1.Concepte importante:

- Abstratizare (Clase)
- Incapsulare
- Agregare
- Mostenire
- Polimorfism

2.Ce e POO?

Modelarea obiectelor din viata reala in programare

3.De ce contruim aplicatii folosind POO?

Clasificam in mod natural obiectele in tipuri.
Folosind POO in dezvoltarea software vom obtine aplicatii mai usor de intretinut, mai scalabile si mai usor de refolosit
In ce mod este diferita abordarea OPP de programarea clasica?
Aplicatia este descompusa in tipuri de date abstracte prin identificarea unor entitati
Un tip de data abstract contine date si defineste modul in care aceste date pot fi folosite

4.Programare Procedurala si Programare Orientata Obiect

- Unitatea de baza in programarea procedura este *functia*, in programarea orientata-obiect este *clasa.*
- Programarea procedurala se concentreaza pe construirea de functii, pe cand POO porneste de la definirea claselor si apoi defineste metode (functii) si membrii (variabile)
- Programarea procedurala separa datele de operatii, pe cand POO se concentreaza pe ambele aspecte.

5. Conceptele de "Clasa" si "Obiect"

"Clasa" este un sablon. Defineste variabile si metode pe care le suporta obiectul.

"Object" este o instanta a clasei. Fiecare obiect are la baza o clasa care defineste datele si comportamentul lor.

6. Membrii unei clase

O clasa are 3 tipuri de membrii:

- ***campuri***: variabile care arata starea unei clase sau a unui obiect
- ***metode***: functii care permit schimbarea starii obiectului sau accesul la valoarea unui camp
- ***clase si interfete imbricate***

7. Declarea membrilor unei clase

tip_data numeMembru; sau
tip_data numeMembru = val_initiala;

Tipuri de data
boolean, char, byte, short, int, long, float, double

Modificatori optionali
modificatori de acces (public, private, protected)
static
final

8.Initializarea membrilor

Nu neaparat cu constante

Daca nu sunt initializati, se considera valoarea implicita in functie de tip

Tip	**Valoare initiala**
boolean	false
char	'\u0000'
byte, short, int, long	0
float	+0.0f
double	+0.0
object reference	null

9. Declararea metodelor
O metoda are 2 parti
Header Costa din: modificatori (optionali), tip de data returnat, numele metodei, lista de parametrii si clauza throws (optional)
Corp
-O clasa poate avea mai mult de o metoda cu acelasi nume atat timp cat au lista de parametrii diferita.

10. Invocarea (apelarea) metodelor
Metodele sunt apelate ca operatii ale claselor sau obiectelor prin folosirea operatorului punct (.)
referinta.metoda(argumente)
Metode statice:
-In afara clasei: "referinta" poate fi sau un nume de clasa sau un obiect instantiat din clasa
-In cadrul unei clase: "referinta" poate fi omis
Metode nestatice:
-"referinta" trebuie sa fie un obiect instantiate

-Parametrii sunt trimisi prin valoare
-Cand parametrul este un obiect, se trimite referinta la obiectul respectiv, dar prin valoare (!!)

11. Metoda "main"

-Este punctul de intrare in program
-Masina virtuala localizeaza clasa in care se gaseste metoda main si incepe executia programului
- Alte metode sunt apelate din main implicit sau explicit
- Trebuie sa fie public, static si void

12. Modificatori de acces
- private: membrii pot fi accesati doar din interiorul clasei
-protected: membrii pot fi accesati in clasele din acelasi packet (vezi "Pachete in Java"), in subclase (vezi "Mostenirea claselor") sau in interiorul clasei

-public: membrii sunt accesibili atat in interiorul cat si in exteriorul clasei

13. Modificatorul static

-O singura copie a membrului exista, comuna tuturor obiectelor din clasa
-Membrii statici pot fi accesati direct in clasa
-Accesarea din afara clasei se poate face in 2 moduri:

Prin intermediul clasei

```
System.out.println(Masina.id);
```

Prin intermediul obiectului

```
System.out.println(m1.id);
```

-Important! Membrii ne-statici pot fi accesati din afara clasei doar prin intermediul obiectelor.

14. Modificatorul final

Dupa initializare valoarea campului nu mai poate fi modificata
Se foloseste, de obicei, pentru definirea constantelor
Campurile definite ca static final trebuie initializate atunci cand se defineste clasa
Campurile ne-statice si final trebuie initializate cand obiectul este initializat

15. Modificatori de clasa

public
Accesibila in mod public
Fara acest modificator clasa poate fi folosita doar in cadrul pachetului in care se afla
abstract
Clasa declarata este abstracta (vezi “Clase Abstracte”)
Nu poate fi instantiata
final
Nu poate fi mostenita (vezi “Mostenirea/Extinderea Claselor”)
Un fisier poate sa contina mai multe clase, dar doar o singura clasa poate sa fie publica. Numele fisierului trebuie sa fie acelasi cu numele clasei publice.

16.Constructori

-Instantierea obiectelor

Un obiect este instantiat prin folosirea instructiunii new
Masina virtuala aloca memorie pentru stocarea noului obiect
Daca nu exista suficient spatiu, modulul “Garbage Collector” va incerca sa elibereze memorie prin stergerea unor obiecte care nu mai sunt folosite. Daca nici in acest mod nu se obtine suficienta memorie, programul va arunca o exceptie de tip OutOfMemoryError (vezi “Exceptii”)
Memoria nu trebuie eliberata explicit

17.Constructori

Definitie

Un constructor reprezinta o modalitate de a initializa un obiect imediat dupa instantierea lui
Constructorii sunt metode (functii), dar nu intorc nici o valoare (!)
Un constructor are acelasi nume cu clasa (!)
Poate avea aceeasi modificatori de acces ca si alte metode
Daca un cosntructor nu este declarat explicit de catre programator, se considera ca exista un constructor fara argumente si care initializeaza campurile obiectului cu valorile implicite.

Supraincarcarea constructorilor
O clasa poate avea mai multi constructori atat timp cat lista de parametrii este diferita.

18. Folosirea lui this
-In cadrul unui constructor se poate folosi this pentru a apela un constructor deja existent. In mod obligatoriu trebuie sa fie prima instructiune din constructor.
-this poate fi folosit si ca referinta la obiectul curent.

19.Pachete Generalitati

Clasele pot fi grupate in pachete (en. package)
Clasele din libraria standard Java sunt grupate in pachete, cum ar fi java.lang si java.util
http://docs.oracle.com/javase/7/docs/api/index.html
Principalul motiv pentru folosirea pachetelor: unicitatea de nume a claselor
Clase cu acelasi nume pot sa fie grupate in pachete diferite

20.Importul claselor

- Exista doua metode de accesare a claselor **publice** dintr-un alt pachet:
  1. Prin folosirea numelui complet al pachetului inaintea numelui clasei:

     java.util. Date today = new **java.util.**Date( );
  2. Folosind intructiunea import in partea de sus a fisierelor ce contin codul sursa. In acest caz nu mai este nevoie de folosirea numelui pachetului inainte de numele clasei
  3. Se poate importa doar o clasa din pachet:

```
import java.util.Date;
Date today = new Date( );
```

  - Se pot importa toate clasele din pachet:

```
import java.util.*;
Date today = new Date( );
```

- Instructiunea *import* faciliteaza accesul doar la clasele din pachetul curent, nu si la clasele din sub-pachete
- In cazul metodelor statice este necesara folosirea numelui clasei
-

21.Crearea pachetelor

- Crearea unui pachet se realizeaza prin scrierea la inceputul fisierelor sursa a instructiunii:
  package numePachet;
- Instructiunea *package* actioneaza asupra intregului fisier sursa la inceputul caruia apare.

- Daca nu este specificat un anumit pachet, clasele unui fisier sursa vor face parte din pachetul implicit (care nu are nici un nume). In general, pachetul implicit este format din toate clasele si intefetele directorului curent de lucru.

22. Adaugarea unei clase intr-un pachet:

1. Se pune numele pachetului in partea de sus a fisierului sursa:

```
package com.hostname.corejava;

public class Employee {

    . . .

}
```

2. Se pune fisierul in directorul care se potriveste cu numele pachetului:
in fisierul “Employee.java” stocat in directorul “somePath/com/hostname/corejava/”

23.Incapsularea datelor

```
public class Body {
        public long idNum;
        public String name = “<unnamed>”;
        public Body orbits = null;
        public static long nextID = 0;
        Body( ) {
                idNum = nextID++;
        }
        Body(String bodyName, Body orbitsAround) {
                this( );
                name = bodyName;
                orbits = orbitsAround;
        }
}
```

Problema: toate campurile sunt “expuse”, putand fi folisite de “toata lumea”

- O versiune imbunatatita:

```
public class Body {
        private long idNum;
        private String name = “<unnamed>”;
        private Body orbits = null;
        private static long nextID = 0;
        Body( ) {
                idNum = nextID++;
        }
        Body(String bodyName, Body orbitsAround) {
                this( );
                name = bodyName;
                orbits = orbitsAround;
        }
}
```

Problema: cum pot fi accesate campurile?

O versiune imbunatatita cu **metode de accesare**:

```
public class Body {
        private long idNum;
        private String name = "<unnamed>";
        private Body orbits = null;
        private static long nextID = 0;
        Body( ) {
                idNum = nextID++; }
        Body(String bodyName, Body orbitsAround) {
                this( );
                name = bodyName;
                orbits = orbitsAround; }
        public long getID() {return idNum;}
        public String getName() {return name;};
        public Body getOrbits() {return orbits;}
}
```

Nota: Campurile *idNum*, *name* si *orbits* sunt read-only din afara clasei

24. Conventii de nume

- ***Pachete***: incep cu litera mica
  - E.g. java.util, java.net, java.io . . .
- ***Clase***: incep cu litara mare
  - E.g. File, Math . . .
- ***Variabile, nume de metode si campuri***: incep cu litera mica
  - E.g. x, out, abs . . .
- ***Constante***: cu litere mari
  - E.g. PI . . .
- ***Nume din mai multe cuvinte***: fiecare cuvant este scris cu litera mare in afara de primul care poate sa cu litera mare in cazul claselor sau cu litera mica in cazul metodelor
  - E.g. HelloWorldApp, getName . . .