



# Chapter 2- Basics of Java

By Biruk M.

# Outline

- All that is to know on Java basics
- Introduction & History
- Basics of Programming constructs
- Core Classes
- Arrays
- ArrayLists
- Methods
- Exercise

# 2.1 History of Java

## C++

- Evolved from C
- Provides object-oriented programming capabilities

## Objects

- Reusable software components that model real-world items

## Java

- Originally for intelligent consumer-electronic devices
- Then used for creating Web pages with *dynamic content*
- Now also used for:
  - Develop large-scale enterprise applications
  - Enhance WWW server functionality
  - Provide applications for consumer devices (cell phones, etc.)

# Cont…

- Java is a high level language.
- 1991 – Sun Microsystems initiates project “Green” with the intent to develop a programming language for digitally controlled consumer devices and computers.
- The language OAK was developed by James Gosling with the goal of using C++’s popularity.
- OAK was later renamed to JAVA and released in 1995.
- Can be used to create two types of programs: applications and applets.
- Robust: no pointers in java and no manual memory handling.

# 2.2 Java Class Libraries

## Classes

- Contain *methods* that perform tasks
  - Return information after task completion
- Used to build Java programs

## Java systems contain

- Environment
- Language
- Class libraries(APIs- Application Programming Interfaces)

## Java APIs

- provides an entire framework in which Java developers can work to achieve true reusability and rapid application development

# cnt’d

- **Powerful**: massive libraries.
- Java programs are portable.
- Java promise: “Write once, run everywhere”.
- Java differs from other programming languages in that it is both compiled and interpreted language.
- Java compilers produce the Java bytecode.
- Java bytecodes are a set of instructions written for a hypothetical computer, known as the Java virtual machine.
- Bytecodes are platform-independent.
- The JVM is an interpreter for bytecode.

# cnt’d

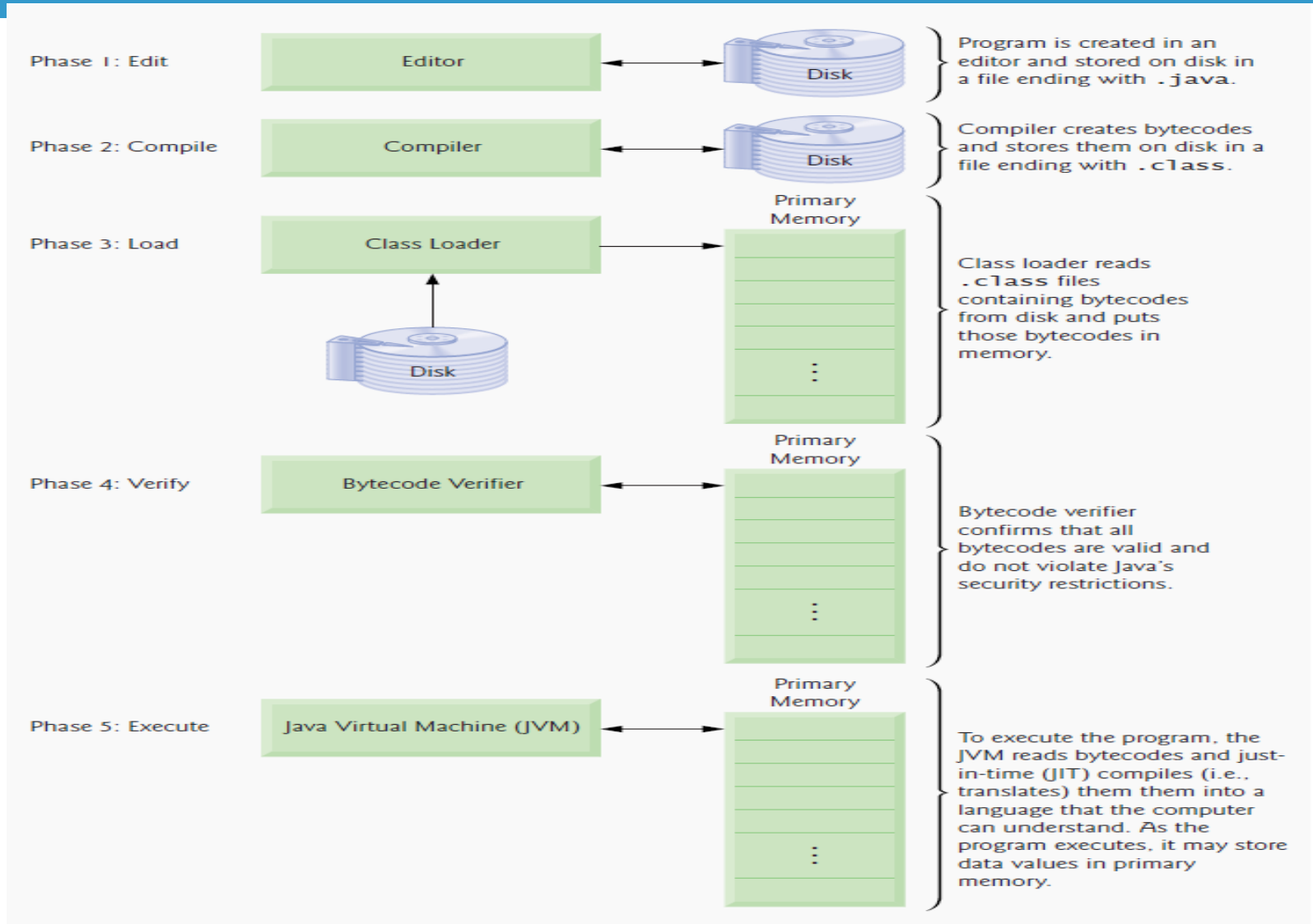
- An interpreter reads the byte code and translates into a sequence of commands that can be directly executed by the computer.
- Because the execution of every Java program is under the control of the JVM, the JVM can contain the program and prevent it from generating side effects outside of the system.
- The JVM must locate classes just as the compiler does

# 2.3 Basics of a Typical Java Environment

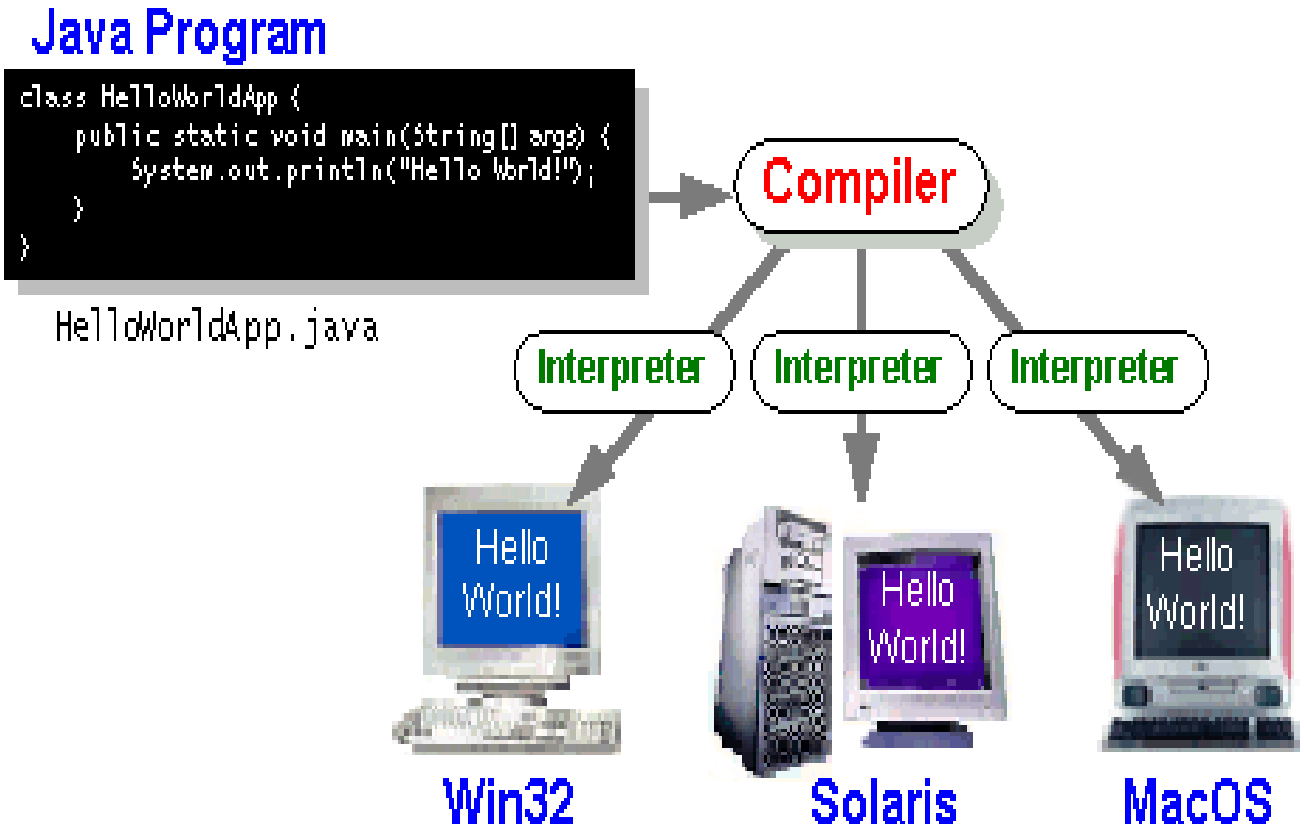
Java programs normally undergo five phases

- Edit
  - Programmer writes program (and stores program on disk)
- Compile
  - Compiler creates *bytecodes* from program
- Load
  - Class loader stores bytecodes in memory
- Verify
  - Verifier ensures bytecodes do not violate security requirements
- Execute
  - Interpreter translates bytecodes into machine language

# cnt'd


Phase 1: Edit
Editor
Disk
Program is created in an editor and stored on disk in a file ending with .java.
Phase 2: Compile
Compiler
Disk
Compiler creates bytecodes and stores them on disk in a file ending with .class.
Phase 3: Load
Class Loader
Disk
Primary Memory
Class loader reads .class files containing bytecodes from disk and puts those bytecodes in memory.
Phase 4: Verify
Bytecode Verifier
Primary Memory
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.
Phase 5: Execute
Java Virtual Machine (JVM)
Primary Memory
To execute the program, the JVM reads bytecodes and just-in-time (JIT) compiles (i.e., translates) them them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

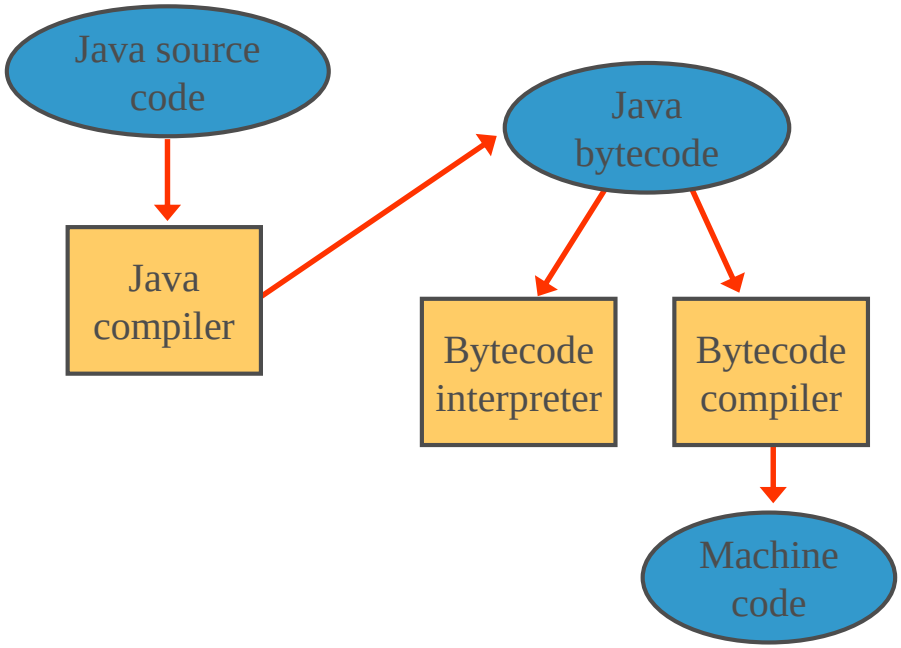
# cnt'd


Java Program
class HelloWorldApp {
public static void main(String[] args) {
System.out.println("Hello World!");
}
}
HelloWorldApp.java
Compiler
Interpreter
Interpreter
Interpreter
Hello World!
Hello World!
Hello World!
Win32
Solaris
MacOS

# Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *architecture-neutral*

1-11


Java source code
Java compiler
Java bytecode
Bytecode interpreter
Bytecode compiler
Machine code

# Development Environments

There are many programs that support the development of Java software, including:

- Sun Java Development Kit (JDK)
- Sun NetBeans
- IBM Eclipse
- Borland JBuilder
- MetroWerks CodeWarrior
- BlueJ
- jGRASP

Though the details of these environments differ, the basic compilation and execution process is essentially the same

# Syntax & Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

# Errors

A program can have three types of errors

- The compiler will find syntax errors and other basic problems (*compile-time errors*)
- If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

# Java Program Structure

In the Java programming language:

- A program is made up of one or more *classes*
- A class contains one or more *methods*
- A method contains program *statements*
- A Java application always contains a method called `main`
- A Java applet doesn't need  a method called `main`

# Structure of simple program

A Java program has the following structure

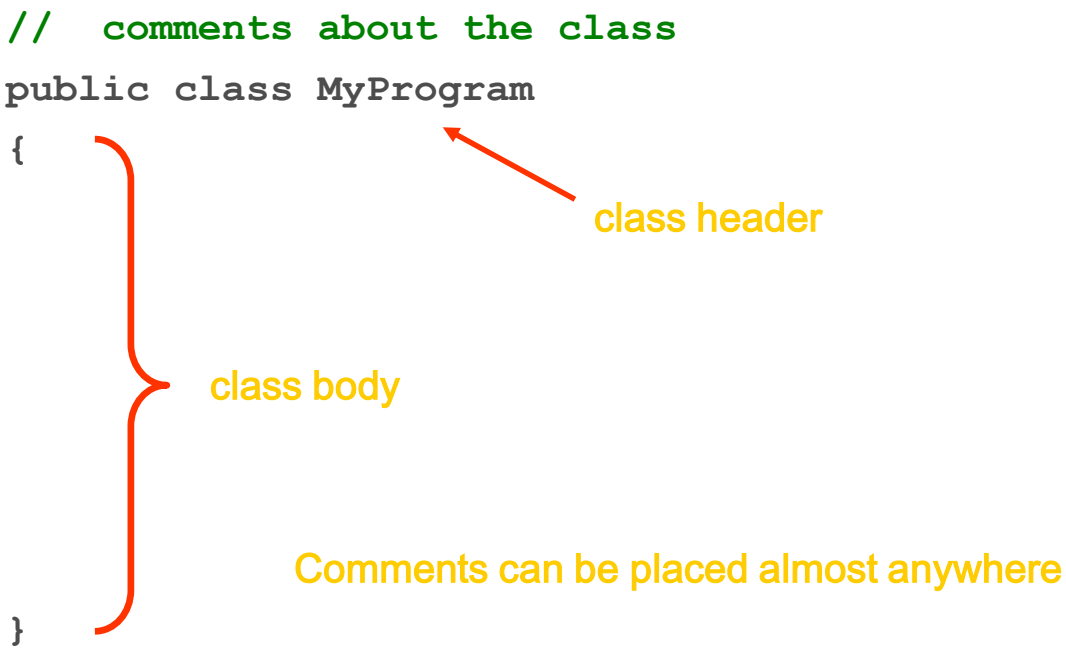
[Comments]

[imported packages]

[Global variable declarations]

[Class Definition]

[Definitions of methods]


// comments about the class
public class MyProgram
{
class header
class body
Comments can be placed almost anywhere
}

```
//  comments about the class
public class MyProgram
{
   //  comments about the method
   public static void main (String[] args)
   {

   }

}
```

# Example:

```
public class Eg
{
   //-------------------------------------------------------------
   //  Prints one's message;
   //-------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("A quote by Mr X:");

      System.out.println ("Whatever you are, be a good one.");
   }
}
```

# Basic Program Constructs

- **Comments:**
- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

1-21

# Keywords

The Java reserved words:

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

1-22

# White Spaces

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many ways
- **Programs should be formatted to enhance readability, using consistent indentation**

# Escape Sequences

- What if we wanted to print a the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println ("I said \"Hello\" to you.");
```

# Some Java escape sequences:

| Escape Sequence | Meaning |
|---|---|
| `\b` | backspace |
| `\t` | tab |
| `\n` | newline |
| `\r` | carriage return |
| `\"` | double quote |
| `\'` | single quote |
| `\\` | backslash |

# Character Strings

A string of characters can be represented as a *string literal* by putting double quotes around the text:

Examples:

```
"This is a string literal."

"123 Main Street"

"X"
```

Every character string is an object in Java, defined by the `String` class

Every string literal represents a `String` object

1-26

# The println Method

We can invoke the `println` method to print a character string

The `System.out` object represents a destination (the monitor screen) to which we can send output

```
System.out.println ("Whatever you are, be a good one.");
```

# The print Method

The `System.out` object provides another service as well

The `print` method is similar to the `println` method, except that it does not advance to the next line

Therefore anything printed after a `print` statement will appear on the same line

# String Concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another
  - `"Peanut butter " + "and jelly"`
- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program
- The + operator is also used for arithmetic addition
- The function that it performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right, but parentheses can be used to force the order

# Addition.java

```
public class Addition
{
   //-----------------------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("24 and 45 concatenated: " + 24 + 45);

      System.out.println ("24 and 45 added: " + (24 + 45));
   }
}
```

# Data types:

Several other variable types are built into Java They can be conveniently classified as ***integer***, ***floating-point,double, short long*** or ***character*** variables. – Primitive standard types

- Floating-point variable types can be expressed as fraction i.e. they are “real numbers”.
- Character variables hold a two byte. They are used to hold 256 different characters and symbols of the ASCII and extended ASCII character sets.
- String: eg: string x; (data type for array of characters)

# cnt'd…

Besides reference types, there are eight other types in Java. These are called *primitive types,* to distinguish them from reference types - to store the locations of objects in the computer's memory.

Their names and values are:

| | |
|---|---|
| `boolean` | either `false` or `true` |
| `char` | 16-bit Unicode characters |
| `byte` | 8-bit whole numbers: integers ranging from –128 to 127 |
| `short` | 16-bit whole numbers: integers ranging from –32,768 to 32,767 |
| `int` | 32-bit whole numbers: integers ranging from –2,147,483,648 to 2,147,483,647 |
| `long` | 64-bit whole numbers: integers ranging from ±9,223,372,036,854,775,807 |
| `float` | 32-bit decimal numbers: rationals ranging from $\pm 1.4 \times 10^{-45}$ to $\pm 3.4 \times 10^{-38}$ |
| `double` | 64-bit decimal numbers: rationals ranging from $\pm 4.9 \times 10^{-324}$ to $\pm 1.8 \times 10^{-308}$ |

# Simple Type Conversion/Casting

A value in any of the built-in types we have seen so far can be converted (type-cast) to any of the other types.

For example: (int) 3.14 // converts 3.14 to an int 3

(double) 2// converts 2 to a double to give 2.0

(string) 122 // converts 122 to a string whose code=122

- ***Integer division always results in an integer outcome.***
- ***Division of integer by integer will not round off to the next integer***
- E.g.: 9/2 gives 4 not 4.5
- E.g 8.2/2=4.1 implicit conversion/promotion

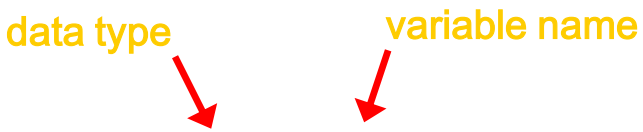
# Identifiers

- An identifier is name associated with a function or data object and used to refer to that function or data object. An identifier must:
- Start with a letter or underscore or $
- Consist only of letters, the digits 0-9, or the underscore symbol _ or $
- Identifiers cannot begin with a digit
- Java is *case sensitive* - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
- *title case* for class names - `Color`
- *upper case* for constants - `MAXIMUM`

# Variables

A *variable* is a name for a location in memory

A variable must be *declared* by specifying the variable's name and the type of information that it will hold




```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

1-35

# Variable Initialization

A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- **When a variable is referenced in a program, its current value is used**

# Assignment

An *assignment statement* changes the value of a variable

The assignment operator is the = sign

```
total = 55;
```

- **The expression on the right is evaluated and the result is stored in the variable on the left**
- **The value that was in `total` is overwritten**
- **You can only assign a value to a variable that is consistent with the variable's declared type**

**Eg:** **int sides = 7; // declaration with initialization**

**sides = 10; // assignment statement**

# Constants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- **Though const is a reserved word in Java it's actually not in use!**

  **However the final keyword let's you define constants and const variables**
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

# Constants

Constants are useful for three important reasons

- First, they give meaning to otherwise unclear literal values
  - For example, MAX_LOAD means more than the literal 250
- Second, they facilitate program maintenance
  - If a constant is used in multiple places, its value need only be updated in one place
- Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

1-39

# Operators

An operator is a symbol that makes the machine to take an action.

Different Operators act on one or more operands and can also have different kinds of operators.

C++ provides several categories of operators, including the following:

- Assignment operator  eg: x=3;or x+=3;
- Arithmetic operator  (+,-,*,/,%)
- Relational operator (>,<,>=,<=,==,!=)
- Increment/decrement operator

# Cont…

E.g. int k = 5;

(auto increment prefix) y= ++k + 10; //y=

(auto increment postfix) y= k++ + 10; //y=

(auto decrement prefix) y= --k + 10; //y=

(auto decrement postfix) y= k-- + 10; //y=

Operator Precedence

| Operators | Associativity | Type |
|---|---|---|
| ( ) | left to right | parentheses |
| * / % | left to right | multiplicative |
| + - | left to right | additive |
| < <= > >= | left to right | relational |
| == != | left to right | equality |
| = | right to left | assignment |

# Basic Classes

1. **System:** one of the core classes the easiest way to display information

- System class is final and all of it's members and methods are **static**
- System.out.println( "Welcome to Java!" );
- System.out.print("…");
- System.exit(0);

**New line usage**

- System.out.println( "Welcome**\nto\nJava\n**Programming!" );

# Cont…

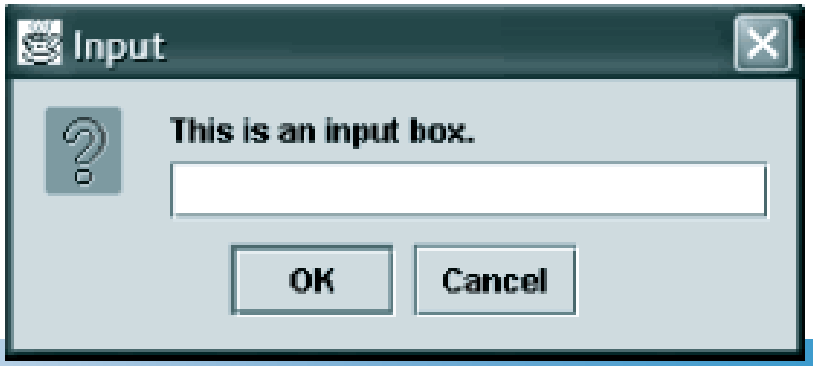
2. Scanner:

- The **java.util.Scanner** class is a simple text scanner which can parse primitive types and strings using regular expressions.
- Methods: next(), nextInt()…
- Eg:- Scanner s=new Scanner (System.in);

  int x=s.nextInt();

**3. JOptionPane**

- The JOptionPane class provides static methods to display each type of dialog box.

# Cont…

**Eg:**

- String x =JOptionPane.showInputDialog( "Enter first integer" );
- JOptionPane.showMessageDialog( null, “result" + sum, “Title",ICON-sign); **ICON-sign**: JOptionPane.PLAIN_MESSAGE

The following statement must be before the program’s class header(tells the compiler where to find the JOptionPane class)

```
import javax.swing.JOptionPane;
```

A program that uses **JOptionPane does not automatically stop executing** when the end of the main method is reached.

Java generates a *thread*, which is a process running in the computer, when a JOptionPane is created.

If the System.exit method is not called, this thread continues to execute.

# Examples:

```
Import java.util.Scanner;
int number1,number2,sum;

Scanner s=new Scanner(System.in);

number1 = s.nextInt();
number2 = s.nextInt();

 sum = number1 + number2;

System.out.println(“Result=”+sum);
```

# Examples:

```
import javax.swing.JOptionPane;

String firstNumber,secondNumber

 int number1,number2,sum

firstNumber =JOptionPane.showInputDialog( "Enter first integer" );

secondNumber = JOptionPane.showInputDialog( "Enter second integer"
    );

number1 = Integer.parseInt( firstNumber );

number2 = Integer.parseInt( secondNumber );

 sum = number1 + number2;


JOptionPane.showMessageDialog( null, "The sum is " + sum,
    "Results", JOptionPane.PLAIN_MESSAGE );

System.exit( 0 );
```

# 4. Type-classes

Eg: Integer, String, Double…

Works with Parse-methods:

- Parse methods convert strings to numeric data types

String x; int y,z;

- Integer.parseInt(x)
- Double.parseDouble(x)
- Integer.*toString(y);*
- **Integer.*compare(y,z)*;-comparison**
- **Integer.*max(y,z);***

***String; charAt(x)***

- String name;
- **for(i=0;i<name.length();i++)**
- **System.out.println(name.charAt(i));**

# 5. Math-class

The **java.lang.Math** class contains methods for performing basic numeric operations such as the **elementary exponential, logarithm, square root, and trigonometric functions**

- **Eg: Math.ceil(x)**
- **Math.floor(x)**
- **Math.PI;**
- **Math.sin(x);**
- **Math.pow(x,y), Math.round(x)**
- **Math.max(x,y) ,Math.toDegrees(x);**
- **Math.random():** a random floating point number between 0 and 1. generates a double value in the range [0,1)
- Eg: Math.random()*100

# Cont…

In order to produce random "integer" values within a specified range,

- you need to manipulate the random( ) method.
- Obviously, we will need to cast our value to force it to become an integer.

The formula is:

- int x = (int) (min + Math.random( ) * (Max - Min + 1));
- Eg: random numbers between 1 to 10
  - Int x= (int) (1 + Math.random( ) * (10 - 1 + 1));
  - Int x= (int) (1 + Math.random( ) * 10);
  - Int x=(int) 1+(0.63*10)=7

# Exercises

1. Addition of two numbers using Scanner
2. Addition of two numbers using JOptionPane
3. Checking the number even or not…using if & Scanner
4. Displaying all squares of 1 to 10..using do while & JOptionPane
5. Insert an integer N and check whether it is perfect square or not
6. Insert integer N and display only the perfect squares from 1 to N using JOptionPane

# 6. Random

Random is a class used to generate random numbers between the given lists/ranges.

Example:

```
public class RANDOM1 {
    public static void main(String[] args) {
        double x []=new double[20];
        Random r=new Random();
                    int sum=0;
        Scanner s=new Scanner(System.in);
                for(int i=0;i<5;i++)
                {x[i]=r.nextInt(100);//nextDouble()*100
//it excludes the top-value so we add 1
//int randomNum = rand.nextInt((max - min) + 1) + min;
                    System.out.println(x[i]);
```

# Basic Control Structures

## 1. Conditional Statements

- Conditional statements **like if, if-else, if-else-if and switch** can also be used in java as of their syntax

**Example of if-statement**

```
public class Even
{
public static void main(String args[] )
{
int x;
for (x=0;x<=10;x++)
if(x%2==0)
System.out.println("evens="+x);
}}
```

# 2. Looping structure

Do-while, for and while repetition statements are also applicable in java

Eg:-displaying all squares of numbers 1 to 10

```
public static void main(String args[])
   { String output=" ,";
       Int   n=10;
System.out.println("the output is=");
Int i=1;
      do {
            output +=i*I;
          i++;
       }
while(i<=n);
```

# For-each

Advanced or Enhanced For loop

The for-each loop introduced in Java5. It is mainly used to traverse **array or collection elements**.

Advantage of for-each loop:

- It makes the code more readable.
- It elimnates the possibility of programming errors.

Syntax :

- **for**(data_type variable : array | collection){}

**Eg:-**

```
int arr[]={12,13,14,44};
   for(int i:arr){
      System.out.println(i);
```

# 3. Branching statement

**Reading Assignment** : How do we use the following Branching statements in Java?

- **Continue, break, goto**

# ???

# Arrays

**A static data structure-**

**Sysntax:**

- **Data type arrayname[]=new dataType[size];**

**Int x[]=new int[20];**

**Example-**

```
import java.util.*;
public class array1 {
 public static void main(String[] args) {
        int x []=new int[20];//array declaration
        int sum=0;
```

# Cont…

```
Scanner s=new Scanner(System.in);
        System.out.println("enter the numbers");
                for(int i=0;i<5;i++)
                        {       x[i]=s.nextInt();
                        System.out.println(x[i]);
                    sum=sum+x[i];
                        }
System.out.println("result="+sum);
}}
```

# Array-Lists

- Arraylist is a class which implements List interface. It is widely used because of the functionality and flexibility it offers.
- Most of the developers **choose Arraylist over Array** as it's a very good alternative of traditional java arrays.
- ArrayList can dynamically grow and shrink as per the need. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy
- Syntax for string list
- **ArrayList<String> obj = new ArrayList<String>();**

# Example

```
import java.util.*;
public class EGARRAYLIST {
 public static void main(String args[]){
        int sum=0;
    ArrayList<Integer> x=new ArrayList<Integer>();
Scanner s=new Scanner(System.in);
        System.out.println("enter the numbers");
        for(int i=0;i<5;i++){
                x.add(s.nextInt());
                sum=sum+x.get(i);}
```

# Cont…

- System.***out***.print("res===,"+sum);
- **for**(**int** i=0;i<5;i++)
- {  //x.add(i);
- System.***out***.println(x.get(i));
- }}
- }
- **Eg-2**

```
ArrayList<String> list=new ArrayList<String>();
 list.add(“banana");
 list.add(“Apple");
 list.add(“orange");
 for(String s:list){
   System.out.println(s);
```

# Exercise(ArrayList)

1. Insert N-integers from the keyboard and calculate the sum and average (Arrays & For-Each)
2. Insert a string using Scanner and count the number of ‘a’ in the string

# Exercise(ArrayList)

1. Generate 5 random numbers between 10 and 20 and find the smallest even of all(ArrayList )
2. Insert a string and sort the characters in ASC order
3. Check if the inserted string is palindrome or not
4. Insert list of names and particular searching string (key) and check if the key is found or not
5. Write a java program that inserts 5 integers and display
    1. Find the smallest
    2. Find only evens
    3. Count only integers>50

# Exercise(ArrayList)

1. Generate 5 random numbers between 2 and 50 and find the largest two (ArrayList)
2. Check if the inserted string is a palindrome or not
3. Insert list of names and particular searching string (key) and check if the key is found or not
4. Generate 10 random numbers between 2 and 50 and display only the perfect square numbers(use ArrayList)

# Methods

- Methods :a subprogram/set of codes that acts on data and often returns a value.
- In java, we have two methods:
- User defined
  - Methods also defined outside main and called in the main whenever necessary. Eg:- public static int sum(int a,int b)
- Buit-in: available methods in API . Eg: nextInt(),

## Passing by value and by reference?

- **Passing by value**
  - When an argument is passed by value, a *copy* of the argument's *value* is passed to the called method.
  - The called method works exclusively with the copy

# example

```
public static void main(String[] args) {
        int x = 1;
        System.out.println ("Before the call, x is " + x);
        square (x);
        System.out.println ("After the call, x is " + x);
    }
    public static void square (int y) {
        y++;
        System.out.println (“y inside the method is " + (y*y));
    }
```

**Output?**

## Passing by Reference

- When an argument is passed by reference, the called method can access the argument's value in the caller directly and modify that data, if necessary.
- Pass-by-reference improves performance by eliminating the need to copy possibly large amounts of data.

Java does *not* allow you to use pointers so we have only calling by value. However we can reference the actual data in array.

# example

```
public static void main(String[] args) {
        int[] array = {10, 20, 30, 40, 50}
 System.out.println("The values array before modify are: ");
  for (int value : array) {
          System.out.print(value + "  ");
        }
        sqaure(array); // pass array reference
        System.out.println("\nThe values of the modified array are: ");
        for (int value : array) {
          System.out.print(value + "  ");
        }
    }
 public static void modifyArray(int[] array2) {
for (int i = 0; i< array2.length; i++)
  {           array2[i] *= 2;
                }
```

# Methods-User defined

- Methods also defined outside main and called in the main whenever necessary.
- Static methods & Member methods(more on chapter 3)

**Example:**

```
import java.util.Scanner;
public class c1 {
    public static void main(String[] args)
    {
          msg();
          c1.sum();
         }
```

# Cont…

```
public static void msg()
{System.out.println("hello");
}
public  static void sum()
{    int x,y,r;
     Scanner s=new Scanner (System.in);
     System.out.println ("enter...");
     x=s.nextInt();  y=s.nextInt();
     r=x+y;
     System.out.println("res="+r);}
```

# Method-Exercises

..

# Questions?

That concludes this chapter

# Thank You

Biruk M.