Programare Java (2)

prof.dr.ing. Ștefan-Gheorghe Pentiuc

Modificator	CLASĂ	METODĂ	VARIABILĂ
public	acces permis tuturor claselor	poate fi apelată din orice clasă	**poate fi consultată/ modificată din orice clasă**
private	-	nu poate fi apelată decât din clasa respectivă	**nu poate fi consultată/ modificată decât din clasa respectivă**
protected	-	poate fi apelată din clasa respectivă, subclasele şi clasele aceluiaşi pachet	**poate fi consultată/ modificată din clasa respectivă, subclasele şi clasele aceluiaşi pachet**
- (fără modificator)	clasa este accesibilă claselor din acelaşi pachet	poate fi apelată din clasa respectivă şi din clasele aceluiași pachet	**poate fi consultată/ modificată din clasa respectivă şi din clasele aceluiași pachet**

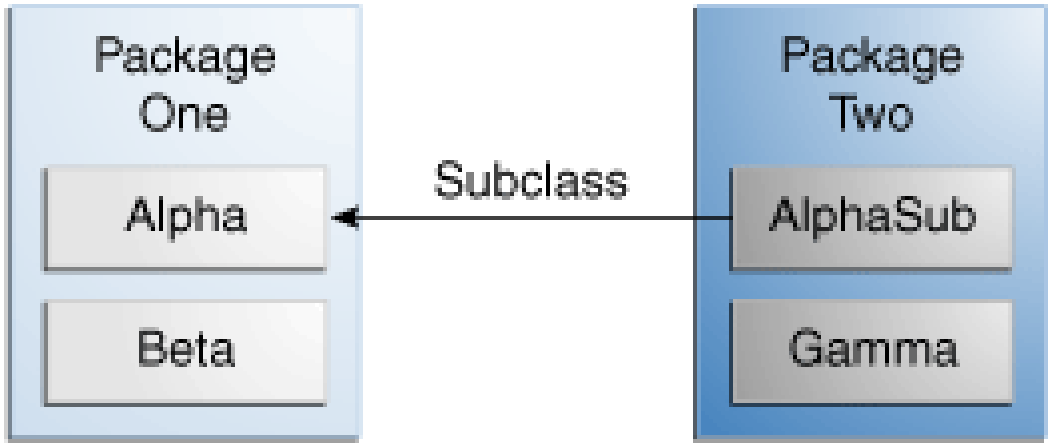
Modificator	CLASĂ	METODĂ	VARIABILĂ
abstract	defineşte o clasă şablon (abstractă)	metodă şablon (fără a defini corpul funcţiei)	-
static	-	nu poate fi redefinita in subclase metoda poate fi referirta **clasa.metoda**()	**variabilă globală obiectelor clasei**
final	nu are subclase	nu mai poate fi redefinită în subclase	**constantă**
final static	-	-	**globală, nu poate fi modificată în subclase**

Accesul la membrii unei clase

Modificator	Clasă	Pachet	Subclasă	Exterior
public	Da	Da	Da	Da
protected	Da	Da	Da	NU
Fără modificator	Da	Da	NU	NU
private	Da	NU	NU	NU

Fără modificator (implicit) = protejat la nivel de pachet (package private)

Vizibilitatea membrilor clasei Alpha



Modificator	Alpha	Beta	Alphasub	Gamma
public	Da	Da	Da	Da
protected	Da	Da	Da	Nu
fără modificator	Da	Da	Nu	Nu
private	Da	Nu	Nu	Nu

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscoNutrol.html

Sfaturi

- utilizati **cel mai restrictiv** mod de acces cu putință
- **private** *nu* va fi utilizat decât dacă este un motiv să nu fie utilizat
- de evitat **public** pentru câmpuri; excepție constantele

```
public final int NMAX=100;
```

Moștenire și modificatorii de acces

Java interzice reducerea accesului în clasele derivate!

Trebuie ca metodele suprascrise in clasele copil sa poată fi accesate la fel ca cele din clasa părinte.

Moștenire

```
package adnotare;

public class Parinte {

    public void afis (String sir) {
        System.out.println("Afis parinte: ” + sir);
    }
}
```

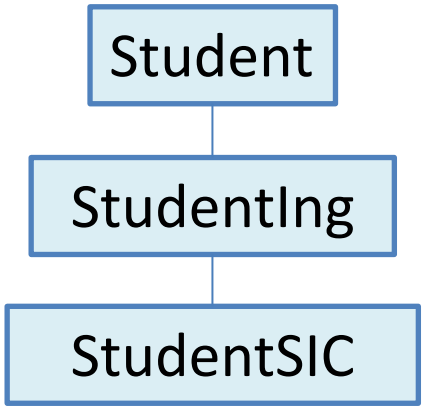
Moștenire

```
package adnotare;

public class Copil extends Parinte {

    protected void afis(String sir){
        System.out.println("Afis copil: "+sir);
    }
}
```

error: afis(String) in Copil cannot override afis(String) in Parinte
protected void afis(String sir){
attempting to assign weaker access privileges; was public

POLIMORFISM

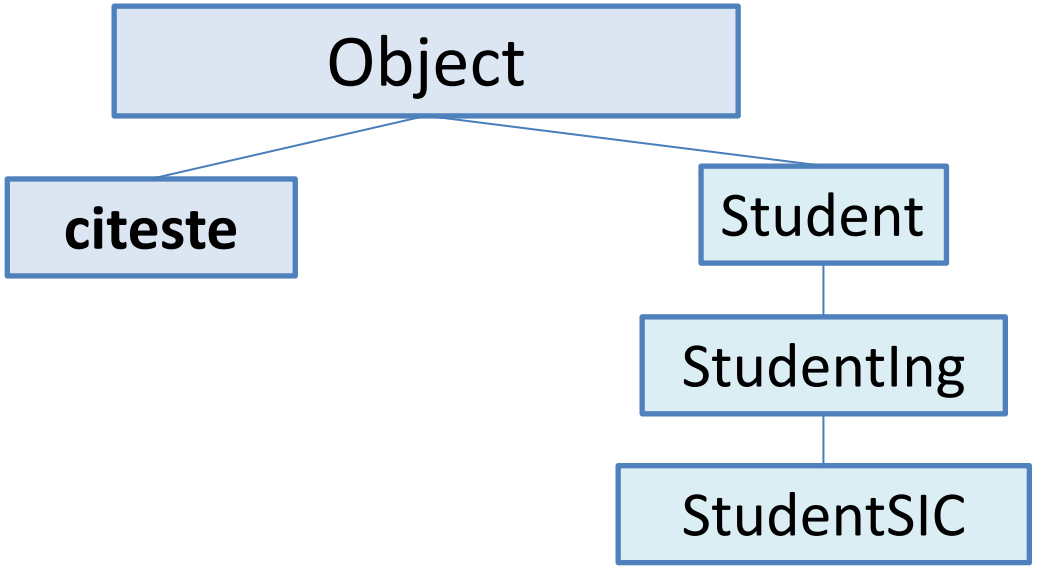
Atribuiri polimorfe




```
Student s;
StudentIng x,y;
StudentSIC d;
citeste c;
//
d= new StudentSIC("Numex","Prenumex",2020,true);

// atribuiri corecte
x=d;  // StudentIng este superclasa lui StudentSIC
s=x;  // Student este superclasa lui StudentIng
s=d;  // Student este si superclasa lui StudentSIC
y=x;  // de acelasi tip
```

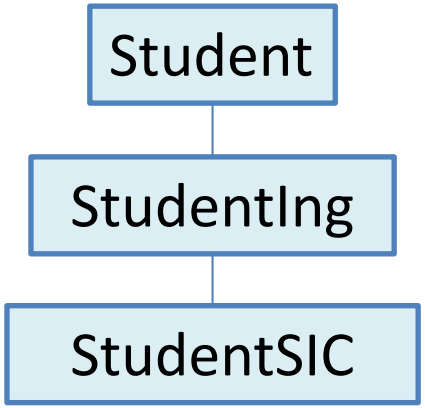
Atribuiri polimorfe

```
Object o;

// atribuiri corecte
o=s;    // Object este superclasa tuturor
o=y;
o=d;
o=c;
```



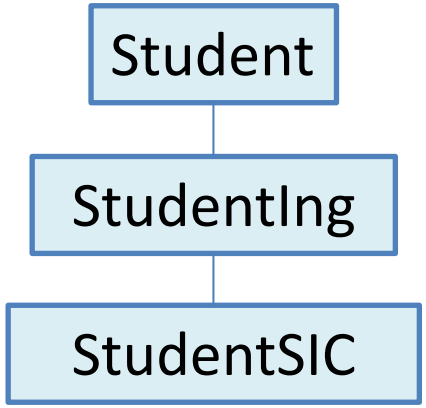
Atribuiri polimorfe INCORECTE

```
Student s;
StudentIng x,y;
StudentSIC d;
citeste c;
Object o;
```



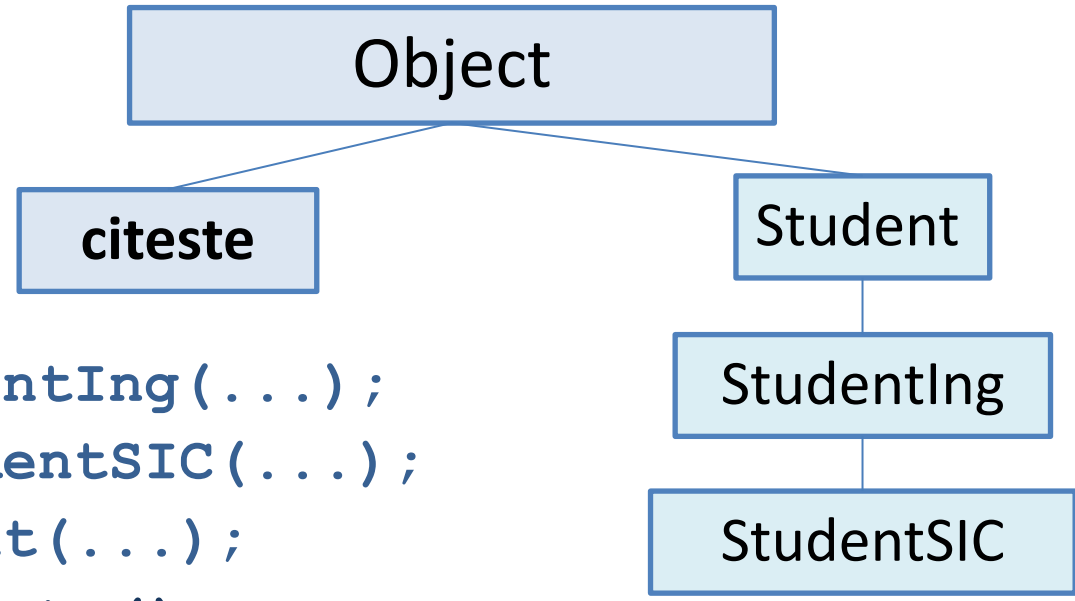

```
/*  atribuiri eronate
d=x;  //  incompatible types
      // required: Studenti.StudentSIC

x=s;  // required: Studenti.StudentIng
d=s;  // required: Studenti.StudentSIC

s=o;  // required: Studenti.Student
x=o;  // required: Studenti.StudentIng
d=o;  // required: Studenti.StudentSIC

c =s;  //required: citeste
c=o;
*/
```

Tablouri polimorfe



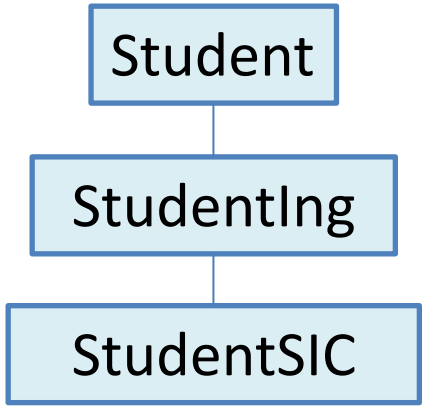

```
StudentIng  d = new StudentIng(...);
StudentSIC  bun = new StudentSIC(...);
Student     a = new Student(...);

Student[] grup=new Student[]{d,bun,a};
for(Student s: grup)
     System.out.println(s);
```

Tablouri polimorfe




```
StudentIng d= new StudentIng(...);
StudentSIC bun=new StudentSIC(...);
Student    a=new Student(...);
citeste    c= new citeste();
Object     o= new Object();

Object[] vector = new Object[]{d,bun,a,c,o};
for(Object v: vector)
     System.out.println(v);
```

Polimorfism

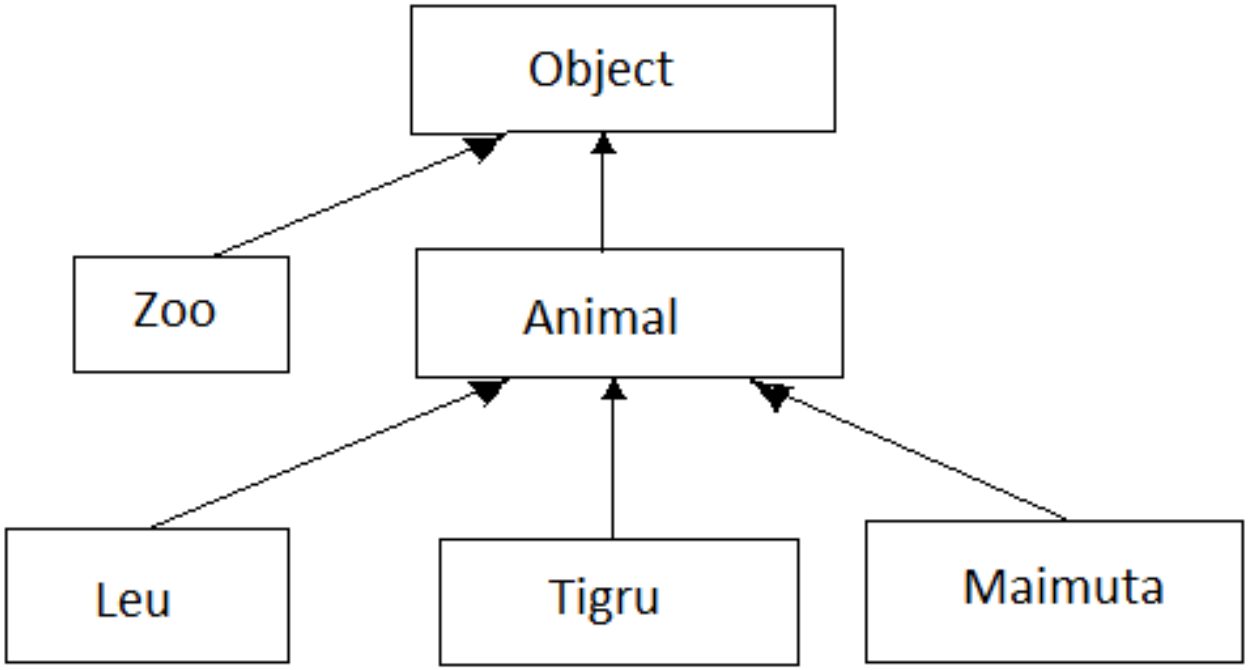



```
class StudentIng extends Student {
int an;
   public void promovat() {
     int an_terminal=4;
     if (this instanceof StudentSIC)
                                  an_terminal=6;
     if(an+1<=an_terminal) an++;
   }

   //...
   public static void main(String args[]){
     Student[] grup =new Student[]{ing1,sic1,sic2,ing3};
     for(Student s: grup) s.promovat();
   }
}
```

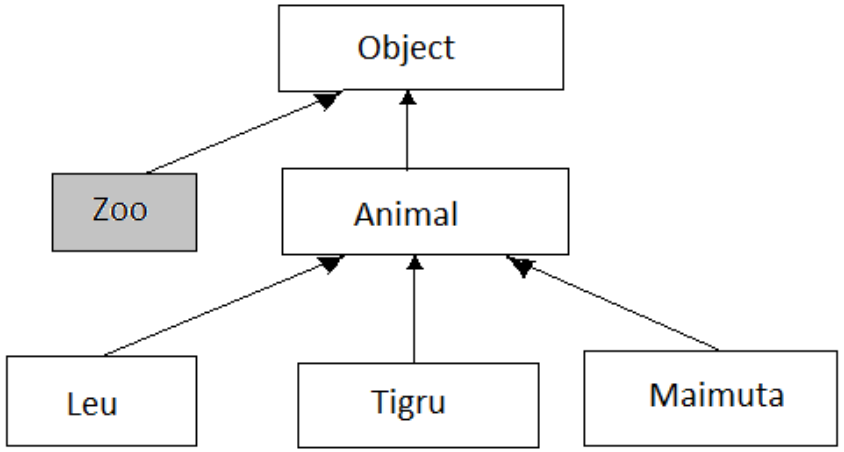
Exemplu complet cu un tablou polimorf

Aplicatie: Gestiunea unei grădini zoologice care adăpostește mai multe animale diferite: lei, tigri, maimuțe.

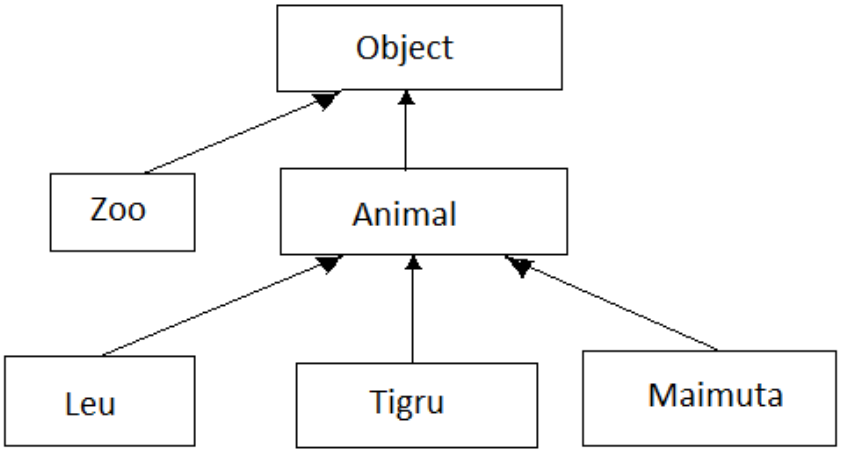
Modelarea lor în POO conduce a ierarhia de clase:



```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}


class Maimuta extends Animal{
}
```



```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}


class Maimuta extends Animal{
}
```




```
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[4];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("In Zoo avem:");

        for (int i = 0; i < zoo.length; i++) {
            System.out.println(zoo[i]+"    ");

    }
}
```

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}


class Maimuta extends Animal{
}
```

```
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[5];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("In Zoo avem:");

        for (int i = 0; i < zoo.length; i++) {
            System.out.println(zoo[i]+"    ");
        }
    }
}
```

Care este afișarea produsă de program ?

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}

class Maimuta extends Animal{
}
```

```
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[5];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("In Zoo avem:");

        for (int i = 0; i < zoo.length; i++) {
            System.out.println(zoo[i]+"    ");
        }
    }
}
```

Care este afișarea produsă de program ?

In Zoo avem:
Leu
ro.usv.Tigru@27d6c5e0
ro.usv.Animal@4f3f5b24
ro.usv.Maimuta@15aeb7ab
null

De ce *toString()* din clasa **Object** a afisat pentru zoo[1] ro.usv.Tigru@27d6c5e0

java.lang.**Class**

Instanțe ale clasei **Class** reprezintă clasele, interfețele în timpul execuției unui program Java.

Pentru fiecare clasă dintr-un program Java si pentru toate tablourile JVM creează câte un obiect de tip **Class**

În continuare se prezintă metode ale claselor **Object** și **Class** preluate din **Java API Documentation**
https://docs.oracle.com/en/java/javase/15/docs/api/index.html

*java.lang.**Object*** **Method Summary**

Modifier and Type	Method	Description
protected **Object**	**clone**()	Creates and returns a copy of this object.
boolean	**equals**(**Object** obj)	Indicates whether some other object is "equal to" this one.
protected void	**finalize**()	**Deprecated.**The finalization mechanism is inherently problematic.
Class<?>	**getClass**()	Returns the runtime class of this Object.
int	**hashCode**()	Returns a hash code value for the object.
void	**notify**()	Wakes up a single thread that is waiting on this object's monitor.
void	**notifyAll**()	Wakes up all threads that are waiting on this object's monitor.
String	**toString**()	Returns a string representation of the object.
void	**wait**()	Causes the current thread to wait until it is awakened, typically by being notified or interrupted.
void	**wait**(long timeoutMillis)	Causes the current thread to wait until it is awakened, typically by being notified or interrupted, or until a certain amount of real time has elapsed.
void	**wait**(long timeoutMillis, int nanos)	Causes the current thread to wait until it is awakened, typically by being notified or interrupted, or until a certain amount of real time has elapsed.

java.lang.Class Method Summary (selecție)

static Class<?>	**forName(String className)**	**Returns the Class object associated with the class or interface with the given string name.**
Constructor<?>[]	getConstructors()	Returns an array containing Constructor objects reflecting all the public constructors of the class represented by this Class object.
Field[]	getFields()	Returns an array containing Field objects reflecting all the accessible public fields of the class or interface represented by this Class object.
Class<?>[]	getInterfaces()	Returns the interfaces directly implemented by the class or interface represented by this object.
Method[]	getMethods()	Returns an array containing Method objects reflecting all the public methods of the class or interface represented by this Class object, including those declared by the class or interface and those inherited from superclasses and superinterfaces.
int	getModifiers()	Returns the Java language modifiers for this class or interface, encoded in an integer.
String	getName()	Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this Class object, as a String.
String	getSimpleName()	Returns the simple name of the underlying class as given in the source code.
Class<? super T>	getSuperclass()	Returns the Class representing the direct superclass of the entity (class, interface, primitive type or void) represented by this Class.
boolean	isArray()	Determines if this Class object represents an array class.
boolean	isEnum()	Returns true if and only if this class was declared as an enum in the source code.
boolean	isInterface()	Determines if the specified Class object represents an interface type.

De ce *toString()* din clasa **Object** a afisat pentru zoo[1] ro.usv.Tigru@27d6c5e0

java.lang.**Class**

String toString() – o reprezentare sub forma “clasa@nrHexazecimal”

Class getClass() – returnează clasa obiectului (un obiect de tip Class)

java.lang.**Class**

String getName() – returneaza numele complet al clasei (pachet.clasă)

String getSimpleName() - returneaza numele clasei fără denumirea pachetului

Exemplu. Fie Leu obj = new Leu();

Obținerea **clasei** unui obiect: obj.getClass() ------> un obiect de tip Class

Numele complet al clasei: obj.getClass().getName() ---> "ro.usv.Leu”

Numele clasei fără den.pachet: obj.getClass().getSimpleName() --> “Leu"

Metoda toString() din clasa **Object**

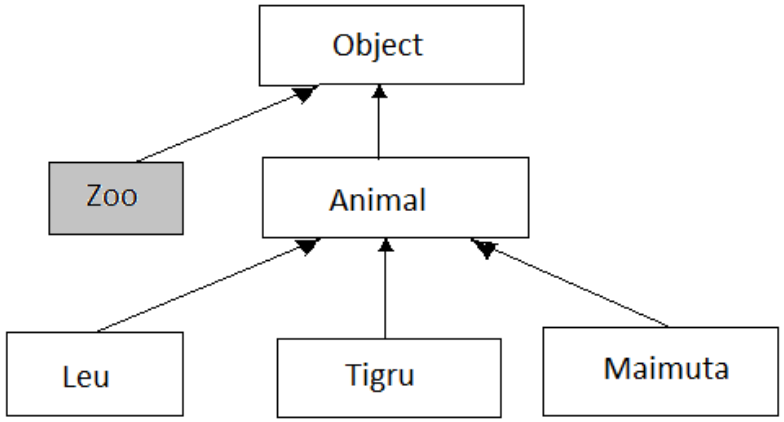
```
public class Object {
    // . . .
    public String toString() {
          return getClass().getName() + "@" +
                    Integer.toHexString ( hashCode() );
    }
    // . . .
    public native int hashCode();
    // . . .
}
```

ro.usv.Tigru@27d6c5e0

getClass().**getName**() @ Integer.toHexString (hashCode())

Pentru a afisa doar numele simplu al claselor obiectelor din tabloul Animal zoo[]

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
    @Override
    public String toString() {
        return  getClass().getSimpleName();
    }
}
```



```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
    @Override
    public String toString() {
        return getClass().getSimpleName();
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}
```

```
class Maimuta extends Animal{
}
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[4];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("In Zoo avem:");

        for (int i = 0; i < zoo.length; i++) {
            System.out.println(zoo[i]+"    ");
    }
}
```

Care este afișarea produsă de program ?

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
    @Override
    public String toString() {
        return getClass().getSimpleName();
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}

class Maimuta extends Animal{
}
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[4];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("In Zoo avem:");

        for (int i = 0; i < zoo.length; i++) {
            System.out.println(zoo[i]+"    ");
        }
}
```

Care este afișarea produsă de program ?

In Zoo avem:
Leu
Tigru
Animal
Maimuta

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
    @Override
    public String toString() {
        return getClass().getSimpleName();
}

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}

class Maimuta extends Animal{
}
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[4];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("In Zoo avem:");

        for (int i = 0; i < zoo.length; i++) {
            System.out.println(zoo[i]+"    ");
        }
}
```

Care este afișarea produsă de program ?

In Zoo avem:
Leu
Tigru
Animal
Maimuta

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
    @Override
    public String toString() {
        return getClass().getSimpleName();
    }

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}

class Maimuta extends Animal{
}
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[4];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("Ce fac “ +
                           "pensionarii din Zoo:");
        for(Animal a: zoo)
                a.ceFace();
    }
}
```

Care este afișarea produsă de program ?

```
class Animal {
    public void ceFace(){
        System.out.println(this + " face ce poate");
    }
    @Override
    public String toString() {
        return getClass().getSimpleName();
    }

class Leu extends Animal {
    @Override
    public void ceFace() {
        System.out.println(this + " se plimba");
    }
    @Override
    public String toString() {
        return "Leu";
    }
}

class Tigru extends Animal {
    @Override
    public void ceFace() {
        System.out.println( this +" zambeste");
    }
}

class Maimuta extends Animal{
}
public class Zoo {
    public static void main(String[] args) {
        Animal[] zoo = new Animal[4];
        zoo[0]= new Leu();
        zoo[1] = new Tigru();
        zoo[2] = new Animal();
        zoo[3] = new Maimuta();
        System.out.println("Ce fac " +
                           "pensionarii din Zoo:");
        for(Animal a: zoo)
                a.ceFace();
    }
}
```

Care este afișarea produsă de program ?

Ce fac pensionarii din Zoo:
Leu se plimba
Tigru zambeste
Animal face ce poate
Maimuta face ce poate

Dacă tabloul zoo[] ar fi de tip Object

```
Object[] zoo = new Object[5];
zoo[0]= new Leu();
zoo[1] = new Tigru();
zoo[2] = new Animal();
zoo[3] = new Maimuta();
zoo[4] = new Punct();          // nu era posibil pt. Animal zoo[]...
System.out.println("Ce fac pensionarii din Zoo:");
for(Object a: zoo){
   ((Animal)a).ceFace();       // era eroare pentru a.ceFace()
}
```

Care este rezultatul executiei ?

Dacă tabloul zoo[] ar fi de tip Object

```
Object[] zoo = new Object[5];
zoo[0]= new Leu();
zoo[1] = new Tigru();
zoo[2] = new Animal();
zoo[3] = new Maimuta();
zoo[4] = new Punct();          // nu era posibil pt. Animal zoo[]...
System.out.println("Ce fac pensionarii din Zoo:");
for(Object a: zoo){
   ((Animal)a).ceFace();       // era eroare pentru a.ceFace()
}
```

```
Ce fac pensionarii din Zoo:
Leu se plimba
Tigru zambeste
Animal face ce poate
Maimuta face ce poate
Exception in thread "main" java.lang.ClassCastException:
class ro.usv.Punct cannot be cast to class ro.usv.Animal (ro.usv.Punct and
ro.usv.Animal are in unnamed module of loader 'app')
        at ro.usv.Zoo.main(Zoo.java:58)
```

Moștenire

```
package adnotare;

public class Parinte {

    public void afis (String sir) {
        System.out.println("Afis parinte: ” + sir);
    }
}
```

Moștenire

```
package adnotare;

public class Copil extends Parinte {

    @Override
    public void afis(String sir){
        System.out.println("Afis copil: " + sir);
    }
}
```

Polimorfism

```
package adnotare;
public class TestParinteCopil {
    public static void main(String[] args) {
        Parinte pp= new Parinte();
        Parinte pc= new Copil();
        Copil cc= new Copil();
//      Copil cp= new Parinte();  //Parinte cannot be converted to Copil

        pp.afis("pp 1");
        pc.afis("pc 2");
        cc.afis("cc 3");
    }
}
```

Polimorfism

```
package adnotare;
public class TestParinteCopil {
    public static void main(String[] args) {
        Parinte pp= new Parinte();
        Parinte pc= new Copil();
        Copil cc= new Copil();
//      Copil cp= new Parinte();  //Parinte cannot be converted to Copil

        pp.afis("pp 1");
        pc.afis("pc 2");
        cc.afis("cc 3");
    }
}
```

```
Afis parinte: pp 1
Afis copil: pc 2
Afis copil: cc 3
```

Polimorfism

```
package adnotare;
public class TestParinteCopil {
    public static void main(String[] args) {
        Parinte pp= new Parinte();
        Parinte pc= new Copil();
        Copil cc= new Copil();
//      Copil cp= new Parinte();  //eroare Parinte cannot be converted to Copil

        pp.afis("pp 1");
        pc.afis("pc 2");
        cc.afis("cc 3");
      System.out.println(" Clasa obj ref de pc "+ pc.getClass().getName() ) ;
    }
}
```

Polimorfism

```
package adnotare;
public class TestParinteCopil {
    public static void main(String[] args) {
    Parinte pp= new Parinte();
    Parinte pc= new Copil();
    Copil cc= new Copil();
//    Copil cp= new Parinte();  //eroare Parinte cannot be converted to Copil

    pp.afis("pp 1");
    pc.afis("pc 2");
    cc.afis("cc 3");
    System.out.println("Clasa obj ref de pc "+ pc.getClass().getName() ) ;
     }
}
```

Afis parinte: pp 1
Afis copil: pc 2
Afis copil: cc 3

Clasa obj ref de pc **adnotare.Copil**

Invocarea metodei virtuale

Mașina virtuală Java (JVM) apelează totdeauna metoda cea mai potrivită pentru **obiectul** referit de o variabilă (*exemplu* Parinte pc = new Copil();).

Nu se apelează metoda definită de tipul variabilei (în exemplu **Parinte**), ci metoda cea mai adecvată tipului **obiectului** referit de variabilă pc (**Copil**).

Dar dacă ar fi metode diferite, însă cu același nume ?

```
package adnotare;

public class Parinte {

    public void afis (Object sir) {
        System.out.println("Afis parinte: ” + sir);
    }
}

public class Copil extends Parinte {

// @Override   error: method does not override or implement a method from a supertype
    public void afis(String sir){
       System.out.println("Afis copil: " + sir);
    }
}
```

Atentie. S-a renuntat la @Override

Există 2 metode diferite afis().
Una în clasa Parinte și una în clasa Copil

```
package adnotare;
public class TestParinteCopil {
    public static void main(String[] args) {
      Parinte pp= new Parinte();
      Parinte pc= new Copil();
      Copil cc= new Copil();
//    Copil cp= new Parinte();  //Parinte cannot be converted to Copil

      pp.afis("pp 1");
      pc.afis("pc 2");
      cc.afis("cc 3");
    }
}
```

Ce se afișează ?

Rezultatul afișării

```
package adnotare;
public class TestParinteCopil {
    public static void main(String[] args) {
    Parinte pp= new Parinte();
    Parinte pc= new Copil();
    Copil cc= new Copil();
//    Copil cp= new Parinte();  //Parinte cannot be converted to Copil

    pp.afis("pp 1");
    pc.afis("pc 2");
    cc.afis("cc 3");
    }
}
```

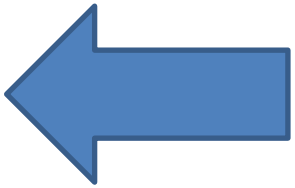
```
Afis parinte: pp 1
Afis parinte: pc 2
Afis copil: cc 3
```

Fie o nouă metodă în clasa Copil

```
package adnotare;

public class Copil extends Parinte {
    @Override
    public void afis(String sir){
        System.out.println("Afis copil: " + sir);
    }

    public void scrie(String sir){
        System.out.println(„Scrie copil: " + sir);
    }
}
```

Ce se afișează ?

```
package adnotare;

public class TestParinteCopil {
    public static void main(String[] args) {
        Parinte pp = new Parinte();
        Copil cc = new Copil();
        Parinte pc = new Copil();
//      Copil cp= new Parinte();
//       err: Parinte cannot be converted to Copil
        pp.afis("pp 1");
        cc.afis("cc 3");
        cc.scrie("pc 3.2");

        pc.afis("pc 2");
        // pc.scrie("pc 2");   -> eroare - necesar cast
        ((Copil)pc).scrie("pc 2");
    }
}
```



Ce se afișează ?

```
package adnotare;

public class TestParinteCopil {
    public static void main(String[] args) {
        Parinte pp = new Parinte();
        Copil cc = new Copil();
        Parinte pc = new Copil();
//      Copil cp= new Parinte();
//       err: Parinte cannot be converted to Copil
        pp.afis("pp 1");
        cc.afis("cc 3");
        cc.scrie("pc 3.2");

        pc.afis("pc 2");
        // pc.scrie("pc 2");   -> eroare - necesar cast
        ((Copil)pc).scrie("pc 2");
    }
}
```

Afis parinte: pp 1
Afis copil: cc 3
Scrie copil: pc 3.2
Afis copil: pc 2
Scrie copil: pc 2

Clasă abstractă

```
public abstract class FiguraGeometrica{
        public abstract double getAria();
}
public class Cerc extends FiguraGeometrica {
      double r;
      public Cerc(double r){
         this.r=r;
      }
      public double getAria() {
            return Math.PI*r*r;
      }
}
```

Clasa abstracta

```
public class Dreptunghi extends FiguraGeometrica {
    double lat, lung;
    public Dreptunghi(double lat, double lung){
        this.lat=lat;
        this.lung=lung;
    }
    public double getAria() {
            return lung*lat;
    }
}
```

Mostenire

```
public class Patrat extends Dreptunghi {
    public Patrat(double lat){
        super(lat, lat);
    }
}
```

Clasa abstracta

```
public class AriaTotala {
   public static double sumaAriilor (FiguraGeometrica figuri[]) {
      double suma=0;
      for (FiguraGeometrica fig: figuri) suma += fig.getAria();
      return suma;
   }
   public static void main(String args[]){
       FiguraGeometrica diverse[] ={new Patrat(2), new Cerc(1),
                                    new Dreptunghi(3, 4)  };
       System.out.println("Suma ariilor este " + sumaAriilor(diverse));
   }
}
```

Interfețe

```
public interface FigGeometrica{
        public abstract double getAria();
}

public class Cerc implements FigGeometrica {
    double r;
    public Cerc(double r){
       this.r=r;
    }
    public double getAria() {
          return Math.PI*r*r;
    }
}
```

Interfețe

```
public class Dreptunghi implements FigGeometrica {
    double lat, lung;
    public Dreptunghi(double lat, double lung){
        this.lat=lat;
        this.lung=lung;
    }
    public double getAria() {
            return lung*lat;
    }
}
```

Interfețe

```
public class Patrat extends Dreptunghi {
        public Patrat(double lat){
            super(lat, lat);
        }
}
```

Nemodificat

Interfețe

```
public class AriaTotala {
   public static double sumaAriilor (FigGeometrica figuri[]) {
      double suma=0;
      for (FigGeometrica fig: figuri) suma += fig.getAria();
      return suma;
   }
   public static void main(String args[]){
       FigGeometrica  diverse[] =  { new Patrat(2), new Cerc(1),
                                     new Dreptunghi(3, 4)  };
       System.out.println("Suma ariilor este " + sumaAriilor(diverse));
   }
}
```

Nemodificat

Interfețe

Sunt utile pentru:

- A modela similaritățile dintre clase între care nu există o relație de moștenire.
- Declararea metodelor care există cu certitudine într-o clasă (sau mai multe).
- A putea lucra cu un obiect de programare fără a-i cunoaște clasa (doar interfața).

Interfețele

- Similare cu ”abstract class”
- Ajută la a pune în evidență metodele comune în clase diferite
- Toate metodele sunt *public*
- În Java o clasă poate implementa mai multe interfețe.

Interfețe

```
interface IChef {
   void cook(Food food);
}
```

```
interface BabyKicker {
   void kickTheBaby(Baby);
}
```

```
interface SouthParkCharacter {
   void curse();
}
```

```
class Chef implements IChef, SouthParkCharacter {
       // overridden methods MUST be public
       // can you tell why ?
       public void curse() { … }
       public void cook(Food f) { … }
}
```

* access rights (Java forbids reducing of access rights)

Când e necesară o interfață?

Atunci când se dorește ascunderea structurii interne a unei clase.

Numai interfața este făcută cunoscută.