PROGRAMARE AVANSATĂ PE OBIECTE

Laborator 3

Maria Cristina Chițu
mariachitu11@gmail.com

Rezumat laborator

Relații între obiecte:

- Agregare si Compunere - **has a**
- Moștenire - **is a**

Upcasting

- convertire **copil** ⇒ **parinte**
- realizată automat

Downcasting

- convertire **parinte** ⇒**copil**
- trebuie facută explicit de către programator
- încercați să evitați folosirea operatorului **instanceof**

Suprascrierea

- înlocuirea functionalității metodei din clasa de bază în clasa derivată
- pastreaza numele și semnatura metodei

Supraincarcarea

- în interiorul clasei pot exista mai multe metode cu acelasi nume, cu condiția ca semnătura (tipul, argumentele) să fie diferită

super

- instanța clasei parinte
- amintiți-vă din laboratorul anterior că **this** se referă la instanța clasei curente

this

Cuvântul cheie this se referă la instanța curentă a clasei și poate fi folosit de metodele, care nu sunt statice, ale unei clase pentru a referi obiectul curent. Apelurile de funcții membru din interiorul unei funcții aparținând aceleiași clase se fac direct prin nume. Apelul de funcții aparținând unui alt obiect se face prefixând apelul de funcție cu numele obiectului. Situația este aceeași pentru datele membru.

Totuși, unele funcții pot trimite un parametru cu același nume ca și un câmp membru. În aceste situații, se folosește cuvântul cheie this pentru *dezambiguizare*, el prefixând denumirea câmpului când se dorește utilizarea acestuia. Acest lucru este necesar pentru că în Java este comportament default ca un nume de parametru să ascundă numele unui câmp.

În general, cuvântul cheie **this** este utilizat pentru:

- a se face diferenta între câmpuri ale obiectului curent și argumente care au același nume
- a pasa ca argument unei metode o referință către obiectul curent (vezi linia (1) din exemplul următor)
- a facilita apelarea constructorilor din alți constructori, evitându-se astfel replicarea unor bucăți de cod (vezi exemplul de la constructori)

Cuvântul-cheie "**final**". Obiecte immutable

Variabilele declarate cu atributul final pot fi inițializate **o singură dată**. Observăm că astfel unei variabile de tip referință care are atributul final îi poate fi asignată o singură valoare (variabila poate puncta către un singur obiect). O încercare nouă de asignare a unei astfel de variabile va avea ca efect generarea unei erori la compilare.

Cuvântul-cheie "**static**"

După cum am putut observa până acum, de fiecare dată când cream o instanță a unei clase, valorile câmpurilor din cadrul instanței sunt unice pentru aceasta și pot fi utilizate fără pericolul ca instaţierile următoare să le modifice în mod implicit.

Singleton Pattern

Pattern-ul Singleton este utilizat pentru a restricționa numărul de instanțieri ale unei clase la un singur obiect, deci reprezintă o metodă de a folosi o singură instanță a unui obiect în aplicație.

Aplicarea pattern-ului Singleton constă în implementarea unei metode ce permite crearea unei noi instanțe a clasei dacă aceasta nu există, și întoarcerea unei referințe către aceasta dacă există deja. În Java, pentru a asigura o singură instanțiere a clasei, constructorul trebuie să fie *private*, iar instanța să fie oferită printr-o metodă statică, publică.

String, StrinBuffer, StringBuilder

Factor / Class	String	StringBuffer	StringBuilder
Mutability	Immutable	Mutable	Mutable
Thread Safety	Not thread safe	Thread safe	Not thread safe
Performance	Very high	Moderate	Very high

Clase abstracte

Dorim să stabilim interfaţa comună pentru a putea crea funcţionalitate diferită pentru fiecare subtip și pentru a ști ce anume au clasele derivate în comun. O clasă cu caracteristicile enumerate mai sus se numește **abstractă**. Creăm o clasă abstractă atunci când dorim să:

- manipulăm un set de clase printr-o **interfaţă comună**
- **reutilizăm** o serie metode si membrii din această clasă in clasele derivate.

Metode abstracte

Pentru a exprima faptul că o metodă este abstractă (adică se declară doar interfaţa ei, nu și implementarea), Java folosește cuvântul cheie abstract:

abstract void f();

Clase abstracte în contextul moștenirii

O clasă care moștenește o clasă abstractă este ea însăși abstractă daca nu implementează **toate** metodele abstracte ale clasei de bază. Putem defini clase abstracte care moștenesc alte clase abstracte ș.a.m.d. O clasă care poate fi instanţiată (adică nu este abstractă) și care moștenește o clasă abstractă trebuie să implementeze (definească) toate metodele abstracte pe lanţul moștenirii (ale tuturor claselor abstracte care îi sunt "părinţi"). Este posibil să declarăm o **clasă abstractă fără** ca ea să aibă **metode abstracte**. Acest lucru este folositor când declarăm o clasă pentru care nu dorim instanţe (nu este corect conceptual să avem obiecte de tipul acelei clase, chiar dacă definiţia ei este completă).

Problema 1

Creati un pachet aferent exercitiului 1.

1. Folositi metodele indexOf, lastIndexOf, length, split pentru procesarea unui sir de caractere.
2. Identificati care este caracterul din sir care are cel mai mare numar de aparitii.
3. Inlocuiti fiecare aparitie a literei 'a' din String cu caracterul '*'. (hint: replace)
4. Eliminati toate caracterele spatiu ' ' (hint: trim)
5. Scrieti o metoda care sa concateneze 2 string-uri primite (s1, s2). Se va returna s1s2 doar daca s1 incepe cu ultimele 3 caractere din s2 (de ex: "ele fac" si "cafele"), altfel se va returna s2s1.

Problema 2

1. Studiati documentatia aferenta claselor **String, StrinBuffer, StringBuilder**.
2. Creati o clasa Main cu metoda main in care sa folositi 10 metode specifice claselor in cateva exemple de baza pentru procesarea string-urilor.

Problema 3

Create o clasa Circle intr-un pachet nou. Aceasta va contine:

- 2 campuri private : radius (double) si colour (String);
- Campurile vor avea valorile 1.0 si 'Red' ca default. (hint: constructor fara parametri → overloading)
- Creati un constructor cu 2 parametri si inca unul care seteaza doar raza, culoarea fiind una default
- Creati setter si getteri pentru membrii clasei (metode publice, ex: setRadius(double r), getRadius())
- Implementati o metoda getArea() pentru calcularea ariei.
- Suprascrieti metoda toString()
- Suprascrieti metoda equals(), verificati daca obiectele apartin aceleasi clase si daca au campurile cu valori egale
- Creati o metoda main intr-o clasa noua TestCircle, in care sa exemplificati definirea si instantierea obiectelor de tip Circle folosind cei 3 parametri. De asemenea, apelati metodele de set si get, getArea, equals si afisati datele corespunzatoare fiecarei instante.

Problema 4

Creati un pachet aferent exercitiului 4.

1. Creati o clasa abstracta Product. Adaugati cateva metode abstracte (2-3), de ex: roundPrice(), dar si cateva campuri member (variabile);
2. Creati clasele care sa extinda clasa Product: FoodProduct, CleaningProduct, FurnitureProduct, ClotheProduct;
3. Pentru fiecare clasa rescrieti metodele abstracte, dar si metoda toString();
4. Creati clasa Main cu metoda main in care sa testate functionalitatile implementate.