PROGRAMARE ORIENTATĂ PE OBIECTE

GENGE BÉLA

Capitolul 3
Tablouri

Definiții

Tabloul reprezintă o structură omogenă cu:

- Același tip de elemente.
- Elemente plasate într-o zonă continuă de memorie.
- Index-ul primului element este 0.

Definiții

Diferența majoră față de C/C++:

- În Java tabloul este un obiect.
- Limitele sunt verificate.
- Elementele sunt inițializate cu:
  - Valori numerice: 0.
  - Caractere: 0 unicode.
  - boolean: false.
  - Referință: null.

Exemple

Tablourile pot fi construite din tipuri primitive și din tipuri referință.

Exemple:

- int a[];
- int[] a;
- int a[], b;
- int[] a, b;

Tablouri bidimensionale

Exemple declarare:

- int t[][] = new int[3][4];
- int t[][] = new int[2][];
  - t[0] = new int[3]
  - t[1] = new int[10]

Accesarea elementelor

Interogarea numărului de elemente:

- Atributul special length.
- Generat automat de mașina virtuală.

Exemplu declarare tablou de întregi și inițializare.

Exerciții

- Să se citească de la tastatură *n* numere întregi și să se stocheze într-un tablou unidimensional (*n* citit de la tastatură).
- Să se încapsuleze conceptul Stivă statică folosind tablouri unidimensionale.
- Să se citească de la tastatură și să se stocheze *n* șiruri de caractere într-un tablou unidimensional (*n* citit de la tastatură).
- Să se citească de la tastatură informațiile specifice conceptului Persoana și să se stocheze într-un tablou *n* instanțe Persoana.
  - Să se determine dacă o persoană (după CNP citit de la tastatură) există în tablou.

String și StringBuffer

Tipul String se utilizează pentru șiruri de caractere care nu se modifică.

- Exemplu:
  - String s = “”;
  - s += “abc”;

Tipul StringBuffer permite modificări asupra șirurilor de caractere.

- Exemplu:
  - StringBuffer s = “”;
  - s.append(“abc”);

Metoda toString()

- Definită în clasa de bază Object.
- În Java fiecare clasă trebuie să suprascrie metoda toString().
- Returnează o reprezentare sub forma unui șir de caractere a stării obiectului.
- Exemplu:
  - Extinderea clasei Persoana cu metoda toString().

Expresii regulate

- O secvență de caractere ce formează un șablon de căutare.

Definirea expresiilor regulate:

- Pachetul java.util.regex.
- Include două clase:
  - Pattern.
  - Matcher.

Expresii regulate – Clasa Pattern

- Asigură construirea șablonului de căutare.
- Construcții principale:

Characters

x	The character *x*
\\	The backslash character
\0*n*	The character with octal value 0*n* (0 <= *n* <= 7)
\0*nn*	The character with octal value 0*nn* (0 <= *n* <= 7)
\0*mnn*	The character with octal value 0*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7)
\x*hh*	The character with hexadecimal value 0x*hh*
\u*hhhh*	The character with hexadecimal value 0x*hhhh*
\x{*h...h*}	The character with hexadecimal value 0x*h...h* (Character.MIN_C
\t	The tab character ('\u0009')
\n	The newline (line feed) character ('\u000A')
\r	The carriage-return character ('\u000D')
\f	The form-feed character ('\u000C')
\a	The alert (bell) character ('\u0007')
\e	The escape character ('\u001B')
\c*x*	The control character corresponding to *x*

Expresii regulate – Clasa Pattern

- Asigură construirea șablonului de căutare.
- Construcții principale:

Character classes

`[abc]`	a, b, or c (simple class)
`[^abc]`	Any character except a, b, or c (negation)
`[a-zA-Z]`	a through z or A through Z, inclusive (range)
`[a-d[m-p]]`	a through d, or m through p: `[a-dm-p]` (union)
`[a-z&&[def]]`	d, e, or f (intersection)
`[a-z&&[^bc]]`	a through z, except for b and c: `[ad-z]` (subtraction)
`[a-z&&[^m-p]]`	a through z, and not m through p: `[a-lq-z]`(subtraction)

Expresii regulate – Clasa Pattern

- Asigură construirea șablonului de căutare.
- Construcții principale:

Predefined character classes

`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`

Expresii regulate – Clasa Pattern

- Asigură construirea șablonului de căutare.
- Construcții principale:

Boundary matchers

^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

Expresii regulate – Clasa Pattern

- Asigură construirea șablonului de căutare.
- Construcții principale:

Greedy quantifiers

X?	*X*, once or not at all
*X**	*X*, zero or more times
X+	*X*, one or more times
X{*n*}	*X*, exactly *n* times
X{*n*,}	*X*, at least *n* times
X{*n*,*m*}	*X*, at least *n* but not more than *m* times

Expresii regulate – Clasa Pattern

- Asigură construirea șablonului de căutare.
- Construcții principale:

Logical operators

XY	*X* followed by *Y*
X|*Y*	Either *X* or *Y*
(*X*)	X, as a capturing group

Special constructs (named-capturing and non-capturing)

(?<name>*X*)	*X*, as a named-capturing group
(?:*X*)	*X*, as a non-capturing group
(?idmsuxU-idmsuxU)	Nothing, but turns match flags i d m s u x U on - off
(?idmsux-idmsux:*X*)	*X*, as a non-capturing group with the given flags i d m s u x on - off
(?=*X*)	*X*, via zero-width positive lookahead
(?!*X*)	*X*, via zero-width negative lookahead
(?<=*X*)	*X*, via zero-width positive lookbehind
(?<!*X*)	*X*, via zero-width negative lookbehind
(?>*X*)	*X*, as an independent, non-capturing group

Expresii regulate – Clasa Match

- Implementează operațiile de căutare asupra șablonului definit prin clasa Pattern.
- Metode uzuale:
  - find(): boolean – caută următoarea apariție pentru șablonul căutat și returnează true în caz de succes.
  - start(): int – returnează poziția de start a rezultatului căutării.
  - end(): int – returnează poziția de sfârșit a rezultatului căutării.
  - group(): String – returnează secvența găsită.
  - group(int group): String – returnează secvența găsită pentru grupul dat.
  - group(String name): String – returnează secvența găsită pentru grupul dat.

Expresii regulate – exerciții

- Fie textul: “aab aaa aaaaaab ab b”.
- Se cere: Să se afișeze secvențele de caractere care includ cel puțin o literă ‘a’ și se termină cu o singură literă ‘b’.

Expresii regulate – exerciții

- Soluția:

```
String s = "aab aaa  aaaaaab ab b";
Pattern p = Pattern.compile("a+b");
Matcher m = p.matcher(s);
while (m.find()){
        System.out.println("Start: " + m.start() + ", End: " + m.end());
        System.out.println(s.substring(m.start(), m.end()));
}
```

Expresii regulate – exerciții

- Scrieți expresia regulată pentru căutarea CNP-ului.

- Fie textul: “abcd 2014-05-24 ….. 1982-12-30 asd asda”
- Se cere: scrieți expresia regulată pentru afișarea tuturor datelor calendaristice.

Expresii regulate – exerciții

- O primă soluție (trebuie îmbunătățită):

```
String s = "abcd 2014-05-24 ….. 1982-12-30 asd asda";
Pattern p = Pattern.compile("[1|2][0-9]{3}-[0|1][1-9]-[0-3][0-9]");
Matcher m = p.matcher(s);

while (m.find()){
        System.out.println("Start: " + m.start() + ", End: " + m.end());
        System.out.println(s.substring(m.start(), m.end()));
}
```

Expresii regulate – exerciții

- Fie textul: “class A{ … class Abc {….. class AbcEfd {”
- Se cere: scrieți expresia regulată pentru afișarea tuturor denumirilor de clase.

Expresii regulate – exerciții

- Soluția:

```
String s = "class A{ …  class Abc  {…..  class   AbcEfd {";
Pattern p = Pattern.compile("class[ \\t]+([A-Z][a-zA-Z]*)[ \\t]*");
Matcher m = p.matcher(s);

while (m.find()){
        System.out.println("Start: " + m.start() + ", End: " + m.end());
        System.out.println(s.substring(m.start(), m.end()));
}
```

Exerciții

- Fie codul dat într-un șir de caractere: JFrame f = new JFrame(“ Titlu fereastra 123 _ … ”);
- Scrieți o aplicație în Java care afișează:
  - Denumirile variabilelor JFrame (ex: f)
  - Denumirile ferestrelor create(ex: Titlu fereastra 123 _ …)