Clase si obiecte (specificatori de acces, constructori, referințe si model de memorie)

Alexandru Olteanu

Universitatea Politehnica Bucuresti
Facultatea de Automatică si Calculatoare, Departamentul Calculatoare
alexandru.olteanu@upb.ro

OOP, 2020

Subiecte de discutie

- Clase si obiecte: ce sunt si cum arata?
  (specificatori de acces, constructori, getters, setters, this)
- Alocarea memoriei
  (referinte, pointeri, alocarea memoriei si modelul de memorie)

Clase si obiecte: ce sunt si cum arata?

Programare Orientată-Obiect vs Programare Procedurală

Programare Procedurală	**Programare Orientată-Obiect**
Procedură: O listă de instrucțiuni care îi spun computerului ce să facă pas cu pas	Obiect: componentă a programului care știe cum să desfășoare anumite acțiuni și cum să interacționeze cu alte elemente din program
printf("Hello World!\n")	System.out.println("Hello World!")
in C, printf este o functie	in Java, System.out este un obiect, println este metoda sa

Programare Orientată-Obiect vs Programare Procedurală

Programare Procedurală	Programare Orientată-Obiect
Procedură: O listă de instrucțiuni care îi spun computerului ce să facă pas cu pas	Obiect: componentă a programului care știe cum să desfășoare anumite acțiuni și cum să interacționeze cu alte elemente din program
• tipuri de date primitive • tipuri de date compuse (a.k.a. composite data type, a.k.a. record, e.g. in C: struct si array)	• tipuri de date primitive • tipuri de date compuse (struct si array in C) • clase (tipuri ce compun date si actiuni)

Exercitiu: despre struct in C

Exercitiu: despre struct in C

Obiectele sunt instante ale claselor

```
int x;                  // tip de date primitiv
MyClass myobject; // tip de date compus, clasa
```

Obiectele sunt instante ale claselor

```
int x;                  // tip de date primitiv
MyClass myobject; // tip de date compus, clasa
```

Clasele sunt tipuri de date compuse, ce contin membrii (aka argumente):

- date sub forma de campuri (adica variabile)
- actiuni sub forma de metode (adica functii)

Obiectele sunt instante ale claselor

```
int x;                  // tip de date primitiv
MyClass myobject;       // tip de date compus, clasa
```

Clasele sunt tipuri de date compuse, ce contin membrii (aka argumente):

- date sub forma de campuri (adica variabile)
- actiuni sub forma de metode (adica functii)
  - constructori
  - getters si setters (accessors / properties)
  - alte metode

Obiectele sunt instante ale claselor

```
public class Punct2D {
  private int x;
  private int y;

  public Punct2D() {
    this.x = 0;
    this.y = 0;
  }

  public Punct2D(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int getX() {
    return x;
  }

  public void setX(int x) {
    this.x = x;
  }
```

Specificatori de acces

- public - permite acces complet din exteriorul clasei curente
- private - limitează accesul doar în cadrul clasei curente
- protected - limitează accesul doar în cadrul clasei curente și al tuturor descendenților ei (conceptul de descendență sau de moștenire va fi explicat cursul următor)
- (default) - în cazul în care nu este utilizat explicit nici unul din specificatorii de acces de mai sus, accesul este permis doar în cadrul pachetului (package private). Atenție, nu confundați specificatorul default (lipsa unui specificator explicit) cu vreunul din ceilalți specificatori!

Obiectele sunt instante ale claselor

```
class Punct2D {
  private:
     int x;
     int y;

  public:
     Punct2D() {
        this->x = 0;
        this->y = 0;
     }

     Punct2D(int x, int y) {
        this->x = x;
        this->y = y;
     }

     int getX() {
        return x;
     }

     void setX(int x) {
```

Specificatori de acces: exemplu

```
class GeneratorNume {
  private String[] vocabular = {new String("Ana"), new
      String("Mihai"), new String("Robert")};
  private Random randomGenerator = new Random();

  public String genNume() {
    return vocabular[randomGenerator.nextInt(3)];
  }
}
...
GeneratorNume gn = new GeneratorNume();
System.out.println(gn.genNume());
```

Cuvantul cheie this

Cuvantul cheie this se refera la obiectul (instanta clasei) din care se apeleaza functia respectiva.

- dezambiguizare intre variabile locale si parametrii
- claritatea codului
- apelul unui constructor
- valoare de return pentru orice functie

Constructori: Default Constructor

Default constructor (no-arg constructor): provides the default values to the object like 0, null etc. depending on the type

```
public class Magazin {
    private String brand;

    public Magazin() {
      brand = "de inchiriat";
    }
}
...
Magazin item = new Magazin();
```

Constructori: Parameterized Constructor

Parameterized constructor: provide different values to the distinct objects

```
public class Magazin {
    private String brand;

    public Magazin(String brand) {
      this.brand = brand;
    }
}
...
Magazin item = new Magazin("IKEA");
```

Constructori: Constructor Overloading

A class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type

```
public class Magazin {
    private String brand;
    private Integer suprafata;

    public Magazin(Integer suprafata) {
      this.suprafata = suprafata;
      this.brand = "de inchiriat";
    }

    public Magazin(Integer suprafata, String brand) {
      this.suprafata = suprafata;
      this.brand = brand;
    }
}
```

Constructori: Copy Constructor

In Java, there is no Copy Constructor per-se, but there are many ways to copy the values of one object into another:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

```
public class Training {
    private int pushups;
    private int crunches;

    public Training(Training otherTraining) {
      this.pushups = otherTraining.pushups;
      this.crunches = otherTraining.crunches;
    }
}
```

Destructori

In C++, se apeleaza la iesirea din scope a variabilei:

```
class Catalog {
  private:
    int* note;

  public:
    Catalog(int count = 20) {
      note = new int[count];
    }

    ~Catalog() {
      delete[] note;
    }
};

int main(){
    Catalog* grupa321CD = new Catalog(30); // -> constructor
    ...
    delete grupa321CD;                     // -> destructor
}
```

Destructori

In C++, se apeleaza la iesirea din scope a variabilei:

```
class Catalog {
  private:
    int* note;

  public:
    Catalog(int count = 20) {
      note = new int[count];
    }

    ~Catalog() {
      delete[] note;
    }
};

int main(){
    Catalog* grupa321CD = new Catalog(30); // -> constructor
    ...
    //delete grupa321CD;
}                                          // -> destructor
```

Destructori

In Java exista functia finalize, dar nu stim cand este apelata pentru ca de gestiunea memoriei se ocupa Garbage Collector

Accessors (Setters and Getters)

Campurile expuse prin intermediul functiilor de tip Setter-Getter se numesc proprietati.
De ce sa folosim Setter/Getter in loc de camp direct?

Getter și Setter: motive

Pot exista nivele de acces diferite pentru setter și getter (sau chiar unul să nu existe)

```
public class Employee {
    private int salary;

    public int getSalary() {
      return salary;
    }
    protected void setSalary(int salary) {
      this.salary = salary;
    }
}
```

Getter și Setter: motive

Se poate ascunde reprezentarea internă:

```
public class Employee {
    private String street;
    private int number;
    private String city;

    public int getAddress() {
      return street+", "+number+", "+city;
    }
}
```

Getter și Setter: motive

Se pot face validări:

```
public class Employee {
    private String email;

    public void setEmail(String email) {
      if (!EmailChecker.isValid(email)) {
        System.out.println(email);
      } else {
        this.email = email;
      }
    }
}
```

Getter și Setter: motive

Se pot face conversii:

```
public class Employee {
    private float height;

    public void setHeight(float height, String measure)
        {
      if (!measure.equals("m")) {
        this.height = height / 0.3048;
      } else {
        this.height = height;
      }
    }
}
```

Getter și Setter vs Boilerplate Code

In C#:

```
public abstract class Foo {
    public virtual string Hello { get; set; }
}
```

In Java:

Project Lombok

Alocarea memoriei

Pointeri vs Referințe în C/C++

```
int x = 13, y = 14;
int *px = &x, &rx = x;
```

Pointeri vs Referințe în C/C++

```
int x = 13, y = 14;
int *px = &x, &rx = x;

int *px, &rx;                    // eroare: referintele sunt
    initializate la creare
```

Pointeri vs Referințe în C/C++

```
int x = 13, y = 14;
int *px = &x, &rx = x;

int *px, &rx;               // eroare: referintele sunt
    initializate la creare

px = &y;  rx = y;           // nu se modifica referinta, ci
    variabila referita
```

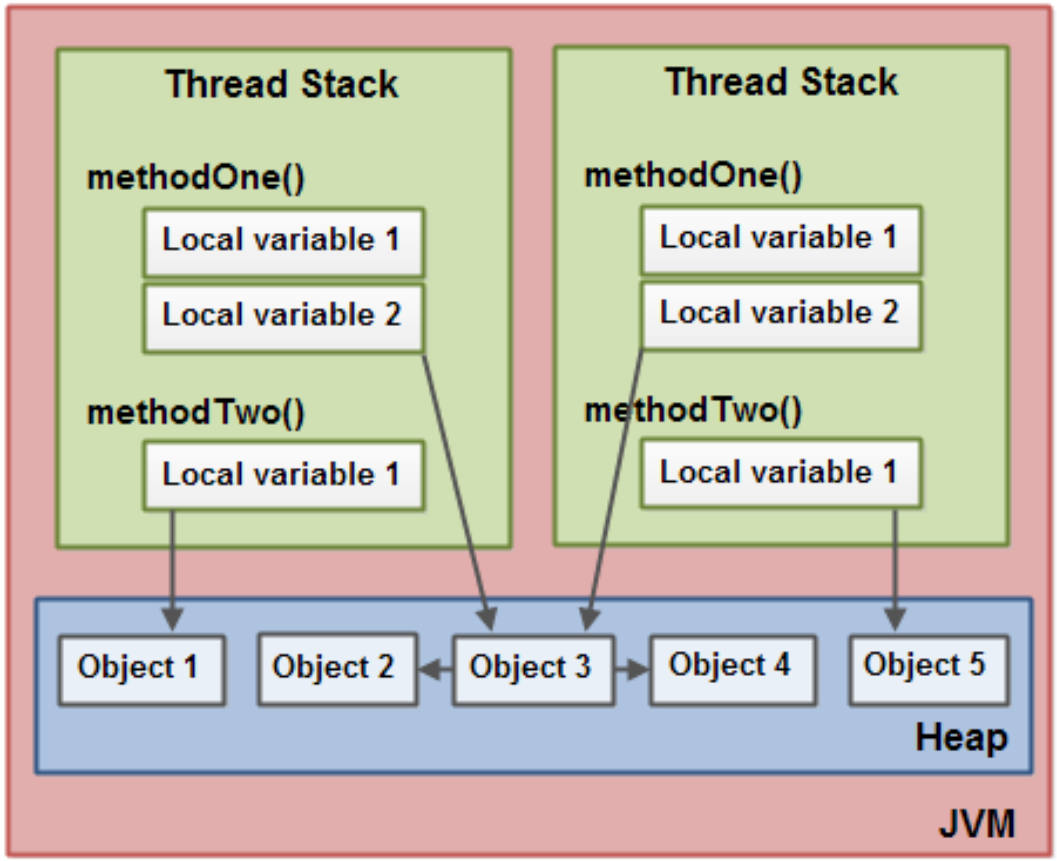
Pointeri vs Referințe în C/C++

```
int x = 13, y = 14;
int *px = &x, &rx = x;

int *px, &rx;                    // eroare: referintele sunt
    initializate la creare

px = &y;   rx = y;               // nu se modifica referinta, ci
    variabila referita

px++;    rx++;                   // nu se modifica referinta, ci
    variabila referita
```

Pointeri vs Referințe în Java

Exercitiu: În Java avem doar referințe, nu avem pointeri

Modelul de memorie al JVM



▸ Java Memory Model on Jenkov.com

Tipuri de date primitive: alocarea memoriei

În Java, variabile primitive pot fi locale unei metode sau membrii unui obiect

```
public class Album {
  public static void main(String[] args) {
    int price;
  }
}
```

```
public class Album {
  int price;
  public static void main(String[] args) {
  }
}
```

Tipuri de date primitive: valori default

variabilele locale nu sunt inițializate

```
public class Album {
  public static void main(String[] args) {
    int price;
    price++; // eroare de compilare
  }
}
```

Tipuri de date primitive: valori default

variabilele locale nu sunt inițializate

```
public class Album {
  public static void main(String[] args) {
    int price;
    price++; // eroare de compilare
  }
}
```

membrii claselor sunt inițializați cu 0 sau null, în funcție de tip

```
public class Album {
  int price;
  public static void main(String[] args) {
    price++; // bad programming style
  }
}
```

▸ Primitive Data Types doc

Alocarea memoriei

La declararea fără inițializare se creează o referință nulă:

```
Magazin spatiuDeInchiriat;
```

Se alocă spațiu pe heap și se apelează un constructor la inițializare:

```
Magazin magazinSport = new Magazin("Articole Sportive");
```

Alocarea memoriei

Două referințe la același obiect: se modifică obiectul

```
public Arena {
  public int seats;
  Arena(int seats) {
    this.seats = seats;
  }
}

Arena arenaNationala = new Arena(55000);
stadionulNational = arenaNationala;
stadionulNational.seats += 600;
System.out.println(arenaNationala.seats); // 55000 ?
    55600
```

Comparații

== tests for reference equality (whether they are the same object)
.equals() tests for value equality (whether they are logically "equal")

```
String name1 = new String("Michael"), name2 = name1,
    name3 = new String("Michael");
System.out.println(name1==name2);
System.out.println(name1==name3);
System.out.println(name1.equals(name3));
```

Comparații

Credeti ca ati inteles?

```
public class MyProgram {

    public static void main(String args[])
    {
        Integer a = Integer.valueOf(1);
        Integer b = Integer.valueOf(1);
        System.out.println(a==b);

        Integer x = Integer.valueOf(10001);
        Integer y = Integer.valueOf(10001);
        System.out.println(x==y);
    }

}
```

Transferul parametrilor

Transferul parametrilor la apelul funcțiilor este crucial pentru o funcționare corectă:

- variabile de tip primitiv
  - se transferă prin copiere pe stivă
  - orice modificare din functie a valorii variabilei NU VA FI VIZIBILA
- obiecte
  - se transferă prin referinta pe stivă
  - orice modificare din functie a referintei obiectului (e.g. p = new Player()) NU VA FI VIZIBILA

Tipuri de date compuse: eliberarea memoriei

Garbage Collector:

- Când un obiect nu mai este folosit, Garbage Collector revendică spațiul său de memorie de pe Heap pentru a fi refolosit

▸ Java Garbage Collection Basics - Oracle

Reading Assignments

- Lab 02: Constructori si Referințe
- Java Memory Model on Jenkov.com
- Rule of three, of five, of zero

inca o data...

Tipuri de date

Tipuri de date:

- primitive
- compuse
- omogene

Tipuri de date

Tipuri de date in C/C++:

- primitive
  - int, char, float, double, void
  - signed, unsigned

Tipuri de date

Tipuri de date in C/C++:

- primitive
  - int, char, float, double, void
  - signed, unsigned
- compuse
  - struct din C
  - struct, class din C++

Tipuri de date

Tipuri de date in C/C++:

- primitive
  - int, char, float, double, void
  - signed, unsigned
- compuse
  - struct din C
  - struct, class din C++
- omogene
  - vectori (a.k.a. arrays)
  - structuri de date (implementate manual sau din STL)

Tipuri de date

Tipuri de date in Java:

- primitive:

```
float pret = 12.5;
```

- compuse - class:

```
Song song1 = new Song("Bucovina", "Mestecanis");
```

- omogene:

```
Song playlist[] = new Song[]{
  new Song("Subcarpati", "Balada Romanului"),
  new Song("Subcarpati", "Frunzulita, iarba deasa")
};
```