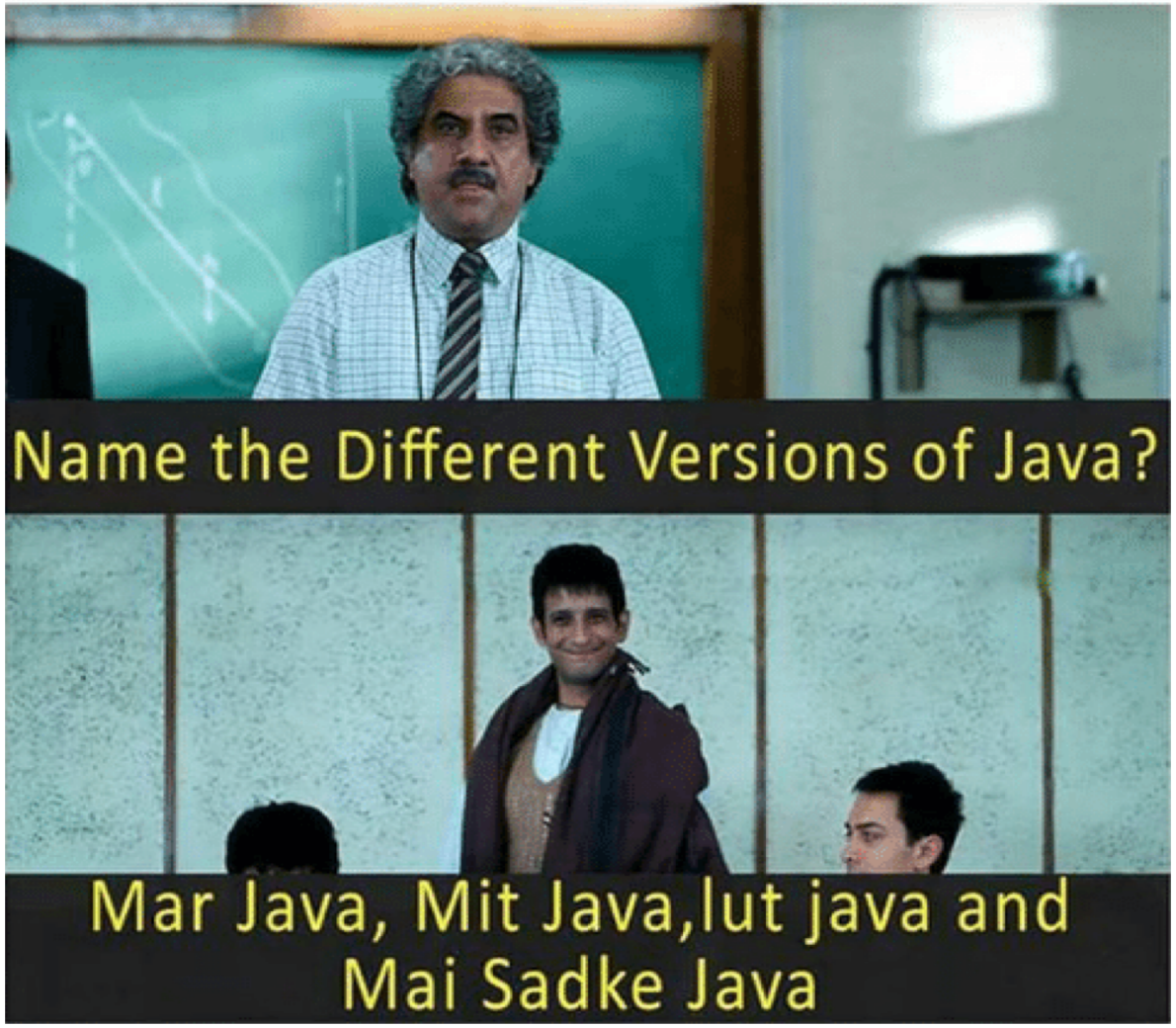
PROGRAMMING

Lecture 2

Sushil Paudel

OH NO!


Name the Different Versions of Java?
Mar Java, Mit Java,lut java and
Mai Sadke Java

PREVIOUS TOPICS

- Language
- Programming language
- Introduction to Java
- Text editors - Notepad, BlueJ
- Java fields
- Java Prerequisites

TODAY'S TOPICS

- Separators
- Comments
- Identifiers
- Keywords
- Java Class Library
- Syntax
- Indentation
- Variable
- Data types
- Operators

SEPARATORS

- Separators are symbols that indicate the division and arrangement of groups of code.
- The structure and function of code is generally defined by the separators.
- The separators used in Java are-

PARENTHESES ( )

- It is used to enclose parameters in method definitions, and enclosing cast types.

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

BRACES { }

- It is used to define a block of code and to hold the values of arrays.

```
public class MyFirstJavaProgram {

	public static void main(String[] args){

		System.out.println("Hello world!");

		int[] numbers = {1, 2, 3, 4, 5};

	}

}
```

BRACKETS []

- It is used to declare array types

```
public class MyFirstJavaProgram {

	public static void main(String[] args){

		System.out.println("Hello world!");

	}

}
```

SEMICOLON

- It is used to separate statements.

```
public class MyFirstJavaProgram {

	public static void main(String[] args){

		System.out.println("Hello world!");

		System.out.println(”I am learning Java!");

	}

}
```

COMMA ,

- It is used to separate identifiers in a variable declaration or values in an array.

```
public class MyFirstJavaProgram {
	public static void main(String[] args){
		int a =10, b= 20;
		int[] numbers = {1, 2, 3, 4, 5};
	}
}
```

PERIOD .

It is used to separate package names from classes and subclasses and to separate a variable or a method from a reference variable.

```
public class MyFirstJavaProgram {

	public static void main(String[] args){

		System.out.println("Hello world!");

	}

}
```

DON’T BE LIKE THIS!

COMMENTS

- Comments are used for documenting the program.
- What is obvious today may not be obvious after sometime.
- What if someone else wants to read your codes?
- The compiler simply ignores the comments.
- Comments don't have any effect in the execution of the program.

SINGLE LINE COMMENT

- It reach till the end of the line only.
- Symbolized by //
- E.g.

```
// Defining the class MyFirstJavaProgram
public class MyFirstJavaProgram {
	public static void main(String[] args){
		System.out.println("Hello world!"); // Printing Hello World
	}
}
```

MULTI LINE COMMENTS

- It is used to write multiple lines of
- Symbolized by /* <comments> */ . E.g

```
public class Hello {
    public static void main (String args[]){
        System.out.println("Hello World");
        /*This is the
          multi line comments,
          I can write any lines of comments here*/
    }
}
```

DOCUMENTATION COMMENTS (JAVADOC)

- Javadoc/Documentation comments structure look very similar to a regular multi-line comment, but the key difference is the extra asterisk at the beginning.
- It is set inside the comment delimiters /** ... */ with one comment per class, interface, or member.
- The comment should appear right before the declaration of the class, interface or member.
- They can be processed by the javadoc tool to generate the API documentation for your classes.

DOCUMENTATION COMMENTS (JAVADOC)

```
/**
* This class just prints the Hello World in the main method
*
* @author  Sushil Paudel
* @version 1.0.1
*/
 public class Hello {

     public static void main (String args[]){

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

      }

 }
```

JAVA CLASS LIBRARY

- Java class library is a collection of thousands of classes included in the JDK and ready to be used.
- Once you have installed Java, you get them as part of the installation and can start building your application code up using the classes of Java Class Library.
  - Data Structures
  - Networking
  - Graphical User Interfaces
  - Image Processing & Multimedia
  - Files, etc.

SYNTAX

- Syntax is a grammatical rule to write programs in any language.
- Java throws compilation error if syntax is wrong.
- A program written using Java and Php have different syntax, but their meaning/output(Semantics) can be same.
  - E.g.

```
System.out.print("Hello World); // Java
print("Hello World"); // Python
Console.Write("Hello World"); // C#
```

INDENTATION

- Giving proper tab/space to similar/related elements to group them together.
- The compiler cares nothing about indentation, but if you indent well, you make your code so much easier to read, which means you'll make it easier to debug.
- It is to display how the braces match up and show the logic of the program in an organized fashion

INDENTATION

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

Above code is a nightmare

INDENTATION

```
public class MyFirstJavaProgram {

	public static void main(String[] args){

		System.out.println("Hello world!");

	}

}
```

KEYWORDS

- In Java certain words are reserved that give special functionality/meaning.
- These are predefined words by Java so it cannot be used as a variable or object name.

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

JAVA KEYWORDS LIST

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

VARIABLES

- Variables are containers that hold values(as in algebra) temporarily in memory.
- The value held by a variable can be changed but its data type remains the same.
- **Syntax for variable declaration**
  - Datatype variableName;
  - Eg.

```
int a;
String color = “red”;
Here, color is a variable that holds the value red.
```

2 MINS INTERMISSION

IDENTIFIERS

- Identifiers are combination of words, numbers and symbols used for naming **classes, methods and variables.**
- Identifiers consists of letters, digits and two special symbols, underscore(_) and dollar(**$**)
- E.g.

```
String firstName;
Double salary;
String bank_branch;
```

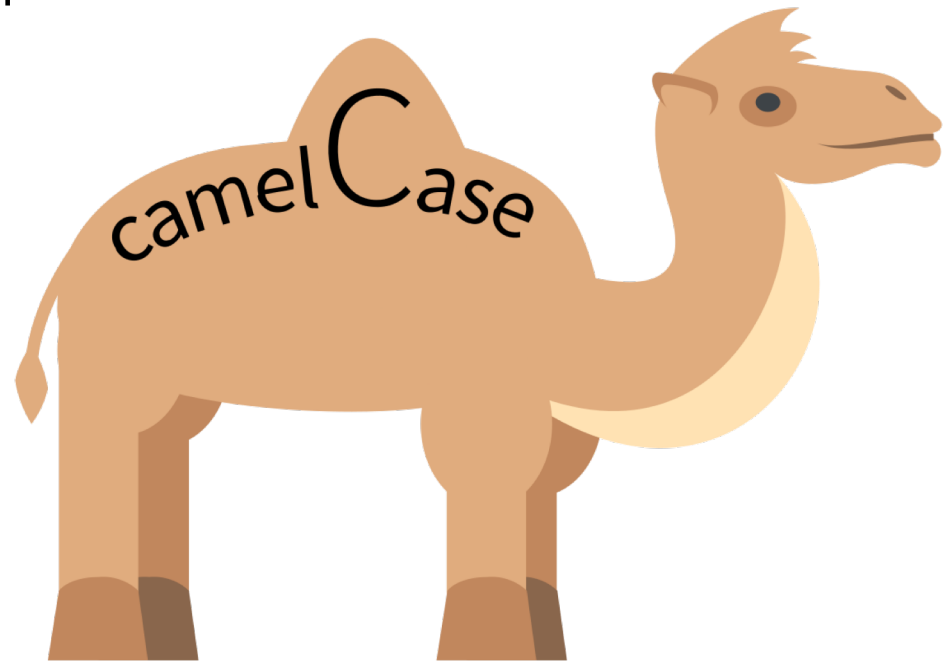
NAMING CONVENTION

Camel case in Java Programming : It consists of compound words or phrases such that each word or abbreviation begins with a capital letter or first word with a lowercase letter, rest all with capital.

bankaccountdetail

BankAccountDetail

myFirstName



RULES FOR IDENTIFIERS

- Identifiers cannot start with digits. E.g.

  String 1name; is wrong

- Identifiers should not contain spaces. E.g.

  int first number is wrong

- Except $ and _ , other symbols are not allowed while declaring identifier. E.g.

  String #hello is wrong

- Reserved keywords are not allowed for declaring identifiers. E.g. **public, new, static, int**
- Note : Java is case sensitive so, **"Name", "NAME" and "name"** are three different terms.

CLASSES AND INTERFACES

- It should start with the uppercase letter.
- It should be a noun such as Color, Button, ButtonWhite, MyJava, MyJavaProgram, System, Thread, etc.
- Use appropriate words, instead of acronyms.

```
public class MyFirstJavaProgram {

      public static void main(String[] args){

            System.out.println("Hello world!");

      }

}
```

METHODS

- It should start with lowercase letter.
- It should be a verb such as print(), println(), calculate(), etc
- If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed()

```
public class MyFirstJavaProgram {

	public static void main(String[] args){

		System.out.println("Hello world!");

	}

}
```

VARIABLE

- It should start with a lowercase letter such as id, name.
- If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as **firstName, lastName.**

```
public class MyFirstJavaProgram {
	public static void main(String[] args){
		int firstNumber = 10;
		String permanentAddress = “Pokhara”;
	}
}
```

NAMING CONVENTION

Package

- It should be a lowercase letter such as java, lang.
- If the name contains multiple words, it should be separated by dots (.) such as **java.util**, **java.lang**.

Constant

- It should be in uppercase letters such as **RED**, **YELLOW**.
- If the name contains multiple words, it should be separated by an underscore(_) such as **MAX_PRIORITY.**

DATA TYPES

- **Data type** defines the values that a variable can take, for example if a variable has int data type, it can only take integer values. There are two data types in Java
  1. Primitive
  2. Objects (Reference)

PRIMITIVE DATA TYPE

- Primitive data types are used to store
  - Numbers (int, float, etc.)
  - Individual characters, Booleans (true/false)
- Primitive data type names are all lowercase.

PRIMITIVE DATA TYPES

There are eight primitive data types in Java:

Data Type	Size	Description
byte	8 bits	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

REFERENCED DATA TYPES

- A reference type is a data type that's based on a class rather than on one of the primitive types
- Reference datatypes in java are those which contains reference/address of dynamically created objects. These are not predefined like primitive data types.
- Eg.

```
Animal animal = new Animal("giraffe");
```

- Object data types have initial uppercase character

DATA TYPES

- All variables in Java (whether primitives or objects), must be explicitly data typed when they are declared.
- This makes Java statically(strongly) typed language.

```
int myVar = 5;
```

where ***int*** *is data type*

myVar *is variable*

5 *is value*

- In the above example, myVar variable cannot store other data types later in the program.

OPERATORS

- An operator is a sign or symbol which performs some operations between one or multiple operands. For example,

```
int a = 5 + 3;
a++;
String message = “Hello” + “World”;
```

ARITHMETIC OPERATOR

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. If A= 10 and B=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)	Increases the value of operand by 1.	B ++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B -- gives 19

RELATIONAL OPERATOR

If A= 10 and B=20, then

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 false
!= (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 false
< (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 false
<= (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.

LOGICAL OPERATOR

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

Operator	Description	Example
&& (logical and)	Called Logical AND operator. Returns true if both statements are true	(A **&&** B) is false
|| (logical or)	Called Logical OR Operator. Returns true if one of the statements is true	(A || B) is true
! (logical not)	Reverse the result, returns false if the result is true.	**!**(A && B) is true !A= false

ASSIGNMENT OPERATOR

=	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

BITWISE OPERATOR

Operator	Description	Example
& (bitwise and)	if both bits are 1, it gives 1, else it gives 0	0 & 0 = 0 0 & 1 = 0 1 & 0 = 0 1 & 1 = 1
| (bitwise or)	If either of the bits is 1, it gives 1, else it gives 0.	0 | 0 = 0 0 | 1 = 1 1 | 0 = 1 1 | 1 = 1
^ (bitwise XOR)	This means that if bits of both the bits are 1 or 0 the result will be 0; otherwise, the result will be 1:	0 ^ 0 = 0 0 ^ 1 = 1 1 ^ 0 = 1 1 ^ 1 = 0
~ (bitwise compliment)	It changes binary digits 1 to 0 and 0 to 1.	~1 = 0 ~0 = 1

BITWISE OPERATOR

```
a    =      0011   1100
b    =      0000   1101
-----------------------------------
a&b  =      0000   1100
a|b  =      0011   1101
a^b  =      0011   0001
~a   =      1100   0011
```

TERNARY/CONDITIONAL OPERATOR

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

E.g

```
int a = 10;
Int b;
b = (a == 1) ? 20: 30;
System.out.println( "Value of b is : " + b );
```

What will be the output?

END OF LECTURE 2

Any Questions?