Core Java

Trainer : Ameya Joshi
Email : ameya.joshi@vinsys.com
Mobile : +919850676160
LinkedIn : https://www.linkedin.com/in/amsalways

1

Course Agenda

- **Java Platform Architecture**
- **Java Programming Language**
- **Classes and Objects**
- **Inheritance and Polymorphism in Java**
- **Exception Handling**
- **IO Streams in Java**
- **Multi threading**
- **Java Database Connectivity**
- **Java 8 Features**

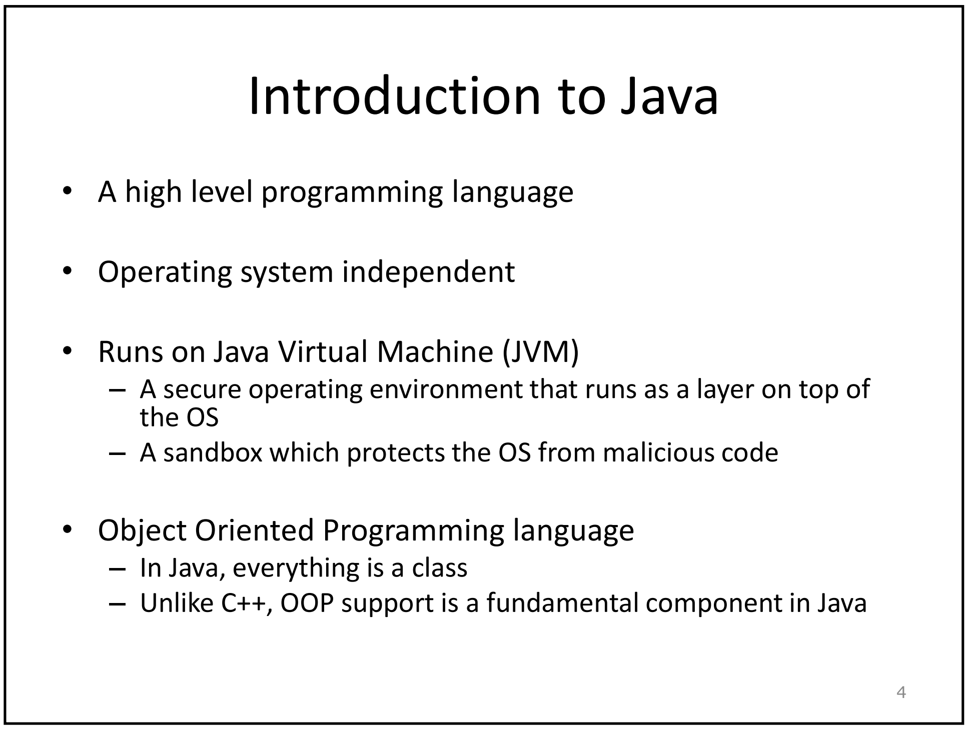
2

2

Core Java

Java Basics

3



Introduction to Java

- A high level programming language
- Operating system independent
- Runs on Java Virtual Machine (JVM)
  - A secure operating environment that runs as a layer on top of the OS
  - A sandbox which protects the OS from malicious code
- Object Oriented Programming language
  - In Java, everything is a class
  - Unlike C++, OOP support is a fundamental component in Java

4

4

Features of Java

- Object Oriented
- Simple
  - Compared to earlier OO languages like C++, it is simple
- Robust
- Secure
  - Absence of pointers

5

5

Features of Java (Contd…)

- Support for Multithreading at language level
- Designed to handle Distributed applications
- Architecture Neutral / Portable:
  - Java code compiled on Windows can be run on Unix without recompilation

6

6

Platform Independence

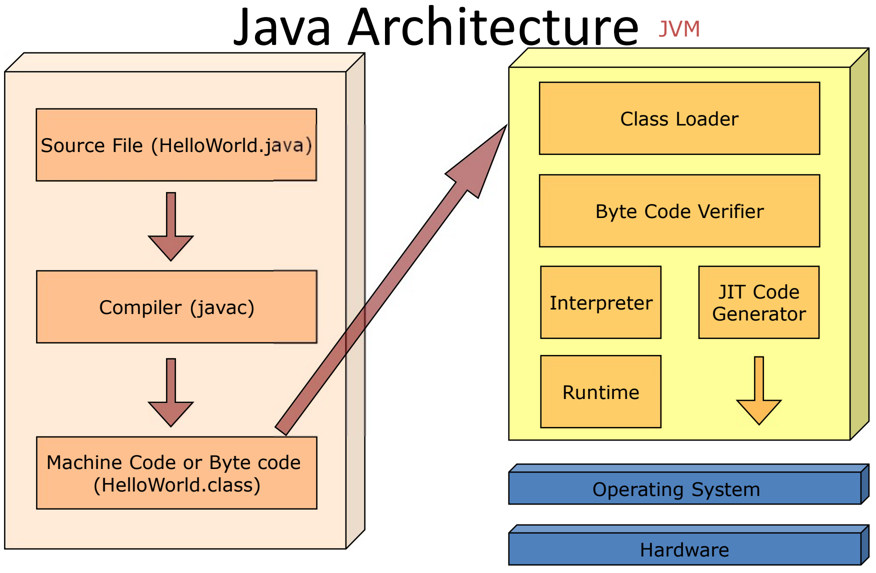
- A platform is the hardware & software environment in which a program runs
- Once compiled, java code runs on any platform without recompiling or any kind of modification

  "Write Once Run Anywhere"
- This is made possible by the Java Virtual Machine (JVM)

7

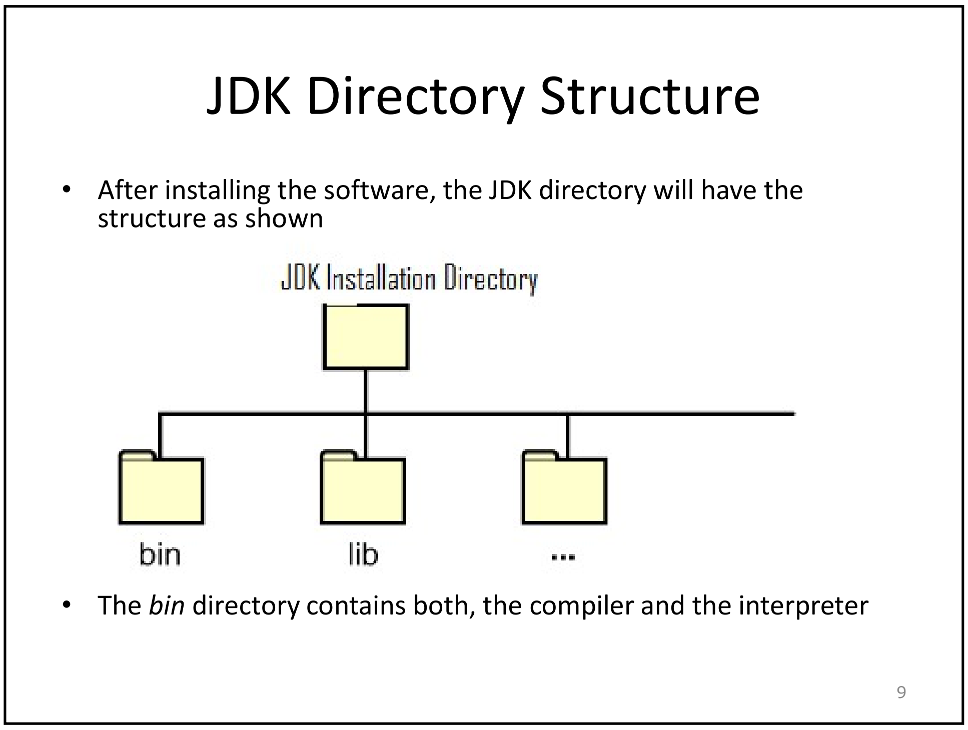
7

Java Architecture



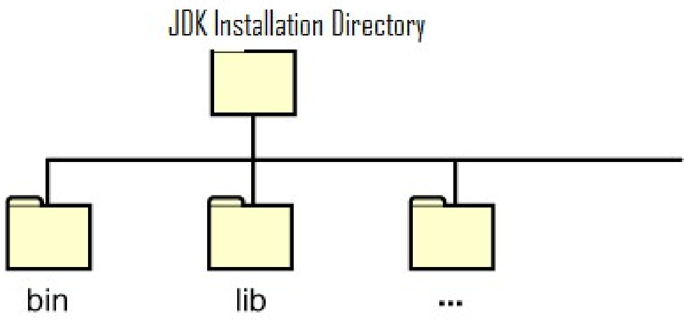
8

8



JDK Directory Structure

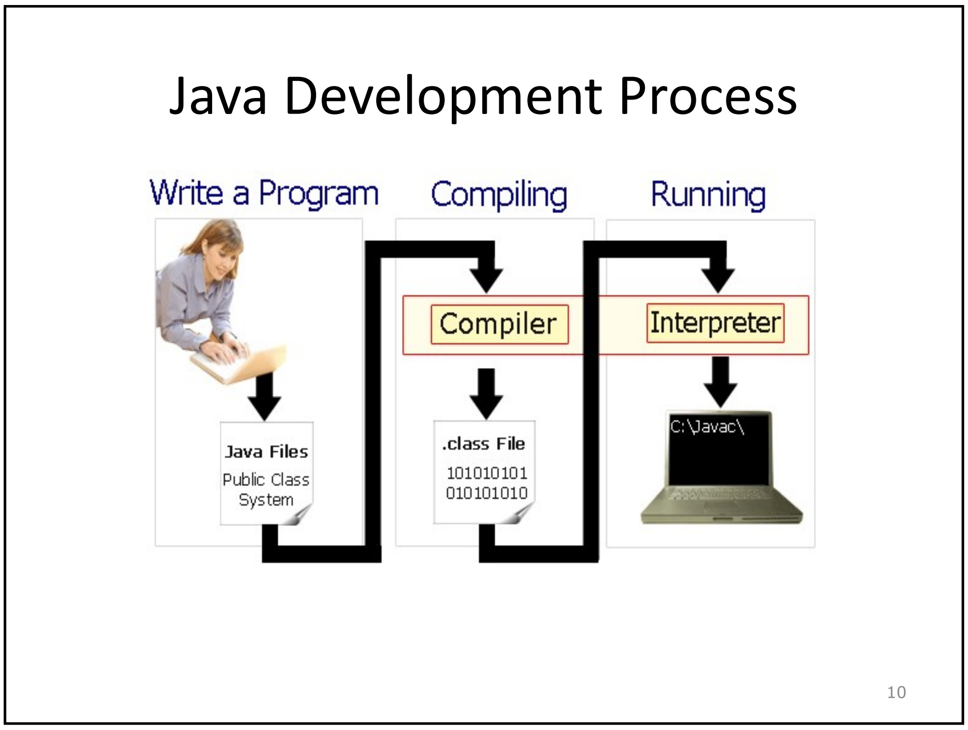
- After installing the software, the JDK directory will have the structure as shown



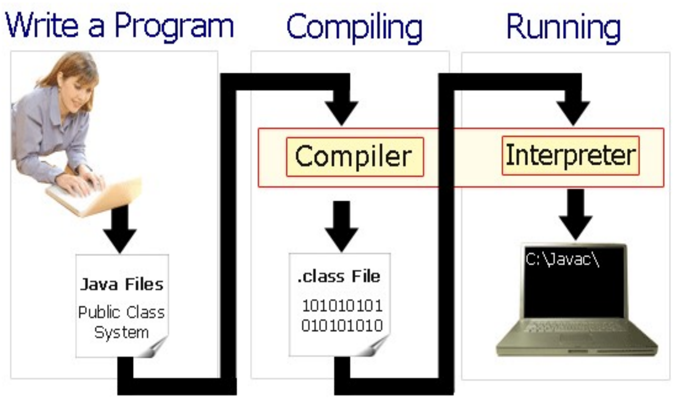
- The *bin* directory contains both, the compiler and the interpreter

9

9



Java Development Process



10

10

Java Virtual Machine (JVM)

- The source code of Java is stored in a text file with the extension .java
- The Java compiler compiles a .java file into byte code
- The byte code will be in a file with extension .class
- The generated .class file is the machine code of this processor
  - Byte code is in binary language
- The byte code is interpreted by the JVM

11

11

Java Virtual Machine (JVM) (Contd…)

- JVM makes Java platform independent
- The JVM interprets the .class file to the machine language of the underlying platform
- The underlying platform processes the commands given by the JVM

12

12

Environment Variables in JVM

- JAVA_HOME: Java Installation Directory
  - Used to derive all other environment variables used by JVM

In Windows	`set C:\Program Files\Java\jdk11.0_171`
In UNIX	`export JAVA_HOME=/var/usr/java`

- CLASSPATH:
  - Used to locate class files

In Windows	`set CLASSPATH=%CLASSPATH%;%JAVA_HOME%\lib\tools.jar`
In UNIX	`set CLASSPATH=$CLASSPATH:$JAVA_HOME/lib/tools.jar`

13

13

Source File Layout - Hello World

- Type the source code using any text editor

```
public class HelloWorldApp {
     public static void main(String[]args){
             System.out.println(“Hello World!”);
     }
}
```

- Save this file as *HelloWorldApp.java*

14

14

To Compile

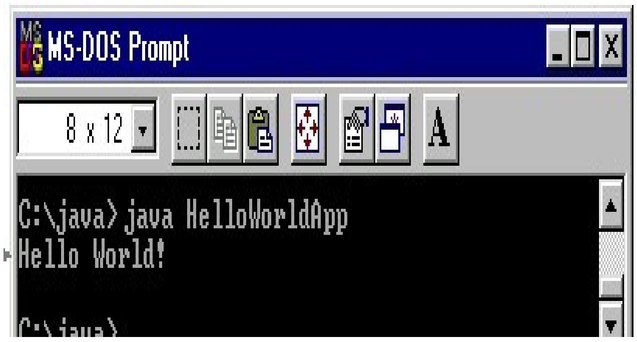
- Open the command prompt
- Set the environment variables
- Go to the directory in which the program is saved
- Type - javac HelloWorldApp.java
  - If it says, “bad command or file name” then check the path setting
  - If it returns to prompt without giving any message, it means that compilation is successful

15

15

To Execute

- Type the command - java HelloWorldApp
- The result will be



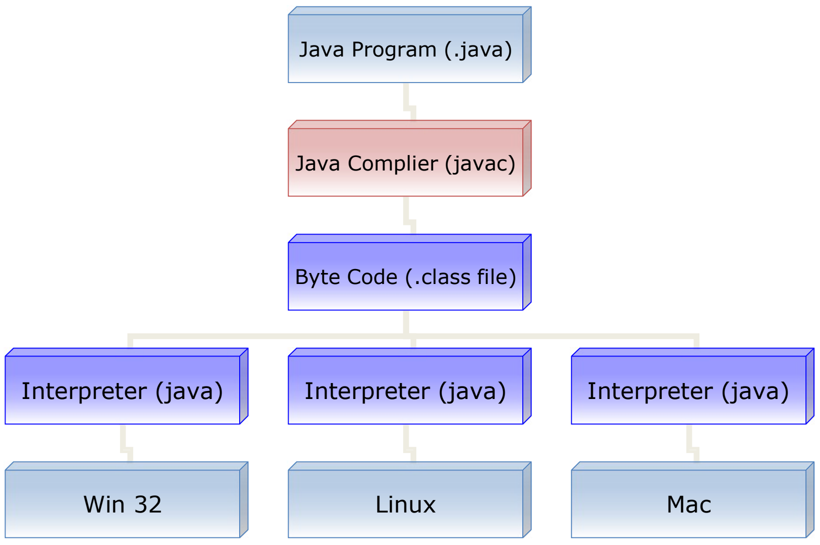
```
C:\java> java HelloWorldApp
Hello World!

C:\java>
```

16

16

Compilation & Execution



17

17

Best Practices

- Only put one class in one source file
- Provide adequate comments in the program
- Properly indent the program
- Follow coding standards for identifiers

18

18

Java Keywords

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

* Keywords not in use now

19

19

Java Identifiers

- Declared entities such as variables, methods, classes & interfaces are Java Identifiers
- May contain letters, digits, underscore(_) & dollar sign ($)

20

20

Data Types in Java

- Java is a strongly typed language
  - Unlike C, type checking is strictly enforced at run time
  - Impossible to typecast incompatible types
- Data types may be:
  - Primitive data types
  - Reference data types

21

21

Primitive Data Types in Java

Integer Data Types	
byte	(1 byte)
short	(2 bytes)
int	(4 bytes)
long	(8 bytes)
Floating Data Types	
float	(4 bytes)
double	(8 bytes)
Character Data Types	
char	(2 bytes)
Logical Data Types	
boolean	(1 bit) (true/false)

- All numeric data types are signed
- The size of data types remain same on all platforms
- *char* data type is 2 bytes as it uses the UNICODE character set. And so, Java supports internationalization

22

22

Variables

- A named storage location in the computer's memory that stores a value of a particular type for use by program.

- Example of variable declaration:

```
DataType          variableName
int               myAge, cellPhone;
double            salary;
char              tempChar;
```

- The data type can either be:
  - built-in *primitive* types (e.g. int, double, char object classes)
  - *reference* data types (e.g. String,BufferedReader)

- Naming Convention →

Variable Name: First word lowercase & rest initial capitalized (Camel Casing)

e.g. thisIsALongVariableName

23

23

Variables (Contd…)

- Using primitive data types is similar to other languages

```
int count;
int max=100;
```

- Variables can be declared anywhere in the program

```
for (int count=0; count < max; count++) {
  int z = count * 10;
}
```

BEST PRACTICE

Declare a variable in program only when required

Do not declare variables upfront like in C

- In Java, if a local variable is used without initializing it, the compiler will show an error

24

24

Access specifiers/modifiers

- default
  - Are accessible in same class and from other class but with limitation.
- public
  - Are accessible from anywhere
- private
  - Are accessible only from within the class
- protected
  - Are similar to private but can be inherited.

25

25

How many of these are valid Java Identifiers?

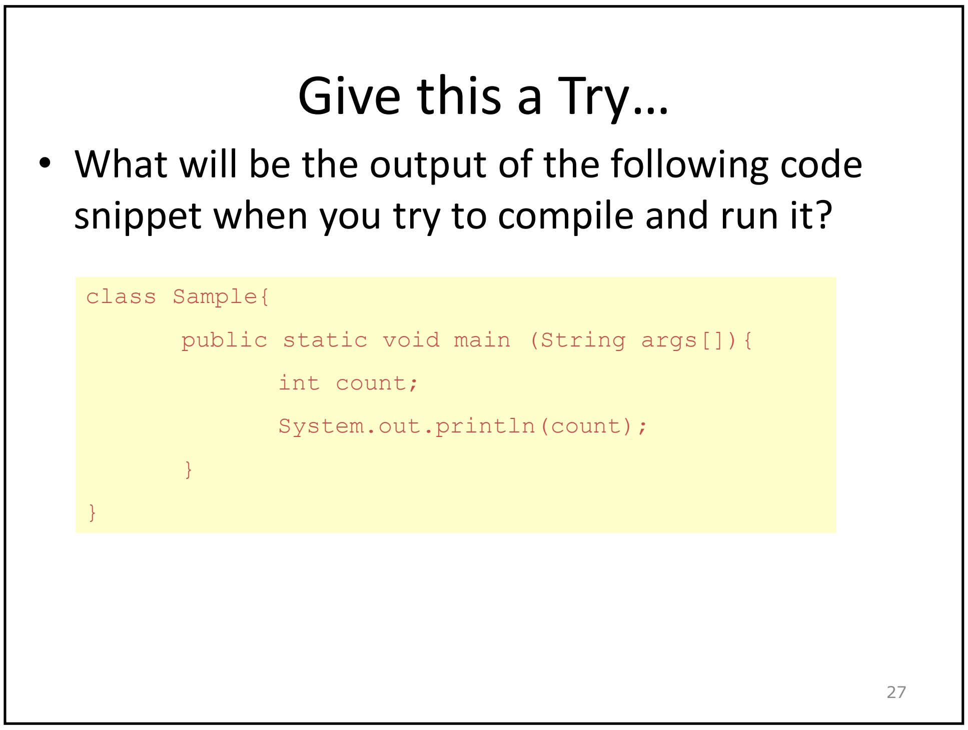
Give this a Try…

```
78class              Class87              sixDogs
User$ID              Jump_Up_             DEFAULT_VAL
False                Private              Average-Age
Hello!               First One            String

A. 5
B. 6
C. 7
D. 8
E. 9
```

26

26



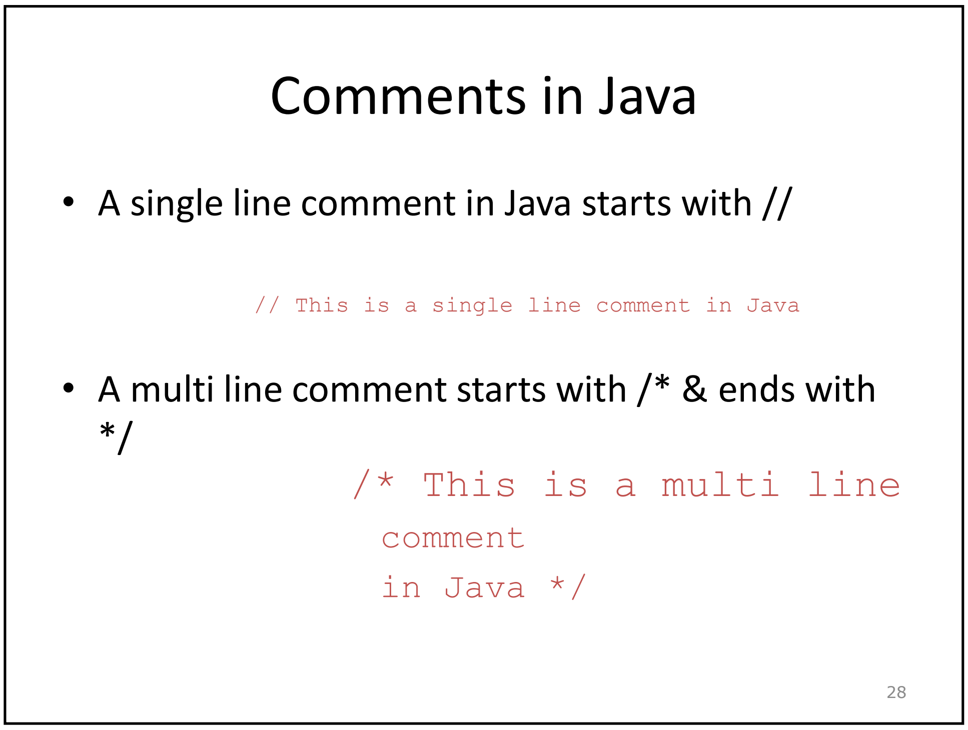
Give this a Try…

- What will be the output of the following code snippet when you try to compile and run it?

```
class Sample{
       public static void main (String args[]){
              int count;
              System.out.println(count);
       }
}
```

27

27



Comments in Java

- A single line comment in Java starts with //

```
// This is a single line comment in Java
```

- A multi line comment starts with /* & ends with */

```
/* This is a multi line
 comment
 in Java */
```

28

28

Reference Data Types

- Hold the reference of dynamically created objects which are in the heap
- Can hold three kinds of values:
  - Class type: Points to an object / class instance
  - Interface type: Points to an object, which is implementing the corresponding interface
  - Array type: Points to an array instance or "*null*"
- Difference between Primitive & Reference data types:
  - Primitive data types hold values themselves
  - Reference data types hold reference to objects, i.e. they are not objects, but reference to objects

29

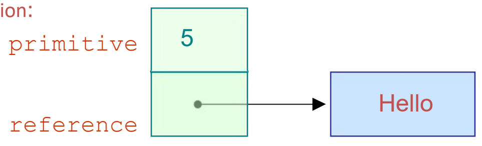
29

Reference Data Types (Contd…)

- Objects & Arrays are accessed using *reference variables* in Java
- A reference variable is similar to a pointer (stores memory address of an object)
- Java does not support the explicit use of addresses like other languages
- Java does not allow pointer manipulation or pointer arithmetic

```
int primitive = 5;
String reference = "Hello" ;
```

- Memory Representation:



30

30

Reference Data Types (Contd…)

- A reference type cannot be cast to primitive type
- A reference type can be assigned 'null' to show that it is not referring to any object

31

31

Typecasting Primitive Data Types

- Automatic type changing is known as *Implicit Conversion*
  - A variable of smaller capacity can be assigned to another variable of bigger capacity

```
 int i = 10;
double d;
 d = i;
```

- Whenever a larger type is converted to a smaller type, we have to explicitly specify the *type cast operator*

```
double d = 10
int i;
i = (int) d;
```

Type cast operator

- This prevents *accidental loss* of data

32

32

Java Operators

- Used to manipulate primitive data types
- Classified as unary, binary or ternary
- Following are different operators in Java:
  - Assignment
  - Arithmetic
  - Relational
  - Logical
  - Bitwise
  - Compound assignment
  - Conditional

33

33

Java Operators (Contd…)

Assignment Operators	=					
Arithmetic Operators	- +	*	/	%	++	
	--					
Relational Operators	> <	>=	<=	==	!=	
Logical Operators	&&	||	!			
Bit wise Operator	& |	^	>>	>>>	<<	
Compound Assignment Operators	+=	-=	*=	/=	%=	
Conditional Operator	?:					

34

34

Precedence & Associativity of Java Operators

- Decides the order of evaluation of operators
- Click below to check all Java operators from highest to lowest precedence, along with their associativity

Give this a Try…

- What is the result of the following code fragment?

```
int x = 5;
int y = 10;
int z = ++x * y--;
```

Control Structures

- Work the same as in C / C++

if/else, for, while, do/while, switch

```
i = 0;
while(i < 10) {
    a += i;
    i++;
}
```

```
for(i = 0; i < 10; i++) {
    a += i;
}
```

```
if(a > 3) {
    a = 3;
}
else {
    a = 0;
}
```

```
switch(i) {
    case 1:
        string = "foo";
    case 2:
        string = "bar";
    default:
        string = "";
}
```

```
i = 0;
do {
    a += i;
    i++;
} while(i < 10);
```

37

37

Control Structures (Contd…)

- Java supports continue & break keywords also
- Again, work very similar to as in C / C++
- Switch statements require the condition variable to be a char, byte, short or int

```
for(i = 0; i < 10; i++) {
    if(i == 5)
        continue;
    a += i;
}
```

```
for(i = 0; i < 10; i++) {
    a += i;
    if(a > 100)
        break;
}
```

38

38

Give this a Try…

What do you think is the output if aNumber is 3?

```
if (aNumber >= 0){


 if (aNumber == 0)
   System.out.println("first string");
else
 System.out.println("second string");
 System.out.println("third string");
}
```

39

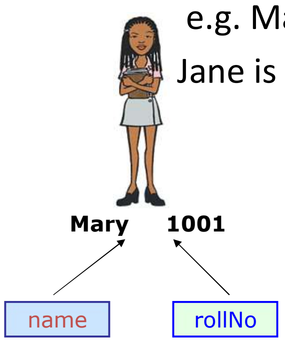
39

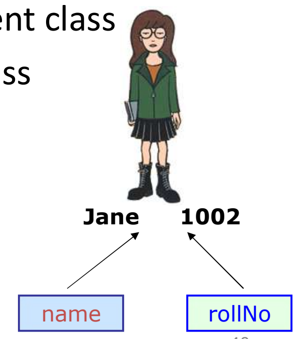
Concept of Class

- A class is a description of a group of objects with common properties (attributes) & behavior (operations)
  - An object is an instance of a class

e.g. Mary is an object of Student class

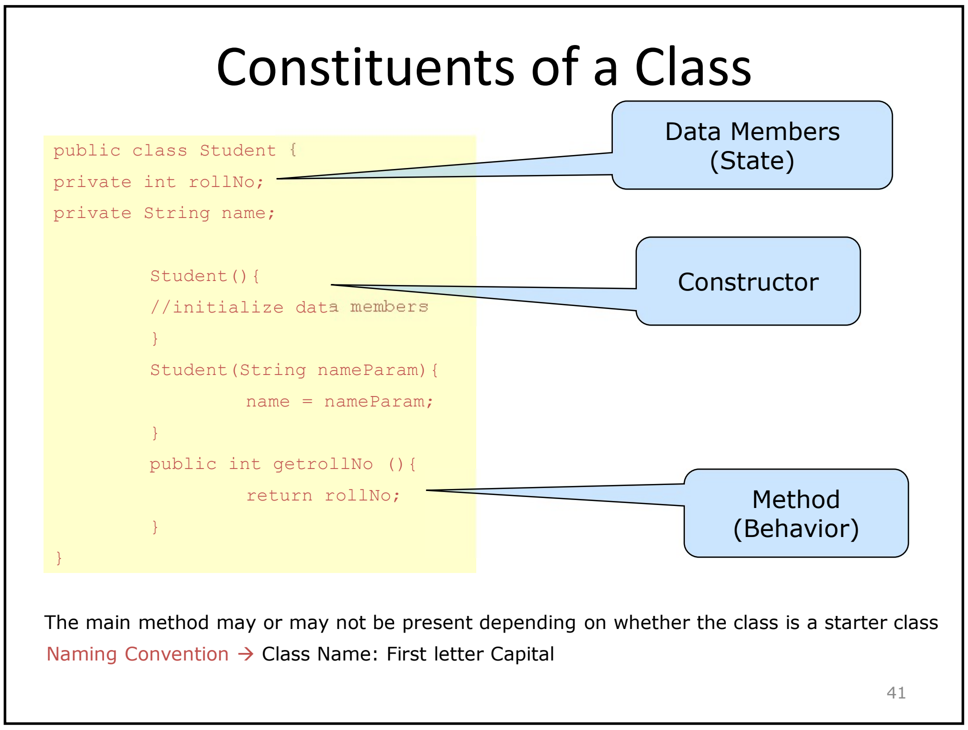
Jane is an object of Student class





40

40



Constituents of a Class

```
public class Student {
private int rollNo;
private String name;

        Student(){
        //initialize data members
        }
        Student(String nameParam){
                name = nameParam;
        }
        public int getrollNo (){
                return rollNo;
        }
}
```

Data Members (State)

Constructor

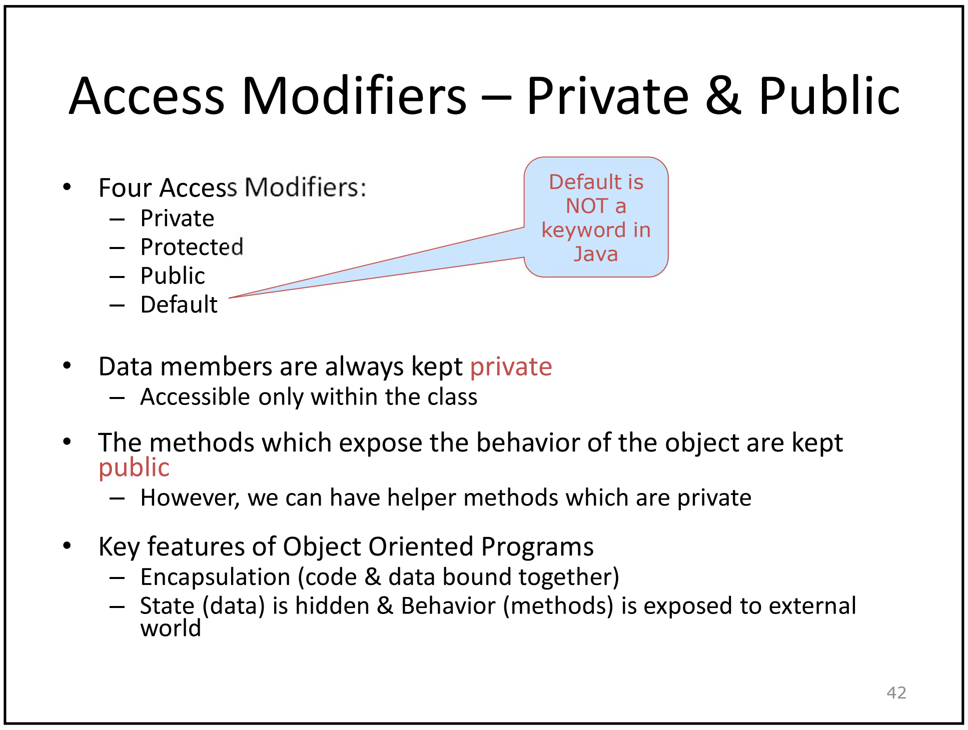
Method (Behavior)

The main method may or may not be present depending on whether the class is a starter class

Naming Convention → Class Name: First letter Capital

41

41



Access Modifiers – Private & Public

- Four Access Modifiers:
  - Private
  - Protected
  - Public
  - Default

Default is NOT a keyword in Java

- Data members are always kept private
  - Accessible only within the class
- The methods which expose the behavior of the object are kept public
  - However, we can have helper methods which are private
- Key features of Object Oriented Programs
  - Encapsulation (code & data bound together)
  - State (data) is hidden & Behavior (methods) is exposed to external world

42

42

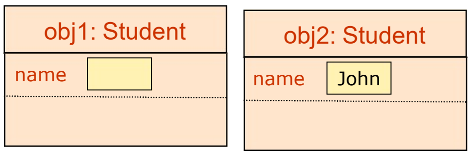
Creating Objects

- The *new* operator creates a object & returns a reference to it
- Memory allocation of objects happens in the heap area
- Reference returned can be stored in reference variables

```
        Student obj1;
        obj1 = new Student();
        Student obj2 = new
Student("John");
```

obj1 is a reference variable

new keyword creates an object and returns a reference to it



43

43

Constructors

- Special methods used to initialize a newly created object
- Called just after memory is allocated for an object
- Initialize objects to required or default values at the time of object creation
- Not mandatory to write a constructor for each class
- A constructor
  - Has the same name as that of the class
  - Doesn't return any value, not even `void`
  - May or may not have parameters (arguments)
- If a class does not have any constructor, the default constructor is automatically added

44

44

Constructors (Contd…)

- In the absence of a user defined constructor, the compiler initializes member variables to its default values
  - Numeric data types are set to 0
  - Char data types are set to null character ('\0')
  - Reference variables are set to *null*

45

45

Lifetime of Objects

```
Student obj1 = new student();
Student obj2 = new student();
```

Both Student objects now live on the heap

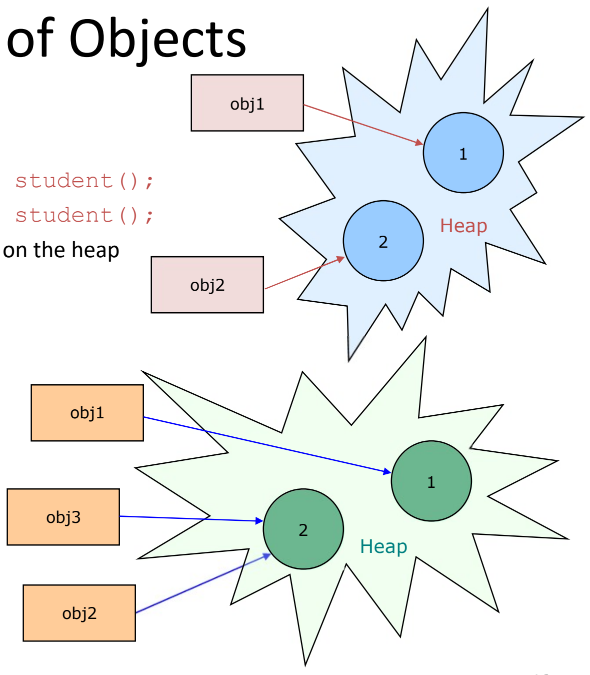
→ References : 2

→ Objects : 2

```
Student obj3 = obj2;
```

→ References : 3

→ Objects : 2



46

46

Garbage Collection

- In C, it is the programmer's responsibility to de-allocate the dynamically allocated memory using the *free()* function
- JVM automatically de-allocates memory (Garbage Collection)
- An object which is not referred by any reference variable is removed from memory by the Garbage Collector
- Primitive types are not objects & cannot be assigned *null*
- *Finalizer method : protected void finalize(){} -> Object finalizer*

47

47

Lifetime of Objects (Contd...)

```
obj3 = obj1;
```

→ References : 3

→ Objects : 2

```
obj2 = null;
```

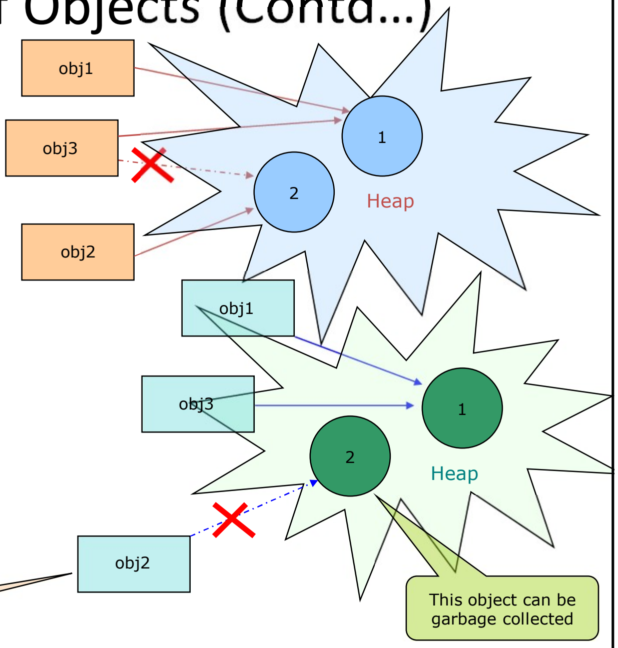
→ Active References : 2

→ Null References : 1

→ Reachable Objects : 1

→ Abandoned objects : 1

Null Reference



48

48

Scope of Variables

- Instance Variables (also called Member Variables)
  - Declared inside a class
  - Outside any method or constructor
  - Belong to the object
  - Lifetime depends on the lifetime of object
- Local Variables (also called Stack Variables)
  - Declared inside a method
  - Method parameters are also local variables
  - Stored in the program stack along with method calls and live until the call ends

49

49

Scope of Variables (Contd…)

- If we don't initialize instance variables explicitly, they are awarded predictable *default initial values*, based only on the type of the variable

Type	Default Value
boolean	false
byte	(byte) 0
short	(short) 0
int	0
long	0L
char	\u0000
float	0.0f
double	0.0d
object reference	null

- Local variables are not initialized implicitly

50

50

Scope of Variables (Contd…)

```
class Student{
   int rollNo;
   String name;
   public void display (int z){
      int x=z+10;
    }
 }
```

rollNo and name are instance variables to be stored in the heap

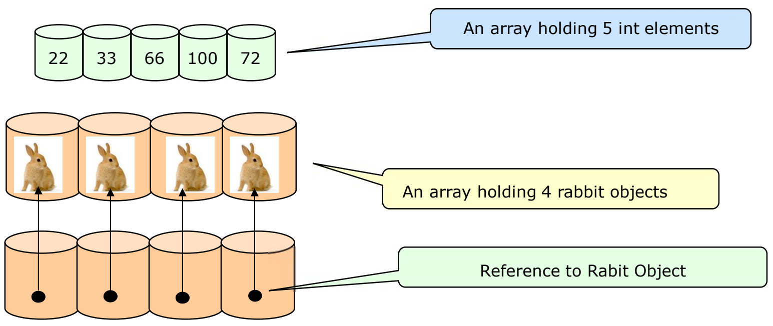
z and x are local variables to be stored in the stack

51

51

Arrays in Java

- A data structure which defines an ordered collection of a fixed number of homogeneous data elements
- Size is fixed and cannot increase to accommodate more elements
- Arrays in Java are objects and can be of primitive data types or reference variable type
- All elements in the array must be of the same data type



22
33
66
100
72
An array holding 5 int elements
An array holding 4 rabbit objects
Reference to Rabit Object

52

52

Arrays in Java (Contd…)

- *Reference variables* are used in Java to store the references of objects created by the operator *new*

```
int x[];
int [] x;
```

- Any one of the following syntax can be used to create a reference to an *int* array

```
//Declare a reference to an int array
int [] x;
//Create a new int array and make x refer to it
x = new int[5];
```

- The reference x can be used for referring to any *int* array

53

53

Arrays in Java (Contd…)

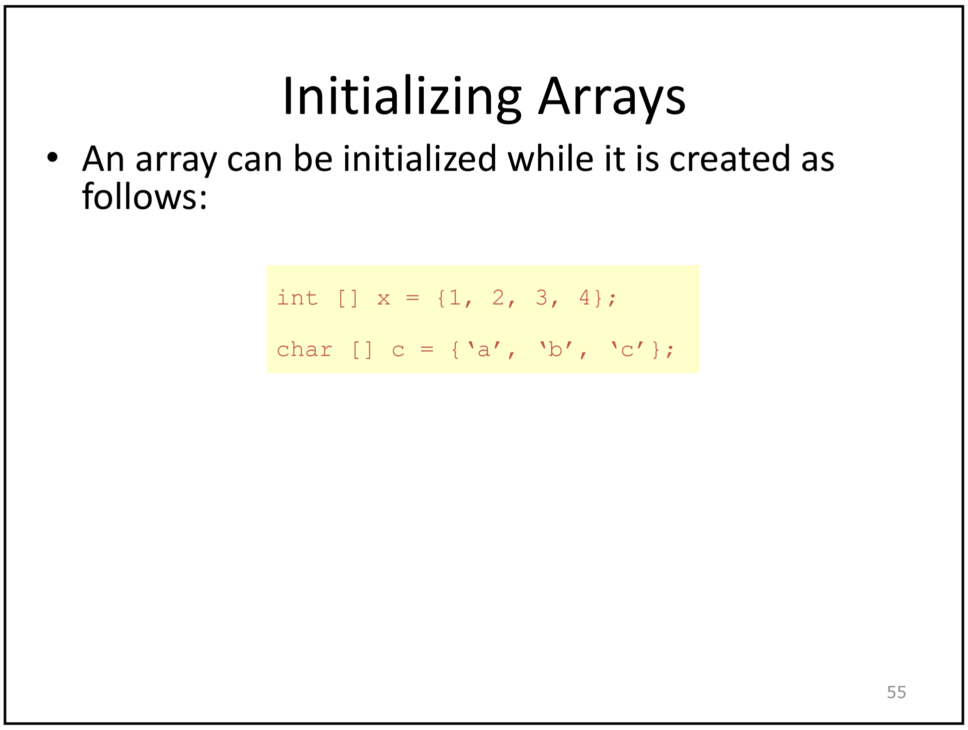
- The following statement also creates a new *int* array and assigns its reference to x

```
int [] x = new int[5];
```

- In simple terms, references can be seen as names of an array

54

54



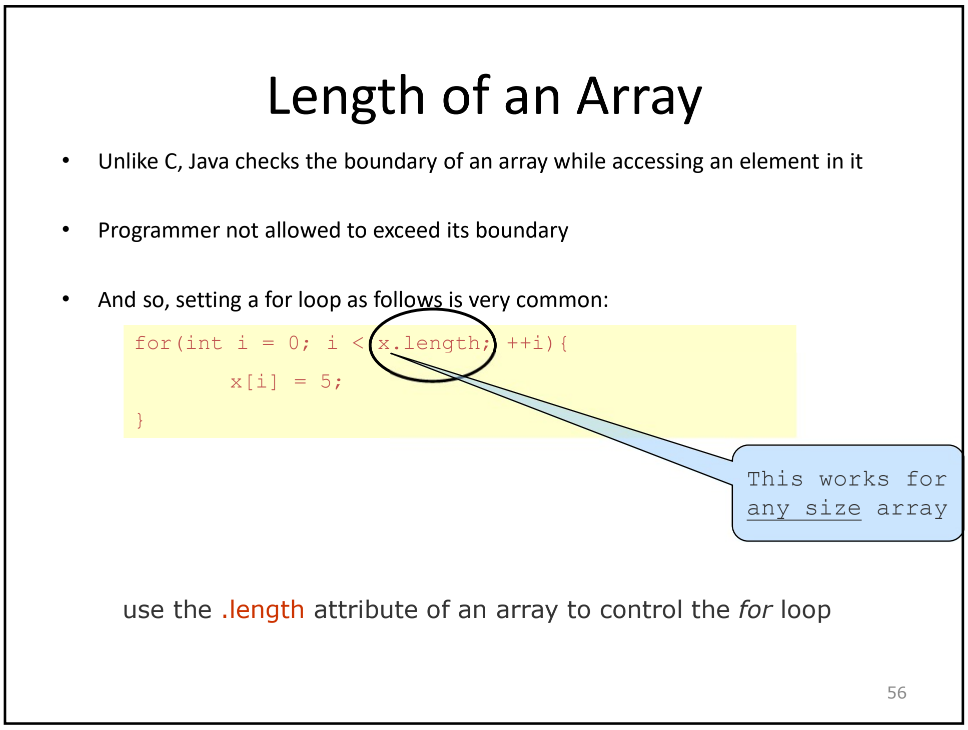
Initializing Arrays

- An array can be initialized while it is created as follows:

```
int [] x = {1, 2, 3, 4};
char [] c = {'a', 'b', 'c'};
```

55

55



Length of an Array

- Unlike C, Java checks the boundary of an array while accessing an element in it
- Programmer not allowed to exceed its boundary
- And so, setting a for loop as follows is very common:

```
for(int i = 0; i < x.length; ++i){
        x[i] = 5;
}
```

This works for any size array

use the .length attribute of an array to control the *for* loop

56

56

Multidimensional Arrays

- A Multi-dimensional array is an array of arrays
- To declare a multidimensional array, specify each additional index using another set of square brackets

```
int [][] x;
//x is a reference to an array of int arrays
x = new int[3][4];
//Create 3 new int arrays, each having 4 elements
//x[0] refers to the first int array, x[1] to the second and
so on
//x[0][0] is the first element of the first array
//x.length will be 3
//x[0].length, x[1].length and x[2].length will be 4
```

57

57

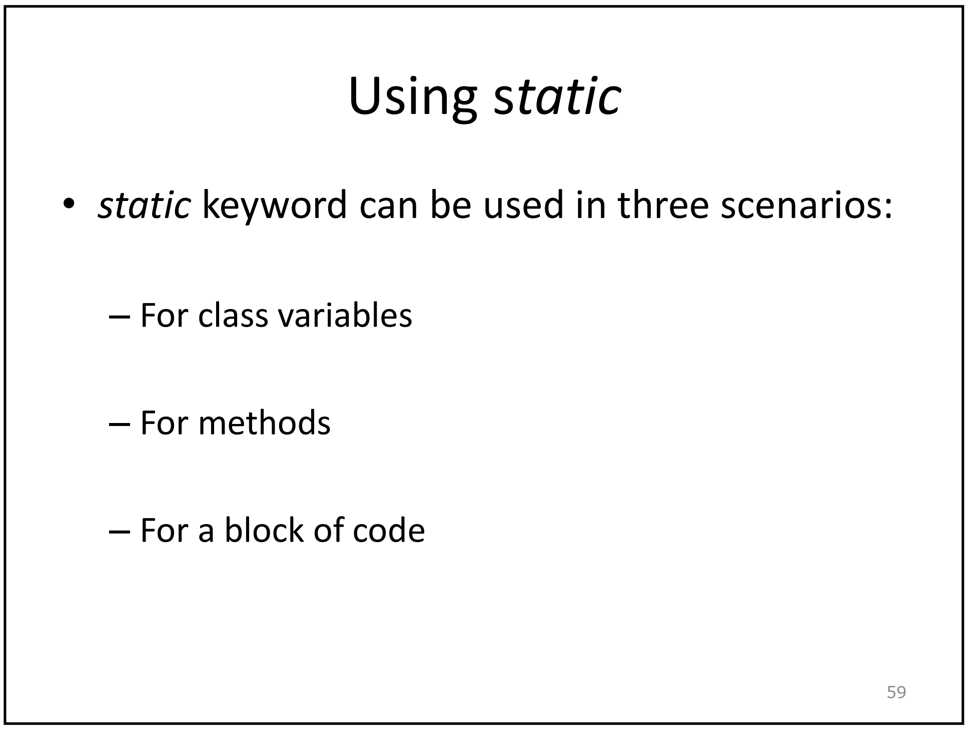
Command Line Arguments

- Information that follows program’s name on the command line when it is executed
- This data is passed to the application in the form of String arguments

```
class Echo {
public static void main (String args[]) {
for (int i = 0; i < args.length; i++)
System.out.println(args[i]);
 }
}
 Try this: Invoke the Echo application as
follows
C:\> java Echo Drink Hot Java
Drink
Hot
Java
```

58

58

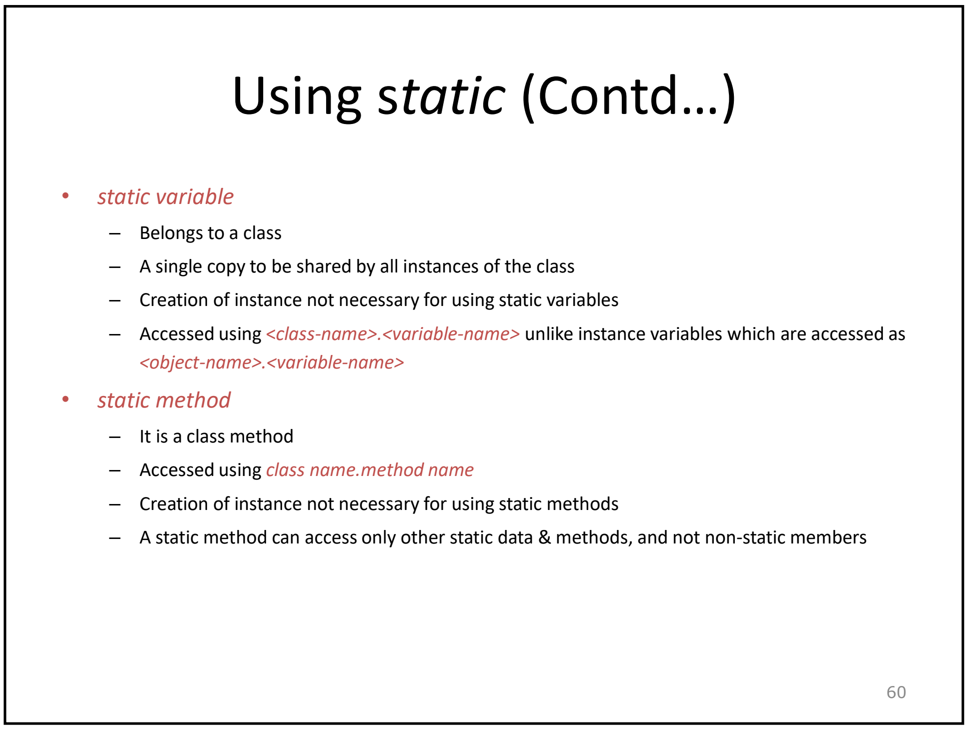


Using *static*

- *static* keyword can be used in three scenarios:
  - For class variables
  - For methods
  - For a block of code

59

59

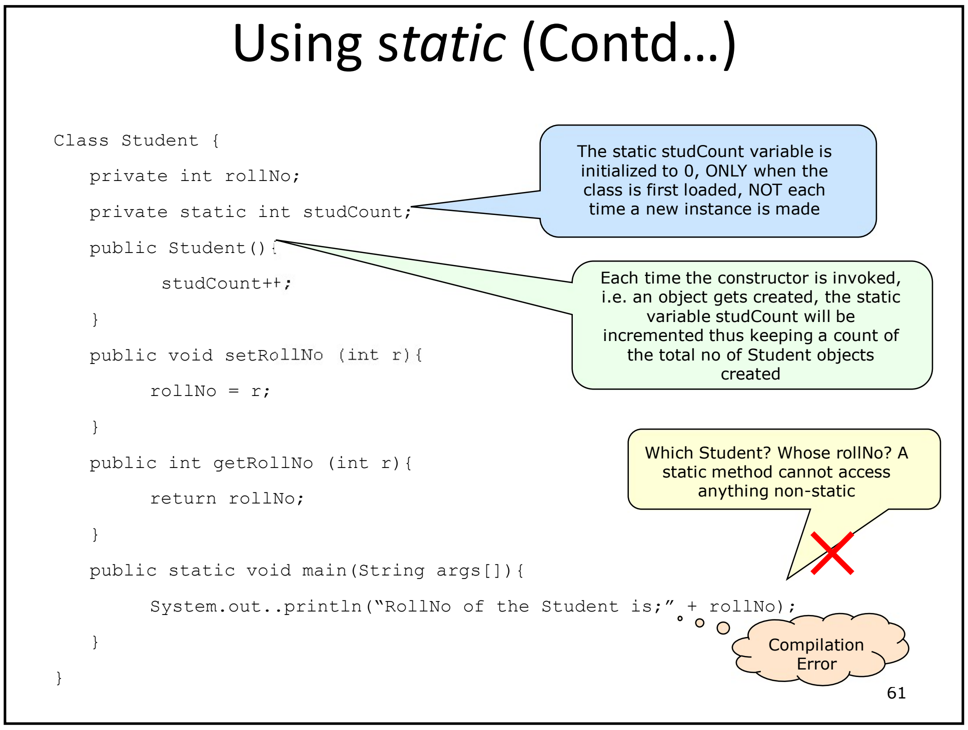


Using *static* (Contd…)

- *static variable*
  - Belongs to a class
  - A single copy to be shared by all instances of the class
  - Creation of instance not necessary for using static variables
  - Accessed using *<class-name>.<variable-name>* unlike instance variables which are accessed as *<object-name>.<variable-name>*
- *static method*
  - It is a class method
  - Accessed using *class name.method name*
  - Creation of instance not necessary for using static methods
  - A static method can access only other static data & methods, and not non-static members

60

60



Using *static* (Contd…)

```
Class Student {
   private int rollNo;
   private static int studCount;
   public Student(){
         studCount++;
   }
   public void setRollNo (int r){
        rollNo = r;
   }
   public int getRollNo (int r){
        return rollNo;
   }
   public static void main(String args[]){
        System.out..println("RollNo of the Student is;" + rollNo);
   }
}
```

The static studCount variable is initialized to 0, ONLY when the class is first loaded, NOT each time a new instance is made

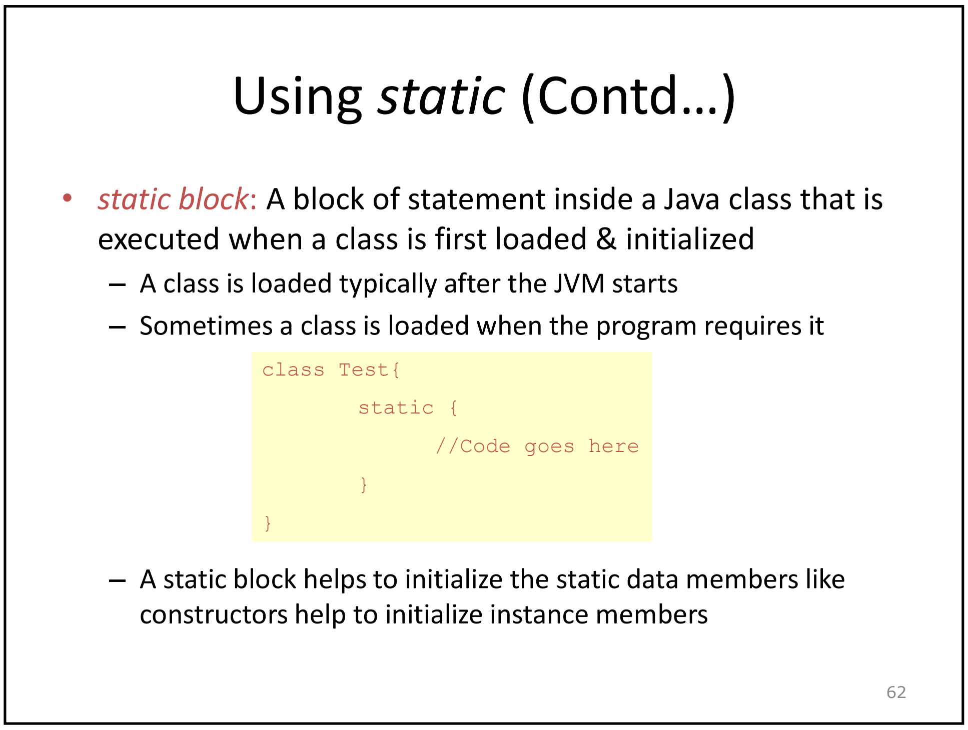
Each time the constructor is invoked, i.e. an object gets created, the static variable studCount will be incremented thus keeping a count of the total no of Student objects created

Which Student? Whose rollNo? A static method cannot access anything non-static

Compilation Error

61

61



Using *static* (Contd…)

- *static block*: A block of statement inside a Java class that is executed when a class is first loaded & initialized
  - A class is loaded typically after the JVM starts
  - Sometimes a class is loaded when the program requires it

```
class Test{
        static {
                //Code goes here
        }
}
```

  - A static block helps to initialize the static data members like constructors help to initialize instance members

62

62