Crearea și distrugerea obiectelor

1. Inițializarea obiectelor standarte
2. Inițializarea obiectelor în constructor
3. Supraîncărcarea metodelor
4. Cuvîntul chiee this
5. Cuvîntul chiee static
6. Distrugerea obiectelor
7. Inițializarea tablourilor

Inițializarea obiectelor standarte primitive

Doar tipurile primitive sunt automat iniţializate la valoarea default dacă acestea sunt definite ca variabile membru ale clasei. Tipurile referinţă nu sunt automat iniţializate. Dacă în cadrul unei clase avem definite unul sau mai multe obiecte acestea vor avea valoarea **null.** „null” este cuvânt cheie în java care desemnează faptul că o variabilă de tip referinţă nu pointează către nici un obiect in memorie.

Dacă în cadrul unei clase este definită o variabilă de tip static atunci aceasta va fi iniţializată o singură dată în momentul în care clasa respectivă este încărcată în cadrul maşinii virtuale. O clasă este încărcată în memorie atunci când este folosită pentru prima data: fie un obiect de tipul respectiv este construit fie un atribut sau o metodă statică a acelei clase este apelat.

Conctructori

Rolul constructorului este de iniţializare a stării unui obiect. Câteva caracteristici importante ale constructorilor din java sunt următoarele:

- Constructorii sunt metode speciale a căror nume este identic cu numele clasei din care fac parte.
- Constructorii nu au nici un tip de return
- În cadrul unei clase pot fi definiţi unul sau mai mulţi constructori care trebuie să difere între ei prin numărul şi tipul parametrilor.
- Dacă în cadrul unei clase programator nu adaugă nici un constructor atunci la compilare automat se va adăuga constructorul *default* care nu are nici un parametru.
- Dacă programatorul adaugă în cadrul clasei cel puţin un constructor atunci construcorul *default* nu mai este automat adăugat.
- Constructorul este apelat automat după ce obiectul a fost construit în memorie.

Inițializarea prin constructor

```
class Rock
 {
Rock() { // это конструктор
System.out.println("Creating Rock");
}

public class SimpleConstructor { public static void main(String[] args)
{
for(int i = 0; i < 10; i++)
 new Rock();
}

}
```

Constructori cu parametri

```
class Rock2
{
Rock2(int i)
{
System.out.println( "Creating Rock number " + i);
} }

 public class SimpleConstructor2
 {
public static void main(String[] args)
{
for(int i = 0; i < 10; i++)
 new Rock2(i);
} }
```

Constructor implicit

```
class Bird
 {
int i;
 }
public class DefaultConstructor
 {
public static void main(String[] args)
 {
Bird nc = new Bird(); // по умолчанию!
}
}
```

Inițializarea în constructor

```
class Tag {
Tag(int marker) {
System.out.println("Tag(" + marker + ")");
} }
 class Card {
Tag t1 = new Tag(1); // Перед конструктором
Card() {
 System.out.println("Card()");
 t3 = new Tag(33);}
Tag t2 = new Tag(2); // После конструктора
void f() {
System.out.println("f()"); }
Tag t3 = new Tag(3); // В конце
}
public class OrderOfInitialization {
public static void main(String[] args) {
 Card t = new Card();
 t.f(); // Показывает завершение конструктора
} }
```

Supraîncărcarea metodelor

```
import java.util.*;
 class Tree {
int height;
 Tree() {
 prt("Planting a seedling");
 height = 0; }
Tree(int i) {
 prt("Creating new Tree that is " + i + " feet tall");
 height = i; }
 void info() {
 prt("Tree is " + height + " feet tall"); }
void info(String s) {
 prt(s + ": Tree is " + height + " feet tall"); }
 static void prt(String s) {
 System.out.println(s); } }
public class Over {
 public static void main(String[] args) {
 for(int i = 0; i < 5; i++)
{ Tree t = new Tree(i);
 t.info();
 t.info("overloaded method"); }
 // Перегруженный конструктор: new Tree(); } }
```

Cuvîntul chiee this

Cuvântul cheie **this** este folosit în cadrul metodelor atunci când se doreşte să se aibă acces la referinţa obiectului curent. Cuvântul cheie **this** este o referinţă către obiectul curent. Acest cuvânt cheie este folosit doar în cazurile speciale când este nevoie să se facă o referinţă explicită la obiectul curent.

```
public class Leaf
{
 int i = 0;
Leaf increment()
{
i++; return this;
 }
void print()
{
System.out.println("i = " + i);
}
public static void main(String[] args)
{
Leaf x = new Leaf();
x.increment().increment().increment().print();
} }
```

Activarea unui constructor în alt constructor

O altă utilitate a cuvântului cheie this este de apelare din cadrul unui constructor a altui constructor în scopul evitării duplicării codului.

```
public class Flower
 { int petalCount = 0;
 String s = new String("null");
 Flower(int petals)
{ petalCount = petals;
System.out.println( "Constr int arg only, petalCount= " + petalCount); }
    Flower(String ss)
{ System.out.println( "Constructor  String arg only, s=" + ss); s = ss; }
    Flower(String s, int petals) {
//! this(petals);  this(s); // Нельзя вызвать два!
 this.s = s; // Другое использование "this"
System.out.println("String & int args"); }
```

Dacă numele parametrului formal al unei metode este identic cu numele unui atribut din cadrul obiectului, atunci în cadrul metodei pentru a face distincţie între cele două se va folosi cuvântul cheie this pentru a specifica faptul că se doreşte folosirea atributului obiectului

continuare

```
Flower() { this("hi", 47);
System.out.println( "default constructor (no
   args)"); }
void print() {
 //! this(11); // Не внутри - не конструктор!
System.out.println( "petalCount = " +
   petalCount + " s = "+ s); }
 public static void main(String[] args)
{ Flower x = new Flower();
 x.print();
} }
```

Cuvîntul chiee static

Cuvântul cheie „**static**” este folosit în java pentru a defini o variabilă sau o metodă care poate fi accesata prin intermediul numelui clasei, fără a fi nevoie să se construiască obiecte de tipul respectiv. În construirea unei aplicaţii orientate pe obiecte utilizarea atributelor sau metodelor statice trebuie să fie evitată şi acestea să fie folosite doar în cazuri de strictă necesitate.

Un atribut static al unei clase este comun tuturor obiectelor de tipul respectiv. Există o singură locaţie de memorie pentru un atribut static indiferent de numărul obiectelor construite.

- O variabilă sau o metodă statică poate fi apelată doar din cadrul unei alte metode statice. Din cadrul unei metode statice a unei clase este imposibilă apelare unui atribut sau metodă membru al clasei dacă acesta nu este static.

continuare

```
class Cup
 { Cup(int marker) { System.out.println("Cup(" + marker + ")"); }
 void f(int marker) {
System.out.println("f(" + marker + ")"); } }
class Cups {
 static Cup c1;
static Cup c2;
static { c1 = new Cup(1);
 c2 = new Cup(2); }
 Cups() {
System.out.println("Cups()"); } }
```

continuare

```
public class ExplicitStatic {
 public static void main(String[] args) {
System.out.println("Inside main()");
 Cups.c1.f(99); // (1) }
// static Cups x = new Cups(); // (2)
 // static Cups y = new Cups(); // (2) }
```

Rezultatul realizării:

```
Run pentru 1
Inside main()
Cup(1)
Cup(2)
f(99)
Run pentru 2
Cup(1)
Cup(2)
Cups()
Cups()
Inside main()
f(99)
```

Inițializarea obiectelor dinamice

```
class Mug {
 Mug(int marker) {
System.out.println("Mug(" + marker + ")"); }
 void f(int marker) {
 System.out.println("f(" + marker + ")");
} }
public class Mugs {
Mug c1;
 Mug c2;
 { c1 = new Mug(1);
 c2 = new Mug(2);
System.out.println("c1 & c2 initialized"); }
 Mugs() {
System.out.println("Mugs()");
}
 public static void main(String[] args) {
 System.out.println("Inside main()");
Mugs x = new Mugs();
} }
```

continuare

Rezultatul realizării:

run-single:
Inside main()
Mug(1)
Mug(2)
c1 & c2 initialized
Mugs()

Inițializarea tabloului

```
public class Arrays
 {
 public static void main(String[] args)
{
 int[] a1 = { 1, 2, 3, 4, 5 };
int[] a2;
a2 = a1;
for(int i = 0; i < a2.length; i++)
a2[i]++;
 for(int i = 0; i < a1.length; i++)
 System.out.println( "a1[" + i + "] = " + a1[i]);
} }
```

Distrugerea obiectelor

Colectorul de gunoaie este un proces de prioritate scazuta care se executa periodic si marcheaza acele obiecte care au referinte directe sau indirecte. Dupa ce toate obiectele au fost parcurse, cele care au ramas nemarcate sunt eliminate automat din memorie (gc).

Un obiect este eliminat din memorie de procesul de colectare atunci cand nu mai exista nici o

referinta la acesta. Referintele (care sunt de fapt variabile) sunt distruse doua moduri:

• natural, atunci cand variabila respectiva iese din domeniul sau de vizibilitate, de exemplu la

terminarea metodei ın care ea a fost declarata;

• explicit, daca atribuim variabilei respective valoare null.

continuare

Inainte ca un obiect sa fie eliminat din memorie, procesul gc da acelui obiect posibilitatea sa ‘curete dupa el’, apeland metoda de finalizare a obiectului respectiv. Uzual , in timpul finalizarii un obiect isi inchide fisierele si socket-urile folosite, distruge referintele catre alte obiecte (pentru a usura sarcina colectorului de gunoaie).

Codul pentru finalizarea unui obiect trebuie scris intr-o metoda speciala numita **finalize** a clasei ce descrie obiectul respectiv.

Spre deosebire de destructorii din C++ , metoda finalize nu are rolul de a distruge obiectul, ci este apelata automat inainte de eliminarea obiectului respectiv din memorie.

Realizarea metodei finalize()

```
class Chair
 { static boolean gcrun = false;
 static boolean f = false;
 static int created = 0;
static int finalized = 0;
 int i;
 Chair() {
 i = ++created;
 if(created == 47)
 System.out.println("Created 47"); }
 public void finalize() {
if(!gcrun) {
// Первый раз вызывается finalize(): gcrun = true;
System.out.println("Beginning to finalize after " + created + " Chairs have been
     created"); }
if(i == 47) {
System.out.println( "Finalizing Chair #47, " + "Setting flag to stop Chair
     creation");
f = true; }
 finalized++;
if(finalized >= created)
System.out.println( "All " + finalized + " finalized"); } }
```

continuare

```
public class Garbage {
public static void main(String[] args) {
 // Пока флаг не установлен, создаются Chairs и Strings:
    while(!Chair.f) {
new Chair();
 new String("To take up space"); }
 System.out.println( "After all Chairs have been created:\n" + "total
    created = " + Chair.created + ", total finalized = " + Chair.finalized);
 // Необязательные аргументы форсируют сборку мусора
    if(args.length > 0) {
if(args[0].equals("gc") || args[0].equals("all")) {
    System.out.println("gc():");
 System.gc(); }
 if(args[0].equals("finalize") || args[0].equals("all")) {
    System.out.println("runFinalization():");
System.runFinalization();
 } }
System.out.println("bye!"); } }
```

Crearea tablourulor prin new

```
import java.util.*;
 public class ArrayNew
{
 static Random rand = new Random();
static int pRand(int mod)
{
 return Math.abs(rand.nextInt()) % mod + 1;
}
 public static void main(String[] args)
 {
int[] a;
 a = new int[pRand(20)];
 System.out.println( "length of a = " + a.length);
 for(int i = 0; i < a.length; i++)
System.out.println( "a[" + i + "] = " + a[i]);
 } }
```

Tablouri cu obiecte nestandarte

```
import java.util.*;
 public class ArrayClassObj
{
 static Random rand = new Random();
 static int pRand(int mod)
{
return Math.abs(rand.nextInt()) % mod + 1;
 }
public static void main(String[] args)
 {
 Integer[] a = new Integer[pRand(20)];
 System.out.println( "length of a = " + a.length);
for(int i = 0; i < a.length; i++)
 { a[i] = new Integer(pRand(500));
 System.out.println( "a[" + i + "] = " + a[i]);
} } }
```

Tablou cu mărimi și tipuri deferite

```
class A
{ int i; }
public class VarArgs
{
static void f(Object[] x)
{ for(int i = 0; i < x.length; i++) System.out.println(x[i]);
}
 public static void main(String[] args)
{
f(new Object[] { new Integer(47), new VarArgs(),
 new Float(3.14), new Double(11.11) });
 f(new Object[] {"one", "two", "three" });
f(new Object[] {new A(), new A(), new A()});
} }
```

Tablouri multidemensionale

```
import java.util.*;
 public class MultiDimArray
{
static Random rand = new Random(); static int pRand(int mod)
{
return Math.abs(rand.nextInt()) % mod + 1;
}
static void prt(String s) { System.out.println(s);
}
public static void main(String[] args)
{
 int[][] a1 = { { 1, 2, 3, }, { 4, 5, 6, }, };
for(int i = 0; i < a1.length; i++)
 for(int j = 0; j < a1[i].length; j++)
 prt("a1[" + i + "][" + j + "] = " + a1[i][j]);
```

continuate

```
// 3-х мерный массив фиксированной длины int[][][] a2 = new
     int[2][2][4];
 for(int i = 0; i < a2.length; i++)
for(int j = 0; j < a2[i].length; j++)
 for(int k = 0; k < a2[i][j].length; k++)
 prt("a2[" + i + "][" + j + "][" + k + "] = " + a2[i][j][k]);
 // 3-х мерный массив с векторами переменной длины
int[][][] a3 = new int[pRand(7)][][];
 for(int i = 0; i < a3.length; i++)
 {
a3[i] = new int[pRand(5)][];
 for(int j = 0; j < a3[i].length; j++)
 a3[i][j] = new int[pRand(5)];
}
 for(int i = 0; i < a3.length; i++)
for(int j = 0; j < a3[i].length; j++)
for(int k = 0; k < a3[i][j].length; k++)
prt("a3[" + i + "][" + j + "][" + k + "] = " + a3[i][j][k]);
```

continuate

```
// Массив не примитивных объектов
Integer[][] a4 = { { new Integer(1), new Integer(2)}, { new Integer(3),
    new Integer(4)},
{ new Integer(5), new Integer(6)}, };
 for(int i = 0; i < a4.length; i++)
 for(int j = 0; j < a4[i].length; j++)
 prt("a4[" + i + "][" + j + "] = " + a4[i][j]);
 Integer[][] a5;
a5 = new Integer[3][];
for(int i = 0; i < a5.length; i++)
 { a5[i] = new Integer[3];
 for(int j = 0; j < a5[i].length; j++)
 a5[i][j] = new Integer(i*j); }
 for(int i = 0; i < a5.length; i++)
 for(int j = 0; j < a5[i].length; j++)
 prt("a5[" + i + "][" + j + "] = " + a5[i][j]);
} }
```