

Objektové modelování, AD7B36OMO

Návrhové vzory

Tvorba objektů

JIŘÍ ŠEBEK

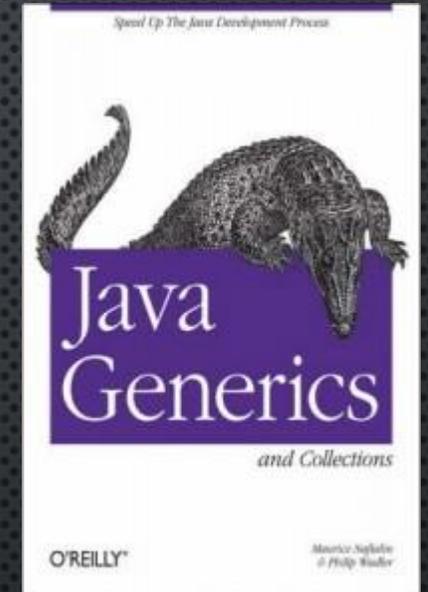
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

<https://edux.feld.cvut.cz/courses/AD7B36OMO>

Přednáška je vytvořena na základě původních přednášek autorů: Karel Richta;Pavel Strnad, 2014

CO TO JSOU GENERIKA?

- UMOŽŇUJE POUŽÍVÁNÍ GENERICKÝCH TYPŮ PŘI DEKLARACI TŘÍD, ROZHRANÍ A METOD
- OD JDK 1.5



CO TO JSOU GENERIKA?



MOTIVACE

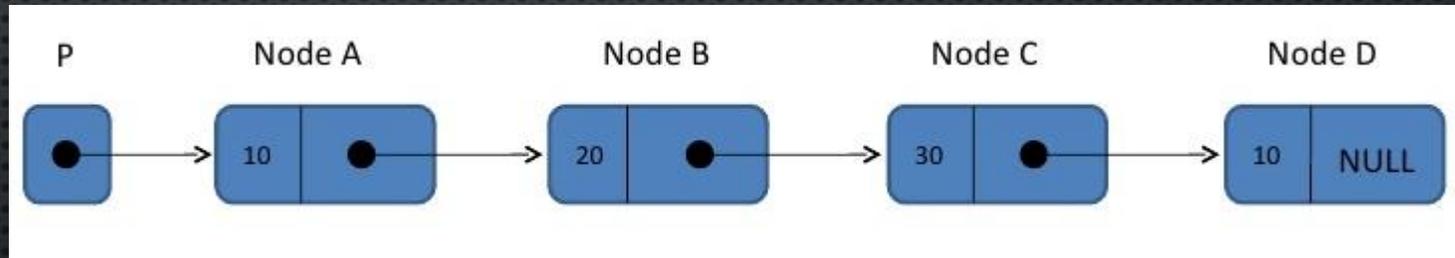
CÍLEM JE:

- NEBÝT OMEZENI DATOVÝMI TYPY
 - TYP **OBJECT** KOMPLIKUJE/ZNEMOŽŇUJE DATOVOU KONTROLU
- ROZŠÍŘENÍ TYPOVÉ KONTROLY
- VYUŽÍT KÓD OPAKOVANĚ S RŮZNÝMI DATOVÝMI TYPY



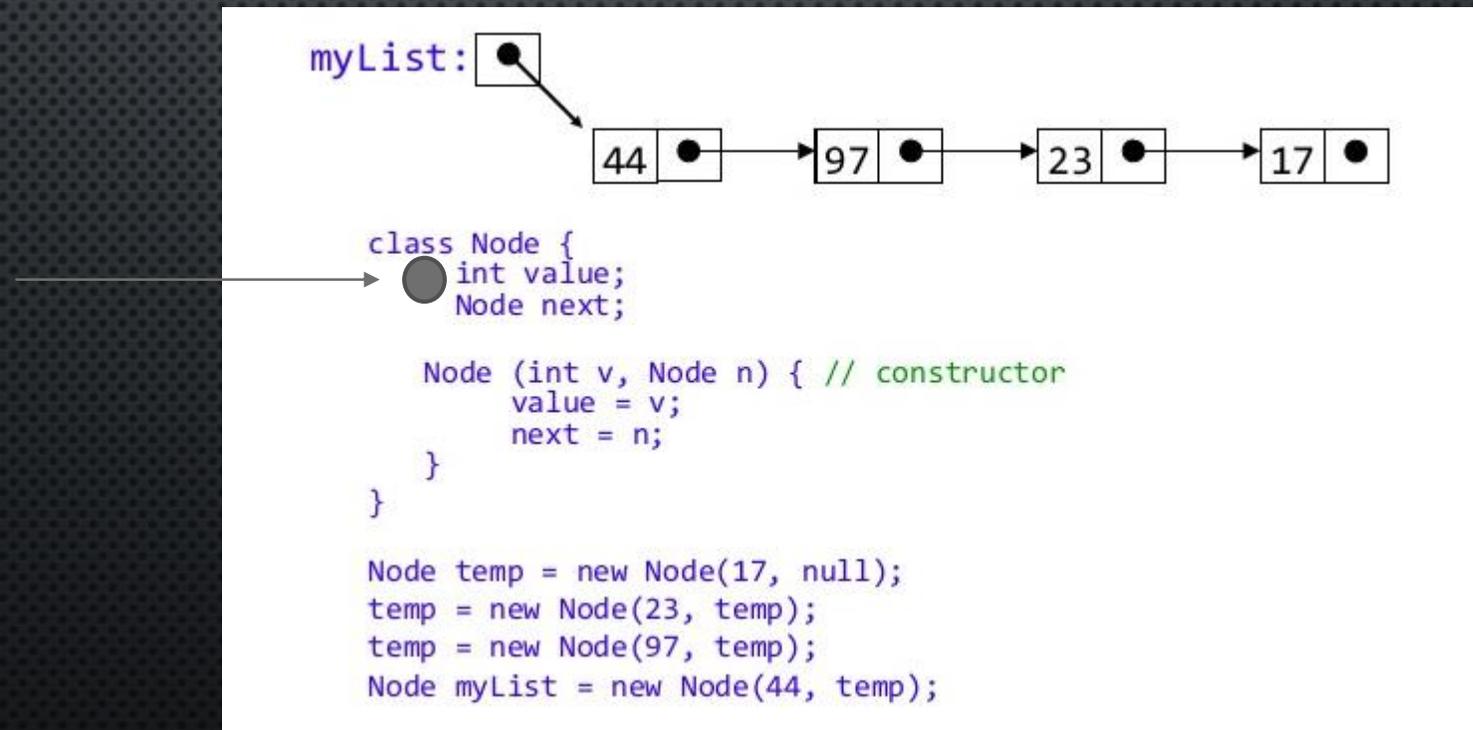
PŘÍKLAD

- SPOJOVÝ SEZNAM (JENDOSMĚRNÝ, OBOUSMĚRNÝ, CYKICKÝ ...)
 - BEZ GENERIK
 - S GENERIKY



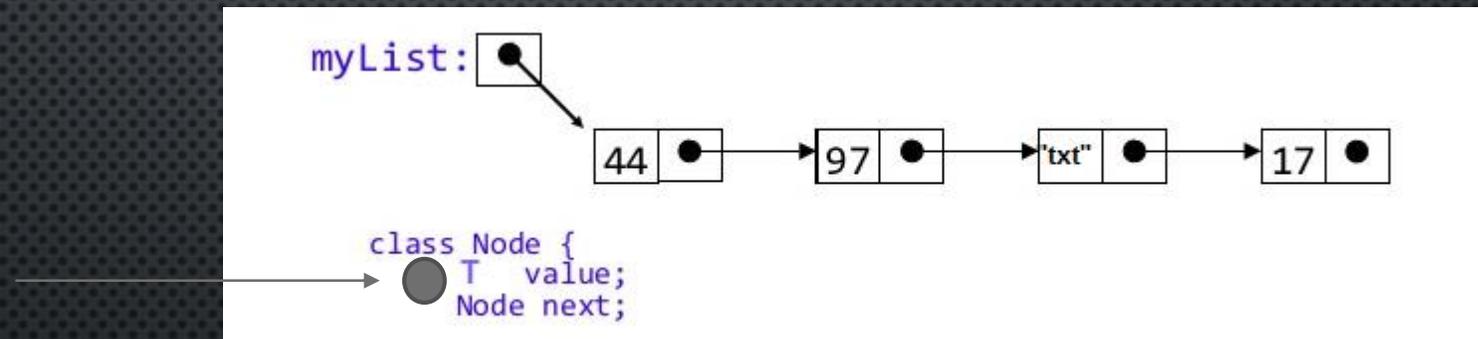
PŘÍKLAD

- SPOJOVÝ SEZNAM (JENDOSMĚRNÝ, OBOUSMĚRNÝ, CYKlickÝ ...)
 - BEZ GENERIK
 - S GENERIKY



PŘÍKLAD

- SPOJOVÝ SEZNAM (JENDOSMĚRNÝ, OBOUSMĚRNÝ, CYKlickÝ ...)
 - BEZ GENERIK
 - S GENERIKY



PŘÍKLAD ARRAYLIST (VŠICHNI POUŽÍVÁME)

```
public class SimpleList {  
    public static void main(String[] args) {  
        List<String> words = new ArrayList<>();  
        words.add("hi");  
        words.add("hello");  
        words.add("hola");  
        System.out.println("First word: " + words.get(0));  
        System.out.println("All words: " + words);  
    }  
}
```

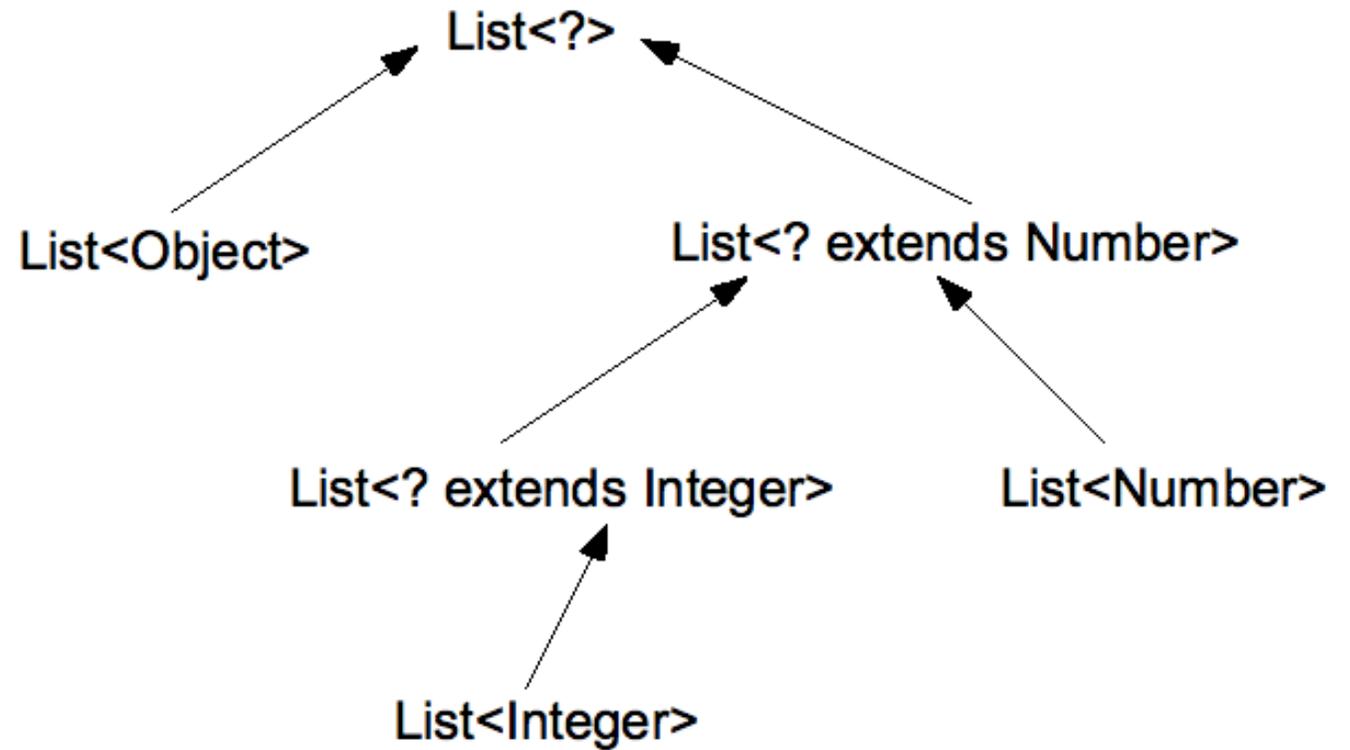
PŘÍKLAD LINKEDLIST (VŠICHNI POUŽÍVÁME)

```
LinkedList<String> list = new LinkedList<String>();  
list.add("abc");  
String s = list.get(0); // není nutnost přetypovat
```

```
LinkedList list = new LinkedList();  
list.add("abc");  
String s = (String)list.get(0); // nutné přetypování
```

DALŠÍ PŘÍKLAD

- SAMOZDŘEJME MŮŽEME VYUŽÍVAT DĚDIČNOST



Taxonomy of the generic *List* types

GENERICKÉ TYPY

JSOU TŘÍDY ČI INTERFACES DEKLARUJÍCÍ TZV. TYPOVÉ PARAMETRY JIMIŽ:

- SYSTEMATIZUJÍ TYPOVOU KONTROLU KOMPILÁTOREM,
- VYJADŘUJÍ JASNĚJI SMYSL, ZLEPŠUJÍ ČITELNOST A ROBUSTNOST PROGRAMU,
- ULEHČUJÍ PSANÍ PROGRAMU V IDE,
- V CLASS-SOUBORECH JSOU VYZNAČENY, ALE LZE JE POMINOUT (TZV. ERASURE),
- V RUNTIME SE NIJAK NEUPLATŇUJÍ – TAM JSOU JEN TZV. HRUBÉ (RAW) TYPY.
- UŽÍVAJÍ SE ZEJMÉNA V KOLEKCÍCH K VYMEZENÍ TYPŮ PRVKŮ.

GENERICKÉ TYPY

TYPOVÉ PARAMETRY LZE UŽÍT JEN NA REFERENČNÍ TYPY, ALE **NIKOLI**:

- VE STATICKÉM KONTEXTU: `STATIC VOID M() { LIST<T> LST= ... }`
- POLE A VYTVOŘENÍ POLE: `E[] E = (E[]) NEW OBJECT[5]; NEW E[3]
NEW ARRAYLIST<STRING>[7]`
- PRIMITIVNÍ TYPY: `LIST<INT>`
- STATICKÉ ATRIBUTY: `STATIC T T`
- VYTVOŘIT INSTANCI: `NEW E()`
- POTOMKY TŘÍDY THROWABLE: `CLASS MYEX<T> EXTENDS EXCEPTION { }`

GENERICKÉ TYPY

```
CLASS PAIR<K, V> {
```

```
    PRIVATE K KEY;
```

```
    PRIVATE V VALUE;
```

```
    PUBLIC PAIR(K KEY, V VALUE) {
```

```
        THIS.KEY = KEY;
```

```
        THIS.VALUE = VALUE;
```

```
    }
```

```
    // ...
```

```
}
```

```
Pair<int, char> p = new Pair<>(8, 'a'); // compile-time error
```

```
Pair<Integer, Character> p = new Pair<>(8, 'a'); // OK
```

GENERICKÉ TYPY

```
PUBLIC STATIC <E> VOID APPEND(LIST<E> LIST) {  
    E ELEM = NEW E(); // COMPILE-TIME ERROR  
    LIST.ADD(ELEM);  
}
```

```
PUBLIC STATIC <E> VOID APPEND(LIST<E> LIST, CLASS<E> CLS) THROWS EXCEPTION {  
    E ELEM = CLS.NEWINSTANCE(); // OK  
    LIST.ADD(ELEM);  
}
```

GENERICKÉ TYPY

```
PUBLIC CLASS MOBILEDEVICE<T> {  
    PRIVATE STATIC T OS; // NEMŮŽEME!  
  
    // ...  
}
```

GENERICKÉ TYPY

```
PUBLIC STATIC <E> VOID RTTI(LIST<E> LIST) {  
    IF (LIST INSTANCEOF ARRAYLIST<INTEGER>) { // COMPILE-TIME ERROR  
        // ...  
    }  
}
```

```
PUBLIC STATIC VOID RTTI(LIST<?> LIST) {  
    IF (LIST INSTANCEOF ARRAYLIST<?>) { // OK  
        // ...  
    }  
}
```

SYNTAX HLAVIČEK

```
... class JménoTřidy [ < Deklarace > ] // generická třída
    [ extends JménoNadTřidy [ < TypArgument, ... > ] ]
    [ implements JménoInterfejsu [ < TypArgument, ... > ], ... ] {...}

... interface JménoInterface [ < Deklarace > ] // generický interface
    [ extends JménoNadInterface [ < TypArgument, ... > ], ... ] {...}

... [ < Deklarace > ] JménoTřidy // generický konstruktor
    ( [ [ final ] ( Typ | TypParm ) jménoParametru, ... ] ) {...}

... [ [ static ] [ final ] | [ abstract ] ] // generická metoda
    [ < Deklarace > ] ( void | ReturnTyp | TypParm ) jménoMetody
    ( [ [ final ] ( Typ | TypParm ) jménoParametru, ... ] ) ( {...} | ; )
```

< Deklarace > je seznam typových parametrů: $\langle T_1, T_2, \dots, T_i, \dots, T_n \rangle$
 T_i ve třídě či interface jsou globální a nelze je užít ve statickém kontextu,
 T_i konstruktoru či metodě jsou lokální a lze je zastínit.

TYPOVÝ PARAMETR

DEKLAROVANÝ V SEZNAMU $\langle T_1, T_2, \dots, T_i, \dots, T_n \rangle$ MŮŽE BÝT VYMEZEN TAKTO:

- T_i [EXTENDS JAVA.LANG.OBJECT] - JE JAKÁKOLI TŘÍDA.
- T_i EXTENDS Z - Z JE JIŽ ZNÁMÁ TYPOVÁ PROMĚNNÁ.
- T_i EXTENDS (R | I) [& I & I ...] - R JE TŘÍDA
I INTERFACE
& - LOGICKÝ SOUČIN.
- TYPOVÉ PARAMETRY SE OBVYKLE ZNAČÍ JEDNOPÍSMENNÝM IDENTIFIKÁTOREM:
- T, E, K, V, N, S, A PRO: TYPE, ELEMENT, KEY, VALUE, NUMBER, SERVICE, ACTION

TYPOVÝ PARAMETR

PŘÍKLADY :

```
CLASS DEMO < T, E EXTENDS Y & K , N EXTENDS J & K > { }
```

```
PUBLIC < Y, K > K DEMO( Y Y, K K, INT I ) { ... RETURN K; }
```

KDE Y JE NAPŘ. TŘÍDA – J A K JSOU INTERFACES.

PARAMETRIZOVANÝ OBJEKT

VYTVOŘÍ KONSTRUKTOR S TYPOVÝMI ARGUMENTY V POČTU A S VYMEZENÍM
FORMÁLNÍCH PARAMETRŮ DLE DEFINICE TYPU:

```
NEW DEMO < A, B, C > ( ... )
```

KDE A, B, C JSOU VHODNÉ REFERENČNÍ TYPY, TJ. TŘÍDY NEBO INTERFACE.

PARAMETRIZOVANÝ OBJEKT

PARAMETRIZOVANÝ OBJEKT LZE REFEROVAT ČTYŘMI ZPŮSOBY:

- `DEMO REF0 ; // RAW: OBJECT, OBJECT, OBJECT`
- `DEMO < A ,B,C > REF1 ; // REF1: A,B,C`
- `REF1 = NEW DEMO < A ,B,C > (...); // DECORATED TYPE`
- `REF1 = NEW DEMO <> (...); // INFERENCE DIAMANTEM <> OD V. 7`
- `REF0 = REF1 ; // ERASURE`
- `REF0 = NEW DEMO (...); // RAW TYPE`
- `REF1 = REF0 ; // UNCHECKED CONVERSION`



task



Intervaly

uzavřený interval $[a,b]$



otevřený interval (a,b)



zleva uzavřený $[a,b)$
zprava otevřený $[a,b)$



zleva otevřený $(a,b]$
zprava uzavřený $(a,b]$



- Prvky v intervalu se musí umět porovnat
- Intervaly se mezi sebou musí umět porovnat

PŘÍKLAD: PARAMETRIZOVANÝ INTERVAL

```
public class Interval ... {  
    // ... low, high;  
  
    public Interval(... ) {  
        if ( low.compareTo( high ) >0 ) throw new  
IllegalArgumentException( );  
        this.low=low; this.high=high;  
    }  
    @Override  
    public int compareTo( ... ) {  
        return this.low.compareTo( that.low );  
    }  
    @Override  
    public String toString( ) { return "Interval["+low+", "+high+"]"; }  
}
```

PŘÍKLAD: PARAMETRIZOVANÝ INTERVAL

```
public class Interval <E extends Comparable<E>>
    implements Comparable<Interval<E>> {
    E low, high;

    public Interval( E low, E high ) {
        if ( low.compareTo( high ) >0 ) throw new IllegalArgumentException( );
        this.low=low; this.high=high;
    }
    @Override
    public int compareTo( Interval<E> that ) {
        return this.low.compareTo( that.low );
    }
    @Override
    public String toString( ) { return "Interval["+low+", "+high+"]"; }
}
class XI <E extends Comparable<E>> extends Interval<E> {
    public XI( E a, E b ) { super(a,b); }
}
```

TYPOVÝ ARGUMENT SE ŽOLÍKEM (WILDCARD)

REFERENCE, TJ. PROMĚNNÉ A PARAMETRY METOD, LZE VYMEZIT JAKO MNOŽINU PŘIJATELNÝCH NEZNÁMÝCH TYPŮ TAKTO:

< ? [EXTENDS TYP | EXTENDS JAVA.LANG.OBJECT] >

- ZDE EXTENDS Zahrnuje i IMPLEMENTS

TYP VYZNAČUJE HORNÍ MEZ VČETNĚ.

NELZE VKLÁDAT NEBO MĚNIT HODNOTY (KROMĚ NULL)

- JEN K NIM PŘISTUPOVAT (A V KOLEKCÍCH I ODSTRAŇOVAT) .

< ? SUPER TYP > - TYP VYZNAČUJE DOLNÍ MEZ VČETNĚ.

LZE VKLÁDAT HODNOTY.

TYPOVÝ ARGUMENT SE ŽOLÍKEM (WILDCARD)

NAPŘ.: PRO VÝPIS LZE UŽÍT NEOMEZENÝ ŽOLÍK:

```
VOID PRINTANY( LIST< ? > C ) {           // NE < OBJECT >
    FOR ( OBJECT O : C ) SYSTEM.OUT.PRINTLN( O );
    C.GET( 0 );                          // LZE - NEMÁ PARAM. TYP
    C.ADD( NEW OBJECT( ) );              // MÁ PARAM. Ě - COMPILE ERROR
}
```

? V REFERENCI

ZADFINUJEME TŘÍDY ... <- A <- B <- C <- D REFEROVANÉ PROMĚNÝMI A, B, C, D .

```
CLASS BOX< V EXTENDS B > { // TATO KRABICE UMOŽNÍ ULOŽIT JEDNU VĚC
    V v ; // SPLŇUJÍCÍ: V INSTANCEOF B
    BOX ( V v ) { THIS.V = v ; } // NUTNO VLOŽIT B NEBO C NEBO NULL
    V GET ( ) { RETURN v ; }
    VOID SET ( V v ) { THIS.V = v ; } // VŽDY LZE VLOŽIT NULL
}
```

? V REFERENCI

KRABICE SLOUŽÍ K ŘÁDNÉMU ULOŽENÍ, REFERENCE VYMEZUJE VKLAD A VÝBĚR VĚCÍ:

```
BOX< B >          x1=NEW BOX<>( B );    x1.SET( B... );    B z1=x1.GET();
BOX< C >          x2=NEW BOX<>( C );    x2.SET( C... );    C z2=x2.GET();
BOX< ? SUPER B >  x3=NEW BOX<>( B );    x3.SET( B... );    B z3=x3.GET();
BOX< ? SUPER C >  x4=NEW BOX<>( B | C ); x4.SET( C... );    B z4=x4.GET();
BOX< ? EXTENDS B > x5=NEW BOX<>( B... );          B z5=x5.GET();
BOX< ? EXTENDS C > x6=NEW BOX<>( C... );          C z6=x6.GET();
BOX< ?[EXTENDS A] > x7=NEW BOX<>( B... );          B z7=x7.GET();
```

```
@ SUPPRESSWARNINGS( "RAWTYPES" )    // POTLAČENÍ PŘED METODOU ČI CLASS
```

```
BOX          x= NEW B<>( B | C );    x.SET( B | C );    B z= x.GET( );
```

DALŠÍ PŘÍKLAD

```
CLASS DEMO < X , Y EXTENDS NUMBER, Z EXTENDS ABSTRACTMAP < K, V >  
    IMPLEMENTS NAVIGABLEMAP < K, V > ... {
```

```
    PRIVATE COMPARATOR< ? SUPER K > COMPARATOR = NULL ;
```

```
    PUBLIC TREEMAP( COMPARATOR< ? SUPER K > C ) { COMPARATOR = C ; ...
```

```
    PUBLIC TREEMAP( MAP< ? EXTENDS K, ? EXTENDS V > M ) { PUTALL( M ); ...
```

```
    ...
```

```
}    Upravte kód nahoře, aby se dal spustit kód zde:
```

```
MAP < STRING, INTEGER > M = NEW TREEMAP < STRING, INTEGER > ( ) ;
```

```
M.PUT ( "AAA" , 1);
```

```
INT I = M.GET ( "AAA" );
```

DALŠÍ PŘÍKLAD

Co kdybychom chtěli dát podmínku na String a Integer?

```
CLASS TREEMAP< K, V > EXTENDS ABSTRACTMAP < K, V >  
    IMPLEMENTS NAVIGABLEMAP < K, V > ... {  
    PRIVATE COMPARATOR< ? SUPER K > COMPARATOR = NULL ;  
  
    PUBLIC TREEMAP( COMPARATOR< ? SUPER K > C ) { COMPARATOR = C ; ...  
  
    PUBLIC TREEMAP( MAP< ? EXTENDS K, ? EXTENDS V > M ) { PUTALL( M ) ; ...  
    ...  
}  
  
MAP < STRING, INTEGER > M = NEW TREEMAP < STRING, INTEGER > ( ) ;  
M.PUT ( "AAA" , 1 ) ;  
INT I = M.GET ( "AAA" ) ;
```



DALŠÍ PŘÍKLAD

PARAMETRIZOVAT METODY LZE I V NEPARAMETRIZOVANÝCH TŘÍDÁCH I INTERFACES.

```
PUBLIC STATIC < U EXTENDS DOUBLE, V > U M( U u, V v ) {  
    U x = u;  
    V y = v;  
    SYSTEM.OUT.PRINTLN( u.GETCLASS( ) +" "+ v.GETCLASS( ) );  
    SYSTEM.OUT.PRINTLN( x+" "+y );  
    RETURN x;  
}
```

Parametrizovaný konstruktor má smysl jen v parametrizovaných třídách.

....

```
NUMBER s = M ( 666.999, NEW STACK( ) ); // ČÍSLA PŘEVÁDÍ AUTOBOXING  
DOUBLE d = M ( .777, "BBB" );  
OBJECT o = M ( 3.14, 3L );
```

GENERIKA A SUBTYPY

DŮLEŽITÁ, BYĚ NEITUITIVNÍ, ZÁBRANA PROTI NARUŠENÍ VYMEZENÍ:

PRO VZTAH SUBTYP --> SUPERTYP (EXTENDS / IMPLEMENTS)

NEPLATÍ: L<SUBTYP> --> L<SUPERTYP>

PŘÍKLAD: CLASS DRIVER EXTENDS PERSON { ... }

```
LIST< DRIVER > DRIVERS = NEW ARRAYLIST<> ( );
```

```
LIST< PERSON > PEOPLE = DRIVERS; // ERROR
```

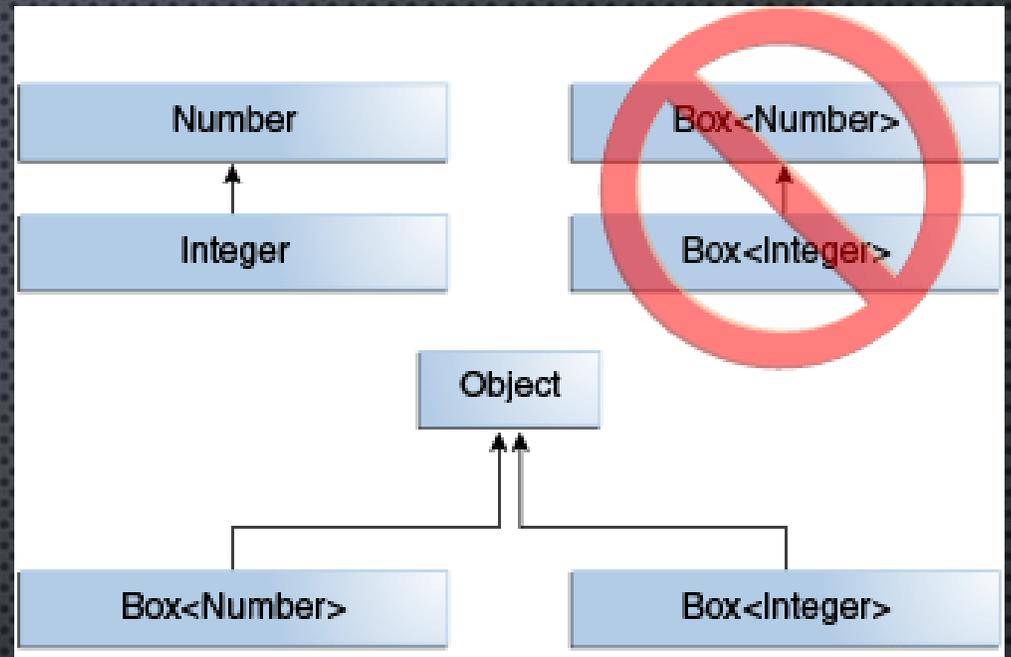
```
PEOPLE.ADD( NEW PERSON( ) ); // NON-DRIVER IN DRIVERS
```

GENERIKA A SUBTYPY

DŮLEŽITÁ, BYĚ NEUTIVNÍ, ZÁBRANA PROTI NARUŠENÍ VYMEZENÍ:

PRO VZTAH SUBTYP --> SUPERTYP

NEPLATÍ: L<SUBTYP> --> L<SUPERTYP>



PŘÍKLAD: `CLASS DRIVER EXTENDS PERSON { ... }`

```
LIST< DRIVER > DRIVERS = NEW ARRAYLIST<> ( );
```

```
LIST< PERSON > PEOPLE = DRIVERS; // ERROR
```

```
PEOPLE.ADD( NEW PERSON( ) ); // NON-DRIVER IN DRIVERS
```

ERASURE

- PROCES VYNUCOVÁNÍ TYPOVÝCH OMEZENÍ POUZE PŘI KOMPILACI A VYŘAZENÍ INFORMACÍ O TYPU PRVKU PŘI BĚHU

```
public static <E> boolean containsElement(E [] elements, E element){  
    for (E e : elements){  
        if(e.equals(element)){  
            return true;  
        }  
    }  
    return false;  
}
```

Před kompilací

```
PUBLIC STATIC BOOLEAN CONTAINS_ELEMENT(OBJECT [] ELEMENTS, OBJECT ELEMENT){  
  
    FOR (OBJECT E : ELEMENTS){  
        IF(E.EQUALS(ELEMENT)){  
            RETURN TRUE;  
        }  
    }  
    RETURN FALSE;  
}
```

Po kompilaci

ERASURE

```
COLLECTION<STRING> CS = NEW ARRAYLIST <>( );
                    CS.ADD( "AAA" );

COLLECTION          CR = CS;                // RAW TYPE
                    CR.ADD( 666 );          // UNSAFE OPERATION

COLLECTION<?>       CX = CS;
                    CX.ADD( "BBB" );       // ERROR

COLLECTION<? SUPER STRING > CX = CS;
                    CX.ADD( "CCC" );

COLLECTION<? EXTENDS STRING > CY = CS;
                    CY.ADD( "DDD" );       // ERROR

COLLECTION<OBJECT> CO = CS;                 // ERROR INCOMPATIBLE TYPES
```

ERASURE - PŘÍKLAD

- NAPIŠTE TŘÍDU HETEROGENNÍ KOLEKCE, DO KTERÉ MŮŽEME VLOŽIT RŮZNÉ OBJEKTY, ALE BUDE SPLŇOVAT COMPARABLE (MÁ METODU COMPARE ;))



ERASURE - PŘÍKLAD

```
LIST< COMPARABLE< ? > > LIST = NEW ARRAYLIST<>( );  
// VLOŽENÍ OBJEKTŮ RŮZNÝCH TŘÍD AVŠAK SPLŇUJÍCÍ COMPARABLE.  
  
COLLECTIONS.SORT( LIST, NEW MYCOMP( ) );           // ŘAZENÍ  
  
@ SUPPRESSWARNINGS( { "RAWTYPES" } )  
CLASS MYCOMP IMPLEMENTS COMPARATOR< COMPARABLE > {  
    @ OVERRIDE  
    @ SUPPRESSWARNINGS( { "UNCHECKED" } )  
    PUBLIC INT COMPARE( COMPARABLE O1, COMPARABLE O2 ) {  
        CLASS C1 = O1.GETCLASS( ), C2 = O2.GETCLASS( );  
        RETURN C1 == C2 ?  
            - O1.COMPARETO( O2 )           // TÉŽE TŘÍDY  
            : - C1.GETNAME( ).COMPARETO( C2.GETNAME( ) ); // RŮZNÝCH TŘÍD  
    }  
}
```

CO GENERIKA NEUMÍ

- JEDNÁ SE O „+“NAD KOMPILÁTOREM (V BYTEKÓDU NEEXISTUJÍ).
- NEMŮŽEME POUŽÍVAT TYPOVÉ PARAMETRY ZA BĚHU PROGRAMU (V DANÝ OKAMŽIK JIŽ NEEXISTUJÍ).
 - DANÉ TYPY PŘEDAT KONSTRUKTORU (PŘI KONSTRUKCI OBJEKTU JE ZNÁT MUSÍME)

REFLEXE – JAK OBEJÍT ZAPOUZDŘENÍ

```
PRIVATE STATIC VOID SET(OBJECT OBJECT, STRING VAR, OBJECT VALUE)
    THROWS NOSUCHFIELDException, ILLEGALArgumentException, ILLEGALAccessException {

    FIELD F = OBJECT.GETCLASS().GETDECLAREDField(VAR);
    F.SETACCESSIBLE(TRUE);
    F.SET(OBJECT, VALUE);

    // TODO: MŮŽEME ODCHYTÁVAT VÝJIMKY A VYHAZOVAT VLASTNÍ VÝJIMKU PŘÍPADNĚ BĚHOVOU
}
```

REFLEXE – RYCHLOST ☹️

PŘÍMÝ PŘÍSTUP: CELKOVÝ ČAS = 5 MS RELATIVNÍ ČAS = 100.00 %

SETTER: CELKOVÝ ČAS = 5 MS RELATIVNÍ ČAS = 100.00 %

REFLEXE: CELKOVÝ ČAS = 151 MS RELATIVNÍ ČAS = 3020.00 %

REFLEXE PŘIPRAVENÁ: CELKOVÝ ČAS = 16 MS RELATIVNÍ ČAS = 320.00 %

ABSOLUTNÍ ČASY JSOU PRO MILION OPAKOVÁNÍ

POKUD NAJDEME FILED JEN POPRVÉ A POTÉ HO JEN VYUŽÍVÁME.

