Computer Science 120 Java

[Writing a Program in Java 3](#_Toc296082087)

[Java Code 8](#_Toc296082088)

[Example Program 8](#_Toc296082089)

[Input and Output Commands 9](#_Toc296082090)

[Example Program 9](#_Toc296082091)

[Formatting Output 9](#_Toc296082092)

[Example Program 11](#_Toc296082093)

[Example Program 11](#_Toc296082094)

[Java Packages 12](#_Toc296082095)

[Input Commands 12](#_Toc296082096)

[Example Program 13](#_Toc296082097)

[Variables and Constants 13](#_Toc296082098)

[Example Program 14](#_Toc296082099)

[Operators 14](#_Toc296082100)

[Casting Data Types 16](#_Toc296082101)

[Mathematical Functions 17](#_Toc296082102)

[Example Program 18](#_Toc296082103)

[Loops 18](#_Toc296082104)

[Example Program 21](#_Toc296082105)

[Example Program 22](#_Toc296082106)

[Decision Structures 22](#_Toc296082107)

[Example Program 24](#_Toc296082108)

[Modular Division 24](#_Toc296082109)

[Testing For Primes 25](#_Toc296082110)

[Separating Digits 25](#_Toc296082111)

[Generating Random Numbers 25](#_Toc296082112)

[Arrays 26](#_Toc296082113)

[Declaring an Array 27](#_Toc296082114)

[Putting Information Into An Array 27](#_Toc296082115)

[Best Practices: Using a Constant for the Array Size 28](#_Toc296082116)

[Example Program 29](#_Toc296082117)

[Two Dimensional Arrays 29](#_Toc296082118)

[Putting Information Into A Two-Dimensional Array 31](#_Toc296082119)

[Modular Programming Using Methods 32](#_Toc296082120)

[Example Program Using Methods 93 34](#_Toc296082121)

[Sorting 35](#_Toc296082122)

[Characters and Strings 36](#_Toc296082123)

[Example Program 38](#_Toc296082124)

[Defining a class 38](#_Toc296082125)

[Using a class 39](#_Toc296082126)

[Creating an array of variables that use the class 39](#_Toc296082127)

[Graphical User Interface 39](#_Toc296082128)

[Frames and Panels 39](#_Toc296082129)

[Displaying a dialog box 39](#_Toc296082130)

[Creating Simple Drawings 40](#_Toc296082131)

[Graphics 41](#_Toc296082132)

[Color 42](#_Toc296082133)

[Using Variables to Change Drawings 43](#_Toc296082134)

[Responding to Mouse Events 44](#_Toc296082135)

[Movement With Arrow Keys 45](#_Toc296082136)

[Continual Movement 47](#_Toc296082137)

[Loading Images 48](#_Toc296082138)

[File Management 50](#_Toc296082139)

[Creating a new file 50](#_Toc296082140)

[Example Program 50](#_Toc296082141)

[Handling Exceptions 50](#_Toc296082142)

[Example Program 50](#_Toc296082143)

[Reading the contents of a text file 51](#_Toc296082144)

[Example Program 51](#_Toc296082145)

[Reading the contents of a file containing numbers 52](#_Toc296082146)

[Example Program 52](#_Toc296082147)

[Writing data to a file 53](#_Toc296082148)

[Example Program 54](#_Toc296082149)

[Appendix A 55](#_Toc296082150)

[ASCII Character Codes 55](#_Toc296082151)

[Appendix B 56](#_Toc296082152)

[Writing a Program Using Notepad 56](#_Toc296082153)

[Compiling and Running a Program in Java 57](#_Toc296082154)

[Appendix C 57](#_Toc296082155)

[Mazes 57](#_Toc296082156)

[Assignments 61](#_Toc296082157)

[Assignment #1 (simple output) 63](#_Toc296082158)

[Assignment #2 (output and formatting) 63](#_Toc296082159)

[Assignment #3 (Input, Math operators and variables) 64](#_Toc296082160)

[Assignment #4 (Loops) 65](#_Toc296082161)

[Assignment #5 (Decision Structures) 65](#_Toc296082162)

[Assignment #6 (Modular Division) 66](#_Toc296082163)

[Assignment #7 (One Dimensional Arrays) 67](#_Toc296082164)

[Assignment #8 (Two Dimensional Arrays) 68](#_Toc296082165)

[Assignment #9 (Methods) 69](#_Toc296082166)

[Project #1 70](#_Toc296082167)

[Assignment #10 (Sorting) 71](#_Toc296082168)

[Assignment #11 (Strings) 72](#_Toc296082169)

[Assignment #12 (Classes) 73](#_Toc296082170)

[Assignment #13 (Graphics) 73](#_Toc296082171)

[Assignment #14 (Files) 74](#_Toc296082172)

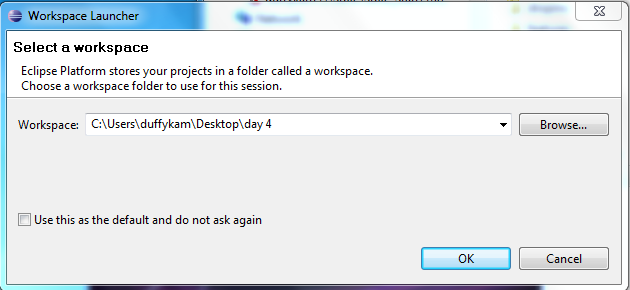
[Final Project 74](#_Toc296082173)

# Writing a Program in Java

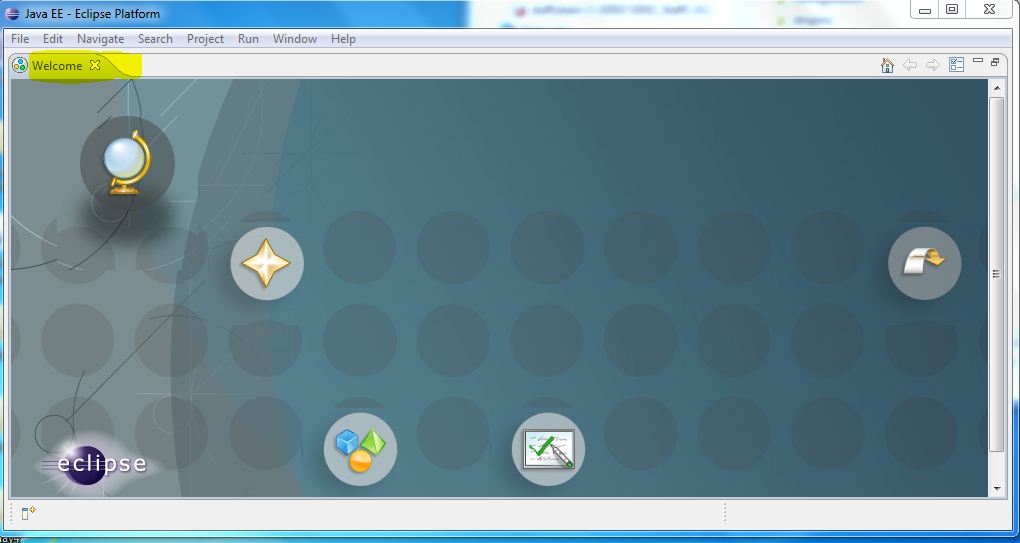


Use the editor Eclipse to create Java text files.

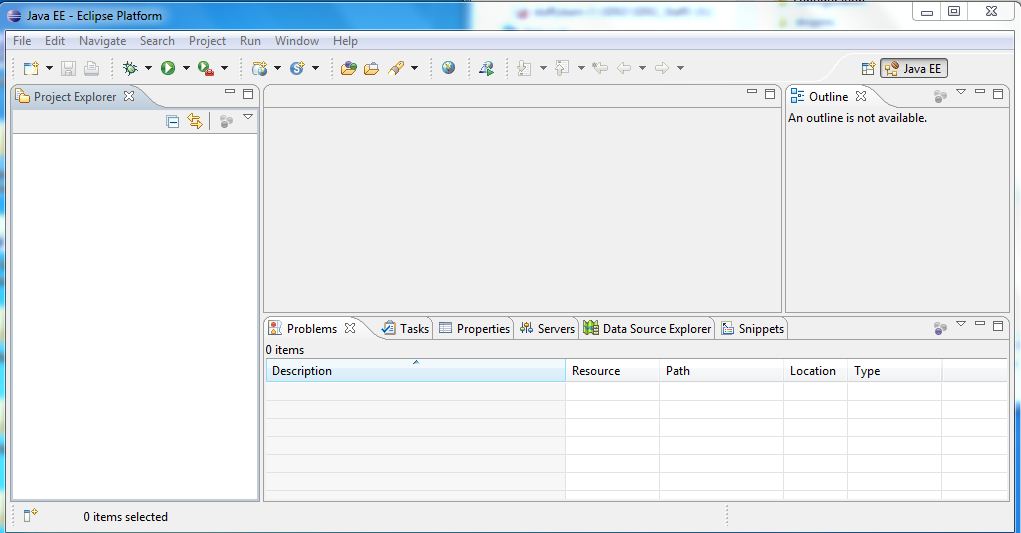
1. Open eclipse :
2. Enter a workspace name for your project. You might call your workspace: Project1. Make sure the workspace is saved on your drive. Click OK.



1. You will see the welcome screen for eclipse. Click the X to close the welcome screen.



1. You will now see the Eclipse editor.

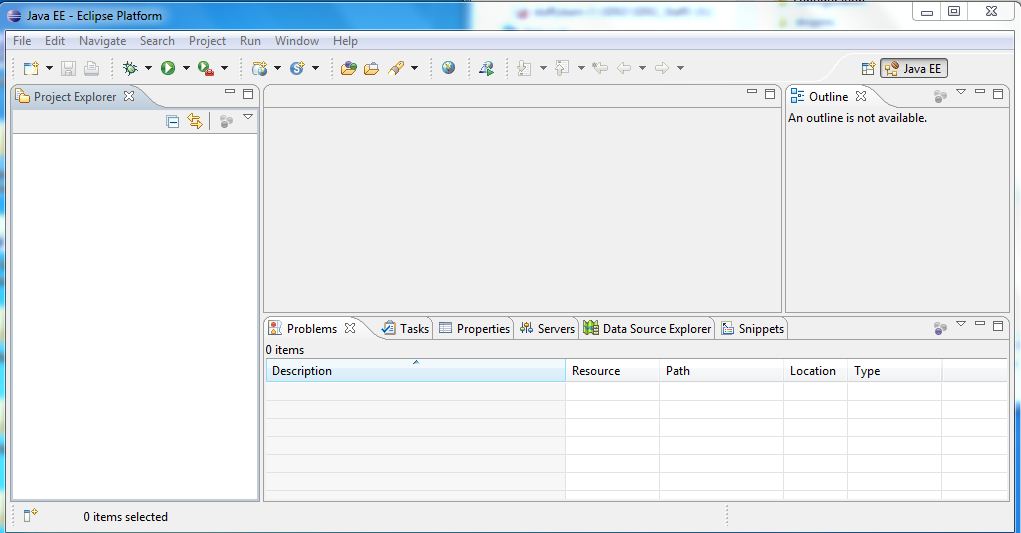


Go to the right hand side of your screen and change the environment

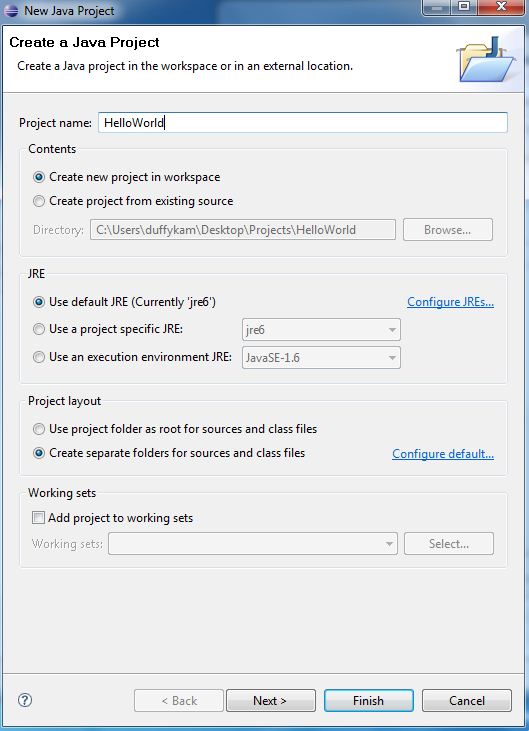
3.jpg

to Java.

Click the box to see your options



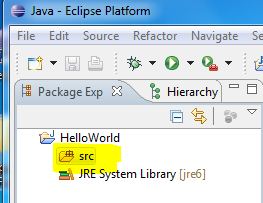
1. Click **File->New->Java Project** to create a new project.



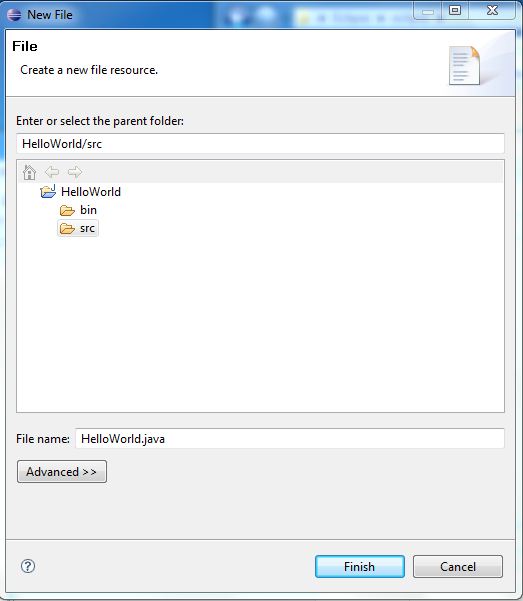
Type the name of the project

Click Finish to go on to the next step

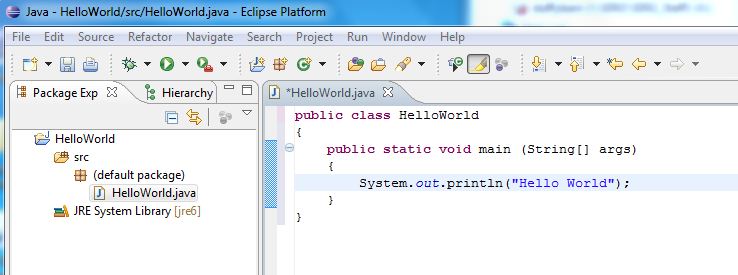
1. Open your java program by clicking the arrow to the left of your program name. This will list the files in your program. Right click on the **src** folder and start a NEW->FILE



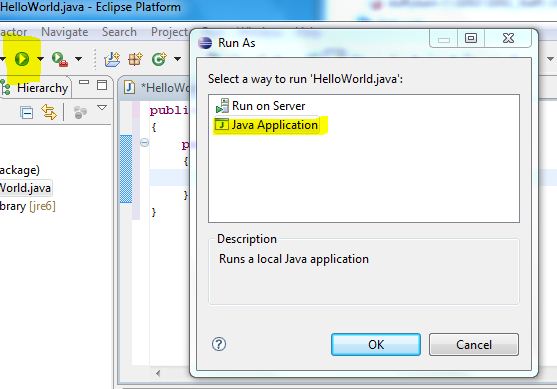
1. Type in the name of the new file. It will be the same name as the java file and will end in **.java** Click finish to end this step.



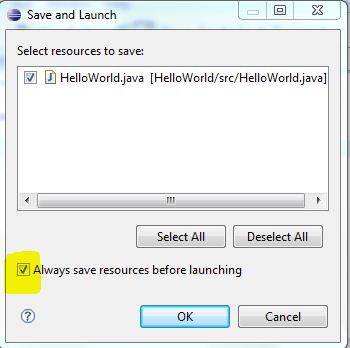
1. You can now type in your program. The upper and lower case letters and punctuation must be correct, while spaces and indentation are decided by the user.



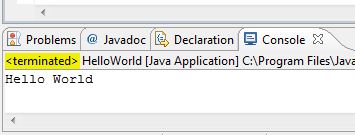
1. Run the program and fix any errors. Click on the Play button to run the program. Run the program as a Java application. Press the OK button to continue.



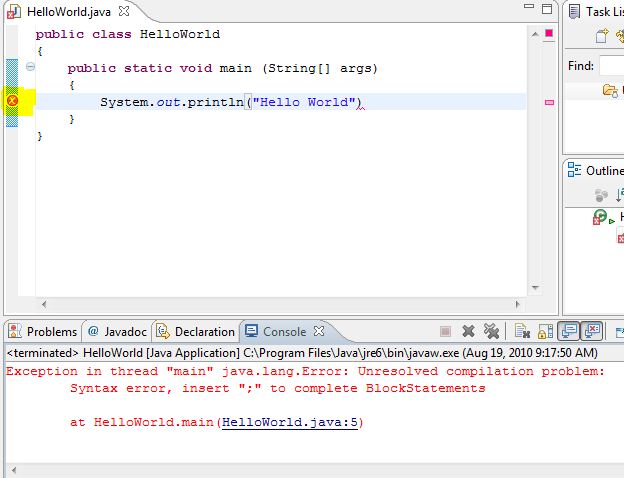
1. The first time you run a program, the editor will ask you if you want to save the resource before it runs the program. If you check the indicated box your program will always be saved before you run the program.



1. The program will run and the output will be displayed at the bottom of your screen. When the program is finished, the message **<terminated>** will appear.



1. Correct any errors that appear in your program. The compiler will give you suggestions as to how to fix your program and will indicate where the error has occurred. If you place your mouse over the X more details regarding the error will appear.



# Java Code

In general, Java code is structured with a class containing functions, made up of statements. Every Java program must have a function named **main**. At run time, program execution starts at the first line of the main function and then continues in order, between the curly braces { }. The Java code is entered in the main function between the braces.

### 

***Example:***

class public Greeting

{

public static void main ( String[] args)

{

*// enter code here*

System.out.println(“Hello World”);

}

}

Greeting is the name of the class.

The main function will always contain the arguments: **public**, **static** and **void** and the bracket will always contain **String[ ] args**.

This main function consists of a single line or statement of code.

Braces In Java, the braces { } mark the beginning and ending of a block of code. In this case, the block of code belonging to the main function, and the block of code belonging to the class Greeting. You will notice that the program is indented so that it easier to see that the braces match up.

Semicolons Semicolons are used to indicate the end of a line of code. Almost all lines of code end with a semicolon. Remember that semicolons mark the end of a line and not the return key.

Example Program

***Example1: Example2:***

public static void main ( String[] args) public static void main ( String[] args)

{ {

int a; System.out.println(“Total Wages :”

a = 2 ; + (HoursWorked \* HourlyWage);

System.out.println(“Your number is ” + a ) ;

} }

\*WARNING : You cannot split a statement between the quotation marks

Comments Comments are used to make a program easier for the programmer to follow. The compiler will ignore any statements that appear after a comment. Comments can be written into the program in two ways:

**//** These are used to indicate that any statement that follows is a comment. It can be used as a line on its own or after a statement.

**/\* \*/** Everything between these symbols is a comment.

***Example1: Example2:***

**//**  This is a comment **/\*** this is a comment

that takes up more than one line **\*/**

int n1 = 10 ; **//** Local variable n1

# Input and Output Commands

To display data to the computer screen use :

1/ **System.out.print (“** *message to be printed* **”);** //stay on the same line after printing

2/ **System.out.println (“** *message to be printed* **”);** //start a new line after printing

The format for these commands is as follows:

Example Program

|  |  |
| --- | --- |
|  |  |
| **Syntax:** | **Example:** |
| public static void main ( String[] args)  {  System.out.println(“any message” + any variable);  } | public static void main ( String[] args)  { int num1=10, num2 =5,sum;  System.out.println(“Hello”);  System.out.println(“Your first number is :” + num1);  System.out.println(“Your second number is :” + num2);  sum = num1+num2;  System.out.println(“The sum of ” + num1+” and ”+  num2+” is ” + sum);  } |
|  |  |

# Formatting Output

The printf() method can be used in place of the print() or println() methods to control the way that output is displayed.

The format command has the following syntax :

**System.out.printf ("***format1 format2 …. ", "Output String1", variable,…..****);***

The specific format is of the form : **% specifier**

**%** :indicates the start of a specifier

|  |  |  |  |
| --- | --- | --- | --- |
| Specifier | Type of variable | Example | Output |
| d | integer | int number=5;  System.out.printf(“The number is %d”, number); | The number is 5 |
| e or E | Decimal in exponential notation | double number=5.123;  System.out.printf(“The number is %e”, number);  double number=123.456;  System.out.printf(“The number is %E”, number); | The number is 5.123000e+00  The number is 1.23456E+02 |
| f | Decimal in usual decimal notation | double number=123.456;  System.out.printf(“The number is %f”, number); | The number is 123.456000 |
| c | characters | char letter=’a’;  System.out.printf(“The letter is %c”, letter); | The letter is a |
| s or S | strings | String message=”Hello”;  System.out.printf(“Here is my message : %s”, message);  String message=”Hello ”;  System.out.printf(“Here is my message : %S”, message); | Here is my message : Hello  Here is my message: HELLO |
| b | boolean | boolean maybe=true;  System.out.printf("My boolean: %b", maybe); | My boolean: true |
| Mixed | Combination of variable types | String message="The numbers are :";  int number1=1;  int number2=5;  System.out.printf(" %s %d and %d", message, number1, number2); | The numbers are : 1 and 5 |
| \n | Move to the next line | String message="The numbers are :";  int number1=1;  int number2=5;  System.out.printf(" %s %d and \n %d", message, number1, number2); | The numbers are : 1 and  5 |
| \t | Insert a tab | int number1=1;  int number2=5;  System.out.printf(" numbers : \t %d and \t %d", message, number1, number2); | numbers: 1 and 5 |

* There are many more specifiers for times and dates. Check online or in a manual for this information.

The specific format can be extended to allow for left or right alignment and specific spacing:

**% alignment width specifier**

alignment : **-** for left alignment

(nothing) for right alignment

width : number of characters to use for output. If the width is greater than the number of characters then

spaces are used to pad the output. This will help line up items in a column. With decimals , you can

also specify how many numbers will appear after the decimal.

|  |  |
| --- | --- |
| Example | Output |
| int number1=1;  int number2=5;  System.out.printf("%14s %13s","First Number","Second Number\n");  System.out.printf("%14d %13d", number1, number2); | First Number Second Number  1 5 |
| int number1=1;  int number2=5;  // change the alignment to left for number 1  System.out.printf("%-14s %13s","First Number","Second Number\n");  System.out.printf("%-14d %13d", number1, number2); | First Number Second Number  1 5 |
| double number1=1.23456;  double number2=0.0012345;  System.out.printf("%-14s %13s","First Number","Second Number\n");  System.out.printf("%-14f %13f", number1, number2); | First Number Second Number  1.234560 0.001235 |
| double number1=1.23456;  double number2=0.0012345;    System.out.printf("%-14s %13s","First Number","Second Number\n");  System.out.printf("%-14.2f %13.7f", number1, number2);  //print 2 digits after the decimal and 7 digits after the decimal | First Number Second Number  1.23 0.0012345 |
| **double** num1=10, num2 =7;  **double** quotient=num1/num2;    System.*out*.printf("The number is %1.2f \n",quotient); // print 2 numbers after  System.*out*.printf("The number is %1.3f \n",quotient); // print 3 numbers after  System.*out*.printf("The number is %1.4f \n",quotient); // print 4 numbers after  System.*out*.printf("The number is %1.5f \n",quotient); // print 5 numbers after | The number is 1.43  The number is 1.429  The number is 1.4286  The number is 1.42857 |

Example Program

|  |  |
| --- | --- |
| **Example:** | **Output:** |
|  |  |
| System.out.printf("%-10s %-5s %-6s %8s", "Team","Wins", Draws", "Losses\n");  System.out.printf("%-10s %-5s %-6s %8s", "USA", "1","2", "0\n");  System.out.printf("%-10s %-5s %-6s %8s", "England", "1","2", "0\n");  System.out.printf("%-10s %-5s %-6s %8s", "Slovenia", "1","1", "1\n");  System.out.printf("%-10s %-5s %-6s %8s", "Algeria", "0","1", "2\n");  //The teams, wins, and draws are left aligned while the losses line  // up on the right. The team name is allowed 10 spaces , the wins have // 5 spaces, the losses 6 and the draws and losses occupy 8 spaces.  // \n is used to move to the next line |  |
| **Example:** | **Output:** |
| System.out.printf("%-10s %8s %8s %8s", "Team", "Wins","Draws", "Losses\n");  System.out.printf("%-10s %8s %8s %8s", "USA", "1","2", "0\n");  System.out.printf("%-10s %8s %8s %8s", "England", "1","2", "0\n");  System.out.printf("%-10s %8s %8s %8s", "Slovenia", "1","1", "1\n");  System.out.printf("%-10s %8s %8s %8s", "Algeria", "0","1", "2\n");  //The teams are left aligned while the wins, draws and losses all line up // on the right. The team name is allowed 10 spaces and the wins, draws // and losses occupy 8 spaces. |  |
|  |  |

The NumberFormat class and the DecimalFormat class are used to format numbers. They are defined in the java.text package. You must import this package in order to use the formatting classes.

Example Program

|  |  |
| --- | --- |
| **Example:** | **Output:** |
| import java.text.NumberFormat;  class Test  {  public static void main ( String[] args)  {  double dollars =123.99;  int num1 = 9753;  double decimalNumber=2.0/3.0;  double ratio = 0.75;  NumberFormat money= NumberFormat.getCurrencyInstance();  System.out.println(money.format(dollars));    NumberFormat number= NumberFormat.getIntegerInstance();  System.out.println(number.format(num1));    NumberFormat decimal= NumberFormat.getNumberInstance();  System.out.println(decimal.format(decimalNumber));    NumberFormat percent= NumberFormat.getPercentInstance();  System.out.println(percent.format(ratio));  }  } | //Declare the format for money and then use it //to print the variable dollar  //Declare the format for integers and then use //it to print the variable num1  //Declare the format for decimals and then use //it to print the variable decimalNumber  //Declare the format for percentages and then //use it to print the variable ratio |
| **Example:** | **Output:** |
| import java.util.\*;  import java.text.DecimalFormat;  class Test  {  public static void main ( String[] args)  {  Scanner input = new Scanner(System.in);  int radius;  double area, circumference;  System.out.println("Enter the radius:");  radius=input.nextInt();  area = Math.PI\*Math.pow(radius,2);  circumference = 2\*Math.PI\*radius;    DecimalFormat fmt = new DecimalFormat ("0.###");  System.out.println("The circle's area :"+ fmt.format(area));  DecimalFormat fmt2 = new DecimalFormat ("0.##");  System.out.println("The circle's circumference:"+fmt2.format(circumference));  }  } | // create a format of 3 numbers after the // decimal  // create a 2nd format of 2 numbers after // the decimal |
|  |  |

# Java Packages

Java has a large number of packages as part of the Java Runtime Environment. These packages contain already written blocks of code. The statement **import java.packageName.Classname** allows the user to use a specific class defined in the package. This saves the programmer from having to write code that already exists for common functions. Java applications automatically import the entire java.lang package.

***Example:*** import java.util.Scanner; // makes a single class available from the package

import java.util.\* ; // makes all member classes available from the package

The import statements are the first lines in the program.

To access the Java packages use the following link : http://java.sun.com/j2se/1.4.1/docs/api/

# Input Commands

To read data from the keyboard use :

1/ nextInt(); // read an integer variable

2/ nextDouble(); // read a double variable

3/ next(); // read a string variable

4/ nextLine(); // read a line up to the end of line character

5/ nextBoolean(); // read a boolean variable

The code for these function is in a Java package java.util which must be imported before the functions are available for use. A Scanner object must also be initialized before using any of the standard input.

Use the code : **Scanner *input* = new Scanner (System.in);** to initialize the Scanner object.

The format for these commands is demonstrated in the examples that follow.

Example Program

|  |  |
| --- | --- |
| **Syntax:** | **Example:** |
| import java.util.\*;  class Test  { public static void main ( String[] args)  {int IntegerVariable;  double DoubleVariable;    Scanner input = new Scanner (System.in);  System.out.println(“Type in an integer”);  IntegerVariable=input.nextInt();  System.out.println(“Type in a double”);  DoubleVariable=input.nextDouble();  }  } | import java.util.\*;  class Test  { public static void main ( String[] args)  {  Scanner input = new Scanner (System.in);  int num1, num2,sum;  System.out.println(“Please enter you first number”);  num1=input.nextInt();  System.out.println(“Please enter you second number:”);  num2=input.nextInt();  sum = num1+num2;  System.out.println(“The sum of ” + num1+” and ”+  num2+” is ” + sum);  }  } |

# Variables and Constants

Variables are used to store information that can be changed. You **must declare** a variable before it can be used. Variables must start with a letter. The rest of the variable name can be letters, numbers or the underscore symbol.

**Warning: Java** is case sensitive: **Num1** is different from **num1** and from **NUM1**

The most common Java data types (variables) are integers, decimals, Boolean and characters.

**Simple Java Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Example Declarations** | **Description** |
| *int*  *long*  *short* | int factorial;  long factorial;  short grade, age;; | Integer data types. In Java:   * int and long are 4 bytes (–2,147,483,648 to 2,147,483,647 * short is 2 bytes (–32,768 to 32,767) |
| *float*  *double*  *long double* | float average;  double span;  long  double carbondate | Floating point (decimal) numbers. These floating point variations allow for different precisions (numbers of significant digits after the decimal point.) In Java:   * float is 4 bytes (decimal numbers from 3.4 x 10-38 to 3.4 x 1038) * double and long double are 8 bytes  (decimal numbers fr0m -1.7 x 10308 to 1.7 x 10308) |
| *char* | char grade; | Technically a one byte integer type., char holds character data. The contents of a char variable is a number ASCII code for the character it represents. Our alpha and numeric character sets are part of the ASCII codes. |
| *Strings* | String sentence; | A string is a sequence of characters. Ex. Sentence = “Hello there”; |
| *bool* | bool ready; | Also an integer but acts as a Boolean value. Possible values are:   * true (a or a non-zero integer) * false (or zero)   ***Example:*** ready = true; |

**Variable Assignments**

Variables can be assigned a value either when they are declared or later in the program:

***Example:*** double radius; **// declare *radius***

radius = 4.0097; **// assign 4.0097 to the variable radius**

… or equivalently:

***Example:*** double radius = 4.0097; **// declare *radius* and initialize at the same time**

char TopGrade=’A’; **// declare *TopGrade* and initialize**

**Constants**

Constants are used to store information that does not change. You define the constants in the main function and then you can refer to them throughout the program. They are useful when you are using the same information again and again in the program.

Example Program

class Contants

{

public static void main ( String[] args)

{

final double Pi = 3.14159;

double radius=10 , area ;

area=Pi \* radius \* radius ;

System.out.println(“The radius of the circle is : ” + area);

}

}

# Operators

**Mathematical Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Function** | **Example**  (of using the operator in an assignment statement) | **Resulting value of the variable** |
| **+** | Addition | sum = 2 + 3; | 5 |
| **-** | Subtraction | net = 5 – 2; | 3 |
| **\*** | Multiplication | wage = 40 \* 15.00; | 600 |
| **/** | Division | velocity = 220/2; | 110 |
| **%** | Modular division  (modular division gives the remainder after dividing the two numbers) | mod\_quotient = 5 % 2; | 1 |

**Assignment Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Function** | **Example**  (these examples trace the value of the variable num) | **Resulting value of the variable "num"** |
| **=** | assignment | num = 32; | **num**  is assigned the value of 32 |
| **+ =** | add, then assign | num **+ =** 1; | 1 is added to num, **num** is now 33 |
| **- =** | subtract, then assign | num **- =** 3; | 3 is added to num, **num** is now 30 |
| **\* =** | multiply, then assign | num **\* =** 2; | num is multiplied by 2, **num** is now 60 |
| **/ =** | divide, then assign | num **/ =** 2; | num is divided by 2, **num** is now 60 |

Order of Operations

As in Math, braces ( ) can affect the order of operations.

|  |  |
| --- | --- |
| ***Example:*** | **Resulting value of the variable num** |
| num = 3 + 5 \* 2 | 13 |
| num = (3 + 5) \* 2 | 16 |

**Relational and Logical Operations**

These operations are used in conditional expressions when testing **while** loops and **if**  statements.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Function** | **Example** |
| **<** | less than | if (count **<** 100) |
| **>** | greater than | while (grade **>** 60) |
| **< =** | less or equal to | if (age **< =** 65) |
| **> =** | greater or equal to | bool pass;  int grade;  (pass = (grade **>=** 60)); |
| **= =** | equal to | while (score == 0) |
| **! =** | not equal to | if (score != 0) |
| **& &** | and | if (score == 0 && round > 1) |
| | | | or | if ((time **<** 20) || | (points **>** 23)) |

**Note:**

* All conditional statements must be enclosed in braces **( )** as they are in the above examples. Additional braces can be added for clarity (last example) or to change the order of operations (like in math)
* A common mistake when programming in Java is using the assignment operator "**=**" when you really mean to be using the comparison operator "**=** **=**" (*equal to)*. Later on when you are programming **if** statements and **while** loops, watch for this. The compiler will not give you an error if you mistakenly use the assignment operator in a conditional expression, but weird stuff will happen in you program.
* The greater than or equal to ">=", less than or equal to "<=" operators must have the greater than or the less than sign first. For example, Java does not recognize "=<" as a valid conditional operator.

**Increment and Decrement Operators**

These unusual operators are used in the Java **for…Loop** (but also can be used in other Java statements.)

1. Increment and decrement operators either increase a variable by one or decrease a variable by one.
2. There are two varieties of the increment and decrement operator:
   * prefix

the increment or decrement operator is placed before the variable

* + postfix

the increment or decrement operator is placed after the operator

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Action** | **Shorthand For:** |
| **i++** | **i**  is evaluated and then incremented by 1 | i = i + 1 |
| **i -- --** | **i**  is evaluated and then decreased by 1 | i = i − 1 |
| **++i** | **i**  is incremented by 1 and then evaluated. |  |
| **- - i** | **i**  is decreased by 1 and then evaluated. |  |

***Note:*** In the examples, the variable used is "**i**". The letters **i** and **j** are commonly used in many programming languages as counters in **for** loops.

The Increment and decrement operators **apply to one variable**; they cannot be applied to an expression. Therefore the following are illegal statements in Java :

|  |  |
| --- | --- |
| **Illegal Statements** | **Reason** |
| (age \* 3)++ | (age \* 3) is not a variable |
| - - (TopScore + Points) | (TopScore – Points) is not a variable |
| 16 + + | "16" is not a variable |

**More About Postfix Versus Prefix Increment/Decrement Operators**

|  |  |
| --- | --- |
| **Example** | **Output** |
| int CustomerNum = 10; |  |
| System.out.println("Now serving customer #" + (CustomerNum++));  System.out.println("Next customer is #" + CustomerNum); | Now serving customer #10  Next customer is #11 |
| System.out.println("Now serving customer #" +( ++CustomerNum));  System.out.println("Are you customer #" + CustomerNum); | Now serving customer #11  Are you customer ##11 |

# Casting Data Types

In QBasic, assigning the result of a division to a variable that is type integer always results in an integer result even if the result was a real number. In such cases you had to be very careful to choose a real number variable for the result

***QBasic Example:*** num1% = 10

answer1% : 3

answer2! : 3.3333

num2% = 3

answer1% = num1 / num2

answer2! = num1 / num2

In Java, you not only must ensure that the variable that will hold the result is of type real (or floating point), you also have to either:

1. make the operands (ie: num1 and num2 in the above example.) floating point variable types too
2. cast the type float on the integer operands

Sometimes it is very important that your variable is an integer even if it is used in an operation (like division) that could result in a floating point number. The casting operation (#2 above) is then used.

Tocast is to *temporarily* allow these variables to be floating point numbers

Please see the example program and output on the next page:

import java.util.\*;

class Test

{ public static void main ( String[] args)

{ int num1, num2, answer1;

float answer2;

num1 = 8;

num2 = 5;

answer1 = num1 / num2;

System.out.println("8 divided by 5 is "+answer1 + " is not correct");

/\* the computer now tries again \*/

answer2 = num1 / num2; //it is not enough to have the answer as a float

System.out.println("8 divided by 5 is "+answer2+ " is not correct");

/\* the computer now tries casting the variables on the right \*/

answer2 = (float) num1 / num2;

System.out.println(" 8 divided by 5 is "+answer2);

}

}



# Mathematical Functions

There are many available mathematical functions in the **Math class** which is included in the java.lang package.

|  |  |
| --- | --- |
| **Function** | **Example** |
| double pow(double number, double power) | square = Math.pow(5, 2.0); // calculates 5^2  cube = Math.pow(number,3.0); // calculates number^3 |
| double sqrt(double number) | root = Math.sqrt(4); // determines square root of 4  answer = Math.sqrt(discriminant); // determines square root of  // the variable "discriminant. |
| int abs(int number) | distance = Math.abs(distance); // determines absolute value |
| double sin(double angle) | ratio = Math.sin(angle); // determines the sine of the angle  // the angle is in radians |
| double cos(double angle) | x= Math.cos(angle); |
| double tan(double angle) | y= Math.tan(5); |
| double asin(double ratio) | angle= Math.asin(ratio); // determines the angle with the given  // sine ratio |
| double acos(double ratio) | A = Math.acos(0.245) ); // determines the angle with the given  // cosine ratio |
| double atan(double ratio) | C = Math.atan(ratio) ); // determines the angle with the given  // tangent ratio |
| double toRadians (double angle) | radianAngle=Math.toRadians( angle); // converts the angle  //from degrees to radians |
| double toDegrees (double angle) | angle=Math.toRadians( radianAngle); // converts the angle  //from radians to degrees |
| double log(double argument) | power = Math.log(argument); // determines the log base e |
| exp(power) | x = Math.exp(power); // determines e^power |
| PI | pi=Math.PI; // retrieves the value of pi |

Example Program

import java.util.\*;

class Test

{

public static void main ( String[] args)

{

Scanner input = new Scanner (System.in);

int num1,num2,sum,product;

double cube, squareroot;

System.out.println("Type in a number");

num1=input.nextInt();

System.out.println("Type in a number");

num2=input.nextInt();

sum=num1+num2;

System.out.println("The sum :" + sum);

product=num1\*num2;

System.out.println("The product :" + product);

cube= Math.pow(num1 , 3);

System.out.println("The cube of " + num1 + " is " + cube);

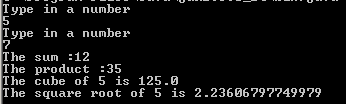
squareroot= Math.sqrt(num1);

System.out.println("The square root of "+ num1 +" is " + squareroot);

}

}

# 



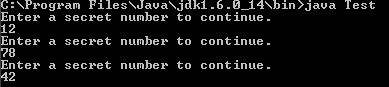
# Loops

The Java language has three looping structures:

1. **while** loop While a condition is true the steps will be repeated. The program checks at the *beginning* of the loop to see if the condition is true.
2. **do while** loop Do the steps as long as a condition is true. The program checks at the *end* of the loop to see if the condition is true.
3. **for** loop Repeat a series of steps for a given number of times.

**The While and Do While Loops**

|  |  |
| --- | --- |
|  |  |
| **Syntax:** | **Example:** |
| while **(***conditional expression)*  **{**  *statements to be repeated*  **}** | import java.util.\*;  class Test  {  public static void main ( String[] args)  {  final int SecretNum=42;  Scanner input = new Scanner (System.in);  int guess=0; // guess must be initialized as it is checked now  while (guess != SecretNum)  {  System.out.println("Enter a secret number to continue.");  guess=input.nextInt();  }  }  } |
|  |  |
| do  **{**  *statements to be repeated*  **}**  while **(***conditional expression)***;** | **Example 2:**  import java.util.\*;  class Test  {  public static void main ( String[] args)  {  final int SecretNum=42;  Scanner input = new Scanner (System.in);  int guess;    do  {  System.out.println("Enter a secret number to continue.");  guess=input.nextInt();  } while (guess != SecretNum);  }  } |



**Notes about the while and do while loops**

* In the **while loop**, there is **no semi-colon** after the braces of the while conditional expression.
* In the **do while loop**, there **is a semi-colon** after the braces of the while conditional expression
* curly braces {} are used to enclosed the statements repeated by the loop.
* The *conditional statement*  **must** be enclosed by round braces ( )
* always check that your loop does not repeat one-too-few or one-too-many times.
* The choice of which loop to use depends on when you want the condition for the loop tested: before the commands in the loop are executed, or after the loop's commands are executed the first time.
* If you know up-front how many times the loop should be repeated (say exactly 10 times), consider using a for loop instead of a while loop.

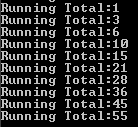
**The For Loop**

**In QBasic**, the For Loop's syntax is:

|  |  |
| --- | --- |
|  |  |
| **Syntax:** | **Example:** |
| **For** *countvariable* = *start* **TO** *endnum*  *statements to be repeated*  **NEXT** *countvariable* | sum% = 0  **For** n%= 1**TO** 100  sum% = sum% + n%  PRINT "Running Total… ", sum%  **NEXT** n% |
|  |  |

**In Java,** the **For** Loop's syntax is as follows for counting up:

|  |  |
| --- | --- |
|  |  |
| **Syntax:** |  |
| **For (** *counter = start value; counter<=end value ; update counter* **)**  **{**  *statements to be repeated*  Declares and initializes the counting variable — in this case the integer-type variable "n". Note the semi-colon.  **}** | |
| **Example:**  Repeat the loop as long as this condition is true. Note the semi-colons.  int sum = 0;  for ( int n = 1; n <= 10; n++ )  {  sum = sum + n;  System.out.println("Running Total:"+sum);  } | |
|  | |



The **For** Loop's syntax for counting down:

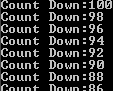
|  |  |
| --- | --- |
|  |  |
| **Syntax:** |  |
| **For (** *counter = start value; counter* ***>=****end value ; update counter* **)**  **{**  *statements to be repeated*  **}** | |
| **Example:**  for ( int n = 10; n >= 0; n-- )  {  System.out.println("The number is :"+n); //This will print the numbers from 10 down to 0  } | |

**Notes about the for loop**

* A for loop can be used to code instructions like "repeat this bunch of steps n times."
* There is no semi-colon after the braces of the for loop statement
* curly braces {} are used to enclosed the statements repeated by the loop
* always check that any variables used in the for loop are properly *declared* *and initialized* before being used.
* always check that your for loop does not repeat one-too-few or one-too-many times.
* the *update-action* portion of the for loop's statement is not limited to "++" or "−−". An update action can be another valid Java statement that updates the value of the counter variable. See the first program example below.
* At first, Java's For Loop syntax seems confusing. Java's For Loop does have its merits though. Whereas QBasic's For Loop is limited to repeated addition, Java's can be used to repeat all sorts of calculations.

Example Program

Using a Loop to count down from 100 by twos



class Test

{

public static void main ( String[ ] args)

{

int i = 0;

// decreases the counter by two each time through the loop. //

for ( i = 100; i >= 0; i -= 2 )

System.out.println("Count Down:"+i);

}

}

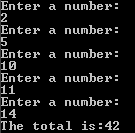
This first example does not use curly braces to enclose the body (aka the statements to be repeated) of the loop. When only one program statement is to be repeated by *any* Java structure (for loops, If…Else statements also), the curly braces are optional.

Some programmers will always use the curly braces for two reasons:

1. increase the readability of their code
2. to avoid errors (or worse yet, unintended results) if they choose later to add more statements to the loop (or whatever) and forget at *that* time to add the curly braces.

Example Program

Using a Loop to Calculating a Running Total



import java.util.\*;

class Test

{

public static void main ( String[] args)

{

//Use a loop to input 5 numbers and find the sum:

Scanner input = new Scanner (System.in);

int i; // declare counter variable, i, used in the for loop

int total = 0; // declare & initialize variables for input and running total

int number = 0;

for ( i=1; i<=5; i++) // loop for input of 5 numbers and running total

{

System.out.println("Enter a number:");

number=input.nextInt();

total += number; // "total += number" is the same as "total = total + number"

}

System.out.println("The total is: " + total );

}

}

# Decision Structures

Decision structures are used to choose between alternate actions. We would like to be able to perform certain operations in one case and other operations in other cases.

The Java language has two types of decision structures:

1. **if…else** The most simple method of choosing between options

**if…else if**

1. **switch case** The switch case decision structure is used to avoid deeply nested **if…else if** statements.

**Notes about the if, if..else and if…else if statements**

* There is no **then** part as there is in QBasic.
* The conditional expression must be enclosed in braces ()
* If there are two options, add an **else**  statement
* If there are more than two options, add **else if**  statements
* You can have as many **else if** statements as you need..

|  |  |
| --- | --- |
|  |  |
| **Syntax:** | **Example:** |
| **if (***conditional expression)*  *statement to be executed;* | **if (**point == True)  System.out.println("You scored"); |
| **if (***condition)*  **{**  *statements to be executed*  **}** | **if (**point = = True)  **{**  score += score;  System.out.println("Score:"+score);  **}** |
| **if (***condition)*  **{**  *statements to be executed*  **}**  **else**  **{**  *statements to be executed*  **}** | **if (**guess != myNum)  **{**  System.out.println("No.Guess again”);  guess=input.nextInt();  **}**  **else**  System.out.println("Genuis!!"); |
| **if (***condition)*  **{**  *statements to be executed*  **}**  **else if** (*condition*)  **{**  *statements to be executed*  **}**  **else if** (*condition*)  **{**  *statements to be executed*  **}** | **if (**grade <60)  **{**  System.out.println("Not a pass");  **}**  **else if** (grade >=50 && grade <75)  **{**  System.out.println("Pass");  **}**  **else if** (grade >=75)  **{**  System.out.println("Well done");  **}** |
|  |  |

**Notes about the switch…case statements**

* Similar to the Select…Case structure in QBasic
* Less versatile than the **if** statement
* Useful alternative to heavily nested **if** statements
* Perfect for implementing menus (especially after you know how to program methods which we do later.)

Example Program

|  |  |
| --- | --- |
| **Syntax:** | **Example:** |
| **switch (***controllingl expression)*  **{**  **case** *constant*1 **:**  steps to be completed  **break;**  **case** *constant*2 **:**step to be completed  **break;**  **.**  **.**  **.**    **case** *constant* n **:**  steps to be completed  **break;**    **default:**  steps to be completed  **break;**  **}** | import java.util.\*;  class Test  {  public static void main ( String[] args)  {  Scanner input = new Scanner (System.in);  int number;  System.out.println("Enter a number between 5 and 7");  number=input.nextInt();    switch (number)  {  case 5 :System.out.println("Too small.");  break;  case 6 :System.out.println("Well done");  break;  case 7 :System.out.println("Too large");  break;  default:  System.out.println("What?");  System.out.println("Not even close.");  break;  } // end of switch statement  }  } |

**Note:**

* The choice of which case in the **switch** statement to execute, is determined by the **controlling statement**that is in the braces after the keyword **switch.**
* The **controlling expression** must always return a value of type: bool (true or false), char, integer.
* The **constant value** is specified after each **case**
* The **break** statement appears at the end of each case.
* When the computer gets to the **break** statement, the entire **switch** statement ends.
* If you forget a break statement, the computer, after completing the steps for a **case** that is true, will continue to the next **case** statement and evaluate if that case is true. It is imaginable that this might sometimes be what you want. But be careful.
* The **default** is used if none of the choices are selected. It can be handy for catching errors and also debugging your program.
* The **default** section is optional.
* If there is no match among your cases listed, the program will do nothing and just continue on with your program.

# Modular Division

Modular division is performed using the % sign. This calculates the remainder when one number is divided by another number.

Ex. remainder = 10 % 2 the remainder will equal 0

remainder = 9 % 2 the remainder will equal 1

remainder = 10 % 4 the remainder will equal 2

### Testing For Primes

Modular division is useful for testing if a number is prime. You could use the following loop to determine if a number was prime :

import java.util.\*;

class Test

{

public static void main ( String[] args)

{

Scanner input = new Scanner (System.in);

int number, k , done;

k=2; // start with the first prime number

done=0;

System.out.println("Type in a number");

number=input.nextInt();

while(k<=number/2 && done==0) // continue until the numbers are too big

{

if ((number % k) ==0) // if the number is divisible by the prime number

{ //the number is not prime

done=1;

}

else

k++; // move on to the next number

}

if ((done) ==0)

{System.out.println("The number "+number+" is prime.");}

else

{ System.out.println("The number "+number+" is not prime."); }

}

}

### Separating Digits

Modular division is also useful for separating the digits of a number.

Ex. The number 243 is made up of the digits 2 , 4 and 3.

You could use the following loop to separate the digits of a number :

do

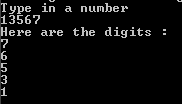
{

digit=number % 10; // the remainder when you divide by 10 is the digit on the right

number=number/10; // divide the number by 10 to get rid of the last digit

System.out.println(digit); //print the digit

}while (number>0); // continue until all the digits are printed



# Generating Random Numbers

You can now use Java's random function to generate a random number. The syntax is:

*number* **= (int ) (Math.random( )\*** *range*  **+** *lowest)* **;**

*number* is an integer that will be used to store the random number

**Math.random( )** calls the random function which generates a random decimal

*range* is a number that specifies how many possible integers can be generated.

*lowest* is a number that specifies the lowest possible integer that can be generated.

|  |  |
| --- | --- |
| **Example** | Output |
| **import** java.util.\*;  **class** TestRandom  {  **public** **static** **void** main ( String[] args)  {    **int** r;    // random number from 1-100  r=(**int** )(Math.*random*( )\* 100 + 1) ;  System.*out*.println("random #" + r);    // random number from 1-5  r=(**int** )(Math.*random*( )\* 5 + 1) ;  System.*out*.println("random #" + r);  }  } | // the program generate different sets of numbers each time the //program runs |

**Note:** By default, **Math.random( )** returns a random number over all possible decimal values. If you want your random number to be from 1 to your range specified, you must specify your range and then add one to your range so that you do not start at 0.(see example above.)

# Arrays

Many computer programs , such as spreadsheets, deal with collections of **related** data.. It can be very time consuming to perform calculations on this data when each variable has a different name. For example, if we wanted to calculate the average of 200 numbers we would have to work with 200 variables. To avoid this problem we use an **array** to store variables **of the same type**.

An array can be thought of as a row of storage spaces where each storage space is accessed by specifying the array name and the number (called the **index**) of the storage space.

Before looking at the syntax of declaring and working with arrays, here is an example of what an array is like:.

**Example**

We have twelve grades:

73 21 98 67 54 87 90 43 56 78 89 95

We need an array variable to store the data. This is how such an array is organized.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 73 | 21 | 98 | 67 | 54 | 87 | 90 | 43 | 56 | 78 | 89 | 95 |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |

Let's call the above array **marks.**

|  |  |
| --- | --- |
| **The parts of any array are:** | **Example (from above)** |
| type of data | integer |
| Name or identifier | marks |
| index values | 0,1,2,3,4,5,6,7,8,9,10,11 |
| components | marks[0], marks[1],…,marks[11] |
| elements | 73, 21, 98, 67, 54, 87, 90, 43, 56, 78, 89, 95 |

Note:

* The terms *element* and *component* are often used interchangeably, which can be confusing.
* The *index* of an array in Java is best thought of as the offset of that component from the first component. So the first component is offset=0 from itself, hence the index=0.
* If you try to access a component with an index outside of the range that you have declared (ie: marks[12]), you get an error. Be careful!

In the program, when we want to work with the data in the array, we do not just refer to the array. We have to refer to the specific component of the array.

|  |  |
| --- | --- |
| **Example** |  |
| System.out.println(marks); | Wrong. Java will not know which component to output. |
| System.out.println(marks[2]); | Correct. Java will output "98" |
| System.out.println(marks[12]); | Wrong. The index "12" is out of range. Valid indices of the array **marks** are 0, 1, … 11. |

### Declaring an Array

Arrays take up a large amount of space and so the amount of space required must be set aside before the array can be used.

Arrays are declared in the same way that variables are declared.

*dataType*[ ] *arrayName* = new *dataType* [*length*] ;

|  |  |
| --- | --- |
| **Example** |  |
| int[ ] marks =new int[12]; | An array of 12 integers. |
| float[ ] balance= new float[100]; | An array of 12 real numbers |
| char [] name= new char[20]; | An array of 20 characters. **Note**: these are equivalent to strings. |
| bool [ ] active = new bool[20]; | An array of 12 boolean values. |

### Putting Information Into An Array

Information can be put in the array two ways:

1. directly (when you declare the array)
2. with a loop. (used when information needs to be input by the user or generated by a formula)

Putting information into an array directly when declaring the array:

|  |  |
| --- | --- |
| **Example** | **Equivalent To:** |
| int[ ] MyNumber= {2,5,7,8,9}; | int[ ] MyNumber=new int[5];  MyNumber[0] = 2;  MyNumber[1] = 5;  MyNumber[2] = 7;  MyNumber[3] = 8;  MyNumber[4] = 9; |
| float [ ] MyGrade = {90, 80.5, 99.9}; | float [ ] MyGrade= new float[3];  MyGrade[0] = 90;  MyGrade[1] = 80.5;  MyGrade[2] = 99.9; |
| String[] friends={“Kermit”,”Roxy”,”Smedley”}; | String[] friends=new String[3];  friends[0]=”Kermit”;  friends[1]=”Roxy”;  friends[2]=”Smedley”; |
| bool [ ] truths = {true, true, true}; | bool [ ] truths= new bool[3];  truths[0] = true;  truths[1] = true;  truths[2] = true; |

**Using a Loop to put information into an array**

Often, a loop is used to initialize an array when:

1. the data is entered by the user
2. the data is read from a file
3. the data is derived from a formula
4. the array components need to be initialed to a start value, like zero

**Using a Loop to put information into an array**

|  |  |
| --- | --- |
| **Example** |  |
| int[ ] Grades= new int[5];  for ( i = 0; i < 5; i++)  { //enter the grades, prompt user with course #  System.out.println("Grade for course # " + (i+1) + ":"); Grades[i]=input.nextInt();  }  System.out.println("Here are the grades:");  for ( i = 0; i < 5; i++)  {  System.out.print(Grades[i]+" ");  } |  |
| final int enrolled = 2017;  int [ ] StudentNum= new int[enrolled];  for ( i = 0; i < enrolled; i++)  StudentNum[i] = 0; | This array has 2017 components. A loop efficiently initializes all values to zero. |
| int i;  final double pi = 3.14;  double [ ] CircleAreas= new double[5];  for ( i = 0; i < 5; i++)  CircleAreas[i] = pi \* Math.pow(i, 2.0);  for ( i = 0; i < 5; i++)  System.out.println("A circle of radius "+ i+" has an area  of " +CircleAreas[i]); |  |

### Best Practices: Using a Constant for the Array Size

It is good programming practice to use a constant for the size of an array. That way if you need to change the size of the array you only need change one line of code.

Example Program

import java.util.\*;

class Test

{

public static void main ( String[] args)

{

Scanner input = new Scanner (System.in);

final int size = 7;

int[] number= {29,51,43,6,3,78,7};

int i;

for ( i=0; i < size ; i++)

{

System.out.println("Number"+ (i+1) +":" +number[i]);

}

}

}

### 



### Two Dimensional Arrays

Arrays can be multidimensional with an index for each dimension.

A two-dimension array can be thought of as a table of rows and columns.

For example, consider the following table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Final Grade** | | | | |
| **Student#** | **Course#1** | **Course#2** | **Course#3** | **Course#4** | **Course#5** |
| **1** | 67 | 78 | 82 | 65 | 78 |
| **2** | 82 | 74 | 93 | 87 | 83 |
| **3** | 78 | 83 | 77 | 68 | 72 |
| **4** | 93 | 97 | 93 | 87 | 85 |
| **5** | 66 | 62 | 57 | 52 | 45 |
| **6** | 74 | 75 | 78 | 76 | 77 |
| **7** | 87 | 89 | 75 | 97 | 93 |
| **8** | 85 | 67 | 83 | 84 | 80 |

This table shows the final grades, for each of eight students, for five courses. This table has:

* eight rows – one per student
* five columns – one per course

To implement an array to hold this data, we would need a two-dimensional array of eight rows and five columns. The following declares such an array:

Do use constants for the size of your array indexes:

* minimizes index-out-of-range errors later
* makes modifying code easy
* meaningful names for the indices make accessing the components of the array less confusing.

final int NumOfStudents = 8;

final int NumOfCourses = 5;

int [ ][ ] TermMarks= new int [NumOfStudents][NumOfCourses];

//declares an array with 8 rows and 5 columns

|  |  |
| --- | --- |
| **The parts of the array are:** | **Example (from above)** |
| Type of data | int |
| Name or identifier | TermMarks |
| index values | rows : 0, 1, 2, 3, 4 , 5 ,6 , 7  columns : 0, 1, 2, 3, 4 |
| components | TermMarks[0][0], TermMarks[0][1],…,TermMarks[0][4]  TermMarks[1][0], TermMarks[1][1],…,TermMarks[1][4]    TermMarks[7][0], TermMarks[7][1],…,TermMarks[7][4] |
| elements | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 67 | 78 | 82 | 65 | 78 | | 82 | 74 | 93 | 87 | 83 | | 78 | 83 | 77 | 68 | 72 | | 93 | 97 | 93 | 87 | 85 | | 66 | 62 | 57 | 52 | 45 | | 74 | 75 | 78 | 76 | 77 | | 87 | 89 | 75 | 97 | 93 | | 85 | 67 | 83 | 84 | 80 | |

Remember, the index is an offset from the first component. So:

* row index =0 is zero away from the first row (ie: student #1)
* row index=1 is one away from the first row (ie: student #2)

*and*

* column index =0 is zero away from the first row (ie: course #1)
* column index=1 is one away from the first row (ie: course #2)

**Syntax for specifying components of the array TermMarks**

|  |  |
| --- | --- |
| **Component** | **Element** |
| TermMarks[0][0] | for student #1, the grade made in the 1st course = 67 |
| TermMarks[4][3] | for student #5, the grade made in the 4th course = 52 |
| TermMarks[7][4] | for student #8, the grade made in the 5th course = 80 |

**Array Components used in statements**

|  |  |
| --- | --- |
| **Statement** |  |
| System.out.println(TermMarks[0][0]); | Outputs **67** (the grade the 1st student made in course #1) |
| grade = TermMarks[4][3] | the variable **grade** is assigned **52** (the grade the 5th student made in course #4) |
| if (TermMarks[7][4] > 50) | **80** (the grade the 8th student made in course #5) is compared to **50** |

### Putting Information Into A Two-Dimensional Array

Just like single dimensional arrays, information can be put in the array directly or with a loop.

Putting information into an array directly when declaring the array.

|  |  |
| --- | --- |
| **Example** | **Equivalent To:** |
| final int NumOfStudents = 8;  final int NumOfCourses = 5;  int [ ][ ] TermMarks={ {67, 78, 82, 65, 78},  {82, 74, 93, 87, 83},  {78, 83, 77, 68, 72},  {93, 97, 93, 87, 85},  {66, 62, 57, 52, 45},  {74, 75, 78, 76, 77},  {87, 89, 75, 97, 93},  {85, 67, 83, 84, 80} }; | final int NumOfStudents = 8;  final int NumOfCourses = 5;  int [ ] [ ] TermMarks = new int[NumOfStudents][NumOfCourses];  TermMarks [0] [0] = 67;  TermMarks [0] [1] = 78;  TermMarks [0] [2] = 82;  TermMarks [0] [3] = 65;  TermMarks [0] [4] = 78;  TermMarks [1] [0] = 82;  TermMarks [1] [1] = 74;  ….  TermMarks [7] [4] = 80; |

Putting information into a two dimensional array using two loops

|  |  |
| --- | --- |
| **Example** | **Output:** |
| final int NumOfStudents = 3;  final int NumOfCourses = 5;  int student, course;  int [ ] [ ] TermMarks = new int[NumOfStudents][NumOfCourses];  //repeat loop for every student  for (student=0; student < NumOfStudents; student++)  {  System.out.println("Enter grades for student # "+ (student+1));  //input a mark for each course  for (course = 0; course < NumOfCourses; course++)  {  System.out.println("Grade for course # "+ (course+1));  TermMarks [student] [course]=input.nextInt();  }  }  //print all students’ marks  for (student=0; student < NumOfStudents; student++)  {  System.out.println("Grades for student # "+ (student+1));  // print marks for every course each student is taking  for (course = 0; course < NumOfCourses; course++)  {  System.out.print(TermMarks[student][course]+" ");  }  System.out.println(); //move down a line after printing marks  } |  |

# Modular Programming Using Methods

In QBasic, you learned how to program sub programs. In Visual Basic, each object could have code in its own private sub.

In Java, sub programs are called methods. Some things about Java methods are very similar to what you have learned about QBasic sub programs. Some things are different. But the idea is the same.

In order to make a program easier to follow and easier to compose, large complex problems are broken into smaller parts called modules. The modules are then shorter and much easier to create. They are linked together to make a complete program. (So far, you have used Java's main method.)

In Java a module is known as a method. A method is made up of:

1. a **method declaration** and
2. a **method body**.

**What is a method declaration**?

A **method declaration** is a single line of code that states the inputs and outputs of the method as well as the access level.

The format is **accessLevel ReturnType MethodName (inputType1 variable1, inputType2 variable2, … inputTypen varaiblen)**

1. accessLevel – determines if other classes can call the method. Ex. A public method can be called by any other method.
2. ReturnType – a method can send back one variable to the main method. ReturnType specifies what type of data (if any) will be sent back from the method. If no data is being sent back, the ReturnType is void.
3. MethodName – name of the method
4. inputType1 variable1, inputType2 variable2, …inputTypen varaiblen - the list of variables sent from the main method to be used by this method.

**What is a method body**?

A **method body** contains the statements of code that do the work of the method.

**Notes:**

* The input variables, listed after the method’s name are known as "arguments".
* A method may have none to many arguments.
* A method can only return one simple data item such as int, char and float or an array of simple data types.
* If you want to return more than one result you need two functions or two calls to the same functions.
* If the method does not return a data type, use **void**.
* The method inputs or variables are enclosed in parentheses. Declare them as you would other variables. These variables belong to the method and can not be accessed from any where else in the program.

**Example method declarations**

|  |  |
| --- | --- |
| **Example** | **Note** |
| public static double get( ) | * is named 'get' * does not receive any input from the calling program * returns data of type double |
| public static double add(double a, double b ) | * is named ‘add’ * two doubles, a and b, will be sent to this method * returns data of type double |
| public static void print(double a, double b, double answer, String word ) | * is named print * three doubles, a,b and answer will be sent to this method as well as one String word. * does not return any data to the calling program |

**The Placement Of The Method**

**IMPORTANT!!!!** The method always comes **AFTER** the Main method**…. NEVER** inside!!!

The methodis just like a normal Java program. It is a complete unit and is typed in before the main method.

If the method returns a value, a return() statement is used in the method body for returning the expected value.

**Example methods**

|  |  |
| --- | --- |
| **Example** | **Note** |
| public static double get( )  { Scanner input = new Scanner (System.in);  double number;  System.out.println("Type in a number");  number=input.nextDouble();  return(number);  } | * This method is used to get a number from the keyboard * The user is prompted to enter a number * The number is returned to the calling program |
| public static double add(double a, double b )  {  double sum;  sum=a+b;  return(sum);  } | * This method is used to add two numbers * Two doubles, a and b, are sent to this method * The sum of the two numbers is returned to the program |
| public static void print(double a, double b, double answer, String word )  {  System.out.println("The "+word+" of "+a+  " and "+b+" is "+answer);  } | * This method is used to print an answer * three doubles, a, b and answer will be sent to this function as well as one String containing the word to be printed. * No data is returned to the calling program |

**Calling a method**

To execute, or call a method you call it by name and give the method any variables that it needs to work. If the method returns a result you need to use a variable when you call the method.

**Examples of calls to a method**

|  |  |
| --- | --- |
| **Method** | **Calls to the method** |
| public static double get( ) | x=get( ); //each call the this method, gets a new  y=get( ); //number |
| public static double add(double a, double b )  // a = x and b = y | result=add(x,y);//x and y are sent to the method  //the answer is returned to result |
| public static void print(double a, double b, double answer, String word )  // a = x , b = y, answer=result , word=”sum” | print(x,y,result,"sum");  //x,y,result and the word “sum” are sent to the //method to be printed |

##### Notes

* Java matches the variables listed in the method declaration with the data in the call to the method. Arguments in a method call must be passed to a method in the same order as the parameter declarations.
* When calling a method, you must send the method the data it expects.
* **When passing an array to a method**, the size of the array is specified in the method declaration*,* ***but not*** in the statement calling the method. Defining the size of any array as a constant and passing that constant to any function using that array is just good programming practice.
* **/\*\* \*/** Used to enclose documentation comments for a method

### Example Program Using Methods Calculate the sum and quotient of two numbers

import java.util.\*;

class Test

{

//----------------------------------- main method ----------------------------------------------------

public static void main ( String[] args)

{

//Scanner input = new Scanner (System.in);

double x,y,result;

x=get();

y=get();

result=add(x,y);

print(x,y,result,"sum");

result=divide(x,y);

print(x,y,result,"quotient");

}

/\*\* Get a number from the user

\* Returns the number to the main method

\*/

public static double get( )

{ Scanner input = new Scanner (System.in);

double number;

System.out.println("Type in a number");

number=input.nextDouble();

return(number);

}

// Add two numbers

public static double add(double a, double b )

{

double sum;

sum=a+b;

return(sum);

}

// Divide two numbers

public static double divide(double a, double b )

{

double quotient;

quotient=a/b;

return(quotient);

}

// Print answers

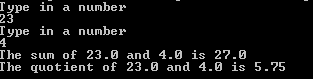
public static void print(double a, double b, double answer, String word )

{

System.out.println("The "+word+" of "+a+" and "+b+" is "+answer);

}

}



# Sorting

It is often useful to put a list in order either according to number or according to the alphabet. In computer science, this is known as sorting. There are numerous methods of sorting as sorting a very long list can be a very time consuming process. One of the most simple forms of sorting is the bubble sort. The bubble sort is so named as the smallest number is “bubbled” up to the front of the list and then the next smallest number is moved next. This is repeated until all the numbers are in order.

Use the following code to sort a list of numbers from smallest to largest :

**class** Assignment12A

{

**public** **static** **void** main (String[] args)

{

**int** [ ] points={10,2,4,7,1,13,20,3};

**int** temp;

**int** i,j;

System.*out*.println();System.*out*.println

("Points in order from lowest to highest");

//points highest to lowest

**for** (i=0; i<7; i++) // compare the number in the front of the list with

{ // every number after it

**for** (j=i+1; j<8; j++)

{

**if** (points[i]>points[j]) //if the number at the front is

{//swap // larger, swap the numbers

temp=points[i];

points[i]=points[j];

points[j]=temp;

}//end of if

}

}

**for** (i=0; i<8; i++)

{

System.*out*.printf("%5d",points[i]);

}

System.*out*.println();

Capture.PNG

}

}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  | Compare the first item in the list |
| i=0, j=1 | 10 | 2 | 4 | 7 | 1 | 13 | 20 | 3 | Swap the 10 and 2 |
| i=0, j=2 | 2 | 10 | 4 | 7 | 1 | 13 | 20 | 3 | No change |
| i=0, j=3 | 2 | 10 | 4 | 7 | 1 | 13 | 20 | 3 | No change |
| i=0, j=4 | 2 | 10 | 4 | 7 | 1 | 13 | 20 | 3 | Swap the 2 and the 1 |
| i=0, j=5 | 1 | 10 | 4 | 7 | 2 | 13 | 20 | 3 | No change |
| i=0, j=6 | 1 | 10 | 4 | 7 | 2 | 13 | 20 | 3 | No change |
| i=0, j=7 | 1 | 10 | 4 | 7 | 2 | 13 | 20 | 3 | No change |
|  |  |  |  |  |  |  |  |  | Compare the second item in the list |
| i=1, j=2 | 1 | 10 | 4 | 7 | 2 | 13 | 20 | 3 | Swap the 10 and the 4 |
| i=1, j=3 | 1 | 4 | 10 | 7 | 2 | 13 | 20 | 3 | No change |
| i=1, j=4 | 1 | 4 | 10 | 7 | 2 | 13 | 20 | 3 | Swap the 4 and the 2 |
| i=1, j=5 | 1 | 2 | 10 | 7 | 4 | 13 | 20 | 3 | No change |
| i=1, j=6 | 1 | 2 | 10 | 7 | 4 | 13 | 20 | 3 | No change |
| i=1, j=7 | 1 | 2 | 10 | 7 | 4 | 13 | 20 | 3 | No change |
|  |  |  |  |  |  |  |  |  | Compare the third item in the list |
| i=2, j=3 | 1 | 2 | 10 | 7 | 4 | 13 | 20 | 3 | Swap the 10 and the 7 |
| i=2, j=4 | 1 | 2 | 7 | 10 | 4 | 13 | 20 | 3 | Swap the 7 and the 4 |
| i=2, j=5 | 1 | 2 | 4 | 10 | 7 | 13 | 20 | 3 | No change |
| i=2, j=6 | 1 | 2 | 4 | 10 | 7 | 13 | 20 | 3 | No change |
| i=2, j=7 | 1 | 2 | 4 | 10 | 7 | 13 | 20 | 3 | Swap the 4 and the 3 |
|  |  |  |  |  |  |  |  |  | Compare the fourth item in the list |
| i=3, j=4 | 1 | 2 | 3 | 10 | 7 | 13 | 20 | 4 | Swap the 10 and the 7 |
| i=3, j=5 | 1 | 2 | 3 | 7 | 10 | 13 | 20 | 4 | No change |
| i=3, j=6 | 1 | 2 | 3 | 7 | 10 | 13 | 20 | 4 | No change |
| i=3, j=7 | 1 | 2 | 3 | 7 | 10 | 13 | 20 | 4 | Swap the 7 and the 4 |
|  |  |  |  |  |  |  |  |  | Compare the fifth item in the list |
| i=4, j=5 | 1 | 2 | 3 | 4 | 10 | 13 | 20 | 7 | No change |
| i=4, j=6 | 1 | 2 | 3 | 4 | 10 | 13 | 20 | 7 | No change |
| i=4, j=7 | 1 | 2 | 3 | 4 | 10 | 13 | 20 | 7 | Swap the 10 and the 7 |
|  |  |  |  |  |  |  |  |  | Compare the sixth item in the list |
| i=5, j=6 | 1 | 2 | 3 | 4 | 7 | 13 | 20 | 10 | No change |
| i=5, j=7 | 1 | 2 | 3 | 4 | 7 | 13 | 20 | 10 | Swap the 13 and the 10 |
|  |  |  |  |  |  |  |  |  | Compare the seventh item in the list |
| i=6, j=7 | 1 | 2 | 3 | 4 | 7 | 10 | 20 | 13 | Swap the 20 and the 13 |
|  | 1 | 2 | 3 | 4 | 7 | 10 | 13 | 20 | The list is now in order |

# 

# Characters and Strings

Characters and Strings are both basic data types. Java provides the java.lang package for dealing with the storing and manipulating of strings. This package is automatically included and does not need to be imported.

**Difference between double and single quotes**

* Single quotes are used for enclosing values of type char.
* Double quotes are used to enclose values of type string.

Characters and Strings are declared just like other variables.

Ex. String sentence = “Here is my sentence”;

String word=”Hello”;

char letter=’a’;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| H | e | l | l | o |
| **0** | **1** | **2** | **3** | **4** |

A string represents an array of characters with each letter taking

up one space. The index of the first letter is 0, the index of the second letter is 1, the index of the third letter is 2, and so on.

Null is used to indicate that there are no characters in a string. Ex. String EmptySentence = null;

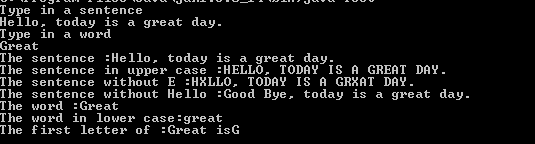
Some of the String class methods include :

|  |  |  |
| --- | --- | --- |
| **Method** | **Example** | **Description** |
| charAt( int index) | Letter=sentence.charAt(i); | Returns the character at the specified index. |
| compareTo(String str) | x=String1.compareTo(string2); | Compares a string to str. Example  Returns -1 if string1 < string 2 cat < zebra  Returns 0 if string1 = string 2 cat = cat  Returns 1 if string1 > string 2 cat > bird |
| concat (String str) | String word=”work”;  newWord= word.concat(“ed”);  // newWord = worked | Returns a new string with str attached to the end of the old string. |
| endsWith(String str) | String sentence = "worked";  **boolean** x;  x= sentence.endsWith("ed");  // x = true | Tests if the string ends with str.  Returns true if the string ends in str or false if the string does not end in str. |
| equals (String str) | String word = "worked";  **boolean** x;  x= sentence.equals("ed");  // x = false | Returns true if the string is equal to string str. |

|  |  |  |
| --- | --- | --- |
| equalsIgnoreCase (String str) | String word = "ED";  **boolean** x;  x=word.equalsIgnoreCase,"ed");  // x = true | Returns true if the string is equal to string str, without regard to case. |
| indexOf(char ch) | String word = "worked";  **int** x;  x=word.indexOf('e');  // x = 4 | Returns the index within the string of the first occurrence of the specified character. |
| indexOf(char ch, int start) | String word = "encyclopedia";  **int** x;  x=word.indexOf('e',3);  // x = 8 | Returns the index within the string of the first occurrence of the specified character, starting the search at the specified index. |
| indexOf(String str) | x =sentence.indexOf(“and”); | Returns the index within the string of the first occurrence of the specified substring str.  Ex. sentence= “ hand” x = 1  sentence= “ brand” x = 2  sentence= “ toast” x = -1 |
| indexOf(String str, int start) | String word = "abracadabra";  **int** x;  x=word.indexOf("ab",3);  // x = 7 | Returns the index within the string of the first occurrence of the specified substring str,  starting the search at the specified index. |
| length() | x = sentence.length(); | Returns an integer that indicates the length  of the string. |
| replace (char old, char new) | newSentence=upperCase.replace  ('E','X'); | Returns a new string with all the occurrences  of the old character being replaced with the  new character. |
| replaceAll (String str, String str2) | newSentence=sentence.replaceAll(“old”,”new”); | Returns a new string with all occurrences of  str replaced with str2 |
| replaceFirst (String str, String str2) | String word = "abracadabra";  newWord=word.replaceFirst("ab","xy");  // newword=xyracadabra | Returns a new string with the first occurrence  of str replaced with str2 |
| startsWith(string str) | String word = "abracadabra";  **boolean** x;  x=word.startsWith("abra");  //x = true | Tests if the string starts with str. |
| substring( int start, int end) | String word = "abracadabra";  String sub;  sub=word.substring(4,8);  // sub = cada | Returns a substring of the string, starting at start and ending at end. |
| toLowerCase() | sentence=sentence.toLowerCase(); | Returns a new string in lowercase. |
| toUpperCase() | big=sentence.toUpperCase(); | Returns a new string in uppercase. |

\* Looking for more ? You can find more string classes by following this link : http://java.sun.com/j2se/1.4.1/docs/api/

## Example Program



import java.util.\*;

class Test

{

public static void main ( String[] args)

{

Scanner input = new Scanner (System.in);

String sentence,word,upperCase, lowerCase, newSentence;

String newString="Good Bye";

char letter;

System.out.println("Type in a sentence"); //enter a sentence

sentence=input.nextLine();

System.out.println("Type in a word"); //enter a word

word=input.next();

System.out.println("The sentence :" + sentence);

upperCase=sentence.toUpperCase(); //convert to upper case

System.out.println("The sentence in upper case :" +upperCase);

newSentence=upperCase.replace('E','X'); //replace E with X

System.out.println("The sentence without E :" +newSentence);

newSentence=sentence.replaceAll("Hello",newString); // replace Hello with GoodBye

System.out.println("The sentence without Hello :" +newSentence);

System.out.println("The word :" + word);

lowerCase=word.toLowerCase(); //convert to lower case

System.out.println("The word in lower case:" + lowerCase);

letter = word.charAt(0); //determine the letter with an index of 0

System.out.println("The first letter of :" + word + " is" + letter);

}

}

# 

# Defining a class

Classes are defined in a separate java file.

**public** **class** Athlete

{

**private** String name; //variables are declared as private, meaning they

**private** String position; // can only be accessed in the class

**private** **int** goals;

**private** **int** assists;

**public** **void** initializeAthlete(String n, String p, **int** g, **int** a)

{

name=n; //This method initializes the variables. Once

position=p; // they have been initialized, they can be used

goals=g; // by any method in this class

assists=a;

}

**public** **void** displayAthlete()

{// display the data associated with Athlete

System.*out*.printf("%s plays %s and has scored %d goals and %d assists", name , position , goals , assists);

}

}

## Using a class

To use a class , create a new java file and write a program that calls the class.

Ex.

**class** Assignment11

{

**public** **static** **void** main (String[] args)

{

Athlete person = **new** Athlete(); // creates a new Athlete

person.initializeAthlete("Amy","defense",21,5);

person.displayAthlete();

}

}

Dot notation is used to access the methods or any public variables.

Ex. person.displayAthlete(); //this will call the method DisplayAthlete that is associated with person

Ex. System.out.println(person.name); //this would display the name of the athlete IF the variable person was public

## Creating an array of variables that use the class

Ex. Create an array of countries, where Country represents a class.

Country[] choice = **new** Country[5];

**for** (**int** i = 0; i<5; i++)

{

choice[i] = **new** Country();

}

# Graphical User Interface

Graphical User Interface related classes are defined in two packages :

Abstract Windowing Toolkit : **import.java.awt.\*;** and Swing : **import.javax.swing.\*;** which was added later and provided components that are more versatile.

### Frames and Panels

A frame is a container that is used to display a graphical user interface. It is displayed as a separate window and has its own title bar. A frame is defined by the JFrame class.

A panel is also a container but it can not be displayed on its own. A panel must be added to another container for it to be displayed.

## Displaying a dialog box

A dialog box can be used to display a message or to get input from the user. The following program will demonstrate the use of a dialog box.

**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**public** **class** Dialog1

{

**public** **static** **void** main ( String[] args)

{

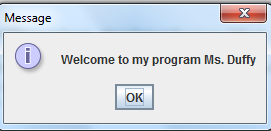
String name;

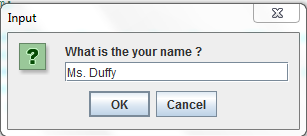
name=JOptionPane.*showInputDialog*("What is the your name ?");

JOptionPane.*showMessageDialog*(**null**,"Welcome to my program "+ name);

}

}





Format of the commands for dialog boxes:

To get input from the user : stringName= JOptionPane.*showInputDialog*("message”);

Ex. name= JOptionPane. *showInputDialog* (" What is the your name ?");

sentence= JOptionPane.*showInputDialog*(" Type in a sentence ");

To display the message : JOptionPane.*showMessageDialog*(**null**,"Message "+ variable);

Ex. JOptionPane.*showMessageDialog*(**null**,"Welcome to my program "+ name);

JOptionPane.*showMessageDialog*(**null**,"Today is "+ day+" and the month is"+month);

The *showInputDialog* method will only return a string and so you must convert the string to an integer if you are going to perform a mathematical calculation.

Ex.

**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**public** **class** Dialog1

{

**public** **static** **void** main ( String[] args)

{

**int** x,y,sum;

String X, Y;

X=JOptionPane.*showInputDialog*("What is the X value ?");

x = Integer.*parseInt*(X.trim());

Y=JOptionPane.*showInputDialog*("What is the Y value ?");

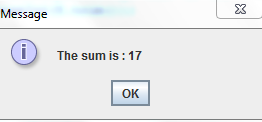
y = Integer.*parseInt*(Y.trim());

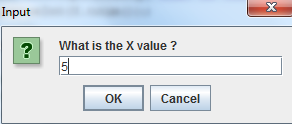
sum = x + y;

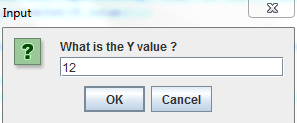
JOptionPane.*showMessageDialog*(**null**,"The sum is : "+ sum);

}

}







To convert from a string to an integer use: integerVariable = Integer.*parseInt*(stringVariable.trim());

Ex. x = Integer.*parseInt*(X.trim());

Answer = Integer.*parseInt*(num1.trim());

## 

## Creating Simple Drawings

The following program will demonstrate how to draw a simple object.

First create a java application DrawPanel. This code will draw an X going from (0,0)-(100,200) and (0,200)-(100,0). You must import java.awt.Graphics; and javax.swing.JPanel; in order to do any graphics.

**import** java.awt.Graphics;

**import** javax.swing.JPanel;

**public** **class** DrawPanel **extends** JPanel

{

**public** **void** paintComponent (Graphics g)

{

**super**.paintComponent(g);

g.drawLine(0,0,100,200);

g.drawLine(0, 200, 100, 0);

}

}

Then create a second file DrawPanelTest. This code will create a window to display the drawing. You must import javax.swing.JFrame; to display the frame.

**import** javax.swing.JFrame;

**public** **class** DrawPanelTest

{

**public** **static** **void** main ( String[] args)

{

// create a panel to contain the drawing

DrawPanel panel = **new** DrawPanel();

// create a new frame to hold the panel

JFrame application = **new** JFrame();

// set the frame to exit when it is closed

application.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

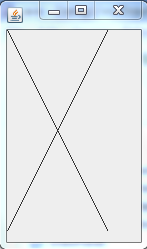
application.add( panel); // add the panel to the frame

application.setSize(150,250); // set the size of the frame

application.setVisible(**true**); // make the frame visible

}

}



## 

## Graphics

To draw objects in Java, the following code is available.

|  |  |
| --- | --- |
|  |  |
| **Syntax:** | **Example:** |
| drawstring (String str, int x , int y); | String str – string that will be drawn on the applet  int x – the horizontal distance in pixels, from the left edge of the applet  int y – the vertical distance in pixels, from the top edge of the applet |
| drawLine(int x1 , int y1, int x2, int y2); | Draws a line from (x1,y1) to (x2,y2) |
| drawRect (int x , int y, int width, int height); | Draws rectangle that has an upper left hand corner at (x,y) and extends the indicated width and height from that point. |
| fillRect (int x , int y, int width, int height); | Draws and paints a rectangle that has an upper left hand corner at (x,y) and extends the indicated width and height from that point. |
| draw3DRect (int x , int y, int width, int height, boolean raised); | Draws a 3D highlighted outline of the specified rectangle. |
| drawOval (int x , int y, int width, int height); | Draws an oval that fits inside the rectangle that has an upper left hand corner at (x,y) and extends the indicated width and height from that point. |
| fillOval (int x , int y, int width, int height); | Draws and paints an oval that fits inside the rectangle that has an upper left hand corner at (x,y) and extends the indicated width and height from that point. |
| drawArc (int x , int y, int width, int height, int startAngle, int arcAngle); | Draws the outline of an arc covering the specified rectangle |
| fillArc (int x , int y, int width, int height, int startAngle, int arcAngle); | Draws and paints an arc covering the specified rectangle |
| drawPolygon(Polygon p); | Draws the outline of a polygon defined by the specified points in the polygon. |
| fillPolygon(Polygon p); | Draws and fills the polygon defined by the specified points in the polygon. |
| drawPolyline(int [ ] xPoints, int [ ] yPoints, int nPoints); | Draws a sequence of connected lines defined by the arrays of x and y coordinates. |
| setBackground (Color.something);  Ex. Color.RED  Color.BLUE  Color.GREEN  Color.WHITE | Changes the colour of the drawing surface. |

## Color

To set the current drawing color use : g.setColor(Color.*nameOfColor*);

Ex. g.setColor(Color.*YELLOW*); g.setColor(Color.*BLACK*);

This code will set he color to one of 13 predefined colors. All objects will be drawn in this color until a new color has been set.

Predefined colors:

BLACK , BLUE , CYAN , DARK\_GRAY , GRAY , GREEN , LIGHT\_GRAY ,

MAGENTA , ORANGE , PINK , RED , WHITE , YELLOW

To set the current drawing to a user defined color use : g.setColor(new Color(int red, int green, int blue);

The user can specify the amount of red, green and blue components.

Ex. g.setColor(**new** Color(255,0,0)); //red g.setColor(**new** Color(0,0,0)); //black

Example : The following program will demonstrate how to create objects of different colors.

First create a java application DrawSmiley. This code will draw a yellow circle with black eyes and a red mouth. You must import java.awt.Color; to use the color methods.

**import** java.awt.Graphics;

**import** javax.swing.JPanel;

**import** java.awt.Color;

**public** **class** DrawSmiley **extends** JPanel

{

**public** **void** paintComponent (Graphics g)

{

**super**.paintComponent(g);

g.setColor(Color.*YELLOW*);

g.fillOval(10,10,200,200);

g.setColor(Color.*BLACK*);

g.fillOval(55,65,30,30);

g.fillOval(135,65,30,30);

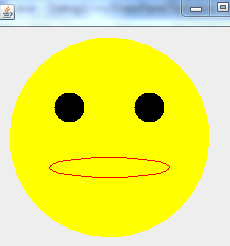
g.setColor(**new** Color(255,0,0));

g.drawOval(50,130,120,20);

}

}

Then create a second file DrawPanelTest. This code will create a window to display the drawing.



**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**public** **class** DrawPanelTest

{

**public** **static** **void** main ( String[] args)

{

DrawSmiley panel = **new** DrawSmiley();

JFrame application = **new** JFrame();

application.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

application.add( panel);

application.setSize(300,300);

application.setVisible(**true**);

}

}

## Using Variables to Change Drawings

The following program will demonstrate how to change a drawing using user input.

In DrawPanel create a variable **choice** that can be used to change the drawing.

**import** java.awt.Graphics;

**import** javax.swing.JPanel;

**public** **class** DrawPanel **extends** JPanel

{

**private** **int** choice;

**public** DrawPanel ( **int** userChoice)

{

choice = userChoice;

}

**public** **void** paintComponent (Graphics g)

{

**super**.paintComponent(g);

**for** (**int** i = 0; i<choice; i++)

{

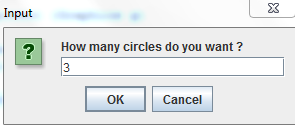
g.drawOval(10,10+50\*i,40,40);

}

}

}

In DrawPanelTest ask the user for the number of circles.



**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**public** **class** DrawPanelTest

{

**public** **static** **void** main ( String[] args)

{

**int** x;

String number;

number=JOptionPane.*showInputDialog*("How many circles do you want ?");

x = Integer.*parseInt*(number.trim());

DrawPanel panel = **new** DrawPanel(x);

JFrame application = **new** JFrame();

application.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

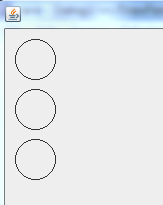
application.add( panel);

application.setSize(300,300);

application.setVisible(**true**);

}

}



## Responding to Mouse Events

In order to have the program respond to mouse events you must implement MouseListener. When implementing the MouseListener, the program must add methods for mouseClicked, mouseEntered, mouseExited, mousePressed and mouseReleased even though you will generally only enter code for the mouseClicked method. The other methods must be included but are left blank.

The following program will demonstrate how to respond to the user clicking somewhere on the screen.

In the class Responding, the location of the mouse is determined using the code :

x=e.getX(); // get the x and y location of the mouse

y=e.getY();

A rectangle is then drawn at this location.

**import** java.awt.\*;

**import** java.awt.event.InputEvent;

**import** java.awt.event.MouseEvent;

**import** java.awt.event.MouseListener;

**import** javax.swing.\*;

**public** **class** Responding **extends** JPanel **implements** MouseListener

{

**int** x = 0;

**int** y = 0;

**public** Responding()

{

addMouseListener(**this**);

}

**public** **static** **void** init() **throws** InterruptedException

{

}

**public** **void** paintComponent(Graphics g)

{

**super**.paintComponent(g);

Font myFont = **new** Font("Arial", Font.*PLAIN*,30); //Set the font to a bigger size

g.setFont(myFont);

g.drawString ("Place the mouse on the screen and click. A square will appear",10,30);

// write text on the screen starting at location (10,30)

}

@Override

**public** **void** mouseClicked (MouseEvent e)

{

Graphics g = getGraphics();

**int** r,gn,b;

r=(**int**) ((Math.*random*()\*255)+1); // generate random amounts of red, green and blue

gn=(**int**) ((Math.*random*()\*255)+1);

b=(**int**) ((Math.*random*()\*255)+1);

x=e.getX(); // get the x and y location of the mouse

y=e.getY();

g.setColor(**new** Color(r,gn,b));

g.fillRect(x,y,30,30); //draw a rectangle at the mouse location

}

@Override

**public** **void** mouseEntered (MouseEvent arg0) { }

@Override

**public** **void** mouseExited (MouseEvent arg0) { }

@Override

**public** **void** mousePressed (MouseEvent arg0) { }

@Override

**public** **void** mouseReleased (MouseEvent arg0) { }

}

In MouseEvent the panel and frame are created.

**import** java.applet.Applet;

**import** java.awt.Dimension;

**import** java.awt.event.ActionEvent;

**import** java.util.\*;

**import** javax.swing.JFrame;

**public** **class** MouseEvent

{

**public** **static** **void** main(String[] args)

{

Responding panel = **new** Responding();

JFrame application = **new** JFrame("Random Rectangles"); //set title of frame

application.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

application.getContentPane().add("Center", **new** Responding());

application.pack();

application.setSize(**new** Dimension(700,700)); // set the size of the frame

application.setVisible(**true**); // make the frame visible

}

}

# Animation

## Movement With Arrow Keys

In order to have an object move using arrow keys you must implement ActionListener and KeyListener. When implementing the KeyListener, the program must add methods for keyPressed, keyReleased and keyTyped, even though you will generally only enter code for the keyPressed method. The other methods must be included but are left blank.

The following program will demonstrate how to move an object using the arrow keys. The code is very similar to the code used in Visual Basic.

The mainline is the standard mainline to create the panel and the frame.

**import** java.awt.Dimension;

**import** javax.swing.JFrame;

**import** javax.swing.JTextArea;

**public** **class** Animation

{

**public** **static** **void** main(String[] args)

{

MoveCircle panel = **new** MoveCircle();

JFrame application = **new** JFrame("Animation");

application.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

application.getContentPane().add("Center", **new** MoveCircle());

application.setSize(**new** Dimension(1000,600)); // set the size of the frame

application.setVisible(**true**); // make the frame visible

}

}

MoveCircle contains the code to move the circle when the arrow keys are pressed. A timer is included that determines how frequently the computer monitors the keys. Without the timer, the computer would not respond to the pressing of the keys.

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.awt.event.KeyListener;

**import** java.awt.event.KeyEvent;

**public** **class** MoveCircle **extends** JPanel **implements** ActionListener, KeyListener

{

Timer t = **new** Timer(5,**this**);

**int** x = 500;

**int** y = 300;

**public** MoveCircle()//Starts timer

{

t.start();

addKeyListener(**this**);

setFocusable(**true**);

setFocusTraversalKeysEnabled(**false**);

}

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

// draw a black circle on the screen at location (x,y)

g.setColor(Color.*black*);

g.fillOval(x,y,40,40);

}

**public** **void** actionPerformed(ActionEvent e) {

// refreshes the screen every time an action is performed

repaint(); //performs any code in paint

}

@Override

**public** **void** keyPressed(KeyEvent e)

{

**int** code = e.getKeyCode();

**switch**(code)

{

**case** KeyEvent.*VK\_DOWN*:

y=y+40; //change y location down

**break**;

**case** KeyEvent.*VK\_UP*:

y=y-40; //change y location up

**break**;

**case** KeyEvent.*VK\_RIGHT*:

x=x+40; //change x location right

**break**;

**case** KeyEvent.*VK\_LEFT*:

x=x-40; //change x location left

**break**;

**case** KeyEvent.*VK\_SPACE*:

// add code for other events

**break**;

}

}

@Override

**public** **void** keyReleased(KeyEvent arg0) { }

@Override

**public** **void** keyTyped(KeyEvent arg0) { }

}

## Continual Movement

In order to have an object move continuously, the movement must be changed in the actionPerformed method.

This program will set a direction according to the arrow keys and then move until another key is pressed.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.event.KeyListener;

import java.awt.event.KeyEvent;

public class AutomaticMove extends JPanel implements ActionListener, KeyListener

{

Timer t = new Timer(5,this);

int x = 500;

int y = 300;

int xVel=0;

int yVel=0;

public AutomaticMove()//Starts timer

{

t.start();

addKeyListener(this);

setFocusable(true);

setFocusTraversalKeysEnabled(false);

}

public void paintComponent(Graphics g) {

super.paintComponent(g);

// draw a black circle on the screen at location (x,y)

g.setColor(Color.black);

g.fillOval(x,y,40,40);

}

public void actionPerformed(ActionEvent e) {

// refreshes the screen every time an action is performed

x=x+xVel; // move the circle to the left or the right

y=y+yVel; // move the circle up or down

repaint(); //performs any code in paint

}

@Override

public void keyPressed(KeyEvent e)

{

int code = e.getKeyCode();

switch(code)

{

case KeyEvent.VK\_DOWN:

xVel=0; // do not move left or right

yVel=5; //change y location down

break;

case KeyEvent.VK\_UP:

xVel=0; // do not move left or right

yVel=-5; //change y location up

break;

case KeyEvent.VK\_RIGHT:

xVel=5; // change x location right

yVel=0; //do not move up or down

break;

case KeyEvent.VK\_LEFT:

xVel=-5; // change x location left

yVel=0; //do not move up or down

break;

case KeyEvent.VK\_SPACE:

// add code for other events

break;

}

}

@Override

public void keyReleased(KeyEvent arg0) {}

@Override

public void keyTyped(KeyEvent arg0) {}

}

## Loading Images

You can create images in paint or find images on the internet and load them on the screen at any location of your choosing. The following program can be used to play a basic tic tac toe game. Remember that you must first create a separate file for the mainline that will create the frame and the panel to display the graphics.

**import** java.awt.\*;

**import** java.awt.event.MouseEvent;

**import** java.awt.event.MouseListener;

**import** java.awt.image.BufferedImage;

**import** java.io.\*;

**import** javax.imageio.ImageIO;

**import** javax.swing.\*;

**public** **class** ShowImages **extends** JPanel **implements** MouseListener

{

//creating the image variables

**private** BufferedImage imgX = **null**;

**private** BufferedImage imgO = **null**;

**private** **int** whoseTurn=0;

**public** ShowImages()

{

addMouseListener((MouseListener) **this**);

}

**public** **void** mousePressed(MouseEvent e) { }

**public** **void** mouseReleased(MouseEvent e) { }

**public** **void** mouseEntered(MouseEvent e) { }

**public** **void** mouseExited(MouseEvent e) { }

**public** **void** mouseClicked(MouseEvent e)

{

Graphics g = getGraphics();

**int** x=e.getX(); // get the x and y location of the mouse

**int** y=e.getY();

//Check the square that has been clicked

**for** (**int** i = 0; i<3; i++)

{

**for** (**int** j=0; j<3;j++)

{ // check the top left block and then move 100 over to check every other block

**if** (x>=(400 + 100\*i) && x<=(500+100\*i) && y>=(100+100\*j) && y<=(200+100\*j))

{ **if** (whoseTurn==0)

{

g.drawImage(imgX, 405+100\*i, 105+100\*j, 90,90,**this**);

whoseTurn=1;

}

**else** **if** (whoseTurn==1)

{

g.drawImage(imgO, 405+100\*i, 105+100\*j, 90,90,**this**);

whoseTurn=0;

}// end check whose turn

}//end check location

}// end for j loop

}// end for i loop

}

The new code *g.drawImage(imgX, 405+100\*i, 105+100\*j, 90,90,****this****);* places the image at the specified location.

The format for this code is : g.drawImage(image imgX, int x , int y, int width, int height , **this**);

This code places the image with the upper left hand corner at (x,y) and extends the indicated width and height from that point. The image will be scaled to the indicated height and width.

Ex. : g.drawImage(imgX, 400 ,100, 50,50 , **this**); // places imageX at location (400,100)

The rest of the program contains the new code :

**try** // load the images

{

imgX=ImageIO.*read*(**new** File("X.jpg"));

imgO=ImageIO.*read*(**new** File("O.jpg"));

}

**catch**(IOException e) //catching if the image was not there

{

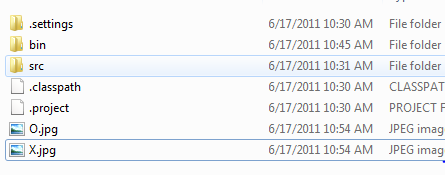
System.*out*.println("could not find image");

g.drawString("Could not find images",100,100);

}

The code *: imgX=ImageIO.read(****new*** *File("X.jpg"));*  reads the image from the file and stores it in imgX.

\*Warning : The files must be stored in your program



in the same place as your bin and src files are saved.

**void** saySomething(String eventDescription, MouseEvent e) { }

**public** **void** paintComponent(Graphics g)

{

**super**.paintComponent(g);

**try** // load the images

{

imgX=ImageIO.*read*(**new** File("X.jpg"));

imgO=ImageIO.*read*(**new** File("O.jpg"));

}

**catch**(IOException e) //catching if the image was not there

{

System.*out*.println("could not find image");

g.drawString("Could not find images",100,100);

}

g.setColor(Color.*WHITE*); // set up the tic tac toe grid

g.fillRect(400, 100, 300, 300);

g.setColor(Color.*BLACK*); // draw the rectangle

g.drawRect(400 , 100, 300, 300);

**for** (**int** i = 1; i<=3; i++) // draw the vertical lines

{g.drawLine(400 , 100+100\*i, 700,100+100\*i);}

**for** (**int** i = 1; i<=3; i++) // draw the horizontal lines

{g.drawLine(400+100\*i , 100, 400+100\*i,400);}

Font myFont = **new** Font("Arial", Font.*PLAIN*,30); //Set the font to a bigger size

g.setFont(myFont);// write text on the screen starting at location (250,30)

g.drawString ("Take turns placing an X or an O on the grid",250,30);

}

} // end of main class

# File Management

## Creating a new file

Files are used for storing and retrieving data. The File class, part of the java.io package, is used to create an object that represents a file. Some of the File class methods include :

|  |  |  |  |
| --- | --- | --- | --- |
| method | format | example | function |
| createNewFile() | nameOfFile.createNewFile(); | textFile.createNewFile(); | Creates a new file. Returns true if the file is created, false otherwise. |
| delete() | nameOfFile.delete(); | dataFile.delete(); | Deletes a file. Returns true if the file is deleted, false otherwise. |
| exists() | nameOfFile.exists(); | if(StudentInfo.exists())  System.out.println(“file exists”); | Determines if a file exists. Returns true if the file exists false otherwise. |

### Example Program

Use the following code to determine if a file exists :

import java.util.Scanner;

import java.io.\*;

public class TestFiles

{

public static void main (String[] args)

{

String filename="c:\\supplies.txt";

File textFile= **new** File(filename);

FileReader in;

BufferedReader readFile;

String sentence;

if (textFile.exists())

{

System.out.println("File exists");

}

else

{

System.out.println("File does not exist");

}

}

}

## Handling Exceptions

An exception is an error affecting program execution. If an exception is not taken care of the application will terminate.. An exception handler is a block of code that performs an action when an exception occurs. The *try-catch-finally* statement can be used to write an exception handler. The *try* statements are the statements that could generate an exception. The *catch* clauses wait for the exception and then executes the code. A separate catch statement is needed for each type of exception. The *finally* clause is optional and executes its statements regardless of what happens in the try-catch portion of the code.

### Example Program

The following code will check for the existence of a file before creating a new one. The createNewFile() method generates an IOException when the filename can not be used to create a file.

import java.util.Scanner;

import java.io.\*;

public class TestFiles

{

public static void main (String[] args)

{

String filename="c:\\supplies.txt";

File textFile= **new** File(filename);

FileReader in;

BufferedReader readFile;

String sentence;

if (textFile.exists())

{

System.out.println("File exists");

}

else

{

try

{

textFile.createNewFile();

System.out.println("New File created");

}

catch (IOException e)

{

System.out.println(" File could not be created");

System.err.println("IOException :"+ e.getMessage());

}

}

}

}

## Reading the contents of a text file

To read an existing file use the FileReader and BufferedReader classes. The FileReader class is used to create an input file stream. The BufferedReader class is used to read text from the stream.

|  |  |  |  |
| --- | --- | --- | --- |
| method | format | example | function |
| FileReader filename | FileReader filename; | FileReader in; | Creates an input stream for the file. |
| close() | nameOfFile.close(); | in.close(); | Closes the input file stream. |
| BufferedReader Reader Stream | BufferedReader streamName; | BufferedReader readFile; | Creates an buffered input stream. |
| read() | var=nameOfBuffer.read(); | letter=readFile.read() | Reads a single character from the input stream. |
| readLine() | var=nameOfBuffer.readLine(); | sentence=readFile.readLine() | Reads a line from the input stream. |
| close() | nameOfBuffer.close(); | readFile.close(); | Closes the input stream. |

### Example Program

Use the following code to read an existing file, line by line. First create a file supplies.txt using notepad and write something in the file.

import java.util.Scanner;

import java.io.\*;

public class TestFiles

{

public static void main (String[] args)

{

String filename="c:\\supplies.txt";

File textFile= **new** File(filename);

FileReader in;

BufferedReader readFile;

String sentence;

try

{

in = new FileReader(textFile);

readFile= new BufferedReader(in);

while((sentence=readFile.readLine())!=null)

{

System.out.println(sentence);

}

readFile.close();

in.close();

}

catch (FileNotFoundException e )

{

System.out.println(" File does not exist");

System.err.println("FileNotFoundException :"+ e.getMessage());

}

catch (IOException e)

{

System.out.println(" Problem reading file");

System.err.println("IOException :"+ e.getMessage());

}

}

}

## 

## Reading the contents of a file containing numbers

When a file contains numeric data, the data must be converted from a sting to a number after it has been read.

|  |  |  |  |
| --- | --- | --- | --- |
| method | format | example | function |
| parseDouble( String test) | variable= Double.parseDouble( String test) | x=Double.parseDouble(score); | Returns the double value of a string. |
| parseInt( String test) | variable=Integer.parseInt( String test) | answer=Integer.parseInt (choice); | Returns the integer value  of a string. |

### Example Program

Use the following code to read test scores and determine the average. First create the file scores.dat in notepad and enter a collection of scores, one per line.

import java.util.Scanner;

import java.io.\*;

public class AvgScore

{

public static void main (String[] args)

{

File textFile= new File("c:\\scores.dat");

FileReader in;

BufferedReader readFile;

String score;

double average,total=0;

int number=0;

try

{ in = new FileReader(textFile);

readFile= new BufferedReader(in);

while((score=readFile.readLine())!=null)

{ number++;

System.out.println(score);

total=total+Double.parseDouble(score);

}

average =total/number;

System.out.println("Average= "+average);

readFile.close();

in.close();

}

catch (FileNotFoundException e )

{ System.out.println(" File does not exist");

System.err.println("FileNotFoundException :"+ e.getMessage());

}

catch (IOException e)

{ System.out.println(" Problem reading file");

System.err.println("IOException :"+ e.getMessage());

}

}

}

## Writing data to a file

To write to a file use the FileWriter and BufferedWriter classes. The FileWriter class is used to create an output file stream. The BufferedWriter class is used to send text into the stream.

|  |  |  |  |
| --- | --- | --- | --- |
| Method | format | example | function |
| FileWriter (filename, boolean append) | FileWriter ( filename,true);  FileWriter ( filename,false); | FileWriter ( in,true);  FileWriter out; | Creates an output stream for the file. If append is true then the new data will be added to the end of the file. If append is false, the old data is overwritten. |
| close() | nameOfFile.close(); | out.close(); | Closes the output file stream. |
| BufferedWriter  Write Stream | BufferedWriter streamName; | BufferedWriter writeFile; | Creates an output stream. |
| write(char c) | nameOfBuffer.write(c); | writeFile.write(letter); | Writes the character to the output stream. |
| write(String str) | nameOfBuffer.write(“words”); | writeFile.write(sentence); | Writes the string to the output stream. |
| newLine() | nameOfBuffer.writeLine(); | writeFile.writeLine(); | Writes a newline character to the output stream. |
| close() | nameOfBuffer.close(); | writeFile.close(); | Closes the output stream. |

### Example Program

This program will ask the use for 5 student names and 5 test scores. The data will be stored in the file StudentScores.dat

**import** java.util.Scanner;

**import** java.io.\*;

**public** **class** CreateDataFile

{

**public** **static** **void** main (String[] args)

{

File dataFile= **new** File("c:\\StudentScores.dat");

FileWriter out;

BufferedWriter writeFile;

Scanner input = **new** Scanner(System.*in*);

**double** score;

String name;

**try**

{

out = **new** FileWriter(dataFile);

writeFile= **new** BufferedWriter(out);

**for** (**int** i=0; i<5; i++)

{

System.*out*.println("Enter student name");

name=input.next();

System.*out*.println("Enter test score");

score=input.nextDouble();

writeFile.write(name);

writeFile.newLine();

writeFile.write(String.*valueOf*(score));

writeFile.newLine();

}

writeFile.close();

out.close();

System.*out*.println("Data written to file");

}

**catch** (IOException e)

{

System.*out*.println(" Problem writing to file");

System.*err*.println("IOException :"+ e.getMessage());

}

}

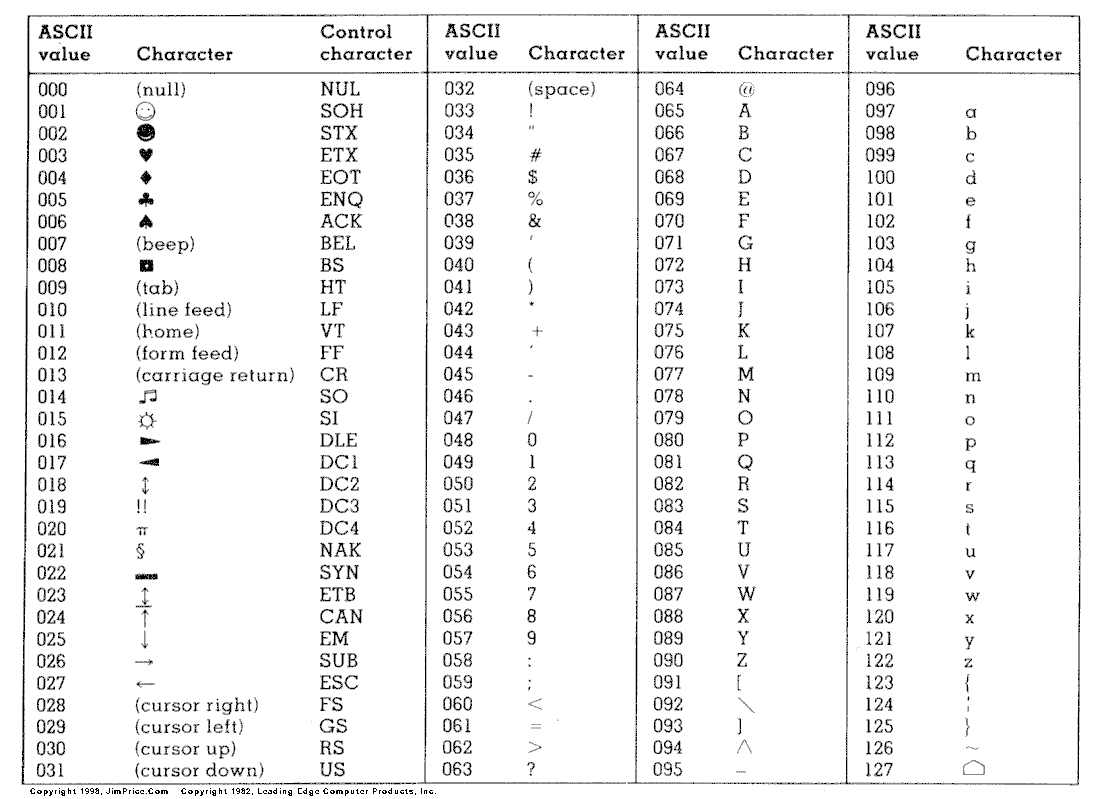
## Appendix A

## ASCII Character Codes

The smallest amount of data that the computer can make use of is the byte (8 bits, a *bit* having the value of zero or one.) The zeroes and ones (binary numbers) that the computer understands is commonly called *machine* language.

Because humans do not normally communicate in zeroes and ones, but use words, the folks at *The American Standards Committee for Information Interchange* (ASCII) have established a standard 'translation' of our alphabet into what the computer does understand. This translation is the infamous ASCII character code.

The majority of the ASCII table contains upper and lower case alphabetic characters, numeric characters, and punctuation marks.

Do note that the lower portion of the ASCII table is devoted to what we call *Control Characters* (Some call these control characters *Escape Sequences.*) These characters are special because they do not represent *symbols* (like our alphabet and punctuation marks) but represent either *actions* (like *line* feed) or states (like Null.) Some of these control characters are reminiscent of how manual type writers work (ie: carriage return and tab.) To learn more about the ASCII codes, do a web search – there are lots of great sites that will tell you lots more.

# Appendix B

## Writing a Program Using Notepad

Use the Windows text editor Notepad to create Java text files.

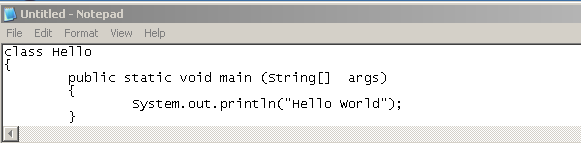
1. Open the text editor by clicking :

**Start- All Programs - Accessories-Notepad**

2. Type in your program. The upper and lower case letters and

punctuation must be correct, while spaces and indentation

are decided by the user.

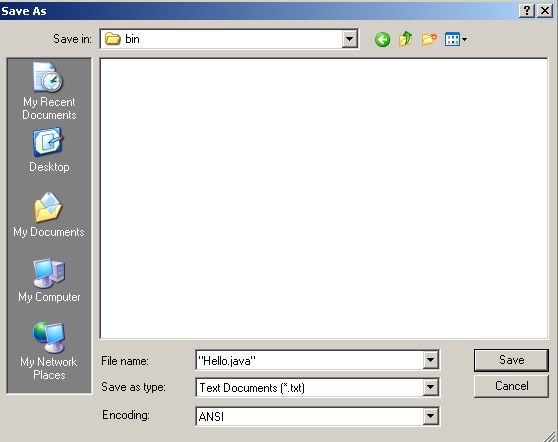


3. Save the file, click **File- Save As**

4. In the **Save In** text box , find the bin folder. In the **File Name** box, type : **“Hello.java”**

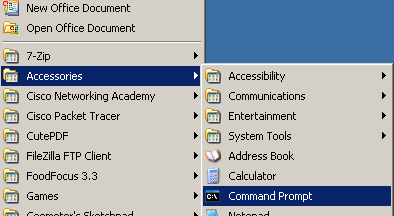
**Hello** matches the name of the class, java is always the ending of the java files and the quotations must be

around the filename.



5.Now click **Save** to save the program.

# 



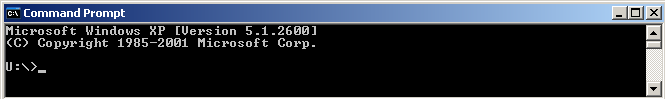
## Compiling and Running a Program in Java

Java programs are run from the Command line.

1. Open the Command line interface by clicking :

**Start-All Programs-Accessories-Command Prompt**

This will open the Command line interface.



1. Change to the bin folder that contains your java program.

For example , use : **C:** to change the drive to the C drive or use : **cd bin** to change to the bin directory

directory. If you are having trouble finding the correct directory, type : **dir** to list the possible directories.



3. To compile the program type : **javac Hello.java**

If the code has been entered properly, you will

now be allowed to run the program.

If there are any errors, the program must be fixed in Notepad, saved and recompiled.

1. To run the program type : **java Hello** and the program should display Hello World.



# Appendix C

## Mazes

Many games can be programmed using arrays to store mazes (PacMan, Snake, Pirate Ships, Wander the World) or grid information ( Battleships). The array stores the information in a chart and the objects are allowed to move according to the numbers in the chart.

Ex. A program can be written that will allow an object to move over 0’s but not 4’s. The 0’s correspond to pathways and the 4’s to walls. You can then add other to numbers to represent other objects. The information is stored in a two dimensional array.

Ex.

**int**[][] map = **new** **int**[][] {{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4},//The map 9 rows 25columns

{4,2,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,4,4,4,4,4,4,4},// 4 = barrier

{4,0,4,0,4,4,4,4,4,4,0,4,1,4,4,4,4,0,4,4,0,0,0,0,0,4},// 3 = gold

{4,0,4,0,4,0,0,0,4,4,0,4,0,0,0,4,4,0,4,4,0,4,4,4,2,4}, // 2 = water

{4,0,4,0,4,0,4,0,4,4,0,4,0,4,0,0,0,0,4,4,0,0,0,4,4,4}, // 1 = fire

{4,0,4,0,4,0,4,0,4,4,0,4,0,4,4,4,4,4,4,4,4,4,0,4,4,4},// 0 = pathway

{4,0,4,0,4,0,4,0,0,4,0,4,0,0,0,0,0,0,0,0,1,4,0,0,0,4},

{4,0,0,0,0,0,4,4,0,0,0,4,4,4,4,4,0,4,4,4,0,0,0,4,3,4},

{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4},

1. This sample program is split into three parts in order to fit on the page. First import all necessary files, create the class variables, create the maze and start the timer.

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.awt.event.KeyListener;

**import** java.awt.event.KeyEvent;

**public** **class** SmallMove **extends** JPanel **implements** ActionListener, KeyListener{

Timer t = **new** Timer(5,**this**); // every 5 milliseconds check for a pressed key

**int** waterCount=0; // count the number of water stops you have reached

**int** gameOver=0; // determine of game has been lost

**int** youWin=0; // determine of game has been won

**int** row = 3; // your location in the maze

**int** column = 5;

**int** x = 40\*(column); // your location on the screen

**int** y = 40\*(row);

**int**[][] map = **new** **int**[ ][ ] {{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4},//The map 17rows 25columns

{4,2,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,4,4,4,4,4},// 4 = barrier

{4,0,4,0,4,4,4,4,4,4,0,4,4,4,4,4,4,0,4,4,0,0,0,0,0,4},// 3 = gold

{4,0,4,0,4,0,0,0,4,4,0,4,0,0,0,4,4,0,4,4,0,4,4,4,2,4}, // 2 = water

{4,0,4,0,4,0,4,0,4,4,0,4,3,4,0,1,0,0,4,4,0,0,0,4,4,4}, // 1 = fire

{4,0,0,0,0,0,4,1,0,0,0,0,0,4,4,4,4,4,4,4,4,4,0,4,4,4},// 0 = pathway

{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}};

**public** SmallMove()//Starts timer

{ t.start();

addKeyListener(**this**); // check to see if a key has been pressed

setFocusable(**true**);

setFocusTraversalKeysEnabled(**false**);

}

**public** **void** KeyTyped(KeyEvent e) {}

**public** **void** KeyReleased(KeyEvent e) {}

@Override

**public** **void** keyReleased(KeyEvent arg0) { }

@Override

**public** **void** keyTyped(KeyEvent arg0) {}

}

1. Then draw the maze and moving circle in the paintComponent. This is also a good place to include instructions and scoreboards.

Ex. **if**(map[i][j] == 1)//Creates red squares

{

g.setColor(Color.*RED*);

g.fillRect(j \* 40, i \* 40, 40, 40);

}

**else** **if**(map[i][j] == 2)//Creates blue square

{

g.setColor(Color.*BLUE*);

g.fillRect(j \* 40, i \* 40, 40, 40);

}

This code will determine what is displayed on the screen depending on what value is stored in the maze. The drawing can be far more complex than a simple colored square.

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

**if** (youWin==1) // print you win or game over

g.drawString("You Win", 400, 400);

**else** **if** (gameOver==1)

g.drawString("Game Over", 400, 400);

**else** // or play the game

{

g.drawString("Capture the gold in the yellow square", 1050, 100); // instructions

g.drawString("The red fire will end the game", 1050, 120);

g.drawString("The blue water will put out the fire", 1050, 140);

**for**(**int** i = 0; i < 7; i++) // draw the maze

{

**for**(**int** j = 0; j < 26; j++)

{

**if**(map[i][j] == 1)//Creates red squares

{

g.setColor(Color.*RED*);

g.fillRect(j \* 40, i \* 40, 40, 40);

}

**else** **if**(map[i][j] == 2)//Creates blue square

{

g.setColor(Color.*BLUE*);

g.fillRect(j \* 40, i \* 40, 40, 40);

}

**else** **if**(map[i][j] == 3)//Creates gold square

{

g.setColor(Color.*YELLOW*);

g.fillRect(j \* 40, i \* 40, 40, 40);

}

**else** **if**(map[i][j] == 4)//Creates gray squares

{

g.setColor(Color.*GRAY*);

g.fillRect(j \* 40, i \* 40, 40, 40);

}

}

}

g.fillOval(x,y,39,39);//Creates you

}// end of game on

}

**public** **void** actionPerformed(ActionEvent e) {

repaint();//redraws maze and player

}

**public** **void** reset()

{

// this is where the code would go to start game over

}

1. The last section of code is the movement of the object. You would then have to write the mainline in a separate file to create the panel and frame.

@Override

**public** **void** keyPressed(KeyEvent e) // controls movement of object

{

**int** code = e.getKeyCode(); // determine which key has been pressed

Graphics g = getGraphics();

**switch**(code)

{

**case** KeyEvent.*VK\_DOWN*: // if the location one down is not a barrier, move

**if** ((map[row+1][column]<4))

{

y=y+40; // update the new location

row = row +1;

}

**break**;

**case** KeyEvent.*VK\_UP*:// if the location one up is not a barrier, move

**if** ((map[row-1][column]<4))

{

y=y-40;// update the new location

row = row -1;

}

**break**;

**case** KeyEvent.*VK\_RIGHT*:// if the location one right is not a barrier, move

**if** ((map[row][column +1]<4))

{

x=x+40;// update the new location

column = column +1;

}

**break**;

**case** KeyEvent.*VK\_LEFT*: // if the location one left is not a barrier, move

**if** ((map[row][column -1]<4))

{

x=x-40;// update the new location

column = column -1;

}

**break**;

**case** KeyEvent.*VK\_SPACE*:

reset(); // reset game after winning or losing

**break**;

}

**if** (map[row][column]==1) // if you have landed on fire, end game

{

gameOver=1;

}

**else** **if** (map[row][column]==2)//if you have landed on water,add to the water count

{

waterCount=waterCount+1;

map[row][column]=0; // set current water to normal square

}

**else** **if** (map[row][column]==3) // if you have landed on gold, win game

{

youWin=1;

}

**if** (waterCount==1) // if you have 1 water, put out 1 fire

map[5][7]=0;

**else** **if** (waterCount==2) // if you have 2 water, put out 2nd fire

map[4][15]=0;

}// end of movement of object

# Appendix D

## Computer Competition Questions

These are sample questions taken from previous computer science competitions.3

1. Safe from Malaria

You have a rectangular piece of mosquito screen with one small hole in it. Its height is 100cm and its width is 50cm. You need a large rectangular piece with no holes. Bring out your scissors. Since this is a mosquito screen, only horizontal or vertical cuts make sense.

Input : Five scenarios, one per line. Each scenario specifies the location of the hole.

On each line, there is a positive whole number ( the distance of the hole from the top of the piece of paper), then one space, and finally a positive whole number (distance of the hole from the left side of the piece).

Output : For each scenario, the area of the largest rectangle without a hole. The output consists of five lines, each containing only a single positive whole number.

Note: The diameter of the hole is negligible.

Example Input : Example output :

50 25 2500

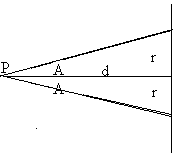
50 25 2500

50 25 2500

50 25 2500

1 1 4950

2. A projector sits on a pedestal and shines toward a wall forming a circular spot of light. Let d be the horizontal distance from the projector to the wall, let r be the radius of the circle, and let A be the angle the light forms with the horizontal.



We want to change the radius of the circle by p percent. For example, if p = -25, then we want to reduce the radius by 25%, if p=50, we want to increase the radius by 50%. Assuming the angle A stays constant, calculate how far, and in which direction, we should move the projector along the horizontal the change the radius by p percent.

**Input** : The first line of input contains a positive integer N specifying the number of data sets that follow. Each data set consists of two lines :

Line 1 contains d, a fractional number between 1 and 40.

Line 2 contains p, a fractional number between -95 and 300

Output : One number for each data set representing the change in the position of the projector. A positive number indicates moving towards the wall and a negative number indicates moving the projector away from the wall.

Ex. d=28.5 and p= -25.0 then the projector is moved 7.125

d=28.5 and p= 50.0 then the projector is moved -14.25

3. Write a program to determine the first 50 safe primes. A safe prime is a prime number of the form 2p+1 where p is also prime.

Ex. We can calculate the first 6 primes and test the formula 2p+1 to see if they produce a safe prime. The first 4 safe primes are 5,7,11 and 23 since 15 and 27 are not prime.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P | 2 | 3 | 5 | 7 | 11 | 13 |
| 2p+1 | 5 | 7 | 11 | 15 | 23 | 27 |

4. Cheapest Supermarket, part1

We have data on the prices of groceries in several different stores. We need to find the store that sells the groceries on our list at the cheapest price.

Grocery List : 10lbs potatoes, 1 dozen eggs, 2 loaves bread and 4 litres of milk.

Input : First, a single line containing the number of stores. Then , one line per store. Each of these lines contains four positive whole numbers, A,B,C and D, separated by spaces. Each number represents the price ( in cents) of :

1. potatoes, per lb (for number A)

2. eggs, per dozen ( for number B)

3. a single loaf of bread (for number C)

4. a litre of milk ( for number D)

Output : An indicator of the cheapest store, and the costs ( in cents) for the items at the store. Use the same format as the example output.

Example input :

3

2 2 2 2

1 1 1 1

3 3 3 3

Example output :

Store 2 at a cost of 17 cents.

5. Where in Floyd are we ?

Floyd’s triangle is built by filling rows of different sizes with consecutive integers, forming a triangle. The starting number is 1, and it is the only element of the first row. Each row thereafter will have one more number than the previous row, and will contain the next consecutive numbers. An illustration of Floyd’s triangle up to the 4th row is shown below.

1

2 3

4 5 6

7 8 9 10

Your task is to implement a program that will give the row and the column position for the three different integers in Floyd’s triangle.

Input : The input consists of 3 positive whole numbers in one line. They are separated by spaces.

Output : Three lines, each indicating the row and column position of the given number in Floyd’s triangle. The format of each line is as follows:

R, then the row number, then a space, then C, then the column number

Example input : 5 9 8

Example Output :

R3 C2

R4 C3

R4 C2

6. Write a program to decode a confidential message. The original message consists of printable ASCII characters (the characters with ASCII values in the range 32 to 126). The successor of a character is the character immediately after it and the predecessor of a character is the character immediately before it. Ex. B is the successor of A. Y is the predecessor of Z. The only exception is the successor of character 126 is 32 and the predecessor of 32 is 126.

The encrypt the message, the message is first partitioned into two rows. The first row contains all the odd characters and the second row contains all the even characters. Each character in the first row is replaced with its successor and each character in the second row is replaced with its predecessor.

Ex. Original Message “Hello, Paul. This is John.”

row 1 : “Hlo al his on” which is encrypted as “Imp!bm!itj!po”

row 2: “el,Pu.Ti sJh.” which is encrypted as “dk+Ot-Sh~rIg-“

Start with the two coded lines and print out one line of decoded text.

# Assignments

Note:

* Assignments are worth **1 mark each** unless specified otherwise.
* Assignments worth more than 1 mark will have a mark value indicated by a number in a box to its left.
* Please save a copy of your assignments in your CS 120 folder using the filename indicated

Assignment #1 (simple output)

1. **filename:** Hello.java

Open Notepad and enter the following code. Run the program to make sure that it works. ***Change the program*** so that it says hello to you.

class Hello

{

public static void main (String[ ] args)

{

System.out.println("Hello World");

}

}

Assignment #2 (output and formatting)

**2.1 filename:** Assignment2A

Write a program that will display your name, grade and favourite activity.

Ex. My name is Ms. Duffy.

I teach grade 11 and grade 12.

My favourite activity is cross country skiing.

**2.2 filename:** Assignment2B

Write a program that will create a chart to display 3 students and the marks for each student in all 5 periods. Make sure the chart is aligned properly.

Ex. Student Name Period 1 Period 2 Period 3 Period 4 Period 5

Amy Smith 98 87 78 95 85

Bob Jones 87 88 89 87 83

Carmen Pike 78 77 76 75 95

**2.3 filename:** Assignment2C

Write a program that will display a decimal number to different levels of significance. The program should display the same number first with only one digit after the decimal, then two digits,…, up to 5 digits after the decimal. Use the printf command to achieve the proper formatting.

Ex. Number = 127.5192683

One decimal 127.5

Two decimals 127.52

Three decimals 127.519

Four decimals 127.5193

Five decimals 127.51927

**2.4 filename:** Assignment2D

Write a program that will calculate the discount on an item. The program should show the original price, the discount and the new price. Use the NumberFormat class to achieve the proper formatting.

Ex. Original Price $127.36

Discount 30%

New Price $89.15

Assignment #3 (Input, Math operators and variables)

1. **filename:** Assignment3A

Input 2 integer numbers, calculate and display the sum, product, quotient and difference. Declare all your variables, ***except*** your variable for the quotient, as type integer. ***Test your program with*** the following numbers :

1. 10 and 2 b) 8 and 3

Ex. Please enter the first number 8

Please enter the second number 3

The sum of 8 and 3 is 11

The product of 8 and 3 is 24

The difference of 8 and 3 is 5

The quotient of 8 and 3 is 2.6666666666666665

**Note** : When you have errors with the quotient in part b), remember to CAST your variables

**3.2 filename:** Assignment3B

Input a number and determine the square, cube and fourth power. Use Java's **pow** function for exponentiation

Ex. Please enter a number 2.5

The square of 2.5 is 6.25

The cube of 2.5 is 15.625

The fourth power of 2.5 is 39.0625

3.3 **filename:** Assignment3C

Calculate the average of 5 numbers. ***Test your program*** on the following data :

a) 1,2,3,4,5 (average = 3) b) 1,2,3,4,6 (average = 3.2)

3.4 **filename:** Assignment3D

Input the radius of a circle and calculate the area. A = πr2 .

Ex. Please enter a radius 10

The area of a circle with radius 10 is 314.159265358793

**3.5 filename:** Assignment3E

3

Write a program to evaluate some formula from another class. You will be marked on the level of difficulty of the formula you choose ( 1-basic math, 2-squares and square roots, 3-trigonometry), your appropriate use of any java math functions the formula might need, and the format of the output. A logical , easy to follow program with well organized output is worth full marks.

Ex. This program will calculate the hypotenuse of a right triangle and the two acute angles.

Please input the first side 3

Please input the second side 4

The two acute angles are 36.8699 and 53.1301 \* the computer works in radians. Convert your angles to

degrees using the toDegrees function..

Assignment #4 (Loops)

1. **filename:** Assignment4A

Use a loop to print the numbers 1 to 100.

1. **filename:** Assignment4B

Use a loop to find the sum of the first 50 numbers.

1. **filename:** Assignment4C

Use a loop to print the square roots of the whole numbers from 50 to 70, rounded to exactly five digits after the decimal place, one per line and justified left.

1. **filename:** Assignment4D

Use a loop to calculate the squares and cubes of the numbers from 1 to 10. Print the results in table format.

Ex. Number Square Cube

1 1 1

2 4 8

3 9 27

………….

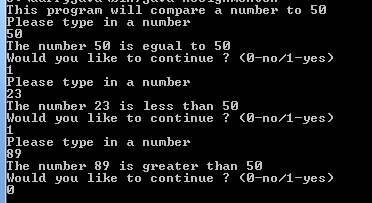
1. **filename:** Assignment4E

Input a number less than 10. Use a loop to keep multiplying the number by 2 until the number is greater than 1000. Print the number of times you were able to multiply by 2 before the number gets too large.

Assignment #5 (Decision Structures)

1. **filename:** Assignment5A

Input a number. Determine if the number is bigger or less than 50. Have the program repeat until the user decides to stop.



1. **filename:** Assignment5B

Input two numbers. Ask the user if they would like to add or multiply the two numbers. Use a **switch** statement to handle the multiply or add option. Print the result. Repeat the program until the user decides to stop

* 1. **filename:** Assignment5C

Design a game to guess a number from 1 to 100. Have the program continue until the user guesses the correct number. Give hints such as “guess higher” and “guess lower” when the user guesses wrong. Print the total number of guesses that the user needed to get the correct answer. Repeat the program.

* 1. **filename:** Assignment5D

Design a game of your own.

5

Assignment #6 (Modular Division)

* 1. **filename:** Assignment6A

Input a number and determine if it divisible by 2. (Hint: use modular division). Repeat the program.

* 1. **filename:** Assignment6B

Modify the previous program to input two numbers. Determine if the first number is evenly divisible by the second. Have the program repeat until the user decides to stop.

* 1. **filename:** Assignment6C

Input a number and determine the sum of all the **digits** of the number.

* 1. **filename:** Assignment6D

Determine the sum of all the **digits** of the numbers from 1 to 100. Print the sum of the digits of each number on a separate line.

Ex. The sum of the digits of 11 is 2

The sum of the digits of 12 is 3

The sum of the digits of 13 is 4

* 1. **filename:** Assignment6E

Determine if a number has exactly one more 1 than 2 in their decimal representation.

Ex 1, 10, 13, 14, 15, 16, 17, 18, 19, 31, 41, 51, 61, 71, 81, 91, 100, 103,…121,…11122678 all have one more 1 than 2

* 1. **filename:** Assignment6F

Determine the **number of integers** between 1 and n that have exactly one more 1 than 2 in their decimal representation.

Ex. n = 103 , there are 18 numbers with one more 1 than 2

1, 10, 13, 14, 15, 16, 17, 18, 19, 31, 41, 51, 61, 71, 81, 91, 100, 103

The program should work for a positive integer n that is less than or equal to 9999.

* 1. **filename:** Assignment6G

A prime number, is a positive integer greater than 1 that has only two positive factors : 1 and p.

Write a program to determine if a number is prime. Have the program repeat until the user decides to stop.

* 1. **filename:** Assignment6H

Write a program that inputs an integer n and outputs a list of the prime numbers that are factors of n.

Assignment #7 (One Dimensional Arrays)

* 1. **filename:** Assignment7\_1

Create an array to accommodate 10 numbers. Allow the user to input the numbers (the user may enter as few as one to as many as 10 number.) Determine the mean of the numbers. Print the numbers and the mean. Allow the user to repeat the program as often as they wish.

Test your program on the following data :

a) 1,2,3,4,5 mean = 3

b) 1,2,3,4,6 mean = 3.2

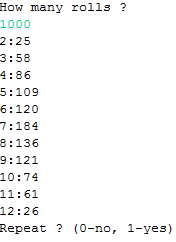
c) 9,10 mean = 9.5

* 1. **filename:** Assignment7\_2

Store the following information in 3 arrays (one array for the planet name, one array for the moons, and one array for the distance). Design a menu that allows the user to choose a planet and view the information associated with that planet. Allow the user to repeat the program as often as they wish.

|  |  |  |
| --- | --- | --- |
| Planet | Moons | Distance |
| Mercury | 0 | 58 |
| Venus | 0 | 108 |
| Earth | 1 | 150 |
| Mars | 2 | 228 |
| Jupiter | 16 | 778 |
| Saturn | 18 | 1427 |
| Uranus | 15 | 2869 |
| Neptune | 8 | 4498 |
| Pluto | 1 | 5900 |

* 1. **filename:** Assignment7\_3



Write a program to simulate the rolling of two dice. A roll is simulated by

generating a random number from 1 to 6. The sum of the two numbers

should be calculated. Each die can show a number from 1 to 6 so the

totals will vary from 2 to 12.Use a one dimensional array to tally the

number of times each possible sum appears. Ask the user for the number

of rolls and then display the frequency count.

Allow the user to repeat the program.

Assignment #8 (Two Dimensional Arrays)

**8.1filename:** Assignment8\_1

Store the 5 marks for each of the 5 students in a two dimensional array. Create a menu that allows the user to view the data and make any changes. Allow the user to change the data as often as they wish. Show the updated information when the user finishes making changes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A- Amy | B- Bill | C- Carmen | D- Dave | E- Ellen |
| Period 1 | **95** | **56** | **67** | **89** | **78** |
| Period 2 | **91** | **59** | **63** | **82** | **80** |
| Period 3 | **97** | **55** | **61** | **85** | **72** |
| Period 4 | **88** | **67** | **69** | **81** | **76** |
| Period 5 | **94** | **62** | **58** | **91** | **87** |

**8.2filename:** Assignment8\_2

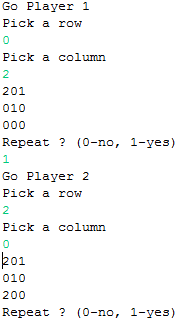
Create a two dimensional array of size 3 x 4. Initialize the array with the following

numbers :

Student Test #1 Test#2 Test#3 Test #4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 87 | 73 | 88 | 92 |
| 2 | 65 | 57 | 69 | 60 |
| 3 | 52 | 57 | 62 | 63 |

Determine the average of each test and each student.



**8.3filename:** Assignment8\_3

Create a very basic tic tac toe game using a 3x3 array.

This is a two player game where each player must specify a

row and a column and the appropriate marker is placed in that row

and column. You may assume that players are smart enough to not

ask for an already occupies space. At the end of each move,

display the board.

**8.4filename:** Assignment8\_4

Create a two dimensional array of maximum size three by ten.

Ask the user how many numbers they would like to use (no more than 10).

Determine the standard deviation of these numbers. Print the calculations in a

well organized chart. The array should contain the numbers, the differences

from the mean and the squares of the differences from the mean.

To determine the standard deviation :

1. Input the numbers
2. Determine the mean
3. Determine the difference between the mean and each number
4. Square the differences
5. Find the average of the squares
6. Calculate the square root of the average

Test your program with the following numbers :

a) 1,2,3,4,5 standard deviation = 1.414 b) 1,2,3,4,6,7,8,9 standard deviation = 2.7386

|  |  |  |
| --- | --- | --- |
| Number | Difference | Square |
| 1 | -2 | 4 |
| 2 | -1 | 1 |
| 3 | 0 | 0 |
| 4 | 1 | 1 |
| 5 | 2 | 4 |

Ex

Mean = 3

Standard deviation = 1.414

**8.5filename:** Assignment8\_5

Write a program to display the first 8 rows of Pascal’s triangle. Use an array of size eight by eight. To create Pascal’s triangle each row starts and ends with a 1. The middle terms are found by adding the two numbers above.

Computer Output:

Example of first six rows of Pascal's triangle.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  1 1  1 2 1  1 3 3 1  1 4 6 4 1 |  |  |  |  | 1 | |  |  |  |  |  |
|  |  |  |  | 1 | | 1 | |  |  |  |  |
|  |  |  | 1 | | 2 | | 1 | |  |  |  |
|  |  | 1 | | 3 | | 3 | | 1 | |  |  |
|  | 1 | | 4 | | 6 | | 4 | | 1 | |  |
| 1 | | 5 | | 10 | | 10 | | 5 | | 1 | |

Assignment #9 (Methods)

* 1. **filename:** Assignment9\_1

Write a program that will input two numbers. Print the sum, product, quotient and difference. Your program will have six functions:

1) one to input a number

2) four, to calculate each of the results and

3) one to print the results in a well organized fashion.

Allow the user to repeat the program.

* 1. **filename:** Assignment9\_2

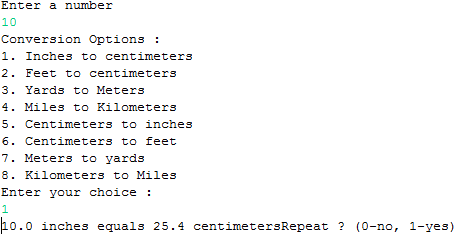
Write a program that will convert a temperature from Fahrenheit to Celsius or Celsius to Fahrenheit . The conversions must be done in methods.

(Note: C = 5 (F-32)/9 *where* C = degrees Celsius and F = degrees Fahrenheit

* 1. **filename:** Assignment9\_3

The following formulas can be used to convert English units of measurement to metric units

Inches \* 2.54 = centimeters



Feet \* 30 = centimeters

Yards \* 0.91 = meters

Miles \* 1.6 = kilometers

Create a program that displays a menu of conversion

choices and then prompts the user to choose a

conversion. The program should have separate

methods for each conversion while the mainline

contains the menu.

* 1. **filename:** Assignment9\_4



2

The area of a triangle can be calculated using the formula:

where : a, b and c are the lengths of the triangle's sides



s, the semi-perimeter, is given by the formula

* + - 1. Write a program using one method to determine the area of a triangle.

Test your program with the following :

i) a = 3, b = 4, c = 5 area = 6

ii) a = 7, b = 8, c = 9 area = 26.8328

1. Modify your program so that it gives an error message if the user types in three side-lengths which will **not** produce a triangle.

Note : A triangle can not be produced if any side is longer than the combination of the other two sides.

Test your program with the following :

i) a = 5, b = 7, c = 13 no solution

ii) a = 5, b = 7, c = 12 area = 0

Project #1

10

Use functions to program the *Game of Life*. This game was invented by the mathematician John Horton Conway. The *Game of Life* consists of a world made up of rows and columns, with *a being* possibly living in any given *cell*.

This make-believe world is dynamic – it changes all the time. Depending on the cells immediately adjacent to being, that being may *die* or may continue on happy as a clam. As well, a new being may be *born* in an unoccupied cell, again depending upon the status of the cells immediately adjacent to that unoccupied cell. Once a being dies, that cell is considered unoccupied.

Of course, given a birth in a cell, the status of other cells may change. To start the 'game', you, the programmer, populate some of the cells with 'beings'. Then, with well formatted outputs, you watch as 'life' happens in this world.

Now for some specifics:

The world is a two dimensional array that is 22 rows by 80 columns. (This size has been determined by the size of the output window and is thus arbitrary.)

Each cell of the array can hold one being.

1. Mark an occupied cell with an asterisk '\*' ( or something better if you want) and
2. Mark an empty cell with a blank ' '.

Life in this array will follow the following set of rules :

1. Each cell (except the ones on the edge of the array) has 8 neighbour cells. For example:

|  |  |  |
| --- | --- | --- |
| Neighbour 1 | Neighbour 2 | Neighbour 3 |
| Neighbour 4 | \* | Neighbour 5 |
| Neighbour 6 | Neighbour 7 | Neighbour 8 |

2. If an occupied cell has 0 or 1 neighbours, it dies of loneliness. For example:

|  |  |  |
| --- | --- | --- |
| (0) ✞ |  |  |
|  |  | (2) ♥ |
| (1) ✞ | (3) ♥ | (2) ♥ |

Dies of loneliness ✞

Completely happy ♥

3. If an occupied cell has more than 3 neighbours, it dies of overcrowding.

|  |  |  |
| --- | --- | --- |
| (3) ♥ | (4) ✞ |  |
| (4) ✞ | (6) ✞ | (4) ✞ |
|  | (4) ✞ | (3) ♥ |

Dies of overcrowding ✞

Completely happy ♥

4. If an unoccupied cell has exactly 3 neighbours, a birth takes place and the cell is now occupied.

|  |  |  |
| --- | --- | --- |
| ♥ | ♥ |  |
| ♥ | (3) ◇ |  |
|  |  |  |

Completely happy ♥

Birth takes place ◇

5. Births and deaths are instantaneous. A new born being can not save a cell that was dying of loneliness

Have the program repeat for many generations to watch the changes that take place. Try various starting configurations.

Life will be evaluated as follows :

1. Proper indentation, logical variable names, efficient use of functions and good use of comments.
2. Display the world.
3. Count the neighbours.
4. Determine the births and deaths.
5. Display the next generation.
6. Repeat for a number of generations.
7. Start with a random population.
8. Allow user to choose between different starting populations.
9. Allow user to create starting population ( Ex. could be based on density).
10. Add something new and useful.

Check your program with :

a) b)

1st generation

\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\*

2nd generation \* \*\*\* \*\*\* \*\*\*

\* \*\*\* \*\*\* \*\*\*

\* \*\*\* \*\*\* \*\*\*

3rd generation \*\*\* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \*

4th generation \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \*\* \*\* \*\* \*\* \*\* \*\*

\* \* \* \* \* \* \* \* \*

\* \* \*

5th generation  **\* \* \* \*\*\* \*\*\* \*\*\***

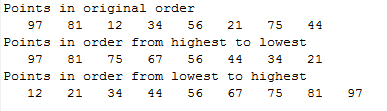
**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\*\*\* \*\*\* \*\*\***

# Assignment #10 (Sorting)



**10.1filename:** Assignment10A

Write a program to sort the following data :

97 81 12 34 56 21 75 44

from smallest to largest and then from largest to

smallest. Display both sets of sorted data.

**10.2filename:** Assignment10B

Write a program to sort the following data according to the numbers of goals scored, points and then according to alphabetical order.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | [Player](http://www.nhl.com/nhlstats/app?fetchKey=20083ALLSASAll&page=Stats&service=page&sort=player.properName&viewName=points) | [Team](http://www.nhl.com/nhlstats/app?fetchKey=20083ALLSASAll&page=Stats&service=page&sort=team.displayAbbrev&viewName=points) | [G](http://www.nhl.com/nhlstats/app?fetchKey=20083ALLSASAll&page=Stats&service=page&sort=goals&viewName=points) | [A](http://www.nhl.com/nhlstats/app?fetchKey=20083ALLSASAll&page=Stats&service=page&sort=assists&viewName=points) | **P** |
| 1 | [Henrik Zetterberg](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8468083) | DET | 11 | 10 | 21 |
| 2 | [Sidney Crosby](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8471675) | PIT | 4 | 17 | 21 |
| 3 | [Marian Hossa](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8466148) | PIT | 9 | 10 | 19 |
| 4 | [Pavel Datsyuk](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8467514) | DET | 9 | 10 | 19 |
| 5 | [Evgeni Malkin](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8471215) | PIT | 9 | 10 | 19 |
| 6 | [Mike Ribeiro](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8467371) | DAL | 3 | 14 | 17 |
| 7 | [Daniel Briere](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8464975) | PHI | 9 | 7 | 16 |
| 8 | [Johan Franzen](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8471309) | DET | 12 | 3 | 15 |
| 9 | [R.J. Umberger](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8469469) | PHI | 10 | 5 | 15 |
| 10 | [Brenden Morrow](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8466160) | DAL | 9 | 6 | 15 |
| 11 | [Ryan Malone](http://www.nhl.com/nhl/app?service=page&page=PlayerDetail&playerId=8467988) | PIT | 6 | 9 | 15 |

# Assignment #11 (Strings)

**11.1filename:** Assignment11A

Write a program that will input two words and print them in alphabetical order. Allow the user to type in the words as often as they want.

Test your program on the following data :

* + 1. cat, dog
    2. alphabet, alpha
    3. cat, Cat
    4. cat, DOG

**11.2filename:** Assignment11B

Write a program that will input a sentence and print it out backwards. Allow the user to type in the sentences as often as they want.

* 1. **filename:** Assignment11C

Write a program that will input a sentence and change it according to the following rules: replace every instance of “FOO” with “OOF” Make sure your program goes through the string as many times as necessary.

Test your program with : FOOOOOPLE

Your output should be : OOOOFOPLE

* 1. **filename:** Assignment11D

Input a string of 1’s and 0’s and determine the number of runs. A “run” is defined as a string of characters with the following properties : 1/ All of the characters are the same

2/ It is not part of a longer run

Ex. AAAABBBAAAA contains 3 runs AAAA, BBB and AAAA

Test your program with : You output should read :

a) 011000111100000111111 this string contains 6 runs

b) 111100 this string contains 2 runs

* 1. **filename:** Assignment11E

Write a program that will input a sentence and count the number of times each letter appears in the sentence. Upper and lower case of the same letter are to be considered the same.

***Example:***  User inputs: "See Spot run."

Program outputs: Letter # of Occurrences

==================

e 2

n 1

o 1

p 1

r 1

s 2

t 1

u 1

Test your program on the following sentence :

The quick brown fox jumps over the lazy dog .

Note : Each letter of the alphabet will be used at least once

# Assignment #12 (Classes)

**12.1 filename:** Assignment12A

Write a program using a class to store data on a single athlete. Record the player’s first name, last name, goals and assists. Write functions to input the data, display the data and update the data.

**12.2 filename:** Assignment12B

Write a program to simulate a bank machine. Use a class to store information from a bank card .Use a menu system to view and change information. The program should allow the changes that a bank machine would make.

Ex. Check balance, deposit money, withdraw money.

# 

# Assignment #13 (Graphics)

**13.1filename:** Assignment13A

Write a program that will use dialog boxes to ask the user the day of the week and the month. Display the information on the screen.

**13.2filename:** Assignment13B

Write a program that will use dialog boxes to ask the user for two numbers. Display the sum, product, quotient and difference on the screen.

**13.3filename:** Assignment13C

Write a program that uses loops to display line art.

**13.4filename:** Assignment13D

Write a program that will draw a circle, square, diamond, rectangle and triangle. Use different colors for each object.

**13.5filename:** Assignment13E

Modify your previous program and ask the user for a choice of which object they would like to see. Display the object of their choice.

**13.6filename:** Assignment13F

Use random numbers to draw Serpinski’s triangle.

**13.7filename:** Assignment13G

Write a program that will draw a rectangle on the screen. When the user clicks inside the rectangle , print the word “HIT” on the screen. When the user clicks outside of the rectangle, print the word “MISS” on the screen.

**13.8filename:** Assignment13H

Modify your program from 13D. Print the words circle, square, diamond, rectangle and triangle. When the user clicks on the word, the specified shape should be drawn on the screen.

Hint : Put the code for the drawing of the shapes in the paintComponent method.

In the mouseClicked method , use the command repaint(); to get the program to draw the new object.

# 

# Assignment #14 (Animation)

**14.1filename:** Assignment14A

Write a program that will move an object across the screen using arrow keys. You object should have some detail and be far more interesting than a circle or square.

**14.2filename:** Assignment14B

Modify your previous program by adding a continually moving object. Use a hit test to determine if your object has collided with the new object.

**14.3filename:** Assignment14C

Write a program that will load one image when you click inside a certain location on the screen. Load a different image when you click somewhere else.

Ex. Battleships – load a ship or an X for a miss

# Assignment #15 (Files)

**15.1 filename:** A15A

Write a program that will open a text file for input. Open the file and convert the contents to upper case. Print the information on the screen.

**15.2 filename:** A15B

Write a program that will open a data file for input. Find the largest and smallest integer and write the numbers on the screen.

Test your program with the following data : 5, 9, 7, 10, 2, 4, 1, 6, 3, 8. Each number must be on its own line in the file.

**15.3 filename:** A15C

Open the file from the last program and determine the average of the numbers. Display the numbers and the average on the screen.

**15.4 filename:** A15D

Open a data file for ouput. Ask the user for 5 names and 5 scores ( 1 score for each name). Write the

information in the file in an organized manner.

**15.5 filename:** A15E

Open a file containing the data from the last program. The data must be stored with each name and each score on its own line. Sort the data from smallest to largest score. Display the data in an organized fashion on the screen. Include the names with the scores.

## Final Project

20

Write a program using graphics to display some concept in mathematics or physics or create a game.

**Some Ideas for Projects**

* 1. The program will draw two lines and find the points of intersection.
  2. The program will calculate the area under a curve.
  3. The program will draw a parabola and determine max/min point and x-intercepts.
  4. The program will draw a fractal or Julia set.
  5. Create a game such as pacman, snake, battleships, space invaders,…