**Caracteristicile POO**

* **Incapsularea –** regruparea in acelasi obiect datele si operatiile(metodele) care-i sunt specifice:

**Attribute –** datele incluse in obiect

**Metode –** functii din obiect

Incapsularea prevede izolarea detaliilor de implementare.

Interfata – in cazul obiectului,reprezinta ceea ce programatorul poate utiliza inafara clasei.

* **Abstractizarea –** identificarea pentru un ansamblu de obiecte caracteristicle lor commune si mecanismele lor.
* **Mostenirea**
* **Polimorfism –** utilizarea aceluiasi cod pentru tipuri de date diferite.

Obiectul are 2 nivele:

* **Intern** – ceea ce nu este accesibil din afara clasei si poate fi utilizat doar in ea.
* **Extern** – ceea ce poate fi utilizat din afara clasei de programator.

Atributele unui obiect nu trebuie sa fie vizibile din exterior!

**Clasa** – rezultatul incapsularii si abstractizarii

**Clasa** defineste un tip

O realizare particulara a unei clase se cheama **instanta** sau **obiect**

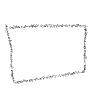
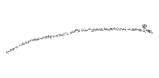
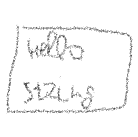
Un **obiect** este o variabila

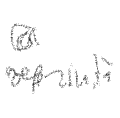
O variabila este doar o referinta la un obiect, daca e vorba de obiect.

De ex:

**String text = new String(“Hello”\_;**

Variabila text are si ea spatiul ei de memorie, ca si obiectul “Hello”, doar ca ea pastreaza in memoria alocata ei o referinta catre obiectul “Hello” si deci prin variabila text noi pur si simplu accesam zona de memorie a obiectului creat de tip String. De aceea trebuie sa specificam ce referinta poate avea variabila in memoria ei.





**null** – daca o variabila ia ca valoare null, inseamna ca ea nu are in zona ei de memorie nici-o referinta.

**In general, cand scriem String obj, noi spunem ca variabila obj poate stoca o referinta catre un obiect de tip String.**

**Inscrierea claselor in fisier/e**

* Daca avem mai multe clase intr-un fisier, de ex Hellow.java, Java(compilatorul) va crea cate un fisier separate pentru fiecare clasa, fisier de tip . class. De ex, daca in fisierul Hellow.java avem clasa Hellow si Bye, se vor crea inca 2 fisiere Hellow.class si Bye.class
* Putem avea si o clasa in fisier, dar Java(compilatrul) tot va crea un fisier .class pentru clasa din fisier
* Cand incercam sa executam o clasa, doar metoda main se executa. Daca ea nu exista si incercam sa executam clasa, primim eroare.

**Crearea obiectelor si claselor**

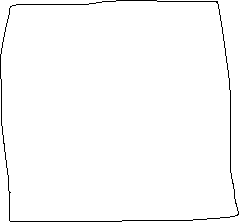
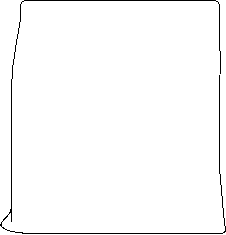
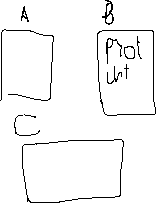
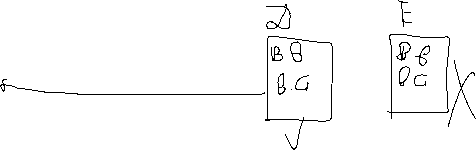
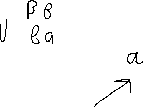
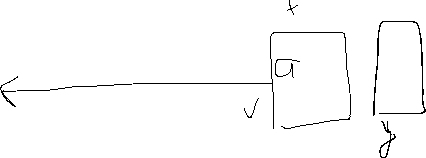
* Daca cream un obiect al carui clasa are membri neinitializati, compilatorul o sa-i initializeze automat, cu 0 pentru int, null pentru obiecte, false pentru boolean si 0.0 pentru double. Acest lucru are loc doar la crearea obiectelor cu membrii neinitializati, nu si in cadrul variabilelor neinitializate dintr-o metoda.
* Fiecare obiect creat are propriul bloc ce contine copii ale membrilor clasei. De aceea, fiecare obiect e diferit de celelalte.

**Public,Private,Protected and Default**

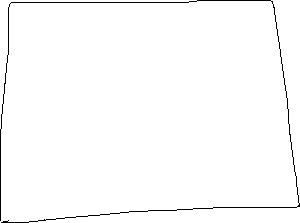
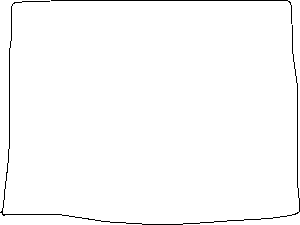
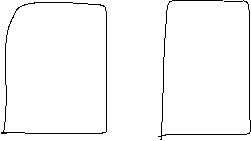
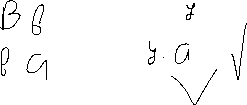
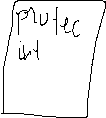
* **Private –** tot ce este private nu poate fi accesat din exteriorul clasei, si nici nu trebuie sa poata fi accesat. Private sunt:
* Atributele
* O parte din metodele ce nu trebuie accesate

Rolul lui este de a ne asigura ca nu putem modifica orice atribut din exterior, ci sa avem anumite metode care ne permit sa o facem cu mecanisme speciale, ce sa ne avertizeze daca sunt greseli sau exceptii. Daca utilizam de multe ori direct atributele din obiect si dorim apoi sa le modificam la nivel de clasa, va trebui sa stergem tot ce am scris cand le-am folosit si sa sriem iar, dar cu o metoda ar trebui doar sa modificam metoda. In general, atributele trebuie setate ca private ca programatorul sa nu trebuiasca sa le citeasca in clasa pentru a putea utiliza clasa.

* **Public –** tot ce poate fi accesat din afara clasei de oriunde si trebuie sa poata fi.
* **Default –** tot ce poate fi accesat doar in acelasi package
* **Protected(nerecomandat) –** tot ce poate fi accesat in acelasi package si in subclasele din alte packageuri. Daca o clasa din alt package mosteneste o clasa ce are membrii protected, ea va putea sa-l acceseze, va putea crea obiecte identice cu numele ei si sa-i acceseze, dar nu va putea sa-i accese prin obiecte de tip clasa parintre create-n ea!



Daca ne aflam in alt package si o clasa mosteneste de la alta clasa din alt package, ea va putea accesa membrii protected, dar daca vom crea obiecte de tip clasa de la care mosteneste, nu vom putea accesa cu aceste obiecte membrii protected chiar daca ne aflam intr-o clasa ce mosteneste acea clasa. Acest lucru e din cauza ca protected pur si simplu permite claselor copil din alte package sa acceseze membrii protected din clasa parinte, dar numai pentru ea si deci pentru obiecte de tip clasa copil! Nu si pentru obiecte de tip clasa parinte definite in clasa copil din alt package, caci pentru el package unde e clasa copil e strain!



**Conventii in Java**

* Denumirile de clase incep cu litere mari si orice cuvant separat e tot cu litera mare la inceput, ca in Python, de ex:

Class MyFirstClass

* Metodele incep cu litera mica si apoi fiecare cuvant nou cu litera mare, fara \_

Public void setSizeOfCircle()

* Variabielele incep cu litera mica si apoi fiecare cuvant nou cu litera mare, fara \_

int sizeOfCircle = 10;

* Metodele getter nu se folosesc pentru obiecte importante de clase, pentru a nu da acces programatorului la ele.
* Variabilele finale se scriu doar cu litere mari