**Programming languages (BSc, 18) Java 10. seminar**

**Liskov Substitution Principle**

public static boolean isSameAuthor(Book book1, Book book2)

{

return book1.getAuthor().equals(book2.getAuthor());

}

public static void main(String[] args)

{

Book book1 = new Book();

Book book2 = new Book("author", "Title", 100);

PrintedBook pbook2 = new PrintedBook("author", "Printed: Title", 100, CoverType.Softcover);

System.out.println(isSameAuthor(book1, book2));

System.out.println(isSameAuthor(book2, pbook2));

}

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it. (Stackoverflow)

**static vs. dynamic type of a variable**

Book book;

book = new Book();

// the static type of book is Book, the dynamic type of book is Book

System.out.println(book.toString()); // Book toString()

book = new PrintedBook("author", "Printed: Title", 100, CoverType.Softcover);

// the static type of book is Book, the dynamic type of book is PrintedBook

System.out.println(book.toString()); // PrintedBook toString()

//PrintedBook pbook3 = (PrintedBook)book1;

//System.out.println(pbook3.toString());

**equals() and hashCode()**

class Vector

{

double[] coords;

public Vector(double x1, double x2)

{

this.coords = new double[2];

this.coords[0] = x1;

this.coords[1] = x2;

}

public String toString()

{

return "(" + this.coords[0] + "," + this.coords[1] + ")";

}

}

class Main

{

public static void main(String[] args)

{

System.out.println(new Vector(2, 3).equals(new Vector(2, 3)));

System.out.println(new Vector(2, 3).equals(new Vector(2, 2)));

HashSet<Vector> exampleSet = new HashSet<Vector>();

exampleSet.add(new Vector(0, 0));

exampleSet.add(new Vector(3, -7));

exampleSet.add(new Vector(3, -7));

System.out.println( "size of HashSet: " + exampleSet.size());

System.out.println( "items of HashSet: " + exampleSet);

}

}

Our expectations of the equals() and hashCode() methods:

* equals() is an equivalence relation (reflexive, symmetric, transitive)
* if a != null, then a.equals(null) is false
  + however null.equals(a) should throw a NullPointerException
* equals() should be consistent with hashcode()
  + equal objects should have equal hashcodes (meaning if a.equals(b) then a.hashCode() == b.hashCode())
  + different objects may still have equal hashcodes, though this is discouraged (meaning it is ideal, but not enforced that if a.equals(b)==false then a.hashCode() != b.hashCode())

**Task1**

Take the Vector class from the previous example and write it's equals() and hashCode() methods. Create unit tests to test them according to our expectations of them.

import java.util.Objects;

class Vector

{

double[] coords;

public Vector(double x1, double x2)

{

this.coords = new double[2];

this.coords[0] = x1;

this.coords[1] = x2;

}

@Override

public String toString()

{

return "(" + this.coords[0] + "," + this.coords[1] + ")";

}

@Override

public boolean equals(Object that)

{

if (that == this) return true;

if (that == null) return false;

if (that instanceof Vector)

{

Vector thatVector = (Vector)that;

return coords[0] == thatVector.coords[0] && coords[1] == thatVector.coords[1];

}

else return false;

}

@Override

public int hashCode()

{

//return (int)(11\*coords[0] + 19\*coords[1]);

return Objects.hash(coords[0], coords[1]);

}

}

import static org.junit.Assert.assertTrue;

import static org.junit.Assert.assertFalse;

import org.junit.Test;

public class VectorTest

{

@Test

public void equalsIsReflexiv()

{

Vector v1 = new Vector(1, 3);

assertTrue(v1.equals(v1));

}

@Test

public void equalsIsSymmetric1()

{

Vector v1 = new Vector(1, 3);

Vector v2 = new Vector(1, 3);

if (v1.equals(v2))

{

assertTrue(v2.equals(v1));

}

if (v2.equals(v1))

{

assertTrue(v1.equals(v2));

}

}

@Test

public void equalsIsSymmetric2()

{

Vector v1 = new Vector(1, 3);

Vector v2 = new Vector(1, 5);

if (v1.equals(v2))

{

assertTrue(v2.equals(v1));

}

if (v2.equals(v1))

{

assertTrue(v1.equals(v2));

}

}

@Test

public void equalsIsTransitive1()

{

Vector v1 = new Vector(1, 3);

Vector v2 = new Vector(1, 3);

Vector v3 = new Vector(1, 3);

if (v1.equals(v2) && v2.equals(v3))

{

assertTrue(v1.equals(v3));

}

}

@Test

public void equalsIsTransitive2()

{

Vector v1 = new Vector(1, 3);

Vector v2 = new Vector(1, 5);

Vector v3 = new Vector(1, 3);

if (v1.equals(v2) && v2.equals(v3))

{

assertTrue(v1.equals(v3));

}

}

@Test

public void equalsWithNullParam()

{

Vector v = new Vector(10, 20);

assertFalse(v.equals(null));

}

@Test(expected = NullPointerException.class)

public void equalsOnNullref()

{

Vector v1 = new Vector(10, 20);

Vector v2 = null;

v2.equals(v1);

}

@Test

public void equalObjHashCodeIsSame()

{

Vector v1 = new Vector(1, 3);

Vector v2 = new Vector(1, 3);

assertTrue(v1.equals(v2));

assertTrue(v1.hashCode() == v2.hashCode());

}

}

import java.util.HashSet;

class Main

{

public static void main(String[] args)

{

System.out.println(new Vector(2, 3).equals(new Vector(2, 3)));

System.out.println(new Vector(2, 3).equals(new Vector(2, 2)));

HashSet<Vector> exampleSet = new HashSet<Vector>();

exampleSet.add(new Vector(0, 0));

exampleSet.add(new Vector(3, -7));

exampleSet.add(new Vector(3, -7));

System.out.println( "size of HashSet: " + exampleSet.size());

System.out.println( "items of HashSet: " + exampleSet);

}

}

### Task2

Modify the equals() method of the Person class of task 7 of lesson 3 so that it will suit our expectations of it and write a hashCode() method accordingly.

package person;

public enum Gender

{

MALE, FEMALE

}

package person;

import java.util.Objects;

public class Person

{

private String firstname, lastname;

private String occup;

private Gender gen;

private int birthYear;

public Person(String firstname, String lastname, String occup, Gender gen, int birthYear)

{

this.firstname = firstname;

this.lastname = lastname;

this.occup = occup;

this.gen = gen;

this.birthYear = birthYear;

}

public String toString()

{

return "(" + firstname + "," + lastname + "," + occup + "," + gen + "," + birthYear + ")";

}

// overload

// dont do this

/\*

public boolean equals(Person that)

{

return this.firstname.equals(that.firstname) && this.lastname.equals(that.lastname) && this.occup.equals(that.occup) && this.gen == that.gen && this.birthYear == that.birthYear;

}\*/

@Override

public boolean equals(Object that)

{

if (that == this) return true;

if (that == null) return false;

if (that instanceof Person)

{

Person thatPerson = (Person)that;

return firstname.equals(thatPerson.firstname) && lastname.equals(thatPerson.lastname) && occup.equals(thatPerson.occup) && gen == thatPerson.gen && birthYear == thatPerson.birthYear;

}

else return false;

}

@Override

public int hashCode()

{

return Objects.hash(firstname, lastname, occup, gen, birthYear);

}

}

package main;

import java.util.HashSet;

import person.\*;

public class Main

{

public static void main(String[] args)

{

HashSet<Person> exampleSet = new HashSet<Person>();

exampleSet.add(new Person("AAA", "BBB", "oc", Gender.MALE, 2000));

exampleSet.add(new Person("AAA2", "BBB2", "oc2", Gender.FEMALE, 1999));

exampleSet.add(new Person("AAA", "BBB", "oc", Gender.MALE, 2000));

exampleSet.add(new Person("AAA", "BBB", "oc", Gender.MALE, 2000));

exampleSet.add(new Person("AAA", "BBB", "oc", Gender.MALE, 2000));

exampleSet.add(new Person("AAA", "BBB", "oc", Gender.MALE, 2000));

System.out.println("size of HashSet: " + exampleSet.size());

System.out.println("items of HashSet: " + exampleSet);

Person tmp = new Person("AAA", "BBB", "oc", Gender.MALE, 2000);

//System.out.println(tmp.equals(tmp)); // equals(Person)

System.out.println(tmp.equals(new Object())); // equals(Object)

}

}

### Task3

Create a generic Bag<T> class representing a bag. A bag is a set that may contain duplicates of it's elements.

Add a HashMap<T, Integer> datamember to it which will be initialized by a parameterless constructor. Add an add(T element) method to it. This will check if the key is already present in the bag: if it's not, add it with value 1, otherwise get it's current value and increment it by 1. (The map will track how many instances of each element is in the bag.)

Add a countOf(T element) method to it returning an int that returns how many duplicates of the element is in the bag. If there is no such key in the bag, return 0.

Add a remove(T element) method for removing an element. This will decrement the value of the given key by 1 in the bag. If the value drops to 0, remove the corresponding key-value pair from the map so that no superfluous data is stored. If the element was not in the bag throw a NotInBagException exception which is a user defined exception class. The NotInBagException exception inherits from Exception and it's constructor expecting a string parameter should call the super class' constructor.

Create a main program which will process an input text file containing a word in each line and counts how many times each word appeared in the input file using the Bag<T> class.

input.txt:

hello

world

interface

abstract

abstract

world

world

world

hello

world

X-Files

protected

abstract

abstract

extends

protected

socket

world

hello

socket

extends

```

import java.util.HashMap;

import java.util.Map;

public class Bag<T> {

private final Map<T, Integer> map = new HashMap<>();

public void add(T element) {

int count = 1;

Integer currentCount = map.get(element);

if (currentCount != null) {

count = currentCount + 1;

}

map.put(element, count);

}

public int countOf(T element) {

Integer currentCount = map.get(element);

if (currentCount != null) {

return currentCount;

} else {

return 0;

}

}

public void remove(T element) throws NotInBagException {

Integer currentCount = map.get(element);

if (currentCount == null) {

throw new NotInBagException("Element: " + element + " doesn't present in Bag.");

}

if (currentCount == 1) {

map.remove(element);

} else {

map.put(element, currentCount - 1);

}

}

@Override

public String toString() {

return map.toString();

}

}

import java.util.Scanner;

import java.io.File;

import java.io.FileNotFoundException;

class Main

{

public static void main(String[] args)

{

/\*

Bag<String> bag = new Bag<>();

bag.add("foo");

bag.add("foo");

bag.add("foo");

bag.add("foo2");

bag.add("foo3");

bag.add("idk");

bag.add("idk");

System.out.println(bag);

\*/

File input = new File(args[0]);

Bag<String> statistics = new Bag<>();

try (Scanner sc = new Scanner(input))

{

while (sc.hasNextLine())

{

statistics.add(sc.nextLine());

}

}

catch (FileNotFoundException e)

{

System.out.println("Unable to access file: " + args[0]);

}

System.out.println("Word statistics: " + statistics);

}

}

public class NotInBagException extends Exception

{

public NotInBagException(String msg)

{

super(msg);

}

}

Input.txt

hello

world

interface

abstract

abstract

world

world

world

hello

world

X-Files

protected

abstract

abstract

extends

protected

socket

world

hello

socket

extends