Cambodian University of Specialties

Faculty of Science and Technology

Java Introduction

Introduction to Java Programming

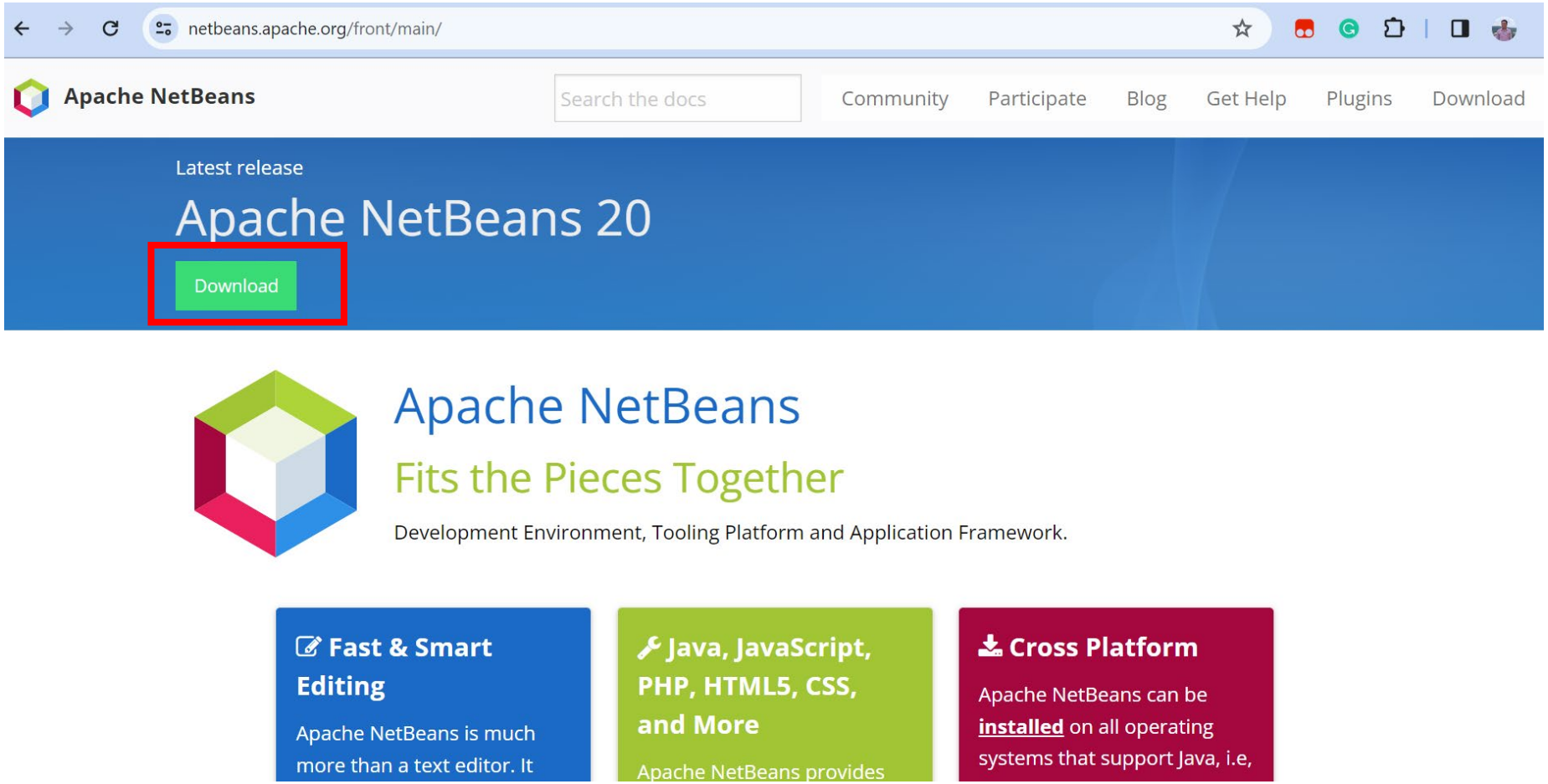
- Installing NetBeans
- Introduction to Java Programming
- Java Program Structure
- Java Data Types
- Java Operators

Introduction to Java Programming

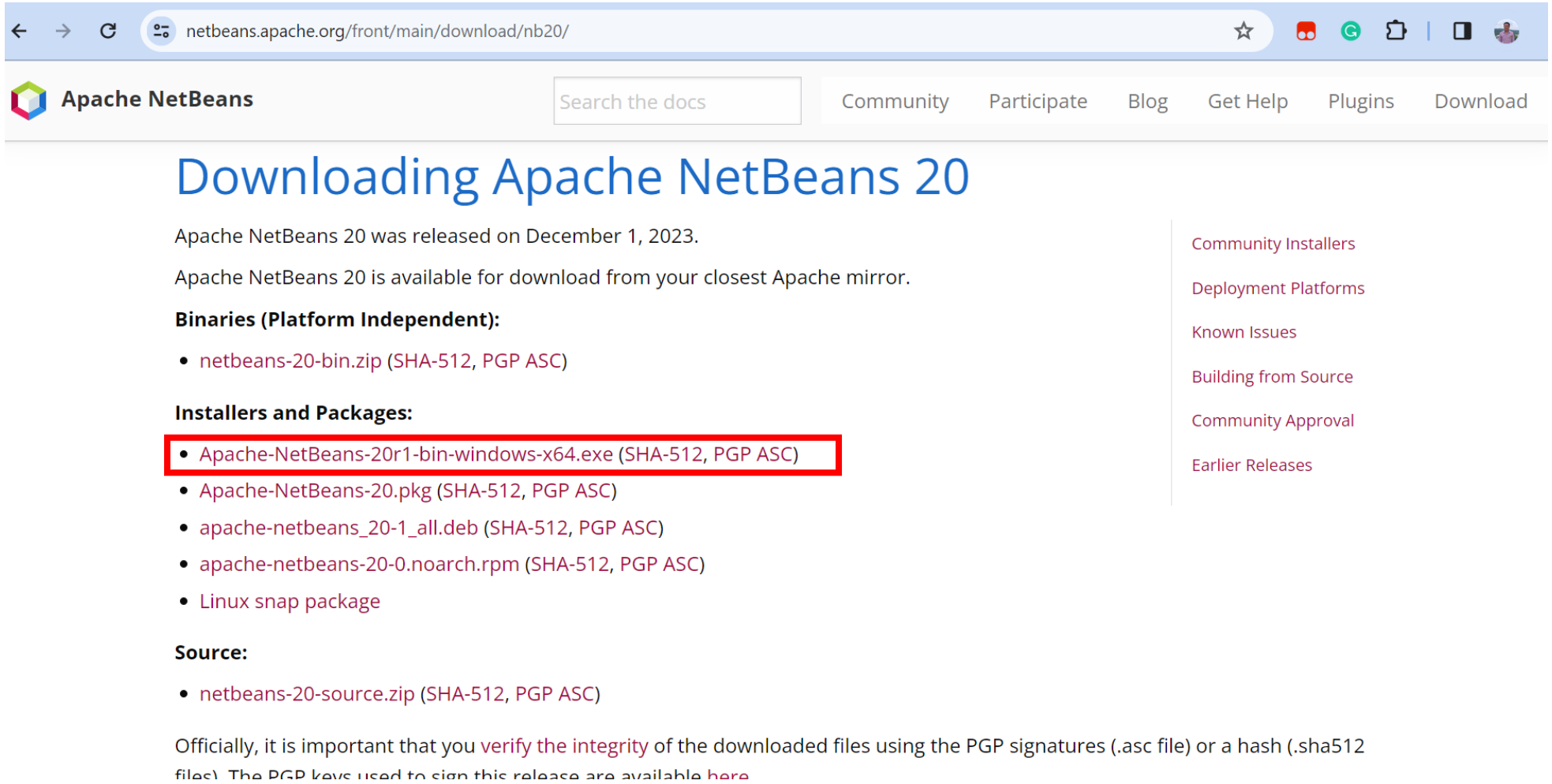
- Java Logical Arithmetic
- Java Methods
- Java Access Modifier

Installing NetBeans

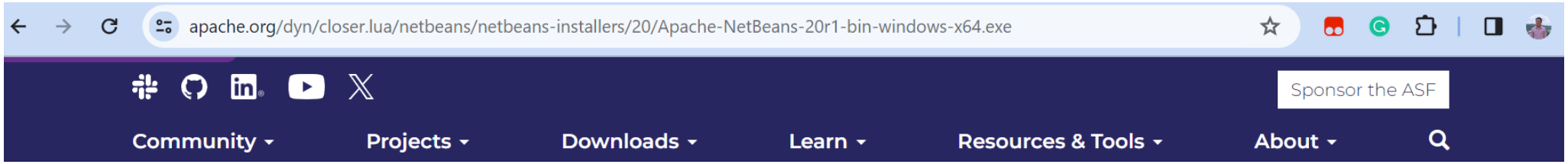
Installing NetBeans



Installing NetBeans



Installing NetBeans



THE APACHE SOFTWARE FOUNDATION ESTABLISHED 1999

We suggest the following location for your download:

https://dlcdn.apache.org/netbeans/netbeans-installers/20/Apache-NetBeans-20r1-bin-windows-x64.exe

Alternate download locations are suggested below.

It is essential that you verify the integrity of the downloaded file using the PGP signature (`.asc` file) or a hash (`.md5` or `.sha*` file).

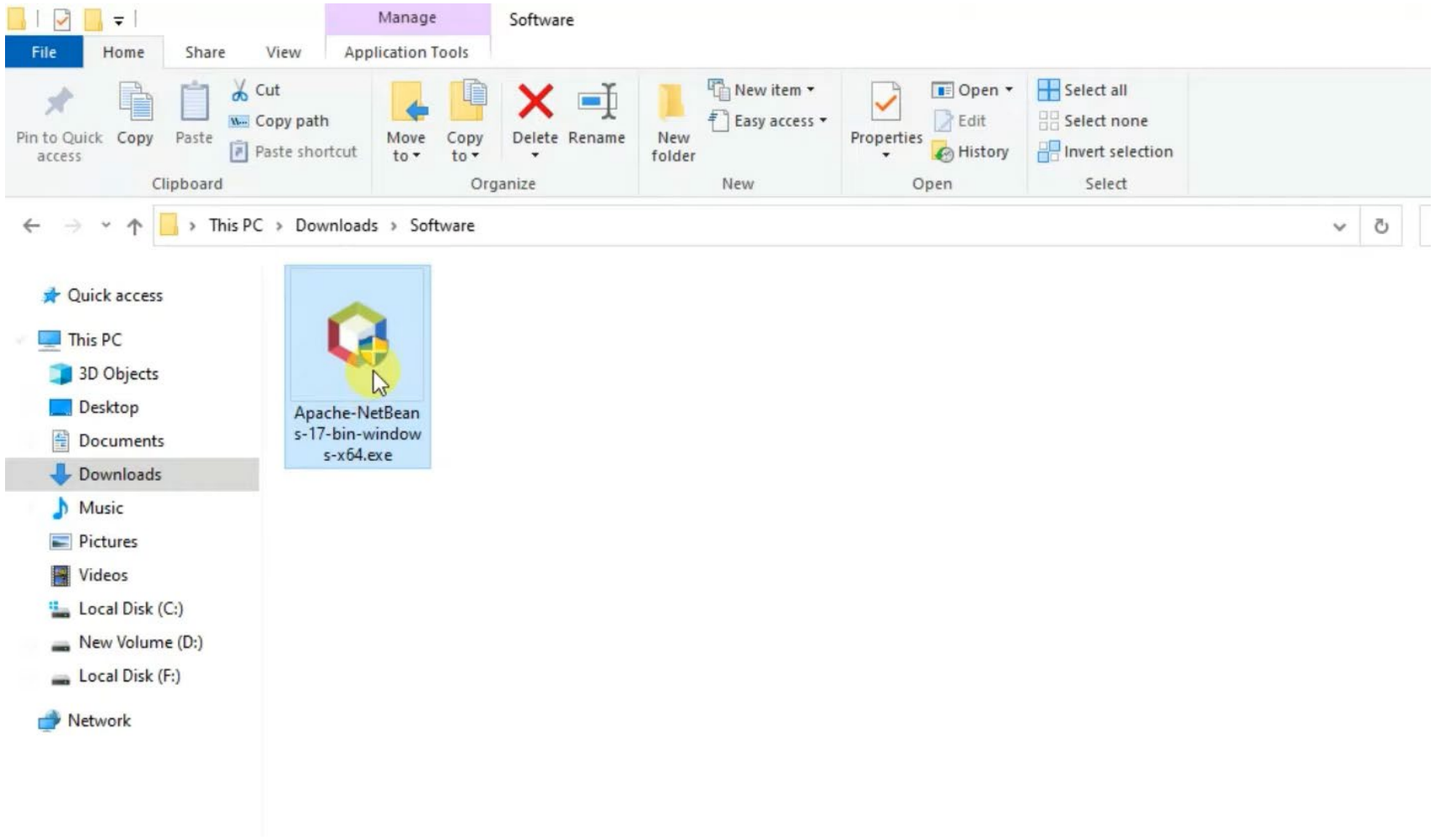
HTTP

https://dlcdn.apache.org/netbeans/netbeans-installers/20/Apache-NetBeans-20r1-bin-windows-x64.exe

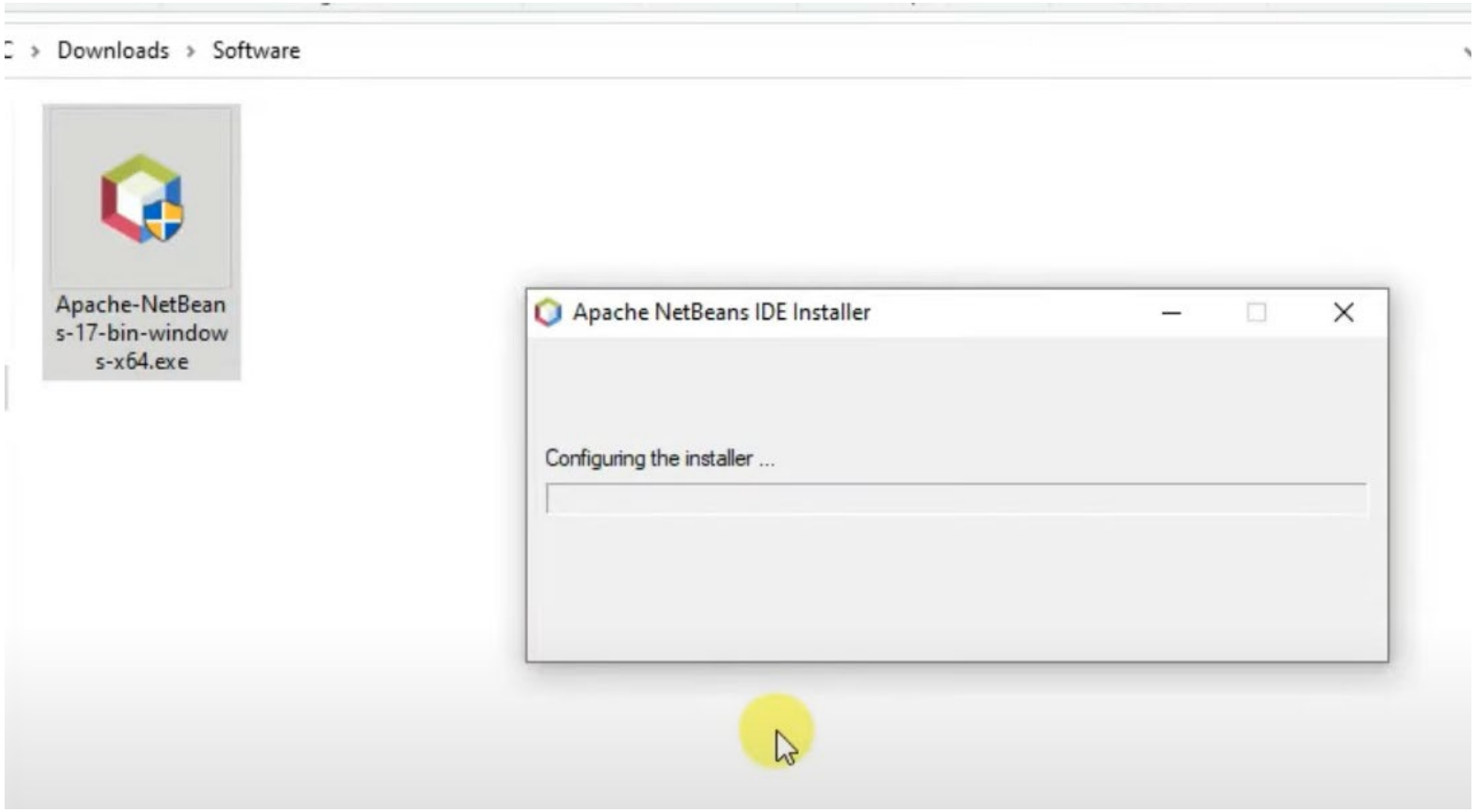
BACKUP SITES

https://dlcdn.apache.org/netbeans/netbeans-installers/20/Apache-NetBeans-20r1-bin-windows-x64.exe

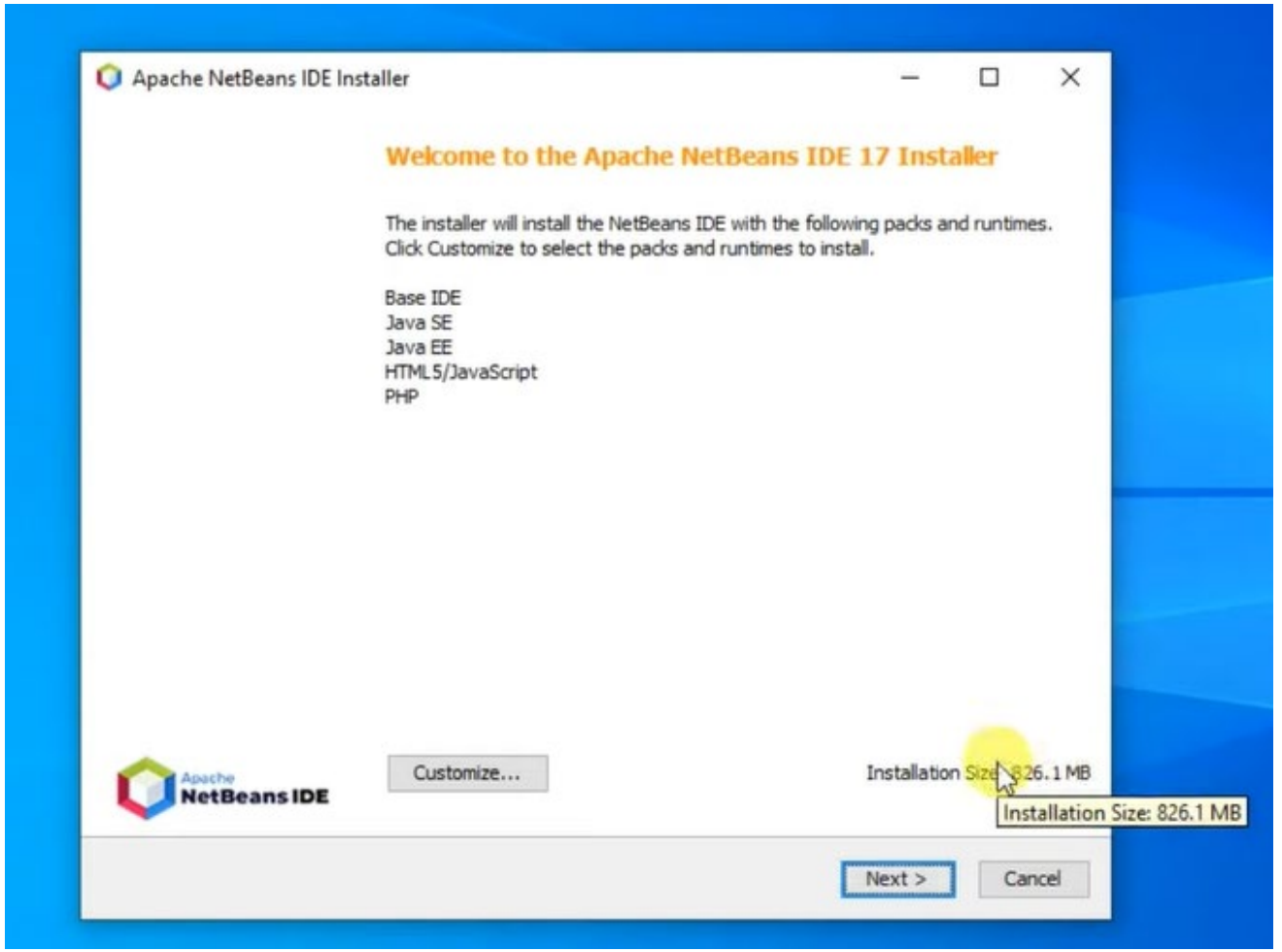
Installing NetBeans



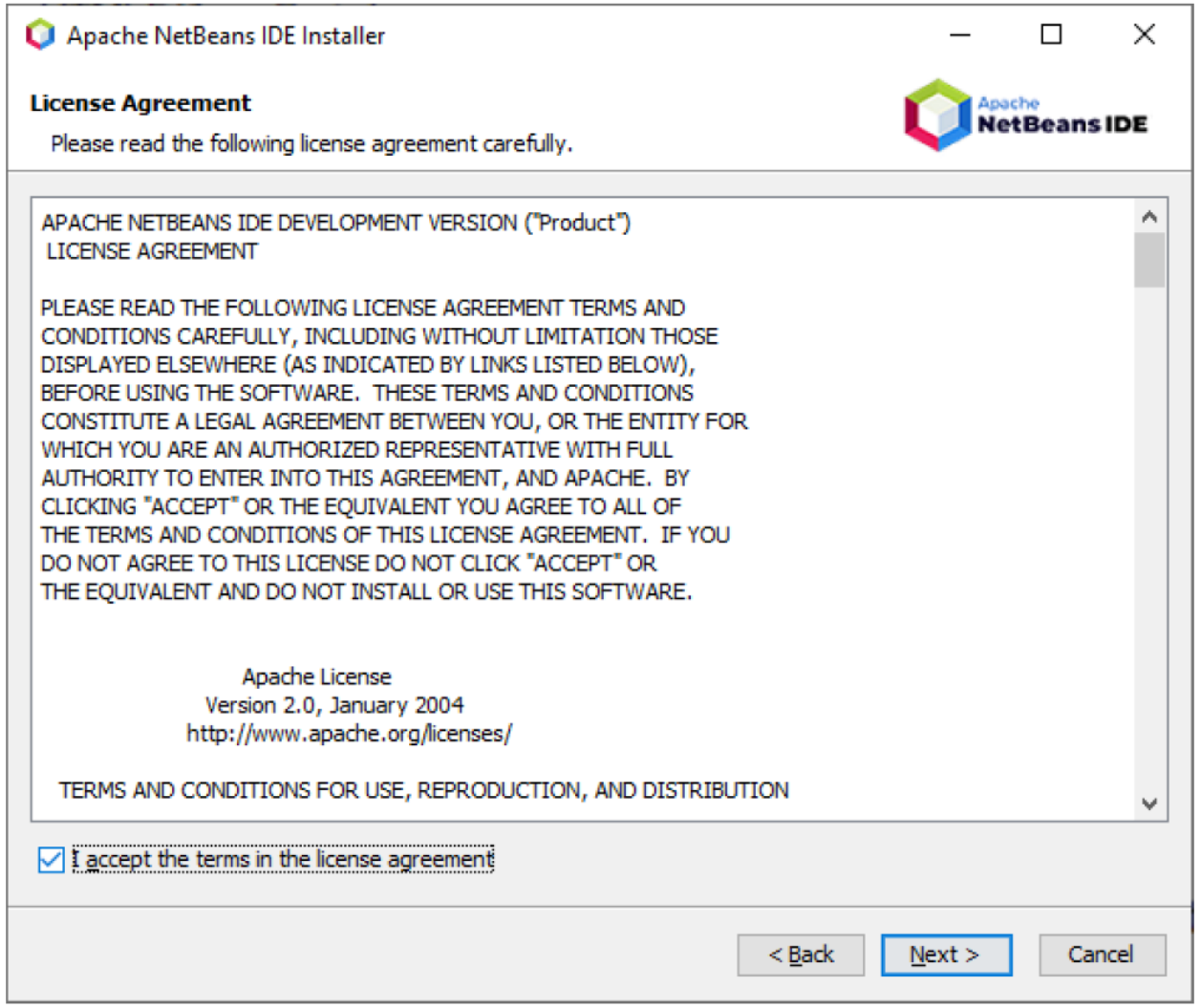
Installing NetBeans



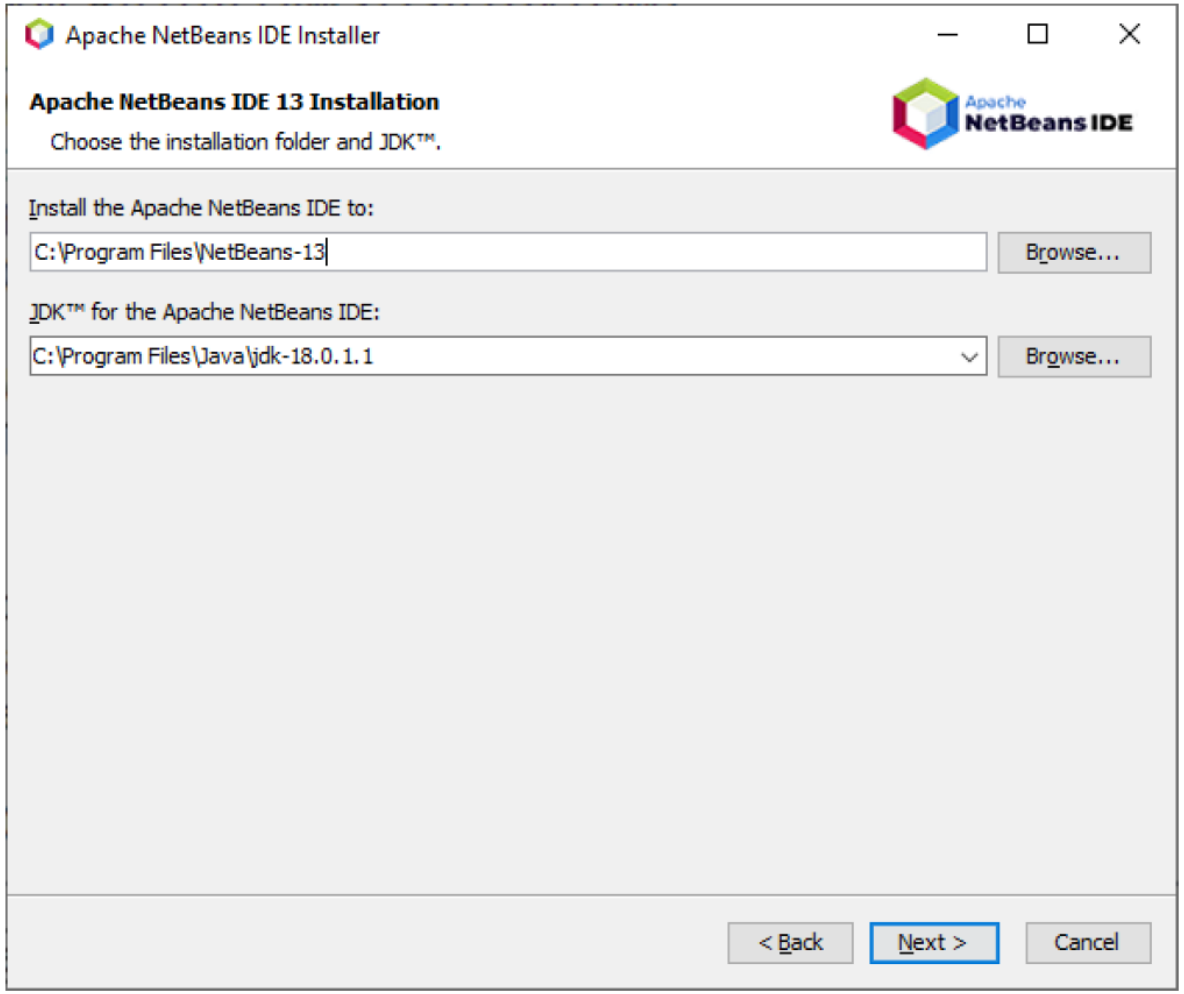
Installing NetBeans



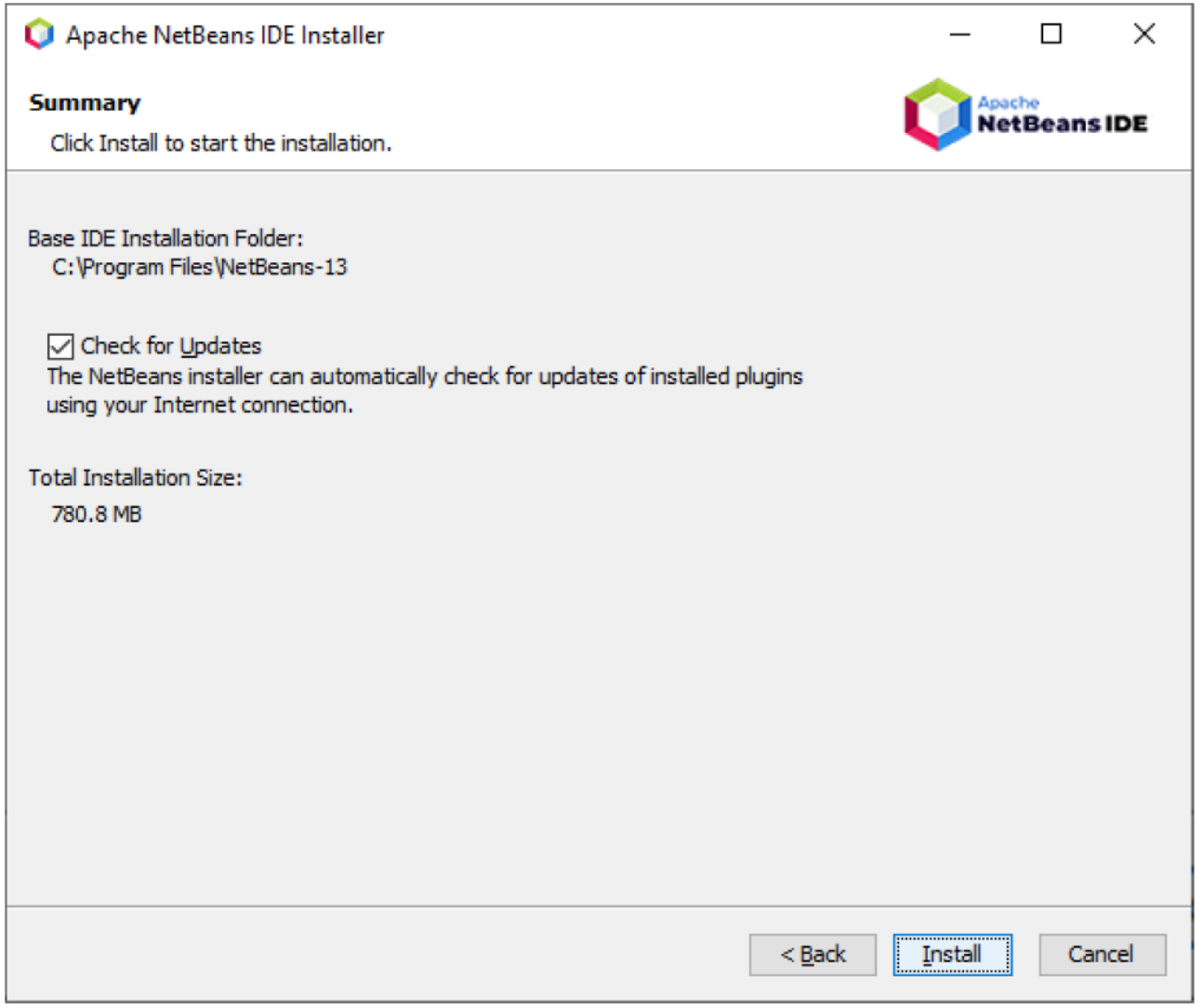
Installing NetBeans



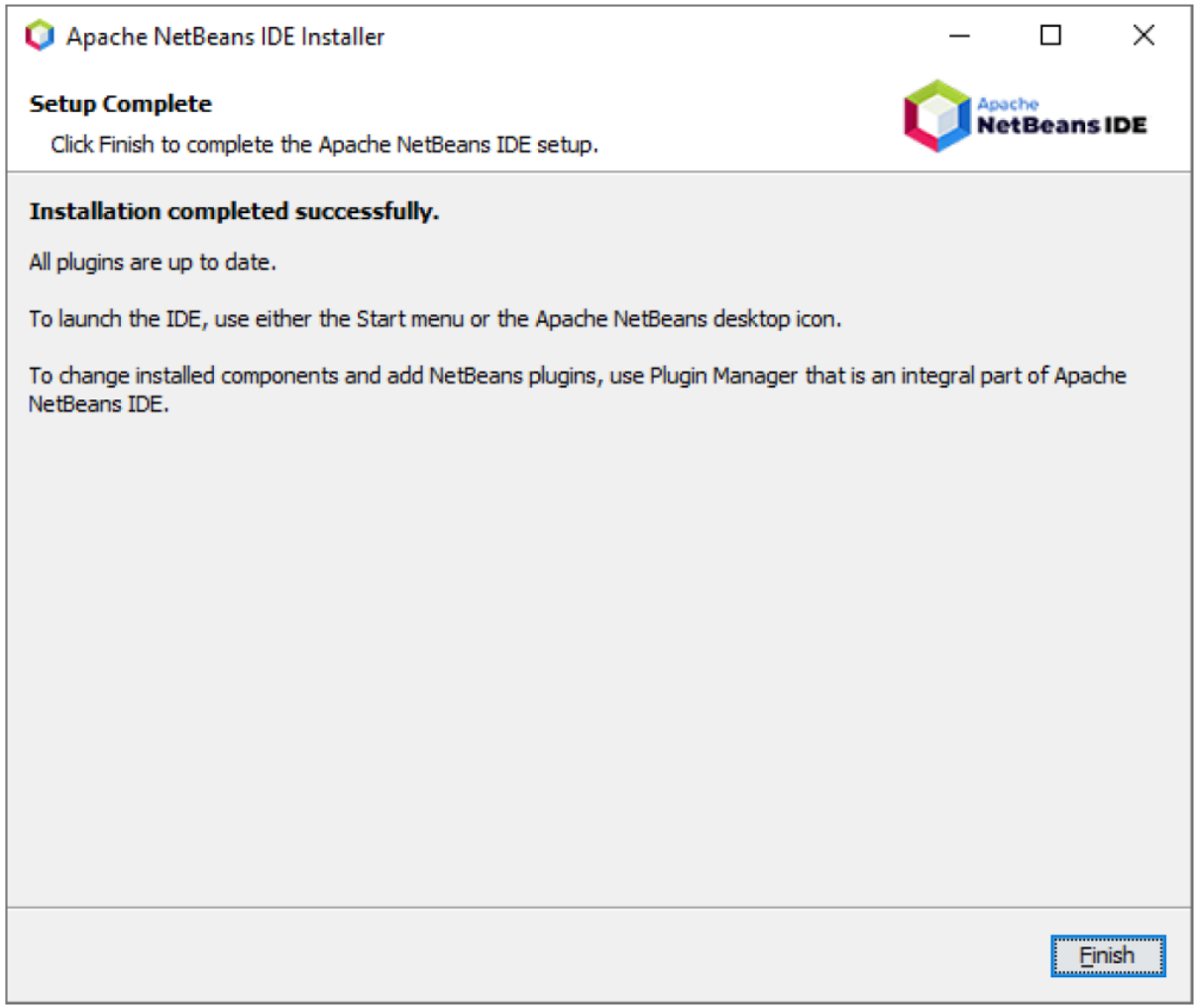
Installing NetBeans



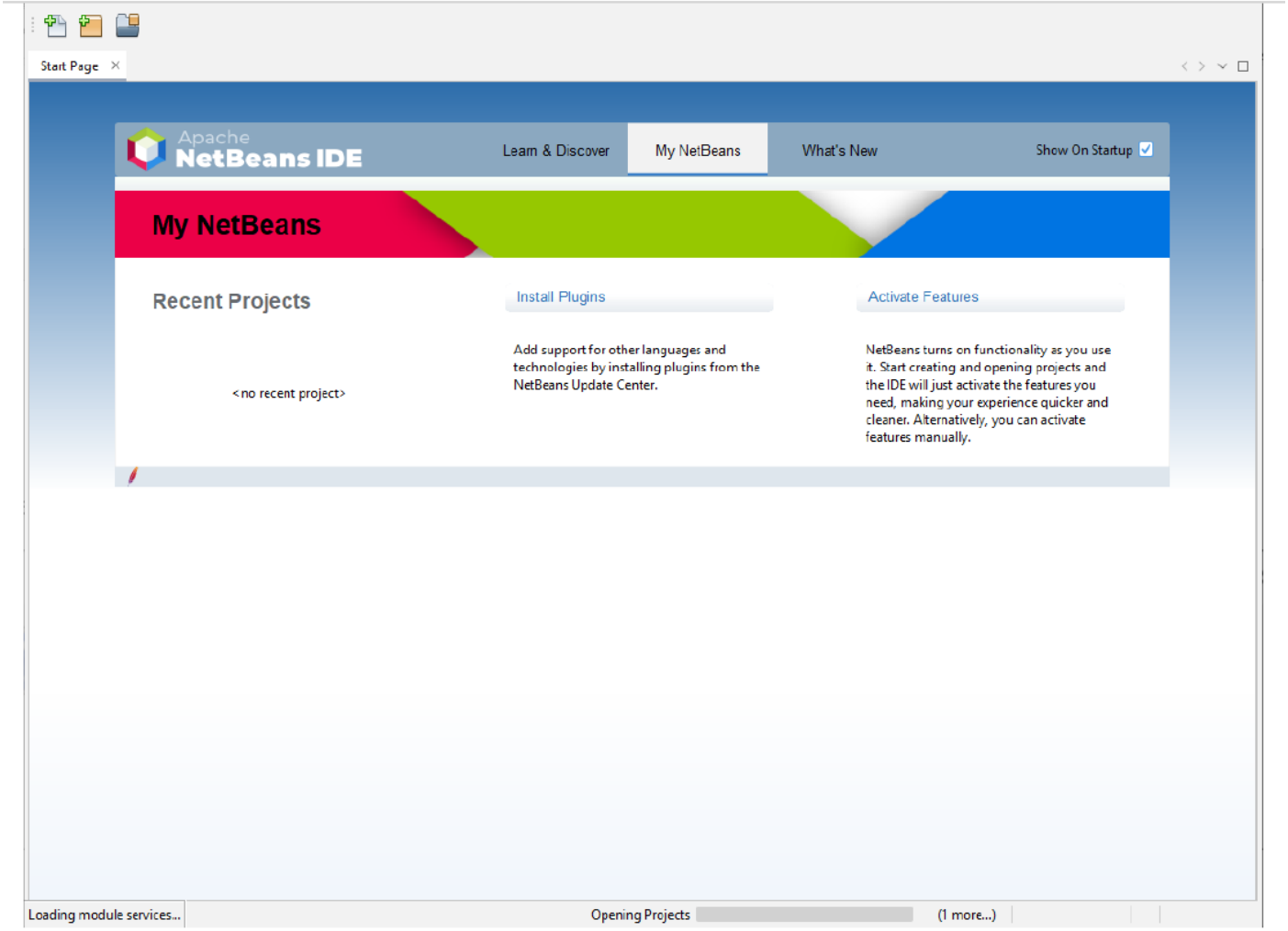
Installing NetBeans



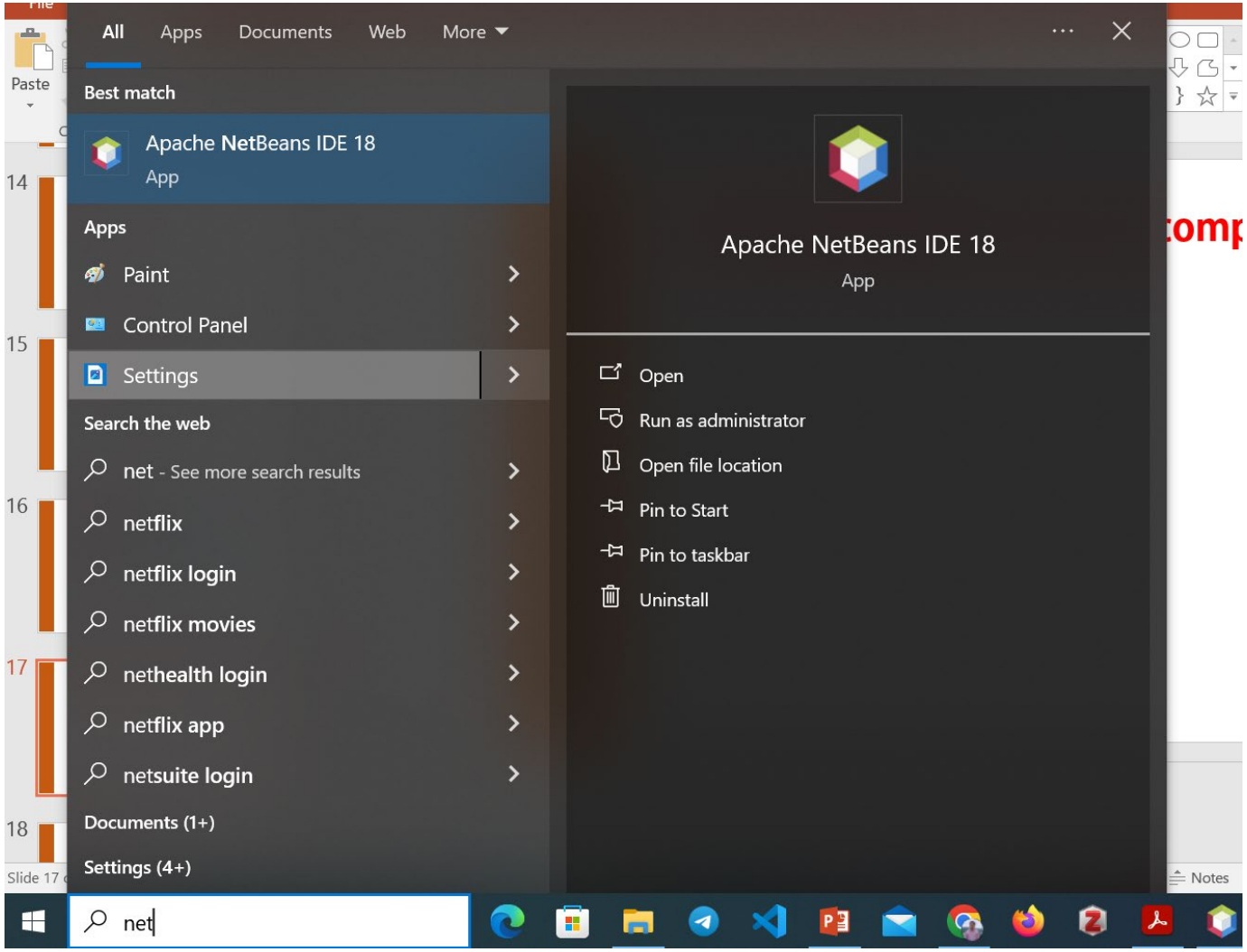
Installing NetBeans



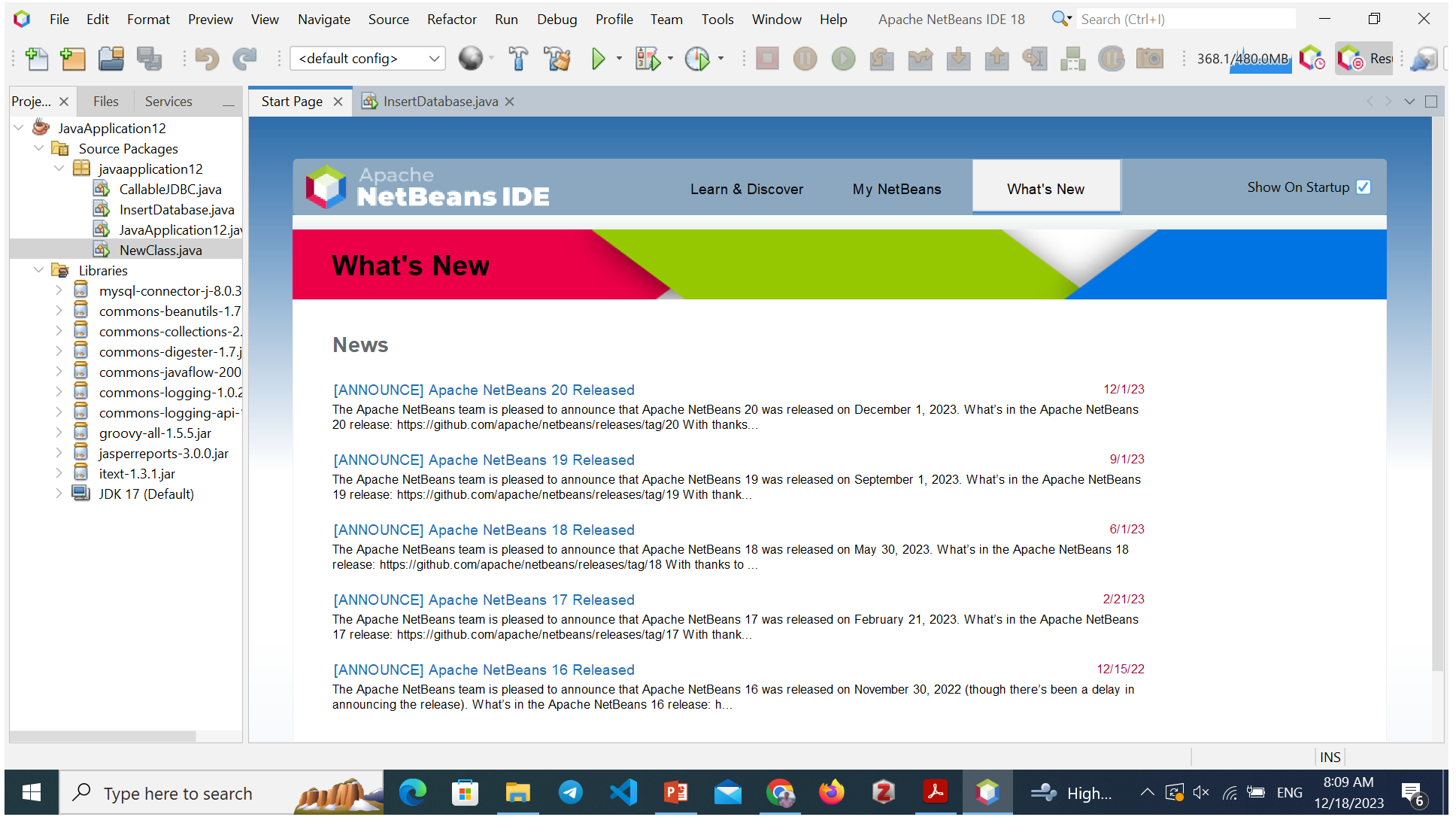
Installing NetBeans



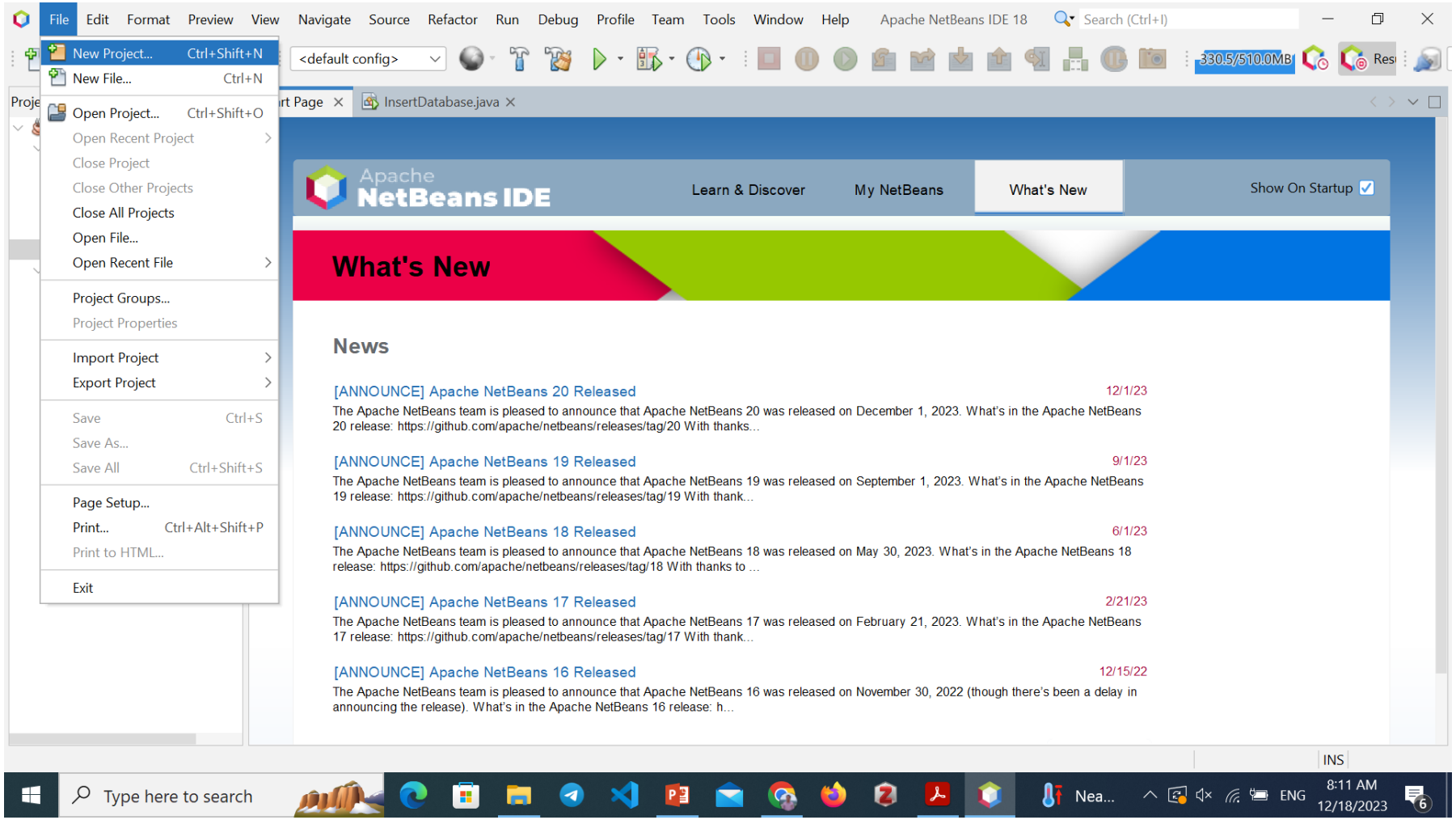
Starting NetBeans on your computer



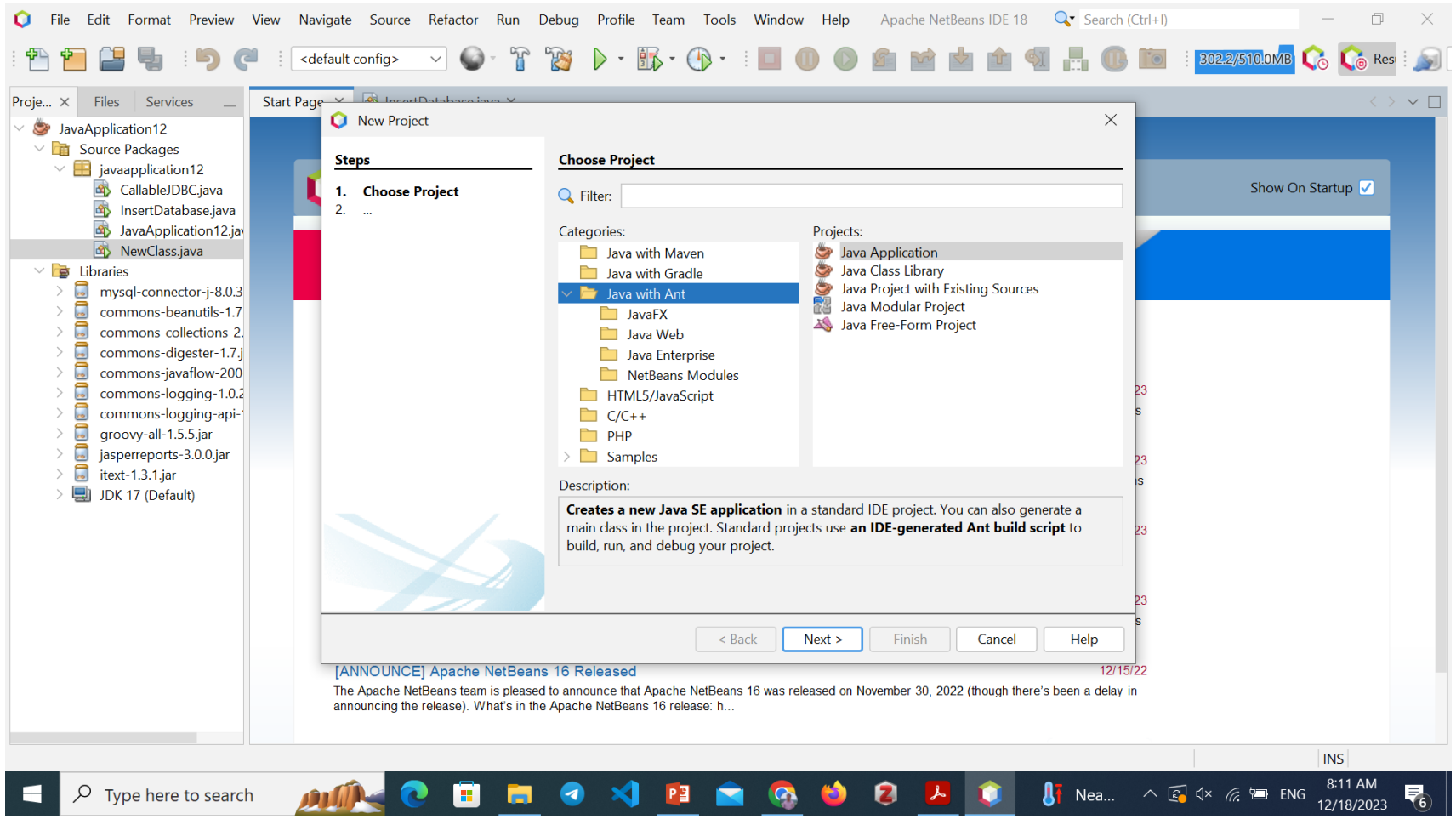
Installing NetBeans



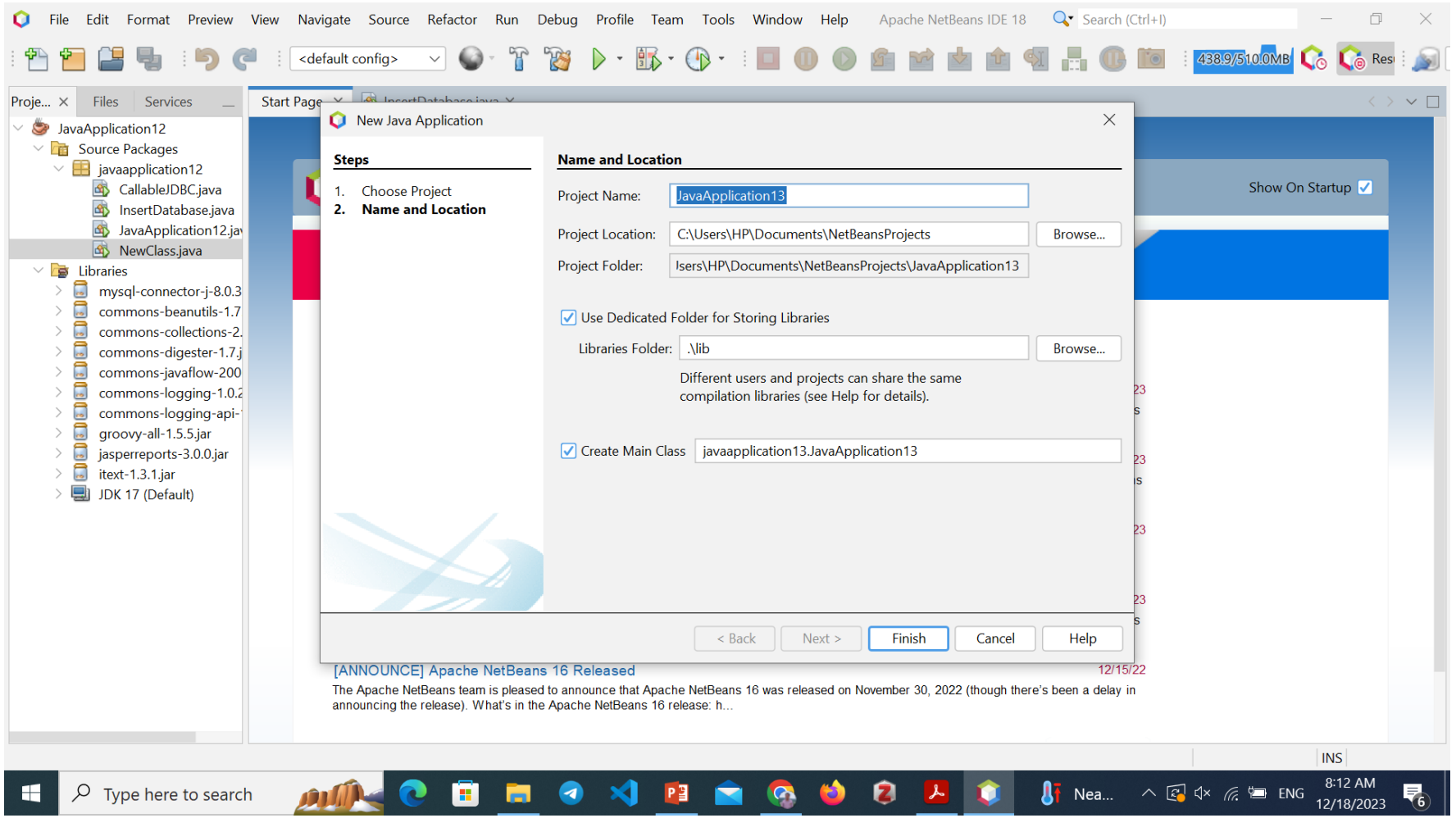
New Project in NetBeans



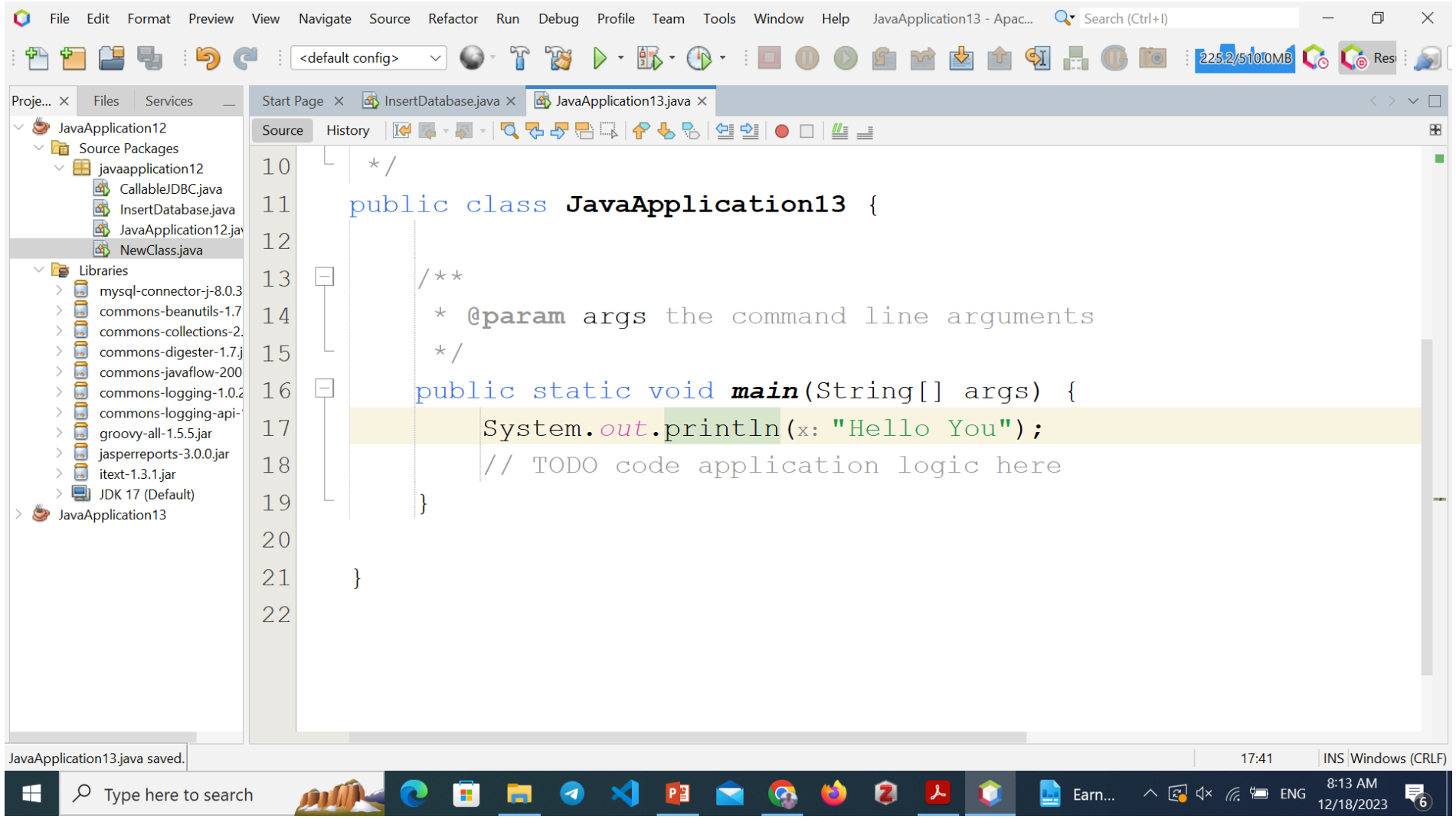
New Project in NetBeans



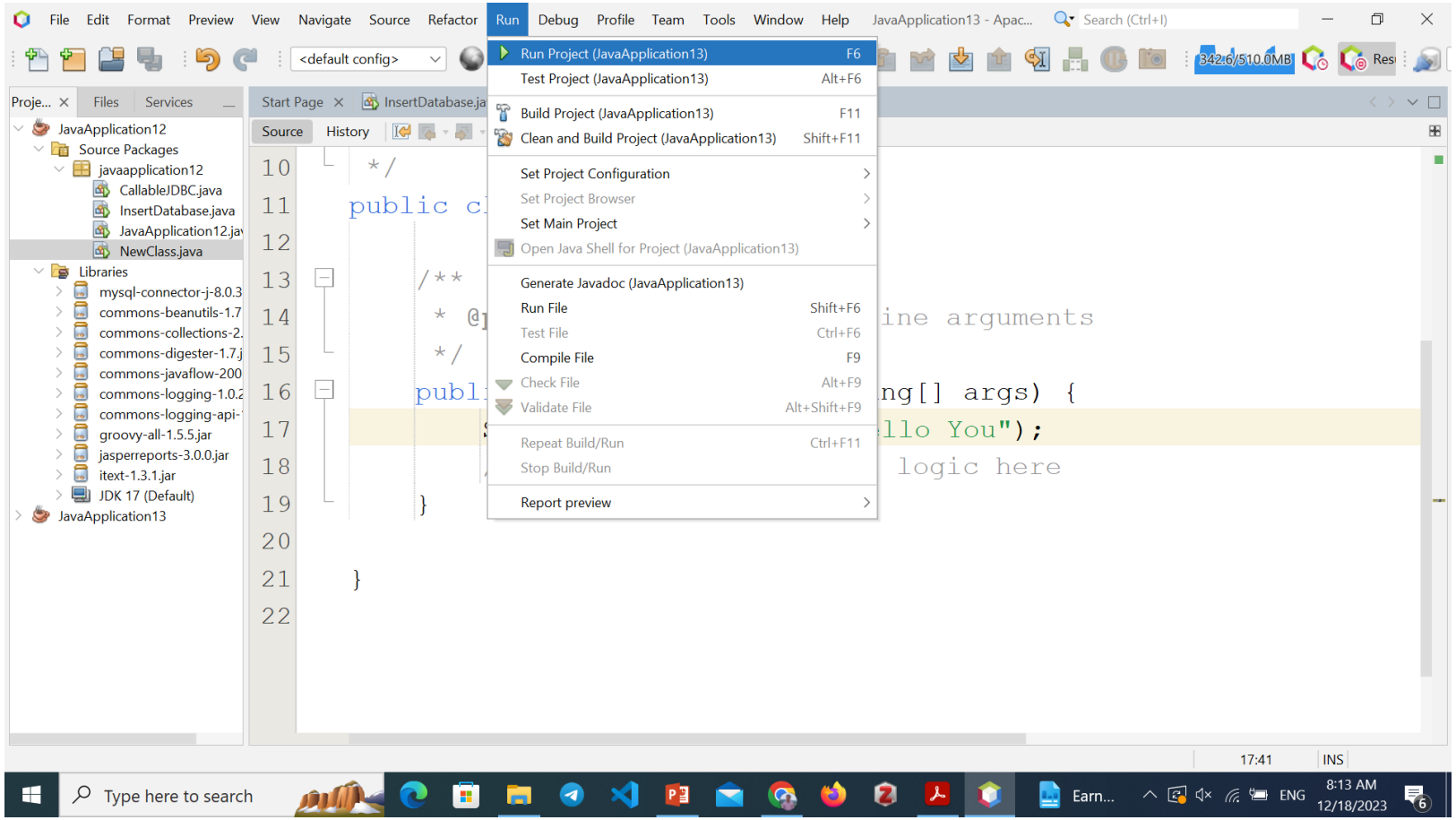
New Project in NetBeans



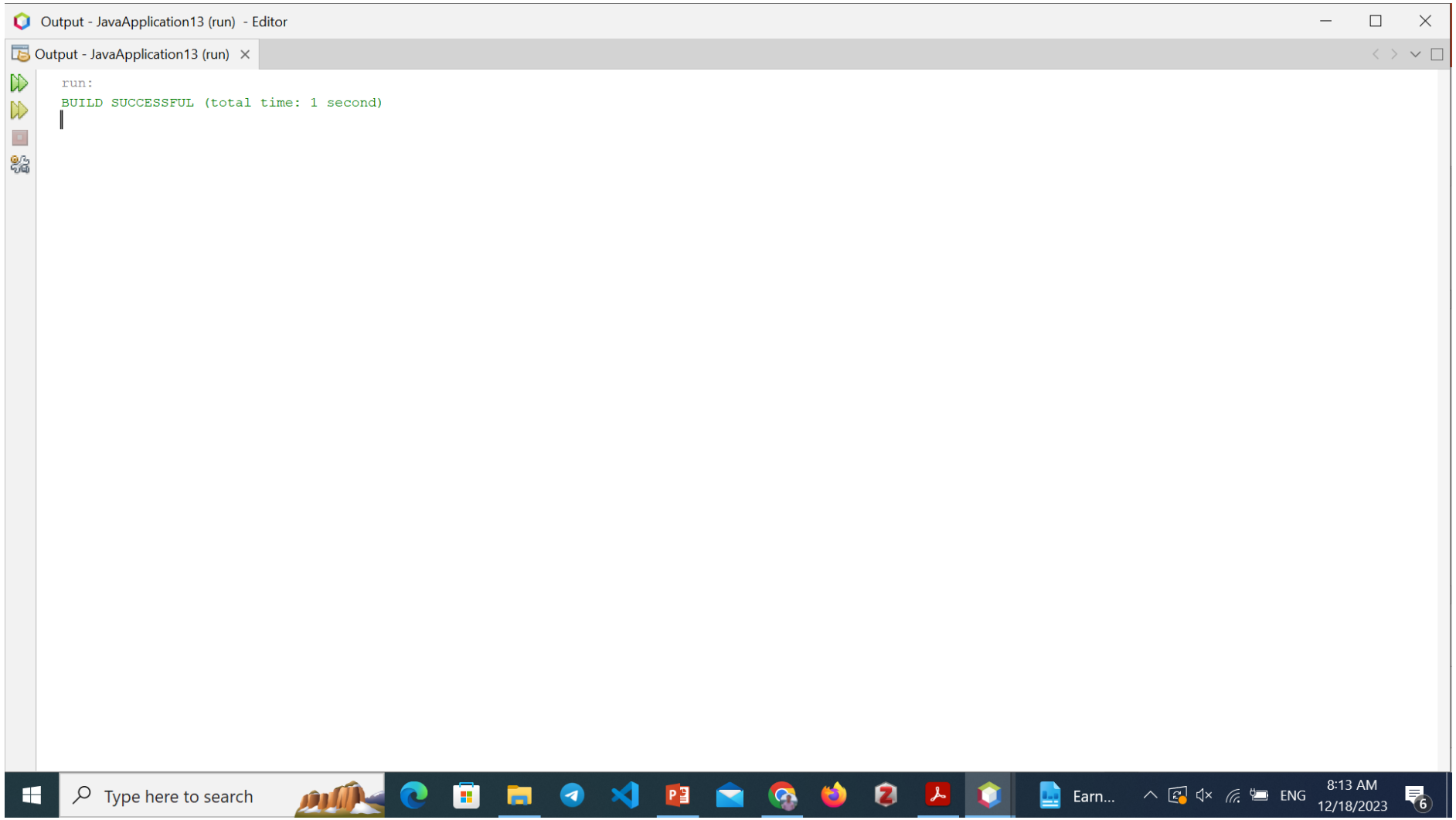
New Project in NetBeans



New Project in NetBeans



New Project in NetBeans



Introduction to Java Programming

- Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Introduction to Java Programming

Types of Java Applications

1. Standalone Application
2. Web Application
3. Enterprise Application
4. Mobile Application

Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

Web Application

An application that runs on the server side and creates a dynamic page is called a web application.

Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering.

In Java, EJB is used for creating enterprise applications.

Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Introduction to Java Programming

```
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Java Data Types

Data types are divided into two groups:

- Primitive data types -

includes byte, short, int, long, float, double, boolean and char

- Non-primitive data types - such

as String, Arrays and Classes (you will learn more about these in a later chapter)

Java 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

Java Variables

Variables are containers for storing data values.

In Java, there are different **types** of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false

Java Variables

- Declaring (Creating) Variables
- To create a variable, you must specify the type and assign it a value:

Syntax

```
type variableName = value;
```

Java Variables

```
public class Main
{
  public static void main(String[] args)
  {
    String name = "John";
    System.out.println(name);
  }
}
```

```
John
```

Java Variables

```
int myNum = 5;

float myFloatNum = 5.99f;

char myLetter = 'D';

boolean myBool = true;

String myText = "Hello";
```

Java Operators

Operators are used to perform operations on variables and values.

- Java divides the operators into the following groups:
- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Java Operators

- Arithmetic operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y

Java Operators

- Arithmetic operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Operators

- Assignment operators

Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3

Java Operators

- Assignment operators

Assignment operators are used to assign values to variables.

Operator	Example	Same As
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Operators

- Assignment operators

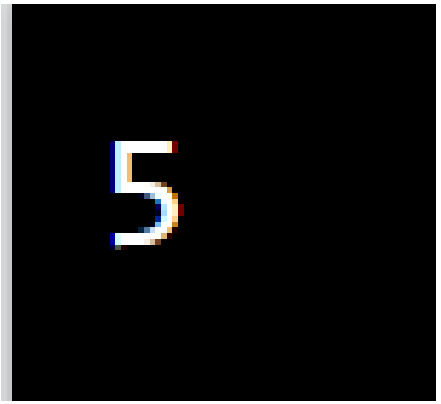
```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x |= 3;
    System.out.println(x);
  }
}
```

```
7
```

Java Operators

- Assignment operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x);
  }
}
```



Java Operators

- Assignment operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x += 3;
    System.out.println(x);
  }
}
```

```
8
```

Java Operators

- Assignment operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x >>= 3;
    System.out.println(x);
  }
}
```

```
0
```

Java Operators

- Assignment operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x <<= 3;
    System.out.println(x);
  }
}
```

```
40
```

Java Operators

- Comparison operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either true or false. These values are known as *Boolean values*, and you will learn more about them in the Booleans and If..Else chapter.

In the following example, we use the **greater than** operator (>) to find out if 5 is greater than 3:

Java Operators

- Comparison operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Java Operators

- Comparison operators

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5;
    int y = 3;
    System.out.println(x > y); // returns true, because 5
is higher than 3
  }
}
```

```
true
```

Java Operators

- Comparison operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x == y); // returns false because 5
is not equal to 3
  }
}
```

```
false
```

Java Operators

- Comparison operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x >= y); // returns true because 5
is greater, or equal, to 3
  }
}
```

```
true
```

Java Operators

- Comparison operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x != y); // returns true because 5
is not equal to 3
  }
}
```

```
true
```

Java Operators

- Logical operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

Java Operators

- Logical operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Java Operators

- Bitwise operators

```
true
```

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x > 3 && x < 10); // returns true
because 5 is greater than 3 AND 5 is less than 10
  }
}
```

Java Operators

- Bitwise operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(!(x > 3 && x < 10)); // returns
false because ! (not) is used to reverse the result
  }
}
```

```
false
```

Java Operators

- Bitwise operators

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(!(x > 3 && x < 10)); // returns
false because ! (not) is used to reverse the result
  }
}
```

```
false
```

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Single-line comments start with two forward slashes (//).

```
public class Main {
  public static void main(String[] args) {
    // This is a comment
    System.out.println("Hello World");
  }
}
```

```
Hello World
```

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Multi-line comments start with /* and ends with */.

```
public class Main {
  public static void main(String[] args) {
    /* The code below will print the words
Hello World
    to the screen, and it is amazing */
    System.out.println("Hello World");
  }
}
```

```
Hello World
```

Java Control Statements

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java if...else Statement

Condition is true

```
int number = 10;

if (number > 0) {
  // code
}

// code after if
```

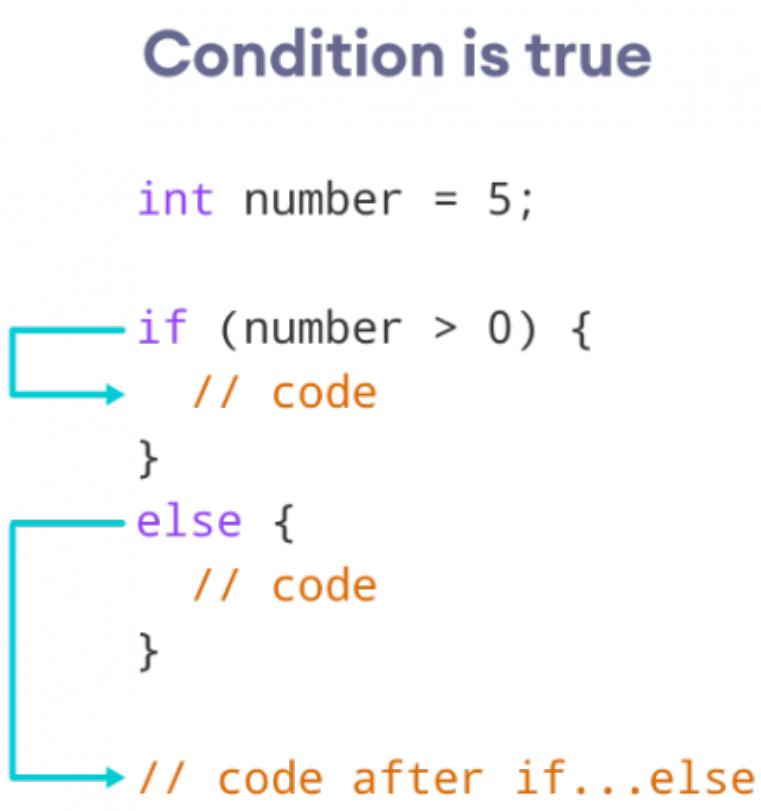
Condition is false

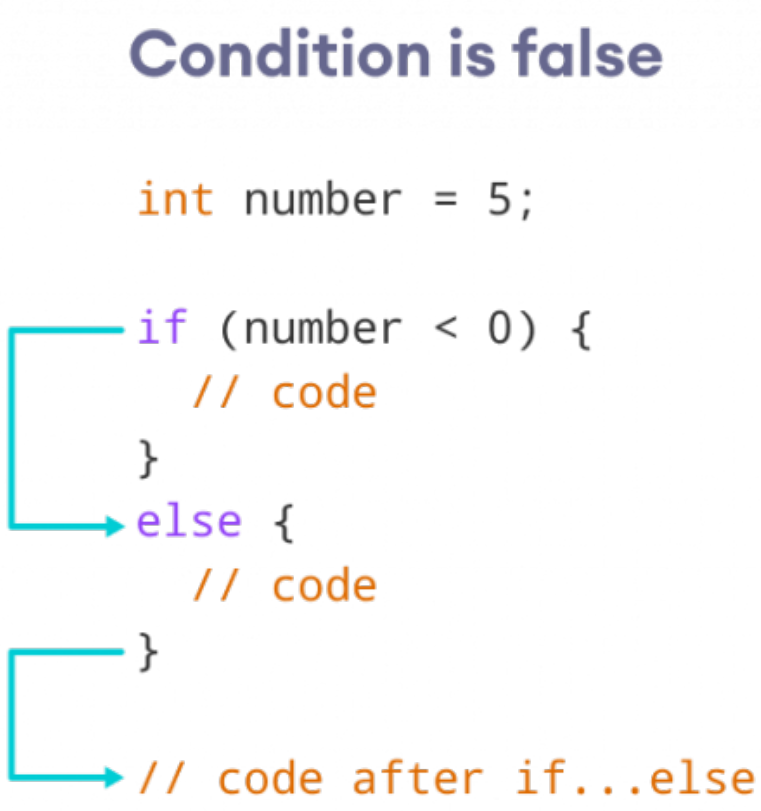
```
int number = 10;

if (number < 0) {
  // code
}

// code after if
```

Java if...else Statement





Java if...else Statement

1st Condition is true

```
int number = 2;
if (number > 0) {
  // code
}
else if (number == 0){
  // code
}
else {
  //code
}

//code after if
```

2nd Condition is true

```
int number = 0;
if (number > 0) {
  // code
}
else if (number == 0){
  // code
}
else {
  //code
}

//code after if
```

All Conditions are false

```
int number = -2;
if (number > 0) {
  // code
}
else if (number == 0){
  // code
}
else {
  //code
}

//code after if
```

Java switch Statement

*Example.java

```
1  package Examples;
2  public class Example{
3      public static void main(String[] args)
4      {
5          int num=23;
6            switch(num){
7            case 1: System.out.println("15");
8            break;
9            case 2: System.out.println("25");
10           break;
11           case 3: System.out.println("35");
12           break;
13           default:System.out.println("Not Exist");
14
15       }
16     }
17 }
```

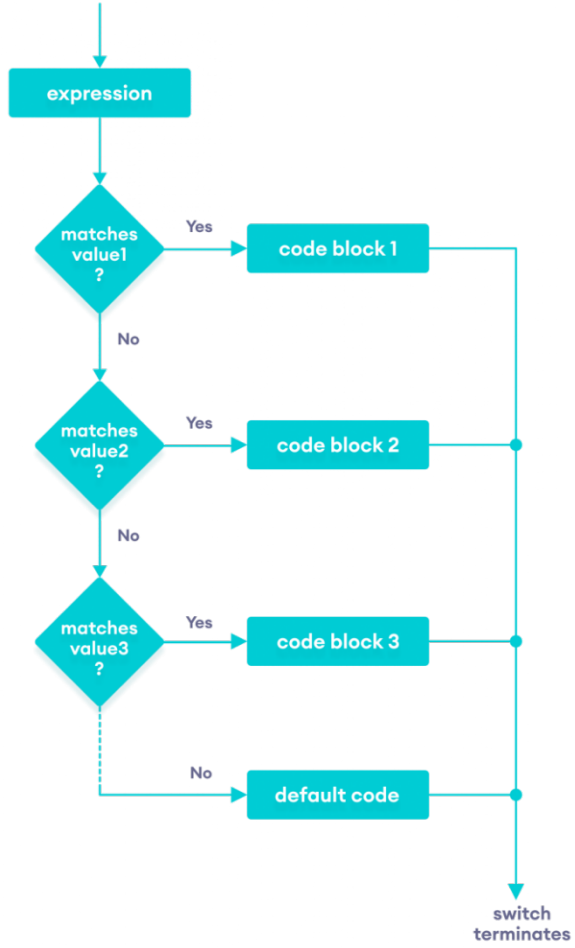
Code

Console

<terminated> Example [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.

```
Not Exist
```

Output



Java switch Statement

```
public class Main {
  public static void main(String[] args) {
    if (20 > 18) {
      System.out.println("20 is greater than
18"); // obviously
    }
  }
}
```

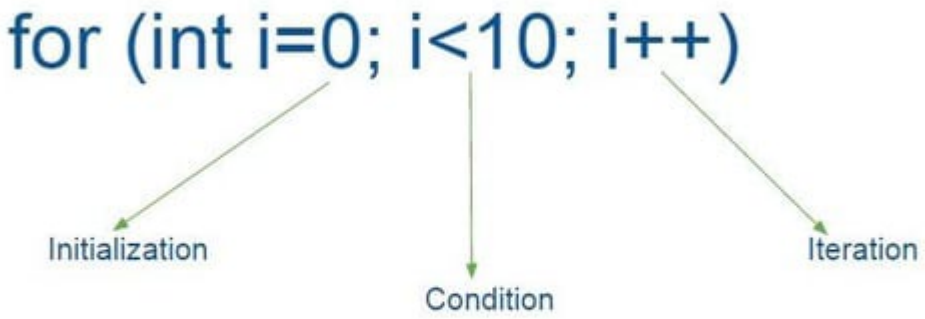
```
20 is greater than 18
```

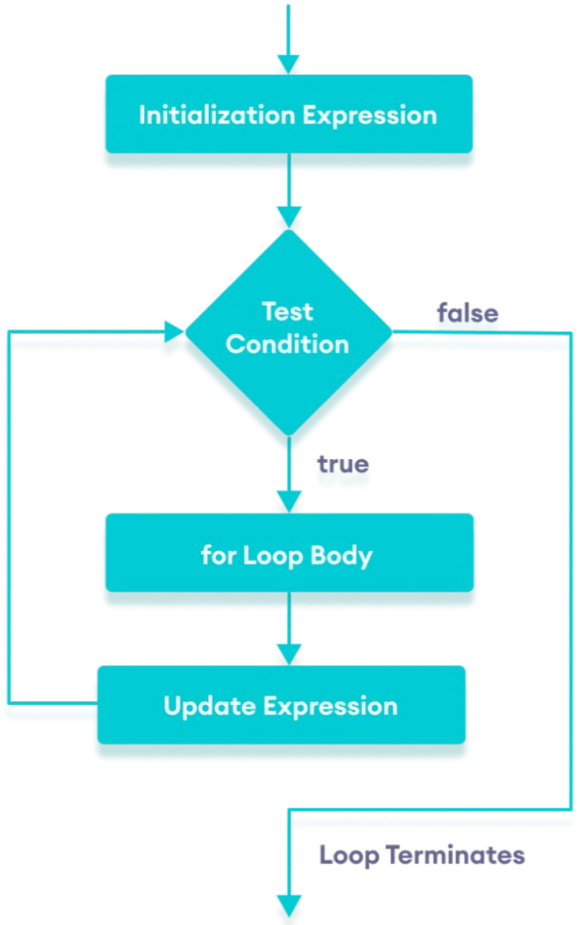
Java switch Statement

```
public class Main {
  public static void main(String[] args) {
    int day = 4;
    switch (day) {
      case 1:
        System.out.println("Monday");
        break;
      case 2:
      case 3:
      case 4:
        System.out.println("Thursday");
      case 7:
        System.out.println("Sunday");
        break;
    }
  }
}
```

```
Thursday
Sunday
```

Java for Loop





Java for Loop

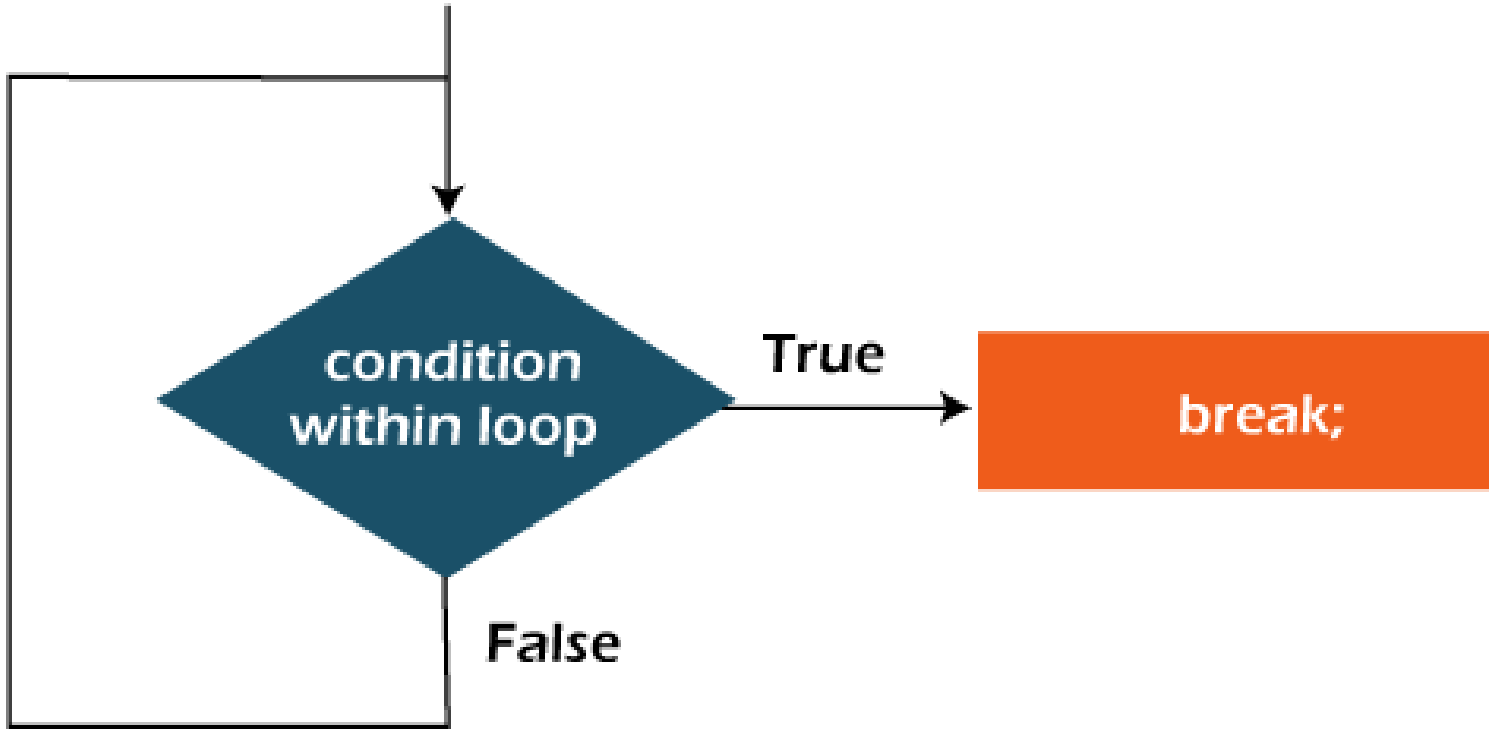
```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(i);
    }
  }
}
```

```
0
1
2
3
4
```

Java 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.
- The Java *break* statement 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.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Java Break Statement



Flowchart of break statement

Java Break Statement

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

```
0
1
2
3
```

Java Access Modifier

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter
protected	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter

Java Access Modifier

Main.java | Second.java

```
public class Main {
  public String fname = "John";
  public String lname = "Doe";
  public String email = "john@doe.com";
  public int age = 24;
}
```

Java Access Modifier

Main.java | Second.java

```
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " +
myObj.fname + " " + myObj.lname);
    System.out.println("Email: " +
myObj.email);
    System.out.println("Age: " + myObj.age);
  }
}
```

```
Name: John Doe
Email: john@doe.com
Age: 24
```

Java Access Modifier

```
public class Main {
  private String fname = "John";
  private String lname = "Doe";
  private String email = "john@doe.com";
  private int age = 24;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " +
myObj.lname);
    System.out.println("Email: " + myObj.email);
    System.out.println("Age: " + myObj.age);
  }
}
```

```
Name: John Doe
Email: john@doe.com
Age: 24
```

Java Access Modifier

```
class Person {
  protected String fname = "John";
  protected String lname = "Doe";
  protected String email = "john@doe.com";
  protected int age = 24;
}

class Student extends Person {
  private int graduationYear = 2018;
  public static void main(String[] args) {
    Student myObj = new Student();
    System.out.println("Name: " + myObj.fname + " " +
myObj.lname);
    System.out.println("Email: " + myObj.email);
    System.out.println("Age: " + myObj.age);
    System.out.println("Graduation Year: " +
myObj.graduationYear);
  }
}
```

```
Name: John Doe
Email: john@doe.com
Age: 24
Graduation Year: 2018
```

Java Methods

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as **functions**.
- Why use methods? To reuse code: define the code once, and use it many times.

Java Methods

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Java Methods

Create a method inside Main:

```
public class Main {
  static void myMethod() {
    // code to be executed
  }
}
```

Java Methods

Example Explained

- myMethod() is the name of the method
- static means that the method belongs to the Main class and not an object of the Main class. You will learn more about objects and how to access methods through objects later in this tutorial.
- void means that this method does not have a return value. You will learn more about return values later in this chapter

Java Methods

```
public class Main {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

```
I just got executed!
```

Java Methods

```
public class Main {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

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

```
I just got executed!
I just got executed!
I just got executed!
```

Java Method Parameters

- Information can be passed to methods as parameter. Parameters act as variables inside the method.
- Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Java Method Parameters

```
public class Main {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
```

```
Liam Refsnes
Jenny Refsnes
Anja Refsnes
```

Java Method Parameters

When a **parameter** is passed to the method, it is called an **argument**. So,

from the example above: fname is a **parameter**,

while Liam, Jenny and Anja are **arguments**.

- Multiple Parameters
- Note that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

Java Method Parameters

Return Values

The void keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

Java Method Parameters

```
public class Main {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
```

```
8
```

Java Scope

In Java, variables are only accessible inside the region they are created. This is called **scope**.

- **Method Scope**
- Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared:

Java Scope

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

    // Code here cannot use x

    int x = 100;

    // Code here can use x
    System.out.println(x);
  }
}
```

```
100
```

Java Scope

In Java, variables are only accessible inside the region they are created. This is called **scope**.

Block Scope

A block of code refers to all of the code between curly braces {}.

Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared:

Java Scope

```
public class Main {
  public static void main(String[] args)
  {
    {
      int x = 100;
      System.out.println(x);
    }
  }
}
```

```
100
```

Java Scope

A block of code may exist on its own or it can belong to an if, while or for statement. In the case of for statements, variables declared in the statement itself are also available inside the block's scope.