

General

Where appropriate, you should do input verification. In other words if the user is supposed to input, for example, a positive number, an appropriate error message should be displayed when the user does not enter a positive number. The user should then get additional chances to enter input.

Java

The version of Java installed in the electronic classrooms here is the Java SDK 1.5.

Output with Java 1.5 uses the standard form: `System.out.println` (or `System.out.print` as appropriate)

Input with Java 1.5 input can be handled by either of the following methods:

- `BufferedReader`
- `Scanner`

Use whichever method you are most experienced with.

Input with `BufferedReader`:

- When using the `BufferedReader` make sure that the first statement you include in your Java program is the statement:

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

- Before doing any input from the keyboard you will need to declare a `BufferedReader` object that is tied to `System.in`. (In the following example the `BufferedReader` object is named "br")

```
BufferedReader br = new BufferedReader (new  
    InputStreamReader(System.in));
```

- Then you may use this object to read Strings (one line at a time) from the keyboard:

```
System.out.println("Please enter a line of input> ");  
String line = br.readLine();  
System.out.println("Please enter another line> ");  
String line2 = br.readLine();
```

The above example prompts the user for input and reads two lines of input from the keyboard. `line1` references the first string, while `line2` references the second..

- Use `Integer.parseInt` or `Double.parseDouble` to convert any Strings to ints or doubles, respectively. Reading in a line of input and converting it to a single numeric type can be done in one statement. For example:

```
int i = Integer.parseInt( br.readLine() );
```

- If a line of input has `NUM_ELEMS` numeric values you can use a String tokenizer to divide the line into tokens and then convert each token into the appropriate numeric value; For example:

```
import java.io.*;
import java.util.*;
...
StringTokenizer line = new StringTokenizer(" ");
try{
    line = new StringTokenizer(br.readLine());
} catch (Exception e) {}
for (i = 0; i < NUM_ELEMS; i++)
    array[i] = Integer.parseInt(line.nextToken());
```

- Finally, any methods that contain a call to `readLine()` (as well as methods that call a methods that calls `readLine()` need a "throws Exception" clause in the method header, unless you want to deal with the Exception handling and place the `readLine()` call in a `try...catch` block. **This type of exception handling is NOT required for this contest.** So it is recommended that you just include the "throws Exception" clause.

```
public static void main(String[] args) throws Exception
{
    // throws exception code in here.
}
```

Input with the Scanner class

- When using the Scanner class for input you will need to include the following import statement as the first line in all your Java files:

```
import java.util.Scanner;
```

- The next step is to declare a Scanner object that is tied to `System.in`

```
Scanner scanIn = new Scanner(System.in);
```

- The scanner object can now be used to read Strings, doubles and ints directly from the keyboard. The methods `nextInt` and `nextDouble` allow you to read integers or doubles from the keyboard; for Example:

```
System.out.println("Enter quantity of toys> ");
int number = scanIn.nextInt();
System.out.println("Enter price for each item> ");
double price = scanIn.nextDouble();
```

- The `nextLine()` method returns the next line of input from the keyboard, `nextLine` uses the newline character to determine the end of input. The `next()` method returns the next word from the input, `next()` uses whitespace (tabs, spaces or newline characters) to determine the end of input.

```
System.out.println("Enter name> ");
String name = scanIn.nextLine();
```

```
System.out.println("Enter name> ");
String anotherName = scanIn.next();
```

- `nextLine()` would return a String consisting of multiple words such as "John Smith", while `next()` would return a single word, "John".
- The Scanner class has methods that test to see if it is possible to read a specific type of input as the next item.
 - `boolean hasNext()`
 - `boolean hasNextDouble()`
 - `boolean hasNextInt()`
 - `boolean hasNextLine()`

These functions return true if it is possible to read any non-empty String, double, int or line of input, respectively. The following example reads an undetermined number of integers from the keyboard. All these functions may BLOCK if there is NO input available. In particular, pressing the enter key will have no apparent effect as the method `hasNextInt()` is expecting some valid input besides whitespace.

```
System.out.println("Enter numbers to add (q to quit)> ");
while (scanIn.hasNextInt())
{
    num = num + scanIn.nextInt();
}
System.out.println("num is " + num);
```

This program will stop when input other than an integer is entered.

- Finally, any methods that contain a call to `next()`, `nextLine()`, `nextInt()`, or `nextDouble()` (as well as methods that call one of the above methods) need a "throws Exception" clause in the method header, unless you want to deal with the Exception handling and place the methods call in a `try...catch` block. **This type of exception handling is NOT required for this contest.** So it is recommended that you just include the "throws Exception" clause.

```
public static void main(String[] args) throws Exception
{
    // Scanner methods in here.
}
```

Other Modifications with Java 1.5

Generics

With Java 1.5 collections are generic classes with type parameters. For example, `ArrayList<E>` collects elements of type `E`; `Map<K, V>` maps keys of type `K` to values of type `V`. When a generic type is used, the type parameters are replaced with the actual types; for example, `ArrayList<Dog>` or `Map<Location, Dog>`. In versions of Java before 1.5 collections store elements of type `Object`.

Java 1.5

```
private ArrayList<Dog> dogs;
private Map<Location, Dog> environment;
```

Pre Java 1.5

```
private ArrayList dogs; // contains Dog objects
private Map environment; // Maps Location objects to Dog objects
```

Autoboxing

Autoboxing is the automatic conversion between primitive types and corresponding wrapper classes.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
// 25 is automatically converted to new Integer(25)
numbers.add(25);
int num = numbers.get(0); // intValue is automatically called
```