Day 19: Manisha Assignment

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.

Explanation:

1. Pair Class:

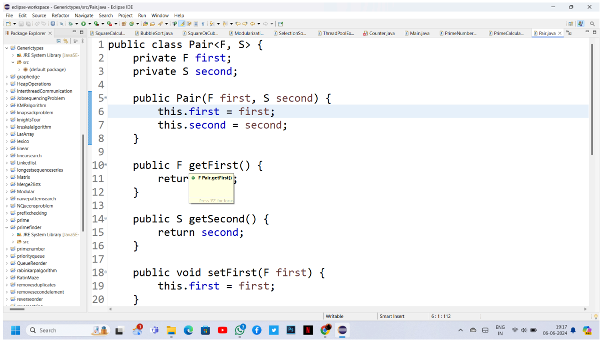
- `F` and `S` are generic type parameters representing the types of the first and second objects, respectively.
- Constructor initializes the first and second objects.
- Getter and setter methods for `first` and `second`.
- `reverse()` method returns a new `Pair` with the types of `first` and `second` swapped.
- `toString()` method provides a string representation of the pair.

2. Main Method:

- Creates an instance of `Pair` with a `String` and an `Integer`.
- Prints the original pair.
- Calls the `reverse()` method to get the reversed pair.
- Prints the reversed pair.

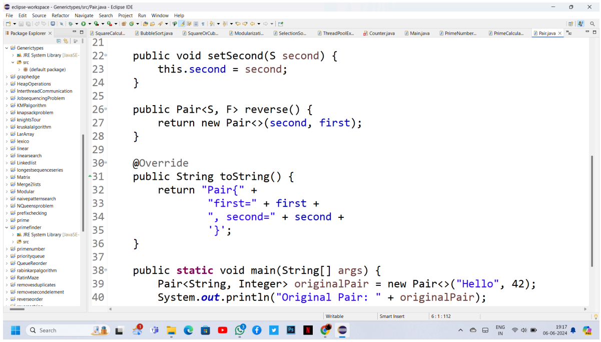
Output:

```
Original Pair: Pair{first=Hello, second=42}
Reversed Pair: Pair{first=42, second=Hello}
```



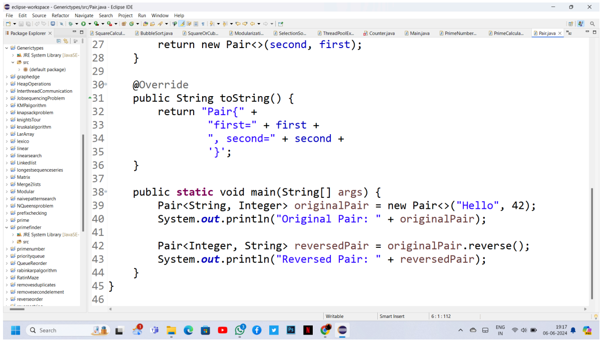
eclipse-workspace - Generictypes/src/Pair.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
SquareCalcul... BubbleSort.java SquareOrCub... Modularizati... SelectionSo... ThreadPoolEx... Counter.java Main.java PrimeNumber... PrimeCalcula... Pair.java
F Pair.getFirst()
Press 'F2' for focus
Writable
Smart Insert
6 : 1 : 112
19:17
06-06-2024

```java
public class Pair<F, S> {
    private F first;
    private S second;

    public Pair(F first, S second) {
        this.first = first;
        this.second = second;
    }

    public F getFirst() {
        retur ;
    }

    public S getSecond() {
        return second;
    }

    public void setFirst(F first) {
        this.first = first;
    }
```



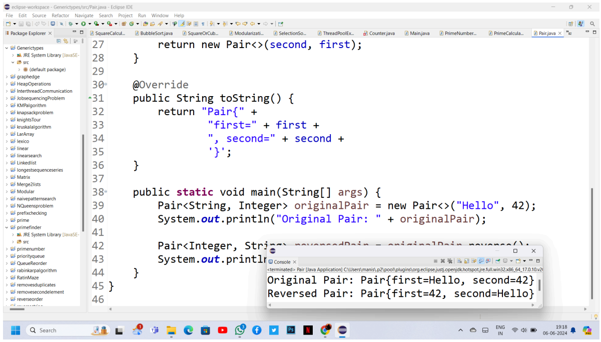
eclipse-workspace - Generictypes/src/Pair.java - Eclipse IDE
Writable
Smart Insert
6 : 1 : 112
19:17
06-06-2024

```java

    public void setSecond(S second) {
        this.second = second;
    }

    public Pair<S, F> reverse() {
        return new Pair<>(second, first);
    }

    @Override
    public String toString() {
        return "Pair{" +
                "first=" + first +
                ", second=" + second +
                '}';
    }

    public static void main(String[] args) {
        Pair<String, Integer> originalPair = new Pair<>("Hello", 42);
        System.out.println("Original Pair: " + originalPair);
```



eclipse-workspace - Generictypes/src/Pair.java - Eclipse IDE
Writable
Smart Insert
6 : 1 : 112
19:17
06-06-2024

```java
        return new Pair<>(second, first);
    }

    @Override
    public String toString() {
        return "Pair{" +
                "first=" + first +
                ", second=" + second +
                '}';
    }

    public static void main(String[] args) {
        Pair<String, Integer> originalPair = new Pair<>("Hello", 42);
        System.out.println("Original Pair: " + originalPair);

        Pair<Integer, String> reversedPair = originalPair.reverse();
        System.out.println("Reversed Pair: " + reversedPair);
    }
}

```



eclipse-workspace - Generictypes/src/Pair.java - Eclipse IDE
Console
19:18
06-06-2024

```java
        return new Pair<>(second, first);
    }

    @Override
    public String toString() {
        return "Pair{" +
                "first=" + first +
                ", second=" + second +
                '}';
    }

    public static void main(String[] args) {
        Pair<String, Integer> originalPair = new Pair<>("Hello", 42);
        System.out.println("Original Pair: " + originalPair);

        Pair<Integer, Strin
        System.out.printl
    }
}

```

```
<terminated> Pair [Java Application] C:\Users\manis\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v2
Original Pair: Pair{first=Hello, second=42}
Reversed Pair: Pair{first=42, second=Hello}
```

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.

Explanation:

1. Generic Method to Swap Elements:

- `swap(T[] array, int index1, int index2)`:
  - This is a generic method where `T` represents the type of the array elements.
  - It swaps the elements at `index1` and `index2` in the array.

2. Main Method:

- Demonstrates the usage of the `swap` method with an `Integer` array and a `String` array.
- Prints the original arrays.
- Swaps elements and prints the modified arrays.

Output:

Original Integer array:

1 2 3 4 5

Integer array after swap:

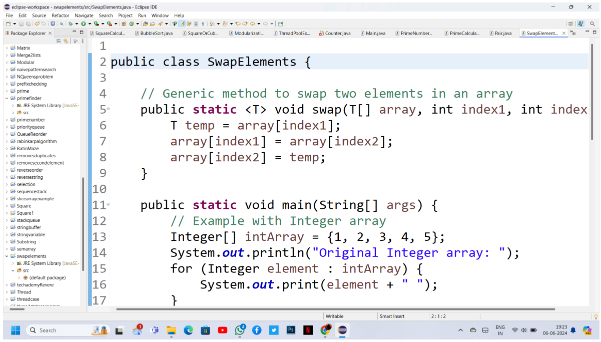
1 4 3 2 5

Original String array:

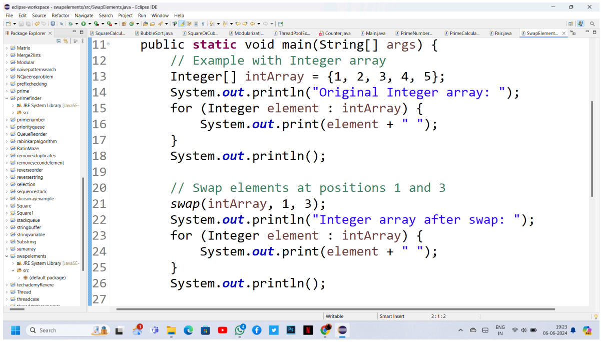
apple banana cherry date

String array after swap:

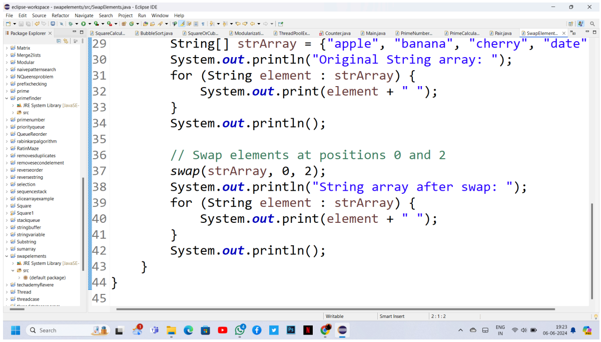
cherry banana apple date



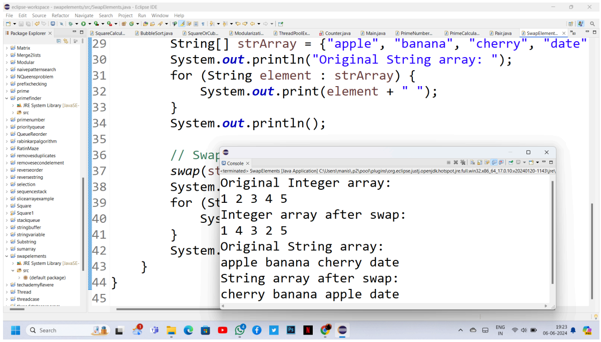
```java

public class SwapElements {

    // Generic method to swap two elements in an array
    public static <T> void swap(T[] array, int index1, int index
        T temp = array[index1];
        array[index1] = array[index2];
        array[index2] = temp;
    }

    public static void main(String[] args) {
        // Example with Integer array
        Integer[] intArray = {1, 2, 3, 4, 5};
        System.out.println("Original Integer array: ");
        for (Integer element : intArray) {
            System.out.print(element + " ");
        }
```



```java
    public static void main(String[] args) {
        // Example with Integer array
        Integer[] intArray = {1, 2, 3, 4, 5};
        System.out.println("Original Integer array: ");
        for (Integer element : intArray) {
            System.out.print(element + " ");
        }
        System.out.println();

        // Swap elements at positions 1 and 3
        swap(intArray, 1, 3);
        System.out.println("Integer array after swap: ");
        for (Integer element : intArray) {
            System.out.print(element + " ");
        }
        System.out.println();

```



```java
        String[] strArray = {"apple", "banana", "cherry", "date"
        System.out.println("Original String array: ");
        for (String element : strArray) {
            System.out.print(element + " ");
        }
        System.out.println();

        // Swap elements at positions 0 and 2
        swap(strArray, 0, 2);
        System.out.println("String array after swap: ");
        for (String element : strArray) {
            System.out.print(element + " ");
        }
        System.out.println();
    }
}

```



```java
        String[] strArray = {"apple", "banana", "cherry", "date"
        System.out.println("Original String array: ");
        for (String element : strArray) {
            System.out.print(element + " ");
        }
        System.out.println();

        // Swap
        swap(st
        System.
        for (St
            Sys
        }
        System.
    }
}

```

<terminated> SwapElements [Java Application] C:\Users\manis\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\

```
Original Integer array:
1 2 3 4 5
Integer array after swap:
1 4 3 2 5
Original String array:
apple banana cherry date
String array after swap:
cherry banana apple date
```

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

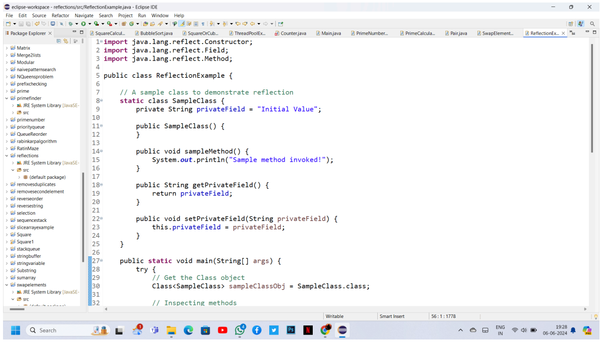
Explanation:

1. **SampleClass:**
   - A simple class with a private field `privateField`, a default constructor, and a method `sampleMethod`.

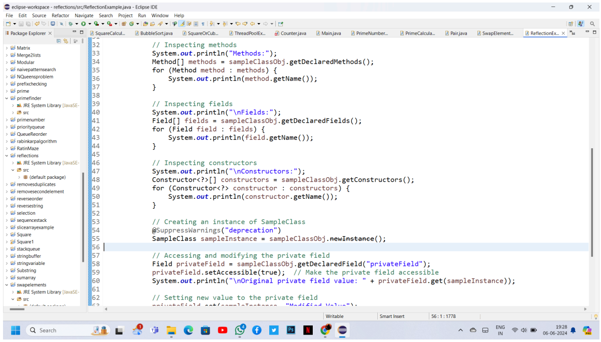
2. **Main Method:**
   - Get the Class object: `Class<SampleClass> sampleClassObj = SampleClass.class;`
   - Inspect Methods: Uses `getDeclaredMethods()` to list all methods.
   - Inspect Fields: Uses `getDeclaredFields()` to list all fields.
   - Inspect Constructors: Uses `getConstructors()` to list all constructors.
   - Create Instance: Uses `newInstance()` to create an instance of `SampleClass`.
   - Access and Modify Private Field:
     - Uses `getDeclaredField("privateField")` to get the `privateField`.
     - Sets the field accessible with `setAccessible(true)`.
     - Prints the original value and modifies it using `set()` method.
   - Invoke Method: Uses `getMethod("sampleMethod")` to get the `sampleMethod` and `invoke()` to call it.

Output:

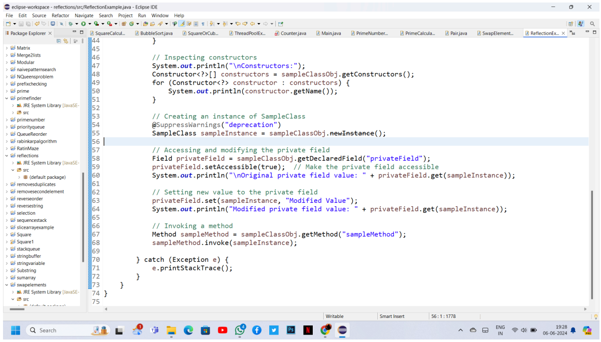
```
Methods:
sampleMethod

Fields:
privateField

Constructors:
ReflectionExample$SampleClass

Original private field value: Initial Value
Modified private field value: Modified Value
Sample method invoked!
```


eclipse-workspace - reflections/src/ReflectionExample.java - Eclipse IDE

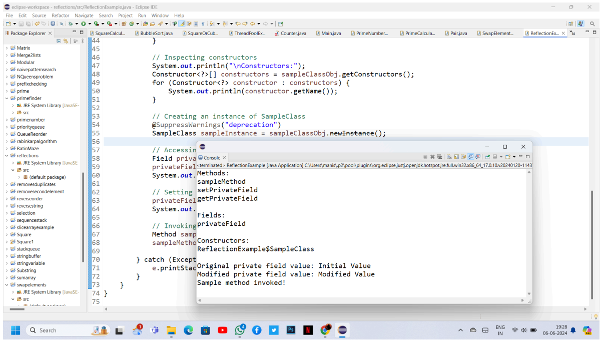
```java
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class ReflectionExample {

    // A sample class to demonstrate reflection
    static class SampleClass {
        private String privateField = "Initial Value";

        public SampleClass() {
        }

        public void sampleMethod() {
            System.out.println("Sample method invoked!");
        }

        public String getPrivateField() {
            return privateField;
        }

        public void setPrivateField(String privateField) {
            this.privateField = privateField;
        }
    }

    public static void main(String[] args) {
        try {
            // Get the Class object
            Class<SampleClass> sampleClassObj = SampleClass.class;

            // Inspecting methods
```


eclipse-workspace - reflections/src/ReflectionExample.java - Eclipse IDE

```java
            // Inspecting methods
            System.out.println("Methods:");
            Method[] methods = sampleClassObj.getDeclaredMethods();
            for (Method method : methods) {
                System.out.println(method.getName());
            }

            // Inspecting fields
            System.out.println("\nFields:");
            Field[] fields = sampleClassObj.getDeclaredFields();
            for (Field field : fields) {
                System.out.println(field.getName());
            }

            // Inspecting constructors
            System.out.println("\nConstructors:");
            Constructor<?>[] constructors = sampleClassObj.getConstructors();
            for (Constructor<?> constructor : constructors) {
                System.out.println(constructor.getName());
            }

            // Creating an instance of SampleClass
            @SuppressWarnings("deprecation")
            SampleClass sampleInstance = sampleClassObj.newInstance();

            // Accessing and modifying the private field
            Field privateField = sampleClassObj.getDeclaredField("privateField");
            privateField.setAccessible(true);  // Make the private field accessible
            System.out.println("\nOriginal private field value: " + privateField.get(sampleInstance));

            // Setting new value to the private field
```


eclipse-workspace - reflections/src/ReflectionExample.java - Eclipse IDE

```java
            }

            // Inspecting constructors
            System.out.println("\nConstructors:");
            Constructor<?>[] constructors = sampleClassObj.getConstructors();
            for (Constructor<?> constructor : constructors) {
                System.out.println(constructor.getName());
            }

            // Creating an instance of SampleClass
            @SuppressWarnings("deprecation")
            SampleClass sampleInstance = sampleClassObj.newInstance();

            // Accessing and modifying the private field
            Field privateField = sampleClassObj.getDeclaredField("privateField");
            privateField.setAccessible(true);  // Make the private field accessible
            System.out.println("\nOriginal private field value: " + privateField.get(sampleInstance));

            // Setting new value to the private field
            privateField.set(sampleInstance, "Modified Value");
            System.out.println("Modified private field value: " + privateField.get(sampleInstance));

            // Invoking a method
            Method sampleMethod = sampleClassObj.getMethod("sampleMethod");
            sampleMethod.invoke(sampleInstance);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```


eclipse-workspace - reflections/src/ReflectionExample.java - Eclipse IDE

```java
            }

            // Inspecting constructors
            System.out.println("\nConstructors:");
            Constructor<?>[] constructors = sampleClassObj.getConstructors();
            for (Constructor<?> constructor : constructors) {
                System.out.println(constructor.getName());
            }

            // Creating an instance of SampleClass
            @SuppressWarnings("deprecation")
            SampleClass sampleInstance = sampleClassObj.newInstance();
```

Console

<terminated> ReflectionExample [Java Application] C:\Users\manis\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143

```
Methods:
sampleMethod
setPrivateField
getPrivateField

Fields:
privateField

Constructors:
ReflectionExample$SampleClass

Original private field value: Initial Value
Modified private field value: Modified Value
Sample method invoked!
```

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..

Explanation:

1. Person Class:

- The `Person` class has two fields: `name` and `age`.

- It includes a constructor to initialize these fields.

- The `toString()` method is overridden to provide a readable string representation of a `Person` object.

2. PersonComparatorExample Class:

- In the `main` method, a list of `Person` objects is created and populated.

- The list is sorted using `Collections.sort()` with a lambda expression that implements the `Comparator` interface, comparing `Person` objects by their `age`.

- The sorted list is printed to the console.

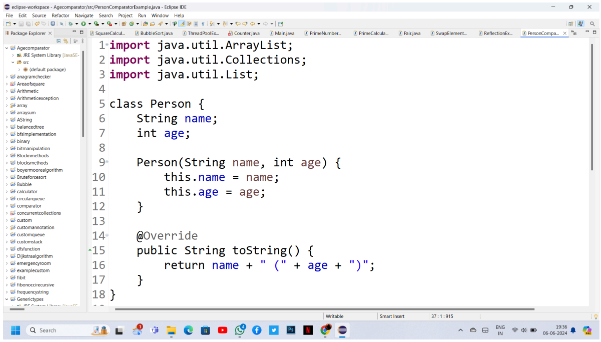
Output

Dave (20)

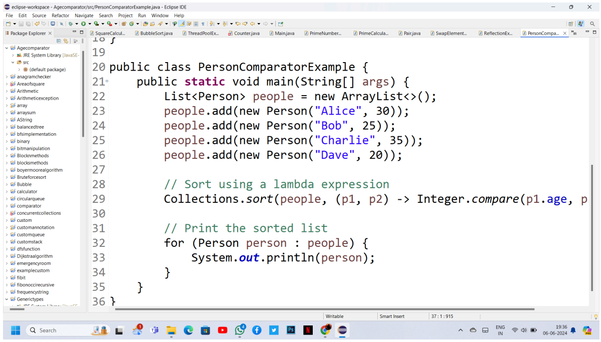
Bob (25)

Alice (30)

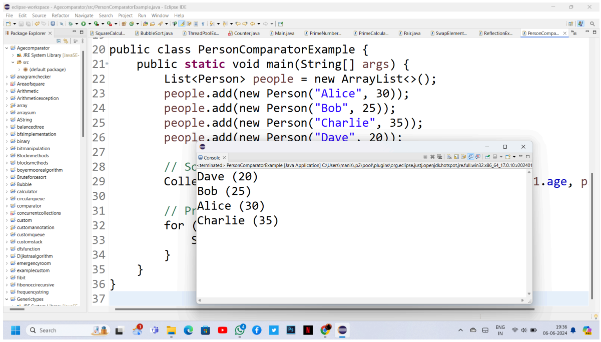
Charlie (35)



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

class Person {
    String name;
    int age;

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

    @Override
    public String toString() {
        return name + " (" + age + ")";
    }
}
```



```java

public class PersonComparatorExample {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Alice", 30));
        people.add(new Person("Bob", 25));
        people.add(new Person("Charlie", 35));
        people.add(new Person("Dave", 20));

        // Sort using a lambda expression
        Collections.sort(people, (p1, p2) -> Integer.compare(p1.age, p

        // Print the sorted list
        for (Person person : people) {
            System.out.println(person);
        }
    }
}
```



```java
public class PersonComparatorExample {
    public static void main(String[] args) {
        List<Person> people = new ArrayList<>();
        people.add(new Person("Alice", 30));
        people.add(new Person("Bob", 25));
        people.add(new Person("Charlie", 35));
        people.add(new Person("Dave", 20));
```

<terminated> PersonComparatorExample [Java Application] C:\Users\manis\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v202401

```
Dave (20)
Bob (25)
Alice (30)
Charlie (35)
```

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.

Explanation:

1. Person Class:

- The `Person` class has two fields: `name` and `age`.

- It includes a constructor to initialize these fields.

- The `toString()` method is overridden to provide a readable string representation of a `Person` object.

2. FunctionalInterfacesExample Class:

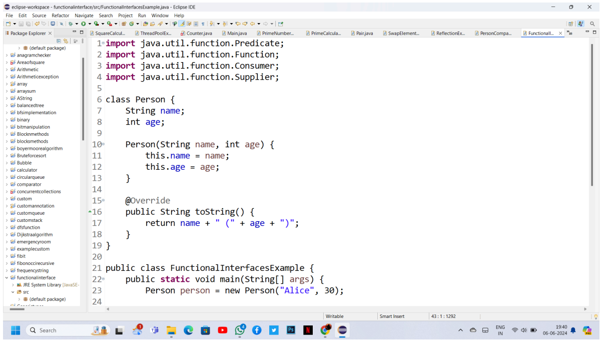
- The `main` method demonstrates the use of different functional interfaces (`Predicate`, `Function`, `Consumer`, and `Supplier`) to operate on a `Person` object.

- Predicate: A functional interface that takes one argument and returns a boolean. The `isAdult` predicate checks if a person's age is greater than 25.

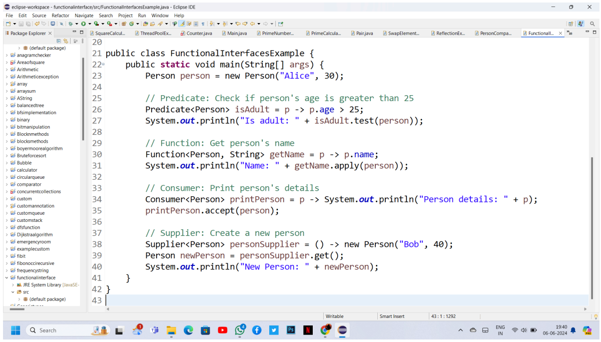
- Function: A functional interface that takes one argument and returns a result. The `getName` function retrieves the name of the person.

- Consumer: A functional interface that takes one argument and performs an action. The `printPerson` consumer prints the person's details.

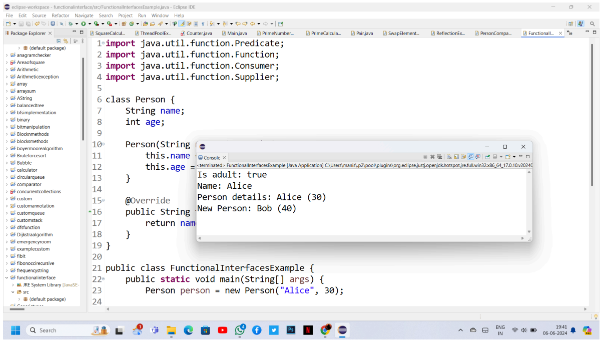
- Supplier: A functional interface that takes no arguments and returns a result. The `personSupplier` creates a new `Person` object.



```java
import java.util.function.Predicate;
import java.util.function.Function;
import java.util.function.Consumer;
import java.util.function.Supplier;

class Person {
    String name;
    int age;

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

    @Override
    public String toString() {
        return name + " (" + age + ")";
    }
}

public class FunctionalInterfacesExample {
    public static void main(String[] args) {
        Person person = new Person("Alice", 30);

```



```java

public class FunctionalInterfacesExample {
    public static void main(String[] args) {
        Person person = new Person("Alice", 30);

        // Predicate: Check if person's age is greater than 25
        Predicate<Person> isAdult = p -> p.age > 25;
        System.out.println("Is adult: " + isAdult.test(person));

        // Function: Get person's name
        Function<Person, String> getName = p -> p.name;
        System.out.println("Name: " + getName.apply(person));

        // Consumer: Print person's details
        Consumer<Person> printPerson = p -> System.out.println("Person details: " + p);
        printPerson.accept(person);

        // Supplier: Create a new person
        Supplier<Person> personSupplier = () -> new Person("Bob", 40);
        Person newPerson = personSupplier.get();
        System.out.println("New Person: " + newPerson);
    }
}

```



```
<terminated> FunctionalInterfacesExample [Java Application] C:\Users\manis\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v2024
Is adult: true
Name: Alice
Person details: Alice (30)
New Person: Bob (40)
```

Output:
Is adult: true
Name: Alice
Person details: Alice (30)
New Person: Bob (40)