Introduction to Computer Science - University Preparation

ICS3U-Grade 11

Introduction - THE PROGRAMMING PROCESS

Introduction.1 Compilers, Interpreters and all that

Programming is at the heart of the functioning of computer systems. What makes a computer unique from other machines is that it is *programmable*, that is, its internal state can be set to perform particular tasks based on an instruction set we call a *program*.

Two types of computers rise out of programmable machines: *General Purpose* and *Specific Purpose.* General Purpose computers are machines that are fully programmable for an indefinite number of tasks. General Purpose programming languages such as Java, C, Pascal, Turing, are used with this end in mind. Desktop computers are General Purpose computers. Specific Purpose computers are pre-programmed computers with a specific goal. These machines have been optimized for speed and space and do not need to do more than perform the one needed task. Image processors, printers, cell phones, video game consoles, are examples of Specific Purpose computers. Specific Purpose languages are used for these machines. Examples of such languages are PostScript for printers, or DataCube for image processing.

At the heart of programming instructions is binary code. All electronic computer systems ultimately work with instructions coded in binary. In the development of computer systems (1930‘s onward) all coding was initially done in binary, thereby making programming very difficult and tedious. Though computer systems are still designed to work with binary instructions, their evolution over time has allowed programmers to use instructions that are more meaningful to humans. These programs are written in languages that are English-like, making it possible to code very sophisticated and abstract problems and solutions. We call these programs the *source code*.

A program written in source code needs to be translated into binary instructions before it can be run by a computer. This translation is done by yet another program. This other program comes in two flavours: *Interpreter* or *compiler.* The interpreter translates one instruction of the source code and immediately runs the translated instruction. Then the next instruction from source code is translated and run. This process continues until the end of the source code is reached. In contrast, the compiler translates the entire source code once and then the *compiled program* is run by the computer without any further aid from the compiler. The compiled program is known as the *executable* and it is made of *binary code*, also called *machine code*. It contains only ones and zeroes.

The interpreter method requires that during the running process both the source code and the interpreter be present in the computer's memory. With the compiler method, once the source code has been compiled, the resulting executable can be installed and run in any machine without the compiler being present. Programs such as Word, PowerPoint, Halo, Windows 7, and thousands of others are executables that have been compiled from source code. They have been written in languages such as C, C++, Java, Pascal, etc. but the manufacturers do not release the source code, only the executables.

Introduction.1 Exercises

1. Why is a printer considered a *Specific Purpose Computer*?

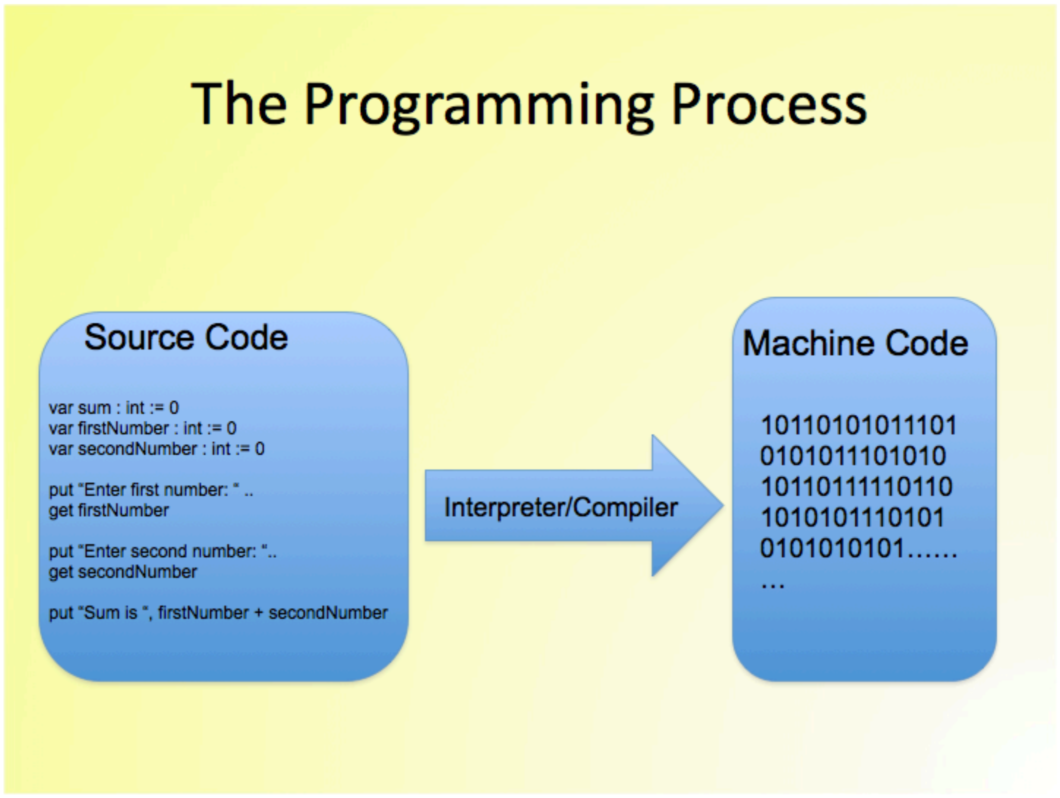
2. In this section we learned the difference between an interpreter and a compiler. Research further on these two types of programs and write down a situation in which it would be more convenient to use an interpreter instead of a compiler. Then write a situation in which it would be more convenient to use a compiler instead of an interpreter.

3. Why do you think that companies that make software do not release their source code?

4. Research what is meant by *Open Source*

Introduction.2 From source code to binary code

This example shows the *standard programming process*. Source code of a Turing program is shown here:



```
var sum : int := 0
var firstNumber : int := 0
var secondNumber : int := 0

put "Enter first number: " ..
get firstNumber

put "Enter second number: "..
get secondNumber

put "Sum is ", firstNumber + secondNumber
```

```
10110101011101
0101011101010
10110111110110
1010101110101
0101010101......
...
```

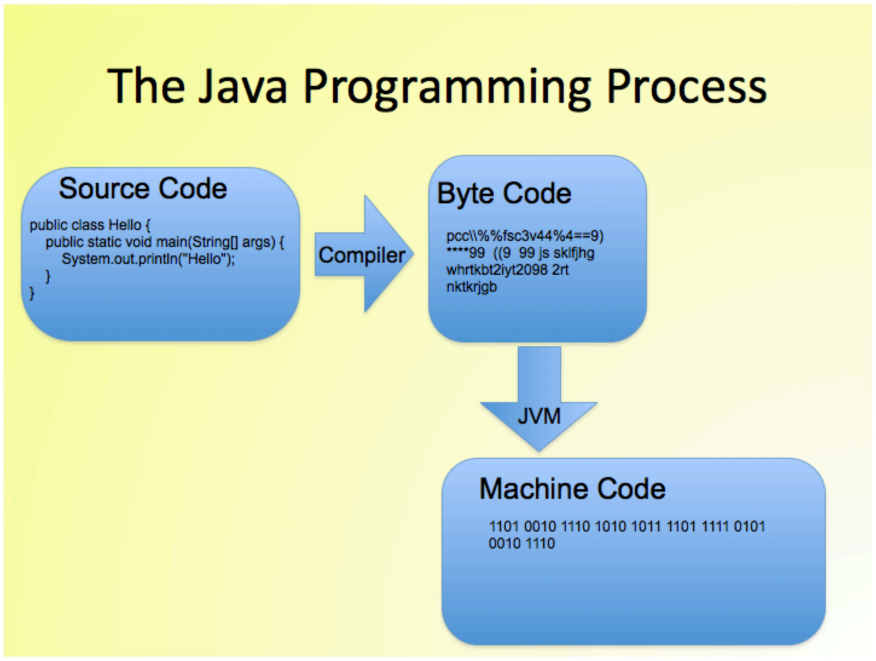
The binary code that is generated from this process is specific to a particular computer processor. A compiler will generate binary code that will run, for example, on only Intel processors. If we wish to run a program in a different processor, say a Motorola, we need to compile our source code again using a compiler that will translate for a Motorola processor.

Because of this, binary code is limited to a particular type processor. With the advent of the internet, programs need to run on computers for which we do not know the processor. This is because programs are sent over the internet lines and they need to run on the machine where they are being sent. There is no way of knowing in what type of computer a program may end up once it is sent over the net. This creates a big problem, but it was solved in the late 1990s with the invention of the *Virtual Machine*.

A Virtual Machine is a *program that simulates a computer*. In other words, with a Virtual Machine we can make a computer behave like a different computer. For example, we can make an Intel processor behave like a Motorola processor. In this way, if a program that has been compiled for an Intel processor needs to run on a Motorola processor, we use a Virtual Machine to make Motorola processor behave like an Intel processor and thus we are able to run our programs. This is how we can run Windows programs on a Mac. Software like Wine or VMFusion are examples of virtual machines.

When we compile programs using Java, we are not generating binary code, but *byte code.* The byte code *does not run directly on the computer* but runs *on top of the Java Virtual Machine (JVM)*. Any computer on the internet that is Java-enabled has a Java Virtual Machine installed, which means that byte code can run on that computer *regardless of the type of processor*. The JVM guarantees that the instructions contained in the byte code always work, no matter what computer is being used. The JVM allows our programs to run on computers without our knowing the type of processor those computers have.

The following diagram shows the Java Programming Process. Java source code compiles to Byte Code which then runs on a virtual machine, and not directly on the processor of the computer.

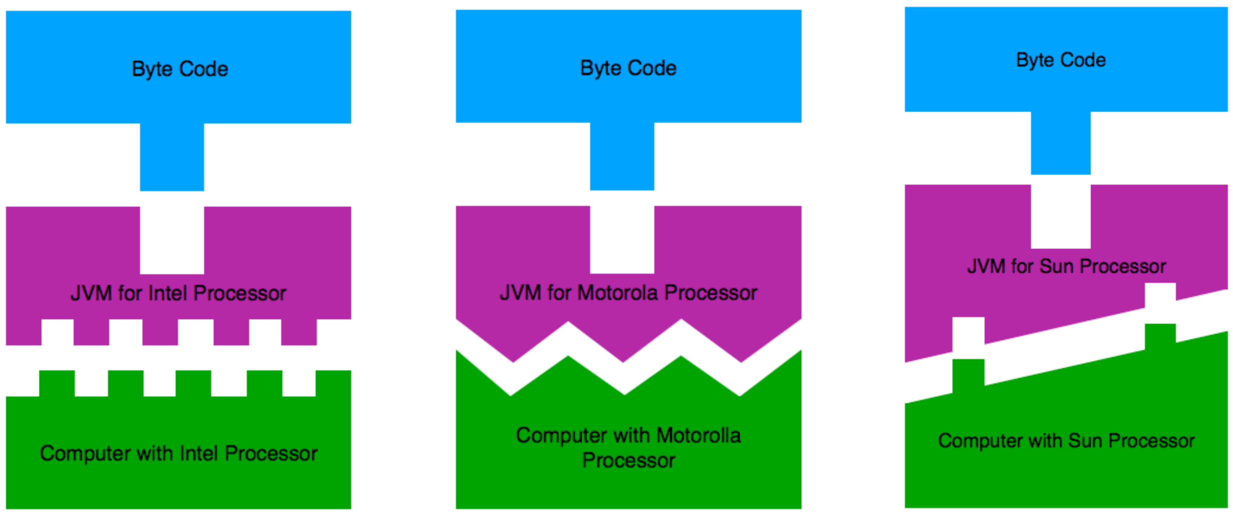

The Java Programming Process

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

```
pcc\\%%fsc3v44%4==9)
****99  ((9  99 js sklfjhg
whrtkbt2iyt2098 2rt
nktkrjgb
```

```
1101 0010 1110 1010 1011 1101 1111 0101
0010 1110
```

The following visual analogy, as conceived by Alexander Mirabella, demonstrates how the *same* byte code interacts with JVMs that have been designed to work with three different computer processors[1]:


Byte Code
JVM for Intel Processor
Computer with Intel Processor
Byte Code
JVM for Motorola Processor
Computer with Motorolla Processor
Byte Code
JVM for Sun Processor
Computer with Sun Processor

Introduction.2 Exercises

1. Define the following:
    a. Binary code
    b. Machine code
    c. Assembly code
    d. Byte code
    e. Source code

2. Define the following:
    a. Instruction set
    b. Virtual machine
    c. ALU
    d. RISC
    e. JVM

3. Why is it that programs compiled for an Intel processor cannot run on a different type of processor?

[1] Visual representation by Alexander Mirabella, 2016. Used with permission.

4. Put the following in order, from the furthest to the closest to machine code: Executable, Source Code, Compiler, JVM, Byte Code

5. What is a general purpose programming language? Give some examples

Introduction.3 Generations of Programming Languages

Over the past eighty years of computer development, programming languages have gotten more and more sophisticated. These languages have been categorized in generations as follows:

Generation	Characteristics	Uses	Example
First	Low-level Controls hardware directly	Machine code	Binary code `1001 0001 1001 1101`
Second	Low-level Controls hardware indirectly through machine code	Mnemonics	Assembly language `MV 3, 10`
Third	High-level and abstract Can control hardware through operating system General purpose	English-like language	C, C++, Java, Turing, Pascal, PL/1 `printf("%d", 35);`
Fourth	High-level and abstract Specific purpose Removed from hardware	English-like language	SQL (Structured Query Language) `list all entries with mark > 72`
Fifth	High-level Mathematical General purpose Self-generating Conceptually based	English-like language	PROLOG, Lisp

1 - FUNDAMENTALS and Input/Output

1.1 The Fundamental Java Program

The essential framework on which all Java programs are based is illustrated with this example we call `MyFirstJavaProgram`

```
import hsa.*;
public class MyFirstJavaProgram {
    public static void main(String[] args) {
        Stdout.println("Hello World");
    }
}

// End of program
```

- The `import` statement allows us to make use of predefined methods such as those for printing, inputting, etc. This statement will be explained fully a bit later, once we have some practice with programming.

- The `public class MyFirstJavaProgram` is a *class declaration*. **Keywords** are used, in this case *public* and *class.* These words are part of the Java language and they direct the compiler. They are not user defined. The *identifier* `MyFirstJavaProgram` gives a name to our class, and we use name conventions when choosing a name for a class. These are,

  - The name of the class starts with *upper case*
  - The name of the class contains *no spaces*
  - The name of the class contains only alphanumeric characters
  - For readability, if there is more than one word in the name of the class we capitalize the start of the new word

- For our program to compile successfully it is necessary to save the program with the very same name as the class followed by the extension *.java.* Our program file needs to have the name

`MyFirstJavaProgram.java`

- The main method of our program where execution always begins is identified by the next heading

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

- Finally, braces {} are used to enclose code. Notice that for every opening brace there is a corresponding closing one. In our program, there is a set of opening and closing braces for the class *MyFirstJavaProgram* and there is also another set for the main method. The innermost opening braces always correspond with the innermost closing braces.

- At the very end we have an example of *comments*. These can be included anywhere in the program and start with two slashes as shown in the example. The compiler will completely ignore the words after the two slashes. The comments can span for one line of text, and they are meant for us to make our programs more readable. Another way of commenting is by using block comment delimiters. Any text inserted between /* and */ is considered by the compiler as comments and will also be ignored. The following text will be bypassed by the compiler,

```
/* This is my first program and
   I think I am going to enjoy it.
   Today we learn about public static void main(..)
   and other characteristics of Java */
```

- The `Stdout.println("Hello World");` statement outputs the text *Hello World* to the output screen. We now spend some time with this statement in the next section.

1.1 Exercises

1. Create a folder within your `Home` folder and call it `Java Programs`. Use this folder to store all your programs

2. Create and run the program above `MyFirstJavaProgram`

3. Modify the program above so that the class is named `MySecondJavaProgram`. Make sure it compiles and runs

4. Write a Java program that prints out your name, your address, city, province, and postal code. The program needs to print out each item in a separate line as if being printed on an envelope.

5. Using these course notes and the internet, find definitions for these items and write a sentence about each of them in your own words:

a) Program	*f) Compiler*
b) Programming Language	*g) Interpreter*
c) Editor	*h) Byte Code*
d) Source code	*i) JVM*
e) Integrated Development Environment	

1.2 Text and Number Output with Java

Sending text to the screen is a fundamental programming operation. Before windowing systems were invented by Xerox and commercialized by Apple and Microsoft, all text was sent to the one screen which had only one window: The entire screen.

The first windowing system was called Smalltalk and was created by the Xerox photocopying company in the 1960's. Not many people paid attention to this system and it was archived for 10 years as nothing more than an interesting pilot project.

When Steve Jobs visited Xerox in the late 1970's for a presentation of Smalltalk, he was absolutely amazed and quickly realized the potential of this system. He soon thereafter created the windowing system for the Apple MacIntosh in the early 1980's. Microsoft followed with Windows 1.0, though it was not until Windows 3.0 in 1990 that Microsoft started to seriously implement windowing systems.

Java allows the programmer to send output to windows as needed. There are three different ways in which a programmer can do this. A programmer can send text output to any or all of the following:

1. The Standard Java Output window
2. The Standard Java Output window using `import hsa.*`
3. The programmer's own defined window also using `import hsa.*`
4. Any combination of the above

We now look at each of these separately

1.2.1 Printing to the Standard Output Window with `System.out.print()` and `System.out.println()`

The `System.out.print()` and `System.out.println()` methods are used to output text, numbers, evaluated expressions, and concatenated expressions. Here is a summary with examples:

- `System.out.print("Hello")` prints the word *Hello* on the output screen. Any text enclosed in double quotes will be printed *as is*
- `System.out.print(24)` prints the number 24 on the output screen
- `System.out.print(2 + 3 * 5)` evaluates the expression in brackets and prints the result on the output screen. In this case 17 will be printed. Mathematical expressions follow BEDMAS
- `System.out.print(5 / 2)` evaluates the expression and prints the result. This is an example of *integer division* since both dividend and divisor are integers. The result is not rounded but truncated to 2
- `System.out.print(5.0 / 2)` evaluates the expression an prints the result. This is an example of *floating point division*. It is enough that either dividend or divisor be a floating point that the result will also be a floating point. In this case 2.5
- `System.out.print("Hello" + " " + "world.")` evaluates the expression and prints the result. This is called *concatenation*, where text is joined by means of the plus sign. The result printed is `Hello world.`
- `System.out.print("You are " + (15 + 1) + " years old")` evaluates the expression. Here numbers and text are mixed, something that, unlike Java, most languages do not allow. Java first evaluates the mathematical expression *(15 + 1)* to yield 16. Then Java converts the 16 to text and joins it with the rest of the expression. Notice that in a single statement the plus sign takes on two different meanings: Arithmetic addition and concatenation. The printed result is *You are 16 years old*

- For every example given above we can replace `System.out.print` with `System.out.println`. The effect will be that a new line feed is printed

- `System.out.println()` prints a blank line

1.2.2 Printing to the Standard Output Window with `Stdout.print()` and `Stdout.println()`

A second way to send text to the Standard Java Window is through the use of also `print()` and `println()` statements. These work in the same way as the ones explained above but contain a powerful feature: *Formatting*. This feature has been implemented in such a way that makes it easier on the programmer.

For us to have access to these printing statements, we need to include the statement

```
import hsa.*;
```

before the header of our class. For example,

```
import hsa.*;
public class NamesOfStudents {
    public static void main(String[] args) {
        < .... program code goes here .... >
    }
}
```

To perform actual printing, we can use the commands `Stdout.print()` and `Stdout.println()` in the same way that we use `System.out.print()` and `System.out. println()`

In the small program above, we can now insert commands like:

- `Stdout.print(“Hello”)` prints the word *Hello*
- `Stdout.print(24)` prints the number 24

- `Stdout.print(2 + 3 * 5)` evaluates to 17 and is printed

These commands offer simple yet powerful formatting as well. For example,

- `Stdout.print(“My Java textbook”, 30)` prints the words `My Java Textbook` on a left justified field of 30 spaces
- `Stdout.print(“My Java textbook”, -30)` prints the words `My Java Textbook` on a right justified field of 30 spaces
- `Stdout.print(18 * 3, 6, 2)` evaluates the expression and prints the result. The result is 54 but a formatting is specified: 6 spaces in total, including 2 decimal places. The command prints 54.00 right justified. Note that the decimal point takes one space.
- `Stdout.print(2.5 / 7, 10, 6)` prints 0.357143 right justified in a field of 10. The division of 2.5 by 7 has more than 6 decimal places. Java rounds the result before outputting it.

1.2.3 Printing to our own defined window with `print()` and `println()`

The third way to send text to the screen is through the use of a window that we define ourselves. To do this, we need to include a statement in our program that creates the window. Once this is done, the window will be available for output and will become visible on the first output command.

Here is the code to send the word `Hello` to our own defined window:

```
import hsa.*;
public class MyFirstOwnWindow {
    public static void main(String[] args) {
        Console c = new Console();
        c.println("Hello");
    }
}
```

We can now use the `print()` and `println()` commands in the same way as we did above for number formatting. For text, only left justification is done the same way as in Stdout. Text right justification is not possible with a user defined window and we need to use other methods, such as tabs, to do this.

To output to our own window, we always need to write the window and the printing command separated with a period. For example, if our window is called `glass`, our printing commands become,

```
glass.println(100 * 2 - 1);
glass.println("hello, how are you");
glass.println("The cat came back");
glass.println("Your test mark is: " + ((87 + 82 + 91) / 3.0));
```

1.2 Exercises

1. Write a Java program with the class name `MovieTicket` that displays in the Standard Java Window the following information, exactly as shown here:

```
----------------------------
AMG Movie Theatres - Toronto
July 15, 2015 8:05 pm
Mr. Holmes, $12.25
Theatre 3 - Not reserved

Payment method: Debit Card
Amount paid   : $12.25
Enjoy the show
----------------------------
```

2. Write a Java program with the class name `LetterE` that displays in the Standard Java Window, but using `Stdout`, the following pattern, exactly as shown here:

```
**********
**
**
*******
**
**
**********
```

3. Write a Java program with the class name `Triangle` that displays in Standard Java Window, using `Stdout`, the following pattern, exactly as shown here:

```
        *
       * *
      *   *
     *     *
    *       *
   *         *
  *           *
 *             *
* * * * * * * * *
```

4. Write, compile and run these three programs. Study their differences and similarities.

```
import hsa.*;
public class Multiply1 {
   public static void main(String[] args) {
        Stdout.print(“3 + 2 = “);
        Stdout.print(3 + 2);
        Stdout.println();
   }
}

import hsa.*;
public class Multiply2 {
   public static void main(String[] args) {
        Stdout.println(“3 + 2 = “ + (3 + 2));
   }
}

import hsa.*;
public class Multiply2 {
   public static void main(String[] args) {
        Stdout.println(“3 + 2 = “ + 3 + 2);
   }
}
```

5. Write, compile and run these two programs. Study their differences and similarities.

```
import hsa.*;
public class Divide1 {
    public static void main(String[] args) {
        Stdout.println("12 / 11 = " + 12 / 11);
    }
}

import hsa.*;
public class Divide2 {
    public static void main(String[] args) {
        Stdout.println("12 / 11 = " + 12.0 / 11);
    }
}
```

6. Write, compile and run these two programs. Study their differences and similarities.

```
import hsa.*;
public class Divide3 {
    public static void main(String[] args) {
        Stdout.print("12/5=");
        Stdout.println(12 / 5, 8, 2);
    }
}

import hsa.*;
public class Divide4 {
    public static void main(String[] args) {
        Stdout.print("12/5=");
        Stdout.println(12.0 / 5, 8, 2);
    }
}
```

7. Write a program that outputs the following with proper alignment, exactly as shown here:

   a. Output needs to be sent to the Java Standard Window using `Stdout`.
   b. Numbers must be output as numbers, not text: *Do not write your numbers in quotation marks.*
   c. Use the adequate format of the commands `Stdout.print()` and `Stdout.println()`:

```
One                   1
Ten                  10
One hundred         100
One hundred         100.00
Thousands          3645.1
Thousandths           0.342

Two hot dogs       $  3.45
Two soft drinks    $  2.76
Full meal          $ 12.59
```

8. In this exercise we experiment with the special tab character `\t`. This character will create a tab in the output. For example: `System.out.print("John\tSmith")` will output

```
John    Smith
```

The word `Smith` is pushed to the next tab position. Tab positions usually happen at 9, 17, 25, 33, i.e. multiples of 8 plus 1. Write a Java program with the class name `Students` that uses the `\t` character. This character can be used in any output window The program must output exactly:

```
Student Number Name    Phone
-------------- ----    ------------
324332546      John    416-464-3321
354657555      Sue     905-443-2222
354665575      Anna    416-435-3633
```

9. Write a program with the class name `StudentMarks` that outputs the following with proper alignment and decimal places. Make sure to use numbers for the marks, not text, and a correct expression to calculate the average. *Do not copy the number* for the class average as given here:

```
Student Number         Quiz1     Quiz2     Quiz3     Average

323144324              67%       70%       78%       71.67%
324435007              89%       73%       86%       82.67%
310755654              57%       64%       74%       65.00%
387102291              34%       54%       57%       48.33%
375842516              67%       64%       58%       63.00%
300644535              95%       67%       75%       79.00%
-----------------------------------------------------------
                                   Class average     68.28%
```

10. Write a Java program with the class name `Calculator`. The program needs to calculate and print the following expressions. Answers on the right column are for you to check your output:

```
a) (24 + 1) * -2                                  (ans. -50)
b) The square root of (125 / 5 * 4)               (ans. 10.0)
c) 4 / 2 + 8 / 4 + 12 / 6 + 16 / 8                (ans. 8)
d) 4 cubed                                        (ans. 64.00)
e) 2 raised to the power of 5                     (ans. 32.00)
```

11. Write a Java program with the class name `Calculations`. Only letters can go in quotes. All numbers need to be printed without quotes and all calculations must be performed by Java, not by you. The program outputs on any window the following three statements:

```
A square has a side of 25.0m
Its area is 625.0 sq m

A rectangle has length 10.30m and height 3.00m
Therefore, its perimeter is 26.60m

A circle has a radius of 10.00cm
Its circumference is 62.80 cm
```

1.3 Variables and Data Types

A fundamental concept of programming languages is the *variable*. A variable is a name associated with a block of memory, where values can be stored, retrieved, and modified. Variables are created by the programmer as part of the program source code.

The concept of storage of values in memory was introduced in the 1950's by *John Von Neumann,* a Hungarian born mathematician and physicist who worked in the Manhattan project. Von Neumann based his work partly on the work of J. Presper Ecker and John William Mauchly, the inventors of the ENIAC computer at the University of Pennsylvania (1946). Von Neumann proposed an architectural plan for the computer that has been used ever since. Thus our modern computer is appropriately called a *Von Neumann Computer*. This early vision of a separate logic for a central processing unit (CPU) that fetches, processes and stores information from and to memory (RAM) set the plan for all future computer systems.

1.3.1 Variables

In Third Generation languages, access to memory is made relatively easy by the use of variables. Variables contain information that can be accessed through declared names. In Java, *all variables need to be declared* before they can be used. A declaration statement tells the Java compiler/interpreter that we intend to use a variable in the program. The declaration also specifies the type of information that we will put in the variable.

There are two kinds of variables in Java: *Primitives* and *objects*. First we will look at primitives. At a later time in the course, we will look at objects which are partially built on primitives.

Primitives are variables whose memory is *automatically allocated at declaration time.* This means that the variable has its own memory space that can only be used by the program that created it. It is the responsibility of the Operating System to make sure that this space is fully protected and not used by any other program. Once declared, primitives can be used right away. In Java, when we use the word *variable*, we mean a *primitive variable*.

Different types of variables can contain different types of data, as explained below. The conventions for naming a variable:

- The name starts with a lower case letter
- If the name has more than one word, each additional word is capitalized
- Only numbers or letters in the name. No underscores or special symbols
- A variable name cannot use a keyword for a name
- It is good practice to use descriptive names

We will see that variables can be used in Java commands such as `print()` and `println()` to output their values.

1.3.2 Data Types

When a variable is declared, we will need to specify its *data type*, i.e., the kind of data that a variable can hold. Here are the data types we use most frequently for Java variables,

declaration	data type	what it can hold
`byte`	integer	integers from -128 to 127
`char`	single character	65,536 different characters
`short`	integer	integers from -32768 to 32767
`int`	integer	integers from -2^31 to 2^31 - 1
`long`	integer	integers from -2^63 to 2^63 - 1
`float`	floating point	-3.4E38 to 3.4E38 with 6-7 decimals of accuracy
`double`	floating point	floats from -1.8E308 to 1.8E308 with 15-17 decimals of accuracy
`boolean`	logical	true or false
`String`	text(*)	text

* *A String is not a primitive but an object. Because it is used so often, Java compilers allow its use as though it were a primitive. We will see later why it is important to remember that it is an object.*

1.3.3 Declaring Variables

In a variable declaration, we use the syntax:

```
    [data type] [identifier];
```

For example,

```
     int age;
     int monthOfYear;
     double average;
     boolean finished;
```

Here are some more examples,

```
      int mountainHeight;
      int lotteryPick1, lotteryPick2, lotteryPick3;
      double probabilityOfAThree;
      boolean gameOver;
      double assignmentMark;
      byte keyboardLetter;
```

Alternatively, we can declare the variable and give it an initial value. The syntax is

```
    [data type] [identifier] = [initial value];
```

For example,

```
     double mark = 84.5;
     int numberOfCars = 0;
     boolean gameOver = false;
```

1.3 Exercises

1. Write a declaration statement for the following:
   a) The number of bank accounts a customer can have
   b) The maximum air speed of an airplane
   c) Whether a person has previously applied for a particular job
   d) The number of people in the classroom
   e) The amount in a bank account
   f) The number of basketballs in a department store
   g) The price of a basketball

2. Write a Java program with the class name `VolumeOfCylinder`. In the program declare and initialize the following variables:

   `radius` is a `double` with a value of `4.18`
   `height` is a `double` with a value of `11.18`

   Use the formula for the volume of the cylinder to calculate and print the volume of a cylinder with the dimensions given above. When the program runs, it should display

   ```
   radius = 4.18u, height = 11.18u, volume = 613.37 cu. u
   ```

3. Write a Java program with the class name `SlopeOfLine`. In the program declare and initialize the following variables:

   `x1` is a `double` with a value of `5`
   `y1` is a `double` with a value of `1`
   `x2` is a `double` with a value of `-3`
   `y2` is a `double` with a value of `5`

   Use the slope formula to determine the slope of the line that passes through the points `(x1, y1)` and `(x2, y2)`. Your code must not use actual numbers after the variables have been declared and initialized, but only the variables. The program output should be

   ```
   Slope of the line through (5, 1) and (-3, 5) is -0.5
   ```

4. Write a Java program with the class name `AverageMark`. In the program declare and initialize the following variables:

   `test1` is a `double` with a value of `75.5`
   `test2` is a `double` with a value of `81.0`
   `test3` is a `double` with a value of `80.0`

   The program outputs the average mark with one rounded decimal place

   ```
   Average mark is 78.8%
   ```

5. Write a Java program with the class name `RectangleGraphic`. In the program declare and initialize the following variables:

`upperLeftX` is an `int` with a value of `60`
`upperLeftY` is an `int` with a value of `110`
`width` is an `int` with a value of `250`
`height` is an `int` with a value of `150`

Define a window with the name `g` and include the command

```
g.drawRect(upperLeftX, upperLeftY, width, height);
```

6. Write a Java program with the class name `NameInfo`. In the program declare and initialize the following variables:

   `name` is a `String` with the value `"Joanne Adams"`
   `age` is an `int` with a value of `16`
   `birth` is a `String` with the value of `"May"`
   `year` is an `int` with a value of `1998`

   Using a single printing statement and making use of the variables but not the actual values, write the program that will print

   ```
   Born in May 1998, Joanne Adams is now 16 years old.
   ```

7. The following program calculates how many times the human heart beats in one month, on average. The program does not compile because it contains 6 errors. Find the errors so that you can compile and run the program:

   ```
   import hsa*;
   public class HeartBeats {
       public void static main(String() args) {
           console window = new Console();

           int beats = 72.0;
           window.print("The human heart beats ");
           window.print(beats * 60 * 24 * 30);
           window.println(" times in one month")
       }
   }
   ```

1.4 Sizes of Variables

A quick and simple introduction to Random Access Memory (RAM) will be helpful for understanding variables. RAM is a large grid where each cell of the grid can hold one exactly one byte. In most programming languages one byte is exactly one character. The following is a RAM that can hold a maximum of 80 bytes:

The words we type on the keyboard are placed in RAM like this:

c	a	t	(sp)	d	o	g			

Java is different from other languages in that it uses two bytes to store a character. Therefore, when storing the words `cat dog`, Java places them in memory this way:

	c		a		t		(sp)		d
	o		g						

With this simple definition in mind, we can now specify how much space each type takes in memory when programming with Java,

declaration	size in memory
`byte`	1 byte
`char`	2 bytes
`short`	2 bytes
`int`	4 bytes
`long`	8 bytes
`float`	4 bytes
`double`	8 bytes
`boolean`	1 byte
`String`	variable multiples of 2 bytes

1.4 Exercises

1. Calculate how much memory in bytes is being used by each of the following cases:
   a) 3 bytes and 2 shorts
   b) 4 floats, 240 characters, and 6 doubles
   c) 20 integers

2. Calculate how much memory in bytes is being used by the primitives of this program when it is running

```
import hsa.*;
public class Quadratic {
    public static void main(String[] args) {
        int a = 1;
        int b = 5;
        int c = 6;

        double x1=(-1.0*b+Math.sqrt(Math.pow(b, 2)-4*a*c))/(2*a);
        double x2=(-1.0*b-Math.sqrt(Math.pow(b, 2)-4*a*c))/(2*a);

        Stdout.println("Root 1: " + x1);
        Stdout.println("Root 2: " + x2);
    }
}
```

3. Write a Java program that uses a total of 44 bytes by declaring only integers, character, and double variable types. Initialize each integer or double variable of your program to a value that is clearly invalid. By 'invalid' we mean that the value can be assigned, the program will compile, but the value is non-sensical. For example, initializing a variable named `height` to -1. Then, initialize each character value to the null character: `'\0'`

1.5 Assignment Statements and Casting

Now that we know how to declare variables, we can use them in different ways. One of the most fundamental ways is through *assignment*, which has the effect of storing a value into the variable.

1.5.1 Assignments

Java allows the following assignment operations with variables:

• A constant to a variable,

```
int cars;
cars = 14;
```

• An expression to a variable,

```
double cost;
double tax;

cost = 125;
tax  = cost * 0.13;
```

• A variable to a variable,

```
int boxes = -1;
int wrapping = -1;

boxes   = 3;
wrapping = boxes;
```

- A constant, an expression or a variable at declaration time

```
boolean done = true;
double diameter = 2 * 10.5;
double d = diameter;
```

- An increment

```
int count = 0;
count = count + 1;
```

- An alternative way to increment

```
int count = 0;
count++;
```

- An expression of variable to itself,

```
int numberOfPeople;

numberOfPeople = 32;
numberOfPeople = numberOfPeople + 3;
```

- Alternatively,

```
int numberOfPeople;

numberOfPeople = 32;
numberOfPeople += 3;
```

1.5.2 Casting

When assigning to a variable, the value assigned and the receiving variable must be

- Of the same type, or
- Implicitly casted to a higher type, for example `int` to `double`, or
- Explicitly casted to a lower type, for example `double` to `int`

- A mathematical integer variable that has more memory space reserved for storage is of a higher type than a variable that has less space reserved for storage.
- A mathematical floating point variable that has more memory space reserved for storage is of a higher type than a variable that has less space reserved for storage.
- A floating point variable is of a higher type than an integer variable.
- A string variable is of a higher type than a character variable.

An implicit casting means that Java will automatically create the correct type from the given value to the assigned variable. For example:

```
int days = 628;
double months = days / 31;    // careful: integer division
```

When 628 is divided by 31, the resulting value is first converted to a `double` before it is assigned to the variable `months`. This is possible because a `double` is a higher type than an `int`. This is know as *expansion*. Because of this, there is no loss of precision and the entire value is fully kept in the new variable.

An explicit casting means that the programmer must specify in the program how the value will be assigned. This is done when a higher type needs to be assigned to a lower type. This is know as *coercion*. In this operation, higher order bytes will be lost. This is done to conserve space or because the higher order bytes are not needed. For example,

```
long time = 1143351756;
int seconds = (int) time;
```

Since the number `1143361756` fits in an integer, it can be casted to the variable `seconds` without loss of information.

Notes on `long` and variable assignments: If the value being assigned to a `long` is a constant and is not in the range of an `int`, then we need to add the letter `L` (upper case or lower case) at the end of the constant for the compiler to accept the value. In the example that we saw above

```
long time = 1143351756;
```

the constant is within the range of an `int`. Therefore the line compiles successfully. However, if the constant is `7142389754`, we need to write the statement like this,

```
long time = 7142389754L;
```

Java will always keep a mathematical integer constant as an `int` unless otherwise specified.

Notes on `float` and variable assignments: When a floating point constant is assigned to a `float` variable, the constant must always be followed by the letter `F` (upper case or lower case) as in this example:

```
float area = 6.78f;
```

Java will always keep a mathematical floating point constant as `double` unless otherwise specified.

From the highest to the lowest the casting order of types in Java is,

declaration	data type	Size in memory
`double`	numbers with decimals	8 bytes
`float`	numbers with decimals	4 bytes
`long`	integers	8 bytes
`int`	integers	4 bytes
`short`	integers	2 bytes
`char`	single character	2 bytes
`byte`	single byte	1 byte

Furthermore, the `char` type can be assigned to the `String` type with implicit casting, but `String` cannot be assigned to `char` even with explicitly casting because `String` is not a primitive. The order is,

declaration	data type	Size in memory
`String`	text	unlimited
`char`	single character	2 bytes

1.5.3 Other Considerations with Variables

Java allows the use of exponential notation when assigning values to a variable. For example

```
double c = 3e8;
```

will store in the variable `c` the value $3x10^8$ or `300000000`

Uninitialized variables will cause a compiler error if they are assigned before initialization. For example,

```
int a, b;
```

```
  a = b;
```

will cause the compiler to generate an error because the variable `b` does not have a value at the time that it is assigned to `a`

1.5 Exercises

1. State the type of each value (some could have more than one type)
   a) -5
   b) 37.0
   c) 'A'
   d) 65
   e) "Hello"
   f) 34 * 5
   g) 77 / 6
   h) 12 / 3.0
   i) 2.7e-4
   j) "31554"
   k) false

2. Rewrite in standard decimal form
   a) 8.77e-3
   b) 299.04e0
   c) -0.0443e6
   d) -23e4

3. Which of the following conversions always require explicit casting?
   a) `long` to `double`
   b) `byte` to `char`
   c) `float` to `double`
   d) `byte` to `long`
   e) `short` to `char`
   f) `double` to `long`

4. What would be printed from this program fragment

```
double x = 0.093e-1;
double y = -3e-3;
System.out.println("x: " + x + " y: " + y);
```

5. Find the three errors in the following program

```
public class ErrorsInProgram {
    public static main(String[] args); {
        int i = 7;
        int j;

        i = j;
        System.out.println("Value of i is " + i);
        System.out.println("Value of j is " + j);
    }
}
```

6. Write the value of each expression. Use a decimal point in your answer if the result is floating point

```
a) (int) 1.8 * 0.6
b) (int) 6.3 / (int) 4.7
c) (float) 2 * 3 + 0.1
d) (double) (7 % 4 / 3)
```

7. Assume that all the variables in this exercise have been declared. Rewrite the following statements using alternative increment or decrement operators

```
a) count = count + 1;
b) people = people - 1;
c) billiards = billiards + 5;
d) cars = cars - 3;
```

8. Find the value of j and k after execution of the following code fragment

```
int j = 0;
int k = 0;
j+=2;
k = j;
k++;
k += j;
k = k * j;
j = 92 % k;
```

1.6 Input from the Keyboard

Input from the keyboard can be done in different ways in Java. Here we look at two different ways. It is a good idea to be consistent with the choice of method.

1.6.1 Input using Stdin or a user defined window (Console)

Data can be input into a variable through the keyboard. In computer jargon, the keyboard is called *the standard input stream or console*. When we include the `import hsa.*` statement, we have access to a set of input methods.They are:

- `Stdin.readByte()`
- `Stdin.readShort()`
- `Stdin.readInt()`
- `Stdin.readLong()`
- `Stdin.readFloat()`
- `Stdin.readDouble()`
- `Stdin.readString()` (reads only up to the first blank)
- `Stdin.readLine()` (reads the entire line)
- `Stdin.readChar()`
- `Stdin.readBoolean()`

Each one of these returns the appropriate type, and the value is assigned to a variable. For example, the following statements prompt the user for a person's age:

```
int age = -1;
Stdout.print("Please enter your age: ");
age = Stdin.readInt();
```

All of these are also available in a user defined window,

```
Console w = new Console();

int age = -1;
w.print("Please enter your age: ");
age = w.readInt();
```

1.6.2 Input using the Scanner Class

Data can also be input through another set of methods that are found in most Java IDE's and implementations. To access these methods we need to include the following statement at the top of the program together with any other `import` statements:

```
import java.util.*;
```

Inside the program, the following statement is necessary

```
Scanner [variable] = new Scanner(System.in);
```

where `[variable]` can be any variable name. For example,

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

Furthermore, Java-RTP does not include the Scanner class automatically, so we need to access it by putting the following statement in the Class Path. The Class Path can be found by choosing from the IDE

File -> Preferences -> Java -> Additional Class Path Directories

We then include the following path to where the Scanner.class resides:

```
\\Tdsbshares\schclass$1276\1276-Pickup\Mr. Mario\Java Classes
```

The methods available are

- `[variable].nextByte()`
- `[variable].nextShort()`
- `[variable].nextInt()`
- `[variable].nextLong()`
- `[variable].nextFloat()`
- `[variable].nextDouble()`
- `[variable].next()` (reads only up to the first blank)
- `[variable].nextLine()` (reads the entire line)
- `[variable].nextBoolean()`

Note to Scanner users: In some implementations of the Scanner class the method `nextLine()` does not process the `CR` and `LF` and the end of the line together but as two different lines of input. Therefore, two `nextLine()` statements are needed to process a single line. Our implementation of Scanner processes CR and LF as a single line of input, so only one `nextLine()` statement is required.

1.6 Exercises

1. Write a computer program that clearly asks the user for 3 test marks, computes the average and outputs the result with a clear label. The answer should round to one decimal place. Make sure to initialize variables to invalid or null values.

2. Write a computer program that uses a boolean variable, initialized to `false`. Use the variable to obtain `true` or `false` from the user and then print out its value. The words in red below represent the value of the variable. For example,

```
I am 16 years old: (True/False)? true
It is true that I am 16 years old
```

3. Why is the following program not compiling?

```
public class Numbers;
     public static void main(String[] args) {
          short height = -1;
          speedOfLight = 3e8;
     }
}
```

4. Write a program that does the following:
   a. Declares an integer variable, asks the user for an integer, coerces the value into a separate short variable, and prints the value of the short
   b. Declares a double variable, asks the user for a number with both an integer part and a fractional part. The program then puts only the integer part of the number into another variable and then prints the resulting integer. For example, if 2.43 is input, then the program puts only 2 in another variable and then the 2 is printed.
   c. Similar to part b but the program puts the fractional part into a variable whose value is then printed with only 2 decimal places. For example, if 2.43 is input, then the program puts the fractional part, i.e. 0.43 in another variable, and then it's printed.
   d. Declares a short variable, gets a value from the user (between 65 and 110), casts the value into a `char`, and prints it out as a character. How is the value that you entered related to the character that is printed?

5. Run through the following program manually, writing out the value of each variable as the program runs. Then write what the program prints.

```
import hsa.*;
public class Tracing {
     public static void main(String[] args) {
          int count = 0;
          int sum   = count;
          count = count + 1;
          sum   = sum + count;
          count = count + 1;
          sum   = sum + count;
          count = count + 1;
          sum   = sum + count;
          count = count + 1;
          sum   = sum + count;
          Stdout.print(sum);
     }
}
```

6. If you invest *P* dollars at a yearly interest rate of *R*, compounded yearly, in *N* years your investment will grow to *A* according to the formula

$$A = P\left(1 + \frac{R}{100}\right)^{N}$$

Write a Java program that asks the user for the initial investment, the yearly rate and the number of years. The program calculates and outputs the value of *A* with two decimal places.

7. Write a Java program that asks the user for two coordinate points in the plane (x1, y1) and (x2, y2). The program outputs the equation of the line through those points. For example, if points are (1, -2) and (3, 4) the output of the program is

```
Equation of the line through (1, -2) and (3, 4) is y = 3.0x + -5.0
```

1.7 Graphics output with hsa

The hsa package provides a collection of methods that provide the programmer with graphics output. When using the hsa package, all output must be sent to a user defined window or console.

1.7.1 Graphics output using a user defined window (Console)

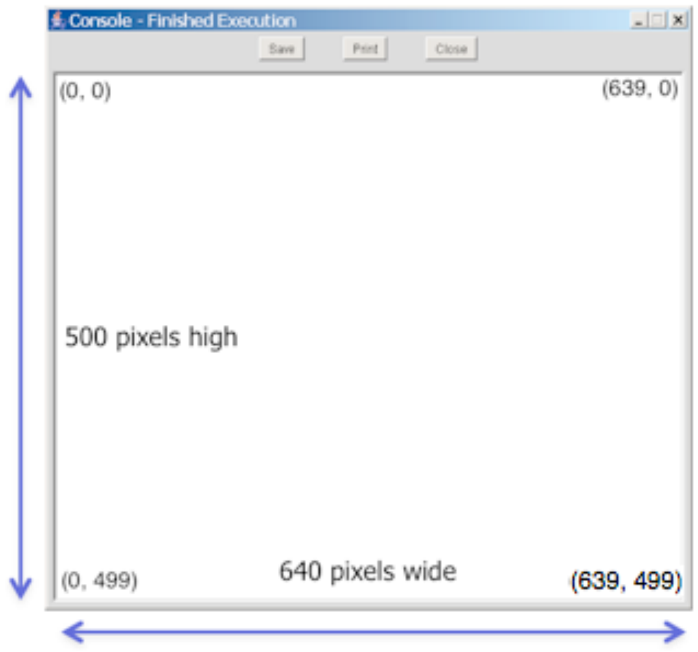
Graphics output using hsa is sent to a user defined window since all the graphics commands available work only with a window that the user creates. Therefore, at the beginning of our code we need the statement

```
    import hsa.*;
```

and inside the main method we need a line such as

```
    Console w = new Console();
```

Java gives us a window as shown here



Notice that in Java, the origin (0, 0) is at the upper left corner of the window. If we want to plot points using the Cartesian coordinate system, we will need to perform a small transformation so that the points show in the correct location. This standard window has 25 rows and 80 columns for writing text with a font size of 14.

The `new Console()` statement can be expanded to allow us to define the window size, the font to use in the window, and the title of the window. Here are the other ways in which it can be used:

`Console w = new Console(rows, columns)`
creates a window with as many rows and columns as specified

`Console w = new Console(rows, columns, fontSize)`
same as above but the size of the writing font can be specified

`Console w = new Console(rows, columns, fontSize, title)`
a title can be specified here, as a string type so it must be in quotes

`Console w = new Console(rows, columns, title)`
same as above but the font size is the standard size of 14

`Console w = new Console(title)`
standard size window and a title can be specified

`Console w = new Console(fontSize)`
25 rows by 80 columns with the specified font size

`Console w = new Console(fontSize, title)`
same as above and a title can be specified

1.7.2 Graphics commands in `hsa`

The following are some of the graphics methods available in `hsa`.
The graphics methods available are

- `drawLine(x1, y1, x2, y2)`
- `drawRect(x, y, width, height)`
- `fillRect(x, y, width, height)`
- `drawOval(x, y, hDiameter, vDiameter)`
- `fillOval(x, y, width, height)`
- `setColor(color)`
- `setColor(new Color(redValue, greenValue, blueValue))`
- `clear()`
- `setCursor(row, col)`
- `drawString(text, x, y)`
- `maxx()`
- `maxy()`

These methods are used for text output:

- `maxrow()`
- `maxcol()`
- `setTextColor(color)`
- `setTextColor(new Color(redValue, greenValue, blueValue))`
- `setTextBackgroundColor(color)`
- `setTextBackgroundColor(new Color(redValue, greenValue, blueValue))`

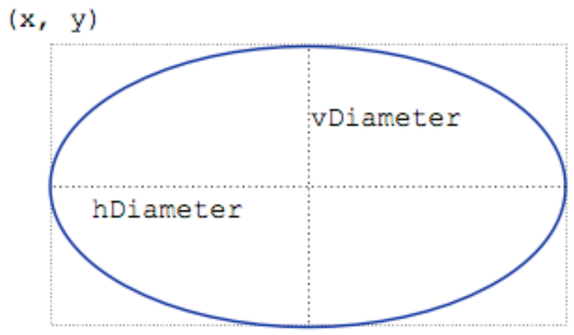
The values that we pass to the methods are called *parameters*. All numeric parameters expect values that will fit into the `int` type:

`drawLine(x1, y1, x2, y2)`
draws a line from `(x1, y1)` to `(x2, y2)` in the current color. To draw a single point, use `drawLine(x, y, x, y)`

`drawRect(x, y, width, height)`
draws a rectangle with upper left corner at `(x, y)` with dimensions `width` and `height`

`fillRect(x, y, width, height)`
fills a rectangle with upper left corner at at `(x, y)` with dimensions `width` and `height`

`drawOval(x, y, hDiameter, vDiameter)`
draws an oval enclosed in an imaginary rectangle with upper left corner at `(x, y)` and with a horizontal diameter `hDiameter` and a vertical diameter `vDiameter`



`fillOval(x, y, width, height)`

draws an oval enclosed in an imaginary rectangle with upper left corner at `(x, y)` and with a horizontal diameter `hDiameter` and a vertical diameter `vDiameter`

`setColor(color)`
sets the current drawing color. The parameter to this method can be one of:

```
Color.white         Color.red           Color.yellow
Color.magenta       Color.blue          Color.green
Color.black         Color.orange        Color.pink
Color.cyan          Color.gray          Color.lightGray
Color.darkGray
```

`setColor(new Color(redValue, greenValue, blueValue))`
sets the current drawing color. The values of `redValue`, `greenValue`, and `blueValue` can be integers from 0 to 255. With this scheme we can create 256x256x256 = 16,777,216 different colors. In order to create a new color of our own, we need to include the following statement at the top of our program,

```
      import java.awt.*;
```

`clear()`
clears the window and moves the cursor to (0, 0) position

`setCursor(row, col)`
moves the cursor to `(row, col)`

`drawString(text, x, y)`
draws the string `text` at location `(x, y)`

`maxx()`
returns the maximum x position of the current window

`maxy()`
returns the maximum y position of the current window

`maxrow()`
returns the maximum row of the current window (first row is 1)
`maxcol()`
returns the maximum column of the current window (first column is 1)

`setTextColor(color)`
sets the text color for subsequent print operations

`setTextColor(new Color(redValue, greenValue, blueValue))`
same as above but with user defined colors

`setTextBackgroundColor(color)`
sets the background text color for subsequent print operations

`setTextBackgroundColor(new Color(redValue, greenValue, blueValue))`
same as above but with user defined colors

1.7 Exercises

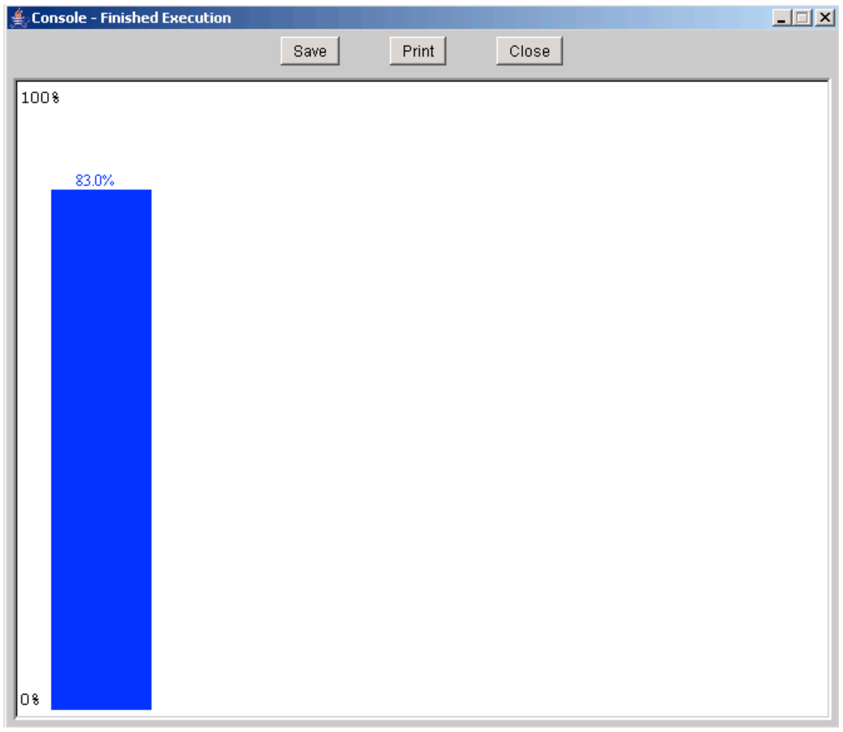
1. Write a Java program that creates a window of 30 rows and 100 columns with a font size 18 and the title `Computer Graphics at Bloor CI`. The program does the following:
   a. draws a cyan filled rectangle that takes up the entire window
   b. draws a black horizontal line that divides the window in half
   c. draws a black vertical line that divides the window in half
   d. draws the label x on the horizontal line, approximately on the far right
   e. draws the label y on the vertical line, approximately on the far top
   f. draws the line `y = x + 15` in the interval [-300, 300]
   g. sets the text background color to cyan
   h. sends the cursor to row 10, column 65
   i. prints the label `y = x + 15`

2. Write a Java program that creates a standard sized window and fills an oval in any location and of any size, with a color whose red, green and blue values are 45, 140, and 221 respectively.

3. Write a Java program that uses `Stdin` to ask the user for two numbers, `x` and `y`. Then the program asks for two other numbers, `width` and `height`. The program then creates a standard sized window and draws the frame of a red rectangle in this window with the information supplied by the user.

4. Write a Java program that uses `Stdin` to ask the user for a mark between 0 and 100 with one decimal place. The program then creates a standard sized window with a label of 0% at the bottom left corner and a label of 100% at the top left corner of the window. The program then draws a blue vertical rectangle that proportionately displays the mark of the student. The mark needs to be rounded. Here is a sample run:

```
Enter a mark: 82.6
```



5. The following image is created by drawing two diagonal lines, a rectangle, and an ellipse. The width of the rectangle is half the width of the window, and the height of the rectangle is half the height of the window. Write a Java program that draws this image, centered in the window *regardless of the dimensions* of the window. Try the program first with a standard window. Then, change the console call to make the window 30 rows and 40 columns. The rest of the code needs to remain unchanged but draw the image correctly. Run the program with other sizes to check that the image is centered correctly. (Note: The console window has a minimum set at 35 columns. The buttons at the top of the window prevent us from creating a narrower window.)

Image on standard window
25 rows, 80 columns

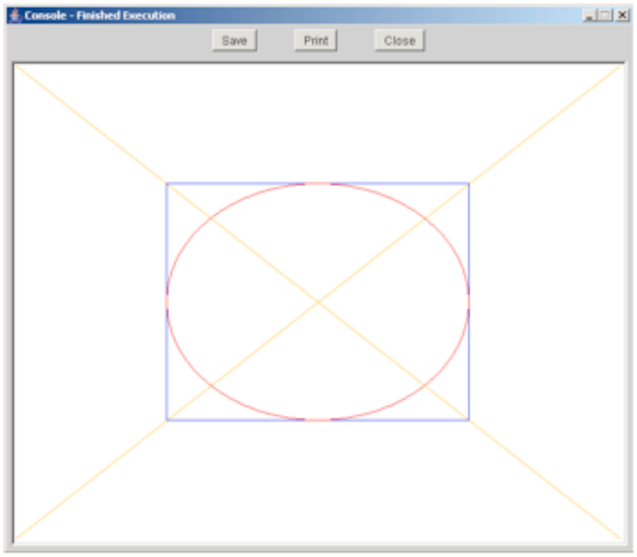
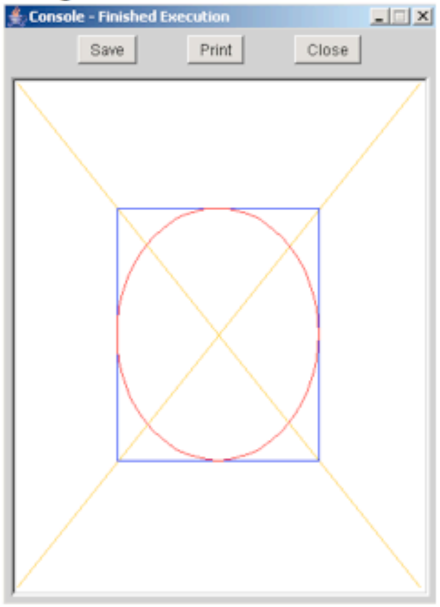


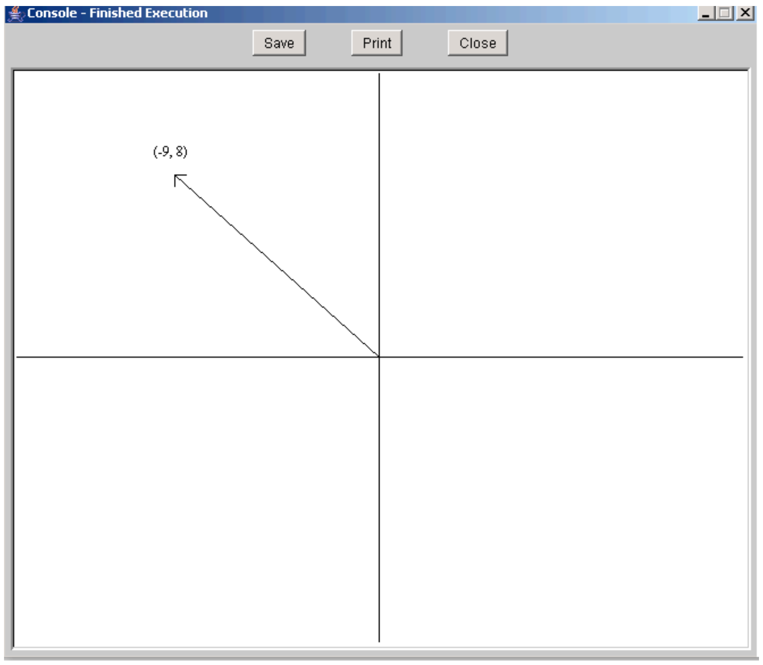
Image on 30 rows, 40 columns



6. Write a Java program that draws a vector in the Cartesian plane from an `x` and a `y` value entered by the user. A vector is drawn from `(0, 0)` to `(x, y)`.

   Draw this in a standard console and let each increment of the axes be 20 pixels. For example,

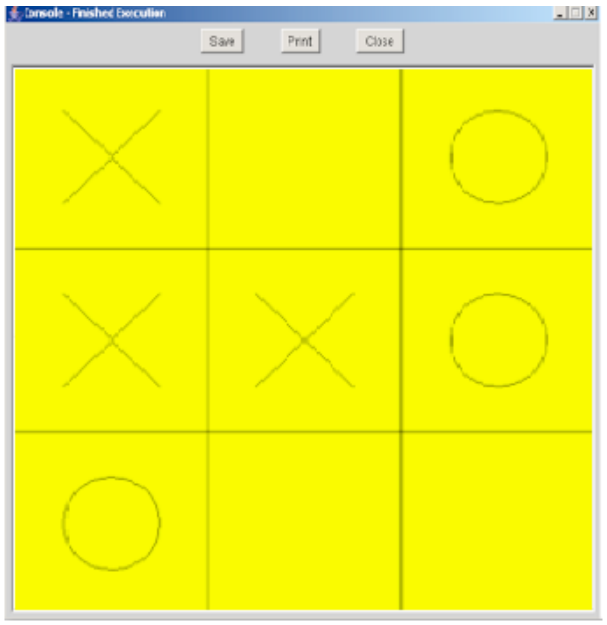
```
Enter x: -9
Enter y: 8
```



   Include in your drawing a label of the coordinates, enclosed in brackets and a few pixels away from the vector so they can be read.

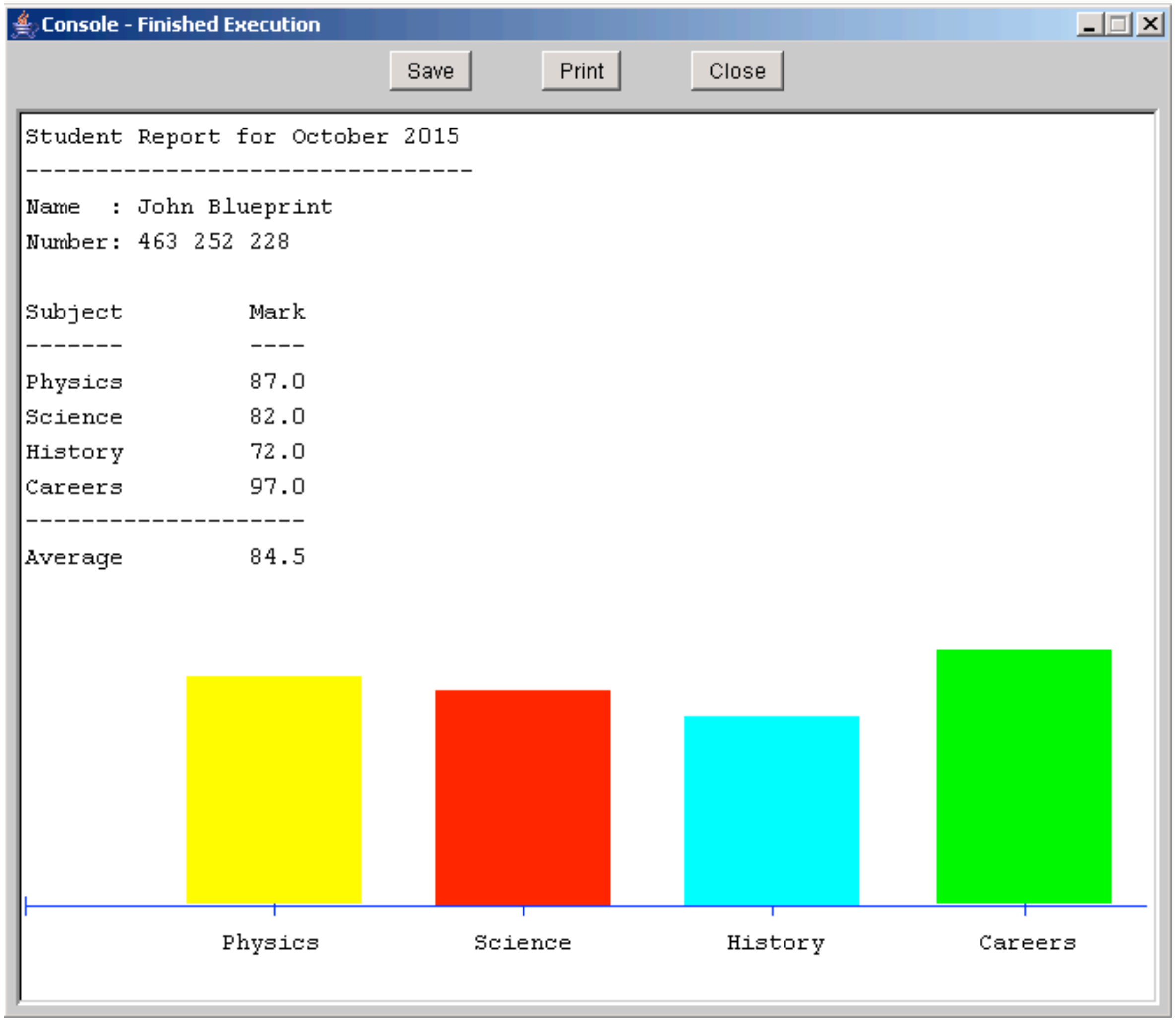
   Drawing the arrow at the tip of the vector requires drawing a small horizontal and a small vertical line from the tip. The direction of these lines can be determined by the signs of the `x` and `y` coordinates provided by the user. Each line can be drawn with a single `c.drawLine()` statement. To determine how far each line needs to be drawn, consider using the absolute function `Math.abs(x)`

7. Write a Java program that draws a Tic Tac Toe board as shown below. All squares have the same dimensions as accurately as possible (because of integer division some widths or heights may be +/-1 pixel off), and the circles and x's are centered also as accurately as possible. These drawings need to adjust their locations automatically if the size of the console window is changed during declaration.



8. Write a Java program that asks for your name (possibly with spaces), your student number (also possibly with spaces), and marks for Physics, Science, Careers, and History. The program prints a simple report along with a bar graph, with axes, as shown in this sample run:

```
Enter your name: John Blueprint
Enter your student number: 463 252 228
Enter your mark for Physics: 87
Enter your mark for Science: 82
Enter your mark for History: 72
Enter your mark for Careers: 97
```



```
Student Report for October 2015
--------------------------------
Name  : John Blueprint
Number: 463 252 228

Subject         Mark
-------         ----
Physics         87.0
Science         82.0
History         72.0
Careers         97.0
--------------------
Average         84.5
```