Obiecte, metode, clase

Conținut

Programarea orientată pe obiecte (OOP) este o idee nouă, în domeniul tehnicilor de programare. Comparativ cu vechile metode de programare OOP oferă un mod de organizare al programelor preluând modelul vieții reale.

În cadrul următoarelor două expuneri se va încerca furnizarea unui răspuns la următoarele întrebări:

- Ce sunt obiectele şi clasele şi cum sunt legate între ele
- Ce rol au cele două părți importante ale unei clase sau obiect: atributele şi comportamentul
- Ce înseamnă moştenirea unei clase şi cum afectează aceasta modul de proiectare al programelor
- Ce semnificație au pachetele şi interfețe

A gândi utilizând obiectele: o analogie

Pentru a oferi o imagine asupra rolului obiectelor şi programării orientate pe obiecte vom considera două analogii între aceasta şi obiectele reale:

Să analizăm jocul Lego. Acesta este format din elemente, ce dispun de posibilitatea de a se culpa între ele. (pe o fața a unui element Lego există un pin de conectare, pe cealaltă fața se află un orificiu care permite conectarea cu alt element). Acest joc dispune de roți, motoraşe, cuplaje, role care permit prin intermediul conectărilor de subansamble realizarea unor castele, roboți, maşini de curse sau absolute orice îşi poate imagina utilizatorul.

Concluzionând, fiecare element Lego este un mic subanasmblu care permite conectarea sa cu alt element, într-un mod predefinit, respectând anumite reguli tehnologice. Acest lucru conduce la creearea unor obiecte masive. În mod similar lucrează şi OOP-ul: pune la un loc elemente mici pentru a creea un produs final masiv.

Un alt exemplu. Unii dintre dumneavoastră ați mers la o firma de calculatoare şi cu experiența de care dispuneți şi cu un pic de ajutor, eventual, ați cumpărat componentele unui calculator: placa de bază, microprocesorul şi radiatorul acestuia, placa video, placa de sunet, CDROM-ul, hard discul, tastatura, monitorul, sursa de alimentare, carcasa, etc. Le-ați asamblat şi v-ați putut bucura, (dacă nu au intervenit unele neplaceri legate de şuruburi căzute pe placa de bază sau conectoare puse invers datorită unor mufe mamă care permiteau acest lucru sau datorită dumneavoastră care le-ați făcut să permită acest lucru), de un calculator nou nouț şi mai ales funcțional.

Cu siguranță că ştiați că pe placa de bază sunt o serie de componente şi circuite, produse de diferite companii, a căror funcție nu sunt tocmai convins că o ştiați. Dar acest lucru nu v-a împiedicat să obțineți un calculator funcțional, chiar dacă nu ați avut nici cea mai mică idee ce rol a avut condensatorul fără terminale de pe placa de bază (pe care poate nu l-ați smuls când ați montat placa pe carcasă!!). Tot ceea ce ați urmărit a fost ca anumite conectoare şi mufe să se potrivească cu elementele de conectică ale celorlalte placi, pentru ca astfel succesul realizării calculatorului să fie deplin. Pe de altă parte, placa de bază comunică cu procesorul, placa video, tastatrura, hard discul, CDROM-ul, fllopy, iar acestea comunică între ele.

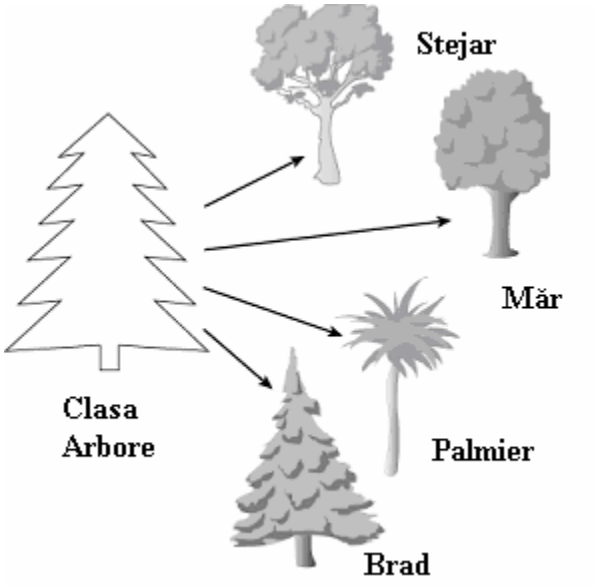
Programarea orientată pe obiecte lucrează în mod similar: utilizatorul poate folosi o serie de componente, dezvoltate anterior (obiecte), fără a fi nevoie ca acesta să le reproiecteze soft, fiecare componentă având însă un rol specific în cadrul programului principal şi fiecare componentă comunicând cu celelalte

Obiecte şi Clase

OOP este realizată într-un mod similar alcătuirii lumii reale: este alcătuit din mici obiecte extreme de diferite. Capacitatea de a combina obiectele este un aspect extrem de important în cadrul OOP. Această tehnică oferă numeroase concepte şi metode pentru a creea şi utiliza obiectele într-un mod cât mai flexibil, iar cea mai importantă modalitate se datorează existenței claselor.

În clipa când construiți un program, nu creeați obiecte. Definiți clase de obiecte, unde prin clasă înțelegeți o macro-structură care caracterizează mai multe obiecte ce au în comun anumite caracteristici. O clasă înglobează toate caracteristicile unui set particular de obiecte. Spre exemplu clasa Arbore, cuprinde toți copacii care au aceleaşi caracteristici dominante (au frunze,

rădăcini, tulpină, creează clorofilă). Conceptul clasă Arbore oferă un model abstract pentru acțiuni de tip: a atinge frunza unui copac, a taia un copac, etc. Pe lângă aceste acțiuni care se pot defini în raport cu un arbore definit conceptual, pot fi create o serie de elemente de tip membru, instanțe ale unei clase – cum ar fi definirea unui brad, palmier, stejar, etc - care însă păstrează caracteristicile unui arbore, chiar daca au alte atribute.



Clasa de tip Arbore şi câteva instanțe ale acesteia.

Definiție
Prin clasă se înțelege un prototip generic pentru un set de obiecte cu caracteristici similare.

Un exemplu sau o instanță a unei clase este un alt nume pentru obiectul actual. Dacă se consideră clasa o reprezentare generală a unui obiect, exemplul este reprezentarea concretă a clasei. Se pune întrebarea: care este diferența dintre o reprezentare a unui obiect şi obiectul în sine? Absolut niciuna. Obiectul este un termen general, dar atât acesta cât şi exemplu sunt reprezentări concrete ale unei clase. Aceşti termeni în cadrul OOP sunt intrerchanjabili: un exemplu de arbore şi un obiect arbore sunt acelaşi lucru.

Definiție
Un exemplu (instanță) este o reprezentare concretă a unei clase. Exemplele şi obiectele sunt unul şi acelaşi lucru.

Să presupunem că doriți să realizați, în Java, un element destinat interfeței utilizator, care va purta numele de buton. O clasă `Buton` defineşte caracteristicile butonului (eticheta acestuia, dimensiunile, aspectul) şi modul în care acesta se comportă. (Este nevoie de un click sau un

dublu click pentru activare? Își va schimba sau nu culoarea când va fi activat? Ce trebuie să facă când este activat?) Odată cu definirea clasei `Buton`, se pot creea uşor exemple ale acestui buton: butoane de toate dimensiunile şi culorile, care dispun de aceleaşi caracteristici ale clasei buton, dar au forma, culoare, dimeniuni şi comportament diferit, bazat pe particularizarea elementelor incluse în cadrul clasei mama. Creind o clasă `Buton`, nu va mai fi nevoie să rescrieți codul pentru fiecare buton individual pe care doriți să îl utilizați în cadrul programului, ci veți importa clasa cu anumite atribute setate în modul particular ce corespunde cerințelor dumneavoastră.

La scrierea unui program Java, programatorul creează şi construieşte un set de clase. La rularea programului, exemplele acestor clase sunt create şi apoi distruse, acolo unde este nevoie. În concluzie, un programator Java va trebui să creeze un set care să acopere toate cerințele programului.

Din fericire, nu sunteți nevoiți să porniți de la început cu creearea claselor Java: mediul Java, la instalare, dispune de un set standard de clase (numite biblioteci de clase) care implementează o serie de elemente de bază destinate programării la nivel fundamental (clase care construiesc funcții matematice de bază, tablouri, şiruri, etc.), precum şi elemente de grafică, animație, sunet şi comunicații. În multe cazuri, bibliotecile standard Java, furnizează destule clase, astfel încât programul Java va putea creea o singură clasă care va apela clasele din cadrul bibliotecii. Pentru programe Java complicate, va fi nevoie să creeați un întreg set de clase şi să definiți interacțiunile dintre ele.

Termen Nou
O bibliotecă de clase are scopul de a putea fi folosită în mod repetat în cadrul diferitelor programe. Biblioteca standard Java conține numeroase clase destinate programării fundamentale în Java.

Comportare şi atribute

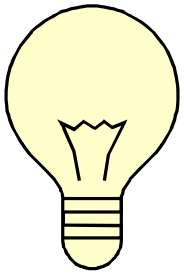
Fiecare clasă scrisă în Java are două elemente de bază: atribute şi comportament. Pentru a înțelege aceste exemple, în continuare vor fi exemplificate câteva structuri de programe.

Atribute

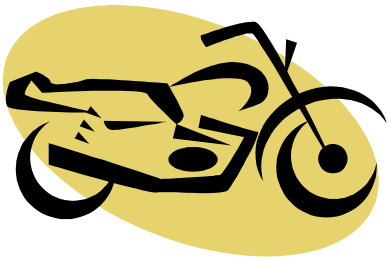
Atributele sunt elemente individuale care diferențiază un obiect de altul, determinând aspectul, starea şi alte calități ale respectivului obiect.

Să creem câteva exemple:

1) Fie o clasă ipotetică ce va purta numele de Bec. Această clasă va dispune de următoarele atribute:

<table>
<tr><td>Numele obiectului</td><td colspan="2">Bec</td><td rowspan="4">  </td></tr>
<tr><td rowspan="3">Atribut şi comportament</td><td rowspan="2">Aprins</td><td>Da</td></tr>
<tr><td>Nu</td></tr>
<tr><td colspan="2">Intensitate luminoasă (mărime numerică)</td></tr>
<tr><td colspan="2">Dimensiune (mărime numerică)</td></tr>
</table>

2) Fie o clasă pe care o numim `Motocicleta`. Această clasă va avea următoarele atribute cu următoarele valori tipice:



- `Culoare`: roşu, verde, argintiu, maro
- `Tip`: sport, standard, dragster
- `Marcă`: Honda, BMW, Bultaco

Ca atribut al unui obiect pot fi incluse şi informaţii referitoare la starea acestuia: dacă motorul este pornit sau nu sau treapta de viteză selectată la momentul curent.

Atributele sunt definite în cadrul claselor ca variabile. Numele şi tipul acestor variabile sunt definite în cadrul clasei, fiecare obiect având propriile valori ale atributelor definite prin intermediul variabilelor. Deoarece fiecare exemplu al unei clase dispune de valori diferite ale atributelor caracteristice clasei, variabilele poartă numele de variabilele exemplului sau variabilile instanței.

Definiție
Variabilele unui exemplu definesc atributele obiectului. Tipul şi numele variabilelor sunt definite în interiorul unei clase, dar valorile lor sunt setate şi modificate în cadrul obiectului.

Variabilele exemplelor pot fi setate inițial, la creearea obiectului şi pot rămâne constante toată viața obiectului sau se pot modifica odată cu rularea programului. Modificând valoarea unei variabile se modifică şi atributul obiectului.

În afara variabilelor exemplului, sunt variabilele clasei, care sunt aceleaşi pentru întreaga clasă şi pentru toate exemplele. Spre deosebire de variabilele exemplelor, variabilele clasei sunt memorate în cadrul clasei.

Comportament

Un comportament al unei clase determină modul în care un exemplu al acelei clase reacţionează. Spre exemplu cum reacţionează respectivul obiect dacă o altă clasă sau obiect îi "cere" să facă ceva sau dacă starea internă a clasei se modifică. Comportamentul este simgurul mod prin care un obiect poate să facă ceva pentru sine, prin sine sau pentru altă clasă. Spre exemplu, revenind la cazul clasei `Motocicleta`, acesta poate avea următorul comportament:

- Pornirea motorului Start Motor
- Oprirea motorului Stop Motor
- Accelerare
- Schimbarea vitezelor
- Decelerare

Pentru a defini un comportament, vor trebui definite metodele, un set de declaraţii Java care definesc un anumit task. Metodele se comportă exact ca nişte funcţii dar sunt definite şi accesibile numai în cadrul clasei. Java nu dispune de metode definite în afara unei clase.

Term nou
Metodele sunt funcţii definite în interiorul unei clase, care operează cu exemple ale acelei clase.

Metodele pot fi folosite pentru a opera în cadrul structurii unui singur obiect, dar totodată şi pentru a oferi facilităţile de comunicare ale obiectelor unul cu altul. O clasă sau un obiect poate chema metodele din cadrul altei clase sau obiect pentru a schimba mediul sau pentru a verifica schimbarea stării respectivului obiect.

Se pot identifica metode specifice unei clase, respective metode de exemplu: dacă metodele de clasă operează în cadrul clasei, metodele de exemplu operează în cadrul exemplelor unui obiect.

Crearea unei clase

Pentru exemplificarea aspectelor prezentate anterior se va construl o clasă `Motocicleta`, pentru ca apoi să poată fi urmărit modul în care apar şi sunt dezvoltate atributele şi comportamentul, respectiv metodele definite în cadrul unei clase Java.

Utilizând un editor de text va fi creat codul sursă Java prin introducerea următoarelor declaraţii (Atenţie Java este un mediu CaseSensitive – caracterele mari, respective caracterele mici sunt interpretate diferit):

```
class Motocicleta {

}
```

În acest moment a fost creeată o clasă. Se observă sărăcia acetsei clase care nu dispune de nici o caracteristică, metodă, comportament!

Să identificăm ce atribute sunt necesare pentru definirea unor obiecte ale acestei clase, respective ulterior ale unor exemple ale obiectelor acestei clase:

```
String marca;
String culoare;
boolean StareMotor = false;
```

Primele două atribute conțin obiecte de tip `String` (prin String se înțelege un şir de caractere, iar `String`, scris cu litera S, este o parte a unei biblioteci standard Java). Cel de-al III-lea atribut este o variabilă logică - `booleană`, care se referă la starea motorului: dacă este pornit `on` sau oprit `off`; valoarea `false` indică faptul că motorul este oprit, în timp ce valoarea `true` indică faptul că motorul este pornit.

Termen Nou
O variabilă booleană este adevărată - true sau falsă - false.

Să adăugăm acum o serie de metode sau comportamente ale clasei. Cea mai simplă şi logică metodă se referă la pornirea motorului. Acest lucru se realizează adăugând următoarele linii în cadrul programului:

```
void startMotor() {
    if (StareMotor == true)
        System.out.println("Motorul era pornit.");
    else {
        StareMotor = true;
        System.out.println("Motorul este pornit.");
    }
}
```

Metoda `startMotor()`testează dacă motorul este deja pornit şi ,dacă acest lucru este realizat afişează un mesaj. Dacă motorul nu este pornit, modifică atributul StareMotor al clasei Motocilceta în starea `true` şi afişează un mesaj. Deoarece metoda `startMotor()` nu întoarce nici o valoare, în cadrul definiției metodei este inclus cuvântul cheie void.

Atenție
Parantezele utilizate după numele acestei metode - `startMotor()` - indică faptul că aceasta nu întoarce nici o

variabilă. Acestea au numai rol de identificator de metodă.

Înainte de compilare să adăugăm o nouă metodă care să permită afişarea atributelor unui exemplu al clasei Motocicleta. Metoda `Paramteri()` listează valorile curente ale tuturor variabilelor unui obiect al unei clase `Motorcicleta`

```
void Parametri() {
    System.out.println("Aceasta motocicleta are culoarea  "
        + culoare + "  si este marca" + marca);
    if (StareMotor == true)
        System.out.println("Motorul este pornit.");
    else System.out.println("Motorul este oprit.");
}
```

Metoda `Parametri()` afişează pe ecran două linii: marca şi culoarea obiectului motocicletă, respectiv starea motorului pornit sau oprit.

Clasa Java dispune acum de 3 variabile şi două metode

```
javac Motorcicleta.java
```

La rularea clasei Motocicleta, programul Java va afişa următoarea eroare:

```
In class Motorcicleta: void main(String argv[]) is not defined
Exception in thread "main":  java.lang.UnknownError
```

Explicația se datorează faptului că Java presupune că respective clasă este o aplicație, motiv pentru care caută metoda principală `main()`. Deoarece acestă metoda nu a fost definită, în interiorul clasei, interpretorul Java a generat respective eroare.

Pentru a utiliza această clasă se construieşte un applet sau o aplicație sau se poate insera o metodă main în cadrul respectivei clase. Apelând la această ultimă metodă în cele ce urmează este un exemplu de metodă `main()` ce va fi adăugată la clasa `Motorcicleta` .

Metoda `main()` pentru clasa `Motorcicleta.java`.

```
 1: public static void main (String args[]) {
 2:    Motocicleta m = new Motocicleta ();
 3:    m.marca = "Yamaha RZ350";
 4:    m.culoare = "Galbena";
 5:    System.out.println("Apel al parametrilor clasei...");
 6:    m.Parametri();
 7:    System.out.println("--------");
 8:    System.out.println("Este pornit motorul...");
 9:    m.startEngine();
10:    System.out.println("--------");
11:    System.out.println("Apel al parametrilor clasei...");
```

```
12:    m.Parametri();
13:    System.out.println("--------");
14:    System.out.println("Este pornit motorul...");
15:    m.startEngine();
16:}
```

Cu ajutorul acestei metode `main()` clasa `Motorcicleta` este o aplicație Java ce poate fi compilată şi rulată fără erori.

```
D:\JDK12~1.2\bin>javac Motocicleta.java

D:\JDK12~1.2\bin>java Motocicleta
Apel al parametrilor clasei...
Aceasta motocicleta are culoarea  Galbena  si este marca Java 500
Motorul este oprit.
--------
Este pornit motorul...
Motorul este pornit.
--------
Apel al parametrilor clasei...
Aceasta motocicleta are culoarea  Galbena  si este marca Java 500
Motorul este pornit.
--------
Este pornit motorul...
Motorul era pornit.
```

În cele ce urmează este realizată o analiză amănunțită a metodei `main()`.

Prima linie va avea o formă identică pentru toate aplicațiile în care aceasta va fi folosită: `main()`.

Cea de-a doua linie , `Motocicleta m = new Motocicleta ();`, crează un nou exemplu al clasei `Motorcicleta` şi memorează o referință la aceasta în cadrul variabilei `m`. După cum se observă şi aici, în cadrul unui program Java nu se lucrează direct cu o clasă , ci cu obiecte create pe baza acestor clase şi apelul metodelor definite pe aceste obiecte.

Liniile 3 şi 4 setează variabilele exemplu pentru obiectul `Motocicleta` : mai precis marca `Yamaha RZ350`, iar culoarea Galbena.

Liniile 5 şi 6 apelează metoda `Parametri()` , definită în cadrul obiectului `Motocicleta.` Noul obiect `Motocicleta` – m oferă atributele de care dispune pentru a fi listate prin intermediul metodei `Parametri()`.

Linia 9 apelează metoda `startMotor()` în cadrul obiectului `Motocicleta` pentru a porni motorul. StareMotor va fi de acum pornită - true.

Linia 11 afişează din nou valorile curente ale obiectului `Motocicleta` prin apelul la metoda `Parametri()`.

Linia 15 încercă să pornească din nou motorul, dar acesta fiind deja pornit, atributul furnizat de obiect duce la afişarea mesajului `Motorul era deja pornit`.

Versiunea finala a programului `Motorcicleta.java`.

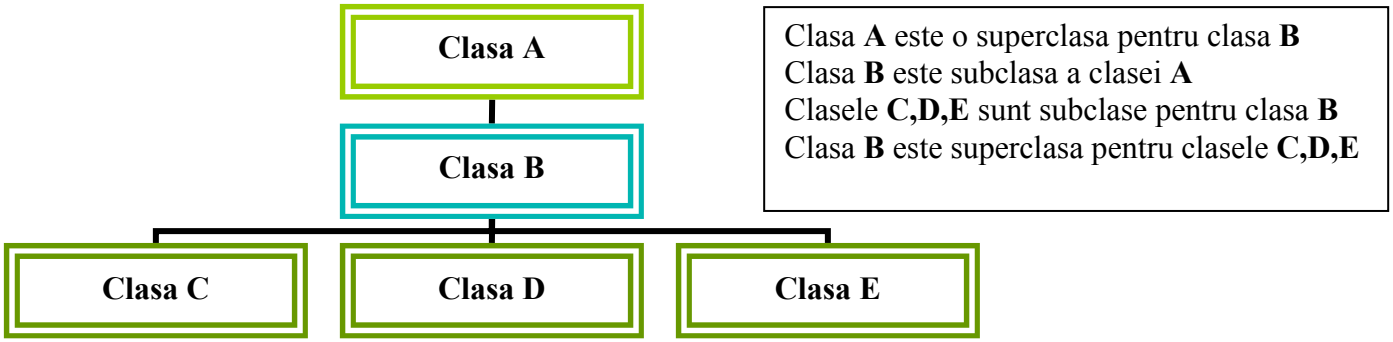
```
1: class Motocicleta
2:{
3:
4:    String marca;
5:       String culoare;
6:       boolean StareMotor = false;
7:       void startMotor() {
8:           if (StareMotor == true)
9:        System.out.println("Motorul era pornit.");
10:          else {
11:        StareMotor = true;
12:        System.out.println("Motorul este pornit.");
13:             }
14:                    }
15:
16:      void Parametri() {
17:             System.out.println("Aceasta motocicleta are
culoarea  "
18:             + culoare + "  si este marca" + marca);
19:      if (StareMotor == true)
20:        System.out.println("Motorul este pornit.");
21:      else System.out.println("Motorul este oprit.");
22:                    }
23:
24:    public static void main (String args[]) {
25:    Motocicleta m = new Motocicleta ();
26:    m.marca = "Yamaha RZ350";
27:    m.culoare = "Galbena";
28:    System.out.println("Apel al parametrilor clasei...");
29:    m.Parametri();
30:    System.out.println("--------");
31:    System.out.println("Este pornit motorul...");
32:    m.startEngine();
33:    System.out.println("--------");
34:    System.out.println("Apel al parametrilor clasei...");
35:    m.Parametri();
36:    System.out.println("--------");
37:    System.out.println("Este pornit motorul...");
38:    m.startEngine();
39:                                          }
40:}
```

Moştenire, interfeţe şi pachete

Moştenire

Moştenirea este un concept crucial în cadrul programării pe obiecte. În esență la scrierea unei noi clase este nevoie numai de specificarea elementelor care diferențiază o clasă de altă clasă anterior formată. În acest mod moştenirea oferă accesul automat la informațiile existente în cadrul clasei moştenite.

Prin prisma moştenirii, toate clasele create de un utilizator sunt aranjate într-o ordine ierarhică strictă, relativ la clasele bibliotecilor folosite. Astfel fiecare clasă are o superclasă - clasa situată deasupra ei în cadul tabloului ierarhic, iar fiecare clasă are una sau mai multe subclase (clase situate sub clasa curentă în cadrul tabloului ierarhic). Clasele situate în zona inferioară sunt moştenite de clasele din zona superioară.



Tablou ierarhic

Subclasele moştenesc toate metodele şi variabilele de la superclase, ceea ce implică faptul că acel comportament de care este nevoie în interiorul unei clase, nu va mai trebui definit din nou de utilizator, acesta fiind definit anterior în cadrul unei superclase. În felul acesta clasa astfel formată devine o moştenitoare a comportamentului claselor moştenite.

Term nou
Moştenirea este un concept al programării orientate pe obiecte, prin care toate clasele sunt aranjate într-o ordine ierarhică strictă. Fiecare clasă ierarhică are o superclasă, iar toate subclasele moştenesc atributele şi comportamentul superclaselor.

În vârful ierarhiei Java se află clasa `Object`; toate clasele moştenesc această superclasă. `Object` este cea mai generală clasă ierarhică, definind comportamentul moştenit de toate clasele din Java. Fiecare clasă aşezată ierarhic inferior, moşteneşte această clasă, şi adăugând informații devine cât mai centrată pe un anumit scop.

Astfel ierarhizarea claselor poate începe cu un concept abstract aflat în vârful tabloului de ierarhizare. Pe măsură ce se coboară în cadrul organigramei clasele nou realizate concretizează conceptual obiectuul ce se doreşte a fi implementat.

În cadrul programelor Java se doreşte realizarea unor clase care moştenesc toate elementele altor clase, elemente la care se adaugă un plus de informație. Spre exemplu, dacă se doreşte realizarea unei noi clase de tip `Buton` care să dispună de o etichetă inclusă în cadrul definiției, se va prelua ca moştenire clasa `Buton`, adăugând la noua clasă `(ButonNou??)` acele elemente care particularizează noua clasă relativ la clasa moştenită. Acest mecanism de definire a unei noi clase prin diferențierea dintre clasa ce se doreşte realizată şi clasa moştenită poartă numele de subclasare.

Subclasarea implică creearea unei noi clase din alte clase situate ierarhic superior. Utilizând subclasarea se vor specifica numai diferențele dintre clasa ce se doreşte a se forma şi clasa moştenită.

Term nou
Subclasarea este procesul de creeare a unei noi clase care moşteneşte superclasele existente.

În cazul în care clasa ce va fi formată nu moşteneşte nici o altă clasă, va fi indicată drept clasă moştenitoare clasa generică `Object`.

Crearea unei ierarhii de clase

Pentru cazul creerii unor structuri largi de clase, în cadrul unor programe complexe, se poate impune ca şi aceste clase să dispună de o ierarhizare proprie, care conduce la o organizare optimală a codului Java, dublată desigur de un efort conceptual ce va fi răsplătit din plin la finalul programului:

- La dezvoltarea unei ierarhizări de clase, se poate separa informația comună mai multor clase, construindu-se o superclasă, după care se poate folosi informația conținută în superclasă de câte ori este nevoie. Fiecare subclasă preia informația comună prin intermediul superclasei sale.
- Modificarea sau inserarea în zona superioară a unui comportament conduce la modificarea automată a comportamentului subclaselor fără a mai fi nevoie de recompilarea şi modificarea celorlalte subclase, deoarece acestea preiau informația din superclasa comună şi nu copiază o anumită zonă care poate fi modificată.

Continuând exemplul cu construcția clasei `Motorcicleta` să presupunem că ați implementat un program care modelează toate caracteristicile unei motociclete. După aceast doriți să construiți o nouă clasă `Autoturisme`.

`Autoturisme` şi `Motocicleta` dispun de numeroase elemente comune, ambele fiind vehicule acționate de un motor. Ambele dispun de transmisie, semnalizatoare, vitezometre. O soluție ar fi

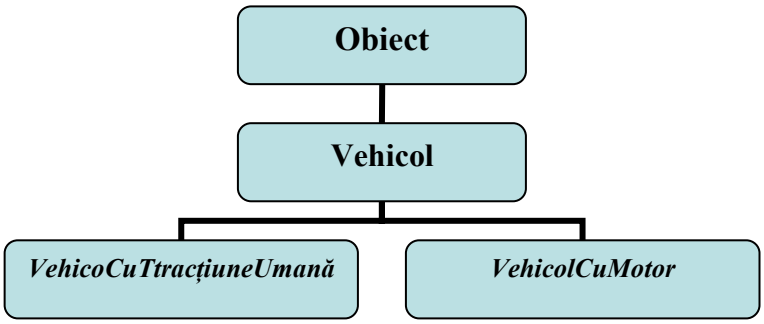
să deschideți clasa `Motocicleta` şi să copiați informațiile necesare construcției noii clase `Autoturisme`.

O soluție mult mai bună este să introduceți ambele clase într-o clasă mult mai generală. Poate nu se justifică numai pentru 2 clase, dar dacă veți fi nevoiți să construiți şi alte clase cum ar fi `Bicicleta`, `Scoter`, `Camion`, şi altele, care dispun de elemente reutilizabile importante, atunci efortul se dovedeşte a fi necesar.

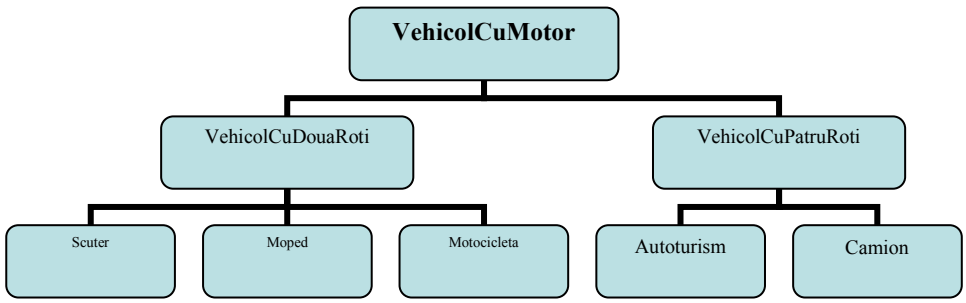
Un exemplu de clasă generală este clasa `Vehicol`. Un vehicol, în general etse definit a fi un mijloc de deplasare dintr-un loc în altul. În cadrul acestei clase se poate descrie comportamentul acestei clase de a permite deplasarea cuiva dintr-un punct a într-un punct b. Această clasă moşteneşte numai caracteristicile clasei `Object`.

Sub `Vehicol`? Pot fi introduce două noi clase: `VehicolCuTractiuneUmană` şi `VehicolCuMotor`.

`VehicolCuMotor` este diferit de `Vehicol` deoarece are un motor, iar comportarea sa include pornirea şi oprirea motorului, existența unui combustibil, schimbarea vitezelor. `VehicolCuTractiuneUmană` dispune de un sistem de transmisie a mişcării de la om la vehicol, de exemplu prin intermediul pedalelor.



Să încercăm acum să particularizăm o serie de subclase sau exemple ale clasei `VehicolCuMotor`. Din această clasă pot fi derivate următoarele clase: `Motocicleta`, `Autoturism`, `Camion` şi altele. Mai mult , poate fi divizată această clasă în alte două csubclase, care să poată îngloba mai multe atribute comune: `VehicolCuDouaRoti` şi `VehicolCuPatruRoti`.



Continuând acum cu exemple de subclase sau incidențe, putem adăuga la clasa `VehicolCuDouaRoti` subclasele: `Motocicleta, Scuter, Moped`.

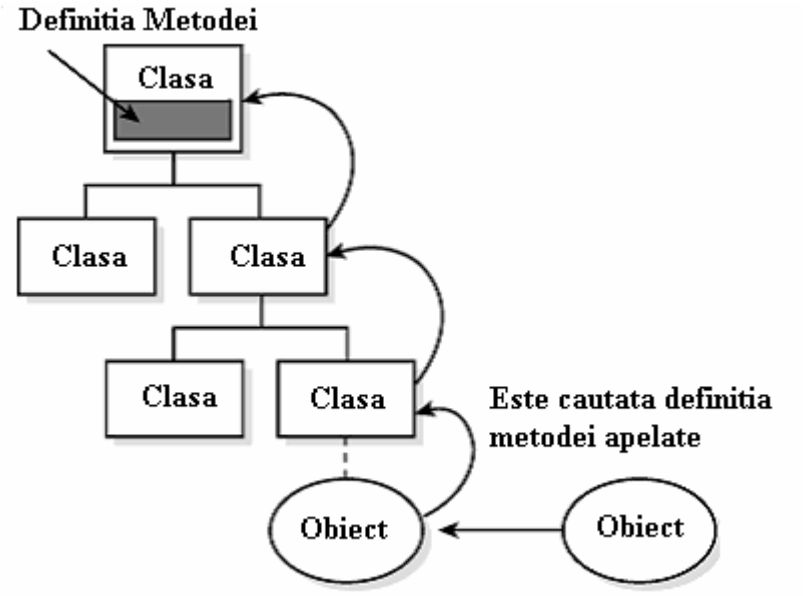
Atributele referitoare la culoare sau marcă vor putea fi integrate acum în cadrul clasei de vehicul, care derivat va forma alte instanțe ale respectivei clase.

Ce acțiune are moştenirea

Acest punct răspunde la întrebarea legată de ce reprezintă moştenirea şi cum se explică faptul că o instanță preia în mod automat variabilele şi metodele unei clase situate ierarhic superior.

La formarea unui exemplu sau instanță a unei clase, efectiv veți particulariza variabilele instanței definite prin intermediul clasei curente, la rândul său definită în cadrul superclasei din care provine. Astfel toate clasele sunt combinate pentru a forma un model al obiectului curent, iar fiecare obiect preia informațiile specifice unei situații concrete.

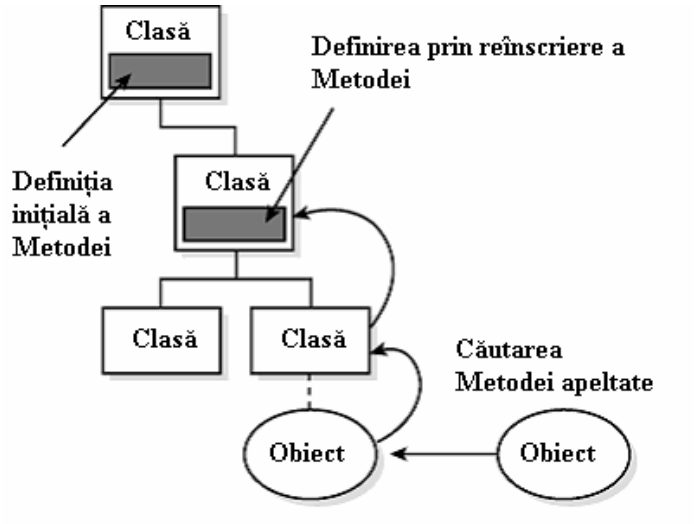
Metodele operează în mod similar. Noile obiecte dispun de acces total la metodele claselor sau superclaselor utilizate dar definirea şi utilizarea acestora se realizează prin apelul dinamic al acestora. Încercând să oferim o definiție plastică prin supercalse şi clase se creează limbajul propriu problemei, iar în cadrul construcției clasei curente etse folosit respectivul limbaj, fără a mai fi nevoie de explicarea termenilor (prin structuri program - desigur). La apelul unei metode a unui obiect particular, Java verifică dacă respectiva metodă este definită în cadrul clasei curente, după care, dacă respectiva definiție nu a fost găsită, căutarea are loc într-una dintre clasele importate sau superclase componente.



Un caz particular este situația în care definiția unei metode are acelați nume, numere, argumente ca şi o metodă definită într-o superclasă. În acest caz este valabilă prima definiție întâlnită plecând de la obiectul curent spre vârf (spre superclasa 0 sau supercalsa `Object`). Această

modalitate poartă numele de reînscriere a metodei şi poate fi realizată intenţionat pentru a ascunde anumite acţiuni ale metodei definite în cadrul superclasei.

Term nou
Supraînscrierea este o metodă de creeare a unei metode în cadrul unei sublase care are aceeaşi semnătură ca şi o metodă din cadrul unei supercalse. Noua metodă ascunde metoda definită în cadrul superclasei.



Moştenire simplă şi moştenire multiplă

Tipul de moştenire permis de Java este moştenirea simplă. Mai precis fiecare clasă poate avea o singură superclasă pe care să o moştenească. Superclasele pot avea însă nenumărate subclase.

În alte limbaje orientate pe obiecte, cum ar fi C++, clasele pot avea mai multe superclase, lucru care oferă o mulţime de variabile şi metode disponibile pentru respectiva clasă, dar, totodată apare şi riscul unor probleme sporite datorită definiţiilor multiple. Acest tip de moştenire poartă numele de moştenire multiplă.

Interfeţe şi pachete

Moştenirea simpla permisă de Java, chiar dacă oferă o claritate deosebită în ceea ce priveşte implementarea claselor, atunci când o anumită metodă sau comportament trebuie să fie copiate în cadrul a diferite ramuri ale ierarhiei claselor se dovedeşte a impune o serie de limitări. Rezolvarea acestei probleme este posibilă prin utilizarea unui nou concept interfeţele. Acestea colectează numele metodelor într-un singur loc permiţând adăugarea acestor metode ca un grup pentru clasele ce au nevoie de ele. Interfeţele conţin numai numele metodelor şi eventual argumentele, fără a conţine şi definiţiile.

Chiar dacă o clasă Java dispune de o singură superclasă, aceasta poate implementa un număr oricât de mare de interfețe. O clasă preia metode intermediul interfeței. Dacă două clase diferite implementează aceeaşi interfață, ceea ce înseamnă că acestea răspund la aceleaşi metode, acest lucru nu înseamnă că şi răspunsul va fi identic.

Termen Nou
O interfață este o colecție de nume de metode, fără definiții, ce pot fi adăugate la clase pentru a furniza un comportament suplimentar neinclus în cadrul definiției clasei sau in definițiile superclasei sau superclaselor anterioare.

Pachetele constituie un mod de grupare a unor clase înrudite sau interfețe într-o singură bibliotecă sau colecție. Acestea constituie grupuri modulare de clase, disponibile acolo unde este nevoie de ele sau utilizate pentru a elimina conflictele potențiale dintre numele claselor din diferite grupuri.

Principalele elemente caracteristice ale pachetelor sunt:

- Clasele bibliotecii Java sunt incluse în cadrul pachetului numit `java`. Aceste clase sunt disponibile în orice implementare Java
- Toate clasele Java sunt disponibile numai în `java.lang` (pachetul de bază din cadrul Kitului Java). Pentru a utiliza aceste clase şi în alte implementări, va trebui ca respectivul pachet să fie importat sau referit explicit.
- Referirea la o clasă din interiorul unui pachet se realizează prin numele pachetului separat de un punct (`.`) urmat de numele clasei. Spre exemplu folosirea clasei `Color` , conținute în pachetul `awt` (`awt` de la Abstract Windowing Toolkit), din cadrul pachetului `java` , se realizează prin următoarea specificare `java.awt.Color`.

Crearea unei Subclase

Un exemplu de creeare a unei subclase va fi realizat utilizând primul exemplu de applet Java realizat, cunoscutul HelloJava. Toate appletele sunt subclase ale clasei `Applet` (parte a pachetului `java.applet`).

În cadrul exemplului următor va fi creeat un applet HelloJavaAgain, care va permite afişarea unui şir de caractere cu un font diferit şi o culoare diferită. Declarația următoare permite construcția unei clase, din superclasa Applet, subpachetul, applet, pachetul java, clasă accesibilă tuturor claselor din cadrul clasei utilizator care va fi creeată.

```
public class HelloAgainApplet extends java.applet.Applet {

}
```

În acest mod a fost creeată o subclasă numită `HelloAgainApplet`. Cuvântul cheie `extends` `urmat de pachetul, clasa java.applet` indică faptul că această clasă este o subclasă a

clasei `Applet`. Deoarece clasa este inclusă în pachetul `java.applet` trebuie specificat numele respectivei clase în mod explicit.

Referitor la cuvântul cheie public, pe lângă accesibilitatea pe care o permite tuturor claselor ce intră în componența aplicației curente, acest cuvânt devine obligatoriu în cazul creerii de applete.

Pentru creearea unei noi clase, aceasta va trebui să dispună de noi elemente (cum ar fi: atribute, metode), față de superclasa pe care o moşteneşte. Noile metode sau atribute vor fi inserate între acoladele care urmează declarației de clasă.

În cadrul acestei aplicații va fi adăugată o variabilă de instanță – obiectul `Font` :

```
Font f = new Font("TimesRoman", Font.BOLD, 36);
```

Această nouă variabilă de instanță este parte componentă a pachetului `java.awt` . Particularizarea obiectului `Font` în acest caz este fontul Times Roman , bold, cu înălțimea de 36 de puncte. Această variabilă de instanță, mai precis această instanță va implica ca toate font-urile utilizate în cadrul metodelor următoare să fie de tip f.

Aşa cum s-a descries, anterior, evoluția appletelor, o metodă standard inclusă în clasa applet este metoda `paint()` care permite afişarea propriu-zisă a applet-ului pe ecran. Rescrierea metodei `paint()`, va indica applet-ului ce să deseneze pe ecran:

```
public void paint(Graphics g) {
    g.setFont(f);
    g.setColor(Color.red);
    g.drawString("Hello Java din nou!", 5, 40);
}
```

Se cuvin a fi realizate două observații în legătură cu metoda `paint()` :

Prima observație este ca metoda este declarată publică. Chiar dacă metoda din cadrul clasei applet este declarată inițial publică, la reînscriere aceasta va trebui declarată din nou publică, în caz contrar apărând o eroare de compilare.

Cea de-a doua observație se referă la argumentul metodei, reprezentat de o instanță a clasei `Graphics`. Clasa `Graphics` oferă o platformă independentă pentru utilizarea fonturilor, culorilor, metodelor şi comportamentelor pentru trasarea liniilor şi diverselor forme.

În cadrul metodei `paint()` se declară trei elemente:

- Fontul implicit utilizat de metodă va fi oferit de variabila de instanță `f`.
- Culoarea de desenare implicită va fi o instanță a clasei `Color`, mai precis culoarea roşie - red.
- Cel de-al treilea element este trimiterea spre afişarea pe ecran a şirului de caractere `"Hello Java din nou!"` începând cu poziția `x` = `5`, respectiv `y` = `25`.

Totodată se cuvine a remarca faptul că obiectele `Graphics` şi `Font` aparţin unui pachet care nu este inclus în cadrul pachetului implicit `java.lang` sau la clasa definită ca superclasă (-`java.applet.Applet` -), în definiţia clasei curente.

Soluţionarea acestei probleme se face prin indicarea întregului pachet care conţine clasele respective, fie prin indicarea claselor numai care vor fi importate din cadrul pachetului. Se reaminteşte că importarea unui pachet durează mult mai mult timp, încărcând memoria calculatorului cu elemente inutile, care nu vor fi utilizate în cadrul aplicaţiei curente. Clasele ce vor trebui importate sunt `Graphics`, `Font` şi `Color`, făcâd parte din pachetul `java.awt`:

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
```

Observaţie

Importarea unui pachet public se realizează prin utilizarea unui asterisk (`*`) în locul specificaţiei de nume de clasă. În exemplu anterior:

```
import java.awt.*;
```

În aceste condiţii forma finală a appletului `HelloJavaAgain.java` va avea următoarea structură:

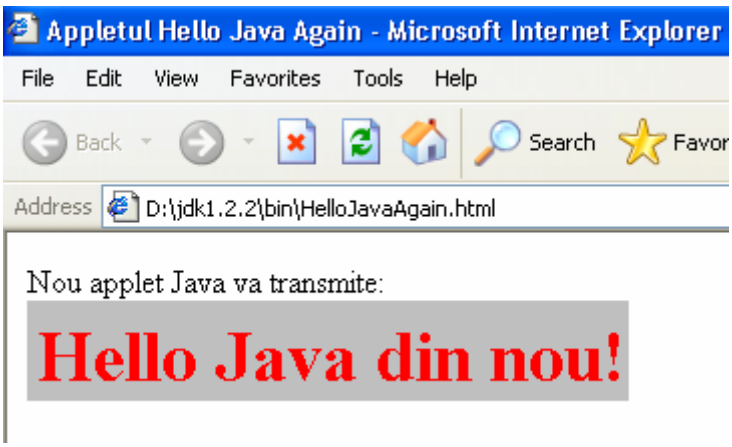
```
 1:import java.awt.Graphics;
 2:import java.awt.Font;
 3:import java.awt.Color;
 4:
 5:public class HelloJavaAgain extends java.applet.Applet {
 6:
 7:  Font f = new Font("TimesRoman",Font.BOLD,36);
 8:
 9:  public void paint(Graphics g) {
10:    g.setFont(f);
11:    g.setColor(Color.red);
12:    g.drawString("Hello Java din nou!", 5, 40);
13:  }
14:}
```

Fişierul HTML care apelează `<APPLET>` -ul va avea următoarea formă:

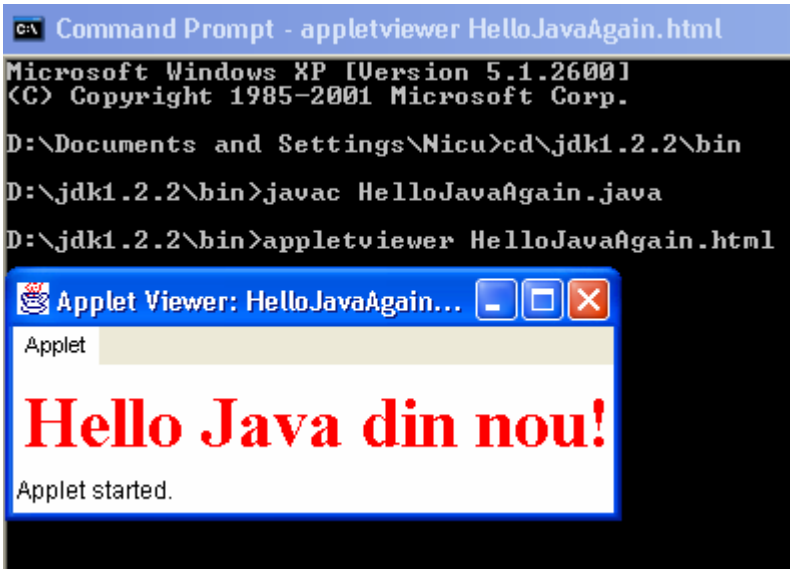
```
<HTML>
<HEAD>
<TITLE>Appletul Hello Java Again</TITLE>
</HEAD>
<BODY>
<P>Nou applet Java va transmite:
```

```
<BR><APPLET CODE="HelloJavaAgain.class" WIDTH=300 HEIGHT=50>
</APPLET>
</BODY>
</HTML>
```

Atenție atât fişierul class, cât şi fişierul HTML vor trebui localizați în cadrul aceluiaşi director. Salvați fişierul HTML cu numele `HelloJavaAgain.html` şi încărcați-l într-un browser compatibil Java. Rezultatul va fi următorul:



sau utilizând appletviewer:



Breviar

Elementele de bază ale programării orientate pe obiecte sunt enumerate în continuare:

clasă: – un model pentru obiecte, care conține variabile şi metode care reprezintă atribute şi comportamente. Clasele moştenesc variabile şi metode de la alte clase.

Metoda unei clase: Metodă definită în cadrul unei clase. Aceasta operează în cadrul clasei, putând fi apelată prin intermediul clasei sau prin intermediul unor instanțe.

Variabilele unei clase: Acest tip de variabilă aparține unei clase şi tuturor instanțelor acestei clase, fiind memorată în cadrul clasei.
Instanță: sau exemplu. Are aceeaşi semnificație cu obiect. Fiecare obiect este o instanță a unei clase.
Metoda unei instanțe: O metodă definită în cadrul unei clase, care operează cu instanțele respectivei clase: Metoda unei instanțe este denumită, în mod usual, metodă.

Variabilele unei instanțe: O variabilă care caracterizează o instanță şi a cărei valoare este memorată în cadrul instanței.

Interfață: O colecție de comportamente abstracte care pot fi implementate de către anumite clase.

Obiect: O instanță concretă a unei clase. Mai mult obiecte care sunt instanțe ale aceleiaşi clase au acces la aceleaşi metode, cu observația că dispun de diferite valori pentru variabilele de instanță.

Pachet: O colecție de clase şi interfețe. Clasele din cadrul altor pachete decât `java.lang` trebuie specificate în mod explicit prin numele lor sau importate.

Subclasă: O clasă situată inferior în cadrul ierarhiei ereditare, având ca părinte o superclasă. Creearea unei noi clase, poartă numerele de subclasare.

Superclasă: O clasă situată superior, în cadrul arborelui de moştenire, oferind subclaselor moştenitoare metodele sale, respective variabilele sale.

Concluzii

1: ***Metodele sunt de fapt funcții definite în interiorul unei clase. Cu toate acestea ele poartă numele de metode şi nu de funcții deoarece în cadrul limbajului Java, metodele acționaeză numai în interiorul claselor, în timp ce funcțiile acționează în afara claselor.***

2: ***În cadrul programării Java, se operează cu obiecte. Anumite attribute sau comportamente, fiind legate de caracterizarea globală a unei categorii de obiecte, devin elemente constituiente ale clasei caracteristice. Pentru caracterizarea unei instanțe se vor specifica attribute, respective metode specifice.***