Streams:

Stream is a new abstract layer introduced in Java 8. Using stream, you can process data in a declarative way similar to SQL statements.

* Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
* Stream is functional in nature. Operations performed on a stream does not modify it's source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
* Stream is lazy and evaluates code only when required.
* The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

You can use stream to filter, collect, print, and convert from one data structure to other etc

**package** hacker;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Collectors;

**class** Product{

**int** id;

String name;

**float** price;

**public** Product(**int** id, String name, **float** price) {

**this**.id = id;

**this**.name = name;

**this**.price = price;

}

}

**public** **class** JavaStreamExample {

**public** **static** **void** main(String[] args) {

List<Product> productsList = **new** ArrayList<Product>();

//Adding Products

productsList.add(**new** Product(1,"HP Laptop",25000f));

productsList.add(**new** Product(2,"Dell Laptop",30000f));

productsList.add(**new** Product(3,"Lenevo Laptop",28000f));

productsList.add(**new** Product(4,"Sony Laptop",28000f));

productsList.add(**new** Product(5,"Apple Laptop",90000f));

// Using Collectors's method to sum the prices.

**double** totalPrice3 = productsList.stream()

.collect(Collectors.*summingDouble*(product->product.price));

System.***out***.println(totalPrice3);

}

}







