

PROGRAMARE AVANSATĂ PE OBIECTE

Conf.univ.dr. Ana Cristina DĂSCĂLESCU





Temtică curs 4

- Agregare și compoziție
- Prelucrarea șirurilor de caractere
- Principilu de desing pentru clasă imutabilă
- Record

AGREGARE ȘI COMPOZIȚIE



- **Agregarea** și **compoziția** reprezintă alte două modalități de interconectare (asociere) a două clase, alături de mecanismul de extindere a claselor (moștenire).
 - Asocierea a două clase se realizează prin încapsularea în **clasa container** a unei referințe, de un tip diferit, către un obiect al **clasei asociate** (încapsulate).
 - **Sintaxa:**

```
class Container
{
    Referinta dataMembra;
}
```



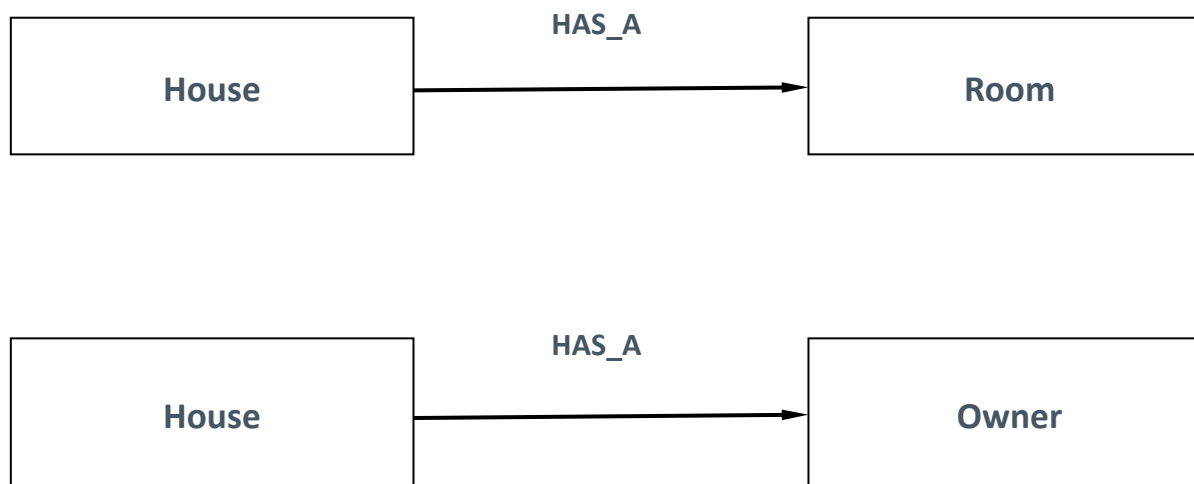
AGREGARE ȘI COMPOZIȚIE

- Conceptual, compoziția este diferită de agregare în raport **de ciclul de viață al obiectului încapsulat**, astfel:
 - ✓ dacă ciclul de viață al obiectului încapsulat este dependent de ciclul de viață al obiectului container, atunci relația de asociere este de tip **compoziție (strong association)**;
 - ✓ dacă obiectul încapsulat poate să existe și după distrugerea containerului său, atunci relația de asociere este de **tip agregare (weak association)**.
- Compoziția și agregarea sunt relații de tip HAS_A, care se folosesc în momentul în care dorim să reutilizăm o clasă existentă



AGREGARE ȘI COMPOZIȚIE

■ Exemplu





■ Implementare agregare

```
class Person{  
    private String name;  
    private String SSN;  
    .....  
}
```

```
class House{  
    private String address;  
    private Person owner;  
    .....  
  
    public House(Person owner,...){  
        this.owner = owner;  
        .....  
    }  
}
```



■ Implementare compoziție

```
class Room{  
    private float width;  
    private float length;  
    .....  
  
    public Room(Room r){  
        this.width = r.width;  
        .....  
    }  
}
```

```
class House{  
    private String address;  
    private Room dining;  
    .....  
  
    public House(Room dining,...){  
        this.dining = new Room( );  
        .....  
    }  
}
```



ȘIRURI DE CARACTERE

- În limbajul Java sunt predefinite 3 clase pentru manipularea la nivel înalt a șirurilor de caractere:
 1. clasa `String`
 2. clasa `StringBuilder`
 3. clasa `StringBuffer`

- De asemenea, șirurile de caractere poate fi implementate și manipulate direct, prin intermediul `tablourilor cu elemente de tip char`



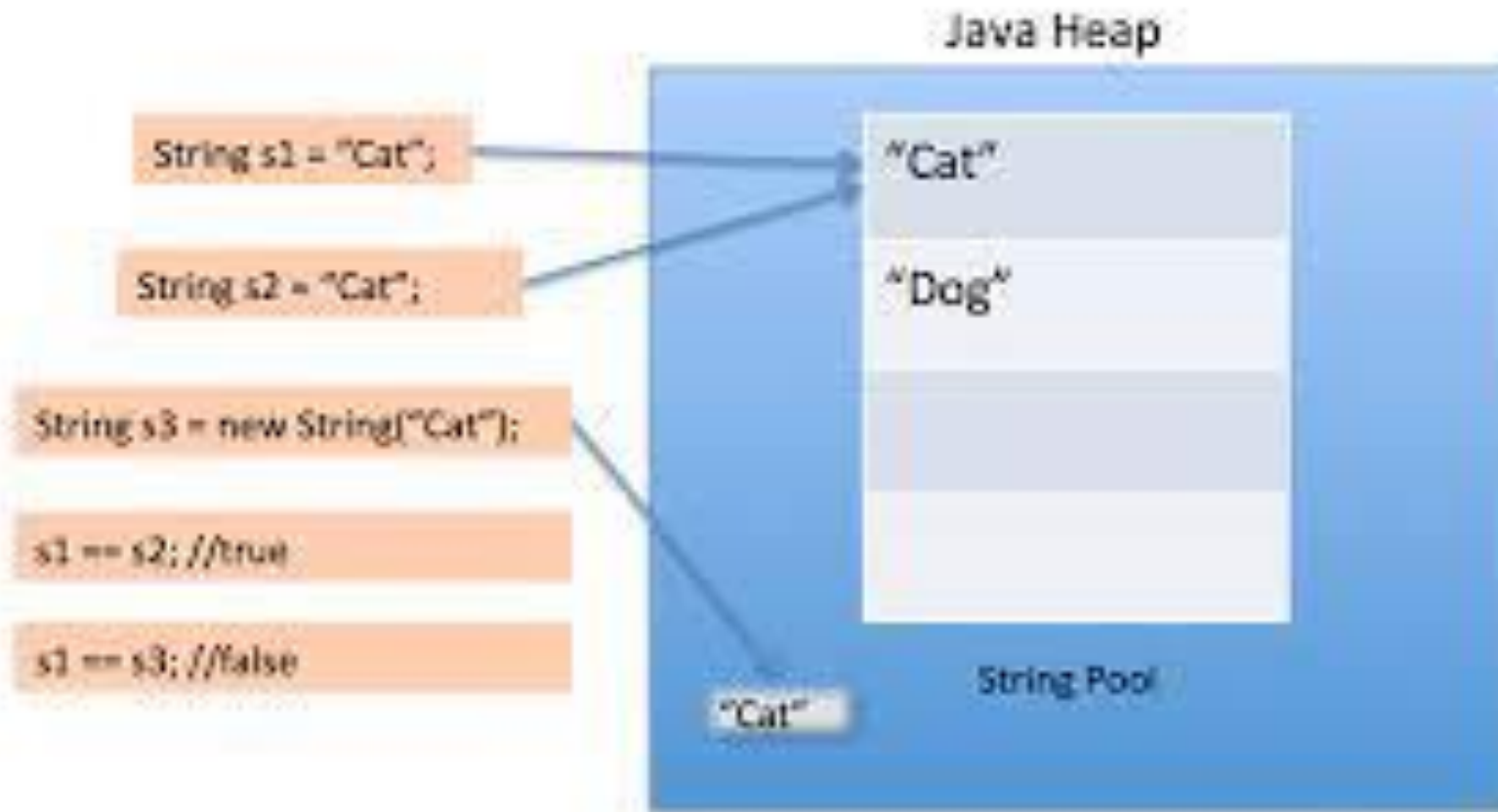
CLASA STRING

- Folosind clasa String, un șir de caractere poate fi instanțiat în două moduri:

1. `String s = "exemplu";`
2. `String s = new String("exemplu");`

- Diferența dintre cele două metode constă în zona de memorie în care va fi alocat șirul respectiv:
 1. Se va utiliza o zona de memorie specială, numită **tabelă de șiruri (string literal/constant pool)**.
 2. Se va utiliza zona de memorie **heap**.

TABELA DE LITERALI





■ Observații

- Operația de comparare a două șiruri din tabela de literali, din punct de vedere al conținuturilor lor, se poate realiza direct, prin compararea referințelor celor două șiruri, utilizând operatorul `==`.
- Un șir de caractere alocat dinamic, folosind operatorul `new`, poate fi plasat în tabela de șiruri folosind metoda `String intern()`:

```
String sir_1 = "exemplu";  
String sir_2 = new String("exemplu");  
System.out.println(sir_1 == sir_2);           // se va afișa false  
sir_2 = sir_2.intern();  
System.out.println(sir_1 == sir_2);           // se va afișa true
```



- Odată creat un șir de caractere, conținutul său nu mai poate fi modificat!!!!
- Orice operație de modificare a conținutului său va conduce la construcția unui alt șir! Astfel, după executarea secvenței de cod:

```
String sir_1 = "programare";
```

```
sir_1.toUpperCase();
```

va crea un nou șir având
conținutul PROGRAMARE

```
System.out.println(sir_1);
```

programare

- Dacă instanțele unei clase nu mai pot fi modificate din punct de vedere al conținutului după ce au fost create, atunci clasă este o **clasă imutabilă**.



- Clasa **String** pune la dispoziția programatorilor metode pentru:
 1. determinarea numărului de caractere:
 - `int length()`
 2. extragerea unui subșir:
 - `String substring(int beginIndex)`
 - `String substring(int beginIndex, int endIndex)`
 3. extragerea unui caracter:
 - `char charAt(int index)`
 4. compararea lexicografică a două șiruri:
 - `int compareTo(String anotherString)`
 - `boolean equals(Object anotherObject)`
 - `boolean equalsIgnoreCase(String anotherString)`



4. transformarea tuturor literelor în litere mici sau în litere mari:

- `String toLowerCase()`
- `String toUpperCase()`

5. reprezentarea unei valori de tip primitiv sau a unui obiect sub forma unui șir de caractere:

- `static String valueOf(boolean b)`
- `static String valueOf(char c)`

- Informații detaliate despre toate metodele din clasa `String` pot fi găsite în pagina: <https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>.



- Clasa `String` există mai multe metode care necesită utilizarea unor *expresii regulate* (regex).
- O *expresie regulată* (regex) este o secvență de caractere prin care se definește un șablon de căutare.
- De obicei, expresiile regulate se utilizează pentru a testa validitatea datelor de intrare (de exemplu, pentru a verifica dacă un șir conține un CNP formal corect) sau pentru realizarea unor operații de căutare/înlocuire/parsare într-un șir de caractere.



➤ Câteva reguli uzuale pentru definirea unei expresii regulate sunt următoarele:

[abc] – șirul este format doar dintr-una dintre literele a sau b sau c

[^abc] – șirul este format din orice caracter, mai puțin literele a, b și c

[a-z] – șirul este format dintr-o singură literă mică

[a-zA-Z] – șirul este format dintr-o singură literă mică sau mare

[a-z][A-Z] – șirul este format dintr-o literă mică urmată de o literă mare

[abc]+ – șirul este format din orice combinație a literelor a, b și c, iar lungimea sa este cel puțin 1

[abc]{5} – șirul este format din orice combinație a literelor a, b și c de lungime exact 5

[abc]{5,} – șirul este format din orice combinație a literelor a, b și c de lungime cel puțin 5

[abc]{5,10} – șirul este format din orice combinație a literelor a, b și c cu lungimea cuprinsă între 5 și 10



- Exemple de utilizare a metodelor care necesită expresii regulate:
 1. pentru a verifica dacă un șir de caractere are o anumită formă particulară se folosește metoda **boolean matches(String regex)**:
 - a) șirul s începe cu o literă mare, apoi conține doar litere mici (cel puțin una!):

```
boolean ok = s.matches("[A-Z][a-z]+");
```
 - b) șirul s conține doar cifre:

```
boolean ok = s.matches("[0-9]+");
```
 - c) șirul s conține un număr de telefon Vodafone:

```
boolean ok = s.matches("(072|073)[0-9]{7}");
```



2. pentru a împărți un șir `s` în subșiruri (stocate într-un tablou de șiruri), în raport de anumiți delimitatori, folosind metoda

```
String[] split(String regex):
```

a) împărțirea textului în caractere:

```
> String[] w = s.split("");
```

b) împărțirea textului în cuvinte de lungime nenulă:

```
> String[] w = s.split("[ .,:;! ?]+");
```



- Obiectele de tip `StringBuilder` sunt mutabile, deci pot fi direct modificate.
- Intern, obiectele de tip `StringBuilder` sunt alocate în zona de memorie heap și sunt tratate ca niște tablouri de caractere. Dimensiunea tabloului se modifică dinamic, pe măsură ce șirul este construit (inițial, șirul are o lungime de 16 caractere):

```
StringBuilder sb = new StringBuilder();  
sb.append("exemplu");
```

- Deoarece nu sunt imutabile, șirurile de tip `StringBuilder` nu sunt thread-safe!



- Clasa `StringBuilder` conține, în afara unor metode asemănătoare celor din clasa `String`, mai multe metode specifice:
 - modificarea lungimii șirului prin trunchiere sau extindere cu caracterul `'\u0000'`:

```
void setLength(int newLength)
```
 - adăugarea la sfârșitul șirului a unor caractere obținute prin conversia unor valori de tip primitiv sau obiecte:

```
StringBuilder append(int x)
StringBuilder append(Object obj)
StringBuilder append(String str)
StringBuilder append(StringBuffer sb)
```
- inserarea în șir, începând cu poziția `offset`, a unor caractere obținute prin conversia unor valori de tip primitiv sau obiecte:
 - `StringBuilder insert(int offset, boolean b)`
 - `StringBuilder insert(int offset, String str)`



- Singura diferență dintre clasa `StringBuilder` și clasa `StringBuffer` constă în faptul că aceasta este thread-safe, adică metodele sale sunt sincronizate, fiind executate pe rând, sub excludere reciprocă! Din acest motiv, metodele sale sunt mai lente decât cele echivalente din clasa `StringBuilder`.