Programming in Java

Dr. Nyein Aye Maung Maung
Dr. Eng (Ritsumeikan University, Japan)
Lecturer
Computer Engineering and Information Technology Dept.
Yangon Technological University

Basic Course Information

- Course Title
  - Programming in Java
- Textbook and Reference materials
  - ***Java™2: The Complete Reference***, Seventh Edition by Herbert Schildt
  - ***Introduction to Java programming:*** comprehensive version by Y. Daniel Liang, 9th edition
- Course Duration
  - 14 Weeks

Course Objectives

- To understand basic principles of the object oriented development process and apply this understanding to the analysis and design of solutions for small scale problems
- To have competence in the use of the Java programming language in the development of small to medium sized application programs
- To have skills to demonstrate knowledge of basic algorithms and data structures, in particular the application of recursive algorithms, simple searching and sorting algorithms and simple linked list structures
- To demonstrate their ability to work on larger, more complex projects by collaboratively designing and then individually implementing applications, in whole or in part, that exercise their competency with the core facilities of the Java APIs for feature-rich GUIs, including components that distribute processing using the event-driven programming, multithreading, networking or JDBC APIs

Course Schedule

Week	Topics
Week 1	Overview of JAVA, Data Types, Variables and Arrays
Week 2	Operators and Control Statements
Week 3	Classes and A closer look at Methods and Classes
Week 4	Inheritance, Overriding and Polymorphism
Week 5	Interfaces and Packages
Week 6	Exception Handling and Multi-threaded Programming
Week 7	String Handling
Week 8	Exploring java.lang and More utilities classes
Week 9	Java Collections Framework
Week 10	Java I/O
Week 11	Basic Graphical User Interface
Week 12	Event Handling
Week 13	Database Programming
Week 14	Applet and Networking

Exam and Grading System

Task	Mark
Final Exam	80%
Practical Assignments	20%

Mark	Grade
85~100	A+
80~85	A
70~80	B
50~70	C
<50	F

Lecture 1

Overview of Java, Variables, Data Types and Arrays

Outline of Class (Lecture 1)

- Overview of Java
- Data types
- Variables
- Arrays

Lecture Objectives

- To introduce
  - what Java is
  - How Java is used for
  - Features of Java
  - Components in Java
- To write a simple Java program
- To explain the basic syntax of a Java program
- To create, compile, and run Java programs
- To explore Java numeric primitive data types
- To use variables to store data
- To describe why arrays are necessary in programming and to apply arrays in application development

Topic 1

Overview of Java

What is Java?

- A programming language and a computing platform to develop applications
- An object-oriented language similar to C++ with advanced and simplified features
- Platform independent, “Write once, Run anywhere”
- Developed by Sun Microsystems
- Designed for easy Web/Internet applications
- Widespread acceptance
- Derive much characters from C and C++

Features of Java

Remove all the complexities such as pointers, operator overloading in C++

Platform independent

Everything is considered to be an “object”

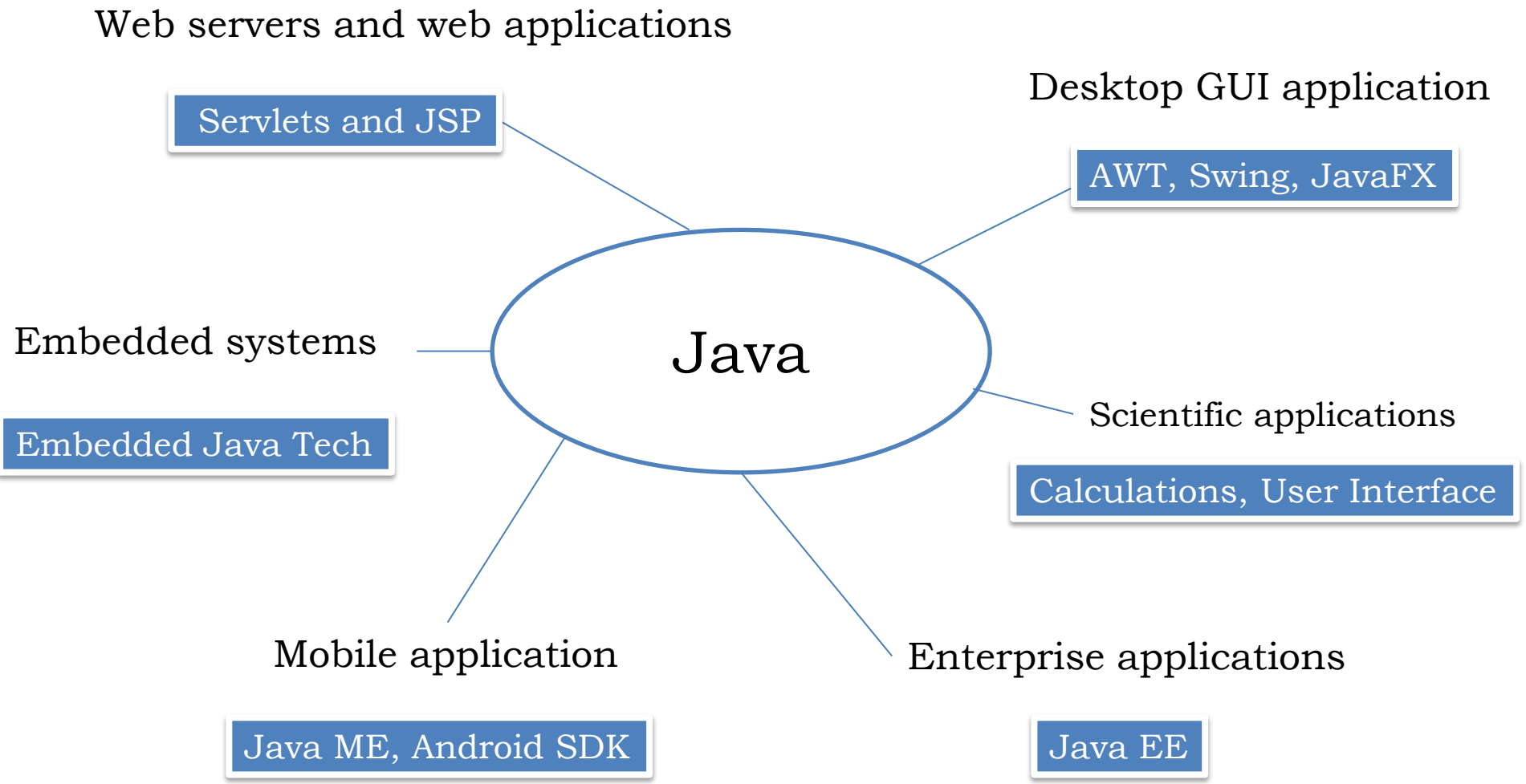
Use byte-codes

Use Remote Method Invocation (RMI)

The use of JIT (Just-In-Time) compilers, Java enables high performance

Support multiple threads of execution, including a set of synchronization primitives

Applications of Java

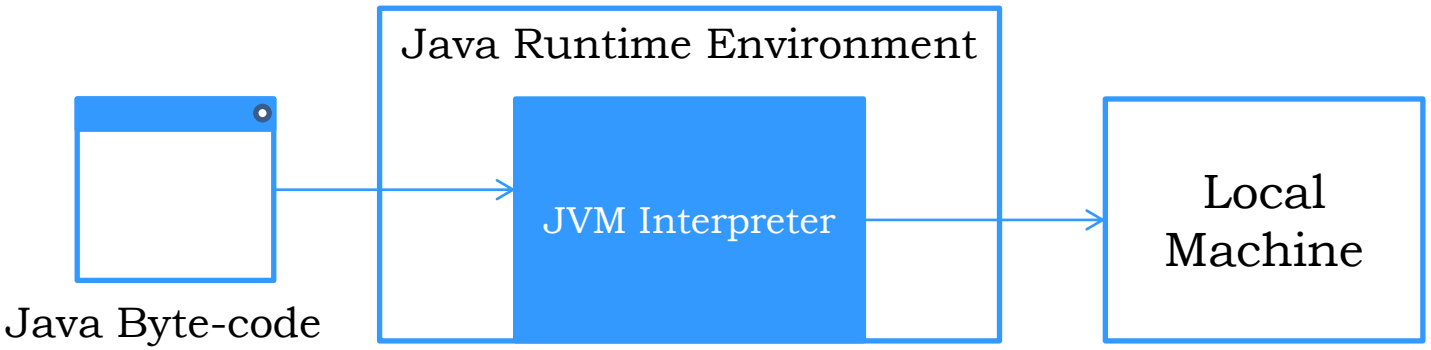


Java Platform

- The computing platform from Oracle that helps users to *run* and *develop* Java applications
- Consists of two essential parts
  1. **Java Runtime Environment (JRE)**, which is needed to *run* Java appli cations and applets; and,
  2. **Java Development Kit (JDK)**, which is needed to *develop* those Java applications and applets

1. Java Runtime Environment (JRE)

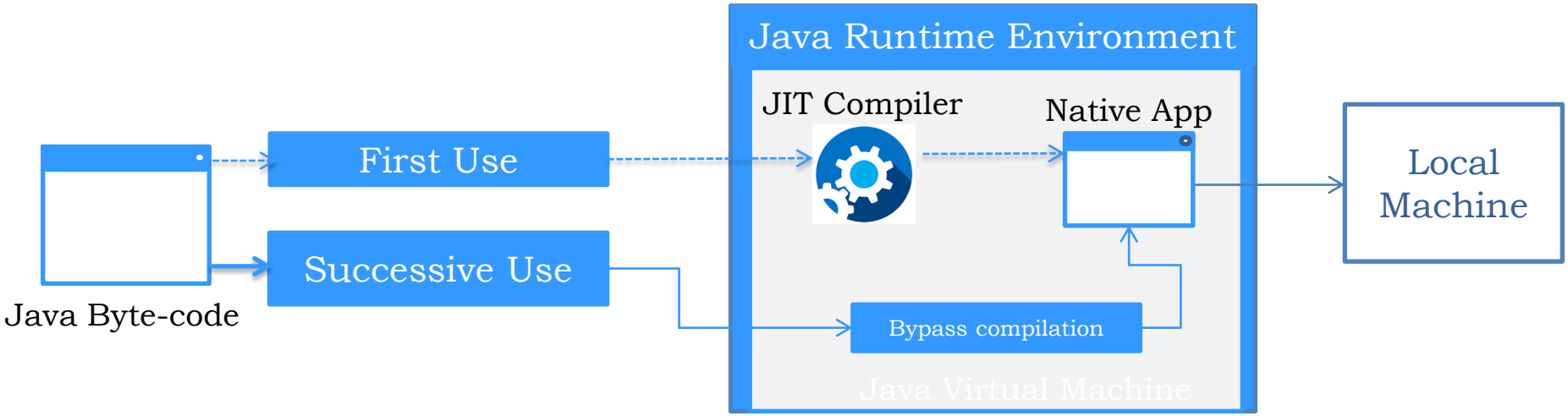
- When a Java application is run, the JRE acts as a liaison between the underlying platform and that application
- It interprets the Java application to run in accordance with the underlying platform
- The part of the JRE that accomplishes this complex liaison agreement is called the Java Virtual Machine (JVM)
- **Java Virtual Machine (JVM)**
  - The JVM acts like a virtual processor, enabling Java applications to be run on the local system. Its main purpose is to interpret (read translate) the received byte-code and make it appear as native code



1. Java Runtime Environment (JRE) (cont')

– Just-in-time Compiler (JIT Compiler)

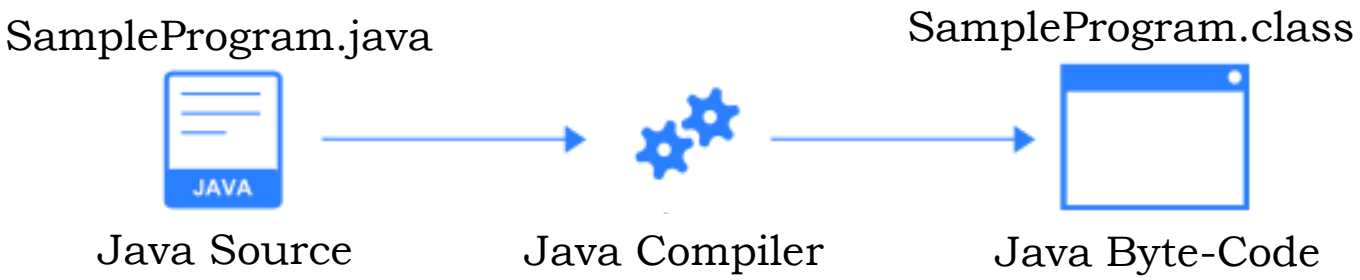
- A more robust JVM introduced since version 1.2
- Instead of interpreting byte-code, it down-right converts the code straight into equivalent native code for the local system, called JIT-compilation.
- This process only occurs when the byte-code is executed for the first time.
- Unless the byte-code itself is changed, the JVM uses the compiled version of the byte-code on every successive execution.
- Saves a lot of time and processor effort, allowing applications to execute much faster at the cost of a small delay on first execution.



2. Java Development Kit (JDK)

- For writing pure code in Java which can then be converted into Java byte-code for mass deployment
- A tool to compile, document and package java programs.
- **Java Compiler**
  - The Java compiler tool (named **javac** in the JDK) is the most important utility found with the JDK to compile Java Source files to Java Byte-code.
  - This process of converting Java source files into their equivalent byte-codes is known as compilation.
  - e.g,

javac SampleProgram.java → **SampleProgram.class**



Major Editions of Java

- **Java 2 Standard Edition (J2SE**): This is the most commonly used form of Java technology. Also referred to as Java Development Kit (JDK), J2SE is a software development kit that includes a set of APIs, tools, and a runtime for developing general-purpose applications.
- **Java 2 Enterprise Edition (J2EE)**: J2EE is the basis for Java-based enterprise applications which specifies a standard for developing multi-tier distributed component-based applications. It provides an integrated technical framework for developing Internet-enabled enterprise, and e-commerce applications.
- **Java 2 Micro Edition (J2ME)**: J2ME provides a lightweight and optimized Java environment for developing applications for mobile devices.

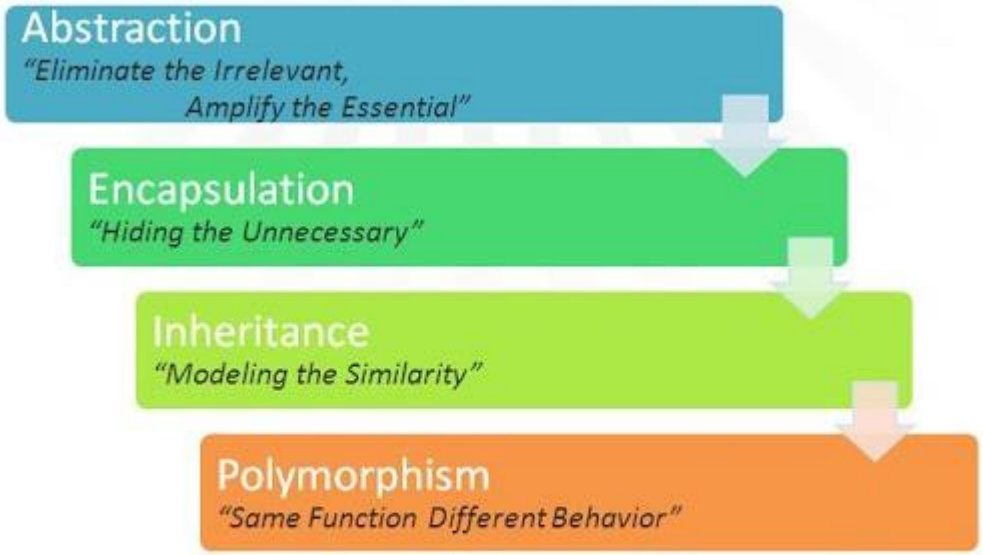
Four Main OOP Principles

(1) Abstraction

(2) Encapsulation

(3) Inheritance and

(4) Polymorphism



(1) Abstraction

- Humans manage complexity through abstraction.
  - For example, people do not think of a car as a set of tens of thousands of individual parts. They think of it as a well-defined object with its own unique behavior. They can ignore the details of how the engine, transmission, and braking systems work. Instead, they are free to utilize the object as a whole.
- Can achieve abstraction in two ways:

a) Abstract Class

- Contain the 'abstract' keyword.
- If a class is declared abstract, it cannot be instantiated, which means you cannot create an object of an abstract class.
- Also, an abstract class can contain abstract as well as concrete methods.

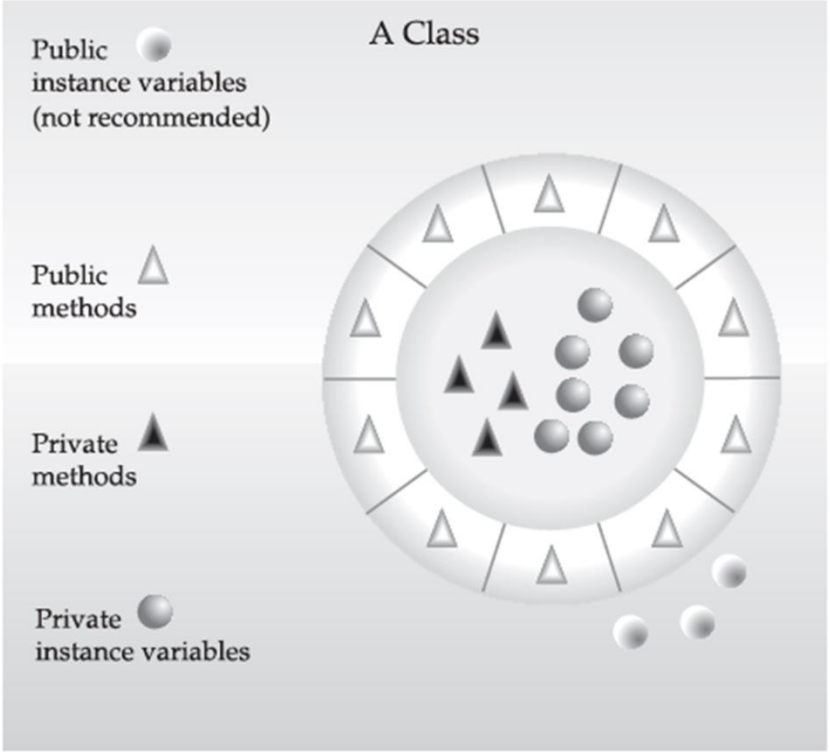
b) Interface

- A Java interface is a bit like a class, except a Java interface can only contain method signatures and fields.
- An Java interface cannot contain an implementation of the methods, only the signature (name, parameters and exceptions) of the method

(2) Encapsulation

- The mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse

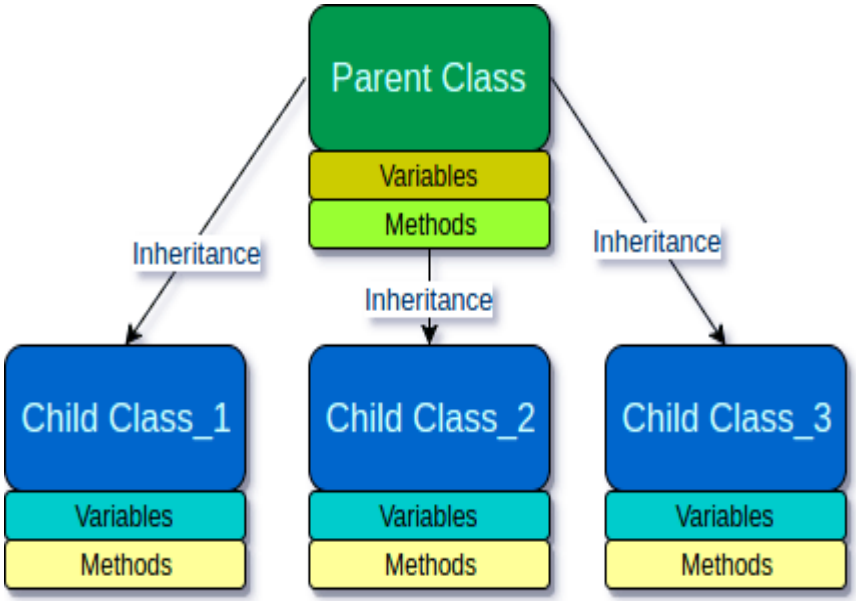
- Achieve encapsulation in Java by:
  - Declaring the variables of a class as private.
  - Providing public setter and getter methods to modify and view the variables values.

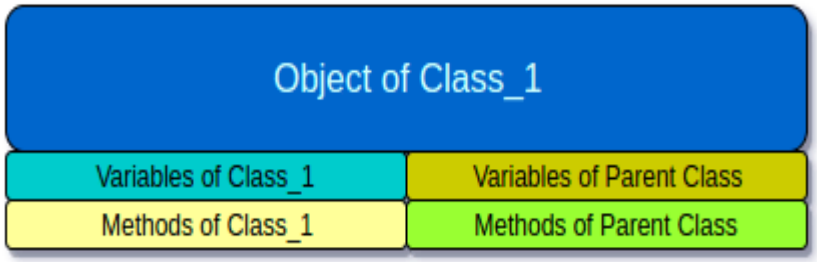


Encapsulation: public methods can be used to protect private data

(3) Inheritance

- Can be defined as the process where one class acquires the properties (methods and fields) of another .
- It allows the reuse of the members of a class (called the superclass or the parent class) in another class (called subclass, child class or the derived class) that inherits from it.
- This way, classes can be built by successive inheritance.
- It helps to reuse the code and establish a relationship between different classes.





(4) Polymorphism

- Polymorphism is the ability of an object to take on many forms.
- The same method will behave differently when it is applied to the objects of different classes
- It also means that different methods associated with different classes can interpret the same message in different ways.
- Example: an object can send a message PRINT to several objects, and each one will use it's own PRINT method to execute the message.

Basic Syntax of Java

- **Case Sensitivity** − Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Eg. *class MyFirstJavaClass*
- **Method Names** − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Eg. *public void myMethodName()*
- **Program File Name** − Name of the program file should exactly match the class name.(if the file name and the class name do not match, your program will not compile). E.g Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*
- **public static void main(String args[])** − Java program processing starts from the main() method which is a mandatory part of every Java program.

A SIMPLE JAVA PROGRAM

Basic syntax of a Java Class

```
< modifier> class < name>
{
        < attribute_declaration>*
        < constructor_declaration>*
        < method_declaration>*
}
```

A SIMPLE JAVA PROGRAM

```
/*
This is a simple Java program.
Call this file "Example.java".
*/
class Example {
// Your program begins with a call to main()
public static void main(String args[]) {
System.out.println("This is a simple Java program.");
}
}
```

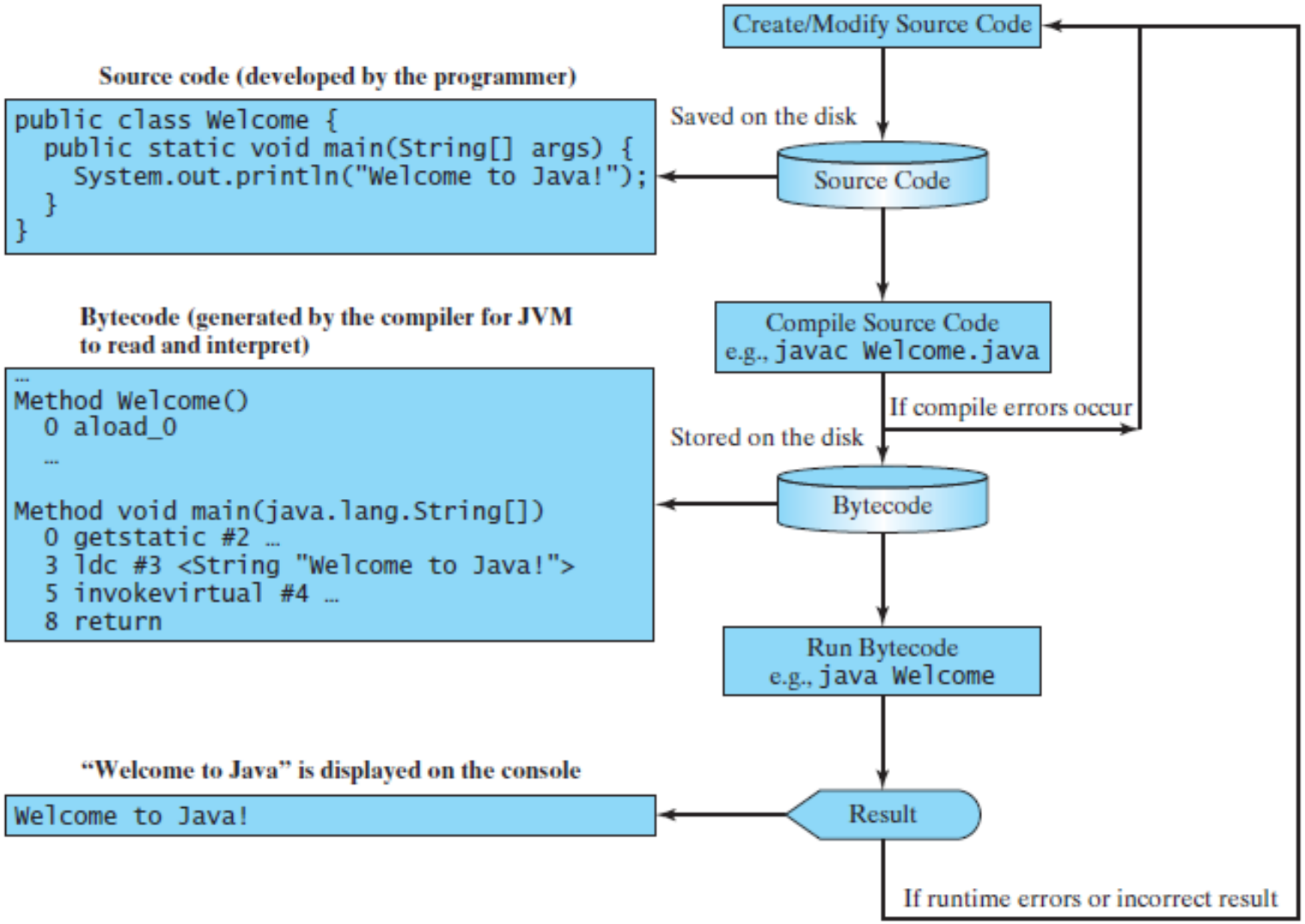
Block comments

Class name

Single line Comment

Main Method

Creating, Compiling and Executing the Java Program



Programming Style and Documentation

"Good programming style and proper documentation make a program easy to read and help programmers prevent errors"

- **Comments**
  - // → line comment, /* … */ → Block comment
- **Proper Indentation and Spacing**
  - Indent each subcomponent or statement at least two spaces more than the construct within which it is nested

```
System.out.println(3+4*4);
```
← Bad spacing style

```
System.out.println(3 + 4 * 4);
```
← Good spacing style

- **Block Styles**

```
Public class test
{
  public static void main(String[] args)
  {
     System.out.println("Next line style");
   }
}
```

```
Public class test{
  public static void main(String[] args){
     System.out.println("End of line style");
   }
}
```

Programming Errors

- Syntax Error or Compiled Error
  - Errors that are detected by the compiler

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

missing void

missing closing quotation mark

- Runtime Errors
  - Runtime errors are errors that cause a program to terminate abnormally
    - Eg. input error, division by zero

- Logic Errors
  - Logic errors occur when a program does not perform the way it was intended to

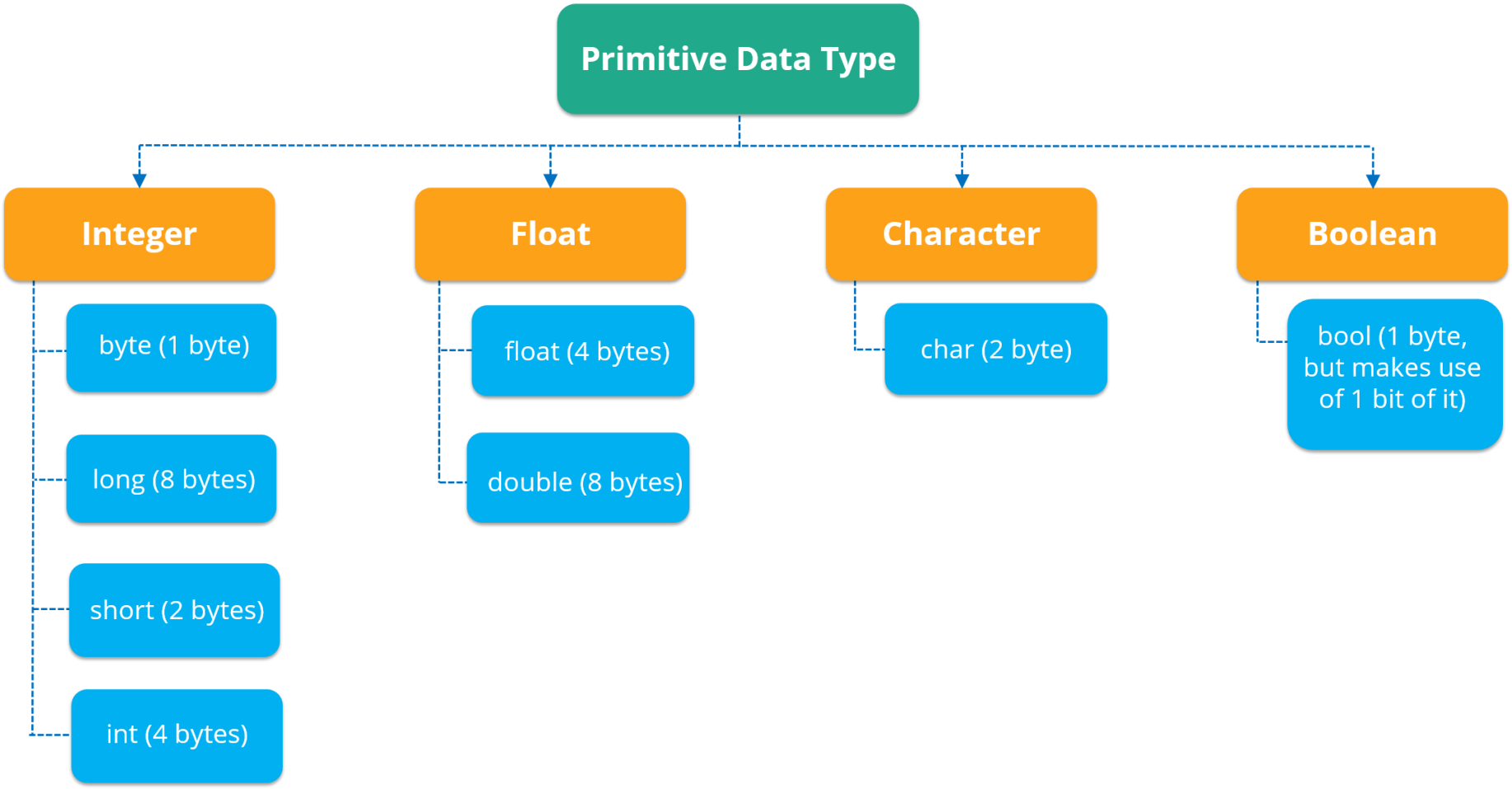
Topic 2

Variables, Data Types and Arrays

Identifiers

- Identifiers are the names that identify the elements such as classes, methods, and variables in a program.
  - Eg. class **MyProgram**
- All identifiers must obey the following rules:
  - A sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
  - Must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - Cannot be a reserved word.
  - Cannot be true, false, or null.
  - Can be of any length.

Primitive Data Types



Range of Data Types

Integer

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Floating-point

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Example 1: Write a program that computes the number of miles that light will travel in 1000 days. Approximate speed of light in miles per second is 186000.

```
// Compute distance light travels using long variables.
class Light {
  public static void main(String args[]) {
    int lightspeed;
    long days;
    long seconds;
    long distance;
    // approximate speed of light in miles per second
    lightspeed = 186000;
    days = 1000; // specify number of days here
    seconds = days * 24 * 60 * 60; // convert to seconds
    distance = lightspeed * seconds; // compute distance
    System.out.print("In " + days);
    System.out.print(" days light will travel about ");
    System.out.println(distance + " miles.");
  }
}
```

Output

```
In 1000 days light will travel about 1607040000000 miles.
```

Example 2: Write a program to compute the area of the circle. Size of radius is 10.

```
// Compute the area of a circle.
class Area {
  public static void main(String args[]) {
    double pi, r, a;
    r = 10.8; // radius of circle
    pi = 3.1416; // pi, approximately
    a = pi * r * r; // compute area
    System.out.println("Area of circle is " + a);
  }
}
```

Output Area of circle is 366.436224

Variables

- A memory location with a name and can be manipulated by programs
- For representing data of a certain type

  datatype identifier;
  datatype identifier1, identifier2, … , identifier_n;
  int a, b, c; // declares three ints, a, b, and c.
  int d = 3, e, f = 5; // declares three more ints, initializing d and f.

- Four kinds of variables in Java
  1. Static
  2. Non-static
  3. Local
  4. Parameters

Variables (cont')

1. Static
   - Variables declared inside the class and outside methods, with the modifier "**static**"
   - Any changes on static variable by an instance of the class is reflected in all other instances of the class
   - "final" modifier should be used during variable declaration to avoid changing the value and to keep it constant
2. Non-static
   - Variables declared inside the class and outside methods, without the modifier "static"
   - Variable value could differ from one instance to another instance of the same class
3. Local
   - Variables declared inside methods
4. Parameters
   - These variables are declared in the signature of a method or constructor or even in exception handlers

Variables (cont')

```
public class Employee {
    public String name; //Non-static variable
    static final String dept= "IT"; //Static variable which cannot be changed
    private double salary; //Non-static variable
    public Employee (String empName)  //Parameter
    {
            name = empName;
   }
  public void setSalary(double empSal) {
            int rate=40; //Local variable
            salary = empSal * rate;
   }
   public static void main(String args[]) {
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
}
}
```

The Scope and Lifetime of Variables

```
class Scope {
        int z; //Known to all instances of the class
        public static void main(String args[]) {
                int x; // known to all code within main
                x = 10;
                if(x == 10) {
                        int y = 20; // known only to this block
                        // x and y both known here.
                        System.out.println("x and y: " + x + " " + y);
                        x = y * 2;
                }
                 y = 100;    // Error! y not known here
                 int y=100; //Error! Cannot declare a variable to have
                                the same name as one in an outer scope
                // x is still known here.
                System.out.println("x is " + x);
        }
}
```

" Variables declared inside a scope are not visible (that is, accessible) to code that is defined outside that scope"

The Scope and Lifetime of Variables (cont')

```
// Demonstrate lifetime of a variable.
class LifeTime
{
        public static void main(String args[])
        {
                int x;
                for(x = 0; x < 3; x++)
                {
                    int y = -1; // y is initialized each time block is entered
                    System.out.println("y is: " + y); // this always prints -1
                    y = 100;
                    System.out.println("y is now: " + y);
                }

        }
}
```

Type Conversion and Casting

- Automatic Conversion
  - The two types are compatible
  - The destination type is larger than the source type

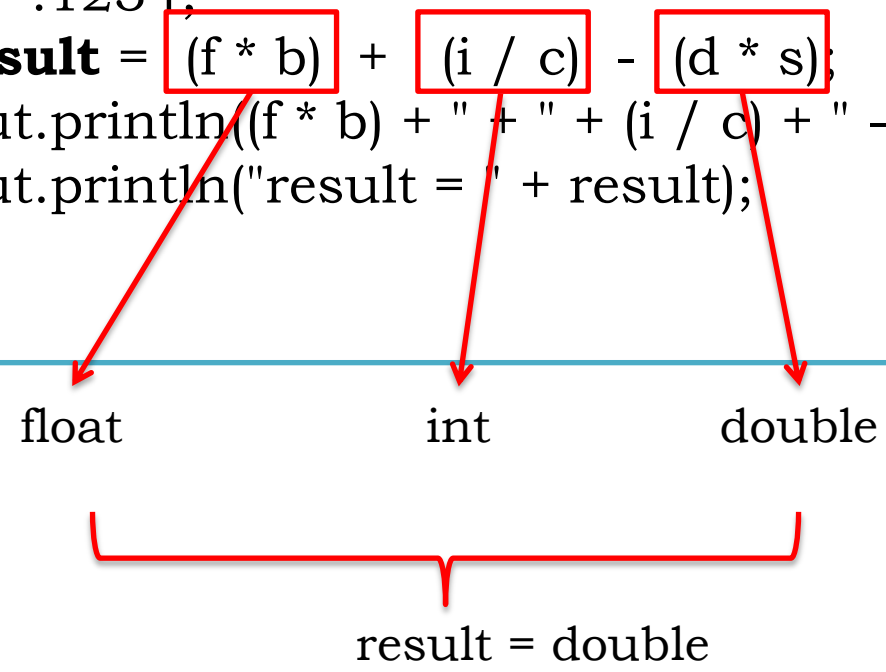
  eg. double d;
  int i=90;
  d=i;
- Casting Incompatible Types
  - (**target type**) value

    int a;
    byte b;
    b=(byte) a;
- Automatic Type Promotion
  - All byte and short values are promoted to int.
  - If one operand is a long, the whole expression is promoted to long.
  - If one operand is a float, the entire expression is promoted to float.
  - If any of the operands is double, the result is double.

```
// Demonstrate Type casts.
class Conversion{
        public static void main(String args[]) {
                byte b;
                int i = 257;
                double d = 323.142;
                System.out.println("\n Conversion of int to byte.");
                b = (byte) i;
                System.out.println("i and b " + i + " " + b);
                System.out.println("\nConversion of double to int.");
                i = (int) d;
                System.out.println("d and i " + d + " " + i);
                System.out.println("\n Conversion of double to byte.");
                b = (byte) d;
                System.out.println("d and b " + d + " " + b);
        }
}
```

i and b	257	1
d and i	323.142	323
d and b	323.142	67

```
//Demonstrate type promotion
class Promote {
        public static void main(String args[])
        {
                byte b = 42;
                char c = 'a';
                short s = 1024;
                int i = 50000;
                float f = 5.67f;
                double d = .1234;
                double result = (f * b) + (i / c) - (d * s);
                System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
                System.out.println("result = " + result);
        }
}
```



Reading Input from the Console

- System.out : the standard output device (display monitor)
- System.in : the standard input device (the keyboard)
- Console input is not directly supported in Java, but can use the Scanner class to create an object to read input from System.in

Scanner input = new Scanner(System.in);

Methods for Scanner Object

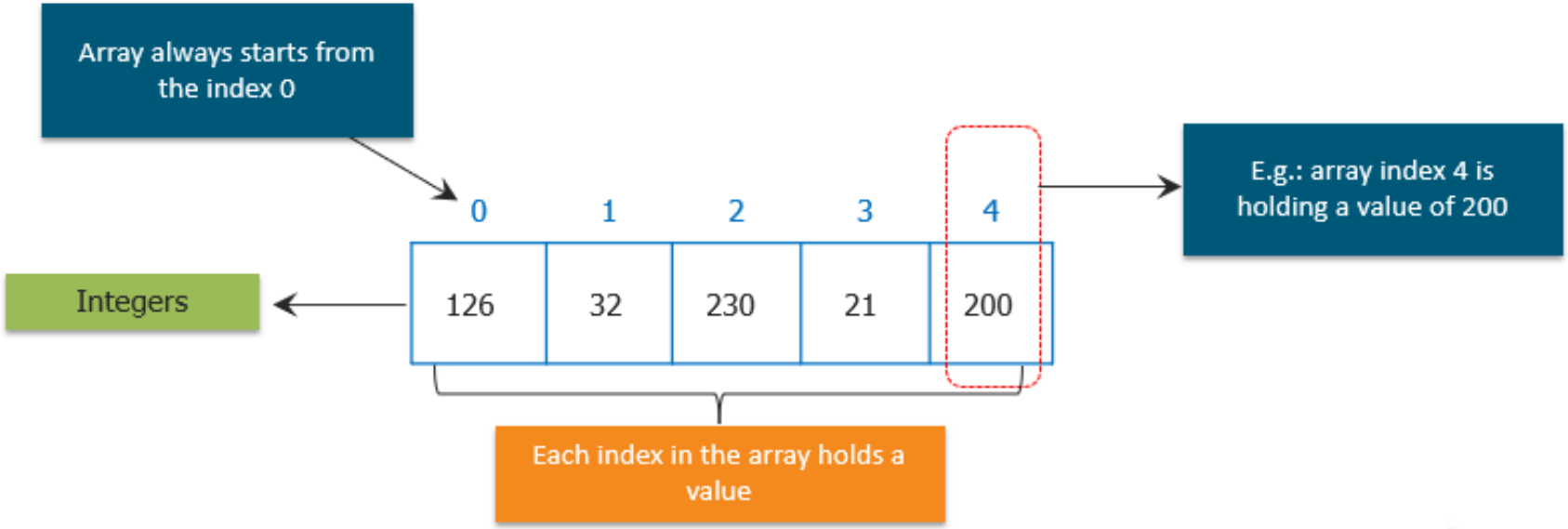
Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.
`next()`	reads a string that ends before a whitespace character.
`nextLine()`	reads a line of text (i.e., a string ending with the *Enter* key pressed).

Example 3: Write a program that reads radius input from the keyboard and computes the area of the circle.

```
//Read radius input from keyboard and compute area of circle.
import java.util.*;
public class ComputeArea{
        public static void main(String[] args) {
                // Create a Scanner object
                Scanner input = new Scanner(System.in);
                // Prompt the user to enter a radius
                System.out.print("Enter a number for radius: ");
                // Read double input
                double radius =input.nextDouble();
                // Compute area
                double area = radius * radius * 3.14159;
                // Display results
                System.out.println("The area for the circle of radius " +
                radius + " is " + area);
        }
}
```

Arrays

- A group of like-typed variables that are referred to by a common name.
- May have one or more dimensions.
- A specific element in an array is accessed by its index.
- Offer a convenient means of grouping related information.



Declaring and Creating Arrays

1. Declare an array variable

   dataType[] arrayName; //preferred) (or)

   dataType arrayName[ ];

   int[ ] days; or int days[ ];

2. Creating an array via the new keyword

   arrayName = new dataType[size];

   days=new int[5];

Steps 1+2 →

dataType arrayNam[ ]=new dataType[size];

int days[ ]=new int[5]

Initializing Arrays

- When an array is created, the elements are assigned default values for its primitive types: 0 for int, 0.0 for double, '\u0000' for char, and false for Boolean type.
- Assigning new values to the elements:

```
days[0] = 11;
days [1] = 12;
days [2] = 13;
days [3] = 14;
days [4] = 15;
```

or

```
int[ ] days = {11,12,13,14,15};
```

- Getting size/length of array

arrayName.length( ) eg. days.length( )

Processing Arrays

1. Initializing arrays with input values:

```
double[ ] myList = new double[10];
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
        myList[i] = input.nextDouble();
```

2. Displaying arrays

```
for (int i = 0; i < myList.length; i++)
{
        System.out.print(myList[i] + " ");
}
```

Processing Arrays (cont')

3. Summing all elements

```
double total = 0;
for (int i = 0; i < myList.length; i++)
 {
            total += myList[i];
}
```

4. Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++)
 {
     if (myList[i] > max)
          max = myList[i];
}
```

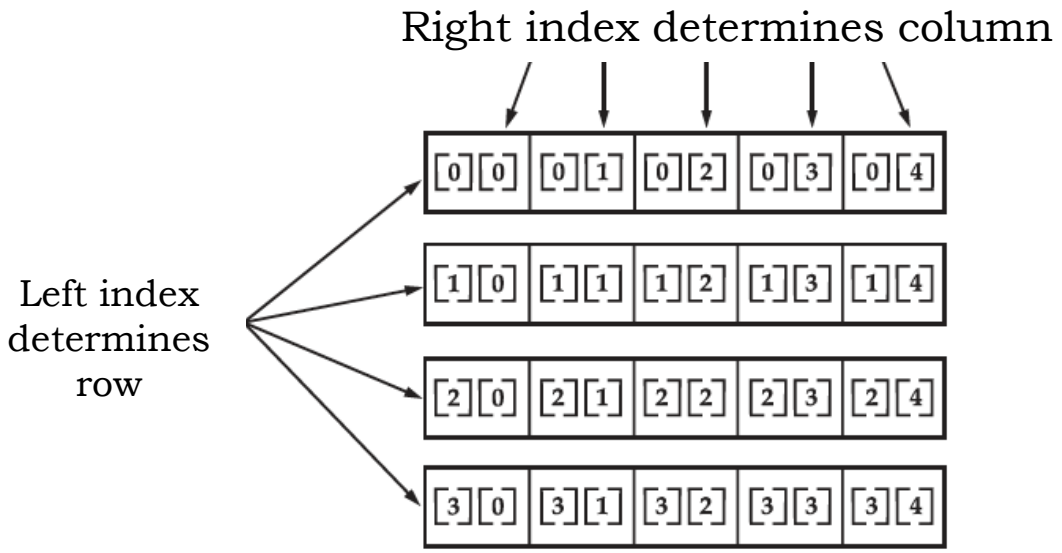
```
// Demonstrate a one-dimensional array. A program that creates an array of number of days in each month
class OneDArray
{
        public static void main(String args[])
        {
            int month_days[]= new int[ 12];
            month_days[0] = 31;
            month_days[1] = 28;
            month_days[2] = 31;
            month_days[3] = 30;
            month_days[4] = 31;
            month_days[5] = 30;
            month_days[6] = 31;
            month_days[7] = 31;
            month_days[8] = 30;
            month_days[9] = 31;
            month_days[10] = 30;
            month_days[11] = 31;
            System.out.println("April has " +  month_days[3] + " days.");
        }
}
```

```
int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

Multidimensional Array

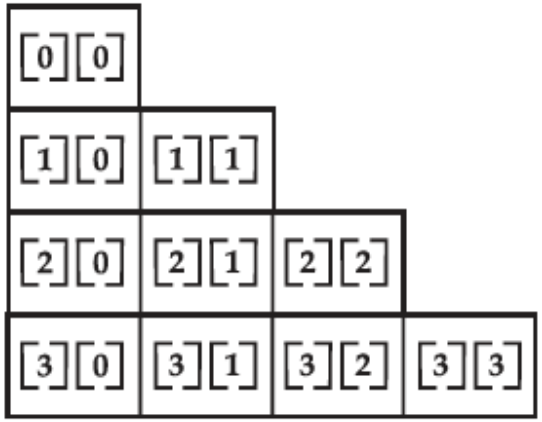
- Array of arrays

dataType arrayName[ ][ ]=new dataType[row][column]



Given: int twoD [ ] [ ] = new int [4][5];

Two dimensional arrays with equal sized second dimension



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

Two dimensional arrays with unequal sized second dimension

```
//Demonstrate a two-dimensional array.
class TwoDArray
{       public static void main(String args[])
        {
                int twoD[][]= new int[4][5];
                int i, j, k = 0;
                for(i=0; i<4; i++)
                    for(j=0; j<5; j++)
                    {
                        twoD[i][j] = k;
                        k++;
                    }
                    for(i=0; i<4; i++)
                    {
                        for(j=0; j<5; j++)
                                System.out.print(twoD[i][j] + " ");
                        System.out.println();
                    }
                    }
                    }
```

Output

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

//Demonstrate an unequal size two-dimensional array.

```
class TwoDAgain
{
  public static void main(String args[])
 {
        int twoD[][] = new int[4][];
        twoD[0] = new int[1];
        twoD[1] = new int[2];
        twoD[2] = new int[3];
        twoD[3] = new int[4];
        int i, j, k = 0;
        for(i=0; i<4; i++)
              for(j=0; j<i+1; j++)
              {
                    twoD[i][j] = k;
                    k++;
              }
        for(i=0; i<4; i++)
        {
              for(j=0; j<i+1; j++)
              System.out.print
              (twoD[i][j] + " ");
              System.out.println();
        }
  }
}
```

Output

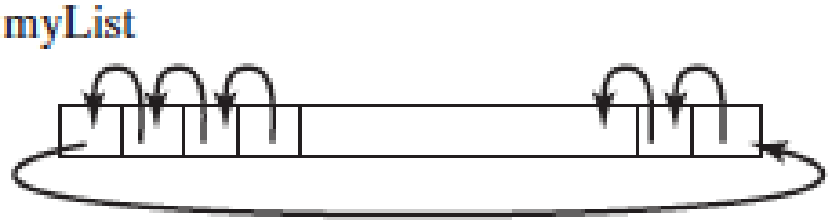
```
0
1 2
3 4 5
6 7 8 9
```

Example 4. Write a program to find an element of an array using the sequential search algorithm.

```
public class SquentialSearch{
public static void main(String[] args){
      int[] arr={23,2,4,56,80,23,4,5,6,10};
      int pos=seqsearch(arr,56, arr.length); //call method for sequential search
      if(pos!=-1)
        System.out.println(" The values is found at the position of "+ pos);
}
//Method for sequential search
public static int seqsearch(int[ ] dataset,int target,int n){
      int found=0;
      int i;
      int pos=-1;
      for(i=0;i<n && found!=1;i++)
        if(target==dataset[i])
         {
            pos=i;
            found=1;
          }
      return pos;
        }
}
```

Assignments

1. Write a Java program that reads a number in inches, converts it to meters.
2. Write a Java program to convert minutes into a number of years and days.
3. Write a Java program that accepts two integers from the user and then prints the sum, the difference, the product, the average, the distance (the difference between integer), the maximum (the larger of the two integers), the minimum (smaller of the two integers).
4. Write a program that searches for an element of the array by using binary search algorithm.
5. Write a program that shifts the array elements one position to the left and fills the last element with the first element.



Next Week Lecture

- **Operators**
- **Control Statements**

Thank you!