Teorie POO

I. CLASSES & OBJECTS:

O clasă definește un set de obiecte care au aceeași structură și comportament. Un obiect este o instanță a unei clase.

Crearea obiectelor se face la rularea programului prin operatorul new și sunt alocate în heap. De ștergere se ocupă colectorul de deșeuri (se șterg singure după un mai pot fi accesate din program).

Constructor: "metodă" mai specială care:

- Are exact același nume ca și clasa
- Nu are tip returnat (nici măcar void)
- Se execută o singura dată la crearea obiectului
- Dacă nu declarăm explicit un constructor în clasă se generează automat unul fără nici un argument (constructor no-arg)

Ordinea inițializărilor:

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

Membrii statici: sunt specifici unei clase, nu unui obiect anume. Metodele statice NU se execută pe un obiect, this nu are sens în metode statice, nu poți accesa membrii instanță cu this.

O variabilă instanță final poate fi atribuită o singură o dată. Obligatoriu să fie inițializată până la sfârșitul oricărui constructor.

Identitatea unui obiect-este acea proprietate a unui obiect care îl distinge de oricare alt obiect. (referința/adresa de memorie unde e alocat) (Nu confunda numele de variabile referință cu identitatea unui obiect!)

Strea obiectului-reprezintă toate proprietățile obiectului plus valorile curente pentru fiecare din acele proprietăți.

Comportamentul obiectului-reprezintă cum anume acționează/reacționează acel obiect în termeni de schimbare a stării sale și de interacțiune cu alte obiecte. În esență, comportamentul unui obiect depinde de starea curentă, cât și de operația efectuatăpe obiect, iar unele operații pot altera starea sa.

Intergața-reprezintă setul de (declaratii de) operații (de obicei publice) pe care un client le poate efectua pe un obiect.

Încapsularea-compartimentarea elementelor unui obiect care formează structura și comportamentul său; servește la separarea interfeței de implementarea abstracțiunii. Impachetarea datelor și a metodelor (ce lucrează cu acele date) în clase, în combinație cu ascunderea implementării.

Semnătura unei metode-nume; numărul, ordinea și tipurile parametrilor formali;

Supraîncărcarea metodelor(Overloading)-metode cu același nume dar semnături diferite. Se poate și între constructori, numai din alți constructori. Dacă e necesar, ea trebuie să fie prima instructiune din constructor (apelul constructorului din superclasă).

Transmiterea parametrilor în JAVA se face exclusiv prin valoare!

II. CÂTEVA METODE IMPORTANTE:

În JAVA există 2 felori de egalitate:

a. FIZICĂ SAU DE IDENTITATE: folosim ==
b. DIN PUNCT DE VEDERE AL STĂRII:definim o metodă pentru acest lucru:equals();

Metoda equals trebuie să fie:reflexiv, simetric, tranzitiv.

Metoda toString( ):întoarce reprezentarea sub formă de șir de caractere a obiectului respectiv.

String-urile sunt obiecte imutabile(obiect ce nu își schimbă starea odată creat).

III. MOȘTENIREA:

Constructori: prima instrucțiune dintr-un cosntructor, e fie un apel la alt constructor al aceleiași clase, fie un apel la un constructor din superclasa directă. Dacă în superclasă există un constructor no-arg (default sau nu) compilatorul introduce automat un apel la acel constructor ca primă instrucțiune.

Ordinea inițializărilor:

1. Apel constructor
2. Apel cosntructor superclasă (*se repetă procedura și pentru superclasă)
3. Se realizează inițializările de la variabilele de stare în ordine textuală
4. Se execută corpul constructorului.

O clasă poate extinde cel mult o clasă. Nu există moștenire multiplă în JAVA.

Clasa Object este automat superclasă pentru orice clasă care nu extinde altceva. E implicit moștenită în mod direct/indirect.

Redefinirea metodelor(OVERRIDING): același nume, aceeași semnătură.

Metode abstracte: doar declarația metodei, implementate prin overriding în subclase.

Clasa abstractă:

- NU poate fi instanțiată
- Folosită pentru specificarea tipului și reutilizarea codului comun

Polimorfism: O variabilă referință declarată de un anumit tip (clasă) poate să refere obiecte ale acelui tip (clase) și a oricărui alt subtip (subclase) de-a sa.

Principiul Open-Closed: Entitățile software (ex. clase,metode) să fie deschise la extensii (să putem extinde, refolosi funcționalitatea lor), dar închise la modificări (să le putem extinde, dar fără să le modificăm codul).

Interfețe:

- NU putem factoriza codul comun diverselor implementări
- O clasă poate implementa mai multe interfețe
- O interfață poate extinde mai multe interfețe
- Seamană foarte mult cu o clasă pur abstractă(conține doar metode abstracte);

IV. EXCEPȚII:

O excepție este un eveniment ce apare la execuția unui program și care disturbă modul uzual în care programul se execută.

Detecția situației anormale și tratarea ei: Uzual în locuri diferite de program, în metode diferite. În locul detecției problemei nu avem uzual suficiente informații pentru a elimina problema (dacă am avea, am rezolva-o direct). În locul detecției nu mai putem continua procesarea.

Sunt obiecte. Clasa lor extinde o clasa specială din biblioteca Java (uzual clasa Exception).

Error și RuntimeException sunt excepții unchecked(fără verificare).Restul sunt checked(cu verificare). Excepțiile unchecked pot fi aruncate oricând, de oriunde.

Instrucțiunea throw poate fi văzută ca un return doar că diferă locul din program unde se revine(în catch).

Instrucțiunea throws la metode specifică ce fel de tipuri de exceptii poate emite metoda. Metoda va putea emite fără erori de compilare orice excepție de un tip/subtip de-a celor puse în clasă.

La overriding: Când o metodă X suprascrie o metodă Y dintr-o clasă de bază, throws-ul metodei X poate conține doar exceptii checked specificate în clauza throws de la Y(sau subtipuri).

Finally se execută oricum am ieși din try.

V. PACHETE:

Vizibilități:

a) Private-respectivul membru al clasei poate fi accesat doar în interiorul clasei
b) Public-respectivul membru al calsei poate fi accesat de oriunde
c) Protected-respectivul membru al clasei poate fi accesat din interiorul clasei, din subclasele sale (pe this) sau din același pachet (pe orice obiect).
d) Default(fără cuvânt cheie)-respectivul membru al clasei poate fi accesat doar din interiorul aceluiași pachet (de oriunde din interiorul pachetului).

O unitate de compilare trebuie să aibă același nume ca și clasa pe care o conține deci putem avea o singură clasă/unitate de compilare.

Numele complet calificat reprezintă numele pachetului.numele_clasei.

Import nu face altceva decât să spună compilatorului unde pe disc va găsi clasa de care avem nevoie.

VI. GENERICITATE:

Dacă avem 2 variabile cu parametri diferiți între < >, nu putem face egalitate(asignare(=)) între ei.

Dacă avem Wildcard limitat superior, adica <? Extends o_clasă>, atunci putem face egalitate cu un alt obiect de tip o_clasă sau subtip al acesteia. De asemenea când apelăm o metodă pe o variabilă declarată cu wildcard, putem folosi doar null ca parametru. Altfel e eroare de compilare.

VII. Reflexie:

getClass()-Întroarce clasa obiectului (NU A REFERINȚEI SALE)

isInstance()-Verifică daca obiectul parametru(NU REFERINȚA) este o instanță a clasei/sublcasei.

equals()-Face egalitate de identitate între clase.

Exemplu:

```
class A{}
class B extends A{}
class Main{
   public static void main(String args[]){
      A a=new B();
      B b=new B();
      Object o=new A();
      System.out.println(A.class.equals(a.getClass()));//false deoarece a este instanță a clasei B, iar equals compară atunci clasa A cu clasa B, care nu sunt egale ca identitate (chiar dacă B extinde A).
      System.out.println(A.class.isInstance(b));//true deoarece b este o instanță a clasei B, care este de asemenea un A, prin extindere.
      System.out.println(o.getClass().equals(Object.class));//false deoarece o.getClass() este clasa A care nu este egală cu clasas Object
      System.out.println(a.getClass().equals(b.getClass()));//true deoarece a.getClass() și b.getClass() sunt clasa B, iar clasa B este egală ca identitate cu clasa B.
      System.out.println(B.class.isInstance(a));//true deoarece a este instanță a clasei B.
   }
}
```