equals (Object obj) – Object (Java SE 21 & JDK 21)

Compare two objects to check for the equality

Its default implementation checks if the references are the same(==)

We can give custom implementation by overriding this method(overriding to compare the values(content))

```
package equalsEx;

public class Employee {
    int id;
    String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public boolean equals(Object obj){
        if (this == obj)  return true;// same reference check
        if (obj == null || getClass() != obj.getClass() )
return false;

        Employee employee = (Employee) obj;
        return id == employee.id &&
name.equals(employee.name);
    }

    public static void main(String[] args) {
        Employee emp1 = new Employee(101,"Krishna");
        Employee emp2 = new Employee(101, "Krishna");

        System.out.println("Are they equal: "+
emp1.equals(emp2));
    }
}
```

o/p:

Are they equal: true

Comparable & Comparator

These are interfaces which are used for sorting objects.

Comparable - java.lang - Comparable (Java SE 21 & JDK 21)

Comparator - java.util - Comparator (Java SE 21 & JDK 21)

- Comparable:
  It is used to define natural ordering of objects.
  Natural Ordering:
  - String: alphabetical order(A-Z, a-z)
  - Numbers: ascending order(1,2,3,4,……)
- Comparator
  It is used to define custom ordering

1. Comparable:
   - From java.lang
   - It has a single method compareTo method, which is used to define natural order of objects.

Syntax:

Public class Employee implements Comparable<Employee>{

@Override

public int compareTo(Employee obj){

// comparison logic

}

}

```
package comparableExample;

public class Employee implements Comparable<Employee>{
    private String name;
    private int age;

    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Employee o) {
        return this.age - o.age;  // Natural sorting with age
    }
}
```

```
package comparableExample;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        employees.add(new Employee("Krishna", 30));
        employees.add(new Employee("Gopal",20));
        employees.add(new Employee("Govind", 50));

        Collections.sort(employees);

        for (Employee emp: employees){
            System.out.println(emp);
        }
    }
}
```

O/P:

Employee{name='Gopal', age=20}

Employee{name='Krishna', age=30}

```
Employee{name='Govind', age=50}
```

```java
@Override
public int compareTo(Employee o) {
    return this.age - o.age;  // Natural sorting with age
}
```

The compareTo() method compares the current object(this) with the specified object(o)

- 0 -> If both objects are equal
- + -> if the current object is greater
- - -> the current object is smaller

1. If this.age is less than o.age, the result will be negative(indicating this comes before o)
2. If this.age is greater than o.age, the result will be positive(indicating this comes after o)
3. If this.age equals o.age, the result will be 0 (both are equal)

Comparator

- From java.util package
- It provides compare method
- It allows custom ordering of objects

SYNTAX:

```
public class MyComparator implements
Comparator<MyClass> {
@override
Public int compare(MyClass obj1, MyClas obj2){
//comparison logic
}
```

```
}


package comparableExample;

public class Employee{
    private String name;
    private int age;



    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
```

```
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }


}
```

```
package comparableExample;

import java.util.Comparator;

public class NameComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee emp1, Employee emp2) {
        return emp1.getName().compareTo(emp2.getName());
//sorting by their names
    }
}
```

```
package comparableExample;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
    public static void main(String[] args) {
```

```
        List<Employee> employees = new ArrayList<>();

        employees.add(new Employee("Krishna", 30));
        employees.add(new Employee("Adarsh",20));
        employees.add(new Employee("Bishal", 50));

        Collections.sort(employees,new NameComparator());
//using comparator

        for (Employee emp: employees){
            System.out.println(emp);
        }
    }
}
```

o/p:

Employee{name='Adarsh', age=20}

Employee{name='Bishal', age=50}

Employee{name='Krishna', age=30}

TASK:

Student Class

Attributes: name, rollNo, marks

Implement comparable to sort students by rollNo and use comparator to sort students by marks (in descending order)

TASK:

Book Class: title, author, price

Implement comparable to sort book by title

Create comparator to sort book by price

Threads*