

Name:	Snigdha Ghosh
UID:	23ICS10004
Session:	622-A

Experiment 2.2 –

PartA –

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

```
class Employee {
```

```
    String name;
```

```
    int age;
```

```
    double salary;
```

```
    Employee(String name, int  
age, double salary) {
```

```
        this.name = name; this.age  
= age; this.salary = salary;  
    }
```

```
@Override
```

```
public String toString() {
```

```
        return name + " | Age: " +  
age + " | Salary: " + salary;  
    }  
}
```

// Main class must be named
Main in Programiz

```
public class Main {  
  
    public static void  
main(String[] args) {  
  
        List<Employee> employees  
= Arrays.asList(  
  
            new Employee("John",  
28, 50000),  
  
            new Employee("Alice",  
24, 60000),  
  
            new Employee("Bob", 30,  
45000)  
  
        );  
  

```

```
System.out.println("\nSorted by  
Name:");
```

```
        employees.stream()  
  
            .sorted((e1, e2) ->  
e1.name.compareTo(e2.name))
```

```
.forEach(System.out::println);
```

```
System.out.println("\nSorted by  
Age:");
```

```
    employees.stream()  
        .sorted((e1, e2) ->  
Integer.compare(e1.age,  
e2.age))
```

```
.forEach(System.out::println);
```

```
System.out.println("\nSorted by  
Salary (Descending):");
```

```
    employees.stream()  
        .sorted((e1, e2) ->  
Double.compare(e2.salary,  
e1.salary))
```

```
.forEach(System.out::println);
```

```
    }
```

```
}
```

Output

```
Sorted by Name:
Alice | Age: 24 | Salary: 60000.0
Bob | Age: 30 | Salary: 45000.0
John | Age: 28 | Salary: 50000.0

Sorted by Age:
Alice | Age: 24 | Salary: 60000.0
John | Age: 28 | Salary: 50000.0
Bob | Age: 30 | Salary: 45000.0

Sorted by Salary (Descending):
Alice | Age: 24 | Salary: 60000.0
John | Age: 28 | Salary: 50000.0
Bob | Age: 30 | Salary: 45000.0

=== Code Execution Successful ===
```

PART B –

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

```
class Student {
```

```
    String name;
```

```
    double marks;
```

```
    Student(String name, double
marks) {
```

```
        this.name = name;
```

```
this.marks = marks;
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
        return name + " | Marks: "
+ marks;
    }
}
```

// Must be Main in Programiz

```
public class Main {
    public static void
main(String[] args) {
    List<Student> students =
Arrays.asList(
        new Student("Ravi", 85),
        new Student("Sneha",
72),
        new Student("Amit", 90),
        new Student("Priya", 65)
    );
}
```

```
System.out.println("Students
with marks > 75 sorted by
marks:");
```

```
    students.stream()
        .filter(s -> s.marks > 75)
```

```
        .sorted((s1, s2) ->
Double.compare(s1.marks,
s2.marks))

        .map(s -> s.name)

.forEach(System.out::println);
    }
}
```

Output

```
Students with marks > 75 sorted by marks:
Ravi
Amit

=== Code Execution Successful ===
```

PART C –

```
import java.util.*;
import java.util.stream.*;
import java.util.Comparator;

class Product {
    String name;
    double price;
    String category;
```

```
Product(String name, double price, String category) {  
    this.name = name; this.price = price; this.category = category;  
}
```

```
@Override
```

```
public String toString() {  
    return name + " | " + category + " | $" + price;  
}  
}
```

```
// Must be Main for Programiz
```

```
public class Main {  
    public static void main(String[] args) {  
        List<Product> products = Arrays.asList(  
            new Product("Laptop", 75000, "Electronics"),  
            new Product("Phone", 50000, "Electronics"),  
            new Product("Shirt", 1500, "Clothing"),  
            new Product("Jeans", 2500, "Clothing"),  
            new Product("Fridge", 30000, "Electronics")  
        );  
  
        // Group by category  
        System.out.println("Products grouped by category:");  
        Map<String, List<Product>> grouped = products.stream()  
            .collect(Collectors.groupingBy(p -> p.category));  
        grouped.forEach((cat, list) -> System.out.println(cat + " -> " + list));  
    }  
}
```

```

// Most expensive product in each category
System.out.println("\nMost expensive product in each category:");
Map<String, Optional<Product>> expensive = products.stream()
    .collect(Collectors.groupingBy(
        p -> p.category,
        Collectors.maxBy(Comparator.comparingDouble(p -> p.price))
    ));
expensive.forEach((cat, prod) -> System.out.println(cat + " -> " + prod.get()));

// Average price of all products
double avgPrice = products.stream()
    .collect(Collectors.averagingDouble(p -> p.price));
System.out.println("\nAverage Price of All Products: " + avgPrice);
}
}

```

Output

Clear

Products grouped by category:

Clothing -> [Shirt | Clothing | \$1500.0, Jeans | Clothing | \$2500.0]

Electronics -> [Laptop | Electronics | \$75000.0, Phone | Electronics | \$50000.0, Fridge | Electronics | \$30000.0]

Most expensive product in each category:

Clothing -> Jeans | Clothing | \$2500.0

Electronics -> Laptop | Electronics | \$75000.0

Average Price of All Products: 31800.0

=== Code Execution Successful ===