Lecture Notes
Strings

1. String Class

- provides 40+ methods for manipulating strings
- provides eleven constructors for creating String objects

2. String Object

- represents a <u>sequence of characters</u>, i.e., a string of characters

a. Creating a String

- String newString = new String(stringLiteral);
- String newString = new String("Welcome");
- String msg = "Welcome";
- Char[ ] charArray = {"W", "e", "l", "c", "o", "m", "e"};
  String msg = new String(charArray);

b. String Components

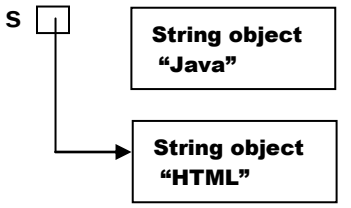
- String variable e.g., newString
- String object .g., String("Welcome")
- String value e.g., "Welcome"

3. Immutable Strings

a. String objects are immutable

String s = "Java";

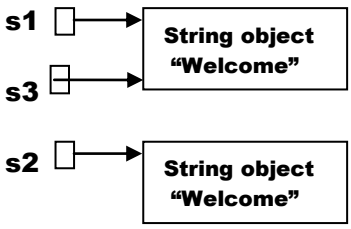
s = "HTML";



JVM uses the concept of "interned" to provide a single storage space for multiple instances of the same sequence of characters

b. Interned Strings

String s1 = "Welcome";
String s2 = new String("Welcome");
String s3 = "Welcome";



s1 == s2 → FALSE
s1 == s3 → TRUE

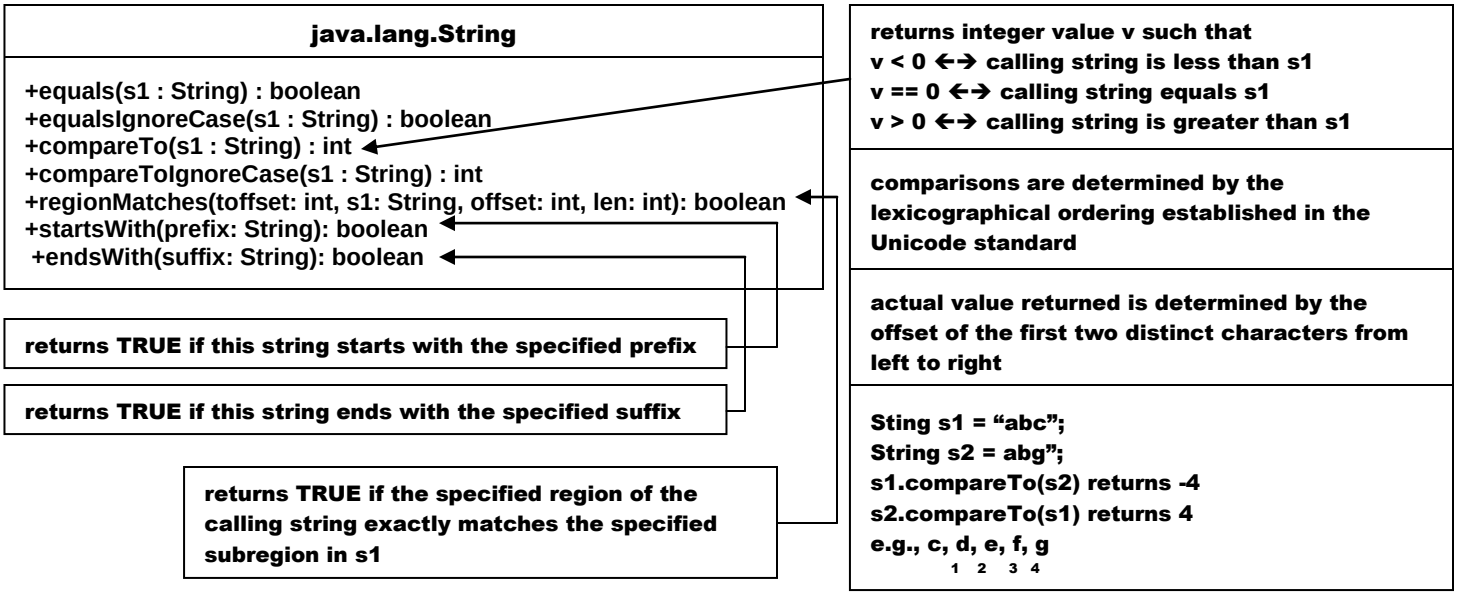
4. String Comparisons

- "==" operator

  if(string1 == string2)
  string1 & string2 refer to the same object
  else
  string1 & string2 refer to different objects
  but in this case,
  <u>the content of the objects string1 and string2 may or may not be the same</u>

- ".equals" operator

  If(string1.equals(string2))
  string1 & string2 have the same contents
  else
  the contents are different



public boolean regionMatches(int toffset, String other, int ooffset, int len)
or
public boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)

toffset:	the starting offset of the subregion in this string (calling string)
other:	the string argument
ooffset:	the starting offset of the subregion in the string argument (called string)
len:	number of characters to compare
ignoreCase:	if true, ignore case when comparing characters

return value: true if the subregions match false otherwise
match is exact if ignoreCase is false
match is case insensitive if ignoreCase is true

```
import java.io.*;
public class Test
{
  public static void main(String args[])
  {
    String Str1 = new String("Welcome to JavJavas_great.com");
String Str2 = new String("JavJavas");
String Str3 = new String("JAVJAVAS");

    System.out.print("Return Value :" );
    System.out.println(Str1.regionMatches(11, Str2, 0, 9));

    System.out.print("Return Value :" );
    System.out.println(Str1.regionMatches(11, Str3, 0, 9));

    System.out.print("Return Value :" );
    System.out.println(Str1.regionMatches(true, 11, Str3, 0, 9));
  }
}
```

called string

calling string

```
Return Value :true
Return Value :false
Return Value :true
```

---

5. String Length, Characters, & Combining Strings

java.lang.String
+length( ): int +charAt(index: int): char +concat(s1: String): String

returns the character located at the specified index in the String s for all index such that 0 <= index <= s.length - 1

String Literals
"Welcome".charAt(0) returns 'W'

Caution
length is a method in the String class thus
s.length( ) returns the number of characters in the string s
but
length is a property of an array object thus
a.length returns the number of elements in the aray a

String values are represented internally using a private array variable; hence the index notation is appropriate when accessing individual characters of the string object.

Data Encapsulation – detailed data structure of the String class is hidden from the user via private modifiers, hence users cannot directly manipulate the internal data structure

The String class provides numerous public methods to access and modify the string content.

Accessing a string out-of-bounds produces a StringIndexOutOfboundsException error, i.e., do not use string s index beyond s.length( ) - 1

String Concatenation

```
String s1 = "Charles";
String s2 = "Putnam";
String s3, s4, s5 = "15", s6;
s3 = s1.concat(s2); ➔ s3 contains the string "CharlesPutnam"
s4 = s1 + " " + s2; ➔ s4 contains the string "Charles Putnam"
s6 = s5 + 45 ➔ s6 contains the string "1545"
```

6. Substrings

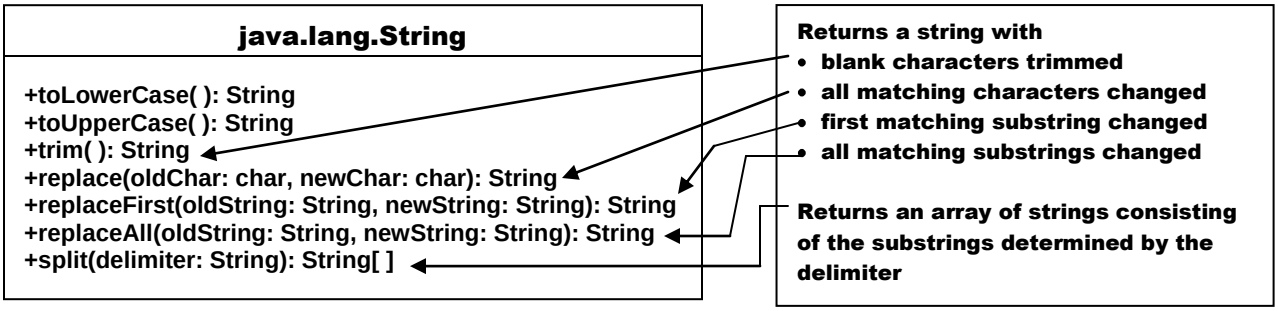
java.lang.String
+substring(beginIndex: int) : String +substring(beginIndex: int, endIndex: int): String

returns substring extending until
- **end of the string**
- **character at endIndex - 1**

String msg = "Welcome to Java".substring(0, 11) + "HTML";
➔
msg now contains the string "Welcome to HTML"

beginIndex == endIndex ➔ returns the empty string with length zero.
beginIndex > endIndex ➔ runtime error

7. Converting, Replacing & Splitting Strings



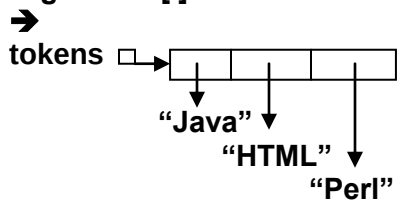

```
public class StringReplaceAllExample
{
   public static void main(String[] args)
  {
      String string = "Find String and replace All String";
      String newString = string.replaceAll("String", "New");
      System.out.println("Output :"+newString);
   }
}
```

Output String : "Find New and replace All New"

```
public class StringReplaceFirstExample
{
   public static void main(String[] args)
  {
      String string = "Find String and replace All String";
      String newString = string.replaceFirst("String", "New");
      System.out.println("Output :"+newString);
   }
}
```

Output String : "Find New and replace All String"

String tokens[ ] = "Java#HTML#Perl".split("#");



```
public class StringSplit
{
  public static void main(String args[]) throws Exception
  {
     new StringSplit().doit();
  }

  public void doit()
  {
     String s3 = "Real-How-To";
     String [ ] temp = null;
     temp = s3.split("-");
     dump(temp);
  }

  public void dump(String [ ]s)
  {
    System.out.println("------------");
    for (int i = 0 ; i < s.length ; i++)
    {
     System.out.println(s[i]);
    }
  System.out.println("------------");
  }
}
```

```
output
------------
Real
How
To
------------
```

8. Matching, Replacing & Splitting by Patterns

REGULAR EXPRESSIONS

a. match any number of characters *

+marches(s: String): boolean
returns TRUE if the invoking (calling) string matches the regular expression in s

s.matches("Java*")
returns true if the string object contains any string
starting with the characters "Java"

s	value returned
Javac is a compiler	TRUE
Java Indonesia	TRUE
I like Java coffee	FALSE

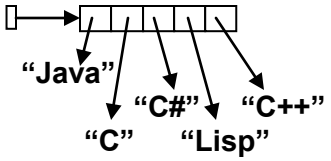
b. match any characters listed [, ,]

- matches( )
- replaceAll( )
- replaceFirst( )
- split( )

String s = “a:b;c,d;e”.replaceAll(“[: ; ,]”, “ “); ➔ “a b c d e”

String [ ] tokens = “Java,C?C#,Lisp.C++”.split(“[. , : ; ?]”);

tokens



regular expression [. , : ; ?]
specifies a set of delimiters for splitting the string

9. Finding a Character or a Substring in a String

java.lang.String	
+indexOf(ch: char): int	first occurrence
+indexOf(ch: char, fromIndex: int): int	first occurrence after fromIndex
+indexOf(s: String): int	first occurrence
+indexOf(s: String, fromIndex: int): int	first occurrence after fromIndex
+lastIndexOf(ch: char): int	last ocurrance
+lastIndexOf(ch: char, fromIndex: int): int	last occurrence before fromIndex
+lastIndexOf(s: String): int	last ocurrance
+lastIndexOf(s: String, fromIndex: int): int	last occurrence before fromIndex

“Welcome to Java”.indexOf(‘o’); ➔ 4

“Welcome to Java”.indexOf(‘o’, 5); ➔ 9

“Welcome to Java”.indexOf(“come”); ➔ 3

“Welcome to Javava”.indexOf(“ava”); ➔ 12

“Welcome to Java”.indexOf(“java”, 5); ➔ -1

“Welcome to Javava”.lastIndex(‘a’); ➔ 16

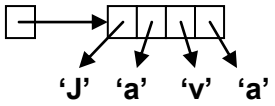
“Welcome to Javava”.lastIndex(“ava”); ➔ 14

“Welcome to Java”.lastIndex(“home”); ➔ -1

10. Conversions STOP HERE (The end for today)

a. Strings ➔ Arrays

i. String s = “Java”;
Char [ ] ch = s.toCharArray( );➔
ch



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

calling string beginning index

calling string ending index

target array

target array beginning index

String [ ] a = {‘W’, ‘e’, ‘l’, ‘c’, ‘o’, ‘m’, ‘e’, ‘ ‘, ‘t’, ‘o’, ‘ ‘, ‘J’, ‘a’, ‘v’, ‘a’};
“Comp 110: ”.getChars(0, 9, a, 0);
➔
“Comp 110: Java”

b. Arrays ➔ Strings

i. String(char [ ]); //Constructor

String [ ] a = {‘W’, ‘e’, ‘l’, ‘c’, ‘o’, ‘m’, ‘e’, ‘ ‘, ‘t’, ‘o’, ‘ ‘, ‘J’, ‘a’, ‘v’, ‘a’};
String str = new String(a);
➔
str → “Welcome to Java”

ii. valueOf(char[ ]); //method
String str = String.valueOf(a);
➔
str → “Welcome to Java”

c. Characters & Numeric Values ➔ Strings

java.lang.String

<u>+valueOf(c: char): String</u>
<u>+valueOf(data: char[ ]): String</u>
<u>+valueOf(d: double): String</u>
<u>+valueOf(f: float): String</u>
<u>+valueOf(i: int): String</u>
<u>+valueOf(l: long): String</u>

returns a string

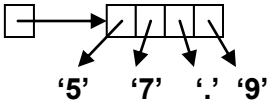
- consisting of the character c
- consisting of the characters in the array
- representing the double value
- representing the float value
- representing the int value
- representing the long value

Underline denotes class or static methods

d. Strings ➔ Numeric Values

i. Double.parseDouble(str);

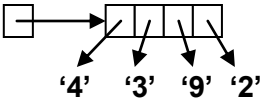
double d = 57.9;
String sd = valueOf(d); ➔ sd



double d1 = Double.parseDouble(sd); d1 == 57.9

ii. Integer.parseInt(str);

int i = 4392;
String si = valueOf(i); ➔ si



int i1 = Integer.parseInt(si); i1 == 4392

Wrapper Classes

Enable primitive data types to be treated as classes
Contain useful methods for processing primitive values

Character	Boolean	Byte	Short
Integer	Long	Float	Double

ABLE was I ere I saw ELBA

Palindromes

```
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;
}
```

11. Character Class

java.lang.Character

+Character(value: char)
+charValue( ): char
+compareTo(anotherCharacter: Character): int
+equals(anotherCharacter: Character): boolean
+isDigit(ch: char): boolean
+isLetter(ch: char): boolean
+isletterOrDigit(ch: char): boolean
+isLowerCase(ch: char): boolean
+isUpperCase(ch: char): boolean
+toLowerCase(ch: char): char
+toUpperCase(ch: char): char

Listing 8.2

```
// main
  …
String s = input.nextLine( );
int [ ] counts = countLetters(s.toLowerCase( ));

for (int i = 0; i < counts.length; i++)
{
  if (counts[ i ] != 0)
     System.out.println((char)('a' + i) + " appears " +
           counts[ i ] + ((counts[ i ] == 1) ? " time" : " times"));
}


// end of main

// method declaration
public stat 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;
}
```

12. StringBuilder/StringBuffer Class

- mutable, i.e.,
  - objects can be modified by adding, inserting, & appending new contents
- methods for modifying StringBuffer objects are <u>synchronized</u>; use the StringBuffer class if the objects may be accessed by multiple tasks concurrently
- methods for modifying StringBuilder objects are <u>more efficient</u> if the objects are to be accessed by single tasks

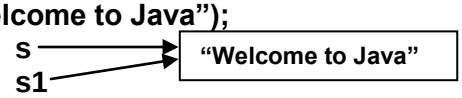
java.lang.StringBuilder

+StringBuilder( ) // empty string builder capacity 16
+StringBuilder(capacity: int) // empty string builder specified capacity
+StringBuilder(s: String)
 	
+append(data: char[ ]): StringBuilder // append char array
+append(data: char[ ], offset: int, len: int): StringBuilder // append subarray
+append(v: aPrimitiveType): StringBuilder // appends primitive type value
+append(s: String): StringBuilder // appends string
 	
+delete(startIndex: int, endIndex: int): StringBuilder
// deletes char from startIndex to endIndex
+deleteCharAt(index: int): StringBuilder // deletes char at specified index
 	
+insert(index: int, data: char[ ], offset: int, len: int): StringBuilder
//inserts a subarray of the data array into builder at the specified index
+insert(offset: int, data: char[ ]): StringBuilder // inserts data into builder at position offset
+insert(offset: int, b: aPrimitiveType): StringBuilder
// inserts converted value into this builder
+insert(offset: int, s: String): StringBuilder
// inserts a string into this builder at the position offset
 	
+replace(startIndex: int, endIndex: int, s: String): StringBuilder
// replaces characters in builder from startindex to endindex with specified string
 	
+reverse( ): StringBuilder // reverses characters in builder
 	
+setCharAt(index: int, cxh: char): void
// sets new character at the specified index in this builder

```
StringBuilder s = new StringBuilder("Welcome to Java");

StringBuilder s1 = s.reverse( );

s.reverse( );          // OK
```



If the string does not require modification use String rather than StringBuilder

13. Additional StringBuilder Methods

java.lang.StringBuilder	
+toString( ): String	returns string object
+capacity( ): int	returns capacity
+charAt(index: int): char	returns character at specified index
+length( ): int	returns number of characters in this builder
+setLength(newLength: int): void	sets a new length in this builder
+substring(startIndex: int): String	returns substring starting at startIndex
+substring(startIndex: int, endIndex: int): String	returns substring from startIndex to endIndex-1
+trimToSize( ): void	reduces storage size used for builder

s.length( ) <= s.capacity( )

capacity

- **automatically increased when additional characters exceed the current capacity**
- **storage space is an internal array**
- **when size is automatically increased an new array is allocated and the values are copied from the old to the new array**
- **new array size = 2*(previous array size + 1)**

- **for efficient program construction**
  - **specify initial capacity larger than builder will require, so that the JVM will never need to reallocate storage space**
  - **do not specify storage space too large as that wastes memory space**
  - **use trimToSize( ) method to reduce the capacity to the actual size**

Listing 8.3

```
// main
    …

  String s = input.nextLine( );
  System.out.println( s + " is a palindrome? " + isPalindrone(s));

// end of main
// methods
```

```
  public static boolean isPalindrome(String s)
  {
     String s1 = filter(s);
     String s2 = reverse(s1);
     return s2.equals(s1);
  }
```

```
 public static String filter(String s)
 {
    StringBuilder sb = new StringBuilder( );
    for( int i = 0; i < s.length( ); i++)
    {
        if (Character.isletterOrDigit(s.charAt(i)))
        {
           sb.append(s.charAt(i));
        }
    }
    return sb.toString( );
 }
```

Remark:
The String method filter( ) uses a StringBuilder object to create a new object containing only letters and digits; before the object is returned is must be converted to a String object

---

```
 public static String reverse(String s)
 {
    StringBuilder sb1 = new StringBuilder(s);
    sb1.reverse( );
    return sb1.toString( );
 }
```

Remark:
The String method reverse( ) uses a StringBuilder object to create the reverse of characters in the internal array; before the object is returned is must be converted to a String object

14. Command-Line Arguments

a. Calling the main Method of another Program

```
public class A
{
  public static void main(String[ ] args)
  {
    String s = {"New York", "Los Angeles","Chicago"}
    B.main(s);
  }
}
```

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

B.main(s); invokes main statement in program B which prints the string s

b. Passing Strings to the main Method

>javac Hello.java
>java Hello

- create data file A consisting of an array of numbers, i.e., a matrix
- create data file B consisting of an array of numbers, i.e., a matrix
- use the program Matrix to compute array C

When a program, with arguments, is invoked on the command line

>java Matrix C = A "*" B

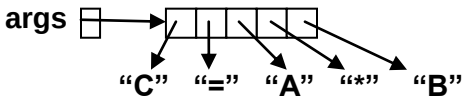
```
public class Matrix
{
  public static void main(String[ ] args)
                . . .
}
```

"*" is required to pass the argument * via the command line;

when * is used on the command line, it refers to all files in the current directory

* is also used as a wildcard in a regular expression

the JVM provides the String array args = new String[5];



If the program has no arguments the array args is created, i.e.,
    args = new String[0];
thus <u>args is empty</u>, i.e., args.length == 0

c. Passing all Files in the Current Directory to the main Method

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

The command line statement

>java Test *

passes the names of all the files in the current directory to the program which then prints the names in the display area

d. Listing 8.4

```
>java Calculator 63 "*" 47
>java Calculator 63 + 47
>java Calculator 63 - 47
>java Calculator 63 / 47

public class Calculator
{
  public static void main(String([ ] args)
  {
     if (args.length != 3)
     {
       System.out.println( . . . );
       System.exit(0);
     }
    int result = 0;
    switch(args[i].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;
       default : System.out.println(" . . . ");
    }
    System.out.println(args[0] + ' ' args[1] + ' ' + args[2] + " = " + result);
  }
}
```

Insures that Calculator is provided with the correct number of arguments

Converts the string args[...] from a string to an integer

15. The File Class

- binary files (Liang Ch 19)
- text (ASCII or Unicode) files

- file system
  - directories
    - files

- absolute file name – complete path + file name
  - complete path includes root directory (Unix) or drive letter (Windows)
  - absolute file names are machine dependent
    - c:\Documents and Settings\ putnam_adm\Desktop
    - /usr/home/faculty/compsci/cputnam

- relative file name – path from the current directory + file name
- <u>absolute/relative path + file name</u> is a string

a. File Class
   - contains the methods for
     - obtaining file properties
     - renaming files
     - deleting files

<u>Directory Separators</u>
Windows \ Backslash
Java / Forward Slash
Unix / Forward Slash

   - wrapper class for the file name and its directory path
     - provides machine dependent complexities of files and path names in a machine-independent fashion

   - does not contain the methods for reading or writing file contents
   - does not contain the methods for creating the actual physical file

Windows
   - new File("c:\\book") creates a File object for the directory c:\book
   - new File("c:\\book\\test.dat") creates a File object for the file c:\book\test.dat

In Java the backslash \ has special meaning and must be written as \\ in a string literal

Unix
   - new File("/usr/home/faculty/compsci/cputnam") creates a File object for the directory /usr/home/faculty/compsci/cputnam
   - new File("/usr/home/faculty/compsci/cputnam/junk") creates a File object for the file /usr/home/faculty/compsci/cputnam/junk

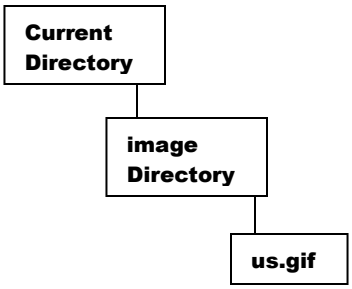
   - constructing a File instance does not create a physical file on the machine
   - a File instance can be created for any filename regardless of whether or not a physical file exists on the machine

   - invoking the exists( ) method on a File object will reveal whether or not the physical file exists

b. Code Portability

- **Do not use absolute file names in application code; if the program uses absolute file names, the program probably will not work on another system, i.e., it will not be portable.**

- **Use relative file names; ; if the program uses file names based on the current directory, the program will be portable as far as the directory paths are concerned.**
  - **new File("Welcome.java") for the file "Welcome.java" in the current directory**
  - **new File("image/us.gif") for the file "us.gif" in the directory image which is a subdirectory of the current directory**



java.io.File

+File(pathname: String)	creates a File object for the pathname: directory or file
+File(parent: String, child: String)	creates a File object for the child: directory or file parent is a String
+File(parent: File, child: String)	creates a File object for the child: directory or file parent is a File
+exists( ): boolean	
+canRead( ): boolean	
+canWrite( ): boolean	
+isDirectory( ): boolean	
+isFile( ): boolean	
+isAbsolute( ): boolean	returns true if the File object is created using an absolute path
+isHidden( ): boolean	returns true if the file represented in the File object is hidden • Windows – mark file hidden in File Properties dialog box • Unix -- filename begins with a period "." Character
+getAbsolutePath( ): String	
+getCanonicalPath( ): String	returns absolute path with redundant names, e.g., "." & ".." removed , symbolic links resolved, and drive letters converted to uppercase
+getName( ): String	
+getPath( ): String	
+getParent( ): String	
+lastModified( ): long	returns time the file was last modified
+length( ): long	returns • size of file if it exists • 0 if the file does not exist • 0 if it is a directory
+listFile( ): File[ ]	returns the files under a directory for a directory File object
+delete( ): boolean	returns true if deletion succeeds
+renameTo(dest: File): boolean	returns true if renaming operation succeeds

c. File Input & Output

i. <u>Writing Data using PrintWriter</u>

Listing 8.6

```
public class WriteData
{
  public static void main(String[ ] args) throws Exception
  {
    java.io.File file = new java.io.File("scores.txt");
    if (file.exists( ))
    {
        System.out.println( . . . );
        System.exit(0);
    }

    java.io.PrintWriter output = new java.io.PrintWriter(file);
    output.print("JTSmith  ");
    output.println(90);
    output.printf("%d", 97.3);
    output.close( );
  }
}
```

Invoking the constructor new PrintWriter(String filename) may throw an Exception; thus all such programs must include the declaration on the main method

Creates File wrapper for "scores.txt"

Creates a PrintWriter object for the text file "scores.txt"; if the physical file "scores.txt" does not exist, the PrintWriter constructor will create it; if the file does exist, the current content will be discarded.

System.out is a standard Java object for the console; new java.io.PrintWriter(file) creates a Java object for the file "scores.txt"

If the file is not properly closed by invoking the file.close( ) method, data may not be properly saved

java.io.PrintWriter

+PrintWriter(filename: String)

+print(s: String): void
+print(c: char): void
+print(cArray: char[ ]): void
+print(i: int): void
+print(l: long): void
+print(f: float): void
+print(d: double): void
+print(b: boolean): void

Also includes the overloaded println methods – prints line separator
Windows – line separator \r\n
Unix – line separator \n

Also includes the overloaded printf methods – see §3.6 Liang

ii. Reading Data using Scanner

- **Scanner breaks input into tokens delimited by whitespace**
- **To read from the keyboard**
  - **Scanner input = new Scanner(System.in)**

System.in is a standard Java object for retrieving data from the keyboard;

- **To read from a file**
  - **Scanner input = new Scanner(new File(filename))**

new File(filename) creates a Java object for reading data from a file

java.util.Scanner

+Scanner(source: File)	**produces values from the specified file**
+Scanner(source: String)	**produces values from the specified string**
+close( )	
+hasNext( ): boolean	**returns true if scanner has more data to be read**
+next( ): String	**returns next token as a string delimited by whitespace**
+nextLine( ): String	**returns a line ending with a line separator**
+nextByte( ): byte **+nextShort( ): short** **+nextInt( ): int** **+nextLong( ): long** **+nextFloat( ): float** **+nextDouble( ): double**	**returns next token as a** • **byte** • **short** • **int** • **long** • **float** • **double**
+useDelimiter(pattern: String): Scanner	**sets the scanner's delimiting pattern**

Listing 8.7

Invoking the constructor new Scanner(File) may throw an exception

```
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");
        if (!file.exists( )) {  System.out.println(" ... "); System.exit(0); }
        Scanner input = new Scanner(file);
        while (input.hasNext( ))
        {    String firstName = input.next( );
             String mi = input.next( );
             String lastName = input.next( );
             Int score = input.nextInt( );
             System.out.println(firstname + " " + mi + " " + lastName + " " + score);
        }
        input.close( );
    }
}
```

It is not necessary to close an input file for data integrity but closing an input file releases resources for other uses

iii. How the Scanner Works

Token Reading Methods

Token Reading Methods read tokens that are separated by delimiters; by default the delimiters are whitespace; a new set of delimiters may be set by the use of the useDelimiter(String regex) method

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

Token Reading Methods skip any leading delimiters, reads a token ended by a terminal delimiter; the token is converted into a value of the type required by the specific token, e.g., byte, short, int, long, float or double. The next( ) method does not convert the token string.

If the converted token does not match the expected type, a runtime exception java.util.InputMismatchException will be thrown

- next( ) method reads a string delimited by delimiters
- nextLine( ) method reads a line ending in a line separator
  - line separators are platform dependent
    - Windows line separator \r\n
    - Unix line separator \n

- To get the line separator for a specific platform use
  String lineSeparator = System.getProperty("line.separator");

- Keyboard entries end the line with the ENTER key which corresponds to \n

- token reading methods do not read the terminating delimiters

- if the nextLine( ) method is invoked after a token reading method, the nextLine( ) method reads characters that start with from this delimiter and ends with the line separator; the line separator is read but is not part of the string returned by nextLine( )

```
Scanner input = new Scanner(new file("test.txt"));
int intValue = input.nextInt( );
String line = input.nextLine( );
```

test.txt
34 567

➔ ((intValue == 34) &&(line contains the characters ' ', '5' '6' '7')

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt( );
String line = input.nextLine( );
```

Keyboard Entries
34, ENTER, 567, ENTER

➔ ((intValue == 34)&&(line contains an empty string))

nextInt( ) reads 34 & stops at the line separator delimiter
nextLine( ) ends after reading the line separator; it returns the characters read before the line separator, i.e., it returns an empty string to the line variable

Listing 8.8

Usage: >java ReplaceText sourceFile targetFile oldString newString

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

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

        File sourceFile = new File(args[0]);
        if (!sourceFile.exists( )){//print message; System.exit(0);}

        File sourceFile = new File(args[1]);
        if (targetFile.exists( )){//print message; System.exit(0);}

        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( );
    }
}
```

iv. GUI File Dialogs

javax.swing.JFileChooser Class

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

public class ReadFileUsingJFileChooser
{
  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");
    }
  }
}
```

Open button clicked → APPROVE_OPTION

Displays a dialog box

Returns the selected file from the dialog box