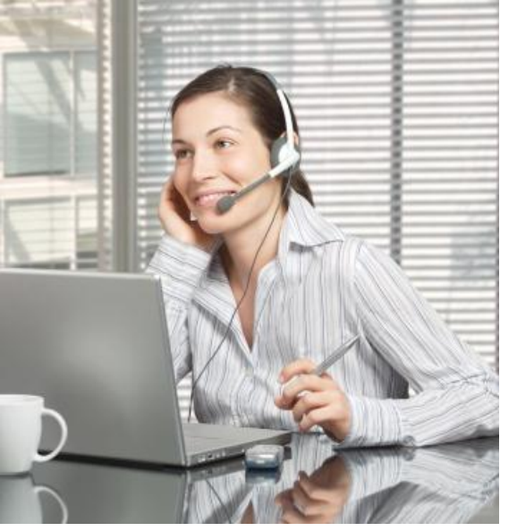
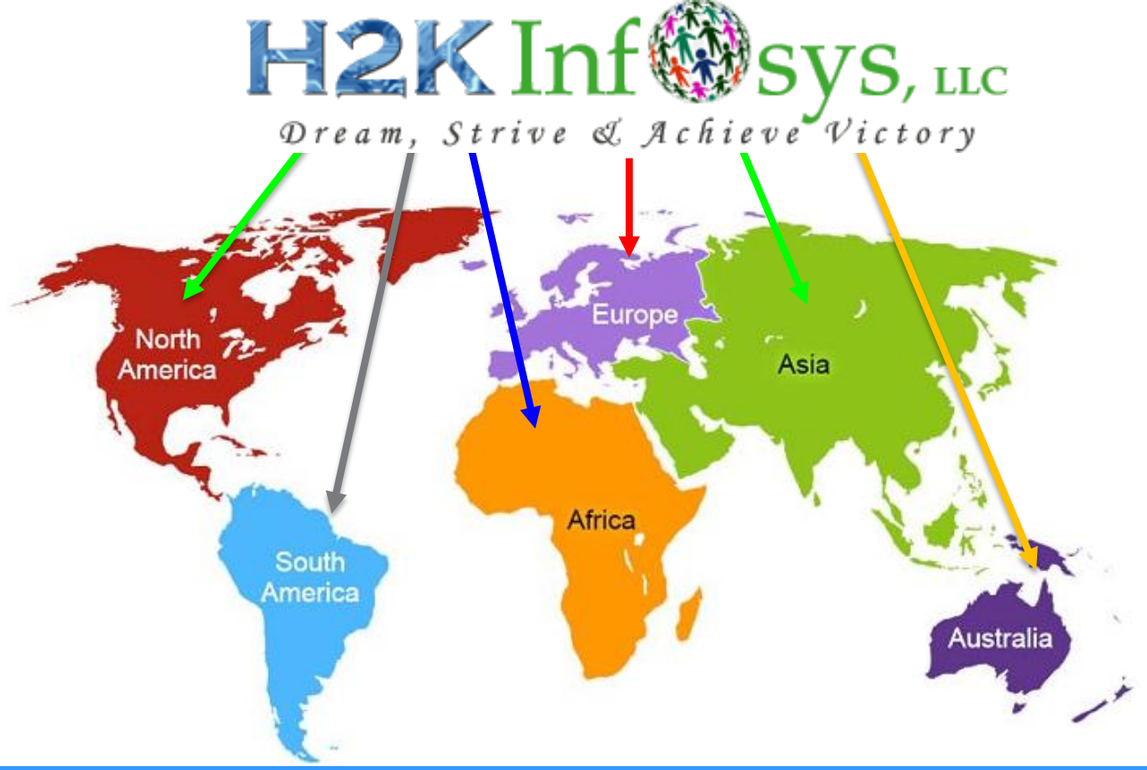


H2K Infosys, LLC

Dream, Strive & Achieve Victory



H2K

Live - Green - Everywhere

H2K Infosys is E-Verify business based in Atlanta, Georgia – United States
Providing Online IT training services world wide.

www.H2KINFOSYS.com

USA - +1-(770)-777-1269, UK - (020) 3371 7615
Training@H2KInfosys.com / H2KInfosys@Gmail.com

H2K INFOSYS PROVIDES WORLD CLASS SERVICES IN

IT Trainings with Real time Project Work for Corporates & Individuals

Special IT Training for MS Students in US

Software Design, Development, QA – Manual, Automation & Performance Testing, Maintenance.

IT Staff Augmentation

Job Placement Assistance

Tech Support

What are we learning today?

- **Java Data types**
- **Language Basics**
- **Conditional Statements**
- **Looping**
- **How to break the Loop**
- **How to continue the Loop**
-

Java Variables

- **Instance Variables**

Instance Variables declared directly under class declaration.
Name instance variable because there value is unique to each instance.

Let's write some instance variables in CustomerTO.java class.

- **Class Variables**

Class variables are declared with 'static' modifiers. So, also called as 'Static Variables'
No matter how many instances of class gets created, exact one copy of Class Variable exists.
If the variable value is changed, all instances will 'see' changed value.

Let's write some static variable.

- **Local Variables**

Methods declared within method to store temporary values, are local variables.

Naming Java Variables

Your Variable and method names reflect your Java Experience.

- Variable names are case-sensitive. Variables are recommended to begin with letters even is $ or _ sign is allowed.
- Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- Its recommended to use camelCase for variables in Java.

- **Java Data types**

The Java programming language is statically-typed, which means that all variables must first be declared before they can be used:

```
int myFirstInt = 1;
```

There are eight primitive data types supported by the Java programming language. We will look into it shortly.

Why Data Type?

A variable's data type determines the values it may contain, plus the operations that may be performed on it.

Format of declaring variable is:

<DataType> <variableName> = <AssignmentValue>;

```
int myFirstVaraiable = 1;
```

Java has Eight Different and very useful primitive data types

Java Data Type

.

Primitive	Definition	Default
byte	The byte data type is an 8-bit signed two's complement integer	0
short	The short data type is a 16-bit signed two's complement integer	0
int	The int data type is a 32-bit signed two's complement integer	0
long	The long data type is a 64-bit signed two's complement integer	0
float	The float data type is a single-precision 32-bit IEEE 754 floating point	0.0f
double	The double data type is a double-precision 64-bit IEEE 754 floating point.	0.0d
boolean	The boolean data type has only two possible values: true and false.	False
char	The char data type is a single 16-bit Unicode character.	'\u0000'
String	Undefined Size. Depends on assignment	Null

Operators

Type	Operators	Example
Unary	++*expr* --*expr* +*expr* -*expr* ~ !	++x, !booleanVar
multiplicative	* / %	firstVar*secondVar
additive	+ -	A + B
Relational	< > <= >= instanceof	A < B, A<=B
Equality	== !=	A==B
logical AND	&&	boolStatOne && boolStatTwo
Logical OR	||	boolStatOne || bollStatTwo
ternary	? :	A>B? return A: return B
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=	A = 0

- **Conditions**

We can check conditions in two ways:
If the boolean expression evaluates to true then the block of code inside the if statement will be executed.

```
if(Boolean_expression){
        //Executes when the Boolean expression is true
} else {
      //Executes when the Boolean expression is false
}
```

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
            }
```

Switch statement

The following rules apply to a switch statement:

- 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.

- **The while loop**

The Java programming language is statically-typed, which means that all variables must first be declared before they can be used:

```
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.

The do...while Loop

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

```
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.

The for loop:

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- A for loop is useful when you know how many times a task is to be repeated.

Looping


```
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.

Enhanced for loop: As of java 5 the enhanced for loop was introduced. This is mainly used for Arrays and Collections.

Looping


```
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.

.

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.

```
int [] numbers = {1, 2, 3, 4, 5};
for(int x : numbers ) {
   if( x == 3) {
         break;
    }
    System.out.println( x );
}
```

- Output will be:

1
2

And loop breaks on x = 3.

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.

```
int [] numbers = {1, 2, 3, 4, 5};
for(int x : numbers ) {
   if( x == 3) {
         continue;
    }
    System.out.println( x );
}
```

What will be output of this?

Questions:

Remember: No Question is Dumb Question.

Thank you

Your feedback is highly important to improve our course material and teaching methodologies. Please email your suggestions.

USA +1-(770)-777-1269
Training@H2KInfosys.com

UK (020) 3371 7615
H2KInfosys@Gmail.com

H2K Infosys is e-Verified business based in Atlanta, Georgia – United States

DISCLAIMER

H2K Infosys, LLC (hereinafter "H2K") acknowledges the proprietary rights of the trademarks and product names of other companies mentioned in any of the training material including but not limited to the handouts, written material, videos, power point presentations, etc. All such training materials are provided to H2K students for learning purposes only. H2K students shall not use such materials for their private gain nor can they sell any such materials to a third party. Some of the examples provided in any such training materials may not be owned by H2K and as such H2K does not claim any proprietary rights for the same. H2K does not guarantee nor is it responsible for such products and projects. H2K acknowledges that any such information or product that has been lawfully received from any third party source is free from restriction and without any breach or violation of law whatsoever.