Use the *Scanner* class for reading text files

File inputFile = new File(“input.txt”);

Scanner in = new Scanner(inputFile);

When writing text files, use the *PrintWriter* class and the *print/println/printf* methods.

PrintWriter out = new PrintWriter(“output.txt”);

out.println(“Hello, World!”);

out.printf(“Total: %8.2f\n”, total);

Close all files when you are done processing them.

in.close();

out.close();

The *next* method reads a string that is delimited by white space.

to read just the words and discard anything that isn't a letter. You achieve this task by calling the *useDelimiter* method on your Scanner object

Scanner in = new Scanner(…);

in.useDelimiter(“[?A-Za-z]+”);

The *Character* class has methods for classifying characters

Character.isDigit(ch)

The *trim* method returns the string with all white space at the beginning and end removed

countryName = countryName.trim();

If a string contains the digits of a number, you use the Integer.parseInt or Double.parseDouble method to obtain the number value. must be a string containing the digits of an integer, without any additional characters. Not even spaces are allowed!

int populationValue = Integer.parseInt(population);

double price = Double.parseDouble(input);

A construct such as %-10s or %10.2f is called a format *specifier*: it describes how a value should be formatted.

First %, Next, there are optional “flags” , Next is the field width, last is format type.

%-10s formats a left-justified string.

%10.2f formats a floating-point number, right-justified

Programs that start from the command line receive the command line arguments in the main method

To signal an exceptional condition, use the throw statement to throw an exception object.

throw new IllegalArgumentException(“Amount exceeds balance”);

When you throw an exception, execution does not continue with the next statement but with an **exception handler**

Place the statements that can cause an exception inside a *try* block, and the handler inside a *catch* clause.

try{ …}

catch (IOException exception){ exception.printStackTrace();}

exception.printStackTrace(); prints out error message that usually show in the java run window.

The *Scanner* constructor with *File* can throw a FileNotFoundException.

*Scanner.next* can throw a NoSuchElementException.

*Integer.parseInt* can throw a NumberFormatException.

Throw an exception as soon as a problem is detected. Catch it only when the problem can be handled.

Do Not Use *catch* and *finally* in the Same *try* Statement

**Checked /Unchecked exceptions**

Descendants of **Runtime**Exception, such as as IndexOutOfBoundsException or IllegalArgumentException indicate errors *in your code*. They are called **unchecked exceptions**.

All other exceptions are **checked exceptions**. These exceptions indicate that something has gone wrong *for some external reason beyond your control*

Checked exceptions are due to external circumstances that the programmer cannot prevent. The compiler checks that your program handles these exceptions.

Add a ***throws*** clause to a method that can throw a checked exception.

Once a *try* block is entered, the statements in a *finally* clause are guaranteed to be executed, whether or not an exception is thrown.

finally{ out.close() ;}

try (PrintWriter out = new PrintWriter(filename))

{ Write output to out.}

In this case java will automatically *close* the file.