K DEEPAK **ASSIGNMENT-(DAY-26)**

DAY – 19 CORE JAVA

Task 1: Generics and Type Safety:

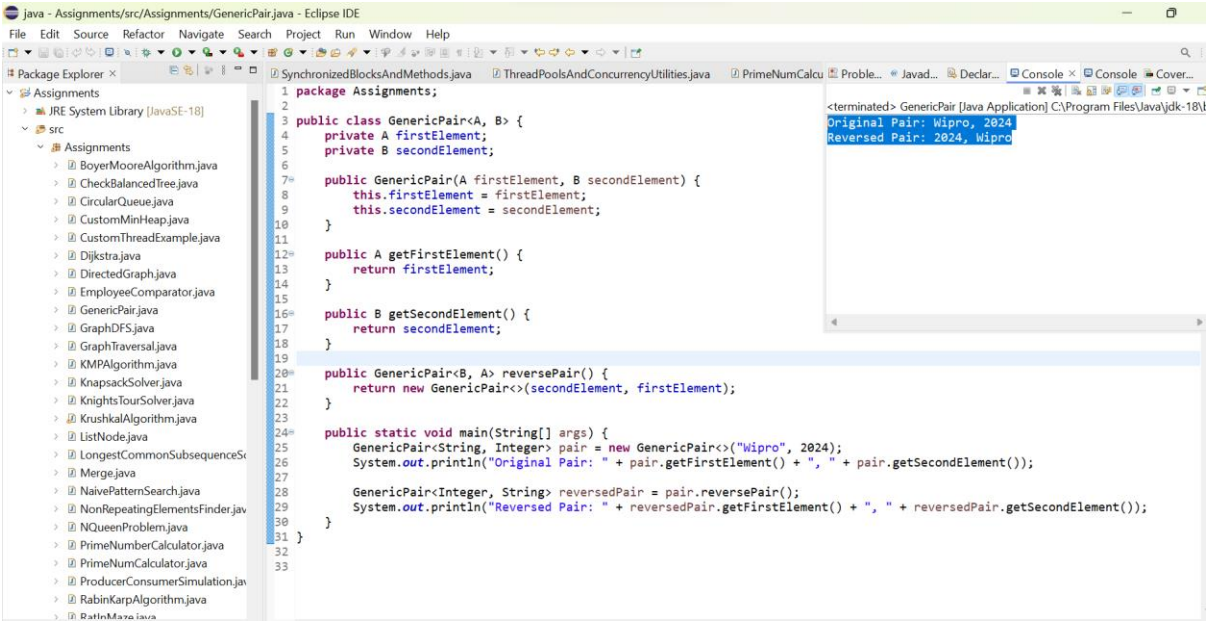
Create a generic Pair class that holds two objects of different types, and write a method to return a reversed version of the pair.

Code:

```
package Assignments;

public class GenericPair<A, B> {
    private A firstElement;
    private B secondElement;

    public GenericPair(A firstElement, B secondElement) {
        this.firstElement = firstElement;
        this.secondElement = secondElement;
    }

    public A getFirstElement() {
        return firstElement;
    }

    public B getSecondElement() {
        return secondElement;
    }

    public GenericPair<B, A> reversePair() {
        return new GenericPair<>(secondElement, firstElement);
    }

    public static void main(String[] args) {
        GenericPair<String, Integer> pair = new GenericPair<>("Wipro", 2024);
        System.out.println("Original Pair: " + pair.getFirstElement() + ", " +
pair.getSecondElement());

        GenericPair<Integer, String> reversedPair = pair.reversePair();
        System.out.println("Reversed Pair: " + reversedPair.getFirstElement() + ",
" + reversedPair.getSecondElement());
    }
}
```

Output:

```
Original Pair: Wipro, 2024
Reversed Pair: 2024, Wipro
```



Task 2: Generic Classes and Methods:

Implement a generic method that swaps the positions of two elements in an array, regardless of their type, and demonstrate its usage with different object types.

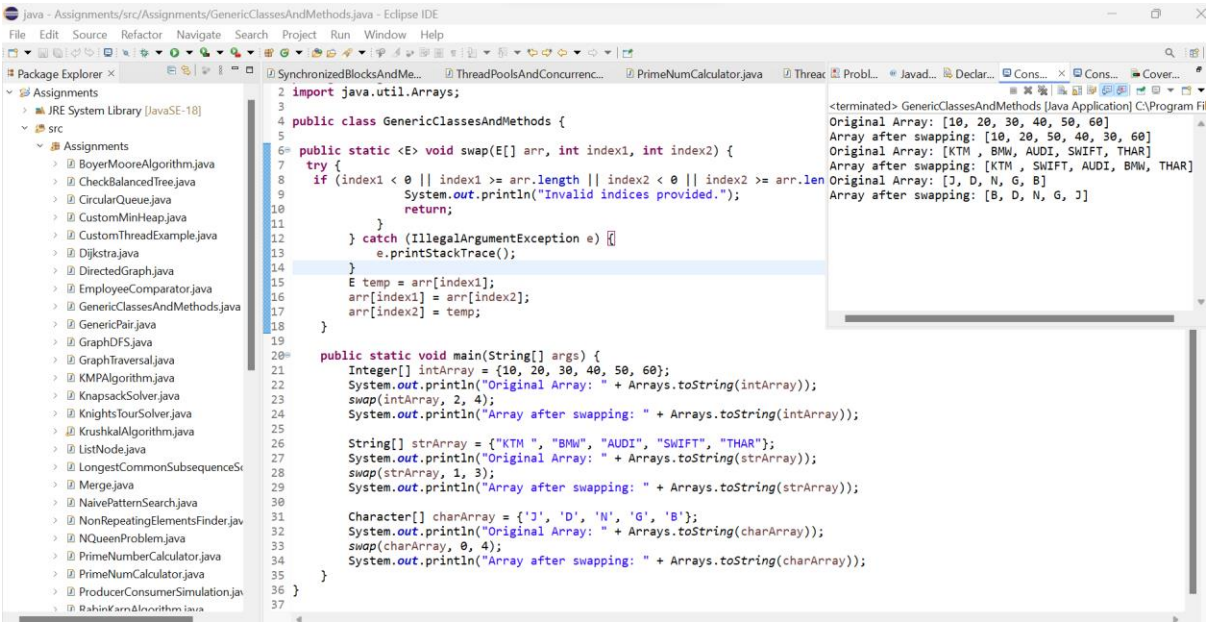
Code:

```
package Assignments;

import java.util.Arrays;

public class GenericClassesAndMethods {

 public static <E> void swap(E[] arr, int index1, int index2) {
  try {
    if (index1 < 0 || index1 >= arr.length || index2 < 0 || index2 >= arr.length){
                System.out.println("Invalid indices provided.");
                return;
            }
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        }
        E temp = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = temp;
    }

    public static void main(String[] args) {
        Integer[] intArray = {10, 20, 30, 40, 50, 60};
        System.out.println("Original Array: " + Arrays.toString(intArray));
        swap(intArray, 2, 4);
        System.out.println("Array after swapping: " + Arrays.toString(intArray));
```

```
        String[] strArray = {"KTM ", "BMW", "AUDI", "SWIFT", "THAR"};
        System.out.println("Original Array: " + Arrays.toString(strArray));
        swap(strArray, 1, 3);
        System.out.println("Array after swapping: " + Arrays.toString(strArray));

        Character[] charArray = {'J', 'D', 'N', 'G', 'B'};
        System.out.println("Original Array: " + Arrays.toString(charArray));
        swap(charArray, 0, 4);
        System.out.println("Array after swapping: " + Arrays.toString(charArray));
    }
}
```

Output:

```
Original Array: [10, 20, 30, 40, 50, 60]
Array after swapping: [10, 20, 50, 40, 30, 60]
Original Array: [KTM , BMW, AUDI, SWIFT, THAR]
Array after swapping: [KTM , SWIFT, AUDI, BMW, THAR]
Original Array: [J, D, N, G, B]
Array after swapping: [B, D, N, G, J]
```



Task 3: Reflection API:

Use reflection to inspect a class's methods, fields, and constructors, and modify the access level of a private field, setting its value during runtime

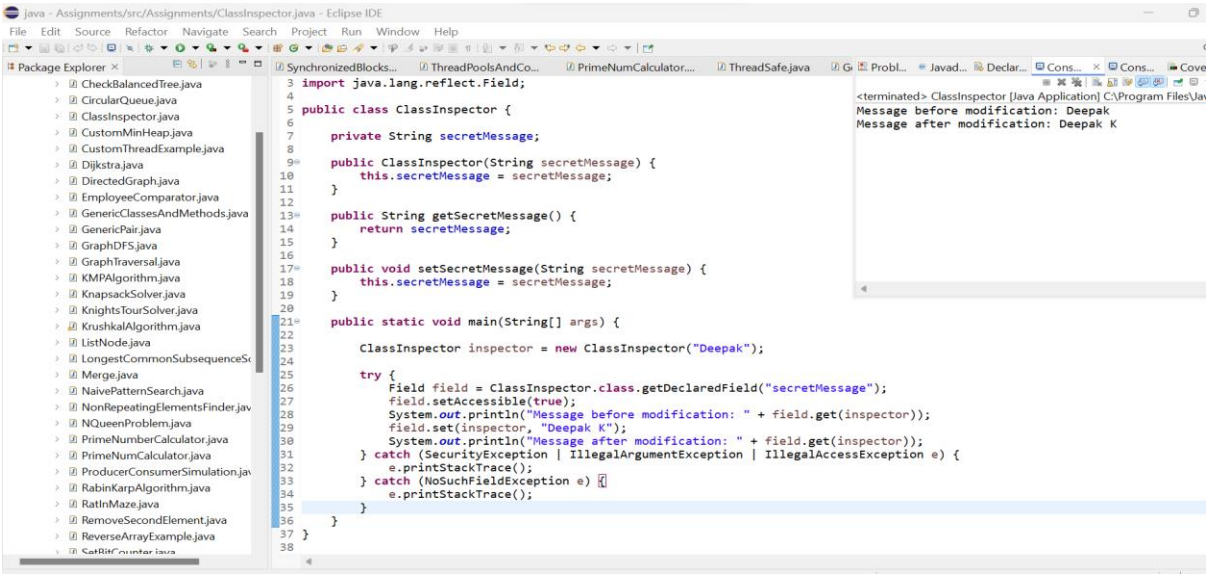
Code:

```
package Assignments;

import java.lang.reflect.Field;

public class ClassInspector {
```

```java
    private String secretMessage;

    public ClassInspector(String secretMessage) {
        this.secretMessage = secretMessage;
    }

    public String getSecretMessage() {
        return secretMessage;
    }

    public void setSecretMessage(String secretMessage) {
        this.secretMessage = secretMessage;
    }

    public static void main(String[] args) {

        ClassInspector inspector = new ClassInspector("Deepak");

        try {
            Field field = ClassInspector.class.getDeclaredField("secretMessage");
            field.setAccessible(true);
            System.out.println("Message before modification: " +
field.get(inspector));
            field.set(inspector, "Deepak K");
            System.out.println("Message after modification: " +
field.get(inspector));
        } catch (SecurityException | IllegalArgumentException |
IllegalAccessException e) {
            e.printStackTrace();
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Message before modification: Deepak
Message after modification: Deepak K
```



Task 4: Lambda Expressions:

Implement a Comparator for a Person class using a lambda expression, and sort a list of Person objects by their age..

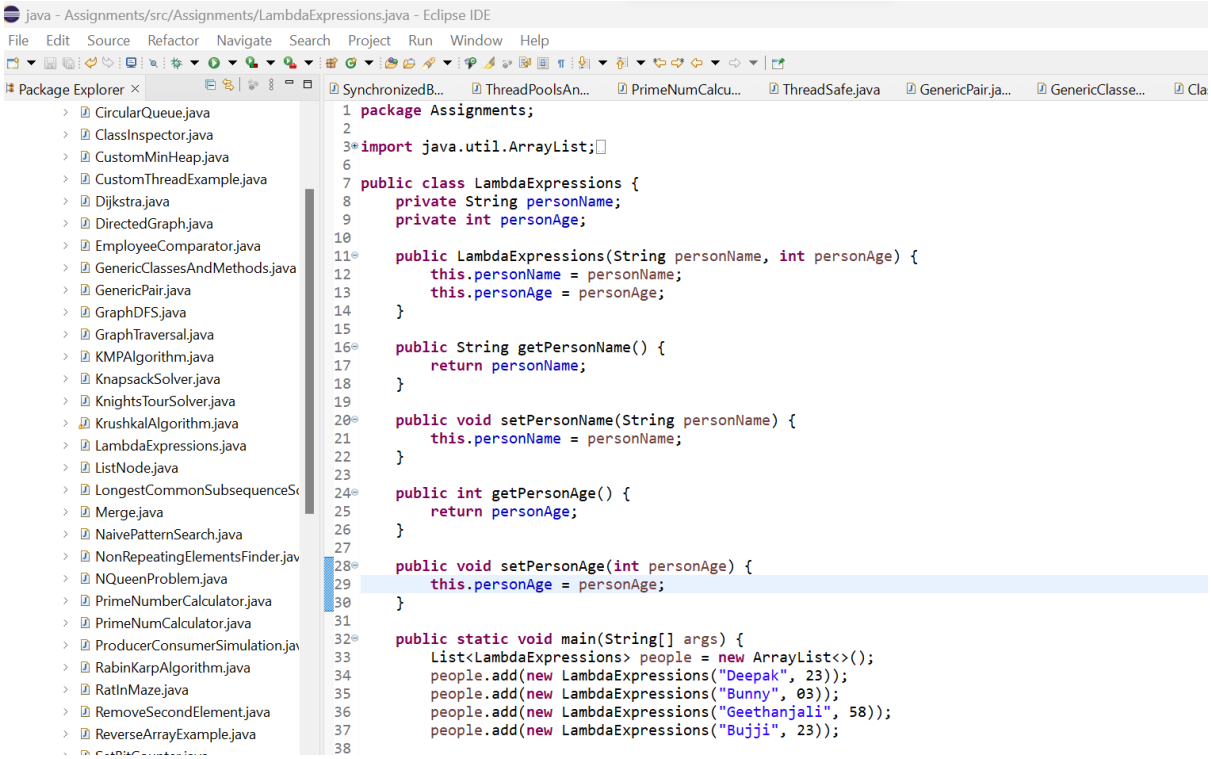
Code:

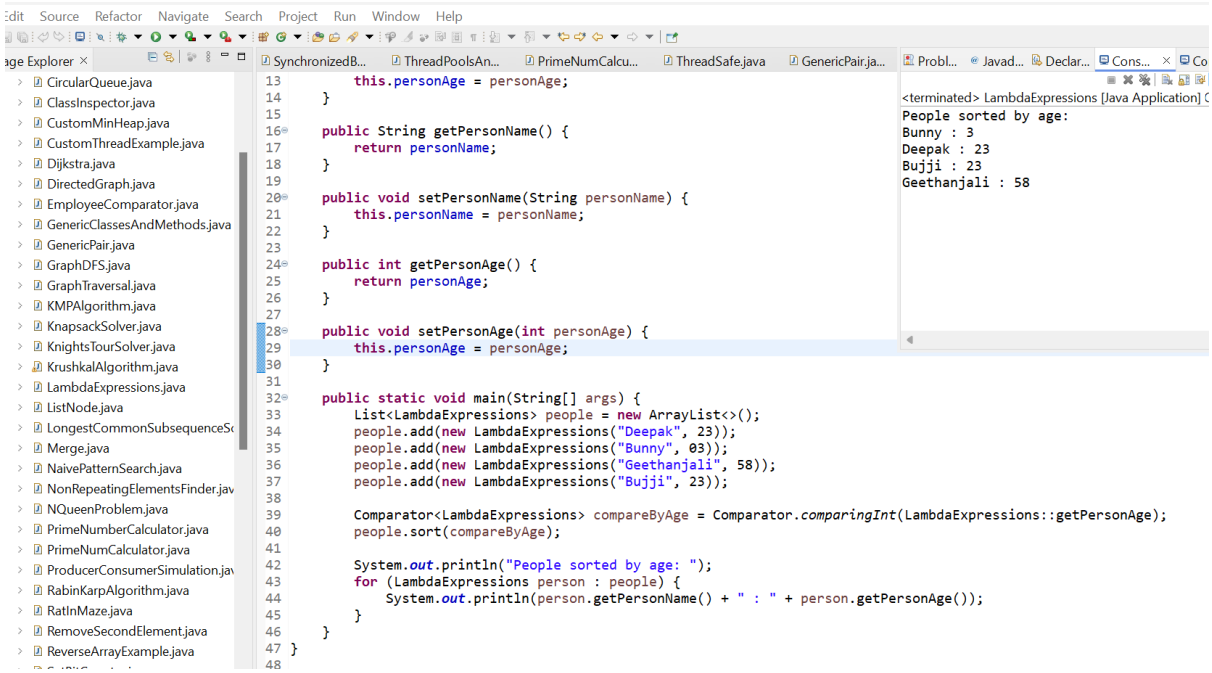
```
package Assignments;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class LambdaExpressions {
    private String personName;
    private int personAge;

    public LambdaExpressions(String personName, int personAge) {
        this.personName = personName;
        this.personAge = personAge;
    }

    public String getPersonName() {
        return personName;
    }

    public void setPersonName(String personName) {
        this.personName = personName;
    }

    public int getPersonAge() {
        return personAge;
    }

    public void setPersonAge(int personAge) {
        this.personAge = personAge;
    }

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

```
        List<LambdaExpressions> people = new ArrayList<>();
        people.add(new LambdaExpressions("Deepak", 23));
        people.add(new LambdaExpressions("Bunny", 03));
        people.add(new LambdaExpressions("Geethanjali", 58));
        people.add(new LambdaExpressions("Bujji", 23));

        Comparator<LambdaExpressions> compareByAge =
Comparator.comparingInt(LambdaExpressions::getPersonAge);
        people.sort(compareByAge);

        System.out.println("People sorted by age: ");
        for (LambdaExpressions person : people) {
            System.out.println(person.getPersonName() + " : " +
person.getPersonAge());
        }
    }
}
```

Output:

```
People sorted by age:
Bunny : 3
Deepak : 23
Bujji : 23
Geethanjali : 58
```





Task 5: Functional Interfaces:

Create a method that accepts functions as parameters using Predicate, Function, Consumer, and Supplier interfaces to operate on a Person object.

Code:

```
package Assignments;


import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;

import java.util.function.Supplier;


public class PersonProcessor {


    public static void handlePerson(

        Individual individual,

        Predicate<Individual> condition,
```

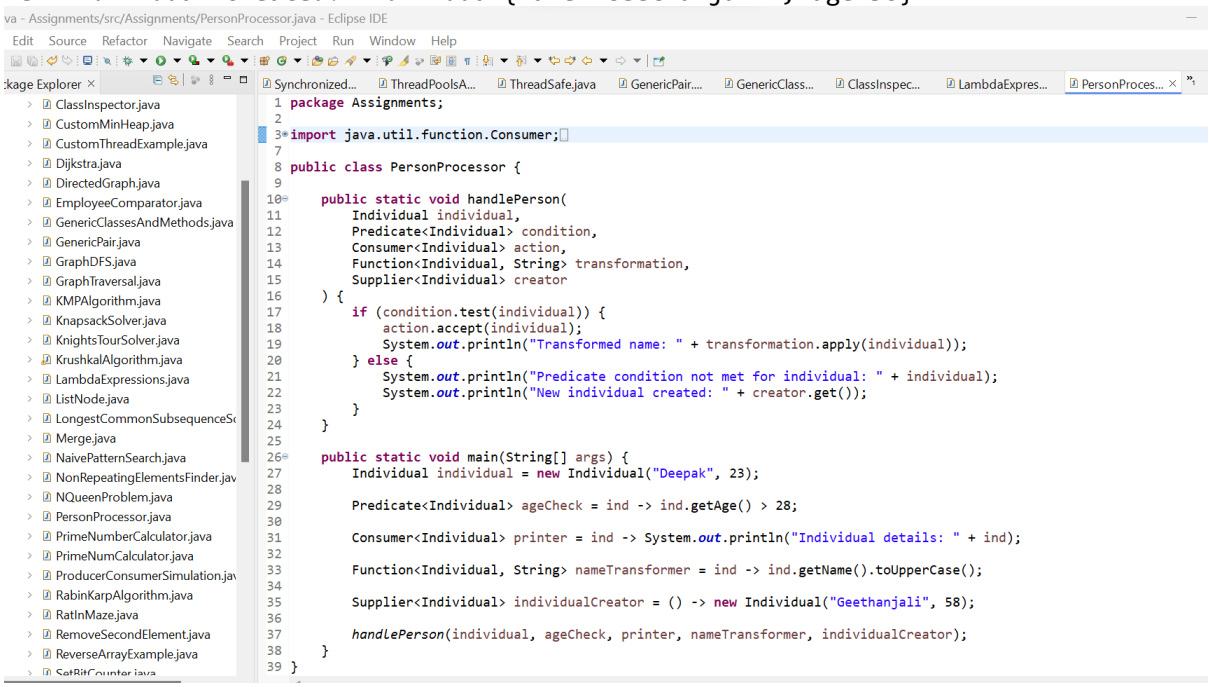
```
        Consumer<Individual> action,
        Function<Individual, String> transformation,
        Supplier<Individual> creator
    ) {
        if (condition.test(individual)) {
            action.accept(individual);
            System.out.println("Transformed name: " +
transformation.apply(individual));
        } else {
            System.out.println("Predicate condition not met for individual: " +
individual);
            System.out.println("New individual created: " + creator.get());
        }
    }

    public static void main(String[] args) {
        Individual individual = new Individual("Deepak", 23);

        Predicate<Individual> ageCheck = ind -> ind.getAge() > 28;

        Consumer<Individual> printer = ind -> System.out.println("Individual
details: " + ind);

        Function<Individual, String> nameTransformer = ind ->
ind.getName().toUpperCase();

        Supplier<Individual> individualCreator = () -> new
Individual("Geethanjali", 58);
```

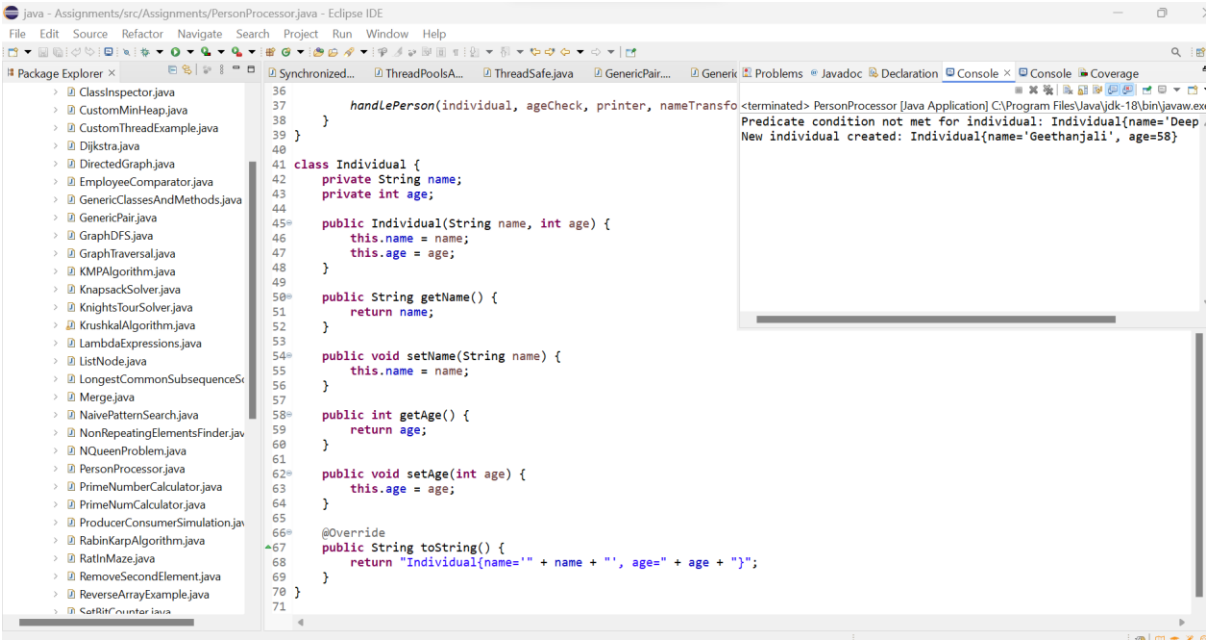
```
        handlePerson(individual, ageCheck, printer, nameTransformer,
individualCreator);
    }
}

class Individual {
    private String name;
    private int age;

    public Individual(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 "Individual{name='" + name + "', age=" + age + "}";
    }
}
```

Output:

```
Predicate condition not met for individual: Individual{name='Deepak', age=23}
New individual created: Individual{name='Geethanjali', age=58}
```




java - Assignments/src/Assignments/PersonProcessor.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ClassInspector.java
CustomMinHeap.java
CustomThreadExample.java
Dijkstra.java
DirectedGraph.java
EmployeeComparator.java
GenericClassesAndMethods.java
GenericPair.java
GraphDFS.java
GraphTraversal.java
KMPAlgorithm.java
KnapsackSolver.java
KnightsTourSolver.java
KrushkalAlgorithm.java
LambdaExpressions.java
ListNode.java
Merge.java
NaivePatternSearch.java
NQueenProblem.java
PersonProcessor.java
PrimeNumberCalculator.java
PrimeNumCalculator.java
RabinKarpAlgorithm.java
RatInMaze.java
RemoveSecondElement.java
ReverseArrayExample.java
Synchronized...
ThreadPoolsA...
ThreadSafe.java
GenericPair....
Problems
Javadoc
Declaration
Console
Console
Coverage
handlePerson(individual, ageCheck, printer, nameTransfo
}
}
class Individual {
private String name;
private int age;
public Individual(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 "Individual{name='" + name + "', age=" + age + "}";
}
}
<terminated> PersonProcessor [Java Application] C:\Program Files\Java\jdk-18\bin\javaw.ex
Predicate condition not met for individual: Individual{name='Deep
New individual created: Individual{name='Geethanjali', age=58}