Lecture 01
Java Fundamentals

Dr. Ahmed ElShafee

Agenda

1. Java program structure
2. Basic Syntax
3. Java Identifiers
4. Java Keywords
5. Comments in Java
6. Java Basic Data Types
7. Java Basic Operators
8. Java IO
9. Casting
10. Java Loops
11. Java Decision Making

Java program structure

- a simple code that would print the words *Hello World*.

```
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
   }
}
```


Basic Syntax

- 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 words 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).
  - 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..

Java Identifiers

- All java components require names.
- Names used for classes, variables and methods are called identifiers.
- In java there are several points to remember about identifiers. They are as follows:
  - All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (-).
  - After the first character identifiers can have any combination of characters.
  - A key word cannot be used as an identifier.
  - Most importantly identifiers are case sensitive.
  - Examples of legal identifiers:age, $salary, _value, __1_value
  - Examples of illegal identifiers : 123abc, -salary

Java Keywords:

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

Comments in Java

```
public class MyFirstJavaProgram{

   /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */

   public static void main(String []args){
      // This is an example of single line comment
      /* This is also an example of single line comment. */
      System.out.println("Hello World");
   }
}
```


Java Basic Data Types

- Variables are reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.
- Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.
- There are two data types available in Java:
  - Primitive Data Types
  - Reference/Object Data Types

1. Primitive Data Types:

- There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word.
- **byte:** Byte data type is a 8-bit signed two's complement integer.
- **short:** Short data type is a 16-bit signed two's complement integer.
- **int:** Int data type is a 32-bit signed two's complement integer.
- **long:** Long data type is a 64-bit signed two's complement integer.
- **float:** Float data type is a single-precision 32-bit IEEE 754 floating point.

- **double:** double data type is a double-precision 64-bit IEEE 754 floating point.
- **boolean:** boolean data type represents one bit of information.
- **char:** char data type is a single 16-bit Unicode character.

2. Reference Data Types:

- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to 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 indicates 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 literals in Java are a sequence of characters between a pair of double quotes.

```
"Hello World"
"two\nlines"
"\"This is in quotes\""
```

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

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


- special escape sequences for String and char literals as well.

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)

Java Basic Operators

- Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:
  - Arithmetic Operators
  - Relational Operators
  - Bitwise Operators
  - Logical Operators
  - Assignment Operators
  - Misc Operators

The Arithmetic Operators:

- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:
- Assume integer variable A holds 10 and variable B holds 20 then:

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 - Increase the value of operand by 1	B++ gives 21
--	Decrement - Decrease the value of operand by 1	B-- gives 19


```
class MySecondJavaProgram{
public static void main(String args[]) {
  int a = 10;
  int b = 20;
  int c = 25;
  int d = 25;
  System.out.println("a + b = " + (a + b) );
  System.out.println("a - b = " + (a - b) );
  System.out.println("a * b = " + (a * b) );
  System.out.println("b / a = " + (b / a) );
  System.out.println("b % a = " + (b % a) );
  System.out.println("c % a = " + (c % a) );
  System.out.println("a++   = " +  (a++) );
  System.out.println("b--   = " +  (a--) );
```

```
 // Check the difference in d++ and ++d
   System.out.println("d++   = " +  (d++) );
   System.out.println("++d   = " +  (++d) );
}
}
```

```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```


The Relational Operators:

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

```
class MyThirdJavaProgram{
public static void main(String args[]) {
   int a = 10;
   int b = 20;
   System.out.println("a == b = " + (a == b) );
   System.out.println("a != b = " + (a != b) );
   System.out.println("a > b = " + (a > b) );
   System.out.println("a < b = " + (a < b) );
   System.out.println("b >= a = " + (b >= a) );
   System.out.println("b <= a = " + (b <= a) );
}
}
```



```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

Java IO

print & println

```
String first_name=“zaki”;
String family_name=“gedan”;
System.out.println(first_name + " " + family_name);
System.out.print(first_name + “ ” + family_name);
```

```
String first_name=“zaki”;
String family_name=“gedan”;
Char space=“ “;
System.out.println(first_name + space + family_name);
System.out.print(first_name + space + family_name);
```


printf (formatted string)

Data type	symbol	function
char	%c	Sysetm.out.printf(“this is char: %c”, ch);
String	%s	Sysetm.out.printf(“my name is %s %s”, first_name, last_name);
integers	%d	Sysetm.out.printf(“%d + %d = %d”, num1, num2, answer);
double	%f	Sysetm.out.printf(“%f + %f = %f”, num1, num2, answer);

Data type	symbol	function
short	%d	Sysetm.out.printf(“%d + %d = %d”, num1, num2, answer);
float	%f	Sysetm.out.printf(“%f + %f = %f”, num1, num2, answer);
byte	%d	Sysetm.out.printf(“%d + %d = %d”, num1, num2, answer);
Boolean	%b	Sysetm.out.printf(“yes this is %b”, 1);
Hexadecimal	%x	Sysetm.out.printf(“0x%x + 0x%x = 0x%x”, num1, num2, answer);
Octal	%o	Sysetm.out.printf(“%o + %o = %o”, num1, num2, answer);

Scanners

1. import library

```
import java.util.Scanner;
```

2. Create object from scanner

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

3. Accepting string input from user

```
String family_name = user_input.next( );
```

* You can accept any type of inputs from the user as follows

```
x = user_input.nextXYZ()
```

where XYZ is one of BigDecimal, BigInteger, Boolean, Byte, Float, or Short

```
 class MyThirdJavaProgram2{
public static void main(String[] args) {
      String first_name, last_name, comment;
      int id,level, gained_credit_hours;
      float GPA;
      Scanner s=new Scanner(System.in);
      System.out.printf("\nEnter Your ID (ineteger):");
      id=s.nextInt();
      System.out.printf("\nEnter Your first name:");
      first_name=s.next();
      System.out.printf("\nEnter Your family name:");
      last_name=s.next();
      System.out.printf("\nEnter Your Level(1,2,3,4):");
      level=s.nextInt();
      System.out.printf("\nEnter Your Gained Credit hours till now (integer):");
```



```
 gained_credit_hours=s.nextInt();
System.out.printf("\nEnter Your your GPA (float):");
      GPA=s.nextFloat();
      System.out.printf("\ntype your comments (ended by Enter):");
      comment=s.next();

System.out.printf("\n**************************************************"
);
      System.out.printf("\nID\t%d\nname\t%s %s\nLevel\t%d\nCH\t%d\nGPA\t%f",
            id, first_name, last_name, level, gained_credit_hours, GPA);
      System.out.println("\nCommnet:\n"+comment);
   }
}
```



```
Enter Your ID (ineteger):4091001
Enter Your first name:mosfata
Enter Your family name:diaa
Enter Your Level(1,2,3,4):4
Enter Your Gained Credit hours till now (integer):3
Enter Your your GPA (float):3.5
type your comments (ended by Enter):no comment
**************************************************
ID      4091001
name    mosfata diaa
Level   4
CH      123
GPA     3.500000
Commnet:
no
```


JOptionPane

1. Import library

```
import javax.swing.JOptionPane;
```

2. Show message

```
JOptionPane.showMessageDialog( null, full_name );
```

3. Accept input string

```
first_name = JOptionPane.showInputDialog("First Name“,"Enter Your Family Name");
```

Casting

Converting string to integer

```
String string_num="10";
int num= Integer.parseInt( string_num );
```

Converting int to string

```
int n=10;
String str="this number equals to: "+n;
```

```
int n=10;
String strn=Integer.toString(n);
String str="this number equals to: "+strn;
```

That applies to all types of numerical variables


```
class MyThirdJavaProgram3{
public static void main(String[] args) {
    String user_input;
    float breadth, hight, area;
    user_input=JOptionPane.showInputDialog("Breadth","0");
    breadth=Float.parseFloat(user_input);
    user_input=JOptionPane.showInputDialog("Hight","0");
    hight=Float.parseFloat(user_input);
    area=breadth*hight;
    JOptionPane.showMessageDialog(null, "Rectangle Area = "+area,"Rectangle",
JOptionPane.INFORMATION_MESSAGE);
        }
}
```


Java Loops - for, while and do...while

- There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.
- Java has very flexible three looping mechanisms. You can use one of the following three loops:
  - while Loop
  - do...while Loop
  - for Loop

While loop

```
while(Boolean_expression)
{
  //Statements
}
```

- When executing, if the *boolean_expression* result is true then the actions inside the loop will be executed.
- This will continue as long as the expression result is true.
- Here key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

```
public class MyFourthJavaProgram{
   public static void main(String args[]){
      int x= 10;

      while( x < 20 ){
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```


The do...while Loop:

```
do
{
   //Statements
}while(Boolean_expression);
```

- Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.
- If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again.
- This process repeats until the Boolean expression is false.


```
public class MyFifthJavaProgram{
   public static void main(String args[]){
      int x= 10;

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```


The for Loop:

```
for(initialization; Boolean_expression; update)
{
   //Statements
}
```

- The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.

- After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step,then Boolean expression). After the Boolean expression is false, the for loop terminates.


```
public class My6thJavaProgram{
  public static void main(String args[]){

    for(int x = 10; x < 20; x = x+1){
      System.out.print("value of x : " + x );
      System.out.print("\n");
    }
  }
}
```

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```


Enhanced for loop in Java:

- As of java 5 the enhanced for loop was introduced. This is mainly used for Arrays.

```
for(declaration : expression)
{
  //Statements
}
```

- **Declaration** . The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression** . This evaluate to the array you need to loop through. The expression can be an array variable or method call that returns an array.


```
public class My7thJavaProgram{
  public static void main(String args[]){
    int [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers ){
      System.out.print( x );
      System.out.print(",");
    }
    System.out.print("\n");
    String [] names ={"James", "Larry", "Tom", "Lacy"};
    for( String name : names ) {
      System.out.print( name );
      System.out.print(",");
    }
  }
}
```

```
10,20,30,40,50,
James,Larry,Tom,Lacy,
```

- **The break Keyword:**
- The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.
- The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.
- The syntax of a break is a single statement inside any loop:

```
break;
```



```
public class My8thJavaPogram{
   public static void main(String args[]){
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ){
         if( x == 30 ){
                break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

```
10
20
```


The continue Keyword:

- The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.
- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

```
continue;
```



```
public class My9thJavaProgram{
   public static void main(String args[]){
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ){
         if( x == 30 ){
                continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

```
10
20
40
50
```

Java Decision Making

The if Statement:

- An if statement consists of a Boolean expression followed by one or more statements.

```
if(Boolean_expression)
{
   //Statements will execute if the Boolean expression is true
}
```

- If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement(after the closing curly brace) will be executed.


```
public class My10thJavaProgram{
  public static void main(String args[]){
    int x = 10;

    if( x < 20 ){
      System.out.print("This is if statement");
    }
  }
}
```

```
This is if statement
```


The if...else Statement:

- An if statement can be followed by an optional *else if...else* statement, which is very usefull to test various conditions using single if...else if statement.

```
if(Boolean_expression 1){
   //Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2){
   //Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3){
   //Executes when the Boolean expression 3 is true
}else {
   //Executes when the none of the above condition is true.
}
```



```
public class My11thJavaProgram{
  public static void main(String args[]){
    int x = 30;

    if( x == 10 ){
      System.out.print("Value of X is 10");
    }else if( x == 20 ){
      System.out.print("Value of X is 20");
    }else if( x == 30 ){
      System.out.print("Value of X is 30");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

```
Value of X is 30
```

Nested if...else Statement:

- It is always legal to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement.

```
if(Boolean_expression 1){
   //Executes when the Boolean expression 1 is true
   if(Boolean_expression 2){
      //Executes when the Boolean expression 2 is true
   }
}
```



```
public class My12thJavaProgram{
  public static void main(String args[]){
    int x = 30;
            int y = 10;

    if( x == 30 ){
      if( y == 10 ){
      System.out.print("X = 30 and Y = 10");
    }
  }
}
```

```
X = 30 and Y = 10
```


The switch Statement:

- A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

```
switch(expression){
   case value :
      //Statements
      break; //optional
   case value :
      //Statements
      break; //optional
   //You can have any number of case statements.
   default : //Optional
      //Statements
}
```


- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
- When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.


```
public class My13thJavaProgram{
  public static void main(String args[]){
    char grade = args[0].charAt(0);

    switch(grade)
    {
    case 'A' :
      System.out.println("Excellent!");
      break;
    case 'B' :
    case 'C' :
      System.out.println("Well done");
      break;
    case 'D' :
      System.out.println("You passed");
    case 'F' :
      System.out.println("Better try again");
      break;
    default :
      System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```



```
$ java Test a
Invalid grade
Your grade is a a
$ java Test A
Excellent!
Your grade is a A
$ java Test C
Well done
Your grade is a C
$
```