Contnets:

Object Oriented Programming (OOP) – Characteristics of OOP – Features of JAVA – Advantages of JAVA – Tools Available of JAVA Programming (JDK, JAVA Packages, various IDEs like NetBeans, eclipse) – Building Java applications.

Objects and Classes – Defining a Class – Declaring attributes, Declaring and Defining methods, Creating Object – Accessing Objects - Constructors – Constructor overloading – Static variables, constants and methods – method overloading – Visibility modifiers, Data field Encapsulation, passing and returning objects as arguments, Array of objects-Exception handling, Try,catch,-Multiple catch & finally statement.

Object Oriented Programming

Object Oriented Programming (OOP) is a programming approach that is based on real world entities known as “objects”. It follows a bottom-up approach to develop applications.

Concepts or characteristics of OOP

Following are the fundamental concepts of object oriented programming:

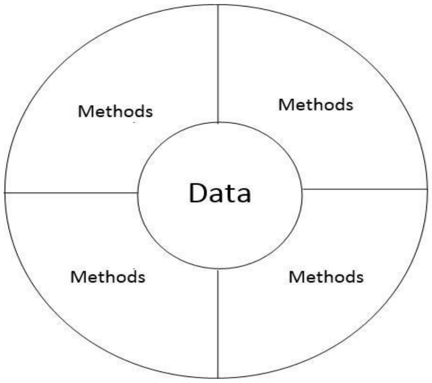
1. Class
2. Object
3. Data abstraction
4. Data Encapsulation
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Communication

Object:-

Objects are the basic runtime entities in an object-oriented system. Objects represent some things or concepts in the real world which have certain **properties (or states or attributes)** and **behavior**. The properties and behavior of objects are defined in its class. So objects are also known as **instances** of class.

Properties or states are **the data** and **behavior** is the **function or method** that acts upon the data.

Object = Data + Methods



Class:-

A class is a user-defined **blueprint** or **data type** using which objects are created. It represents or defines the set of **properties (attributes or states)** and **behaviors** that are common to all objects belonging to that class.

A class may be thought of as a ‘data type’ and an object as a ‘variable’ of that data type.

Eg: - **mango, orange** and **apple** are objects of the class **fruit**.

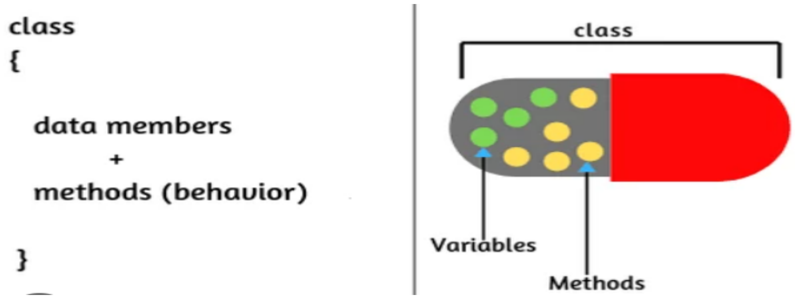
Data abstraction

Data Abstraction is the property in which only the essential details are displayed to the user and non-essential details are kept hidden from the outside world.

Example:**An ATM machine** which can be used for cash transfer, withdrawal, inquiring account balance, etc. We utilize ATM machines to achieve different functionalities but when we put the card in the ATM, we have no idea what operations are happening within the ATM machine.

Data Encapsulation

The wrapping up of data and methods into a single unit (called class) is known as encapsulation. It provides the important security effect of **data hiding or information hiding**.

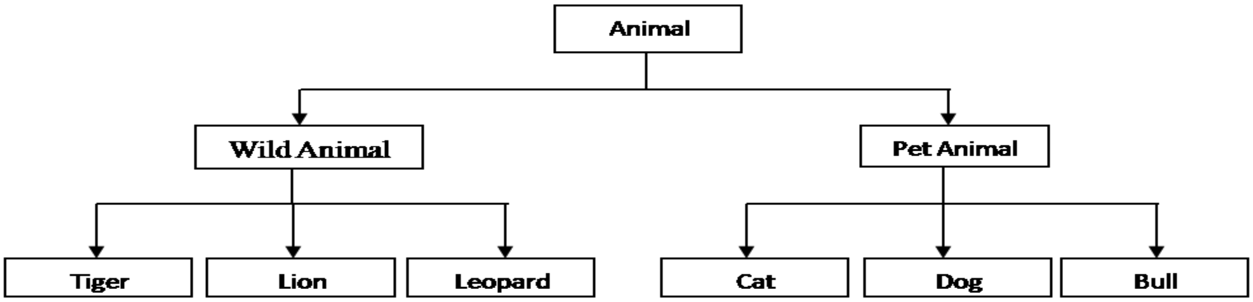


```
class
{
  data members
        +
  methods (behavior)

}
```

Inheritance

Inheritance is an important concept of OOP by which one class is allowed to inherit the features (properties and methods) of another class. Inheritance supports the concept of "**reusability**".



Polymorphism

It is another major concept of OOP which is the ability to take more than one form. **Polymorphism** is a manner in which **object oriented** systems allow the same operator name or function name to be used for multiple operations.

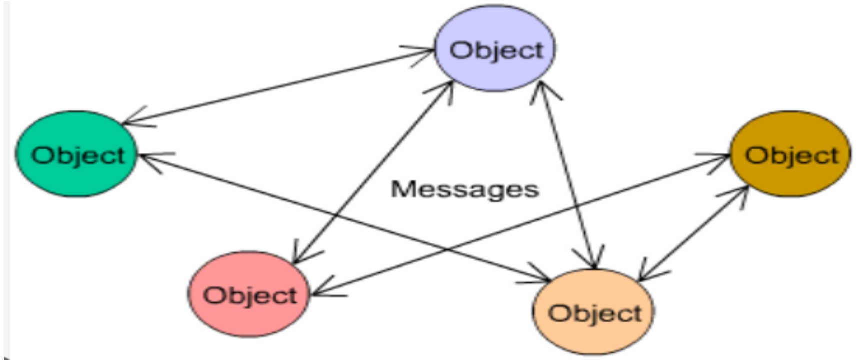
There are two types of polymorphism, they are:

- **Run-time polymorphism (method overriding)** – Method to be invoked is determined at execution time or at runtime. Run-time polymorphism is a **dynamic binding** process.
- **Compile-time polymorphism (method overloading)** – Method to be invoked is determined at the compile time. It is a **static binding** process.

Example:

Data	Method	Operation
x, y	Add(x, y)	12+40 : Add two numbers
s1, s2	Add(s1, s2)	"good" + "bad" : concatenate two strings
img, doc	Add(img, doc)	Image + document : paste an image to a document

Message Passing



- Objects communicate with one another by sending and receiving information.
- A message for an object is a request for execution of a procedure
- The object receiving the message will invoke a method (function) and generates the desired output
- Message passing involves specifying the **name of the object**, **name of the method(message)** and the **information** to be sent
- Example:- **Employee.salary(empid);** Here **Employee is the object**, **salary is the message** and **empid is the** parameter that contains **information**

Difference between OOP and POP:

	OOP (Object oriented Programming)	**POP (Procedure Oriented Programming)**
Program Organization	Program is divided into objects	Program is divided into functions
Approach	Bottom-up approach	Top-down approach
Importance	Importance is given to **data** than to functions	Importance is given to **functions** ; not to data
Inheritance	Inheritance allows reuse of existing code	Inheritance is not allowed
Access specifier & Data Access	Uses access specifier (public, private, protected) Objects access local data and can be accessed in a controlled manner	It doesn't use access specifier. Functions use global data for sharing. Data can be accessed freely from function to function
Data Hiding	Encapsulation is used to hide data	No data hiding
Maintainability	Adding new data and functions is easy	Adding new data and functions is not easy
Languages	Eg:-C++ , Java, C#	Eg:- C, Pascal

The Java Language

Java is a **general-purpose**, **object-oriented, platform-neutral** programming language developed by Sun Microsystems of USA in 1991.Its original name was "Oak" Oak was renamed "Java" in 1995.

JAVA FEATURES

Java developers wanted to create a simple compact interactive language that is reliable, portable and distributed. Sun microsystems officially describes Java with the following features: These features have made java a suitable language for WWW applications as well as general-purpose stand-alone applications

1. Compiled and Interpreted
2. Platform-Independent and Portable
3. Object-Oriented
4. Robust and Secure
5. Distributed
6. Familiar, simple and Small
7. Multithreaded and Interactive
8. High Performance
9. Dynamic and Extensible

Compiled and Interpreted:-

- Java source code is translated to machine code in two stages.
- In the **first stage Java compiler** translates the source code to an intermediate code known as "**byte code**".
- Byte codes are not machine instructions. Therefore, in the **second stage , Java Interpreter** generates machine code. So, Java is both compiled and interpreted

Platform-Independent and Portable

- Portable or platform-Independent means the same program can be executed in any device located anywhere , any time. It doesn't have any dependency on OS, Processors and other system resources.
- For example, Java applets can be executed in any browsers installed in any device without any modification
- Java ensures portability (platform-Independency) in two ways:
  - Java compiler generates bytecode that can be implemented on any machine
  - The size of primitive data types are machine-independent

Object-Oriented

- Java is a true object-oriented language. Almost everything in java is an object. All program codes and data reside within **objects** defined using **class** concept..

Robust and Secure

- Java is a robust language.
- It has strict compile-time and run-time checking for data types.
- Strong garbage collection system handles memory-management problems
- Compile-time and run-time exception handling captures serious errors and avoids system crashes

- Absence of pointers ensures authorized memory access which provides security
- Provides various access controls - private, protected

Distributed

- Java is designed as a distributed language for creating applications on networks. Popular language for Network Programming.
- Java applications can open and access remote objects on Internet as easily as they can do in a local system that enables multiple programmers to collaborate on single project from different locations

Simple, Small and Familiar

- Java is a simple language. Many features that may cause unreliability in C and C++ such as pointers, preprocessor header files, operator overloading, multiple inheritance etc. are not part of Java.
- Java uses many constructs of C and C++ . This familiarity is a striking feature of Java.

Multithreaded and Interactive

- Multithreaded means handling multiple tasks simultaneously
- Java applications need not wait to finish one task before beginning another task.
- For example, we can begin an audio clip while scrolling a page and at the same time download an applet from a distant computer.
- Multithreading helps to improve the performance of graphical interactive applications

High Performance

- Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a **just-in-time** compiler
- Incorporation of multithreading enhances the overall execution speed of java programs.

Dynamic and Extensible

- Java is a dynamic language
- Java is capable of dynamically linking in new class libraries, methods and objects
- Native methods from other languages such as C and C++ can be dynamically linked to Java applications

JAVA ENVIRONMENT

Java Environment includes::

1) Java Development Kit (JDK) - A set of development tools
2) Application Programming Interface (API) - Hundreds of classes and methods as part of Java Standard Library (JSL)

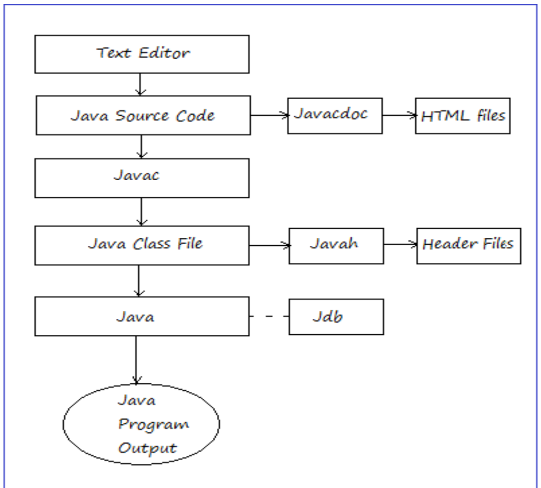
Java Development Kit

- JDK comes with a collection of tools that are used for developing and running java programs. They Include:

	Tool	Description
1	appletviewer	Enables us to run java sports without actually using java-compatible browsers (applets are java programs that run in web browsers)
2	javac (Java compiler)	The java compiler translates the java source code to bytecode files
3	java (Java Interpreter)	Java Interpreter runs applets and applications by reading and interpreting bytecode files
4	javadoc	Create HTML-format documentation from Java source code file
5	javah	Produces header files for use with native methods (for C header files) To create interface between java and C
6	javap	Java disassembler , which enables us to convert bytecode files into a program description
7	jdb	Java debugger, which helps to find errors in our programs

Process of building and running Java application

1. Create a source code file using a **text editor**
2. Compile the source code using the compiler **javac**
3. Execute the byte code using the interpreter **java**
4. **jdb** is used to find errors, if any, in the source code
5. Compiled java program can be converted to source code using the disassembler **javap**



Process of building & running Java programs

Application Programming Interfaces (API)

- The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages
- Most Commonly used packages are:

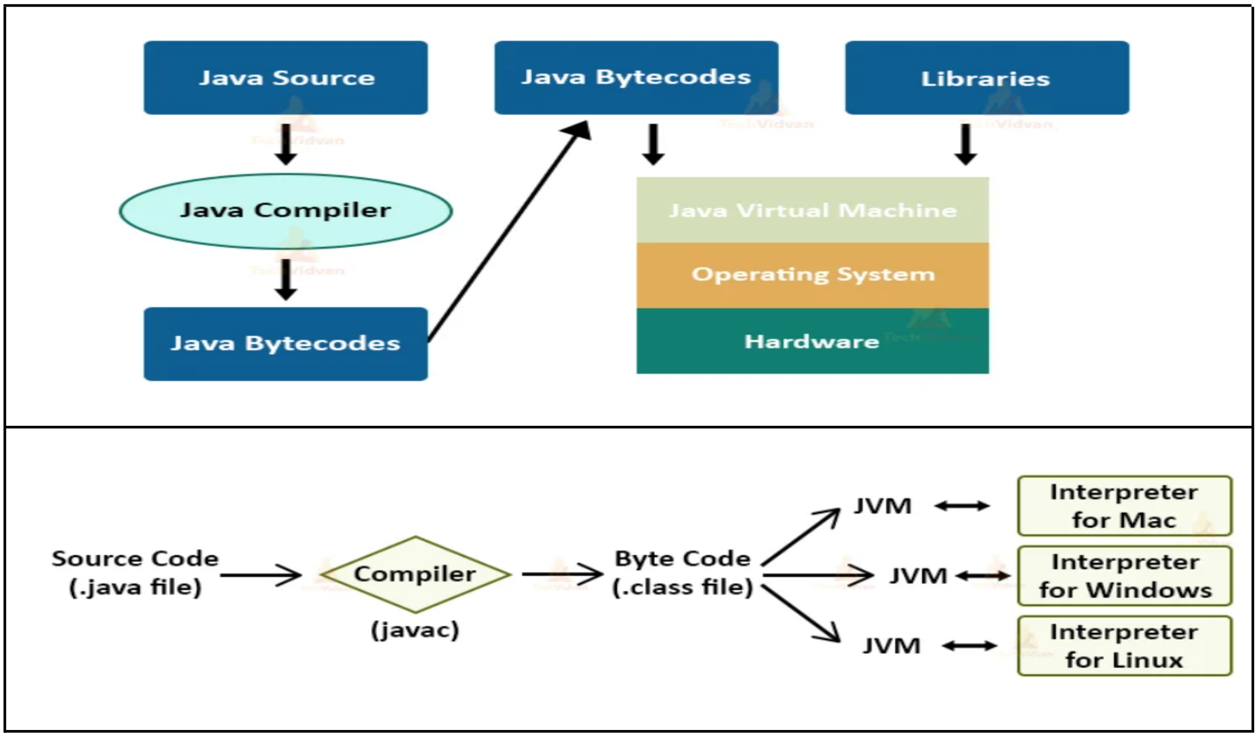
	Package (API)	**Description**
1	Language Support Package (**java.lang**)	Classes and methods for implementing basic features of java(classes for **primitive types, strings, math functions, threads and exceptions**)
2	Utilities Package (**java.util**)	A collection of classes to provide utility functions such as **date, time, vectors, hash tables** etc.
3	Input/Output Package (**java.io**)	A collection of classes required for input / output manipulation.
4	Networking Package (**java.net**)	A collection of classes for communicating with other computers via the Internet.
5	AWT package (**java.awt**)	The Abstract Window Tool Kit package contains classes that implement platform-independent graphical user interface.
6	Applet Package (**java.applet**)	This includes a set of classes that allows us to create Java applets.

Java Runtime Environment

- The Java Runtime Environment (JRE) facilitates the execution of programs developed in Java.
- JRE comprises the following:
  - **Java Virtual Machine (JVM)** :- It is a program that interprets the intermediate Java bytecode and generates the desired output. **JVM and bytecode makes Java portable.**
  - **Runtime class libraries** : Set of core class libraries that are required for the execution of Java programs
  - **User Interface Toolkits** :-**AWT** and **Swing** are examples of toolkits that support interactive Java applications
  - **Deployment Technologies:**

- **Java plug-in** : Enables the execution of java applied on the browser
- **Java Web Start:** Enables remote-deployment of an application. User can directly launch an application without really installing it in their device.

Working of JVM in executing Java Programs

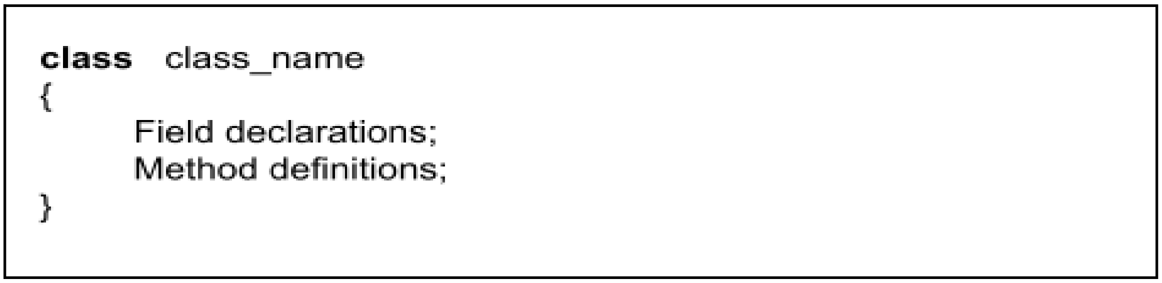


Java IDEs

- **An integrated development environment** (IDE) is a software for coding programs that combines common developer tools into a single GUI.
- It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging
- Examples:-
    - **NetBeans & Eclipse** are most commonly used IDEs for java development

CLASS, OBJECT & METHOD

- A class is a template that defines the properties and behaviour of objects
- It is a user defined data type
- A class consists of two types of members:
  - ***Data fields***
    - Data fields are also known as data members   or instance variables or member variables
  - ***Methods***
    - Methods are functions that define the operations or actions that the objects can perform using its data. It shows the behaviour of objects
- **Syntax of defining a class**



```
class   class_name
{
        Field declarations;
        Method definitions;
}
```

- ***Syntax of Field declarations:-***

```
data_type     var1, var2, …. ;
```

Example:

int length, breadth;

- **Method Definition**
  - Methods are functions that are used to implement behavior of objects
  - Methods define the operations that an object can perform using its data.

Syntax of Method definition

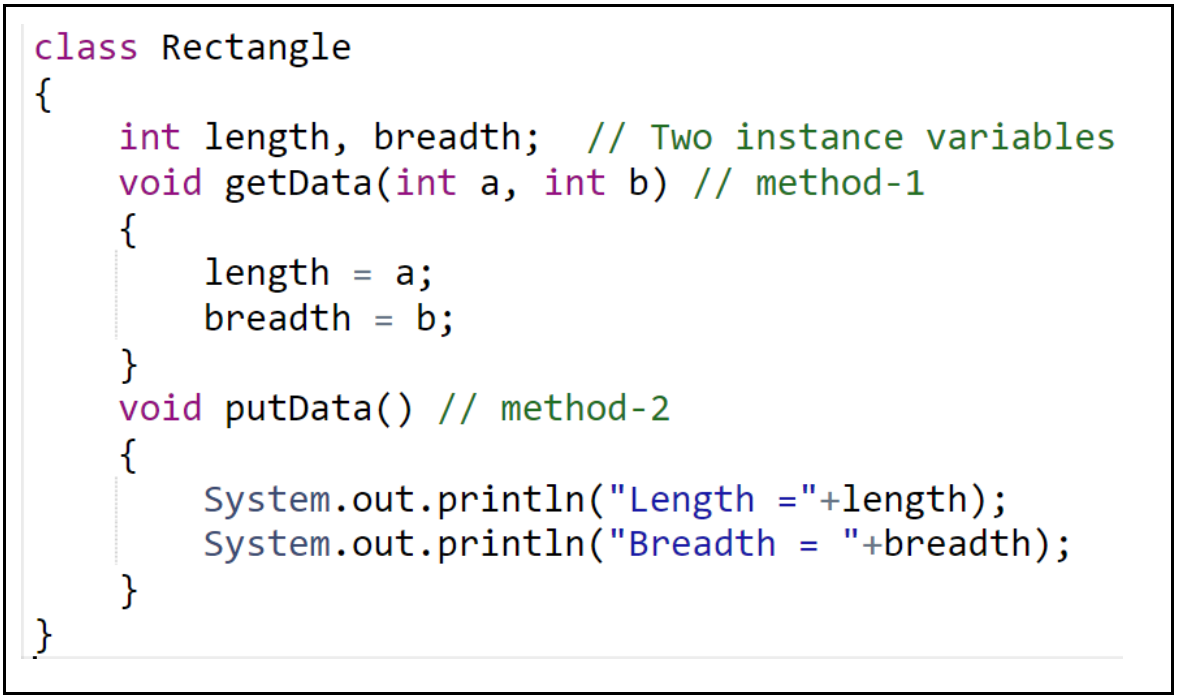
```
return_type    method_name( parameter declaration )
{
        /// Body of the method
}
```

The syntax consists of 4 parts:

1) Name of the method
2) Data type of the value returned by the method. If it returns nothing, the return_type will be **void**

3) A list of parameters (generally these are the input data to the method)
4) Body of the method (Executable statements to perform the operation)

Example for a class definition : Define a class **Rectangle** with **two instance variables - length & breadth** and **two methods - getData()** to get data to the instance variables **putData()** to display the value of instance variables

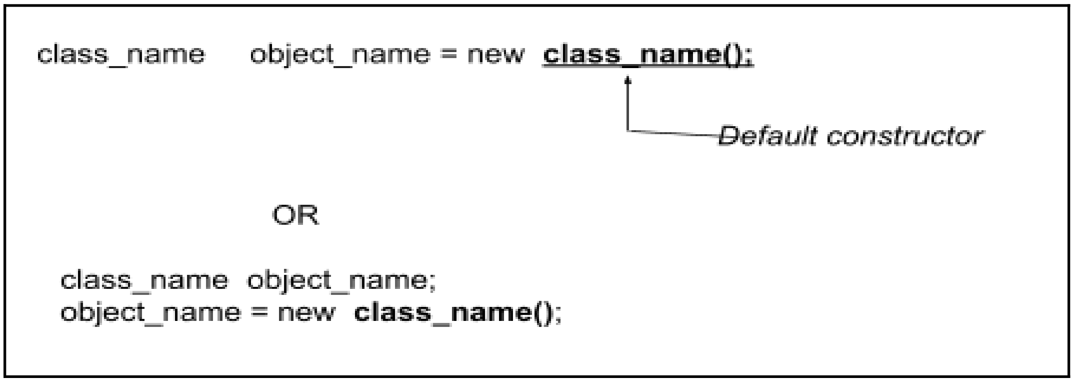


```
class Rectangle
{
    int length, breadth;  // Two instance variables
    void getData(int a, int b) // method-1
    {
        length = a;
        breadth = b;
    }
    void putData() // method-2
    {
        System.out.println("Length ="+length);
        System.out.println("Breadth = "+breadth);
    }
}
```

CREATING OBJECTS

- An object is an instance of a class
- Creating an object is also known as instantiating an object
- When we create an object, the memory space required to store its instance variables is allocated. ***Each object has its own instance variables***
- Objects are created using the ***new*** operator
- After creation, it returns a *reference* of the object created

Syntax of creating object



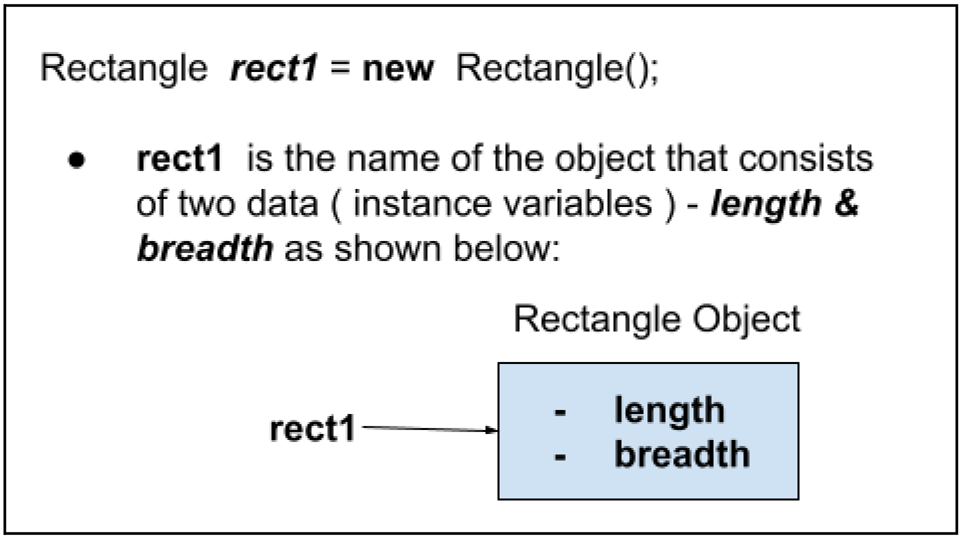
```
class_name    object_name = new  class_name();
```

Default constructor

OR

```
class_name  object_name;
object_name = new  class_name();
```

Example: To create an object **rect1** of the class Rectangle



```
Rectangle rect1 = new Rectangle();
```

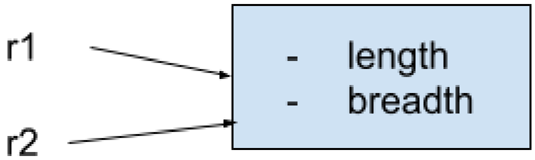
- **rect1** is the name of the object that consists of two data (instance variables) - ***length & breadth*** as shown below:

- Suppose we have two Rectangle objects and when we assign one object to the other , it just assigns the reference so that both points the same object

Rectangle r2;

Rectangle r1 = new Rectangle();

r2 = r1; // it just assign the reference so that **r1** and **r2** points the same object as shown below:



- if we assign **r1=null;** , no issues, you can access the object using the reference **r2**

ACCESSING CLASS MEMBERS

- To access class members from outside the class, we use dot operator (member operator) as shown below:

object_name .variable_name = value;

object_name.method_name(actual_parameters);

Example:

To assign values to the variables ***length*** & ***breadth*** of the Rectangle object ***rect1***

```
rect1.length = 15;
rect1.breadth = 20;
```

To call the methods ***getData()*** and ***putData()***

```
rect1.getData(15, 20);
rect1.putData();
```

Structure of a java Program

Documentation or Comment Section	← Suggested Section
package Statement	← Optional
import Statements	← Optional
interface definitions	← Optional
class definitions	← Optional
main() method class	Essential section because the main() method is the starting point of program execution.

Documentation or comment section

- Comment section is optional still it is suggested to include comments.
- Comments are non-executable statements that provide helpful information about the program, author etc.
- Three types of comments in Java:
  1) Single line comment (any line starting with //)
  2) Multi-line comments (multi-line comments are enclosed between /* and */)
  3) Documentation comments (The documentation comments are placed between /** and */. These comments will be included in the **HTML** documentation file generated while using **javadoc** tool)

package Statement

- Optional section
- It informs the compiler that the classes defined in the program belong to this package.
- Package in java is a collection of classes.

import Statements

- Optional section
- It instructs the interpreter to load the specified class or all classes of the specified package.
- This way, we can access classes from other java packages

interface definitions

- Optional section
- An interface in Java is like a class with a set of method declarations
- It is used when we need to implement multiple inheritance

Class Definitions

- Classes are user defined data types that define the state and behavior of objects.
- A java program may contain many classes.
- All classes must be declared in this section

Main method class

- It is an essential section of a Java program
- Program execution always starts from the main**()** method.
- The class in which the **main() method** is defined is known as **Main Class**.

Example Program :

1) Write a program to define a Rectangle class with following members
2) Write a program to define a Student class with following members

CONSTRUCTOR

- Constructor is a special method with the same name as its class.
- A Constructor is invoked automatically and initializes an object when the object is created
- Typically, we use constructor to give initial values to instance variables defined by the class
- Constructors have no explicit return type. It implicitly returns the object reference
- Java automatically provides a default constructor for every class. Once we define a constructor the default constructor is no longer used

Example:

```
class  MyClass
{
      int x;
          MyClass() // constructor
          {
               x=10;
          }
}
```

- The constructor **MyClass()** assigns the value 10 to the instance variable x

```
class   DemoClass
{
       public static void main(String args[])
      {
                    MyClass  obj1 = new MyClass();
                    MyClass  obj2 = new MyClass();
                    System.out.println(obj1.x + "  " + obj2.x);
   }
}
```

In the line **MyClass obj1 = new MyClass(); ,** the constructor **MyClass()** is called and the returning object is assigned to **obj1**. After construction, **obj1.x** has the value **10** . The same thing happens for **obj2**

So the output of the program is:

```
10   10
```

Parameterized Constructors

- It is possible to declare parameters in constructor method
- Values to parameters are passed while creating objects

Example:

```
class MyClass
{
      int x;
```

```
             MyClass( int k) // constructor for MyClass with parameter
             {
                x = k;
             }
}
```

The constructor **Myclass()** assigns the instance variable **x** of MyClass the value of **k.**

```
class   DemoClass
{
    public static void main()
    {
          MyClass obj1 = new MyClass( 55 );
          MyClass obj2 = new MyClass(  12 );
           System.out.println(obj1.x + "  " + obj2.x);
    }
}
```

In the line MyClass obj1 = new MyClass(55); , the constructor MyClass() is called with value 55 passed to the parameter k , which is used to initialise the instance variable x

So the output of the program is:

```
55  12
```

Method Overloading

- In Java, two or methods within the same class can have the same name, as long as their parameter declarations are different. Then we can say that the methods are overloaded. The process is known as method overloading.
- Either the type of parameters or the number of parameters or both must be different
- When an overloaded method is called, the version of the method whose parameters match with the actual arguments is executed.
- Method overloading is one of the ways that Java implements compile-time polymorphism.

Sample Programs :

1) ***Overload the method area() to calculate area of different geometrical shapes***
2) ***Overload the method sum() to add two integers, two float numbers, to strings etc.***

Constructor Overloading

- A class can have more than one constructors with different parameter list , either in their data type or in their number
- While creating objects, the constructor whose parameters match with the actual arguments will be invoked.

Sample Programs :

1) Define a Time class with multiple constructors
2) Define Rectangle class with multiple constructors

Static variables

- **Static variables are class variables shared by all objects of the class**
- **Static members (variables & methods) can be accessed using class name from other classes**
- **Static methods can only call other static methods**
- **Static methods can only access static data**

Major difference between Static variable & Instance Variable

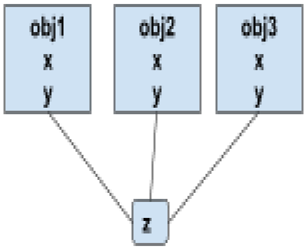
Instance Variable and methods	**Static Variables & Static methods**
Each object of the class has their own instance variables	Static variables are **class variables** and the **same variable is shared by all object** of the class
Instance variables and methods are **accessed using objects** from outside the class	Static variables and methods can be **accessed using class_name** without creating objects
The object reference **‘this’ can be used** to refer the current object	The object reference **‘this’ cannot be used** in a static method

```
class   Test
{
      int x,y;
      static int c;
……………
}
class Main
{
```

Memory usage of objects obj1, obj2 and obj3 are as shown below:

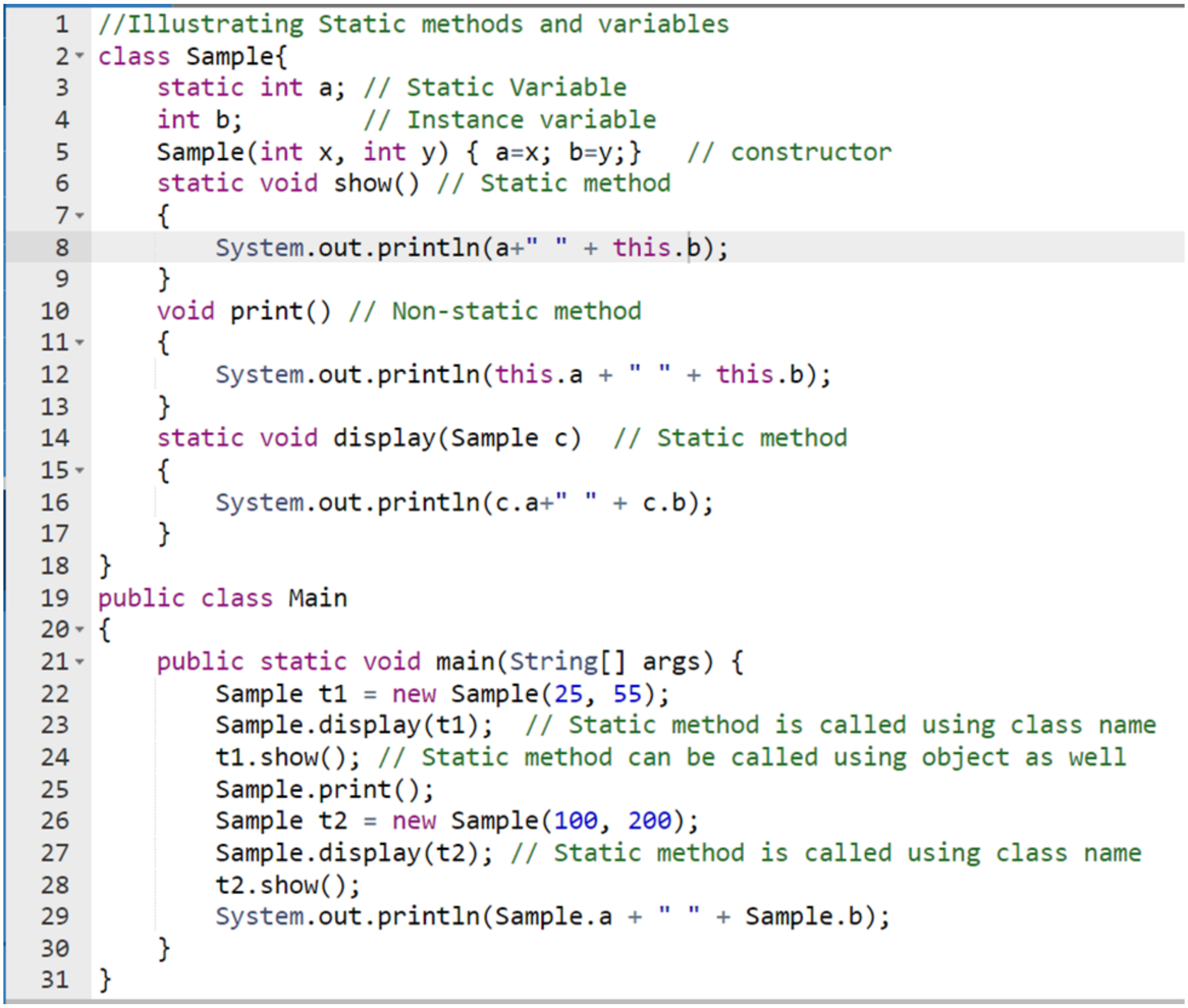
```
  public static void main(String argos[])
  {
         Test  obj1 = new Test();
         Test  obj2 = new Test();
          Test obj3 = new Test();
          ………….

  }
}
```



Static variable - z - has a common allocation and same variable is shared by all objects where as each object has separate allocation for instance variables - x and y

See the program below as an illustration for static members



```
 1  //Illustrating Static methods and variables
 2  class Sample{
 3      static int a; // Static Variable
 4      int b;        // Instance variable
 5      Sample(int x, int y) { a=x; b=y;}   // constructor
 6      static void show() // Static method
 7      {
 8          System.out.println(a+" " + this.b);
 9      }
10      void print() // Non-static method
11      {
12          System.out.println(this.a + " " + this.b);
13      }
14      static void display(Sample c)  // Static method
15      {
16          System.out.println(c.a+" " + c.b);
17      }
18  }
19  public class Main
20  {
21      public static void main(String[] args) {
22          Sample t1 = new Sample(25, 55);
23          Sample.display(t1);  // Static method is called using class name
24          t1.show(); // Static method can be called using object as well
25          Sample.print();
26          Sample t2 = new Sample(100, 200);
27          Sample.display(t2); // Static method is called using class name
28          t2.show();
29          System.out.println(Sample.a + " " + Sample.b);
30      }
31  }
```

```
Main.java:8: error: non-static variable this cannot be referenced from a static context
        System.out.println(a+" " + this.b);
                                   ^
```

```
Main.java:25: error: non-static method print() cannot be referenced from a static
context
                    Sample.print();
                          ^
Main.java:29: error: non-static variable b cannot be referenced from a static context
                    System.out.println(Sample.a + " " + Sample.b);
                                                               ^
3 errors
```

Visibility Modifiers (Access Modifiers)

Access to members of a class can be controlled using **three access specifiers:**

1. **public**
2. **private**
3. **protected**

Usage syntax

- public int count;
- private int mark;
- public Rectangle() { …… }

public Access

- public members of a class can be accessed from methods defined in any other classes of our application
- In other words, public variables or public methods have the widest visibility and are accessible everywhere.

Private Access

- When a member of a class is specified as private, that member can be accessed only by other members of its class.
- In other words, private members are accessible only from the methods of their own class.
- Private members have the highest degree of protection

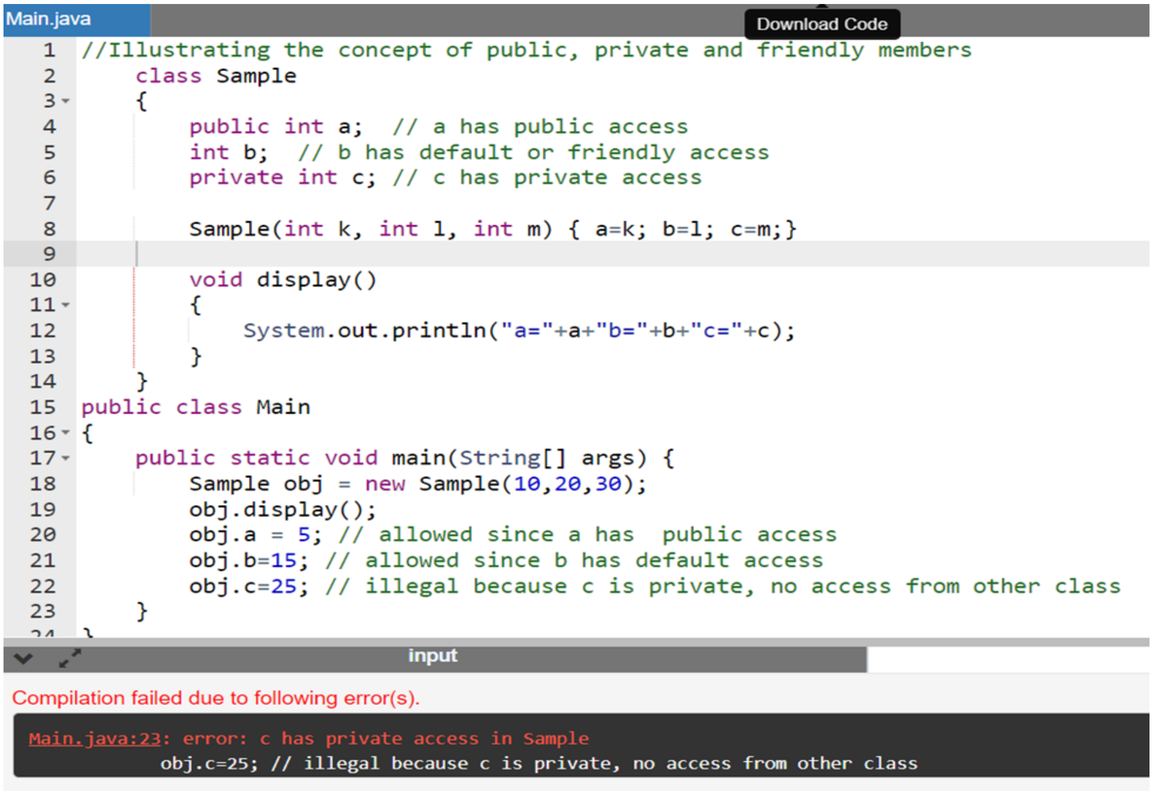
Default access (friendly access)

- The default access setting (in which no modifier is used) is the same as public unless two or more packages are involved in our application. A package is a grouping of related classes.
- Default access is also known as friendly access

Difference between public access & friendly access

- public modifier makes fields visible in all classes regardless of their package while friendly access (default access with no modifier) makes fields visible only in the same package, but not in other packages

Illustraion of public vs. private access



```java
//Illustrating the concept of public, private and friendly members
    class Sample
    {
        public int a;   // a has public access
        int b;   // b has default or friendly access
        private int c; // c has private access

        Sample(int k, int l, int m) { a=k; b=l; c=m;}

        void display()
        {
            System.out.println("a="+a+"b="+b+"c="+c);
        }
    }
public class Main
{
    public static void main(String[] args) {
        Sample obj = new Sample(10,20,30);
        obj.display();
        obj.a = 5; // allowed since a has  public access
        obj.b=15; // allowed since b has default access
        obj.c=25; // illegal because c is private, no access from other class
    }
}
```

Compilation failed due to following error(s).

```
Main.java:23: error: c has private access in Sample
            obj.c=25; // illegal because c is private, no access from other class
```

** * **protected specifier** is connected with the concept of Inheritance. So will be discussed in detail with the topic Inheritance in Module-2

Passing and returning objects as arguments

- **Methods can have objects as parameters.**
- **Also, methods can return an object as a result after processing.**
- If we need to manipulate two objects within a method, one object can be used to call the method and the second object can be passed as a parameter. If the result is an object, that can be returned. So, the method is called as follows:

result_object = object1.method_name(object2);

- Another means of manipulating two objects is pass both objects as parameters and call the method using result_object as shown below:

result_object.method_name(object1, object2);

- Third way of manipulating two objects is to write a static method with two objects as parameters and return result_object . Static method is then invoked using class_name

result_object = class_name.static_method(object1,object2);

Different ways to manipulate two Objects

Example: Complex Number Addition : (Two operands c1 & c2 ,   result is sum)

SlNo	Method Definition in Complex class	Method Call in main()	Explanation

1	Complex add (Complex c)
	{
	    Complex s = new Complex();
	    s.real = this.real + c. real ;
	    s.imaginary = this.imaginary + c.imaginary;
	    return s;
	}
		sum = c1.add(c2) ;	● add() method called using first operand c1 and second operand c2 is passed as parameter.
			● The sum is another Complex number which is returned from add() function
			● The keyword *this* representing the object using which the method is called is optional. this represents c1

2	void add (Complex a, Complex b)
	{
	  this.real = a.real + b.real ;
	  this.imaginary = a.imaginary + b.imaginary;
	}
		sum.add(c1,c2);	● add() method called using result object sum and both operands c1 and c2 are passed as parameters.
			● No need to return sum. Real part and imaginary part of sum is stored in the object used for calling the function.
			● The keyword *this* representing the object using which the method is called is optional. this represents sum

3	static Complex add(Complex x, Complex y)
	{
	Complex s = new Complex();
	s.real = x.real + y. real ;
	s.imaginary = x.imaginary + y.imaginary;
	return s;
	}
		sum=Complex.add(c1,c2);	● add() is defined as a static method so that it can be called using class name
			● Two operands c1 and c2 are passed as parameters and the result object sum is returned.
			● Static method can be called using class name.
			● The keyword this is not allowed in static method

Example 2: *To compare two rectangles r1 & r2 (Logic is test whether length and breadth of both objects are equal)*

SlNo	Method Definition in Rectangle class	Method Call in main()	Explanation

1	int equals(Rectangle r)
	{
	if (this.length == r.length && this.breadth == r.breadth)
	return 1;
	else
	return 0;
	}
		val = r1.equals(r2) ;
		if (val==1)
		System.out.println(“r1 & r2 are equal);
		else
		System.out.println(“r1 and r2 are not equal);
			● equals() method called using first rectangle r1 and second rectangle r2 is passed as parameter.
			● Returns the integer value 1 if both rectangles are equal ; otherwise the method returns 0
			● Here , this represents r1

3	**static int equals(Rectangle x,Rectangle y)**
	{
	if (x.length == y.length && x.breadth == y.breadth)
	return 1;
	else
	return 0;
	}
		val=Rectangle.equals(r1,r2);	● **equals() is defined as a static method so that it can be called using class name**
			● **Two operands r1 and r2 are passed as parameters and the result is an integer value 0 or 1**

Other Exercises to study:

1) Write a program to add **two Distance** objects. Distance is specified in **Feet & Inch (Hint : You have to define Distance class with two instance variables – Feet & Inch , Create two objects of Distance class . Add them , Print both distances and their sum. 1 Feet = 12 Inches)**

2) Define a class **Point** with **two instance variables x and y** to represent points in a two dimensional coordinate system. Write methods for

   - Create two points - Eg:-p1(6, 10) p2(12, 20)
   - Print the points
   - Find the mid point
   - Find the distance between the two points

3) Define Time class with method to add two Time objects

Array of objects

ARRAYS

- An array is a collection of variables of the same data type, referred to by a common name.
- In Java, arrays can have one or more dimensions. One dimensional array is more common.
- Arrays are a convenient method of grouping together related variables
- **In Java, arrays are implemented as objects**

One-Dimensional Arrays

- One dimensional array is a list of variables
- In Java, an array is created in two steps:
  - o **First**, declare the array
  - o **Second**, dynamically allocated memory space using **new** operator

Syntax of declaring one-dimensional array

```
data-type[ ]  array-name        =  new   data-type[size] ;
```

- **data-type** refers to the type of elements in the array
- **size** refers to the maximum number of elements in the array
- Memory space for array elements are dynamically allocated using **new** operator

Example for creating an array:

An **int** array of 10 elements with the name **sample**

```
int[ ]  sample    =   new   int[10] ;
```

This can also be written in two steps:

```
int[ ]  sample;
sample   =   new   int[10] ;
```

Accessing array elements

- Individual elements within an array are accessed using index.
- Index describes the position of an element within an array
- Zero is the index of first element. Since **sample** has 10 elements its index range from 0 to 9
- **sample[0]** is the indexed variable for **first element**. Last element is **sample[9]**

Assigning values to arrays (Use assignment operator)

```
Sample[0] = 15;
Sample[1] = 25;
………..
Sample[9] = 95;
```

- Arrays can also be initialized when they are created

```
data-type[]  array-name = { val1, val2, val3, ….. , valn } ;
```

Here no need to explicitly use the new operator

Example :

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

Array of more dimensions

```
 data-type[ ] [ ] array-name = new   data-type[sze-1][size-2];
data-type[ ] [ ] …[ ]array-name = new   data-type[sze-1][size-2]….[size-N];
```

USING THE length MEMBER

- Since arrays are implemented as objects, there is an instance variable **length** that hold the size (number of elements) of the array

```
// code to print all elements of the array sample
for(i=0 ; i< sample.length  ; i++)
          System.out.println(sample[i]);
```

FOR – EACH style Loop

- For-Each style loop allows to traverse or cycle through a collection of objects
- For - each style loop is also known as enhanced for loop

Syntax

```
 for ( datatype  itr_var : collection)
      Statement block
```

data-type – refers to the type of elements in the collection

itr_var – is the iteration variable that will receive elements from collection

Example 1: To print the elements of array sample using FOR-EACH type LOOP

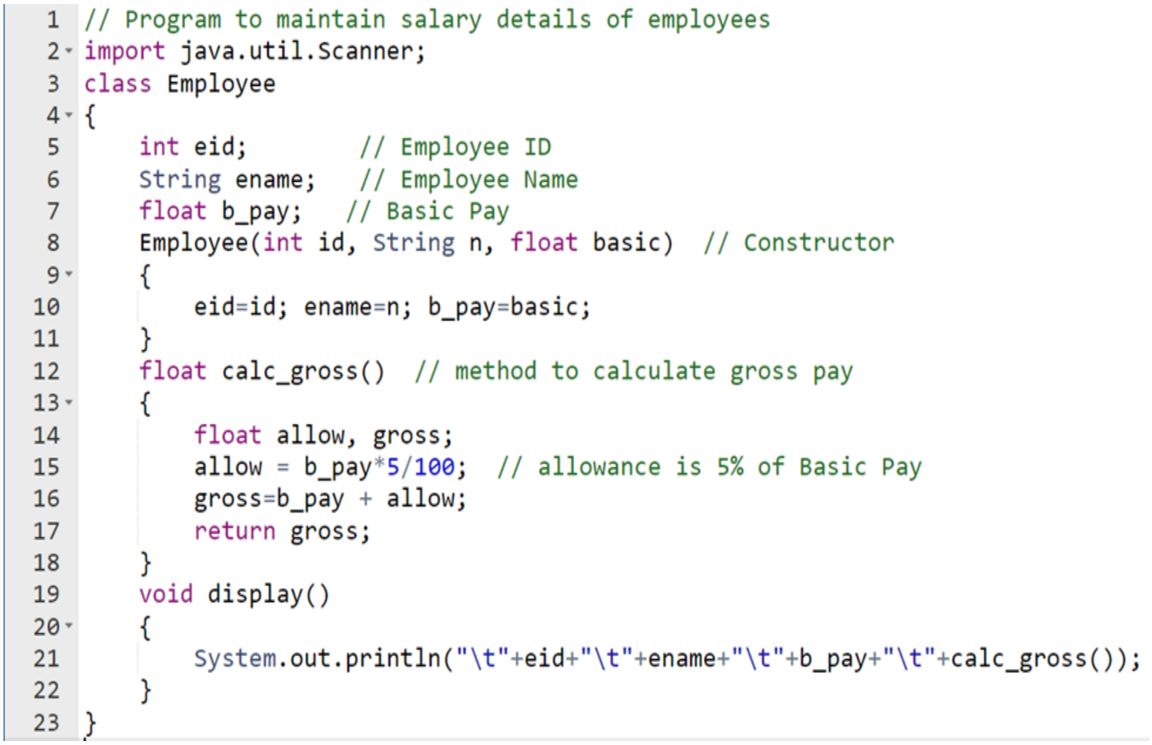
```
for ( int  x : sample)
     System.out.println(x);
```

Example 2: to find the sum of elements of the array sample

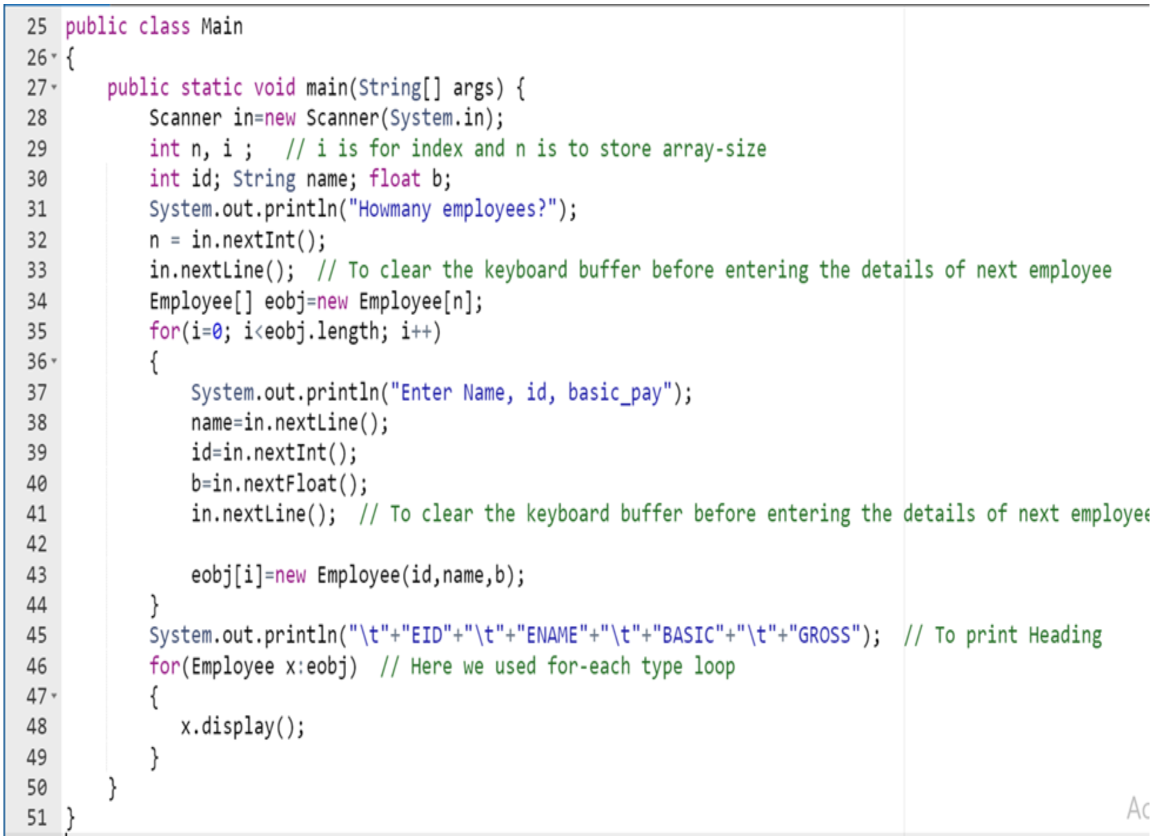
```
int  x, sum;
sum = 0;
for ( x : sample)
      sum  = sum + x;
System.out.println(sum);
```

Sample program programs using array of objects

1) Program to maintain (read, calculate total & grade, print) the mark details of students in a class
2) Program to maintain the salary details (read, calculate gross salary, print) of employees in a company



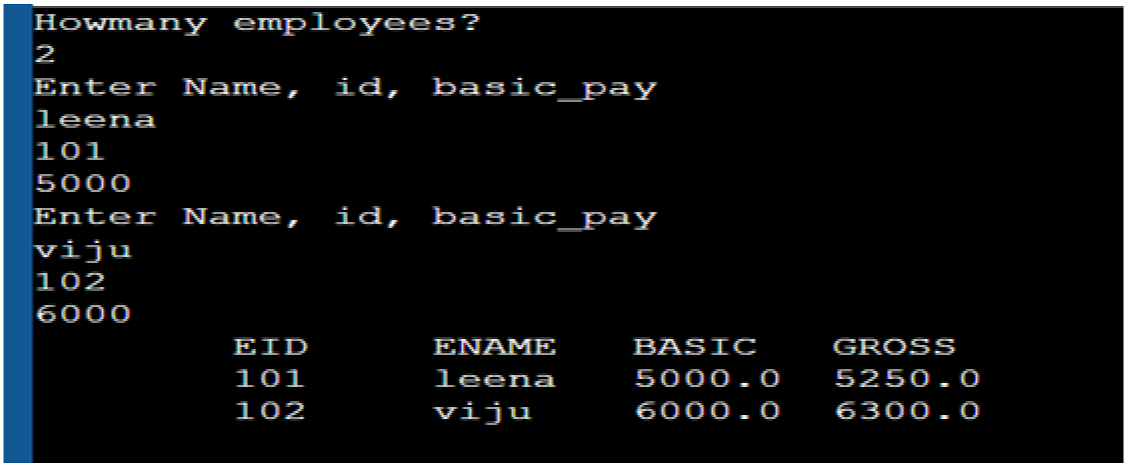
```
// Program to maintain salary details of employees
import java.util.Scanner;
class Employee
{
    int eid;        // Employee ID
    String ename;   // Employee Name
    float b_pay;   // Basic Pay
    Employee(int id, String n, float basic)  // Constructor
    {
        eid=id; ename=n; b_pay=basic;
    }
    float calc_gross()  // method to calculate gross pay
    {
        float allow, gross;
        allow = b_pay*5/100;  // allowance is 5% of Basic Pay
        gross=b_pay + allow;
        return gross;
    }
    void display()
    {
        System.out.println("\t"+eid+"\t"+ename+"\t"+b_pay+"\t"+calc_gross());
    }
}
```



```java
public class Main
{
    public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        int n, i ;   // i is for index and n is to store array-size
        int id; String name; float b;
        System.out.println("Howmany employees?");
        n = in.nextInt();
        in.nextLine();  // To clear the keyboard buffer before entering the details of next employee
        Employee[] eobj=new Employee[n];
        for(i=0; i<eobj.length; i++)
        {
            System.out.println("Enter Name, id, basic_pay");
            name=in.nextLine();
            id=in.nextInt();
            b=in.nextFloat();
            in.nextLine();  // To clear the keyboard buffer before entering the details of next employee

            eobj[i]=new Employee(id,name,b);
        }
        System.out.println("\t"+"EID"+"\t"+"ENAME"+"\t"+"BASIC"+"\t"+"GROSS");  // To print Heading
        for(Employee x:eobj)  // Here we used for-each type loop
        {
           x.display();
        }
    }
}
```

Output



```
Howmany employees?
2
Enter Name, id, basic_pay
leena
101
5000
Enter Name, id, basic_pay
viju
102
6000
        EID     ENAME   BASIC   GROSS
        101     leena   5000.0  5250.0
        102     viju    6000.0  6300.0
```

Errors & Exceptions

- A mistake in a program lead to an error causing the program to produce unexpected results
- Errors in a program may be broadly classified into **two categories**:
    - Compile-time errors
    - Run-time errors

Compile-time errors :– These are the syntax errors that are detected and displayed by the java compiler. When there is compilation error, the compiler will not generate the byte code **(.class file**)

Common causes for syntax errors:- missing semicolon, missing brackets, misspelling of keywords and identifiers, undeclared variables, incompatible types etc.

Run time errors :- These are errors that may occur during program execution and produces wrong results due to wrong logic or may terminate due to errors such as stack overflow

Examples: Division by zero, Accessing array elements out of boundary, use negative size for an array, Use null reference of objects to access methods etc.

EXCEPTIONS

- An **exception** is a condition that is caused by a run-time error in the program.
- When a Java interpreter encounters a run-time error, it creates an exception object and throws it (to inform us about the error). If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program. If we want the program to continue with execution of the remaining code, then we should try to catch the exception object and take corrective actions by displaying proper error messages. This task is known as ***exception handling***.

Exception Handling Steps:

1) **Hit** the Exception (Find the problem)
2) **Throw** the exception (inform that the error has occurred)
3) **Catch** the exception (Receive the error information)
4) **Handle** the exception (Take corrective actions)

Exception Handling Mechanism in Java

- The basic concept of exception handling are throwing an exception and catching it as shown in figure above.
- Java exception handling is managed via **five** keywords:

1) try -

- program statements that we want to monitor are contained within **try** block
- if an exception occurs in try block, it is thrown

2) catch –

- Exception thrown from try block are caught by **catch** statement and handled in catch block

3) throw

- It is used to manually throw an exception

4) throws

- An exception that is thrown out of a method must be specified using **throws** keyword

5) finally

- Any mandatory or compulsory code to be executed while exiting from **try** block is mentioned in **finally** block.

General form of try ….catch exception handling block

```
try
{
        // block of code to monitor for errors
}
catch(ExceptionType  object)
{
        // Exception handling code
}
```

- When an exception occurs in the monitored block, the exception object will be thrown and caught by the object specified in catch block
- If no exception is thrown, **try** block will end normally and **catch** block will be bypassed
- Th type of the exception must match with the type specified in **catch**

```
// An example for demonstrating exception handling
// Division by zero exception

class ExceptionDemo1
{
        public static void main(String[] args) {
                int a = 25, b = 5,  c = 5;
                int temp;

                try
                {
                        temp=a/(b-c);
                   System.out.println("temp="+temp);
                }
                catch(Exception  e)
                {
                        System.out.println("Error is "+e.getMessage());
                }
        }
}
```

Using multiple catch

- It is possible to associate more than one catch clause with a tr.
- Each **catch** must catch a different type of exception

```
try
{
        // block of code to monitor for errors
}
catch(ExceptionType1  object)
{
        // Exception handling code for ExceptionType1
}
catch(ExceptionType2  object)
{
        // Exception handling code for ExceptionType2
}
```

Example: this example handles Division by zero (an **ArithmeticException**) & array index error (**ArrayIndexOutOfBoundException**)

```
class ExceptionDemo1
{
        public static void main(String[] args) {
                int[ ] x={ 100, 50, 30, 25, 75, 128, 256};
                int[ ] y= { 2, 0, 4, 4, 0,5} ;
                for(int  i=0;  i<x.length  ; i++)
                {
                try
                {
                        System.out.println( x[i] + " /  " + y[i] + " is "+  x[i] / y[i] );
                }
                catch(ArithmeticException e)
                {

                        System.out.println("Division by zero is not allowed!!! ");
                }
                catch(ArrayIndexOutOfBoundException  e)
                {
                        System.out.println("Array index exceed the boundary!!! ");
                }
                }
        }
}
```

Output

```
100 /  2 is 50
Division by zero is not allowed!!!
30 /  4 is 7
25 /  4 is 6
Division by zero is not allowed!!!
128 /  5 is 25
Array index exceed the boundary!!!
```

Throwing an exception (throw)

- The examples that we found earlier are catching the exceptions generated automatically by JVM
- **It is possible to manually throw an exception**
- First we have to create the object of Exception class using new operator. Then throw the object

Syntax

```
        throw new ExceptionClass() ;
```

Example:

This example throws an exception if the mark entered is greater than 100 or less than 0

```
import java.util.Scanner;
class ThrowDemo
{
        public static void main(String[] args) {
                int mark;
                Scanner in=new Scanner(System.in);
                try
                {
                        System.out.print( "enter mark:");
                        mark=in.nextInt();
                        if(mark<0 || mark>100)
                        {
                        throw new ArithmeticException();
                        }
                   System.out.println("Mark="+mark);
                }
                catch(ArithmeticException e)
                {
                        System.out.println ("Mark Cannot  > 100 or -ve !!!!");
                }
                System.out.println("After Try/catch block .....");
        }
}
```

Example-2:

```
import java.util.Scanner;
import java.lang.Exception;
class MarkException extends Exception
{
        String errorMessage;
        MarkException(String msg)
        {
        errorMessage=msg;
        }
}
class ThrowDemo
{
        public static void main(String[] args) {
                int mark;
                Scanner in=new Scanner(System.in);
                try
                {
                        System.out.print( "enter mark:");
                        mark=in.nextInt();
                        if(mark<0)
                        {
                        throw new MarkException("Mark cannot be negative !!!");
                        }
                        else if(mark>100)
                        {
                        throw new MarkException("Mark cannot be more than 100 !!!");
                        }
                        else
                        {
                      System.out.println("Mark="+mark);
                        }
                }
         catch(MarkException e)
        {
                        System.out.println (e.errorMessage);
        }
                System.out.println("After Try/catch block .....");
        }
}
```

Use of finally block

- **catch** blocks are executed only when a matching error occurs in the corresponding try block
- It is possible to write a compulsory code to be executed regardless of whether an error occurred or not in the **try** block. Those codes are written inside **finally** block at the end of **try/catch** as shown below

- **finally** block is used to perform some house-keeping operations such as **closing files, releasing memory** etc.

```
try
{
    // block of code to monitor for errors
}
catch(ExceptionType1  object)
{
    // Exception handling code for ExceptionType1
}
catch(ExceptionType2  object)
{
    // Exception handling code for ExceptionType2
}
// ….
finally
{
    // finally code
}
```

Example with finally caluse

```
 class ExceptionDemo1
{
    public static void main(String[] args)
           {
            int[ ] x={ 100, 50, 30, 25, 75, 128, 256};
            int[ ] y= { 2, 0, 4, 4, 0,5} ;
           for(int  i=0;  i<x.length  ; i++)
                {
            try
                {
                    System.out.println( x[i] + " /  " + y[i] + " is "+  x[i] / y[i] );
                }
                catch(ArithmeticException e)
                {
                   System.out.println("Division by zero is not allowed!!! ");
                }
                catch(ArrayIndexOutOfBoundException  e)
                {
                    System.out.println("Array index exceeds the boundary!!! ");
                    }
                finally
                {
                    System.out.println(“Leaving try block !!!! “);
                }
    }
}
```

Using throws

- If a method generates an exception that it does not handle using **try/catch** blocks, it must declare that exception in a **throws** clause as shown below:

```
return-type   method_name(parameter-list)  throws Exception-List
{
     // body of the method
}
```

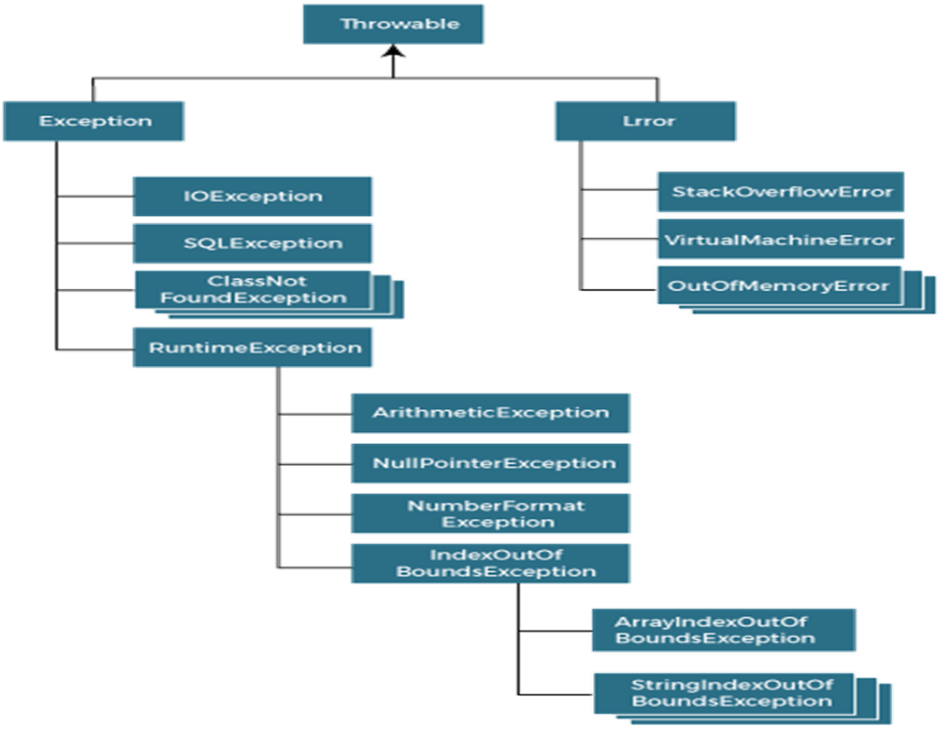
- Exceptions that are subclasses of **Error** or **RuntimeException** don't need to be specified in a throws clause
- All other exceptions do need to be declared. Failure to do so will generate a compile-time error

Example

```
public static void main( String[] args)  throws IOException
{
     // body of the method
}
```

Java Exception Hierarchy

- In Java, all exceptions are subclasses of **Throwable**



-

Categories of Built-in Exceptions in Java

There are mainly **two types** of exceptions: checked and unchecked

1) **Checked Exceptions** :-

- The classes that directly inherit the **Throwable** class **except RuntimeException** and **Error** are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.
- Checked exceptions should be explicitly handled in the program code with the help of **try – catch** block or using **throws** keyword

2) **Unchecked Exceptions :-**

- The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
- These exceptions are typically handled by JVM (Java Virtual machine).
- They are not essentially handled in the program code.
- Error is also considered as Unchecked Exception

Common Java Exceptions

Exception Type (Exception classes)	**Cause of Exception**
ArithmeticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	Caused by wrong array indexes
FileNotFoundException	Caused by an attempt to access a non-existent file
IOException	Caused by general I/O failure, 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 a new object

Using Scanner class for Interactive input in Java

- To give input data interactively from key board while executing a Java program, use the following **methods** of **Scanner class** defined in **java.util package**
    - **nextInt() - to read an integer value**
    - **nextFloat() – to read a float value**
    - **nextLine() – to read strings**
    - **nextByte() – to read next input from keyboard as a byte**
    - **nextBoolean() – To read next input as a Boolean value**
    - **nextLong() -**
    - **nextShort() -**

Steps:

1. import java.util.Scanner; class in the import section of the program

2. Create an object of Scanner class

Scanner obj_name = new Scanner(System.in);

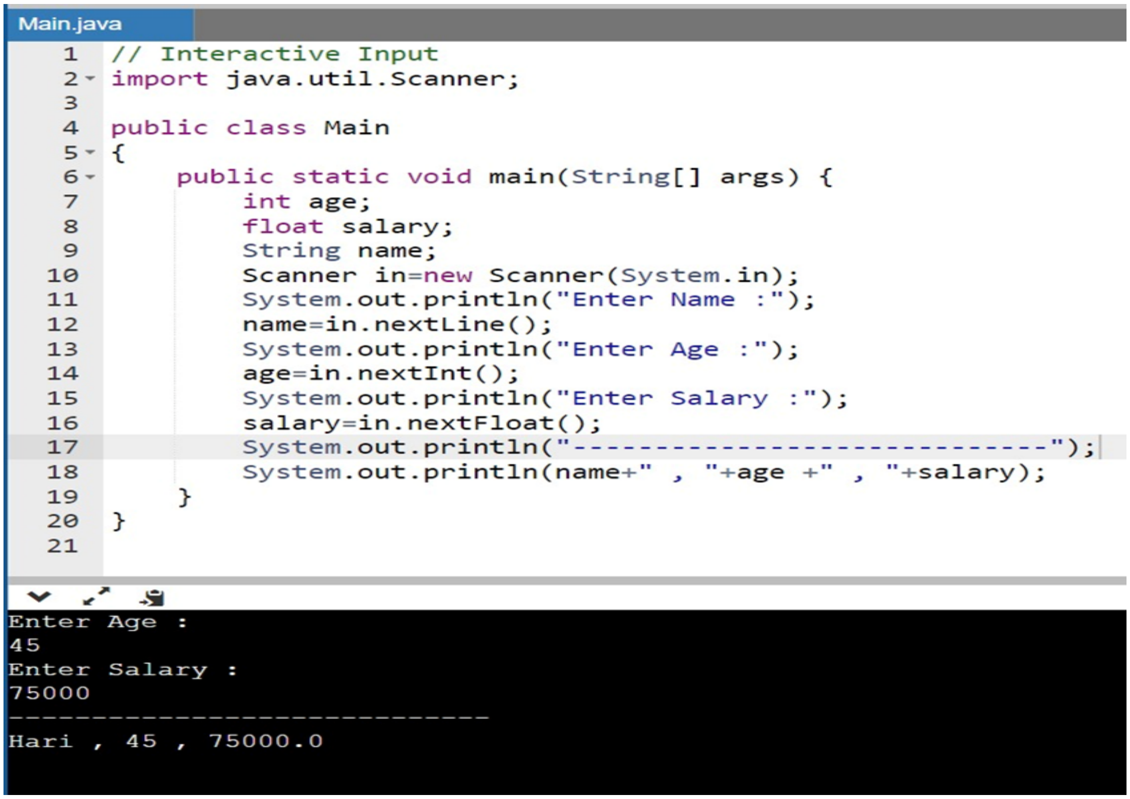
3. Then call the required method using the Scanner object

integer_variable = obj_name.nextInt();

float_variable = obj_name.nextFloat();

String_object = obj_name.nextLine();

Example:



Main.java

```
// Interactive Input
import java.util.Scanner;

public class Main
{
    public static void main(String[] args) {
        int age;
        float salary;
        String name;
        Scanner in=new Scanner(System.in);
        System.out.println("Enter Name :");
        name=in.nextLine();
        System.out.println("Enter Age :");
        age=in.nextInt();
        System.out.println("Enter Salary :");
        salary=in.nextFloat();
        System.out.println("-----------------------------");
        System.out.println(name+" , "+age +" , "+salary);
    }
}
```

```
Enter Age :
45
Enter Salary :
75000
-----------------------------
Hari , 45 , 75000.0
```