1.A Generics and Type Safety

public class Pair<T, U> {

private T first;

private U second;

public Pair(T first, U second) {

this.first = first;

this.second = second;

}

public T getFirst() {

return first;

}

public U getSecond() {

return second;

}

public Pair<U, T> reverse() {

return new Pair<>(second, first);

}

public static void main(String[] args) {

Pair<Integer, String> intStringPair = new Pair<>(42, "Hello");

System.out.println("Original Pair: " + intStringPair.getFirst() + ", " + intStringPair.getSecond());

Pair<String, Integer> reversedPair = intStringPair.reverse();

System.out.println("Reversed Pair: " + reversedPair.getFirst() + ", " + reversedPair.getSecond());

}

}

2.A Generic Classes and Methods

import java.util.List;

public class ArraySwapper {

public static <T> void swap(T[] array, int index1, int index2) {

T temp = array[index1];

array[index1] = array[index2];

array[index2] = temp;

}

public static void main(String[] args) {

Integer[] intArray = {1, 2, 3};

String[] stringArray = {"apple", "banana", "cherry"};

swap(intArray, 0, 2);

System.out.println("Swapped Integer array: " + java.util.Arrays.toString(intArray));

swap(stringArray, 1, 2);

System.out.println("Swapped String array: " + java.util.Arrays.toString(stringArray));

}

}

3.A Reflection API

class Animal { }

class Dog extends Animal {

public void display() {

System.out.println("I am a dog.");

}

}

public class Main {

public static void main(String[] args) {

try {

Dog d1 = new Dog();

Class<?> dogClass = d1.getClass();

String name = dogClass.getName();

System.out.println("Name: " + name);

int modifier = dogClass.getModifiers();

String mod = Modifier.toString(modifier);

System.out.println("Modifier: " + mod);

Class<?> superClass = dogClass.getSuperclass();

System.out.println("Superclass: " + superClass.getName());

} catch (Exception e) {

e.printStackTrace();

}

}

}

4.A Lambda Expressions

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

class Person {

private String firstName;

private String lastName;

private int age;

public Person(String firstName, String lastName, int age) {

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public int getAge() {

return age;

}

@Override

public String toString() {

return firstName + " " + lastName + " (" + age + " years old)";

}

}

public class PersonComparatorExample {

public static void main(String[] args) {

List<Person> personList = new ArrayList<>();

personList.add(new Person("Alice", "Smith", 30));

personList.add(new Person("Bob", "Johnson", 25));

personList.add(new Person("Carol", "Brown", 35));

personList.sort(Comparator.comparingInt(Person::getAge));

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

}

}

5.A Functional Interface

import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;

import java.util.function.Supplier;

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public void processPerson(Predicate<Person> filter,

Function<Person, String> transform,

Consumer<String> action,

Supplier<Person> supplier) {

if (filter.test(this)) {

String transformedName = transform.apply(this);

action.accept("Processed: " + transformedName);

} else {

Person newPerson = supplier.get();

System.out.println("Created new person: " + newPerson.name);

}

}

public static void main(String[] args) {

Person person = new Person("Alice", 30);

person.processPerson(

p -> p.age >= 18

p -> p.name.toUpperCase(),uppercase

System.out::println,

() -> new Person("Bob", 25)

);

}

}