JDK. Mediul de programare Eclipse. Aplicatii simple in Java. Tablouri de variabile, clase pt. siruri de caractere.

JDK. The Eclipse programming environment. Simple Java applications. Arrays of variables, classes for arrays of characters.

Obiective:
- JDK
- implementarea unor aplicatii simple in Java
- compilarea si rularea aplicatiilor Java
- familiarizarea cu mediul de programare Eclipse
- implementarea aplicatiilor Java care lucreaza cu siruri de variabile si cu date de tip *String*, *StringBuffer*, *StringBuilder*, *StringTokenizer*

Objectives:
- JDK
- implementing simple Java applications
- compiling and running Java applications
- getting familiar with the Eclipse programming environment
- implementing Java applications that work with arrays of variables and *String*, *StringBuffer*, *StringBuilder*, *StringTokenizer* data

JDK - Java Developer's Kit
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Principalele utilitare Java sunt:

- *javac (Java compiler)* => compilatorul Java, transformă fişierele sursă **.java* în fişiere bytecode *.*class*.

  javac nume_fişier.java

- *java (Java interpreter)* => interpretorul care lansează în execuţie bytecode-urile Java.

  java nume_clasa_main

- *appletviewer (Java applet viewer)* => utilitar care permite execuţia applet-urilor, în ipoteza în care acestea nu sunt încărcate dintr-un browser de Web. Pentru folosirea acestui utilitar, este necesară existenţa unui fişier *html* adecvat.

  appletviewer nume_fişier.html

- *javadoc (Java API documentaţion generator)* => creează documentaţia programelor sursa Java în format *html*. Documentaţia este creeată pe baza comentariilor din program, menţionate sub forma */**… */*

  javadoc nume_fişier.java

- *javah (Java header and stub file generator)* => creează fişiere antet în limbajul C şi fişiere primare din clasele Java ce permite interacţiunea dintre codul Java şi codul C.
- *javap (Java class file disassembler)* => dezasamblează fişierele bytecode **.class* şi afişează o reprezentare a codului de octeţi.

javap nume_fişier.class

- *jdb (Java language debugger) =>* instrument pentru urmărirea şi detectarea erorilor din cod.

Paşii pe care trebuie să-i parcurgă un programator cu scopul de a rula un program Java:

1. ***scrierea codului sursă*** înseamnă crearea unui fişier **.java* care conţine în interior instrucţiuni Java. Prin comparaţie, putem asemăna acest lucru cu crearea fişierelor **.c* sau **.cpp* în C++. Reamintim că nu este nevoie de un editor specializat pentru scrierea acestor fişiere sursă, programatorul putând folosi orice editor de texte în acest scop (*Notepad, etc.*).
2. ***compilarea*** are că efect "traducerea" instrucţiunilor scrise în fişierul sursă în format bytecode şi stocarea lor într-unul sau mai multe fişiere **.class* funcţie de câte clase au existat în programul sursă. Numele fişierului *class* care va fi executat este cel al clasei care conţine metoda *main* şi poate să difere de numele textului sursă dacă clasa nu este publică. Fişierul bytecode este creeat numai în ipoteza în care compilatorul nu a găsit greşeli în fişierul sursă.
3. ***interpretarea şi lansarea în execuţie*** este transformarea fişierului sau fişierelor *class* în cod-maşină, urmata de rularea efectivă a aplicaţiei.

Compilarea fişierului sursă se realizează cu ajutorul utilitarului *javac.exe*. Sintaxa din linia de comandă este următoarea:

javac MyApplication.java

Efectul este lansarea compilatorului. După terminarea analizei codului sursă, pot apărea două situaţii:

- dacă au existat erori, acestea sunt afişate pe terminal. Se specifică tipul erorii, linia unde a fost detectată eroare şi se indică care componentă din linia respectivă de cod a creeat probleme.
- dacă nu a fost detectată nici o eroare la compilare, este creeat fişierul *MyApplication.class* existând în fişierul sursă o clasă cu acest nume unde este localizată metoda *main*.

Interpretarea şi lansarea în execuţie se face diferit în funcţie de natura aplicaţiei. Dacă este vorba de o aplicaţie de sine stătătoare, se foloseşte următoarea instrucţiune din linia de comandă:

java MyApplication

Tablouri de variabile (unidimensionale, multidimensionale)

Alăturări de elemente de același tip (date primitive sau instanțe de clasă), accesate pe baza indexului lor.

Declarare: *tip_date nume_variabila[];*
Alocare memorie: *nume_variabila = new tip_date[nr_elemente];*

Accesul la elemente se face pe baza indexului (int);

Proprietatea *.length* => numărul de elemente din șir

Clasa *Arrays*

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

=> operații pe variabile de tip șir (căutări, sortări, copieri, etc.)

Clasa String

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

=> permite tratarea unitara a şirurilor de caractere; toate variabilele literale Java care conţin o secvenţă de caractere pot fi prelucrate prin intermediul acestei clase.

Exemplu:

```
//declararea şi iniţializarea unei variabile de tip String
String sir = new String("abc");
```

! Variabilele de tip *String* nu pot fi modificate odată ce au fost iniţializate.

Clasa *String* are constructori care permit:

- iniţializarea unui obiect de tip *String* cu un şir vid de caractere
- iniţializarea unui obiect de tip *String* cu un şir de caractere primit ca parametru
- iniţializarea unui obiect de tip *String* cu un şir de *bytes* primit ca parametru
- iniţializarea unui obiect de tip *String* cu un alt obiect de tip *String* primit ca parametru
- iniţializarea unui obiect de tip *String* cu un alt obiect de tip *StringBuffer* primit ca parametru

Clasa *String* include metode pentru:

- examinarea individuală a fiecărui caracter din secvenţa stocată
- compararea a doua variabile de tip *String*
- căutarea unei anumite porţiuni dintr-un *String*
- extragerea unei anumite porţiuni dintr-un *String*
- crearea unei copii a unei variabile de tip *String*, cu sau fără transformarea caracterelor din majuscule în minuscule
- determinarea lungimii şirului de caractere conţinut de un obiect de tip *String*
- etc.

Java pune la dispoziţie un mecanism prin care se permite operatorului *+* să fie folosit pentru concatenarea variabilelor de tip *String*. Se permite de asemenea concatenarea variabilelor de tip *String* cu variabile de alte tipuri, inclusiv cele elementare (*byte*, *short*, *int*, etc.). Acest lucru este posibil datorită faptului că orice clasă Java este considerată ca fiind derivată (direct sau indirect din clasa *Object*) iar aceasta poate transforma în variabile *String* datele conţinute (metoda *toString*).

Exemplu:

```
String s1 = new String("abc");
String s2 = s1 + "def";       //şirul "def" este convertit automat în String
String s3 = s1 + 5;//se concatenează şirul iniţial cu şirul de caractere "5"
System.out.println(s1 + s2 + 100.23f);
//etc.
```

Clasa StringBuffer

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html

=> *permite*, ca şi clasa *String* tratarea unitară a şirurilor de caractere; toate variabilele literale Java care conţin o secvenţă de caractere pot fi prelucrate prin intermediul acestei clase.

Exemplu:

```
//declararea şi iniţializarea unei variabile de tip StringBuffer
StringBuffer sb = new StringBuffer("abc");
```

! Spre deosebire de o variabilă de tip *String*, un obiect de tip *StringBuffer* poate fi modificat pe parcursul existenţei sale.

Clasa *StringBuffer* are constructori care permit:

- iniţializarea unui obiect de tip *StringBuffer* cu un şir vid de caractere
- iniţializarea unui obiect de tip *StringBuffer* cu un şir vid de caractere de o anumită lungime
- iniţializarea unui obiect de tip *StringBuffer* cu un alt obiect de tip *String* primit ca parametru

Clasa *StringBuffer* include metode pentru:

- examinarea individuală a fiecărui caracter din secvenţa stocată
- compararea a doua variabile de tip *StringBuffer*
- căutarea unei anumite porţiuni dintr-un *StringBuffer*
- extragerea unei anumite porţiuni dintr-un *StringBuffer*
- crearea unei copii a unei variabile de tip *StringBufer*, cu sau fără transformarea caracterelor din majuscule în minuscule
- modificarea unui obiect de tip *StringBuffer* prin inserarea (la o anumită poziţie sau la sfârşit) oricăror tipuri de date Java
- modificarea unui obiect de tip *StringBuffer* prin alterarea oricărei părţi a conţinutului său
- determinarea lungimii şirului de caractere conţinut de un obiect de tip *StringBuffer*

Clasa StringBuilder

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html

=> se doreşte a fi un înlocuitor al clasei *StringBuffer*. Mecanismele, funcţionalităţile şi metodele pe care le oferă sunt foarte asemănătoare cu cele ale clasei mai sus amintite.

Clasa StringTokenizer

https://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html

=> permite separarea unui obiect de tip *String* în subsecţiuni (porțiuni), pe baza unor anumiţi separatori. Un obiect de acest tip nu face distincţia între tipurile datelor care sunt incluse în *String* şi le tratează pe toate la nivel de caracter sau şir de caractere.

Clasa *StringTokenizer* are constructori care permit:

- iniţializarea unui obiect de tip *StringTokenizer* cu un alt obiect de tip *String* primit ca parametru
- iniţializarea unui obiect de tip *StringTokenizer* cu un alte 2 obiecte de tip *String* (care reprezintă conţinutul variabilei şi şirul de caractere considerat drept separator) primite ca parametru

Clasa *StringTokenizer* include metode pentru:

- determinarea numărului de porţiuni obţinute după secţionarea potrivit separatorilor declaraţi
- parcurgerea iterativă a întregii secvențe de secțiuni
- extragerea porţiunii curente
- determinarea existenţei sau inexistenței de porţiuni adiţionale

- etc.

Lucru individual

1. Scrieți o aplicație Java care definește o cheie de autentificare de tipul: XXXXX-XXXXX-XXXXX-XXXXX, unde X reprezintă un caracter ce poate fi cifră sau literă. Scrieți un program care verifică dacă această cheie are exact 4 grupuri de caractere a câte 5 caractere fiecare și separate prin caracterul '-'. De asemenea, calculați numărul de cifre și litere din cheia de autentificare. Numărul de cifre trebuie să fie mai mare decât numărul de litere, iar numărul de litere nu poate să fie 0.
În cazul în care nu este îndeplinită cel puțin o condiție din cele menționate anterior, afișați mesajul: "Cheie de autentificare incorectă!"

==========

2. Fie un algoritm de criptare în cadrul căruia se ia un text de intrare 'A' format din litere mici și mari. Separat se definește un șir de caractere "B" cărora li se asociază aleator câte un număr întreg de la 1 la 100. Algoritmul verifică dacă literele din șirul B există în șirul A și face suma numerelor asociate acestora. La suma finală se adaugă pozițiile din șirul A la care au fost găsite caracterele din șirul B. Dacă suma finală este mai mare de 100, criptarea a fost validă. Afișați un mesaj corespunzător.

Exemplu:
Șirul A = "aTmPpDsst"
Șirul B ="ams"
Valorile asociate elementelor din șirul B: 11 33 7

Suma: (11+33+7+7)+(1+3+7+8)=77 -> CRIPTARE NEVALIDĂ

==========

3. Scrieti o aplicatie Java care defineste o valoare intreaga si apoi calculeaza si afiseaza reprezentarea ei binară, octală și hexazecimală. Scrieți funcții de calcul a valorilor în diferite baze.

==========

4. Implementați jocul X-0 naiv ca și joc automat. Programul va selecta aleator la fiecare pas o poziție pe care o va completa fie cu X, fie cu 0 în mod alternativ. Poziția selectată nu poate fi una completată deja. Jocul se termină atunci când nu mai există casuțe libere sau când unul dintre jucători a obținut o linie, coloană sau diagonală completă. Afișați pe ecran fiecare pas al algoritmului sub forma unei matrici, caracterul * va reprezenta o casuță necompletată. De ex:

```
X       *       *
*       *       *
*       *       *
--
X       *       *
*       0       *
*       *       *
--
X       *       *
*       0       *
X       *       *
--
X       *       0
*       0       *
X       *       *
```

X * 0
X 0 *
X * *

Joc terminat!

---

*Extindeți aplicația astfel încât dimensiunile tablei de joc să poată fi variabile.

==========

5. Se defineste un sir de variabile de tip String care va fi populat cu toate cartile dintr-un pachet de joc. Se vor extrage aleator carti in pachet pana cand cartea curenta va fi de trefla cu valoarea mai mare decat 8. Sa se afiseze la fiecare pas cartea curenta si numarul de carti deja extrase.

Indiciu: Folosiți un generator de numere aleatoare. Cartile deja extrase sunt eliminate.

==========

6. Implementati algoritmii cunoscuti de ordonare a sirurilor numerice (bubble sort, insertion sort, quick sort, etc.) si aplicati-i asupra unui sir de valori intregi predefinite.

Individual work

1. Write a Java application which defines an authentication key with the format: XXXXX-XXXXX-XXXXX-XXXXX, where X is a character which can be either a digit or a letter. The application should verify if this key has exactly 4 groups of characters with 5 characters each, and separated by the symbol '-'. Also, compute the number of digits and letters from the authetication key. The number of digits should be greater than the number of letters, and the number of letters cannot be 0.
If any of the above conditions are not met, display the message: "Invalid authetication key!"

==========

2. Assume that there is a cryptographic algorithm which takes an input text 'A' composed of lower and upper case characters. Separately a character string 'B' is defined. Each character from B has an associated random interger value between 1 and 100. The algorithm checks if the letters from B are found in A and adds the associated numerical values. To the final sum value, the algorithm also adds the positions from string A where characters from string B were found. If the final sum is larger than 100, the encryption was valid. Display a message with the result.
Example:
String A = "aTmPpDsst"
String B ="ams"
Associated numerical values for string B: 11 33 7
Sum: (11+33+7+7)+(1+3+7+8)=77 -> INVALID ENCRYPTION

==========

3. Write a Java application which defines an integer value and the computes and displays it binary, octal and hexadecimal representation. Write methods for each of the base convertions.

==========

4. Implement the naive dots and crosses game (X-O) as an automated game. The application will randomly select at each step a position from the board at which to place the next symbol, alternating between X and O. The selected position cannot be an already filled square. The game ends when either there are no more empty squares on the board, or one of the symbols wins by completing a line, a column or a diagonal. Display on the screen each step of the algorithm as a matrix. Unfilled squares will be represented by the * character.
Example:

```
 X  *  *
 *  *  *
 *  *  *
--
 X  *  *
 *  0  *
 *  *  *
--
 X  *  *
 *  0  *
 X  *  *
--
 X  *  0
 *  0  *
 X  *  *

 X  *  0
```

```
 X  0  *
 X  *  *
```

Game over!

---

*Extend the application so that the dimensions of the board (number of rows and columns) can be changed.

==========

5. Define an array of String variables that will be populated with all the playing cards from a complete package. A series of randomly picked cards will be extracted until the current card will have the clubs sign and a value greater than 8. At each step, the current card and the number of already extracted cards will be displayed.

Hint: Use a random numbers generator. The cards which were already used are eliminated.

==========

6. Implement the already known sorting algorithms (bubble sort, insertion sort, quick sort, etc.) and apply them upon an array of predefined integer variables.