

Core Java (Java 8)

By Saurabh Sharma

Course Outline

Session/Day	Module Name
1	<ul style="list-style-type: none">▪ Introduction to Java▪ Introduction of IDE (Net Beans or Eclipse). How to configure and create java application with IDE.
2	<ul style="list-style-type: none">▪ Inheritance and Polymorphism▪ Classes in Java
3	<ul style="list-style-type: none">▪ Generics and Collections▪ Enum, Var-Args, Scanner class.
4	<ul style="list-style-type: none">▪ Exceptions and Assertions▪ Input and Output Streams▪ Input and Output Streams(NIO.2)
5	<ul style="list-style-type: none">▪ Threads And Concurrency▪ Inner and Anonymous classes
6	<ul style="list-style-type: none">▪ Building Database Applications with JDBC▪ Localization

Course Objectives

At the end of this course, you should be able to:

- ❑ How to start with Java
- ❑ Define inheritance and polymorphism
- ❑ Identify the classes in Java
- ❑ Define collections and Generics
- ❑ Describe Exceptions and Assertion
- ❑ Identify Input and output Streams
- ❑ Explain and define Threads and JDBC

Prerequisites:

- ❑ Must be computer literate
- ❑ Learners are supposed to have understanding of basic concept of a variables, objects & classes
- ❑ Must have worked on any generation of programming language

Core Java

Session - 1

Java: An introduction

Session 1: Objectives

After completion of this module, you should be able to:

- ❑ Understand history of Java
- ❑ Understand the concept of Java
- ❑ Identify the types of Java programs and first Java programs
- ❑ Explain variables and data types
- ❑ Identify various expressions, statements and blocks of Java
- ❑ Identify operators
- ❑ Understand arrays, control flow and Classpath

Session 1: An Introduction

- ❑ What is java?
- ❑ Why Java?

Session 1: Brief History Of Java

- ❑ Founder of Java Language → By team named “Green” with members lead by James Arthur Gosling
- ❑ Originally called Oak (1991) → The name Oak was used by Gosling after an oak tree that stood outside his office
- ❑ First version of Java was released: 1995
- ❑ FOSS (GPL):
 - FOSS is a free and open source software and GNU General Public License
 - The GNU General Public License is a free, copy-left license for software and other kinds of works
- ❑ Netscape Navigator Internet browser was the first Java enabled browser

Session 1: Features Of Java

- ❑ **Simple:** The language syntax is based on the familiar programming language 'C++'. Java does not support pointers which are a notorious source of bugs. Memory is automatically allocated and deallocation is done by the garbage collector and must not be explicitly programmed
- ❑ **Object oriented:** Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships
- ❑ **Architectural Neutral:** As Java programs are compiled to byte-code machine, compiled programs will run on every architecture which implements the Java virtual machine. whether it is an Window PC, a Macintosh, Linux or a Solaris system. So the Java programs can be used on any machine irrespective of its architecture and hence the Java program is Architectural Neutral. Slogan of Java – Write Once , Run Anywhere
- ❑ **Secure:** Java is mostly used in network environment, a lot of emphasis has been placed on security to enable construction of virus free and tamper free systems

Session 1: Features of Java(Continued)

- **Portable type:** The size of data types are always same irrespective of the system architecture. That is an int in Java is always 32 bit unlike in C/C++ where the size may be 16-bit or 32-bit depending on the compiler and machine. Hence when ported on different machines, there does not occur any change in the values the data types can hold. And also Java has libraries that enables to port its application on any systems like UNIX, Windows and the Macintosh systems
- **Distributed:** Java's networking capabilities are both strong and easy to use. Java applications are capable of accessing objects across the net, via URLs as easy as a local file system
- **Robust:** A main source of errors in the C programming language is pointer manipulation. The explicit use of pointers has been removed from Java and in this way Java programs are easier to debug. On the other hand Java is a strongly typed language and therefore extensive compile-time checking for potential type errors is done. Java requires explicit method declarations and there are no implicit declarations (as in C, for example). The exception handling model of Java allows to group all the error handling code in one place via the try/catch/finally construct

Session 1: Features of Java(Continued)

- ❑ **Multithreaded:** Multithreading is the ability for one program to do more than one task at once. It is easy to implement in java compare to other language
- ❑ **Dynamic:** Java manipulates memory in a dynamic way. Classes are loaded by demand even across a network. The distributed nature of Java really shines when combined with its dynamic class loading capabilities. Together, these features make it possible for a Java interpreter to download and run code from across a network
- ❑ **Interpreted:** The Java Interpreter can execute Java byte code, directly on any machine to which the interpreter has been ported. Interpreted code is slower than compiled code
- ❑ **High performance:** The byte codes can be translated at run time into machine code for the particular CPU on which the application is running
- ❑ **Applet programming:** This is one of the important features which has attracted the users of the Internet. Applets are Java programs that are typically loaded from the Internet by a browser

Session 1: Programming Approaches

❑ Structured Approach

- ❑ Based on functions
- ❑ goto branching
- ❑ C, C++, COBOL, Pascal
- ❑ Some disadvantages: No constructs for encapsulation, chances of code repetition, No strong data hiding concept, difficult to debug

❑ Object-oriented Approach

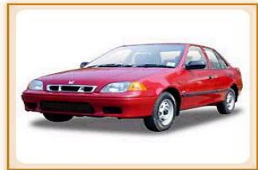
- ❑ Smalltalk, Java, C# etc.

Session 1: Object-Oriented Programming Concepts

- ❑ Object
- ❑ Class
- ❑ Abstraction
- ❑ Encapsulation
- ❑ Inheritance

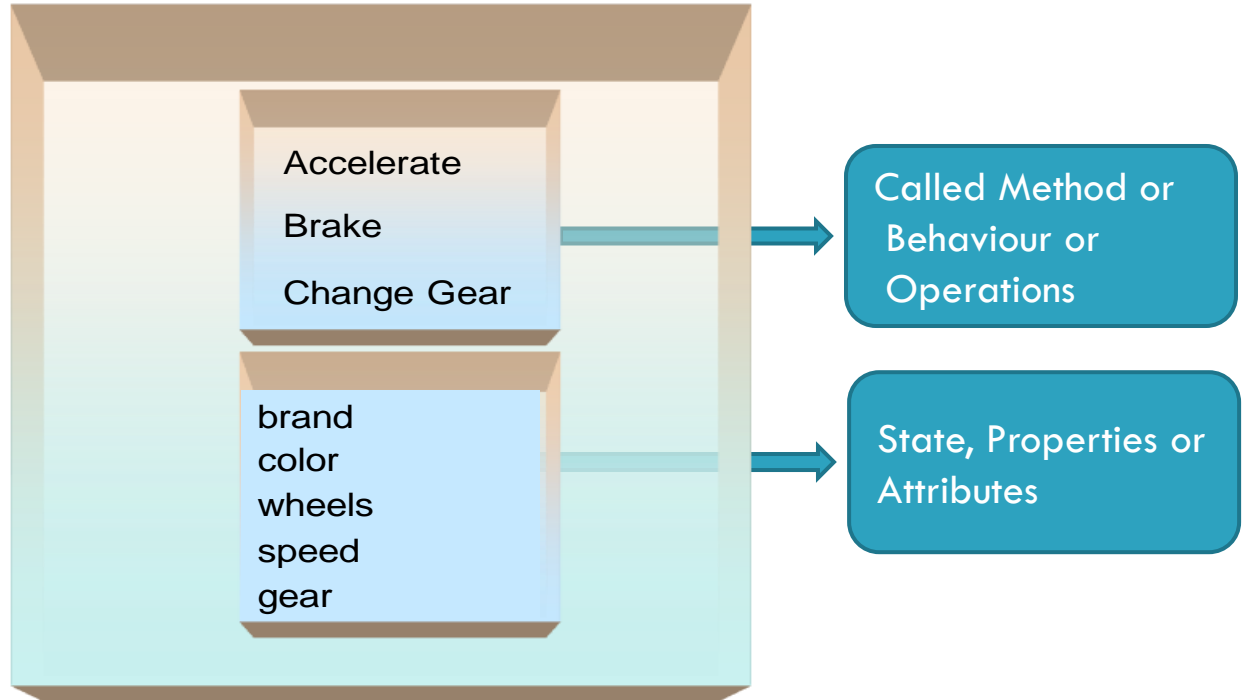
Session1: Object

- Any thing in a real world that can be either physical or conceptual. An object in object oriented programming can be physical or conceptual
- Conceptual objects are entities that are not tangible in the way real-world physical objects are
- Car is a physical object. While college is a conceptual object
- Conceptual objects may not have a real world equivalent. For instance, a Stack object in a program
- Object has state and behavior



What is the state and behavior of this Car?

Session1: Example Object



Session 1:Attributes And Operations

- ❑ The object's state is determined by the value of its properties or attributes
- ❑ Properties or attributes → member variables or data members
- ❑ The object's behavior is determined by the operations that it provides
- ❑ Operations → member functions or methods

Session 1: Putting It Together

A Car:

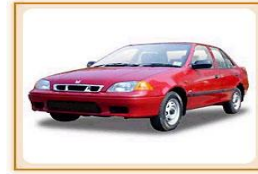
- ❑ It's a real-world thing
- ❑ Can be accelerate to increase speed
- ❑ It has real features like the color, wheels, brand etc
- ❑ It also has conceptual features like power
- ❑ A Car manufacturing factory produces many Cars based on a basic description / pattern of what a Car is

Object

Method

Member variables

class



Session 1: Class

- ❑ A class is a collection of attributes and behavior. It is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind
 - ✓ **blueprint:** A class can't do anything on its own. The life of class begins by instantiating it
 - ✓ **defines:** It provides something that can be used later
 - ✓ **Objects:** This is the instance of class
- ❑ A class is an abstraction

Session 1: Abstraction

- ❑ *Abstraction denotes essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. -Grady Booch*
- ❑ Abstraction makes only the relevant details of an object visible
- ❑ Example
 - For a Doctor → you are a Patient
 - Name, Age, Old medical records
 - For a Teacher → you are a Student
 - Name, Roll Number/RegNo, Education background

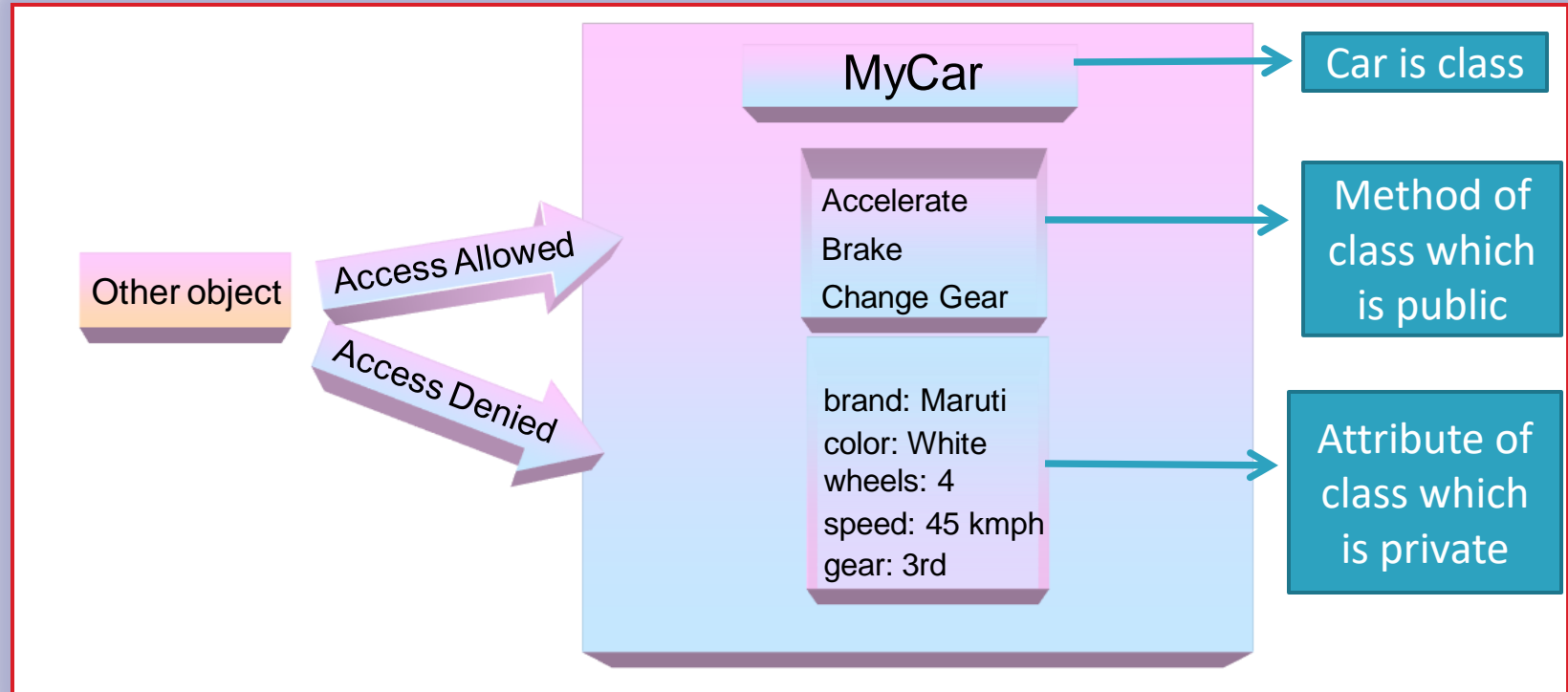
Session 1: Encapsulation

- ❑ *Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics*

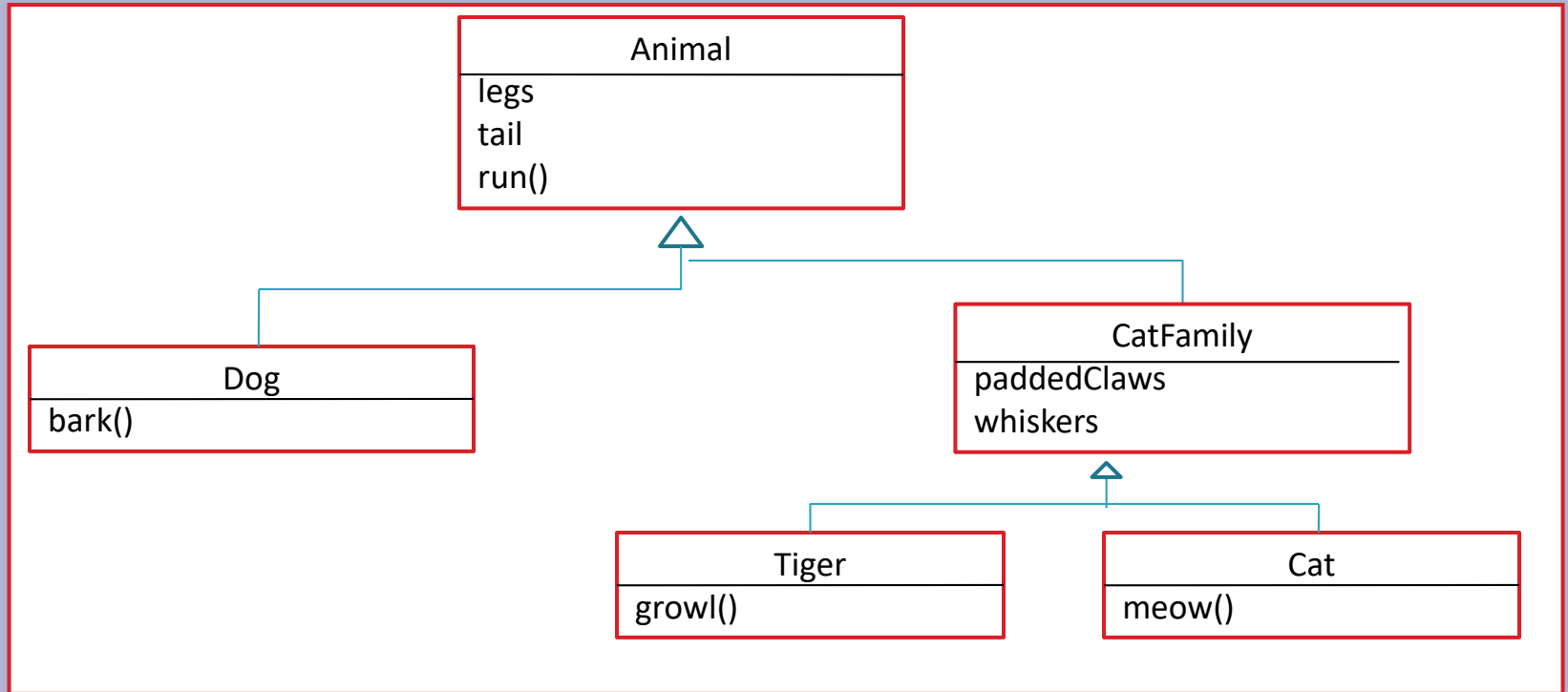
- Grady Booch

- ❑ Encapsulation is binding data and operations that work on data together in a construct
- ❑ Encapsulation involves Data and Implementation Hiding
- ❑ By using the concept of encapsulation you can keep data safe as making them private and you can access that by its public methods. Nobody can access these private data outside the class so you can say encapsulation is like data hiding

Session 1: Example Encapsulation



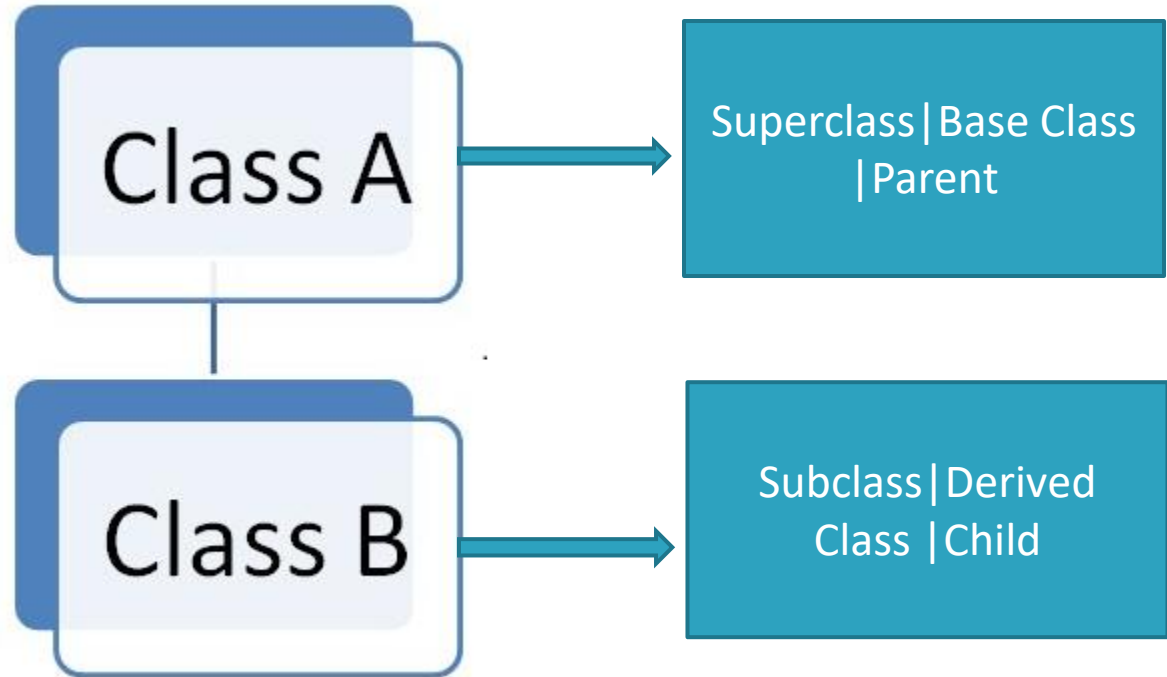
Session1: Inheritance(Continued)



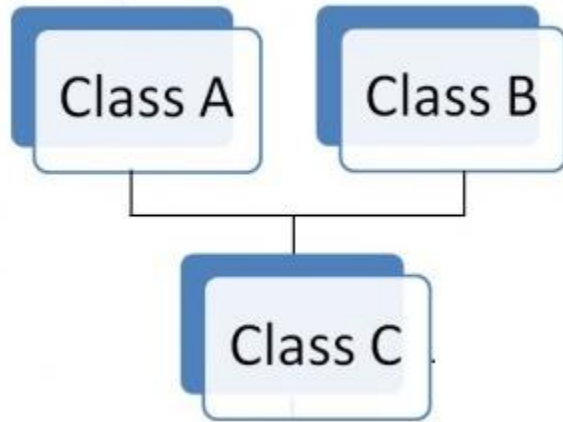
Session1: Types Of Inheritance

- Single inheritance
- Multiple inheritance

Session 1: Single Inheritance



Session 1: Multiple Inheritance



Java does not support multiple inheritance

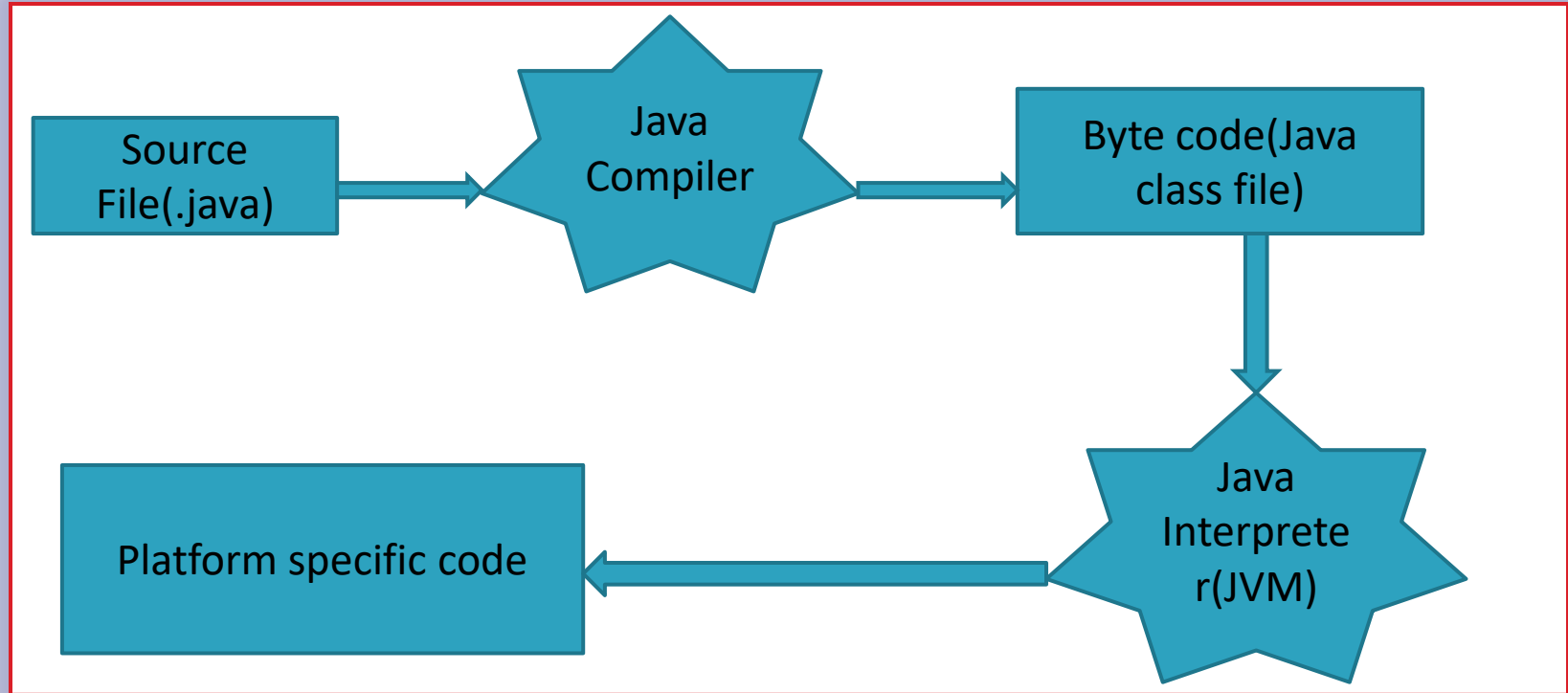
Session 1: Polymorphism

- ❑ Derived from two Latin words - Poly, which means many, and morph, which means forms
- ❑ It is the capability of an action or method to do different things based on the object that it is acting upon
- ❑ In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class

Session 1: Components Of Java Architecture

- ❑ Java Programming Environment
- ❑ Java Virtual Machine (JVM)
- ❑ Java API (Application Programming Interface)

Session 1: Components Of Java Architecture(Continue)



Session 1: JVM(Java Virtual Machine)

- ❑ A Java virtual machine (JVM) is an implementation of the Java Virtual Machine Specification, interprets compiled java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java program's instructions. Java was designed to allow application programs to be built that could be run on any platform without having to be rewritten or recompiled by the programmer for each separate platform. A Java virtual machine makes this possible because it is aware of the specific instruction lengths and other particularities of the platform
- ❑ The Java virtual machine is called "virtual" because it is an abstract computer defined by a specification. To run a Java program
- ❑ The Java Virtual Machine Specification defines an abstract rather than a real machine or processor. The Specification specifies an instruction, a set of registers, a stack, a "garbage heap," and a method area. Once a Java virtual machine has been implemented for a given platform, any Java program (which, after compilation, is called bytecode) can run on that platform. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to a real processor instruction) or the bytecode can be compiled further for the real processor using what is called a just-in-time compiler
- ❑ JVM is platform dependent(For every machine, we have different JVM)

Session 1: Java API

- ❑ Java API is a set of classes and interfaces that comes with the Java development Kit. Java API has huge collection of library routines that performs basic programming tasks such as looping, displaying GUI form etc
- ❑ In the Java API classes and interfaces are packaged in packages. All these classes are written in Java programming language and runs on the JVM. Java classes are platform independent but JVM is not platform independent. You have find different downloads for each OS
- ❑ Example of Java API- `java.lang`, `java.io`, `java.util`, `java.awt`, `java.net` etc

Session 1: Structure Of Java Application

❑ Class Components

- A class Consists of data members and methods

Class Name

Variable
declaration

Method1
Method2
.
.
Method n

Session 1: Java Class : Syntax

```
1. [modifier] class classname
2. {
3. // Variables definitions
4.     modifier datatype data_variable1;
5.     modifier datatype data_variable2;
6.     .....
7. // Method definitions
8. modifier returnType method1(datatype parameter11, datatype parameter12,...)
9. {
10.     //method definition
11.     stmt 1;
12. }
13. modifier returnType method2(datatype parameter21, datatype parameter22,....)
14. {
15.     //method definition
16.     stmt 1 ;
17.     .....
18. }
```


Session 1: Class Example

```
class Car
```

```
{
```

```
String brand;
```

```
String color;
```

```
int wheels;
```

```
int speed;
```

```
int gear;
```

```
void accelerate(){}  
void brake(){}  
void changeGear(){}  
}
```

Variable Declaration

Method Name

Return type

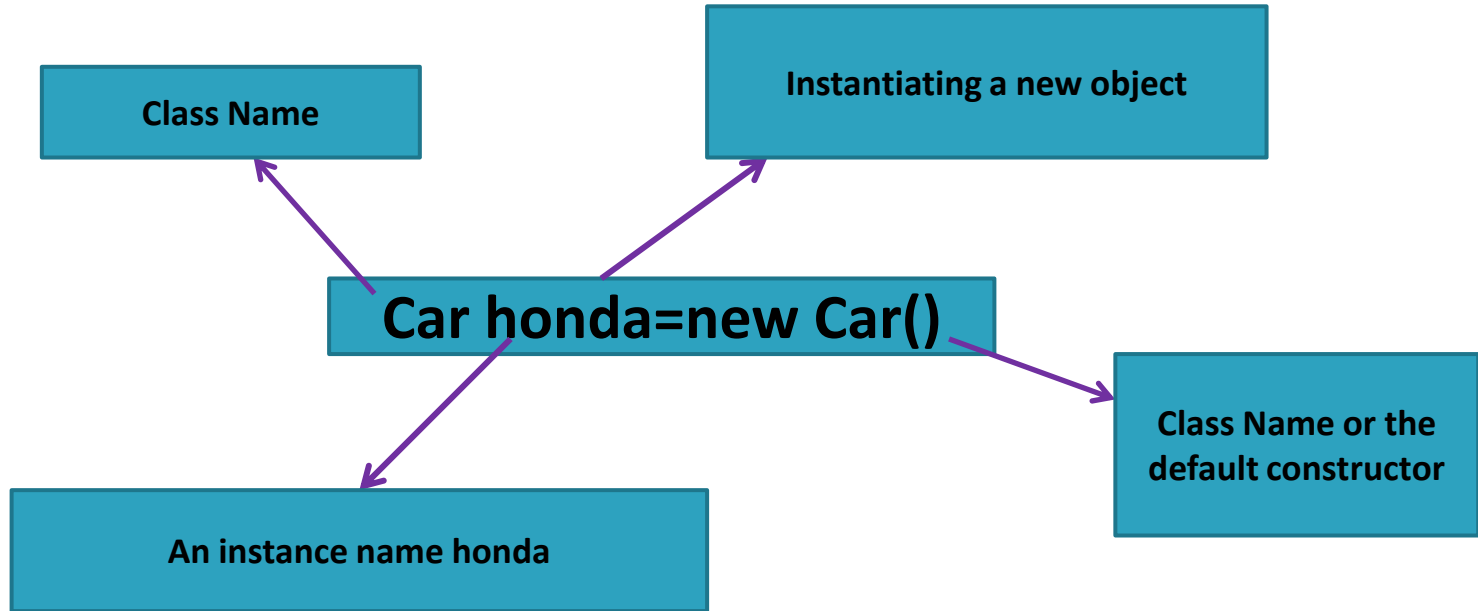
Session 1: Class Example

❑ Steps to create an object of class

- Declaration ➡ Car obj; ➡ No memory allocation
- Instantiation ➡ obj=new Car() ➡ Object created

Reference to Car object

Session 1: Object Of a Class(Continued)



Session 1: Accessing Data Members of a Class

- ❑ Dot Operator(.) is used to access the data members of a class outside the class by specifying the data member name or the method name

Example– `Car obj=new Car();`

`obj.brand="Honda";`

`obj.color="red";`

`obj.brake();`

Session 1: First Java Program

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

Special statement used to display data on console.
'println' causes the next print statement to be printed
in the next line.

main() is a method from where program
execution begins.

Save the file as HelloTest.java. A public class name and file name must
be same.

Session 1: Environment To Compile And Execute

- Compile java programs
 - ▣ From command prompt
 - ▣ Through an IDE (Integrated development environment) like
 - Eclipse
 - NetBeans
 - JBuilder
 - RAD (Rapid application development)

Session 1: Compiling HelloTest.java From Command Prompt

- ❑ Assume that file is saved in C drive by the name HelloTest.Java
- ❑ Steps to Compile HelloTest.Java application from DOS command prompt
 - Set the path variable to the bin directory of the installation folder of Java
 - Open the DOS command prompt and type the following command:

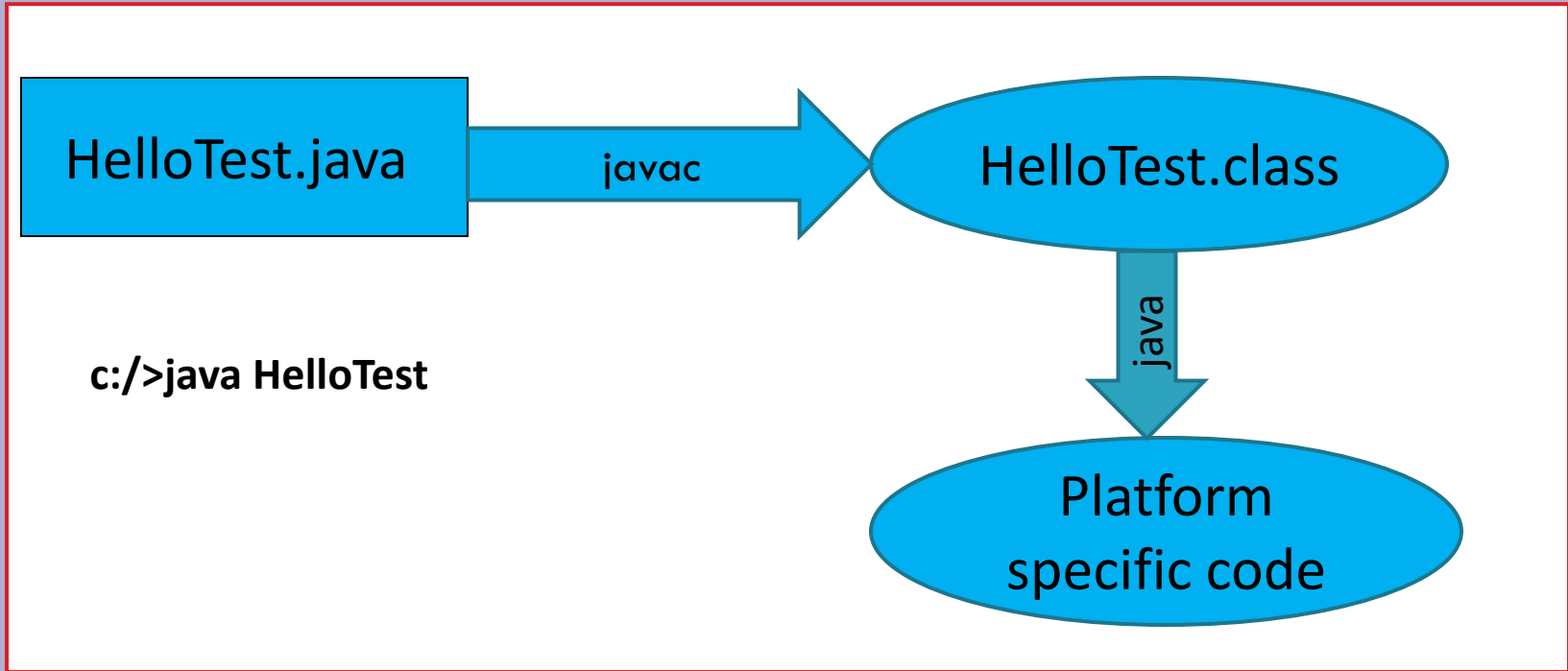
C:\>javac HelloTest.java

- ❑ Run the Application

- ❑ Use Java command followed by .class file name without using the extension to run the Java application as shown below:

Java HelloTest

Session 1: Compilation And Execution Model (From Command Prompt)



Session 1: Classpath

- ❑ Classpath is an environment variable that tells the Java Virtual Machine where to look for user-defined classes and packages in Java programs
- ❑ The Classpath is the connection between the Java runtime and the filesystem. It defines where the compiler and interpreter look for .class files to load. The basic idea is that the filesystem hierarchy mirrors the Java package hierarchy, and the Classpath specifies which directories in the filesystem serve as roots for the Java package hierarchy

Session 1: Setting Classpath Variable In Windows

Steps for setting Classpath on Windows XP

- ❑ The Classpath variable can be set on Windows with the following steps
- ❑ Click the Start button in the lower left of the screen
- ❑ Select My Computer or Computer from the pop-up menu
- ❑ Right click on Computer or My Computer select properties
- ❑ Click the Advanced tab
- ❑ Click the Environment Variables button near the bottom and you will see two lists of variables
- ❑ Inside System variables list for a variable named Classpath. If you find it, click Edit. If you don't find it, click New and enter Classpath in the Variable name field
- ❑ The Variable value field is a list of file paths of directories or jar files. The first thing in this list should be the current directory, which is represented in windows just as it is in Unix, as a single period. If there's more than one thing in this list, separate items with a semicolon

Session 1: Setting Classpath In Command Line

- ❑ To set the classpath temporarily in command line
- ❑ In the command prompt, enter the command
set CLASSPATH=%CLASSPATH%;JAVA_HOME\lib;
- ❑ Command to set CLASSPATH in Unix/Linux
export CLASSPATH= \${CLASSPATH}:/opt/Java/JDK1.7/lib
- ❑ Also don't forget to include current directory, denoted by dot(.) to include in CLASSPATH, this will ensure that it will look first on current directory and if it found the class it will use that even if that class also exists in another directory which exists in CLASSPATH

Session 1: IDE(Integrated Development Environment)

- ❑ Integrated development environment (sometimes also called integrated debugging environment or interactive development environment or integrated design environment) is an GUI interface that allows programmers to build and test (and sometimes design) their application
- ❑ Typically IDE consists of
 - An editor where code can be written
 - A compiler: Some IDEs (like all of them listed in previous slide) compile as and when the code is written. In eclipse, redline underline is used to indicate compilation errors and yellow underline is used to indicate warnings
 - Run and Debug features
 - Tools to create language specific components
 - Context sensitive help (In eclipse, ctrl spacebar)
 - Tools for auto-build (packaging a JEE application in eclipse)

Session 1: Setup JDK

- ❑ <http://java.com/en/download/index.jsp> or find appropriate link in <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- ❑ Download Java based on the type of OS
 - ▣ Windows
 - ▣ Linux
 - ▣ Mac OS
 - ▣ Solaris
- ❑ Install JDK

Session 1: Basic Elements Of Java

- ❑ Variables, Literals , Keywords
- ❑ Data types
- ❑ Expressions, Statements and Blocks
- ❑ Operators
- ❑ Arrays
- ❑ Controls flow

The topics mentioned above come under language basics

Session 1: Variables

□ Variables

- A variable is the name that refers to a memory location where some data value is stored
- Each variable that is used in a program must be declared
- Variable name must begin with
 - ✓ a letter (A-Z, a-z, or any other language letters supported by UTF 16)
 - ✓ An underscore (_)
 - ✓ A dollar (\$)after which it can be any sequence of letters/digits
- Digits: 0-9 or any Unicode that represents digit
- Length of the variable name is unlimited
- Java reserved words should not be used as variable names

Session 1: Variable Naming Convention

- ❑ Must begin with lower case
- ❑ Must always begin with a letter, not "\$" or "_"
- ❑ Use meaningful names
- ❑ If a variable consists of two or more words, then the second and subsequent words should start with upper case. It should be in camel case
- ❑ For example, firstName

Session 1: Keywords/ Reserved words

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

Day 1: Primitive Data Types

- ❑ Primitive data types are basic data types
 - Integer type: **byte, short, int, long**
 - Floating point types: **float, double**
 - Character data types : **char**
 - Boolean data type: **boolean**

Session 1: Description Of Primitive Data Types

Group	Data Type	#bits	Range	Default value
Integer	byte	8	-128 to 127	0
	short	16	-32768 to 32767	0
	int	32	-2,147,483,648 to 2,147,483,647	0
	long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0

Session 1: Description Of Primitive Data Types(Continued)

Group	Data Type	#bits	Range	Default value
Floating point	float	32	- 3.4* e038 to +3.4*e038	0.0
	double	64	-1.7*e308 to +1.7*e308	0.0
Boolean	boolean	1	true or false	false
Character	char	16	A single character	Unicode character

Session 1: Literals

- A literal is the value assigned to a variable. For example 4, “A” is a literal
- String literals
 - ▣ Ex : “Nityo Infotech Pvt. Ltd”
- Boolean literals
 - ▣ Ex: true, false
- Character literals
 - ▣ Ex : ‘a’, ‘b’, ‘1’, ‘2’
- Integer literals
 - ▣ Ex: 123, 99877, 6753
- Floating literals
 - ▣ Ex: 323.544, 2.439f, 934937.955

Session 1: Conversions

- ❑ Widening Conversions (achieved by automatic or implicit conversion)
- ❑ Narrowing Conversions (achieved by casting or explicit conversion)
- ❑ All primitives are convertible to each other except for **boolean**

Screen 1: Automatic or implicit conversion

- ❑ The conversion in below direction happens automatically

byte → **short** → **int** → **long** → **float** → **double**
 ↑
 char

- ❑ When an arithmetic conversion happens between different numeric types, the result of the conversion is always of the higher numeric data type
- ❑ If an arithmetic operation happens between any integer type except **long**, the result is an **int**
- ❑ Similarly when arithmetic operation happens between floating points the result is **double**

Session 1: Loss of information

▣ There could be loss of some information during conversion of the following:

- **int → float**
- **long → float**
- **long → double**

Example

int i=12345678;

float f= i; → 1.2345678E7

Session 1: Assignment conversions with integers

- ❑ `int i=5;`
`int b=5;`
`int k=i+b; // ok`
- ❑ `byte b1=25; // ok`
`short s=25; // ok`

But `byte b=1234; // error`
- ❑ `int b=10;`
`byte b1=b; //error`
But `byte b1=(byte)b; //ok` this is called casting or explicit conversion
- ❑ `final int b=10;`
`byte b1=b; //ok`
- ❑ Integer literals are `int`
- ❑ When the integer literals are within the range of the specifies integer data types, the conversion automatically happen. In case the literal is beyond range of compiler flags an error

Session 1: Operators

- Arithmetic Operators
- Comparison Operators
- Boolean Operators
- Unary Operators
- Bitwise Operators
- isinstance

Session 1: Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular(Remainder)

- Operators precedence
 - First priority
 - * / %
 - Second priority
 - + -
 - Operators with higher priority are applied first
 - Operators with same priority are applied from left to right

Session 1: Arithmetic Operators conversions

- **byte b1=2, b2=3;**

int i=b1+b2; // ok

But **byte b=b1+b2; // error**

Same is the case with **short**

- Since arithmetic operations between integer types results in int (except when it involves long),
b1+b2 gives error (possible loss of precision)

- **byte b=b+1; // error**

But **byte b+=1; // ok**

because compiler converts this as **byte=(byte)(b+1);**

Session 1: Arithmetic Operators conversions(Continued)

- ❑ **byte c=10; // ok**

byte c1=-c; // error

byte c2=c1+1; // Error : possible loss of precision

But **byte c2=++c1; // ok**

- ❑ Compound assignment operator and pre/post increment/ decrement operator does the conversions required

Session 1: Explicit Conversion Or Casting

- ❑ Any conversion between primitives (excluding **boolean**) that is not possible implicitly can be done explicitly
- ❑ Conversions like (a) **double to long**, (b) **char to byte** etc
- ❑ This is done through casting

Example:

```
int k=10;
```

```
char b=k; // error
```

Casting makes the error disappear:

```
char b=(char)k;
```

```
byte b=(byte)128;
```

- ❑ *What will be the value when you print b?*

Session 1: Comparison Operators

==	Equality
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal

All above operators will return boolean value i.e. true or false

Session 1: Boolean Operators

Operators	Type	Definition
&& Example X && Y	Conditional AND	If both X and Y are true, result is true. If either X or Y are false result is false. If X is false then Y is not evaluated.
 Example X Y	Conditional OR	If either X or Y are true, result is true. If X is true then Y is not evaluated.
& Example X & Y	Boolean AND	If both X and Y are true, result is true. If either X or Y are false result is false. Both X and Y are evaluated before test.
 Example X Y	Boolean OR	If either X or Y are true, result is true. Both X and Y are evaluated before test.

Session 1: Unary Operators

Operator	Type	Definition
++	Increment	<pre>int a=2; int b=a++; // b=2 and a=3 Post increment Int b=++a; // b=3 and a=3 Pre increment</pre>
--	Decrement	<pre>int a=2; int b=a--; // b=2 and a=1 Int b=--a; // b=1 and a=1</pre>

Session 1: Integer Bitwise Operators

Operand 1	Operand2	&		^	~
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

It convert integer data type to their binary form then perform operations according to table.

Session 1: instanceof Operator

- Instanceof operator in java can be used to find whether an object is an instance of a particular class or not. This is handy when you pass an object as an instance of Object in a method argument and later inside your method code you may check what class is the passed argument an instance of and accordingly cast the generic Object instance to a specific object

Example:-

```
Class Animal{  
  
    public static void main(String as[]){  
  
        Animal obj=new Animal();  
  
        System.out.println(obj instanceof Animal); //true  
  
    }  
}
```

Session 1: Control statement

- ❑ Conditional statement

- **if** statement
- **switch** statement

- ❑ Loop statement

- **for** statement
- enhanced **for** statement
- **while** statement
- **do-while** statement

- ❑ Branch statement

- **break**
- **continue**
- **return**

Session 1: if statement

if (*conditional expression*) *<statement>*

else *<statement>*

```
if (x < 0) x = 2;
```

```
if (x > y) m = x; else m = y;
```

- ❑ **if** statement are same as that in C or C++ except that the condition must always evaluate to a **boolean** value

Session 1: if Statement(Continued)

- ❑ Nested if statement

if (<conditional expression>)

if (< conditional expression >)

// ...

<statement>

- ❑ Multiple if statement

if (< conditional expression 1> <statement 1>

else if (< conditional expression 2>) < statement 2>

...

else if (< conditional expression n>) < statement n>

else < statement>

Session 1: Switch Statement

- ❑ Syntax of switch statement

```
switch ( <expr.> ) {  
    case <expr. 1> : <statement 1>; break;  
    case <expr. 2> : < statement 2>;break;  
        :  
    case <expr. n> : < statement n>;break;  
    default : < statement>;  
}
```

- ❑ **switch** expression must be either integer value (not long) or enum or char. From Java SE 7 or later, **String** can also be used in **switch** statement's expression
- ❑ **case** expression must evaluate to a constant value
- ❑ **break** statement after every set of case statements will prevent fall-through

Session 1: Example of Switch Statement

```
switch (month) {  
    case 1: System.out.println("January");  
        break;  
    case 2: System.out.println("February");  
        break;  
    case 3: System.out.println("March");  
        break;  
    case 4: System.out.println("April");  
        break;  
    case 5: System.out.println("May");  
        break;  
    case 6: System.out.println("June");  
        break;
```



Session 1: Example Of Switch Statement Continued

```
case 7: System.out.println("July");  
        break;  
case 8: System.out.println("August");  
        break;  
case 9: System.out.println("September");  
        break;  
case 10: System.out.println("October");  
        break;  
case 11: System.out.println("November");  
        break;  
case 12: System.out.println("December");  
        break;  
Default: : System.out.println("Invalid Month No"); }
```

Session 1: for Loop

- Repeat the sequence of statement as many as defined
- Syntax of for statement

Initialization **condition** **iteration**



for (<expr 1> ; <expr 2> ; <expr 3>) <statement>

The diagram illustrates the syntax of a for loop. Three labels are positioned above the code: 'Initialization' above '<expr 1>', 'condition' above '<expr 2>', and 'iteration' above '<expr 3>'. Three blue arrows point from each label to its respective expression in the code.

- **Initialization** expression is used to initialize variable
 - more than one variable initialization is separated by commas “,”.
 - can also include initialization with declaration
 - executed only once when the loop begins

Session1: for Loop Continued

- **Condition** expression
 - Must evaluate to a **boolean** value
 - Loop will continue till the condition is **true**
 - Only one condition can be specified, multiple conditions can be combined using logical operators
 - is evaluated at the beginning of each loop
- The **iteration** expression
 - is usually an increment or decrement expression of the variable initialized in the initialization expression
 - is evaluated at the beginning of each loop

Session 1: for Loop Example

```
class Test
{
    public static void main(String as[])
    {
        for(int i=1; i<=10; i++)
        {
            System.out.println(2*i);
        }
    }
}
```

Output

2
4
6
8
10
12
14
16
18
20

Session 1: while and do-while Loop

- ❑ Syntax of while statement

while (cond. Expr.) <Statement>

Example:

```
1.      int i = 1; s = 0;
2.      while (i <= 10) { // summation from 1 to 10
3.          s += i; ++i;
4.      }
```

- ❑ **while** and **do-while** statement is used to iterate through a set of statements till the condition remains true
- ❑ **while** evaluates the condition before at the beginning of each iteration whereas **do-while** evaluates condition only at the end of each iteration
- ❑ **do-while** guarantees that the loop statements are executed at least once

Session 1: break And continue

- **break** is used to move control to out of the block. It is also used with switch statement
- **continue** is used to skip the rest of the statements in the loop and move control to the start of next repetition

Example :

```
int i = 1;
while (true) {
    if (i == 5)
        break;
    System.out.println(i);
    i++;
}
```

Output



1
2
3
4

Session 1: Label break Statement

- ❑ Can be used like goto statement

Example:-

```
class Test
{
    public static void main(String as[])
    {
        Outer:
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<2; j++)
            {
                System.out.println("Hello");
                break Outer;
            }
        }
    }
}
```

```
System.out.println("Hi");
}
System.out.println("End Of Main");
}
}
```

Output:
Hello
End Of Main

Session 1: continue Statement

- ❑ To move control to the start of next repetition

Example:

```
class Test{  
    public static void main(String as[]){  
        int a=0;  
        while(a<10){  
            a++;  
            if(a==5)  
                continue;  
            System.out.println(a);  
        }  
    }  
}
```

Output:

1
2
3
4
6
7
8
9
10

Session 1: Label continue Statement

Example:-

```
class Test
{
    public static void main(String as[])
    {
        Outer:
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<2; j++)
            {
                System.out.println("Hello");
                continue Outer;
            }
        }
    }
}
```

```
System.out.println("Hi");
}
System.out.println("End Of Main");
}
}
```

Output:

```
Hello
Hello
Hello
End Of Main
```

Session 1: return Statement

- To terminate the execution of method, then pass the method of caller that control




Example:-

```
class Test
{
    public static void main(String as[])
    {
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<2; j++)
            {
                System.out.println("Hello");
                return;
            }
        }
    }
}
```

```
System.out.println("Hi");
}
System.out.println("End Of Main");
}
}
```

Output:
Hello

Session 1: Array

- ❑ An array is an object that holds a fixed number of values of a same type
- ❑ Arrays in java are like objects, they are created in heap
- ❑ Array elements are automatically initialized to the default value based on the type of the array
- ❑ **new** keyword has to be used to create an array
- ❑ Types Of Array
 1. One-dimensional arrays
Ex- `int arr[];`  This is called dimension.
 2. Two-dimensional arrays
Ex- `int arr[][];`  Two dimension
 3. Multi-dimensional arrays
Ex- `int arr[][][];`  Three or multi dimension

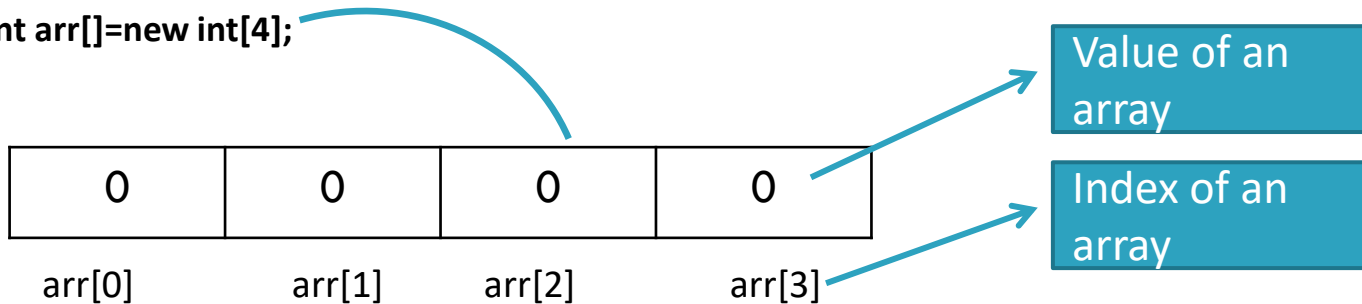
Session 1: Array Continued

□ How to create an Array

1. `int arr[]=new int[4];` //Creation of an Single Dimension Array
2. `int arr[][]=new int[4][2];` //Creation of Two Dimension Array
3. `int arr[][][]=new int[2][3][4];` //Creation of Multi Dimension Array

□ Structure of An Array in case of Single Dimension

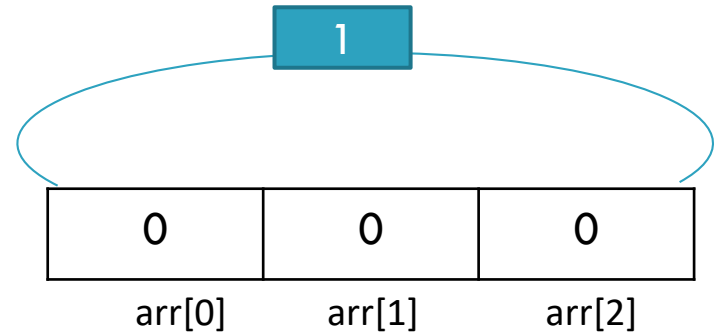
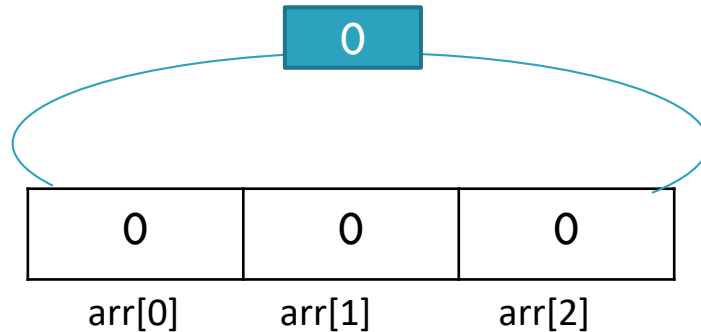
`int arr[]=new int[4];`



Session 1: Array Continued

- Structure of two Dimension Array

int arr[][]=new int[2][3];



- **int arr[][]=new int[][]{{1,2,3},{46,7,9},{6,4,2,1}};**
- **int arr[][]={{1,2,3},{46,7,9},{6,4,2,1}};**

Session 1: Array Continued

- Structure of multi dimension Array

```
int arr[][][]=new int[2][3][4];
```

```
int arr[][][]=new int[][][]{{{1,2,3},{4,6,8}},{{46,7,9},{6,4,2,1}}, },{{41,71,9},{61,4,2,1}}};
```

```
int arr[][][]={{{{1,2,3},{4,6,8}},{{46,7,9},{6,4,2,1}}, },{{41,71,91},{61,41,2,1}}};
```

Session 1: for Loop For Array Iteration

Example:

1. `int a[]={4,5,6,7,8,87,54};`
2. `for(int i=0; i<a.length; i++)`
3. `System.out.println(a[i]);`

❑ Or we can use enhanced **for loop** (known as for-each) for convenient way to iterate

Example:

4. `for(int i:a)`
5. `System.out.println(i);`

The variable in the **for-each** statement must be of same type as that of elements in the array

Session 1: Access Specifiers

- ❑ Specify the access to our code for other classes – whether other classes can access or not and if permitted, to what extent they can access. Java includes access modifiers also which are quite different from access specifiers
- ❑ Type of access specifiers
 - **Private:-** Members are accessible only to the class
 - **default(no specifier):-** Members are accessible only to the class or other class within same package
 - **Protected:** - Members are accessible only to the class or other class within same package but outside the package it will be accessible within the subclass only
 - **Public:** - Members are accessible to the class and to other classes within the package or outside the package

Session 1: Non Access Modifiers

- ❑ A modifier is a keyword placed in a class, method or variable declaration to determine how it is used in other classes
- ❑ Types of Non Access Modifiers
 - final
 - static
 - abstract
 - synchronized
 - transient
 - native
 - Strictfp

Session 1: Non Access Modifiers(Continued)

❑ **final**

- A final variable cannot be changed
- A final method cannot be overridden in the subclass
- A final class cannot be inherited

❑ **static**

- In Java, a *static* member is a member of a class that isn't associated with an instance of a class
- Only variable , method and inner class can be static
- Instance(object) is not required for calling the static variable or method. It belongs to the class
- A **single copy** of static variable and method to be shared by all instances of the class

Session 1: Non Access Modifiers(Continued)

❑ **abstract**

- Used to declare class that provides common behavior across a set of subclasses
- Only class and method can be abstract
- Abstract class provides a common root for a group of classes
- An abstract keyword with a method does not have any definition
- In detail we discuss in next day session

❑ **synchronized**

- Only method can be synchronized
- It is used in multithreaded environment
- In detail we discuss in multithreading session

Session 1: Non Access Modifiers(Continued)

❑ **transient**

- A transient variable is a variable that may not be serialized i.e. JVM understands that the indicated variable is not part of the persistent state of the object
- In detail we discuss in IO serialization session

❑ **native**

- Used only with methods
- It notify the compiler that the method has been coded in a language other than Java
- It indicates that method lies outside the Java Runtime Environment

Session 1: Non Access Modifiers(Continued)

❑ **strictfp**

- strictfp can be used to modify a class or a method, but never a variable
- Modifying the class as strictfp means that any method code in the class will conform to the IEEE 754 standard rules for floating points
- Without that modifier floating points used in the methods might behave in a platform-dependent way

Session 1: Method Overloading

- ❑ Overloading refers to the methods in a class having same name but different arguments
- ❑ If same method names are used to define multiple methods in a class, then the methods are said to be overloaded
- ❑ Methods are identified with the parameters set based on:
 - the number of parameters
 - types of parameters
 - order of parameters
- ❑ Method overloading is used to perform same task with different available data
- ❑ Example:
 1. class Calculate {
 2. int sum(int a,int b) { return a+b;}
 3. int sum(int a,int b,int c) { return a+b+c;}
 4. float sum(float a, float b, float c) { return a+b+c;}
 5. }

Session 1: Constructors

- ❑ The constructor is a special method for every class that helps initialize the object members at the time of creation
- ❑ A constructor method has the same name as the that of the class
- ❑ does not have a return type
- ❑ Can be called when object gets created
- ❑ It is used to initialized instance variable of class
- ❑ It can use any access specifier but most often defined with 'public' so that every program can create an instance
- ❑ If there is no constructor in class, a default constructor is used to create instances of the class. A default Constructor is provided by compiler
- ❑ A class can have more than one constructors

Session 1: Constructors Example

```
1. public class Employee{
2.     private String name;
3.     private int empId;
4.     private String department;
5.     public Employee(String n, int id, String dep ){
6.         name=n;
7.         empId=id;
8.         department=dep;
9.     }
10. }
```


Session 1: this Reference

- “this” is a reference to the current object
- “this” is used to
 - ▣ refer to the current object when it has to be passed as a parameter to a method or constructor

Line 1. obj1.methodName(this);

- ▣ refer to the current object when it has to be returned in a method
- ▣ Refer the instance variables if parameter names are same as instance variables to avoid ambiguity

Line 2. public void method1(String name){

Line 3. this.name=name;

}

Session 1: this Reference(Continued)

- The “this” keyword with parenthesis i.e. this() with or without parameters is used to call another constructor within same class
- The this() can be called only from a constructor and must be the first statement in the constructor

Example: The default constructor for the Employee class can be redefined as below

```
Line 1.      public Employee(){  
Line 2.                  this("Saurabh",2383,"L&K");  
Line 3.      }
```

Session 1: Compiler Generated Constructor

- When no constructors are defined for a class, compiler automatically inserts a constructor that takes no argument

Example:-

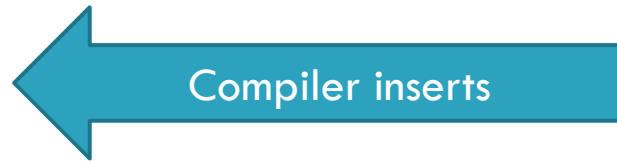
```
public class Employee{
```

```
String name
```

```
int empld
```

```
.....
```

```
}
```



Compiler inserts

```
Public Employee()  
{  
    super();  
}
```

Day 1: Package

- ❑ A package is a namespace that organizes the a set of related classes and interfaces. It also defines the scope of a class. An application consists of thousands of classes and interfaces, and therefore, it is very important to organize them logically into packages
- ❑ Packages are implemented using file system directories
- ❑ Package must be the first statement in the java source file
- ❑ Syntax for creating packages:
package package name[.package name];
- ❑ Types of Java packages
 - Built in/Java defined packages
 - User defined packages

Session 1: Benefits of Package

- ❑ Modularity
- ❑ Easier Application Design
- ❑ Hiding Information
- ❑ Eliminates name collision
- ❑ Better Performance
- ❑ Overloading

Session 1: import Keyword

- ❑ The classes in the packages are to be included into the applications as required
- ❑ The classes of a package are included using import keyword
- ❑ The import keyword followed by fully qualified class name is to be placed before the application class definition
 - `import packaename.classname ;`
 - `// Class Definition`
 - Ex: `import Java.util.Date;`
 - `import Java.io.*;`
 - `// Class Definition`
- ❑ above indicates to import all the classes in a package
- ❑ All the classes in Java.lang package are included implicitly

Session 1: User Defined Packages

- ❑ When we write a Java program, we create many classes. We can organize these classes by creating our own packages
- ❑ The packages that we create are called user-defined packages
- ❑ A user-defined package contains one or more classes or interface that can be imported in a Java program

Session 1: User Defined Packages(continued)

Syntax

1. `package <package_name>`
 `// Class definition`
2. `public class <classname1>`
3. `{`
4. `// Body of the class`
5. `}`

Example

```
package vehicle;  
public class Car  
{  
    String brand;  
    String color;  
}
```


Session 1: Importing User Defined Packages

- You can include a user-defined package or any Java API using the import statement
- The following syntax shows how to implement the user-defined package
 1. `import vehicle.Car;`
 2. `public class Honda extends Car`
 3. `{`
 4. `// Body of the class`
 5. `}`

Session 1: Static Imports

- Makes the static members of a class available to code directly without explicitly specifying the class name

- Syntax:

import static packageName.ClassName.*;

(imports all the static members)

Or

import static packageName.ClassName.staticMember;

(imports only the particular 'staticMember')

Session 1: Example Of Static Imports

```
1. import static java.lang.System.*;
2. class Test
3. {
4.     public static void main(String as[])
5.     {
6.         out.println("Hello");
7.     }
8. }
```



Instead of System.out.println

Session 1: Java Source File Declaration Rule

- A java source file can contain:
 - a single package statement which will be the first statement
 - any number of import statements. They should be between package and class statement
 - any number of classes with default access specifier. In such cases, the file name could be 'anything'.java
 - only one public class. If there is a public class then the file must be named after the public class name
 - The package and the import statements apply to all the classes in the source file

Day 1: Questions

