

18 Object relations

Overview

- Object identity and equality
- Algebraic properties of equality
- Hash code methods
- Comparison and orderings

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 2

Well behaved classes

- Objects that are handled by the JVM or many standard classes should have
 - *No-arg constructor*
 - *String representation*
 - *Serialization (for streaming)*
 - *Cloning (deep copy)*
 - **Equality and hashCode methods** ←

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 3

Identity ≠ equality

```
public class Person
{
    private long pnr;
    private String name;

    public Person(long pnr, String name)
    {
        this.pnr = pnr; this.name = name;
    } ...
}

ArrayList<Person> l = new ArrayList<Person>();
Person p1 = new Person(123, "Bob");
Person p2 = new Person(123, "Bob");
l.add(p1);
l.contains(p1);
l.contains(p2);
```

true or false? **true**
 true or false? **false!**

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 4

Identity ≠ equality (2)

- The Java standard says:


```
public boolean contains(Object o)
    - Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that
      (o==null ? e==null : o.equals(e))
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 5

Identity ≠ equality (2)

- The operator == is inadequate for object equality tests because it means **object identity** (address equality).
- If instances of a user defined class are going to be stored in collections the class must override the method


```
public boolean equals(Object other)
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 6

Naïve implementation of equals

```
public boolean equals(Object o)
{
    Person other = (Person)o;
    return pnr == other.pnr &&
        name.equals(other.name);
}
```

2. **ClassCastException**

1. **NullPointerException**

1. **NullPointerException**

2. **ClassCastException**

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 7

Properties of the equals method

- When the equals method is overridden, it must satisfy the following properties for any objects x , y and z

Reflexivity: $x.equals(x)$ must return true.
Symmetry: $x.equals(y)$ returns true iff $y.equals(x)$ returns true.
Transitivity: if $x.equals(y)$ returns true and $y.equals(z)$ returns true, then $x.equals(z)$ must return true.
Null: $x.equals(null)$ must return false.
hashCode: if $x.equals(y)$ returns true then $x.hashCode() == y.hashCode()$.
Consistency: the value of $x.equals(y)$ must not change if x and y do not change.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 8

The instanceof operator

- If obj denotes some object and C is a class then
 $obj instanceof C$
is **true** iff the type of obj is C , or some subclass of C (transitivity), and **false** otherwise. In particular,
 $null instanceof C$
is always **false**.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 9

Correct (?) implementation of equals

```
public boolean equals(Object o)
{
    if ( o instanceof Person ) {
        Person other = (Person)o;
        return pnr == other.pnr &&
            name.equals(other.name);
    }
    return false;
}
```

Person p1 = new Person(123, "Bob");
Person p2 = new Person(123, "Bob");
Person p3 = new Person(123, "Bo");
p1.equals(p1) **true**
p1.equals(p2) **true**
p1.equals(p3) **false**

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 10

Can objects of different types be equal?

- Two lists are equal if they contain equal elements in the same order.
- Fine! So l1 and l2 are equal?*

```
List<Integer> l1 = new ArrayList<Integer>();
l1.add(1);
l1.add(2);
List<Integer> l2 = new LinkedList<Integer>();
l2.add(1);
l2.add(2);
l1.equals(l2);
```

true or false?

true!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 11

Can objects of different types be equal? (2)

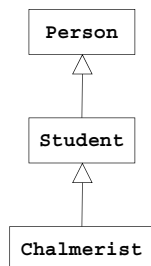
- List equality is a general property of lists
- regardless of their implementations.
- Two lists are *equal* if they contain the same number of elements, and the elements are pairwise *equal*.*
- Thus the lists in the previous slide are equal!*

Two list elements e1 and e2 are equal if (e1==null ? e2==null : e1.equals(e2))

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 12

Equality and inheritance



- Can a **Person** be compared to a **Student**?
- Can a **Student** be compared to a **Chalmerist**?
- ... *would that make sense?*
- *What should equality mean in such cases?*

Explore the [ladok_1](#) project!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 13

Equality and inheritance (2)

```

public class Student extends Person
{
    private int credits;

    public boolean equals(Object other) {
        if ( other instanceof Student ) {
            return super.equals(other) &&
                credits == ((Student)other).credits;
        } else
            return false;
    }
    ... constructor and methods omitted
}
  
```

Delegate equality test for the base class part to the base class's equals

We may want to extend the equality test in the base class to include the additional instance variables of the subclass.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 14

Equality and inheritance (3)

```

public class Chalmerist extends Student
{
    private char program;

    public boolean equals(Object other) {
        if ( other instanceof Chalmerist ) {
            return super.equals(other) &&
                program == ((Chalmerist)other).program;
        } else
            return false;
    }
    ... constructor and methods omitted
}
  
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 15

Equality and inheritance (4)

```

Person p = new Person(870218,"Liza");
Student s = new Student(870218,"Liza",160);
Chalmerist c1 = new Chalmerist(870218,"Liza",160,'D');
Chalmerist c2 = new Chalmerist(870218,"Liza",160,'F');

p.equals(s); // true
p.equals(c1); // true
s.equals(c1); // true
c1.equals(c2); // false
s.equals(p); // false! equals is not symmetric - why?
  
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 16

Equality and inheritance (5)

- *What is the problem about?*

```

class Student extends Person
Person p;
Student s;
  
```

A **Student** **is** a **Person**
but a **Person** **is not** a **Student**

thus

`s instanceof Person` **is true**

but

`p instanceof Student` **is false!**

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 17

Why is symmetry important?

- Ex. Storing elements of different subtypes in data structures may give different search results depending on the insertion order.

```

ArrayList<Person> ladok = new ArrayList<Person>();
Person p = new Person(870218,"Liza");
Student s = new Student(870218,"Liza",160);
  
```

// consider one of the following scenarios

```

ladok.add(p);
ladok.contains(s);
  
```

```

ladok.add(s);
ladok.contains(p);
  
```

One will return true, and the other false because contains will use our overridden equals methods!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 18

Final classes and final methods

- *Final classes* may not have subclasses.

```
public final class Person
public class Student extends Person
```

forbidden

- *Final methods* may not be overridden in subclasses.

```
public final boolean equals(Object other)
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 19

Non-overridable equals method

- Use this scheme if the `equals` method in class C should *not* be overridable in subclasses.

```
public final boolean equals(Object other) {
    if ( this == other )
        return true; // reflexivity
    if ( other instanceof C ) {
        C tmp = (C)other;
        // Compare the class C instance variables in this
        // object with their counterparts in the other object,
        // and return the result of the comparison.
    }
    return false;
}
```

So, `equals` will be *inherited*
- but it may *not* be *redefined*.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 20

Checking the dynamic type

- Instances of the standard class `Class` contain type information about classes.
- The method `Object.getClass()` returns the run-time (dynamic) type of an object as a `Class` object.

```
Person p = new Person(...);
Student s = new Student(...);
p.getClass() == s.getClass() // false
p = s;
p.getClass() == s.getClass() // true
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 21

Overridable equals method

- Use this scheme if the `equals` method in class C may be redefined in subclasses.

```
public boolean equals(Object other) {
    if ( this == other )
        return true; // reflexivity
    if ( other != null &&
        getClass() == other.getClass() )
    {
        C tmp = (C)other;
        // Compare the class C instance variables in this
        // object with their counterparts in the other object,
        // and return the result of the comparison.
    }
    return false;
}
```

So, `equals` may be redefined
- but only objects of the same
dynamic type can be equal.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 22

The Person hierarchy refactored

- Are all instance variables relevant in an equality test?

```
public class Person
{
    private String pnr;
    private String name;
    private String address;
    ...
    public final boolean equals(Object other) {
        if ( this == other )
            return true;
        if ( other instanceof Person )
        {
            Person tmp = (Person)other;
            return (pnr == null ?
                tmp.pnr == null : pnr.equals(tmp.pnr));
        }
        else
            return false;
    }
    // constructor and methods omitted
}
```

search key

Explore the [ladok_2](#) project!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 23

The Person hierarchy refactored

```
public class Student extends Person
{
    private int credits;
    ...
}

public class Chalmerist extends Student
{
    private char program;
    ...
}
```

no equals in this class

no equals in this class

- After all, a "Chalmerist" can change name, move to a new place, grade, change program, ...
- and still be the same (equal) person!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 24

Hash code methods

- *What is hashing?*
- *What is a hash function?*
- *What properties should a hashCode method have?*
- *How do we define a hashCode method?*

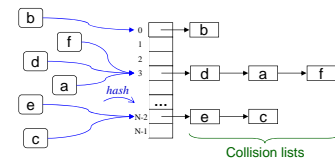
Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 25

What is hashing?

- A *hash function* maps a data object to a non-negative integer - a hash value.
- Hash functions are used in hash tables such as the `HashSet` and `HashMap` classes in the Java collections framework.

Hash function	
Object	Value
a	3
b	0
c	N-2
d	3
e	N-2
f	3



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 26

Hash functions in Java

- The prototype for hash functions in Java is defined in `class Object`
- ```
public int hashCode()
```
- The method call `x.hashCode()` returns a hash value for the object `x`.
  - Standard classes redefine this method.
  - User defined classes which redefine the equals method must redefine hashCode!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 27

## Hash functions in Java (2)

- The value returned by `hashCode()` in class `Object` is typically based on the address of the calling instance.
- Distinct objects are likely to have distinct hash codes.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 28

## The hashCode contract

For any objects `x` and `y`

1. If `x.equals(y)` returns `true` then `x.hashCode() == y.hashCode()` must be `true`.
2. If `x.equals(y)` returns `false` then `x.hashCode() != y.hashCode()` is not a requirement, but it gives better performance in hash tables.
3. During the same program execution `x.hashCode()` must consistently return the same result, provided no information used in the `equals` method is changed.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 29

## Forgot to override hashCode?

- Suppose we enter an instance of the last version of the `Person` class into a `HashSet`

Explore the *hashCode* project!

```
HashSet<Person> s = new HashSet<Person>();
Person p = new Person("871138", "NN", "Gothenburg");
s.add(p);
...
s.contains(new Person("871138", "NN", "Gothenburg"))
```

True or false?

False!  
`x.equals(y)` but  
`x.hashCode() != y.hashCode()`

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 30

## Rules for `hashCode` overriding

1. In an overriding of `hashCode()` in class `C` the value shall depend on the *significant instance variables* that are used in the `C.equals()` method.
2. The hash value must *not* depend on any variables that are *not* used in `C.equals()`.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 31

## Ex. `Person.hashCode()`

```
public class Person
{
 private String pnr;
 private String name;
 private String address;
 ...
 public final boolean equals(Object o) { ... }

 public int hashCode() {
 return pnr.hashCode();
 }
 // constructor and other methods omitted
}
```

Compute the hash value from the search key - only

Delegate to `String.hashCode()`

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 32

## How to compute `hashCodes`

1. For instance variables `x` of elementary data type the hash code depends on the value of `x`.
2. For instance variables `r` of class `C`, the hash code depends on the value returned by `r.hashCode()`.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 33

## How to compute `hashCodes` (2)

```
public int hashCode() {
 int code = 123; // Arbitrary value
 for each significant instance variable x
 if x is of primitive data type
 code = 37*code + v(x)
 else
 code = 37*code + x.hashCode()
 return code;
}
```

For computation rules for `v(x)`, refer to [1].[1] Joshua Bloch, *Effective Java*

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 34

## Ex. a simple `hashCode` method

```
public class C {
 private char c;
 private boolean b;
 private String s;
 private float foo;

 public boolean equals(Object other) { ... }

 public int hashCode() {
 int code = 123;
 code = 37*code + (int)c;
 code = 37*code + (b ? 1 : 0);
 code = 37*code + s.hashCode();
 return code;
 }
}
```

Suppose the `equals` method depends on the significant variables `c`, `b` and `s`

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 35

## Ordering objects

```
public interface Comparable<T> {
 int compareTo(T other);
}
```

Instances of classes that implement this interface are *comparable to each other*.

```
public interface Comparator<T> {
 int compare(T lhs, T rhs);
 boolean equals(Object other);
}
```

Instances of classes that implement this interface can *compare objects of type T*.

|        |
|--------|
| return |
| < -1   |
| == 0   |
| > 1    |

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 36

## java.lang.Comparable

```
public class Person
implements Comparable<Person>
```

```
{
 ...
 public int compareTo(Person other) {
 return pnr.compareTo(other.pnr);
 }
}
```

Example. Order Person objects by person id in lexicographical order

Delegate the comparison to the String class (for example).

// constructor and other methods omitted

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 37

## Properties of compareTo

- Overridings of the compareTo method should satisfy the following properties for any objects x, y and z

$$\text{sgn}(a) = \begin{cases} -1 & \text{if } a < 0 \\ 0 & \text{if } a = 0 \\ 1 & \text{if } a > 0 \end{cases}$$

*Opposite order:*

$\text{sgn}(x.\text{compareTo}(y)) == -\text{sgn}(y.\text{compareTo}(x))$

*Transitivity 1:*

if  $x.\text{compareTo}(y) > 0$  and  $y.\text{compareTo}(z) > 0$  then  $x.\text{compareTo}(z) > 0$

*Transitivity 2:*

if  $x.\text{compareTo}(y) == 0$  then for all z,

$\text{sgn}(x.\text{compareTo}(z)) == \text{sgn}(y.\text{compareTo}(z))$

*Consistency with equals:*

if  $x.\text{equals}(y)$  then  $x.\text{compareTo}(y) == 0$

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 38

## Properties of compareTo

### Comments

- $x.\text{compareTo}(\text{null})$  throws `NullPointerException`
  - even though  $x.\text{equals}(\text{null})$  returns false.
- Consistency with equals is recommended but not required.
- Inheritance can make things complicated, just as for equals. *Beware!*

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 39

## java.util.Comparator

```
public class PersonComparator
implements Comparator<Person>
{
 public int compare(Person lhs, Person rhs) {
 return (lhs.getPnr()).compareTo(rhs.getPnr());
 }

 public boolean equals(Object other)
 // it is safe to not override this method
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 40

## java.util.Comparator

```
public someClass {
 ...
 public someMethod(..., Comparator<some type> comp) {
 ...
 if (comp.compare(x,y) == ...)
 ...
 }
 // constructor and other methods omitted
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 41

## Using the Person comparator

```
public class SortedLADOK
{
 private Set<Person> database =
 new TreeSet<Person>(new PersonComparator());

 public void print()
 {
 for (Person p : database)
 System.out.println(p);
 }
 // constructor and other methods omitted
}
```

This loop will print the persons in the database in increasing order according to the given comparator object.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 18 42