Programozási nyelvek és paradigmák

Egyenlőségvizsgálat és objektummásolás

Kozsik Tamás (2020)

Egyenlőségvizsgálat Javában

```
package java.lang;
public class Object {
    ...
    public boolean equals( Object that ){
        // alapértelmezett implementáció: object identity
        return this == that;
    }
}
```

Invariáns paramétertípus!

```
public class Point {
    private int x, y;
    ...
    @Override public boolean equals( Object that ){
        if( that == null ) return false;
        if( that == this ) return true;
        if( that.getClass().equals(this.getClass()) ){
            Point point = (Point)that;
            return x == point.x && y == point.y;
        } else return false;
    }

    @Override public int hashCode(){
        return x + y;     // lehetne jobb is...
    }
}
```

Ilyet nem csinálunk!

```
public class Point {
    private int x, y;
    ...
    /* @Override */ public boolean equals( Point that ){
        return this.x == that.x && this.y == that.y;
    }
}
```

Eiffelben bináris művelettel

```
class ANY
...
feature
    frozen twin: attached like Current
        ...

    is_equal( other: attached like Current ): BOOLEAN
        ...
end
```

Egyenlőségvizsgálat Eiffelben

```
class ANY
...
feature
    is_equal( other: attached like Current ): BOOLEAN
        ...

    frozen equal( a: detachable ANY; b: like a ): BOOLEAN
        do
            if a = Void then
                Result := b = Void
            else
                Result := b /= Void and then a.is_equal(b)
            end
        end
...
end
```

Használat

```
class FOO
feature
    dummy( a, b: FRACTION )
        do
            if a = b then ... end
            if a ~ b then ... end
            if equal(a,b) then ... end
            if a.is_equal(b) then ... end
        end
```

Egyenlőségfogalom versus altípusosság

- Egyenlőség: ekvivalencia-reláció (RST)
- Altípusosság – LSP: részbenrendezés (RAT)

A kettő nem illeszkedik jól egymáshoz

- Polimorfizmus? Különböző típusú objektumok egyenlősége?

```
a.is_equal(b)
```

Egyenlőségfogalom Javában: `java.lang.Object.equals(java.lang.Object)`

- Reflexive: for any non-null reference value `x`, `x.equals(x)` should return true.
- Symmetric: for any non-null reference values `x` and `y`, `x.equals(y)` should return true if and only if `y.equals(x)` returns true.
- Transitive: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns true and `y.equals(z)` returns true, then `x.equals(z)` should return true.
- Consistent: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return false.

Az `equals` működése

`a.equals(b)`

- Ha `a == null`, váltsunk ki `NullPointerException` kivételt
- Egyébként `a.getClass()` $\rightarrow$ dinamikus típus
- Válasszuk ki a dinamikus típusnak megfelelő implementációt

Az összes törzs együttesen adja az implementációt!

Kitérő

OOP alapelv: használjunk elágazás helyett felüldefiniálást!

- Választás elágazással: fix számú lehetőség
- Választás felüldefiniálással: bővíthető

Monomorf vagy polimorf implementáció?

- `instanceof`: `ArrayList` és `LinkedList`
- `getClass().equals`: `Time` és `ExactTime`

Monomorf implementáció (1)

```
public class Time {

    private int hour, minute;

    @Override public boolean equals( Object that ){
        if( that == null ) return false;
        if( that == this ) return true;
        if( this.getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour &&
                   minute == t.minute;
        } else return false;
    }

    @Override public int hashCode(){
        return 60 * hour + minute;
    }

}
```

Monomort implementáció (2)

```
public class ExactTime extends Time {

    private int second;

    @Override public boolean equals( Object that ){
        if( that == null ) return false;
        if( that == this ) return true;
        if( this.getClass().equals(that.getClass()) ){
            ExactTime t = (ExactTime)that;
            return hour == t.hour &&
                   minute == t.minute && second == t.second;
        } else return false;
    }

    @Override public int hashCode(){
        return 60 * super.hashCode() + second;
    }

}
```

Polimorf implementáció

```
package java.util;
public class AbstractList<T> implements List<T> {

    ...

    @Override public boolean equals( Object that ){
        if( that instanceof List ){
            // sekély összehasonlítás
            ...
        } else return false;
    }

    @Override public int hashCode(){
        ...
    }

}
```

Polimorf implementáció

```
package java.util;
public class AbstractList<T> implements List<T> {

    ...

    @Override public boolean equals( Object that ){
        if( that instanceof List ){
            // sekély összehasonlítás
            ...
        } else return false;
    }

    @Override public int hashCode(){
        ...
    }

}
```

```
(new LinkedList<String>()).equals(new ArrayList<String>())
```

Altípusosság és egyenlőségvizsgálat

Van, hogy összhangba hozható

- öröklődésre tervezés
- polimorf `equals`

Van, hogy nem hozható összhangba

- öröklődés kiváltása kompozícióval
- privát öröklődés
- monomorf maradhat az `equals`

Öröklődés kiváltása kompozícióval (Java)

```
public class ExactTime {

    private Time time = new Time();
    private int second;

    @Override public boolean equals( Object that ){
        if( that == null ) return false;
        if( that == this ) return true;
        if( this.getClass().equals(that.getClass()) ){
            ExactTime et = (ExactTime)that;
            return time.equals(et.time) &&
                   second == et.second;
        } else return false;
    }

    @Override public int hashCode(){
        return 60 * time.hashCode() + second;
    }

}
```

Eiffel egyenlőségvizsgálata

- monomorf
- statikus típusrendszer
  - bináris művelet
  - kovariáns paraméter
  - `like Current`
- statikus típusbiztonság?

Egyenlőségvizsgálat referenciatípusokon

- Azonosság (object identity)
  - Referenciák egyenlősége

    ```
    a = b
    ```
- Tartalmi egyenlőség (object equality)
  - Sekély (shallow)
  - Egyedi (custom)

    ```
    a ~ b
    ```
  - Mély (deep)

Egyenlőségvizsgálat kifejtett (expanded) típusokon

- Tartalmi egyenlőség (object equality)
  - Sekély (shallow)
  - Egyedi (custom)

    `a = b`

    `a ~ b`
  - Mély (deep)

Sekély egyenlőségvizsgálat

Két objektum mezői páronként az =-vel összehasonlítva egyenlők

```
standard_equal(a,b)
a.standard_is_equal(b)
```

standard_is_equal (sekély egyenlőségvizsgálat)

```
class ANY
...
feature
    frozen standard_is_equal (other: like Current): BOOLEAN
        -- `other' attached to an object of the same type as
        -- current object, and field-by-field identical to it?
        require
            other_not_void: other /= Void
        external
            "built_in"
        ensure
            same_type: Result implies same_type(other)
            symmetric: Result implies
                                other.standard_is_equal(Current)
        end
...
invariant
    reflexive_equality: standard_is_equal(Current)
    ...
end
```

standard_equal (sekély egyenlőségvizsgálat)

```
class ANY
...
feature
    frozen standard_equal(a:detachable ANY; b:like a):BOOLEAN
        -- Are `a' and `b' either both void or attached to
        -- field-by-field identical objects of the same type?
        -- Always uses default object comparison criterion.
        ensure
            definition:
                Result = (a = Void and b = Void) or else
                         (  (a /= Void and b /= Void)
                            and then a.standard_is_equal(b)
                         )
        end
...
end
```

Példa

```
class POINT feature x, y: INTEGER end
```

```
local
    a, b: INTEGER                          -- expanded!
    p, q: attached POINT
do
    a := 42
    b := 42
    create p    -- (0,0)
    create q    -- (0,0)
    ... a = b ...                          -- True
    ... standard_equal(a,b) ...            -- True
    ... p = q ...                          -- False
    ... standard_equal(p,q) ...            -- True
```

Mély egyenlőségvizsgálat

Két objektum mezői páronként *mélyen* összehasonlítva egyenlők (rekurzív definíció!)

```
deep_equal(a,b)
a.is_deep_equal(b)
```

is_deep_equal (mély egyenlőségvizsgálat)

```
class ANY
...
feature
    frozen is_deep_equal (other: like Current): BOOLEAN
        -- Are `Current' and `other' attached to isomorphic
        -- object structures?
        require
            other_not_void: other /= Void
        external
            "built_in"
        ensure
            shallow_implies_deep:
                standard_is_equal (other) implies Result
            same_type:
                Result implies same_type (other)
            symmetric:
                Result implies other.is_deep_equal(Current)
        end
end
```

deep_equal (mély egyenlőségvizsgálat)

```
class ANY
...
feature
    frozen deep_equal (a: detachable ANY; b: like a): BOOLEAN
        -- Are `a' and `b' either both void
        -- or attached to isomorphic object structures?
        ensure
            shallow_implies_deep:
                standard_equal (a, b) implies Result
            both_or_none_void:
                (a = Void) implies (Result = (b = Void))
            same_type:
                (Result and (a /= Void)) implies
                    (b /= Void and then a.same_type (b))
            symmetric:
                Result implies deep_equal (b, a)
        end
end
```

Példa

```
class INTEGER_LIST
create set
feature
    item: INTEGER
    maybe_next: detachable like Current
    set( value: INTEGER; next: detachable like Current )
        do item := value ; maybe_next := next end
end
```

```
    p, q: attached INTEGER_LIST
do
    create p.set( 1, create {INTEGER_LIST}.set(0,Void) )
    create q.set( 1, create {INTEGER_LIST}.set(0,Void) )
    ... p = q ...                          -- False
    ... standard_equal(p,q) ...            -- False
    ... deep_equal(p,q) ...                -- True
```

Egyedi egyenlőségvizsgálat

Ahogy a programozó definiálja az adott osztályra vonatkozóan

```
a ~ b
equal(a,b)
a.is_equal(b)
```

is_equal (felüldefiniálható egyenlőségvizsgálat)

```
class ANY
...
feature
    is_equal (other: like Current): BOOLEAN
            -- Is `other' attached to an object considered
            -- equal to current object?
        require
            other_not_void: other /= Void
        external
            "built_in"    -- same as standard_is_equal!
        ensure
            symmetric: Result implies other ~ Current
            consistent: standard_is_equal(other) implies Result
        end
end
```

Java és Eiffel különbségek

- *invariáns* versus *kovariáns* paramétertípus
- üres referencia paraméterként *megengedett* versus *nem megengedett*
- alapértelmezett implementáció: *azonosság* versus *sekély egyenlőség*
- primitív típusokon *nem értelmezett* versus kifejtett típusokon *értelmezett*
- hasheléssel *egybekötött* versus *nem egybekötött*

equal és ~ rögzített megvalósítása

```
class ANY
...
feature
    frozen equal (a: detachable ANY; b: like a): BOOLEAN
            -- Are `a' and `b' either both void or attached
            -- to objects considered equal?
        do
            if a = Void then
                Result := b = Void
            else
                Result := b /= Void and then a.is_equal(b)
            end
        ensure
            definition:
                Result = (a = Void and b = Void) or else
                         ( (a /= Void and b /= Void)
                           and then a.is_equal (b)    )
        end
end
```

Példa

```
class SEGMENT
inherit ANY redefine is_equal end
create make
feature
    a, b: attached POINT
    display: detachable CANVAS
    log: detachable LOGGER      -- ignored in equality

    is_equal( other: like Current ) : BOOLEAN
        do
            Result :=
                a ~ other.a and then        -- recurse
                b ~ other.b and then        -- recurse
                display = other.display     -- shallow-like
        end
...
end
```

Összefoglalva

- Referenciatípusnál:

$$\texttt{=} \subset \texttt{standard\_equal} \subseteq \texttt{equal} \equiv \texttt{\~{}}$$

$$\texttt{=} \subset \texttt{standard\_equal} \subseteq \texttt{deep\_equal}$$

- Kifejtett típusnál:

$$\texttt{standard\_equal} \subseteq \texttt{=} \equiv \texttt{equal} \equiv \texttt{\~{}}$$

$$\texttt{standard\_equal} \subseteq \texttt{deep\_equal}$$

Egyenlőségvizsgálat és másolás

is_equal(b)	frozen standard_is_equal(b)	frozen is_deep_equal(b)
frozen equal(a,b)	frozen standard_equal(a,b)	frozen deep_equal(a,b)
copy(b)	frozen standard_copy(b)	frozen deep_copy(b)
frozen twin	frozen standard_twin	frozen deep_twin
frozen clone(b)	frozen standard_clone(b)	frozen deep_clone(b)

```
a.copy(b)          -- a-ba másoljuk b-t
a := b.twin        -- másolat b-ről
a := clone(b)      -- másolat b-ről (obsolate)
```

Egyenlőségvizsgálat és másolás: szignatúrák

is_equal(other: like Current): BOOLEAN	frozen standard_is_equal(other: like Current): BOOLEAN	frozen is_deep_equal(other: like Current): BOOLEAN
frozen equal(some: ANY; other: like some): BOOLEAN	frozen standard_equal(some: ANY; other: like some): BOOLEAN	frozen deep_equal(some: ANY; other: like some): BOOLEAN
copy (other: like Current)	frozen standard_copy (other: like Current)	frozen deep_copy (other: like Current)
frozen twin: like Current	frozen standard_twin: like Current	frozen deep_twin: like Current
frozen clone (other: ANY): like other	*frozen standard_clone (other: ANY): like other*	*frozen deep_clone (other: ANY): like other*

Lásd itt.

Sekély, mély és egyedi másolat

- `standard_copy(b)`: `b` mezőit értékül adjuk `Current` mezőinek
  - `ensure standard_equal(Current,b)`
- `deep_copy(b)`: `b` mezőit mélyen másoljuk `Current` mezőibe
  - `ensure deep_equal(Current,b)`
- `copy(b)`: amit implementálunk
  - alapértelmezés `ANY`-ben: `standard_copy`
  - `ensure equal(Current,b)`