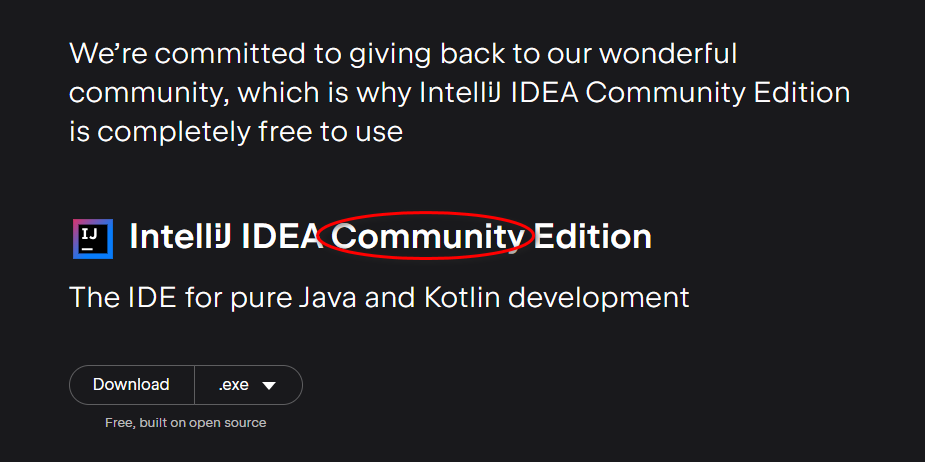
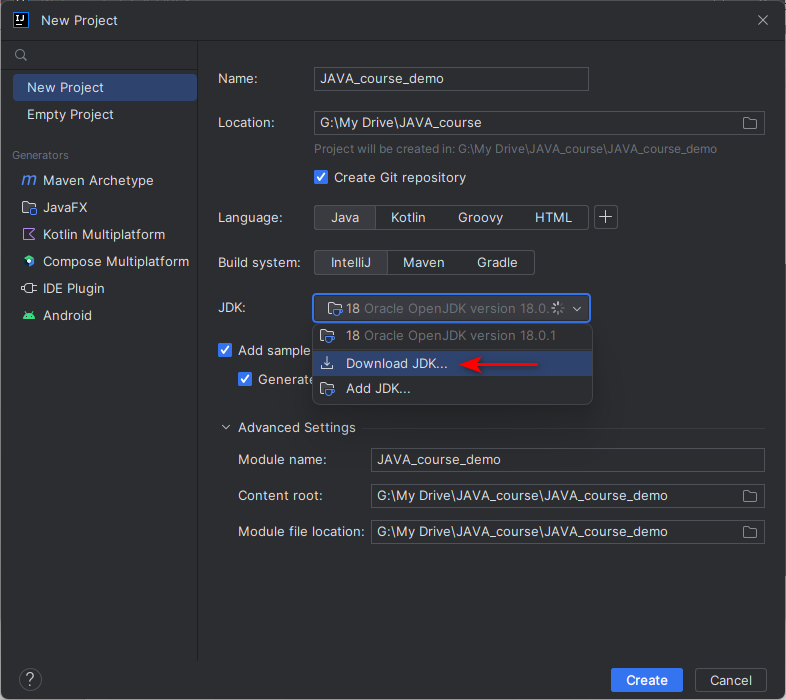
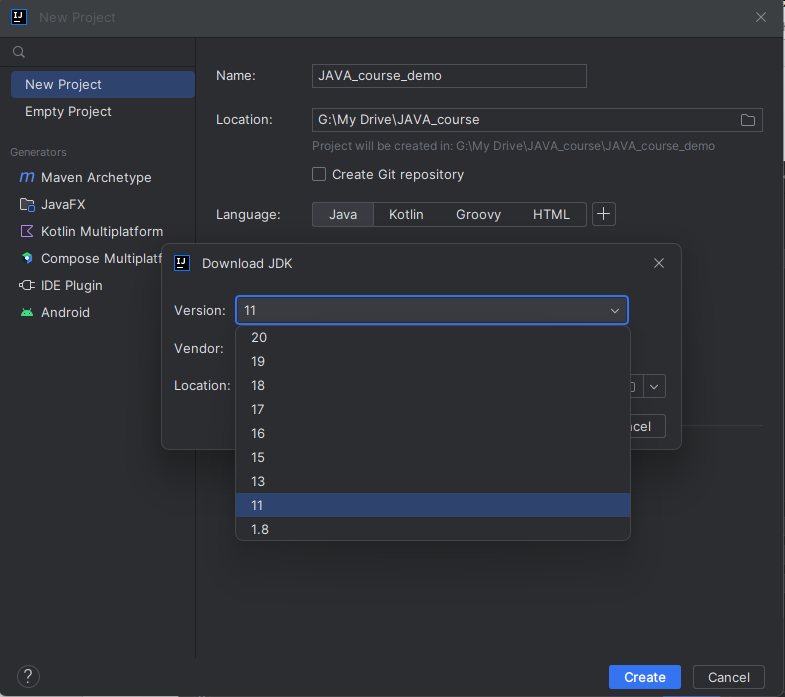
Step 1: Vào trang <https://www.jetbrains.com/idea/download/?section=windows>

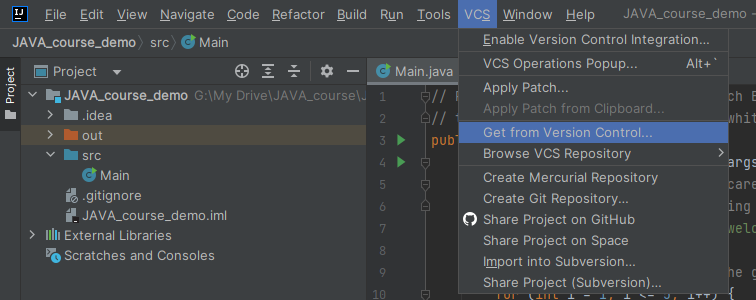
Download bản IntelliJ IDEA Community Edition thay vì bản thương mại IntelliJ IDEA Ultimate

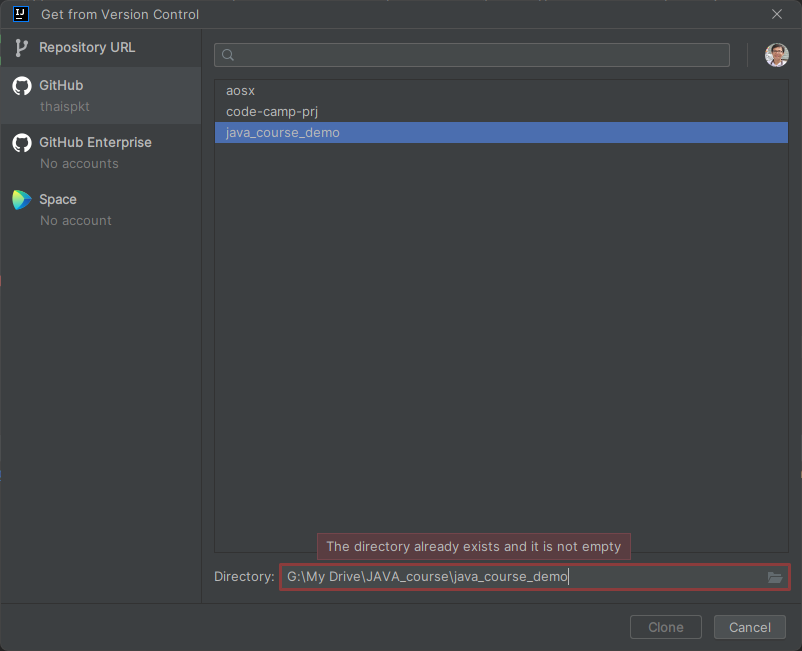


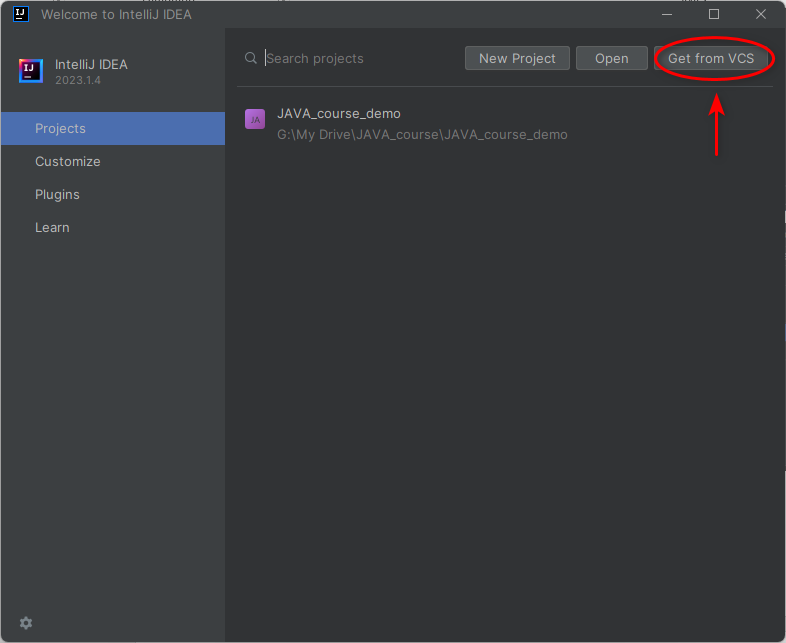


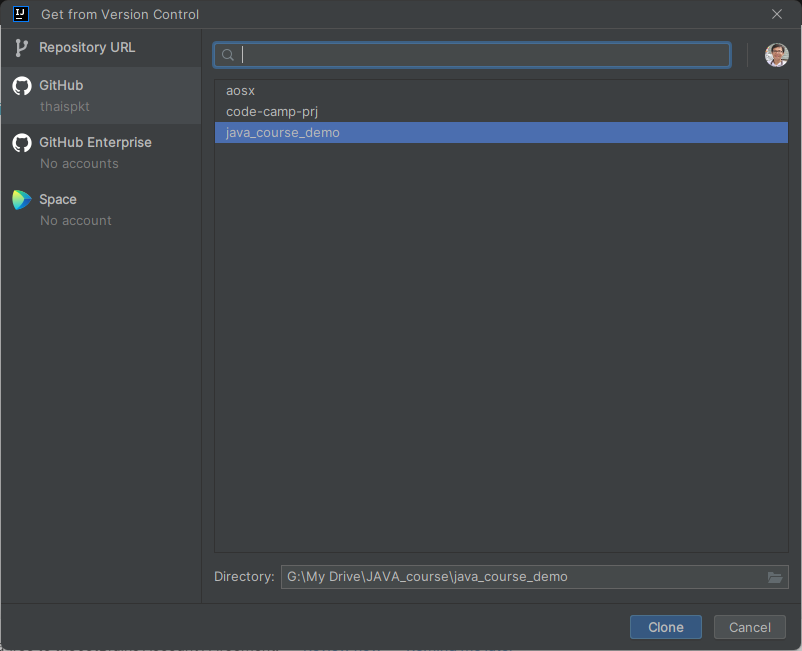
Chọn JDK version 11 vì đây là bản free

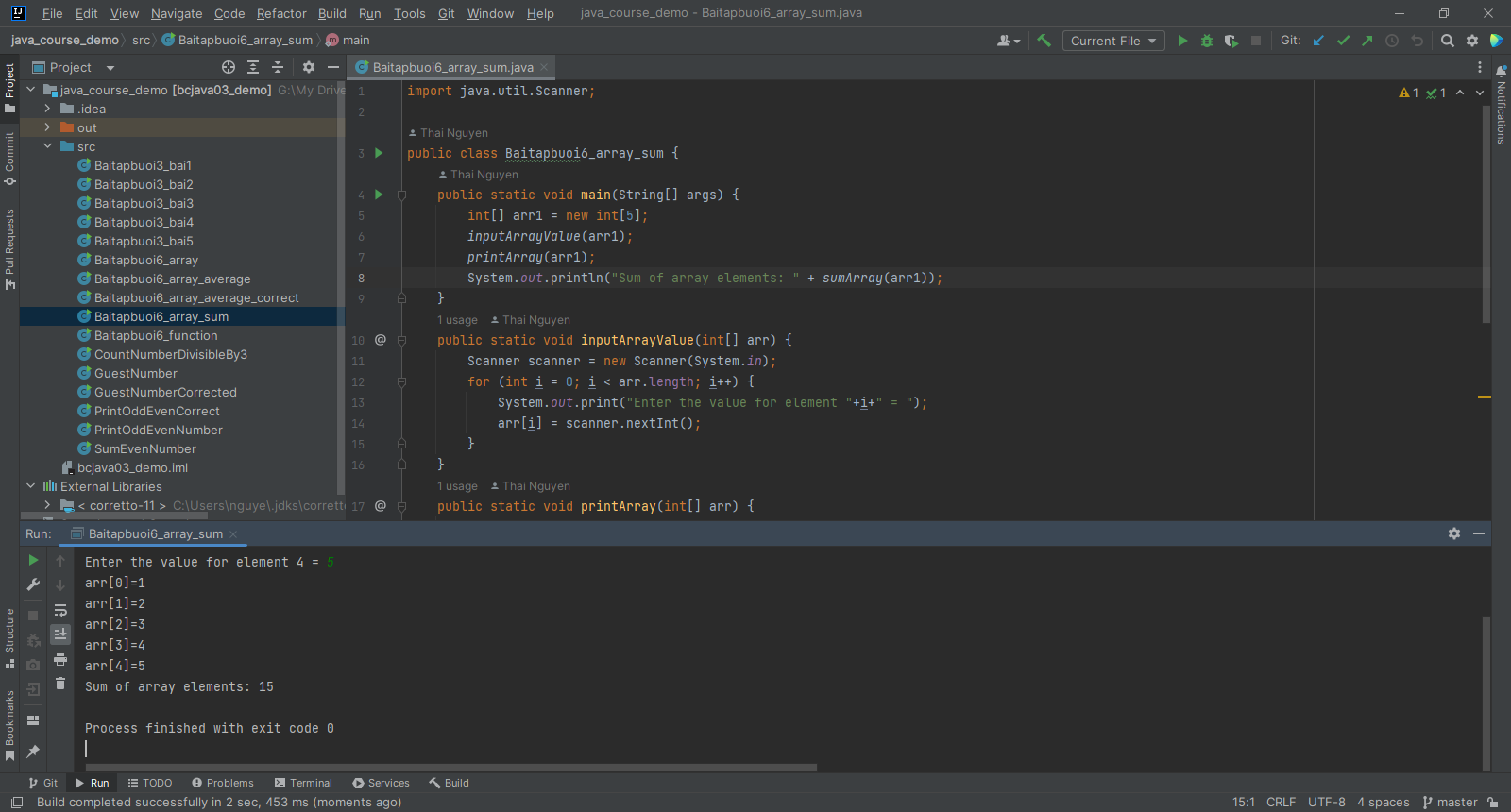


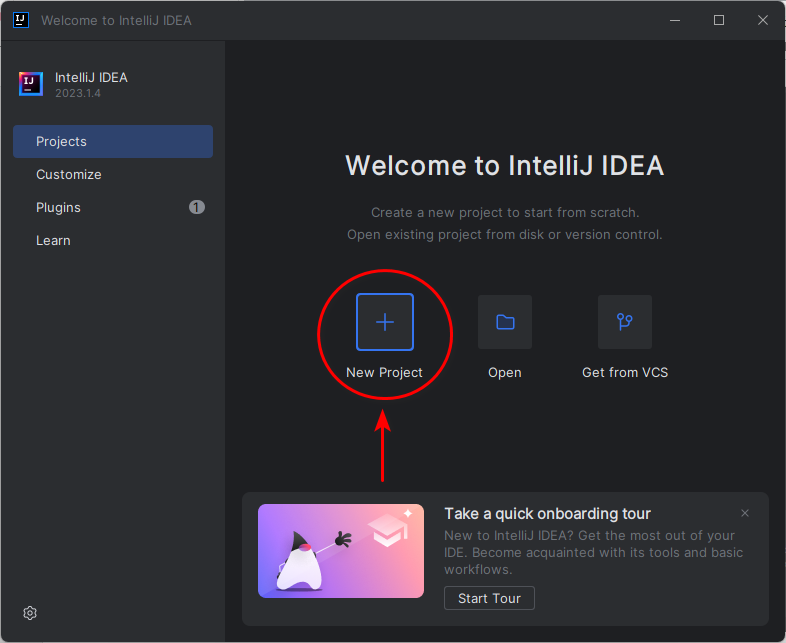












IntelliJ IDEA Community Edition and IntelliJ IDEA Ultimate are both commercial integrated development environments (IDEs) for Java development. However, there are some key differences between the two editions.

* IntelliJ IDEA Community Edition:
  + Free to use for personal and commercial projects.
  + Supports Java, Kotlin, and Scala development.
  + Includes a code editor, debugger, and a variety of other tools for Java development.
  + Does not include support for other languages, such as Python, JavaScript, and PHP.
  + Does not include support for enterprise features, such as code quality tools and version control integration.
* IntelliJ IDEA Ultimate:
  + Paid subscription required for use.
  + Supports Java, Kotlin, Scala, Python, JavaScript, and PHP development.
  + Includes a code editor, debugger, and a variety of other tools for all of the supported languages.
  + Includes support for enterprise features, such as code quality tools and version control integration.

Here is a table summarizing the key differences between IntelliJ IDEA Community Edition and IntelliJ IDEA Ultimate:

|  |  |  |
| --- | --- | --- |
| Feature | IntelliJ IDEA Community Edition | IntelliJ IDEA Ultimate |
| Price | Free | Paid subscription required |
| Languages supported | Java, Kotlin, and Scala | Java, Kotlin, Scala, Python, JavaScript, and PHP |
| Features | Code editor, debugger, and a variety of tools for Java development | Code editor, debugger, and a variety of tools for all of the supported languages |
| Enterprise features | No | Yes |

Ultimately, the best choice for you will depend on your needs and budget. If you are a Java developer who needs a free IDE, then IntelliJ IDEA Community Edition is a good option. If you need support for other languages or enterprise features, then IntelliJ IDEA Ultimate is a better choice.

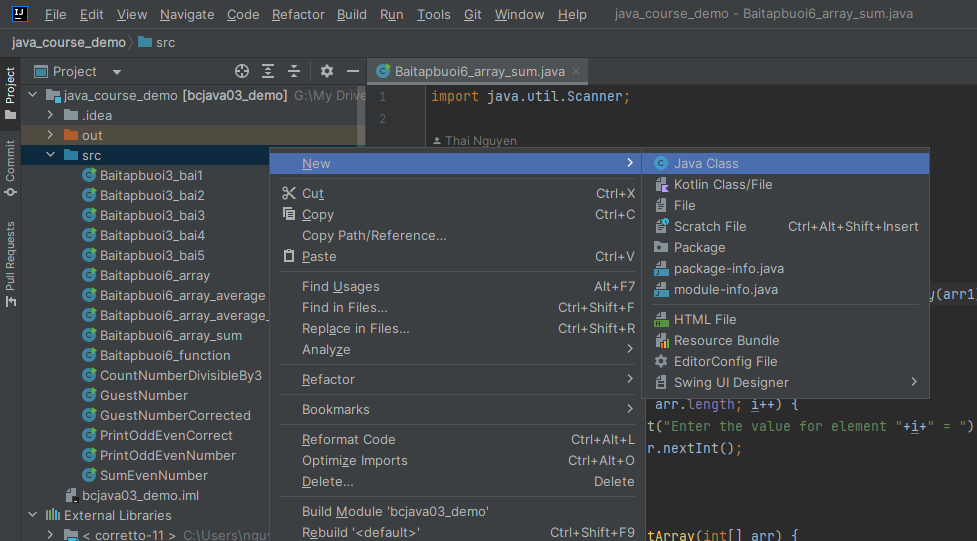
Difference between IntelliJ IDEA Community Edition and IntelliJ IDEA Ultimate

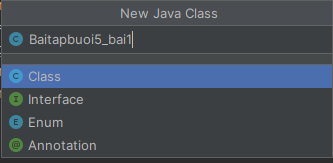
ChatGPT

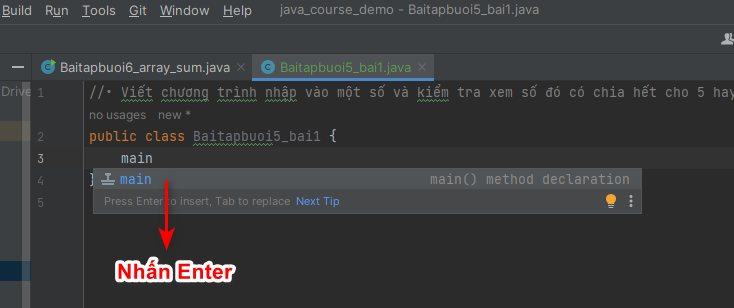
IntelliJ IDEA is an Integrated Development Environment (IDE) developed by JetBrains and is widely used for Java, Kotlin, and other programming languages. There are two main editions of IntelliJ IDEA: Community Edition and Ultimate Edition. Here are the key differences between the two:

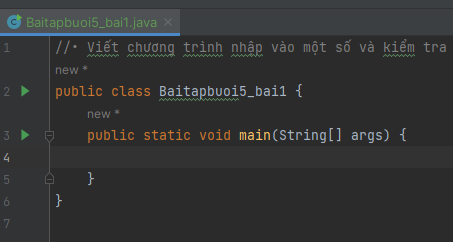
1. Licensing and Cost:
   * IntelliJ IDEA Community Edition: This edition is free and open-source, and it can be used without any cost.
   * IntelliJ IDEA Ultimate Edition: This edition is a commercial product and requires a paid license. It comes with additional features and tools not available in the Community Edition.
2. Features and Functionality:
   * IntelliJ IDEA Community Edition: It provides essential features for Java and Kotlin development, such as code completion, refactoring, version control integration, and support for various frameworks like Spring, JavaFX, and Android.
   * IntelliJ IDEA Ultimate Edition: It includes all the features available in the Community Edition and adds more advanced tools for web development, enterprise development, and support for additional languages like JavaScript, TypeScript, SQL, and more. It also includes support for application servers, database tools, and advanced code analysis.
3. Plugins and Extensions:
   * IntelliJ IDEA Community Edition: It supports a wide range of plugins available from the JetBrains plugin repository and the IntelliJ IDEA community.
   * IntelliJ IDEA Ultimate Edition: In addition to the plugins available for the Community Edition, it also supports plugins specifically designed for the Ultimate Edition, which may provide more specialized functionality for enterprise development, web frameworks, and other advanced use cases.
4. Target Users:
   * IntelliJ IDEA Community Edition: It is suitable for individual developers, hobbyists, and small projects that need basic Java and Kotlin development capabilities.
   * IntelliJ IDEA Ultimate Edition: It is targeted at professional developers and teams working on large-scale projects or projects that require advanced web, enterprise, and database development features.

In summary, the IntelliJ IDEA Community Edition is a free and open-source version that provides essential Java and Kotlin development features. On the other hand, IntelliJ IDEA Ultimate Edition is a commercial product with a paid license that offers more advanced features and tools for web development, enterprise development, and additional language support. The choice between the two editions depends on the specific needs and requirements of the project and the development team.



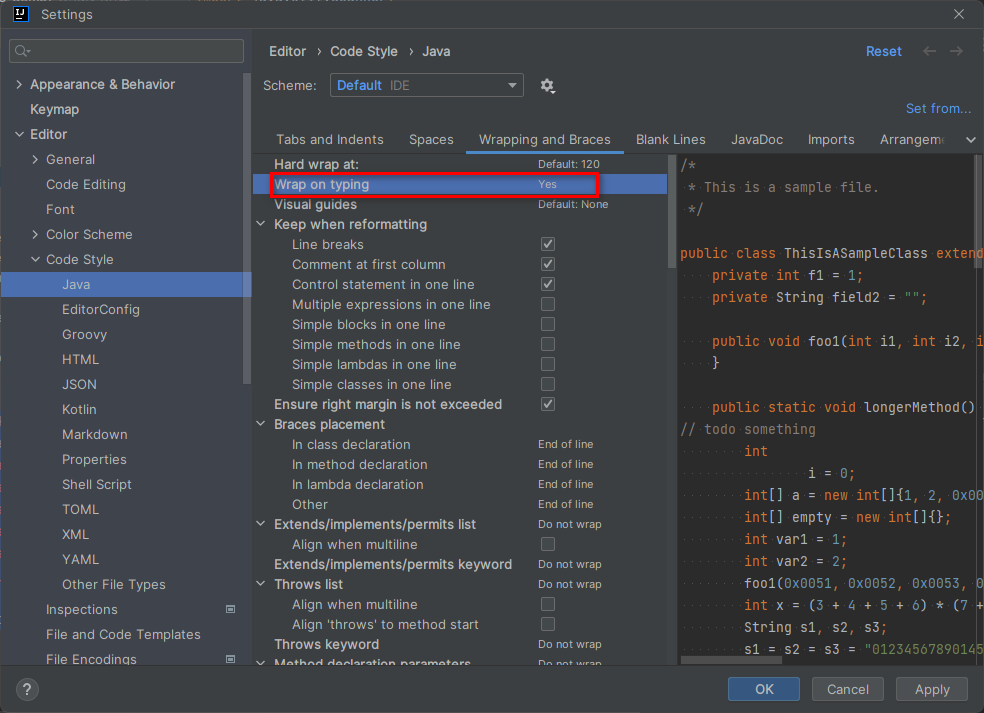




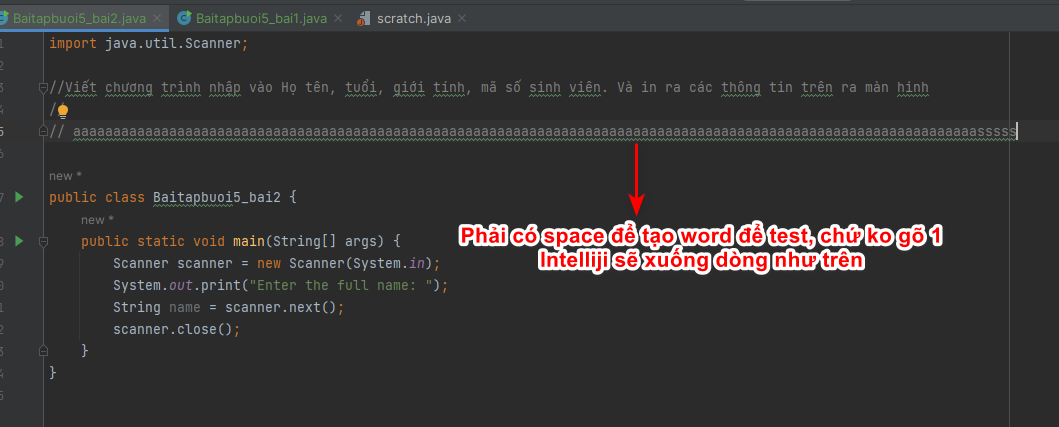


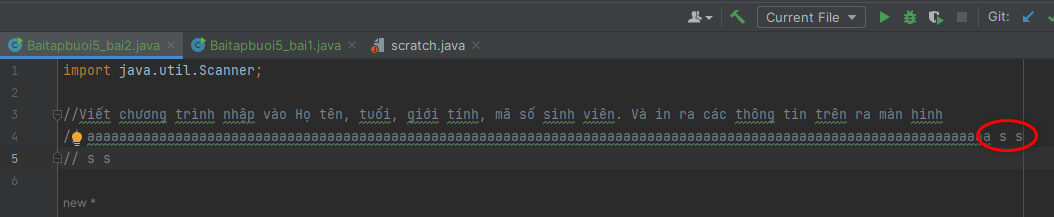
# Word wrap

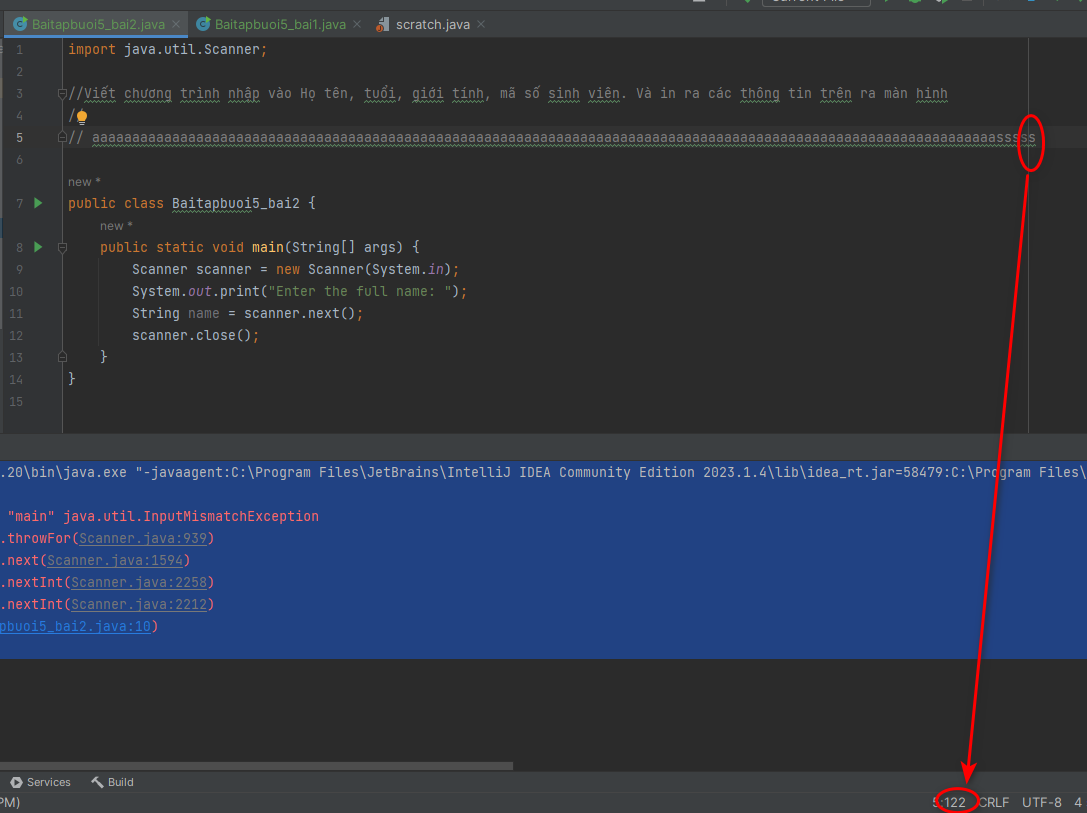
Cách config: Vào Settings/Editors/Code Style, set Yes cho Wrap on typing



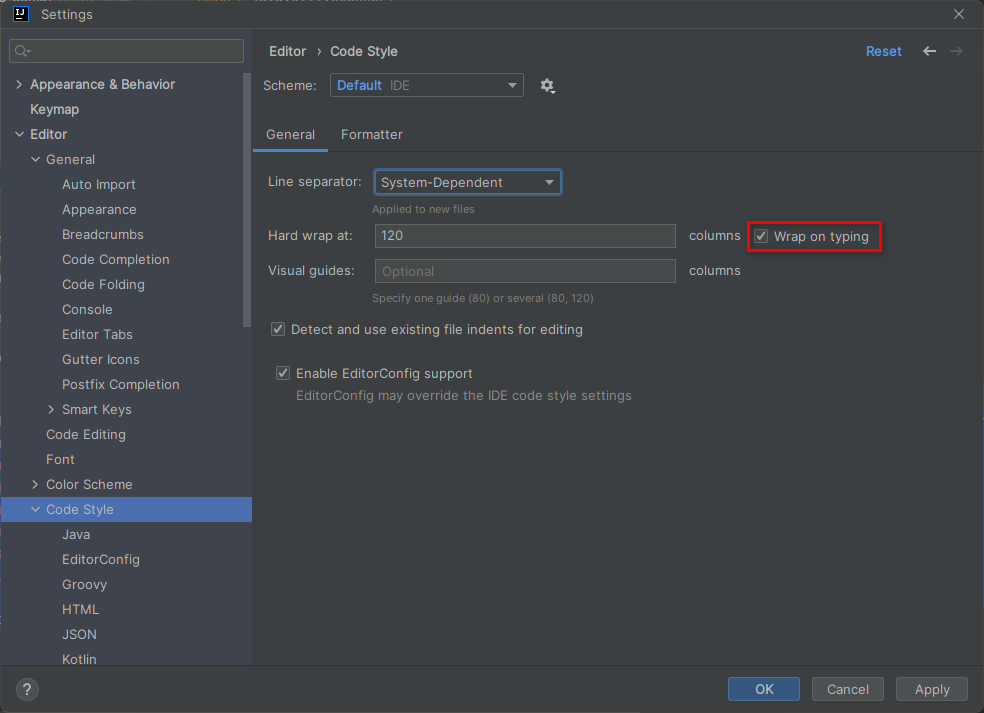
Khi test thì không gõ liên tục fffff mà phải gõ cách khoảng phải có space nếu không sẽ



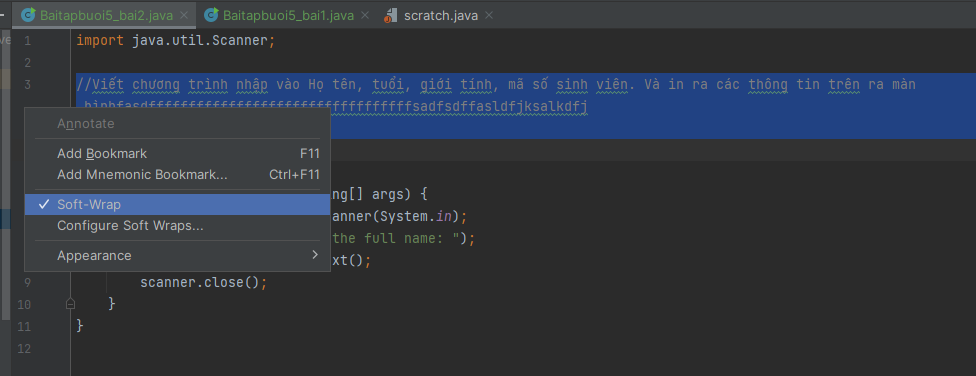


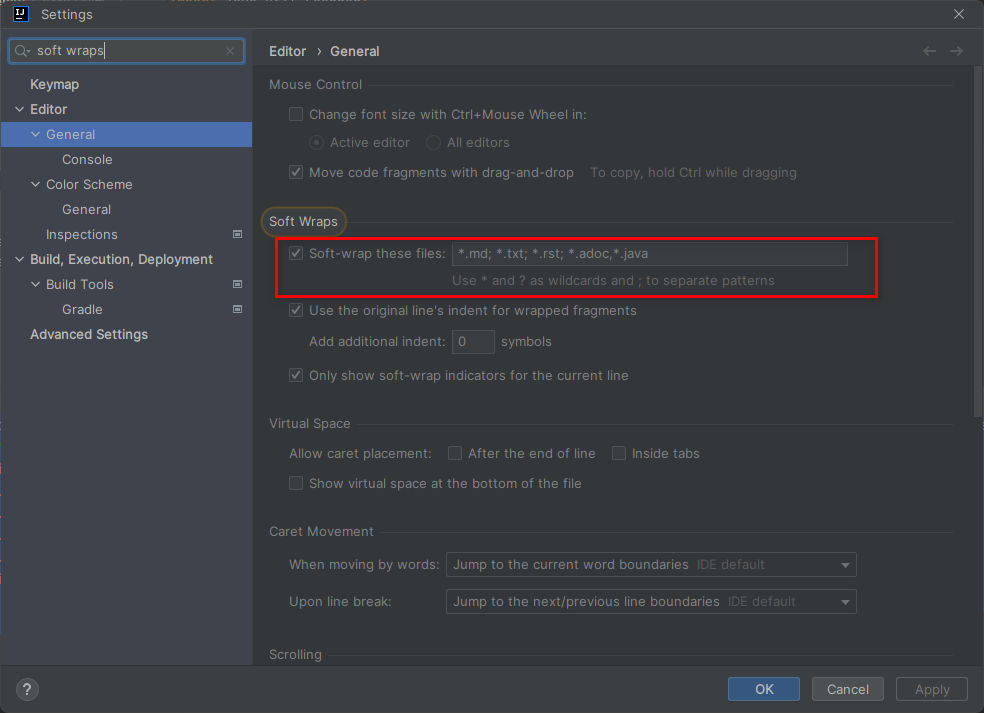


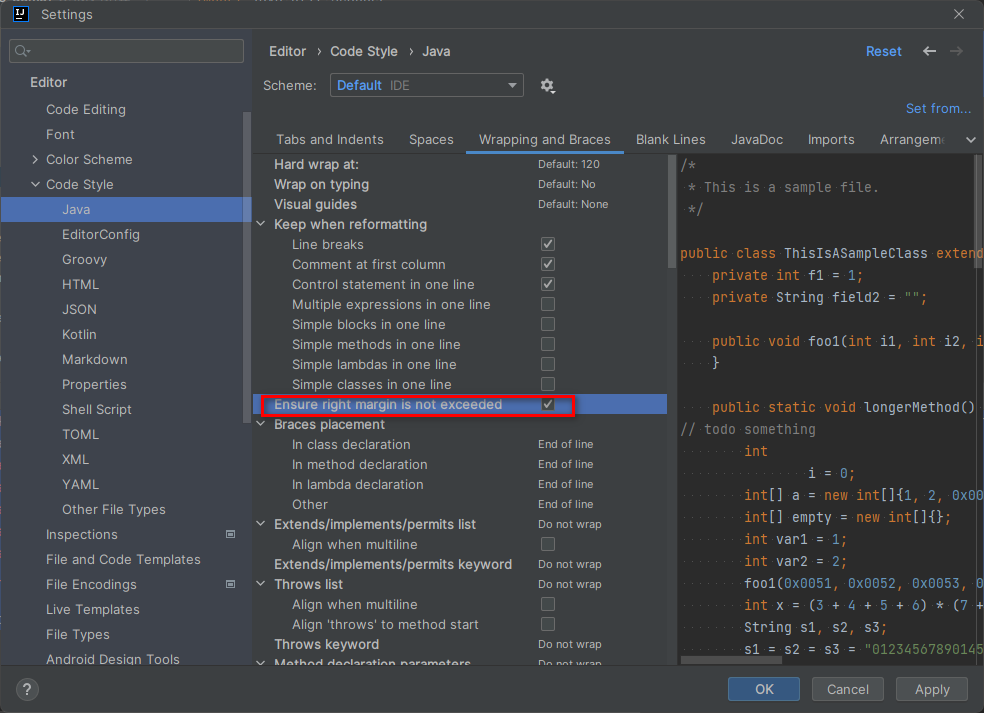
Những cách bên dưới không work



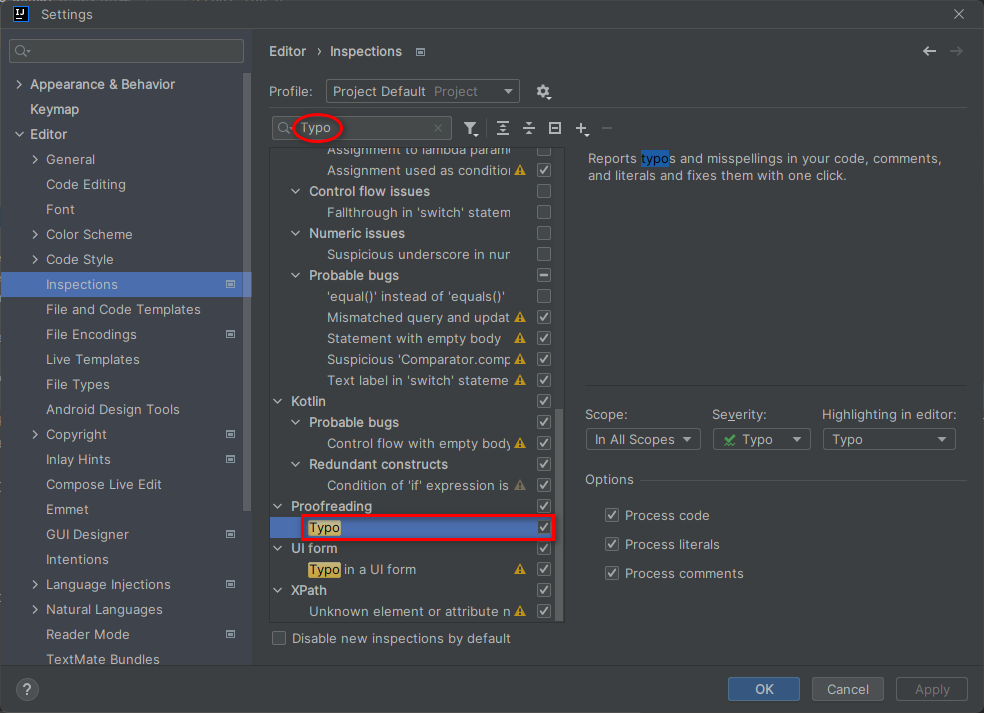
You can also right click the gutter (the vertical bar where line number is shown) and select "Use Soft wraps



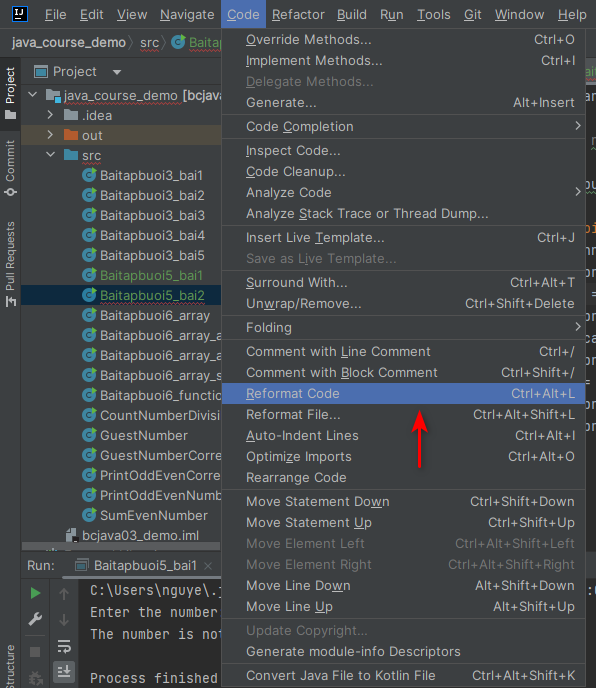


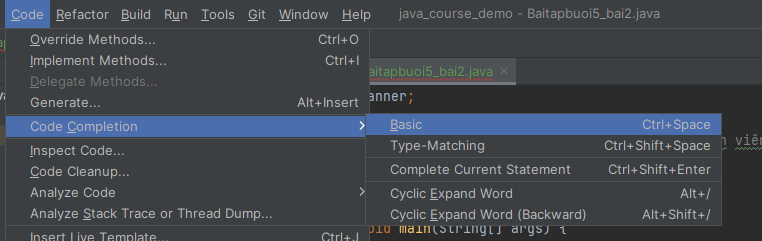


# Disable typo check



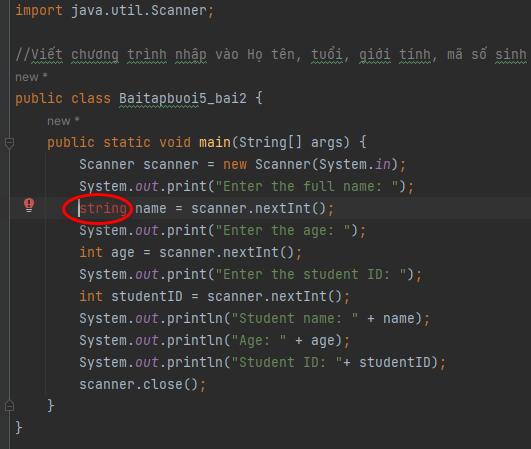
# Intelliji format code







# Khai báo biến kiểu string



**string**: WRONG

In Java, you can declare variables of type **String** using the following syntax:

String variableName; // Declaration of a String variable

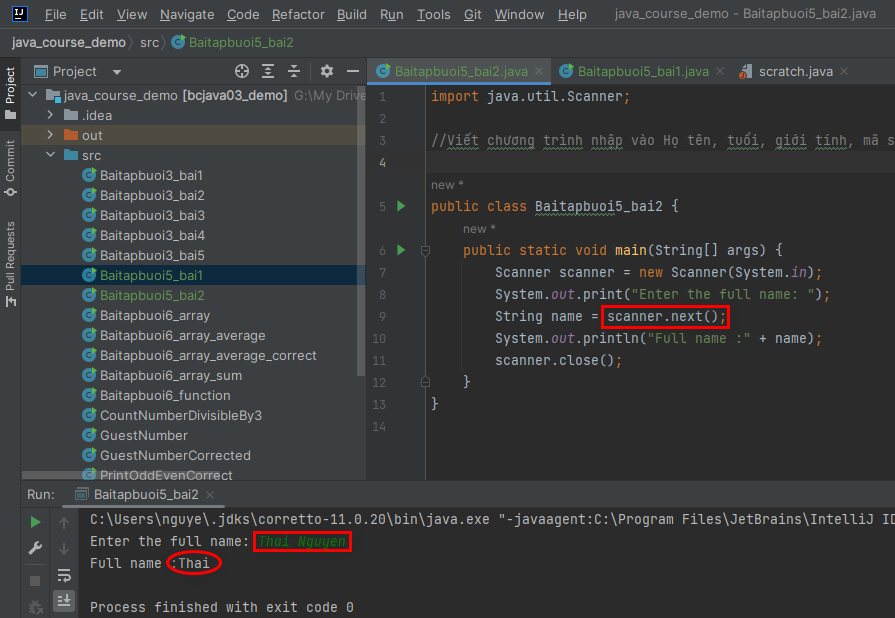
You can also assign a value to the **String** variable at the time of declaration:

String variableName = "Hello"; // Declaration and initialization of a String variable

String variableName = "Hello"; // Declaration and initialization of a String variable

# scanner.next(): Scanner get input as string

import java.util.Scanner;  
  
//Viết chương trình nhập vào Họ tên, tuổi, giới tính, mã số sinh viên. Và in ra các thông tin trên ra màn hinh  
  
public class Baitapbuoi5\_bai2 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter the full name: ");  
 String name = scanner.next();  
 System.*out*.println("Full name :" + name);  
 scanner.close();  
 }  
}



Cause: scanner.next() to read input with spaces

Explain why exception has been thrown in below program ?

public class Baitapbuoi5\_bai2 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the full name: ");

String name = **scanner.next();**

System.out.print("Enter the age: ");

int age = scanner.nextInt();

System.out.println("Student name: " + name);

System.out.println("Age: " + age);

scanner.close();

}

}

Enter the full name: Nguyen Duy

Enter the age: Exception in thread "main" java.util.InputMismatchException

at java.base/java.util.Scanner.throwFor(Scanner.java:939)

at java.base/java.util.Scanner.next(Scanner.java:1594)

at java.base/java.util.Scanner.nextInt(Scanner.java:2258)

at java.base/java.util.Scanner.nextInt(Scanner.java:2212)

at Baitapbuoi5\_bai2.main(Baitapbuoi5\_bai2.java:10)

Ans1:

The exception is thrown in the given program because of a mismatch in the input. The issue is that after reading the full name using **scanner.next()**, the **Scanner** still has the newline character (**\n**) in its buffer. When you try to read the next input with **scanner.nextInt()**, it attempts to parse the newline character as an integer, resulting in a **java.util.InputMismatchException**.

Here's what happens step by step:

1. The program prompts for the full name, and you enter "Nguyen Duy".
2. The **scanner.next()** method reads "Nguyen" and leaves "Duy" (including the newline character) in the buffer.
3. The program then prompts for the age, but when it tries to read the age with **scanner.nextInt()**, it encounters the newline character in the buffer. The newline character cannot be parsed as an integer, leading to the **InputMismatchException**.

To fix this issue, you can add an extra call to **scanner.nextLine()** after reading the name to consume the leftover newline character before reading the age:

public class Baitapbuoi5\_bai2 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); System.out.print("Enter the full name: "); String name = scanner.next(); scanner.nextLine(); // Consume the leftover newline character System.out.print("Enter the age: "); int age = scanner.nextInt(); System.out.println("Student name: " + name); System.out.println("Age: " + age); scanner.close(); } }

By adding **scanner.nextLine()**, the newline character is consumed, and the program will function as expected without throwing the **InputMismatchException**.

public class Baitapbuoi5\_bai2 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter the full name: ");  
 String name = scanner.next();  
 scanner.nextLine(); // Consume the leftover newline character  
  
 System.*out*.print("Enter the age: ");  
 int age = scanner.nextInt();  
 System.*out*.println("Student name: " + name);  
 System.*out*.println("Age: " + age);  
 scanner.close();  
 }  
}

Write a C program and C++ program that read a word (string) and a interger by a space

ChatGPT

Sure! Below are the C and C++ programs that read a word (string) and an integer separated by a space:

C program:

#include <stdio.h>

int main() {

    char word[100];

    int number;

    printf("Enter a word and an integer separated by a space: ");

    scanf("%s %d", word, &number);

    printf("You entered: %s and %d\n", word, number);

    return 0;

}

C++ program:

#include <iostream>

#include <string>

int main() {

    std::string word;

    int number;

    std::cout << "Enter a word and an integer separated by a space: ";

    std::cin >> word >> number;

    std::cout << "You entered: " << word << " and " << number << std::endl;

    return 0;

}

Both programs prompt the user to enter a word (string) and an integer separated by a space. In the C program, we use %s format specifier with scanf to read the word, and %d format specifier to read the integer. In the C++ program, we use std::cin to read the word as a std::string object, and the integer using normal int variable. The entered word and integer are then displayed back to the user.

Process finished with exit code 1

Here's a Java program that takes input for name, age, gender, and student ID from the user and then prints out the entered information on the screen:

import java.util.Scanner;

public class UserInfoPrinter {

    public static void main(String[] args) {

        // Create a Scanner object to read input from the user

        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter their name

        System.out.print("Enter your name: ");

        String name = scanner.nextLine();

        // Prompt the user to enter their age

        System.out.print("Enter your age: ");

        int age = scanner.nextInt();

        scanner.nextLine(); // Consume the newline character left by nextInt()

        // Prompt the user to enter their gender

        System.out.print("Enter your gender: ");

        String gender = scanner.nextLine();

        // Prompt the user to enter their student ID

        System.out.print("Enter your student ID: ");

        String studentID = scanner.nextLine();

        // Print out the entered information

        System.out.println("\n-----------------------------");

        System.out.println("Name: " + name);

        System.out.println("Age: " + age);

        System.out.println("Gender: " + gender);

        System.out.println("Student ID: " + studentID);

        System.out.println("-----------------------------");

        // Close the Scanner

        scanner.close();

    }

}

## Troubleshoting

What wrong in below program ? Why can't I type Student ID when run program

|  |
| --- |
| Enter the full name: Thai Nguyen  Enter the age: 12  Enter the student ID: Student name: Thai Nguyen  Age: 12  Student ID:  Process finished with exit code 0 |

// BEGIN PROGRAM

public class Baitapbuoi5\_bai2 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the full name: ");

        String name = scanner.nextLine();

        System.out.print("Enter the age: ");

        int age = scanner.nextInt();

        System.out.print("Enter the student ID: ");

        String studentID = scanner.nextLine();

        System.out.println("Student name: " + name);

        System.out.println("Age: " + age);

        System.out.println("Student ID: "+ studentID);

        scanner.close();

    }

}

// END PROGRAM

Ans1:

The issue in your program is related to the use of scanner.nextLine() after scanner.nextInt(). **When you use scanner.nextInt(), it reads only the integer value and leaves the newline character (Enter key) in the input buffer.** Then, when you use scanner.nextLine() to read the student ID, it consumes the newline character left in the buffer from the previous nextInt() call, resulting in an empty string for the student ID.

To fix this issue, you can add an extra scanner.nextLine() after scanner.nextInt() to consume the newline character before reading the student ID. Here's the modified code:

import java.util.Scanner;

public class Baitapbuoi5\_bai2 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the full name: ");

        String name = scanner.nextLine();

        System.out.print("Enter the age: ");

        int age = scanner.nextInt();

        // Consume the newline character left in the buffer

        scanner.nextLine();

        System.out.print("Enter the student ID: ");

        String studentID = scanner.nextLine();

        System.out.println("Student name: " + name);

        System.out.println("Age: " + age);

        System.out.println("Student ID: " + studentID);

        scanner.close();

    }

}

With this change, your program will work as expected, and you will be able to enter the student ID after entering the age.

String name = scanner.next(); WRONG

String name = scanner.nextline(); RIGHT

What happend when we don't close scanner ?

// Close the Scanner scanner.close();

ChatGPT

When you don't close the Scanner, it may lead to resource leaks and unexpected behavior. The **Scanner** class in Java uses underlying I/O resources to read input from various sources, such as **System.in**, files, or network streams. If you don't close the **Scanner**, these resources may not be released properly, and it can result in resource leaks.

Resource leaks can be problematic, especially when your program is reading input from standard input (e.g., user input). Leaving the **Scanner** open means that the standard input stream (**System.in**) is not properly closed when your program exits. This may cause the input stream to be in an inconsistent state, and any subsequent attempt to read input might fail or behave unexpectedly.

In the specific case of **System.in**, not closing the **Scanner** might prevent the program from terminating as expected when reading from the standard input. The program may appear to be waiting for input indefinitely, and you might need to forcefully terminate the program.

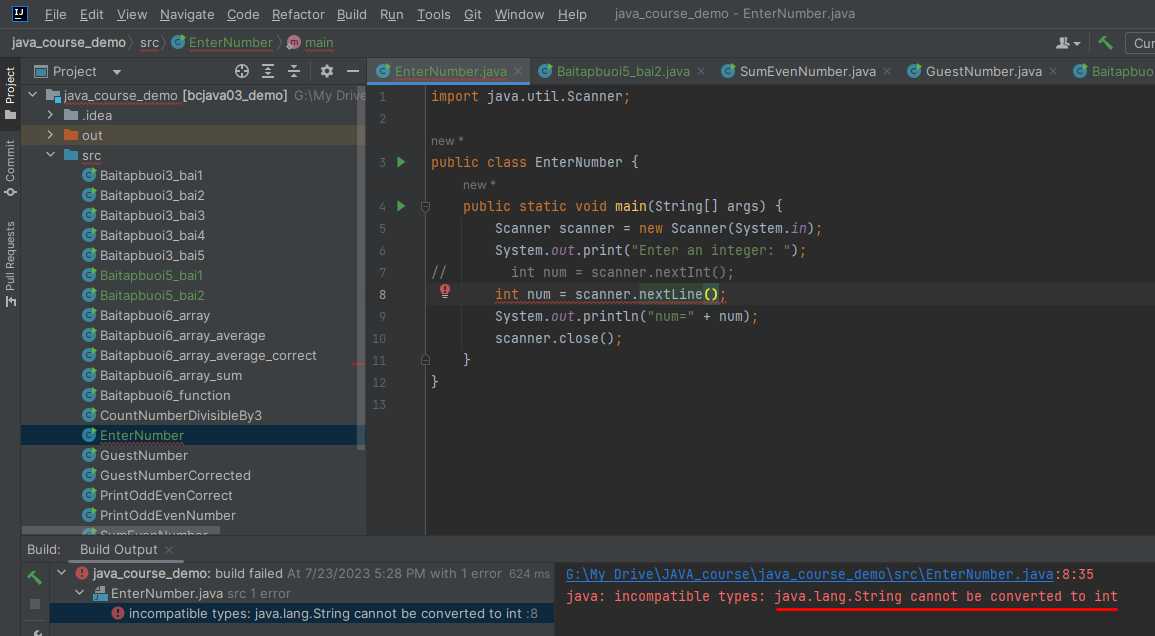
To avoid these issues, it is good practice to close the **Scanner** object when you are done using it. You can close the **Scanner** by calling the **close()** method, as shown in the code provided in the previous example:

javaCopy code

// Close the Scanner scanner.close();

It is a best practice to close any resources you open, such as files, network connections, or **Scanner** objects, to ensure proper cleanup and avoid resource leaks. Using try-with-resources is a recommended approach to handle resource management in Java, as it automatically closes resources when they are no longer needed.

## int num = scanner.nextLine()



# Close scanner

What happend when we don't close scanner ?

// Close the Scanner

scanner.close();

Ans1:

When you don't close the Scanner, it may lead to resource leaks and unexpected behavior. The Scanner class in Java uses underlying I/O resources to read input from various sources, such as System.in, files, or network streams. **If you don't close the Scanner, these resources may not be released properly, and it can result in resource leaks.**

Resource leaks can be problematic, especially when your program is reading input from standard input (e.g., user input). **Leaving the Scanner open means that the standard input stream (System.in) is not properly closed when your program exits. This may cause the input stream to be in an inconsistent state,** and any subsequent attempt to read input might fail or behave unexpectedly.

In the specific case of System.in, not closing the Scanner might prevent the program from terminating as expected when reading from the standard input. The program may appear to be waiting for input indefinitely, and you might need to forcefully terminate the program.

To avoid these issues, it is good practice to close the Scanner object when you are done using it. You can close the Scanner by calling the close() method, as shown in the code provided in the previous example:

java

Copy code

// Close the Scanner

scanner.close();

It is a best practice to close any resources you open, such as files, network connections, or Scanner objects, to ensure proper cleanup and avoid resource leaks. Using try-with-resources is a recommended approach to handle resource management in Java, as it automatically closes resources when they are no longer needed.

Ans2:

If you don't close the Scanner object, it will continue to hold a reference to the underlying input stream. This can cause a resource leak.

A resource leak occurs when an object is no longer needed but its resources are not released. In this case, the Scanner object is no longer needed after the main() method has finished executing. However, if the Scanner object is not closed, it will continue to hold a reference to the underlying input stream. **This will prevent the input stream from being garbage collected, and the resources associated with the input stream will be wasted.**

To avoid this problem, you should always close the Scanner object when you are finished using it. You can do this by calling the close() method on the Scanner object.

The following code shows how to close the Scanner object:

In this code, **the scanner.close() statement is placed at the end of the main() method. This ensures that the Scanner object is closed even if an exception is thrown during the execution of the main() method**.