Lecture #13-14
Strings & Text I/O

1. **String Class**
   - **A String is a sequence of characters**
   - **In Java, a string is an object**

2. **Constructing a String**

   String welcomeJavaString = new String("Welcome to Java Programing!");
   String welcomeJavaString = "Welcome to Java Programing!";

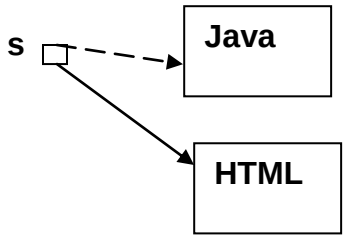
   char[ ] charArray = {'W', 'e', 'l', 'c', 'o', 'm', 'e'};
   String welcome = new String(charArray);

   Remark: The String Variable holds a reference to a String Object which, in turn, holds the String Value, i.e., the message.

3. **String objects are immutable, i.e., the contents of the object cannot be changed**

   String s = "Java";

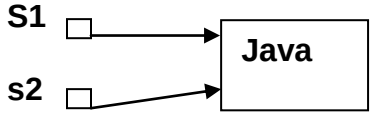
   String s = "HTML";



4. **Interned Strings are created by having more than one string literal with the same character sequence**

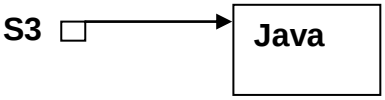
   String s1 = "Java";

   String s2 = "Java";



Interned string object, i.e., created for string "Java"

   String s3 = new String("Java");



separate string object created by new String("Java") statement

s1 == s2 ➔ TRUE **s1 == s3 ➔ FALSE**

5. **String Comparisons**

   a. **The statement "s1 == s2" compares the values of the reference variables s1 & s2, i.e., do they reference the same object?**

      "s1 == s2" performs an equality comparison of the contents of the reference variables.

   b. **The statement "s1.equals(s2)" compares the contents of the objects referenced by s1 & s2, i.e., do they contain the same contents?**

      "s1 == s2" performs an equality comparison of the contents of the objects.

   c. **java.lang.String UML Definitions Liang page 266**

      i. **s2.equals(s1); returns boolean**

      ii. **s2.equalsIgnoreCase(s1); returns boolean case insensitive**

      iii. **s2.compareTo(s1) returns int s2 > s2 ➔ returns n > 0**
      s2 == s1 ➔ returns 0
      s2 < s1 ➔ returns n < 0
      s2 > s1 refers to lexicographical ordering as determined by the Unicode Ordering, e.g.,
      s1 = "abc";
      s2 = "abg";
      s1.compareTo(s2) statement returns -4
      because 'c' is less than 'g' by 4 in the Unicode Table

      iv. **s2.compareToIgnoreCase(s1); returns int case insensitive**

      v. **+regionMatches(toffset: int, s1: String, offset: int, len: int);boolean**

      String s1 = "Auger";
      String s2 = "Burgermeister";
      regionMatches(3, s1, 4, 2); returns TRUE
      regionMatches(3, s1, 11, 2); returns TRUE

      vi. **+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int);boolean case insensitive**

      vii. **s2.startsWith("Bu"); returns TRUE**
      s1.startsWith("Bu"); returns FALSE

      viii. **s2.endsWith("et"); returns FALSE**

      ix. **s1.endsWith("er"); returns TRUE**

6. **String Length, Characters & Combining Strings**

a = "Java Programming";

int n = a.length;

n == 16 →TRUE

char ch = a.charAt(6);

ch == 'r';

b = " is Fun!";

c = a.concat(b);

c contains the string "Java Programming is Fun!"

Remarks:

- **Length is a method in the String class – hence use -- length( )**
- **Length is a property of an array object – hence use – length;**

- **String objects are represented internally by using a private array variable; the array can only be accessed via the public methods provided by the String class**

- **Java allows the use of string literals directly, e.g.,**
  "Java Programming".charAt(5); returns the character 'P'

- **The index n in the method s.charAt(n); must be bound by**
  0 <= n <= s.length( ) – 1
  Otherwise a StringIndexOutOfBoundsException will occur
- **Concatenation Options**
  - **String s3 = s1.concat(s2);**
  - **String s3 = s1 + s2;**
  - **Srting s1 = "File A"; String s3 = s1 + 2;**
    ➔ s3 contains the string "File A2"

7. **Substrings**

String s1 = "Old picadors are quick on their feet!."

String s2 = s1.substring(13);

s2 contains the substring "are quick on their feet!."

String s3 = s1.substring(13, 24);

S3 contains the substring "are quick o"

8. Converting, Replacing & Splitting Strings

+ toLowerCase( ) : String
+ toUpperCase( ) : String

+ trim( ) : String
" Java ".trim( ); ➔ "Java"

+ replace(oldChar: char, newChar: char) : String"
"Java".replace(a,e); ➔ "Jeve"

+ replaceFirst(oldString: String, newString: String) : String
"November is a novel month".replaceFirst("ove", "esse");
➔" Nessember is a novel month".

+ replaceAll(oldString: String, newString: String) : String
"November is a novel month".replaceAll("ove", "esse");
➔" Nessember is a nessel month".

+ split(delimiter: String): String [ ]
String [ ] tokens = "November is a novel month".split("o");
➔tokens contains the following substrings
"N" "vermber is a n" "vel m" "nth"

9. Matching, Replacing & Splitting Patterns (regular expressions)

a. * denotes any string, i.e.,
in a search for "Java*" will return any string which starts with "Java"

e.g., the matches( ) method returns TRUE for the following statements:

i. "Java is another name for coffee".matches("Java*");
ii. "Java is a programming language".matches("Java*");

b. ? denotes a single character, i.e.,
in a search for "Java?" will return any string which consists of "Java" with a single character appended as a suffex

e.g., the matches( ) method returns TRUE for the following statements:

i. "Java1".matches("Java*");
ii. "Java2".matches("Java*");
iii. "Java".matches("Java*");

The following methods may be used with regular expressions:

- **matches( );**
- **replaceAll( );**
- **replaceFirst( );**
- **split( );**

c. The bracket notation, e.g., [#, %, :] denotes the use of any of the bracketed characters, e.g.,

i. String s = "a&b#c:d%f&g#2".replaceAll("[# % :]", "/"); produces the string s which consists of the string "a&b/c/d/f&g/2"

ii. String [ ] tokens = "Java, C?C#,C++:Lisp&Cobol".split("[, ? : &]"); produces the array
tokens[6] == {Java, C, C#, C++, Lisp, Cobol }

10. Finding Characters or Substrings in a String

i. indexOf(ch: char): int
returns index of first occurrence

ii. indexOf(ch: char, fromIndex; int): int
returns index of first occurrence after fromIndex

iii. indexOf(s: String): int
returns index of first occurrence

iv. indexOf(s: String, fromIndex; int): int
returns index of first occurrence after fromIndex

v. lastIndexOf(ch: int): int
returns index of last occurrence

vi. lastIndexOf(ch: int, fromIndex; int): int
returns index of last occurrence after fromIndex

vii. lastIndexOf(s: String): int
returns index of last occurrence

viii. lastIndexOf(s: String, fromIndex; int): int
returns index of last occurrence after fromIndex

11. Conversion of a String to an Array of Characters

```
Char [ ] chars = "Java".toCharArray( );

getChars(int srcBegin, int srcEnd, char [ ] dst, int dstBegin);

copy a substring of a string
from index srcBegin to index srcEnd-1
into a character array dst starting from index dstBegin

e.g.,
char [ ] dst = {'J', 'A', 'V', 'A', '1', '3', '0', '1'};
"CS3720".getChars(2, 6, dst, 4);
Yields dst == {'J', 'A', 'V', 'A', '3', '7', '2', '0'};
```

12. Conversion of an Array of Characters to a String

```
String str = new String(new char [ ] {'J', 'A', 'V', 'A'});
String str = String.valueOf(new char [ ] {'J', 'A', 'V', 'A'});
```

13. Conversion of Characters & Numeric Values to Strings

ix. valueOf(c: char): String
    returns String consisting of character 'c'

x. valueOf(data: char [ ]): String
    returns String consisting of characters in the array

xi. valueOf(d: double): String
    returns String consisting of digits in double d

xii. valueOf(f: float): String
    returns String consisting of digits in float f

xiii. valueOf(i: int): String
    returns String consisting of digits in int i

xiv. valueOf(l: long): String
    returns String consisting of digits in long l

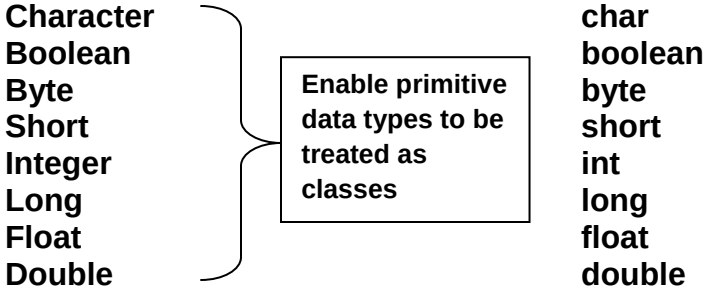
Remark:

```
Double.parseDouble(str);     converts string str to double
Integer.parseInteger(str);   converts string str to integer
```

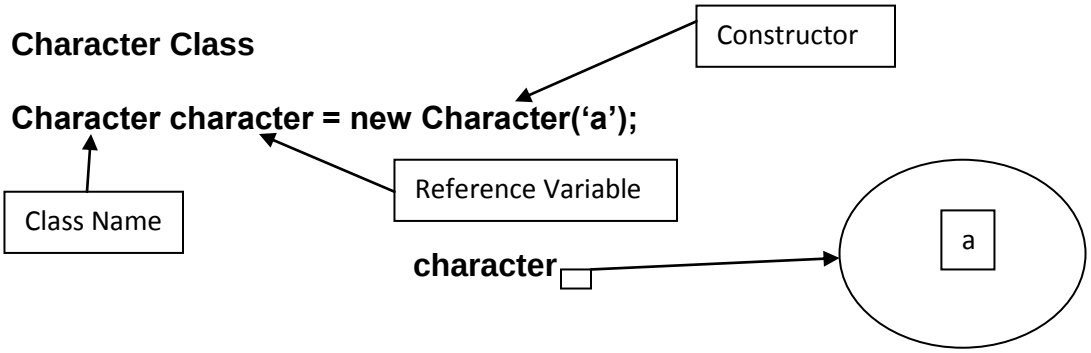
Remark: Liang page 272-273 Listing 8.1 Palindrome

```
String s = input.nextLine( );


public static boolean isPalindrome(String s)
{
  int low = 0;
  int high = s.length( ) – 1;

  while( low < high )
  {
    if(s.charAt(low) != s.charAt(high))
          return false;

    low++;
    high- -;
  }
  return true;
}
```

14. Wrapper Classes



15. Character Class



Character(value: char) — **Constructor**
charValue( ): char — **returns value stored in Character Class**

e.g.,
Character charObject = new Character('b');
char ch = charObject.charValue( ); ➔ ch contains 'b'

compareTo(anotherCharacter: Character): int
int i = charObject.compareTo(new Character('a')); ➔ i == 1
int i = charObject.compareTo(new Character('b')); ➔ i == 0
int i = charObject.compareTo(new Character('c')); ➔ i == -1
int i = charObject.compareTo(new Character('d')); ➔ i == -2

equals(anotherCharacter: Character): boolean
charObject.equals(new Character('a')); ➔ false
charObject.equals(new Character('b')); ➔ true

isDigit(ch: char): boolean
isLetter(ch: char): boolean
isLetterOrDigit(ch: char): boolean

isLowerCase(ch: char): boolean
isUpperCase(ch: char): boolean

toLowerCase(ch: char): boolean
toUpperCase(ch: char): boolean

Remark: Liang page 274-275 Listing 8.2 Counting Letters

```
String s = input.nextLine( );

int [ ] counts = countLetters(s.toLowerCase( ));

public static int [ ] countletters(String s)
{
   int [ ] counts = new int[26];

   for(int i = 0; i < s.length( ); I++;)
   {
      if(Character.isLetter(s.charAt(i)))
        counts[s.charAt(i) – ‘a’]++;
   }
   return counts;
}
```

16. StringBuilder & StringBuffer Classes

The value of a String object is fixed once the object is created.

The values of objects created by either the StringBuilder or Stringbuffer Classes can be modified after the object is created.

The methods for modifying the values in StringBuffer objects are synchronized; such objects can be accessed by multiple tasks concurrently.

The StringBuilder Class is more efficient if the objects are to be accessed by a single task.

StringBuilder( )	**constructs an empty object with capacity 16**
StringBuilder(capacity: int)	**constructs an object with specified capacity**
StringBuilder(s: String)	**constructs an object with specified string**

append(data: char [ ]): StringBuilder
append(data: char [ ], offset: int, len: int): StringBuilder
“Welcome to Java Programming”.append({‘C’, ‘o’, ‘b’, ‘o’, ‘l’}, 11);
produces “Welcome to Cobol Programming”.

append(v: aPrimitiveType): StringBuilder
“Java”.append(1); produces “Java1”

append(s: String): StringBuilder

“Java”.append(“ is a computer language”);

produces “Java is a computer language”

delete(startIndex: int, endIndex: int): StringBuilder

“Java is a computer language”.delete(9, 17);

produces “Java is a language”

deleteCharAt(index: int): StringBuilder

“Java”.deleteCharAt(3); produces “Jav”

insert(index: int, data: char[ ], offset: int, len: int): StringBuilder

insert(offset: int, data: char[ ]): StringBuilder

insert(offset: int, b : aPrimitiveType): StringBuilder

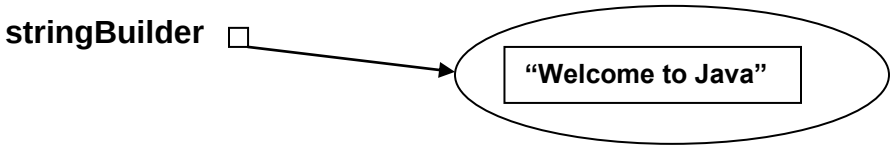
insert(offset: int, s: String): StringBuilder

replace(startIndex: int, endIndex: int, s: String): StringBuilder

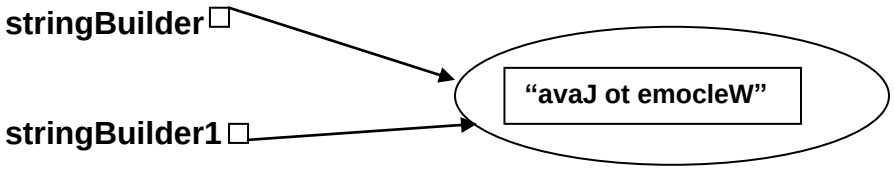
reverse( ): StringBuilder

setCharAt(index: int, ch : char): void

StringBuilder stringBuilder = new StringBuilder(“Welcome to Java”);



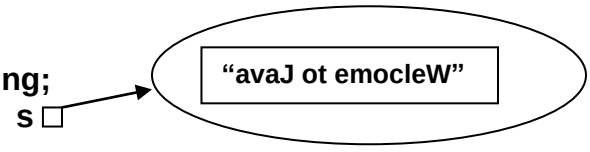
StringBuilder stringBuilder1 = stringBuilder.reverse( );



Remark: If a string does not require modification, use the String Class, since it allows run-time optimizations such as sharing interned strings.

toString( ): String

String s = stringBuilder1.toString;



capacity( ): int

int n = stringBuilder1.capacity( ); ➔ n == 15

charAt(index: int): char

char ch = stringBuilder1.charAt(1); ➔ ch == 'v'

setLength(newLength: int): void

stringBuilder1.setLength(20);
int n = stringBuilder1.capacity( ); ➔ n == 20

length( ): int

int n = stringBuilder1.length( ); ➔ n == 15

substring(startIndex: int): String

String s = stringBuilder1.substring(7); ➔ s contains "emocleW"

substring(startIndex: int, endIdex: int): String

String s = stringBuilder1.substring(7, 10); ➔ s contains "emo"

trimToSize( ): void

stringBuilder.trimToSize( );
int n = stringBuilder1.length( ); ➔ n == 15

Remark: Liang pages 279-280 Listing 8.3 Palindromes w/ Non-alphanumeric Char

17. Command Line Arguments public static void main(String [ ] args)

```
public class A
{
  public static void main(String [ ] args)
  {
     String [ ] strings = {"new York", "boston", "Atlanta"};
      B.main(strings)
  }
}
```

Invokes method B & passes the string variable to the called method

```
public class B
{
  public static void main(String [ ] args)
  {
     for(int i = 0; i < args.length; i++)
    System.out.println(args[i]);
  }
}
```

a. Compiling from the Command Line

> javac Hello.java

creates the java run-time file for this program

b. Passing string variables from the Command line

> java Hello Charles Robert Putnam

args[ ] = {"Charles", "Robert", "Putnam"}

i.e., args[0] denotes "Charles"

args[1] denotes "Robert"

args[2] denotes "Putnam"

args[n] for n > 2 are undefined, i.e., out-of-bounds

args.length == 3

Remark: Liang pages 282-283 Listing 8.4 Calculator

```
public static void main(String[ ] args
{
   If(args.length != 3) System.exit(0);
   int result = 0;
   switch(args[1].charAt(0))
   {
        case '+':  result = Integer.parseInt(args[0]) +
                           Integer.parseInt(args[2]);
                   break;
        case '-':  result = Integer.parseInt(args[0]) -
                           Integer.parseInt(args[2]);
                   break;
        case '*':  result = Integer.parseInt(args[0]) *
                           Integer.parseInt(args[2]);
                   break;
        case '/':  result = Integer.parseInt(args[0]) /
                           Integer.parseInt(args[2]);
                   break;
   }
   System.out.println(args[0] + "  " + args[1] + "  " + args[2] + " = " + result);
}
```

Note: Both JDK & Unix use the symbol * as a wildcard in regular expressions; hence it cannot be directly used as a Command Line argument, but must be enclosed in quotation marks.

```
> java Calculator 63 "*" 43
> java Calculator 63 + 43
> java Calculator 63 - 43
> java Calculator 63 / 43
```

18. File Class (Storage of Input & Output Data between Processing Runs)

path designations

a. absolute path designations

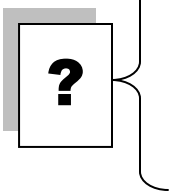
- **Windows** c:\home\fac\cputnam\Comp110\file-name
- **Unix** /home/fac/cputnam/Comp110/file-name

b. relative path designations

- **Windows** .\sub-dir\file-name ..\directory\file-name
- **Unix** ./sub-dir/file-name ../directory/file-name

File Class : Wrapper Class

- **file path & file name – string**
- **methods for**
  - **obtaining file properties**
  - **renaming files**
  - **deleting files**
- **hides the machine-dependent complexities of files & path names**



- **new File("c:\\home\\Project7") creates a file object for c:\home\Project7**

  Remark: "\" is a special character in Java, Windows, & Unix hence "\\" must be used in new File("c:\\home\\Project7")

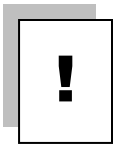
  Remark: Creating a file instance using new File("c:\\home\\Project7") does not create a file on the machine; it only creates a File Class containing the file path, file name and the methods listed above.

- **exists(HW) : boolean** **returns TRUE if HW exists**
- **isDirectory(HW) : boolean** **returns TRUE if HW is a directory**
- **isFile(HW) : boolean** **returns TRUE if HW is a file**

Remark: Do Not Use Absolute Paths ➔ Not Portable to UNIX or Other Platforms

For the file Welcome.java in the current directory,
create a file object by using new File("Welcome.java")

For the file us.gif in the image subdirectory of the current directory
create a file object by using new File("image/us.gif")



Java uses the forward slash "/" as a directory separator; a file object created by new File("image/us.gif") is portable to Unix, Mac O/S, Windows , etc

Remark: Liang pages 284-285 Listing 8.5 Use of File Object Creation & Methods

File(pathname: String)	**Creates File Object for specified directory or file**
File(parent: String, child: String)	**Creates File Object for specified child under the directory String parent; Child may be directory or file**
File(parent: File, child: String)	**Creates File Object for specified child under the directory File parent; Child may be directory or file**
exists( ): boolean	**Returns TRUE if file exists**
canRead( ): boolean	**Returns TRUE if file exists & can be read**
canWrite( ): boolean	**Returns TRUE if file exists & can be written**
isDirectory( ): boolean	**Returns TRUE if File Object represents a Directory**
isFile( ): boolean	**Returns TRUE if File Object represents a file**
isAbsolute( ): boolean	**Returns TRUE if File Object was created using an absolute pathname**
isHidden( ): boolean	**Returns TRUE if the file represented by the File Object is hidden**

Remark: The <u>hidden property</u> is system dependent

- **Windows ➔ File Properties Box ➔ mark as hidden**
- **Unix ➔ name begins with a period "."**

getAbsolutePath( ): String	**Returns absolute path & file name**
getCanonicalPath( ): String	**see Liang page 284 figure 8.15**
getName( ): String	**see Liang page 284 figure 8.15**
getPath( ): String	**see Liang page 284 figure 8.15**
getParent( ): String	**see Liang page 284 figure 8.15**
lastModified( ): long	**see Liang page 284 figure 8.15**
length( ): long	**file ➔ returns length; directory ➔ returns 0**
lastFile( ): File [ ]	**Returns the files listed in the directory**
delete( ): boolean	**Returns TRUE if the deletion is successful**
renameTo(dest: File): boolean	**Renames the file, returns TRUE if successful**

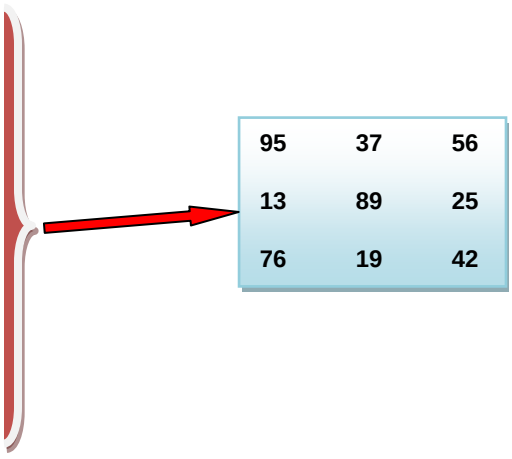
19. File I/O

a. Writing Data using PrintWriter Objects

PrintWriter output = new PrintWriter(filename);

- **creates a file**
- **enables the use of the methods provided in the PrintWriter Class**

```
public class WriteData
{
   public static void main(String [ ] args) throws Exception
   {
      java.io.File file = new java.io.File("scores.txt");
            // creates File Object for "scores.txt"

            // next line checks to see if actual file exists
      if (file.exists( ))
      {
         System.out.println("File already exists");
         System.exit(0);
      }

      java.io.PrintWriter output = new java.io.PrintWriter(file);
      /* file "scores.txt" is created; if file already exists it will discard
                                    the current values, i.e., empty the file */

      output.print(95); output("    ");
      output.print(37); output("    ");
      output.print(56); output("    ");
      output.println("");
      output.print(13); output("    ");
      output.print(89); output("    ");
      output.print(25); output("    ");
      output.println("");
      output.print(76); output("    ");
      output.print(19); output("    ");
      output.print(42); output("    ");
      output.println("");

      output.close( );     /* closes file buffers, i.e., writes contents of file
                              buffers to file & deletes the file buffers;
                              if not invoked, the file will not save all of the
                                                          provided data */
   }
}
```



b. Reading Data using Scanner Objects

Scanner breaks the input into tokens delimited by whitespace characters

- **Keyboard Input**

  Scanner input = new Scanner(System.in);

- **File Input**

  java.io.File file = new java.io.File("scores.txt");
  Scanner input = new Scanner(file);

Scanner(source: File)	**scans values from the specified file**
Scanner(source: String)	**scans values from the specified string**
close( )	**closes Scanner**
hasNext( ): boolean	**returns TRUE if there is more data**
next( ): String	**returns next token delimited by whitespace**
nextLine( ): String	**returns line delimited by a line separator**
nextByte( ): byte	**returns next token as a byte**
nextShort( ): short	**returns next token as a short**
nextInt( ): int	**returns next token as an int**
nextLong( ): long	**returns next token as a long**
nextFloat( ): float	**returns next token as a float**
nextDouble( ): double	**returns next token as a double**

useDelimiter(pattern: String): Scanner
sets the delimiting pattern for the specified Scanner Object

```
import java.util.Scanner;

public class ReadData
{
  public static void main(String [ ] args) throws Exception
  {
     java.io.File file = new java.io.file("scores.txt"); // creates File Object
    Scanner input = new Scanner(file);                   // creates Scanner Object
     long [ ][ ] A = new long [3][3];
    while(input.hasNext( ))
    {
        for (i=0; i < 3; i++)
           for (j = 0; j < 3; j++)
             A[ i ][ j ] = input.nextLong( );
        input.close( );              /* not necessary for data integrity but it is
                                        considered to be good practice since it
                                        releases resources, i.e., deletes input
                                        buffers & associated structures, thus not
                                        contributing to the degradation of overall
                                        system performance */
    }
  }
}
```

20. Scanner Operations

```
next( ): String
nextLine( ): String
nextByte( ): byte
nextShort( ): short
nextInt( ): int
nextLong( ): long
nextFloat( ): float
nextDouble( ): double
```

Token-Reading Methods

read tokens separated by tokens; normally whitespace

useDelimiter(String regular expression) to set a new delimiter pattern, i.e., to change the token delimiters

token reading procedure:

1. skip any leading whitespace
2. convert next data item to specified type
3. if the token does not match the expected type, throw a run-time Exception, i.e., **java.util.InputMismatchException**
4. next( ) reads a string delimited by delimiters
5. token reading methods do not read the delimiter after the token

Remark:

- line separators are platform dependent
  - Windows line separator: \r\n
  - Unix line separator: \n
  - Keyboard text: \n
- String lineSeparator = System.getProperty("line.separator"); will return the line separator use on the current platform
- token reading methods do not deliver the tokens to the system

Behavior of nextLine( ) method
reads a line ending with a line separator

if the nextLine( ) method is invoked after a token reading method, it reads characters that are between this delimiter and the line separator; the line separator is read but is not returned as part of the string

Warning: reading keyboard entries
See Liang page 288-289

21. Command Line Argument Program

> java ReplaceText sourcefile targetFile oldString newString

> **ReplaceText creates the targetFile from the sourceFile by replacing all occurrences of the string oldString by the string newString**

```
import java.io.*;
import java.util.*;

public class ReplaceText
{
  public static void main(String [ ] args) throws Exdceptin
  {
    if (args.length != 4) System.exit(0);

    File sourceFile = new  File(args[0]);
    if ( !sourceFile.exists() )
    {
       System.out.println(args[0] + " does not exist");
       System.exit();
    }

    File targetFile = new  File(args[1]);
    if (targetFile.exists())
    {
       System.out.println(args[1] + " already exists");
       System.exit();
    }

    Scanner input = new Scanner(sourceFile);
    Printwriter output = new Printwriter(targetFile);

    while (input.hasNext())
    {
       String s1 = input.nextLine( );
       String s2 = s1.replaceAll( args[2], args[3] );
       output.println(s2);
    }
    input.close( );
    output.close( );

  }
}
```

22. GUI File Dialogs

javax.swing.JfileChooser class
user can choose a file & display the contents

```
import java.util.Scanner;
import javax.swing.JfileChooser;

public class ReadFileUsingFileChooser
{
  public static void main(String [ ] args) throws Exception
  {
    JFileChooser fileChooser = new JFileChooser( );
    if (fileChooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION  )
    {
        java.io.File file = fileChooser.getSelectedFile( );
        Scanner input = new Scanner(file);
        while (input.hasnext( )) System.out.println(input.nextLine());
        input.close( );
    }
    else
        System.out.println("No file selected");
  }




}
```