**package** topic3\_1;

/\*\*

\*

\* The compareTo interface represents a contract for objects that can be compared.

\* Classes implementing this interface should provide an implementation of the compareTo method

\* to define the natural ordering of objects.

\*

\*/

**public** **interface** compareTo {

}

**package** topic3\_1;

/\*\*

\*

\*The Person class represents a person with a first name and a last name.

\*It implements the PersonInterface and Comparable interfaces.

\*/

**public** **class** Person **implements** PersonInterface, Comparable<Person> {

**private** String firstName = "Mark";

**private** String lastName = "Reha";

**private** **boolean** running;

/\*\*

\* Constructs a Person object with the specified first name and last name.

\*

\* **@param** firstName, the first name of the person

\* **@param** lastName, the last name of the person

\*/

**public** Person(String firstName, String lastName)

{

**this**.firstName = firstName;

**this**.lastName = lastName;

}

/\*\*

\* Constructs a Person object by copying the values from another Person object.

\*

\* **@param** person, The Person object to copy from

\*/

**public** Person(Person person)

{

**this**.firstName = person.getFirstName();

**this**.lastName = person.getLastName();

}

/\*\*

\* Gets the first name of the person.

\*

\* **@return** the first name

\*/

**public** String getFirstName()

{

**return** firstName;

}

/\*\*

\* Gets the last name of the person.

\*

\* **@return** the last name

\*/

**public** String getLastName()

{

**return** lastName;

}

@Override

**public** **boolean** equals(Object other)

{

**if**(other == **this**)

{

System.***out***.println("I am here in other == this");

**return** **true**;

}

**if**(other == **null**)

{

System.***out***.println("I am here in other == null");

**return** **false**;

}

**if**(getClass() != other.getClass())

{

System.***out***.println("I am here in getClass() != other.getClass()");

**return** **false**;

}

Person person = (Person)other;

**return**(**this**.firstName == person.firstName && **this**.lastName == person.lastName);

}

@Override

**public** String toString()

{

**return** "My classs is " + getClass() + " " + **this**.firstName + " " + **this**.lastName;

}

@Override

**public** **void** walk() {

System.***out***.println("I am walking");

running = **false**;

}

@Override

**public** **void** run() {

System.***out***.println("i am running");

running = **true**;

}

@Override

**public** **boolean** isRunning() {

**return** running;

}

@Override

**public** **int** compareTo(Person p) {

**int** value = **this**.lastName.compareTo(p.firstName);

**if**(value == 0)

{

**return** **this**.firstName.compareTo(p.firstName);

}

**else**

{

**return** value;

}

}

}

**package** topic3\_1;

**public** **interface** PersonInterface {

**public** **void** walk();

**public** **void** run();

**public** **boolean** isRunning();

}

**package** topic3\_1;

**import** java.util.Arrays;

/\*\*

\*

\* This class is used to test the functionality of the Person class.

\*

\*/

**public** **class** Test{

/\*\*

\* The main method is the entry point of the program

\*

\* **@param** args, command- line arguments

\*/

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

//Create new Person Objects

Person person1 = **new** Person("Justine", "Reha");

Person person2 = **new** Person("Brianna", "Reha");

Person person3 = **new** Person(person1);

//Test Object equality

**if**(person1 == person2)

System.***out***.println("These persons are identical using ==");

**else**

System.***out***.println("These persons are not identical using ==");

//Test Object equality

**if**(person1.equals(person2))

System.***out***.println("These persons are identical using equals()");

**else**

System.***out***.println("These persons are not identical using equals()");

//Test copy Constructor

**if**(person1.equals(person3))

System.***out***.println("These copied persons are identical using equals()");

**else**

System.***out***.println("These copied persons are not identical using equals()");

//print the Objects

System.***out***.println(person1);

System.***out***.println(person2.toString());

System.***out***.println(person3);

// make a Person walk and run

person1.walk();

person1.run();

System.***out***.println("Person 1 is running: " + person1.isRunning());

person1.walk();

System.***out***.println("Person 1 is running: " + person1.isRunning());

}

//Create a bunch of Persons and compare them so they are sorted on Last Name

Person[] persons = **new** Person[4];

{

persons[0] = **new** Person("Justine", "Reha");

persons[1] = **new** Person("Brianna", "Reha");

persons[2] = **new** Person("Mary", "Reha");

persons[3] = **new** Person("Mark", "Reha");

Arrays.*sort*(persons);

**for**(**int** x=0; x < 4; ++x)

{

System.***out***.println(persons[x]);

}

}

}

/\*\*

\* Notes----

\* -----how and why the output was displayed------

\* The output confirms that the == operator compares object references, while the equals() method compares the values of the class member variables.

\* The overridden toString() method provides a convenient string representation of the Person objects when printing them.

\*

\*---how both the equals() and toString() methods---

\* the toString() method is automatically called to obtain the string representation of the object.

\* The equals() method is used for comparing Person objects. When you use the equals() method to compare two Person objects, it will compare their firstName and lastName values to determine if they are equal.

\* The updated code ensures that the equals() and toString() methods are overridden in the Person class, providing customized behavior for equality comparison and string representation of Person object

\*

\* -----@Override annotation is used for and why it is good practice to add this to overridden methods-----

\* The **@Override** annotation is used in Java to indicate that a method in a subclass is intended to override a method in its superclass.

\* It is good practice to add the **@Override** annotation to overridden methods because it serves as a compile-time check, ensuring that the method signature in the subclass matches the method being overridden in the superclass.

\* If the method signature doesn't match, the compiler will generate an error, alerting you to the issue.

\* This helps prevent accidental mistakes and ensures that your code behaves as intended when overriding methods.

\*/