Limbaje de programare. Laborator 2.

1. **Operatorul de atribuire**

La atribuirea intre 2 variabile referinte catre obiecte, se face copierea intre referinte, ambele variabile referend acelasi obiect

//: operators/Assignment.java

class Tank {

int level;

}

public class Assignment {

public static void main(String[] args) {

Tank t1 = new Tank();

Tank t2 = new Tank();

t1.level = 9;

t2.level = 47;

System.out.println("1: t1.level: " + t1.level + ", t2.level: " + t2.level);

t1 = t2;

System.out.println("2: t1.level: " + t1.level + ", t2.level: " + t2.level);

t1.level = 27;

System.out.println("3: t1.level: " + t1.level + ", t2.level: " + t2.level);

}

}

1. **Aliasing**

La trimiterea unui obiect ca si argument intr-o metoda, se intampla Aliasing, adica atat variabila din metoda cat si cea din exterior vor arata catre acelasi obiect

//: operators/PassObject.java

// Passing objects to methods may not be

// what you’re used to.

class Letter {

char c;

}

public class PassObject {

static void f(Letter y) {

y.c = ‘z’;

}

public static void main(String[] args) {

Letter x = new Letter();

x.c = ‘a’;

System.out.println("1: x.c: " + x.c);

f(x);

System.out.println("2: x.c: " + x.c);

}

}

1. **Operatori pe String: + si +==**

public class StringOperators {

public static void main(String[] args) {

int x = 0, y = 1, z = 2;

String s = "x, y, z ";

System.out.println(s + x + y + z);

print(x + " " + s); // Converts x to a String

s += "(summed) = "; // Concatenation operator

System.outprintln(s + (x + y + z));

print("" + x); // Shorthand for Integer.toString()

}

}

1. Instructiuni pentru controlul fluxului programului

Scrieti un program care genereaza 25 de numere random intregi (clasa Random). Pentru fiecare din aceste numere utilizati o instructiune if-then-else pentru a testa daca numarul generat este mai mare, mai mic sau egal cu un al 2-lea numar intreg generat random.

1. Foreach – selecteaza un element dintr-un array

//: control/ForEachFloat.java

import java.util.\*;

public class ForEachFloat {

public static void main(String[] args) {

Random rand = new Random(47);

float f[] = new float[10]; // crearea arrayului

for(int i = 0; i < 10; i++)

f[i] = rand.nextFloat(); // initializarea fiecarui element din array

**for(float x : f)**

System.out.println(x);

}

}

1. Exercitiu

Creati un program care citeste ca si intrare un numar intreg si afiseaza toate numele lui Fibonacci cel mult egale cu numarul intreg citit la intrare. De exemplu, pentru valoarea 5 se afiseaza lista 1, 1, 2, 3, 5

1. Constructori

//: initialization/SimpleConstructor2.java

// Constructors can have arguments.

class Rock2 {

Rock2(int i) {

System.out.print("Rock " + i + " ");

}

}

public class SimpleConstructor2 {

public static void main(String[] args) {

for(int i = 0; i < 8; i++)

new Rock2(i);

}

}

1. Supraincarcarea metodelor – acelasi nume de metoda utilizat la mai multe metode cu argument diferinte

//: initialization/Overloading.java

// Demonstration of both constructor

// and ordinary method overloading.

class Tree {

int height;

Tree() {

System.out.print("Planting a seedling");

height = 0;

}

Tree(int initialHeight) {

height = initialHeight;

System.out.print("Creating new Tree that is " + height + " feet tall");

}

void info() {

System.out.print("Tree is " + height + " feet tall");

}

void info(String s) {

System.out.print(s + ": Tree is " + height + " feet tall");

}

}

public class Overloading {

public static void main(String[] args) {

for(int i = 0; i < 5; i++) {

Tree t = new Tree(i);

t.info();

t.info("overloaded method");

}

// Overloaded constructor:

new Tree();

}

}

1. Utilizarea this

//: initialization/Leaf.java

// Simple use of the "this" keyword.

public class Leaf {

int i = 0;

Leaf increment() {

i++;

return this; // se returneaza obiectul curent

}

void print() {

System.out.println("i = " + i);

}

public static void main(String[] args) {

Leaf x = new Leaf();

x.increment().increment().increment().print();

}

}

1. Utilizarea finalize

In programul de mai jos, se presupune ca o carte (Book) trebuie sa fie in mod necesar “checkin” inainte ca aceasta sa fie distrusă. Observam ca exista o carte pentru care nu se face checkin si la apelul fortat al garbage collectorului, va aparea mesajul de eroare din finalize.

//: initialization/TerminationCondition.java

// Using finalize() to detect an object that

// hasn’t been properly cleaned up.

class Book {

boolean checkedOut = false;

Book(boolean checkOut) {

checkedOut = checkOut;

}

void checkIn() {

checkedOut = false;

}

protected void finalize() {

if(checkedOut)

System.out.println("Error: checked out");

// Normally, you’ll also do this:

// super.finalize(); // Call the base-class version

}

}

public class TerminationCondition {

public static void main(String[] args) {

Book novel = new Book(true);

// Proper cleanup:

novel.checkIn();

// Drop the reference, forget to clean up:

new Book(true);

// Force garbage collection & finalization:

System.gc();

}

}

1. **Ordinea de initializare – ordinea in care datele membre sunt definite in clasa**

//: initialization/OrderOfInitialization.java

// Demonstrates initialization order.

// When the constructor is called to create a

// Window object, you’ll see a message:

class Window {

Window(int marker) { System.out.print("Window(" + marker + ")"); }

}

class House {

Window w1 = new Window(1); // Before constructor

House() {

// Show that we’re in the constructor:

System.out.print("House()");

w3 = new Window(33); // Reinitialize w3

}

Window w2 = new Window(2); // After constructor

void f() { System.out.print("f()"); }

Window w3 = new Window(3); // At end

}

public class OrderOfInitialization {

public static void main(String[] args) {

House h = new House();

h.f(); // Shows that construction is done

}

}

1. **Initializarea statica**

//: initialization/StaticInitialization.java

// Specifying initial values in a class definition.

class Bowl {

Bowl(int marker) {

System.out.print("Bowl(" + marker + ")");

}

void f1(int marker) {

print("f1(" + marker + ")");

}

}

class Table {

static Bowl bowl1 = new Bowl(1);

Table() {

System.out.print("Table()");

bowl2.f1(1);

}

void f2(int marker) {

System.out.print("f2(" + marker + ")");

}

static Bowl bowl2 = new Bowl(2);

}

class Cupboard {

Bowl bowl3 = new Bowl(3);

static Bowl bowl4 = new Bowl(4);

Cupboard() {

System.out.print("Cupboard()");

bowl4.f1(2);

}

void f3(int marker) {

System.out.print("f3(" + marker + ")");

}

static Bowl bowl5 = new Bowl(5);

}

public class StaticInitialization {

public static void main(String[] args) {

System.out.print("Creating new Cupboard() in main");

new Cupboard();

System.out.print("Creating new Cupboard() in main");

new Cupboard();

table.f2(1);

cupboard.f3(1);

}

static Table table = new Table();

static Cupboard cupboard = new Cupboard();

}

1. **Initializarea prin explicit static**

//: initialization/ExplicitStatic.java

// Explicit static initialization with the "static" clause.

class Cup {

Cup(int marker) {

System.out.print("Cup(" + marker + ")");

}

void f(int marker) {

System.out.print("f(" + marker + ")");

}

}

class Cups {

static Cup cup1;

static Cup cup2;

static { // aici e blocul static explicit

cup1 = new Cup(1);

cup2 = new Cup(2);

}

Cups() {

System.out.print("Cups()");

}

}

public class ExplicitStatic {

public static void main(String[] args) {

System.out.print("Inside main()");

Cups.cup1.f(99); // (1)

}

// static Cups cups1 = new Cups(); // (2)

// static Cups cups2 = new Cups(); // (2)

}

1. **Initializare non-statica**

//: initialization/Mugs.java

// Java "Instance Initialization."

class Mug {

Mug(int marker) {

System.out.print("Mug(" + marker + ")");

}

void f(int marker) {

System.out.print("f(" + marker + ")");

}

}

public class Mugs {

Mug mug1;

Mug mug2;

{ // bloc non-static de initializare

mug1 = new Mug(1);

mug2 = new Mug(2);

print("mug1 & mug2 initialized");

}

Mugs() {

System.out.print("Mugs()");

}

Mugs(int i) {

System.out.print("Mugs(int)");

}

public static void main(String[] args) {

print("Inside main()");

new Mugs();

print("new Mugs() completed");

new Mugs(1);

print("new Mugs(1) completed");

}

}

1. **Initializarea sirurilor de obiecte**
2. Creati un sir de obiecte de tip String si initializati fiecare obiect din sir.
3. Tipariti sirul folosind un for.
4. Creati o clasa care are un constructor cu un argument de tip String.
5. In constructor, tipariti valoarea primita ca si argument.
6. Creati un sir de referinte catre obiecte din clasa creata, dar fara a crea obiectele din sir.
7. La executia programului, vedeti daca mesajul din constructor este tiparit.
8. Initializati sirul cu obiecte din clasa creata.