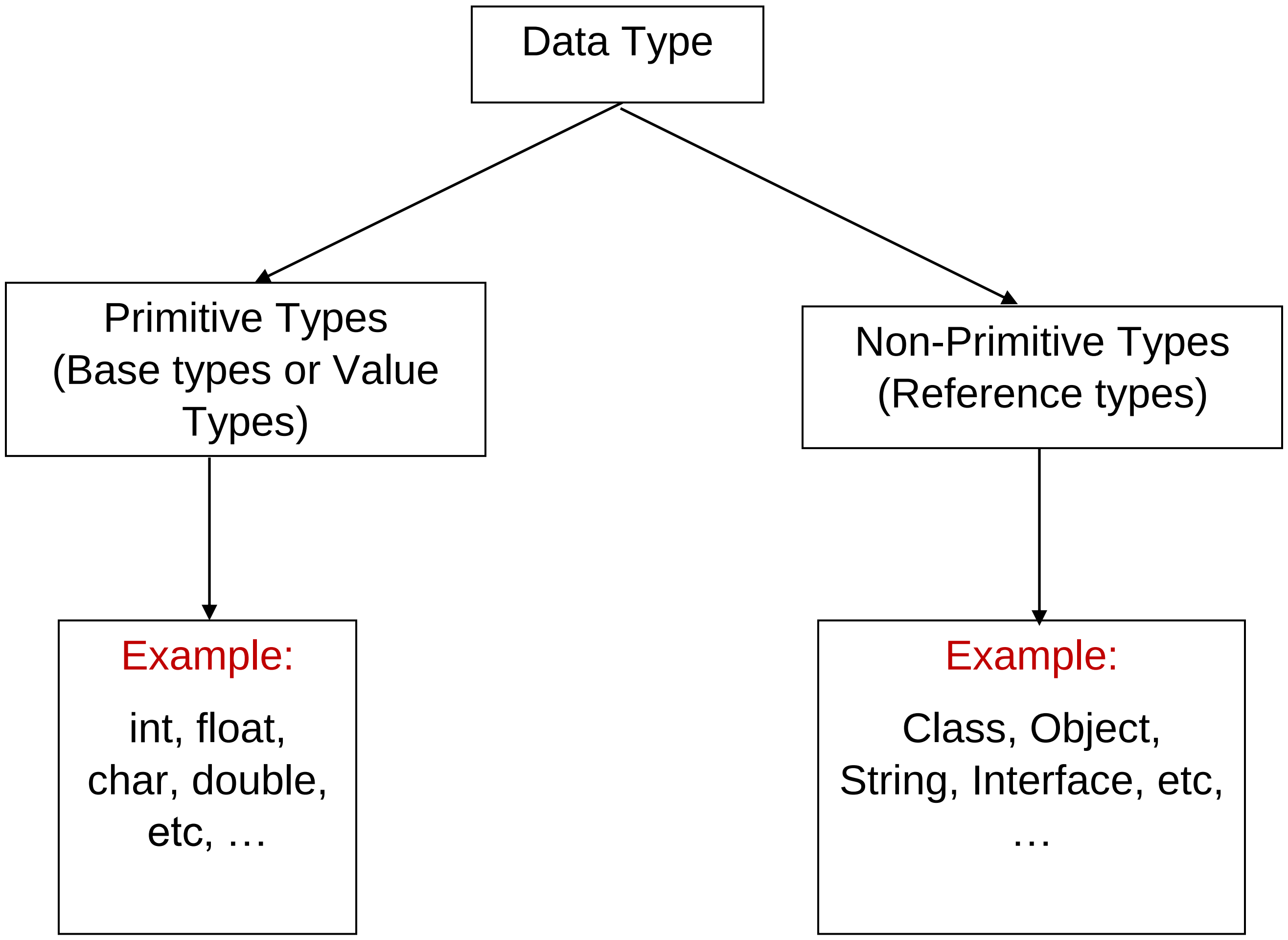
JAVA INTRODUCTION II

DATA TYPES

- Data types plays an important role in the definition of variables
- It specifies the type of data that a variable can store data like int, float, char, etc, …
- Java has two types. They are primitive types (base types) and non-primitive types (reference / object types)

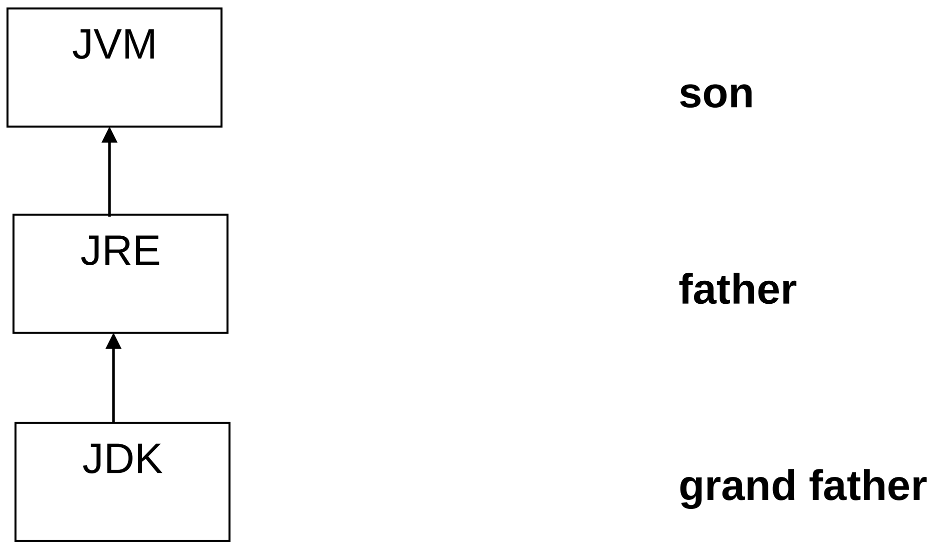

Data Type
Primitive Types (Base types or Value Types)
Non-Primitive Types (Reference types)
Example:
int, float, char, double, etc, …
Example:
Class, Object, String, Interface, etc, …

APPLICATIONS OF JAVA

- Java is not only used in software applications, but it is also used in designing hardware controlling software components
- Following are some of the usage of Java
  - Desktop Applications such as acrobat reader, media player, antivirus etc.
  - Web Applications
  - Mobile Operating System like Android
  - Embedded System
  - Robotics
  - Banking applications
  - Games

JDK, JVM, JRE

- For the development and execution of java applications / applets, we need to know about the software tools and runtime tools such as JDK, JRE and JVM
  1. JDK → used for the development of java applications
  2. JRE → used for the runtime of java applications
  3. JVM → used for the execution of java byte codes to target output



JDK (Java Development Kit)

- It is a superset of JRE
- It is a software development environment used for developing java applications and applets (provides an environment to user for developing java applications / applets)
- It includes
    1. Java Runtime Environment (JRE)
    2. An interpreter / loader (java.exe)
    3. A compiler (javac.exe)
    4. A document generator (Javadoc)
    5. Other tools needed in java development, etc, …
- JDK=JRE+JVM

JRE (Java Runtime Environment)

- Also called as Java Runtime (Runtime Environment)
- It includes
  1. Java Virtual Machine (JVM)
  2. Standard class libraries (java.lang.*, java.util.*, java.sql.*, etc ,…)
  3. Other components to run (execute) java applications / applets
- It does not contain java compiler, interpreter and other software tools needed to write java program

Installation

- No need to install JRE, because JDK usually consists of both development (Ex. compiler, interpreter, etc, …) and runtime environments (Ex. JVM) in it
- If JDK is installed in a computer, then JRE will be installed automatically along with JDK

Storage

- It is smaller than JDK. So it needs less storage space.
- It will be installed automatically, if JDK is installed in a machine (computer)

Usage

- JRE must be installed on machine in order to execute pre compiled Java Programs (.class) / java bytecode

DIFFERENCE BETWEEN JDK AND JRE

S.N	JDK	JRE
1.	Provides tool environment for developing java applications	Provides a runtime environment for execution of java applications
2.	It includes compiler and interpreter	It does not contain compiler and interpreter
3.	JDK=JRE+JVM	JRE=JVM+ standard libraries (java.lang.*, java.util.*, java.io.*, etc, …)
4.	It is mainly targeted for java development	It can't compile java programs with it. It is targeted for java runtime support

JVM (Java Virtual Machine)

- It is virtual machine that runs the java byte code (.class) / pre compiled java programs
- It is also called as runtime engine / runtime manager
- It provides a platform independent way of executing java code. That means that, compile once in any machine and run it anywhere (any machine)
- The Oracle JVM is developed using C programming language
- There are many JVM implementations developed by different organizations

Installation

- No need to install JVM separately, because it is included in the JRE
- JVM will be installed automatically, if JDK is installed in a computer.

Byte Code Support

- The JVM doesn't understand java source code (.java), so that, the source code is compiled into bytecode (.class) which can be understood by the JVM.
- The JVM provides a platform-independent way of executing code, thus making Java platform independent.

Runtime Support

- Actually, JVM runs the java program and uses the standard class libraries (Ex. java.util.*, java.lang.*, etc, …) and other supporting files provided in JRE (Java Runtime Environment)
- So if we want execute java programs, then we need to install JRE in a system

Usage

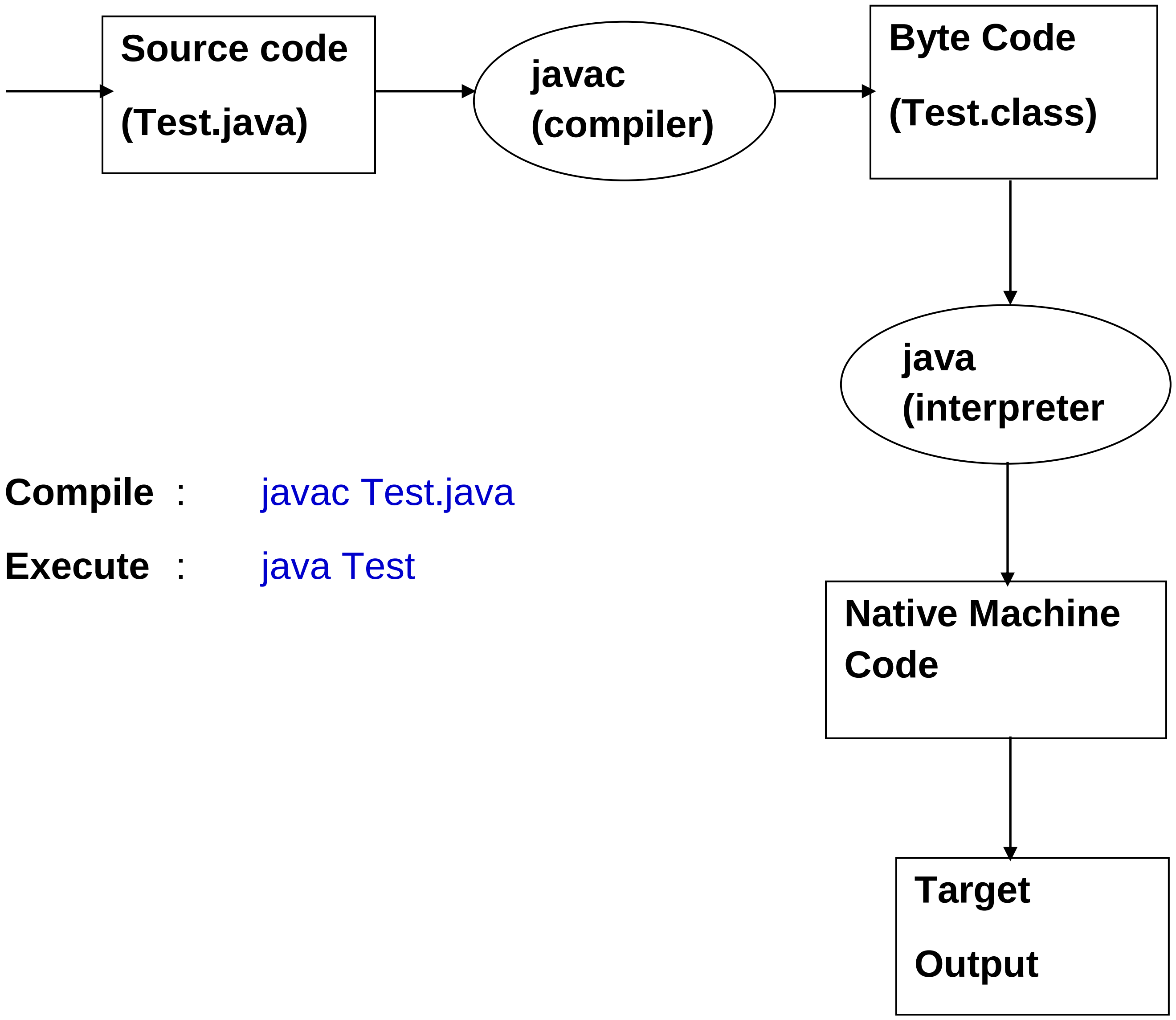
- JVM is used to execute java byte code (.class) to target output

Tasks of JVM

- It performs the following main tasks
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

COMPILATION & EXECUTION (EXECUTION PROCESS)

- Java supports both compilation and interpretation
- In compilation stage, the java source code is converted to byte code (.class)
- In the execution stage, the JVM (java interpreter) converts byte code to target output



Compile : javac Test.java

Execute : java Test

COMPILATION

- The java compiler (javac) translates the java program into byte code / object code
- It produces **.class file**
- Java byte code is not the machine language for any traditional CPU. So Interpreter translates bytecode to machine language (native machine code) and executes it

SYNTAX

javac <filename.java>

Example

➔ javac Test.java

➔ Test.class (byte code will be created, after the successful compilation)

EXECUTION

- The JVM (Java Virtual Machine) converts the byte code into native machine code and return the target output with help of JIT compiler (Just in Time)

JIT Compiler

- Compiled when needed (during run time)
- JIT stands for Just-in-Time means that byte code gets compiled when it is needed, not before runtime
- JIT compiler is a component of JVM
- It improves the performance of JVM / java applications by compiling bytecodes to native machine code at run time.
- It is enabled by default and it does need processor time and memory usage

Working Operation

- After the byte code (which is architecture neutral) has been generated by the java compiler (javac), the execution will be handled by the JVM (java).
- The byte code will be loaded into JVM by the loader and then each byte instruction is interpreted (using java interpreter)
- When we need to call a method multiple times, we need to interpret the same code many times and this may take more time than is needed. So that JIT compilers are used at runtime to reduce the repeated instructions (calling same variable /method multiple times)
- When the byte code is loaded into JVM (its run time), the whole code will be compiled rather than interpreted, thus saving time.
- JIT compilers works only on run time, so we do not have any binary output.

Use of JIT Compiler

- It is used to improve the performance of JVM by dynamically compiling / translating java byte code into native machine code during execution time
- JIT increase the program execution speed

DIFFERENCE BETWEEN JVM AND JIT

S.N	JVM	JIT COMPILER
1.	The main goal of JVM is to provide platform independence by the execution of java byte codes	The main purpose of JIT compiler is used to improve the performance of JVM by the translation java byte code to native machine code
2.	JVM is Java Virtual Machine	JIT is a part of JVM. It is a Just-in-Time compilation

SYNTAX

java <byte code name>

Example

```
➔ java Test

➔ Target Output
```

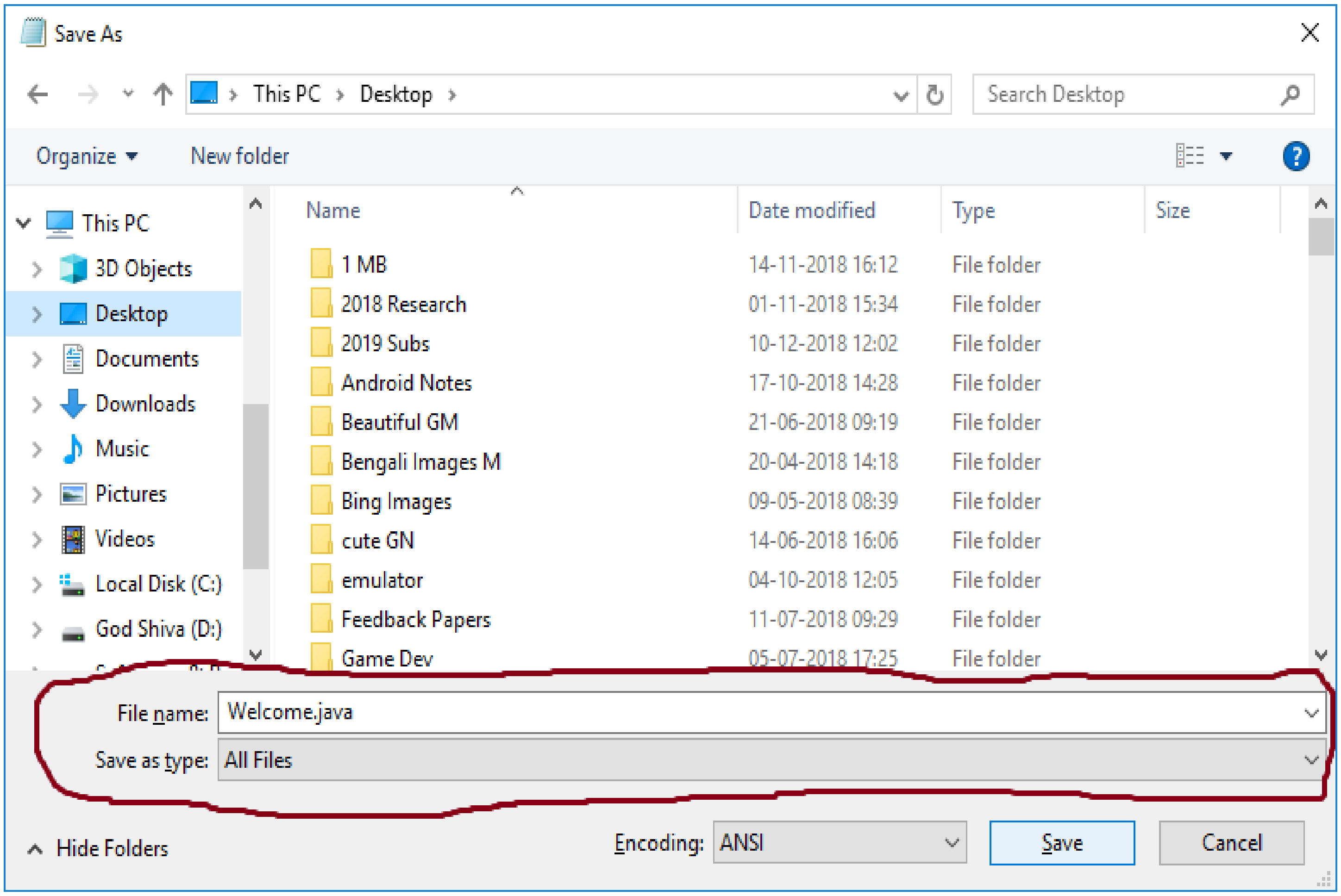
I. HELLO WORLD USING MANUEL WAY

(USING EDITOR)

BASIC REQUIRMENT STEPS

1. Install the JDK if you don't have installed it, download the JDK and install it.	
2. Set path of the jdk/bin directory.	(set path=<path-of-jdk/bin>
3. Create the java program	(.java)
4. Compile the java program	(javac)
5. Run the java program	(java)

1. FILE CONFIGURATION (Saving a file)



Name	Date modified	Type	Size
1 MB	14-11-2018 16:12	File folder	
2018 Research	01-11-2018 15:34	File folder	
2019 Subs	10-12-2018 12:02	File folder	
Android Notes	17-10-2018 14:28	File folder	
Beautiful GM	21-06-2018 09:19	File folder	
Bengali Images M	20-04-2018 14:18	File folder	
Bing Images	09-05-2018 08:39	File folder	
cute GN	14-06-2018 16:06	File folder	
emulator	04-10-2018 12:05	File folder	
Feedback Papers	11-07-2018 09:29	File folder	
Game Dev	05-07-2018 17:25	File folder	

File name: Welcome.java

Save as type: All Files

Encoding: ANSI

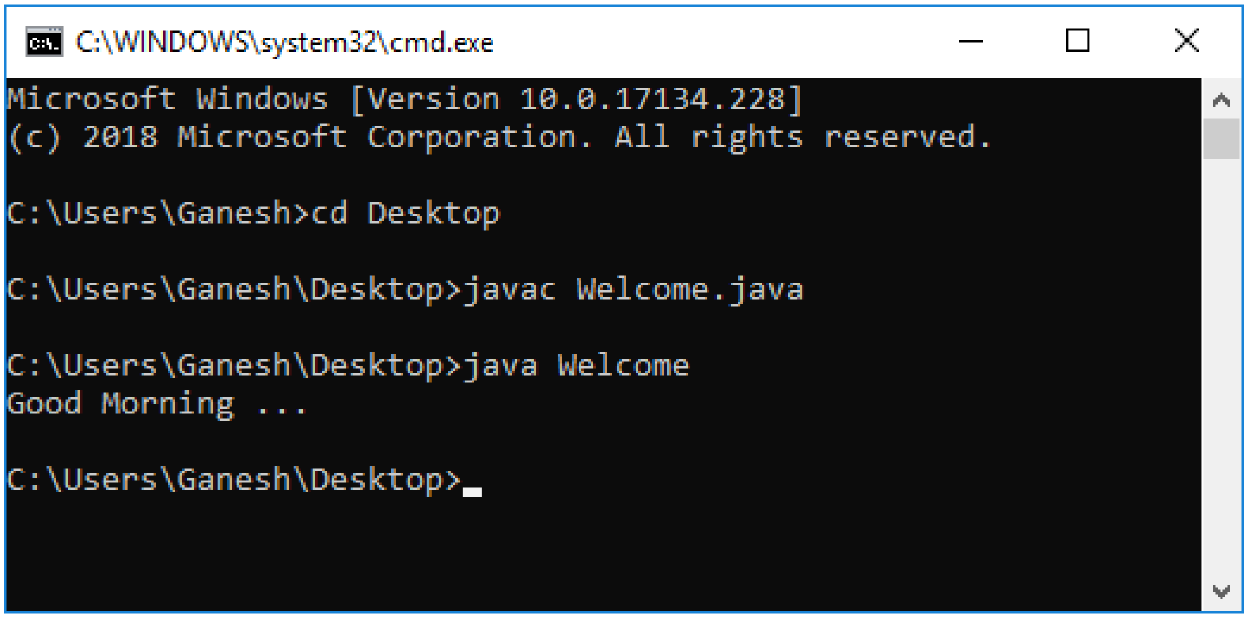
Language	:	**java**
Application Type	:	**Console Application**
Tool	:	**Notepad**
Compiler	:	**javac**
Interpreter	:	**java**

2. SOURCE CODE

(Welcome.java)

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

3. OUTPUT

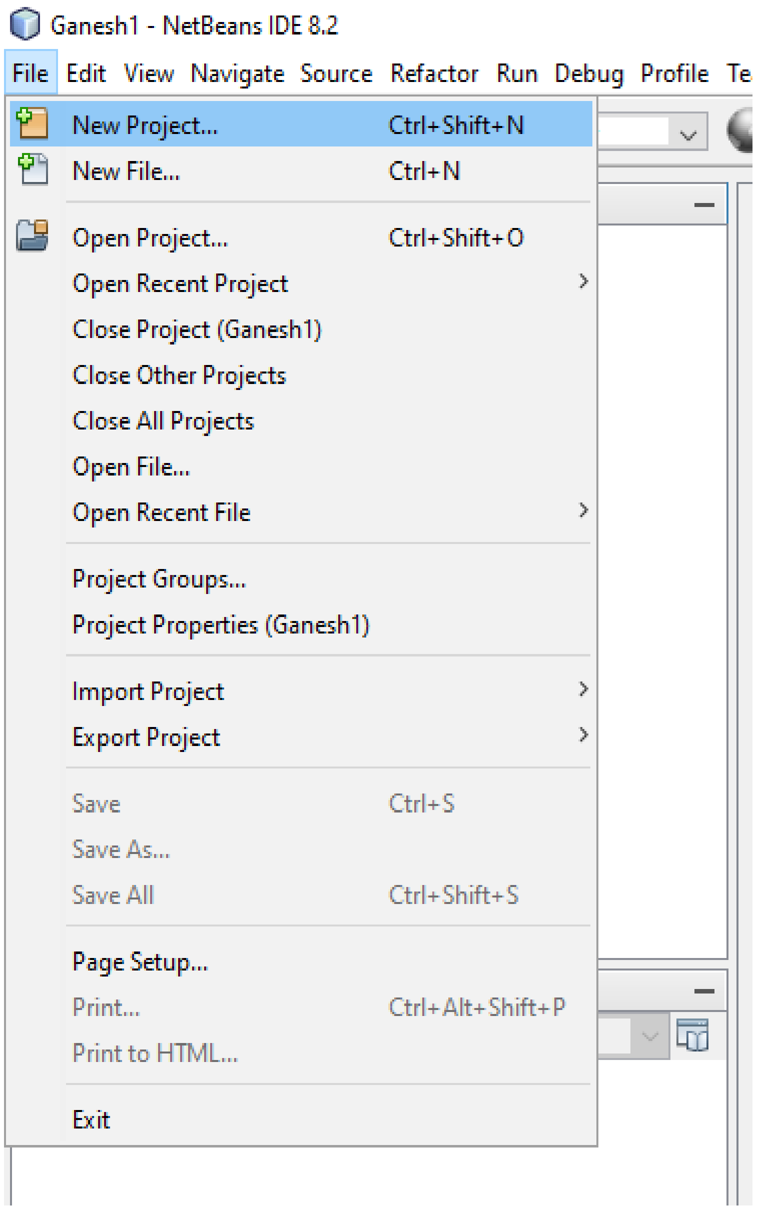


```
Microsoft Windows [Version 10.0.17134.228]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ganesh>cd Desktop

C:\Users\Ganesh\Desktop>javac Welcome.java

C:\Users\Ganesh\Desktop>java Welcome
Good Morning ...

C:\Users\Ganesh\Desktop>_
```

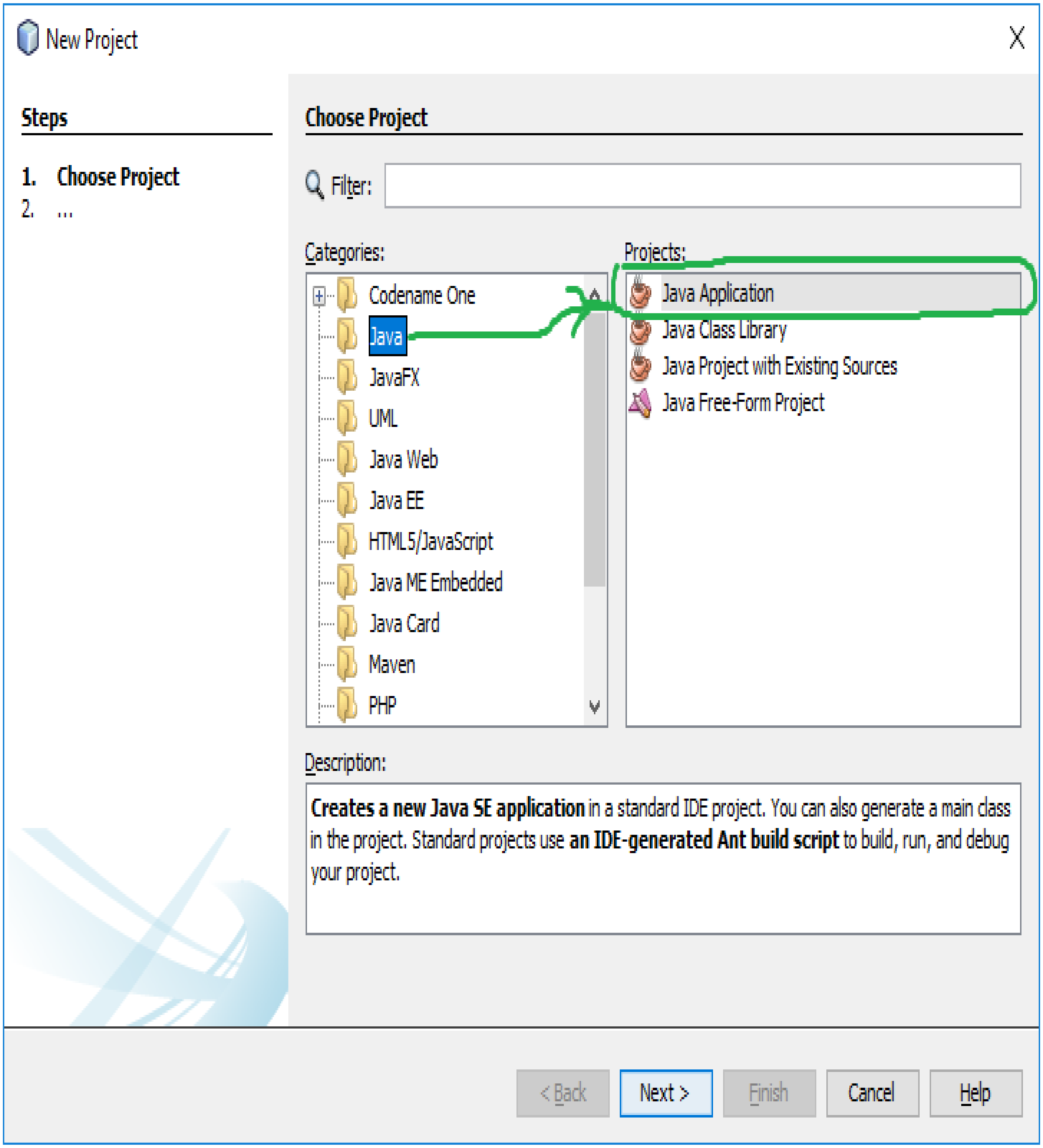
II. HELLO WORLD USING AUTOMATIC WAY

(USING NETBEANS IDE)

STEP 1: PROJECT CREATION IN NETBEANS IDE (8.2)

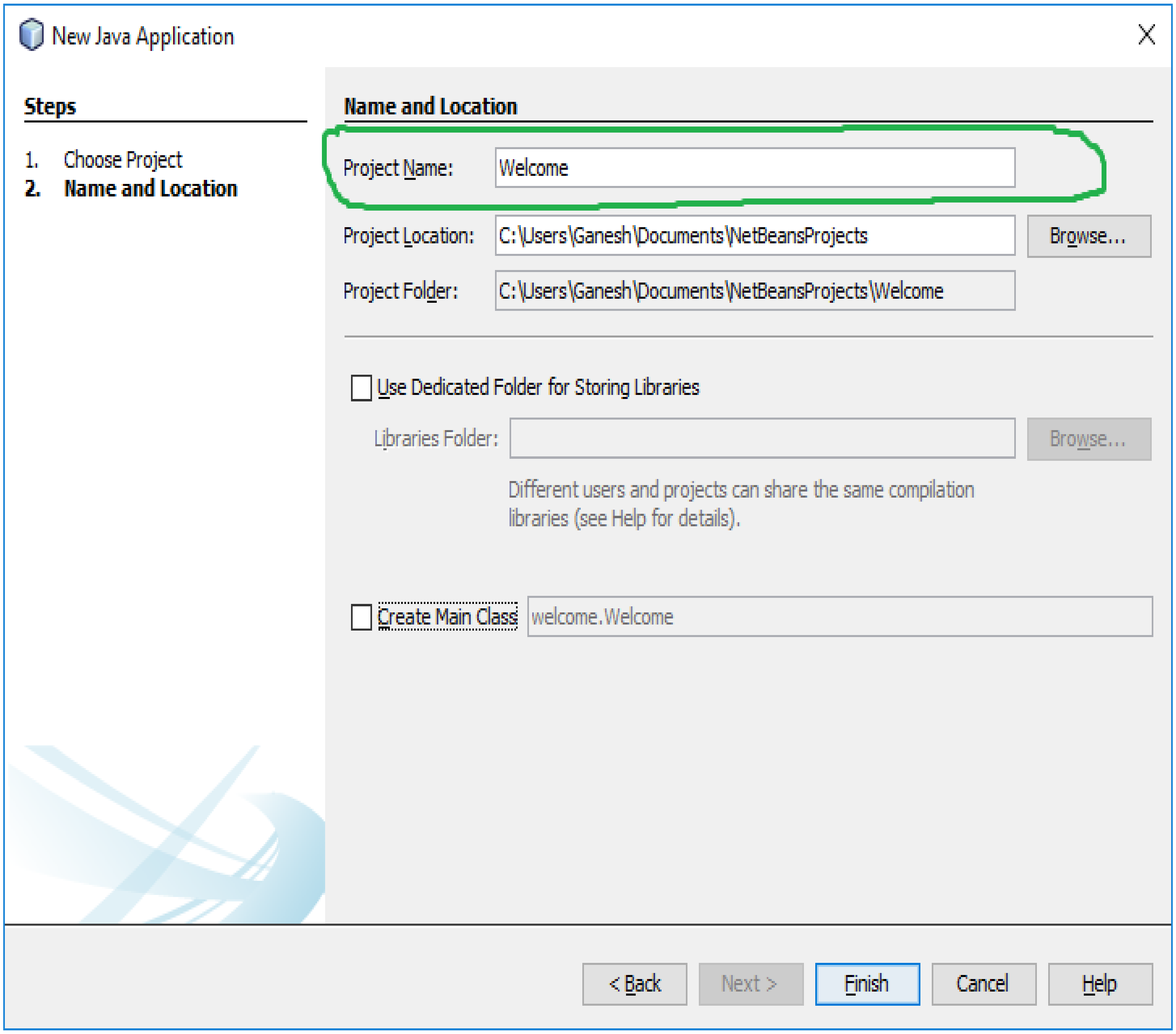


STEP 2: PROJECT TYPE SELECTION



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

STEP 3: SETTING PROJECT NAME & ITS LOCATION



New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: Welcome

Project Location: C:\Users\Ganesh\Documents\NetBeansProjects

Project Folder: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome

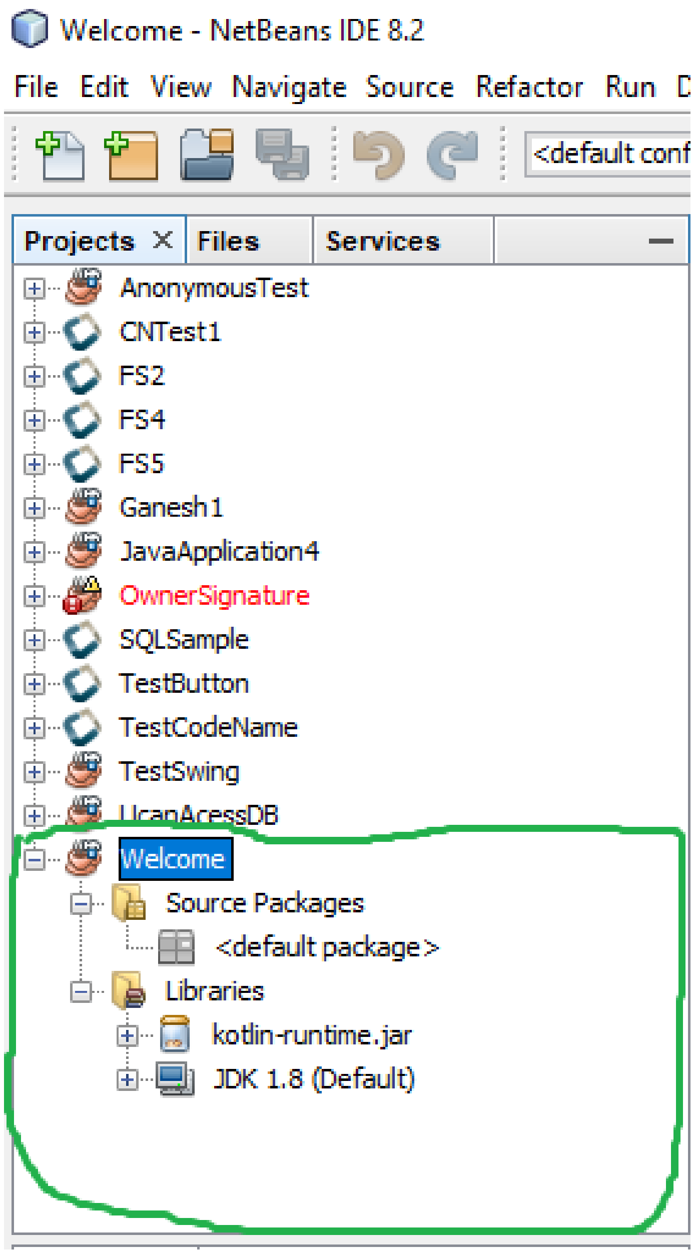
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

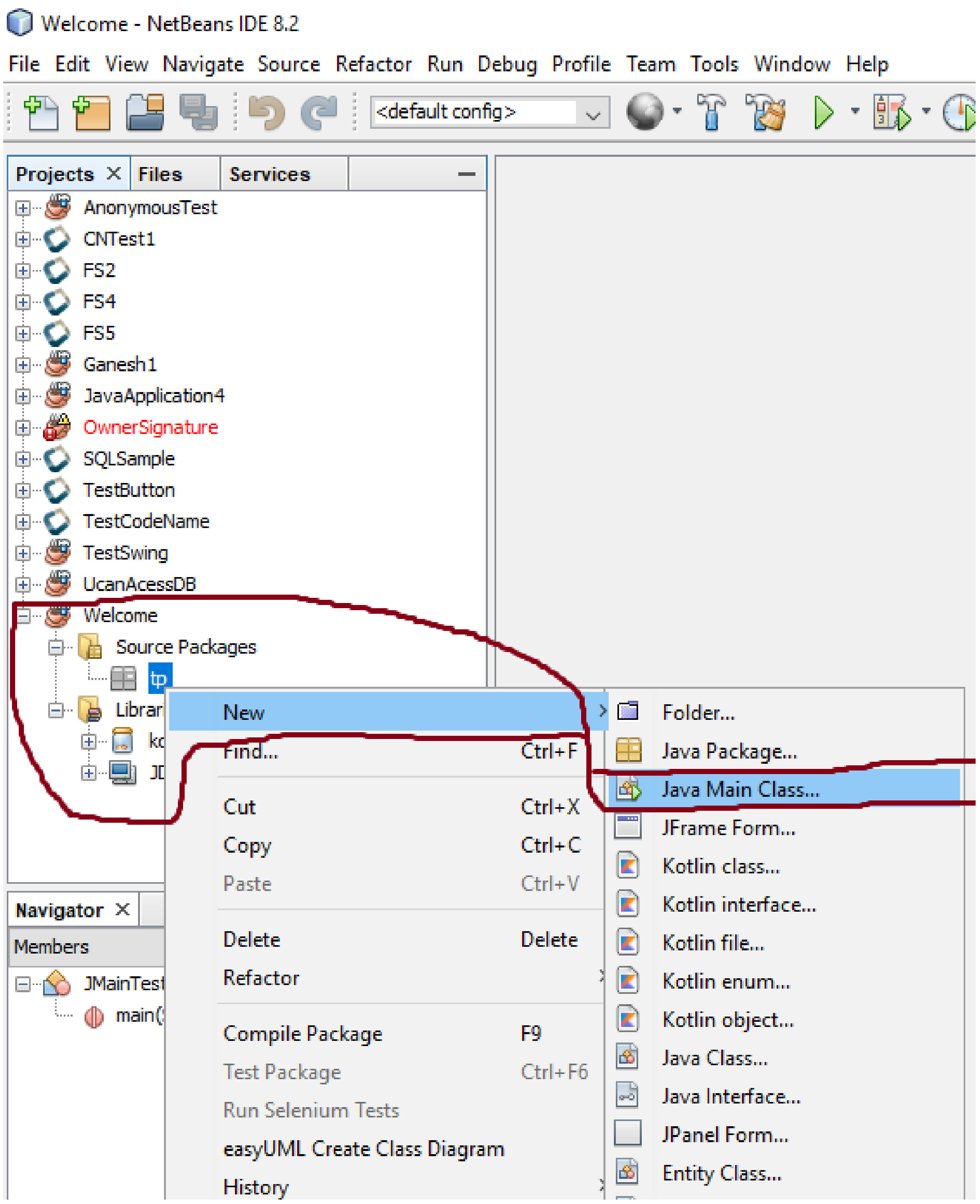
Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class welcome.Welcome

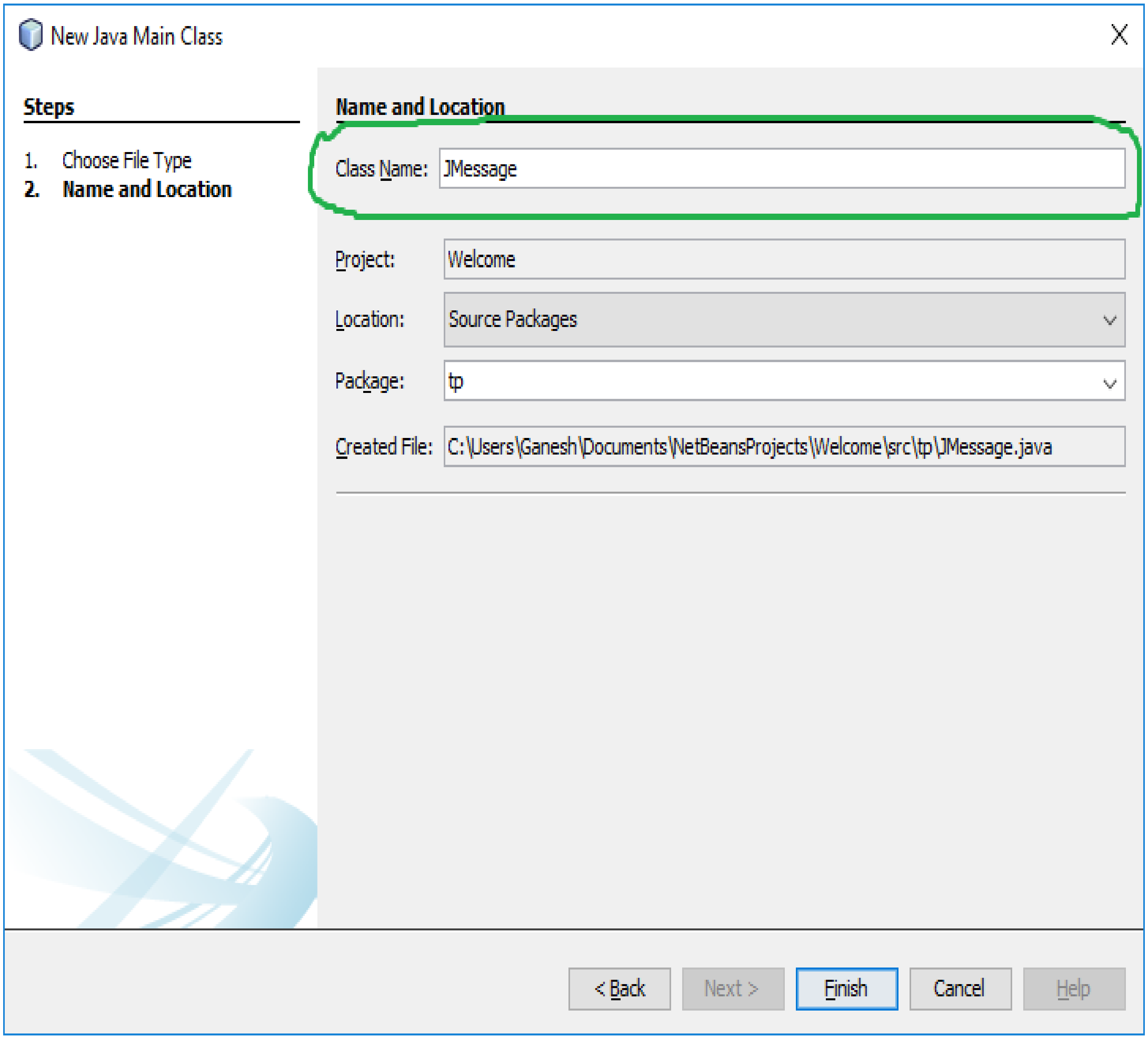
STEP 4: PROJECT VERIFICATION IN NETBEANS



STEP 5: ADD MAIN CLASS TO CURRENT PROJECT FOLDER



STEP 6: SET NAME TO MAIN CLASS


New Java Main Class

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: JMessage

Project: Welcome

Location: Source Packages

Package: tp

Created File: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\src\tp\JMessage.java

NOTE

- Don't add .java extension when creating main class / other class in the project. Because Netbeans IDE will automatically add .java extension.

Language	:	**java**
Application Type	:	**Console Application**
Tool	:	**NetBeans IDE**
Compiler	:	**javac**
Interpreter	:	**java**

1. SOURCE CODE

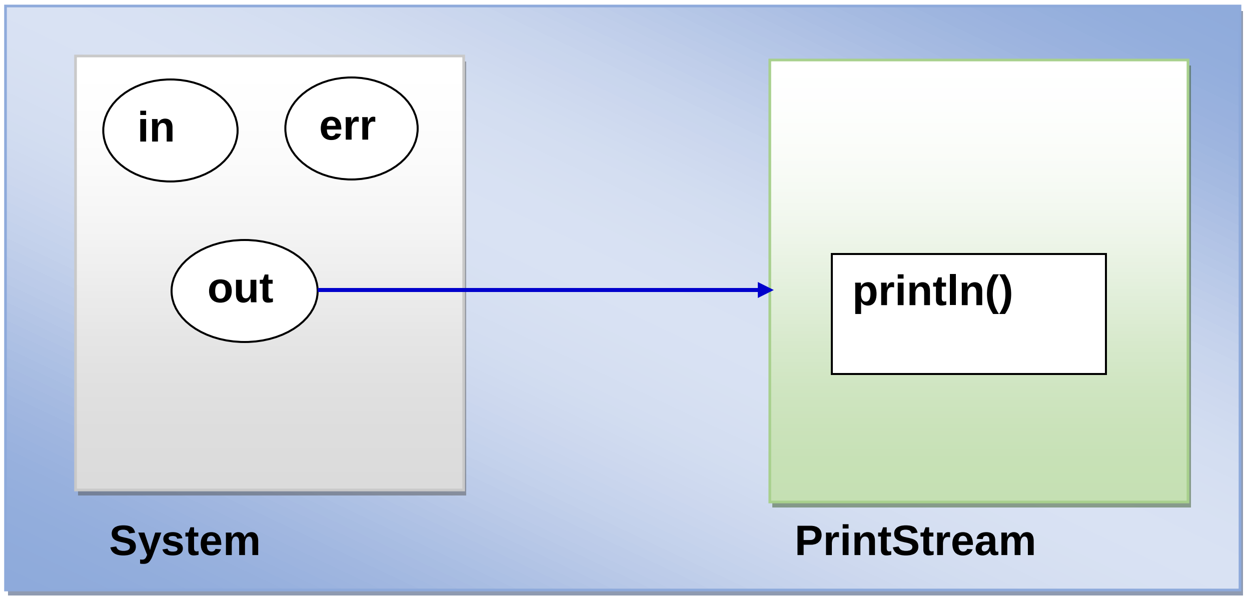
(JMessage.java)

```
package tp;              // user defined package
public class JMessage
{
    public static void main(String[] args)
    {
            System.out.println("Welcome to chennai ...");
    }
}
```

Where,

System	→	built-in class (Pre-defined class)
out	→	static object type of PrintStream class
println()	→	print string (text message)
class	→	reserved keyword used to define a class
public	→	access modifier which gives global scope (accessed anywhere in the programming)

void	→	return type of the main() method. It does not return anything
main	→	represents the starting point of the program (starts the program execution)
String[] args	→	string array which is used to receive the command line inputs. **By default**, the inputs of command line arguments **must be string array** (String [] args)
static	→	it represents main() as a static method. The main advantage of using static method is that, there is no need create an object to call the static methods
System.out.println()	→	used for print statement. System is a built-in class, out is an object of PrintStream class and println() is an instance method of PrintStream class.



System

- It is placed in java.lang.* package
- It is used for input / output operations
- It includes built-in static class fields and methods
- Static Fields are

  1. In → used for getting input
  2. Out → used for printing output on console
  3. Err → used for printing errors as a output

in

- It is used to read input from standard console (Ex. keyboard)
- in is a static member of the System class and is an instance of java.io.PrintStream class
- First, it is reference type (Object type) of PrintStream class
- PrintStream is Built-in class which is placed in java.io.* package
- Here in is actually defined as a static object in System class.

out

- out is a static member of the System class and is an instance of java.io.PrintStream class
- First, it is reference type (Object type) of PrintStream class
- PrintStream is Built-in class which is placed in java.io.* package
- Here out is actually defined as a static object in System class.
- Here out object is used to send the data to standard output device (e.g. dos window / terminal / IDE terminal)

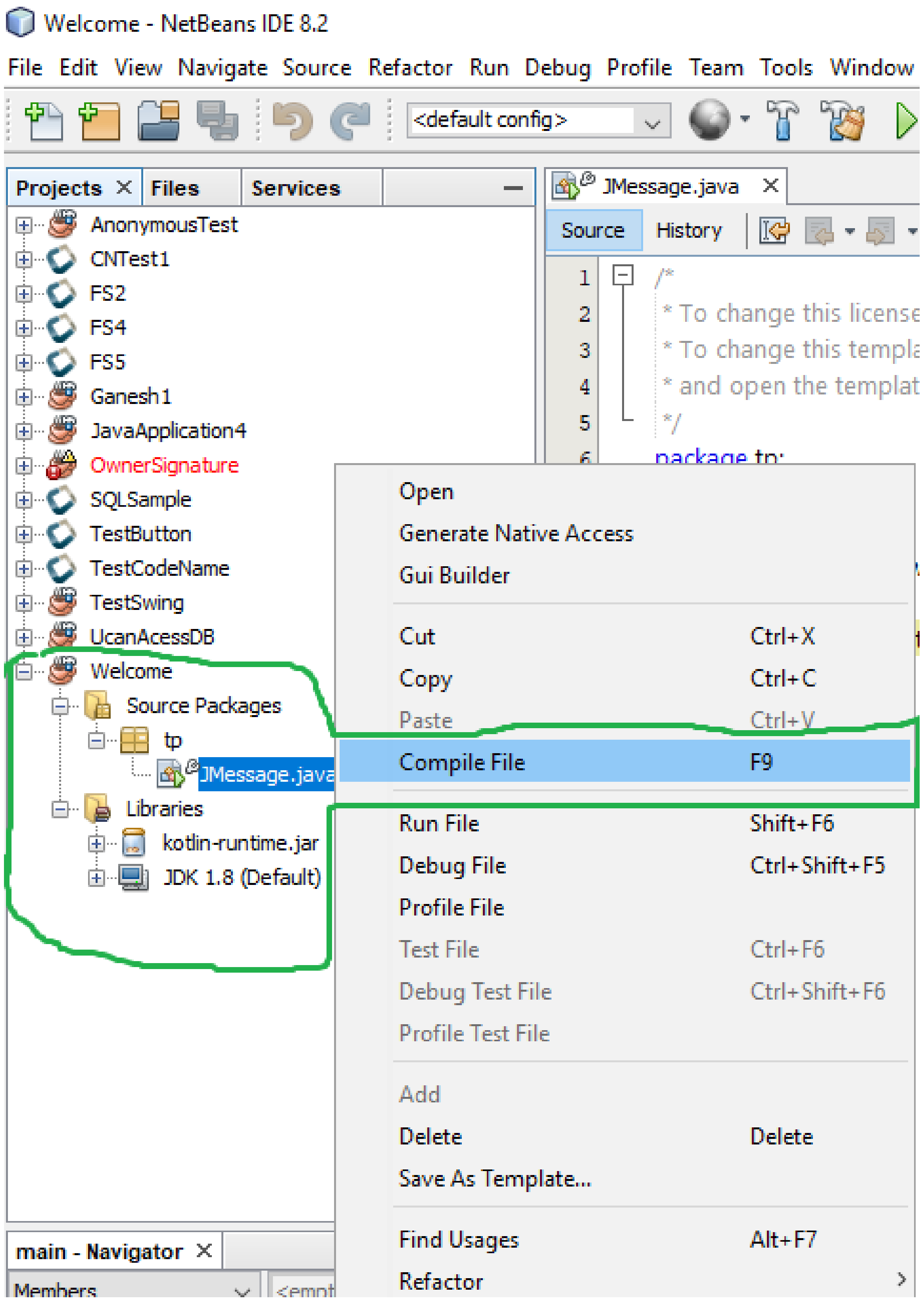
println()

- It is built-in instance method of PrintStream class
- PrintStream is Built-in class which is placed in java.io.* package
- Here it is used to print the message as a line.

err

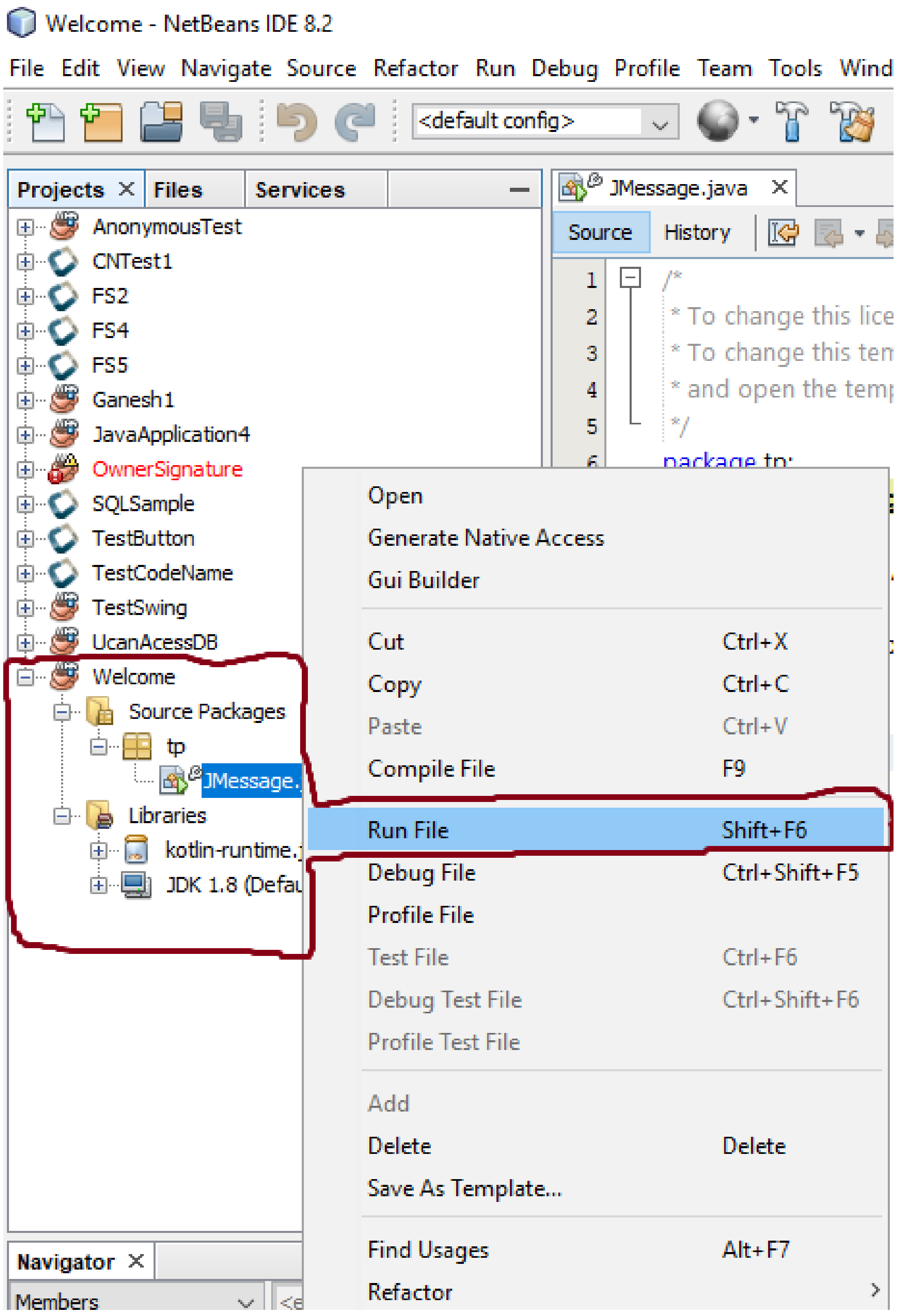
- It works like out object except it is normally only used to output error tasks (print / show error message on the output screen / console)
- It is first object of PrintStream class and second static object of System class

2. COMPILATION



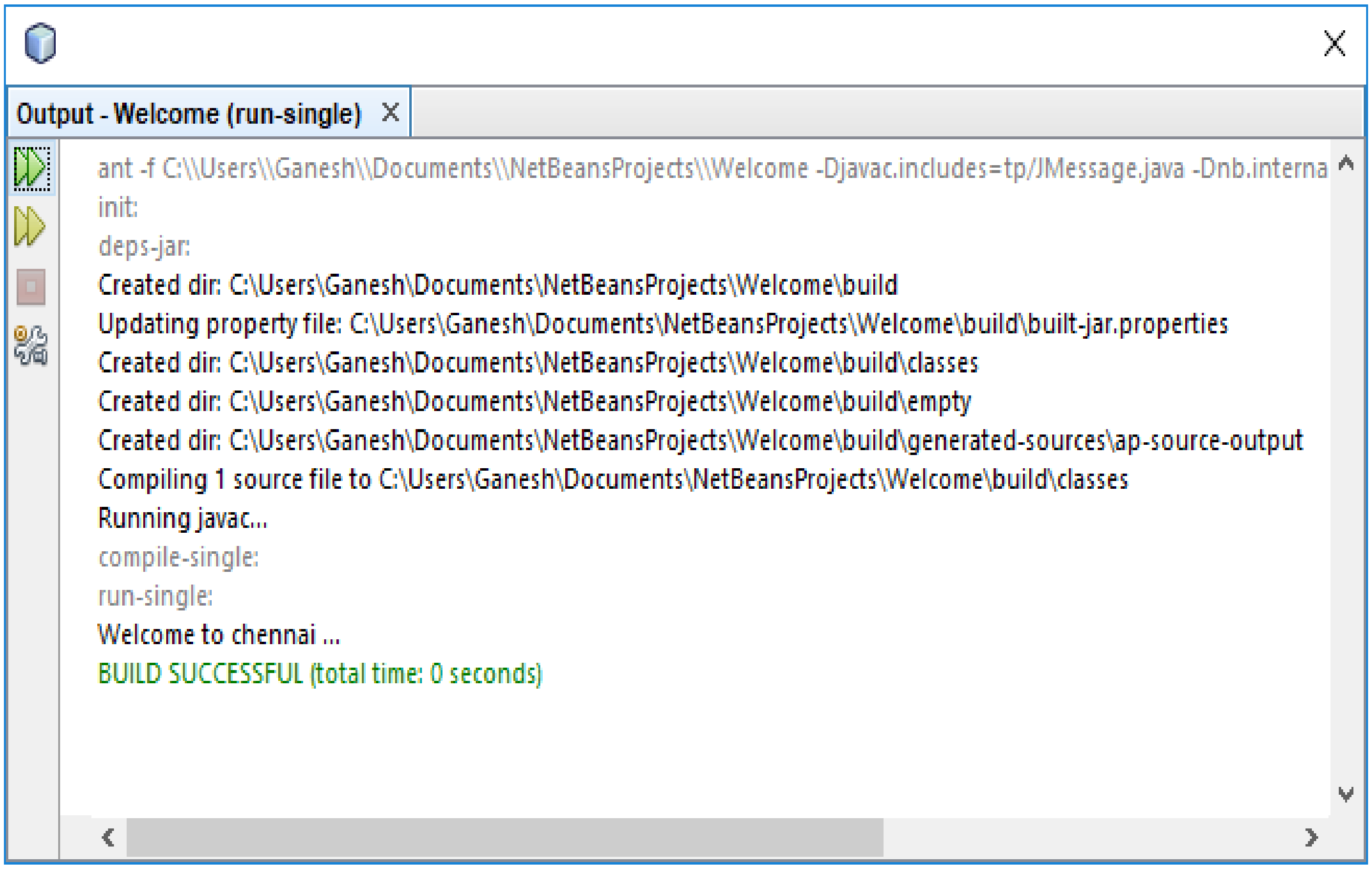
```
/*
 * To change this license
 * To change this templa
 * and open the templat
 */
package tp;
```

3. EXECUTION



```
/*
 * To change this lice
 * To change this ten
 * and open the temp
 */
package tp;
```

4. OUTPUT



```
ant -f C:\\Users\\Ganesh\\Documents\\NetBeansProjects\\Welcome -Djavac.includes=tp/JMessage.java -Dnb.interna
init:
deps-jar:
Created dir: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\build
Updating property file: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\build\built-jar.properties
Created dir: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\build\classes
Created dir: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\build\empty
Created dir: C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\build\generated-sources\ap-source-output
Compiling 1 source file to C:\Users\Ganesh\Documents\NetBeansProjects\Welcome\build\classes
Running javac...
compile-single:
run-single:
Welcome to chennai ...
BUILD SUCCESSFUL (total time: 0 seconds)
```

JAVA INPUT STATEMENTS

- Like c/c++ language, java supports three input statements
- Types
    1. Assignment Inputs (Static input system)
    2. Command Line Inputs
    3. Dynamic Inputs (Runtime Inputs)

1. Assignment Inputs (Static Inputs)

- Providing the input arguments (values) before the program execution is called as assignment input arguments
- We can directly assign the value to variable using equal operator (=)
- Fixed values

Syntax

<modifier> <variable name>=value;

Example

```
int num=95;
```

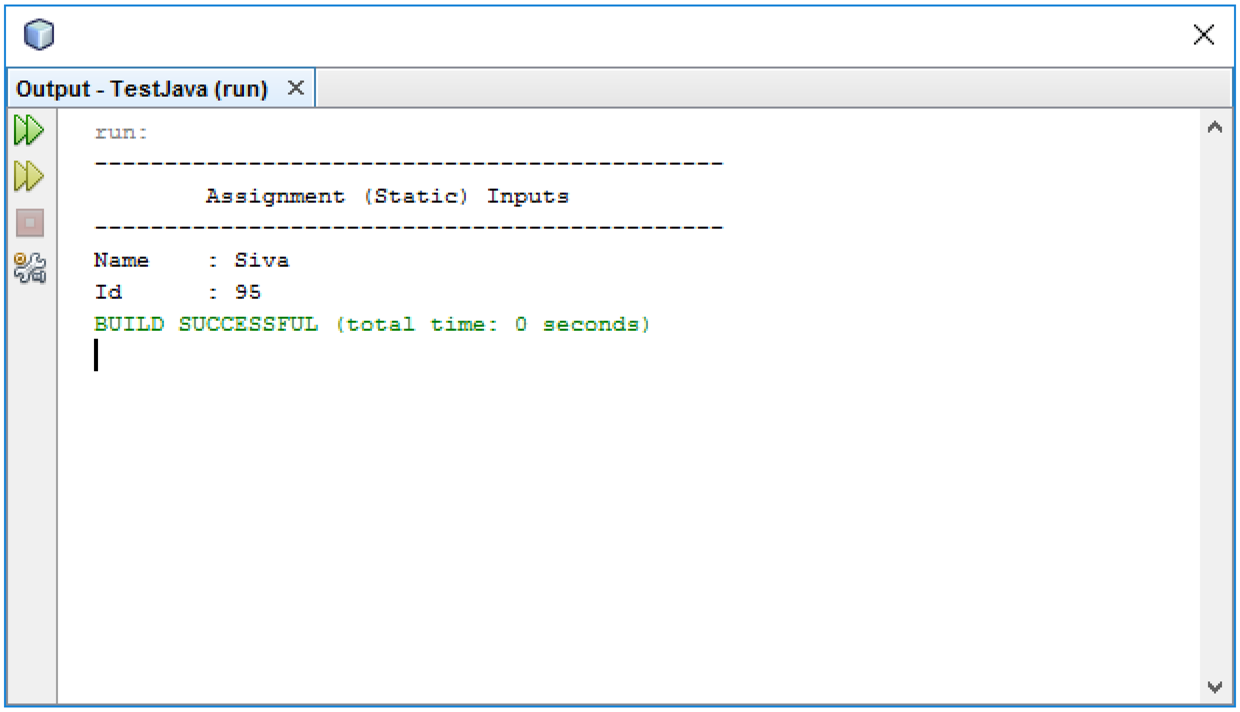
I. EXAMPLE OF ASSIGNMENT INPUTS

1. SOURCE CODE

(JStaticInputs.java)

```
public class JStaticInputs
{
// main() definition
    public static void main(String[] args)
    {
// local variable definition
        int id=95;
        String name="Siva";
// print data (int, string) to output screen (IDE terminal)
        System.out.println("--------------------------------------------");
        System.out.println("\tAssignment (Static) Inputs");
        System.out.println("--------------------------------------------");
        System.out.println("Name\t: "+name);
        System.out.println("Id\t: "+id);
    }
}
```

2. OUTPUT



```
run:
---------------------------------------------
        Assignment (Static) Inputs
---------------------------------------------
Name     : Siva
Id       : 95
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Command Line Inputs

- Providing necessary input values (arguments) to the main() method at the time of program execution
- We can pass input arguments to main() method either manual (e.g. dos command window) or automatic(e.g. IDE)
- By default, all the command line parameters are string array
- An array index starts from 0 to n-1

Manual

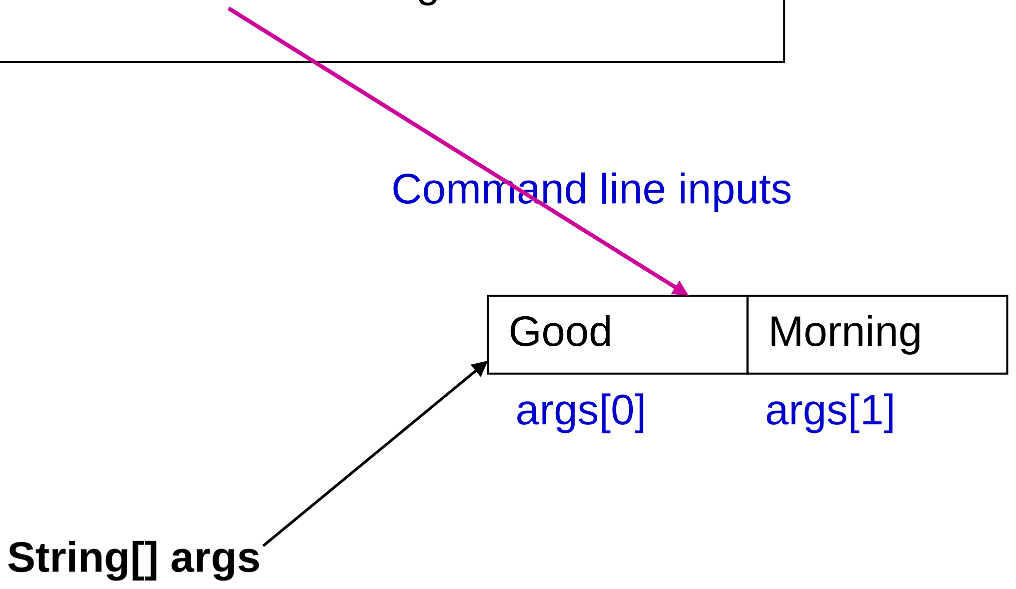
- In the manual approach, the java interpreter takes only one argument.
- The argument is string as a line of text. By default, this inputs are converted as string array (String[] arr)

Syntax

```
javac <filename.java>                          // compilation
java <bytecode-name> <list of cmd inputs>   // interpretation with command line arguments
```

Example

```
javac Test.java
java Test Good Morning
```



Automatic

- In the automatic approach, the user has to specify the command line inputs to the "program arguments" section of current project folder in netbeans IDE.

II. EXAMPLE OF COMMAND LINE INPUTS

(using Notepad-Manual Approach)

1. SOURCE CODE

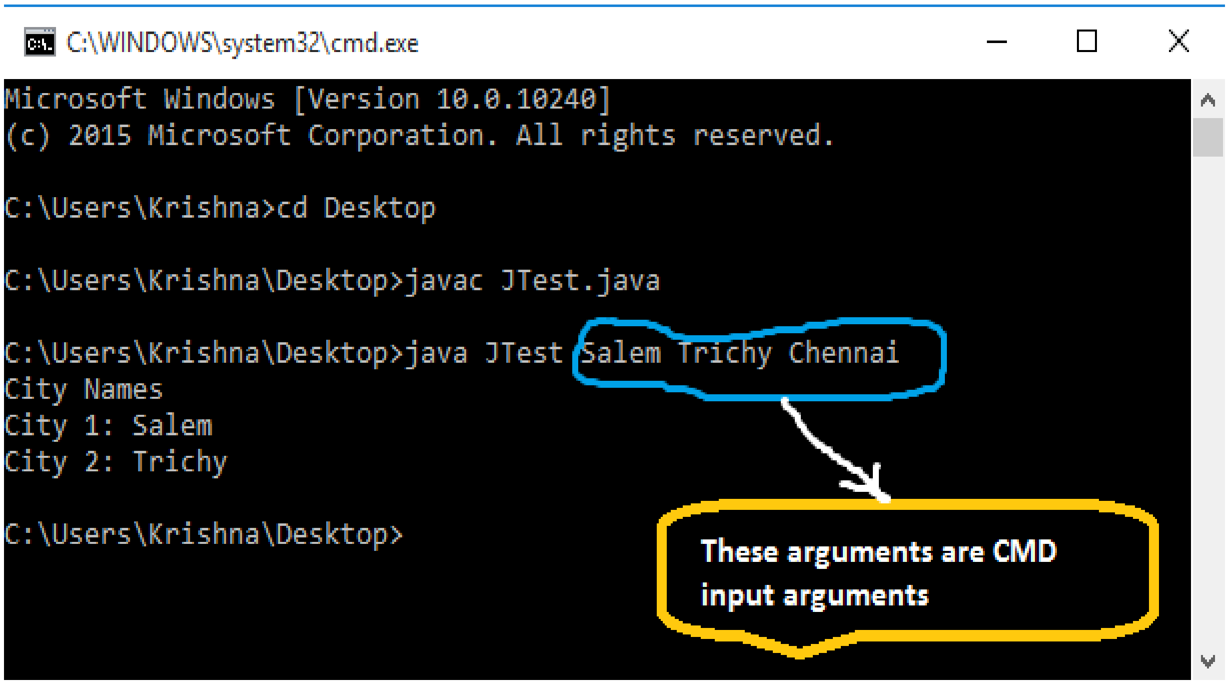
(JTest.java)

```
public class JTest
{
// main() method definition
        public static void main(String[] arr)throws Exception
        {
// checking CMD inputs using the built-in length property of array
                if(arr.length!=0)
                {
                        System.out.println("City Names");
                        System.out.println("City 1: "+arr[0]);
                        System.out.println("City 2: "+arr[1]);
                }
                else
                {
                        System.out.println("No Arguments\nPlease provide the input
Arguments");
                }
        }
}
```

- length is a built-in property of array in java.
- It returns the total number of elements in array as integer.

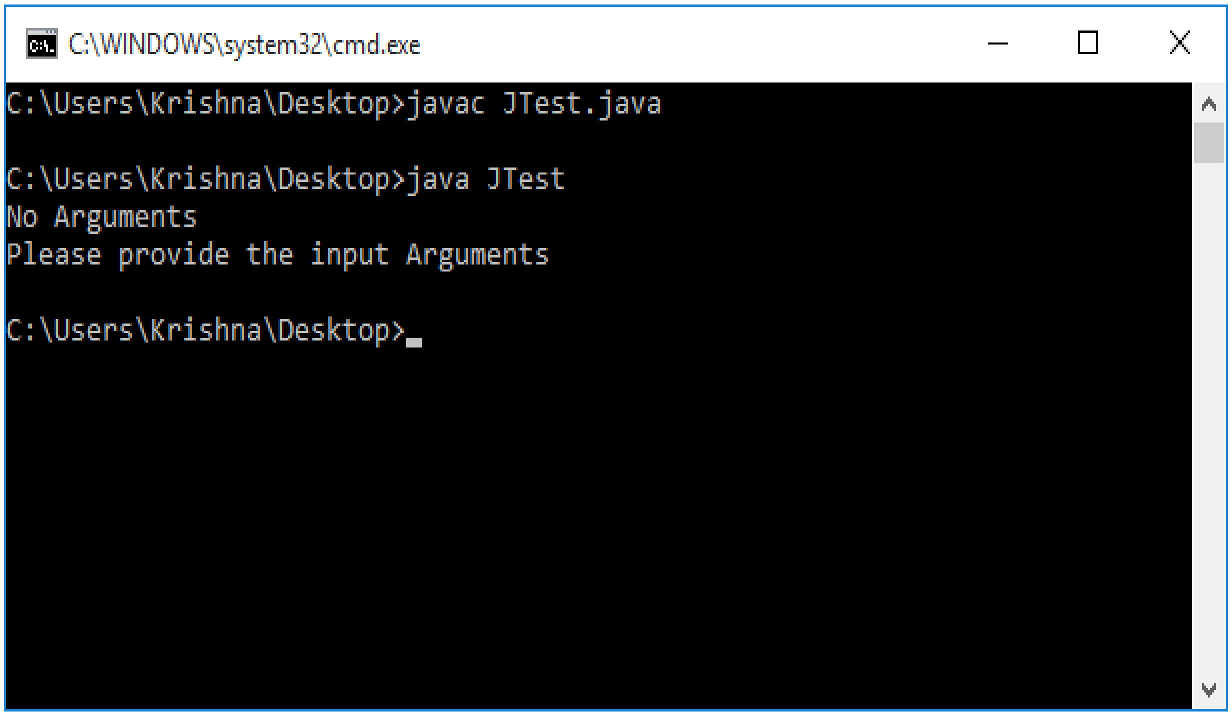
2. OUTPUT

2.1 SUCCESS CASE (IF ARGUMENTS ARE GIVEN)


C:\WINDOWS\system32\cmd.exe
These arguments are CMD input arguments

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Krishna>cd Desktop

C:\Users\Krishna\Desktop>javac JTest.java

C:\Users\Krishna\Desktop>java JTest Salem Trichy Chennai
City Names
City 1: Salem
City 2: Trichy

C:\Users\Krishna\Desktop>
```

2.2 FAILURE CASE (IF NO ARGUMENTS ARE GIVEN)



```
C:\Users\Krishna\Desktop>javac JTest.java

C:\Users\Krishna\Desktop>java JTest
No Arguments
Please provide the input Arguments

C:\Users\Krishna\Desktop>_
```

III. EXAMPLE OF COMMANDLINE INPUTS

(using IDE-Automated Approach)

IDE : Netbeans IDE 8.2

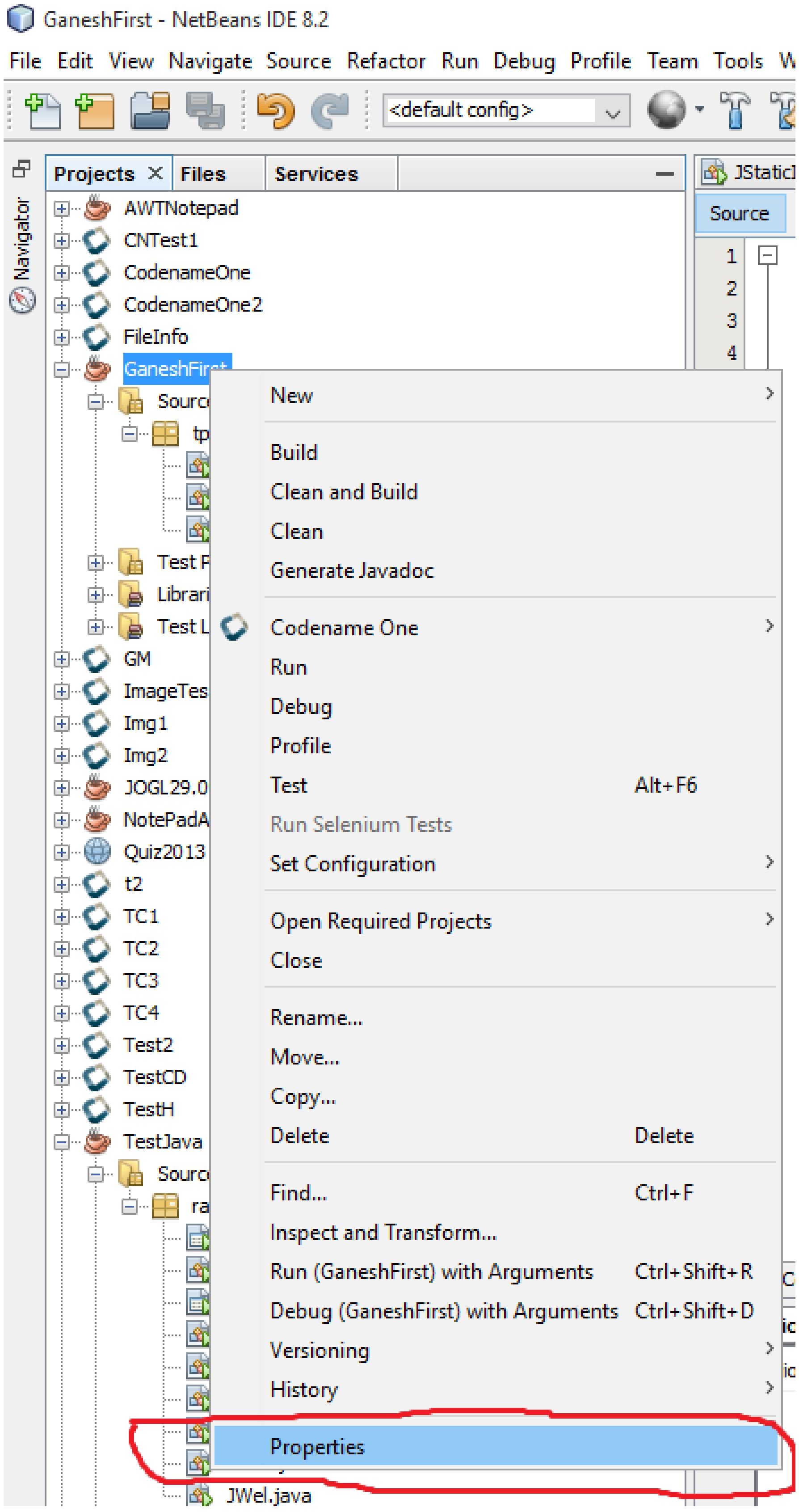
1. SOURCE CODE

(JCommandLine.java)

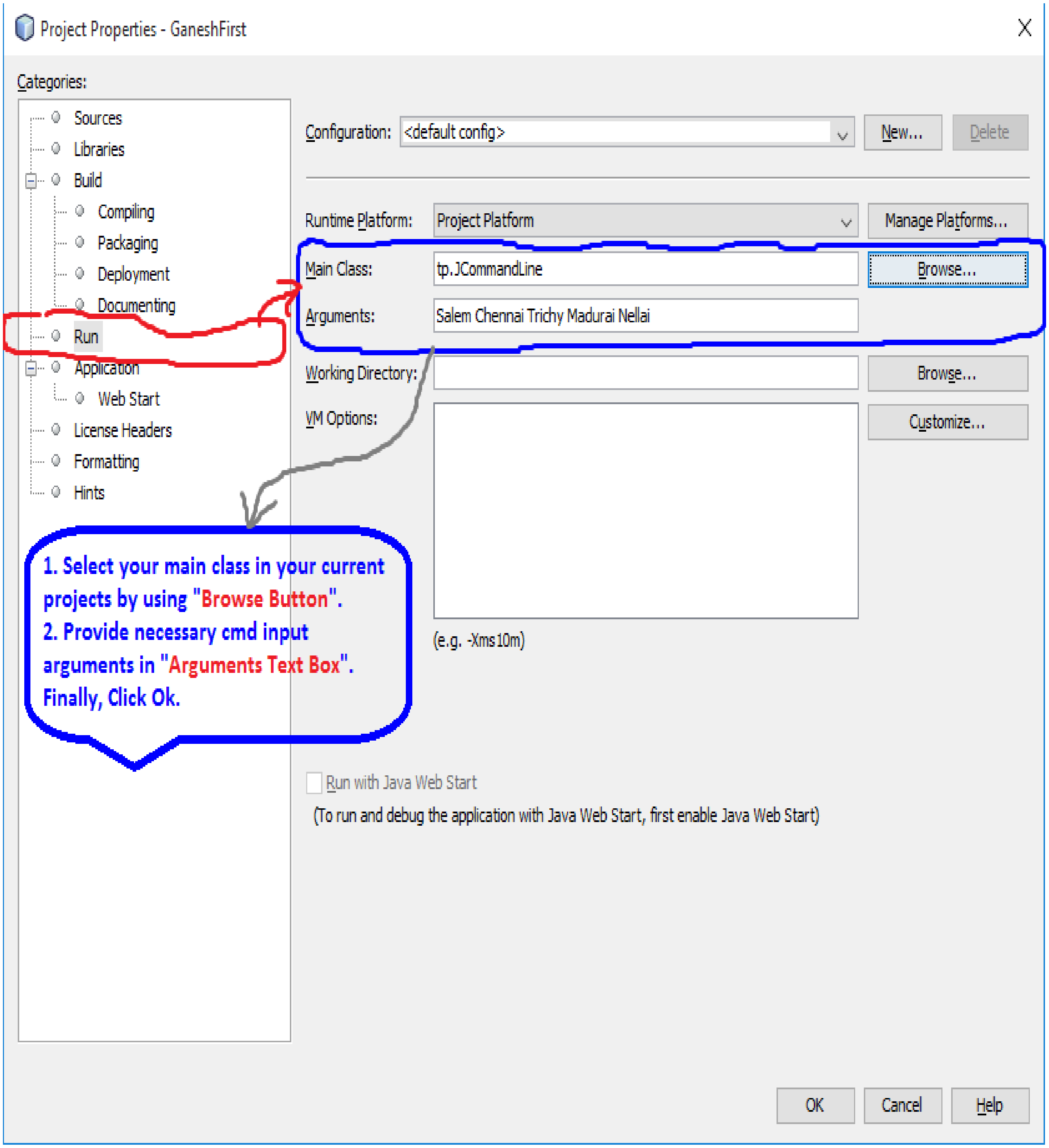
```
public class JCommandLine
{
// main() method definition
    public static void main(String[] args)
    {
        System.out.println("----------------------------------------");
        System.out.println("\tCommand Line Arguments");
        System.out.println("----------------------------------------");
// checking CMD inputs using the built-in length property of array
        if(args.length!=0)
        {
                System.out.println("City Names");
                for(int i=0;i<args.length;i++)
                {
                    System.out.println(args[i]);
                }
        }
        else
        {
                System.out.println("No Arguments\nPlease provide the input
Arguments");
        }
    }
}
```

2. PROJECT CONFIGURATION FOR CMD INPUTS

STEP 1: Select "Properties" by right-clicking on current project folder



STEP 2: Select “Run” from Left Menu and provide input arguments (separated by space) to Arguments Text Box on Right side


Project Properties - GaneshFirst

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Deployment
  - Documenting
- Run
- Application
  - Web Start
- License Headers
- Formatting
- Hints

Configuration: <default config>

Runtime Platform: Project Platform

Main Class: tp.JCommandLine

Arguments: Salem Chennai Trichy Madurai Nellai

Working Directory:

VM Options:

(e.g. -Xms10m)

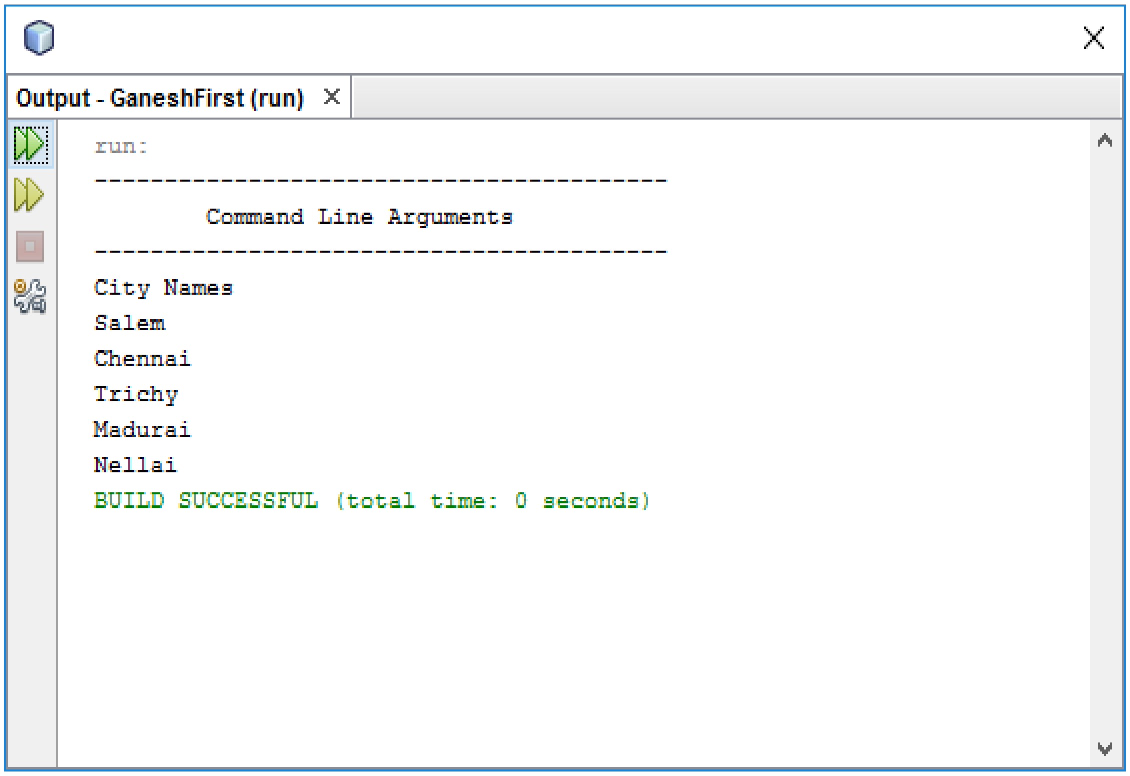
Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)

1. Select your main class in your current projects by using "Browse Button".
2. Provide necessary cmd input arguments in "Arguments Text Box". Finally, Click Ok.

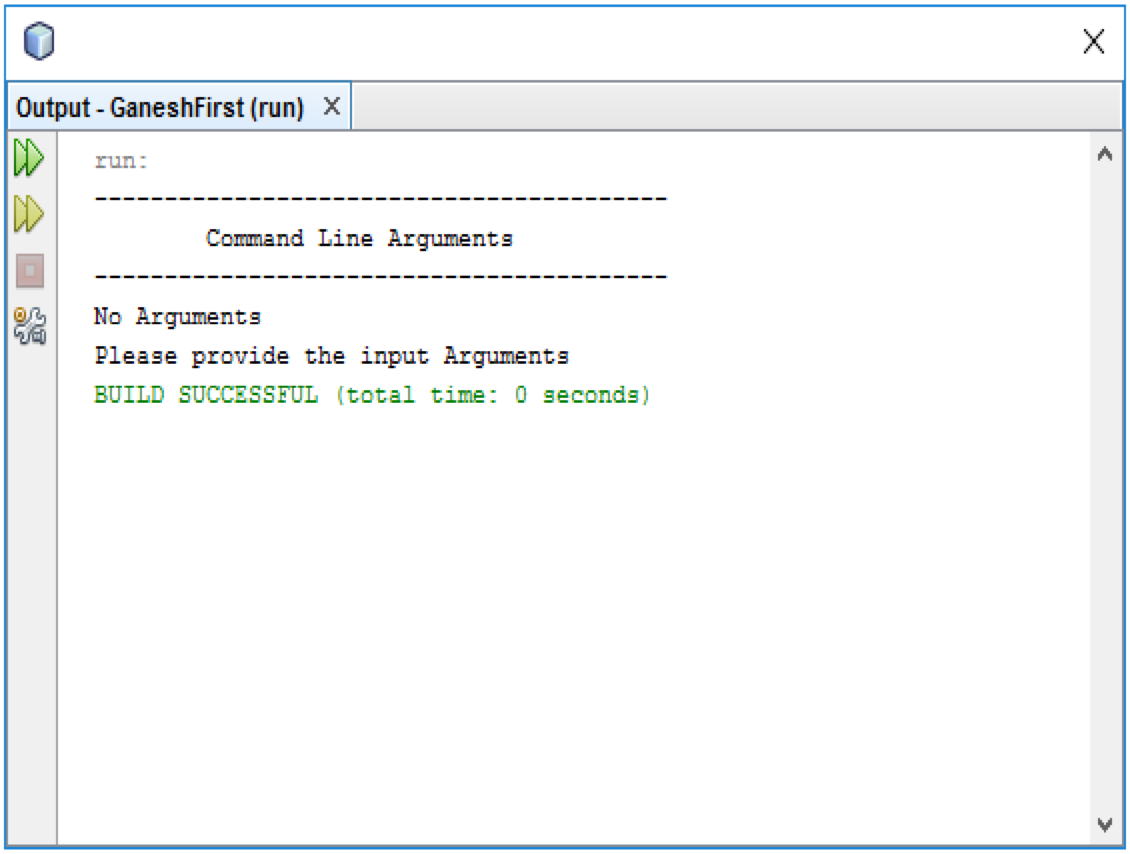
3. OUTPUT

3.1 SUCCESS CASE (IF ARGUMENTS ARE GIVEN)



```
run:
-----------------------------------------
        Command Line Arguments
-----------------------------------------
City Names
Salem
Chennai
Trichy
Madurai
Nellai
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.2 FAILURE CASE (IF NO ARGUMENTS ARE GIVEN)



```
run:
-----------------------------------------
        Command Line Arguments
-----------------------------------------
No Arguments
Please provide the input Arguments
BUILD SUCCESSFUL (total time: 0 seconds)
```

III. DYNAMIC INPUTS / INTERACTIVE INPUTS / RUNTIME INPUTS

- It is possible to give values to variables interactively through the keyboard at the time of program execution (Providing values at the time of runtime)
- Dynamic inputs are done by using stream concepts in java

Streams

- In java, the input and output (I/O) is achieved with help of stream concept
- Stream is the sequence of bytes (objects)
- Streams are generally classified as two types like input stream, output stream

Input Streams

- If bytes flow from device to the main() function, then this process is called input
- Ex. Keyboard

Output Streams

- If bytes flow from main() to device, then this process is called output
- Ex. Display screen (Monitor)

Runtime Inputs

- Java provides several ways to perform dynamic inputs (runtime inputs) namely DataInputStream, BufferedReader, Scanner classes, etc, …

DATAINPUTSTREAM CLASS

- It is a predefined class and available in java.io.* package
- It is used to read java primitive data types (Ex. int, float, double, char, string, etc, …) from the input stream in a machine
- First we need to wrap an input stream (System.in) in a DataInputStream class and then read java primitive types from the DataInputStream. Because it reads data (numbers) instead of bytes

BUILT-IN INSTANCE METHODS OF DATAINPUTSTREAM CLASS

S.N	METHODS	DESCRIPTION
1.	read()	• It is a built-in instance method of DataInputStream class • It is used to read a single byte of data from keyboard • Return type : int
2.	readLine()	• Used to read line of text (string) from keyboard • It is similar to gets() in c language • It is similar to scanf() in c language and cin in c++ language • Return type: **String**
3.	readInt()	• Used to read an integer value • Return type: **int**
4.	readFloat()	• Used to read a float value • Return type: **float**
5.	readDouble()	• Used to read a double value • Return type: **double**

6.	readByte()	• Used to read a byte value • Return type: **byte**
7.	readBoolean()	• Used to read a boolean value • Return type: **boolean**
8.	readChar()	• Used to read a character value • Return type: **boolean**
9.	read(byte[])	• It is used to read the number of bytes from the input stream and store them into byte array (byte []) • Return type: **int**

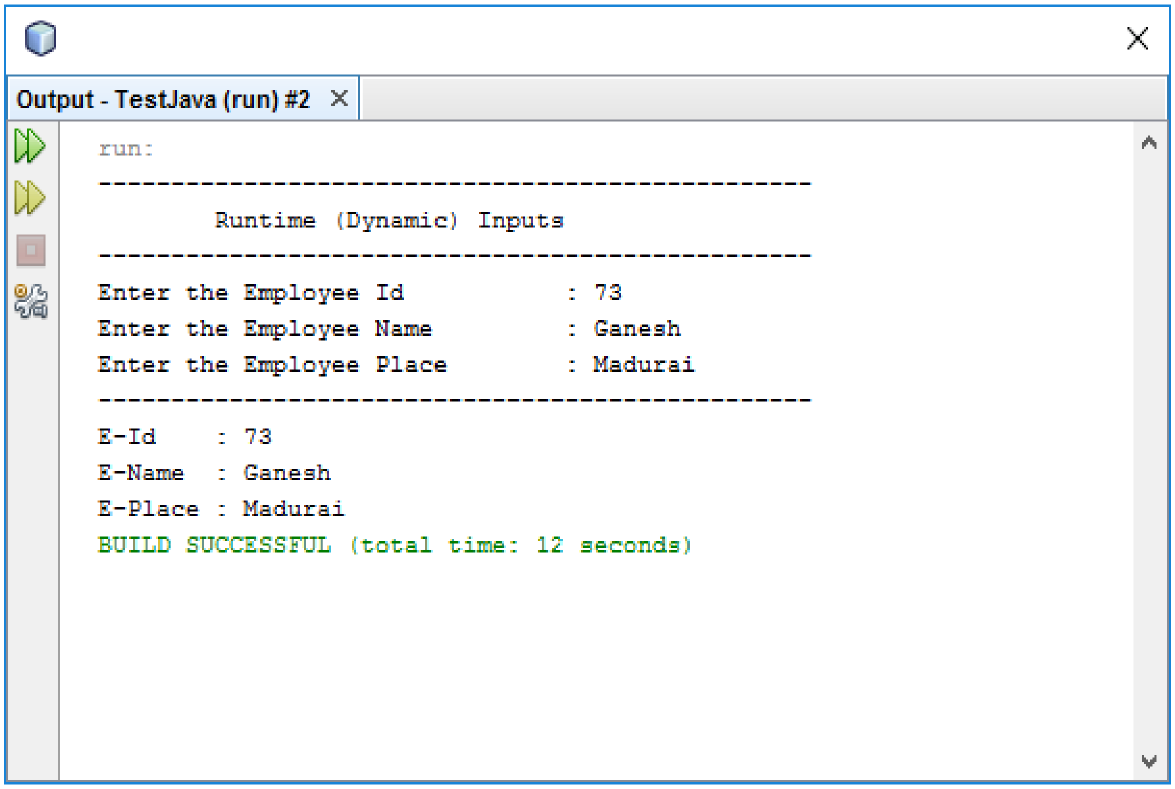
IV. EXAMPLE OF DYNAMIC INPUTS (RUNTIME INPUTS)

1. SOURCE CODE

(JRuntimeInputs.java)

```
import java.io.*;
public class JRuntimeInputs
{
    public static void main(String[] args) throws Exception
    {
// local variables declarations
        String name,place;
        int id;
// DataInputStream definition
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("-------------------------------------------------");
        System.out.println("\tRuntime (Dynamic) Inputs");
        System.out.println("-------------------------------------------------");
    // read employee id
        System.out.print("Enter the Employee Id \t\t: ");
        String sid=ds.readLine();
    // convert string format number to integer number using parse method
        id=Integer.parseInt(sid);
    // read employee name
        System.out.print("Enter the Employee Name \t: ");
        name=ds.readLine();
    // read employee place
        System.out.print("Enter the Employee Place \t: ");
        place=ds.readLine();
    // print employee details
        System.out.println("----------------------------------------------");
        System.out.println("E-Id\t: "+id);
        System.out.println("E-Name\t: "+name);
        System.out.println("E-Place\t: "+place);
    }
}
```

2. OUTPUT



Output - TestJava (run) #2

```
run:
------------------------------------------------
        Runtime (Dynamic) Inputs
------------------------------------------------
Enter the Employee Id           : 73
Enter the Employee Name         : Ganesh
Enter the Employee Place        : Madurai
------------------------------------------------
E-Id    : 73
E-Name  : Ganesh
E-Place : Madurai
BUILD SUCCESSFUL (total time: 12 seconds)
```