Copilot

Using `Comparator` with Java Streams can be very powerful in real-time scenarios. Here are some practical use cases and examples:

1. Sorting Employee Records by Salary

In a company, you might need to sort employee records by their salary to generate payroll reports.

```
List<Employee> sortedBySalary = employees.stream()
    .sorted(Comparator.comparingDouble(Employee::getSalary))
    .collect(Collectors.toList());
```

2. Sorting Products by Price in an E-commerce Application

An e-commerce platform might need to sort products by price to display them in ascending or descending order.

```
List<Product> sortedByPrice = products.stream()
    .sorted(Comparator.comparingDouble(Product::getPrice))
    .collect(Collectors.toList());
```

3. Sorting Transactions by Date

In a banking application, sorting transactions by date can help in generating account statements.

```
List<Transaction> sortedByDate = transactions.stream()
    .sorted(Comparator.comparing(Transaction::getDate))
    .collect(Collectors.toList());
```

4. Finding the Highest and Lowest Scores in a Game

In a gaming application, you might want to find the highest and lowest scores.

```
Optional<Player> highestScore = players.stream()
    .max(Comparator.comparingInt(Player::getScore));

Optional<Player> lowestScore = players.stream()
    .min(Comparator.comparingInt(Player::getScore));
```

5. Sorting Orders by Customer Name

In a retail application, sorting orders by customer name can help in organizing delivery schedules.

```
List<Order> sortedByCustomerName = orders.stream()
    .sorted(Comparator.comparing(Order::getCustomerName))
    .collect(Collectors.toList());
```

6. Prioritizing Tasks by Deadline

In a task management application, sorting tasks by their deadlines can help in prioritizing work.

```
List<Task> sortedByDeadline = tasks.stream()
    .sorted(Comparator.comparing(Task::getDeadline))
    .collect(Collectors.toList());
```

7. Sorting Events by Start Time

In an event management system, sorting events by their start time can help in scheduling.

```
List<Event> sortedByStartTime = events.stream()
    .sorted(Comparator.comparing(Event::getStartTime))
    .collect(Collectors.toList());
```

8. Sorting Students by Grades

In an educational application, sorting students by their grades can help in generating report cards.

```
List<Student> sortedByGrades = students.stream()
    .sorted(Comparator.comparingDouble(Student::getGrade))
    .collect(Collectors.toList());
```

9. Sorting Files by Size

In a file management system, sorting files by their size can help in managing storage.

```
List<File> sortedBySize = files.stream()
    .sorted(Comparator.comparingLong(File::getSize))
    .collect(Collectors.toList());
```

10. Sorting Books by Title

In a library management system, sorting books by their title can help in organizing the catalog.

```
List<Book> sortedByTitle = books.stream()
    .sorted(Comparator.comparing(Book::getTitle))
    .collect(Collectors.toList());
```

These examples demonstrate how `Comparator` can be used with Java Streams to handle various real-time scenarios effectively. If you have a specific scenario in mind, feel free to share, and I can provide a more tailored example!