1. Obiectele si incapsularea (dupa R. Baldwin - *http://www.developer.com/java/article.php/935351*)

1.1. Introducere

In continuare vor fi descrise aspectele esentiale (proprietatile *necesare* si *semnificative*) ale programarii orientate spre obiecte - POO (folosind Java). Altfel spus, va fi prezentata *esenta* POO (folosind Java).

Cele trei *concepte esentiale* ale POO sunt:

- **Incapsularea** (*encapsulation*) – de care ne vom ocupa acum,
- Mostenirea (*inheritance*) si
- Polimorfismul (*polymorphism*).

1.2. Ce este un program orientat spre obiecte?

Un **program** orientat spre obiecte (OO) consta dintr-un **grup de obiecte care coopereaza**, **comunicand prin** trimiteri de **mesaje**, pentru a indeplini un **obiectiv comun**. Fiecare obiect are o **responsabilitate** clara.

1.3. Ce este un object?

Un obiect este o **constructie software** care ***incapsuleaza*** (regrupeaza, protejeaza, si ofera controlul accesului) **date**, impreuna cu **abilitatea de a utiliza sau modifica aceste data**, intr-o **entitate software**.

1.4. Ce este incapsularea?

Incapsularea este principiul conform caruia o entitate software (in particular un obiect) trebuie sa aiba interfata complet separata de implementare. Toate datele si codul implementarii trebuie sa fie complet ascunse (*hidden*) in spatele interfetei.

Ideea este ca dupa crearea unei interfete (metodele publice ale unei clase), cat timp interfata ramane consistenta, aplicatia poate interactiona cu obiectele. Acest lucru ramane adevarat chiar daca rescriem in intregime codul scris intr-o metoda data.

1.5. O analogie cu lumea reala

Conceptele abstracte, cum ar fi cele de obiect sau incapsulare, pot fi adesea intelese prin comparatia cu analogii din lumea reala. O astfel de analogie, destul de buna desi imperfecta, este **sistemul radio al unei masini**.

Astfel se va putea stabili **o legatura intre programarea orientata spre obiecte** (POO) **si lumea reala**. Sistemul radio pentru masina va ilustra si **va permite discutarea diverselor aspecte ale obiectelor software**.

1.6. Abilitatea de a stoca date (informatii care formeaza starea interna)

Radioul unei masini are probabil abilitatea de a stoca date, si de a permite utilizarea si modificarea acelor date dupa dorinta. Totusi, datele pot fi utilizate si modificate doar prin utilizarea unei interfete cu utilizatorul uman, furnizata de producatorul radioului.

Datele sunt probabil stocate in radioul masinii intr-o lista de frecvente (sa presupunem ca **5**) care corespund statiilor de radio favorite ale utilizatorului.

1.7. Utilizarea datelor stocate (accesul la starea interna)

Radioul ofera un mecanism (interfata cu utilizatorul uman) care permite utilizarea datelor stocate in interior.

Cand este apasat un buton selector de frecventa, radioul se acordeaza automat pe frecventa corespunzatoare butonului. In acest caz, obiectul utilizator uman trimite un mesaj catre radio cerandu-i sa execute o anumita actiune.

Daca in prealabil a fost stocata o frecventa favorita in locatia de stocare corespunzatoare acelui buton, apasarea butonului *(trimiterea mesajului)* va duce la redarea in difuzoarele radioului a programului curent al statiei radio de pe acea frecventa.

Daca nu a fost stocata o frecventa favorita in prealabil in locatia de stocare corespunzatoare acelui buton, probabil ca difuzoarele radioului vor produce un huruit. Asta nu inseamna ca obiectul radio nu a raspuns corect la mesaj, ci doar ca raspunsul sau s-a bazat pe date gresite (pe lipsa datelor pe care trebuia sa le ofere utilizatorul).

Se observa faptul ca in cazul sistemului radio al masinii, utilizarea datelor stocate se face doar prin intermediul interfetei cu utilizatorul (utilizatorul nu are acces direct la frecventele stocate).

1.8. Modificarea datelor stocate (schimbarea starii interne)

Interfata cu utilizatorul uman face posibila si stocarea sau modificarea celor 6 valori ale frecventelor stocate in interior. O posibila procedura pentru stocarea unei valori este urmatoarea:

- se acordeaza manual radioul pe frecventa dorita,
- se apasa unul dintre butoane si se tine apasat pentru cateva secunde.

Cand radioul scoate un sunet (beep), utilizatorul stie ca noua valoare pentru frecventa a fost stocata in locatia de stocare care corespunde acelui buton.

In acest caz, utilizatorul trimite un mesaj catre obiectul radio prin care ii *cere sa isi schimbe starea*. Sunetul scos de radio poate fi interpretat ca o *valoare returnata* de obiectul radio pentru a indica indeplinirea sarcinii cerute.

Se poate spune ca *obiectul si-a schimbat starea* atunci cand a fost modificata (cel putin) una dintre valorile stocate in interior.

Putem spune si ca atunci cand un *raspunde la un mesaj*, un obiect:

- fie *indeplineste o sarcina*,
- fie *isi schimba starea interna* (prin modificarea valorilor informatiilor stocate),
- fie *returneaza o valoare* (care *da informatii despre starea sa interna*),
- fie o realizeaza *o combinatie* a celor mai de sus.

1.9. Indeplinirea unei sarcini

Atunci cand este apasat un buton (*trimis un mesaj*), obiectul radio se va acorda automat pe frecventa corespunzatoare acelui buton.

1.10. Persistenta

Abilitatea unui obiect (precum radioul unei masini) de a stoca valori si a-si aminti apoi acele valori (asa cum radioul poate stoca si isi poate reaminti o lista de statii favorite) se numeste *persistenta*.

1.11. Starea

Starea unui obiect la un moment dat este determinata de ansamblul valorilor stocate in acel obiect. In cazul nostru, chiar daca avem doua radiouri identice, starea unui obiect radio la un moment dat poate fi diferita de starea celuilalt obiect radio.

Totusi, este posibil ca doua radiouri identice sa contina aceeasi lista de frecvente in acelasi timp. Chiar si in acest caz, in care starile celor doua obiecte sunt identice, ele raman doua obiecte separate si distincte.

1.12. Trimiterea unui mesaj

O persoana care „vorbeste in limbaj orientat spe obiecte” (*OOP-speak*) ar putea spune ca apasarea unui buton selector de frecventa de pe panoul frontal al radioului *trimite un mesaj* catre obiectul radio, cerandu-i sa *indeplineasca o anumita sarcina* (*acordarea pe o anumita statie radio*).

Acea persoana ar putea sa spuna si ca stocarea unei noi frecvente corespunzatoare unui anumit buton determina *trimiterea unui mesaj* catre obiectul radio, cerandu-i sa *isi schimbe starea*.

1.13. Invocarea (apelul) unei metode

Limbajul Java e putin mai specific decat „limbajul orientarii spe obiecte”. In limbaj Java se poate spune ca apasarea unui buton selector de frecventa de pe panoul frontal al radioului *invoca o metoda* a obiectului radio (*asupra obiectului radio*). Comportamentul obtinut prin *executia metodei invocate* este *indeplinirea unei sarcini*.

De asemenea, putem spune ca stocarea unei noi frecvente corespunzatoare unui anumit buton determina invocarea unei metode *setter* (*mutator*, de stabilire/modificare a starii) a obiectului radio.

1.14. Comportamentul

In plus fata de *stare*, obiectele au si *comportament* (*behavior*). Comportamentul complet al unui obiect este determinat prin combinarea (inlantuirea) comportamentelor metodelor individuale ale acelui obiect.

De exemplu, unul dintre comportamentele expuse (*exhibited*) de obiectul radio este abilitatea de a *reda* (*play*) programul statiei radio de la o anumita frecventa. Atunci cand frecventa este selectata prin apasarea unui buton selector, radioul stie cum sa translateze undele radio de pe acea frecventa in unde audio compatibile cu gama auditiva, si sa trimita acele unde audio prin difuzoare.

Astfel, obiectul radio se comporta intr-un mod specific ca raspuns la un mesaj care ii cere sa se acordeze pe o anumita frecventa.

1.15. De unde vin obiectele?

Pentru a produce in masa sisteme radio de masina, cineva trebuie sa creeze mai intai un set de planuri (*blueprints* - schite, modele, tipare) pentru obiectele radio. De indata ce planurile sunt disponibile, pot fi produse milioane de sisteme radio aproape identice.

1.16. O clasa defineste un set de planuri

Acelasi lucru este valabil si pentru obiectele software. Pentru a crea un obiect software in Java, este necesar sa fie definit mai intai un plan (un set de planuri).

In Java acel plan (sau set de planuri) este numit *clasa* (*class*).

Clasa este definita de programatorul Java. De indata ce definitia clasei este disponibila, pot fi produse milioane de obiecte aproape identice.

1.17. O instanta a unei clase

Despre un sistem radio nou, proaspat achizitionat de la o fabrica producatoare de sisteme radio pentru masini, am putea spune ca este un *exemplu* de obiect produs pe baza setului de planuri utilizat pentru a defini sistemul radio.

In cazul obiectelor software, spunem ca un obiect este creat ca *exemplu* sau *instanta* (*instance*) a unei clasei.

1.18. Instantierea (crearea) unui obiect

Crearea unui obiect software pe baza planurilor oferite de clasa caruia ii apartine se numeste *instantiere*. Clasa si obiectul reprezentate in mediul BlueJ:

clasa Java: 

obiect Java: 

1.19. Exemple de cod Java

Este momentul pentru un prim cod Java.

Presupunand ca avem acces la definitia clasei, sunt mai multe moduri diferite prin care poate fi creat un obiect in Java. Cea mai des intalnita cale este utilizarea unei sintaxe de genul urmator.

```
Radio myObjRef = new Radio();
```

1.20. Ce inseamna asta?

Din punct de vedere tehnic, expresia din partea dreapta a semnului egal:

```
new Radio();
```

aplica operatorul **new** unui *constructor* al clasei numite **Radio** pentru a obtine *alocarea si initializarea memoriei* necesare unui nou obiect, alfel spus *crearea* unui nou obiect.

Deocamdata este de ajuns sa spunem faptul ca un constructor este un bloc de cod care face parte din definitia clasei, care are acelasi nume cu clasa, si care *ofera ajutor in momentul crearii unui nou obiect* (prin nume - care specifica clasa, si prin parametri - care specifica *valorile necesare initializarii starii* obiectului).

1.21. O referinta la un obiect

Expresia din partea dreapta a egalului *returneaza* o *referinta* catre noul obiect (in nici un caz nu returneaza direct un obiect!). Acea referinta este singura cale prin care obiectul nou creat poate fi accesat si utilizat.

1.22. Ce se poate face cu o referinta?

Referinta poate fi apoi utilizata pentru a *se trimite mesaje* catre noul obiect (*invocand metodele* care apartin noului obiect, asa cum sunt ele *definite in clasa* obiectului).

1.23. Salvarea referintei

Pentru a putea utiliza referinta ulterior (iar prin intermediul acesteia obiectul nou creat), este necesara salvarea ei.

Expresia din partea stanga a semnului egal

```
Radio myObjRef
```

declara o *variabila* de *tip referinta la un obiect al clasei* **Radio** (pe scurt, o *referinta de tip* **Radio**) numita **myObjRef**.

Deoarece acest tip de variabila este utilizata pentru a stoca o referinta la un obiect, o putem numi *variabila referinta*.

De aceea *clasele*, care sunt *tipuri utilizate pentru declararea acestor variabile referinta*, sunt numite si *tipuri referinta*.

1.24. Ce inseamna asta?

Declararea unei variabile creaza spatiul de memorie necesar acelei variabile (in acest caz spatiul de memorie necesar referintei, in nici un caz nu spatiul de memorie necesar obiectului, care este alocat de catre expresia din partea dreapta a semnului egal).

Valorile pot fi apoi stocate in acel spatiu de memorie si accesate ulterior folosind numele dat variabilei in momentul declararii (in cazul nostru **myObjRef**).

1.25. Atribuirea valorilor

Semnul egal duce la atribuirea (salvarea) valorii referintei returnate de expresia din partea dreapta in variabila referinta numita **myObjRef** (creata de expresia din partea stanga).

1.26. Alocarea memoriei

De indata ce s-a terminat executia liniei de cod de mai sus, in memorie au fost alocate si populate (initializate cu valori) doua segmente diferite si distincte:

- un segment de memorie (probabil mai mare) a fost alocat (de catre expresia din partea dreapta) pentru a stoca obiectul propriu-zis. Acest segment de memorie a fost populat conform planurilor continute in definitia clasei numite **Radio**.

- un alt segment de memorie (relativ mic) a fost ulterior alocat (de catre expresia din partea stanga) pentru variabila referinta numita **myObjRef**, care contine referinta catre obiect.

1.27. Invocarea unei metode a(supra) obiectului

Presupunand ca definitia clasei **Radio** defineste o metoda (al carei scop este sa simuleze comportamentul obtinut ca raspuns la apasarea unui buton selector de frecventa de pe panoul frontal al radioului) cu urmatorul format (numit *semnatura*):

```
public void playStation(int stationNumber)
```

1.28. Ce inseamna asta?

In contextul obiectului nostru radio, acest format inseamna ca executia metodei numite **playStation()** va duce la selectarea pentru a fi redata a programului statiei radio identificata prin valoarea intreaga pasata sub numele **stationNumber**.

1.29. Cuvintele cheie public si void

Tipul returnat **void** inseamna ca metoda nu returneaza nici o valoare.

Modificatorul (calificatorul) **public** inseamna ca butonul poate fi apasat de oricine se afla in masina si care il poate ajunge la el.

1.30. Semnatura metodei

Continuand expunerea terminologiei (jargonului) orientarii spre obiecte, se poate spune ca linia de cod anterioara constituie semnatura metodei, desi unii atuori considera ca semnatura metodei este urmatoarea (numita de alti autori amprenta – *fingerprint* sau *footprint*):

```
playStation(int stationNumber)
```

1.31. Invocarea metodei

Listingul urmator adauga o linie de cod in care este invocata metoda numita **playStation**.

```
Radio myObjRef = new Radio();
myObjRef.playStation(3);
```

1.32. Sintaxa invocarii metodei

A doua linie de cod (*boldface*) e un exemplu de sintaxa folosita pentru a trimite un mesaj catre un obiect Java, altfel spus pentru a *invoca o medota* a(supra) acelui obiect.

1.33. Reunirea numelui metodei obiectului cu referinta catre obiect

Sintaxa necesara invocarii unei metode a(supra) obiectului Java reuneste numele metodei cu referinta catre obiect, folosind operatorul punct (*dot* sau *period*).

1.34. Apasarea unui buton radio

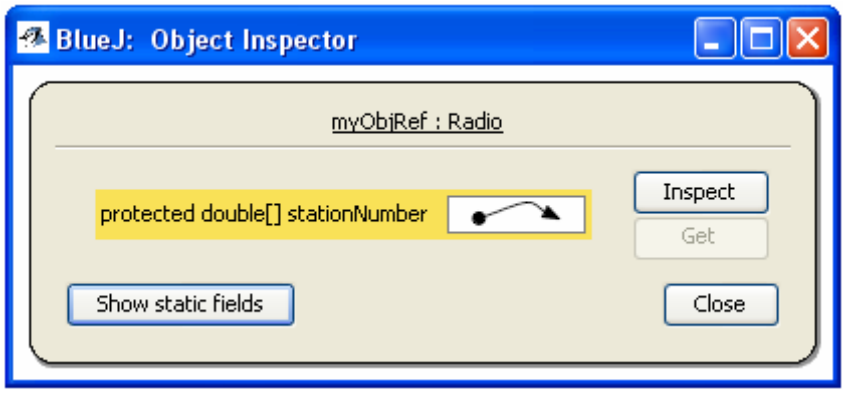
Conform discutiei de pana acum, pasarea valorii 3 catre metoda invocata, va conduce la simularea apasarii butonului cu numarul 3 de pe panoul frontal al radioului masinii (al patrulea buton, avand in vedere ca numerele butoanelor vor fi 0, 1, 2, 3, 4).

1.35. Inspectarea continutului obiectului myObjRef in BlueJ

Mediul de invatare/dezvoltare Java numit **BlueJ** permite inspectarea vizuala:

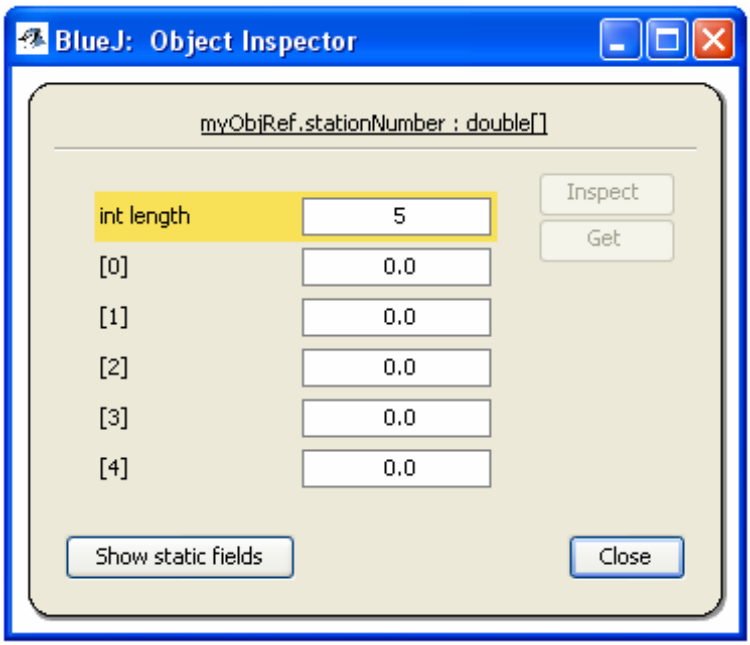
- a declaratiilor campurilor unui obiect (care formeaza starea interna a obiectului) cat si

- a continuturilor campurilor obiectelor, in mod recursiv, permitand vizualizarea valorilor campurilor obiectului interne a obiectelor (adica a stari interne a obiectului).



Se poate observa ca variabila referinta de tip **double[]**, numita **stationNumber**, contine o valoare referinta.

1.36. Inspectarea continutului campului (*field*) stationNumber in BlueJ



Se poate observa acum ca variabila referinta numita **stationNumber** contine o valoare care refera un tablou de 5 valori de tip **double** (al carui tip este **double[]**).

2. Clasele

(dupa R. Baldwin - *http://www.developer.com/java/article.php/943981*)

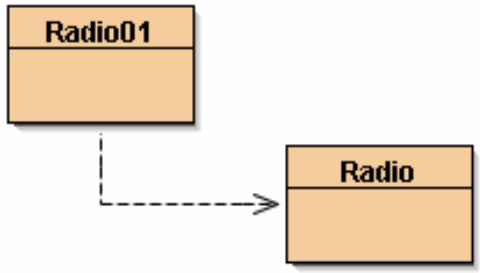
2.1. Ce este o clasa?

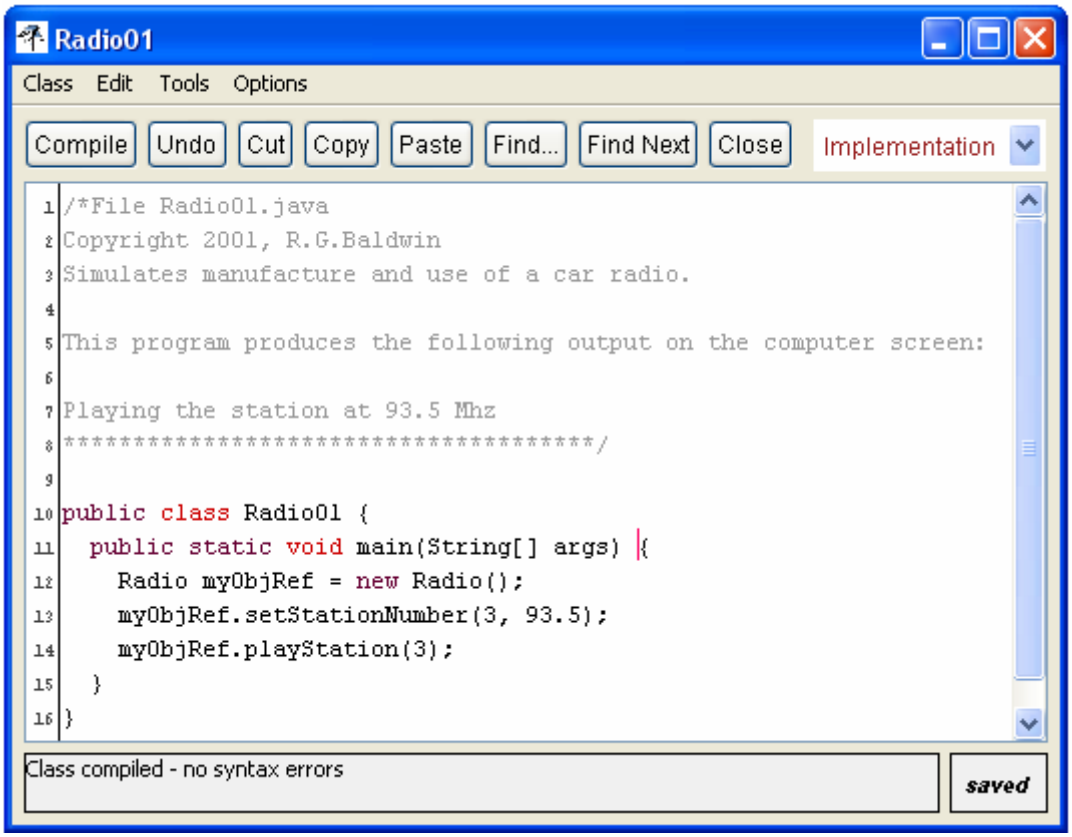
O clasa este un plan folosit pentru crearea mai multor obiecte. Definitia unei clase este asemanatoare cu planurile dupa care pot fi produse milioane de sisteme radio pentru masini aproape identice.

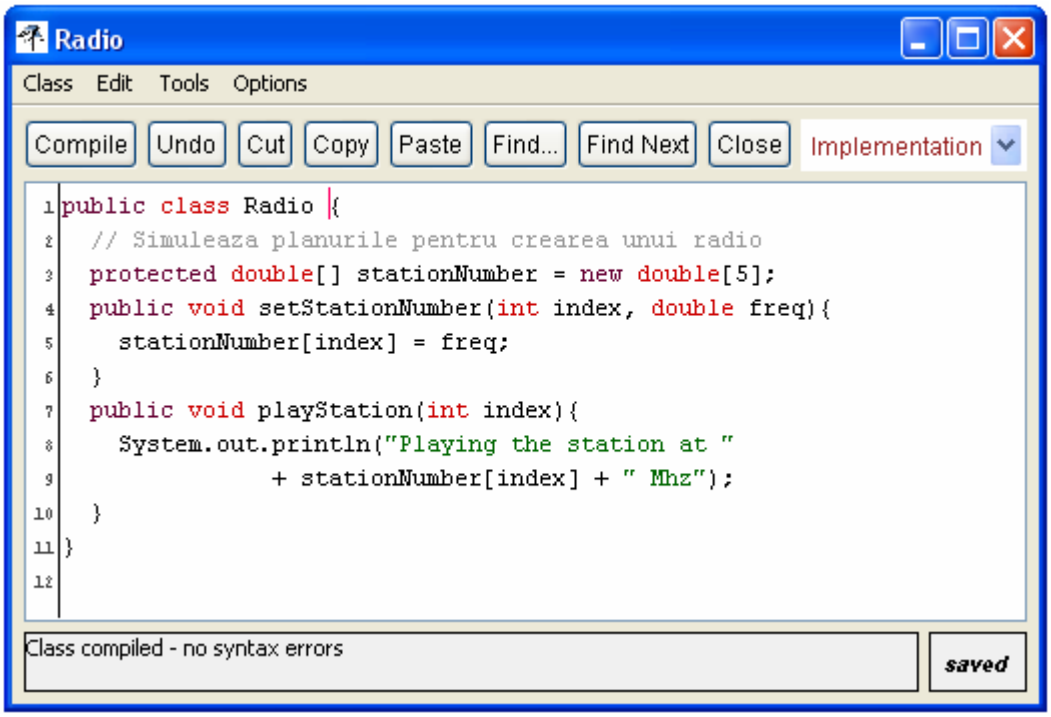
2.2. Un program Java simplu

Pentru a incepe cat mai eficient, si ca suport pentru conceptele introduse in continuare, este util ca in aceasta prezentare sa fie oferit si discutat un program Java simplu.

Codul Java urmator reprezinta o aplicatie simpla care simuleaza crearea si utilizarea unui sistem radio pentru masina. Acest program contine definitiile a doua clase. Reprezentarea lor in mediul BlueJ:







2.3. Clasa numita Radio01

Una dintre definitiile de clase, numita **Radio01**, este prezentata in intregime in continuare.

```
public class Radio01{
  public static void main(String[] args){
    Radio myObjRef = new Radio();
    myObjRef.setStationNumber(3, 93.5);
    myObjRef.playStation(3);
  }
}
```

Clasa numita **Radio01** consta doar dintr-o metoda **main()**. Metoda **main()** a unei aplicatii Java este executata de sistemul de executie Java (numit JVM, de la *Java Virtual Machine*) atunci cand este executata aplicatia. Astfel, metoda **main()** este punctul de intrare (*entry point*, *driver*) sau metoda principala a aplicatiei.

2.4. Clasa principala (*driver* sau de testare)

Prima linie din codul metodei **main()** simuleaza constructia unui obiect radio si utilizarea lui de catre un utilizator uman.

2.5. Constructia unui obiect Radio

Codul aplica operatorul **new** constructorului clasei **Radio**, ceea ce duce la crearea unui nou obiect pe baza planurilor specificate in clasa numita **Radio**.

```
Radio myObjRef = new Radio();
```

2.6. Salvarea unei referinte catre obiectul Radio

Codul anterior declara o variabila referinta de tip **Radio** in care stocheaza valoarea referinta catre noul obiect.

2.7. Programarea butoanelor radio

Urmatoarea linie de cod simuleaza procesul asocierii (frecventei) unei statii radio particulare cu un anumit buton.

```
myObjRef.setStationNumber(3, 93.5);
```

Dupa cum am mai spus, in cazul unui sistem radio real acest lucru poate fi realizat prin acordul manual pe statia radio dorita urmat de tinerea butonului radio apasat pana cand acesta scoate un *beep*.

Lina de cod de mai sus realizeaza asocierea butonului cu o statie radio simulata prin invocarea unei metode numite **setStationNumber** asupra referintei obiectului de tip **Radio**. Aceasta invocare reprezinta un mesaj trimis catre obiect, prin care i se cere obiectului sa isi schimbe starea interna.

Parametrii pasati (argumentele) metodei conduc la asocierea butonului cu numarul 3 cu frecventa 93.5 MHz. Valoarea 93.5 este stocata intr-o variabila interna a obiectului (variabila membru sau variabila instanta) care reprezinta (simuleaza) butonul cu numarul 3.

2.8. Trimiterea unui mesaj catre obiect

In limbajul orientarii spre obiecte, in linia de mai sus este *trimis un mesaj* catre un biect de tip **Radio** object, prin care i se cere obiectului sa isi schimbe starea interna in concordanta cu valorile pasate ca parametri.

2.9. Apasarea unui buton al radioului

In final, codul urmator invoca o metoda numita **playStation()** a obiectului **Radio**, si ii paseaza o valoare intreaga 3 (numar de buton) ca parametru.

```
myObjRef.playStation(3);
```

2.10. Un alt mesaj

Acest cod *trimite un mesaj* catre obiect, cerandu-i sa realizeze o actiune (indeplineasca o sarcina). In acest caz, actiunea ceruta prin acest mesaj este:

- acordeaza-te pe frecventa anterior asociata cu butonul numarul 3,
- reda in difuzoare programul de pe statia radio pe care o gasesti pe acea frecventa.

2.11. Cum este simulatata redarea programului radio?

Acest program simplu nu reda de fapt un program radio, ci afiseaza un mesaj pe ecranul calculatorului, simuland astfel selectia si redarea programului de radio ales.

2.12. Clasa numita Radio

Definitia clasei pentru clasa **Radio01** este prezentata in intregime in continuare.

```
public class Radio{
  // Simuleaza planurile pentru crearea unui radio
  protected double[] stationNumber = new double[5];
  public void setStationNumber(int index, double freq){
    stationNumber[index] = freq;
  }
  public void playStation(int index){
    System.out.println("Playing the station at "
                + stationNumber[index] + " Mhz");
  }
}
```

Se poate observa ca acest cod nu contine nici un constructor explicit. Daca nu este definit un constructor atunci cand este definita clasa, o versiune implicita a constructorului este in mod automat (implicit) oferita. Acesta este si cazul nostru.

2.13. Planurile pentru un obiect

Codul de mai sus furnizeaza planurile dupa care poate fi construit un obiect care simuleaza un sistem radio real.

Un obiect instantiat (un obiect fiind o instanta a unei clase) dupa codul de mai sus simuleaza un sistem radio fizic.

2.14. O variabila instanta (variabila membru a obiectului)

Codul urmator declara si initializeaza ceea ce se numeste o *variabila instanta* (*variabila membru* a unui obiect, sau *camp* al unui obiect).

```
  protected double[] stationNumber = new double[5];
```

2.15. De ce se numeste variabila instanta?

Numele de *variabila instanta* vine de la faptul ca fiecare instanta a unei clase (fiecare obiect) are o astfel de *variabila membru* (sau *camp*, in Java).

Fiecare sistem radio produs dupa acelasi set de planuri are abilitatea de a asocia o frecventa cu fiecare buton selector de pe panoul frontal al radioului.

2.16. Variabilele de clasa – o paranteza

Trebuie remarcat faptul ca Java (ca si alte limbeje OO, printre care si C++) suporta ceea ce se numeste *variabila de clasa*, care este cu totul altceva decat *variabila instanta*.

Variabilele de clasa sunt partajate intre toate obiectele create dintr-o clasa data. Este ca si cum ele ar fi *membre ale unui nucleu comun obiectelor dintr-o anumita clasa*, nucleu *care tine de acea clasa*.

Altfel spus, oricate obiecte ar fi instantiate dintr-o anumita clasa, ele impart o singura copie a fiecarei *variabile de clasa*.

O posibila analogie intre variabilele de clasa si obiectele fizice (radiouri in cazul nostru) ar fi numarul de serie al fiecarui obiect radio produs dupa acelasi set de planuri. Se poate considera ca numarul de serie corespunzator ultimului obiect creat este stocat intr-o variabila a clasei, incrementat cu fiecare nou produs creat, si copiat intr-o variabila instanta a noului obiect. Astfel, variabila de clasa mentine permanent

ultimul numar de serie alocat, iar fiecare obiect isi are propriul numar de serie stocat in variabila instanta corespunzatoare.

Desi variabilele de clasa sunt simplu de utilizat in Java, ele nu fac parte dintre conceptele orientarii spre obiecte. Este bine sa se evite utilizarea acestor variabile.

2.17. Referinta catre un tablou de obiecte

Acum sa ne intoarcem la varabila numita **stationNumber**. Fara a intra in multe detalii, aceasta variabila este de asemenea o variabila referinta, care insa refera un *obiect tablou.*

Obiectul tablou incapsuleaza un tablou uni-dimensional cu 5 elemente de tip **double**. Indexurile elementelor tablourilor Java incep cu valoarea zero, astfel incat valorile index pentru acest tablou se intind de la 0 la 4 inclusiv.

2.18. Persistenta

Tabloul de obiecte este locul in care sunt stocate datele care asociaza frecventele statiilor radio cu butoanele fizice simulate.

Fiecare element din tablou corespunde unui buton selector de frecventa de pe panoul frontal al radioului. Astfel, sistemul radio simulat printr-un obiect al clasei **Radio** are 5 butoane selectoare de frecventa simulate.

Tabloul de obiecte exista atunci cand linia de cod de mai sus se termina de executat. Fiecare element al tabloului este in acel moment initializat cu valoarea 0.0 (valoarea zero, in virgula mobila, in dubla-precizie).

In memorie sunt alocate si populate doua segmente distincte: unul pentru a stoca tabloul propriu-zis, si altul alocat ulterior pentru variabila referinta numita **stationNumber**, care contine referinta catre tablou.

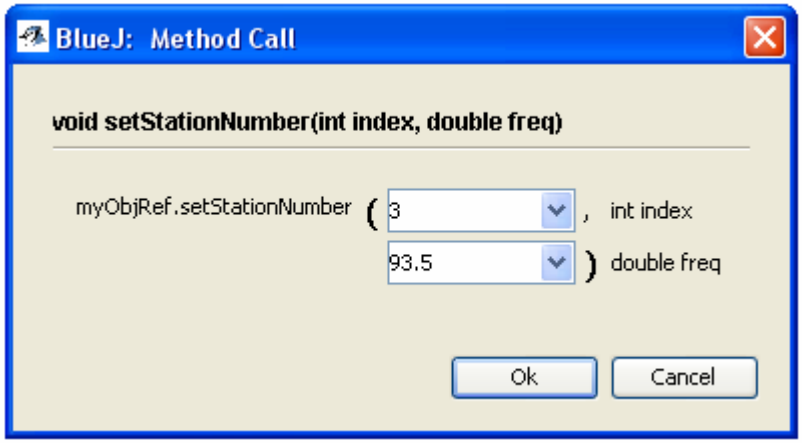
2.19. Asocierea unei statii radio cu un buton

In continuare este prezentata intreaga definitie a metodei **setStationNumber()**.

```
public void setStationNumber(int index, double freq){
  stationNumber[index] = freq;
}
```

Aceasta este metoda folosita pentru a simula comportamentul legat de asocierea unui buton cu o anumita statie radio.

2.20. Invocarea metodei setStationNumber() in BlueJ



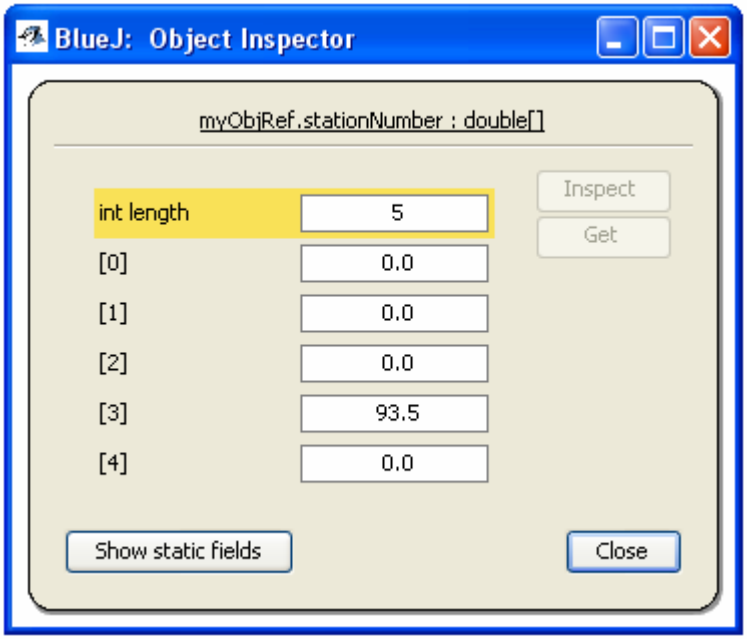
Metoda primeste doi parametri:

- un intreg care corespunde unui numar de buton
- o valoare frecventa care urmeaza a fi asociata cu butonul indicat.

2.21. Salvarea valorii frecventei

Codul metodei stocheaza valoarea frecventei intr-un element al tabloului discutat anterior. Numarul elementului este specificat de valoarea unui **index** care este plasat intre paranteze drepte in expresia de atribuire. Sintaxa este asemenatoare celei utilizate de majoritatea limbajelor de programare de nivel inalt.

2.22. Inspectarea continutului campului (*field*) stationNumber in BlueJ



2.23. Apasarea unui buton pentru a selecta o statie radio

In continuare este prezentata intreaga definitie a metodei **playStation()**.

Aceasta metoda simuleaza rezultatul apasarii de catre utilizator al unui buton de pe panoul frontal al radioului pentru a selecta o un anumit program pentru a fi redat.

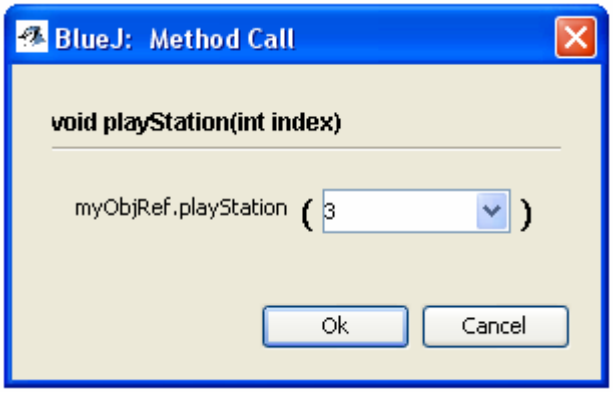
```
public void playStation(int index){
  System.out.println("Playing the station at "
            + stationNumber[index] + " Mhz");
}
```

2.24. Selectarea si redarea programului unei statii radio

Metoda primeste ca parametru o valoare index intreaga. Acest index corespunde numarului butonului apasat de utilizator.

Metoda simuleaza redarea unei statii radio prin extragerea valorii frecventei din tablou si afisarea acelei valori pe ecranul calculatorului, impreuna cu un text suplimentar.

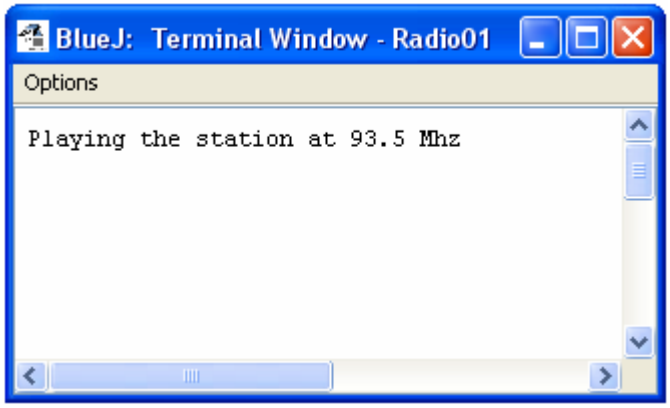
2.25. Invocarea metodei playStation() in BlueJ



Atunci cand este invocata de metoda **main()** a acestui program, metoda scrie pe ecranul calculatorului urmatorul mesaj:

```
Playing the station at 93.5 Mhz
```

2.26. Iesirea programului in BlueJ



2.27. Sintaxa definitiei unei clase

Sunt mai multe elemente care pot sa apara intr-o definitie de clasa, incluzand:

- variabile instanta
- variabile de clasa
- metode de instanta
- metode de clasa
- constructori
- blocuri de initializare statica
- clase interne (statice, non-statice, locale, anonime)

2.28. Aplicarea principiului KISS (*Keep it short and simple*)

Pentru a usura intelegerea conceptelor esentiale, pentru inceput vor fi ignorate pe cat posibil conceptele auxiliare sau exceptiile de la orientarea spre obiecte, adica variabilele de clasa, metodele de clasa, blocurile de initializare statica si clasele interne. De aceea **definitiile claselor vor contine urmatoarele elemente**:

- variabile instanta
- metode de instanta
- constructori

2.29. Constructorii

Un constructor este utilizat o singura data pe durata de viata a unui obiect. El participa la sarcina de a crea *(instantia)* si initializa obiectul. Dupa creare, starea si comportamentul unui obiect depind doar de variabilele si metodele de instanta.

3. Variante de program care nu lucreaza cu obiecte din noua clasa

In continuare sunt prezentate 3 variante de program care nu declara obiecte din noua clasa (Radio00x).

Pot fi urmariti astfel pasii prin care se poate ajunge de la un program nestructurat in mai multe functii la un program structurat procedural si apoi la un program care foloseste campuri declarate la nivelul clasei (ultima varianta fiind un pas catre programul orientat spre obiecte prezentat mai sus).

3.1. Tot codul intr-o metoda (functie)

Definitia unei clase Java numita **Radio001** in care tot codul este scris in metoda principala, **main()**:

```
public class Radio001 {                    // tot codul in metoda principala

  public static void main(String[] args) {
    double[] stationNumber = new double[5];
    int index = 3;
    double freq = 93.5;
    stationNumber[index] = freq;
    System.out.println("Playing the station at " +
                       stationNumber[index] + " Mhz");
  }
}
```

3.2. Orientare spre proceduri (delegare catre alte metode)

Definitia unei clase Java numita **Radio002** in care metoda principala **deleaga sarcini catre doua alte metode de clasa** (globale, declarate **static**):

```
public class Radio002 {                    // „orientare spre proceduri”

  public static void main(String[] args) {
    double[] stationNumber = new double[5];
    int index = 3;
    double freq = 93.5;
    setStationNumber(stationNumber, index, freq);   // delegare sarcina
    playStation(stationNumber, index);              // delegare sarcina
  }

  public static void setStationNumber(double[]stationNumber,
                                      int index, double freq){
    stationNumber[index] = freq;                    // sarcina delegata
  }

  public static void playStation(double[] stationNumber, int index){
    System.out.println("Playing the station at "
                + stationNumber[index] + " Mhz");   // sarcina delegata
  }
}
```

3.3. Orientare spre clase (utilizarea campurilor la nivel de clasa)

Definitia unei clase Java numita **Radio003** in care exista un **camp (atribut) de clasa** (global, declarat **static**) si o metoda principala care deleaga sarcini catre doua alte metode de clasa (declarate **static**):

```
public class Radio003 {                    // „orientare spre clase”

  private static double[] stationNumber = new double[5]; // camp (atribut)
                                                         // la nivel de clasa
  public static void main(String[] args) {
    int index = 3;
    double freq = 93.5;
    setStationNumber(index, freq);
    playStation(index);
  }

  private static void setStationNumber(int index, double freq){
    stationNumber[index] = freq;                        // utilizare camp
  }

  private static void playStation(int index){
    System.out.println("Playing the station at "
                + stationNumber[index] + " Mhz");       // utilizare camp
  }
}
```