**Question 1:**

In java, out in and error are 3 instance variables or objects in system class. The type of in out and error is stream. Out and error belong to print stream. type of out and error is print stream. What stream does in belongs to or what is type of in?

**Answer 1:**

1. **System.in:**

This is an instance of the **InputStream** class. It is used for reading input from the console or another input source.

1. **System.out:**

This is an instance of the PrintStream class. It is used for standard output, meaning it is used to print data to the console or another output destination.

1. **System.err:**

Similar to System.out, System.err is also an instance of the PrintStream class. It is typically used for printing error messages to the console or another error output destination.

**How to read data in primitive or appropriate type ?**

2 Ways:

1. *Writing ur own methods.*
2. *Using java built – in functionality.*

**java.util** is a **package** in Java that provides a collection of utility classes, data structures, and other commonly used components. It is part of the Java Standard Library and is widely used in Java programming for various purposes. Some of the key classes and interfaces provided by the java.util package include:

1. **Collections Framework:**
   * **List**: Interface for ordered collections (e.g., **ArrayList**, **LinkedList**).
   * **Set**: Interface for unordered collections with no duplicate elements (e.g., **HashSet**, **TreeSet**).
   * **Map**: Interface for key-value pairs (e.g., **HashMap**, **TreeMap**).
   * **Queue**: Interface for a collection that supports element insertion and removal at both ends (e.g., **LinkedList**, **PriorityQueue**).
2. **Date and Time:**
   * **Date**: Represents a specific instant in time.
   * **Calendar**: Abstract class for calendar-related operations.
   * **GregorianCalendar**: A concrete implementation of the **Calendar** class.
3. **Utilities:**
   * **Arrays**: Provides methods for manipulating arrays.
   * **Collections**: Contains static methods for operating on collections.
   * **Scanner**: Allows reading input from various sources.
4. **Random Number Generation:**
   * **Random**: Provides methods for generating random numbers.
5. **String Tokenizer:**
   * **StringTokenizer**: Breaks a string into tokens.
6. **Observer Design Pattern:**
   * **Observer** and **Observable**: Interfaces and classes for implementing the Observer design pattern.
7. **Miscellaneous:**
   * **UUID**: Represents a universally unique identifier.
   * **Properties**: Represents a persistent set of properties, key-value pairs that can be loaded from or saved to a stream.

**import java.util.\*;**, it means that all the classes and interfaces in the **java.util** package are being imported, allowing you to use them in your code without fully qualifying their names.

**Scanner Class in java.util package:**

The **Scanner** class in the **java.util** package is used for reading input of various types from different sources like the console, files, or strings. It provides methods to parse primitive types and strings.

Java.util.Scanner – java built in functionality.

**Two ways of importing:**

1. **Single Type Import:**

You can import a **specific class** or interface using the import statement followed by the fully qualified name of the class. For example:

**import java.util.Scanner;**

1. **Wildcard Import:**

If you want to import all classes/interfaces within a package, you can use the wildcard \*. For example:

**import java.util.\*;**

**Question 2:**

If we import one class, then only that class will be used. But if we import a package, then will the computational be heavy? Will it be complex?

**Answer 2:**

No. Whole package will not be loaded into memory (RAM). Only a chunk will be loaded into memory **-> JIT Compiler**

Only the reference class will be loaded into the memory not the whole package. The only thing which has reference will be loaded into the memory. So importing a class or whole package will have no effect.

**Additional answer to read more:**

* Importing a single class or importing a package does not significantly affect the computational heaviness or complexity of your Java program. The primary purpose of importing classes or packages is to provide a convenient way to reference and use the classes or interfaces defined in those packages.

**JIT Compiler:**

* Import statements are primarily a compile-time feature and are used by the Java compiler to resolve class names during the compilation process.
* The JIT compiler, which operates at runtime, does not take import statements into consideration. Its main task is to convert the compiled bytecode into native machine code for efficient execution on the specific hardware.

**Memory Usage:**

* Importing a single class or a package does not significantly affect memory usage at runtime.
* The compiled bytecode (class files) contains symbolic references to the classes and methods used in the code. The actual loading of classes and their bytecode occurs at runtime when they are needed (class loading is a dynamic process in Java).
* Importing a package does not mean that all classes in the package are loaded into memory when the program starts. Only the classes that are actually used during the execution of the program will be loaded.

**Creating an Instance In Java:**

* Using **new operator** and using parenthesis.

**In C++,** we create object of class like this : Scanner obj;

But in Java, we use new operator.

**In Java**, if we write this Scanner obj then obj will not be instance of Scanner class, it will be a simple variable containing nothing.

Creating instance of Scanner class in Java:

**Scanner obj = new Scanner();**

In Java, it is necessary to mention constructort () – default.

**Some things in Java:**

* There is no concept of **garbage value** in Java. Default value is null in Java.
* There are **no pointers** in Java.
* There is **no destructor** in Java.
* There is **no garbage collector** that u have to invoke.
* It is not the responsibility of developer to release / free memory.

**Ques:**

We are creating an instance on heap (new). Then how will memory be released?

**Ans:**

Memory management is totally the responsibility of underline virtual machine i.e.; **JVM**.

JVM will invoke a service periodically which is named as garbage collector and it will destroy all those things whose reference doesn’t exist.

We do not know when it will be invoked.

**A way to invoke garbage collector:**

There is a way but it is not a guaranteed way. We can request JVM to invoke garbage call. We can request it by sending a call System.gc(). But it is not guaranteed.

Calling **System.gc()** is a way to suggest to the Java Virtual Machine (JVM) that it might be a good time to run the garbage collector, but it doesn't guarantee that the garbage collector will actually run immediately. The decision of when and how to run the garbage collector is ultimately up to the JVM.

* The **System.gc()** method is a standard Java method that is part of the **System** class.
* When you call **System.gc()**, you are making a suggestion to the JVM, but it doesn't force an immediate garbage collection. The JVM may choose to ignore the suggestion.

**=>**

This code creates a new instance of the Scanner class in Java and associates it with the standard input stream (System.in).

**Scanner obj = new Scanner(System.in);**

* **Scanner:** This is a class in the **java.util package** that allows you to read various types of input, such as numbers and strings, from different sources like the console, files, or strings.
* **obj:** This is the **name of the variable** that refers to the newly created Scanner object.
* **new Scanner(System.in):** This part of the code **creates a new instance** of the Scanner class, and the **constructor of Scanner** takes an argument specifying the input source.
* **System.in** represents the standard input stream, typically connected to the keyboard for console input.

When you create a **Scanner object obj** with **System.in** as its argument, you are essentially telling the **Scanner object obj** to read input from the standard input stream.

After this line of code is executed, you can use the **obj** variable to interact with the **Scanner** object and read input from the standard input stream. For example, you might use methods like **nextInt()**, **nextDouble()**, or **nextLine()** to read different types of input.

**Several utility functions:**

These methods are used to read input of specific types from the standard input stream (System.in) or another input source.

|  |  |
| --- | --- |
| **Methods** | **Type of values they return** |
| obj.nextInt() | Returns next token as int |
| .nextLine() | Returns the rest of the current line, including any space characters. |
| .nextByte() | byte |
| .nextDouble() | double |
| .nextLong() | long |
| .nextFloat() | float |
| .nextBoolean() | boolean |
| .next() | string |
| .nextShort() | short |

We will get data using these functions.

**Eg:**

Scanner scanner = new Scanner(System.in);

System.out.print("Enter an integer: ");

int intValue = scanner.nextInt();

System.out.println("You entered: " + intValue);

**Taking input from user in calculator:**

* 1. **First import java.util.Scanner;**
  2. **Create instance of scanner class.**
  3. **Call any utility function.**
* Everything will be written inside the class except 2 things:

1. Packages
2. Inheritance relationship

* **Coding:**

1. **Input.java file:**
   * + Import package:

- **Import java.util.\*;**

* + - Write class and inside class write main function.
    - In main function, initialize some variables like - **String degree, batch, session, id;**

**String –** reference / object type like in C++

* + - Creating instance of Scanner class

**Scanner in = new Scanner(System.in);**

**System.in –** Source from where input will be taken.

* + - Prompt and read input.

**System.out.println("What's your Degree?");**

**degree = in.nextLine();** // Read one line from the console.

Cursor will keep on blinking until it gets Ascii value of enter (until we press enter key).When it will get Ascii value of enter, function will be called and input will be saved to degree variable.

* + - Similar steps for other variables
    - At last, remember to close the **Scanner** object in when you are done using it to release associated resources. You can do this by calling the **close()** method:

**in.close();**

When we open an external connection (file, socket base, network of another computer etc), we close the connection when our work gets finished.

**Connection cooling** – concept

If there are 100 or 1000 of users on a system and they open multiple connections, then they will remain in memory until they destroy or close the connection. Maybe some clients will have to wait unnecessarily for their turn.

If we do not write in.close(), there is no issue now because it is a single program on a single system. But its better to write it.

* + - Display output

**System.out.println("Your Roll Number: " + degree+batch+section+id);**

**+** will merge or concatenate all of these in String

**Ques:**

Is there any **concept of operator overloading** in Java?

**Ans:**

No

**Ques:**

Then why + is concatenating them in string. While + adds. + is showing multiple behavior. Then isn’t the + operator overloaded? So is there operator overloading in Java?

**Ans:**

There is operator overloading in built-in. Developer can’t specifically overload any operator.

* There is no diamond prblm in Java but there is inheritance relationship in Java which is written outside of class.
* If we will run input.java code without the first line (import java.utils.\*), then we will get an error of unknown Scanner. It cant find Scanner class, Scanner class will be in the scope when the pckg will be imported.

Open cmd, go to the location where ur Input.java file is. Then compile code by typing **javac Input.java** on cmd. Bytecode will be generated. Then type **java Input** and code will run and then enter respective inputs and output will be shown.

**Javac “.java”** will compile all the java files in the folder.

1. **Input1.java file (Variant of upper code):**

Java provides 2 things:

1. API
2. JVM

**GUI Components:**

We can make ourselves and there are also built-in GUI components.

**Built-in Java GUI Components:**

1. I/P GUI Component
2. O/P GUI Component

**Two packages** will be used while trying to make GUI Components:

1. java.awt (Abstract Window Toolkit)- part of the Java Standard Edition (SE)
2. javax.swing (extension of **java.awt** )- part of the Java Foundation Classes (JFC)

* All **packages** of java are with its name. Some packages will have only jave like **java.\_\_\_** and some will have x written with java like **javax.\_\_\_\_** x shows extension.
* Packages with name of java only ( java.\_\_\_) are part of Core Java.
* x is introduced per need, as extension it is introduced.
* It is just a naming convention.

**Code:**

* First import java package to use built-in GUI component of java

**import javax.swing.\*;**

* Then like previous code. Write class then inside it main function and inside main initialize variables.
* Prompt and read input

**degree = JOptionPane.showInputDialog(“Enter your degree: “);**

* **JOptionPane:** is a class. It is part of javax.swing package.
* **showInputDialog**: is static function of this class.
* The parameter in the function body will be shown.
* Input will be saved in degree variable.
* String is return type of showInputDialog.

**Question 3 (HW):**

In Scanner class, we had an option of chosing appropriate data type like by using .nextInt(), we get data in integer type.

In this case, return type of showInputDialog is a string and we don’t have a option of chosing appropriate data type.

Suppose there is a prgrm to take input of 2 nos and to add them. If we will try to do it here, it will concatenate those numbers because return type of this class is string like this “100” + “200” = “100200”.

So, how to convert this data into appropriate type?

How to convert String into integer or double, then add 2 nos and show their result in this case?

**Answer 3:**

1. We use **JOptionPane.showInputDialog** to get two numbers as input from the user. The inputs are initially stored as **String**.
2. We use **Integer.parseInt** to convert these **String** inputs to **int**. If you were working with decimal numbers, you could use **Double.parseDouble** instead.
3. We perform the addition of the two int values.
4. We display the result using JOptionPane.showMessageDialog.

**Integer.parseInt** is a method in Java that parses a string as a signed decimal integer. It is part of the Integer class in the Java API.

Here's the signature of the parseInt method:

**public static int parseInt(String s) throws NumberFormatException**

In Java, the **Integer class is part of the java.lang package**, which is automatically imported into every Java program. The java.lang package contains fundamental classes and is implicitly imported, so you don't need to explicitly import it in your code.

import javax.swing.JOptionPane;

public class AddTwoNumbers {

public static void main(String[] args) {

// Get the first number as a String

String input1 = JOptionPane.showInputDialog("Enter the first number:");

// Get the second number as a String

String input2 = JOptionPane.showInputDialog("Enter the second number:");

try {

// Convert the String inputs to integers

int num1 = Integer.parseInt(input1);

int num2 = Integer.parseInt(input2);

// Perform addition

int sum = num1 + num2;

// Display the result

JOptionPane.showMessageDialog(null, "Sum: " + sum);

} catch (NumberFormatException e) {

// Handle the case where the input is not a valid integer

JOptionPane.showMessageDialog(null, "Invalid input. Please enter valid integers.");

}

}

}

* Prompt and read input for other variables
* Then to display output

**JOptionPane.showMessageDialog(null, “RollNumber: “ +degree+batch+session+id);**

* What is null here?
* Reference of null is for top level container.
* Where on the screen, GUI should be displayed. On which frame? We pass it as first parameter. Null shows on default on the screen.
* The **showMessageDialog method** is a **static** method of the JOptionPane class, and it is often used to display a simple message dialog box. The first parameter to this method represents the **parent component or owner frame of the dialog.**
* When **null is passed as the parent component**, it means that the dialog is not associated with any specific component or frame. In other words, the dialog will be displayed in the center of the screen or in a default position determined by the windowing system.
* method signature for showMessageDialog:

**public static void showMessageDialog(Component parentComponent, Object message);**

1. **MyFirstProg.java file:**

String args[] – **Command Line arguments.**

If I go to cmd and run this code by **java MyFirstProg**, it gives me **Hello World** in output.

If I pass some arguments like java **MyFirstProg hello bit f21 FCIT 123 234 123 22 334**, it gives me no errors and still returns **Hello World** in output.

**Command Line Arguments:**

We are giving some arguments while executing the file, these arguments are called **command line arguments.**

These arguments will be saved in **String args[].** All data will be here.

And these will be **saved on the base of space**.

Like at index 0, hello will be saved, at index 1, bit will be saved.

All of them will be saved in form of String.

1. **CmdLineArgs.java file:**

Displaying command line arguments.

We can use a loop.

for(int i = 0; I < args.length; i++)

{

System.out.println(“args[“ + i + “]: “ + args[i]);

}

args.length -> length is property of args.

Array is also an object.

If we will pass nothing on cmd. If we will pass no command line arguments, then length will be zero and the loop will terminate.

**Question 4 (HW):**

What if we want to pass space as cmd line argument? Like on the basis of space, arguments are being passed to array indexes. It is terminating on the basis of space. But what if I want to pass space as argument? Like if I want to pass Fakhra Rabbani as whole data in an index, so how to do it?

**Anwer 4:**

The space character separates command-line arguments.

1. When you want to pass a command line argument that includes spaces as a single argument (without being split into multiple arguments), you can enclose the entire argument in **double quotes**. This way, the entire string, including spaces, will be treated as a single command line argument.

**java CmdLineArgs "Fakhra Rabbani" arg2 arg3**

1. **Account Class:**

**Local variable** – variable inside a function boundary

**Instance variable** – outside a function but inside class boundary

Like C++, if we do not write a constructor then **default constructor** will be provided in Java. Default constructor will initialize instance variable with default value. There are no garbage values in Java.

|  |  |
| --- | --- |
| **Data types** | **Default Value** |
| Int | 0 |
| Double | 0.0 |
| String | null |
| Boolean | false |
| Obj (for object references eg instance of a class) | null |
| char | '\u0000' (null character) |
| byte | 0 |
| short | 0 |
| float | 0.0f |
| long | 0L |

Just like C++, if u have any constructor in a class, then Java will not provide its default constructor.

**Account class main:**

* Create instance of Scanner class for input of appropriate type.
* Create instance of Account class for use. (Default constructor provided by Java)

**Account class functions:**

* Create an instance variable balance
* deposit function – Adds deposited amount to balance.
* withdraw function – If balance is less than amount to be withdrawn then return 0.0 as no amount has been withdrawn, amount cant be withdrawn. Otherwise, withdraw amount from balance (balance – amount) and return the amount that is withdrawn.
* getbalance – simply returns balance

**Execution** of Account class code will start from main.

**println ->** prints and then breaks line and takes control to next line

**print ->** prints line but doesn’t break it. Control remains there.

**TASK (HW):**

Write a program to implement a calculator to perform following **functionalities:**

1. Addition of 2 numbers
2. Subtraction
3. Multiplication
4. Division

In main function, create a switch menu, accept 2 numbers from user using **GUI components**, and make appropriate calls to the function.

**Instructions:**

1. Do not accept any negative number.
2. Provide 2 constructors in the class.