

Curs 3 PPOO

Prof. univ. dr. Cristian CIUREA

Departamentul de Informatică și Cibernetică Economică

cristian.ciurea@ie.ase.ro

Java fundamentals

- ▶ Clase
- ▶ Shallow Copy vs. Deep Copy
- ▶ Moștenire (derivare)
- ▶ Polimorfism
- ▶ Clase abstracte
- ▶ Interfețe

Clase

Concepte de bază ale POO:

- ▶ Ce este o clasă? Ce este un **obiect**?
- ▶ Fiecare obiect conține date (**atribute/ câmpuri/ variabile instanță**) definite în cadrul clasei;
- ▶ O clasă definește o serie de funcții (**metode/ operații**) care aplicate unui obiect definesc **interfața** acestuia;
- ▶ Datele obiectelor sunt ascunse și pot fi accesate doar prin funcții definite în interiorul clasei => **încapsulare**;
- ▶ **Starea** unui obiect este definită prin atributele sale;
- ▶ **Comportamentul** unui obiect este definit prin metodele sale;
- ▶ Conceptul de **transmitere a unui mesaj unui obiect** este echivalent cu apelul unei metode.

Clase

- Sintaxa pentru definirea unei clase:

```
[attributes][access_modifier] class class_name [extends  
base_class][implements interface1, interface2, ...]  
{  
    access_modifier attribute1; //instance variable  
    access_modifier attribute2; //instance variable  
    ...  
    access_modifier method1;  
};
```

Clase

Modificatorii de acces:

- ▶ **public**
- ▶ **private**
- ▶ **protected**
- ▶ **default**

Atributele:

- ▶ **final**
- ▶ **abstract**

Clase

Instance members & methods visibility for access modifiers

Visibility	Public	Protected	Private	Default
Same class	X	X	X	X
Class in same package	X	X		X
Subclass in same package	X	X		X
Subclass in other package	X	X		
Class outside the package	X			

Clase

Cuvinte cheie:

- ▶ **extends:** permite derivarea clasei dintr-o altă clasă de bază;
- ▶ **implements:** permite derivarea clasei din una sau mai multe interfețe.

Clase

Atributele clasei:

- ▶ variabile instanță sau atribute ale obiectelor;
- ▶ variabile statice (un fel de variabile globale).

Metodele clasei:

- ▶ funcții constructor;
- ▶ funcții accesori (get() și set());
- ▶ alte metode de prelucrare.

Clase

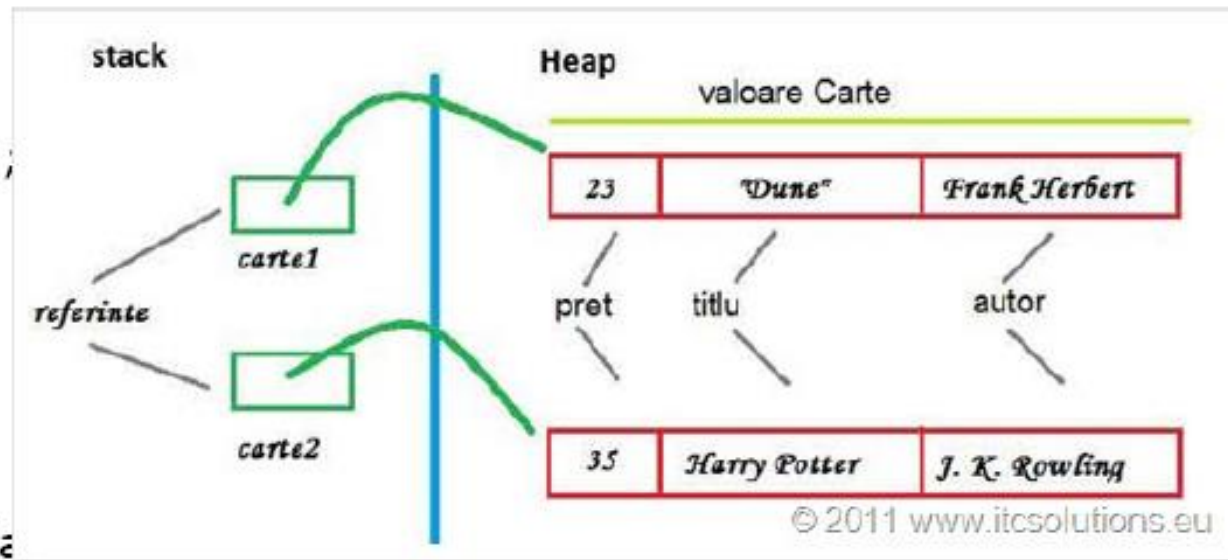
- ▶ Într-un fișier Java (.java) pot fi definite mai multe clase;
- ▶ Într-un fișier Java (.java) poate fi definită o singură clasă publică;
- ▶ Fișierul sursă Java conținând clasa publică poartă denumirea clasei publice;
- ▶ Prin compilarea unui fișier sursă Java ce include mai multe clase se obțin fișiere bytecode (.class) pentru fiecare clasă.

Clase

- ▶ Crearea obiectelor se face prin apelul operatorului **new** care apelează constructorul clasei
- ▶ Accesarea metodelor și atributelor unui obiect se realizează cu ajutorul operatorului punct (.)

Clase

```
class Book{  
    //define attributes - instance variables  
    float price;  
    String title;  
    String author;  
}  
...
```



```
Book book1 = new Ca  
book1.price = 23;  
book1.title = "Dune";  
book1.author = "Frank Herbert";
```

Clase

Atributele **constante**:

- ▶ definite cu **final**;
- ▶ nu se poate modifica valoarea acestora odată ce au fost inițializate;

```
class Test
{
    public final int attribute_1 = 10;
    //MUST be initialized in constructor
    public final int attribute_2;
}
```

Clase

Atributele **constante**:

- ▶ sunt inițializate în constructor sau la definire (dar nu în ambele situații);
- ▶ sunt echivalente variabilelor constante din C++.

Clase

Atributele **statice**:

- ▶ sunt definite prin cuvântul **static**;
- ▶ reprezintă atribute ce nu aparțin unui obiect;
- ▶ pot fi definite și **final**;
- ▶ inițializarea se face la definire sau într-un bloc de inițializare;
- ▶ sunt considerate variabile definite la nivelul clasei (un fel de variabile globale);
- ▶ sunt accesate prin numele clasei.

Clase

static attributes:

syntax:

```
class Test
{
    public static int attribute_1 = 10;
    public static final int attribute_2;
    //initialisation block
    //when is executed ?
    {
        attribute_2 = 45;
    }
}
```

Clase

Funcții statice:

- ▶ Sunt funcții care nu aparțin unui obiect;
- ▶ Sunt “funcții globale” care aparțin unei clase de obiecte;
- ▶ Au acces doar la ceilalți membri statici ai clasei;
- ▶ Sunt apelate prin numele clasei;
- ▶ Nu pot avea referința *this*.

Clase

Funcții membre:

- ▶ Constructori (se poate implementa inclusiv constructor de copiere);
- ▶ Nu există destructor (doar funcțai finalize());
- ▶ Funcții de acces (get și set);
- ▶ Nu se pot supraîncărca operatori ca în C++.

Clase

Constructors:

- syntax:

```
class class_Name {  
    public class_Name( ) {...}  
};
```

- use (because objects are managed by references, an object is created using *new* operator):

```
public static void main () {  
    class_Name object_1 = new class_Name();  
    class_Name object_2 = new class_Name(input params)  
}
```

Clase

Copy constructor:

- **syntax:**

```
class class_name {  
    public class_name(class_name existing_object) { ... }  
};
```

- **use:**

```
public static void main () {  
    class_name object_1 = new class_name(...);  
    class_name object_2 = new class_name(object_1);  
}
```

Clase

Operatorul =

- ▶ Copiază bit cu bit valoarea sursei în zona de memorie destinație (cele două zone sunt identice ca tip și structură);
- ▶ În cazul obiectelor unei clase, copiază valoarea referinței obiectului sursă în referința obiectului destinație.

Clase

Proprietățile (getteri și setteri):

- ▶ Oferă acces (citire/scriere) la attributele private ale clasei;
- ▶ Presupun validarea datelor de intrare;
- ▶ Sunt definite în zona publică a clasei;
- ▶ Definite prin două metode prefixate cu **get** pentru citire și **set** pentru scriere.

Clase

```
class Test {  
    private String name;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Shallow copy vs. Deep copy

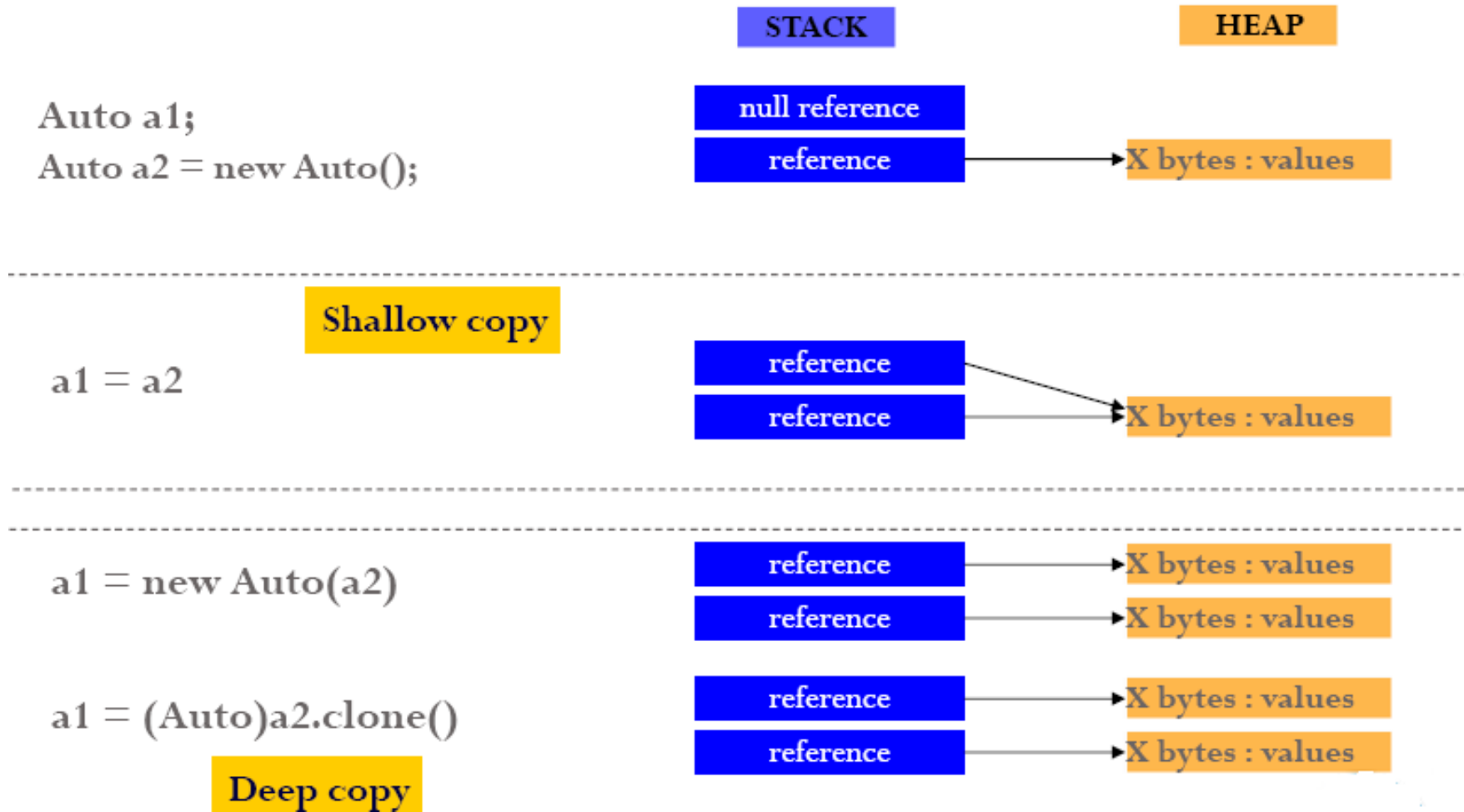
Shallow copy:

- ▶ Copiază valorile a două referințe între ele;
- ▶ Realizat implicit prin operatorul egal “=”;

Deep copy:

- ▶ Copiază valorile obiectelor (nu referințele);
- ▶ Realizat prin metode speciale cum ar fi constructorul de copiere, metoda clone().

Shallow copy vs. Deep copy



Moștenire (derivare)

- ▶ Este implementată atunci când există o relație de tipul “is a” între o subclasă și o clasă de bază;
- ▶ Se poate moșteni o singură clasă de bază;
- ▶ Apelul constructorului clasei de bază se face cu *super*.

```
class SpecialProduct extends Product
{
    private float _discount;
    ...
}
```

```
public SpecialProduct(double cost, float profit, float discount)
{
    super(cost,profit);
    if(discount>0) _discount=discount;
}
```

Moștenire (derivare)

through inheritance the subclass gets all the methods and attributes

```
class Base{
```

```
    int attribute1;
```

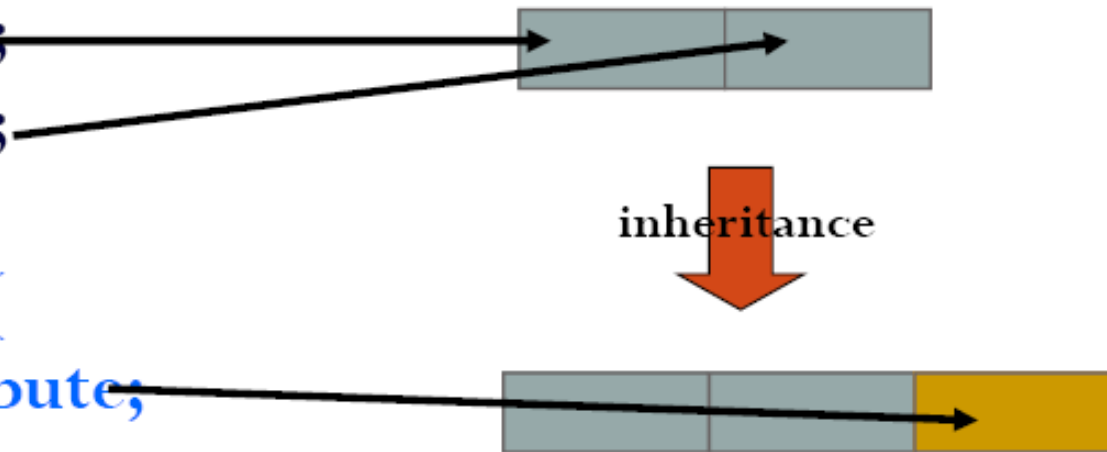
```
    int attribute2;
```

```
};
```

```
class Subclass: Base{
```

```
    int new_attribute;
```

```
};
```

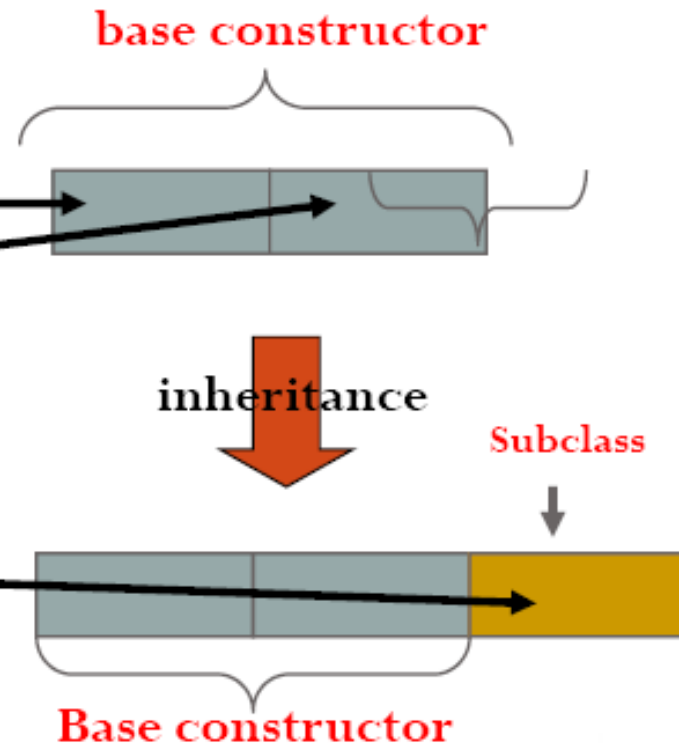


Moștenire (derivare)

each constructor manages the area of its class

```
class Base{  
    int attribute1;  
    int attribute2;  
};
```

```
class Subclass: Base{  
    int new_attribute;  
};
```




Moștenire (derivare)

in the base class and the subclass you can define methods with the same header - overloading

```
class Base{  
    int Method1(int a){...}  
};  
class Subclass: Base{  
    int attribute_new;  
    int Method1(int a){...}  
    int Method2(int a){ super.Method1(a);}  
};
```

call to the base class



Polimorfism

Overriding

```
class Person {  
    public string SaySomething() {...}  
}
```



same signature

```
class Student extends Person {  
    public string SaySomething() {...}  
}
```

Overloading

```
class Person {  
    public void Eat() {...}  
    public void Eat(Food food) {...}  
}
```



different parameters list

```
class Student extends Person {  
    public void Eat(int calories) {...}  
    public void Eat(string foodName) {...}  
    public string Eat(string foodName,  
        string drinkName) {...}  
}
```

Static Binding vs. Dynamic Binding

Există două tipuri de binding:

- ▶ **statică** (cunoscută și sub denumirea de legare **timpurie**).
- ▶ **dinamică** (cunoscută și sub numele de legare **târzie**).

Static Binding vs. Dynamic Binding

- ▶ Când tipul obiectului este determinat la compilare (de către compilator), acesta este cunoscut sub numele de legare statică.
- ▶ Dacă există vreo metodă privată, finală sau statică într-o clasă, există o legare statică.

```
class Dog{  
    private void eat(){System.out.println("dog is eating...");}  
  
    public static void main(String args[]){  
        Dog d1=new Dog();  
        d1.eat();  
    }  
}
```

Static Binding vs. Dynamic Binding

- ▶ Când tipul obiectului este determinat în timpul rulării, acesta este cunoscut sub numele de legare dinamică.

```
class Animal{  
    void eat(){System.out.println("animal is eating...");}  
}
```

```
class Dog extends Animal{  
    void eat(){System.out.println("dog is eating...");}  
  
    public static void main(String args[]){  
        Animal a=new Dog();  
        a.eat();  
    }  
}
```


Clase abstracte

Metode virtuale pure (abstracte):

- ▶ Funcții virtuale care nu au implementare în clasa părinte;
- ▶ Clasa părinte trebuie să fie abstractă;
- ▶ Clasa derivată trebuie să le implementeze dacă nu este abstractă.

Clase abstracte

PURE VIRTUAL methods - ABSTRACT:

```
abstract class Superclass{  
public abstract int Method1(int a) ;  
};
```

```
class Subclass extends Superclass{  
public int Method1(int a){...}  
};
```

Clase abstracte

Clasele abstracte:

- ▶ Conțin cel puțin o metodă abstractă (virtuală pură), dar nu este obligatoriu;
- ▶ Pot conține attribute și alte metode;
- ▶ Sunt o interfață pentru alte clase care trebuie să implementeze un set de metode comune.

Clase abstracte

Abstract classes:

- you can't instantiate an abstract class;
- used for class hierarchies

```
abstract class AbstractClass{  
    int attribute1;  
    public abstract int Method1(int a);  
};  
....  
AbstractClass ba1;  
AbstractClass ba1 = new AbstractClass();  
}
```

Interfețe

- ▶ Interfețele sunt clase abstracte care conțin doar funcții abstracte;
- ▶ Sunt o interfață pentru alte clase care trebuie să implementeze un set de metode comune;
- ▶ Sunt definite prin cuvântul cheie *interface*.


Interfețe

Interfaces:

```
interface IOperations {  
    void Operation1( );  
    void Operation2( );  
};
```

override interface method

```
class Base implements IOperations {  
    public void Operation1() {...}  
    public void Operation2() {...}  
}
```



Interfețe

Abstract classes

- contain abstract methods + attributes + non-abstract methods
- a class can extend only one base class (abstract or not)
- can be used as *reference type*

vs. Interfaces

- contain only abstract methods
- a class can implement on or many interfaces
- can be used as *reference type*

Bibliografie

- ▶ [1] Jonathan Knudsen, Patrick Niemeyer - *Learning Java*, 3rd Edition, O'Reilly.
- ▶ [2] <http://www.itcsolutions.eu>
- ▶ [3] <http://www.acs.ase.ro>
- ▶ [4] <http://docs.oracle.com/javase/tutorial/index.html>