Introduction to Java Programming

T.LOGESWARI

What is Java

- Java is a **programming language** and a **platform**.
- Java is a high level, robust, secured and object-oriented programming language.
- **Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.
-

Where it is used?

- According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:
- Desktop Applications such as acrobat reader, media player, antivirus etc.
- Web Applications such as irctc.co.in, javatpoint.com etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games etc.

Java Applications

- We can develop two types of Java programs:
  - Stand-alone applications
  - Web applications (applets)
  - Enterprise Application
  - Mobile Application

Types of Java Applications

1) Standalone Application

- It is also known as desktop application or window-based application.
- An application that we need to install on every machine such as media player, antivirus etc.
- AWT and Swing are used in java for creating standalone applications.

2) Web Application

- An application that runs on the server side and creates dynamic page, is called web application.
- Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

- An application that is distributed in nature, such as banking applications etc.
- It has the advantage of high level security, load balancing and clustering.
- In java, EJB is used for creating enterprise applications.

4) Mobile Application

- An application that is created for mobile devices.
- Currently Android and Java ME are used for creating mobile applications.

Applets v/s Applications

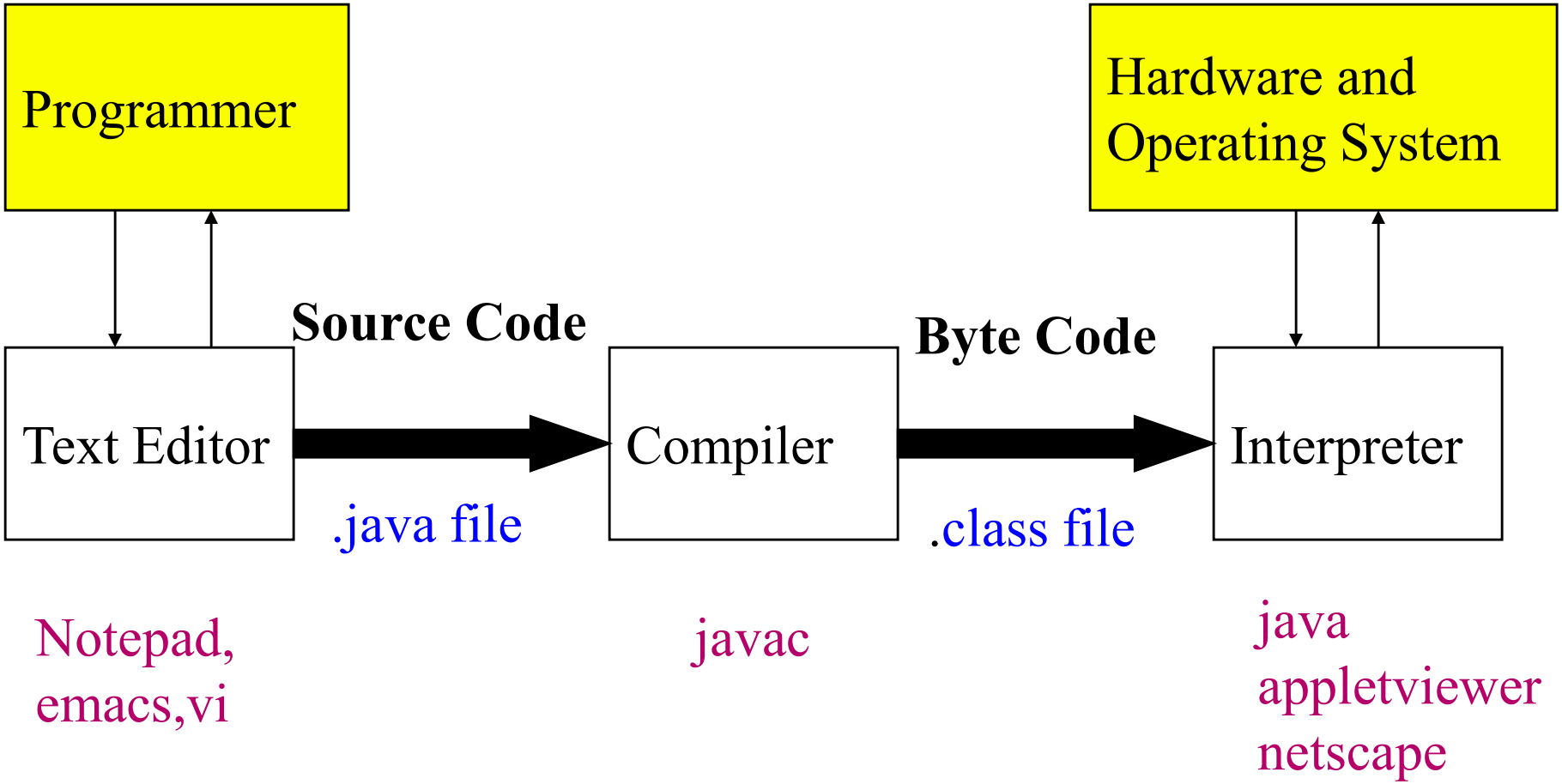
- Different ways to run a Java executable are

  Application- A stand-alone program that can be invoked from command line . A program that has a "main" method

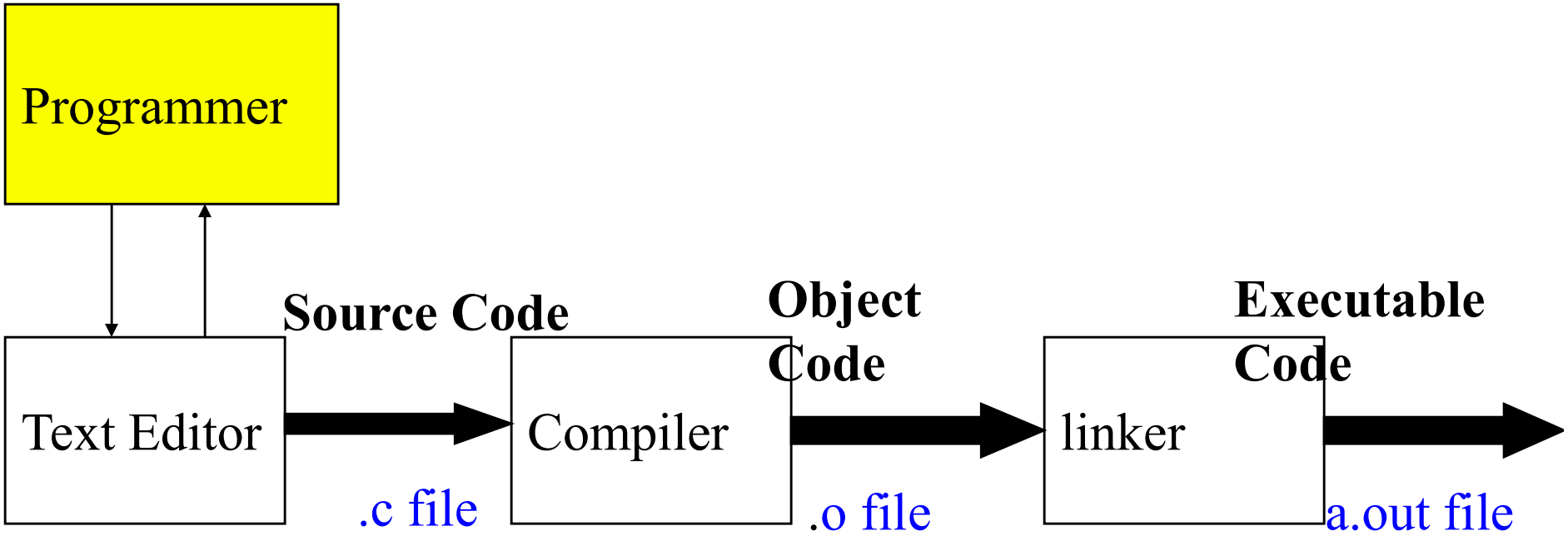
  Applet- A program embedded in a web page , to be run when the page is browsed . A program that contains no "main" method

- Application –Executed by the Java interpreter.
- Applet- Java enabled web browser.

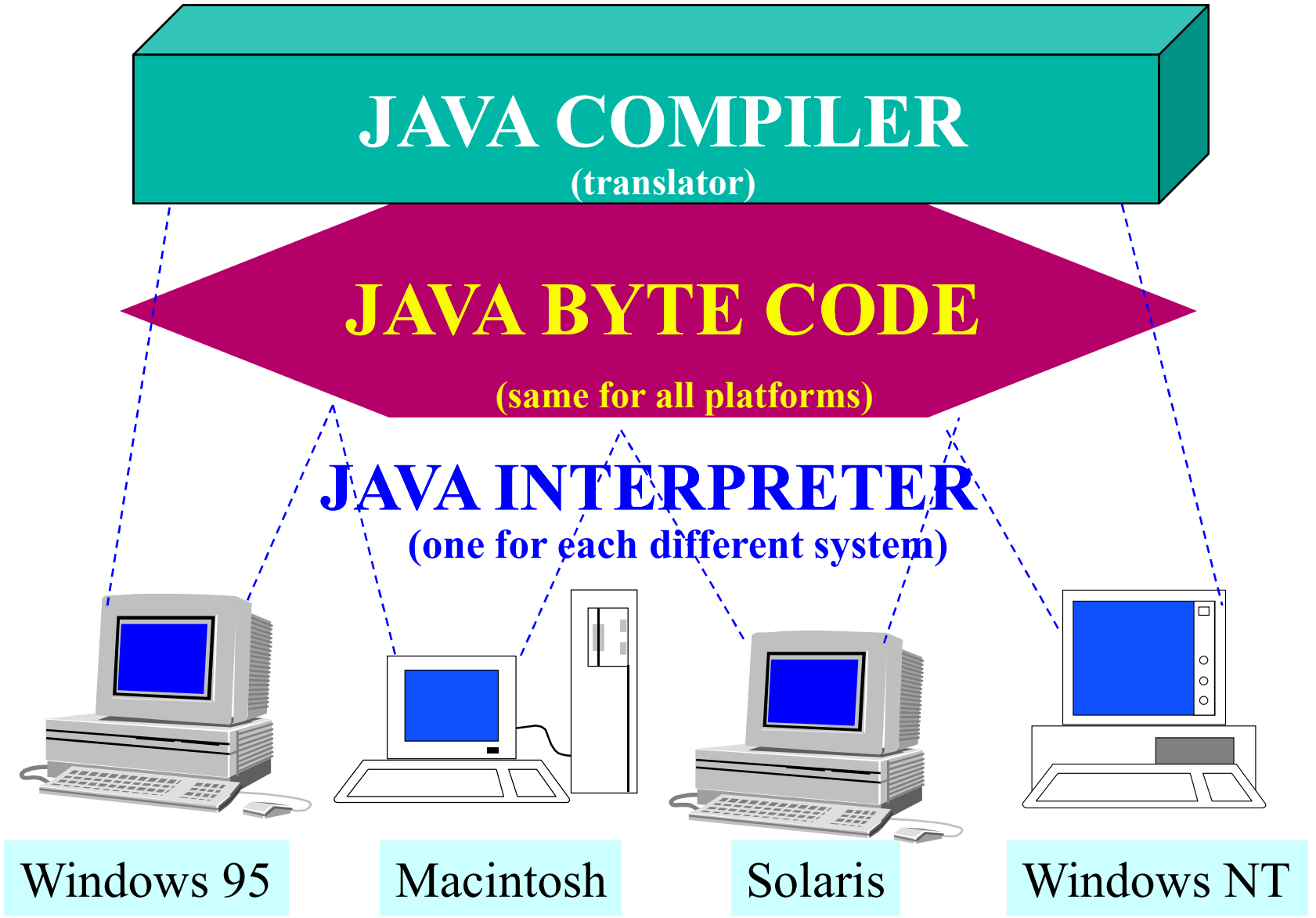
Java is Compiled and Interpreted



Compiled Languages



Total Platform Independence



Architecture Neutral & Portable

- Java Compiler - Java *source code* (file with extension .java) to *bytecode (*file with extension .class*)*
- *Bytecode* - an intermediate form, closer to machine representation
- A interpreter (virtual machine) on any target platform interprets the bytecode.

Getting Started with Java Programming

- A Simple Java Application
- Compiling Programs
- Executing Applications

A Simple Application

Example 1.1

```
//This application program prints Welcome
//to Java!
package chapter1;

public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

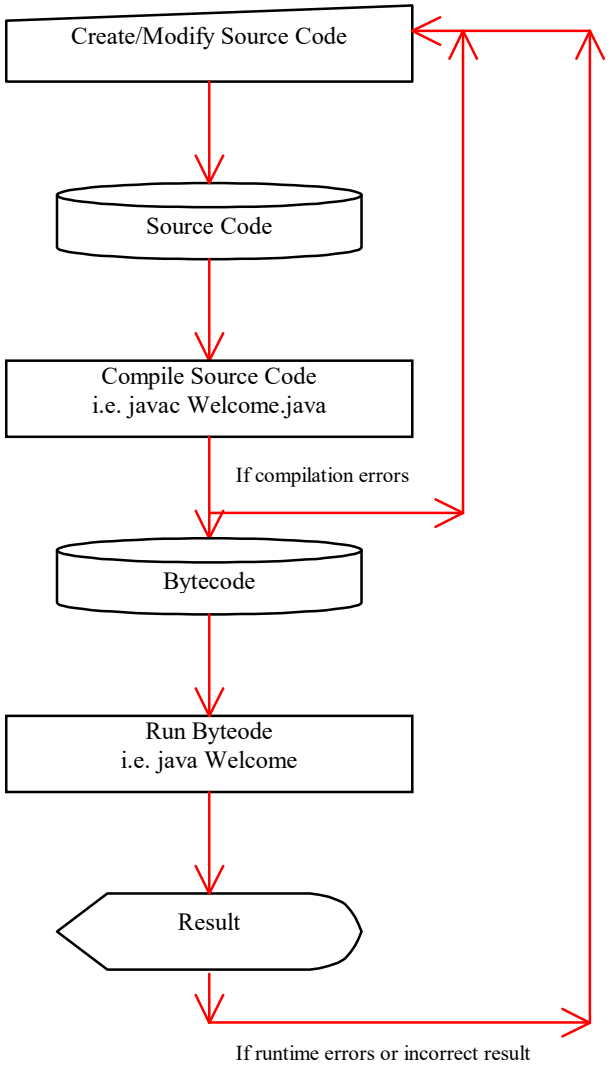
Source

Run

NOTE: To run the program, install slide files on hard disk.

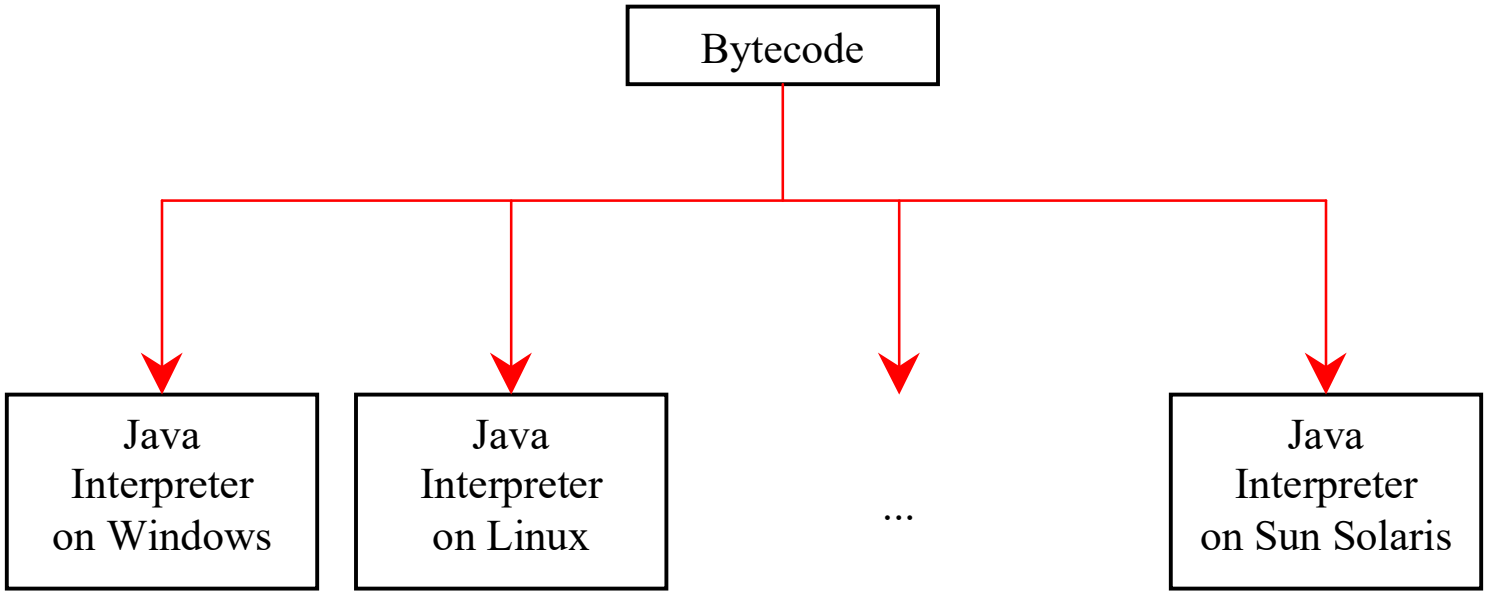
Creating and Compiling Programs

- On command line
  - `javac file.java`



Executing Applications

- On command line
  - `java classname`



Example

```
javac Welcome.java

java Welcome

output:...
```

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

•In Java, comments are preceded by two slashes (//) in a line, or enclosed between /* and */ in one or multiple lines.

•When the compiler sees //, it ignores all text after // in the same line.

• When it sees /*, it scans for the next */ and ignores any text between /* and */.

Package

•The second line in the program (package chapter1;) specifies a package name, chapter1, for the class Welcome.

• Forte compiles the source code in Welcome.java, generates Welcome.class, and stores Welcome.class in the chapter1 folder.

Reserved Words

•Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program.

• For example, when the compiler sees the word class, it understands that the word after class is the name for the class.

•Other reserved words in Example 1.1 are public, static, and void.

Modifiers

•Java uses certain reserved words called modifiers that specify the properties of the data, methods, and classes and how they can be used.

• Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected.

•A public datum, method, or class can be accessed by other programs.

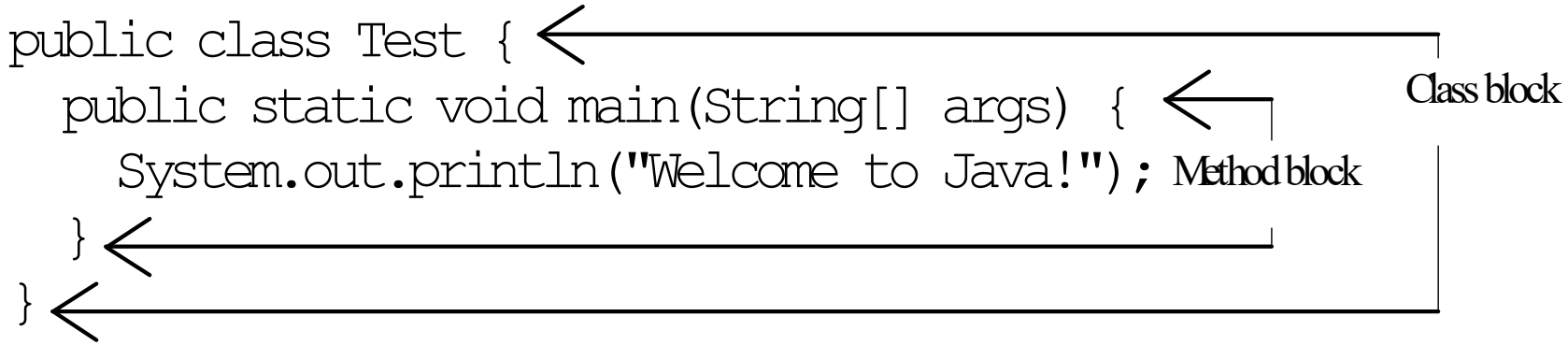
• A private datum or method cannot be accessed by other programs.

Statements

- A statement represents an action or a sequence of actions.
- The statement System.out.println("Welcome to Java!") in the program in Example 1.1 is a statement to display the greeting "Welcome to Java!" Every statement in Java ends with a semicolon (;).

Blocks

•A pair of braces in a program forms a block that groups components of a program.



Classes

•The class is the essential Java construct.

•A class is a template or blueprint for objects.

• To program in Java, you must understand classes and be able to write and use them.

•For now, though, understand that a program is defined by using one or more classes.

Methods

What is System.out.println?

•It is a method: a collection of statements that performs a sequence of operations to display a message on the console.

• It can be used even without fully understanding the details of how it works.

•It is used by invoking a statement with a string argument.

•The string argument is enclosed within parentheses. In this case, the argument is "Welcome to Java!" You can call the same println method with a different argument to print a different message.

main Method

•The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

•The main method looks like this:

```
public static void main(String[] args) {
// Statements;
}
```

Program Processing

- Compilation

  `# javac hello.java`

  results in HelloInternet.class

- Execution

  `# java HelloInternet`

  Hello Internet

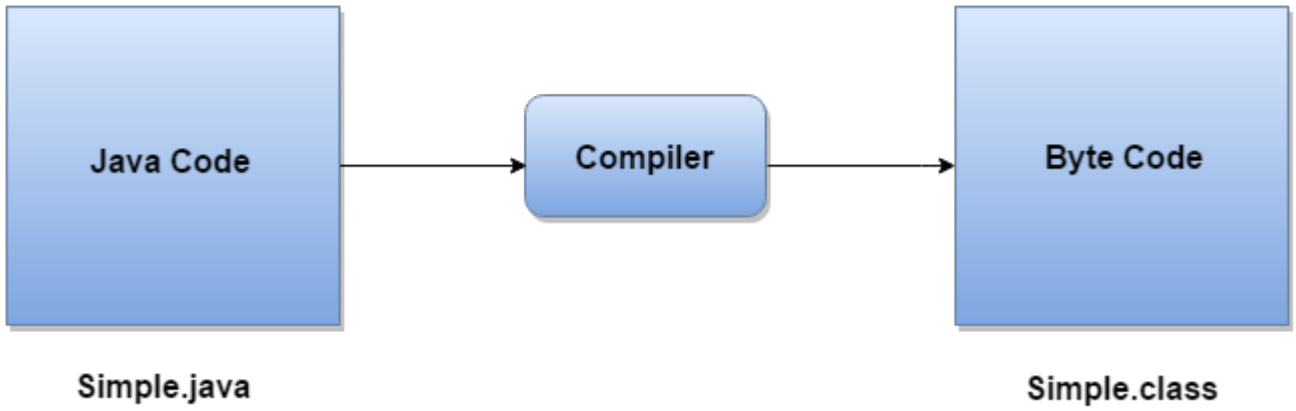
  #

Summary

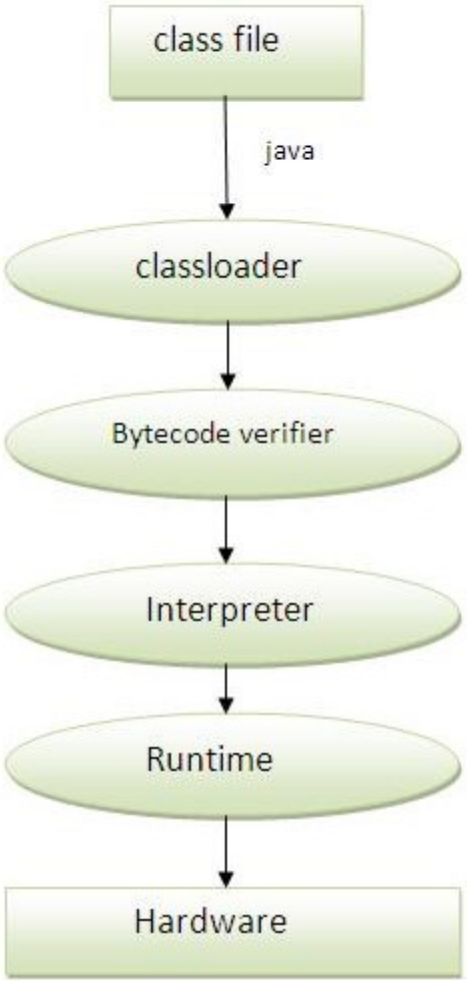
- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method.
  - The core advantage of static method is that there is no need to create object to invoke the static method.
  - The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.

- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used print statement.

- What happens at compile time?
- At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



- What happens at runtime?
- At runtime, following steps are performed:



Classloader: is the subsystem of JVM that is used to load class files.

Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

Interpreter: read bytecode stream then execute the instructions.

INTRODUCTION

Dr.T.Logewari

- **JVM** - JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.
- **JRE** - JRE (Java Runtime Environment) contains JVM, supporting libraries, and other components to run a Java program. However, it doesn't contain any compiler and debugger.
- **JDK** - JDK (Java Development Kit) contains JRE and tools such as compilers and debuggers for developing Java applications.

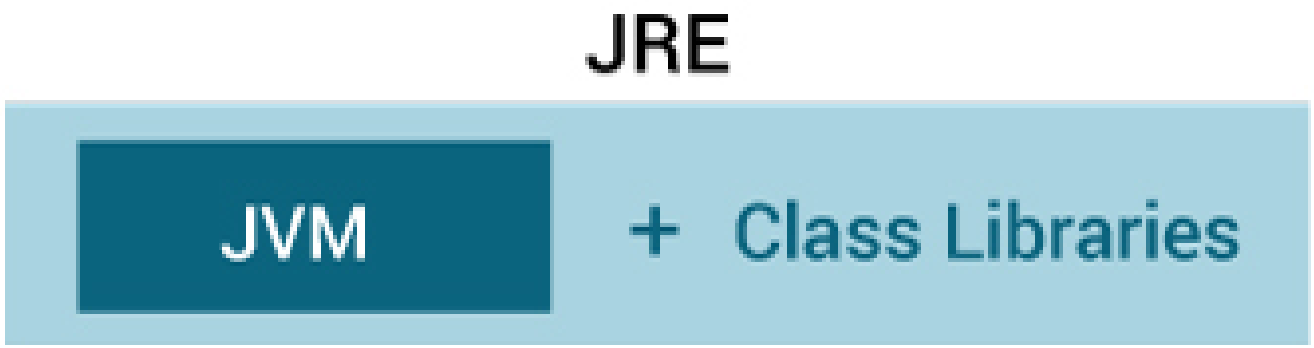
What is JVM?

- JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.
- When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

- Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer).
- Since, JVM executes the Java bytecode which is platform independent, Java is platform-independent.

What is JRE?

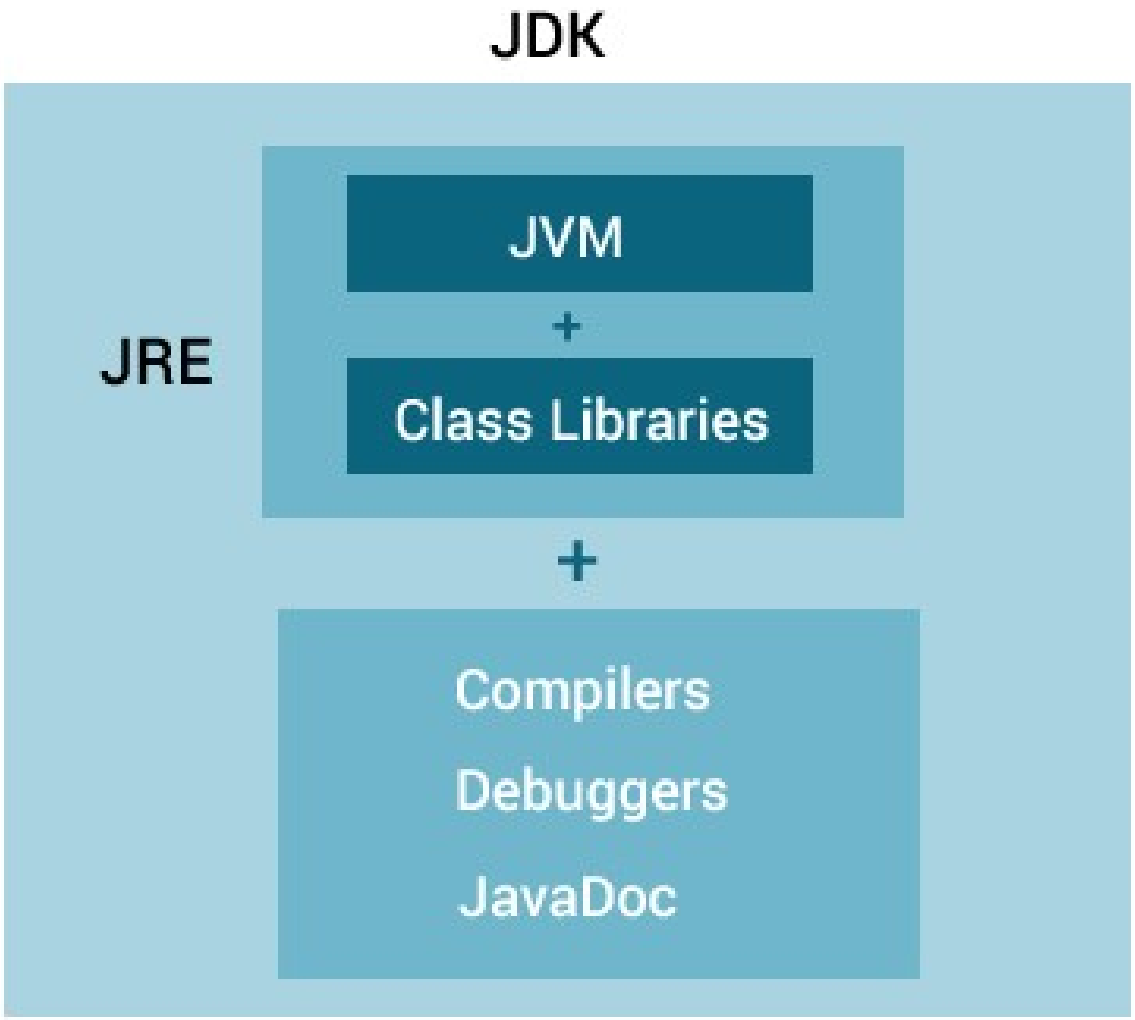
- JRE (Java Realtime Environment) is a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications written in Java programming.
- JRE is the superset of JVM.



What is JDK?

- JDK (Java Development Kit) is a software development kit to develop applications in Java.
- When you download JDK, JRE is also downloaded, and don't need to download it separately.
- In addition to JRE, JDK also contains number of development tools (compilers, JavaDoc, Java Debugger etc).
-

Relationship between JVM, JRE, and JDK.



Command Line Arguments

- A java application can accept any number of arguments from the command line
- The user enter command line argument when invoking the application and specifies them after the name of the class to be run

Why they used

- To turn on debugging input, to indicate a file name to read or write from, or for any other information that might want our java program to know.
- Java application are stand alone program, it is useful to be able to pass argument or option to that program to determine how the program is going to run
- In this case we pass argument by using command line arguments

Find Factorial of Given number using CLA

```
Class factorial {
Pubic static void main(String args[]){
Int num = Integer.parseInt(args[0]);
//take argument as command line
Int result = 1
While(num>o){
Result = result *num;
Num--;
}
```

```
System.out.println("factorial of given
                                    number:"+result);
}
}
```

Output

```
Java factorial 4
Factorial of given number is : 24
```

Java Variables

- A variable is a location in memory (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier).

How to declare variables in Java?

```
int speedLimit = 80;
```

Here, speedLimit is a variable of int data type, and is assigned value 80.

Meaning, the speedLimit variable can store integer values.

- Java is a statically-typed language. It means that all variables must be declared before they can be used.

Rules for Naming Variables in Java

- Java programming language has its own set of rules and conventions for naming variables.
- Variables in Java are case-sensitive.
- A variable's name is a sequence of Unicode letters and digits. It can begin with a letter, $ or _.
- However, it's convention to begin a variable name with a letter. Also, variable name cannot use whitespace in Java.

Variable Name	Remarks
speed	Valid variable name
_speed	Valid but bad variable name
$speed	Valid but bad variable name
speed1	Valid variable name
spe ed	Invalid variable name
spe”ed	Invalid variable name

4 types of variables in Java programming language:

- Instance Variables (Non-Static Fields)
- Class Variables (Static Fields)
- Local Variables
- Parameters

Java Primitive Data Types

Java Keywords

- Keywords are predefined, reserved words used in Java programming that have special meanings to the compiler. For example:

```
int score;
```

- Here, int is a keyword. It indicates that the variable score is of integer type (32-bit signed two's complement integer).
- You cannot use keywords like int, for, class etc as variable name (or identifiers) as they are part of the Java programming language syntax.

Here's the complete list of all keywords in Java programming.

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

Java identifiers

- Identifiers are the name given to variables, classes, methods etc.
- Consider the above code;

```
int score;
```

- Here, score is a variable (an identifier). You cannot use keywords as variable name. It's because keywords have predefined meaning. For example,

```
int float;
```

Rules for Naming an Identifier

- Identifier cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter, $ or _. The first letter of an identifier cannot be a digit.
- It's convention to start an identifier with a letter rather and $ or _.
- Whitespaces are not allowed.
- Similarly, you cannot use symbols such as @, #, and so on.

Operators

- Operators are special symbols (characters) that carry out operations on operands (variables and values).
- For example, + is an operator that performs addition.

Arithmetic Operators

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning
+	Addition (also used for string concatenation)
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Remainder Operator

```
class ArithmeticOperator
{
public static void main(String[] args)
 {
String start, middle, end, result;
start = "Talk is cheap. ";
 middle = "Show me the code. ";
end = "- Linus Torvalds";
result = start + middle + end;
   System.out.println(result);
}
}
```

ASSIGNMENT OPERATORS

The *assignment operator* is the single equal sign, =. This behaviour of assignment operator is similar to other programming languages like C and C++.

The general syntax is:

```
var = expression;
```

- Examples of the assignment operator

```
int x=10;
int y,z;
y=10+20-5;
z=y-10;
```

- The assignment operator allows creating a chain of assignments. For example, consider the below code:

```
int x, y, z;
x = y = z = 100; // set x, y, and z to 100
```

Assignment Operator

- Assignment operators are used in Java to assign values to variables. For example,

  int age; age = 5;

- The assignment operator assigns the value on its right to the variable on its left. Here, 5 is assigned to the variable age using = operator.

```
Class AssignmentOperator
 {
 public static void main(String[] args)
{
 int number1, number2;
// Assigning 5 to number1
 number1 = 5;
System.out.println(number1);
// Assigning value of variable number2 to number1
 number2 = number1;
   System.out.println(number2);
} }
```

Output

5

5

The shorthand assignment operator

- The operator perform shortcut in common programming operation
- It is also called compound assignment operator
- Syntax

var1 operator = var2

Java Assignment Operators

operator	Example	Equivalent to
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x / 5
<<=	x <<= 5	x = x << 5
>>=	x >>= 5	x = x >> 5
&=	x &= 5	x = x & 5
^=	x ^= 5	x = x ^ 5

Unary Operators

- Unary operator performs operation on only one operand.

operator	Meaning
+	Unary plus (not necessary to use since numbers are positive without using it)
-	Unary minus; inverts the sign of an expression
++	Increment operator; increments value by 1
--	decrement operator; decrements value by 1
!	Logical complement operator; inverts the value of a boolean

UNARY OPERATORS

Operator	Description	Example
+	The "+" operator refers to the positive value	int a=+10; int b= +a;
-	The "-" operator refers to the negative value	int a=-10; int b=-a;

++	Increment Operator - to increase the value of operand by 1	int a=10;
		a++;
		++a;

–	Decrement Operator - to decrease the value of operand by 1	int a=10; a–; --a;

Unary plus

```
class UnaryOperator {
public static void main(String[] args)
 { double number = 5.2;
   System.out.println(number++);
   System.out.println(number);
   System.out.println(++number);
   System.out.println(number); } }
```

Output

```
5.2
 6.2
 7.2
 7.2
```

- When System.out.println(number++); statement is executed, the original value is evaluated first.
- The number is increased only after that. That's why you are getting 5.2 as an output.
- Then, when System.out.println(number); is executed, the increased value 6.2 is displayed.
- However, when System.out.println(++number); is executed, number is increased by 1 first before it's printed on the screen.

Equality and Relational Operators

- The equality and relational operators determines the relationship between two operands.
- It checks if an operand is greater than, less than, equal to, not equal to and so on.
- Depending on the relationship, it results to either true or false.
- Equality and relational operators are used in decision making and loops

Operator	Description	Example
==	equal to	5 == 3 is evaluated to false
!=	not equal to	5 != 3 is evaluated to true
>	greater than	5 > 3 is evaluated to true
<	less than	5 < 3 is evaluated to false
>=	greater than or equal to	5 >= 5 is evaluated to true
<=	less then or equal to	5 <= 5 is evaluated to true

```
class RelationalOperator
 { public static void main(String[] args)
{
 int number1 = 5, number2 = 6;
 if (number1 > number2) {
    System.out.println("number1 is greater than
    number2.");
} else
 { System.out.println("number2 is greater than
    number1.");
 } } }
```

Output
number2 is greater than number1.

type comparison operator

- In addition to relational operators, there is also a type comparison operator **instanceof** which compares an **object to a specified type**. For example,

INSTANCE OF OPERATOR

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side then the result will be true.

Following is the example:

```
String name = "Srikanth";
boolean result = name instanceof String;
// This will return true since name is type of String
```

In the above example, we have created string object and reference variable is name. String is a class in Java. If the reference variable is of type String, then result is true.

```
class instanceofOperator
{
 public static void main(String[] args)
 { String test = "asdf";
boolean result;
result = test instanceof String;
   System.out.println(result);
 } }
```

Logical Operators

- The logical operators || (conditional-OR) and && (conditional-AND) operates on boolean expressions.
- Here's how they work.

Operator	Description	Example
||	conditional-OR; true if either of the boolean expression is true	false || true is evaluated to true
&&	conditional-AND; true if all boolean expressions are true	false && true is evaluated to false

```
class LogicalOperator
{ public static void main(String[] args)
 { int number1 = 1, number2 = 2, number3 = 9;
   boolean result;
// At least one expression needs to be true for
   result to be true
 result = (number1 > number2) || (number3 >
   number1);
// result will be true because (number1 >
   number2) is true
 System.out.println(result);
```

```
// All expression must be true from result to be
   true
 result = (number1 > number2) && (number3 >
   number1);
// result will be false because (number3 >
   number1) is false
System.out.println(result);
} }
```

Output
true
false

- Which operator are called short circuit logical operator?

  && and || are short circuit operator

- What type of values can be used as operands of the logical operator?

  The logical operator must have operands of type boolean

What is the difference between normal logical operators (& and |) and short circuit logical operators (&& and ||)?

- Using &&, if the left side of the expression is false, the entire expression is assumed to be false (the value of the right side doesn't matter), so the expression returns false, and the right side of the expression is never evaluated.
- Using &, both sides of the expression are evaluated.
- Using || if the left side of the expression is true, the entire expression is assumed to be true (the value of the right side doesn't matter), so the expression returns true, and the right side of the expression is never evaluated.
- Using |, both sides of the expression are evaluated.

Ternary Operator

- The conditional operator or ternary operator ?: is shorthand for if-then-else statement.
- The syntax of conditional operator is:

variable = Expression ? expression1 : expression2

Here's how it works.

- If the Expression is true, expression1 is assigned to variable.
- If the Expression 1is false, expression2 is assigned to variable.

```
class ConditionalOperator
{ public static void main(String[] args)
 { int februaryDays = 29;
String result;
result = (februaryDays == 28) ? "Not a leap year"
   : "Leap year";
System.out.println(result);
 } }
```

Output

Leap year

Bitwise and Bit Shift Operators

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR

Additional Operator

- Semicolon :
- Curly bracket{}
- Parentheses()
- Square bracket[]
- Comma ,
- Single quote’
- Double quote’’

DOT OPERATOR

The dot operator (.) is used to access the instance variables and methods of class using an object.

Example:

```
dog.age
dog.bark();
```

NEW OPERATOR

The new operator is used to create objects, that is, instances of classes and arrays.

Example: Animal dog=new Animal();

Java Expressions

- Expressions consist of variables, operators, literals and method calls that evaluates to a single value.
- let's take an example,
- int score; score = 90;Here, score = 90 is an expression that returns int.

Double a = 2.2, b = 3.4, result;

result = a + b - 3.4;

Here, a + b - 3.4 is an expression.

```
if (number1 == number2)
    System.out.println("Number 1 is larger than
    number 2");
```

Here, number1 == number2 is an expression that returns Boolean.

Similarly, "Number 1 is larger than number 2" is a string expression.

Precedence of Arithmetic Operator

```
int myInt = 12 - 4 * 2;
```

What will be the value of myInt? Will it be (12 - 4)*2, that is, 16? Or it will be 12 - (4 * 2), that is, 4?

- When two operators share a common operand, 4 in this case, the operator with the highest precedence is operated first.
- In Java, the precedence of * is higher than that of -. Hence, the multiplication is performed before subtraction, and the value of myInt will be 4.

Operators	Precedence
postfix increment and decrement	++ --
prefix increment and decrement, and unary	++ -- + - ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=

bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>=>>>=

Example: Operator Precedence

```
class Precedence
 { public static void main(String[] args)
 { int a = 10, b = 5, c = 1, result;
 result = a-++c-++b;
System.out.println(result);
} }
```

output will be:
2

- The operator precedence of prefix ++ is higher than that of - subtraction operator.
- Hence,
- result = a-++c-++b;is equivalent to

```
result = a-(++c)-(++b);
```

Associativity of Operators in Java

- If an expression has two operators with similar precedence, the expression is evaluated according to its associativity (either left to right, or right to left). Let's take and example.

```
a = b = c;
```

Here, the value of c is assigned to variable b. Then the value of b is assigned of variable a. Why? It's because the associativity of = operator is from right to left.

Operators	Precedence	Associativity
postfix increment and decrement	++ --	left to right
prefix increment and decrement, and unary	++ -- + - ~ !	right to left
multiplicative	* / %	left to right
additive	+ -	left to right
shift	<< >> >>>	left to right
relational	< > <= >= instance of	left to right

equality	== !=	left to right
bitwise AND	&	left to right
bitwise exclusive OR	^	left to right
bitwise inclusive OR	|	left to right
logical AND	&&	left to right
logical OR	||	left to right
ternary	? :	right to left
assignment	= += - = *= /= %= &= ^=	left to right

MATHEMATICAL FUNCTIONS

To make the programmers life easier, the Math class provides a number useful methods. All Math methods are called by this syntax: Math.method(parameters).

Note:

- The Math class is part of java.lang package.
- Since it is in the java.lang package, the Math class need not be imported. The lang package is default for all the programs.

Method	Description	Example
abs(x)	absolute value of x	abs(23.7) is 23.7abs(0.0) is 0.0abs(23.7) is 23.7
ceil(x)	rounds x to the smallest integer not less than x	ceil(9.2) is 10.0ceil(-9.8) is -9.0
cos(x)	trigonometric cosine of x (x in radians)	cos(0.0) is 1.0
exp(x)	exponential method	exp(1.0) is 2.71828exp(2.0) is 7.38906
floor(x)	rounds x to the largest integer not greater than x	floor(9.2) is 9.0floor(-9.8) is -10.0
log(x)	natural logarithm of x (base e)	log(Math.E) is 1.0log(Math.E * Math.E) is 2.0
max(x, y)	larger value of x and y	max(2.3, 12.7) is 12.7max(-2.3, -12.7) is -2.3
min(x, y)	smaller value of x and y	min(2.3, 12.7) is 2.3min(-2.3, -12.7) is -12.7
pow(x, y)	x raised to the power y (i.e., x y)	pow(2.0, 7.0) is 128.0pow(9.0, 0.5) is 3.0
sin(x)	trigonometric sine of x (x in radians)	sin(0.0) is 0.0
sqrt(x)	square root of x	sqrt(900.0) is 30.0
tan(x)	trigonometric tangent of x (x in radians)	tan(0.0) is 0.0

Classes and Object

Dr.T.Logeswari

Class

- All java programs activity occurs within a class
- A class is a template that define the form of an object.
- It specifies both data and code that will operate on that data
- Java uses a class to construct object
- Object are instances of a class

Class Definition

- A class contain data member and methods
- Data member(instance variable)
  - These variables that store data items .they are also referred to as fields or member variables of a class
- Methods
  - These define the operation you can perform for the class

Definition of class and object

- A class is a template that define the form of an object
- A class is a generic template for creating object
- Class = data + methods
- A object is an instance of a class

Defining a class

- A class is created by using the keyword class

[access specifier] [class modifier] class Class Name[extend Super ClassName][implements interfaceList]

{

[variable declaration]

[method declaration]

}

- The element between the pair of square bracket[] are optional
- The access modifier specifies who can access this class
- The class modifier specify the behavioral restriction on this class
- Class then followed by class name
- The class may or may not contain variable declaration or method declaration

Adding variable to class

- We can declare the variable inside the class
- The variable inside the class are of two types
  - The class variable and instance variable
- The class variable will always have the modifier **static** in front of them
- Example

Static int age = 30;

- The same variable defined instance variable would be

Example

Int age =30;

Adding method to class

- The methods are function that manipulate the data defined by the class and in many cases provide access to that data
- The other part of the program will interact with a class through its methods
- The method will perform some task
  - A method contains one or more statement. In well written java code each method perform only one task

- In general we can give a method whatever name we like. However the main() is reserved for entry point for program execution
- We should not use java keyword for method names
- The general form

[modifier]return type method-name(parameter-list)

{

//body of method

}

- Return type – it specifies the type of data returned by the method (any valid data type)
  - If does not return a value then return type must be void
- Method – the name of the method is specified by name. this can be legal identifier
- Parameter-it is variable that receive the value of the argument passed to the method when it is called
- Modifier- it is a list of method modifier that declare various attributes of method

Creating Object

- A object is created by instantiating a class
- The process of creating an object of a class is called as instantiation and created object is called as an instance
- To create a new object java uses the keyword new
- The object are created using new operator with the name of the class

- The general form

<class name><reference-variable>= new<class name>([arguments])

Class name – the name of the class

Reference variable – it can refer to an object

New – operator to create an object

Argument - optional

Accessing Class Member

- We can access the member of the class using dot operator
- The dot operator links the name of the object with the name of a member
- The general form

  to access variable = object.variable

  to access methods = object.methods()

Constructor

- New object is created, the garbage value will be stored in variable initially.
- Accessing these value leads some unwanted result
- To avoid we use member function such as getdata() and setdata() to provide initial value of object.
- But the initialization can be done only after creating the object

- Therefore the mechanism to initialize an object during its creation using special member function known as constructor

Constructor

- If we want to set the default values for instance variables at the time of creation of an object, then we should use constructor
- A constructor initialize an object when it is created.
- It has the same name as its class name
- It have no explicit return type
- We will use constructor, to give initial values to the instance variable defined by the class

- A class have constructor, whether we define one or not, java automatically provides default constructor that initialize all member variable to zero
- The constructor are of two types
- Constructor with no argument(default argument)

  A(){}
- Constructor with arguments(parameterized constructor

  A(int a){}

- The constructor can be overloaded.
- Example

A(int a) { } A(byte b){ } A(long l){}

The constructor are having the same name and same number of argument but different type of argument

Default Constructor	Parameterized constructor
The default constructor is useful to initialize all object with same data	Parameterized Constructor is useful to initialize each object with different data
It does not have any argument	It will have 1 or more argument
When data is not passed at the time of creating an object, default constructor will be called	When data is not passed at the time of creating an object, parameterized constructor will be called

Constructor	Methods
The constructor is used to initialize the instance variable of class	A method is used for any general purpose task like calculation
A Constructor name should always be same as class name	A method name and class name is same or different
A constructor is called at the time of creating an object	A method can be called after creating the object
A constructor is called only once per object	A method can be called any number of times on the object
A constructor is called and executer automatically	A method is executed only when we want it

Using this Keyword

- **this** refers to the current object
- Whenever it is required to point an object from a functionality which is under execution then use **this** keyword
- It always points to an object that is executing the block in which **this** keyword is present

- The use of **this** is to call constructor from another constructor, specifically one in the current class
- The process of calling constructor from other constructor is called constructor chaining
- The constructor call should be the first statement in the constructor (example)
- The second function of **this** is to avoid namespace conflict between a methods or constructor parameter list and its variable

Method Overloading

- In java, two or more methods within the same class can have the same name but with different number of arguments and type of arguments. The method are said to be **overloaded** and the process is referred to as **method overloading**

Method overloading is one of the ways that java implements **polymorphism**

- Each method has a signature (method, number of arguments and type of arguments)
- The method overloading is not based on the return type
- **One method to overload another, the type or number of argument must be different**
- The below method are overloaded

  public void aMethod(string s){}

  public void amethod(){}

  public void amethod(int i, string s){}

  public void amethod(inti , int j){}

All the above method are unique within one class because each of them has different signature

- **What is polymorphism**?

  Defining more than one functionality with the same name is nothing but a polymorphism

- **What are the types of polymorphism?**

  two types – compile time(static)

  -- Run time(dynamic)

- **What is compile time or static polymorphism**?

  Defining more than one functionality with the same name but with different arguments in the same class is known as static polymorphism

- **How static polymorphism is achieved in java?**

  It is achieved using method overloading

Static variables and methods

- A class member must be accessed through an object of its class
- But it is possible without creating an object create a class member
- The keyword is **static**
- Keyword can be used in three scenarios
  - Static variables
  - Static methods
  - Static block of code

Static variables

- The instance variable are non static and it is part of an object
- The static variables are special type of variables that are not associated with an object, they associated with class
- The static variables are also called as class variables
- It can be accessed without an object

Declaring static variables

```
class staticDemo
{
        int x,y;
        Static int z;
}
```

You can directly accessed by the class name and doesnot need any object

Syntax

<class-name>.<variable-name>

Static methods

- The methods can be declared as static
- A static method is associated with a class rather than the instances
- The static methods are also called as class members
- The most common example of a static member is main()
- The main() is declared as static because it must be called by the operating system when our program begin

Declaring static methods

```
   class StaticDemo
{
   int x,y;
Static int z;
Void static method1()
{
System.out.println(z);
}
}
```

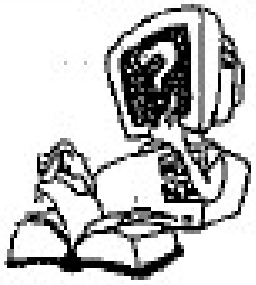
Static and non static blocks

- In real time scenarios block are used to provide information regarding the project(ie version , copy right, name of the company developed the project) this is the use of static block
- If we have many constructor in a class and if every constructor has some common statement. Then instead of repeating those statement in each constructor, we place in non static block(avoid duplication of code)

INHERITANCE

INHERITANCE

- Java classes can be reused in several ways. Reusing class can be accomplished by inheritance.
- The mechanism of deriving a new class from an old one is called **inheritance**.
- The already existing class from which a new class is created is known as the base class or **super class** or **parent class** and the newly created class is called the **subclass** or **derived class** or **child class.**



What is an inheritance? What is the use of it?

Inheritance can be defined as the process of acquiring properties of one object from other object. The important use of an inheritance is a code reusability and achieving polymorphism.

- In the language of java, a class that is inherited is called a super class
- The class that does the inheriting is called subclass
- A subclass is a specialized version of a super class
- It inherits all of the variable and method defined by the super class and add its own, unique variable and methods

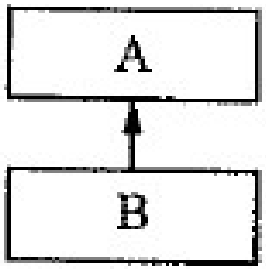
TYPES OF INHERITANCE

There are different types of inheritance like:

- Single inheritance (only one super class)
- Multiple inheritance (several super classes)
- Hierarchical inheritance (one super class, many subclasses)
- Multilevel inheritance (Derived from derived class)

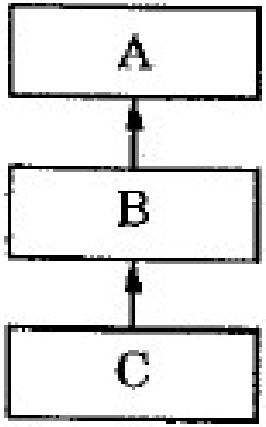
- **Single Inheritance:** The one class extends from another class

Example:



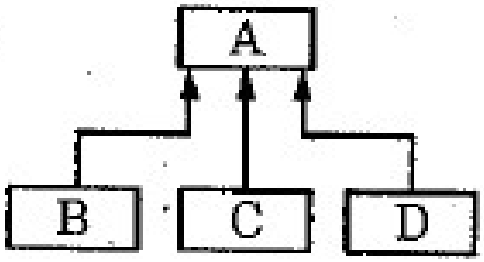
- **Multilevel Inheritance:** one class extending another class as in a hierarchical structure is termed as multilevel inheritance.

Example :



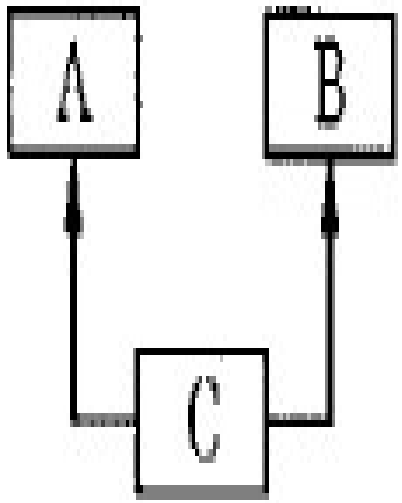
- **Hierarchical Inheritance:** Many classes extending from single super class. One super class and many sub-classes.

Example :



- **Multiple Inheritance**: The one class extends from more than one class.

Example :



The Java supports only single, multilevel, and hierarchical inheritance. It does not support multiple inheritance. The multiple inheritance will be achieved in different way by using Interfaces.

Single Inheritance	class A class B **extends** A
Multilevel Inheritance	class A class B **extends** A class C **extends** B
Hierarchical Inheritance	class A class B **extends** A class C **extends** A class D **extends** A
Multiple Inheritance	class A **extends** B, C // Not allowed

Which keyword is used for inheritance?

Extends keyword will be used to create subclasses.

Which inheritance is not supported in Java?

Multiple Inheritance.

Defining SubClass

- Java support inheritance by allowing one class to incorporate another class into its declaration
- This is done by using extends keyword
- The subclass adds to (extends) the superclass
- The general form is

  class < subclass name > extends <superclass> {

<body of the class: method and variable>}

Overriding Methods

- When we create a subclass of a class, it inherits behavior of the original class
- The subclass can reuse this inherited behavior.
- If you want to modify some of the inherited behavior to match the specialized behavior it is supposed to implement.
- If you modify the behavior by redefining the inherited method in the subclass

- This redefining is popularly known as method overriding.
- The method in the super class is called as overridden method
- The method in the subclass is called overriding method

example

- Consider a class **camera** and its subclass **SLRcamera**.
- The camera class has a **shoot()** method implementing basic photographying.
- The subclass **SLRcamera** is specialized camera needing more adjustment while shooting
- So therefore, it is likely to redefine the shooting behavior

```
Class camera {
    Void shoot() {
        // common code for all cameras
}
}
Class SLRcamera extends camera {
    Void shoot()  {
        // very specific code for SLRcamera
}
}
```

METHOD OVERRIDING

- In a class hierarchy, when a method in a subclass has the same name and type signature as its methods in super class then the method in the subclass is said to be **overriding.**

Example

```
class Base
 {
   Base()
   {
    System.out.println("Constructor of super class");
   }
   void fun1()
   {
    System.out.println("Function inside Super class is
     called");
   }
 }
```

```
class Sub extends Base
{
  Sub()
  {
   System.out.println("Constructor of sub class");
  }
  void fun1()
  {
   System.out.println("Function inside Sub class is called");
  }
}
class MethodOverloadingDemo
{
   public static void main (String args[])
   {
     Sub s=new Sub();
     s.fun1();
   }
}
```

Note:

- Whenever u create an object of a base class,
  - –it will call the super class constructor first
  - –Then the sub class constructor will be called and finally
  - –The functions invoked will be called based on the definition

What is method overriding?

The process of overriding the method present in superclass in subclass is called method overriding. The method should have same name, same signature, and same return type.

What are the rules of method overriding?

- The method must have *same method name* as the method in the superclass. In addition, *the type and order of arguments must be same.*
- The overriding method must have *same return type* as the overridden method
- The access modifiers in subclass but they must be *less restrictive* than the original method in super class.
- The overridden method may not throw any checked exceptions at all. If it throws, the exception must be either same as the exception thrown by the superclass method or the exception must be a subclass of the exception thrown by the superclass method.
- We cannot override a final method of superclass

How dynamic polymorphism is achieved in Java?

It is achieved using method overriding.

What is dynamic polymorphism?

The dynamic polymorphism is the binding of method call to the method body will happen at runtime.

Method Overloading	Method Overriding
Any access modifier can be used	Overriding methods cannot be more restrictive than overridden method
Signature has to be different	Signature has to be the same
Which method to be called will be decided at the time of compilation	Which method to be called will be decided at the time of runtime based on type of object
Method can be static or non static	The static methods don’t participate in overriding
There is no limit on number of overloaded methods a class can have	Each parent class method may be overridden at most once in any sub clsaa

SUPER KEYWORD

- The super keyword is used to refer super class object
- It is used for the following purpose
  - To call superclass method from subclass. If both subclass and superclass contain the same method. That is when the methods are overridden
  - If subclass and superclass contain the same variable, then we can access the superclass variable using super keyword
  - super is used to call superclass constructor explicitly

- What are the rules of using super?
  - The super should be the first statement inside the constructor
  - The **super** and **this** cannot be used together
  - The **supe**r cannot be used inside static methods
  - If we do not write super, then java provides the super() by default

This	Super
It refer to current object	It refer to superclass object
It is used to call the constructor of same class	It is used to call the constructor of superclass
It is used to differentiate between instance variable and local variable of same name	It is used to differentiate between instance variable of subclass and superclass

FINAL CLASS

- Final classes are those classes which cannot be inherited, that is, a final class cannot be subclassed.

- If a class has to be prevented from being inherited, the class can be declared as **final**.

- This is achieved in Java using the keyword **final**

Example

```
final class Base
 {
   Base()
   {
    System.out.println("Constructor of super class");
   }
   void fun1()
   {
    System.out.println("Function inside Super class is
     called");
   }
 }
```

```
class Sub extends Base
{
  Sub()
  {
   System.out.println("Constructor of sub class");
  }
 /* void fun1()
  {
   System.out.println("Function inside Sub class is called");
  }*/
}
class FinalDemo
{
   public static void main (String args[])
   {
     Sub s=new Sub();
     s.fun1();
   }
}
```

When the above program is compiled, it would give the following error.

- ***FinalDemo.java:12: cannot inherit from final Base***
- ***class Sub extends Base***

Final Variables and Methods

- All methods and variables can be overridden by default in subclasses.
- To prevent the subclasses from overriding the members of the super class, they can be declared as final using the modifier as '**final'**.

Example:

```
final  int SIZE = 100;
```

- the value of a final variable can never be changed.
- Final method cannot be altered.

ABSTRACT METHOD

- Abstract Method is a method which does not have any definition which means
  - A method without any implementation
  - Declaration of such method has only the method signature followed by semicolon
  - There will be no body for the method.

Example

```
abstract void method();
```

ABSTRACT CLASS

- It is a class in which contain at least one or more abstract method
- An Abstract class is denoted by the modifier abstract.
- An abstract class can only serve as a base class.
- It cannot be instantiated.

```
abstract class classname
{
   {
        \\variables and Methods Declaration
   }
}
```

```
Abstract class test{
Int a,b,c;
Abstract void method1()
Abstract void method2()
 void method3(){
}
}
```

The above class contain both abstract and normal method. The abstract method does not have body and normal method have body

```
abstract class A
{
 abstract void abfun();
}
class B extends A
{
    void abfun()
 {
   System.out.println("The abstract function completed in
    the subclass");
 }}
class AbstractDemo
{
 public static void main(String args[])
 {
    B b=new B();
            b.abfun();
}}
```

What is an abstract method?

The method which does not have body is called as abstract method. It just contains only method signature.

What is an abstract class?

An abstract class is a class which contains 0 or more abstract methods.

Can we create an object of abstract class?

No. Generally the abstract class consists of incomplete methods like abstract methods. We cannot create an object if the class contains incomplete methods.

Can we declare a class as abstract and final?

No. The abstract class has no life without sub classing. The final classes cannot be sub classed. They both are contradictory.

What is the difference between class and abstract class?

Class	Abstract Class
The class does not contain abstract methods.	It contains abstract methods.
The class can be instantiated.	Abstract classes cannot be instantiated

Note: Important things about Abstract methods and Classes

- Abstract classes cannot be instantiated.
- Any class that contains abstract methods *must* be an abstract class. Otherwise, a compile-time error is thrown.
- Every subclass of an abstract class must provide an implementation for all the abstract methods. If not, the subclass must declare itself as abstract and defer the implementation to it subclasses.
- Abstract method implies that its implementation lies in the subclasses, but declaring abstract methods as private or final prevents overriding and hence makes it impossible to provide any implementation in subclasses. Therefore, compiler does not allow abstract method to be private or final.
- Abstract class can also have *concrete methods or normal methods* besides the abstract methods.
- An abstract class can be declared *without any abstract methods* in it. In that case, the only restriction it has is that it cannot be instantiated.

Garbage Collection

- When we create an object with a new keyword, java allocates heap memory to the newly created object, This memory remains allocated throughout the lifecycle of the object.
- When the object is no more referenced, this allocated heap memory is eligible to be released back to heap as a free memory.
- The mechanism java uses to automatically release the allocated memory is called as the automatic garbage collection

Requesting a garbage collection

- Java provides the facility to request the JVM to perform garbage collection
- When we make such request, the chance that garbage collection will occur in near future increases
- We are making only request and JVM does not guarantee that it will comply with our request

How can we request garbage collection

- We can request garbage collection in two ways

  using RunTime class

  ```
  RunTime runtime = Runtime.getRuntime();
  runtime.gc();
  ```

  using System Class

  ```
  System.gc();
  ```

Finalize() method

- We can do some clean up operation just before an object is garbage collected.
- These operation are known as finalization
- The use of finalization is to release resources held by the object
- This method is a member of the java.lang.object class
- The every class has the object class as its superclass, this method is automatically inherited in all the class

- We can override the finalize method in our class to perform any finalization necessary for objects of that class
- **Following code shows how the dog class can override the finalize() method**

```
Class dog
    protected void finalize() throws
    throwsable {
    system.out.println("garbage collecting
    the dog                    object..");
}
```

- **What is finalization?**

The cleanup operations that performed just before an object is garbage collected is known as finalization

- **Which method should be overridden to perform cleanup activities?**

Protected void finalize() throws Throwable{

}

- **Which is super class for any class in java?**

Java.lang.object class is superclass for all java classes

Access Specifiers

- The access specifier determines the scope or the accessibility of a member of the class.

- JAVA offers four access specifiers
  - private
  - protected
  - public
  - default

Public Access Specifier

- By placing the modifier ***public*** before a member declaration, that member is made available for all the functions inside that class as well as to functions of its derived class.
- By creating an object of a class inside a function of any class, the public members of the class can be accessed.
- Public keyword is necessary to enable web browser or applet viewer to show the applet.
- Ex. public class Square

  { public x,y,size; }

Protected Access Specifier

- By placing the modifier ***protected*** before a member declaration, that member is made available for all the functions inside that class as well as to functions of its derived class in other package.

- It is also accessible to other classes in the same package.

Default Access Specifier

- If the user chooses not to place a modifier in front of the member declaration of a class, the member is created with the default properties.

- This means that they are accessible to all the classes in the same package.

Private Access Specifier

- By placing the modifier ***private*** before a member declaration, that member is made available only inside that class and not to any other class.

Situation	public	protected	default	Private
Accessible to class from same package?	yes	yes	yes	no
Accessible to class from different package?	yes	no, unless it is a subclass	no	no

Array

Arrays

- An array is a collection of homogenous variables that are referred by the common name.

- A specific element in an array is accessed by its index position.

- The number of variables that can be stored in an array is called the array dimension.

- Ex. Int roll_no[5];

Steps to create an Array

1. Declaring array
2. Allocating array(Creating)
3. Initializing array

Declaring array

- An array is declared by specifying the data type of element it is going to hold.
- The array declaration is usually the data type followed by a pair of square bracket followed by the name of the array.

```
datatype[] arrayname;  // syntax
```

example

```
int[] number;
```

- Another way is declaring an array is put the pair of square bracket after the arrayname as

  int number[];

Both declaration is valid

But first form is recommended as it is better readability

Memory model after array declaration diagram

Int[] int Array; // Array declaration

Allocating array or creating array

- The actual array construction with a new keyword involves the memory allocation and the array object creation at runtime

- The following example shows how a simple array is declared at compile time and constructed at runtime

```
Int[]array;   // array declaration
Int array = new int[5];  // array construction at
   runtime
Or
Int[] intarray = new int[5]; // array construction
   at                                    runtime
```

Memory model after the integer array declaration and creation

Int array = new int[5]; // array construction at runtime

Arrays Size

- When an array object is created, we need to specify how many elements it is going to hold
- It is the array size
- We can specify a variable or an expression as a value of the array size

```
Int number = 10;
Int total = number * 2;
Int[] intArray;   // array declaration
Intarray = new int[total]; // array construction at
                                                runtime
```

Initializing array

- Array can either have all of their element initialized at the time of declaration or the elements can be individually initialized after declaration

Initializing Array after declaration

- This is example of an array having all of its element individually after the declaration

```
Int[] myintarray = new int[5];
Myintarray[0]=10;
Myintarray[1]=20;
Myintarray[2]=30;
Myintarray[3]=40;
Myintarray[4]=50;
```

Initializing Array at the time of declaration

- Array can also be created with **an array initializer** without using new operator
- **An array initializer is a code block with a comma separated list of array element, enclosed by a pair of curly braces**
- Example create an array of strings

```
int[] numbers ={ 10,20,30,40,50};
```

Anonymous Array

```
Int[]number;  // array declaration
number ={10,20,30,40,50}; // error
number =new int[]{10,20,30,40,50};//ok
```

- In the above code, number array is declared first.
- However when it is initialized with the initializer block, compile error occur saying" array constant can only be used in initializer
- The last statement can be used instead of second statement. The last statement is called anonymous arrays

Array of Object References

- An array of object references type is created by specifying the object type and size of the array.
- For example an array of Date object is declared and constructed as

```
Date[]birthDate = new Date[5];
```

Array Types - One-Dimensional Array

- The general form of a one-dimensional array declaration is:

  type var-name[];

- Here, type declares the base type of the array. The base type determines the data type of each element that comprises the array.

  int month_days[];

- The array month_days can be linked with an actual, physical array of integers by allocating using ‘new’ keyword and assign it to month_days. ‘new’ is a special operator that allocates memory.
- The general form is

  array-var = new type[size];

```
class AutoArray
{
 public static void main (String args[])
 {
 intmonth_days[]=
   {31,28,31,30,31,30,31,31,30,31,30,31};
 System.out.println("April has " + month_days[3] +
   "days.");
 }
}
```

Output:

April has30days.

Two-Dimensional Array

- In Java, multidimensional arrays are actually array of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.
- For example, the following declares a two-dimensional array variable called twoD.

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

- This allocates a 4 by 5 array and assigns it to twoD.
- Internally this matrix is implemented as an array of arrays of int.

Two-Dimensional Array

```
class FillArray
{
public static void main(String args[])
{
int[][] m;
m=new int[4][5];
for(int row=0;row<4;row++)
{
for(int col=0;col<5;col++)
{
m[row][col]=row+col;
System.out.print("\t elements are" +m[row][col]);
}
}
}
}
```

Strings

- The most important java data types is String
- In many other programming languages String is an array of characters
- This is not the case with java
- String are object in java
- The string class is part of java.lang.package

How strings are created

- We can construct a string just like we construct any other type of object: by using new and call the string constructor

Example

string str = new string(“hello”);

This create a string object called str that contain the character string “Hello”.

- Another example

  string str = “java string are object”

In this case str is initialized to the character sequence “ java string are object”.

Once we created string object, we can use it anywhere that a quoted string is allowed

Why string are called immutable

- String object are created by either using new operator or enclosing a sequence of character in double quotes
- The string object created by either way is immutable
- It means once we create a string object by specifying a sequence of character, that object will always represent that same sequence of character throughout its life

JAVA STRINGS

- Java uses String class to encapsulate string of characters.
- String is a sequence of characters

STRING CONSTRUCTORS

String class provides number of constructors

a. String s=new String() – create instance of string with no characters.
b. String (char[]) – create a string initialized by an array of characters.
c. String(char chars[],int startIndex, int numChars)
d. String(String strObj)-create string object that contains the same character sequence as another string .

Methods of string class

- The string class has many important methods.
- Following are the commonly used methods
- Concat()
- Replace()
- Tolowercase()
- Touppercasse()
- Trim()

Concat()

This method create a new string by appending the content of stringobject passed as argument to the content of string on which the method is invoked

public String concat(String str)

Examples:

String str = “skyward”

Ssytem.out.println(str.concat(“publisher”);

Output: skyward publisher

Replace()

public String replace(char oldChar, char newChar)

Returns a new String resulting from replacing all occurrences of ***oldChar*** in this String with ***newChar***.

Examples:

"mesquite in your cellar".replace('e', 'o')
returns "mosquito in your collar"

public String toLowerCase()

Converts all of the characters in this `String` to lower case.

Examples:
"DOSA".toLowerCase() returns "dosa"

public String toUpperCase()

Converts all of the characters in this `String` to upper case.

Examples:
"india".toUpperCase() returns "INDIA"

public String trim( )

Removes white space from both ends of the String.

Example

```
public class StringDemo
{
    public static void main(String s[])
    {
        char ch;
        String str = "This Is A Test";
        String upper = str.toUpperCase();
        String lower = str.toLowerCase();
        String concat = str.concat("In Java");
        String trm = "     Hello World        ".trim();
        String replac = "Hello".replace('l','w');
        ch = "abc".charAt(2);
        System.out.println(ch);
        System.out.println("Uppercase   " + upper);
        System.out.println("Lowercase   " + lower);
        System.out.println("Concatenate " + concat);
        System.out.println("Trimming              " + trm);
        System.out.println("Replace     " + replac);
    }
}
```

Output:

c

Uppercase THIS IS A TEST

Lowercase this is a test

Concatenate This Is A TestIn Java

Trimming Hello World

Replace Hewwo

STRING BUFFER

- The String class is immutable (constant), i.e. Strings in java, once created and initialized, cannot be changed.
- The String is a final class, no other class can extend it, and you cannot change the state of the string.
- String values cannot be compare with '==', for string value comparision, use equals() method. String class supports various methods, including comparing strings, extracting substrings, searching characters & substrings, converting into either lower case or upper case, etc.

STRING BUFFER

- String buffer represent the characters in java in a growable and modifiable manner.
- It is providing convenient way to modify strings.
- It defines three constructors

A) Stringbuffer ()-reserves room for 16 character

B) Stringbuffer (int num)-accepts int and set the size

C) Stringbuffer (String str)-accepts string and assign room for 16 more characters.

STRING BUFFER

- For example,

String str = “Hello”;

StringBuffer stringBuffer = **new** StringBuffer(str);

Here, capacity of stringBuffer object would be 5 + 16 = 21.

STRING BUFFER

- For example,

```
StringBuffer stringBuffer = new StringBuffer(“Hello World”);
System.out.println(stringBuffer.length());
System.out.println(stringBuffer.capacity());
```

This will print,
11
27

STRING BUFFER METHODS

1. Append ()→used to concatenate string at the end of string buffer object.

Stringbuffer append (String str)

Stringbuffer append (int num)

Stringbuffer append (object obj)

2. charAt() and setCharAt()

- Char charAt(int where)-index of the character is obtained.
- Void setCharAt(int where,char ch)-specifies the index of the character being set ,ch specifies new character .

STRING BUFFER METHODS

1. Delete and deleteCharAt()-used to delete a character.

Stringbuffer delete(int startindex,int endindex)

StringBuffer deleteCharAt(int loc)-delete the character at the index specified by loc.

2. ensureCapacity()-it is used to set the size of the buffer.

Syntax:

Void ensureCapacity(int capacity)

3.getChars()-used to copy a substring into an array.

Syntax:

Void getChars(int sourceStart,int sourceEnd,char target[],int targetstart)

STRING BUFFER METHODS

1. Insert()-insert one string into another.

I. Stringbuffer insert(int index, string str)

II. Stringbuffer insert(int index, char ch)

III. Stringbuffer insert(int index, object obj)

2. length() –used to find the length of the string buffer.

capacity()-used to find total allocated capacity of the buffer.

int length()

Int capacity()

STRING BUFFER METHODS

1. Replace()- replaces one set of characters with another set inside a string buffer.

Syntax: stringbuffer replace(int startindex,int endindex,string str)

2. reverse()-reverse the character in the string buffer.

Syntax: stringbuffer reverse()

3.setlegnth()-used to set the length of the buffer

Void setLengthrin(int len)

4.substring()-returns the substring of a string.

String substring(int startindex,int endindex)

STRING BUFFER METHODS

1. When to use String and when StringBuffer?

If there is a need to change the contents frequently, StringBuffer should be used instead of String because StringBuffer concatenation is significantly faster than String concatenation.

STRING BUFFER METHODS

```
import java.io.*;

public class stringBuffer{
public static void main(String[] args) throws Exception{
BufferedReader in =
new BufferedReader(new InputStreamReader(System.in));
String str;
 try{
System.out.print("Enter your name: ");
str = in.readLine();
  str += ",     This is the example of SringBuffer class and it's functions.";
```

STRING BUFFER METHODS

```
StringBuffer strbuf = new StringBuffer();
strbuf.append(str);
System.out.println(strbuf);
strbuf.delete(0,str.length());
  //append()
strbuf.append("Hello");
strbuf.append("World");
//print HelloWorld
System.out.println(strbuf);
  //insert()
strbuf.insert(5,"_Java ");
//print Hello_Java World
System.out.println(strbuf);
```

STRING BUFFER METHODS

```
    //reverse()
   strbuf.reverse();
   System.out.print("Reversed string : ");
   System.out.println(strbuf);
//print dlroW avaJ_olleH
   strbuf.reverse();
   System.out.println(strbuf);
//print Hello_Java World
    //setCharAt()
   strbuf.setCharAt(5,' ');
   System.out.println(strbuf);
//prit Hello Java World
    //charAt()
   System.out.print("Character at 6th position : ");
   System.out.println(strbuf.charAt(6));
//print J
    //substring()
   System.out.print("Substring from position 3 to 6 : ");
    System.out.println(strbuf.substring(3,7));
```

Vector class

- A Vector class is Java's basic list class.
- A list is an ordered collection of items.
- The operations that can be performed on a list are:
  - Creating a new list
  - Adding an element to the list
  - Removing an element from the list
  - Finding an element from the list
- A list differs from an array in that a list can grow whereas the size of the array is fixed.
- When an element from the List is removed, the space that was occupied by that element will be occupied by another element, whereas when an element of an array is removed, the space occupied by it will not be available for any other purpose and may go waste.

Vector Methods

Method	Description
v.add(o)	adds Object o to Vector v
v.add(i, o)	Inserts Object o at index i, shifting elements up as necessary.
v.clear()	removes all elements from Vector v
v.contains(o)	Returns true if Vector v contains Object o
v.firstElement(i)	Returns the first element.
v.get(i)	Returns the object at int index i.
v.lastElement(i)	Returns the last element.
v.listIterator()	Returns a ListIterator that can be used to go over the Vector. This is a useful alternative to the for loop.
v.remove(i)	Removes the element at position i, and shifts all following elements down.
v.set(i,o)	Sets the element at index i to o.
v.size()	Returns the number of elements in Vector v.
v.toArray(Object[])	The array parameter can be any Object subclass (eg, String). This returns the vector values in that array (or a larger array if necessary). This is useful when you need the generality of a Vector for input, but need the speed of arrays when processing the data.

CREATING VECTORS

- Import vector class from the java.util package.

Import java.util.Vector;

To create vector ,use three steps

i) Declare variable to hold vector

Vector v;

ii) Declare a new vector object and assign it to the vector variable

v=new Vector(); ex. v=new Vector(5);

iii) Store things in the vector

CREATING VECTORS

```
v.addElement(new Integer(1));    // add first vector element
v.addElement(new Float(1.9999)); // add another vector element
for (int i=2; i<10; i++) {
int lastInt = ((Number) v.lastElement()).intValue();
v.addElement(new Integer(i + lastInt));
} // recursively add more elements
```

```
System.out.println(v);
would print
[1, 1.9999, 3, 6, 10, 15, 21, 28, 36, 45]
```

CREATING VECTORS

```
import java.util.Vector;
public class MainClass {
  public static void main(String args[]) {
      Vector v = new Vector(5);
      for (int i = 0; i < 10; i++) {
         v.add(i);
      }
      System.out.println(v);
  }
}
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

ACCESSING ,CHANGING&REMOVING VECTORS

- v.size();→returns the current number of elements/current size of the vector.
- v.removeElementAt(0);→used to delete element of the vector.
- trimToSize()→used to shrunk the capacity.
- V.elementAt(i)→used to get a specific element form the vector.

```
for (int i=0; i<v.size(); i++) {
  System.out.println("v[" + i + "] = " + v.elementAt(i));
}
```

Wrapper Classes

- The java.lang package includes a number of classes that “wrap” a primitive data type in a reference object.
- These classes constitute the ***wrapper classes***.
- The wrapper classes provide object versions of the primitive data types.
- These classes include methods for converting the value from one data type to another.
- The wrapper classes are final.

Important Wrapper Classes

- Integer
- Long
- Byte
- Float
- Double
- Character
- Boolean
- Void

Number Class

- The Number class is the super class for the object wrappers for the int, long, float and double types.

- Any class that expects an instance of a Number may be passed an Integer, Long, Float or Double class.

Integer - Class

The Integer class provides a wrapper for the ***int*** data type.

It contains methods for converting integers to strings and vice versa.

- The constructor takes either of the following form:

- public Integer(int val)
- public Integer(String s) throws NumberFormatException

- In the second form, if the String contains non-numeric character, the NumberFormatException is thrown.
- This class includes methods for fetching information from System properties.

Example

```
class IntegerDemo
{
   public static void main(String args[])
  {
     Integer i=new Integer(7);
     Integer j=new Integer(5);
     Integer k=new Integer(5) ;
     System.out.println(“ Equivalent float Value is :”+ i.floatValue()) ;
     System.out.println(“ Equivalent double Value is :”+ i.doubleValue()) ;
     System.out.println(“ Equivalent byte Value is :”+ i.byteValue()) ;
     System.out.println(“ Equivalent long Value is :”+ i.longValue()) ;
     System.out.println(“ Equivalent String Value is :”+ i.toString()) ;
     System.out.println(“ The objects i and j are equal”+ i.equals(j));
     System.out.println(“ The objects i and k are equal”+ i.equals(k));
     System.out.println(“ The objects k and j are equal”+ k.equals(j));
     System.out.println(“ The int equivalent of string given in command
     line:”+Integer.parseInt(a[0]));
   }
}
```

Long

- The Long class provides a wrapper for the ***long*** data type.
- It contains methods for converting long to strings and vice versa.

The constructor takes either of the following form:

- public Long(long val)
- public Long(String s) throws NumberFormatException

- In the second form, if the String contains non-numeric character, the NumberFormatException is thrown.
- This class includes methods for fetching information from System properties.

Example

```
class LongDemo
{
   public static void main(String a[])
  {
     Long i=new Long (7);
     Long j=new Long (5);
     Long k=new Long (5) ;
     System.out.println(“ Equivalent float Value is :”+ i.floatValue()) ;
     System.out.println(“ Equivalent double Value is :”+ i.doubleValue()) ;
     System.out.println(“ Equivalent byte Value is :”+ i.byteValue()) ;
     System.out.println(“ Equivalent int Value is :”+ i.intValue()) ;
     System.out.println(“ Equivalent String Value is :”+ i.toString()) ;
     System.out.println(“ The objects i and j are equal”+ i.equals(j));
     System.out.println(“ The objects i and k are equal”+ i.equals(k));
     System.out.println(“ The objects k and j are equal”+ k.equals(j));
     System.out.println(“ The long equivalent of string given in command
      line:”+Long.parseLong(a[0]));
  }
}
```

Byte

- The Byte class provides a wrapper for the ***byte*** data type.
- It contains methods for converting byte to strings and vice versa.

- The constructor takes any one of the following form:

  public Byte(byte val)

  public Byte(String s) throws NumberFormatException

- In the second form, if the String contains non-numeric character, the NumberFormatException is thrown.
- This class includes no method for fetching information from System properties.

Example

```
class ByteDemo
{
   public static void main(String a[])
  {
     byte b=7,d=5,c=5;
     Byte i=new Byte(b);
     Byte j=new Byte(d);
     Byte k=new Byte(c) ;
     System.out.println(" Equivalent float Value is :"+ i.floatValue()) ;
     System.out.println(" Equivalent double Value is :"+ i.doubleValue()) ;
     System.out.println(" Equivalent long Value is :"+ i.longValue()) ;
     System.out.println(" Equivalent int Value is :"+ i.intValue()) ;
     System.out.println(" Equivalent String Value is :"+ i.toString()) ;
     System.out.println(" The objects i and j are equal"+ i.equals(j));
     System.out.println(" The objects i and k are equal"+ i.equals(k));
     System.out.println(" The objects k and j are equal"+ k.equals(j));
     System.out.println(" The byte equivalent of string given in command
     line:"+Byte.parseByte(a[0]));
  }
}
```

Short

The Short class provides a wrapper for the ***Short*** data type.

- It contains methods for converting Short to strings and vice versa.

- The constructor takes any one of the following form:

- public Short(Short val)
- public Short(String s) throws NumberFormatException

- In the second form, if the String contains non-numeric character, the NumberFormatException is thrown.

Example

```
class ShortDemo
{
   public static void main(String a[])
  {
     short b=7,d=5,c=5;
     Short i=new Short(b);
     Short j=new Short(d);
     Short k=new Short(c) ;
     System.out.println(" Equivalent float Value is :"+ i.floatValue()) ;
     System.out.println(" Equivalent double Value is :"+ i.doubleValue()) ;
     System.out.println(" Equivalent long Value is :"+ i.longValue()) ;
     System.out.println(" Equivalent int Value is :"+ i.intValue()) ;
     System.out.println(" Equivalent String Value is :"+ i.toString()) ;
     System.out.println(" The objects i and j are equal"+ i.equals(j));
     System.out.println(" The objects i and k are equal"+ i.equals(k));
     System.out.println(" The objects k and j are equal"+ k.equals(j));
     System.out.println(" The Short equivalent of string given in command
     line:"+Short.parseShort(a[0]));
  }
}
```

Float

- The Float class provides a wrapper for the ***float*** data type.

- The following constructors are supported by the Float class.

  - public Float (float value)
  - public Float (double value)
- public Float (string s) throws Number Format Exception

Double

- The Double class provides a wrapper for the double data type. This class supports the following constructors:
- public Double (double value)
- public Double (string s) throws Number Format Exception

Character

- The Character class provides a wrapper for the ***char*** data type.

- It contains methods for converting characters to numeric digits and vice versa, to check whether a given character is an alphabet, number and so on.

- This class has a single constructor.

Boolean

- The Boolean class provides a wrapper for the boolean data type. It has two types of constructors.
  - public Boolean(boolean Value)
  - public Boolean(String str)

Void

- The wrapper class ***Void*** is used for rounding out the set of wrappers for primitive types.

- This wrapper class has no constructor or method and contains only the TYPE attribute that is common to all the wrapper classes.

Multithreading

Dr.T.Logeswari

Why do we need threads?

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU
- Prioritize your work depending on priority

Example

- Consider a simple web server
- The web server listens for request and serves it
- If the web server was not multithreaded, the requests processing would be in a queue, thus increasing the response time and also might hang the server if there was a bad request.
- By implementing in a multithreaded environment, the web server can serve multiple request simultaneously thus improving response time

Main Thread

- Default Thread in any Java Program
- JVM uses to execute program statements
  - **Program To Find the Main Thread**

```
Class Current
{
  public static void main(String args[])
  {
     Thread t=Thread.currentThread();
     System.out.println("Current Thread: "+t);
     System.out.println("Name is: "+t.getName());
  }
}
```

Output:

```
C:\> javac Current.java
C:\>java Current
Current Thread: Thread[main,5,main]
Name is: main
```

Creating threads

- In java threads can be created by extending the Thread class or implementing the Runnable Interface
- It is more preferred to implement the Runnable Interface so that we can extend properties from other classes
- Implement the run() method which is the starting point for thread execution

Running threads

- Example

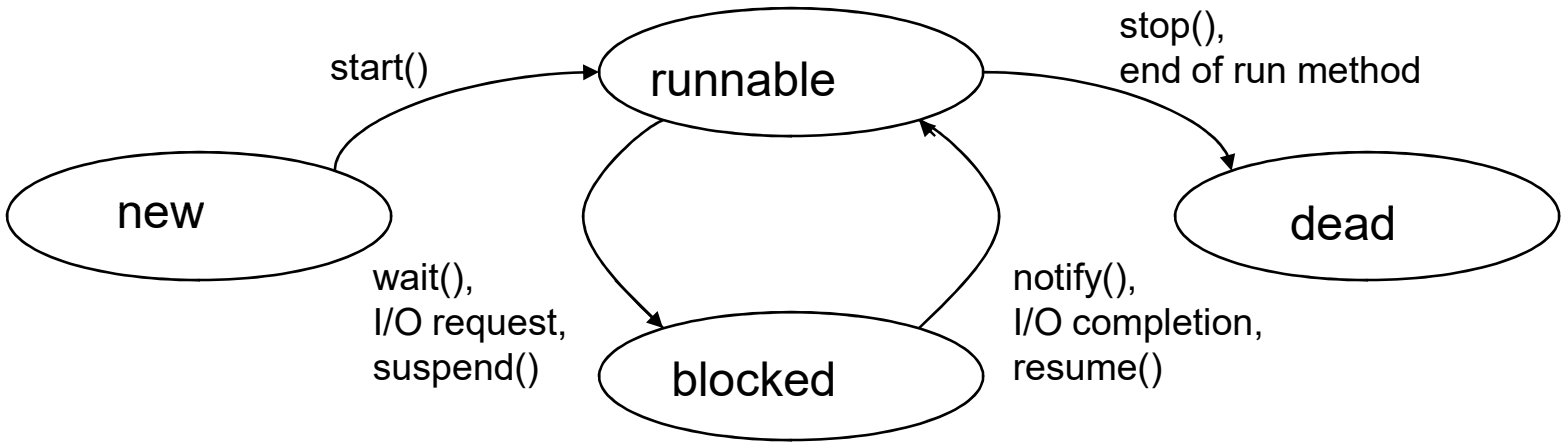
```
class mythread implements Runnable{
        public void run(){
                System.out.println(“Thread Started”);
        }
}
        class mainclass {
        public static void main(String args[]){
                Thread  t = new Thread(new mythread()); // This is the way to instantiate a  thread implementing runnable interface
                t.start(); // starts the thread by running the run method
                }
}
```

- Calling t.run() does not start a thread, it is just a simple method call.
- Creating an object does not create a thread, calling start() method creates the thread.

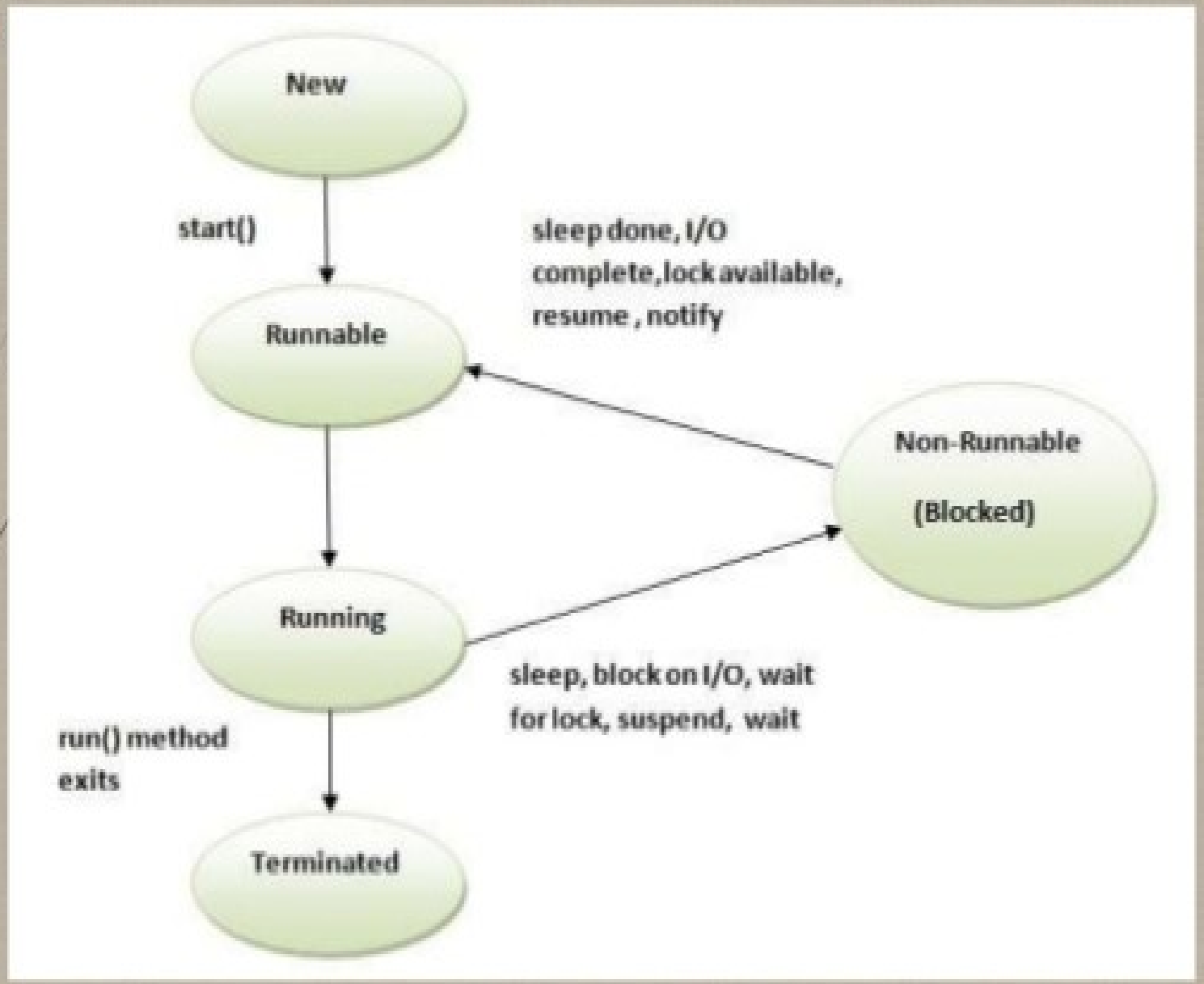
States of Java Threads

- 4 separate states
  - new: just created but not started
  - runnable: created, started, and able to run
  - blocked: created and started but unable to run because it is waiting for some event to occur
  - dead: thread has finished or been stopped

States of Java Threads



Life cycle of a Thread



- **New**

  The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

- **Runnable**

  The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

- **Running**

  The thread is in running state if the thread scheduler has selected it.

- **Non-Runnable (Blocked)**

  This is the state when the thread is still alive, but is currently not eligible to run.

- **Terminated**

  A thread is in terminated or dead state when its run() method exits.

Controlling Java Threads

- _.*start()*: begins a thread running
- *wait()* and *notify()*: for synchronization
  - more on this later
- _.*stop()*: kills a specific thread (deprecated)
- _.*suspend()* and *resume()*: deprecated
- _.*join()*: wait for specific thread to finish
- _.*setPriority()*: 0 to 10 (MIN_PRIORITY to MAX_PRIORITY); 5 is default (NORM_PRIORITY)

Java Thread Scheduling

- highest priority thread runs
  - if more than one, arbitrary
- *yield()*: current thread gives up processor so another of equal priority can run
  - if none of equal priority, it runs again
- *sleep(msec)*: stop executing for set time
  - lower priority thread can run

❖ Thread Synchronization

A shared resource may be corrupted if it is accessed simultaneously by multiple threads.

Example: two unsynchronized threads accessing the same bank account may cause conflict.

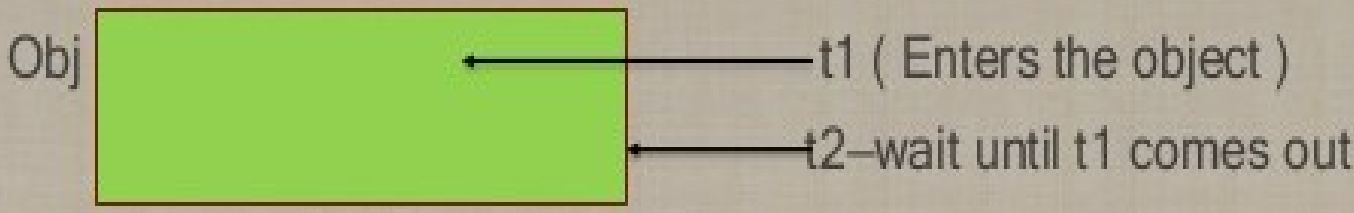
- Known as a *race condition* in multithreaded programs.

❑ A *thread-safe* class does not cause a race condition in the presence of multiple threads.

Synchronized

- **Problem** : race conditions
- **Solution** : give exclusive access to one thread at a time to code that manipulates a shared object.
- Synchronization keeps other threads waiting until the object is available.
- The synchronized keyword synchronizes the method so that only one thread can access the method at a time.

```
public synchronized void xMethod() {
   // method body
}
```



Synchronization

- Synchronization is prevent data corruption
- Synchronization allows only one thread to perform an operation on a object at a time.
- If multiple threads require an access to an object, synchronization helps in maintaining consistency.

Example

```
public class Counter{
    private int count = 0;
    public int getCount(){
            return count;
    }
          public setCount(int count){
            this.count = count;
    }
}
```

- In this example, the counter tells how many an access has been made.
- If a thread is accessing setCount and updating count and another thread is accessing getCount at the same time, there will be inconsistency in the value of count.

Fixing the example

```
public class Counter{
    private static int count = 0;
    public synchronized int getCount(){
            return count;
    }

    public synchoronized setCount(int count){
             this.count = count;
    }
}
```

- By adding the synchronized keyword we make sure that when one thread is in the setCount method the other threads are all in waiting state.
- The synchronized keyword places a lock on the object, and hence locks all the other methods which have the keyword synchronized. The lock does not lock the methods without the keyword synchronized and hence they are open to access by other threads.

What about static methods?

```
public class Counter{
    private int count = 0;
    public static synchronized int getCount(){
            return count;
    }
          public static synchronized setCount(int count){
            this.count = count;
    }
}
```

- In this example the methods are static and hence are associated with the class object and not the instance.
- Hence the lock is placed on the class object that is, Counter.class object and not on the object itself. Any other non static synchronized methods are still available for access by other threads

Common Synchronization mistake

```
public class Counter{
    private int count = 0;
    public static synchronized int getCount(){
            return count;
    }
            public synchronized setCount(int count){
              this.count = count;
    }
}
```

- The common mistake here is one method is static synchronized and another method is non static synchronized.
- This makes a difference as locks are placed on two different objects. The class object and the instance and hence two different threads can access the methods simultaneously.

Object locking

- The object can be explicitly locked in this way

```
synchronized(myInstance){
try{
wait();
}catch(InterruptedException ex){

}
    System.out.println("Iam in this ");
    notifyAll();
}
```

- The synchronized keyword locks the object. The wait keyword waits for the lock to be acquired, if the object was already locked by another thread. Notifyall() notifies other threads that the lock is about to be released by the current thread.
- Another method notify() is available for use, which wakes up only the next thread which is in queue for the object, notifyall() wakes up all the threads and transfers the lock to another thread having the highest priority.

Further Reading

- http://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html
- http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html

Exception Handling

Types of error in java

- Compile time error(forgetting semicolon in program)
- Run time error (main() has no arguments)
- Logical error(flaw in the logic of the program)

- An exception is an abnormal condition that arises in a code sequence at a run time.
- The Exception class defines the possible error conditions that the program may encounter.
- It is an event ,which occurs during the execution of a program ,that disturbs the normal flow of the program instructions.

Exceptions occurs when the

- user is trying to open a file that does not exist
- network connection is disconnected
- Operands which are manipulated do not fall within a prescribed range
- class file is missing

- What is exception?

  An exception is an error that occurs at run time
- What is an exception handling?

  whenever exception handling occurs in the program, the concept of handling the class object and avoiding them from reaching back to JVM (nothing but exception handling). Java to deal with handling the error in an organized fashion is called exception handling

- How java handles exception?

  java provides the exception handling constructs like try catch, try catch finally to manage run time error

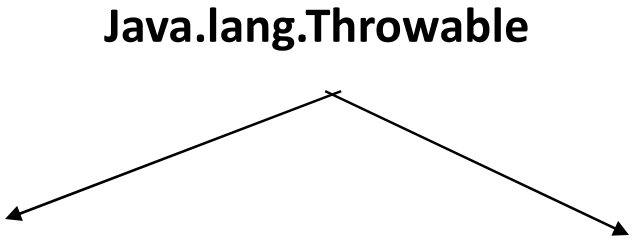
Types of Exception

- In java, all classes are represented by classes.
- All exception classes are derived from a class called throwable
- The Exception are classified into two types
  - Exception
  - Error

•Error and exception are subclasses of throwable

•Object class is the super class of Throwable class

Exception Hierarchy



java.lang.error **java.lang.Exception**

java.lang.Error

It represents normally a series of non-executable code such as running out of memory or being unable to locate a class.(stackOverflowerror)

java.lang.Exception

It represents unusual conditions that arise in the course of program executions, such as reaching end of file, attempting to reference an array element outside the actual source of the array. (divide by zero)

- **Throwable** is at the top of the exception class hierarchy.
- All exception types are the subclasses of the built-in class **Throwable**.
- ***Throwable*** class has two subclasses that partition the exception into two distinct branches.
- One branch is headed by **Exception.** This class is used for exceptional conditions that user programs should catch. This class creates custom exception types.
- There is an important subclass of Exception, called **Run-time Exception**.

- The other branch is topped by **Error**, which defines exceptions that are not expected to be caught under normal circumstances by program.

- Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.

Runtime or unchecked exception

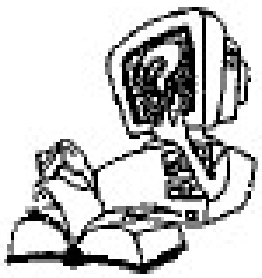
we don't handle some of the exceptions. ***The exception objects for which compiler does not compel to handle them is known as unchecked exceptions.***

RuntimeExceptions are:

- ArithmeticException (often the result of dividing by 0)
- ClassCastException (trying to perform an illegal cast operation)
- IndexArrayOutOfBoundsException
  (e.g., accessing element 15 of a 10-element array)
- NullPointerException (trying to access an object when the variable contains null)

Checked exception

- The checked exception are checked at the compile time by the compiler and it compel to handle the exception in the program (IOException)



How Exceptions are classified?

Exceptions are of two types: Error and Exception.

What is the difference between Error and Exception?

An exception is an error which can be handled. An error is an error which cannot be handled. Errors are generated by the JVM itself and hence it cannot be handled.

Example: OutofMemory.

What is Throwable?

The Throwable is a class represents all errors and exceptions which may occur in Java

Which is the Super class of all Exceptions?

Exception

What is checked exception? Give an example.

The checked exceptions are checked at the compile time by the compiler and it compels to handle the exception in the program.

Example: IOException, FileNotFoundException, InterruptedException etc.,

What is unchecked exception? Give an example.

The unchecked exceptions are checked by the JVM and compiler does not compel to handle the exception in the program.

Example: ArithmeticException, ClassCastException, IndexArrayOutOfBoundsException etc.,

Types of Errors

- An error may produce an incorrect output or may terminate the execution of the program abruptly.

- It is therefore important to detect and manage properly, all the possible error conditions in the program.

Errors are broadly classified into two categories:

- Compile-time errors and
- Run-time errors

Compile Time Errors

- All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as **Compile-Time** errors. Whenever the compiler displays an error, it will not create the **.class** file.

- Some of the compile time errors are:

- Missing semicolons
- Missing (or mismatch of) brackets in classes and methods
- Misspelling of identifiers and keywords
- Incompatible types in assignments/initialization etc.

Run Time Errors

Some of the run-time errors are:

- Dividing an integer by zero.
- Accessing an element that is out of the bounds of an array.
- Trying to store a value into an array of an incompatible class or type.
- Passing a parameter that is not in a valid range or value for a method.

Keywords in Exception Handling

- Java exception handling is managed via five keywords: **try, catch, throw, throws and finally.**
- To handle a run-time error, simply enclose the code that requires to be monitored inside a **try** block.
- Immediately following the try block, a **catch** clause that specifies the exception type that has to be caught is included

A general form of an exception-handling block

```
try
{
   // block of code to monitor for errors
}
catch(ExceptionType1   exOb){
  // exception handler for ExceptionType1
}
 catch(ExceptionType2   exOb) {
 // exception handler for ExpectionType2
}
     //. . .

finally
{
   // block of code to be executed before try block ends
}
```

Exception Type	Cause of Exception
ArithmeticException	Caused by mathematical errors such as division by zero.
ArrayIndexOutOfBounds	Caused by an array indexes. Exception
ArrayStoreException	Caused when a program tries to store the wrong type of data in an array.
FileNotFoundException	Caused by an attempt to access a Nonexistent file.
IOException	Caused by general I/O failures, such as Inability to read from a file.
NullPointerException	Caused by referencing a null Object.
NumberFormatException	Caused when a conversion between strings and number fails.

OutofMemoryException	Caused when there is not enough memory to allocate new object.
Security Exception	Caused when an applet tries to perform an action not allowed by the browser's security setting.
StackOverflow Exception	Caused when the system runs out of stack space.
StringIndexOutOf BoundsException	Caused when a program attempts to access a nonexistent character position

The following program includes a try block and a catch clause which processes the ArithmeticException generated by the division-by-zero error:

```
class Exc2
{
   public  static void main(String args[])
   {
    int d,a;
        try
        {        // monitor  a block of code.
           d = 0;
           a = 42/d;
          System.out.println("This will not be printed.");
        }
      catch (ArithmeticException e)
        {    // catch divide-by zero error
              System.out.println ("Division by zero");
        }
      System.out.println ("After catch statement.");
  }
}
```

Multiple Catch statements

It is possible to have more than one catch statement in the catch block as illustrated:

```
. . . . . .
. . . . . .
 try
{
 statement;            // generates an exception
}
catch (Exception-Type-1 e)
{
  statement;          // processes exception type 1
}
catch  (Exception-Type-2 e)
{
  statement;          // processes exception type 2
}
catch (Exception-Type-N e)
{
  statement ;         // processes exception type N
}
```

Exception (contd…)

- When an exception in a try block is generated, the Java treats the multiple catch statements like cases in a switch statement.

- The first statement whose parameter matches with the exception object will be executed and the remaining statements will be skipped.

Throw

- It is possible to throw an exception explicitly, using the throw statement. The general form of throw is shown here:

  throw ThrowableInstance

- ThrowableInstance must be an object of type Throwable or a subclass of Throwable.

  There are two ways by which a Throwable object can be obtained
  - using a parameter into a Catch clause or
  - creating one with the new operator.

- The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.
- The nearest enclosing try block is inspected to see if it has a catch statement that matches the type of exception. If it does find a match, control is transferred to that statement.
- If not, then the next enclosing try statement is inspected and so on.
- If no matching catch is found, then the default exception handler halts the program and prints the stack trace.

```
// Demonstrate throw.
   class ThrowDemo
   {
     static void demoproc()
     {
       try
       {
           throw new NullPointerException("demo");
       }
        catch(NullPointerException e)
        {
          System.out.println("Caught inside demoproc.");
           throw e; // rethrow the exception
        }
     }
```

```
    public static void main(String args[])
    {
     try
     {
        demoproc();
     }
    catch (NullPointerException e)
    {
      System.out.println("Recaught: " + e);
    }
  }
}
```

Output:

Caught inside demoproc.
Recaught: java.lang.NullPointerException: demo

Program description

- In this program, the class ThrowDemo has a method called demoproc() which uses a throw statement to throw a **NullPointerException**.

- The function itself provides a try and catch clause to handle the exception thrown.

- The catch clause after handling the exception, once again throws it. Therefore, whenever any function calls the function demoproc(), it has to handle the exception thrown by the catch clause inside the demoproc().

- Therefore, the main function encloses the call to the function demoproc() inside the try catch clause.

What is the difference between throw and throws

- The throws clause is used when the programmer does not want to handle the exception in the method and throw it out of method
- The throw clause is used when the programmer want to throw an exception explicitly and wants to handle it using catch block.
- Hence throw and throws are contradictory

finally block

When a **finally** block is defined, this is guaranteed to execute, regardless of whether or not an exception is thrown.

```
class FinallyDemo
{
    // Through an exception out of the method.
  static void procA()
  {
     try
    {        System.out.println("inside procA");
       throw new RuntimeException("demo");
    }
    finally
    {
          System.out.println("procA-s finally");
    }  } // procA
```

```
      // Return from within a try block.
static void procB()
{
   try
   {
      System.out.println("inside proB");
      return;
   }
    finally
    {
        System.out.println("proB's finally");
    }
 } // procB
```

```
// Execute a try block normally.
        static void procC()
        {           try
                      {               System.out.println("inside procC");
                       }
                      finally
                      {                                 System.out.println("procC's finally");
                       }
       } // procC
       public static  void main(String args[])
      {           try
                   {
                          procA();
                   } catch (Exception e)
                  {                         System.out.println("Exception caught");
                   }
                    procB();
                    procC();
       }  // main
  } //class
```

Applet Programming

- An applet is a java class that can be downloaded and executed by the web browser. It is a specific type of java technology. An applet runs in the environment of the web browser.
- The applet can be executed by 2 methods:
    - Using HTML document
    - Using appletviewer

Applet Programming

- **APPLET=JAVA Byte Code + HTML**

What is an Applet?

- **Applet is a small application that is embedded in a HTML page , which is accessed and transported over the internet, automatically installed into the client machine and runs as part of a web page.**

- **Applets are great for creating dynamic and interactive web application.**

Using HTML Document

- Once an applet has been compiled, it is included in a HTML file using the APPLET tag.
- When the HTML file is loaded in the browser, the applet will be automatically executed

```
/*
<applet code ="MyApplet" width=200 height = 60>
</applet>
*/
```

Description of Applet Tag

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer. The syntax for the standard APPLET tag is shown here.

```
<APPLET
   [CODEBASE = codebaseURL]
   [CODE = appletFile]
   [ALT = alternateText]
   [NAME = appletInstanceName]
   [WIDTH = pixels HEIGHT = pixels]
   [ALIGN = alignment]
   [VSPACE = pixels] [HSPACE = pixels]
   [<PARAM NAME = AttributeName VALUE = AttributeValue>]
   [<PARAM NAME = AttributeName2 VALUE = AttributeValue>]
…
</APPLET>
```

CODEBASE

CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file.

CODE

CODE is a required attribute that gives the name of the file containing user applet's complied .class file. This file is relative to the code base URL of the applet, which is the directory that the HTML file was in.

ALT

The **ALT** tag is an optional attribute used to specify a short text message that should be displayed.

NAME

NAME is an optional attribute used to specify a name for the applet instance. To obtain an applet by name, use getApplet(), which is defined by the AppletContext interface.

WIDTH AND HEIGHT

WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

ALIGN

ALIGN is an optional attribute that specifies the alignment of the applet with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTOP, ABSMIDDLE and ABSBOTTOM.

VSPACE AND HSPACE

These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet. HSPACE specifies the space, in pixels, on each side of the applet.

PARAM NAME AND VALUE

The PARAM tag allows the user to specify applet specific arguments in an HTML page. Applets access their attributes with the getParameter() method.

Using appletviewer

- The applet code can be run using an appletviewer. In this case, the appletviewer is typed at the command prompt followed by the name of the java file. The general format is:

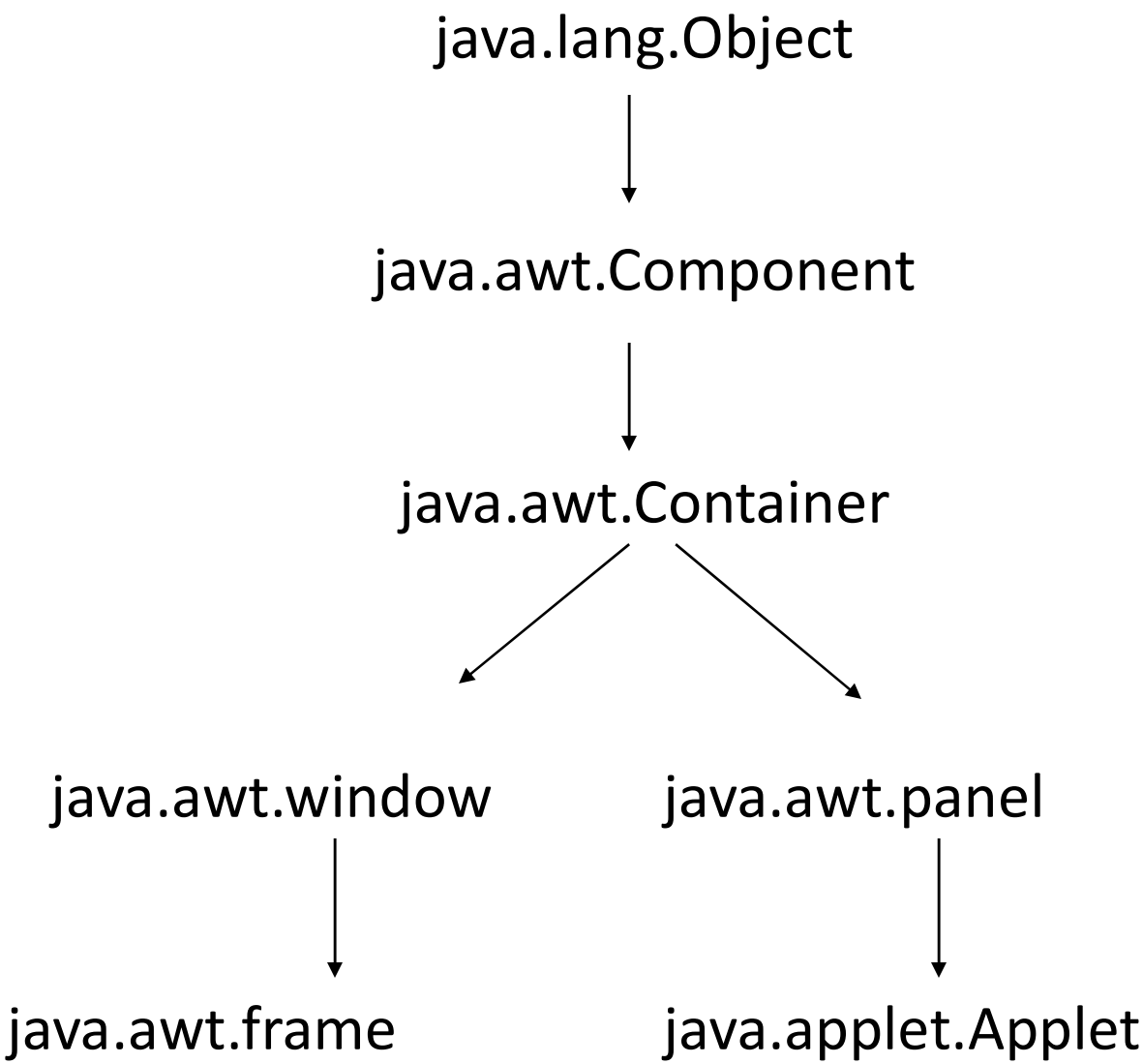
  Prompt> appletviewer filename.java

- The applet tag has to be included in the java file itself and has to be commented (i.e. enclosed between /* and */).

Applet Hierarchy

- All applets are the subclasses of the class **Applet. Applet** class belongs to the package ***java.applet***.
- All user-defined applet must import **java.applet** package.
- Applet extends from a class called Panel present in the package ***java.awt***.
- This class provides support for Java's windows-based graphical user interface.
- Thus, Applet provides all of the necessary support for window-based activities.

Hierarchy of Applet class



The Applet Class

The class **java.applet.Applet** is a subclass of java.awt.panel.

An applet is a window-based program.

Applets are event driven.

Event driven means for every interaction from the user, a particular action takes place in the applet.

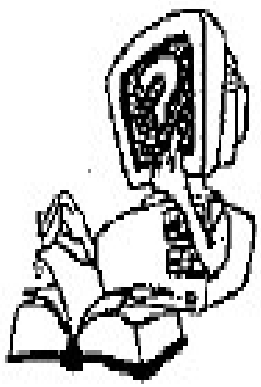
An applet will wait until an event occurs.

The AWT notifies the applet about an event by calling an event handler that has been provided by the applet.

Once this happens, the applet must take appropriate action and then quickly return control to the AWT

Applet differ from application

Applets	Applications
Applets do not have main() method.	Applications have main() method.
Applet runs under the control of a java compatible container, such as a web browser.	An application is independently executed.
Applets only are executed within a Java browser or appletviewer.	The application programs can be compatible executed using Java Interpreter.
Applet is restricted NOT to utilize file system and network resources. This means, applets cannot have access to file system as well as network resources for security reasons.	Applications can utilize the network and file system resources. But applications have access to both file system and network resources



Explain the process of execution of an applet?

The applets are like Java programs, we need to create the program. Once the applet is created, we compile and obtain its byte code. This byte code is embedded in HTML page. The HTML page can be viewed using web browsers like Internet Explorer or Mozilla. The browser will execute the applet's byte code present in the HTML page using applet engine present in the browser.

Where the applets are executed?

Applets are executed by a program called applet engine which is similar to java virtual machine that exist inside the web browser.

What are local and remote applets?

An applet developed and executed in the local machine is called local applet. No need of internet connection or network connection for local applets. The remote applets are developed by someone else and stored on remote machine. The internet/network connection is required to load that applet.

What is the flow of executing remote applet?

1. The user sends a request for an HTML document to remote machine Web server.
2. The Web server returns an HTML document to the user's browser. The HTMLdocument contains the <APPLET> tag that identifies the applet.
3. The bytecode corresponding to that applet is transferred to the user's host. This bytecode was created previously by the Java compiler using the Java source code file for that applet.
4. The applet engine on the user's host interprets the bytecode and provides display.
5. The user then can use the applet with no further downloading from the remote machine Web server. This is because the bytecode contains all the information necessary to run the applet.

Java Applet with 5 methods

```
import java.awt.*;
import java.applet.*;
class Myclass extends Applet {
public void init () {
/* All the variables, methods and images initialize here
will be called only once because this method is called only
        once when the applet is first initializes */
}
public void start () {
/* the components needed to be initialize more than once
in your applet are written here or if the reader
switches back and forth in the applets. This method
can be called more than once.*/
}
public void stop () {
/* This method is the counterpart to start (). The code,
used too stop the execution is written here*/
}

public void destroy() {
/* This method contains the code that result in to release in to release
the resources to the applet before it is
finished. This method is called only once. */
}
public void paint (Graphics g) {
/* write the code in this method to draw, write, or color
things on the applet pane are*/
}
}
```

Life Cycle of an Applet

- The init() method provides the capability to load applet parameters and perform any necessary initialization processing.
- The start() method serves as the execution entry point for an applet, when it is initially executed and restarts the applet.
- The stop() method provides the capability to stop() an applet's execution when the Web page containing the applet is no longer active.
- The destroy() method is used at the end of an applet's life cycle to perform any termination processing.

It is important to understand the order in which the various methods of the applet class are called. When an applet begins, the AWT calls the following methods, in the sequence:

1. init()
2. start()
3. paint()

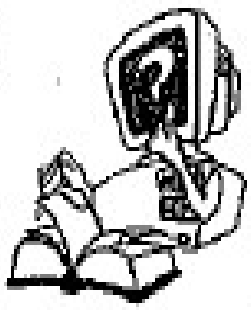
When an applet is terminated, the following methods are called:

1. Stop(), called when the applet is minimized.
2. Destroy(), called when the applet is closed.

1. Java applications are designed to run the homogeneous and more secure area.
2. Java applets are designed to run the heterogeneous and unsecured environment
3. Applets are not capable of communicate to the server.
4. Not capable of reading and writing the users file system.
5. Applet – is a window based program & it works on event driven architecture

What is applet life cycle?

An applet is born with *init()* method and starts executing with *start()* method. To stop the applet, *stop()* method is called and to terminate the applet *destroy()* method is called. Once the applet is terminated, we should reload the HTML page to get the applet start once again from *init()* method. This way of executing the methods are called as life cycle of an applet.



What package must be included when creating an applet?

The package java.applet must be included when creating an applet.

What are the five methods that most applets will override?

The five methods are init( ), start( ), stop( ), destroy( ), and paint( ).

What method outputs to the applet's window?

The paint( ) method displays output in an applet's window.

Which methods are called only once in the life cycle of an applet?

init() and destroy() methods are called only once in the life cycle of an applet.

What is the order of method invocation in an applet?

It is important to understand the order in which the methods are executed. When an applet begins, the following methods are called in this sequence:

1. init( ) 2. start( ) 3. paint( )

When an applet is terminated, the following sequence of method calls takes place:

1. stop( ) 2. destroy( )

Important Note:

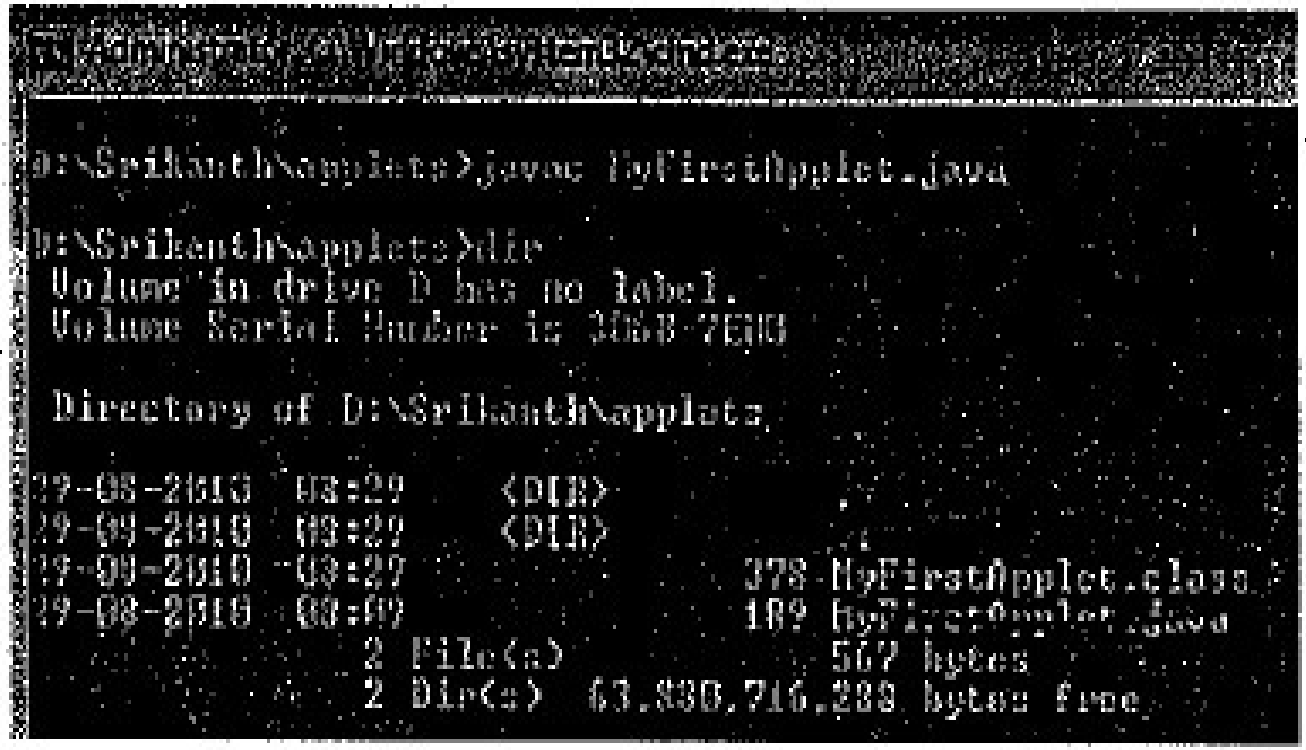
- None of the life cycle methods are compulsory while creating an applet.
- All methods should be public; otherwise they are not available to browser to execute.

1.Writing an applet code

```
import java.applet.*;
import java.awt.*;
public class paint extends Applet
{
      public void paint(Graphics g)
{
 g.drawString("welcome ",40,60);
}
}
```

2.Compile applet code &generate byte code

- Save the applet code and generate the byte code using Java Compiler as shown below.



Now, we have got byte code (MyFirstApplet.class) and this byte code is required to embedded with HTML page.

3.Create an HTML page

1. Open a notepad or any HTML editor. Type the below code.
2. In the body of the HTML page, enter the Applet tag as given below
3. Save the file as AppletExample.html in the same directory where we have stored Applet code

```
1 <HTML>
2     <HEAD>
3     <TITLE> This is my first Applet </TITLE>
4         <BODY>
5          <Applet Code=MyFirstApplet.class width= 400 height=500>
6          </Applet>
7         </BODY>
8     <HEAD>
9 </HTML>
```

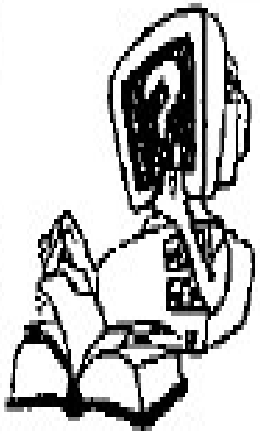
4.Execute using applet viewer

There are two ways in which we can run an applet: inside a browser or with a special development tool that displays applets. The tool provided with the standard Java JDK is called **appletviewer**. We can also run applets in web browser like Internet explorer.

```
appletviewer AppletExample.html
```

Important Note:

- It is very clear from the output that, title mentioned in the HTML code is not displayed in the output. The reason is that, when we execute the applet using an *appletviewer*, all the html tags are ignored and only the applet tag will be considered for execution.
- When we execute the same using web browser, we may not get the applet's window frame like above and it will display all the contents mentioned in the html tags. Example: title of the html page.



What are the different ways of executing an applet?

There are two ways in which we can run an applet:

- Using Java Enabled Web browser.
- Using an applet viewer..

What is an appletviewer?

An *appletviewer* is tool provided with the standard Java JDK to execute an applet.

Which tag is used to add an applet inside an HTML?

<APPLET> tag is used to add an applet inside an HTML.

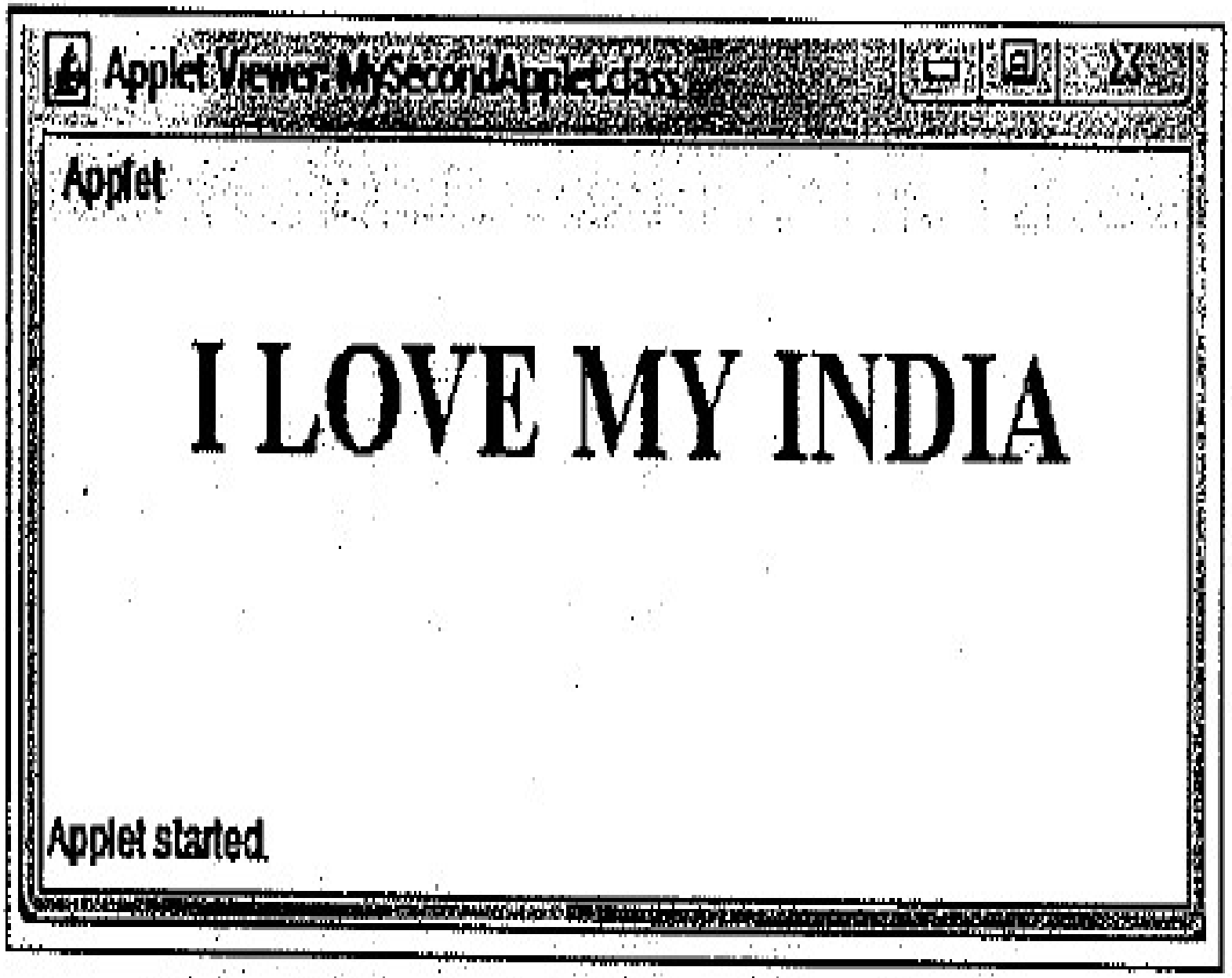
Can we execute an applet without writing HTML?

Yes. We can execute an applet without writing HTML by embedding <APPLET> tag inside Java Applet Code. This should be in comments.

Writing and executing applet without HTML

- For quick testing of an applet, simply include a comment near the top of applet's source code file that contains the APPLET tag. Here the applet source code and applet tag are in the same Java file. The source looks like

```
1  import java.applet.Applet;
2  import java.awt.Font;
3  import java.awt.Graphics;
4
5  /* <Applet Code=MySecondApplet.class width= 400 height=500>
6     </Applet>
7  */
8
9  public class MySecondApplet extends Applet {
0
1      public void paint(Graphics g){
2          Font f = new Font("TimesRoman", Font.BOLD, 36);
3          g.setFont(f);
4          g.drawString("Welcome to Java",50,50);
5      }
6
7  }
```


Applet Viewer: MySecondApplet.class
Applet
I LOVE MY INDIA
Applet started.

Passing parameters to applet

Step 1: Create Applet and Compile

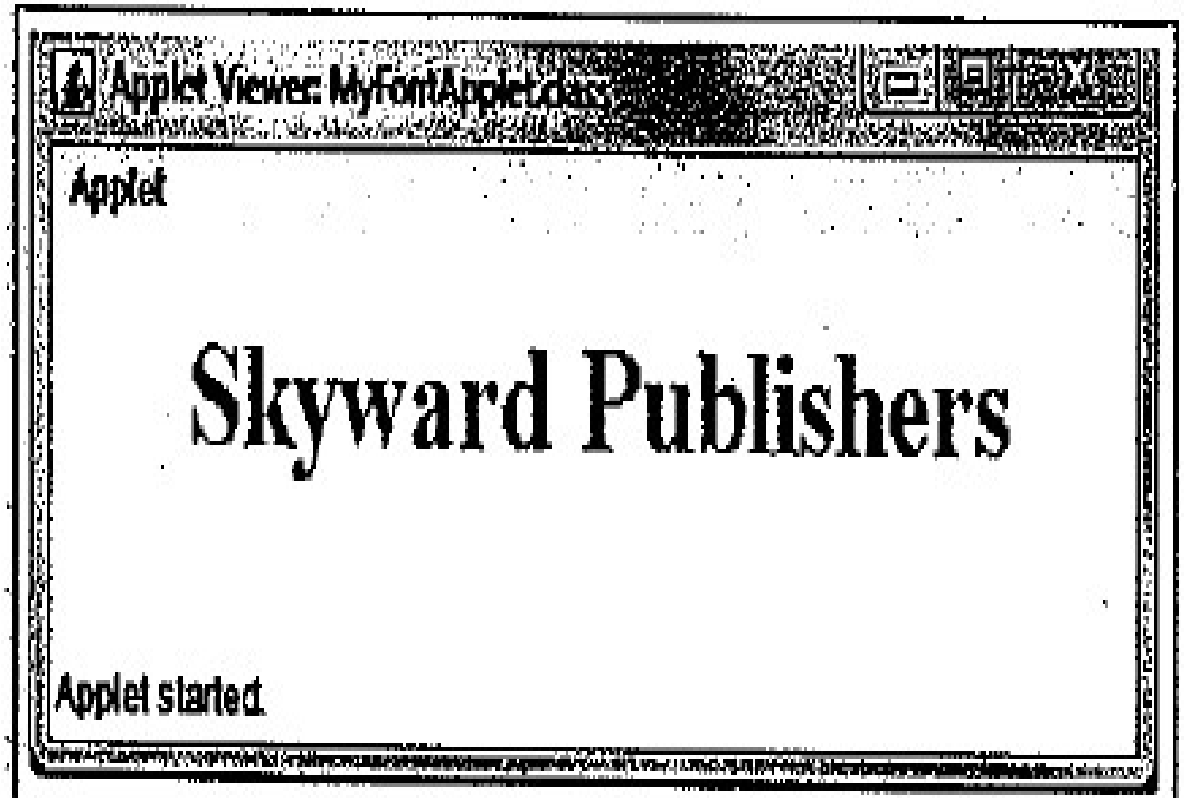
```
1  import java.applet.Applet;
2  import java.awt.Font;
3  import java.awt.Graphics;
4
5  public class MyFontApplet extends Applet {
6
7      String fontName;
8      int fontSize;
9
10     public void init() {
11         fontName = getParameter("font");
12         fontSize = Integer.parseInt(getParameter("size"));
13     }
14
15     public void paint(Graphics g) {
16         /* Create font object with fontName and fontSize */
17         Font f = new Font(fontName, Font.BOLD, fontSize);
18         g.setFont(f);
19         g.drawString("Skyward Publishers", 50, 50);
20     }
21
22 }
```

Step 2: Create HTML called FontExample.html

```
1<HTML>
2    <HEAD>
3    <TITLE> Parameter Passing to an Applet </TITLE>
4       <BODY>
5        <Applet Code="MyFontApplet.class width= 400 height=500>
6            <PARAM NAME="font VALUE="TimesRoman">
7            <PARAM NAME=size VALUE="36">
8        </Applet>
9       </BODY>
10   </HEAD>
11</HTML>
```

Step 3: Execution and Output

```
D:\Srikanth\Applets>appletviewer FontExample.html
```



PASSING PARAMETER TO APPLET

We know that how to pass parameters to main() method by using command line arguments. Similarly we can pass parameters to an applet. Applets can get different input from the HTML file that contains the <APPLET> tag through the use of applet parameters. To set up and handle parameters in an applet, we need two things:

- A special parameter tag in the HTML file.
- Code in our applet to read those parameters.

Applet parameters come in two parts: a parameter name, which is simply a name we give, and a value, which is the actual value of that particular parameter.

In the HTML file that contains the embedded applet, we indicate each parameter using the <PARAM> tag, which has two attributes for the name and the value, called NAME and VALUE. The <PARAM> tag goes inside the opening and closing <APPLET> tags:

Example:

```
<APPLET CODE="MyFontApplet.class" WIDTH=100 HEIGHT=100>
<PARAM NAME=font VALUE="TimesRoman">
<PARAM NAME=size VALUE="36">
</APPLET>
```

Parameters are passed to our applet when it is loaded. In the *init()* method of applet, we can read the value of the parameter using the ***getParameter()*** method. The *getParameter()* takes one argument-a string representing the name of the parameter we are looking for-and returns a string containing the corresponding value of that parameter.

Important Note:

- The names of the parameters as specified in <PARAM> and the names of the parameters in getParameter() must match identically, including having the same case. In other words, <PARAM NAME="name"> is different from <PARAM NAME="Name">. If parameters are not being properly passed to applet, make sure the parameter cases match.

Example:

```
String fontName;
String fontSize;

public void init()
{
 fontName=getParameter("font");
 fontSize=getParameter("size");
}
```

Here, we have used two Strings *fontName* and *fontSize* to receive the values of font and size parameters. The string *fontName* will get "***TimeRoman***" and *fontSize* will get "***36***".

ALIGNING THE APPLET DISPLAY

The ALIGN attribute defines how the applet will be aligned on the page. This attribute can have one of nine values: LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, or ABSBOTTOM. In the case of ALIGN=LEFT and ALIGN=RIGHT, the applet is placed at the left or right margin

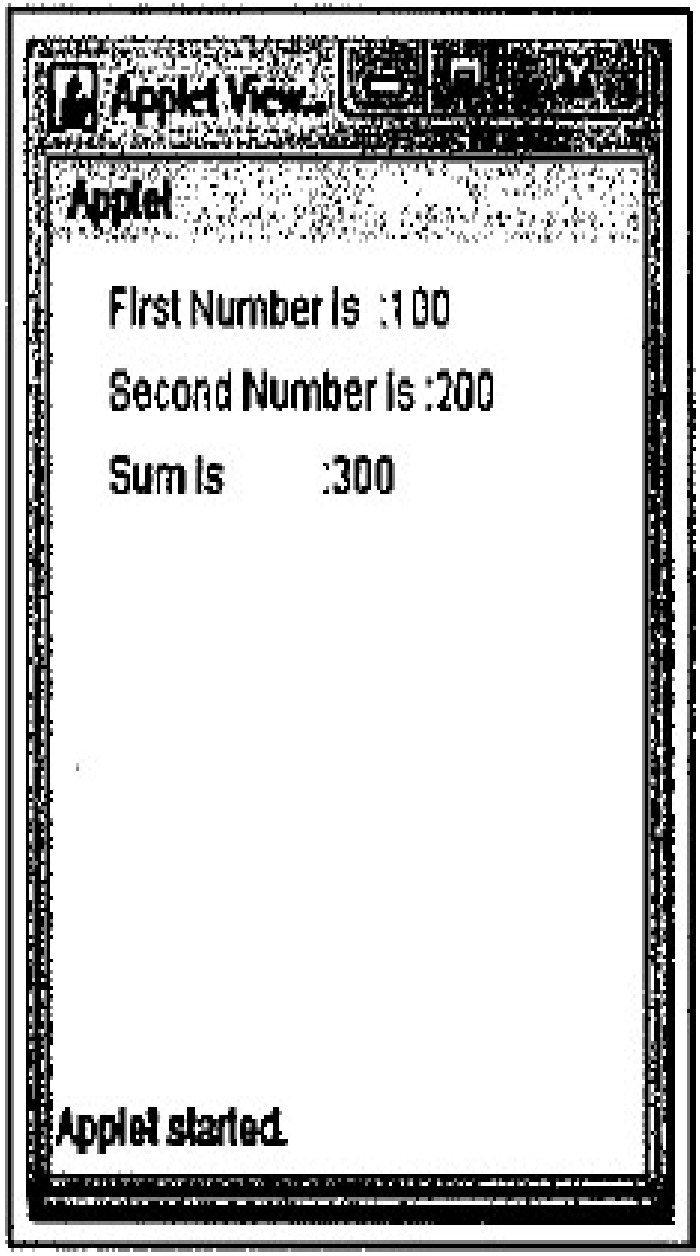
```
<HTML>
    <HEAD>
    <TITLE> Parameter Passing to an Applet </TITLE>
        <BODY>
            <Applet Code=MyFontApplet.class width=400 height=100 ALIGN=left>
                <PARAM NAME=font VALUE="TimesRoman">
                <PARAM NAME=size VALUE="36">
            </Applet>
            <br>The professional team of Skyward Publishers has entered into the market of</br>
            <br>computer books bringing excellent content.The team is committed to excellence<br>
            <br>excellence in quality of content; excellence in the dedication of its authors;<br>
            <br>excellence in the attention to detail; and excellence in understanding the<br>
            <br>needs of students.<br>
        </BODY>
    <HEAD>
</HTML>
```

DISPLAYING NUMERIC VALUE

We know to display the Strings in an applet using drawstring() method. To display, numeric values there is no method like drawInt(). First, we should convert the numeric values into String and use drawString() method to display.

```
import java.applet.Applet;
import java.awt.Graphics;

/* <Applet Code=NumericApplet.class width= 200 height=200>
   </Applet>
*/

public class NumericApplet extends Applet {

    public void paint(Graphics g){

        int first=100;
        int second=200;
        int sum=first+second;

        /* Convert int to Strings using String.valueOf() methods */
        String firstStr  = "First Number is  :"+String.valueOf(first);
        String secondStr = "Second Number is :"+String.valueOf(second);
        String sumStr = "Sum is :"+String.valueOf(sum);

        g.drawString(firstStr, 20, 20);
        g.drawString(secondStr, 20, 40);
        g.drawString(sumStr, 20, 60);
    }

}
```

Output

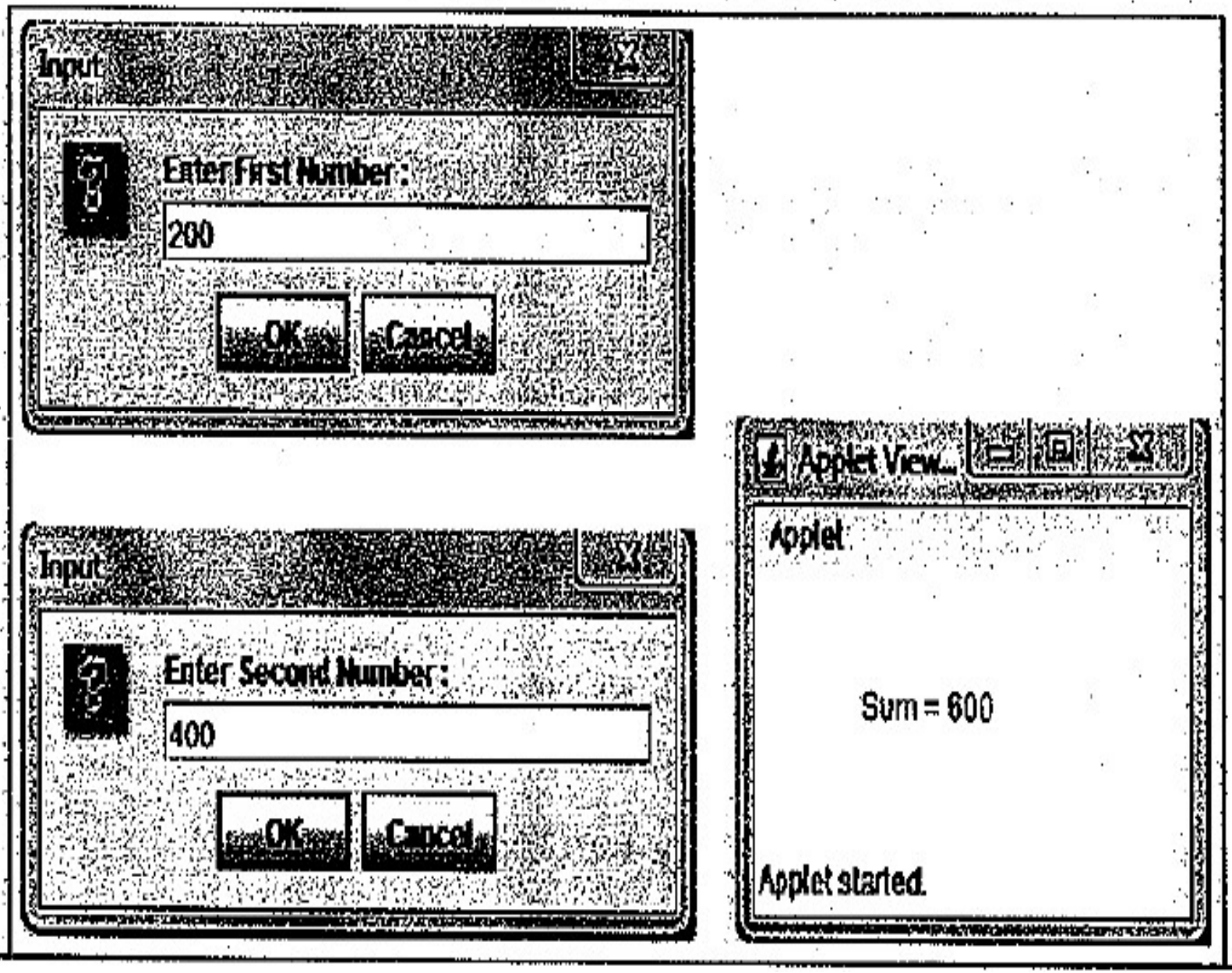

Applet Viewer
Applet
First Number is :100
Second Number is :200
Sum is :300
Applet started.

GETTING INPUT FROM USER

We can add any graphical user interface components like text box, button, checkbox etc., to an applet for developing interactive application. In applet, we can only display strings and we have seen how to convert numeric values to strings in the last section. In the below program, we have used *a JOptionPane* and it is part of the java swing library and it is used do a few things like entering data, display messages or a combination of those. We have used a method ***showInputDialog()*** of JOptionPane class to read the input data from the user. The data entered will be always Strings. If we want numeric values then we should convert it to type we wanted.

```
import java.awt.*;
import javax.swing.*;

/* <Applet Code=InputApplet.class width= 100 height=100>
</Applet>
*/

public class InputApplet extends JApplet {
    String firstStr;
    String secondStr;
    String sumStr;

    public void init() {

        firstStr = JOptionPane.showInputDialog("Enter First Number :" );
        secondStr = JOptionPane.showInputDialog("Enter Second Number :" );

        /* Both firstStr and SecoudStr are Strings. Convert to int and add the numbers
         * Store the result in integer variable sum
         */
        int sum=Integer.parseInt(firstStr)+Integer.parseInt(secondStr);

        /* Convert integer variable sum to String */
        sumStr="Sum = "+ String.valueOf(sum);
    }

    public void paint(Graphics g) {
        g.drawString(sumStr,50,150);
    }
}
```

Output


Input
Enter First Number :
200
OK
Cancel
Input
Enter Second Number :
400
OK
Cancel
Applet Viewer
Applet
Sum = 600
Applet started.

13.11 update() Method

update() method is called when applet has requested that a portion of its window be redrawn. The default version first fills an applet with default back ground color and then calls paint() method. If we fill background with a different color, user will experience a flash of default background each time update is called. To avoid this problem, we should override update method. So that it performs all necessary display activities. Then call update() in paint().

```
public void update(Graphics g)
{
}
public void paint(Graphics g)
{
   update(g);
}
```

13.12 repaint() Method

An applet writes to its window only when its update() or paint() method is called. Whenever our applet needs to update the information displayed in its window, it simply calls repaint().

void repaint() which causes window to be repainted.
void repaint(int left,int width,int height) specifies a region that will be repainted.

Advantages of Applet:

- Applets are cross platform and can run on Windows, Mac OS and Linux platform.
- Applets can work all the version of Java Plug-in.
- Applets runs in a sandbox, so the user does not need to trust the code, so it can work without security approval.
- Applets are supported by most web browsers.
- Applets are cached in most web browsers, so will be quick to load when returning to a web page.

Disadvantages of Applet

- Java plug-in is required to run applet
- Java applet requires JVM so first time it takes significant start-up time
- If applet is not already cached in the machine, it will be downloaded from internet and will take time
- Its difficult to design and build good user interface in applets compared to other technologies.

Applet Display Methods

- Applets are displayed in a window and they use the AWT to perform input and output.
- To output a string to an applet, drawString() which is a member of the ***Graphics*** class is used.
- An object of Graphics class provides the surface for painting.

  void drawString(string message, int x, int y)
- Here, message is the string to be output beginning at the coordinate (x,y).
- In a Java window, the upper-left corner is location (0,0).
- To set the background color of an applet's window, use setBackground().
- To set foreground color, use setForeground().

AppletContext and showDocument()

- The showDocument() method defined by the AppletContext interface is used to allow an applet to transfer control to another URL.
- AppletContext is an interface that lets the user to get the information from the applet's execution environment.
- Within an applet, once the user has obtained the applet's context, the user can bring another document into view by calling showDocument().
- This method has no return value and throws no exception if it fails.
- There are two showDocument() methods.
- The method showDocument(URL) displays the document at the specified URL.
- The method showDocument(URL, target) where target refers to the location where the document is to be displayed.

Method **Description**

- Applet getApplet(String appletName)
  - Returns the applet specified by appletName if it is within the current applet context.
- Enumeration getApplets()
  - Returns an enumeration that contains all the applets within the current applet context.
- Image getImage(URL url)
  - Returns an Image object that encapsulates the image found at the location specified by url.
- void showDocument(URL url)
  - Brings the document at the URL specified by url into view.
- void showStatus(String str)
  - Display string given in ‘str’ in the status window.

Example

```
import java.awt.Graphics;
import java.awt.*;
import java.applet.Applet;

/*<applet code= “Rec.class” width=250 height=250></applet>*/

public class Rec extends Applet
{
    public void paint(Graphics g)
    {
        g.drawLine(10,10,60,50);
        g.fillRect(100,10,60,50);
        g.drawRoundRect(190,10,60,50,15,15);
        g.fillRoundRect(70,90,140,100,30,40);
    }
}
```