Moștenire, pachete proprii

Inheritance, user defined packages

Obiective:

- înțelegerea mecanismelor de moștenire
- definirea și utilizarea pachetelor

Objectives:

- understanding and applying the inheritance mechanisms
- defining and using packages

Moştenirea claselor

În limbajul de programare Java este permisă doar moştenirea simplă, adică o clasă poate fi derivată dintr-o singură clasă de bază. Cuvântul cheie folosit pentru a specifica moştenirea este *extends*.

Sintaxa:

```
class NumeClasaDerivata extends NumeClasaDeBaza{
        //implementare clasa derivată
}
```

Pentru a suplini neajunsurile moştenirii simple, Java pune la dispoziţie un pseudo-mecanism de moştenire multiplă prin intermediul interfeţelor: o clasă poate implementa oricâte interfeţe.

Sintaxa:

```
class NumeClasaDerivata extends NumeClasaDeBaza implements Int1, Int2, IntN{
        //implementare clasa derivată
}
```

O clasă derivată moşteneşte de la clasa de bază toţi membrii marcaţi cu specificatorii de vizibilitate *public* sau *protected*. Membrii *private* nu se propagă prin moştenire. Dacă însă clasa de bază are metode sau clase interioare care trec de moştenire şi care accesează anumiți membri *private*, atunci şi clasa derivată va putea accesa acei membri folosind metodele sau clasele interioare în discuţie.

Upcasting şi downcasting

Mecanismul de *casting* permite transformarea unui obiect dintr-un tip de date în altul (compatibil). Ierarhiile de moştenire a claselor generează tipuri compatibile.

```
class BicicletaDeTeren extends Bicicleta{
    //...
}
```

Mecanismul de instanţiere a clasei derivate sau a celei de bază este cel cunoscut:

```
Bicicleta b0 = new Bicicleta();
```

BicicletaDeTeren b1 = new BicicletaDeTeren();

Mecanismul de *upcasting* se referă la generalizarea tipului unui obiect dintr-o clasă derivată, tipul acestuia fiind transformat în cel al clasei de bază. Acest mecanism este implicit şi nu necesită forţări de tip de nici un fel.

```
// b1 este transformat în tipul lui b0 apoi se egalizează
b0 = b1;
```

sau

```
/* instanța anonimă a clasei derivate este transformată in tipul clasei de bază iar valoarea este asignată obiectului b2 */
    Bicicleta b2 = new BicicletaDeTeren();
```

Mecanismul de *downcasting* se referă la acordarea de facilități noi unui obiect dintr-o clasă de bază prin transformarea și egalizarea lui cu un obiect dintr-o clasă derivată. Acest mecanism merge doar folosind operatorul *cast ()* şi este restricţionat la clase care se afla în aceeaşi ierarhie de moştenire.

BicicletaDeTeren b3 = (BicicletaDeTeren)b0;

Suprascrierea metodelor unei clase de bază

Java permite redefinirea implementării metodelor moştenite dintr-o clasă de bază, cu păstrarea semnăturii (tip returnat, nume şi listă de parametri) acestora. Acest mecanism poartă numele de *suprascriere* (*overridding*).

Există o restricţie cu privire la specificatorul de vizibilitate al metodei suprascrise, şi anume: *o metodă nu poate fi suprascrisă pentru a deveni mai privată*.

Accesul la membrii clasei de bază

Este posibil ca dintr-o clasă derivată să se dorească accesarea membrilor super-clasei. Acest lucru se poate face folosind cuvântul cheie *super*, operatorul "." şi numele metodei care urmează să fie apelată.

Sunt cazuri în care se doreşte apelarea constructorului clasei de bază, din constructorul clasei derivate. Acest lucru este posibil folosind acelaşi cuvânt cheie, iar apelul trebuie să fie făcut înainte de orice altă instrucţiune din constructorul clasei derivate.

Moştenirea interfeţelor

Limbajul de programare Java permite moştenirea interfeţelor. Spre deosebire de clase, o interfaţă poate să moştenească oricâte interfeţe de bază. Cuvântul cheie care specifică moştenirea interfeţelor este identic cu cel al claselor, şi anume *extends*.

Sintaxa:

```
interface IDerivat extends IBaza1, Ibaza2 [, ...]{
    //declaraţii
}
```

Pachete Java

Un pachet Java este o colecţie de clase şi interfeţe grupate în general după funcţionalitate cu scopul de a beneficia de protecţia oferită de acest tip de grupare.
Un pachet Java se declară folosind cuvântul cheie *package*. Declararea unui pachet Java trebuie făcută înainte de orice altă instrucţiune din program (înainte şi de secţiunea de importări ale altor pachete şi clase din antetul programului).

Sintaxa:

package nume_pachet;

Pentru a putea fi folosite ca atare, fişierele sursă şi bytecode rezultate prin compilarea componentelor unui pachet trebuie stocate într-un subdirector care poartă numele pachetului. Astfel ele vor fi găsite de JVM în momentul importării în aplicaţii terţe (bineînţeles, cu respectarea tuturor regulilor referitoare la *classpath*.

Lucru individual

1. Definiți o interfață PiesaDeSah, ce definește prototipul funcției muta(). Creați clasele aferente tipurilor distincte de piese de sah și implementați metoda muta conform regulilor de mișcare a pieselor pe tabla de șah. Metoda are ca parametri de intrare poziția curentă a piesei și direcția de deplasare dată prin coordonate geografice (N,S,E,V,NE,NV, SE,SV) și returnează poziția finală. Atenție la piesele ce pot să se deplaseze mai mult de o căsută pe tabla de șah.
==========

2. Definiți o clasă Vehicul ce are ca și atribute numărul maxim de pasageri, culoarea și viteza maximă de deplasare. Extindeți clasa Vehicul în cadrul clasei VehiculMotorizat ce include atributele de poziționare geografică (coordonate GPS – creați o clasă pentru obiecte de tip localizare geografică -GeoLoc) și viteza de deplasare. VehicululMotorizat se poate deplasa dintr-un punct A într-un punct B (punctele sunt obiecte de tip Point ce au două atribute GeoLoc) prin intermediul metodei deplasează(Point B) și care returnează timpul necesar deplasării solicitate.
Creați o nouă clasă Avion ce extinde clasa VehiculMotorizat și care adaugă metodei deplasează(Point B) și atributul altitudine. Metoda va returna timpul necesar deplasării ținând cont de faptul că avionul se va deplasa pe arcul de cerc dat de punctele A, B și altitudinea maximă la care ajunge avionul (considerați că altitudinea e atinsă doar la jumătatea traseului de deplasare).
==========

3. Fie un pachet de clase și interfețe Java denumit dbInteraction ce permite interacțiunea cu o bază de date pe baza autentificării unui utilizator. Pachetul conține următoarele componente:
- o clasă ce definește obiecte de tip Person și care au atributele private nume, prenume, adresă de e-mail, userID și parolă, alături de metodele accesor și mutator corespunzătoare.
- o interfață pentru autentificare cu metodele createUser(), deleteUser() și login(). Metodele au ca parametru de intrare un obiect de tip Person si returnează un șir de variabile de tip Person. Pentru login() se definește în interfață o listă de utilizatori default: admin, dbAdmin și superUser.
- o clasă abstractă VerifyPerson ce extinde clasa Person și implementează metode de verificare a formatelor pentru nume, prenume, adresă de e-mail, userID și parolă după următoarele specificații:
    * numele și prenumele nu pot să conțină alte simboluri decât cele alfabetice
    * lungimea numelui și a prenumelui nu poate să fie mai mare de 50 de caractere
    * adresa de e-mail trebuie să fie de forma: [a-zA-Z._]@[a-zA-Z.].[a-zA-Z]{2-5}
- clasa abstractă definește, dar nu implementează metodele de verificare a userID și a parolei

Creați o aplicație ce interacționează cu acest pachet astfel:
- scrieți o clasă ce implementează interfata de autentificare și care actualizează lista de persoane conform cu acțiunea specificată.
- scrieți clasa care extinde clasa abstractă Verify și implementează metodele verificare a userID și a parolei după următoarele specificații:
    * userID poate să conțină doar litere, cifre și simbolul “.”
    * parola trebuie să aibă lungimea minimă de 8 caractere, cel puțin un caracter uppercase și un simbol non-alfanumeric
- scrieți o clasă de test ce verifică toate funcționalitățile aplicației.
==========

4. Definiti un pachet Java numit imageProcessor care contine o clasa numita MyImage. Clasa contine toate metodele necesare initializarii si modificarii valorilor dintr-o matrice de pixeli de dimensiune m x n. Fiecare pixel este o instanta a

unei alte clase numita Pixel (inclusa de asemenea in pachet) care contine 3 variabile intregi R, G si B care pot avea valori intre 0 si 255.
Clasa MyImage mai contine metode pentru:

- eliminarea pixelilor care au componentele RGB sub anumite valori primite ca parametri
- eliminarea componentelor R G sau B din toti pixelii
- transformarea valorilor pixelilor in tonuri de gri prin aplicarea formulei 0.21 R + 0.71 G + 0.07 B. Noile componente R G si B vor fi egale cu noua valoare calculata

Nota: fiecare operatiune este cronometrata.

Importati pachetul intr-o aplicatie Java care instantiaza clasa MyImage. Programul aloca valori aleatoare componentelor culorilor pixelilor. Aplicati metodele din clasa asupra obiectului creat. Afisati rezultatele si timpii de transformare.
==========

5. Dezvoltati o ierarhie de clase care definesc o serie de forme si un program care calculeaza cantitatea de vopsea necesara acoperirii diferitelor obiecte create. Ierarhia consta din clasa de baza *Shape* si clasele derivate - *Sphere*, *Rectangle* si *Cylinder*. Singurul atribut pe care il are clasa de baza este numele acesteia si singura metoda abstracta e cea care calculeaza aria.

Parcurgeti urmatoarele etape:

A. Scrieti clasa abstracta *Shape* cu urmatoarele componente:

- o variabila *shapeName* de tip *String*
- o metoda abstracta *area()*
- o metoda *toString()* care returneaza numele formei geometrice

B. Fisierul *Sphere.java* contine clasa *Sphere* derivata din *Shape*. O sfera are *radius* drept atribut privat intreg si implementeaza corpul metodei abstracte mostenite *area()*.

C. Definiti clasele similar *Rectangle* si *Cylinder* derivate de asemenea din clasa de baza *Shape*. Un dreptunghi este definit de *length* si *width* in timp ce un cilindru este caracterizat de *radius* si *height*.

Nota: Fiecare din clasele derivate suprascriu metoda mostenita *toString* si includ metode mutator si accesor specifice variabilelor proprii.

D. Fisierul *Paint.java* contine o interfata al carei unic membru este o constanta de tip float numita *paintPerSurfaceUnit*.
E. Fisierul *PaintThings.java* implementeaza interfata *Paint* si contine un program care calculeaza cantitatea de vopsea necesara acoperirii diferitelor forme create.
Instantiati trei forme: *deck* <- *Rectangle*, *bigBall* <- *Sphere* si *tank* <- *Cylinder*. Apelati metodele corespunzatoare pentru a seta atributele specific cu valori citite de la tastatura.
Calculati cantitatea de vopsea necesara fiecarui obiect.

Individual work

1. Define an interface called ChessPiece which defines the prototype for the method move(). Create the specific classes for each of the chess pieces and implement the move() method according to the rules of movement on the chess board. The method takes as input the chess piece's current location and the direction of the move given by the geographical coordinates (N, S, E, V, NE, NV, SE, SV), and returns the final position of the piece. Pay attention to the pieces which can move a different number of cells!!

==========

2. Define a class called Vehicle which has as attributes the maximum number of passengers, its color and the maximum speed. Extend the Vehicle class within the MotorizedVehicle class which also includes the geo-location attributes (GPS coordonates – create a class for this type of object GeoLoc) and the moving speed of the vehicle. MotorizedVehicle can move from point A to point B (the points are specified using a Point object which has two GeoLoc attributes) by using the move(Point B) method and which returns the total duration of the trip.

Create a new class called Airplane which extends the MotorizedVehicle class and which add the attribute altitude to the move(Point B) method. This method will return the time needed to take the trip by taking into account that the plane will travel on arc of a circle specified through points A and B and the maximum altitude reached by the plane (the maximum altitude is reached half-way between A and B).

==========

3. Consider a package of classes and interfaces called dbInteraction which enables the interaction with a database based on a user's authentication. The package includes the following components:

- a class which defines objects of type Person with the private attributes: name, surname, e-mail address, userID and password along with the getter and setter methods
- an interface for authentication with the methods createUser(), deleteUser() and login(). The methods take as intput a Person object and return an array of Person variables. For the login() method there is a default list of users: admin, dbAdmin and superUser.
- an abstract class VerifyPerson which extends the Person class and implements the mthods which check the formats of the name, surname and e-mail address with the following specs:
    * the name and surname ca only contain alphabetic characters
    * the length of the name and surname cannot be greater than 50 characters
    * the e-mail address should be formatted as: [a-zA-Z._]@[a-zA-Z.].[a-zA-Z]{2-5}
- the abstract class defines but does not implement the the methods which check the userID and the password

Create an application which interacts with this package like so:

- Write a class which implements the authentication interface and which updates the list of persons accordingly
- Write a class which extends the abstract class Verify and implements the check methods for userID and password according to the following specs:
    - The userID can only contain alfanumeric symbols plus the "." Symbol

o The password should be at least 8 characters long, with at lease one uppercase symbol and a non-alphanumeric symbol

- Write a Test class which tests if all the package's functionalities are correct.

==========

4. Define a Java package named imageProcessor which contains a class called MyImage. The class has all the necessary methods used for initializing and modifying the values from a m x n pixels matrix. Each pixel is an instance of another class named Pixel (also included in the package) which contains 3 integer variables R, G and B with possible values between 0 and 255.

The class MyImage defines methods for:

- cancelling the pixels that have the RGB values below some values received as parameters
- deleting the R G or B components from all the pixels
- transforming the pixels into grayscale tones by using the formula 0.21 R + 0.71 G + 0.07 B. The new R G and B components will be equal with this formula's results.

Note: each operation is timed.

Import the defined package into a Java application that creates a MyImage instance. The program generates randomly the values for the pixels' components. Apply the methods stored inside the class upon the created instance. Display the results and the necessary amount of time specific to each operation.

==========

5. In this lab exercise you will develop a class hierarchy of shapes and write a program that computes the amount of paint needed to paint different objects. The hierarchy will consist of a parent class Shape with three derived classes - Sphere, Rectangle, and Cylinder. For the purposes of this exercise, the only attribute a shape will have is a name and the method of interest will be one that computes the area of the shape (surface area in the case of three-dimensional shapes). Do the following.

A. Write an abstract class Shape with the following members:

- an instance variable shapeName of String type
- an abstract method area()
- a toString() method that returns the name of the shape

B. The file Sphere.java contains the Sphere class which is a descendant of Shape. A sphere has a radius as a private integer variable and implements the body of the inherited abstract method area().

C. Define similar classes for a rectangle and a cylinder. The classes Rectangle and Cylinder are also derived from the Shape class. A rectangle is defined by its length and width. A cylinder is defined by a radius and height.

Note: Each of the derived classes override the toString method and define specific mutator and accessor methods.

D. The file Paint.java contains an interface that has a float constant paintPerSurfaceUnit.

E. The file PaintThings.java implements the Paint interface and contains a program that computes the amount of paint needed to paint various shapes.

Instantiate the three shape objects: deck <- Rectangle, bigBall <- Sphere and tank <- Cylinder. Make the appropriate method calls to assign each object's specific attributes with data read from the keyboard. Compute the amount of paint needed for covering each created shape.