

## Lesson 06 Demo 08

### Implementing Comparator with Lambda Expressions

**Objective:** To implement a comparator with lambda expressions in Java to simplify sorting logic and make the code more concise

**Tools Required:** Eclipse IDE

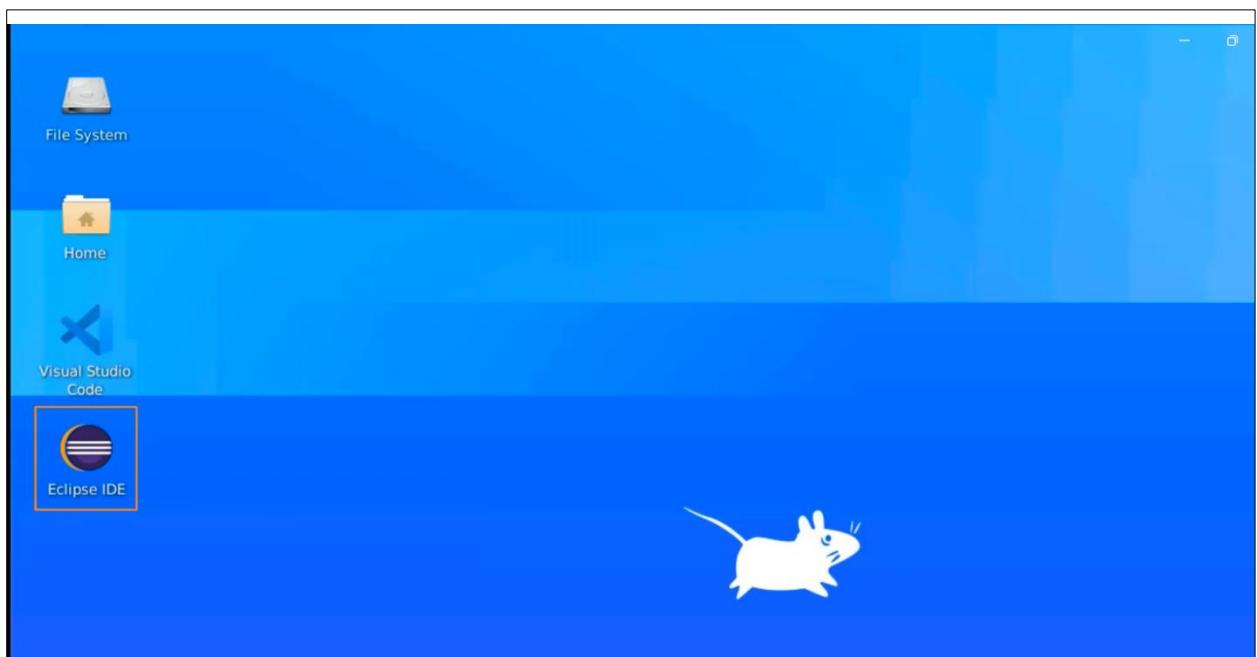
**Prerequisites:** None

Steps to be followed:

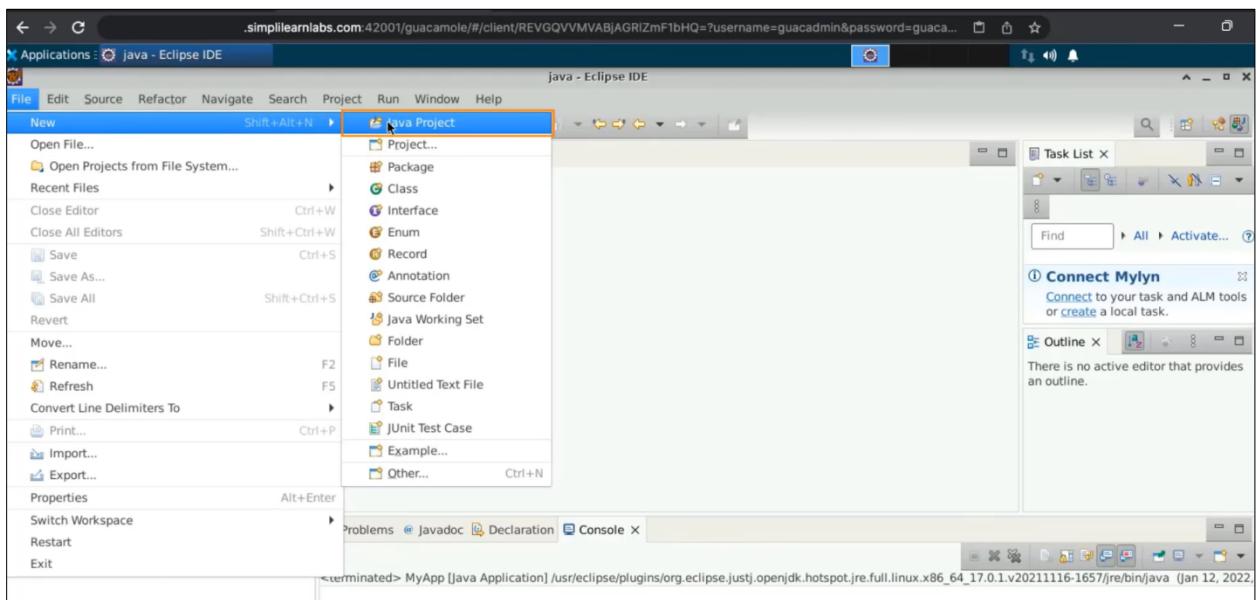
1. Create a new java project
2. Generate constructors with the fields and create the default constructor
3. Create a class and a method that returns an ArrayList
4. Add product objects, with sample data
5. Use the sort method that takes a list as input
6. Create a comparator object and execute the code
7. Write an anonymous class, or a Lambda expression

## Step 1: Create a new java project

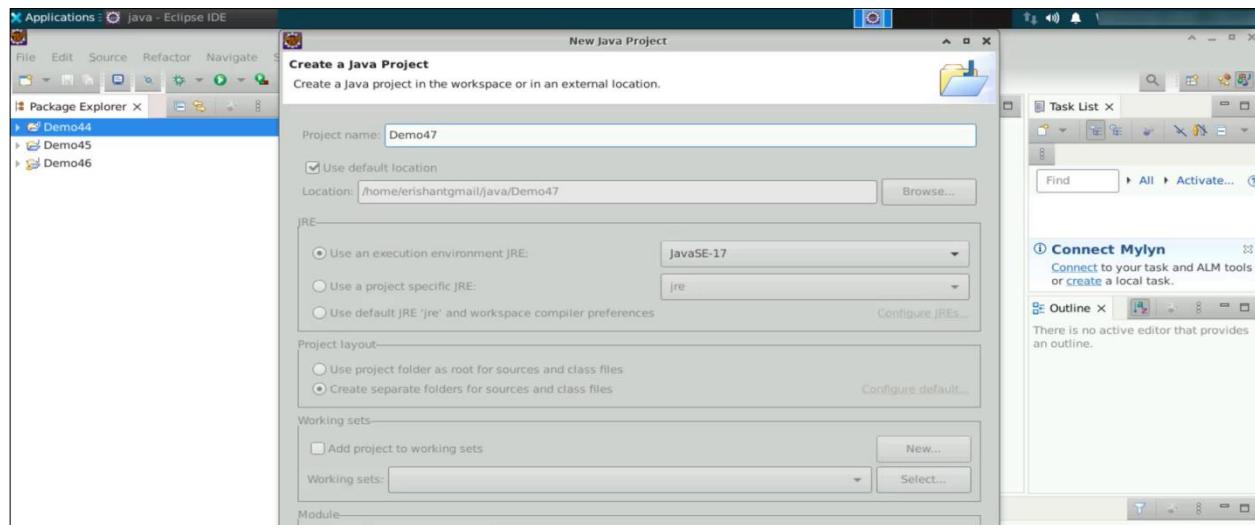
### 1.1 Open the Eclipse IDE



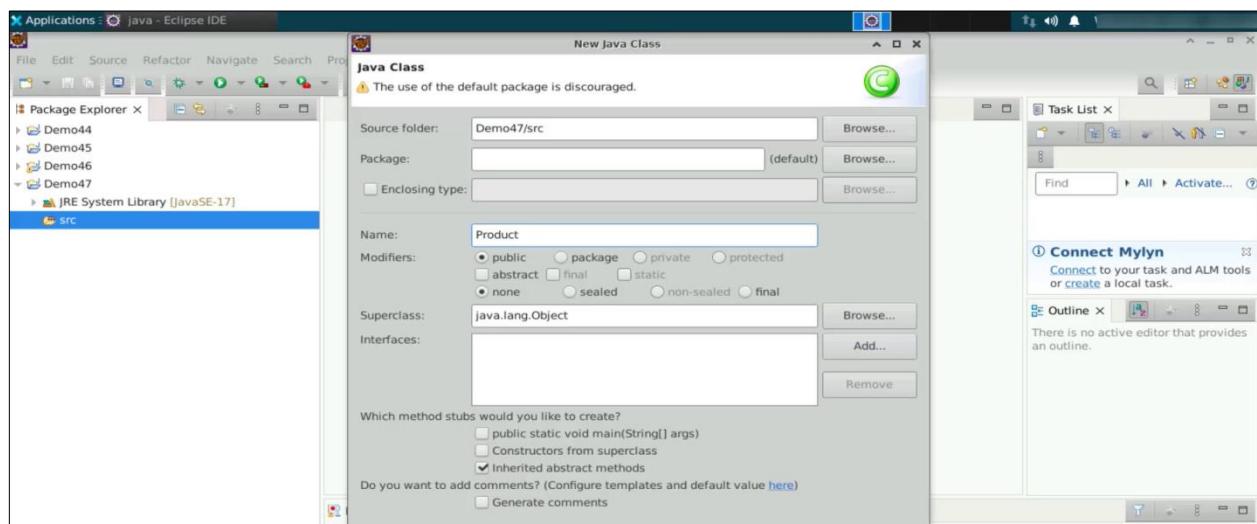
### 1.2 Select File, then New, and click Java project



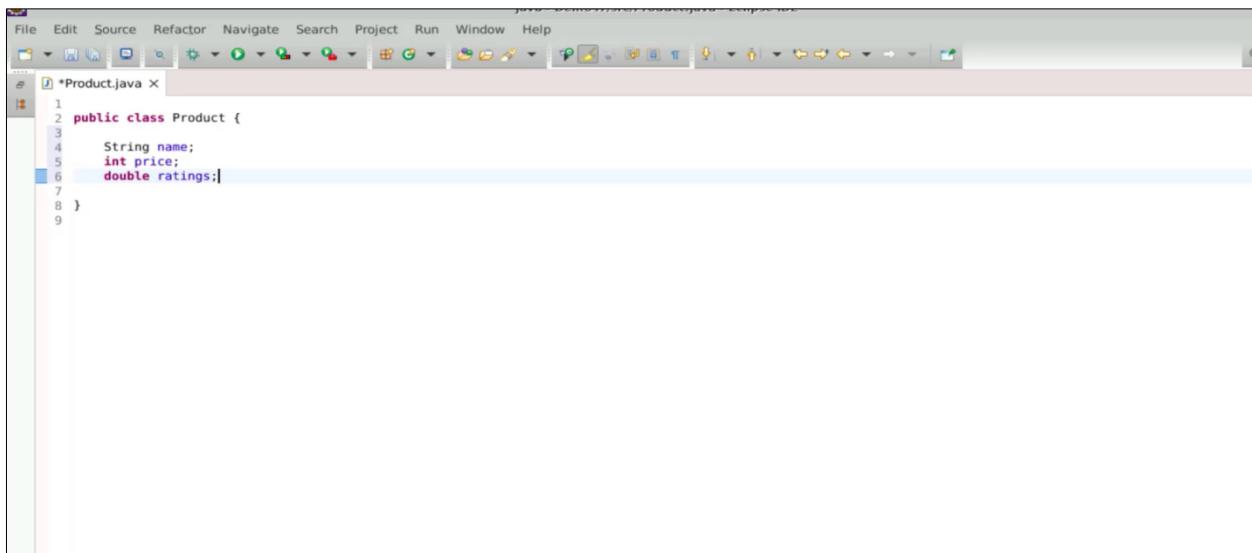
1.3 Enter the project name **Demo47**, uncheck **Create a module-info.java file**, and click **Finish**



1.4 Right-click on the **Demo47** project in the **src** folder and create a new class. Name this class **Product**. Select the main method and click **Finish**



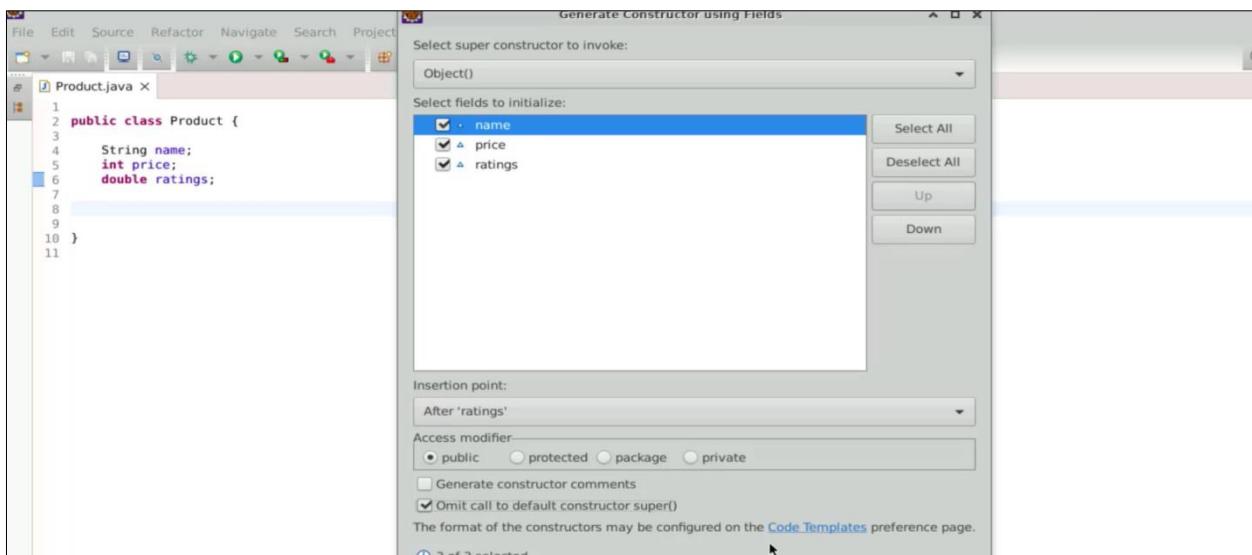
1.5 In the Product class, you will create variables for the product **name**, **price**, and **ratings**



```
File Edit Source Refactor Navigate Search Project Run Window Help
*Product.java X
1
2 public class Product {
3
4     String name;
5     int price;
6     double ratings;
7
8 }
9
```

## Step 2: Generate constructors with the fields and create the default constructor

2.1 Now, generate constructors with the fields and create the default constructor for the Product class



2.2 After completing the above step, navigate to the source code and select **Generate toString()** method. The **toString()** method is used to display the data stored in your Product object. This method defines the structure of your Product object

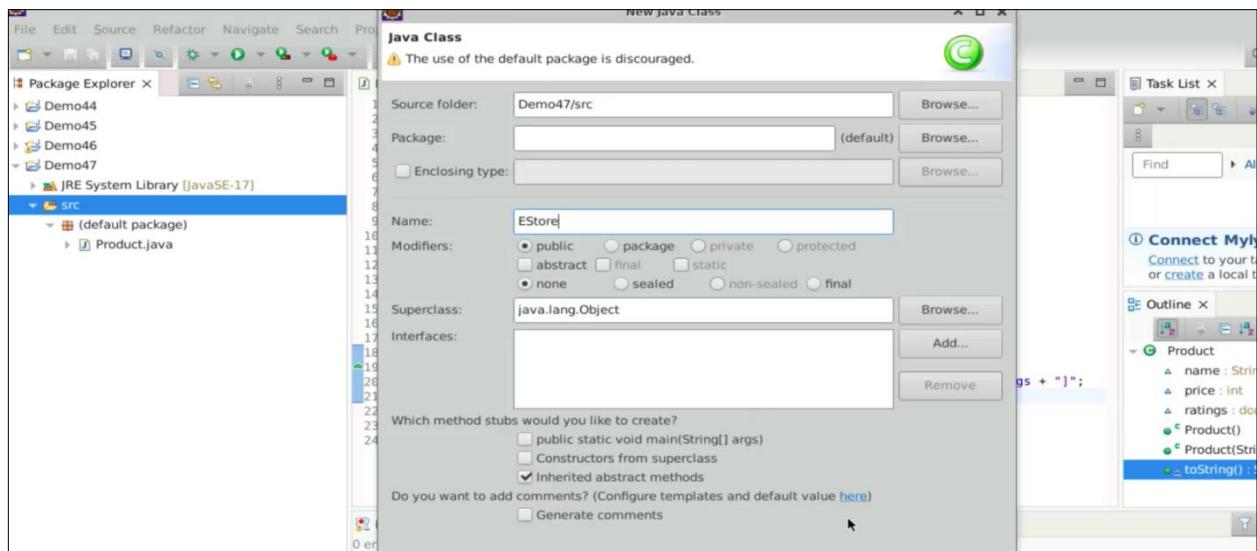
```

1  public class Product {
2
3     String name;
4     int price;
5     double ratings;
6
7     public Product() {
8
9    }
10
11    public Product(String name, int price, double ratings) {
12        this.name = name;
13        this.price = price;
14        this.ratings = ratings;
15    }
16
17    @Override
18    public String toString() {
19        return "Product [name=" + name + ", price=" + price + ", ratings=" + ratings + "]";
20    }
21
22
23 }
24

```

### Step 3: Create a class and a method that returns an ArrayList

#### 3.1 Create another class with name EStore



- 3.2 Within the **EStore** class, create a method that returns an **ArrayList**. This **ArrayList** will be of type **Product**. Name this method **getProductsFromStore**. This method will retrieve an **ArrayList** of **Product** objects. Initialize this **ArrayList** as a new instance

```
File Edit Source Refactor Navigate Search Project Run Window Help
Product.java *EStore.java X
1 import java.util.ArrayList;
2
3 public class EStore {
4
5     ArrayList<Product> getProductsFromStore(){
6
7         ArrayList<Product> products = new ArrayList<Product>();
8
9
10    }
11
12
13 }
14
```

- 3.3 This **ArrayList** will be returned by the **getProductsFromStore** method. To populate the **ArrayList**, add a few **Product** objects directly. For example, you can use the syntax **products.add(new Product("Apple iPhone", 70000, 4.8))**; This adds a new **Product** object with the name **Apple iPhone**, price of **70000**, and ratings of **4.8**.

```
File Edit Source Refactor Navigate Search Project Run Window Help
Product.java *EStore.java X
1 import java.util.ArrayList;
2
3 public class EStore {
4
5     ArrayList<Product> getProductsFromStore(){
6
7         ArrayList<Product> products = new ArrayList<Product>();
8
9         products.add(new Product("Apple iPhone", 70000, 4.8));
10
11     return products;
12
13 }
14
```

## Step 4: Add product objects, with sample data

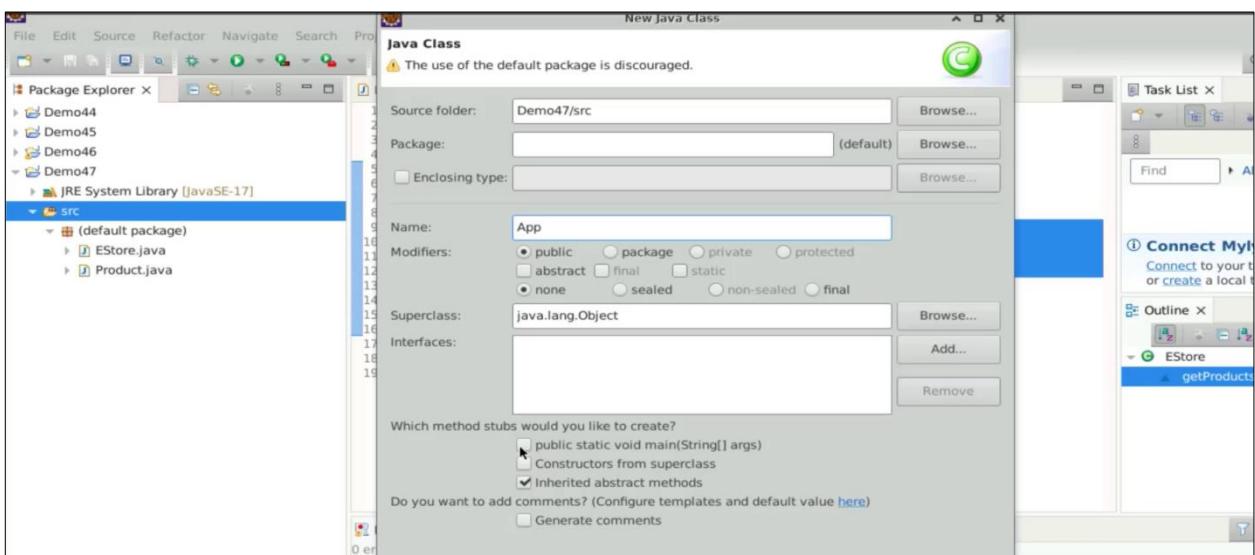
- 4.1 Similarly, you will add a few more product objects. For example, add the following products: **Samsung LED TV, Hitachi Fridge, Skechers, and a MacBook**. You can modify the prices and ratings for these products as needed

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java X
1 import java.util.ArrayList;
2
3 public class EStore {
4
5     ArrayList<Product> getProductsFromStore(){
6
7         ArrayList<Product> products = new ArrayList<Product>();
8
9         products.add(new Product("Apple iPhone", 70000, 4.8));
10        products.add(new Product("Samsung LEDT TV", 50000, 4.2));
11        products.add(new Product("Hitachi Fridge", 65000, 4.0));
12        products.add(new Product("Skechers Shoe", 3000, 4.3));
13        products.add(new Product("Macbook", 120000, 5.0));
14
15    }
16
17    return products;
18
19

```

- 4.2 Create another new Java file as a new class, name this as **App** with the main method



- 4.3 Within the **App** class, retrieve the ArrayList of products from your EStore. Create a new instance of the **EStore** class, Then, use the **getProductsFromStore** method to obtain the ArrayList of products

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
1 import java.util.ArrayList;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         EStore estore = new EStore();
8         ArrayList<Product> products = estore.getProductsFromStore();
9
10        System.out.println("EStore Products");
11        products.forEach((product) -> System.out.println(product));
12
13    }
14
15 }
16

```

- 4.4 Iterate through the product ArrayList and print each product using a for-each loop with a Lambda expression. These are the products available in the **EStore**, displayed exactly as you added them. These are the default products listed in the **EStore**.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
1 import java.util.ArrayList;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         EStore estore = new EStore();
8         ArrayList<Product> products = estore.getProductsFromStore();
9
10        System.out.println("EStore Products");
11        products.forEach((product) -> System.out.println(product));
12
13    }
14
15 }
16

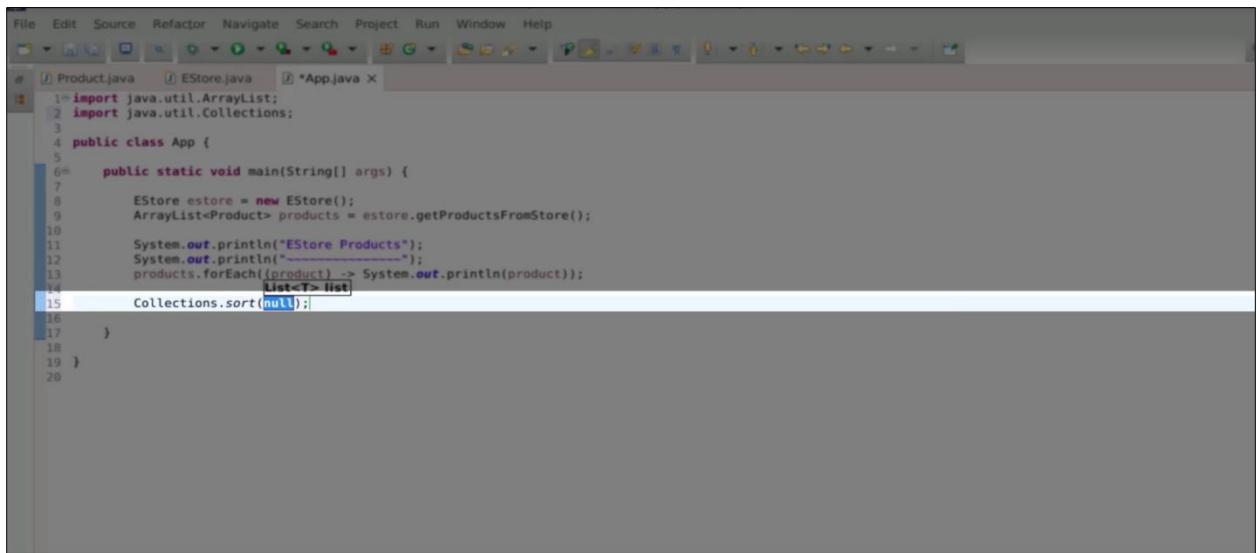
```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justmyjava/EStore Products

| Product | name            | price  | ratings |
|---------|-----------------|--------|---------|
| Product | Apple iPhone    | 70000  | 4.8     |
| Product | Samsung LEDT TV | 50000  | 4.2     |
| Product | Hitachi Fridge  | 65000  | 4.0     |
| Product | Skchechers Shoe | 3000   | 4.3     |
| Product | Macbook         | 120000 | 5.0     |

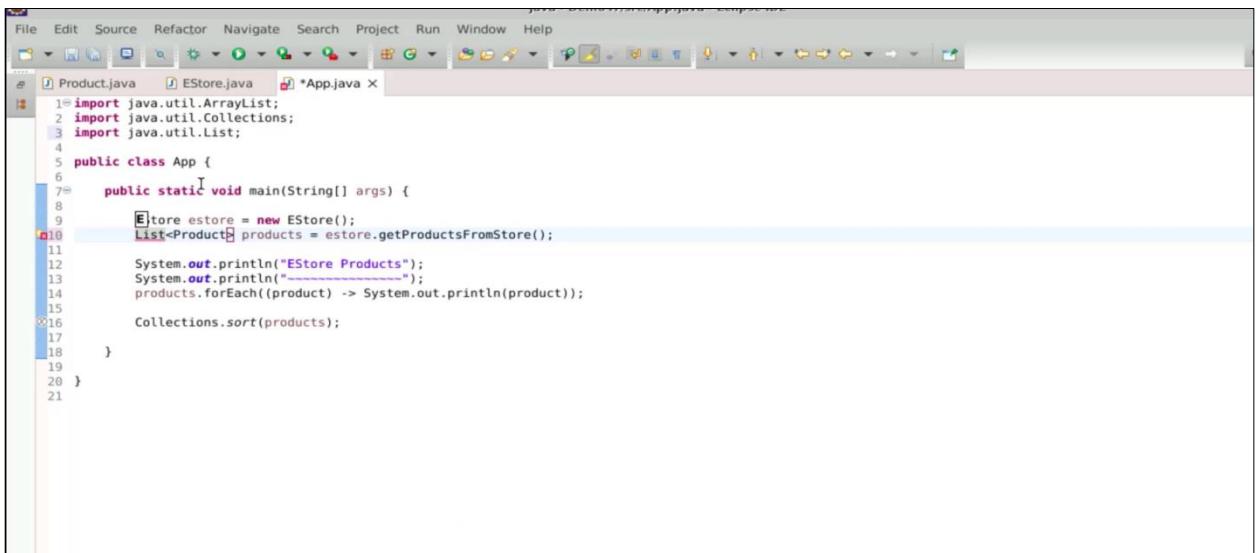
## Step 5: Use the sort method that takes a list as input

5.1 Now, let us say you want to sort the products. To achieve this, you can use the sort method provided by the Collections class. This **sort** method takes a List as input



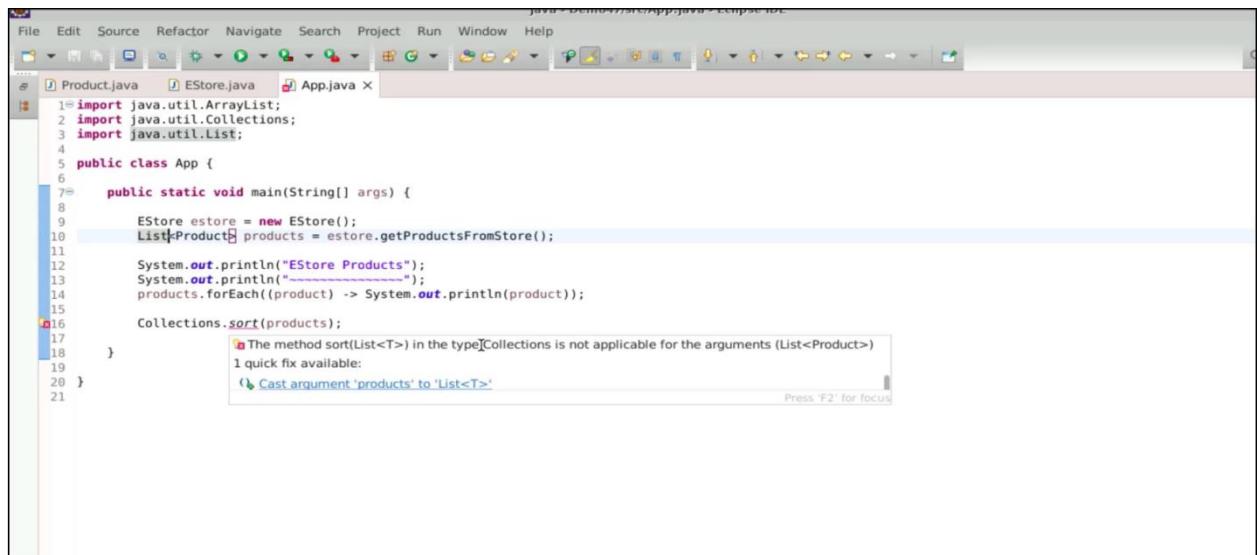
```
File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java *App.java X
1o import java.util.ArrayList;
2 import java.util.Collections;
3
4 public class App {
5
6    public static void main(String[] args) {
7
8        EStore estore = new EStore();
9        ArrayList<Product> products = estore.getProductsFromStore();
10
11        System.out.println("EStore Products");
12        System.out.println("-----");
13        products.forEach((product) -> System.out.println(product));
14
15        Collections.sort(list);
16
17    }
18
19 }
20
```

5.2 When you attempt to pass the products list as input, you might encounter an error. To resolve this, ensure that the products list is of type List.



```
File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java *App.java X
1o import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.List;
4
5 public class App {
6
7    public static void main(String[] args) {
8
9        EStore estore = new EStore();
10       List<Product> products = estore.getProductsFromStore();
11
12       System.out.println("EStore Products");
13       System.out.println("-----");
14       products.forEach((product) -> System.out.println(product));
15
16       Collections.sort(products);
17
18    }
19
20 }
```

- 5.3 Despite changing the list type, the **sort** method might still show an error. The reason for this is that **Collections.sort** is not applicable for a List of type Product.

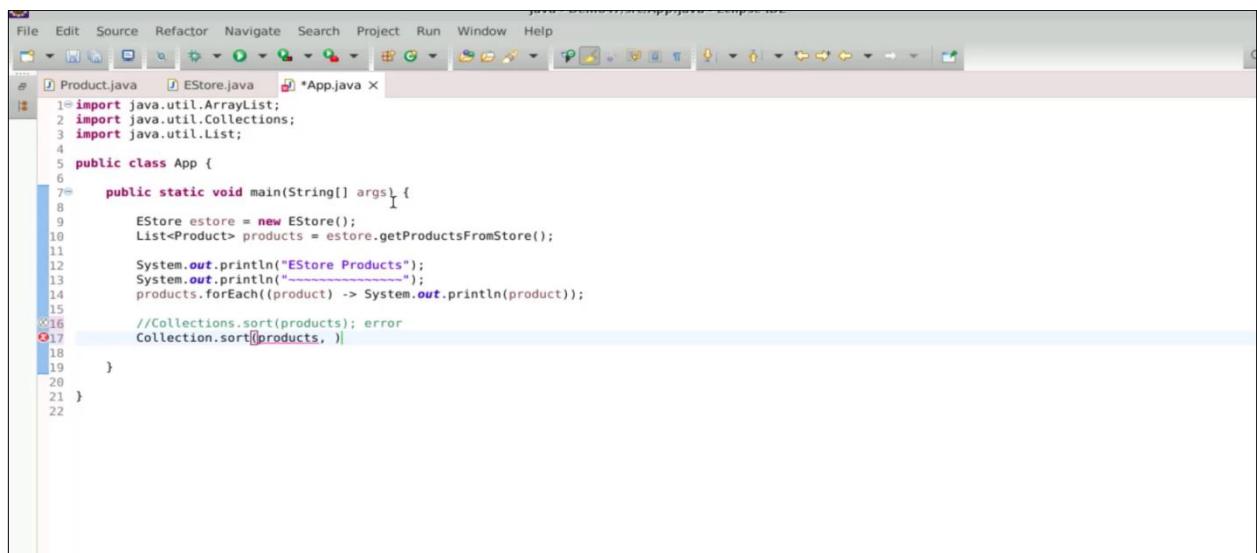


The screenshot shows the Eclipse IDE interface with three tabs open: Product.java, EStore.java, and App.java. The code in App.java is as follows:

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.List;
4
5 public class App {
6
7     public static void main(String[] args) {
8
9         EStore estore = new EStore();
10        List<Product> products = estore.getProductsFromStore();
11
12        System.out.println("EStore Products");
13        System.out.println("-----");
14        products.forEach((product) -> System.out.println(product));
15
16        Collections.sort(products);
17    }
18
19
20 }
```

A red error icon appears next to the call to `Collections.sort(products);`. A tooltip message is displayed: "The method sort(List<T>) in the type Collections is not applicable for the arguments (List<Product>)" followed by "1 quick fix available: Cast argument 'products' to 'List<T>'".

- 5.4 To overcome this issue, you need to use a comparator. Add the appropriate comparator object as an argument to the **Collections.sort** method, along with the products list.



The screenshot shows the Eclipse IDE interface with the same three tabs: Product.java, EStore.java, and App.java. The code in App.java has been modified:

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.List;
4
5 public class App {
6
7     public static void main(String[] args) {
8
9         EStore estore = new EStore();
10        List<Product> products = estore.getProductsFromStore();
11
12        System.out.println("EStore Products");
13        System.out.println("-----");
14        products.forEach((product) -> System.out.println(product));
15
16        //Collections.sort(products); error
17        Collection.sort(products, );
18
19
20 }
21
22 }
```

The error icon and tooltip from the previous screenshot are no longer present, indicating that the code is now valid.

## Step 6: Create a comparator object and execute the code

6.1 Before proceeding, let's create a comparator object. We will use the **Comparator** class from **java.util** with the type **Product**. Create a new comparator, referred to as **comparator01**, as a new instance of Comparator. In the anonymous class for **comparator01**, override the **compare** method to compare two **Product** objects. For sorting based on the name, use **return o1.getName().compareTo(o2.getName());**

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8     public static void main(String[] args) {
9
10         EStore estore = new EStore();
11         List<Product> products = estore.getProductsFromStore();
12
13         System.out.println("EStore Products");
14         System.out.println("-----");
15         products.forEach((product) -> System.out.println(product));
16
17         //Collections.sort(products); error
18
19         Comparator<Product> comparator01 = new Comparator<Product>() {
20             public int compare(Product o1, Product o2) {
21                 return o1.name.compareTo(o2.name);
22             }
23
24
25         Collection.sort(products, )
26
27     }
28
29
30 }
31

```

6.2 Now, pass **comparator01** as an argument to **Collections.sort**. You will notice that the error is resolved. Use a loop to iterate through the sorted list and print the data. Display **EStore products after sorting** to indicate the output.

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8     public static void main(String[] args) {
9
10         EStore estore = new EStore();
11         List<Product> products = estore.getProductsFromStore();
12
13         System.out.println("EStore Products");
14         System.out.println("-----");
15         products.forEach((product) -> System.out.println(product));
16
17         //Collections.sort(products); error
18
19         Comparator<Product> comparator01 = new Comparator<Product>() {
20             public int compare(Product o1, Product o2) {
21                 return o1.name.compareTo(o2.name);
22             }
23
24
25         Collections.sort(products, comparator01);
26
27         System.out.println("EStore Products After Sort");
28         System.out.println("-----");
29         products.forEach((product) -> System.out.println(product));
30
31     }
32

```

6.3 When you run the program, you will see that the data is sorted based on the names.

The products are displayed in the order: Apple, Hitachi, MacBook, Samsung, and Skechers.

Note that products starting with **Sa** come before those starting with **SK**

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8     public static void main(String[] args) {
9
10         EStore estore = new EStore();
11         List<Product> products = estore.getProductsFromStore();
12
13         System.out.println("EStore Products");
14         System.out.println("-----");
15         products.forEach(product -> System.out.println(product));
16
17         //Collections.sort(products); error
18
19         Comparator<Product> comparator1 = new Comparator<Product>() {
20             public int compare(Product o1, Product o2) {
21                 return o1.name.compareTo(o2.name);
22             }
23
24         };
25
26         Collections.sort(products, comparator1);
27
28         System.out.println("EStore Products After Sort");
29         System.out.println("-----");
30         products.forEach(product -> System.out.println(product));
31     }
32

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.oe/EStore Products

Product [name=Apple iPhone, price=70000, ratings=4.8]  
 Product [name=Samsung LEDT TV, price=50000, ratings=4.0]  
 Product [name=Hitachi Fridge, price=65000, ratings=4.0]  
 Product [name=Skechers Shoe, price=3000, ratings=4.3]  
 Product [name=Macbook, price=120000, ratings=5.0]  
 EStore Products After Sort

Product [name=Apple iPhone, price=70000, ratings=4.8]  
 Product [name=Hitachi Fridge, price=65000, ratings=4.0]  
 Product [name=Macbook, price=120000, ratings=5.0]  
 Product [name=Samsung LEDT TV, price=50000, ratings=4.2]  
 Product [name=Skechers Shoe, price=3000, ratings=4.3]

6.4 If you prefer, you can use a lambda expression to define the comparator instead of the anonymous class. Replace **comparator01** with a lambda expression that takes two objects, **o1** and **o2**, and returns **o1.getName().compareTo(o2.getName())**. You can refer to this as **lambdaComparator**

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java *App.java X
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8     public static void main(String[] args) {
9
10         EStore estore = new EStore();
11         List<Product> products = estore.getProductsFromStore();
12
13         System.out.println("EStore Products");
14         System.out.println("-----");
15         products.forEach(product -> System.out.println(product));
16
17         //Collections.sort(products); error
18
19         Comparator<Product> comparator1 = new Comparator<Product>() {
20             public int compare(Product o1, Product o2) {
21                 return o1.name.compareTo(o2.name);
22             }
23
24         };
25
26         Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
27
28         Collections.sort(products, lambdaComparator1);
29
30         System.out.println("EStore Products After Sort");
31         System.out.println("-----");
32         products.forEach(product -> System.out.println(product));
33

```

6.5 Even if you pass **lambdaComparator**, the output will be the same. Running the program will yield a sorted list based on names. This approach simplifies the comparator definition by using a lambda expression

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore Run App App.java X
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8    public static void main(String[] args) {
9
10        EStore estore = new EStore();
11        List<Product> products = estore.getProductsFromStore();
12
13        System.out.println("EStore Products");
14        System.out.println("-----");
15        products.forEach((product) -> System.out.println(product));
16
17        //Collections.sort(products); error
18
19        Comparator<Product> comparator1 = new Comparator<Product>() {
20            public int compare(Product o1, Product o2) {
21                return o1.name.compareTo(o2.name);
22            }
23        };
24
25        Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
26
27        //Collections.sort(products, comparator1);
28        Collections.sort(products, lambdaComparator1);
29
30        System.out.println("EStore Products After Sort");
31        System.out.println("-----");

```

Output:

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.op
EStore Products
-----
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Skchechers Shoe, price=3000, ratings=4.3]
Product [name=Macbook, price=120000, ratings=5.0]
EStore Products After Sort
-----
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Macbook, price=120000, ratings=5.0]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Skchechers Shoe, price=3000, ratings=4.3]

```

6.6 Alternatively, you can further simplify the code by directly using the lambda expression in the **Collections.sort** method. Replace the entire comparator definition with **Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name()));**

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java *App.java X
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8    public static void main(String[] args) {
9
10        EStore estore = new EStore();
11        List<Product> products = estore.getProductsFromStore();
12
13        System.out.println("EStore Products");
14        System.out.println("-----");
15        products.forEach((product) -> System.out.println(product));
16
17        //Collections.sort(products); error
18
19        Comparator<Product> comparator1 = new Comparator<Product>() {
20            public int compare(Product o1, Product o2) {
21                return o1.name.compareTo(o2.name);
22            }
23        };
24
25        Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
26
27        //Collections.sort(products, comparator1);
28        //Collections.sort(products, lambdaComparator1);
29        Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
30
31        System.out.println("EStore Products After Sort");
32        System.out.println("-----");

```

6.7 Run the code, as this is a single line of code with which you can sort your product, so you can see the same output

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8     public static void main(String[] args) {
9
10         Estore estore = new Estore();
11         List<Product> products = estore.getProductsFromStore();
12
13         System.out.println("Estore Products");
14         System.out.println("-----");
15         products.forEach(product -> System.out.println(product));
16
17         //Collections.sort(products); error
18
19         Comparator<Product> comparator1 = new Comparator<Product>() {
20             public int compare(Product o1, Product o2) {
21                 return o1.name.compareTo(o2.name);
22             }
23         };
24
25         Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
26
27         //Collections.sort(products, comparator1);
28         //Collections.sort(products, lambdaComparator1);
29         Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
30
31         System.out.println("Estore Products After Sort");

```

The screenshot shows the Eclipse IDE interface with three open files: Product.java, EStore.java, and App.java. The App.java file contains Java code for sorting a list of products based on their names. The code uses both a traditional anonymous inner class comparator and a lambda expression comparator. The output window displays the sorted list of products, showing them ordered by name.

6.8 If you want to sort based on the price, create another comparator called **comparatorTwo**. In this comparator, return **o1.getPrice() - o2.getPrice()**. This means you are comparing and returning an integer from the **compare** method for the two objects.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Comparator;
4 import java.util.List;
5
6 public class App {
7
8     public static void main(String[] args) {
9
10         Estore estore = new Estore();
11         List<Product> products = estore.getProductsFromStore();
12
13         System.out.println("Estore Products");
14         System.out.println("-----");
15         products.forEach(product -> System.out.println(product));
16
17         //Collections.sort(products); error
18
19         Comparator<Product> comparator1 = new Comparator<Product>() {
20             public int compare(Product o1, Product o2) {
21                 return o1.name.compareTo(o2.name);
22             }
23         };
24
25         Comparator<Product> comparator2 = new Comparator<Product>() {
26             public int compare(Product o1, Product o2) {
27                 return o1.price - o2.price;
28             }
29         };
30
31         Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);

```

This screenshot shows the Eclipse IDE interface with the same three files. The App.java file has been modified to include a second comparator, comparator2, which sorts products by price. The code structure is similar to the previous example but includes the logic for price-based sorting.

6.9 Now you have **comparator2** which works on the criteria of sorting. The criteria to sort will be price, and here on **comparator1** you got criteria of sort as a name

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
6 public class App {
7
8     public static void main(String[] args) {
9
10        EStore estore = new EStore();
11        List<Product> products = estore.getProductsFromStore();
12
13        System.out.println("EStore Products");
14        System.out.println("-----");
15        products.forEach((product) -> System.out.println(product));
16
17        //Collections.sort(products); error
18
19        // Criteria of Sort as Name
20        Comparator<Product> comparator1 = new Comparator<Product>() {
21            public int compare(Product o1, Product o2) {
22                return o1.name.compareTo(o2.name);
23            }
24        };
25
26        // Criteria for sort will be price
27        Comparator<Product> comparator2 = new Comparator<Product>() {
28            public int compare(Product o1, Product o2) {
29                return o1.price - o2.price;
30            }
31        };
32
33        Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34
35        //Collections.sort(products, comparator1);
36        //Collections.sort(products, lambdaComparator1);

```

6.10 Similarly, you can create a lambda expression for **comparator2** by using **LambdaComparator2**. Replace the definition with **(o1, o2) -> o1.getPrice() - o2.getPrice()**

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
10        EStore estore = new EStore();
11        List<Product> products = estore.getProductsFromStore();
12
13        System.out.println("EStore Products");
14        System.out.println("-----");
15        products.forEach((product) -> System.out.println(product));
16
17        //Collections.sort(products); error
18
19        // Criteria of Sort as Name
20        Comparator<Product> comparator1 = new Comparator<Product>() {
21            public int compare(Product o1, Product o2) {
22                return o1.name.compareTo(o2.name);
23            }
24        };
25
26        // Criteria for sort will be price
27        Comparator<Product> comparator2 = new Comparator<Product>() {
28            public int compare(Product o1, Product o2) {
29                return o1.price - o2.price;
30            }
31        };
32
33        Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34        Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36        //Collections.sort(products, comparator1);
37        //Collections.sort(products, lambdaComparator1);
38        Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39
40        System.out.println("EStore Products After Sort");

```

- 6.11 Next, let us write a more sophisticated version to sort the products based on price.  
 Replace the lambda expression with **o1.getPrice() - o2.getPrice()**

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
10     EStore estore = new EStore();
11     List<Product> products = estore.getProductsFromStore();
12
13     System.out.println("EStore Products");
14     System.out.println("-----");
15     products.forEach((product) -> System.out.println(product));
16
17     //Collections.sort(products); error
18
19     // Criteria of Sort as Name
20     Comparator<Product> comparator1 = new Comparator<Product>() {
21         public int compare(Product o1, Product o2) {
22             return o1.name.compareTo(o2.name);
23         }
24     };
25
26     // Criteria for sort will be price
27     Comparator<Product> comparator2 = new Comparator<Product>() {
28         public int compare(Product o1, Product o2) {
29             return o1.price - o2.price;
30         }
31     };
32
33     Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34     Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36     //Collections.sort(products, comparator1);
37     //Collections.sort(products, lambdaComparator1);
38     //Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39     Collections.sort(products, (o1, o2) -> o1.price - o2.price);
40

```

- 6.12 Running the program, you will observe that the product with the lowest price is at the top. The sorting is done in ascending order.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java X
10     EStore estore = new EStore();
11     List<Product> products = estore.getProductsFromStore();
12
13     System.out.println("EStore Products");
14     System.out.println("-----");
15     products.forEach((product) -> System.out.println(product));
16
17     //Collections.sort(products); error
18
19     // Criteria of Sort as Name
20     Comparator<Product> comparator1 = new Comparator<Product>() {
21         public int compare(Product o1, Product o2) {
22             return o1.name.compareTo(o2.name);
23         }
24     };
25
26     // Criteria for sort will be price
27     Comparator<Product> comparator2 = new Comparator<Product>() {
28         public int compare(Product o1, Product o2) {
29             return o1.price - o2.price;
30         }
31     };
32
33     Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34     Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36     //Collections.sort(products, comparator1);
37     //Collections.sort(products, lambdaComparator1);
38     //Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39     Collections.sort(products, (o1, o2) -> o1.price - o2.price);
40

```

Output in Console View:

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justop/EStore Products
-----
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Skcechers Shoe, price=3000, ratings=4.3]
Product [name=Macbook, price=120000, ratings=5.0]
EStore Products After Sort
-----
Product [name=Skcechers Shoe, price=3000, ratings=4.3]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Macbook, price=120000, ratings=5.0]

```

### 6.13 If you return o2.getPrice() - o1.getPrice() instead, the sorting will be in descending order

```

File Edit Source Refactor Navigate Search Project Run Window Help
# Product.java EStore.java *App.java X
10     EStore estore = new EStore();
11     List<Product> products = estore.getProductsFromStore();
12
13     System.out.println("Estore Products");
14     System.out.println("-----");
15     products.forEach((product) -> System.out.println(product));
16
17     //Collections.sort(products); error
18
19     // Criteria of Sort as Name
20     Comparator<Product> comparator1 = new Comparator<Product>() {
21         public int compare(Product o1, Product o2) {
22             return o1.name.compareTo(o2.name);
23         }
24     };
25
26     // Criteria for sort will be price
27     Comparator<Product> comparator2 = new Comparator<Product>() {
28         public int compare(Product o1, Product o2) {
29             return o1.price - o2.price;
30         }
31     };
32
33     Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34     Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36     //Collections.sort(products, comparator1);
37     //Collections.sort(products, lambdaComparator1);
38     //Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39     Collections.sort(products, (o1, o2) -> o2.price - o1.price);
40

```

### 6.14 Running the code again, you will see that the sorting is done in descending order. The sequence of objects is reversed.

```

File Edit Source Refactor Navigate Search Project Run Window Help
# Product.java EStore Run App *App.java X
10     EStore estore = new EStore();
11     List<Product> products = estore.getProductsFromStore();
12
13     System.out.println("Estore Products");
14     System.out.println("-----");
15     products.forEach((product) -> System.out.println(product));
16
17     //Collections.sort(products); error
18
19     // Criteria of Sort as Name
20     Comparator<Product> comparator1 = new Comparator<Product>() {
21         public int compare(Product o1, Product o2) {
22             return o1.name.compareTo(o2.name);
23         }
24     };
25
26     // Criteria for sort will be price
27     Comparator<Product> comparator2 = new Comparator<Product>() {
28         public int compare(Product o1, Product o2) {
29             return o1.price - o2.price;
30         }
31     };
32
33     Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34     Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36     //Collections.sort(products, comparator1);
37     //Collections.sort(products, lambdaComparator1);
38     //Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39     Collections.sort(products, (o1, o2) -> o2.price - o1.price);
40

```

Console Output:

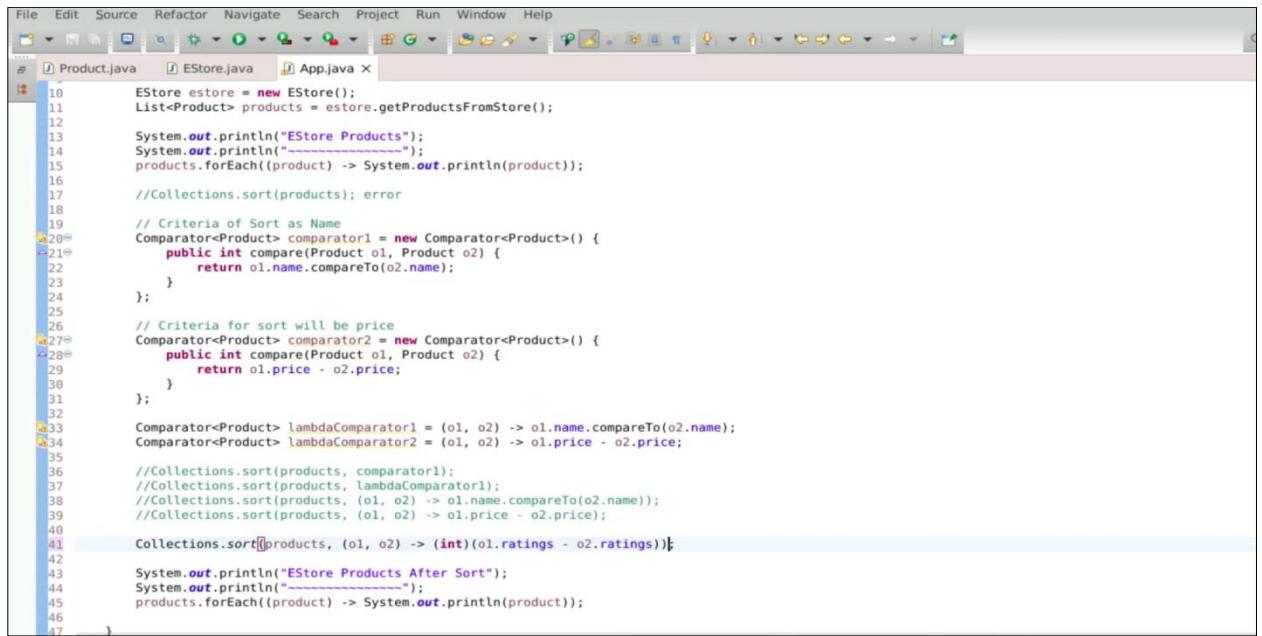
```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.console
EStore Products
-----
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Skechers Shoe, price=3000, ratings=4.3]
Product [name=Macbook, price=120000, ratings=5.0]
EStore Products After Sort
-----
Product [name=Macbook, price=120000, ratings=5.0]
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Skechers Shoe, price=3000, ratings=4.3]

```

## Step 7: Write an anonymous class, or a Lambda expression

7.1 Now, to sort the products based on ratings, modify the code to use **ratings** instead of **price** or **name**. However, when subtracting the ratings, you need to cast the result as an integer.



```
File Edit Source Refactor Navigate Search Project Run Window Help
Product.java EStore.java App.java
10     Estore estore = new Estore();
11     List<Product> products = estore.getProductsFromStore();
12
13     System.out.println("EStore Products");
14     System.out.println("-----");
15     products.forEach(product -> System.out.println(product));
16
17     //Collections.sort(products); error
18
19     // Criteria of Sort as Name
20     Comparator<Product> comparator1 = new Comparator<Product>() {
21         public int compare(Product o1, Product o2) {
22             return o1.name.compareTo(o2.name);
23         }
24     };
25
26     // Criteria for sort will be price
27     Comparator<Product> comparator2 = new Comparator<Product>() {
28         public int compare(Product o1, Product o2) {
29             return o1.price - o2.price;
30         }
31     };
32
33     Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34     Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36     //Collections.sort(products, comparator1);
37     //Collections.sort(products, lambdaComparator1);
38     //Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39     //Collections.sort(products, (o1, o2) -> o1.price - o2.price);
40
41     Collections.sort(products, (o1, o2) -> (int)(o1.ratings - o2.ratings));
42
43     System.out.println("EStore Products After Sort");
44     System.out.println("-----");
45     products.forEach(product -> System.out.println(product));
46
47 }
```

7.2 Run the code again. Now, you will see the data sorted based on the ratings of the products

The screenshot shows the Eclipse IDE interface with the Product.java file open in the editor. The code implements sorting logic for a list of products based on name and price using Comparator and lambda expressions. The output window shows the original unsorted data followed by the sorted data after the sort operation.

```
File Edit Source Refactor Navigate Search Project Run Window Help
# Product.java EStore.java App.java X
10     EStore estore = new EStore();
11     List<Product> products = estore.getProductsFromStore();
12
13     System.out.println("EStore Products");
14     System.out.println("-----");
15     products.forEach(product -> System.out.println(product));
16
17     //Collections.sort(products); error
18
19     // Criteria of Sort as Name
20     Comparator<Product> comparator1 = new Comparator<Product>() {
21         public int compare(Product o1, Product o2) {
22             return o1.name.compareTo(o2.name);
23         }
24     };
25
26     // Criteria for sort will be price
27     Comparator<Product> comparator2 = new Comparator<Product>() {
28         public int compare(Product o1, Product o2) {
29             return o1.price - o2.price;
30         }
31     };
32
33     Comparator<Product> lambdaComparator1 = (o1, o2) -> o1.name.compareTo(o2.name);
34     Comparator<Product> lambdaComparator2 = (o1, o2) -> o1.price - o2.price;
35
36     //Collections.sort(products, comparator1);
37     //Collections.sort(products, lambdaComparator1);
38     //Collections.sort(products, (o1, o2) -> o1.name.compareTo(o2.name));
39     //Collections.sort(products, (o1, o2) -> o1.price - o2.price);
40
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.op
EStore Products
-----
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Skcechers Shoe, price=3000, ratings=4.3]
Product [name=Macbook, price=120000, ratings=5.0]
EStore Products After Sort
-----
Product [name=Apple iPhone, price=70000, ratings=4.8]
Product [name=Samsung LEDT TV, price=50000, ratings=4.2]
Product [name=Hitachi Fridge, price=65000, ratings=4.0]
Product [name=Skcechers Shoe, price=3000, ratings=4.3]
Product [name=Macbook, price=120000, ratings=5.0]
```

By following these steps, you have successfully implemented a comparator with lambda expressions in Java to simplify sorting logic and make the code more concise.