**STREAM**

**https://www.geeksforgeeks.org/stream-in-java/**

**Intermediate Operations:**

1. **map:**The map method is used to map the items in the collection to other objects according to the Predicate passed as argument.  
   List number = Arrays.asList(2,3,4,5);  
   List square = number.stream().map(x->x\*x).collect(Collectors.toList());
2. **filter:** The filter method is used to select elements as per the Predicate passed as argument.  
   List names = Arrays.asList("Reflection","Collection","Stream");  
   List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
3. **sorted:** The sorted method is used to sort the stream.  
   List names = Arrays.asList("Reflection","Collection","Stream");  
   List result = names.stream().sorted().collect(Collectors.toList());

**Terminal Operations:**

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.  
   List number = Arrays.asList(2,3,4,5,3);  
   Set square = number.stream().map(x->x\*x).collect(Collectors.toSet());
2. **forEach:** The forEach method is used to iterate through every element of the stream.  
   List number = Arrays.asList(2,3,4,5);  
   number.stream().map(x->x\*x).forEach(y->System.out.println(y));
3. **reduce:** The reduce method is used to reduce the elements of a stream to a single value.  
   The reduce method takes a BinaryOperator as a parameter.

List number = Arrays.asList(2,3,4,5);  
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

|  |
| --- |
| //a simple program to demonstrate the use of stream in java  **import** java.util.\*;  **import** java.util.stream.\*;    **class** Demo  {  **public** **static** **void** main(String args[])    {        // create a list of integers      List<Integer> number = Arrays.asList(2,3,4,5);        // demonstration of map method      List<Integer> square = number.stream().map(x -> x\*x).                             collect(Collectors.toList());      System.out.println(square);        // create a list of String      List<String> names =                  Arrays.asList("Reflection","Collection","Stream");        // demonstration of filter method      List<String> result = names.stream().filter(s->s.startsWith("S")).                            collect(Collectors.toList());      System.out.println(result);        // demonstration of sorted method      List<String> show =              names.stream().sorted().collect(Collectors.toList());      System.out.println(show);        // create a list of integers      List<Integer> numbers = Arrays.asList(2,3,4,5,2);        // collect method returns a set      Set<Integer> squareSet =           numbers.stream().map(x->x\*x).collect(Collectors.toSet());      System.out.println(squareSet);        // demonstration of forEach method      number.stream().map(x->x\*x).forEach(y->System.out.println(y));        // demonstration of reduce method  **int** even =         number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);        System.out.println(even);    }  } |

//remove item from a list  
  
List<String> fruits = new ArrayList<>();  
fruits.add("Apple");  
fruits.add("Banana");  
fruits.add("Stawberry");  
  
Iterator<String> fruitIterator = fruits.iterator();  
  
while(fruitIterator.hasNext()) {  
 if(fruitIterator.next().contains("Apple")) {  
 fruitIterator.remove();  
 }  
}

System.*out*.println (fruits);

// defference between List - Set  
  
List<String> data = Arrays.*asList*("ab","fc","cd","ab","bc","cd");  
  
List<String> list = new ArrayList<>(data);// will add data as is  
Set<String> set1 = new HashSet<>(data); // will add data keeping only unique values  
Set<String> set2 = new TreeSet<>(data); // will add data keeping unique values and sorting  
Set<String> set3 = new LinkedHashSet<>(data); // will add data keepig only unique values and preserving the original order  
  
System.*out*.println(list);  
System.*out*.println(set1);  
System.*out*.println(set2);  
System.*out*.println(set3);

UnModiFiableList -->ReadOnly

public class UnModifiablList {  
  
 private List<Integer> intList = new ArrayList<>();  
  
 public void addValue(Integer value) {  
 intList.add(value);  
 }  
  
 public List<Integer> getIntList() {  
 return Collections.*unmodifiableList*(intList);

Check Funtional Interface

package Main;  
  
@FunctionalInterface  
public interface bar {  
 void barInt();  
 default int batDouble() { return 0};  
}

GENERIC CLASS

Unlike other identifiers, type parameters have no naming constraints. However their names are commonly the first  
letter of their purpose in upper case. (This is true even throughout the official JavaDocs.)  
Examples include T for "type", E for "element" and K/V for "key"/"value"

public class TestGeneric<T> {  
  
 private T value;  
  
 public void setValue(T value) {  
 this.value=value;  
 }  
  
 public T getValue(){  
 return this.value;  
 }  
}

**DELETE FILE**

File file = new File("d:\\InputData\\JavaTakeNote.docx");  
if(file.exists()) {  
 boolean deleteStatus = file.delete();  
 if(!deleteStatus) {   
 throw new IOException("Unable to delete file");  
 }  
} else {  
 throw new IOException("File does not exist !" );  
}

**Iterating on each file within a folder**

File file = new File("d:\\InputData");  
for(File seletedFile : file.listFiles()) {  
 String property = seletedFile.isFile() ? "file" :"directory";  
 System.*out*.println (property + " " + seletedFile.getAbsolutePath() );  
  
}

**RECURSE FOLDER**

public static void recurseFolder (String path) {  
 File file = new File(path);  
 for(File seletedFile : file.listFiles()) {  
 String property = seletedFile.isFile() ? "file" :"directory";  
 if(property.equals("directory")){  
 *recurseFolder*(seletedFile.getAbsolutePath());  
 }  
  
 System.*out*.println (property + " " + seletedFile.getAbsolutePath() );

}  
}

**COPY FILE**

public static void copyFile(String sourceFile, String desFile) {  
 File source = new File (sourceFile);  
 File des = new File(desFile);  
 try {  
 FileChannel sourceChanel = new FileInputStream(source).getChannel();  
 FileChannel desChanel = new FileOutputStream(des).getChannel();  
 sourceChanel.transferTo(0,sourceChanel.size(),desChanel);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

*// Read from baseReader, one line at a time*BufferedReader reader = new BufferedReader( baseReader );  
String line;  
while((line = reader.readLine()) != null) {  
*// Remember: System.out is a stream, not a writer!*System.out.println(line);  
}

**READ FILE USE SCANNER**

public static void readFile(String sourceFile) {  
 try (Scanner sc = new Scanner(new File(sourceFile))) {  
  
 while (sc.hasNextLine()) {  
 System.*out*.println(sc.nextLine());  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
}

public static void readWoedByWord(String sourceFile) {  
 try (Scanner sc = new Scanner(new File(sourceFile))) {  
  
 while (sc.hasNext()) {  
 System.*out*.println(sc.next());  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
}

**CREATE A NEW FOLDER**

public static void makeFolder() {  
 String path ="d:\\InputData\\NewFolder";  
 File file = new File (path);  
 file.mkdir();  
  
}

**READ FILE**

public static void readFile() {  
  
 String pathFile = "d:\\InputData\\strorage.xml";  
 try {  
 BufferedReader br = new BufferedReader(new FileReader(pathFile));  
 String s =null;  
 while ((s=br.readLine())!=null){  
 System.*out*.println(s);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**SINGLETON PARTON USE ENUM**

package Main;  
  
public enum SingleTon {  
  
 *INSTANCE*;  
  
 private SingleTon () {}  
  
 public void f() {  
 System.*out*.println("CODE HERE");  
  
 }  
}