Unit – I Introduction to Java

- The Java programming language: history, evolution, features
- Introduction to the Java programming environment, JDK, JRE
- Introduction to the IDE
- Data types and wrapper classes, operators
- Control structures
- String handling
- Basic Input-output

History of Java

- **Java** is a high-level object-oriented programming language developed by **James Gosling** and his colleagues at **Sun Microsystems in 1991**.

- Initially it was known as **Oak Language**.
- In 1995 the Java 1.0 was official released to the world

- ***J**ames gosling, **A**rthur **V**an hoff, **A**ndy bechtolsheim*

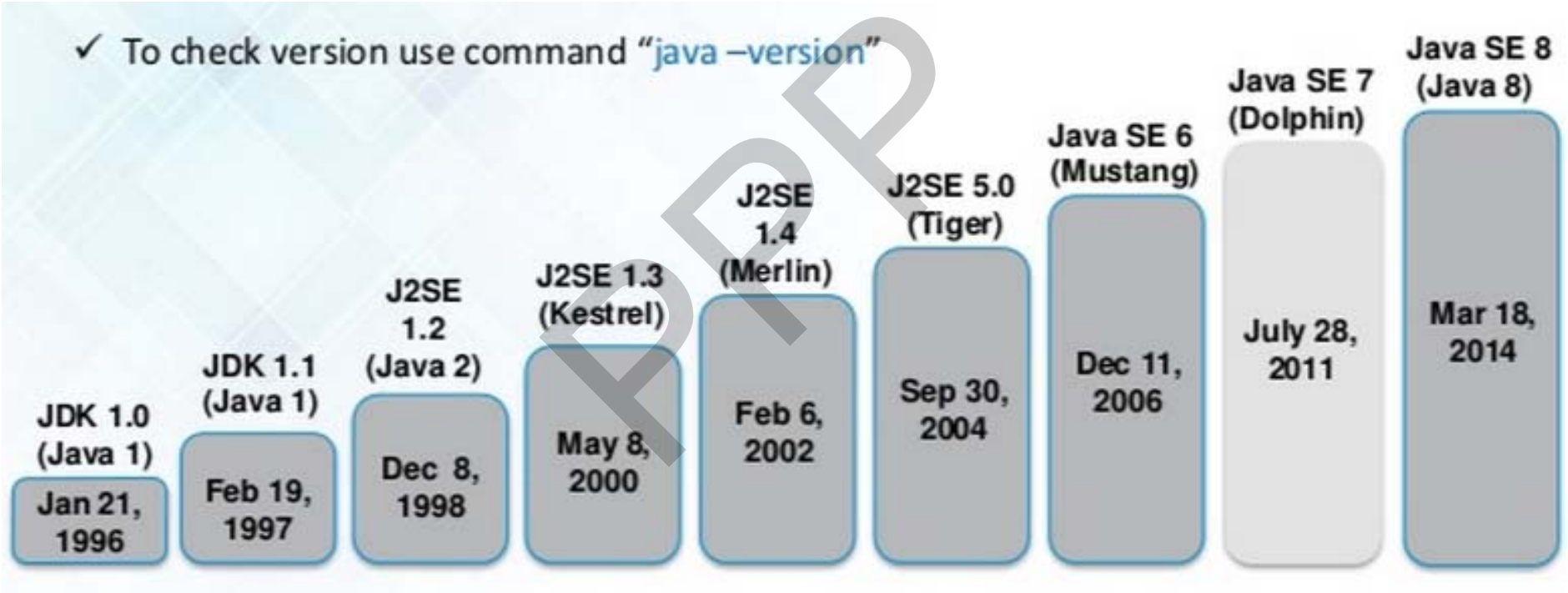
- It was developed keeping in mind the consumer electronics and communication equipments. Thus need of a platform independent language.

- The company promoted this software product with a slogan named **"Write Once Run Anywhere"** that means it can develop and run on any device equipped with **Java Virtual Machine (JVM)**

- This language is supported in all kinds of OS including Linux, Windows, Solaris etc.

- Internet Demands **Portability and Platform Independent**.

Java Evoution



Java Evoution

Java SE 8 **(LTS)**	Mar-14
Java SE 9	Sep-17
Java SE 10	Mar-18
Java SE 11 **(LTS)**	Sep-18
Java SE 12	Mar-19
Java SE 13	Sep-19
Java SE 14	Mar-20
Java SE 15	Sep-20
Java SE 16	Mar-21
Java SE 17 **(LTS)**	Sep-21

- A **Java LTS** (long-term support) release is a **version** of **Java** that will remain the industry standard for several years.
- **E.g. Java** 8 was released in 2014, it will continue to receive updates until 2020, and extended support will end by 2025

Features of Java

- **Simple:**
  - Easy to learn & use effectively. Extends C & C++ so easy for programmers
- **Object Oriented:**
  - Everything in java is object oriented. All program code , methods and data resides in a class.
- **Robust :**
  - A program must execute reliably in a various system.
  - Main reason for program failure are
    - **Memory management** problem &
    - Mishandled **exception condition**.
  - Java supports **Garbage Collection** and use inbuilt E**xception Handler** to make it robust.

Features of Java

- **Multithreading :**
  - Creating an interactive network program.
  - A program may do many things simultaneously.
- **Architectural-Neutral:**
  - Java designer main goal is that a code is long life and portable.
  - If processor, OS upgrade java program is run successful.
  - "Write once, run anywhere, any time, forever."
- **Interpreted and High Performance:**
  - Java enable the creation of cross-platform programs by **byte code.**
  - Byte code interpreted by **JVM (Java Virtual Machine)**.
  - Byte code converted into native code by **JIT (Just In time compilation**).
  - Java perform very well in low power CPU.
- **Distributed:**
  - Java handles TCP/IP protocols.
  - Allow two objects on different computers execute remotely with **RMI** (**Remote Method Invocation**).
- **Dynamic:**
  - Java allocates memory at runtime for classes, methods and objects. This feature allows programms to allocate memory when required.

Introduction To The Java Programming Environment

Java is a concurrent, class-based, object-oriented programming and runtime environment, consisting of:

- A programming language - Java
- Java APIs
- JVM

Introduction To The Java Programming Environment

A programming language Java

•Java is a powerful and has been exceptionally successful in business and enterprise computing.

•Java is an object-oriented programming language that runs on almost all electronic devices.

•The current steward of Java is Oracle Corporation (who acquired Sun Microsystems, the originator of Java).

•Other corporations, such as Red Hat, IBM, Hewlett-Packard, SAP, Apple, and Fujitsu are also heavily involved in producing implementations of standardized Java technologies.

•There is also an open source version of Java, called **OpenJDK**, which many of these companies collaborate on.

•Java actually comprises several different, but related environments and specifications—Java Mobile Edition (**Java ME**), Java Standard Edition (**Java SE**), and Java Enterprise Edition (**Java EE**).

Introduction To The Java Programming Environment

Java APIs

•Java APIs are integrated pieces of software that come with JDKs. APIs in Java provides the **interface between two different applications and establish communication.**

• APIs in Java include classes, interfaces, and user Interfaces.

•They enable developers to integrate various applications and websites and offer real-time information.

Introduction To The Java Programming Environment

Compiler & Interpreter

- A **Compiler** searches all the errors of a program and lists them.
- If the program is error free then it converts the code of program into machine code and executed.

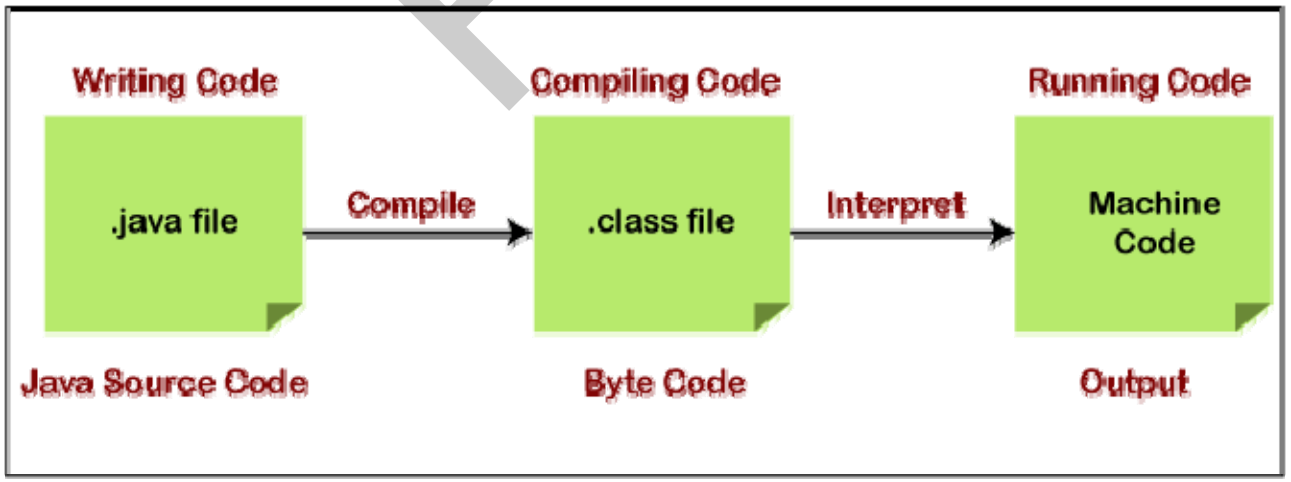
- An **Interpreter** checks the errors of a program statement by statement.

- Java uses a clever mixture of Compiler & Interpreter.

Introduction To The Java Programming Environment

Execution of Java Program is divide into two process

- Source Code --→ Java Compiler --→ Byte Code
- Byte Code --→ JVM --→ Machine readable file



Introduction To The Java Programming Environment

Java's Magic : The Bytecode & JVM

•The output after compilation of java source code is not an executable code but it is a **Bytecode**.

•Bytecode is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (**JVM**)

•JVM is an **interpreter** for bytecode.

•Bytecode provides both: Security & Portability

•**Portability:** Translating a Java program into bytecode helps makes it much easier to run a program in a wide variety of environments

•**Security:** Execution of every Java program is under the control of the JVM, the JVM can contain the program and prevent it from generating side effects outside of the system

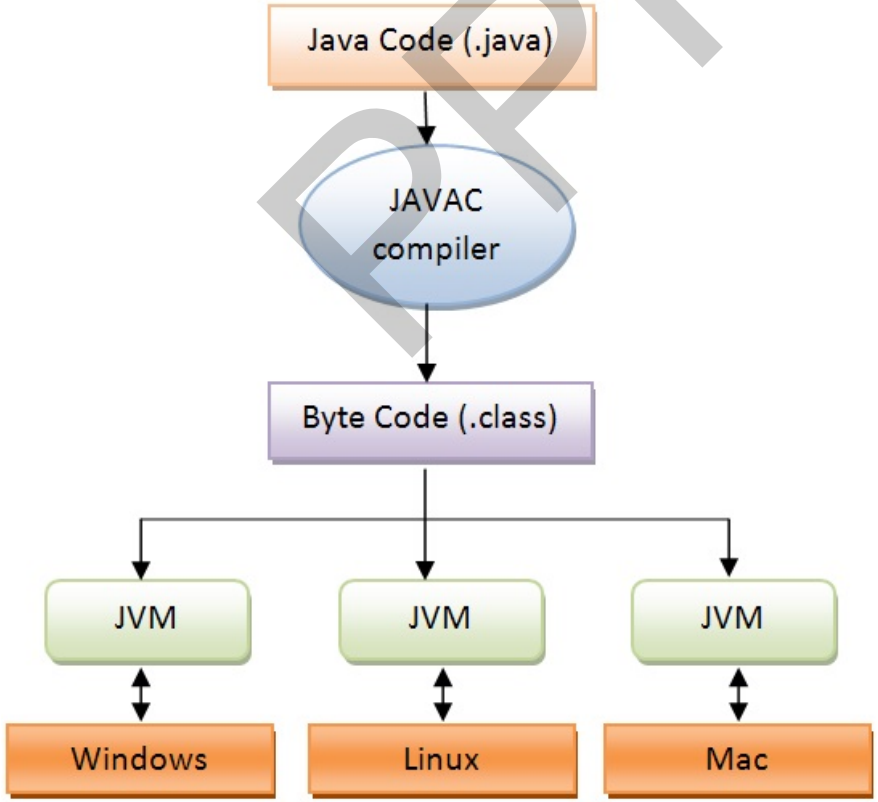
Introduction To The Java Programming Environment

JIT (Just-In-Time)

- JVM also has JIT compiler.
- JIT is a part of JVM.
- JIT execute selected portion of Bytecode.
- It means that the JIT compiles code as it is needed, during execution.

Introduction To The Java Programming Environment

- Any computer that has JVM installed to it can execute the bytecode.
- Any platform with JVM can install the bytecode. This makes Java platform-independent programming language.



Introduction To The Java Programming Environment

JDK (Java Development Kit)

- JDK is a collection of tools that are used for developing & execution of Java Program. It contain any development tools like a **compiler, debugger,** etc

1. **javac -** compiler used to compile Java source code.
   Syntax : **javac filename.java** (Source file name extension is ".java")

2. **java –** interpreter used to execute Java Bytecodes.
   Syntax : **java filename**

3. **appletviewer -** Used to view and test applets.
   Syntax : **appletviewer url**

4. **Javadoc -** This is the Java Documentation tool.
   Generates detailed documentation in HTML form for any .java source code or package.
5. **jdb -** Java debugger which help to find errors in program.

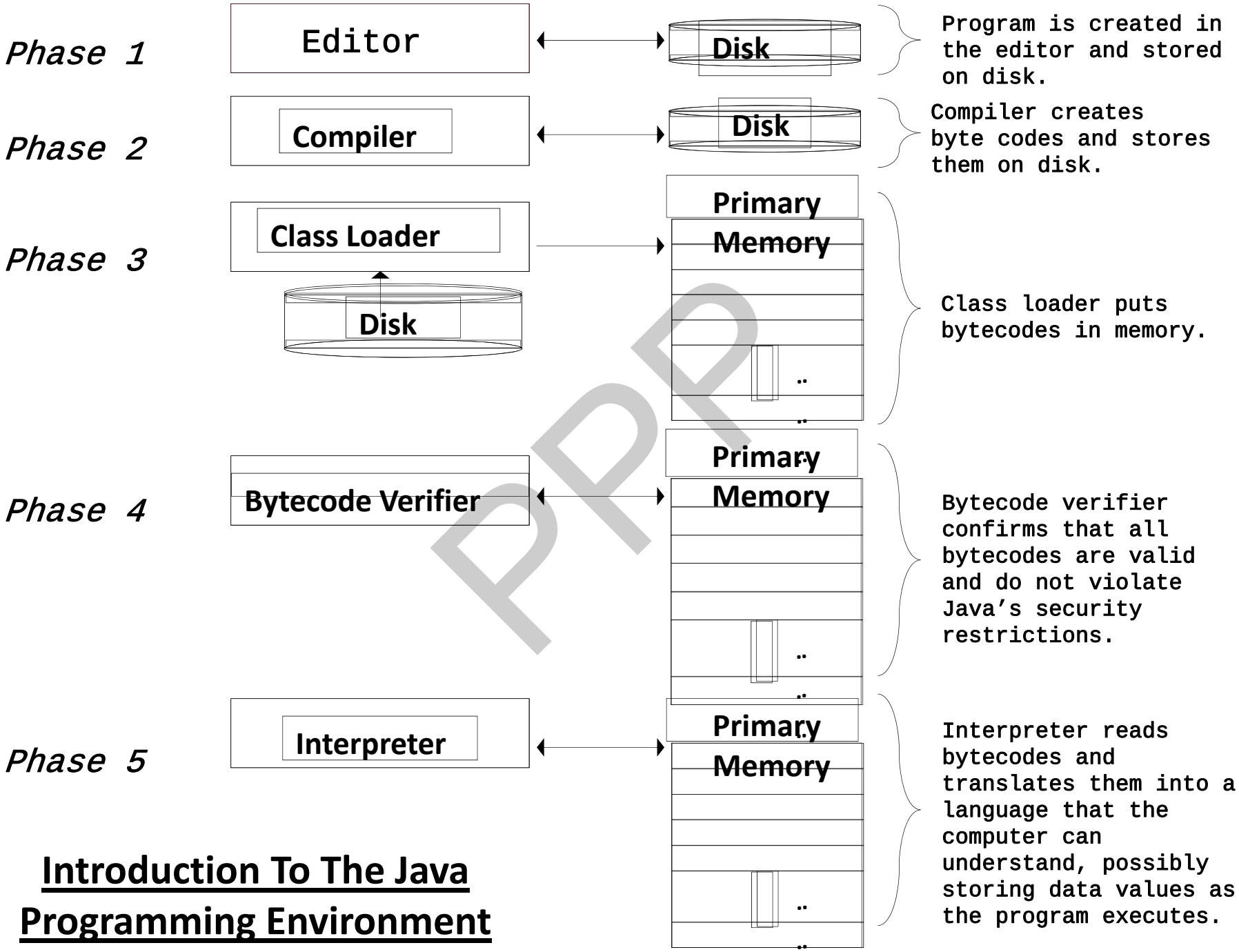
Introduction To The Java Programming Environment

Java Runtime Environment (JRE)

•It is an installation package which provides environment to **only run(not develop)** the java program(or application) onto your machine.

•It consists of **JVM, Java binaries, and other classes** for the smooth execution of the program.

•It runs on top of a computer's operating system software and provides the class libraries and other resources that a specific Java program needs to run.



Introduction To The Java Programming Environment

Introduction to the IDE (Integrated Development Environment)

- It is a programming environment that provides comprehensive facilities to computer programmers for software development.
- It combines all the basic tools that developers need to write or test software.
- This type of environment allows an application developer to write code while compiling, debugging and executing it at the same place.
- It can be a standalone application or a part of one or more compatible applications.
- **Popular IDEs**
  - NetBeans
  - jEdit
  - Eclips
  - JBuilder
  - JCreator

Keywords in Java

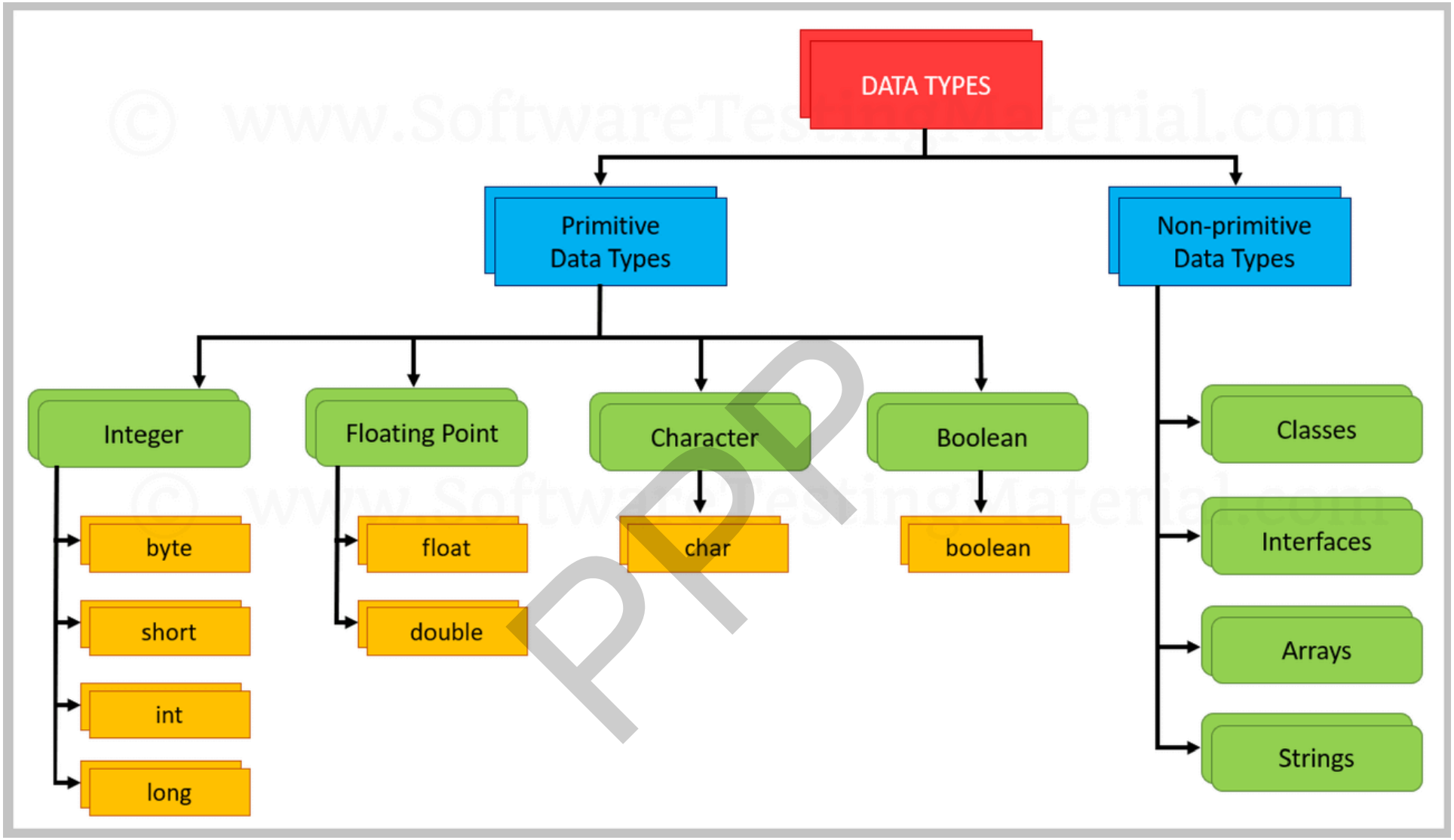
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue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

Data types

- **There are two types of data types in Java:**

  - **Primitive data types:**
    - These are the predefined data types of Java. They specify the size and type of any standard values.
    - The primitive data types are **boolean, char, byte, short, int, long, float and double.**

  - **Non-primitive data types:**
    - These are created by the programmer and is not defined by **Java** (except for String).
    - It can be used to call methods to perform certain operations
    - The non-primitive data types are **Classes, Interfaces, and Arrays.**

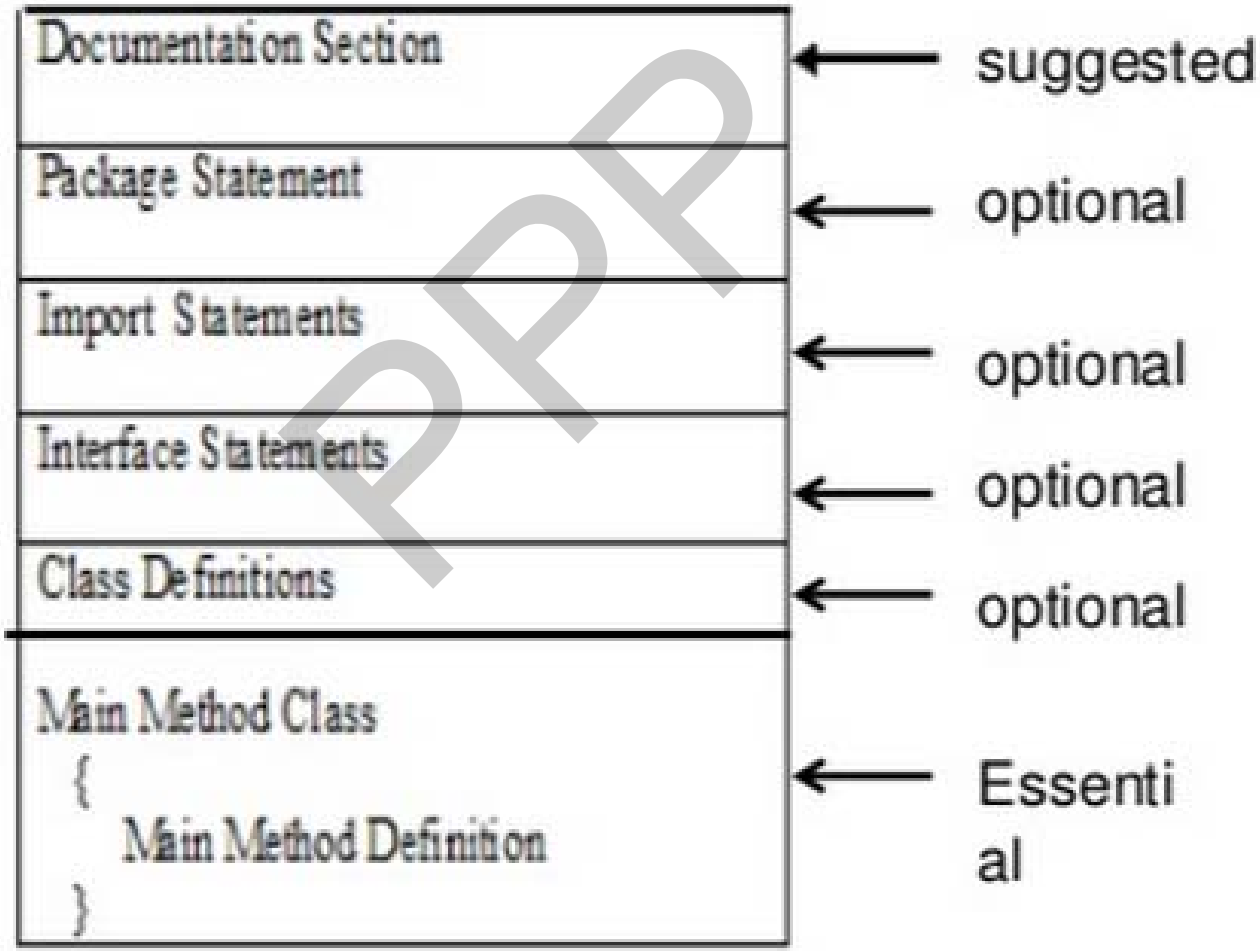


Data types

– **Primitive data types:**

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Basic structure of java program



Document Section

- **Two types of comments**
  - **Single line comment**
    - // insert comments here.
  - **Block comment**
    - /*

      insert comments here.

      */

- **Package Statement**
  - package Package-Name
  - **E.g.** package student
- **Import Statement**
  - Import package-name.class-name
  - **E.g.** import java.io.*
- **Interface Statement**
  - It is similar to class but includes a group of method declaration. Used for multiple inheritance.
- **Class definition**
  - A Java program can contain multiple class definitions
- **Main Method Class**
  - Every java stand-alone application requires a ‘main’ method.
  - The main method creates objects of various classes and establishes a connection between them.

Softwares To Run Java Programs

- The **NetBeans IDE** runs on the Java platform, which means that you can use it with any operating system for which there is a JDK available. These operating systems include Microsoft Windows, Solaris OS, Linux, and Mac OS X.
- **To write your first program, you'll need softwares:**
- **The Java SE Development Kit**
  - Latest version of Java is **Java 17 or JDK 17** released on **2021**
  - For Microsoft Windows, Solaris OS, and Linux:
  - **https://www.oracle.com/java/technologies/downloads/**
- **The NetBeans IDE (Latest version 12.2)**
  - Latest version of NetBeans is **12.6** released on **2021**
  - For all platforms:
  - **https://netbeans.apache.org/download/index.html**

First Java Program In NetBeans

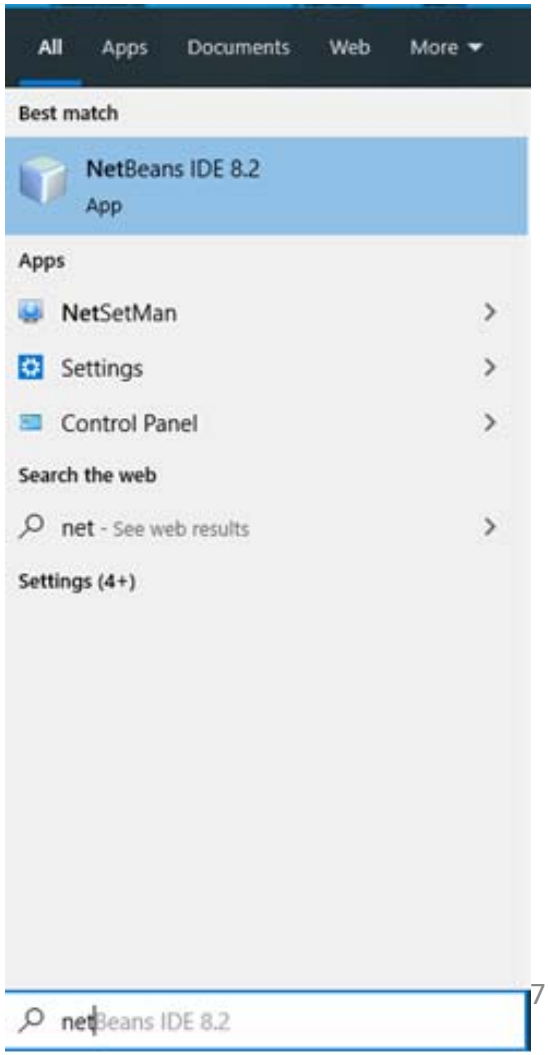
Creating Your First Application

Your first application, FirstApp, will simply display the message "This is my first java program" To create this program, you will:

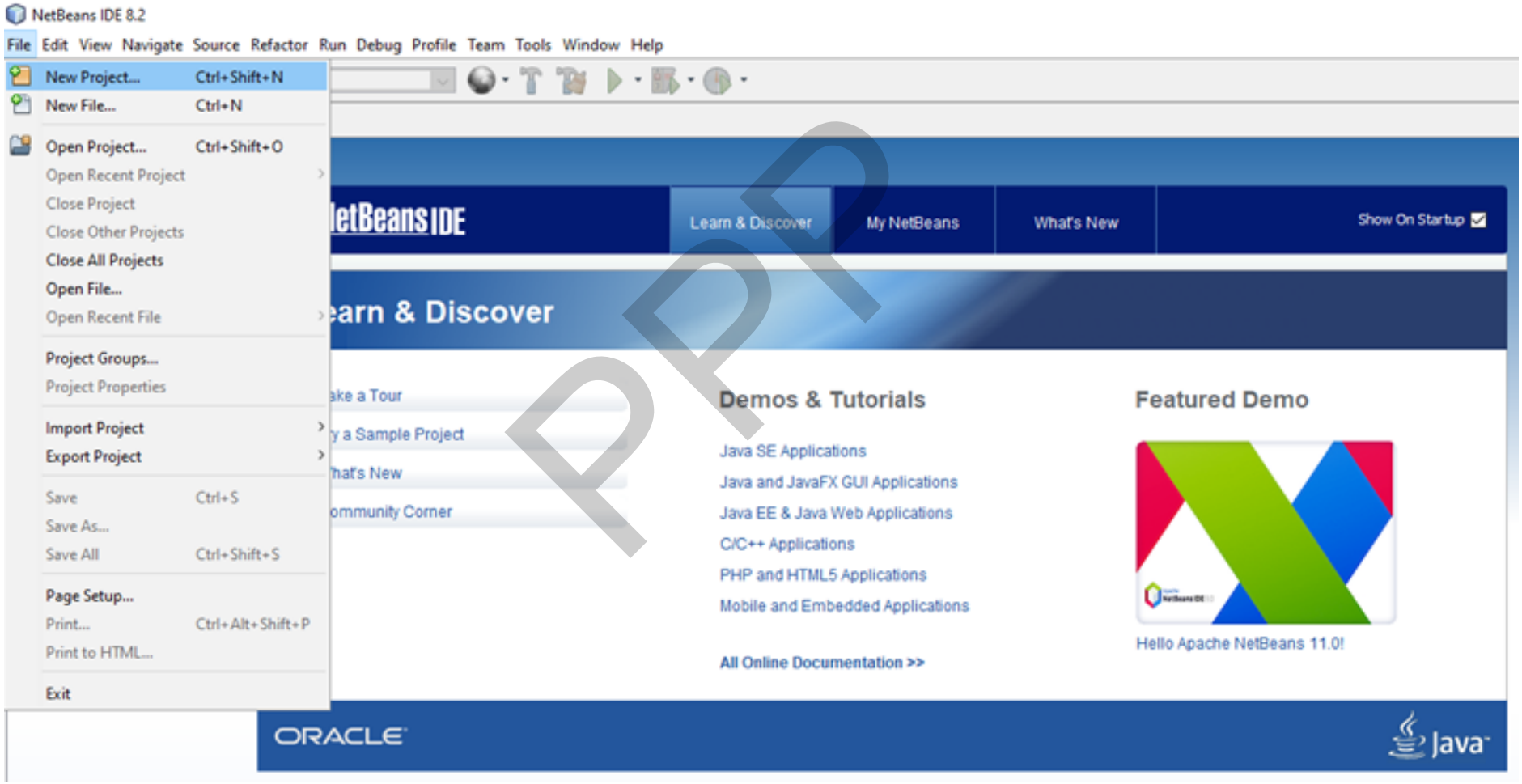
1.Launch the NetBeans IDE. :

On Microsoft Windows systems,

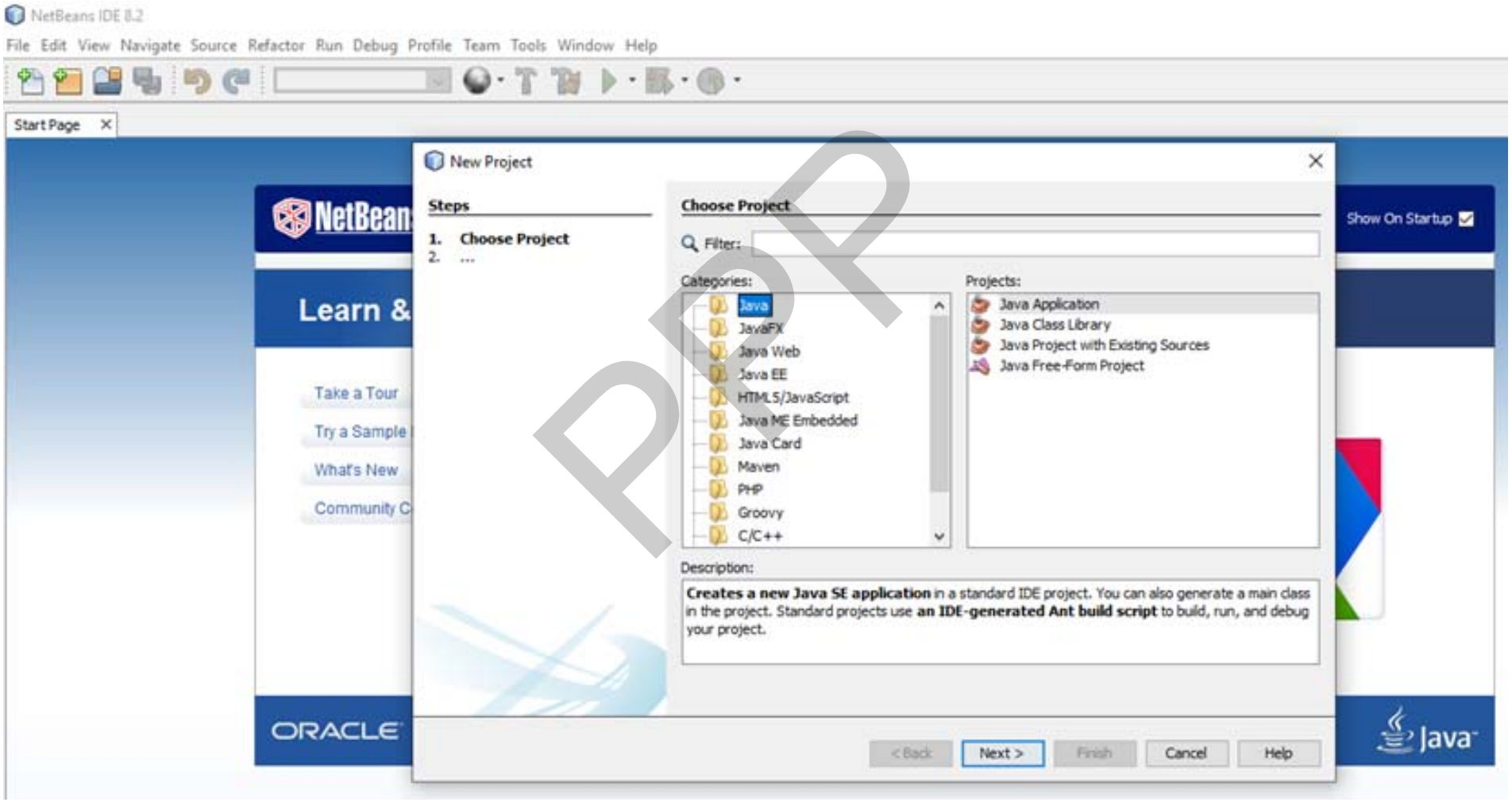
you can use the NetBeans IDE item in the Start menu.



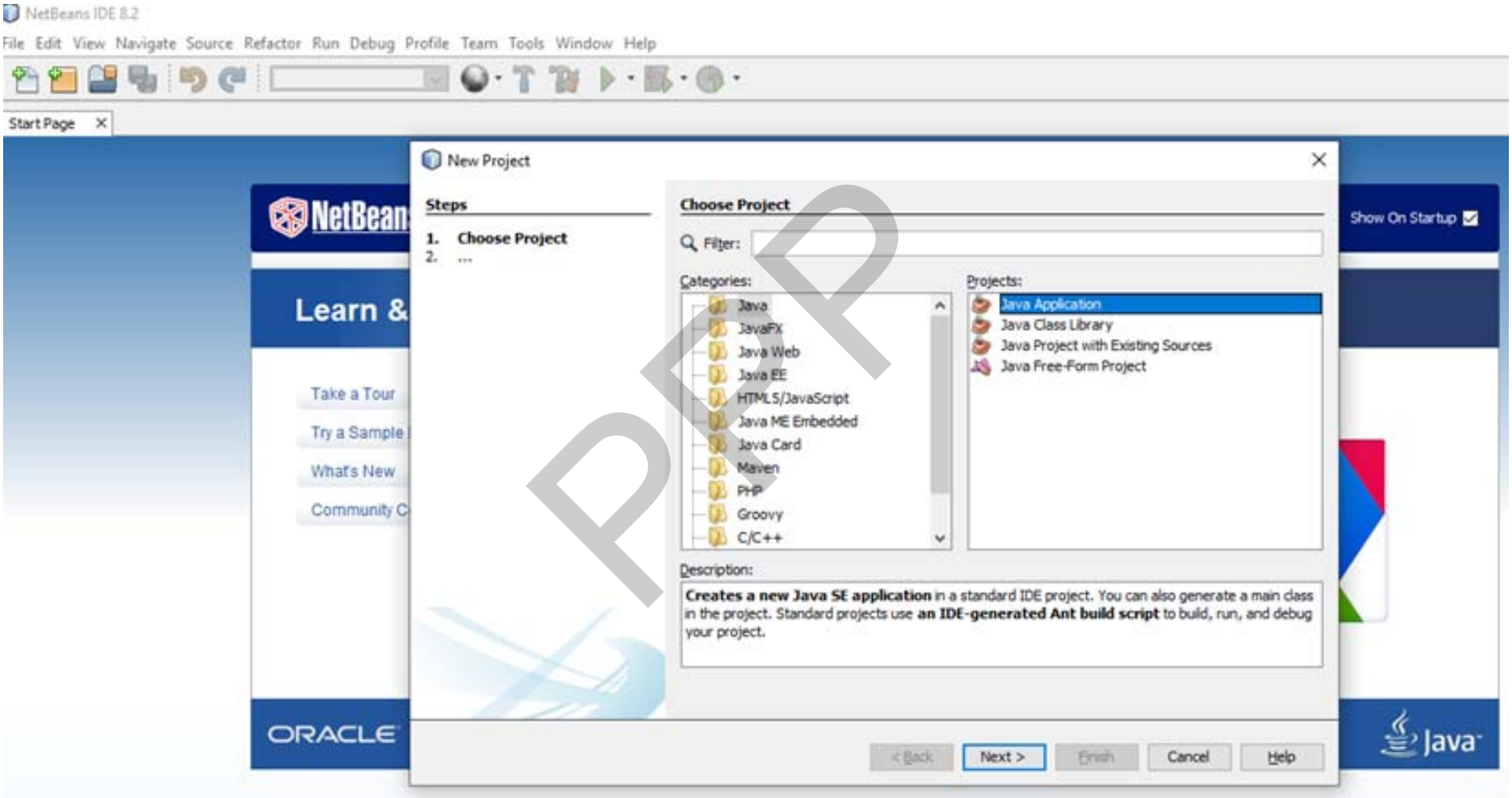
In the NetBeans IDE, choose **File** | **New Project...**.



In the **New Project** wizard, expand the **Java** category as shown in the following figure:



Select **Java Application** as shown in the following figure:



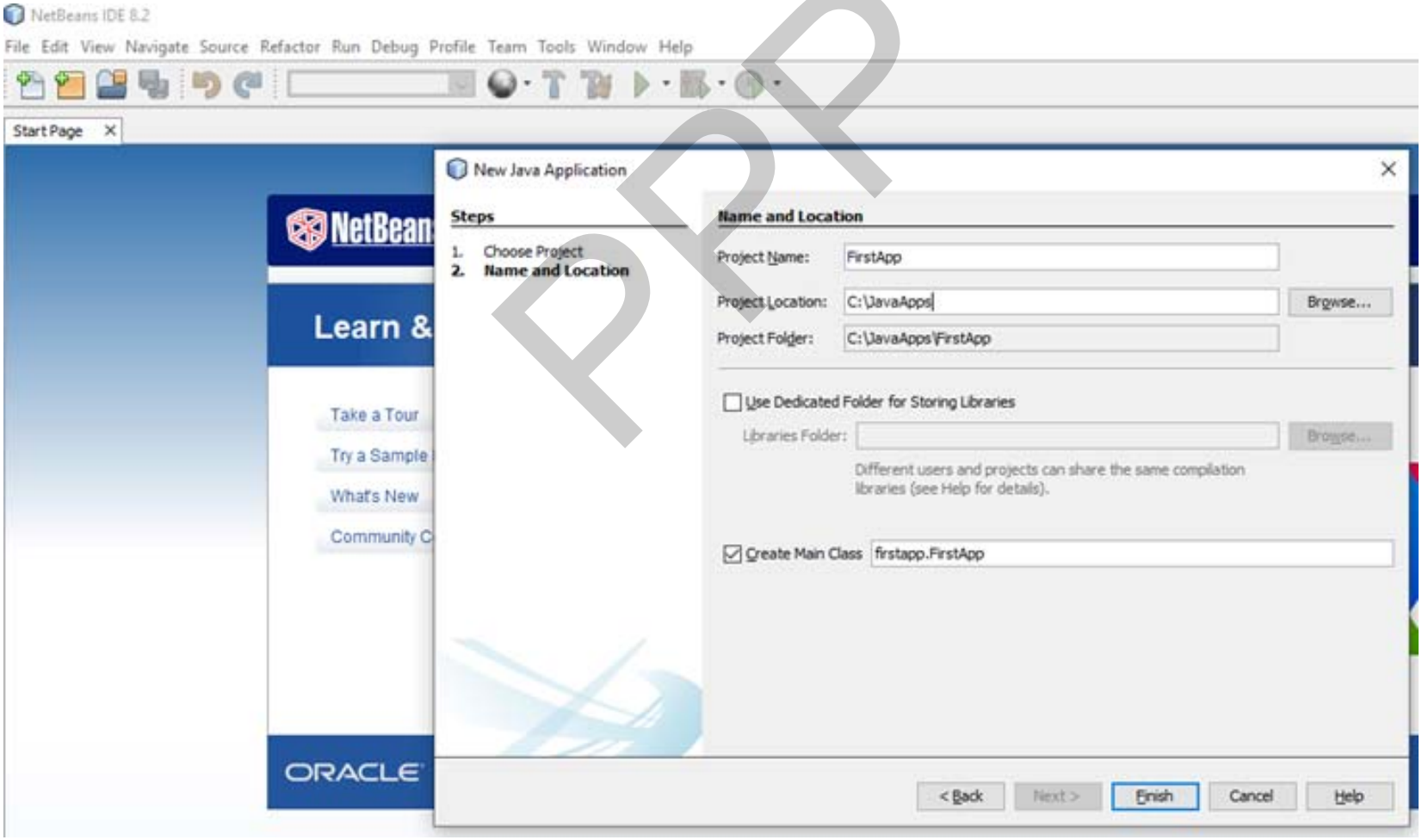
In the **Name and Location page** of the wizard, do the following (as shown in the figure below):

In the **Project Name** field, type **FirstApp**.

In the **Project Location**: C:**JavaApps**

In the **Create Main Class** field, type firstapp.FirstApp.

Click the **Finish** Button.

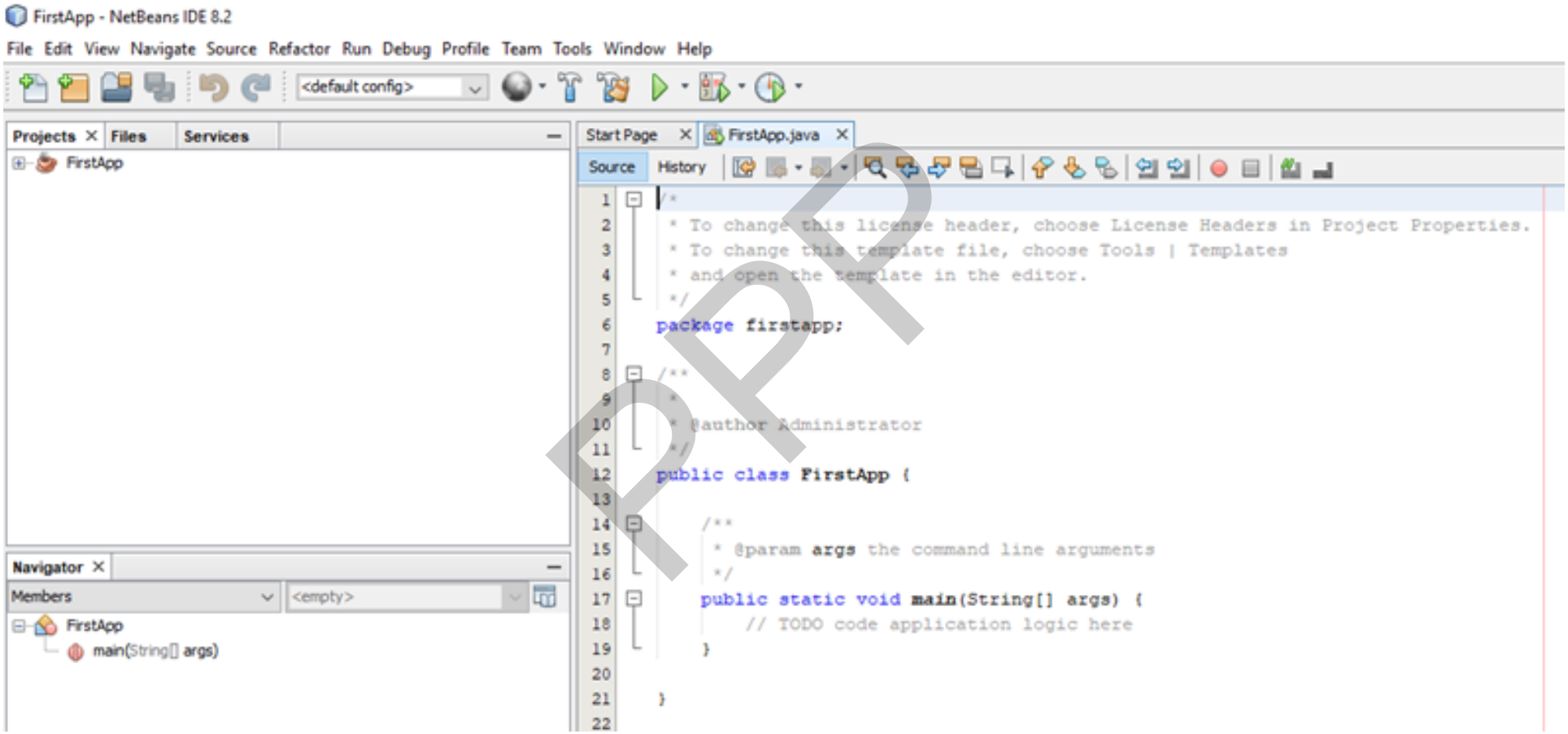


The project is created and opened in the IDE. You should see the following components:

The **Projects** window, which contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.

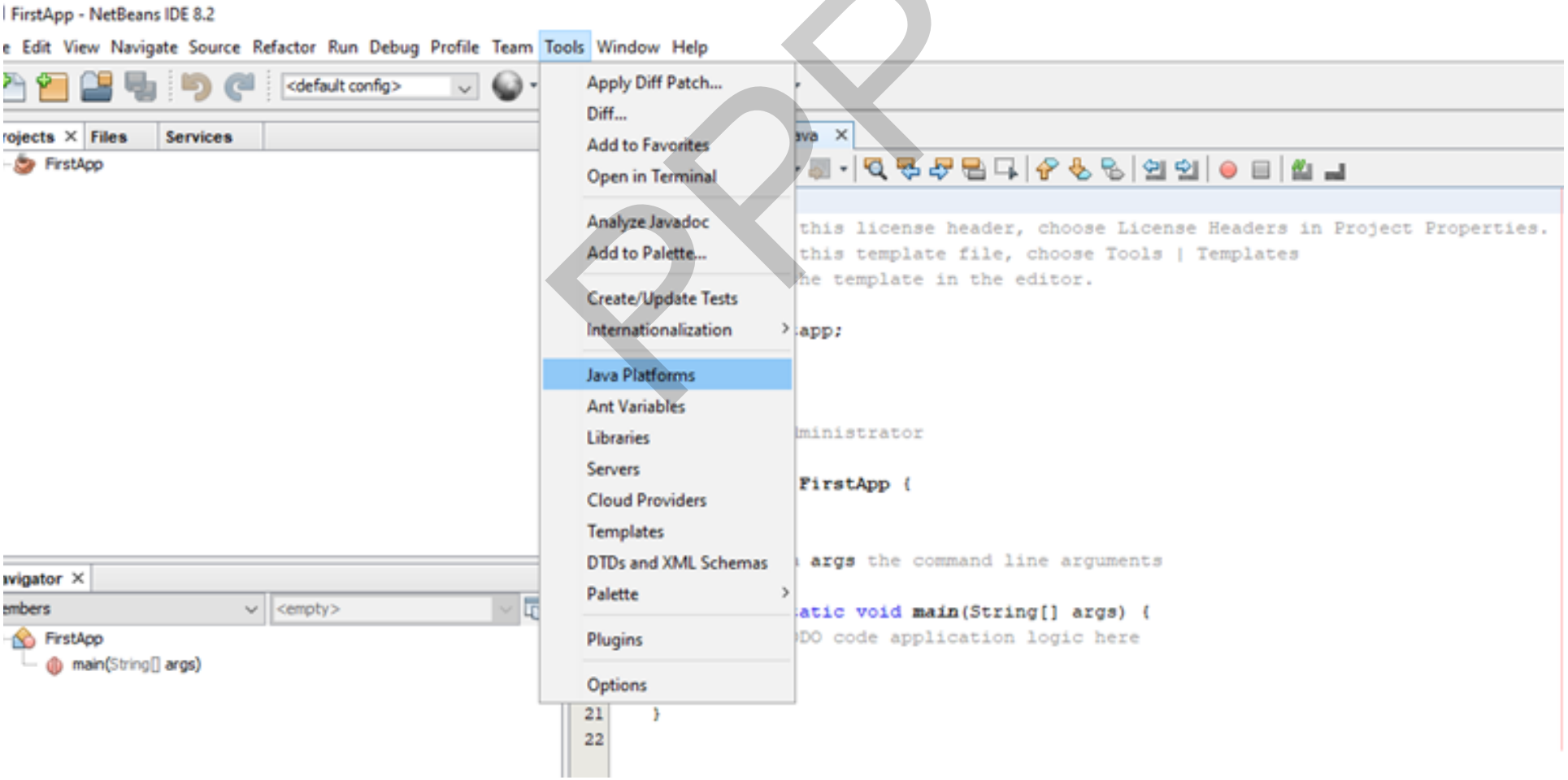
The **Source Editor** window with a file called **FirstApp.java** open.

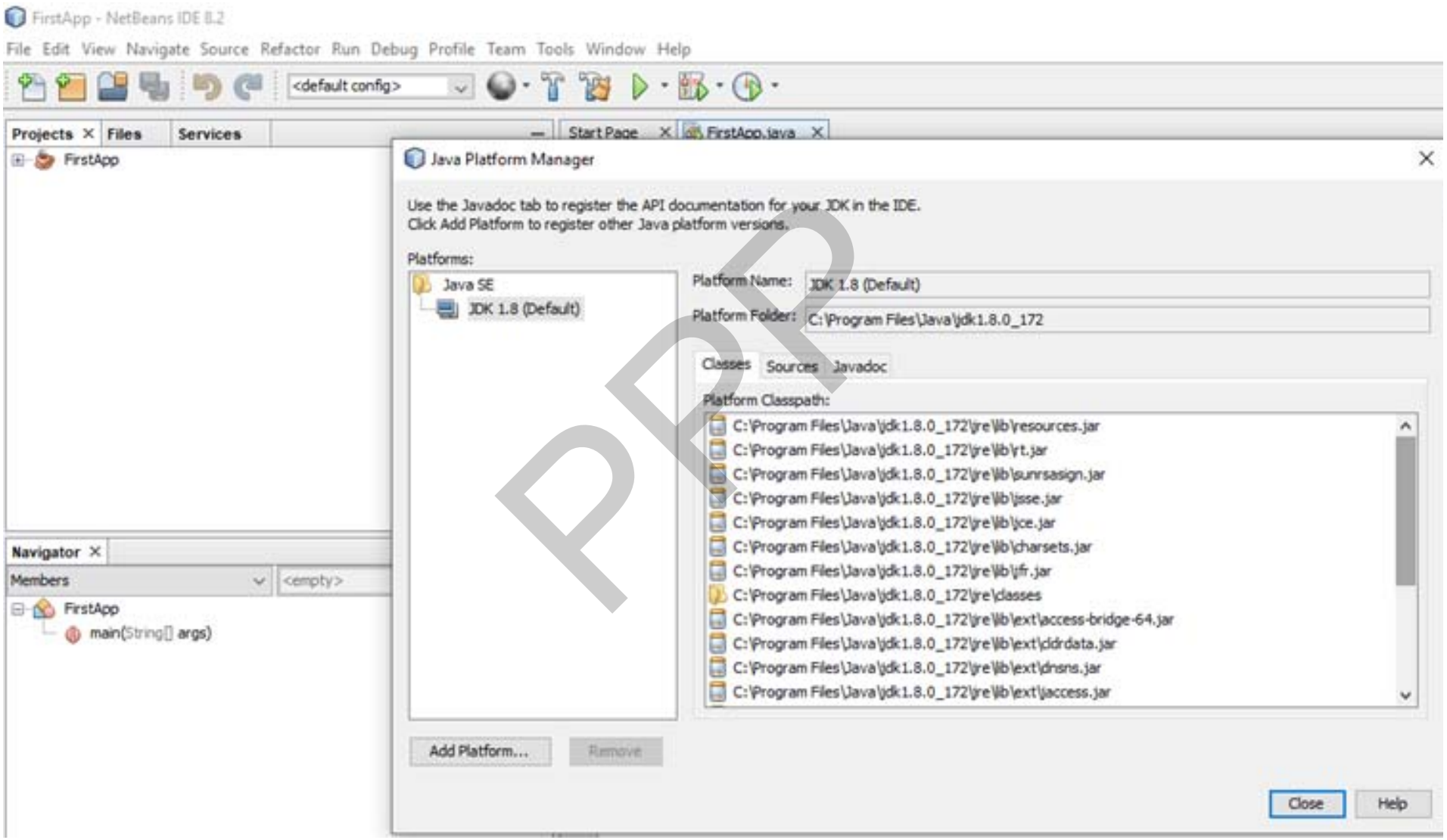
The **Navigator window**, which you can use to quickly navigate between elements within the selected class.


FirstApp - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services
FirstApp
Navigator
Members
<empty>
FirstApp
main(String[] args)
Start Page
FirstApp.java
Source
History
1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package firstapp;
7
8 /**
9 *
10 * @author Administrator
11 */
12 public class FirstApp {
13
14 /**
15 * @param args the command line arguments
16 */
17 public static void main(String[] args) {
18 // TODO code application logic here
19 }
20
21 }
22

Add JDK 8 to the Platform List **(if necessary)**

It may be necessary to add JDK 8 to the IDE's list of available platforms. To do this, choose Tools | Java Platforms as shown in the following figure:



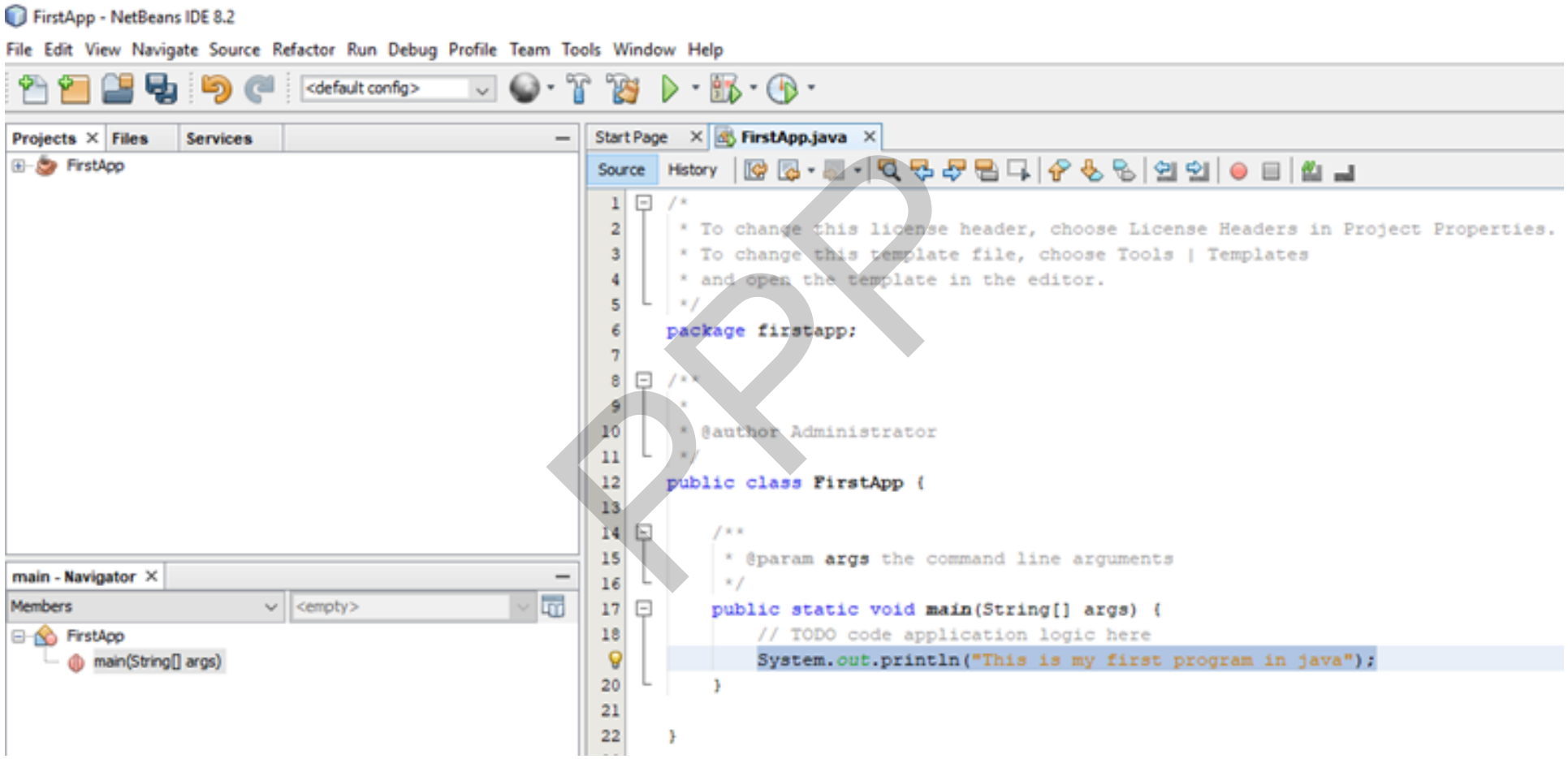

FirstApp - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services
FirstApp
Start Page
FirstApp.java
Navigator
Members
<empty>
FirstApp
main(String[] args)
Java Platform Manager
Use the Javadoc tab to register the API documentation for your JDK in the IDE.
Click Add Platform to register other Java platform versions.
Platforms:
Java SE
JDK 1.8 (Default)
Platform Name:
JDK 1.8 (Default)
Platform Folder:
C:\Program Files\Java\jdk1.8.0_172
Classes
Sources
Javadoc
Platform Classpath:
C:\Program Files\Java\jdk1.8.0_172\jre\lib\resources.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\rt.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\sunrsasign.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\jsse.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\jce.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\charsets.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\jfr.jar
C:\Program Files\Java\jdk1.8.0_172\jre\classes
C:\Program Files\Java\jdk1.8.0_172\jre\lib\ext\access-bridge-64.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\ext\cldrdata.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\ext\dnsns.jar
C:\Program Files\Java\jdk1.8.0_172\jre\lib\ext\jaccess.jar
Add Platform...
Remove
Close
Help

Add Code to the Generated Source File

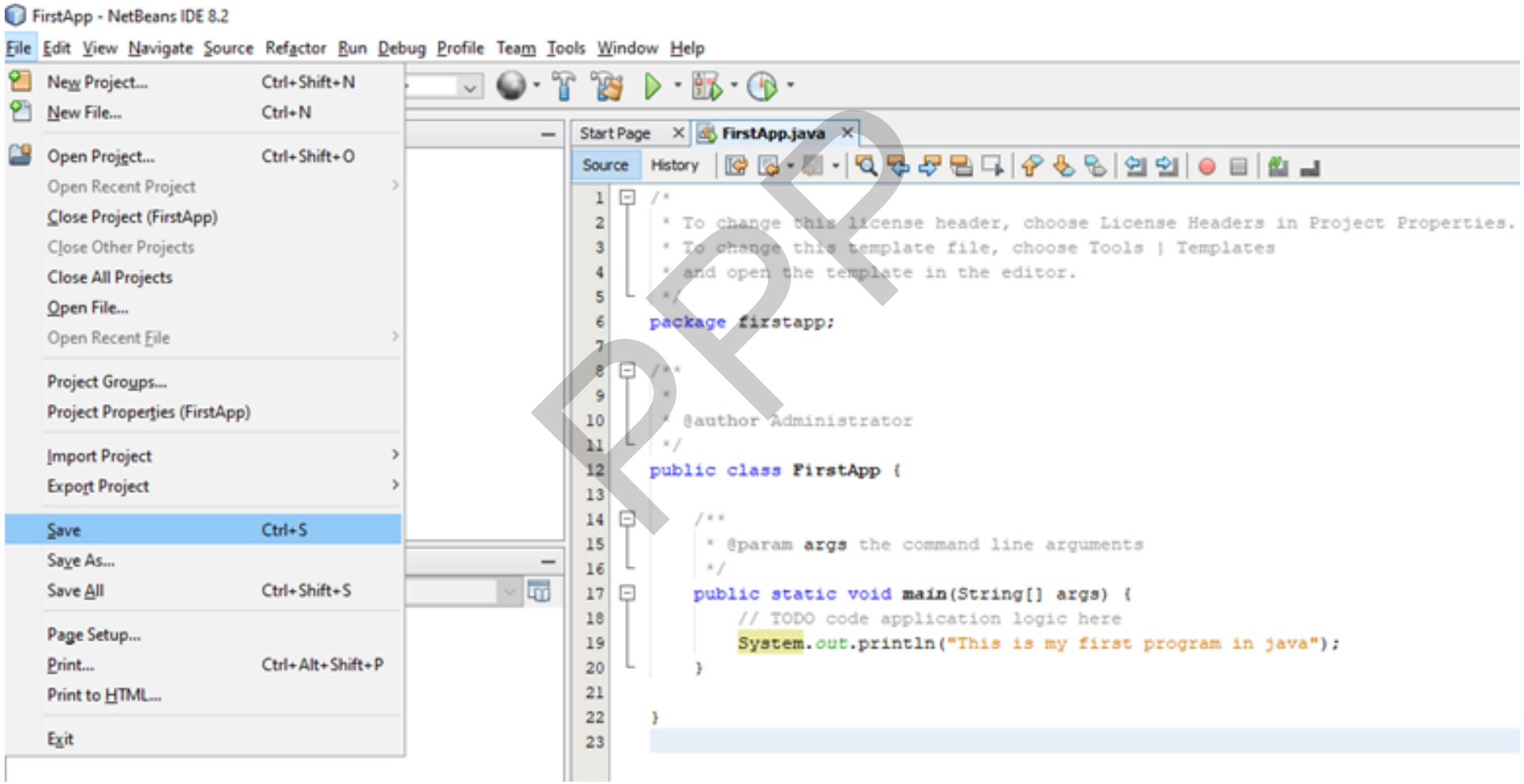
When you created this project, you left the Create Main Class checkbox selected in the New Project wizard.

The IDE has therefore created a skeleton class for you. You can add the "This is my first program in java" message to the skeleton code.

System.out.println("This is my first program in java");

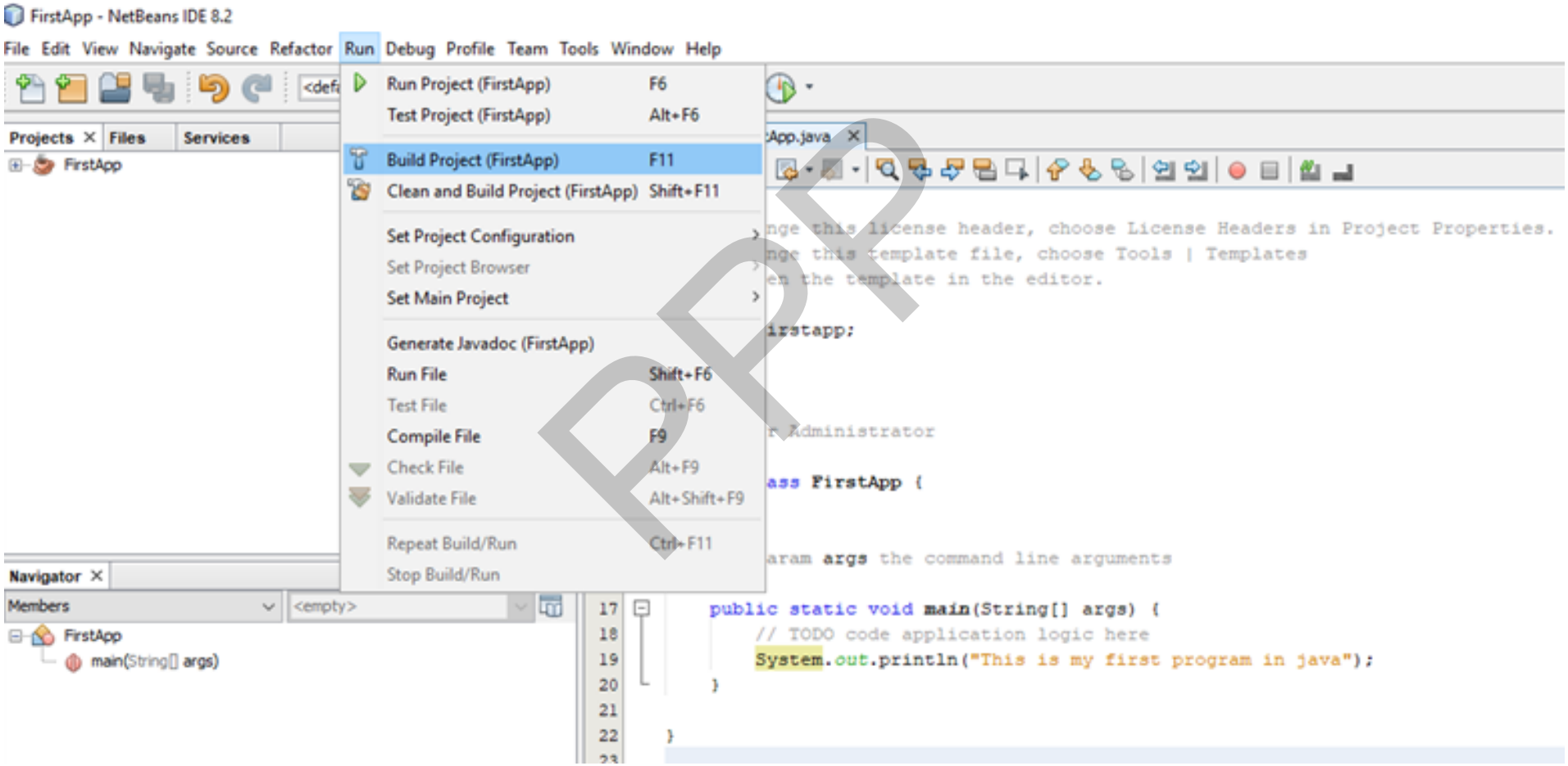

FirstApp - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services
FirstApp
main - Navigator
Members
<empty>
FirstApp
main(String[] args)
Start Page
FirstApp.java
Source
History
1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package firstapp;
7
8 /**
9 *
10 * @author Administrator
11 */
12 public class FirstApp {
13
14 /**
15 * @param args the command line arguments
16 */
17 public static void main(String[] args) {
18 // TODO code application logic here
19 System.out.println("This is my first program in java");
20 }
21
22 }

Save your changes by choosing **File** | **Save**.

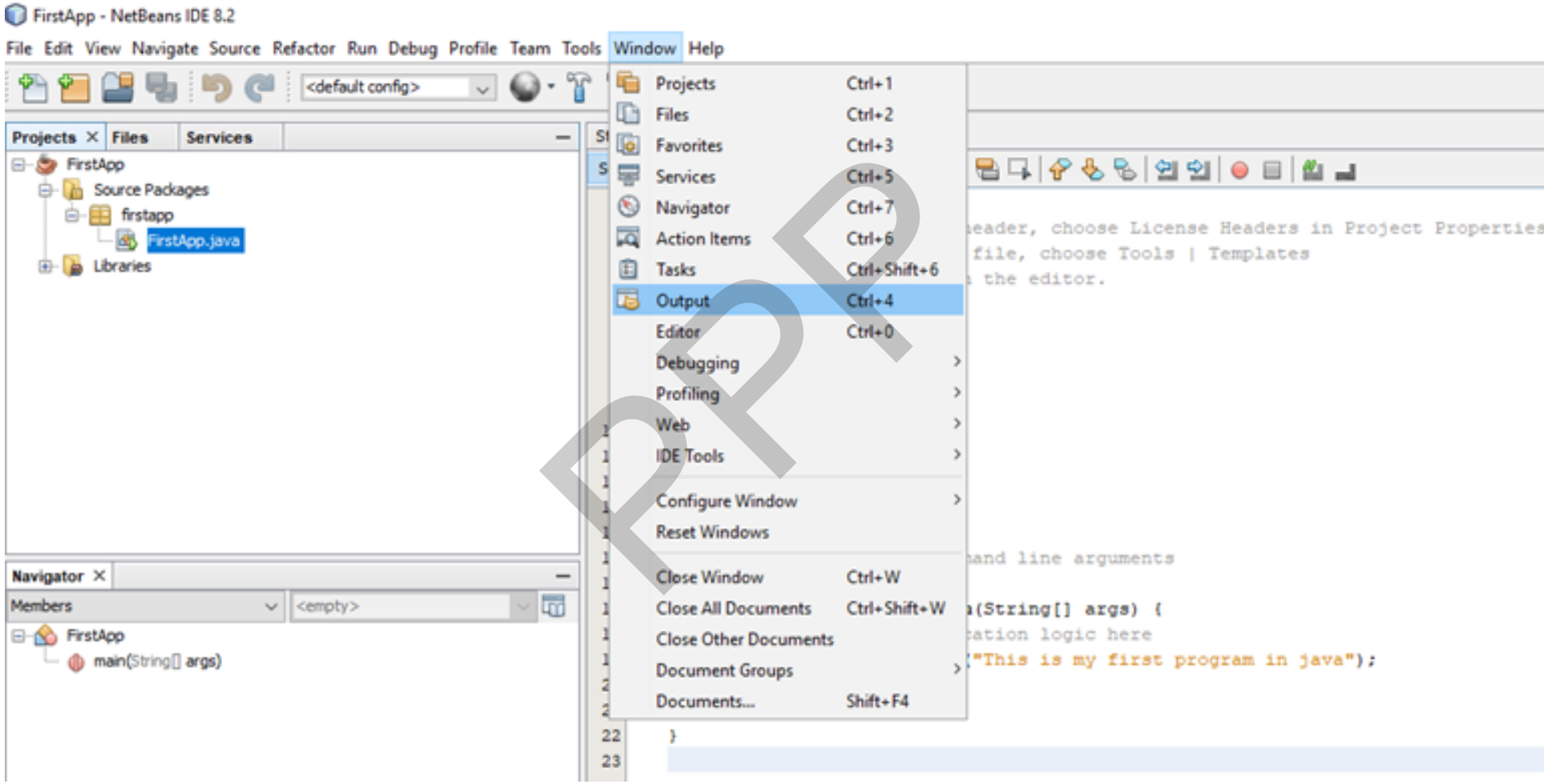


Compile the Source File into a .class File

To compile your source file, choose **Run** | **Build Project (Hello World App)** from the IDE's main menu. **(Short cut key F11)**



If output window does not shown than click on **Windows -> Output** as shown below.

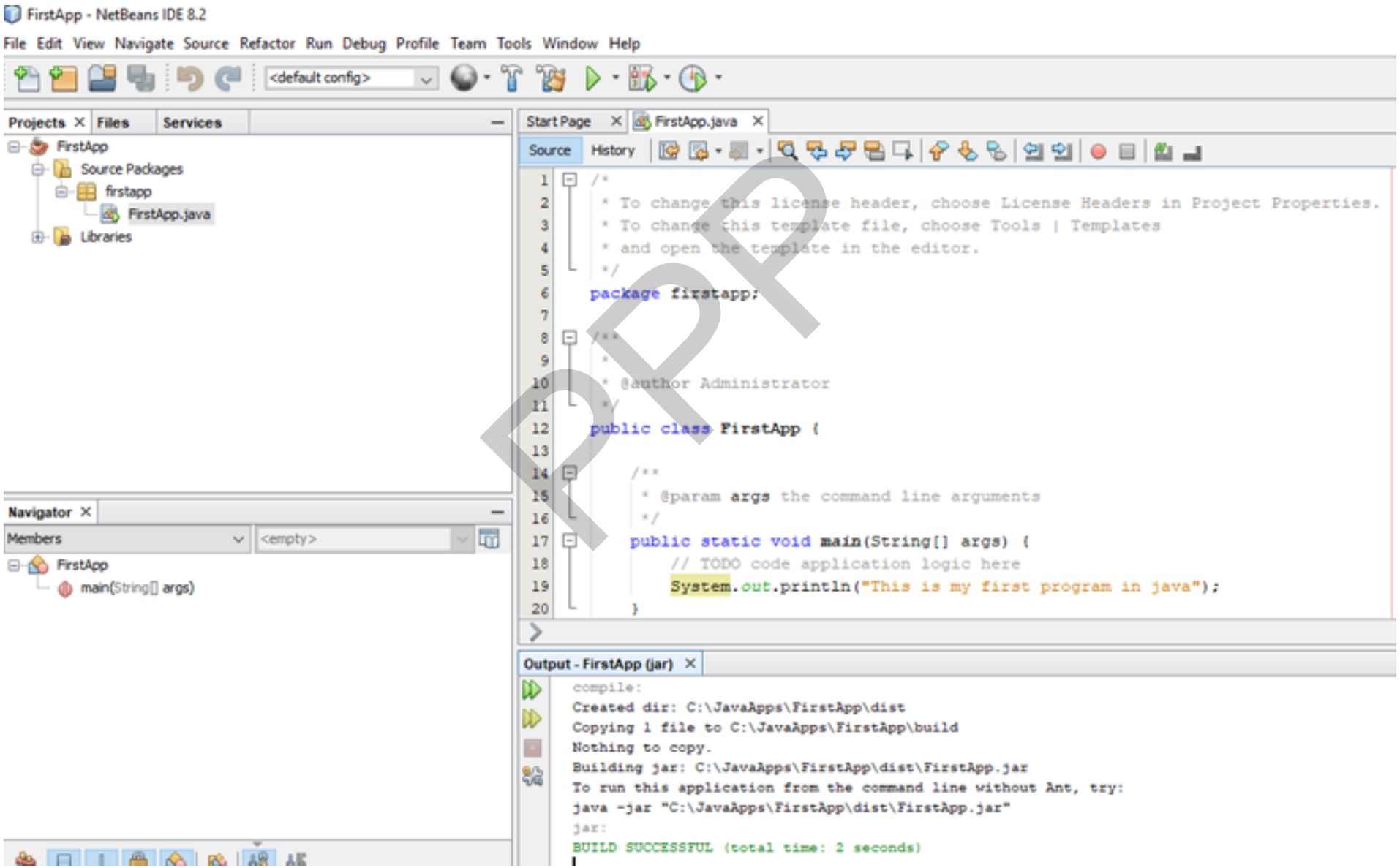


If the build output concludes with the statement **BUILD SUCCESSFUL**, congratulations! **You have successfully compiled your program!**

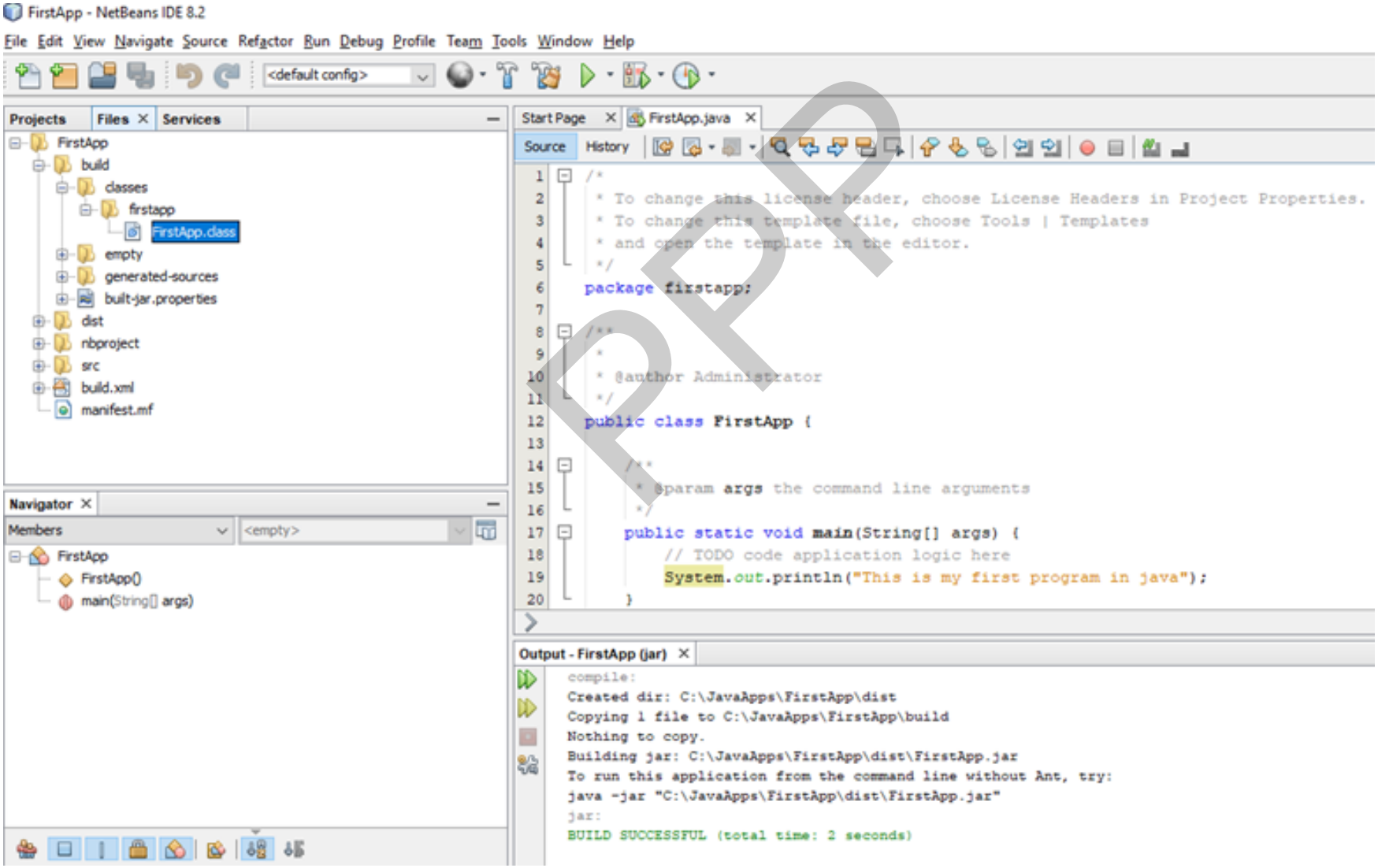
If the build output concludes with the statement **BUILD FAILED**, you probably have a **syntax error in your code**.

Errors are reported in the Output window as **hyperlinked text**. You **double-click** such a hyperlink **to navigate to the source of an error.**

You can then fix the error and once again choose Run | Build Project.

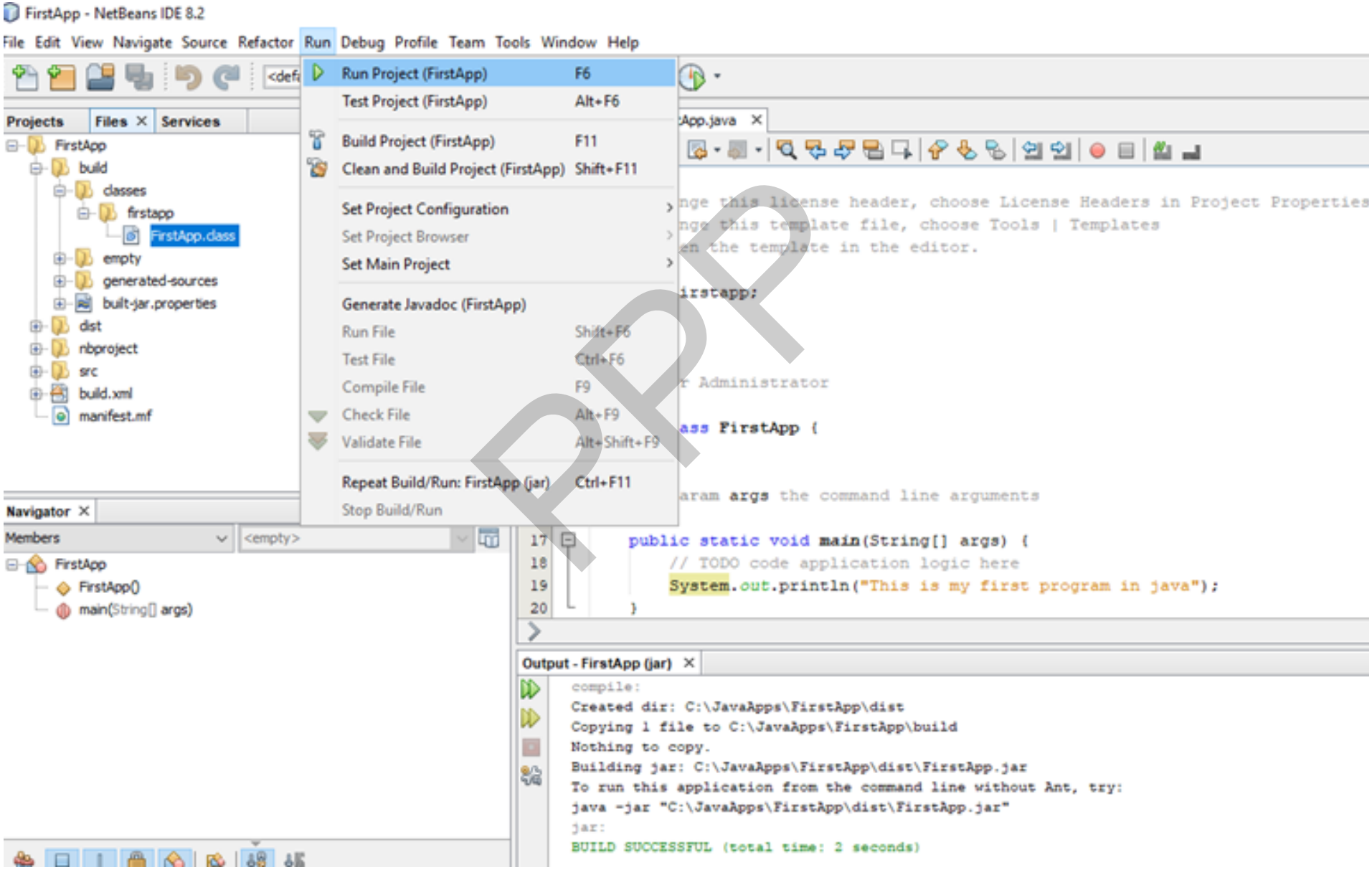

FirstApp - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services
FirstApp
Source Packages
firstapp
FirstApp.java
Libraries
Navigator
Members
<empty>
FirstApp
main(String[] args)
Start Page
FirstApp.java
Source
History
1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package firstapp;
7
8 /**
9 *
10 * @author Administrator
11 */
12 public class FirstApp {
13
14 /**
15 * @param args the command line arguments
16 */
17 public static void main(String[] args) {
18 // TODO code application logic here
19 System.out.println("This is my first program in java");
20 }
Output - FirstApp (jar)
compile:
Created dir: C:\JavaApps\FirstApp\dist
Copying 1 file to C:\JavaApps\FirstApp\build
Nothing to copy.
Building jar: C:\JavaApps\FirstApp\dist\FirstApp.jar
To run this application from the command line without Ant, try:
java -jar "C:\JavaApps\FirstApp\dist\FirstApp.jar"
jar:
BUILD SUCCESSFUL (total time: 2 seconds)

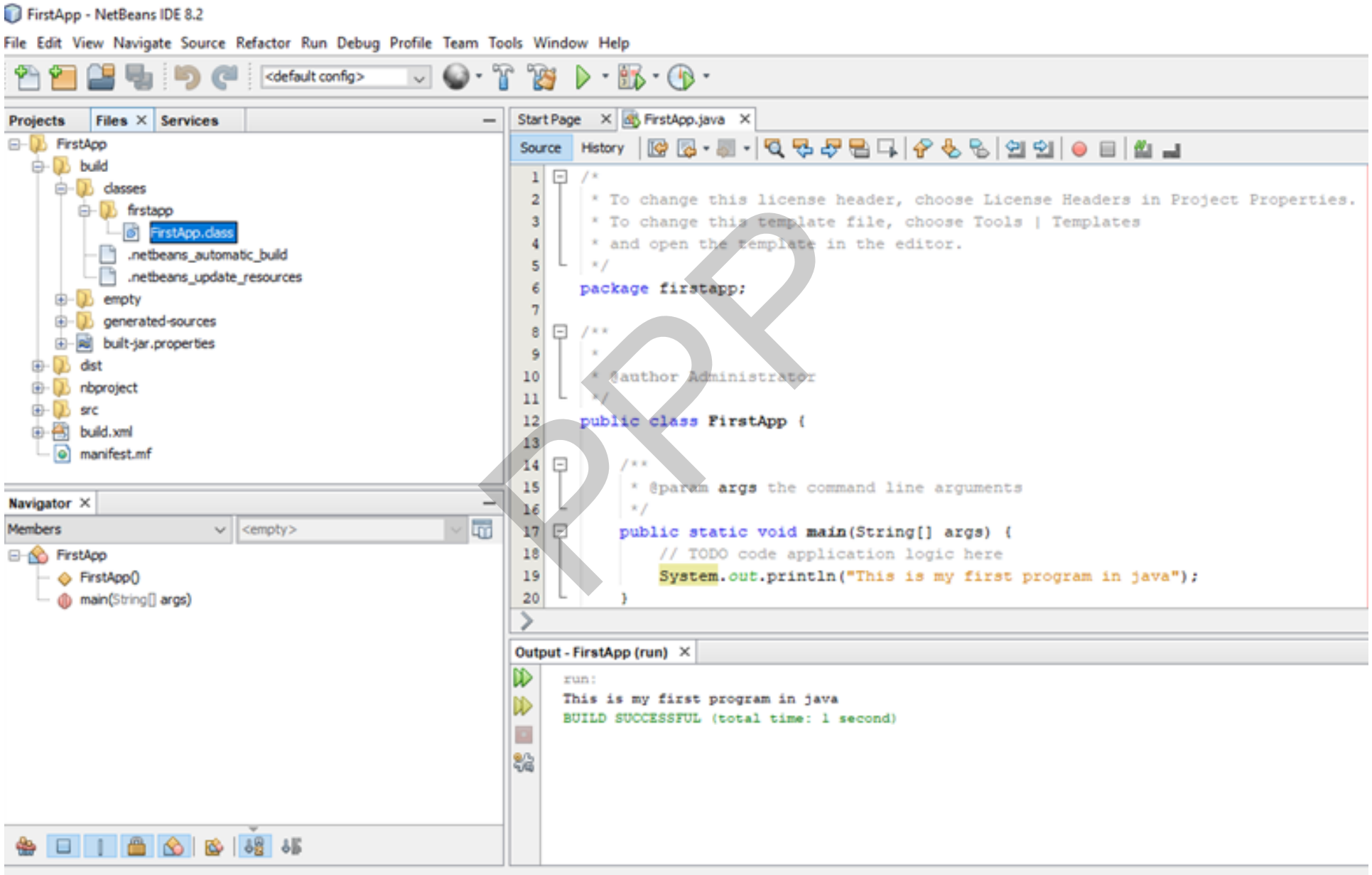
When you build the project, the bytecode file **FirstApp.class** is generated You can see where the new file is generated by opening the **Files** window and expanding the **FirstApp/build/classes/firstapp** node as shown in the following figure.



Run the Program

From the IDE's menu bar, choose **Run** | **Run Main Project**. **(Short cut key F6)**
The next figure shows what you should now see.




FirstApp - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects
Files
Services
FirstApp
build
classes
firstapp
FirstApp.class
.netbeans_automatic_build
.netbeans_update_resources
empty
generated-sources
built-jar.properties
dist
nbproject
src
build.xml
manifest.mf
Navigator
Members
<empty>
FirstApp
FirstApp()
main(String[] args)
Start Page
FirstApp.java
Source
History
1 /*
2 * To change this license header, choose License Headers in Project Properties.
3 * To change this template file, choose Tools | Templates
4 * and open the template in the editor.
5 */
6 package firstapp;
7
8 /**
9 *
10 * @author Administrator
11 */
12 public class FirstApp {
13
14 /**
15 * @param args the command line arguments
16 */
17 public static void main(String[] args) {
18 // TODO code application logic here
19 System.out.println("This is my first program in java");
20 }
Output - FirstApp (run)
run:
This is my first program in java
BUILD SUCCESSFUL (total time: 1 second)

Code Explanation

- **Every line of code that runs in Java must be inside a class.**
  In our example, we named the class FirstAPP.
  A class should always start with an uppercase first letter.
  •**Note: Java is case-sensitive:**
  "MyClass" and "myclass" has different meaning.

- **The name of the java file must match the class name.**
  In our example the file name is **FirstApp.java** and within this file the **class is FirstApp**

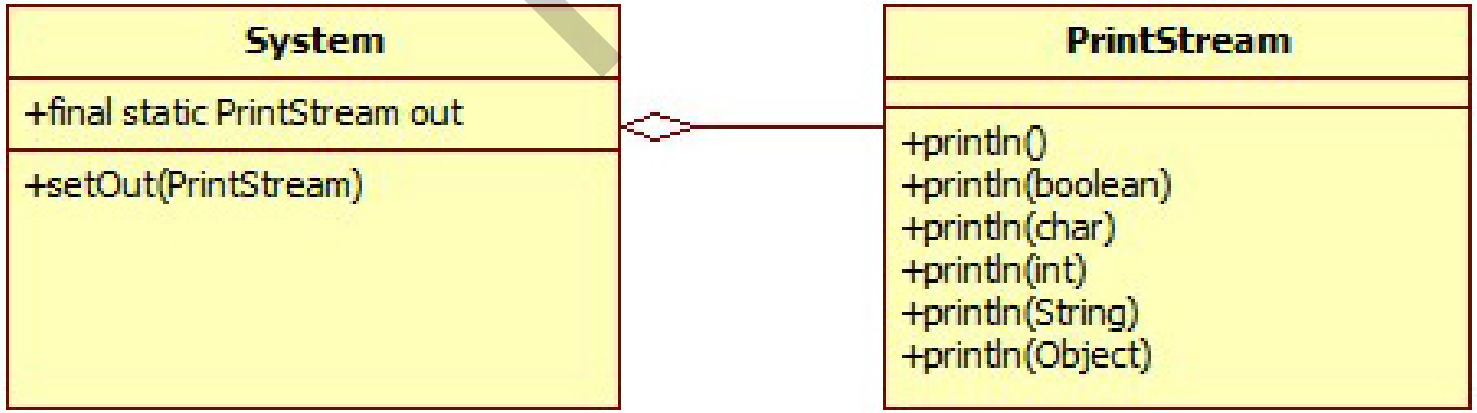
- Every program must contain the **main() method.** Any code inside the main() method will be executed.

- Inside the main() method, we can use the **println() method** to display message on the screen

What is System.out.println?

- **System** – is a final class
- **out** – is a static member field of System class and is of type PrintStream.
- **println** – is a method of PrintStream class.

- **System**. **out** is the output stream connected to the console.



What is class & object?

- **Everything in Java is associated with classes and objects.**

- **Class**
  it describes an data fields (**called variables**) and defines the operations (called **methods**).

- **Object**
  An object is created from a class.

  To create an object of **MyClass**, specify the **class name**, followed by the **object name**, and use the keyword **new**:

  E.g. MyClass obj1 = new MyClass();

Basic Input Output

Identifier

- An identifier may be any sequence of uppercase and lowercase letters, numbers or the underscore and dollar-sign characters.

- Identifiers must not begin with a number.

- It is case sensitive.

- Identifiers are used for class names, method names, and variable names.

Basic Input Output

Declaring Variable

Type identifier [= value] [, identifier [= value] …] ;

E.g. int count ; float salary ;
int basic=10000, hra, da ;

E.g.

```
int myNum = 15;
System.out.println(myNum);
```

E.g.

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

Basic Input Output

java.util.Scanner class

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Basic Input Output

To accept student_no, name & marks and display it.

```
import java.util.Scanner;
public class Student {
public static void main (String args[])
   {        int rollno;
            String name;
            double marks;
Scanner sc = new Scanner (System.in);
System.out.print("Enter Student Rollno \t");
rollno=sc.nextInt();
System.out.print("Enter Student name \t");
name = sc.next();
System.out.print("Enter Student Marks \t");
marks =sc.nextDouble();

System.out.println("Student Rollno:\t"+rollno);
System.out.println("Student Name:\t"+name);
System.out.println("Student Rollno:\t"+marks);
sc.close();
}
}
```

Naming Convention

Identifier Type	Rules for Naming	Examples
Packages	**All-lowercase letters**	java.lang java.io
Classes	**Mixed case with the first letter of each internal word capitalized**.	Integer, System, Math
Methods	**The first letter lowercase, with the first letter of each internal word capitalized**.	Integer.valueOf(s), BufferedReader.readLine()
Variables	**The first letter lowercase, with the first letter of each internal word capitalized**.	int employeeId double itemPrice int numberOfUsers
Constants	**All uppercase with words separated by underscores ("_").**	**static int** MIN_WIDTH = 4;
Interfaces	Interface names should be capitalized **like class names**.	

Operators

- Java operators are mainly categorized into the following four groups:
  - Arithmetic operators,
  - Bitwise Operators,
  - Relational operators,
  - Logical operators

Arithmetic Operators

Operation	Java Operator	Example	Value (x = 10, y = 7, z = 2.5)
Addition	+	x + y	17
Subtraction	-	x - y	3
Multiplication	*	x * y	70
Division	/	x / y	1
		x / z	4.0
Modulo division (remainder)	%	x % y	3

Order of Precedence

( ) evaluated first, inside-out

*, /, or % evaluated second, left-to-right

+, – evaluated last, left-to-right

Example: Sum of two integer

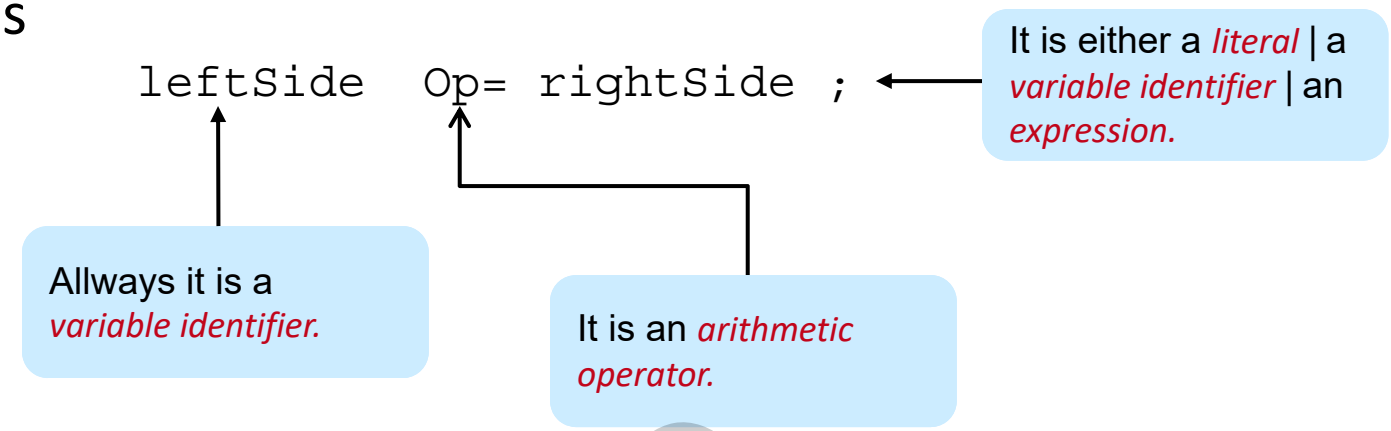
```
public class Sum {

 // main method
    public static void main( String args[] ){
                  int a, b, sum;
                  a = 20;
                  b = 10;
                  sum = a + b;
               System.out.println(a + " + " + b + "  = " + sum);
    } // end main

} // end class Sum
```

Arithmetic/Assignment Operators

Java allows combining arithmetic and assignment operators into a single operator:

Addition/assignment	**+=**
Subtraction/assignment	**–=**
Multiplication/assignment	***=**
Division/assignment	**/=**
Remainder/assignment	**%=**

- The syntax is



- This is equivalent to:

```
leftSide = leftSide Op rightSide ;
```

If X = 10 then

- X+=5; ⇔ x = x + 5; o/p x = 15
- X*=5; ⇔ x = x * 5; o/p x = 50
- x%=5; ⇔ x = x % 5; o/p x = 0

- X *= y+w*z; ⇔ x = x * (y+w*z);

Increment/Decrement Operators

Only use ++ or – – when a variable is being incremented/decremented as a statement by itself.

x++; is equivalent to x = x+1;
x--; is equivalent to x = x-1;

++x; is equivalent to x = x+1;
--x; is equivalent to x = x-1;

The position of the ++ or -- is important here. It is postfix or prefix.

x++ means to return the value of x first then increment (++) it after
++x means to increment (++) first then return the value of x

Regardless of prefix or postfix, the variable is sure to be incremented by 1 or decremented by 1.

Increment/Decrement Operators

```
public static void main( String args[] ){
                int a, b;
                a = 10;
                a = a + 1;
                System.out.println( "Value of a is ::"+ a);
                      a++;
                System.out.println( "Value of a is ::" + a);

                      ++a;
                System.out.println( "Value of a is ::" + a);

                      a = 10;
                      b = a++;
                System.out.println( "Value of a is ::" + a);
                System.out.println( "Value of b is ::" + b);

                      a = 10;
                      b = ++a;
                System.out.println( "Value of a is ::" + a);
                System.out.println( "Value of b is ::" + b);
  }
```

Relational Operators

- Relational operators compare two values
- They Produce a *boolean* value (**true** or **false)** depending on the relationship

Operation	Is true when
a >b	**a** is greater than **b**
a >=b	**a** is greater than or equal to **b**
a ==b	a is equal to **b**
a !=b	**a** is not equal to **b**
a <=b	**a** is less than or equal to **b**
a <b	**a** is less than **b**

Logical Operators

Symbol	Logical Operation Name
&&	AND
||	OR
!	NOT

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

Operators Precedence

Parentheses	(), inside-out
Increment/decrement	++, --, from left to right
Multiplicative	*, /, %, from left to right
Additive	+, -, from left to right
Relational	<, >, <=, >=, from left to right
Equality	==, !=, from left to right
Logical AND	&&
Logical OR	||
Assignment	=, +=, -=, *=, /=, %=

Logical Operators (Bit Level)

- **AND** **&**
- **OR** **|**
- **XOR** **^**
- **NOT** **~**

Logical Operators (Bit Level)

& | ^ ~

```
int a = 10; // 00001010 = 10
int b = 12; // 00001100 = 12
```

&
AND

```
a       00000000000000000000000000001010    10
b       00000000000000000000000000001100    12
a & b   00000000000000000000000000001000     8
```

|
OR

```
a       00000000000000000000000000001010    10
b       00000000000000000000000000001100    12
a | b   00000000000000000000000000001110    14
```

^
XOR

```
a       00000000000000000000000000001010    10
b       00000000000000000000000000001100    12
a ^ b   00000000000000000000000000000110     6
```

~
NOT

```
 a      00000000000000000000000000001010    10
~a      11111111111111111111111111110101   -11
```

Shift Operators (Bit Level)

- **Shift Left <<**
- **Shift Right >>**

<< Left

```
a          00000000000000000000000000000011      3
a << 2     00000000000000000000000000001100     12
```

>> Right

```
a          00000000000000000000000000000011      3
a >> 2     00000000000000000000000000000000      0
```

Control Statements

In Java, control statements can be divided into the following three categories:

- Selection / Conditional Statements
- Iterative / Looping Statements
- Jump / Breaking Statements

Types of Selection Statements

- Simple if
- if … else
- Nested if
- Switch

Simple IF Statement

•Syntax:

```
if ( Condition )
{
    Statements ;
}
```

E.g.

```
if ( num > 0 ) {
    pcount = pcount + 1;
}
```

•If single statement in the body part than no need of braces. But if multiple statements than braces are required.

if –else Statement

•Syntax:

```
 if (expression )
{
      statements;
}
 else
{
     statements;
}
```


E.g. if (a > b) { System.out.println(" A > B"); }
else { System.out.println(" B > A"); }

Conditional Operator : Ternary Operator (? :)

The meaning of **ternary** is composed of three parts.

The **ternary operator (? :)** consists of three operands.

It is used to evaluate Boolean expressions.

The operator decides which value will be assigned to the variable.

It can be used instead of the if-else statement.

It makes the code much more easy, readable, and shorter.

Syntax : variable x = (expression) ? value if true : value if false

```
public static void main( String args[] )
{           int a, b , ans;
            a = 10; b = 20;

            ans = ( a > b ) ? a : b ;
            System.out.println( "Value of ans ::" + ans);
}
```

If …. else if Statements

Syntax:

```
If(condition-1)
{       Statements;   }
else if (condition-2)
{       Statements;   }
.
.
else if (condition-n)
{       Statements;   }
else
{       Statements;   }
```

If …. else if Statements

```
import java.util.Scanner;
 public static void main( String args[] )
{          int percentage;
Scanner sc = new Scanner (System.in);
System.out.print("Enter value of day :: \t");
percentage=sc.nextInt();
                if percentage > 70
                System.out.println( " Grade is Distinction");
                else if percentage > 60
                System.out.println( " Grade is First Class");
                else if percentage > 35
                System.out.println( " Grade is Second Class");
        else
                System.out.println( " Grade is Fail");
}
```

Nested If Statements

```
If(condition)
{
if (condition)
{       Statements;     }
else
{       Statements;     }
}
else
{
if (condition)
{       Statements;     }
else
{       Statements;     }
}
```

```
if ( a > b )
{       If ( a > c )
        {
                System.out.println(" A is the biggest number");
        }
        else
        {
                System.out.println(" C is biggest number");
        }
}
else
{       If ( b > c )
        {
                System.out.println(" B is the biggest number");
        }
        else
        {
                System.out.println(" C  is biggest number");
        }
```

Switch Statement

```
switch (expression) {
case value_1 :
                statement(s);
                break;
case value_2 :
                statement(s);
                break;
. . .
case value_n :
                statement(s);
                break;
default:
                statement(s);
}
```

Switch Statement

```
import java.util.Scanner;
 public static void main( String args[] )
{           int day;
Scanner sc = new Scanner (System.in);
System.out.print("Enter value of day :: \t");
day=sc.nextInt();
switch (day) {
        case 1 : System.out.println( "Sunday"); break;
        case 2 : System.out.println( "Monday"); break;
        case 3 : System.out.println( "Tuesday"); break;
        case 4 : System.out.println( "Wednesday"); break;
        case 5 : System.out.println( "Thursday"); break;
        case 6 : System.out.println( "Friday"); break;
        case 7 : System.out.println( "Saturday"); break;
}
```

Types of Looping Structure

Loops are used to execute a set of instructions/functions repeatedly when some conditions become true.

There are three types of loops in Java

- while loop
- do … while loop
- for loop

Comparison of Looping Structures

Comparison	for loop	while loop	do while loop
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	for(initialize; condition; increment/decrement)
	{
	// code
	}
		while(condition)
		{
		//code
		}
			Do
			{
			//code
			}
			while(condition);

Example	for(int i=1;
	i<=10;
	i++) {
	System.out.println(i);
	}
		int i=1;
		while(i<=10){
		System.out.println(i);
		i++;
		}
			int i=1;
			do{
			System.out.println(i);
			i++;
			}
			while(i<=10);

Syntax for infinitive loop	for(;;)
	{ //code }
		while(true)
		{ //code }
			Do
			{ //code }
			while(true);

Break Statement

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
- **Syntax :** break ;
- **Example:**

```
public class Sample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }
      System.out.println(i);
   } } }
```

Continue Statement

- The continue statement is used in loop control structure when you need to **jump to the next iteration of the loop** immediately.
- It can be used with for loop or while loop.
- It continues the current flow of the program and skips the remaining code at the specified condition.
- In case of an inner loop, it continues the inner loop only.
- **Syntax :** continue;
- **Example:**

```
public class Sample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //continue the loop . it will skip the rest statement
         continue;
      }
      System.out.println(i);
   } } }
```

String Handling

Java Strings are Immutable

Once we create a string, we cannot change that string.

•// create a string

String example = "Hello! ";

•Now suppose we want to change the string.

•// add another string "World" to the previous string example

example = example.concat(" World");

•concat() method to add another string World to the previous string.

•**How it works?**

- JVM takes the first string "Hello! "
- Creates a new string by adding "World" to the first string
- Assign the new string "Hello! World" to the example variable
- The first string "Hello! " remains unchanged

String Handling

Escape character in Java Strings

•The escape character is used to escape some of the characters present inside a string.

•**To include double quotes inside a string.**

•// include double quote
String example = "This is the "String" class";

•Since strings are represented by double quotes, the compiler will treat "This is the " as the string. Hence, the above **code will cause an error.**

•To solve this issue, we use the escape character \ in Java. For example,
•// use the escape character
String example = "This is the \"String\" class.";
•Now escape characters tell the compiler to escape double quotes and read the whole text.

String Handling

- In Java, a string is a sequence of characters.
- For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.
- We use double quotes to represent a string in Java.

For example,

```
// create a string
String str_var = "Java programming";
```

- Here, we have created a string variable named str_var. The variable is initialized with the string Java Programming.
- String is non-primitive data type.
- string variables are instances of the String class.

E.g.

```
public static void main(String[] args) {
   // create strings
   String str_var = "Java";
   // print strings
   System.out.println(str_var);
  }
```

String Handling

String Operations:

```
public static void main(String[] args) {
    // create strings
    String first = "Java";
    String second = "Programming";
    // get the length of greet
    int length = first.length();
    System.out.println("Length: " + length);
    // Combine two strings
    String joinedString = first.concat(second);
    System.out.println("Joined String: " + joinedString);
    // compare first and second strings
    first = "Java";
    second = "Java";
    boolean result1 = first.equals(second);
    System.out.println("Strings first and second are equal: " +
result1);
  }
```

String Handling

Using + operator

Java uses "+" operator to concatenate two string objects into single one. It can also concatenate numeric value with string object.

E.g.

```
public static void main(String[] args) {
        String s = "Hello";
        String str = "Java";
        String str1 = s+str;
        String str2 = "Java"+11;
        System.out.println(str1);
        System.out.println(str2);

System.out.println(str1.toUpperCase());
System.out.println(str1.toLowerCase());

   }
```

String Handling : Methods of String

int length(): It returns the length of a String.

public boolean isEmpty():
This method returns true if the given string has 0 length.
If the length of the specified Java String is non-zero then it returns false.

char charAt(int index):
It returns the character at the specified index.
Specified index value should be between 0 to length() -1 both inclusive.
It throws IndexOutOfBoundsException if index < 0 or >= length of String.

String toUpperCase(): to covert string to UpperCase
String toLowerCase(): to covert string to LowerCase

boolean equals(Object obj):
Compares the string with the specified string and returns true if both matches else false.

String Handling : Methods of String

int compareTo(String string): This method compares the two strings based on the Unicode value of each character in the strings. It returns 0, +ve or -ve value.

String concat(String str): Concatenates the specified string "str" at the end of the string.

int indexOf(int ch): Returns the index of first occurrence of the specified character ch in the string.

int indexOf(String str): This method returns the index of first occurrence of specified substring str.

String substring():
Extracts a substring from the string and returns it.

String trim(): Returns the substring after omitting leading and trailing white spaces from the original string.

String Handling

Creating strings using the new keyword

•Since strings in Java are objects, we can create strings using the new keyword as well.

E.g.

```
public static void main(String[] args) {

   // create a string using new
   String name = new String("Java String");

   System.out.println(name);  // print Java String
 }
```

String Handling

Create String using literals vs new keyword

- String strObject = new String("Java"); &
  String strLiteral = "Java";

  Both expression gives you String object

•In Java, the JVM maintains a string pool to store all of its strings inside the memory. **The string pool helps in reusing the strings.**

•**While creating strings using string literals**, the value of the string is directly provided. Hence, the compiler first checks the string pool to see if the string already exists. **If the string already exists**, the new string is not created. Instead, the **new reference points to the existing string**. **If the string doesn't exist, the new string is created.**

•However, while creating **strings using the new keyword**, the value of the string is not directly provided. Hence the **new string is created all the time.**

String Handling

String Comparison

•To compare string objects, Java provides methods and operators both. So we can **compare string in following three ways.**

•Using **equals()** method
•Using **==** operator
•By **compareTo()** method

Using equals() method

•equals() method compares two strings for equality. Its general **syntax : boolean equals (Object str)**
•It compares the content of the strings. It will return true if string matches, else returns false.

Using == operator

•The double equal (==) operator **compares two object references to check whether they refer to same instance.** This also, will return true on successful match else returns false.

String Handling

compareTo() method

•String compareTo() method **compares values and returns an integer value which tells if the string compared is less than, equal to or greater than the other string.**

•It compares the String based on natural ordering i.e alphabetically. Its general syntax is.

•**Syntax:** int compareTo(String str)

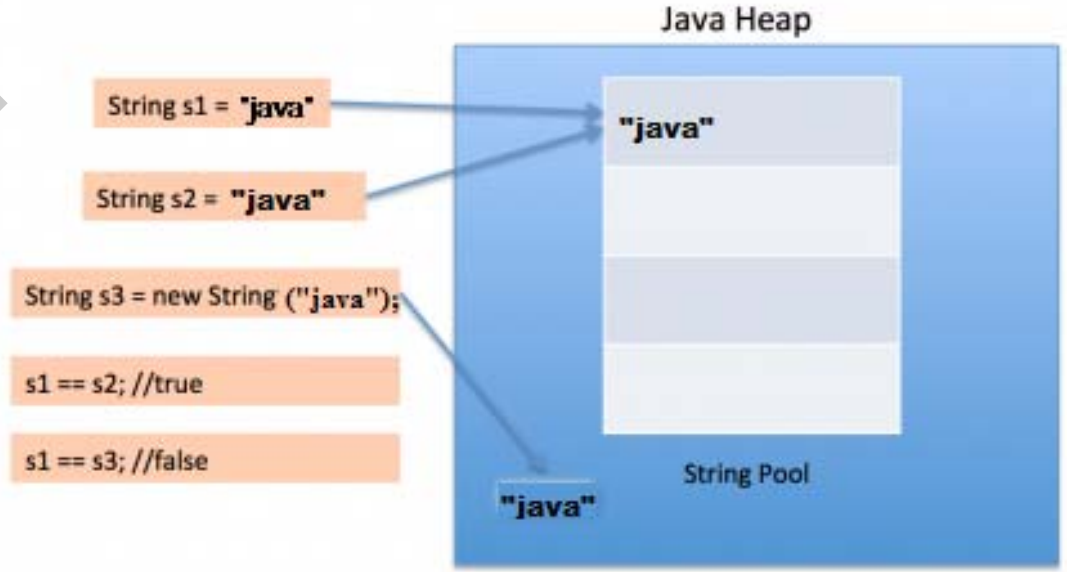
String Handling

compareTo() method

```
public static void main(String[] args) {
        String s1 = "Abhi";
        String s2 = "Viraaj";
        String s3 = "Abhi";

        int a = s1.compareTo(s2);     //return less than  0 for s1 < s2
        System.out.println(a);

        a = s1.compareTo(s3);     //return 0 for s1 == s3
        System.out.println(a);

        a = s2.compareTo(s1);     //return greater than 0 for s2 > s1
        System.out.println(a);
  }
```

String Handling

```
public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java";
        String s3 = new String ("Java");

        boolean b = (s1 == s2);     //true
        System.out.println(b);

        b =       (s1 == s3);     //false
        System.out.println(b);
  }
```

String Handling

Substr method

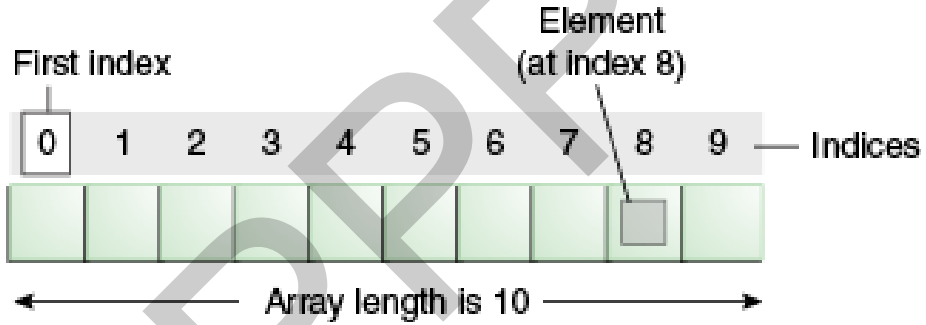
```
String s1="Computer Science";
String substr = s1.substring(0); // Starts with 0 and goes to end
System.out.println(substr);

String substr2 = s1.substring(5,10); // Starts from 5 and goes to 10
System.out.println(substr2);

String substr3 = s1.substring(5,15); // Returns Exception
```

Array

• An array is a collection of similar type of elements which has contiguous memory location.

•Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

•We can store primitive values or objects in an array in Java.



Advantages

Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.

Random access: We can get any data located at an index position.

Disadvantages

Size Limit: We can store only the **fixed size of elements in the array**. It doesn't grow its size at runtime. To solve this problem, **collection** framework is **used** in Java which **grows automatically.**

Array

- **Declare a one-dimensional array**
  - data type arrary-name [ ]; **OR**
  - data type [ ]arrary-name; **OR**
  - data type[ ] arrary-name;
- **Declaration and instantiation array**
  - data type array-name = new data type [size];

    E.g. int a[]=**new int**[5];
- **Declaration, instantiation and initialization of an array**
  - data type array-name [ ] = { list of values };

    E.g. int a[] = {1,2,3,4,5};
- **Length of an array**
  - array-name.length

Array

- **Length of an array**
  - **array-name.length**

```
public static void main(String args[])
{
int a[]={1,2,3,4,5};

//printing array
for(int i=0; i< a.length; i++)
    System.out.println(a[i]);
}
```

Array

Loop through an Array with For … Each

It is used exclusively to loop through elements in arrays:

Syntax:

```
for (type variable : arrayname) {
  ...
}
```

E.g.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    for (String i : cars) {
            System.out.println(i); }
```

for each String element (called i - as in index) in cars, print out the value of i.

It does not require a the length property of an Array.

Array

- **Declare a two-dimensional array**
  - data type arrary-name [ ] [ ]; **OR**
  - data type [ ] [ ] arrary-name; **OR**
  - data type[ ] [ ] arrary-name; **OR**
  - data type [ ] arrary-name [ ];

- **Allocate space for a two-dimensional array**
  - data type array-name = new data type [size] [size] ;
  - **E.g. int**[ ][ ] arr=**new int**[3][3]; //3 rows and 3 columns

- **Initialization of an array**
  - data type array-name [ ] [ ] = { list of values };
  - **E.g. int** arr[ ][ ] = { {1,2,3}, {2,4,5} , {4,4,5} };

- **Length of an array**
  - array-name.length // Indicate the no. of rows.
  - array-name [index] .length // Indicate the no. of columns.

//Java Program to demonstrate the addition of two matrices in Java

```
class AddMatrices{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];

//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
        c[i][j]=a[i][j]+b[i][j];
        System.out.print(c[i][j]+" ");
}
System.out.println();//new line
} } }
```