Chapter 10 – Using Input and Output

Console Input Using Character Streams

* + System.in = byte stream
  + To convert it to char stream, need to wrap it in the **Reader**
  + Console Reader class
  + **Buffered Reader**
    - Supports buffered input stream
    - Can't be constructed directly from System.in
    - You must convert it into character stream with **InputStreamReader** = converts bytes to characters
    - Use constructor: *InputStreamReader (InputStream inputStream)*
    - System.in refers to the object of type InputStream, it can be used for inputStream
    - Next using object produced by InputStreamReader, construct BufferedReader:
      * BufferedReader(Reader inputReader)
    - *BufferedReader br = new BufferedReader(new InputStreamReader(System.in))*
      * Converts byte stream to character stream and links it to buffered reader
      * br = character based stream that is linked to console through System.in
  + **Reading Characters = read()**
    - Buffered Reader has **read()**. Has 3 versions:
      * Int read() throws IOException
        + Reads unicode character
        + Returns -1 when attempt was made at the end of the stream
      * Int read(char data[]) throws IOException
        + Reads char from the input stream and puts into data[] until

Full

End of stream is reached

Error occurs

* + - * + -1 when end of stream reached
      * Int read(char data[], int start, int max) throws IOException
        + Reads input into data[] begin at start until max characters are stored
        + -1 when end of the stream reached
      * All throw IOException on error
  + **Reading Strings = readLine()**
    - String readLine() throws IOException
    - Returns String object
    - **Null** at the end of the stream

Console Output Using Character Streams

* + System.out is permissable to use to write in the console, but recommended to use only for
    - Debugging purposes
    - Sample Programs
  + Real world programs preffered method is **PrintWriter** = character based
  + Character based classes makes it easier to internationalize your program
  + **PrintWriter** defines several constructors
    - PrintWriter(OutputStream outputStream, boolean flushingOn)
      * outputStream is object of OutputStream
      * flushingOn controls weather Java flushes the output stream every time **println()** method is called
      * If true = flushing automatically takes place
      * If false = flushing is not automatic
      * Supports print() and println() for all types including Object
      * Can be used the same way as System.out
      * If argument is **not a primitive type**
        + PrintWriter method calls the **object's toString()** method and then prints out the result
      * Console
        + Write using PrintWriter, speficy System.out for output stream and flush the stream after each call to println()
        + PrintWriter pw = new PrintWriter(System.out, true);
* File I/O Using Character Streams
  + Byte oriented file handling is often most common
  + Advantage of character streams:
    - Operate directly on Unicode characters
  + To perform character-based file I/O, you eill use the **FileReader** and **FileWriter** classes.
  + **FileWriter**
    - Creates a **Writer** to use to write to a file
    - 2 commonly used constructors:
      * FileWriter(String fileName) throws IOException
      * FileWriter(String fileName, boolean append) throws IOException
    - fileName = path name of the file
    - If append = true, output is appened to the end of the file, if false = overwritten.
    - Derived from **OutputStreamWriter, Writer** can access methods of these classes
  + **FileReader**
    - Creates a **Reader** that you can use to read the contents of a file.
    - Commonlt used constructor:
      * FileReader(String fileName) throws FileNotFoundException
      * fileName = full path of the file
    - Derived from **InputStreamReader, Reader** can access methods of all of these classes

Type Wrappers to Convert Numeric Strings

* + Println() method automatically converts numeric values into their human readable form from bytes.
  + Read() method does not provide parallel functionality that reads and converts string containing numberic value into its internal, binary format.
  + Java privides Type Wrappers
    - Classes to encapsulate or wrap the primitive types
    - Needed because primitive types are not objects and that limits their use to some extent:
      * Can't be passed by reference
  + Type wrapper clasees are:
    - Double
      * Static double parseDouble(Strinfgstr) throws NumberFormatException
    - Float
      * Static float parseFloat(String str) throws NumberFormatException
    - Long
      * Static long parseLong(String str) throws NumberFormatException
    - Integer
      * Static int parseInt(String str) throws NumberFormatException
    - Short
      * Static short parseShort(String str) throws NumberFormatException
    - Byte
      * Static byte parseByte(String str) throws NumberFormatException
    - Character
    - Boolean
  + Offer wide array methods to integrate the primitive types into Java's object hirearchy.
  + Numeric wrappers define methods to convert a numeric string into its corresponding binary equivalent
  + Parsing methods give easy way to convert a numeric value which is read as a string from a keyboard or a text file, into its proper internal format.
  + **Scanner** 
    - Reads formatted (human-radable) format and converts it into its binary form
    - Read input from variaety of sources:
      * Console
      * Files
      * Etc.
    - Can read numeric values enetered by keyboard and assign its value to a variable.
    - To read from keyboard link it to console input: Scanner(InputStream from)
      * Creates Scannertthat uses stream specified by from
      * Scanner consoleInput = new Scanner(System.in);
      * Works, because System.in = InputStream
      * consoleInput can be used to read input from keyboard
      * Procedure:
        1. Determine if specific type of input is available by one of the Scanner's hasNextX methods, where X is the type of data desired.
        2. If input is avialble, read it by calling one of Scanner's nextX methods
    - Defines 2 sets of methods that enable to read input
      * hasNext
        1. hasNextInt() returns true only if

Next item in the stream is the human readable form of an interger.

* + - * 1. hasNextDouble()
        2. Etc.
        3. Return true if desired data type is next avaialble item in the data stream and false otherwise
      * Next methods
        1. nextInt()
        2. nextDouble()
        3. Convert the human radable form of the data into its internal, binary represenation and returns result.
    - Scanner conin = new Scanner(System.in);

Int i;

If (comin.hasNextInt()) I = conin.nextInt();

* + Usually both methods need to be used together
  + If next can't find the type it is looking for throws **InputMismatchException**