Looping

• Generalized loops
  – Do-while
  – While
  – For

• Structured loop exit statements
  – Break
  – Continue

• Input from keyboard
  – Streams and TokenStream Class
    – See JavaGently, 4.1-4.3
Generalized Loop Construct

- Generalized loop construct:

  `<loop-header>` { //either of statements
    `<statements>_{1}` //can be empty
    `<test>
    `<statements>_{2}
  }

- Execution: `<loop-header>` `<stmts>_{1}` `<test>
  `<stmts>_{2} `<stmts>_{1} `<test> `<stmts>_{2} `<stmts>_{1}
  `<test> exit loop
For Loop

\[ \text{for (start ; check ; update) \{ block \}}; \]
While Loop

<while-loop> → while ( <conds> ) <block>;
where the variables in <conds> are initialized before the loop starts and <block> should contain statement(s) changing the values of variables in <conds>

• Execution: repeat the following:
<conds> <block> <conds> <block> ... <conds> exit loop
While Loop

\[
\begin{align*}
\text{<loop-header>} & \{ \\
& \quad \text{//initialize <conds>} \\
& \quad \text{while}\{ \\
& \quad \quad \text{<statements>}_1 \\
& \quad \quad \text{<test>} \\
& \quad \quad \text{<statements>}_2 \\
& \quad \} \\
& \} \\
\end{align*}
\]
**Do-while Loop**

\[ \text{do-while-loop} \rightarrow \text{do} \{ \text{<block>} \text{ while <cond> } \} ; \]

where the variables in <cond> are changed by statements in <block>

- Execution model: <block> <cond> <block> <cond>... <cond> exit loop
- Test here is AFTER loop body statements
- Always do first iteration
Do-while loop

loop-header> {  //initialize <conds>
   do{
      <block>
      while <cond>
      <stmts><cond>1
      <stmts><cond>2
   }  }
Loops

- **Do-while** loops always perform their first iteration; **While** and **for** loops check their test before doing the first iteration.
- **Do-while** loops perform their check *after* the loop body, whereas **while** and **for** loops perform their check *before* their loop body.
- **Do-while** and **while** loops are used in situations where counting loop iterations isn’t appropriate.
Uses of Loops

- **For** loops are used when number of iterations is known in advance.
- **While** and **do-while** loops are used when a condition signals the end of processing in the loop.
- `for (; ; ){...}` is equivalent to `while (true) {...}`
- Need a way to exit an indeterminate loop
  - `break` - exit current block
  - `continue` - start next iteration
class Summation extends Object{
    public static void main(String[] args) {
        //sums all numbers until their sum reaches 500
        //
        int sum=0, i = 1;
        while (sum < 500){
            sum = sum + i;
            i++;
        }
        System.out.println("sum of numbers from 1 to " + (i-1) + " is " + sum);
    }
}

why (i - 1)?
For Loop, Same Example

class Summation extends Object{
public static void main(String[] args) {
//sums all numbers from 1 to 1000
//but stops at an n, when sum from 1 to n reaches 500
//
    int sum=0,i;
    f1: for (i = 1; i<=1000; i++) {
        sum = sum + i;
        if (sum>500) break f1;
    }
    System.out.println("sum of numbers from 1 to " + i + " is " + sum);
}
}

10 romulus!111> java Summation
sum of numbers from 1 to 32 is 528
Break and Continue

loop1: for ( ... ){
    ...
    loop2: for (....){
        ...
        if (...) continue loop1;
        if (...) break loop1
        loop3: while (...){
            ...
            if (...) break loop3;
            if (...) continue loop2;
        }
    }
}
Nested While Loop with Break

class Summation extends Object{
    public static void main(String[] args) {
        //sums 1..n for all numbers from 1 to 1000 and prints sums,
        //but stops at an n, when sum from 1 to n reaches 500
        int sum,i=1,j;
        w1: while (i < 1000){
            sum = 0;
            j = 1;
            w2: while (j <= i){
                sum = sum + j;
                if (sum>500)
                    {System.out.println("sum greater "
                        "than 500 for sum 1 to j="+j);
                        break w1;}
                j++;
            }
        System.out.println("sum of numbers from 1 to "
            + i + " is " + sum);
        i++;
    }
}
Output of Example

7 romulus!111> java Summation
sum of numbers from 1 to 1 is 1
sum of numbers from 1 to 2 is 3
sum of numbers from 1 to 3 is 6
...
sum of numbers from 1 to 31 is 496
sum greater than 500 for sum 1 to j=32
8 romulus!111>
Another Nested Loop Example

class Summation extends Object{
    public static void main(String[] args) {
        //sums all numbers from 1 to 1000
        //but stops at an n, when sum from 1 to n reaches 500
        //and doesn't add in any multiples of 10
        int sum, i, j;
        w1: for(i=1; i < 1000; i++) {
            sum = 0;
            if (i%10 == 0) continue w1;
            j = 1;
            w2: while (j <= i) {
                sum = sum + j;
                if (sum>500){System.out.println("sum greater than 500 for sum 1 to j= " + j);
                    break w1;}
                j++;
            }
            System.out.println("sum of numbers from 1" "to "+ i + " is " + sum);
        }
    }
}
Output from 2nd Nested Loop

40 romulus!111> java Summation
sum of numbers from 1 to 1 is 1
sum of numbers from 1 to 2 is 3
sum of numbers from 1 to 3 is 6
sum of numbers from 1 to 4 is 10
...
sum of numbers from 1 to 28 is 406
sum of numbers from 1 to 29 is 435
sum of numbers from 1 to 31 is 496
sum greater than 500 for sum 1 to j= 32
Input

• How to input values to your program from your terminal?
• How to input values to your program from a file?
• *Stream* - a sequence of values
  – Input stream is typed from keyboard or is on a file
• Java Development toolKit (JDK) contains standard i/o library; See `java.io` in Javadocs
Deprecated Methods

• Our textbook is based on **JDK 1.0** whereas the newest is **JDK 1.1**; we have made changes to the Java Gently Text class to avoid problems, both the class and with these updates.

• New library updates allow replacement of some methods with others, where necessary.

• Methods which are about to be replaced thusly are called *deprecated* and compiler will complain when you use them.

• Usually they are unavailable in the next release.
Input

• What to do? change any use of a deprecated method to the replacement method
  – For example, *BufferedReader* is new class used for input and supported by Java in current release
  – *DataInputStream* (see textbook) is a formerly supported class which will work now, but not in next release of JDK
Input

- Java’s library functions allow reading of input a line at a time as a String
- Reading an entire line in from the keyboard as a String isn’t convenient
- Better to break input into pieces, (e.g., an entire integer, an entire double numeric value)
- TokenStream allows reading of individual data items by type
Input

• Using an input stream presents possibility of something going wrong during input process such as running out of input

• Java notifies you if something goes wrong by throwing an exception to handle these situations
  – For I/O, unusual conditions may lead to IOException

    ```java
    public static void main(String[] args) throws IOException {
        // ..... }
    ```
TokenStream Class

• Available by importing `cs111.io.*`
  – See `/usr/local/class/cs111/packages/src/cs111/io/*`

TokenStream class:
  TokenStream(); TokenStream(String fileName);
  String readString();
  String readString(String prompt);
  int readInt();
  int readInt(String prompt);
  double readDouble();
  double readDouble(String prompt);
  char readChar();
  char readChar(String prompt);
TokenStream Methods

- `readInt()`, `readDouble()` - read 1 item of numerical data of the appropriate type
- `readString()` - reads 1 string from a line of input
- `readChar()` - reads 1 char item from a line of input

```java
//inp is keyboard
TokenStream inp = new TokenStream();
int i = inp.readInt();
double d = inp.readDouble();
```
TokenStream Constructors

- *TokenStream()* - used to construct an input stream from the keyboard
- *TokenStream(String fileName)* - used to construct an input stream from a file
- Several input streams can be used by the same program (not true of Text class in textbook)
TokenStream Class

• Allows spaces between data items, but not parts of the same string
  – *Please enter your name* > barbara ryder
  – if program is executing a `readString()` , it will only see “barbara”

• Ignores blank lines
How to Find End of Input?

- Store a count of number of items as the first input and keep a running count (inflexible)
- Use a special termination value to mark end of input (somewhat restrictive)
  - e.g., -1 entered as an item count; 999 as an age;
- Use an end-of-file exception to mark the end of the input items

<table>
<thead>
<tr>
<th>Count</th>
<th>Mark</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
import java.io.*;
import cs111.io.*;
class SumfromKeyboard extends Object{
public static void main(String[] args) throws IOException {
    //sums a sequence of numbers entered from the keyboard
    TokenStream inp = new TokenStream();//create keyboard
    double sum = 0.0, d; //stream
    System.out.print("Enter count of numbers to be summed");
    int n = inp.readInt();
    System.out.println("Enter numbers");
    for (int i=0; i<n; i++)){//executed n times
        d = inp.readDouble();
        sum += d;
    }
    System.out.println("sum = " + sum);
}}
import java.io.*;
import cs111.io.*;

class SumwithMark extends Object{
    public static void main(String[] args) throws IOException {
        //sums a sequence of numbers entered from the keyboard
        TokenStream inp = new TokenStream();
        double sum = 0.0, d = 0.0;
        System.out.println(" Enter numbers to be summed,
         ending with -1 ");
        for (;d != -1.0;){//note empty init and incr
            d = inp.readDouble();
            if (d!= -1.0) sum += d; //don't add mark
        }
        System.out.println("sum = " + sum);
    }
}
import java.io.*;
import cs111.io.*/;

class SumEOF extends Object{
    public static void main(String[] args) throws IOException {
        //sums a sequence of numbers entered from the keyboard
        TokenStream inp = new TokenStream();
        double sum = 0.0;
        System.out.println(" Enter numbers to be summed," + "
            "ending with control-D ");
        try{ //type control-d to signal end of input
            for (;;){ //indefinite loop or loop forever
                d = inp.readDouble();
                sum += d;
            } } 
        catch (EOFException e) { } 
        System.out.println("sum = " + sum); } }