



PERSISTENT

Persistent University

# *Basic Java : Language Fundamentals*





PERSISTENT

# Objectives :

- At the end of this module, you will be able to understand :
  - Identifiers and Keywords
  - Primitive data types
  - Object reference types
  - Literals and variables
    - Binary literals and underscore in numeric literals
    - Scoping and parameter passing (by value & by reference)
  - Comments and Javadoc
  - Flow Control Statements: Enhanced for loop and switch with String
  - Arrays
  - Variable arguments





- Are used to name Java language entities class, variable, methods and labels .
- Begin with a letter, underscore character (\_), or dollar sign (\$).
- Subsequent characters consist of these characters and the digits 0–9.
- Identifiers are case sensitive.

name

salary\_1002

!theValue

salaryEmp1002

1002Salary

AGE

Age

age



PERSISTENT

# Naming Conventions

- Try to keep your entity names in the program simple and descriptive.
- Class
  - Class names should be **nouns**, in mixed case with the first letter of each internal word capitalized.
- Interfaces
  - Interface names should be capitalized like class names.

```
class Customer{} class OrderLine{}
```

```
interface Shape{}
```



## Some points to remember about identifiers

- Methods
  - Methods should be verbs, in mixed case with the first letter lowercase. Internal words start with capital letters.
- Variables
  - Variable Names should be in mixed case with the first letter lowercase. Internal words start with capital letters.
- Constants
  - constants should be all uppercase with words separated by underscores ("\_").

```
run(); showTransaction(); getAllCustomers();
```

```
int age; String firstName;
```

```
int MIN_WIDTH=4; MAX_WIDTH=999;
```



PERSISTENT

# Keywords

- Reserved identifiers, can't be used to denote other entities.

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



PERSISTENT

# Declaring Variables

```
int gear=1;
```

```
float salary=1000.00f;
```

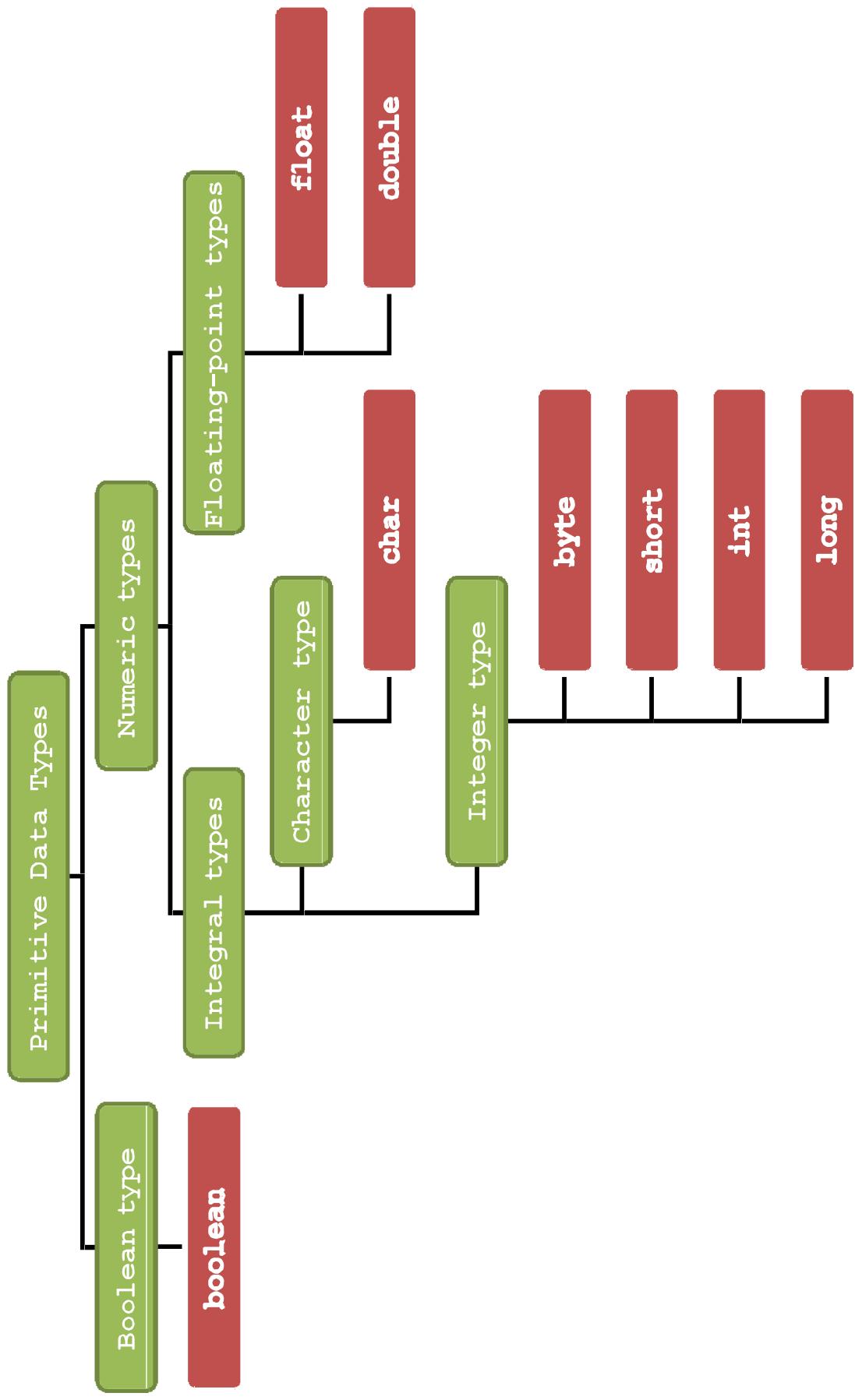
```
char grade='A';
```





PERSISTENT

# Primitive Data Types





Primitive type	Size	Minimum	Maximum	Wrapper type	Default Values
<b>boolean</b>	1 bit	—	—	Boolean	false
<b>char</b>	16-bit	Unicode 0	Unicode $2^{16}-1$	Character	'\u0000'
<b>byte</b>	8-bit	-128	+127	Byte	0
<b>short</b>	16-bit	$-2^{15}$	$+2^{15}-1$	Short	0
<b>int</b>	32-bit	$-2^{31}$	$+2^{31}-1$	Integer	0
<b>long</b>	64-bit	$-2^{63}$	$+2^{63}-1$	Long	0L
<b>float</b>	32-bit	IEEE754	IEEE754	Float	0.0f
<b>double</b>	64-bit	IEEE754	IEEE754	Double	0.0d

int gear;

float salary;

char grade;



PERSISTENT

## Primitive Data Types

- Stored on stack.
- Hold the values themselves.





PERSISTENT

## Reference Types

- These are non primitive data members.
- The types of reference variables are
  - Class type
  - Interface type
  - Arrays





PERSISTENT

## Reference DataTypes

- **Class type** : object of a class like String, Scanner or any custom object
- **Interface type** : reference of any interface that holds object of a class implementing that interface.
- **Arrays** : arrays are reference data members.
  - Reference data members stored on heap.
  - Refer the location where value is stored.





## Boolean and Character Literals

- Boolean Literals
  - represent truth values, true or false

```
boolean t = true;  
boolean f = 0;
```

*// Compiler error!*

- Char Literals

- are 16-bit unsigned integers, (Unicode character set).
- are quoted in single-quotes('). E.g. 'a'.

```
char a = 'a';  
char b = '@';  
char letterN = '\u004E';
```

*// The letter 'N'*



PERSISTENT

## Integer Literals

- Default value is always "int".
- To specify long, append with suffix L (or l).
- Decimal Literals(base10)
- Octal Literals(base 8)
- Hexadecimal Literals(base 16)

1, 010, 0x0001

1226836372234L





PERSISTENT

## Floating Point Literals

- default value is always "double".
- may be explicitly designated with the suffix D (or d).
- float must be specified with the suffix F (or f).
- Scientific notation (with Exponent) E.g. 7E-4.

```
double d = 110599.995011D; double g = 987.897;
```

```
float f = 23.467890;
```

```
float f = 23.467890f;
```

// Compiler  
error!





- String Literals
  - Sequence of characters
  - Must be quoted in quotation marks. E.g. "Java"
  - Are objects of the class *String*.
- Reserved Literals – null, true, false





## Integral Types as Binary Literals

- This feature is added in JDK 1.7
- The integral types as byte, short, int and long, can be represented using binary number system.
- To represent a binary literal, a prefix as 0b or 0B need to be added.

```
- // An 8-bit 'byte' value: byte aByte = (byte)0b00100001;  
- // A 16-bit 'short' value: short aShort = (short)0b1010000101000101;  
- // A 32-bit 'int' value: int anInt1 =  
 0b10100001010001010000101000101;  
- // A 64-bit 'long' value. long aLong =  
 0b1010000101000101101000010100011010000  
 00101000101L;
```





PERSISTENT

# Underscores Between Digits in Numeric Literals

- This feature is added in JDK 1.7
- Underscores are permitted in numeric literals.
- Underscores can be placed where required to increase readability; like between hundreds and thousands, thousands and lakhs etc.
- This is used to group numbers in a bigger literal value (especially of long data type).

```
long carCost = 4585663L;
```

```
long carCost = 45_85_663L;
```





PERSISTENT

# Underscores Between Digits in Numeric Literals

- Following statements will cause compile time error

```
carCost = _23456; - Do not use underscore at the  
beginning of a number.  
carCost = 234567_ ; - Do not use underscore at the end of a  
number.  
  
carCost = 1234_.89; - Do not use underscore before  
decimal point.  
carCost = 2345._90; - Do not use underscore after  
decimal point.  
busCost = 2345678.90_f; - Do not use underscore before  
any suffix as f / F and l / L.
```





PERSISTENT

# Underscores Between Digits in Numeric Literals

- Some points to remember regarding usage of underscore.
  - Do not use the underscore at the beginning.
  - Do not use in String representation of a numeric value

```
String num = "10_00_298"; // causes  
NumberFormatException at runtime.
```





PERSISTENT

## Scoping and Parameter Passing

- Scope of a variable refers to the accessibility of a variable.
- Variables can be declared at several places:
  - In a class body as class fields. Variables declared here are referred to as class-level variables.
  - As parameters of a method or constructor.
  - In a method's body or a constructor's body.
  - Within a statement block, such as inside a while or for block.





PERSISTENT

## Scoping and Parameter Passing

- Variables declared at class level are termed as Global variables and those are accessible throughout the class.
- E.g.

```
class Person{  
    int age;  
    String name;  
}
```

Here age and name are class level variables. Also termed as global variables and available to all methods and other constructs in that class.





## Scoping and Parameter Passing

- Variables declared within any block like method or constructor or a block are termed as Local variables.
- These variables are accessible only within that construct.
- E. g.

```
for(int num = 0; num < 10; num++){
    System.out.println("Num: " + num);
}
```

- num variable is declared within this construct so available only within this block.





PERSISTENT

## Scoping and Parameter Passing

- Parameters are passed to any method or constructor.
- Java uses pass by value for both primitives and reference data members.





PERSISTENT

## Comments

- Single line comment

**//This comment ends at the end of this line  
int x;//from comment-start seq. to the end of line**

- Multiple-line Comment

```
/* comment on line1  
comment on line2  
... comment on lineN */
```





PERSISTENT

## Comments

- Documentation Comment

```
/**  
 * This class implements a gizmo  
 * @author K.A.M  
 * @version 2.0  
 */
```





PERSISTENT

## Javadoc Comments

- Javadoc recognizes special comments `/** ... */` which are highlighted blue by default in Eclipse (regular comments `//` and `/* ... */` are highlighted green).
- Javadoc allows you to attach descriptions to classes, constructors, fields, interfaces and methods in the generated html documentation by placing Javadoc comments directly before their declaration statements.





PERSISTENT

## Javadoc Comments

- Here's an example using Javadoc comments to describe a class, a field and a constructor:

```
/** Class Description of MyClass */
public class MyClass
{
    /** Field Description of myIntField */
    public int myIntField;

    /**
     * Constructor Description of MyClass()
     * public MyClass() { // Do something ... }
    */
}
```





PERSISTENT

## Javadoc In Eclipse

- Eclipse can generate Javadoc comments for classes and methods.
  - Place the cursor in the text of class or method declaration.
  - Right Click->Source->Add Javadoc Comment.
  - Javadoc comments with appropriate tags are generated, but you still have to
  - Write the descriptions.





PERSISTENT

## Flow control Statements : Enhanced For Loop

- Also termed as for-each loop.
- Mainly used to traverse over arrays and collections.
- Makes the code more readable
- Eliminates the possibility of programming errors.





PERSISTENT

## Flow control Statements : Enhanced For Loop

- Cycles over an array or collection in strict sequential manner.
- Automatically cycles from lowest index to highest index
- `for(type variable : array | collection){}`





PERSISTENT

## Flow control Statements : Enhanced For Loop

```
for(type variable : array | collection){//Statements to be  
executed}
```

- The type specifies the data type of the variable retrieved from the implementation.
- The variable will hold the value, one at a time.
- Array or collection is getting cycled.





```
package com.features;
public class EnhancedForLoopDemo1 {
    public static void main(String[] args) {
        //declare an int array
        int int_array[] = {12,45,78,34,90,34,56,23};
        //iterate over array using enhanced for loop
        for (int value : int_array) {
            System.out.println(value);
        }
    }
}
```

Output:

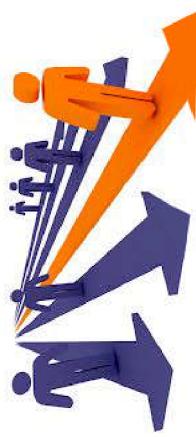
```
12
45
78
34
90
34
56
23
```



PERSISTENT

## Advantages of Enhanced For Loop

- No new keyword is required to be introduced in Java. The capability is enhanced for the for statement without adding any new keyword.
- Pre existing code is not broken.
- More useful for search operation over unsorted implementations.



Copyright reserved © 2013 Persistent University



PERSISTENT

# Flow control Statements : String in switch expression

- String objects can be used for comparison in switch statements.
- Switch does case-sensitive comparison with case statements.
- Switch gives a more efficient and cleaner code than if - else if - else code.
- equals() method is used for the comparison.





# Flow control Statements : String in switch expression

```
public class StringSwitchDemo {  
    public static void main(String[] args) {  
        String season = "Summer";  
        switch (season) {  
            case "Summer":  
                System.out.println("Its Summer");  
                break;  
            case "summer":  
                System.out.println("Its summer");  
                break;  
            default:  
                System.out.println("Wrong value");  
                break;  
        }  
    }  
}
```

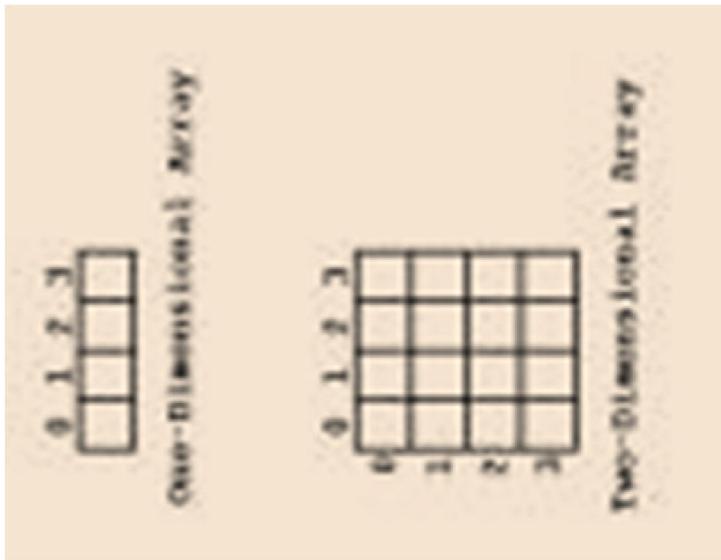




PERSISTENT

# Array

- **What is an array ?**
  - a group of like-typed variables referred to by a common name
- **Why use arrays ?**
  - group related information together
- **Types**
  - Single - dimension
  - Multi - dimensional





# Arrays in Java

- **Declaration**

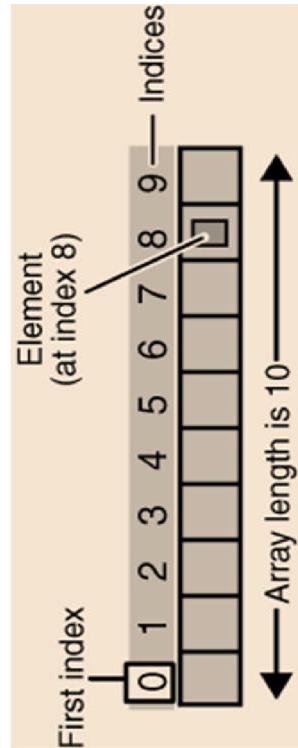
- `int marks[];`

- **Allocation**

- `marks = new int[10];`

- **Accessing**

- `marks[0] = 65;`
- `marks[1] = 56;`
- `marks[2] = 89;.....so on`



- **Initialization**

- `int temps[] = { 45, 33, 40, 37, 11 };`



# Features of arrays in Java

- **Alternative syntax**

- int [] marks, temps;

- **Bounds-checking**

- Compile-time

- Run-time

- **Treated as objects**

- The length attribute

- **The for....each form of the for loop**

- Use for each loop for iterating

```
int [] marks;  
int index = 0;  
  
marks = new int[10];  
  
for( ; index < marks.length; ) {  
    marks[index] = ++index;  
}  
  
/* oops ! don't do this */  
marks[14] = 88;  
  
for(int mark : marks) {  
    System.out.println(mark);  
}
```

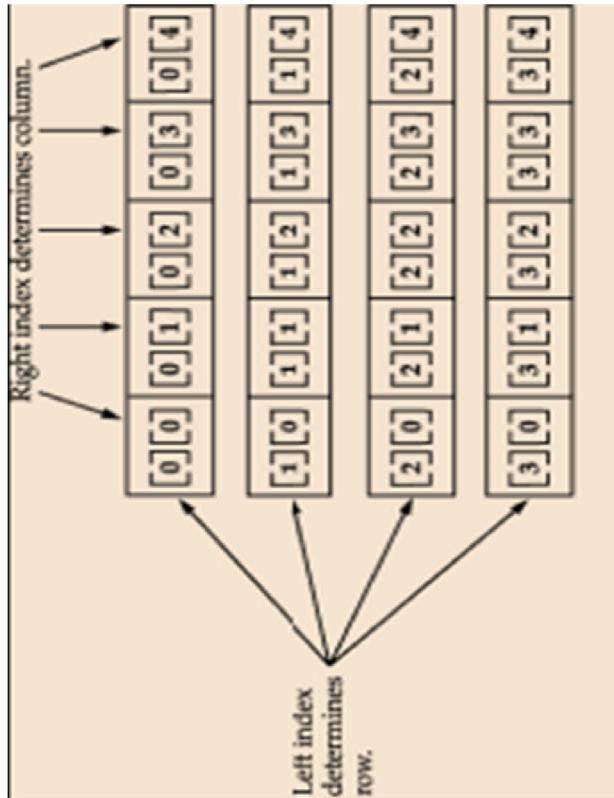




## Even

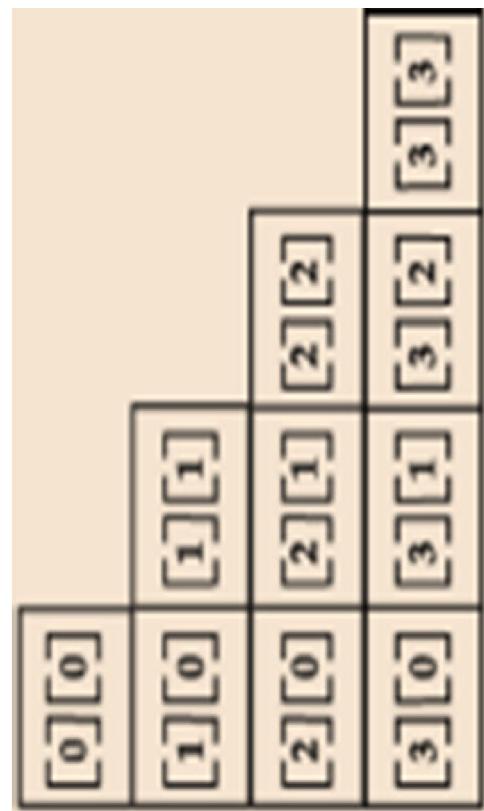
```
int twoD[] [] = new int[4][5];
```

```
int twoD[] [] = new int[4] [] ;  
twoD[0] = new int[1] ;  
twoD[1] = new int[2] ;  
twoD[2] = new int[3] ;  
twoD[3] = new int[4] ;
```



## Uneven

```
int twoD[] [] = new int[4] [] ;  
twoD[0] = new int[1] ;  
twoD[1] = new int[2] ;  
twoD[2] = new int[3] ;  
twoD[3] = new int[4] ;
```





# Multi-dimensional arrays

```
/* Even two-d arrays */
int twoD[][] = new int [4][5];
int x, y;
for(x = 0 ; x < 4 ; x++) {
    for(y = 0 ; y < 5 ; y++) {
        twoD[x][y] = x + y;
    }
}
```

```
/* Un-even two-d arrays */
int twoD[][] = new int [4][];
twoD[0] = new int[1];   twoD[1] = new int[2];
twoD[2] = new int[3];   twoD[3] = new int[4];
for(int x = 0 ; x < twoD.length ; x++) {
    for(int y = 0 ; y < twoD[x].length ; y++) {
        twoD[x][y] = x + y;
    }
}
```



PERSISTENT

## Varargs

- Variable argument allows the programmer to pass 0 or multiple arguments of same data type to a method.
- Before varargs it was required to either overload the method to accept different number of arguments or pass an array as an argument.
- This approach is effective if number of arguments to be passed are not known.





PERSISTENT

## Varargs syntax

- Varargs uses ellipsis i.e. three dots for the declaration

```
data_type... variableName
```





## Rules for using varargs

- There can be only one variable argument in the parameters passed to the method.

```
public void print(String... names, int... nums){} //causes  
                                         compilation error  
  
public void print(String name, int... nums){} //correct  
                                         syntax
```

- Variable argument must be the last parameter in argument list for methods.

```
public void print(String... names, int num){} //causes  
                                         compilation error  
  
public void print(int num, String... names){} //correct syntax
```





**1. Which three are valid declarations of a char? (Choose three.)**

- A. `char c1 = 064770;`
- B. `char c2 = 'face';`
- C. `char c3 = 0xbeef;`
- D. `char c4 = \u0022;`
- E. `char c5 = '\iface';`
- F. `char c6 = '\uface';`

**2. Which two are valid declarations of a String? (Choose two.)**

- A. `String s1 = null;`
- B. `String s2 = 'null';`
- C. `String s3 = (String) 'abc';`
- D. `String s4 = (String) '\ufeed';`
- E. `String s5 = "strings rule";`



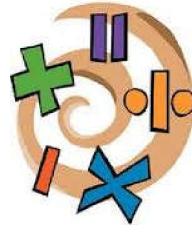


**3. Which one is a valid declaration of a boolean? (Choose one.)**

- A. boolean b1 = 0;
- B. boolean b2 = 'false';
- C. boolean b3 = false;
- D. boolean b4 = Boolean.false();
- E. boolean b5 = no;

**4. Which three are valid declarations of a float? (Choose three.)**

- A. float f1 = -343;
- B. float f2 = 3.14;
- C. float f3 = 0x12345;
- D. float f4 = 42e7;
- E. float f5 = 2001.0D;
- F. float f6 = 2.81F;





PERSISTENT

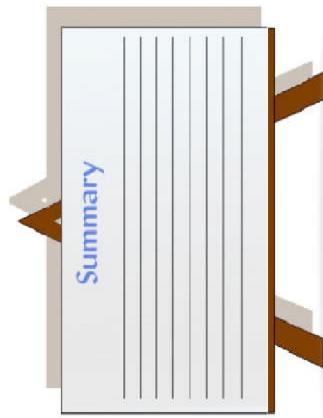
## Summary : Session #

With this we have come to an end of our session, where we discussed about

- Language Fundamentals as identifiers, keywords, data types etc

At the end of this session, we expect you to

- Develop java applications using these concepts





PERSISTENT

## Reference Material : Websites & Blogs

- <http://www.javaprepare.com/java-tutorial/java-language-fundamentals/>
- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/>
- <http://www.java-examples.com/java-language-fundamentals>



- ***Head First Java***

- By: *Kathy Sierra, Bert Bates*
- Publisher: *O'Reilly Media, Inc.*

- ***Java Complete Reference***

- By *Herbert Schildt*



## Key Contacts :

### *Java Interactive :*

- Asif Immanad

[asif\\_immanad@persistent.co.in](mailto:asif_immanad@persistent.co.in)

- Nisha Waikar

[nisha\\_waikar@persistent.co.in](mailto:nisha_waikar@persistent.co.in)

- Varsha Mulik

[varsha\\_mulik@persistent.co.in](mailto:varsha_mulik@persistent.co.in)

### *Persistent University :*

- Poorva Kulkarni

[poorva\\_kulkarni@persistent.co.in](mailto:poorva_kulkarni@persistent.co.in)

### *Vice President:*

- Shubhangi Kelkar

[shubhangi\\_kelkar@persistent.co.in](mailto:shubhangi_kelkar@persistent.co.in)



PERSISTENT

Persistent University

Thank You !!!

