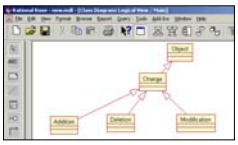
CPSC 233: Introduction to Computers II

Object-oriented programming



Object-oriented design

And a whole lot ole fun (you'll have a …)

James Tam

Administrative Information

Contact Information

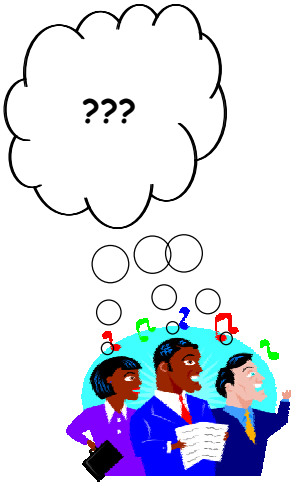
- Office: ICT 707
- Phone: 210-9455
- Email: **tamj@cpsc.ucalgary.ca**

Office hours

- Office hours: MW 12:00 – 12:50
- Email: (any time)
- Appointment: phone or call
- Drop by for urgent requests (but no guarantee that I will be in!)

James Tam

Feedback



Dilbert © United Features Syndicate

How You Will Be Evaluated

Assignments (30%)

- Assignment 1 (Due Monday January 26, worth 2% of overall grade): *Writing a simple Java program*
- Assignment 2 (Due Monday February 2, worth 2% of overall grade): *Introduction to classes*
- Assignment 3 (Due Friday February 13, worth 5% of overall grade): *Dynamic memory allocation through arrays of references*
- Assignment 4 (Due Friday February 27, worth 7% of overall grade): *Writing programs with multiple classes*
- Assignment 5 (Due Friday March 26, worth 8% of overall grade): *Inheritance and exceptions*
- Assignment 6 (Due Friday April 16, worth 6% of overall grade): *Designing a simple graphical-user interface, file input and output*

How You Will Be Evaluated (3)

Exams (70%)

- Midterm exam (30%): Written in-class on Friday March 5.
- Final exam (40%): Scheduled by the Registrar's Office sometime between April 20 – 30.

Note: You must pass both the assignment component and the exam component in order to get a C- or higher in the course.

Course Resources

Course website:

- http://www.cpsc.ucalgary.ca/~tamj/233

Course textbook:

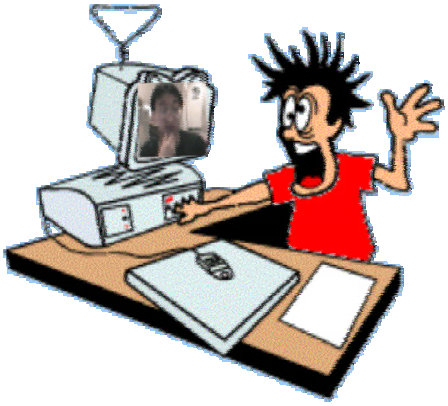
- Big Java by Cay Horstmann (Wiley)

Another good website:

- http://developer.java.sun.com/developer/infodocs/

CPSC 231: What Was It Like

A whole lot of work!



CPSC 233: What To Expect

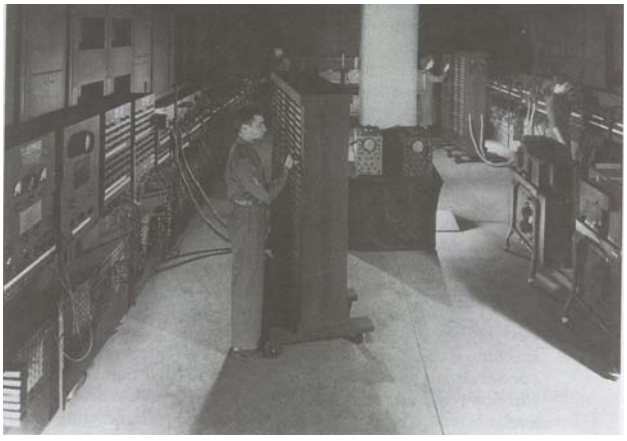
Even more work!!!

Pascal-Java Transition

- History behind Java
- Creating, compiling and executing programs
- Basic program structure
- Text based output
- Documentation
- Variables and constants
- Text based input
- Decision making
- Loops
- Some Java libraries

Java: History

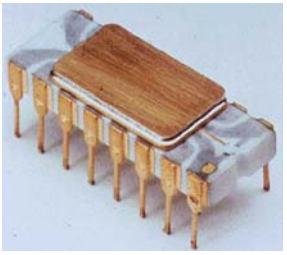
Computers of the past



Picture from "The History of Computing Technology" by Michael R. Williams

Java: History (2)

The invention of the microprocessor revolutionized computers



Java: History (3)

It was believed that the next step for microprocessors was to have them run intelligent consumer electronics

Java History (4)

Sun funded an internal research project "Green":

- Result: A programming language called "Oak"

Blatant advertisement: James Gosling was a graduate of the U of C Computer Science program.

Wav file from "The Simpsons" © Fox

Java History (5)

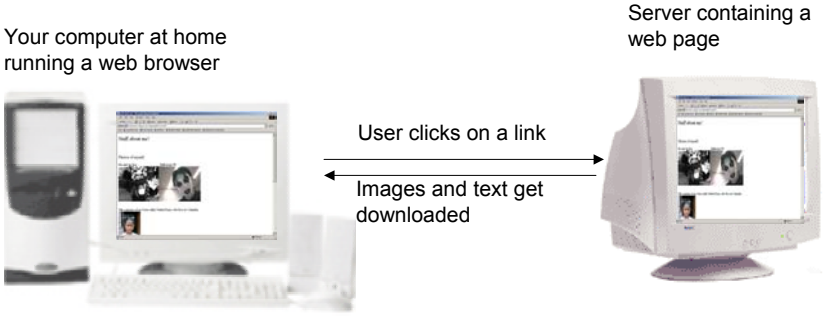
- Problem: There was already a programming language called Oak.
- The "Green" team met at a local coffee shop to come up with another name...Java!

Java: History (6)

- The concept of intelligent devices didn't catch on
- Project Green and work on the Java language was nearly canceled

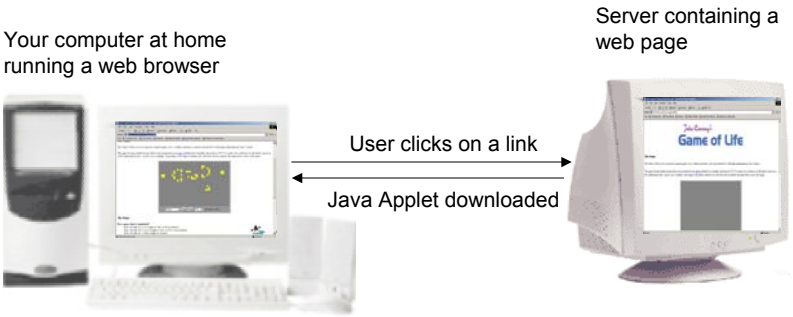
Java: History (7)

- The popularity of the Internet resulted in re-focusing Java on computers.
- Prior to the advent of Java web pages allowed you to download only text and images.



Java: History (8)

- Java enabled web browsers allowed for downloading of programs (Applets)

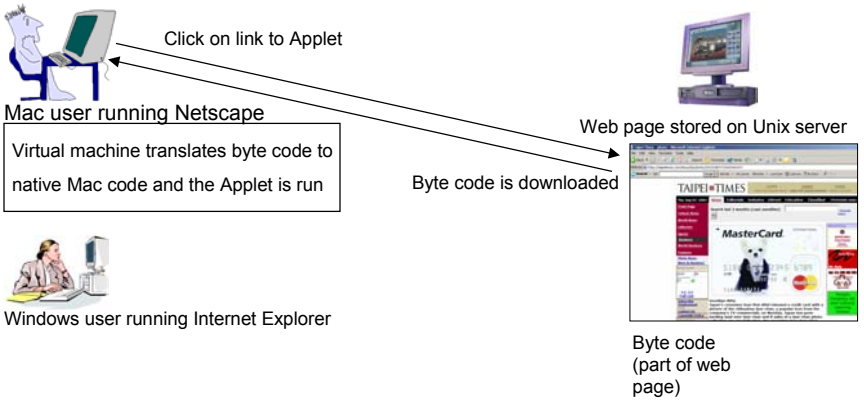


Java version of the Game of Life: http://www.bitstorm.org/gameoflife/

Online checkers: http://www.darkfish.com/checkers/index.html

Java: Write Once, Run Anywhere

Consequence of Java's history: platform-independence



Java: Write Once, Run Anywhere

Consequence of having web-based roots: platform-independent

Mac user running Netscape

Web page stored on Unix server

Click on link to Applet

Byte code is downloaded

Windows user running Internet Explorer

Virtual machine translates byte code to native Windows code and the Applet is run

Java: Write Once, Run Anywhere (2)

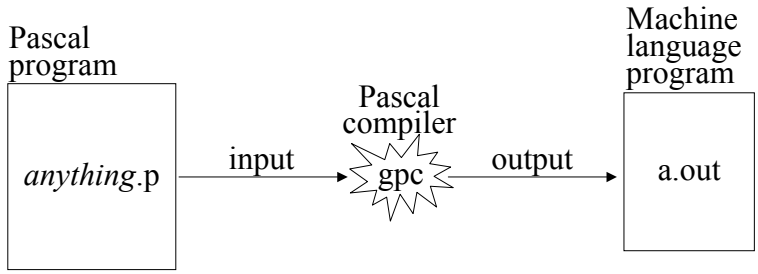
But Java can also create standard (non-web based) programs

Dungeon Master (Java version)

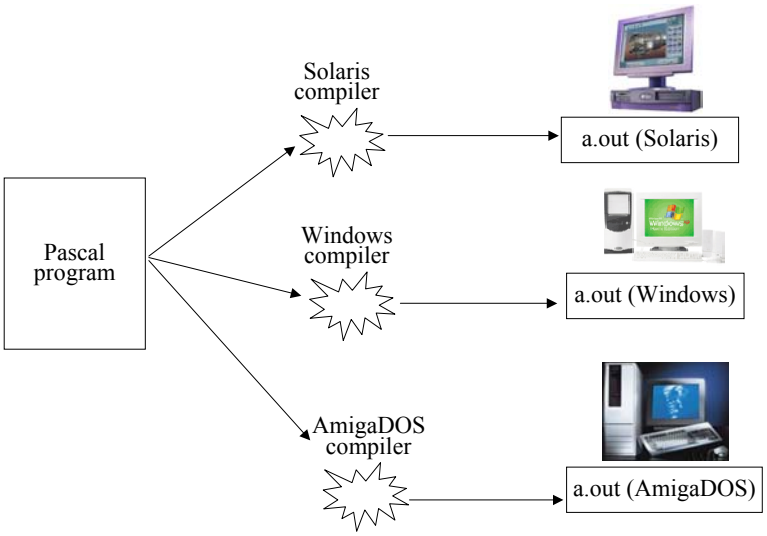
http://www.cs.pitt.edu/~alandale/dmjava/

Don't play this game on the CPSC network!

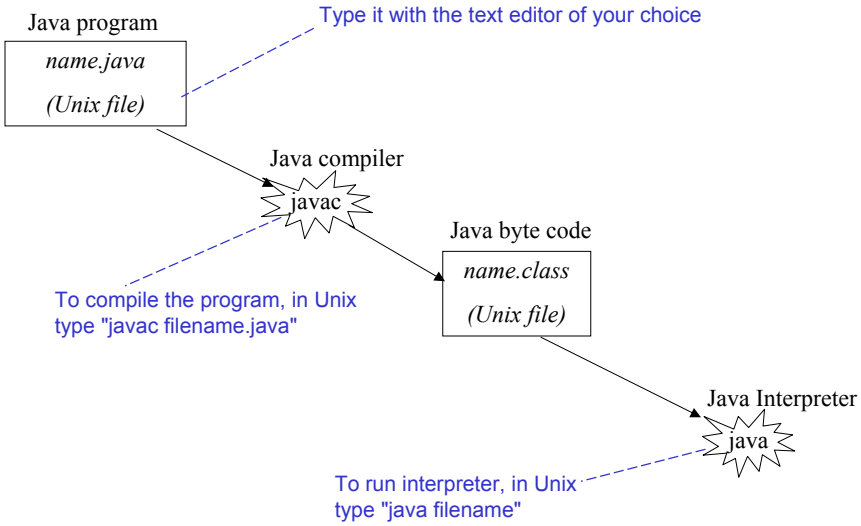
Review: Compiling Pascal Programs



Compiling Pascal Programs On Different Operating Systems



Creating, Compiling And Running Java Programs: On The Computer Science Network



Java Vs. Java Script

Java

- A complete programming language developed by Sun.
- Can be used to develop either web based or stand-alone software
- Many standardized code libraries available
- For more complex and powerful programs

Java Script

- A small language that's mostly used for web-based applications
- Good for programming simple effects for your web page e.g., roll-overs
- e.g., http://www.discoverit.co.uk/webdesign/javascript.htm

Which Java?

Java 2 SDK (Software Development Kit), Standard Edition 1.4.2

- JDK (Java development kit) – for developing Java software
- JRE (Java Runtime environment) – for only running Java software
  - Java Plug-in – a special version of the JRE designed to run through web browsers

http://java.sun.com/j2se/1.4.2/download.html

Which Java?

Java 2 SDK (Software Development Kit), Standard Edition 1.4.2

- JDK (Java development kit) – for developing Java software
- JRE (Java Runtime environment) – for only running Java software
  - Java Plug-in – a special version of the JRE designed to run through web browsers

http://java.sun.com/j2se/1.4.2/download.html

James Tam

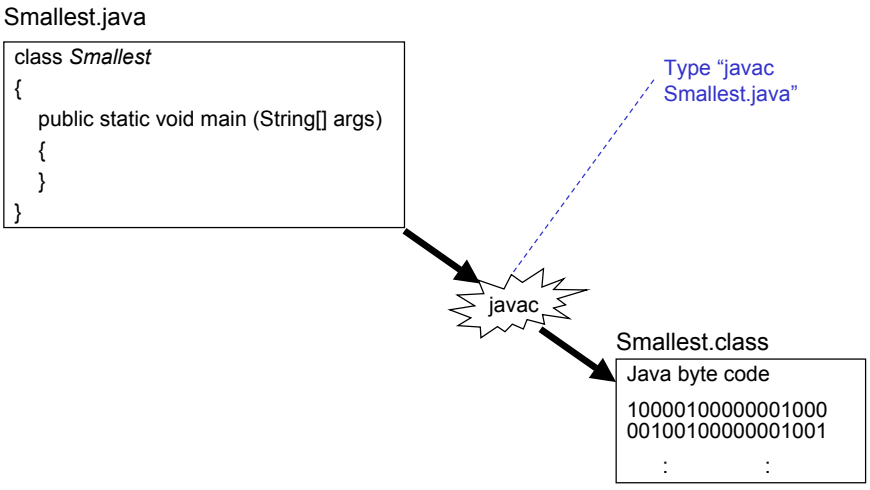
Smallest Compilable And Executable Pascal Program

```
program smallest;
begin
end.
```

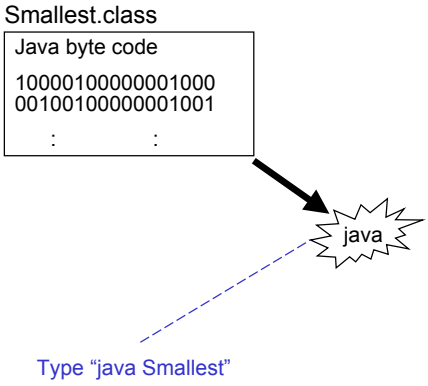

Smallest Compilable And Executable Java Program

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

Compiling The Java Smallest Program



Running The Smallest Java Program



The Semicolon In Pascal

Pascal

- Used to separate statements within a block
- This is okay in Pascal:

```
program test (output);
begin
 writeln("one");
 writeln("two")
end.
```


The Semicolon In Java

Java

- Follows each statement
- This is not okay in Java:

```
class BadExample
{
    public static void main (String [] args)
    {
        System.out.println("one");
        System.out.println("two")
    }
}
```

Documentation / Comments

Pascal

(* Start of documentation

*) End of documentation

Java

Multi-line documentation

/* Start of documentation

*/ End of documentation

Documentation for a single line

// Everything until the end of the line is treated as a comment

Output In Pascal And Java

Pascal

write('...');

writeln('..');

Java

System.out.print("…");

System.out.println("…");

Java Output

Format:

System.out.println(<*string or variable name one*> + <*string or variable name two*>..);

Examples (Assumes a variable called num has been declared.):

- System.out.println("Good-night gracie!");
- System.out.print(num);
- System.out.println("num=" +num);

Output : Some Escape Sequences

Escape sequence	**Description**
\t	Horizontal tab
\r	Carriage return
\n	New line
\”	Double quote
\\	Backslash

Some Built-In Types Of Variables In Java

Type	Description
byte	8 bit signed integer
short	16 but signed integer
int	32 bit signed integer
long	64 bit signed integer
float	32 bit signed real number
double	64 bit signed real number
char	16 bit Unicode character
boolean	1 bit true or false value
String	A sequence of characters between double quotes (“”)

Java Vs. Pascal Variable Declarations

Pascal

Format:

<*variable name*> : variable type;

Example:

num : integer;

Java

Format:

variable type <*variable name*>;

Example:

long num1;

double num2 = 2.33;

Location Of Variable Declarations

```
class <name of class>
{
   public static void main (String[] args)
   {
         // Local variable declarations occur here

         << Program statements >>
               :          :

   }
}
```


Constants In Pascal Vs. Java

Pascal:

Format:

```
const
    <CONSTANT NAME> = <Value>;
```

Example:

```
const
    SIZE =  5;
```

Java

Format:

```
final <constant type> <CONSTANT NAME> = <value>;
```

Example:

```
final int SIZE = 100;
```

Location Of Constant Declarations

```
class <name of class>
{
   public static void main (String[] args)
   {
         // Local constant declarations occur here
         // Local variable declarations

         < Program statements >>
               :        :

   }
}
```


Java Keywords

abstract	boolean	break	byte	case	catch	char
class	const	continue	default	do	double	else
extends	final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long	native
new	package	private	protected	public	return	short
static	super	switch	synchronized	this	throw	throws
transient	try	void	volatile	while		

Variable Naming Conventions In Java

Compiler requirements

- Can't be a keyword nor can the special constants true, false or null be used
- Can be any combination of letters, numbers, underscore or dollar sign (first character must be a letter or underscore)

Common stylistic conventions

- The name should describe the purpose of the variable
- Avoid using the dollar sign
- With single word variable names all characters are lower case
  - e.g., double grades;
- Multiple words are separated by capitalizing the first letter of each word except for the first word
  - e.g., String firstName = "James";

Constant Naming Conventions In Java

Compiler requirements

- Can't be a keyword nor can the special constants true, false or null be used
- Can be any combination of letters, numbers, underscore or dollar sign (first character must be a letter or underscore)

Common stylistic conventions

- The name should describe the purpose of the constant
- Avoid using the dollar sign
- All characters are capitalized
  - e.g., float SIZE = 100;
- Multiple words are separated with an underscore between each word.
  - e ,g, float CORPORATE_TAX_RATE = 0.46;

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
1	expression++ expression--	Post-increment Post-decrement	Right to left
2	++expression --expression + - ! ~ (type)	Pre-increment Pre-decrement Unary plus Unary minus Logical negation Bitwise complement Cast	Right to left

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
3	* / %	Multiplication Division Remainder/modulus	Left to right
4	+ -	Addition or String concatenation Subtraction	Left to right
5	<< >>	Left bitwise shift Right bitwise shift	Left to right

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
6	< <= > >=	Less than Less than, equal to Greater than Greater than, equal to	Left to right
7	== !=	Equal to Not equal to	Left to right
8	&	Bitwise AND	Left to right
9	^	Bitwise exclusive OR	Left to right

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
10	|	Bitwise OR	Left to right
11	&&	Logical AND	Left to right
12	||	Logical OR	Left to right

Common Java Operators / Operator Precedence

Precedence level	Operator	Description	Associativity
13	=	Assignment	Right to left
	+=	Add, assignment	
	-=	Subtract, assignment	
	*=	Multiply, assignment	
	/=	Division, assignment	
	%=	Remainder, assignment	
	&=	Bitwise AND, assignment	
	^=	Bitwise XOR, assignment	
	|=	Bitwise OR, assignment	
	<<=	Left shift, assignment	
	>>=	Right shift, assignment	

Post/Pre Operators

```
class Example1
{
   public static void main (String [] args)
   {
      int num = 5;
      System.out.println(num);
      num++;
      System.out.println(num);
      ++num;
      System.out.println(num);
      System.out.println(++num);
      System.out.println(num++);
   }
}
```

Post/Pre Operators (2)

```
class Example1A
{
   public static void main (String [] args)
   {
      int num1, num2;
      num1 = 5;
      num2 = ++num1 * num1++;
      System.out.println("num1=" + num1);
      System.out.println("num2=" + num2);
   }
}
```


Unary, Complement And Casting Operators

```
class Example2
{
   public static void main (String [] args)
   {
      int num = 5;
      float fl;
      System.out.println(num);
      num = num * -num;
      System.out.println(num);
      num = ~num;
      System.out.println(num);
      fl = num;
      System.out.println(num + " " + fl);
      num = (int) fl;
      System.out.println(num + " " + fl);
   }
}
```

Bit Shifting And Bitwise Operators

```
class Example3
{
   public static void main (String [] args)
   {
      int num1, num2;
      num1 = 4;
      num2 = 3;
      System.out.println(num1 + " " + num2);
      num1 <<= num2;
      System.out.println(num1 + " " + num2);
      num1 =  8;
      num1 = num1 >> num2;
```


Bit Shifting And Bitwise Operators (2)

```
      System.out.println(num1 + " " + num2);
      num1 = 5;
      num1 = num1 & num2;
      System.out.println(num1 + " " + num2);
      num1 = 4;
      num1 |= num2;
      System.out.println(num1 + " " + num2);
   }
}
```

Advanced Output (*Optional*)

You can employ the predefined code in TIO (http://www.cse.ucsc.edu/~charlie/java/tio/)

To use:

(In Unix):

- Create link from the directory where your Java code resides to the following directory /home/profs/tamj/tio
- Do this by typing the following in that directory:
  ln -s /home/profs/tamj/tio tio

(At the start of the Java program include the following statement):

- import tio.*;

Advanced Output (2)

Statement	Effect
Console.out.printf(<*variable or string1* > + <*variable or string 2*> …); ***MUST EVENTUALLY BE FOLLOWED BY A PRINTFLN!***	Prints contents of field
Console.out.printfln((<*variable or string1* > + <*variable or string 2*> …);	Prints contents of field and a new line
Console.out.setWidth(<*integer value*>);	Sets the width of a field
Console.out.setDigits(<*integer value*>);	Sets the number of places of precision
Console.out.setJustify(Console.out.LEFT); Console.out.setJustify(Console.out.RIGHT);	Left or right justify field

Advanced Output: An Example

```
import tio.*;
class Output1
{
   public static void main (String [] args)
   {
      int num = 123;
      double db = 123.45;
      Console.out.setJustify(Console.out.LEFT);
      Console.out.setWidth(6);
      Console.out.setDigits(1);
      Console.out.printf("Start line");
      Console.out.printf(num);
      Console.out.printf(db);
      Console.out.printf("End of line");
      Console.out.printfln("");
   }
}
```


Text-Based Java Input

You can employ the predefined code in TIO (http://www.cse.ucsc.edu/~charlie/java/tio/)

To use:

(In Unix):

- Create link from the directory where your Java code resides to the following directory /home/profs/tamj/tio
- Do this by typing the following in that directory:
  ln -s /home/profs/tamj/tio tio

(At the start of the Java program include the following statement):

- import tio.*;

Text-Based Java Input (2)

1	Console.in.readChar()	Reads in a character Returns an integer
2	Console.in.readInt()	Reads some characters Returns an integer
3	Console.in.readLong()	Reads some characters Returns a long
4	Console.in.readFloat()	Reads some characters Returns a float
5	Console.in.readDouble()	Reads some characters Returns a double

Text-Based Java Input (3)

6	Console.in.readWord()	Reads in a word Returns a String
7	Console.in.readLine()	Reads in a line Returns a String

Text-Based Java Input (4)

Caution! The input routines (2 – 6) accept a series of characters that end with white space but *the white space is still left* on the input stream. Leading white space is removed.

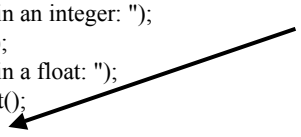
Work-around: Follow each of these input statements with a readLine() as needed.

Text-Based Java Input: An Example

```
import tio.*;
class Input1
{
   public static void main (String [] args)
   {
      int in;
      float fl;
      String st;

      System.out.print("Type in an integer: ");
      in = Console.in.readInt();
      System.out.print("Type in a float: ");
      fl = Console.in.readFloat();

      System.out.print("Type in a sentence: ");
      st = Console.in.readLine();
}
```

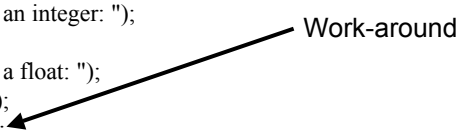


Text-Based Java Input: An Example

```
import tio.*;
class Input1
{
   public static void main (String [] args)
   {
      int in;
      float fl;
      String st;

      System.out.print("Type in an integer: ");
      in = Console.in.readInt();
      System.out.print("Type in a float: ");
      fl = Console.in.readFloat();
      st = Console.in.readLine();

      System.out.print("Type in a sentence: ");
      st = Console.in.readLine();
}
```



Decision Making

Pascal

- If-then
- If-then, else
- If-then, else-if
- Case-of

Java

- If
- If, else
- If, else-if
- Switch

Decision Making: If

Format:

```
if (Boolean Expression)
    Body
```

Example:

```
if (x != y)
    System.out.println("X and Y are not equal");

if ((x > 0) && (y > 0))
{
    System.out.println();
     System.out.println("X and Y are positive");
}
```


Decision Making: If, Else

Format:

```
if (Boolean expression)
    Body of if
else
    Body of else
```

Example:

```
if (x < 0)
    System.out.println("X is negative");
else
    System.out.println("X is non-negative");
```

If, Else-If

Format:

```
if (Boolean expression)
    Body of if
else if (Boolean expression)
    Body of first else-if
    :    :          :
else if (Boolean expression)
    Body of last else-if
else
    Body of else
```


If, Else-If (2)

Example:

```
if (gpa == 4)
 {
    System.out.println("A");
 }
 else if (gpa == 3)
 {
    System.out.println("B");
 }
 else if (gpa == 2)
 {
    System.out.println("C");
 }
```

If, Else-If (2)

```
else if (gpa == 1)
{
   System.out.println("D");
 }
 else
 {
   System.out.println("Invalid gpa");
 }
```


Alternative To Multiple Else-If's: Switch

Format:

```
switch (variable name)
{
  case <integer value>:
  Body
  break;

  case <integer value>:
  Body
  break;
      :
  default:
  Body

}
```

1 The type of variable can be a byte, char, short, int or long

Alternative To Multiple Else-If's: Switch (2)

Format:

```
switch (variable name)
{
   case '<character value>':
   Body
   break;

   case '<character value>':
   Body
   break;
       :
   default:
   Body

}
```

1 The type of variable can be a byte, char, short, int or long

Loops

Pascal Pre-test loops

- For-do
- While-do

Java Pre-test loops

- For
- While

Pascal Post-test loops

- Repeat-until

Java Post-test loops

- Do-while

While Loops

Format:

While (*Expression*)
  Body

Example:

```
int i = 1;
while (i <= 1000000)
{
    System.out.println("How much do I love thee?");
    System.out.println("Let me count the ways: ", + i);
    i = i + 1;
}
```


For Loops

Format:

for (initialization; Boolean expression; update control)
  Body

Example:

```
for (i = 1; i <= 1000000; i++)
{
    System.out.println("How much do I love thee?");
    System.out.println("Let me count the ways: ", + i);
}
```

Do-While Loops

Format:

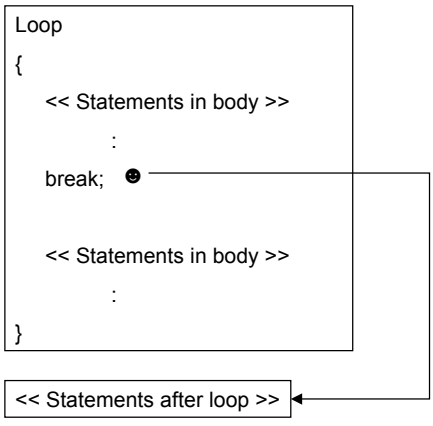
```
do
    Body
while (Boolean expression);
```

Example:

```
char ch = 'A';
do
{
    System.out.println(ch);
    ch++;
}
while (ch != 'K');
```


Ending Loops Early: Break

When this statement is reached the loop ends. (You "break out of" the loop).



Ending Loops Early: An Example

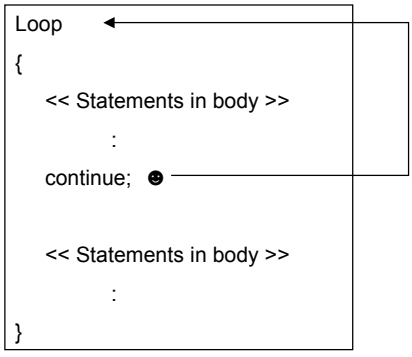
```
import tio.*;
class BreakExample
{
   public static void main (String [] args)
   {
      int number, sum;
      sum = 0;
```


Ending Loops Early: An Example (2)

```
      while (true)
      {
         System.out.print("\tPositive number (negative to quit): ");
         number = Console.in.readInt();
         Console.in.readLine();
         if (number >= 0)
            sum += number;
         else
            break;
      }
      System.out.println("Sum is..." + sum);
   }
}
```

Skipping An Iteration Of A Loop: Continue

When this statement is reached control returns to the beginning of the loop. (You swing back up to the top of the loop).



Skipping An Iteration of A Loop: Continue

```
for (i = 1; i <= 10; i++)
{
  if (i % 2 == 0)
  {
      continue;
  }
  System.out.println("i=" + i);
}
```

Some Useful Java Libraries[1]

Library	Purpose
java.lang	The core part of the Java language e.g., Math functions
java.util	Extra utilities e.g., Random number generators, automatically resizable arrays
java.io	Input and output
java.awt	The original library for developing GUI's (graphical user interfaces)
:	: :

1) Note: The use of the code in any of these libraries (except java.lang) requires the use of an import statement.

You Should Now Know

- How Java was developed and the impact of it's web-based roots on the development of this language
- The basic structure required in creating a simple Java program as well as how to compile and run programs
- How to perform text based input and output in Java
- Methods of documenting a Java program
- The declaration of constants and variables
- What are the common Java operators and how they work
- The structure and syntax of decision making and looping constructs