Laborator 1

Tehnologia Java

Tehnologia Java este alcatuita dintr-un conglomerat de concepte dintre care amintim:

- limbajul de programare Java – dezvoltat pe baza paradigmei obiect orientate;
- platforma de dezvoltare Java consituita din *masina virtuala Java – JVM* si setul de pachete de dezvoltare oragnizate in *Java Application Programming Interface.*

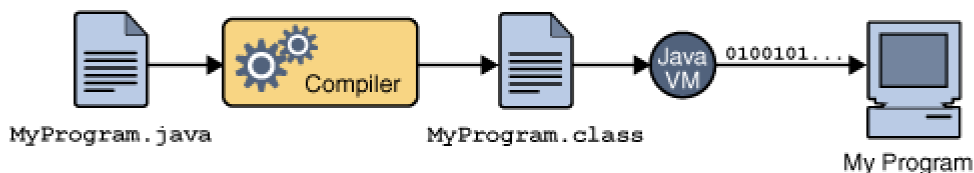
Limbajul de programare java reprezinta un limbaj de programare de nivel inalt definit de urmatoarele caracteristici esentiale:

- independet de arhitectura
- portabil
- multithreaded
- dinamic
- obiect orientat
- distribuit
- robust

Pentru o descriere mai detaliata a conceptelor precedente se poate consulta documentul *The Java Language Environment* realizat de James Gosling si Henry McGilton ce se poate gasi la adresa http://java.sun.com/docs/white/langenv/.

Un cod Java este scris inntr-un fisier cu extensia *.java* care va reprezenta codul sursa. Pentru asigura portabilitatea si indepenta fata de arhitectura si platforma fisierele sursa, care descriu aplicatia Java construita, sunt compilate utilizand compilatorul distributiei Java – *javac* – in fisiere intermediare ce contin bytecode (i.e. nu se obtine codul nativ ce poate fi executat direct pe procesorul sistemului). Aceste fisiere au extensia *.class* si descriu aplicatia intr-un limbaj masina specifica JVM. Rularea unei aplicatii impune astfel implicarea unei instante a JVM pe sistemul unde se doreste rulata aplicatia Java dezvoltata.

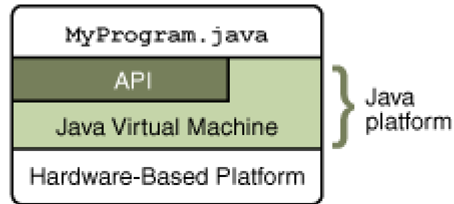
Figura de mai jos suprinde procesul de compilare si executie al unei aplicatii Java.



Masina virtuala Java este o aplicatie capabila sa interpreteze limbajul unui fisier *.class* si sa il transpunsa in limbaj corespunzator masinii unde este lansata o instanta a JVM. Masina virtuala java vine disponibila cu distributia JDK – *Java Development Kit*. Exista distributii JDK disponibile pentru fiecare OS important: MS Windows, Solaris OS, Mac OS, Linux/UNIX, AIX. Astfel acelasi fisier *.class* obtinut in urma compilarii pe un anume sistem poate fi apoi distribuit si executat oriunde exista o instanta a JVM (independent de arhitectura/platforma).

Plaforma Java reprezinta mediul de executie pentru aplicatiile Java si contine masina virtuala Java si un set de componente software pentru furnizarea diverselor capabilitati (e.g. networking, GUI,

formatare si prelucrare documente, etc.) necesare dezvoltarii aplicatiilor complexe. API-ul reprezinta un set de librarii de componente software organizate in ceea ce tehnologia Java denumeste pachet. Notiunea de pachete si organizare a unei aplicatii Java v-a fi discutata mai pe larg in sectiunile urmatoare.



Obiecte, Clase, Ierarhii

Obiect **–** reprezinta o abstractizare a tuturor entitatilor caracterizate de cel putin o *stare* si un *comportament.* Un obiect software este utilizat pentru a incapsula trasarturile caracteristice ale obiectelor din lumea reala si reprezinta entitatea elementara paradigmei obiect-orientate. Astfel obiectele software sunt utilizate in descrierea oricarui concept dorit. Notiunea de *stare* este incapsulata la nivelul unui obiect printr-un set de variabile denumite in Java *campuri – fields*. Notiunea de *comportament* este incapsultata intr-un obiect printr-un set de functii denumite in Java *metode – methods.*

Exemple de obiecte:

Obiect	State (Field variables)	Behaviour (Methods)
Notebook	*LCD, Keyboard, CPU, GPU, I/O Ports, Sound Card, Motherboard, WebCam*	*Powered-On, Idling, Transferring data, Running Java Apps, OS loading, Booting*
Teacher	*Knowledge, Affiliated students, Affiliated courses*	*Teaching, Resting, Grading, Yelling*

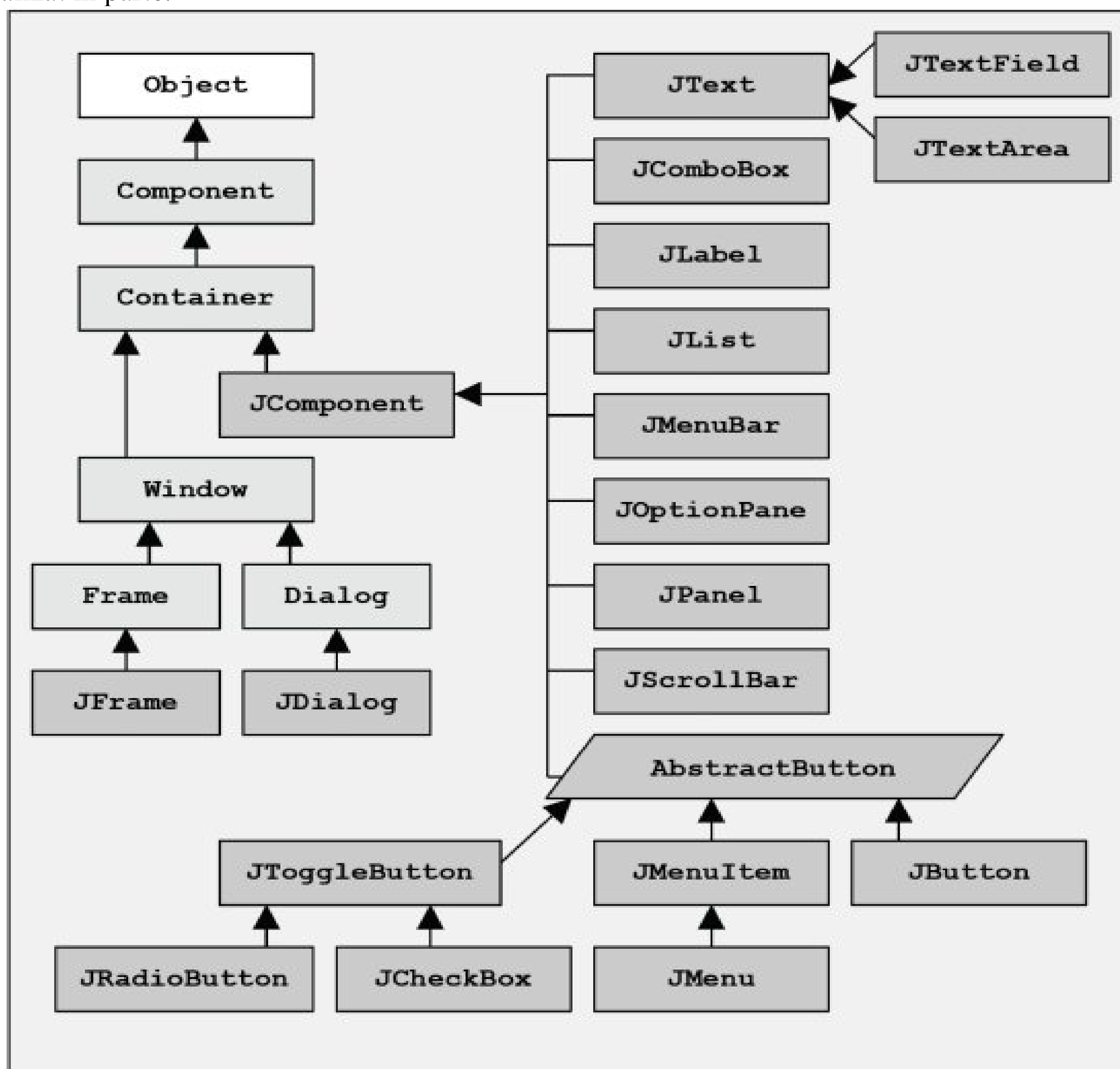
Organizarea unei aplicatii urmarind o structura orientata pe obiecte induce o serie de avantaje:

- modularitate
- incapsularea informatiei
- reutilizarea codului
- usurarea depanarii (debugging)

Clasa **–** Integreaza conceptul de tip de obiecte. Similar lumii reale multe obiecte sunt de acelasi tip (exista mai multe motociclete, stickdrives sau telefoane). Astfel o clasa de obiecte confera sablonul necesar generarii de noi obiecte. Formal un obiecte – e.g. motocicleta – reprezinta o instanta a unei clase de obiecte recunoscute ca fiind motociclete. In limbajul Java se vor crea mai intai tipurile de obiecte (i.e. se incapsuleaza notiunile de stare si comportament la nivel de clasa) iar apoi se genereaza unde si cand este cazul un obiect folosind tipul de obiecte. Detaliile acestei metodologii vor fi prezentate in sectiunile urmatoare unde se descriu particularitatile limbajului Java.

Ierarhii **–** Notiunea de ierarhie defineste unul din cele mai importante concepte din programarea obiect orientata si anume mostenirea. Prin aceasta relatie de mosternire se stabileste o structura arborescenta avand drept noduri clasele descrise. In conditiile in care mai multe tipuri de obiecte prezeinta anumite trasaturi caracteristice in comun atunci se poate realiza un tip general de baza mostenit de fiecare tip

specalizat in parte.



In figura de mai sus se prezinta o ierarhie de tipuri de obiecte Java, mai exact ierarhia catorva componente Swing care descriu tipuri de obiecte de interfata grafica. Astfel clasa Component reprezinta o specializare a clasei Object pe care o mosteneste, tipurile JButton, JToggleButton, JRadioButton mostenesc tipul abstract AbsractButton. Mostenirea este importanta deoarece permite definirea unui tip comun si a unui comportament comun intre tipuri de obiecte similare permitand astfel ca in momentul specializarii unei clase de baza (clasa de la care se mosteneste) codul adaugat sa fie minimal si sa descrie concret diferenta fata de clasa de baza.

EXERCITIUL 1: Identificati obiectele/tipurile de obiecte – i.e. variabile membre care descriu starea si metode membre care descriu comportamentul – necesare implementarii jocului “X si 0”?

Concepte de baza ale limbajului Java

Pentru studierea limbajului exemplele se vor dezvolta folosind NetBeans IDE[1].

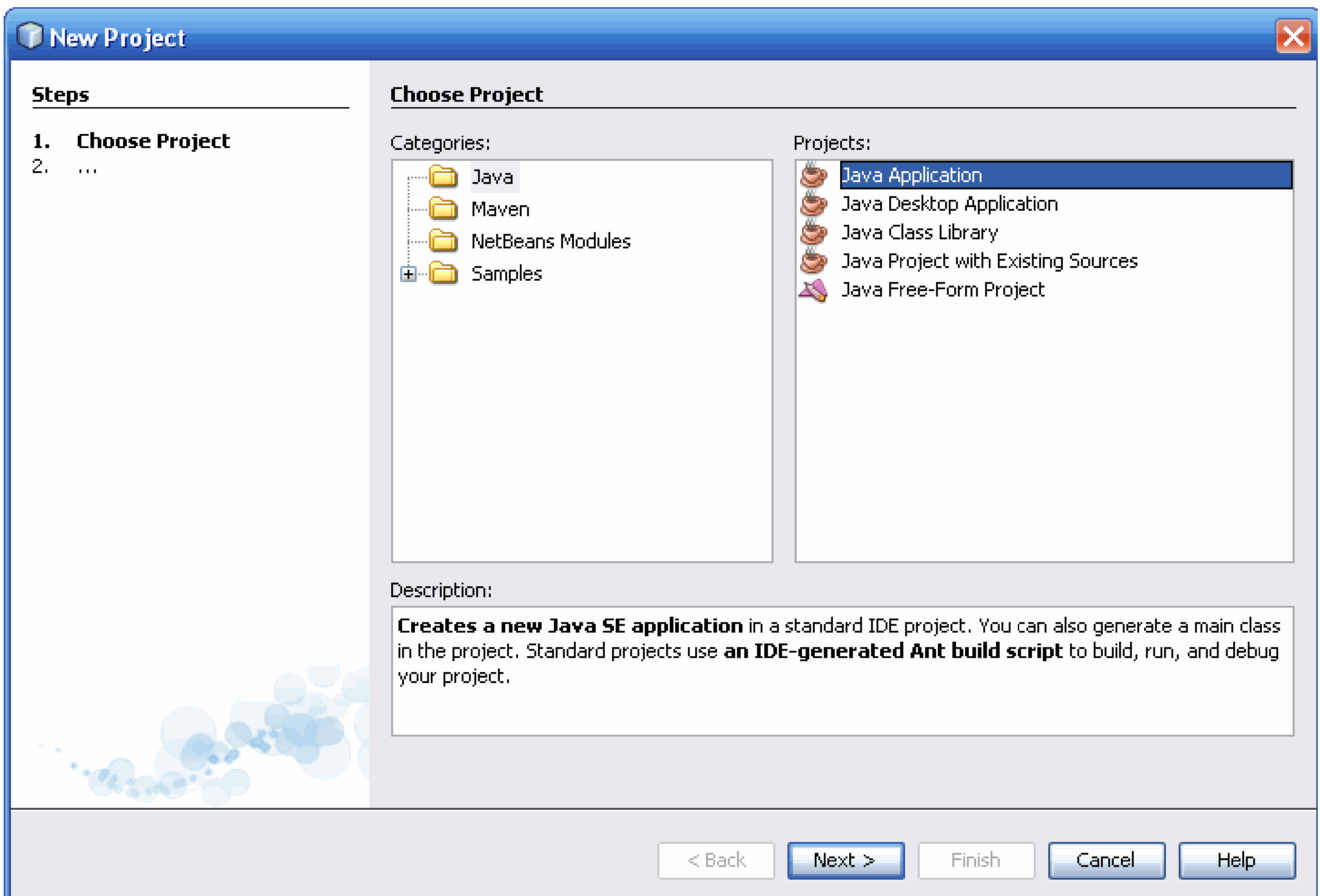
1 Integrated Development Environment

NetBeans - “Hello World”

Prerechizite: JDK 1.6
NetBeans IDE

Se va implementa ca prima aplicatie in mediul NetBeans aplicatia “Hello World” care afiseaza un simplu text pe monitorul device-ului cu care se lucreaza. Pentru aceasta se urmaresc pasii:

- se creaeaza un proiect IDE:
  - Se lanseaza in executie NetBeans IDE
  - Din meniul File se selecteaza optiunea New Project



  - Din fereastra aparauta se selecteaza din zona Categories optiunea General iar din zona Projects se alege Java Application
  - In urmatoarea fereastra se introduc numele proiectului: `Hello World Project`, locatia acestuia (se poate lasa setarea default), se bifeaza casuta Set as Main Project, se bifeaza casuta Create Main Class unde se introduce `helloworldproj.HelloWorldProj`
  - se apasa butonul Finish

Interfata GUI a NetBeans este separata intr-o serie de zone pentru prezentarea diverselor informatii si pentru a permite interactiunea cu diversele parti ale aplicatiei. Astfel in sectiunea Projects se poate naviga prin arborele de documente si fisiere ale proiectelor create, in sectiunea Navigator se poate naviga rapid intre elementele componente ale unei clase iar in sectiunea centrala se realizeaza

editarea documentelor selectate.

- se completeaza fisierul HelloWorldProj.java generat cu codul adecvat
  - prin selectia fisierului HelloWorldProj.java din sectiunea Projects se acceseaza in partea centrala editorul Java disponibil in mediul NetBeans. In acest edtior se completeaza metoda `main` astfel:
  - ```
    public static void main(String[] args) {
      System.out.println("Hello World");
    }
    ```
- se compileaza fisierul HelloWorld pentru a obtine un fisier .class
  - din bara de meniu a NetBeans se selecteaza meniul Run | Build Main Project
  - alternativ se apasa F11
- se lanseaza in executie aplicatia generata
  - din bara de meniu a NetBeans se selecteaza meniul Run | Run Main Project
  - alternativ se apasa F6

Variabile

Java defineste notiunea de obiect pe baza de campuri - *fields* – dar exista de asemeni si notiuena de variabila. Exista 4 tipuri de variabile:

- variabile membre – instance variables – sau campuri non-static; un obiect stocheaza starea in astfel de variabile declarate fara cuantificatorul static si sunt unice instantei obiect create (i.e. o variabila membra este asignata unei singure instante obiect)
- variabile la nivel de clasa – class variables – sau campuri static; reprezinta orice camp declarat cu cuantificatorul static si reprezinta o variabila partajata intre toate instantele obiect generate pe baza clasei respective
- variabile locale – sunt variabilele definite la nivel de metoda sau blocuri
- parametri – reprezinta variabilele primite ca argument de catre metodele obiectelor; este important deretinut ca parametrii sunt clasificati ca variabile si nu drept campuri ale clasei.

In exemplul urmator se creeaza clasa corespunzatoare unei table de joc clasice pentru “X si 0”:

```
public class ClassicX0GameBoard {
	// class variables which are also constants through adding final

	static final int ROWS = 3;
	static final int COLUMNS = 3;
	//1 instance variables
	int color;

	public ClassicX0GameBoard(int color /*parameter variable*/) {
		this.color = color;
	}
	public void setColor(int color) throws Exception {
		// local variables
		int upperLimit = 255;
		int lowerLimit = 0;

		if (color <= upperLimit && color >= lowerLimit) {
			this.color = color;
		} else {
			throw new Exception("Color doesn ot exist");
		}
	}
```
```

```java
        public static void main(String[] args) {
                //declaration
                int[] someRandomArray;
                someRandomArray = new int[3];

                //definition
                someRandomArray[0] = 13;
                someRandomArray[1] = 27;
                someRandomArray[2] = 5;
        }
}
```

Denumirea variabilelor urmareste o serie de conventii in Java care sunt detaliate la [aceasta adresa].

Tipurile primive ce se pot utiliza in Java sunt byte, short, int, long, float, double, boolean, char si sunt detaliate la [aceasta adresa].

Pe langa tipurile primitive Java permite formarea nativa de vectori. Lungimea unui vector este determinata la crearea acestuia (declarare) si nu mai poate fi modificata ulterior.

```java
//declaration
int[] someRandomArray;
someRandomArray = new int[3];

//definition
someRandomArray[0] = 13;
someRandomArray[1] = 27;
someRandomArray[2] = 5;
```

Operatori

Lista operatorilor Java este similara cu cea a altor limbaje de nivel inalt si este descris prin tabelul de mai jos, in ordinea precendetei.

Operatori	Precedenta
postfix	*expr*++ *expr*--
unary	++*expr* —*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|

logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Expresii, Instructiuni si Blocuri

O expresie reprezinta o constructie care evalueaza la o singura valoare si este formata din variabile, operatori si invocari de metode.

someRandomArray[0] = 13 //reprezinta o expresie de asignare

Expresia de mai sus este transformata intr-o unitate completa de executie prin aduagarea elementului termina ";" si formeaza o instructiune/declaratie executiva **STATEMENT**.

Un bloc reprezinta o grupare de zero sau mai multe instructiuni delimitata de acolade. Un bloc se poate utiliza oriunde este admisa utilizarea unei instructiuni.

```
if (color <= upperLimit && color >= lowerLimit) /*start of block 1*/ {
      this.color = color;
}/*end of block 1*/ else /*start of block 2*/{
      throw new Exception("Color doesn ot exist");
}/*end of block 2*/
```

Rolul blocurilor de instructiuni este de a organiza si delimita anumite sectiuni de cod fie pentru claritate fie pentru a indica o anumita ordine de executie.

Instructiuni de control

Instructiunile de control permit formarea unei alte ordini de executie decat cea secventiala in ordinea intalnirii instructiunilor din codul elaborat.

In Java exista urmatoarele instructiuni de control:

- if-then si if-then-else
- switch
- while si do-while
- for
- break, continue si return

Clase

Pentru a explica mai bine urmatoarele concepte se vor utiliza obiectele identificate pentru jocul "X si 0".

Pentru acest jos se identifica urmatoarele obiecte de baza:

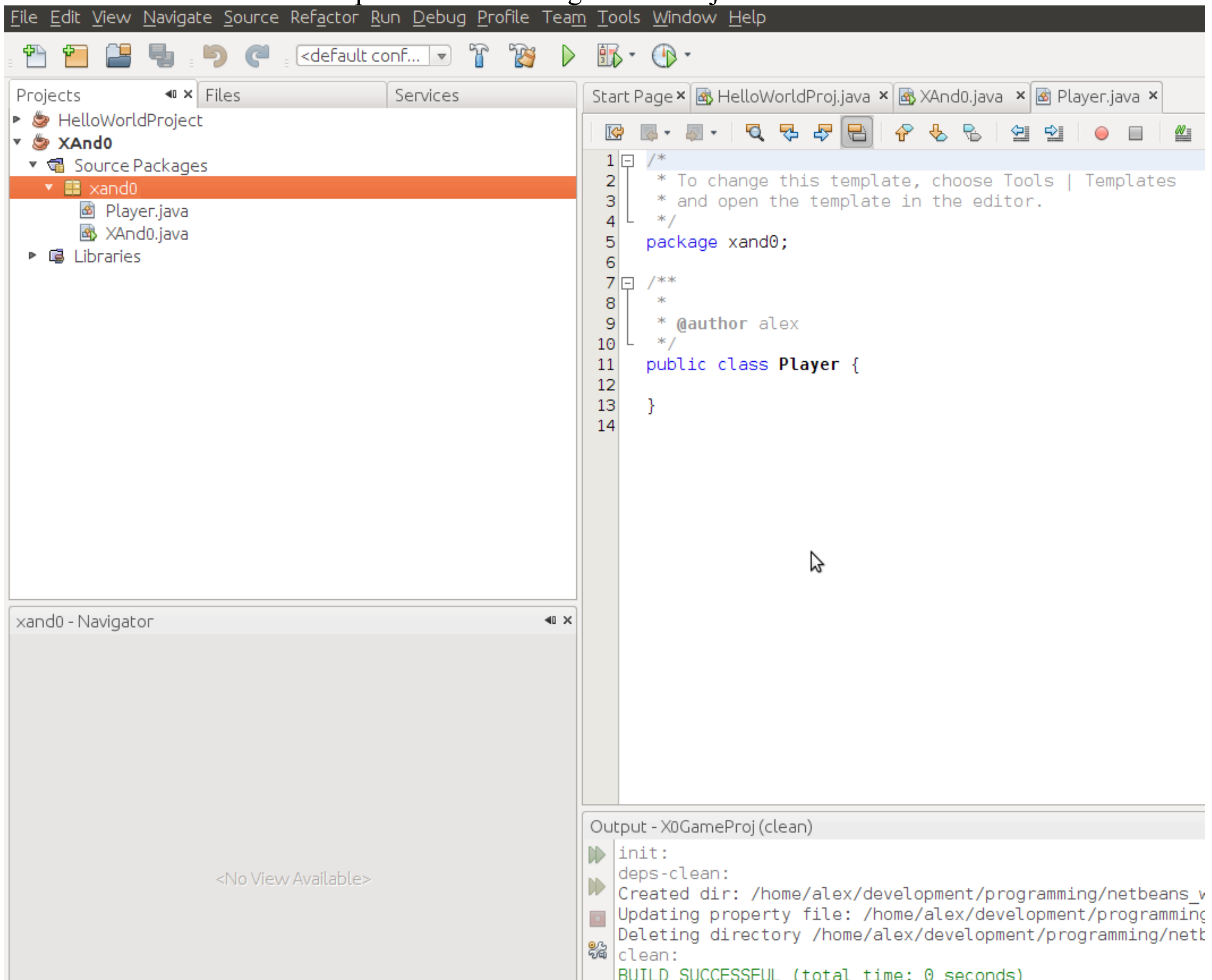
a) jucatorii care reprezinta obiecte instante ale unui tip de obiecte ce va fi reprezentat de clasa Player;
b) tabla de joc care reprezinta o instanta a unui tip identificat prin clasa GameBoard;
c) un arbitru/mediu care suprinde notiunea generala de logica jocului (i.e. regulile jocului) precum si ordinea pe tabla de joc a participantilor;

Pentru inceput se creeaza proiectul XAnd0 in mediul de dezvoltare NetBeans: File|New Project.

Odata creat proiectul se adauga in pachetul xand0 o clasa Java denumita Player - click buton dreapta al mouse-ului atunci cand cursorul se afla deasupra pachetului din zona Projects a ferestrei

mediului de dezvoltare NetBeans; se selecteaza New| Java class.

Se obtine o structura a proiectului ca in figura de mai jos:



Se completeaza clasa Player astfel incat sa contina urmatoarele variabile membre – fields, metode membre de tip getter si setter precum si 2 constructori:

```
package xand0;

public class Player {
    // fields
    String name;
    Double age;
    Double experience;
    Double score;

    // constructors
    public Player(String name) {
        this.name = name;
        age = 0.0; // irrelevant
        experience = 0.0; // no experience
    }

    public Player(String name, Double age, Double experience) {
```

```java
        this.name = name;
        this.age = age;
        this.experience = experience;
    }

    // getters and setters
    public Double getAge() {
        return age;
    }

    public void setAge(Double age) {
        this.age = age;
    }

    public Double getExperience() {
        return experience;
    }

    public void setExperience(Double experience) {
        this.experience = experience;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getScore() {
        return score;
    }

    public void setScore(Double score) {
        this.score = score;
    }
}
```

Declaratia unei clase cuprinde:

- modificatori de acces: *public*, *private* sau *package* (package este utilizat default daca nu este specificat explicit altfel)
- numele clasei
- numele clasei parinte pe care o deriveaza precedata de cuvantul cheie *extends*
- o lista de nume de interfete, separate prin virgula, si precedata de cuvutnul cheie *implements*
- corpul clasei cuprins intre acolade; in corpul clasei se declara si definesc variabilele membre si metodele membre;

```java
public class Xand0Player extends Player implements PlayerInterface {
    // class body
}
```

In exemplul de mai sus clasa Xand0Player prezinta modificatorul de acces public (i.e. oricine poate utiliza aceasta clasa) deriveaza clasa Player (i.e. subclaseaza, mosteneste de la) si implementeaza interfata PlayerInterface.

La inceputul corpului clasei se declara variabilele membre sau campurile clasei – fields:

- zero sau mai multi modificatori de acces – *public, pacakge, protected, private*
- tipul campului
- numele campului

```
public class Xand0Player extends Player implements PlayerInterface {
    // class body

   // fields
    public String name;
    Double age;
    Double experience;
    Double score;
}
```

Declararea unei metode membre a unei clase implica 6 componente:

- modificatorii de access
- tipul returnat – tipul valorii returnate de metoda sau void in cazul in care nu se returneaza nimic; prentru a returna o valoare se invoca cuvantul cheie *return* la finalul corpului metodei urmat de expresia sau declaratia/instructiunea care evalueaza la o expresie si urmat de elemntul terminal ";"
- numele metodei – in Java exista o serie de conventii de denumire pentru numele clasei, numele campurilor si numele metodelor; aceste conventii se pot consulta aici
- o lista de parametri, separati prin virgula, cuprinsa in paranteze rotunde; in absenta oricarui parametru se pastreaza o lista goala ()
- o lista de exceptii – exceptiile for fi discutate in laboratorul urmator
- corpul metodei cuprins intre acolade

```
public class Xand0Player extends Player implements PlayerInterface {
    // class body

   // fields
    public String name;
    Double age;
    Double experience;
    Double score;
    // class body

   // methods
    public String generateGreeting(String name, int years) {
        // local variable
        String greeting = "May the moon pass you for " + (12*years) +
                              "," + name;
        return greeting;

        /* one could have used:
           return "May the moon pass you for " + (12*years) +
                              "," + name;
        */
    }
}
```

In limbajul Java o metoda poate fi supraincarcata - *overloaded* – prin modificarea signaturii metodei. Signatura unei metode este data de numele metodei si tipurile parametrilor.

Astfel intr-o clasa se admite:

```
public class Plot {
    // fields
      ....

    // constructors
      ....

    public void draw(Square s) {
        ...
    }

    public void draw(Circle c) {
        ...
    }

    public void draw(Point p) {
        ...
    }
}
```

O clasa contine de asemeni constructori ce sunt utilizati in crearea instantelor de obiecte pe baza sablonului furnizat de clasa. Declaratia construtorului este identica cu cea pentru o metoda membra a clasei cu exceptia tipului returnat care lipseste (v. clasa Playerde mai sus pentru care s-au definit doi constructori). Orice clasa are cel putin un contructor. In cazul in care nu se defineste explicit nici un constructor, limbajul Java aloca un constructor implicit fara nici un argument care apeleaza contructorul fara argumente al clasei parinte.

Un lucru important de retinut este ca parametrii se transmit mereu prin valaore - *passed by value*. Astfel ca daca se reinitializeaza un parametru in corpul metodei aceasta modificareeste vizibila doar in spatiul corpului metodei; la finalizarea metodei aceasta modificare dispare.

Pentru fixarea acestei particularitati executati exemplele de mai jos si evaluati rezultatele.

1. se creeaza clasa TestParameterPassing.java in cadrul proiectului xand0
2. se adauga modifica clasa incat sa arate astfel:

```
package xand0;

/**
 *
 * @author alex
 */
public class TestParameterPassing {
    public static void test(double d) {
        System.out.println("in test method value of d is: " + d);
    }
    public static void main(String[] args) {
        double d = 13.0;
        test(d);

        System.out.println("in main method value of d is: " + d);
    }
}
```

Metoda main transforma o clasa intr-o instanta de executie Java (i.e. cand se invoca codul compilat Java la nivelul masinii virtuale se incepe executia metodei main).

3. se executa codul de mai sus din mediul NetBean (click pe butonul dreapta al mouse-ului

atunci cand cursorul acestuia se afla deasupra numelui clasei TestParameterPassing din zona Projects; se selecteaza optiunea Run File; alternativ se apasa Shift + F6);
se observa ca valoarea parametrului pasat este aceasi cu cea a obiectului Double definit in main;

4. se modifica metoda statica test astfel:

```
package xand0;

/**
 *
 * @author alex
 */
public class TestParameterPassing {
    public static void test(double d) {
        d = 14.0;
        System.out.println("in test method value of d is: " + d);
    }
    public static void main(String[] args) {
        double d = 13.0; // tipul double este un tip primitiv
        test(d);

        System.out.println("in main method value of d is: " + d);
    }
}
```

se observa ca desi in metoda test s-a modificat parametrul la o noua valoare, odata finalizata metoda variabila pasata ca parametru a ramas nemodificata

5. desi argumentele sunt transmise prin valoare in cazul referintelor (i.e. obiectelor) starea interna a obiectelor referite poate fi modificata iar acestemodificari se pastreaza si dupa finalizarea metodelor unde au fost primiti ca parametri

```
package xand0;

import java.awt.Point;

/**
 *
 * @author alex
 */
public class TestParameterPassing {
    public static void test(Point p) {
        p.x = 24;
        System.out.println("in test method value of p is: " + p);

        p = new Point(10000,20000);
        System.out.println("in test method value of p is: " + p);
    }
    public static void main(String[] args) {
        Point p = new Point(13,25);
        System.out.println("in main method value of p is: " + p);

        test(p);

        System.out.println("in main method value of p is: " + p);
    }
}
```

se observa ca in aceste conditii obiectul p si-a modificat doar starea in urma executiei metodei test; incearcarea de reinitializare a obiectului p, primit ca parametru prin

valoare la nivelul metodei test, nu produce modificarea asteptata la nivelul intregului cod.

Obiecte

Pentru implementarea unei functionalitati specifice, in orice aplicatie Java se creeaza o multitudine de obiecte care intereactioneaza prin invocari de metode.

Pentru a evidentia procesul de creare de obiecte se adauga in proiectul XAnd0, la nivelul pachetului xand0, clasa Environment care arata astfel:

```
package xand0;

/**
 *
 * @author alex
 */
public class Environment {
    public static void main(String[] args) {
        /* declaration, instantiation and initialisation of an object*/
        Player player1 = new Player("Alex");
    }
}
```

Linia scrisa cu bold mai sus descrie crearea unui obiect de tip Player si implica urmatoarele 3 faze:

- declaratia obiectului: **`Player player1 ...`**
- instantierea obiectului: operatorul *new* creaza un obiect pe baza tipului descris de clasa din declaratia obiectului; instantierea presupune la runtime alocarea de memorie pentru obiectul creat si returnarea unei referinte la acel obiect (i.e. variabila player1 reprezinta o referinta la un obiect de tip Player).
- initializarea obiectului: invocarea operatorului *new* determina la runtime, dupa instantiere, apelul constructorului specificat imediat dupa operator, constructor ce trebuie sa existe definit la nivelul clasei specificate in declaratia obiectului.

Declaratia obiectului poate avea loc oriunde, precedent instaierii si initializarii obiectului:

```
package xand0;

/**
 *
 * @author alex
 */
public class Environment {
    public static void main(String[] args) {
        /* declaration, instantiation and initialisation of an object*/
        Player player1;
            ...
            ...
            ...
        player1 = new Player("Alex");
    }
}
```

Odata creat un obiect acesta este mentinut la nivelul JVM pana cand nu mai exista nici o

referinta catre aceste, moment in care este distrus de catre GarbagCollector (mecanismul este ceva mai complex de atat si nu face obiectul acestui material de studiu).

Odata obtinuta referinta catre un obiect (e.g. variabila player1 de mai sus) se pot accesa:

- campurile obiectului (care prezinta modificatoruld e acces adecvat)
- metodele obiectului (care prezinta modificatoruld e acces adecvat)

print utilizarea operatorului **.** :

```
package xand0;

/**
 *
 * @author alex
 */
public class Environment {
    public static void main(String[] args) {
        Player player1 = new Player("Alex");

        System.out.println(player1.name);
        player1.setAge(30.0);
    }
}
```

Invocarea unei metode a obiectului presupune specificarea corecta a listeri de argumente ce trebuie sa conicida cu lista de parametri din declaratia metodei (i.e. tipul obiectelor trebuie sa fie acelasi).

Operatorul this. Modificatori de acces

Dupa cum s-a observat in definirea clasei Player, la nivelul constructorilor s-a utilizat operatorul *this*. In contextul unei metode membre sau al unui contructor *this* reprezinta o referinta la *obiectul curent* – i.e. obiectul a carui metoda sau al carui constructor este apelat. In cazul construcorului s-a utilizat operatorul *this* pentru a diferentia intre variabilele parametru si variabilele membre ale clasei deoarece acestea aveau acelasi nume. Prin this se poate referi orice membur al obiectului curent din interiorul unei metode membre sau a unui constructor.

Modificatorii de acces existenti in Java sunt:

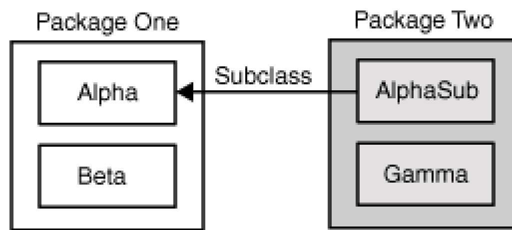
- public
- protected
- private
- package sau default

Posibilitatile de utilizare ale acestor modificatori sunt descrise prin tabelul urmator:

Access Level

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
`package`/*no modifier*	Y	Y	N	N
`private`	Y	N	N	N

Fie urmatoarele 4 clase organizate in pachete conform cu figura de mai jos:



Vizibilitatea membrilor (campuri/metode) clasei Alpha este afectata in functie de modificatorii de acces aplicati, iar efectul este descris prin tabelul urmator:

Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
`package`/*no modifier*	Y	Y	N	N
`private`	Y	N	N	N

Interfete si Mostenire

Interfete

In cazul in care se doreste dezvoltarea unei aplicatii cu o arhitectura flexibila si extensibila atunci componentele software implicate stabilesc un "contract" de interactiune astfel incat sa se asigure o interfata clara de acces la acestea. Un astfel de "contract" poarta denumirea de *interfata* iar acesta nu impune modul de dezvoltare sau mecanismele care determina comportamentul specific fiecarei componente ci doar modalitatea prin care o entitate poate interactiona cu respectiva componenta/instanta obiect.

In cazul proiectului "X si 0" se aborda o directie de dezvoltare astfel incat sa se poata dezvolta orice joc de tip board-game.

Astfel notiunile de jucator si celula a tablei de joc pot fi elaborate pe baza interfetelor:

```
package xand0.model;
public interface PlayerInterface {
        public Double computeExperience();
}

package xand0.model;

public interface BoardCellInterface<E> {
        /*
         * A cell on a board game either consumes/receives information from
the player (e.g. adds and X or an 0
         * in the X and 0 game) or it produces/specifies an action to the
player (e.g. Monopoly like games)
         */
        public void consume(E info);
        public E produce();
}
```

OBS: interfetele nu pot fi instantiate; o interfata nu contine variabile; intr-o interfata se definesc doar signaturile metodelor de acces precedate de valoarea returnata si respectiv modificatori de acces.

Implementarea interfetelor se face prin specificarea cuvatului *implements* in declaratia unei

clase urmat de lista de nume de interfete, ce se doresc implementate, separate prin virgula.

Clase abstracte

O clasa abstracta este o clasa care prezinta in declaratia ei cuvantul cheie *abstract* si poate include sau nu metode abstracte. Similar interfetelor o clasa abstracta nu poate fi instantiata insa poate fi sublclasata si poate contine variabile membre precum si metode ne-bastracte.

O metode abstracta este declarata similar ca metodele unei interfete precizand insa cuvantul cheie abstract in declaratia metodei. O clasa care subclaseaza o clasa abstracta ramane abstracta daca nu implementeaza metodele abstracte ale clasei parinte iar in acest caz clasa trebuie declarata de asemeni explicit ca fiind abstracta.

In proiectul XAnd0 clasa `Player` a fost declarata ca abstracta deorece, in functie de carateristicile fiecarui joc in parte, metoda `computeExperience()`, declarata in interfata `PlayerInterface`, trebuie implementata in clasele specilizate ce descriu un jucator in functie de jocul pe care il practica.

```
package xand0.model;
/*
 * this class is left as an abstract class because itdoes not implement the
PlayerInterface
 */
public abstract class Player implements PlayerInterface {
	// fields
	protected Integer playerID;
	protected String name;
	protected Double age;
	protected Double experience;

	// getters and setters
	public Integer getPlayerID() {
		return playerID;
	}
	public String getName() {
		return name;
	}
	public Double getAge() {
		return age;
	}
	public void setPlayerID(Integer playerID) {
		this.playerID = playerID;
	}
	public void setName(String name) {
		this.name = name;
	}
	public void setAge(Double age) {
		this.age = age;
	}

	// for the experience field the PlayerInterface.computeExperience()
method acts both as setter & getter
}
```

Mostenire

Conceptul de **mosternire/derivare/subclasare/specializare** reprezinta una din cele mai importante caracteristici ale programarii obiect orientate.

In Java o clasa mosteneste campuri si metode de la o clasa prin derivarea acelei clase:

- o clasa care este derivata dintr-o alta clasa poarta denumirea de **subclasa** (clasa derivata, clasa extinsa sau clasa copil) iar clasa din care este derivata este denumita **superclasa** (clasa de baza sau clasa parinte)
- in ierarhia deobiecte Java orice clasa – exceptie clasa Object – are o singura superclasa pe linie ierarhica directa (i.e. deriveaza si mosteneste o singura clasa); in cazul in care nu se specifica explicit nici o superclasa in definitia unei clase atunci aceasta deriveaza impliti clasa Object

Principalul avantaj mostenirii consta in reutilizarea codului. Astfel daca la crearea unei noi clase se constata ca exista deja o alta clasa, care implementeaza o parte a functionalitatii si codului dorit in noua clasa, atunci, prin derivare, noua clasa mosteneste membrii – variabile si metode mebre si clase imbricate – clasei parinte evitand astfel reimplementarea unui cod similar cu aceasi functionalitate. Constructorii unei clase nu sunt membrii ai clasei si nu sunt mosteniti insa acestia pot fi accesati de la nivelul clasei derivate.

Nu toti membrii unei clase sunt mosteniti ci doar cei care prezinta modificatorii de acces *public* si *protected.* In cazul in care clasa parinte si clasa copil sunt in acelasi pachet Java (vezi sectiunea urmatoare) atunci sunt mosteniti si membrii cu modificatorul de access *package.*

Intr-o clasa derivata:

- campurile mostenite pot fi utilizate direct ca si cum ar fi declarate in clasa derivata;
- o variabila membra mostenita poate fi ascunsa in clasa derivata prin declararea unei variabile membre cu acelasi nume; pentruexemplificare incercati urmatoarele exemple:
  - se creaza clasele A si B;
  - B mosteneste pe A si ascunde campul *baseField* prin redefinire;
  - folosind cuvantul cheie *super* se pot accesa membrii clasei de baza; se foloseste *super* pentru a observa diferenta intre campul clasei de baza si campul clasei derivate

```
package xand0.test;

class A {
      public int baseField;

      public A() {
            baseField = 103;
      }
}

public class B extends A {
      public int baseField;
      public B() {
            baseField = 10013;
      }

      public void displayFields() {
            System.out.println(super.baseField);
            System.out.println(baseField);
      }

      public static void main(String[] args) {
            B b = new B();
```

```
                b.displayFields();
        }
    }
```

- rulati exemplul in NetBeans;
- comentati linia `public int baseField;` din clasa B si observati efectul.

- in clasa derivata se pot declara campuri noi care nu exista in superclasa
- metodele mostenite pot fi utilizate direct ca si cum ar fi declarate in clasa derivata
- daca se declara o metoda cu aceasi signatura ca o metoda mostenita din clasa de baza atunci se realizeaza suprascrierea metodei mostenite; cand se invoca o metoda suprascrisa se executa corpul metodei din clasa derivata
- in clasa derivata se poate apela constructorul clasei de baza; pentru aceasta se utilizeaza cuvantul cheie super pe prima linie a corpului unui din constructorii clasei derivate:

```
public class B extends A {
      public int baseField;
      public B() {
            super();
            baseField = 10013;
}
```

Pachete Java

Pentru o organizare eficienta a tipurilor – rapiditate in gasirea claselor, evitarea conflictelor denume, contrulul nivelului de acces – in Java proiectele si librariile sunt organizate in unitati specializate denumite pachete.

Un *pachet* – *package* – reprezinta o grupare de tipuri similare pentru asigurarea protectiei la nivel de acces si administrarea eficienta a spatiului de nume, Reaminitim ca notiunea abstracta de tip se refera atat la clase cat si la interfete, enumerari si adnotari.

Tipurile care sunt deja furnizate de platforma Java sunt organizate in diverse pachete suprinse in documentati specifica Java API.

In cadrul proiectului inceput in acest laborator s-a definit pachetul:

xand0.model

unde vor fi incluse clasele specifice implementarii logicii si entitatilor care participa in jocul "X si 0".

Pentru crearea unui pachet trebuie alesunnume de pachet (e.g. xand0.model) iar apoi se adauga, ca prima linie de cod, `package xand0.model;` in toate clasele care vor fi grupate in acest pachet.

In cazul in care clasele se creeaza folosind NetBeans numele pachetului poate fi specificat in fereastra de generare a clasei iar organizarea si adaugarea liniei package este realizata automat.

Stabilirea numelor de pachete urmareste un set de conventii clare (ca si in cazul denumirilor de clase, campuri si metode):

- numele sunt scrise cu litere minuscule
- in cazul in care este disponibil,se poate utiliza domeniul de Internet detinut pentru a incepe denumirea pachetului: com.etti.xand0.model
- pachetele care apartin platformei Java incep de regula cu java. sau javax.
- numele pachetelor trebuie sa inceapa cu o litera si nu pot fi denumite dupa un tip de baza (e.g. int).

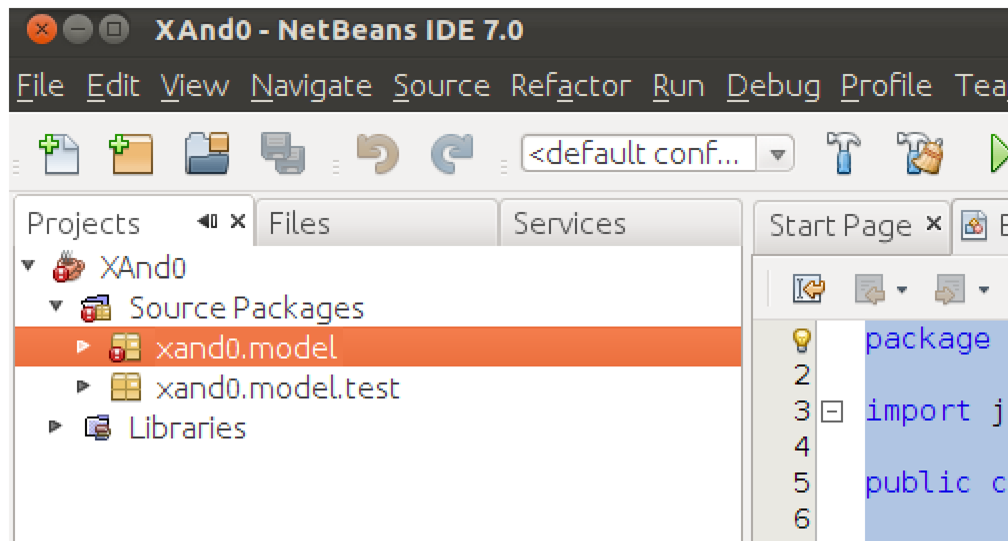
Tipurile care formeaza un pachet sunt denumite membrii pachetului. In cadrul unui pachet membrii au acces conform cu modificatorii de acces public, protected sau private. In afara pachetului accesul este permis doar la tipurile publice iar accesarea lor se face prin:

- referirea cu inregul nume, calificat cu numele pachetului
  - `xand0.model.Xand0Player`
- importarea clasei unde este necesar accesul catre aceasta
  - ```
    import xand0.model.Xand0Player;
    ...
    public class ...
    ...
      Xand0Player player1 = new Xand0Player(...);
    ```
- importarea intregului pachet al clasei ce se doreste accesata
  - `import xand0.model.*;`

NOTA: O descriere detaliata a tuturor conceptelor precedente se poate consulta la aceasta adresa.

Teme si exercitii

1. Sa se realizeze in NetBeans urmatoarea structura de pachete in cadrul proiectului Xand0:

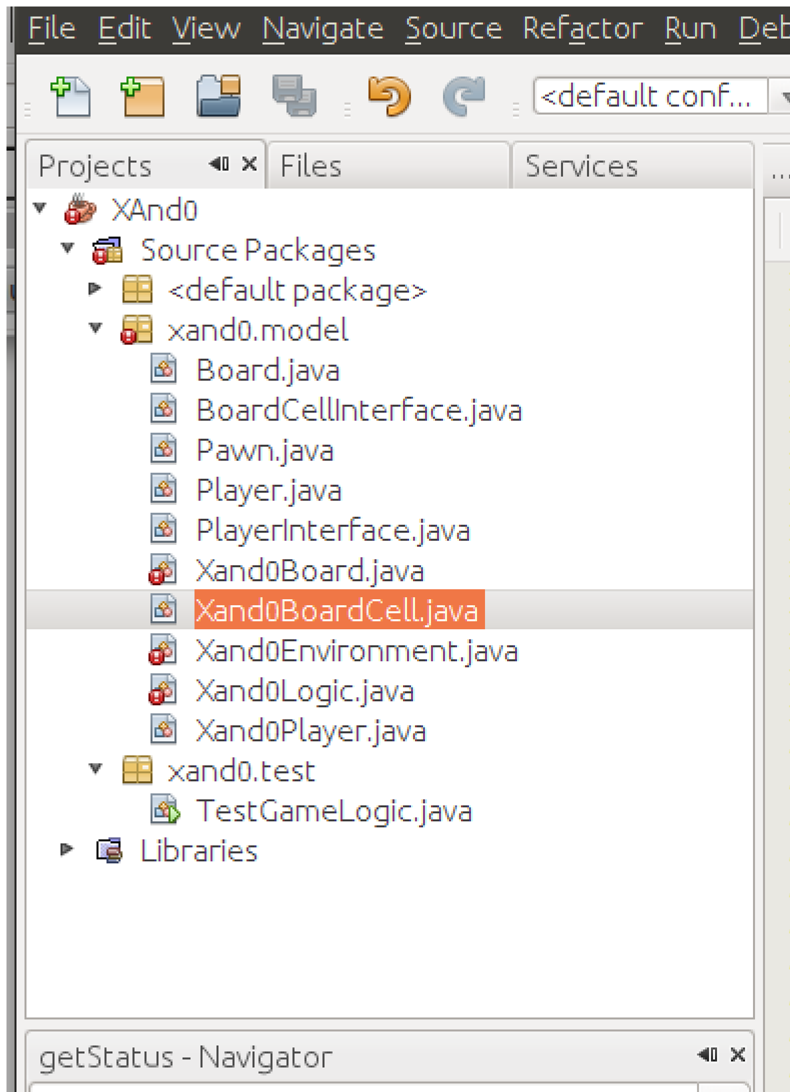


2. Sa se adauge in pachetul xand0.model urmatoarele clase:
3. Pentru implementarea jocului Xand0 s-au identificat urmatoarele tipuri necesare:
   - Xand0Player
   - Xand0Board
   - Xand0BoardCell
   - Xand0Logic
   - Xand0Environment

Pentru implementarea acestor tipuri astfel incat sa se asigure cat mai multa flexibilitate in dezvoltare si o arhitectura extensibila s-au definitsuplimentar tipurile asbtracte:

- PlayerInterface
- BoardCellInterface
- Board
    ```

- Pawn
- Player



Completati proiectul XAnd0 cu aceste tipuri: codul se poate extrage din arhiva atasata acestui document Xand0.rar.

4. Completati in locul a "..." din clasele Xand0Board, Xand0Environment si Xand0Logic codul corespunzator astfel incat sa se realizeze functionalitatea ceruta in comentariul ce precede "...".
5. Testati functionalitatea obtinuta prin utilizarea clasei xand0.test.TestGameLogic