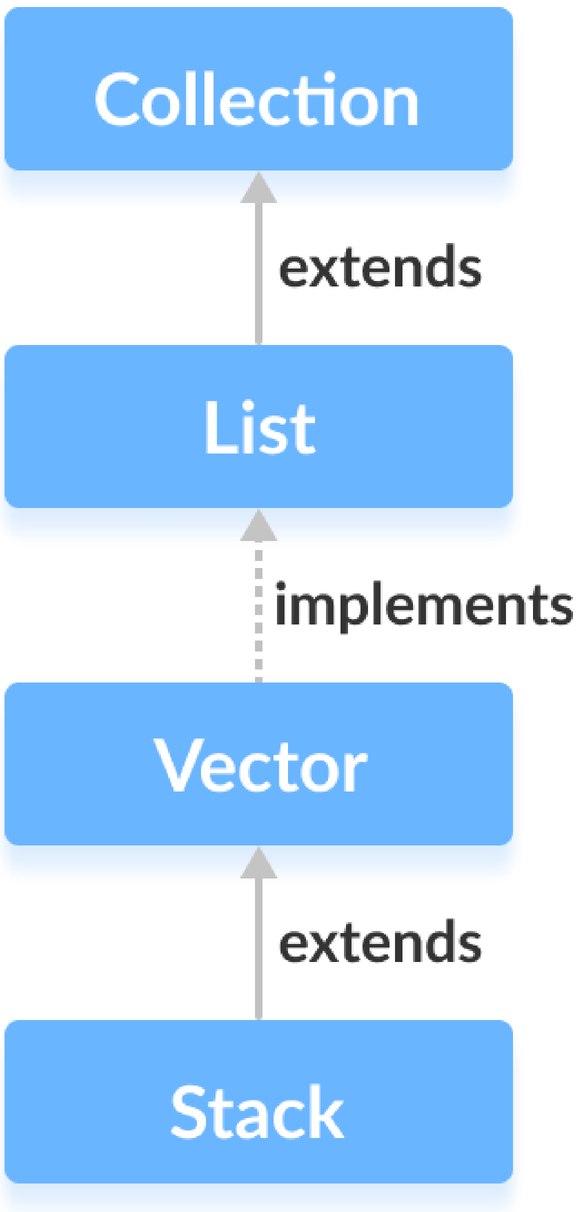
STACK, PACKAGE AND EXCEPTION HANDLING

Java Stack Class

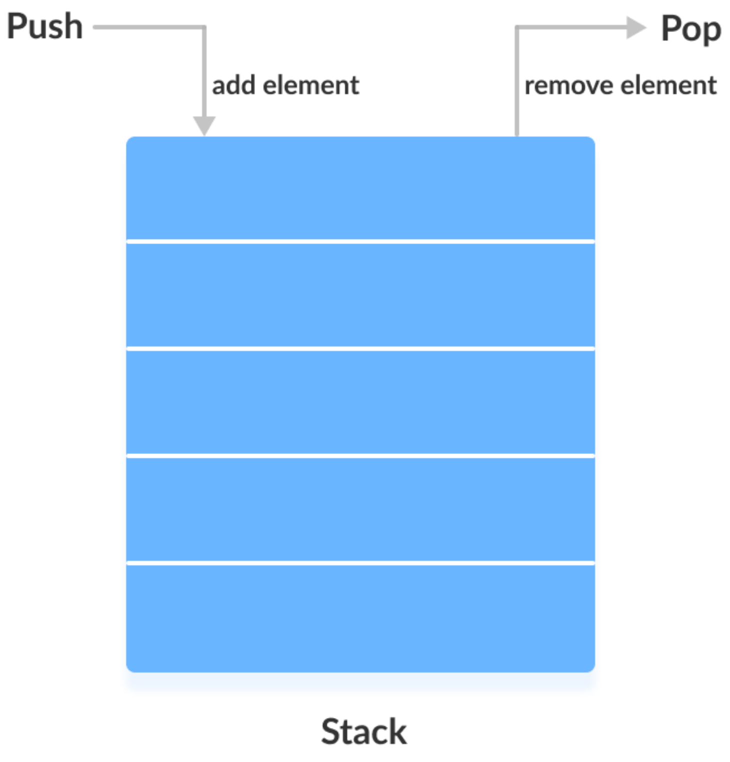
The Java collections framework has a class named Stack that provides the functionality of the stack data structure.

The Stack class extends the Vector class.



Stack Implementation

In stack, elements are stored and accessed in **Last In First Out** manner. That is, elements are added to the top of the stack and removed from the top of the stack.



Creating a Stack

In order to create a stack, we must import the java.util.Stack package first. Once we import the package, here is how we can create a stack in Java.

```
Stack<Type> stacks = new Stack<>();
```

Here, Type indicates the stack's type. For example,

```
// Create Integer type stack
Stack<Integer> stacks = new Stack<>();


// Create String type stack
Stack<String> stacks = new Stack<>();
```

Stack Methods

Since Stack extends the Vector class, it inherits all the methods Vector. To learn about different Vector methods, visit Java Vector Class.

Besides these methods, the Stack class includes 5 more methods that distinguish it from Vector.

1. **push() Method**
2. **pop() Method**
3. **peek() Method**
4. **search() Method**
5. **empty() Method**

push() Method

To add an element to the top of the stack, we use the push() method. For example,

```
import java.util.Stack;


class Main {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();


        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");


        System.out.println("Stack: " + animals);
    }
}
```

Output

```
Stack: [Dog, Horse, Cat]
```

pop() Method

To remove an element from the top of the stack, we use the pop() method. For example,

```
import java.util.Stack;


class Main {
    public static void main(String[] args) {
        Stack<String> animals= new Stack<>();


        // Add elements to Stack
        animals.push("Dog");
        animals.push("Horse");
        animals.push("Cat");
        System.out.println("Initial Stack: " + animals);


        // Remove element stacks
        String element = animals.pop();
        System.out.println("Removed Element: " + element);
    }
}
```

Output

```
Initial Stack: [Dog, Horse, Cat]
Removed Element: Cat
```

peek() Method

The peek() method returns an object from the top of the stack. For example,

```
import java.util.Stack;


class Main {

    public static void main(String[] args) {

        Stack<String> animals= new Stack<>();


        // Add elements to Stack

        animals.push("Dog");

        animals.push("Horse");

        animals.push("Cat");

        System.out.println("Stack: " + animals);


        // Access element from the top

        String element = animals.peek();

        System.out.println("Element at top: " + element);


    }

}
```

Output

```
Stack: [Dog, Horse, Cat]

Element at top: Cat
```

search() Method

To search an element in the stack, we use the search() method. It returns the position of the element from the top of the stack. For example,

```
import java.util.Stack;


class Main {

   public static void main(String[] args) {

      Stack<String> animals= new Stack<>();


      // Add elements to Stack

      animals.push("Dog");

      animals.push("Horse");

      animals.push("Cat");

      System.out.println("Stack: " + animals);


      // Search an element

      int position = animals.search("Horse");

      System.out.println("Position of Horse: " + position);

   }

}
```

Output

```
Stack: [Dog, Horse, Cat]

Position of Horse: 2
```

empty() Method

To check whether a stack is empty or not, we use the empty() method. For example,

```
import java.util.Stack;


class Main {

   public static void main(String[] args) {

      Stack<String> animals= new Stack<>();


      // Add elements to Stack

      animals.push("Dog");

      animals.push("Horse");

      animals.push("Cat");

      System.out.println("Stack: " + animals);


      // Check if stack is empty

      boolean result = animals.empty();

      System.out.println("Is the stack empty? " + result);

   }

}
```

Output

```
Stack: [Dog, Horse, Cat]

Is the stack empty? false
```

Java Packages & API

A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- **Built-in Packages** (packages from the Java API)
- **User-defined Packages** (create your own packages)

Built-in Packages

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.

The library contains components for managing input, database programming, and much much more. The complete list can be found at Oracles website: https://docs.oracle.com/javase/8/docs/api/.

The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

To use a class or a package from the library, you need to use the import keyword:

Syntax

import *package*.*name*.*Class*;   // Import a single class

import *package*.*name*.*;   // Import the whole package

Import a Class

If you find a class you want to use, for example, the Scanner class, **which is used to get user input**, write the following code:

Example

import java.util.Scanner;

In the example above, java.util is a package, while Scanner is a class of the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read a complete line:

Example: Using the Scanner class to get user input:

```
import java.util.Scanner;
class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");


    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  } }
```

Import a Package

There are many packages to choose from. In the previous example, we used the Scanner class from the java.util package. This package also contains date and time facilities, random-number generator and other utility classes.

To import a whole package, end the sentence with an asterisk sign (*). The following example will import ALL the classes in the java.util package:

Example

```
import java.util.*;
```

Built-in Packages

These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:

1. **java.lang:** Contains language support classes(e.g classes which defines primitive data types, math operations). This package is automatically imported.
2. **java.io:** Contains classes for supporting input / output operations.
3. **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4. **java.applet:** Contains classes for creating Applets.
5. **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc). 6)
6. **java.net:** Contain classes for supporting networking operations.

USER-DEFINED PACKAGES: These are the packages that are defined by the user. First we create a directory myPackage (name should be same as the name of the package). Then create the MyClass inside the directory with the first statement being the package names.

```
// Name of the package must be same as the directory

// under which this file is saved

package myPackage;


public class MyClass

{

    public void getNames(String s)

    {

        System.out.println(s);

    }

}
```

Now we can use the MyClass class in our program.

```
/* import 'MyClass' class from 'names' myPackage */

import myPackage.MyClass;


public class PrintName

{

   public static void main(String args[])

   {

      // Initializing the String variable

      // with a value

      String name = "GeeksforGeeks";


      // Creating an instance of class MyClass in
      // the package.
      MyClass obj = new MyClass();


      obj.getNames(name);
   }
}
```

Note : MyClass.java must be saved inside the myPackage directory since it is a part of the package.

Using Static Import

Static import is a feature introduced in Java programming language (versions 5 and above) that allows members (fields and methods) defined in a class as public static to be used in Java code without specifying the class in which the field is defined. Following program demonstrates static import:

```
// Note static keyword after import.
import static java.lang.System.*;


class StaticImportDemo {
    public static void main(String args[])
    {
        // We don't need to use 'System.out'
        // as imported using static.
        out.println("GeeksforGeeks");
    }
}
```

Output:

```
 GeeksforGeeks
```

Java Exceptions

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

Java try and catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

Exception Handling in Java is one of the effective means to handle runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

What are Java Exceptions?

In Java, Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

Syntax

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Consider the following **example**:

This will generate an error, because myNumbers[10] does not exist.

```
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

The output will be something like this:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at Main.main(Main.java:4)
```

If an error occurs, **we can use try...catch to catch the error** and execute some code to handle it:

Example

```
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

The output will be:

```
Something went wrong.
```

Finally

The finally statement lets you execute code, after try...catch, regardless of the result:

Example

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    }
   catch (Exception e) {
      System.out.println("Something went wrong.");
    }
   finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

The output will be:

```
Something went wrong.

The 'try catch' is finished.
```

The **throw** keyword

The **throw** statement allows you **to create a custom error**.

The **throw** statement is used together with an **exception type**. There are many exception types available in
Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, Security Exception, etc:

Example

Throw an exception if **age** is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

The output will be:

```
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.
        at Main.checkAge(Main.java:4)
        at Main.main(Main.java:12)
```

Using **throws** keyword:

The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Example

```
import java.io.FileNotFoundException;

import java.io.FileReader;


public class ThrowsExample {

    public static void main(String[] args) throws FileNotFoundException {

        try {

            FileReader file = new FileReader("file.txt");

            // Do something with the file

        } catch (FileNotFoundException e) {

            System.out.println("File not found: " + e.getMessage());

            throw e; // Re-throwing the exception

        }

    }

}
```

In this example, the **main()** method declares that it may throw a **FileNotFoundException** by using the **throws** keyword. Inside the method, we attempt to create a **FileReader** object for a file named "file.txt". If the file is not found, a **FileNotFoundException** is thrown. We catch the exception, handle it by printing a message, and then re-throw it using the **throw** keyword. This propagates the exception to the calling method or the JVM.

1. Write a Java program to implement a stack with push and pop operations. Find the top element of the stack and check if it is empty or not.

Click me to see the solution

2. Write a Java program to sort the elements of a given stack in ascending order.

Click me to see the solution

3. Write a Java program to sort the elements of the stack in descending order.

Click me to see the solution

4. Write a Java program to reverse the elements of a stack.

Click me to see the solution

5. Write a Java program to find the maximum and minimum elements in a stack.

Click me to see the solution

6. Write a Java program to remove all elements from a stack.

Click me to see the solution

7. Write a Java program to count all stack elements.

Click me to see the solution

8. Write a Java program to implement a stack that checks if a given element is present or not in the stack.

Click me to see the solution

9. Write a Java program to remove duplicates from a given stack.

Click me to see the solution

10. Write a Java program to find the top and bottom elements of a given stack.

Click me to see the solution

11. Write a Java program to rotate the stack elements to the right direction.

Click me to see the solution

12. Write a Java program to rotate the stack elements in the left direction.

Click me to see the solution

13. Write a Java program to remove a specific element from a stack.

Click me to see the solution

14. Write a Java program to swap the top two elements of a given stack.

Click me to see the solution

15. Write a Java program to get the nth element from the top of the stack.

Click me to see the solution

16. Write a Java program to get the nth element from the bottom of the stack.

Click me to see the solution

17. Write a Java program to implement a stack and move the nth element from the top of the stack to the top.

Click me to see the solution

18. Write a Java program to merge two stacks into one.

Click me to see the solution

19. Write a JavaScript program that implements a stack and checks if the stack is a subset of another stack.

Click me to see the solution

20. Write a Java program that implements a stack and checks if two stacks are equal.

Click me to see the solution

21. Write a Java program that implements a stack and finds common elements between two stacks.

Click me to see the solution

22. Write a Java program that implements a stack and find elements that are in the first stack but not in the second stack.

Click me to see the solution

23. Write a Java program that implements a stack and creates a new stack that contains all elements from two stacks without duplicates.

Click me to see the solution

24. Write a Java program that implements a stack and creates a new stack from a portion of the original stack.

Click me to see the solution

25. Write a Java program that implements a stack and creates a new stack that contains only elements that are in either the first or the second stack, but not in both.

Click me to see the solution

26. Write a Java program that implements a stack and checks if all elements of the stack satisfy a condition.

Click me to see the solution

27. Write a Java program that implements a stack and checks if at least one element of the stack satisfies a condition.

Click me to see the solution

28. Write a Java program that implements a stack and create a new stack by removing elements that do not satisfy a condition.

Click me to see the solution

29. Write a Java program to implement a stack using a linked list.

Click me to see the solution

30. Write a Java program that throws an exception and catch it using a try-catch block.

Click me to see the solution

31. Write a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd.

Click me to see the solution

32. Write a Java program to create a method that reads a file and throws an exception if the file is not found.

Click me to see the solution

33. Write a Java program that reads a list of numbers from a file and throws an exception if any of the numbers are positive.

Click me to see the solution

34. Write a Java program that reads a file and throws an exception if the file is empty.

Click me to see the solution

35. Write a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.

Click me to see the solution

36. Write a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.

Click me to see the solution