Programare Orientată pe Obiecte

Clase & Obiecte

Dr. Petru Florin Mihancea

V20180924

1

Diferența între clasă și obiect ?

Definiții

O clasă definește un set de obiecte care au aceeași structură și comportament

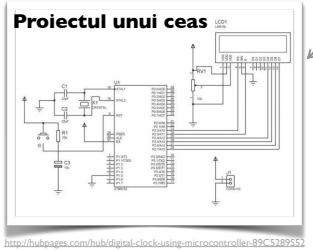
Un obiect este o instanță a unei clase

Booch - OO Analysis and Design

Clasa vs. Obiect

http://www.timetools.co.uk

Obiecte



http://hubpages.com/hub/digital-clock-using-microcontroller-89C5289552

Clasa

Implementarea unui obiect este dată/definită de clasa acelui obiect. Clasa specifică datele interne ale unui obiect și reprezentarea lor și definește operațiile obiectului

GOF, 1995

Definiție (parțială)

Programarea orientată pe obiecte
este o metodă de implementare a programelor
în care acestea sunt organizate ca și
colecții de obiecte ce cooperează între ele [...],

fiecare obiect fiind o instanță a unei clase

Booch - OO Analysis and Design

2

Simulare în C

Cum ar fi în limbajul C (I) ?

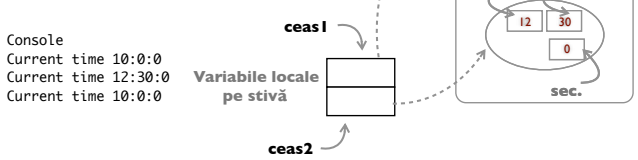
C nu este object-oriented
dar putem "simula"

```
typedef struct _Clock {
        int hour, minute, seconds;
} Clock;

void setTime(Clock *this, int hour, int minute, int second) {
    this->hour = ((hour >= 0 && hour < 24) ? hour : 0);
    this->minute = ((minute >= 0 && minute < 60) ? minute : 0);
    this->seconds = ((second >= 0 && second < 60) ? second : 0);
}

void print(Clock *this) {
    printf("Current time %d:%d:%d\n", this->hour, this->minute,
           this->seconds);
}
```

~ un fel de clasă

Cum ar fi în limbajul C (II) ?

```
int main(int argc, char** argv) {
   Clock *ceas1 = malloc(sizeof(Clock));
   Clock *ceas2 = malloc(sizeof(Clock));
   setTime(ceas1, 10, 0, 0);
   print(ceas1);
   setTime(ceas2, 12, 30, 0);
   print(ceas2);
   print(ceas1);
   ...
  //Eliberare memorie
   ...
}
```



3

Clase și Obiecte în Java

Dr. Petru Florin Mihancea

Ceasuri în Java

```
class Clock {

   private int hour, minute, seconds;

   public void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }

   public void print() {
      System.out.println("Current time " + hour + ":" + minute + ":" + seconds);
   }

}
```

Variabile/câmpuri instanță

```
class Clock {

  //Modificatori tip nume1, nume2 , ..., numeN = initVal;
 private int hour, minute, seconds;

 ...
}
```

definesc reprezentarea datelor unui obiect

fiecare obiect are propriul exemplar (locație de memorie) pt. fiecare din variabile sale instanță

Metode instanță

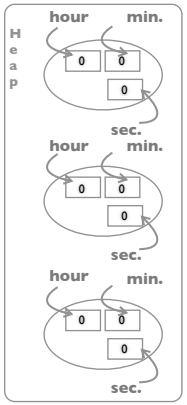
```
class Clock {
   ...
   //Modificatori tip_returnat nume(tip param1, ..., tip paramN) ...
   public void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }
   ...
}
```

definesc comportamentul / operațiile ce se pot executa pe acel obiect

Crearea obiectelor

la **rularea programului** prin operatorul **new** și sunt alocate în **heap**

```
class Main {
    public static void main(String argv[]) {
        ...
        new Clock();
        new Clock();
        new Clock();
        ...
    }
}
```



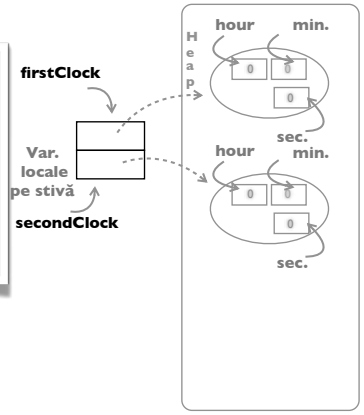
Ștergerea obiectelor

De acest lucru se ocupă colectorul de deșeuri

În esență, acesta șterge singur din timp în timp obiectele care nu mai pot fi accesate din program

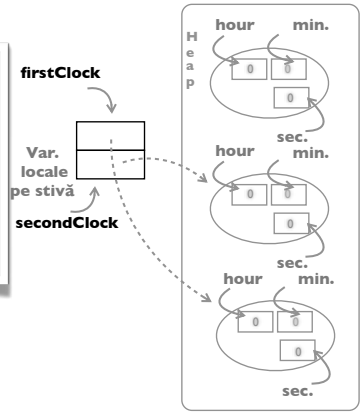
Referirea obiectelor

```
class Main {
    public static void main(String argv[]) {
        ...
        //Variabile referință
        //Aici variabile locale dar pot fi și câmpuri
        //NumeClasa numeVariabila;
        Clock firstClock, secondClock;
        firstClock = new Clock();
        secondClock = new Clock();
```



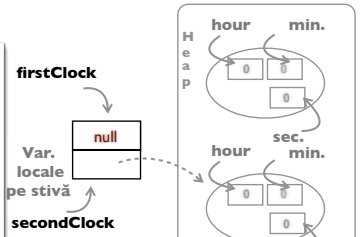
Referirea obiectelor

```
class Main {
    public static void main(String argv[]) {
        ...
        //Variabile referință
        //Aici variabile locale dar pot fi și câmpuri
        //NumeClasa numeVariabila;
        Clock firstClock, secondClock;
        firstClock = new Clock();
        secondClock = new Clock();
        firstClock = new Clock();
```



Referirea obiectelor

```
class Main {
    public static void main(String argv[]) {
        ...
        //Variabile referință
        //Aici variabile locale dar pot fi și câmpuri
        //NumeClasa numeVariabila;
        Clock firstClock, secondClock;
        firstClock = new Clock();
        secondClock = new Clock();
        firstClock = new Clock();
        firstClock = null;
        ...
    }
}
```

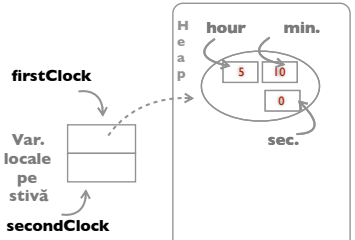


- **null - valoare ce arată că o referință nu referă niciun obiect**
- **nu e echivalent cu 0 !**

Invocarea operațiilor

```
class Main {
    public static void main(String argv[]) {
        //operatorul . (punct)
        Clock firstClock, secondClock;
        firstClock = new Clock();
        firstClock . setTime(5, 10, 0);
```

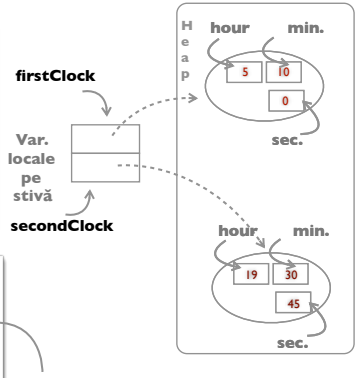
```
class Clock {
    ...
    public void setTime(int h, int m, int s) {
        hour = (h >= 0) && (h < 24) ? h : 0;
        minute = (m >= 0) && ( m < 60) ? m : 0;
        seconds = (s >= 0) && ( s < 60) ? s : 0;
    }
    ...
}
```



Dr. Petru Florin Mihancea

Invocarea operațiilor

```
class Main {
    public static void main(String argv[]) {
        //operatorul . (punct)
        Clock firstClock, secondClock;
        firstClock = new Clock();
        firstClock . setTime(5, 10, 0);
        secondClock = new Clock();
        secondClock . setTime(19, 30, 45);
        ... // tot . (punct) și acces la câmpuri
    }
}
```



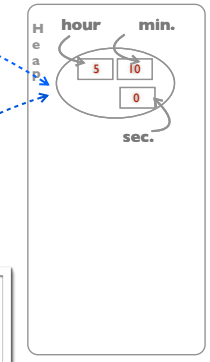

```
class Clock {
    ...
    public void setTime(int h, int m, int s) {
        hour = (h >= 0) && (h < 24) ? h : 0;
        minute = (m >= 0) && ( m < 60) ? m : 0;
        seconds = (s >= 0) && ( s < 60) ? s : 0;
    }
    ...
}
```

De unde știe pe câmpurile cărui obiect lucrează ?

Dr. Petru Florin Mihancea

Referința this

```
class Clock {
    ...
    public void setTime(int h, int m, int s) {
        this.hour = (h >= 0) && (h < 24) ? h : 0;
        this.minute = (m >= 0) && ( m < 60) ? m : 0;
        this.seconds = (s >= 0) && ( s < 60) ? s : 0;
    }
    ...
}
```



în timpul execuției unei metode instanță, this referă spre obiectul din fața apelului
(obiectul receptor)

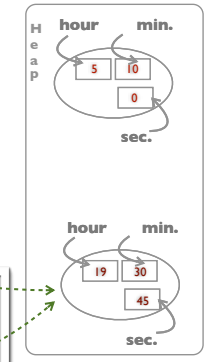
```
...
Clock firstClock, secondClock;
firstClock = new Clock();
secondClock = new Clock();
firstClock . setTime(5, 10, 0);
secondClock . setTime(19, 30, 45);
...
```

Referința this

```
class Clock {
    ...
    public void setTime(int h, int m, int s) {
        this.hour = (h >= 0) && (h < 24) ? h : 0;
        this.minute = (m >= 0) && ( m < 60) ? m : 0;
        this.seconds = (s >= 0) && ( s < 60) ? s : 0;
    }
    ...
}
```

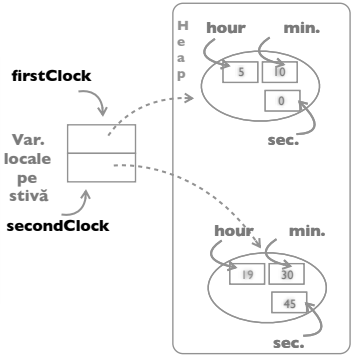
în timpul execuției unei metode instanță, this referă spre obiectul din fața apelului
(obiectul receptor)

```
...
Clock firstClock, secondClock;
firstClock = new Clock();
secondClock = new Clock();
firstClock . setTime(5, 10, 0);
secondClock . setTime(19, 30, 45);
...
```



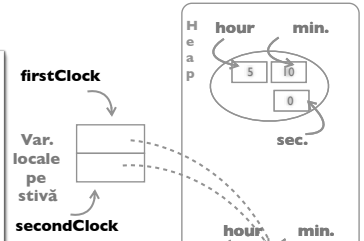
Atenție la aliasing !

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock, secondClock;
        firstClock = new Clock();
        firstClock . setTime(5, 10, 0);
        secondClock = new Clock();
        secondClock . setTime(19, 30, 45);
```



Atenție la aliasing !

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock, secondClock;
        firstClock = new Clock();
        firstClock .setTime(5, 10, 0);
        secondClock = new Clock();
        secondClock .setTime(19, 30, 45);
        firstClock = secondClock;
        firstClock .setTime(0, 0, 0);
        secondClock.print();
        ...
    }
}
```



Output
Current time 0:0:0

4

Constructorii

Ce este un constructor ?

“Metode” mai speciale care

- **au exact același nume ca și clasa**
- **nu au tip returnat (nici măcar void)**
- **se poate executa o dată la crearea obiectului**

```
class Clock {
 ...
  public Clock(int h, int m, int s) {
     setTime(h,m,s);
  }
 ...
}
```

Utilizarea

```
class Exemplu {
   public static void main(String argv[]) {
      //...
      Clock firstClock, secondClock, thirdClock;
      firstClock = new Clock(10, 0, 0);
      secondClock = new Clock(12, 30, 40);
      thirdClock = new Clock();//Eroare de compilare
      //...
   }
}
```

Dacă nu declarăm niciun constructor în clasă se generează automat unul fără nici un argument

Un constructor fără argumente se mai numește constructor no-arg

Ordinea inițializărilor (aprox)

```
class Student {
   private int numarMatricol = 0;
   public Student(int nr) {
      numarMatricol = nr;
   }
   public void print() {
      System.out.println(numarMatricol);
   }
   public static void main(String argv[]) {
      Student s1 = new Student(10);
      s1.print();
   }
}
```

Output
10

1. Se execută inițializatorii variabilelor instanță în ordine textuală
2. Se execută corpul constructorului

5

Accesul la membrii unei clase

Modificatori de access

Modificatori/Specificatori de acces (I)

```
class Clock {

 private int hour, minute, seconds;

 ...
}
```

private

respectivul membru al clasei (câmp/metodă) poate fi accesat **doar în interiorul clasei**

public

respectivul membru al clasei (câmp/metodă) poate fi accesat de **oriunde**

Atenție: dacă lipsește nu e nici public nici private (vom vedea mai încolo)

Modificatori/Specificatori de acces (II)

```
class Specifier{
    public int publicAttribute;
    private int privateAttribute;
    public void publicMethod() {
        publicAttribute = 20;
        privateAttribute = privateMethod();
    }
    private int privateMethod() {
        return 10;
    }
    public void otherMethod(Specifier s) {
        publicAttribute = s.privateAttribute;
        privateAttribute = s.privateMethod();
        s.publicMethod();
    }
}
class ClientSpecifier{
   public static void main(String[] args) {
      Specifier s = new Specifier();
      s.publicMethod();
      s.privateMethod();//Linie cu eroare de compilare
   }
}
```

Euristică Importantă a Programării Orientate pe Obiecte

All data should be hidden (private) within its class

Riel's Heuristic 2.1

De ce ?

```
class A {
    ...
    public void method1(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class D {
    ...
    public void method4(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class ComplexNumber {
    public double x,y;
}
```

```
class B {
    ...
    public void method2(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
cl
}
```

Constatăm la un moment dat ca ar fi mai bine să reprezentăm prin modul și unghi un număr complex

De ce ?

```
class A {
    ...
    public void method1(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class D {
    ...
    public void method4(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class ComplexNumber {
    public double module, angle;
}
```

```
class B {
    ...
    public void method2(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class C {
    ...
}
```

1. Trebuie schimbate numele (se poate face relativ ușor)

De ce ?

```
class A {
    ...
    public void method1(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class D {
    ...
    public void method4(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

```
class ComplexNumber {
    public double module, angle;
}
```

```
class B {
    ...
    public void method2(ComplexNumber n) {
        ... // computation based on n.x, n.y
    }
    ...
}
```

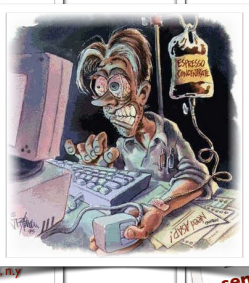
```
class C {
    ...
}
```

2. Dar s-a schimbat și semantica datelor deci trebuie schimbat tot codul client !!!

De ce ?

```
class A {
    ...
    public void method1(ComplexNu
        ... // computation based on n.x
    }
    ...
}
```

```
class D {
    oid method4(ComplexNumber n) {
        computation based on n.x, n.y
}
```




```
class B {
    ...
    public void method2(ComplexNu
        ... // computation based on n.x, n.y
    }
    ...
}
```

Dar s-a schimbat și **semantica datelor** deci trebuie schimbat **tot codul client** !!!

De ce ?

```
class A {
    ...
    public void method1(ComplexNumber n) {
        ... // computation based on n.getX/Y()
    }
    ...
}
```

```
class D {
    ...
    public void method4(ComplexNumber n) {
        ... // computation based on n.getX/Y()
    }
    ...
}
```

```
class ComplexNumber {
    private double module,angle;  //+ init methods
    public double getX() {return module * Math.cos(angle);}
    public double getY() {return module * Math.sin(angle); }
}
```

```
class B {
    ...
    public void method2(ComplexNumber n) {
        ... // computation based on n.getX/Y()
    }
    ...
}
```

```
class C {
    ...
    public void
}
```

Putem limita modificările codului la nivelul clasei (și a codului de instanțiere)

6

Membrii statici

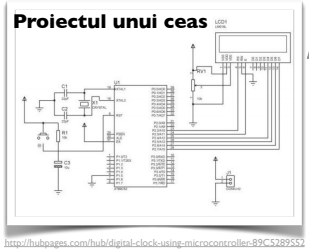
Modificatorul static

Întrebare ?

http://www.timetools.co.uk

Obiecte

Proiectul unui ceas



http://hubpages.com/hub/digital-clock-using-microcontroller-89C5289552

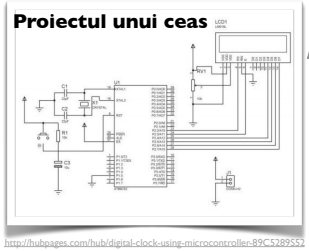
Clasa

Vrem un contor care să numere câte obiecte ceas am creat. Este acesta o variabilă instață a unui obiect ceas ?

Întrebare ?

http://www.timetools.co.uk

Obiecte



http://hubpages.com/hub/digital-clock-using-microcontroller-89C5289552

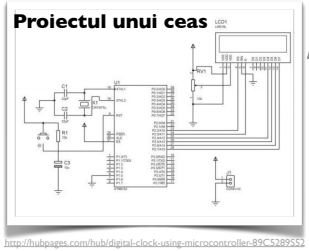
Clasa

NU. Dar a cui este ?

Întrebare ?

http://www.timetools.co.uk

Obiecte



http://hubpages.com/hub/digital-clock-using-microcontroller-89C5289552

Clasa

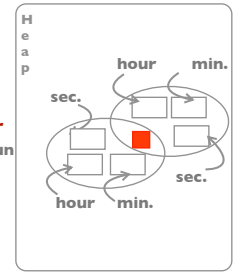
Este o variabilă a clasei !

Modificatorul static

```
class Clock {
    ...
    //numără câte ceasuri am creat
    //variabilă/câmp/metodă de clasă
    private static int count;
    public Clock(int h, int m, int s) {
        count++;
        setTime(h, m, s);
    }
    public static int getCount() {
        return count;
    }
}
```

counter

toate obiecte au un sigur exemplar



Modificatorul static (II)

metodele statice NU se execută pe un obiect al clasei

- **this nu are sens în metodele statice**
- **nu poți accesa membrii instanță via this (implicit ori explicit)**
- **utilizare NumeClasă.numeMetodă(params)**
  ex. Clock.getCount()

```
class Clock {
   private int hour, minute, seconds;
   private static int count;
   public static int oMetodaStatica(Clock c) {
      c.hour = 0;
      hour = 0; //Eroare de compilare
      this.hour = 0; //Eroare de compilare
   }
    ...
}
```

7

Variabile final

Modificatorul final pt. variabile

final

O astfel de variabilă poate fi atribuită o singură dată (altfel e eroare de compilare)

Un câmp instanță final trebuie să fie sigur inițializat până la sfârșitul oricărui constructor (altfel eroare de compilare)

```
class CatalogNote {

   //Exemplu de utilizare: definirea de constante simbolice
   //Prin conventie, numele lor se scrie nu litere mari
   public final static int NOTA_MAXIMA = 10;

}
```

8

Noțiuni teoretice

Identitatea obiectului

... este acea proprietate a unui obiect care îl distinge de oricare alt obiect

Booch - OO Analysis and Design

referința / adresa de memorie unde e alocat

NU confundați numele de variabile referință cu identitatea unui obiect !

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock, secondClock;
        firstClock = new Clock(0,0,0);
        firstClock .setTime(5, 10, 0);
        secondClock = new Clock(0,0,0);
        secondClock . setTime(19, 30, 45);
        firstClock = secondClock;
        firstClock .setTime(0, 0, 0);
        secondClock.print();
        ...
    }
}
```

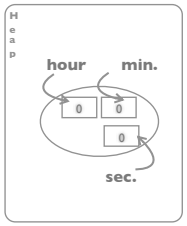
Starea obiectului

... reprezintă toate proprietățile obiectului plus valorile curente pentru fiecare din aceste proprietăți

Booch - OO Analysis and Design

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock;
        firstClock = new Clock(0, 0, 0);
        //Momentul A
```

Starea la momentul A



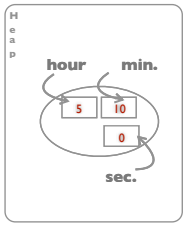
Starea obiectului

... reprezintă toate proprietățile obiectului plus valorile curente pentru fiecare din aceste proprietăți

Booch - OO Analysis and Design

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock;
        firstClock = new Clock(0, 0, 0);
        //Momentul A
        ...
        firstClock . setTime(5, 10, 0);
        //Momentul B
```

Starea la momentul B



Starea obiectului

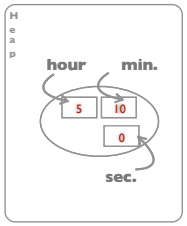
... reprezintă toate proprietățile obiectului plus valorile curente pentru fiecare din aceste proprietăți

Booch - OO Analysis and Design

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock;
        firstClock = new Clock(0, 0, 0);
        //Momentul A
        ...
        firstClock . setTime(5, 10, 0);
        //Momentul B
        ...
        firstClock.print(); //Current time 5: 10:0
```

Starea la momentul B+

Starea se conservă !



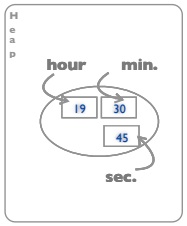
Starea obiectului

... reprezintă toate proprietățile obiectului plus valorile curente pentru fiecare din aceste proprietăți

Booch - OO Analysis and Design

```
class Main {
    public static void main(String argv[]) {
        ...
        Clock firstClock;
        firstClock = new Clock(0, 0, 0);
        //Momentul A
        ...
        firstClock .setTime(5, 10, 0);
        //Momentul B
        ...
        firstClock.print(); //Current time 5: 10:0
        ...
        firstClock .setTime(19, 30, 45);
        //Momentul C
```

Starea la momentul C



Comportamentul obiectului

... reprezintă cum anume acționează/ reacționează acel obiect în termeni de schimbare a stării sale și de interacțiune cu alte obiecte

Booch - OO Analysis and Design

setTime - ora/min/sec se schimbă la valoarea indicată sau 0 dacă valoarea dată e incorectă

print - tipărește pe ecran ora indicată

poate apare schimbare de stare în anumite condiții

ce se tipărește depinde de starea curentă

Comportamentul obiectului (II)

În esență, comportamentul unui obiect depinde de starea curentă cât și de operația efectuată pe obiect, iar unele operații pot altera starea sa

Exemplu
Cum se comportă acest obiect ?

http://www.e-automate-cafe.ro/

Interfața

... reprezintă setul de** (declarații de) **operații** (de obicei publice) **pe care un client le poate efectua pe un obiect

```
class Clock {
   private int hour, minute, seconds;
   public void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }
   public void print() {
      System.out.println("Current time " + hour + ":" + minute + ":" + seconds);
   }
}
```

restul ține de implementarea obiectului/clasei

Încapsularea

... compartimentarea elementelor unui obiect care formează structura și comportamentul său; încapsularea servește la separarea interfeței de implemntarea abstracțiunii

Booch - OO Analysis and Design

Încapsularea (II)

... împachetarea datelor și a metodelor** [ce lucrează cu acele date] **în clase în combinație cu ascunderea implementării

Eckel - Thinking in Java

```
class Clock {
            int hour, minute, seconds;
          void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }
          void print() {
      System.out.println("Current time " + hour + ":" + minute + ":" + seconds);
   }
}
```

Încapsularea (II)

... împachetarea datelor și a metodelor** [ce lucrează cu acele date] **în clase în combinație cu ascunderea implementării

Eckel - Thinking in Java

```
class Clock {
   private int hour, minute, seconds;
   public void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }
   public void print() {
      System.out.println("Current time " + hour + ":" + minute + ":" + seconds);
   }
}
```

Încapsularea (II)

... împachetarea datelor și a metodelor** [ce lucrează cu acele date] **în clase în combinație cu ascunderea implementării

Eckel - Thinking in Java

```
class Clock {
    [redacted]
    public void setTime(int h, int m, int s) {
        [redacted]
    public void print() {
        [redacted]
}
```

Încapsularea (II)

Încapsularea
(variație după autori)

... împachetarea datelor și a metodelor** [ce lucrează cu acele date] **în clase în combinație cu ascunderea implementării

Information/data
hiding

Eckel - Thinking in Java

```
class Clock {
    [redacted]
    public void setTime(int h, int m, int s) {
        [redacted]
    public void print() {
        [redacted]
}
```

9

Unified Modelling Language
Diagrama de clase

sau

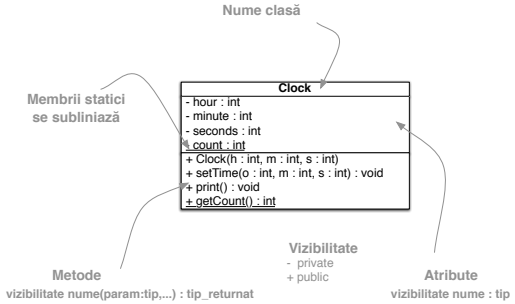
Să vorbim grafic despre clase

Dr. Petru Florin Mihancea

```java
class Clock {

   private int hour, minute, seconds;
   private static int count = 0;

   public Clock(int h, int m, int s) {
      count++;
      setTime(h, m, s);
   }

   public void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }

   public void print() {
      System.out.println("Current time " + hour + ":" + minute + ":" + seconds);
   }

   public static int getCount() {
      return count;
   }

}
```

Dr. Petru Florin Mihancea

Reprezentarea claselor



10

Supraîncărcarea metodelor

Overloading

Semnătura unei metode

```
class Clock {
   ...
   //Modificatori tip_returnat nume(tip param1, ..., tip paramN) ...
   public void setTime(int h, int m, int s) {
      ...
   }
   ...
}
```

Semnătura unei metodei

- **nume**
- **ordinea și tipurile parametrilor formali**

  implicit și numărul parametrilor :)

Supraîncărcarea (overloading)

```
class Clock {
   ...
   public void setTime(int h, int m) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = 0;
   }
   public void setTime(int h, int m, int s) {
      hour = (h >= 0) && (h < 24) ? h : 0;
      minute = (m >= 0) && ( m < 60) ? m : 0;
      seconds = (s >= 0) && ( s < 60) ? s : 0;
   }
   ...
}
```

Metode cu același nume dar semnături diferite

Exemplu

PrintStream

…
+print(b : boolean) : void
+print(c : char) : void
+print(i : int) : void
+print(s : double) : void
+print(s : String) : void
…
+print(b : boolean) : void
+println(b : boolean) : void
+println(c : char) : void
+println(i : int) : void
+println(s : double) : void
+println(s : String) : void
...

```
int a = 1;
String b = "something";
System.out.println(a);
System.out.println(b);
```

Merge și la constructori :)

```
class Clock {
   ...
   public Clock() {
      count++;
      hour = 12;
      minute = seconds = 0;
   }
   public Clock(int h, int m, int s) {
      count++;
      setTime(h, m, s);
   }
   ...
}
```

```
Clock a = new Clock();
Clock b = new Clock(10,20,30);
a.print();
b.print();
```

OUTPUT

```
Current time 12:0:0
Current time 10:20:30
```

Apeluri la constructori

```
class Clock {
   ...
   public Clock() {
         this(12,0,0);
   }
   public Clock(int h, int m, int s) {
      count++;
      setTime(h, m, s);
   }
   ...
}
```

numai din alți constructori

dacă e necesară această apelare, trebuie să fie prima instrucțiune

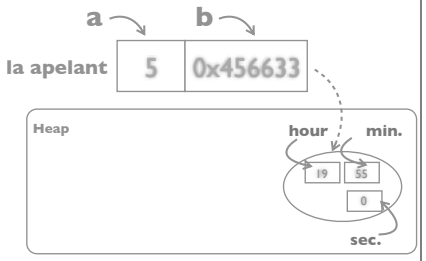
11

Transmiterea parametrilor

Transmiterea parametrilor Java

prin valoare

```
void caller() {
    int a = 5;
    Clock b = new Clock();
    b.setTime(19,55,0);
    someobj.called(a, b);
```

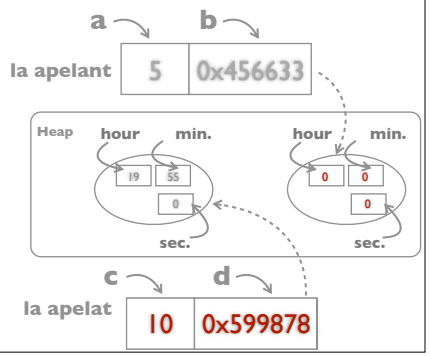


Transmiterea parametrilor Java

prin valoare

```
void caller() {
    int a = 5;
    Clock b = new Clock();
    b.setTime(19,55,0);
    someobj.called(a, b);
```

```
void called(int c, Clock d) {
    c = 10;
    d.setTime(0,0,0);
    d = new Clock(19,55,0);
}
```

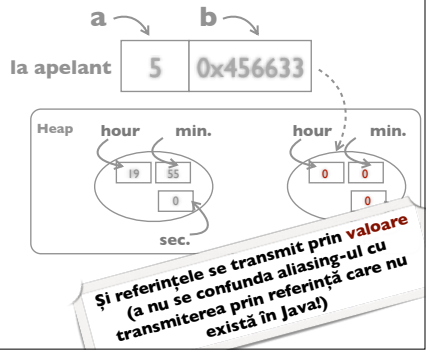


Transmiterea parametrilor Java

prin valoare

```
void caller() {
    int a = 5;
    Clock b = new Clock();
    b.setTime(19,55,0);
    someobj.called(a, b);
    //a este tot 5 !
    //b refera același obiect ceas (!)
    //dar ora a fost schimbată la 0:0:0
}
```

```
void called(int c, Clock d) {
    c = 10;
    d.setTime(0,0,0);
    d = new Clock(19,55,0);
}
```



Și referințele se transmit prin valoare (a nu se confunda aliasing-ul cu transmiterea prin referință care nu există în Java!)

12

Atenție la ...

Atenție la ce referă o referință !

```
class Main {
   public static void main(String argv[]) {
      Clock firstClock;
      firstClock = null;
      firstClock.setTime(0,0,0);
      firstClock.print();
   }
}
```

OUTPUT

```
Exception in thread "main" java.lang.NullPointerException
   at nullpointer.Main.main(Main.java:7)
```

Nume identice

```
class Clock {
   private int hour, minute, seconds;
   ...
   public void setTime(int hour, int minute, int seconds) {
      hour = (hour >= 0) && (hour < 24) ? hour : 0;
      minute = (minute >= 0) && ( minute < 60) ? minute : 0;
      seconds = (seconds >= 0) && ( seconds < 60) ? seconds : 0;
   }
   ...
}
```

X

```
class Clock {
   private int hour, minute, seconds;
   ...
   public void setTime(int hour, int minute, int seconds) {
      this.hour = (hour >= 0) && (hour < 24) ? hour : 0;
      this.minute = (minute >= 0) && ( minute < 60) ? minute : 0;
      this.seconds = (seconds >= 0) && ( seconds < 60) ? seconds : 0;
   }
   ...
}
```

© Petru Florin Mihancea