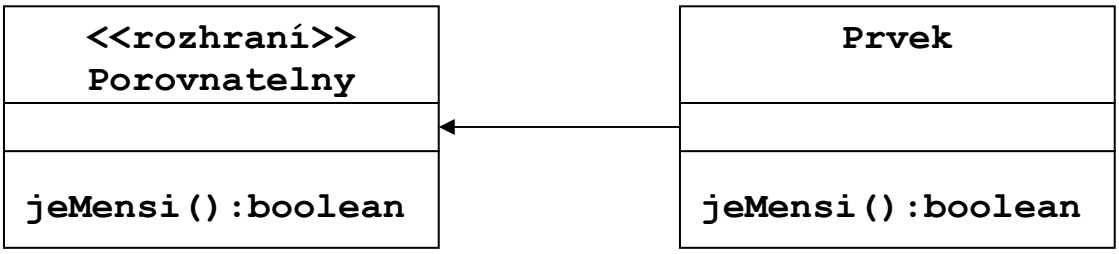
```
import java.util.Stack;

class IntZasobnikApl {
  public static void main(String[] arg) {
  Stack zasobnik = new Stack(); //čeho ?

  if (zasobnik.isEmpty())
    System.out.println("zasobnik je prazdny");

  zasobnik.push('A');
  zasobnik.push(3);
  zasobnik.push(2);
  zasobnik.push(1);

  System.out.println(zasobnik.pop());
  System.out.println(zasobnik.pop());
  System.out.println(zasobnik.pop());
  System.out.println(zasobnik.pop());

  }
}
```

```
import java.util.Stack;

class IntZasobnikAplChecked {
  public static void main(String[] arg) {

  Stack <Integer> zasobnik = new Stack <Integer> ();
  // teď je pro int

  if (zasobnik.isEmpty())
    System.out.println("zasobnik je prazdny");

  zasobnik.push(4);
  zasobnik.push(3);
  zasobnik.push(2);
  zasobnik.push(1);

  System.out.println(zasobnik.pop());
  System.out.println(zasobnik.pop());
  System.out.println(zasobnik.pop());
  System.out.println(zasobnik.pop());

  }
}
```

Rozhraní

-rozhraní definuje soubor metod, bez jejich implementace

-třída, která implementuje toto rozhraní (tj. jakoby jej zdědí), musí implementovat (tj. jakoby překrýt) všechny metody rozhraní



- deklarace rozhraní

```
interface jmeno {
// hlavičky metod
}
```

- implementující třída

```
class trida implements rozhrani {
// deklarace metod
}
```

- k instancím (objektům) implementující třídy přistupujeme pomocí referenční proměnné typu rozhraní

- obdoba přístupu k instancím podtřídy pomocí referenční proměnné nadtřídy

Musíme pro každý typ řazených prvků napsat novou třídu (program) ?

```
interface Porovnatelny {
  boolean mensi(Porovnatelny v);
}
```

- generická třída řazení vkládáním prvků iplementujících rozhraní **Porovnatelny**

```
class Razeni {
  static void razeniVkladanim(
                            Porovnatelny[] a) {
    for(int i = 1; i < a.length; i++) {
      int j = i;
      Porovnatelny v = a[j];

      for( ; j > 0 && v.mensi(a[j - 1]); j--)
        a[j] = a[j - 1];
      a[j] = v;
    }
  }
}
```

- pro řazení celočíselných klíčů definujeme třídu **IntPrvek**

```
class IntPrvek implements Porovnatelny {
  int hodnota;
  IntPrvek(int x) {
    hodnota = x;
  }
  int getInt () {
    return hodnota;
  }
  public boolean mensi(Porovnatelny v) {
    return hodnota < ((IntPrvek) v).hodnota;
  }
}


class RazeniApl {
  public static void main(String [] args) {
    int[] a = new int [3];

    a[0] = 4;
    a[1] = 3;
    a[2] = 5;

  // vytvoříme pole intPole prvků třídy IntPrvek
  IntPrvek[] intPole = new IntPrvek[a.length];
  for(int i = 0; i < a.length; i++)
    intPole[i] = new IntPrvek(a[i]);

  // zavoláme metodu razeniVkladanim() třídy Razeni
  // s parametrem intPole
  Razeni.razeniVkladanim(intPole);

  for(int i = 0; i < a.length; i++)
    System.out.println(intPole[i].getInt());
  }
}
```