

I. History of Computing, Calculating Devices, Terminology, Binary Numbers

Manually-operated devices:

Chinese abacus - used for counting (3 A.D. and perfected in 1100's)

Napier's bones - rearrangeable rods used for multiplication of large numbers (1617)

Slide ruler - use for multiplication and division of large numbers using logarithms

Mechanical Devices:

Pascaline - adding & subtracting gear-driven machine by Blaise Pascal (1642) Used by his father in his tax collection work. Leibniz in 1670 improved on this by adding a crank on the side. Note: These machines were limited to simple calculations and required a human operator to make them work. They were used though for nearly 300 years.

Analytical Engine - designed by Charles Babbage an English mathematician (1833) The first general-purpose programmable computer and the forerunner of modern computers. He could not find support for his design so it was never built in his lifetime. (1 addition per second)

Electro-Mechanical Devices:

Tabulating Machine - developed by Dr. Herman Hollerith (1880's) to calculate the 1890's census. (It took 7.5 years to calculate the 1880's census and the population was growing by 25% each decade.) It manipulated punched cards using gears and wheels powered by electricity. Trivia: Dr. Hollerith in 1896 founded the Tabulating Machine Company and a few years later merged with 12 other companies to form IBM.

Mark I - largest electro-mechanical device ever built (by IBM in 1944). It was 51 feet long, 8 feet high, weighed 5 tons, consisting of 500 miles of wire. It could perform 3.3 additions or subtractions per second.

Electronic Computers:

First generation computers (1946 - 1958) - used vacuum tubes

ENIAC - a computer without mechanical parts (developed by Eckert and Mauchly in 1946). Instead of mechanical switches it used vacuum tubes. It was developed for the U.S. Army to calculate ballistic tables. It occupied 15,000 square feet of space to hold the thousands of vacuum tubes in it. It could perform 5000 additions per second. Limitations of vacuum tubes are that they generated high heat (120 degrees if not cooled), the tubes burned out frequently, and to change the program could take up to two days. In 1952 these two men sold the first commercial computer, *UNIVAC* (to the Census Bureau). In 1953 IBM sold its first computer and within two years dominated the industry.

Second generation (1959 - 1964) - used transistors instead of vacuum tubes.

Third generation (1965 - 1970) - integrated circuits (electronic circuit etched on a silicon chip).

Fourth generation (1971 - now) - microprocessors (multiple functions on a single chip).

Fifth generation (under development) - atomic level research, etc.

Types of Digital Computers

- 1) Hand held - small programmable calculators
- 2) Microcomputer - "personal" computer or PC with limited memory used by individuals
- PC - personal computer
- 3) Minicomputer - usually used by one person possibly in a workstation environment for a small business
- 4) Mainframe - large, fast computers used by medium and large sized companies usually with multiple terminals attached (frequently at removed locations)
- 5) Supercomputer - most powerful, fastest computer. The newest ones can perform 3 trillion calculations per second (just 10 - 15 years ago the speed was in the hundreds of millions of operations per second).

Terms

Computer - a machine that can

- 1) accept data,
- 2) store & retrieve data and instructions,
- 3) process that data, and
- 4) give or show the results

(Trivia note: The word computer comes from the Latin word computare which means "to compute".)

Input - feeding or putting data or instructions into the computer
(ex. via: keyboard, disk, cards, modem)

Output - obtain or see the processed data from the computer
(ex. via: printer, monitor, modem)

I/O (Input/Output)

CPU (Central Processing Unit) - controls the operation of the computer (i.e. the brains of the computer). It consists of two sections:

- 1) Control Unit - the component that coordinates and supervises all operations of the stored program (i.e. a traffic cop)
- 2) Arithmetic/logic Unit - the component that performs arithmetic and logic operations (i.e. a calculator and decision maker)

Note: The speed of a microprocessor (really the speed of its internal clock) is measured in terms of megahertz (MHz) which means millions per second.

Types of Microprocessors:

- 1) CISC (Complex Instruction Set Computer) Note: used in all computers until 1994
- 2) RISC (Reduced Instruction Set Computer) Note: newer and faster processor

Storage and Memory

a) RAM (Random Access Memory) - primary or main storage used for *temporary* storage of data or instructions.

Note: This memory is part of the main circuit board or motherboard

Note: This information is lost when the computer is shut off and is therefore sometimes called volatile storage. (RAM ran away)

b) Auxiliary or secondary storage - used for permanent storage of data outside or away from the motherboard

(ex. tape drive, floppy disk i.e.. magnetic record, CD, or hard disk which can be internal)

c) ROM (Read Only Memory) - permanent storage of data on memory chips on the main circuit board. Note: This information is not lost when the computer is shut off.

(Usually it is internal instructions to the computer so that it will operate correctly, put there by the manufacturer.)

Bus - the system of connections that links the microprocessor to the RAM, ROM, and input & output ports

Interface - a connecting link that allows two different components or system to work together

IDE (Integrated Debugging Environment) - We will be using the *Ready to Program with Java Technology IDE*

Data Storage and Representation:

When a key is depressed on a computer keyboard electrical impulses are sent down the pathways of the microchip opening or closing tiny circuits . Each opening or closing represents a single unit of information. This opening or closing (on or off) - similar to a light switch turned on or off - can be represented by the digits of 0 or 1 (note: open circuit or off = 0 and closed circuit or on = 1). There are only two digits (0 and 1) possible since there are only two states possible (open/on or closed/off). The number system of base 2 which is called the binary number system can therefore be used to represent what has so far been discussed.

Groups of these circuits next to each other are needed to represent numbers or letters. It turns out that computers use groups of 8 circuits represented by binary digits (bits) to represent a single number, letter, or character.

BITS (binary digits) - the smallest unit of information that can be recognized by a computer represented by the digits 0 or 1.

BYTE - storage location for a single number, letter, or character containing 8 bits.

Note: Nibble - half of a byte i.e. 4 bits, kilobyte - 2^{10} (1024 bytes), megabyte - 2^{20} (1,048,576 bytes), gigabyte - 2^{30} (1,073,741,824 bytes), terabyte - 2^{40} (1,099,511,627,776 bytes)

Computer Programming I - Unit 1 Lecture

Binary and Decimal Number Systems

Introduction:

ex. 345 3 hundreds + 4 tens + 5 ones (units)
 10^2 10^1 $10^0 = 1$

What is the largest single digit in the decimal (base 10) system?

How many digits are there in the decimal system?

Binary Numbers:

Binary Number System - base 2 system consisting of two digits 0 and 1

Place value in base 2:

_____	_____	_____	_____	_____	_____	_____	_____
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

A) Write the binary (base 2) numbers below as decimal (base 10) numbers

1

10

101

1100

1010

11001100

10110011

111111111

Computer Programming I - Unit 1 Lecture

Binary and Decimal Number Systems continued:

_____	_____	_____	_____	_____	_____	_____	_____	_____
128	64	32	16	8	4	2	1	

A) Write the decimal numbers below as binary numbers

1

2

3

4

5

8

10

17

31

50

102

210

248

Computer Programming - the process of planning a sequence of steps for a computer to follow in order to solve a problem

- 1) Analysis & Specification - define the purpose of the problem by clarifying the input and output
- 2) General solution using an algorithm - sequence of logical steps
Note: similar to an outline and a geometry proof
Note: uses pseudocode which is a geeky English
- 3) Convert the algorithm to a programming language - code the program
Program - a planned logical sequence of instructions to be performed by a computer in a language it understands (see below)
- 4) Test and fix any errors - check for errors and debug
Note: *Bug* - an error in a program
Note: *Debug* - fix or remove errors in a program
- 5) Document and maintain the program - i.e. modify and keeping it up to date
Documentation - written text and comments that make a program easier for others to understand, use, and modify
Note: documentation can be internal (in the program) or external (outside the program (ex. flowcharts, diagrams, descriptions))

Programming Language - a set of rules, symbols, and special reserved words used to construct a program

- 1) Low-Level
 - a) machine language - uses binary code thus directly communicating with the computer
 - b) assembler language - uses mnemonics to represent machine language code
- 2) High-Level - languages that are closer to ordinary English
ex. Pascal, FORTRAN, COBOL, BASIC, Modula-2, LISP, Ada, Java, C, and C++
Note: C and C++ (a better C) have features built into it that allows it to have the power and flexibility of the low-level languages.

Note:

For the computer to understand and use a high-level language it needs to be translated to machine language by either a program called an Interpreter or a Compiler.

A program written with a high-level language is called source program (or source code). After translation by the compiler we now have a machine language program called the object program (or object file). A special linker program then combines the

Computer Programming I - Unit 1 Lecture

object file with needed machine code to produce an executable file that, unless changes are made, it will not need to be compiled again.

Java Translation and Execution

A Java compiler translates Java source code into Java bytecode. A Java interpreter translates and executes the bytecode. This bytecode is not associated with any particular machine so is machine independent or also called architectural neutral. This bytecode is written for the Java Virtual Machine (JVM). A JVM has been written for every major operating system. JVM is like a simulated CPU that lives on top of the operating system. The Java compiler and interpreter are part of the Java Software Development Kit (SDK) or also called the Java Development Kit (JDK) and can be downloaded free from Sun Microsystem Web site (java.sun.com)

Miscellaneous Terms

Hardware - the physical equipment of a computer system

Peripherals - devices attached to the computer (See page 18)

Software - any material that allows one to operate the computer (ex. programs, written instructions etc.)

OS (Operating System) - instructions that manage the hardware resources
(ex. DOS, MS-DOS, ProDOS)

GUI (Graphic User Interface) ex. Mac and Windows

Basic Control Structures

One of the (if not the most important) principal contribution of structured programming is the concept that any program can be written using only four logical structures (flow of control). By simplifying the structure of programs we make them easier to code and to test. Because of their simplicity these programs are easier to modify when revision is necessary. The four control structures are:

- 1) Sequence - series of statements one directly after another consisting of no loops or branches (ex. a straight road)
- 2) Selection (conditional or decision) - a branch in either of two directions as a result
of testing a condition as being true or false (ex. a fork in a road))
- 3) Loop (iteration or repetition) - a cycle of one or more statements until a condition is met or not met (ex. repeatedly going around a block)
- 4) Subprogram (subroutine, procedure, or function) - mini-programs consisting of the above structures that are then connected together (ex. preparing the car and going on a long trip)

II. Introduction to Program Construction, Syntax, and Semantics

Note: Syntax - the formal rules (grammar & punctuation) for how instructions are to be written so that they will be understood by the computer

Semantics - the rules that determine the meaning of the instructions

I. *Identifiers* - the name associated with a particular computer function or data object such as a variable, or a constant;

Syntax rules:

- 1) Can only consist of letters (a-z, A-Z), digits (0-9), or underscore characters (_)
Note: No periods nor blank (white space) are allowed!
- 2) Must start with either a letter (a-z, A-Z) or an underscore character (_)
Note: You should avoid starting with an underscore as internal commands in the computer use this format and errors can then occur.
- 3) Maximum length is determined by the computer system
- 4) Reserved words - those that have a special meaning in Java cannot be used
Note: On page 31 of the blue book is a list the reserved words
- 5) Identifiers are case-sensitive meaning that uppercase letters are different than lower case letters.
ex. MrG mrg MRG mrG are all 4 different, distinct identifiers that cannot be interchanged

Proper identifiers: x, x1, X1, x_1, _x1, XX, x1X1, X12_y13, sum, rateOne, PI

Improper identifiers: void, 1x, five\$, hi there, int, Mr.G., 56, PROG.CPP

II. Data Types - a specific set of data values along with a set of operations on those values

Note: Every variable (identifier) in a Java program must be declared (see the next section for the proper syntax for how to do that). When declaring the variable you are telling the computer what it represents by the kind of data you will be storing in it.

We will only be concerned at this time with a few of these:

1) Primitive Data Types

A) Integer Types

1) **int** (range: -2,147,483,648 to 2,147,483,647)

2) **long** - also called **long int**

(range: -9223,372,036,854,775.808 to 9223,372,036,854,775.807)

ex. 0, 1, -5, 15000 (note: no comma allowed)

(Note: There are two other integer types, that we will not use, called short and byte.)

B) Floating Point Types

1) **float** (approximate range: -3.4×10^{-38} to 3.4×10^{38})

2) **double** (approximate range: -1.8×10^{-308} to 1.8×10^{308})

(Note: There are objects, that we will not use, that allow any size of number to be used called BigInteger and BigDecimal.)

Note: Earlier C++ Java books mainly use float while later ones, due to less concern with running out of memory, use double.

ex) .3, -127.54, -0.4, 1.5e3, -4.5678e-2, 18.0

(Note: The **.0** must be there or it will be considered an integer type.)

2) Other Data Types

A) **char** (short for character) - consists of a single character from the keyboard designated by enclosing it in single quotes

(Note: This is a not nonnumeric type in Java but is in C++.)

ex.) 'a', 'A', '&', ' ' (a blank space - whitespace), '5'

B) **boolean** - consists the word true or false

(Note: There is no **boolean** type in C++.)

III. Variable Declarations - a statement that associates an identifier with a data type (so that the programmer can refer to that item later in the program by that name)

A) Declaring a single variable:

ex.1) **double** rate ;

ex.2) **int** count ;

Note: These are computer statements and therefore must end in a semicolon.

Note: Sometimes it is easier to read these lines from right to left.

Note: Every variable must be declared before that variable can be used.

You can declare the variable anywhere in the program preceding its use.

By convention we usually declare the variables:

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

right after the starting brace { (my preference) or

b) immediately preceding the usage of the variable

B) Declaring multiple variables in a single line - separate them by a comma:

ex.1) **int** time, hours ;

ex.2) **double** pay, wages2, myPay ;

Note: To be declared in the same line all the variables must be of the same type.

IV. Assignment Statement - a statement that stores the value of an expression into a variable

A) Syntax: Variable = expression ;

ex.1) **int** number ; // the declaration statement
number = 10 ;

ex.2) **double** payRate ; // the declaration statement
payRate = 5.55 ;

ex.3) **char** yesNo ; // the declaration statement
yesNo = 'Y' ;

ex.4) **boolean** repeat ; // the declaration statement
repeat = false ;

B) Semantic (meaning): The value or expression on the right is evaluated and then this result is assigned or stored in the computer's memory (like a post office box) with the variable on the left side as the memory location's name.

Note: You should read assignment statements from right to left and think of the = as an assignment not as an equal representing equality. See V. for why.

Computer Programming I - Unit 1 Lecture

V. Operators - symbols that perform a predefined function on the data with which it is used

A) Assignment operator (=): (Note: see page 69 in the blue book)

B) Arithmetic *Binary* Operators:

<u>Operation</u>	<u>Algebra</u>	<u>Computer</u>
Addition	+	+
Subtraction	-	-
Multiplication	X, ·, (), xy, 2x	*
Division	$2\overline{)6}$, \div , $\frac{x}{y}$	/ (see below for further explanation) or % (see below for further explanation)

1) / (division)

a) The forward slash / for division when dividing two integers just gives only the integer quotient for the result

ex.1) $8 / 2$ is 4 but

ex.2) $7 / 2$ is 3 the decimal part of the answer is not given

b) When dividing two floating point decimals the result is a floating point decimal.

ex.1) $7.0 / 2.0$ is 3.5

2) % (called the *modulus*) - the % symbol is used only for integer division and the result is the remainder

ex.1) $7 \% 2$ is 1

ex.2) $3 \% 5$ is 3

ex.3) $5 \% 5$ is 0

More assignment statements (assume all variables have already been declared):

ex.1) `number = 2 + 5 ;`

ex.2) `overtimePay = 1.5 * pay ;` // What would happen if the * were left out?

ex.3) `salary = rate * hours + overtimePay ;`

☆ex.4) `count = count + 1 ;`

☆ex.5) `down = down - 1 ;`

Note: You can only have a single variable on the left side of the = sign.

Invalid assignment statements:

ex.1) `x + 5 = y ;`

ex.2) `count + 1 = count ;`

ex.3) `y = 2x ;` // Why invalid?

V. *Operators* - continued

C) *Unary Arithmetic Operators* (ones that take just one operand):

1) simple ones: + and -

ex.1) -7

ex.2) +7 (Note: The + as a unary operator is rarely used since if there is no sign then the number is automatically positive.)

VI. *Named Constants* (also called a declared or symbolic constant.) -

a location in memory, referenced by an identifier, where the data value that is stored there cannot be changed by a different assignment statement

Syntax: **final** DataType Identifier = LiteralValue;

(Note: This statement must both declare and assign a value to a variable.)

ex.1) **final int** MAXNUMBER = 3;

ex.2) **final char** BLANK = ' ' ; (Note: This represents a blank.)

ex.3) **final double** PI = 3.14;

ex.4) **final double** PI;
PI = 3.14;

ex.5) **final double** PI = 3.14;
PI = 3.2415; // this is an error

Note: By convention the constant identifiers use all capital letters and variable identifiers use lower case letters except caps are used for runTogether variables.

Note: The constant statement can appear either before the `main()` statement or somewhere after the starting brace `{`, before the constant is used in the program. (More about its placement in a later chapter.)

VII. Initializing Variables - giving a value to a variable at the time that it is declared

Note: This is the same as an assignment statement.

A) Observe the following program portion:

```
{
  int firstNum, secondNum, firstAnswer, secondAnswer;    // variable declaration
  firstNum = 10;                                         // variable initialization
  firstAnswer = firstNum;                                // assigning an initialized value to a variable
  secondAnswer = secondNum;                              // oops
}
```

What is the value of firstAnswer and secondAnswer ?

A variable has no meaningful value until a program gives it one. If no value is given then the variable has a "garbage value" - a random number left in the memory or a number from a previous program which has no meaning and usefulness in the current program. GIGO (Garbage In Garbage Out)

B) Initializing Variables in Declarations (a shortcut)

```
ex.1a) double number ;           // the declaration statement
       number = 10.5 ;           // the initialization statement
```

or

```
ex.1b) double number = 10.5 ; /* the declaration and initialization
                               combined together */
```

```
ex.2a) int x, y, z ;             // the declaration statement
       x = 3 ;                   // initialization statement
       y = 4 ;                   // initialization statement
       z = 5 ;                   // initialization statement
```

or

```
ex.2b) int x = 3, y = 4, z = 5 ; // the declaration and initialization combined together
```

C) Multiple Assignments or Initializing of Variables with the Same Value (a shortcut)

```
ex.1a) int x, y, z ;             // the declaration statement
       x = y = z = 3 ;           // assignment/initialization statement
```

VIII. Output Introduction - c.println();

```
ex.) import hsa.Console;
    public class PrintName
    {
        static Console c;                // The output console
        public static void main(String[ ] args)
        {
            c = new Console( );
            c.println("My name is Bob.");
            c.print(" Hello World!");
        }                                // end of the main method
    }                                    // end of the PrintName
    class
```

A) **print()** and **println()** - are *methods* (like a function) that performs some action which is to show something on an output screen. It must be associated (connected) to what Java calls an *object* by a period actually called a *dot operator*. The object is `c`. If using our lab book to output you must have the import statement of **import hsa.Console;**. An object can only be used only if we know what *Class* it is from. The class for `c` (or screen) is the Console class. The Console class is written by the author of our lab book and creates an output window screen designed by him. (More about Classes and objects later.)

B) `print()` and multiple outputs in the same line

```
ex.1a) c.print("The answer is "); // Note the blank space after the word is
        c.print(5);
        c.print(".");
```

OUTPUT: The answer is 5.

Note: Normally all lines output (`print`) immediately after the preceding output if using a `print()` and will be on separate lines if using `println()` - see next overhead
OR

```
ex.1b) print("The answer is " + 5 + ".");
```

OUTPUT: The answer is 5.

This last ex.) is an example called *concatenation* i.e. connecting strings together. The `+` makes everything into one long string that is then output. The `+` is a special method that is part of the String class which is found in the **java.lang** package which is a package that is automatically imported into all Java programs (so we never need to import it) Note: **java.lang** also contains some Math classes and methods as we will find out later and the also contains the System class.

VIII. *Output Introduction* continued

C) Starting New Lines in Output (two methods)

- 1) To start a new output line, you can include `\n` (called an *escape character*) in a quoted literal string. The `\n` is typed as two symbols with no space between them.

```
ex.1) print("Hi\n");  
      print("Bye\n");
```

OR

```
ex.2a) print("Hi\n" +  
            "Bye\n");
```

```
or ex.2b) print("Hi\n" + "Bye\n");
```

OUTPUT: Hi
Bye

- 2) You can also start a new line by using **println()**

```
ex.1) println("Hi");  
      println("Bye");
```

OUTPUT: Hi
Bye

- 3) Both in one example

```
ex.1a) print("The answer is\n");  
      println(5);  
      println("Bye"); // or print("Bye");
```

OR

```
ex.1b) println("The answer is");  
      print(5 + '\n'); // or print(5 + "\n");  
      println("Bye"); // or print("Bye");
```

OR

```
ex.1c) println("The answer is");  
      println(5);  
      println("Bye"); // or print("Bye");
```

OUTPUT: The answer is
5
Bye

Note: The last example is much better!

C) More Escape characters - \t, \", \', \\,
(Note: see page 66 in the blue book for these and more that we will not study)

- 1) \t - tabs over
- 2) \" - shows a double quote mark ”
- 3) \' - shows a single quote mark ‘
- 4) \\ - shows a single backslash mark \

ex.1) `c.print("Hi\\\'Bye\''")`

Output: `Hi\'Bye"`

IX. Program Construction, Structure, and Layout

```
ex.) // Area of Circle Program
    /* Lecture Example
       for Programming I */
    import hsa.Console; // or import hsa.*;
    public class SomeClassName
    {
        static Console c;
        public static void main(String[ ] args)
        {
            c = new Console( );
            final double PI = 3.14;
            double area;
            int radius = 2;
            area = PI * radius * radius;
            c.print("area = " + area);
        }
    }
```

- A) Comments (two types) - comments or remarks that are ignored by the computer.
(Note: These are optional but recommended.)
- 1) // single line comment
 - 2) /* ... */ single or multiple line comment

B) **import** statement - To create an object, you must specify the class from which the object is to be created. The class can be located in one of two locations. It can be located in the same directory as the program, or it can be located in a group of classes called a package. If the class is located in the same directory as the program, the Java compiler finds it automatically. If the class is located in a package, the program must specify the name of the package using an import statement. The format of the import statement is: **import** [package-name].[Class-name];

This text uses classes from two sources: packages that are part of the Java class library and the package supplied with this book. There are a number of packages in the Java class library and they all have names that start with java. For example, the classes used to read from and write to files are all located in the java.io package. Several of the classes used in this book are created specifically for use in a teaching environment. These classes are all located in the *hsa* package. For example, to use the *Console* class, you must include the statement **import hsa.Console;**

- C) **public class** SomeClassName - all Java programs must be started by a heading which contains the name of the class. The modifier **public** is used so that other programs can access this one, if necessary.
- D) Braces { and } - they mark the start and end of a block of code
Note: A block is a sequence of zero or more statements enclosed by a { } pair.
- E) Output Shortcut - the two lines **static** Console c; and c = **new** Console(); are used to create a new object called c which is of Console class type. This new Console object is the output window on which the results of programs will be displayed. Yes this is a shortcut; we will see in a later chapter how to output without using the Console class supplied by our book's author but by using the System class (supplied in all Java programs found in java.lang)
Note: The modifier **static** is an advanced topic studied in the AP class. You can read about it on page 57 of our blue book.
Note: The reserved word **new** is always used when creating (instantiating) an object. More about creating objects later.
- F) **public static void** main(String[] args) - a method
this is the heading of the *main* method that this class contains. In any Java program one of the classes must contain a *main* method and there can only be one of these! This is the point where the computer actually starts carrying out the program. The three modifiers **public static void** must be used and again is an advanced topic. the variable args (which could be any variable but by convention everyone uses args) is called a parameter and is of type String array which is another advanced topic.
Note: The class file that contains the *main* method is called the application or client file or program.
Note: A method heading does NOT end in a semicolon!
- E) Statement(s) - special executable computer instructions ending with a semicolon
Note: The null statement is a line consisting of only a semicolon, which does nothing, but is allowed.

X. Introduction to Objects - Terminology

Recall: Data Types - **int, char, double, float** ex.) **int** number;

- A) Object - a "special" variable created from a *class* not from a built-in data type
- B) Class - a programmer-defined data type out of which objects and methods can be created
- C) Member Methods - functions associated with a class that can only be accessed (called) by the objects also in that class

Dot Operator syntax- the period (dot) used in the call to a member method that connects the object to the member method. It starts with the class object (variable), then a dot, then the member method name, then the argument.

ex.) **c.print(x)**

c is an object of the Console class and `println()` is a member function of the Console class

//Example of a programmer defined class:

//Only a portion of the application and implementation files for the Example class

```
import hsa.Console; // or import hsa.*;
public class Example
{
    static Console c;
    public static void main(String[ ] args)
    {
        c = new Console( );
        Circle circleA, circleB;
        circleA = new Circle( );
        circleB = new Circle( );
        circleA.output( );
        circleB.output( );
        circleA.area( );
        circleB.area( );
    }
    public class Circle
    {
        public void output( )
        {
            c.println("Center is " + x + "," + y);
            c.println("Radius is " + r);
        }
        public void area( )
        {
            double pi = 3.14;
            c.println("The area is " + pi*r*r);
        }
    }
}
```

Name the following from the example above:

1) the three classes (i.e. the *type* like **int** or **double**)

Answer: Console, Example, Circle

2) the three objects (i.e. the *variables* of the class)

Answer: c, circleA, circleB

3) the three class member methods associated with the objects

Answer: println() , output() , area()