CSCI 125 Lecture 2

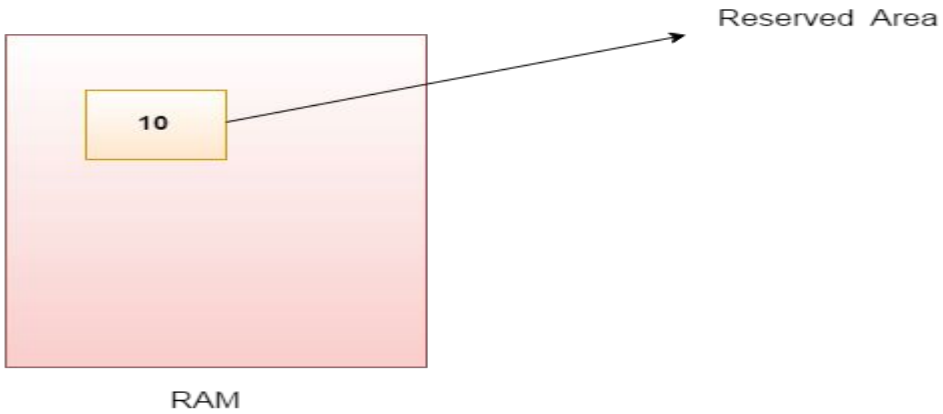
Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in Java: primitive and non-primitive.

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.



Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

PRe-req

What is a Bit?

By its simplest definition, a bit is just a smaller unit of information than a byte. It reflects the basic logical process of a transistor: a single unit of information reflecting a zero (no charge) or a one (a completed, charged circuit). There are eight bits in one byte of information. Alternately, and more commonly in modern connotations, bits (and their successively larger relatives, such as kilobits, megabits and gigabits) are used to measure rates of data transfer. The abbreviation “Mbps” is one of the most commonly misinterpreted in all of modern computing: it refers to “megabits,” not “megabytes,” per second.

What is a Byte?

A byte represents eight bits, and is the most commonly used term relating to the amount of information stored within a computer's memory. The term doesn't refer to "eight bits" in a loose, simply mathematical sense, but to a specific set of eight bits which operate as a cohesive unit within a computer system. The byte was first named in 1956, during the design of the IBM Stretch computer. At the time, it was more closely related to "bit;" its name is a deliberate misspelling of "bite" in order to avoid accidental confusion. When abbreviated, the "B" is capitalized, so as to set it apart from its smaller relative; "Gb" is short for "gigabit," and "GB" is short for "gigabyte."

Number base

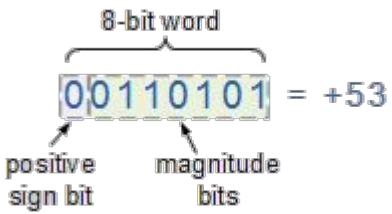
A number base is the number of digits or combination of digits that a system of counting uses to represent numbers. A base can be any whole number greater than 0. The most commonly used number system is the decimal system, commonly known as base 10. Its popularity as a system of counting is most likely due to the fact that we have 10 fingers.

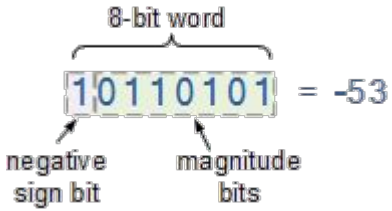
Binary is the most commonly used non-base 10 system. It is used for coding in computers. Binary is also known as Base 2. This means it is composed of only 0's and 1's. For example 9 in binary/base 2 is 1001. Let's see how this works.

Base 10	Base 2	2^4	2^3	2^2	2^1	2^0
1	**1**	0	0	0	0	1
9	**1001**	0	1	0	0	1
16	**10000**	1	0	0	0	0

Signed

in digital circuits there is no provision made to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of “0’s” and “1’s”. When used together in microelectronics, these “1’s” and “0’s”, called a **bit** (being a contraction of **BI**nary digi**T**), fall into several range sizes of numbers which are referred to by common names, such as a *byte*





Primitive Data Types

There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000
-

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

Object Data Types

-

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.
- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.
- Example: Animal animal = new Animal("giraffe");

Java Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable. For example

```
byte a = 68;
```

```
char a = 'A';
```

byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.

Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example −

```
int decimal = 100;
```

```
int octal = 0144;
```

```
int hexa =  0x64
```

String and char types of literals can contain any Unicode characters. For example −

```
har a = '\u0001';
```

```
String a = "\u0001";
```

Java language supports few special escape sequences for String and char literals as well. They are −

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

Unicode System

Why java uses Unicode System?

Before Unicode, there were many language standards:

ASCII (American Standard Code for Information Interchange) for the United States.

ISO 8859-1 for Western European Language.

KOI-8 for Russian.

GB18030 and BIG-5 for chinese, and so on.

a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

Java - Variable Types

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration −

```
data type variable [ = value][, variable [ = value] ...] ;
```

```
int a, b, c;          // Declares three ints, a, b, and c.
int a = 10, b = 10;   // Example of initialization
byte B = 22;          // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
 char a = 'a';         // the char variable a iis initialized with value 'a'
```

Operators in Java

Operator in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

```
10
12
12
10
```

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros	A >>>2 will give 15 which is 0000 1111

The Logical Operators

Assume Boolean variables A holds true and variable B holds false, then −

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Miscellaneous Operators

Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

```
variable x = (expression) ? value if true : value if false
```

```
public class Test {

   public static void main(String args[]) {
      int a, b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

```
Value of b is : 30
Value of b is : 20
```

First Java Program | Hello World Example

```
public class MyFirstJavaProgram {


   /* This is my first java program.
    * This will print 'Hello World' as the output
    */


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

About Java programs, it is very important to keep in mind the following points.

- Case Sensitivity − Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.
- Class Names − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  Example: *class MyFirstJavaClass*
- Method Names − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  Example: *public void myMethodName()*

- Program File Name − Name of the program file should exactly match the class name.
  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).
  But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.
  Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

- public static void main(String args[]) − Java program processing starts from the main() method which is a mandatory part of every Java program.

The main() Method

A Java program needs to start its execution somewhere. A Java program starts by executing the main method of some class. You can choose the name of the class to execute, but not the name of the method. The method must always be called main.

The three keywords public, static and void have a special meaning. Don't worry about them right now. Just remember that a main() method declaration needs these three keywords.

After the three keywords you have the method name. To recap, a method is a set of instructions that can be executed as if they were a single operation. By "calling" (executing) a method you execute all the instructions inside that method.

The Java Main Class

If only a single Java class in your Java program contains a main() method, then the class containing the main() method is often referred to as the *main class*.

You can have as many classes as you want in your project with a main() method in. But, the Java Virtual Machine can only be instructed to run one of them at a time. You can still call the other main() methods from inside the main() method the Java Virtual Machine executes (you haven't seen how yet) and you can also start up multiple virtual machines which each execute a single main() method.

System.out.println()

Inside the `main()` method, we can use the `println()` method to print a line of text to the screen:

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

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

  }
```

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by Java (will not be executed).

Java Output

```
System.out.println(); or

System.out.print(); or

System.out.printf();
```

- `print()` - It prints string inside the quotes.
- `println()` - It prints string inside the quotes similar like `print()` method. Then the cursor moves to the beginning of the next line.
- `printf()` - Tt provides string formatting

```
class Output {

    public static void main(String[] args) {



        System.out.println("1. println ");

        System.out.println("2. println ");



        System.out.print("1. print ");

        System.out.print("2. print");

    }
```

```
1. println

2. println

1. print 2. print
```

Example: Printing Variables and Literals

```
class Variables {

    public static void main(String[] args) {

        Double number = -10.6;

        System.out.println(5);

        System.out.println(number);

    }

  }
```

When you run the program, the output will be:

```
5

-10.6
```

Example: Print Concatenated Strings

```
class PrintVariables {

    public static void main(String[] args) {



        Double number = -10.6;



        System.out.println("I am " + "awesome.");

        System.out.println("Number = " + number);

    }

  }
```

Output

```
I am awesome.

Number = -10.6
```

```
import java.util.Scanner;



class Input {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = input.nextInt();

        System.out.println("You entered " + number);

        // closing the scanner object

        input.close();

    }

  }

  Output
  Enter an integer: 23

  You entered 23
```

Example: Get float, double and String Input

```
import java.util.Scanner;

class Input {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        // Getting float input
        System.out.print("Enter float: ");
        float myFloat = input.nextFloat();
        System.out.println("Float entered = " + myFloat);
        // Getting double input
        System.out.print("Enter double: ");
        double myDouble = input.nextDouble();
        System.out.println("Double entered = " + myDouble);
        // Getting String input
        System.out.print("Enter text: ");
        String myString = input.next();
        System.out.println("Text entered = " + myString);    }}
```

```
Enter float: 2.343
Float entered = 2.343
Enter double: -23.4
Double entered = -23.4
Enter text: Hey!
Text entered = Hey!
```

LAB

Git Hub

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

This tutorial teaches you GitHub essentials like *repositories*, *branches*, *commits*, and *Pull Requests*. You'll create your own Hello World repository and learn GitHub's Pull Request workflow, a popular way to create and review code.

Step 1. Create a Repository

A **repository** is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a *README*, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. *It also offers other common options such as a license file.*

Your `hello-world` repository can be a place where you store ideas, resources, or even share and discuss things with others.

To create a new repository

1. In the upper right corner, next to your avatar or identicon, click and then select **New repository**.
2. Name your repository `hello-world`.
3. Write a short description.
4. Select **Initialize this repository with a README**.

Step 2. Create a Branch

Branching is the way to work on different versions of a repository at one time.

By default your repository has one branch named `main` which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to `main`.

When you create a branch off the `main` branch, you're making a copy, or snapshot, of `main` as it was at that point in time. If someone else made changes to the `main` branch while you were working on your branch, you could pull in those updates.

To create a new branch

1. Go to your new repository `hello-world`.
2. Click the drop down at the top of the file list that says **branch: main**.
3. Type a branch name, `readme-edits`, into the new branch text box.
4. Select the blue **Create branch** box or hit "Enter" on your keyboard.

Step 3. Make and commit changes

On GitHub, saved changes are called *commits*. Each commit has an associated *commit message*, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why.

1. Click the `README.md` file.
2. Click the pencil icon in the upper right corner of the file view to edit.
3. In the editor, write a bit about yourself.
4. Write a commit message that describes your changes.
5. Click **Commit changes** button.

Step 4. Open a Pull Request

Pull Requests are the heart of collaboration on GitHub. When you open a *pull request*, you're proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show *diffs*, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

By using GitHub's @mention system in your pull request message, you can ask for feedback from specific people or teams, whether they're down the hall or 10 time zones away.

You can even open pull requests in your own repository and merge them yourself. It's a great way to learn the GitHub flow before working on larger projects.

Step 5. Merge your Pull Request

In this final step, it’s time to bring your changes together – merging your `readme-edits` branch into the `main` branch.

1. Click the green **Merge pull request** button to merge the changes into `main`.
2. Click **Confirm merge**.
3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.

Home work

Create a Project

Create a class that prints your name, email, student id

Create a class that adds two integers and prints the sum

Create a class that takes an integer and prints the integer + 1

Create your github repository

Upload code on github

Refrences

https://www.computersciencedegreehub.com/faq/what-is-the-difference-between-a-bit-and-a-byte/