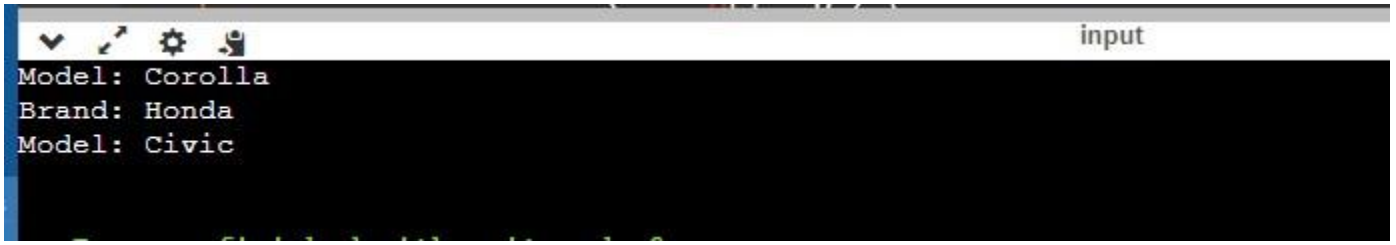
JAVA ASSIGNMENTS

1) **Write a program to cover all Java OOPS concepts. Topics need to cover:**

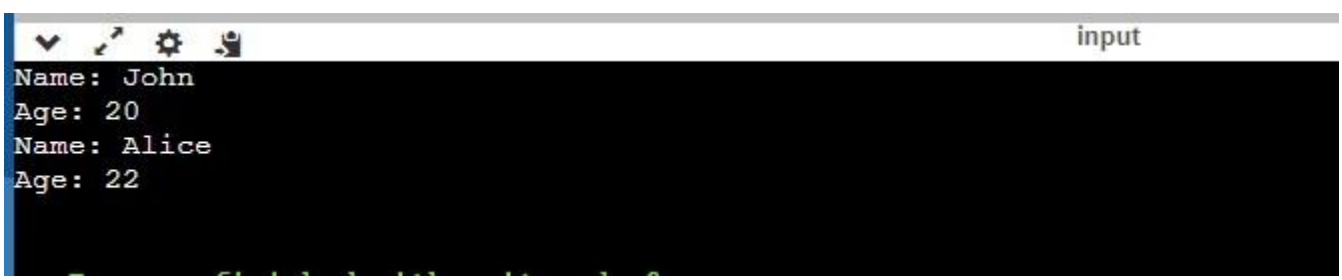
   a) **Class and Object**

```
package info;

// Class and Object
class Vehicle {
String brand;
String model;

public void displayInfo() {
System.out.println("Brand: " + brand);
System.out.println("Model: " + model);
}
}

public class Main {
public static void main(String[] args) {
// Creating objects of Car class
Vehicle car1 = new Vehicle();
car1.brand = "Toyota";
car1.model = "Corolla";
car1.displayInfo();

Vehicle car2 = new Vehicle();
car2.brand = "Honda";
car2.model = "Civic";
car2.displayInfo();
}
}
```



b) Class constructor

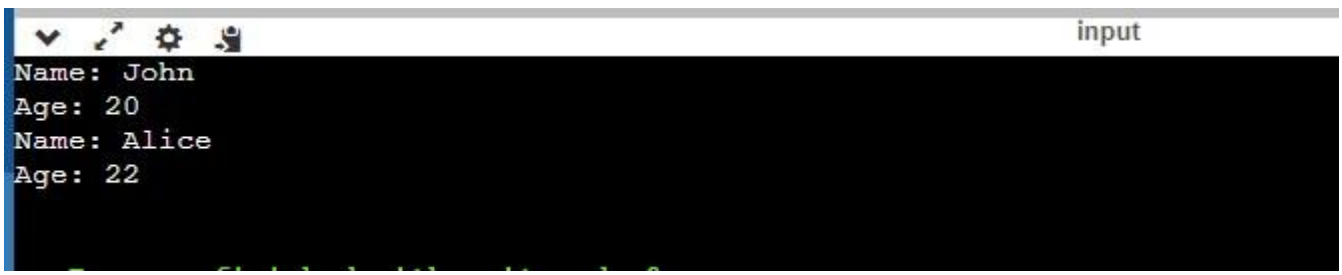
```
package info;
// Class and Constructor
class Student {
String name;
int age;

// Constructor
public Student(String name, int age) {
this.name = name;
this.age = age;
}

public void displayInfo() {
System.out.println("Name: " + name);
System.out.println("Age: " + age);
}
}

public class Main {
public static void main(String[] args) {
// Creating object using constructor
Student student1 = new Student("John", 20);
student1.displayInfo();

Student student2 = new Student("Alice", 22);
student2.displayInfo();
}
}
```



c) Polymorphism

```
// Class and Constructor
class Student {
String name;
int age;
```

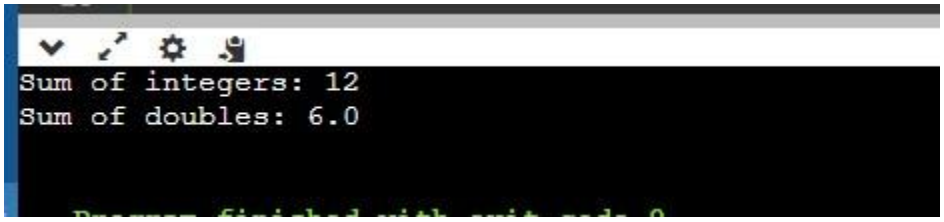
```
// Constructor
public Student(String name, int age) {
this.name = name;
this.age = age;
}

public void displayInfo() {
System.out.println("Name: " + name);
System.out.println("Age: " + age);
}
}

public class Main {
public static void main(String[] args) {
// Creating object using constructor
Student student1 = new Student("John", 20);
student1.displayInfo();

Student student2 = new Student("Alice", 22);
student2.displayInfo();
}
}
```



d) Method overloading

```
// Method Overloading
class Calculator {
public int add(int a, int b) {
return a + b;
}

public double add(double a, double b) {
return a + b;
}
}

public class Main {
```

```
public static void main(String[] args) {
Calculator calculator = new Calculator();
System.out.println("Sum of integers: " + calculator.add(5,
7));
System.out.println("Sum of doubles: " + calculator.add(3.5,
2.5));
}
}
```



e) Method overriding

```
package info;

// Method Overriding
class Animal {
public void sound() {
System.out.println("Animal makes a sound");
}
}

class Dog extends Animal {
@Override
public void sound() {
System.out.println("Dog barks");
}
}

public class Main {
public static void main(String[] args) {
Animal animal = new Dog();
animal.sound();
}
}
```

```
Dog barks
```

f) Inheritance

```
// Inheritance
class Animal {
public void eat() {
System.out.println("Animal is eating");
}
}

class Dog extends Animal {
public void bark() {
System.out.println("Dog is barking");
}
}

public class Main {
public static void main(String[] args) {
Dog dog = new Dog();
dog.eat();
dog.bark();
}
}
```

```
Animal is eating
Dog is barking
```

g) Interface

```
package info;

//Interface
interface Animal {
void sound();
}
```

```
class Dog implements Animal {
@Override
public void sound() {
System.out.println("Dog barks");
}
}

public class Inface {
public static void main(String[] args) {
Dog dog = new Dog();
dog.sound();
}
}
```

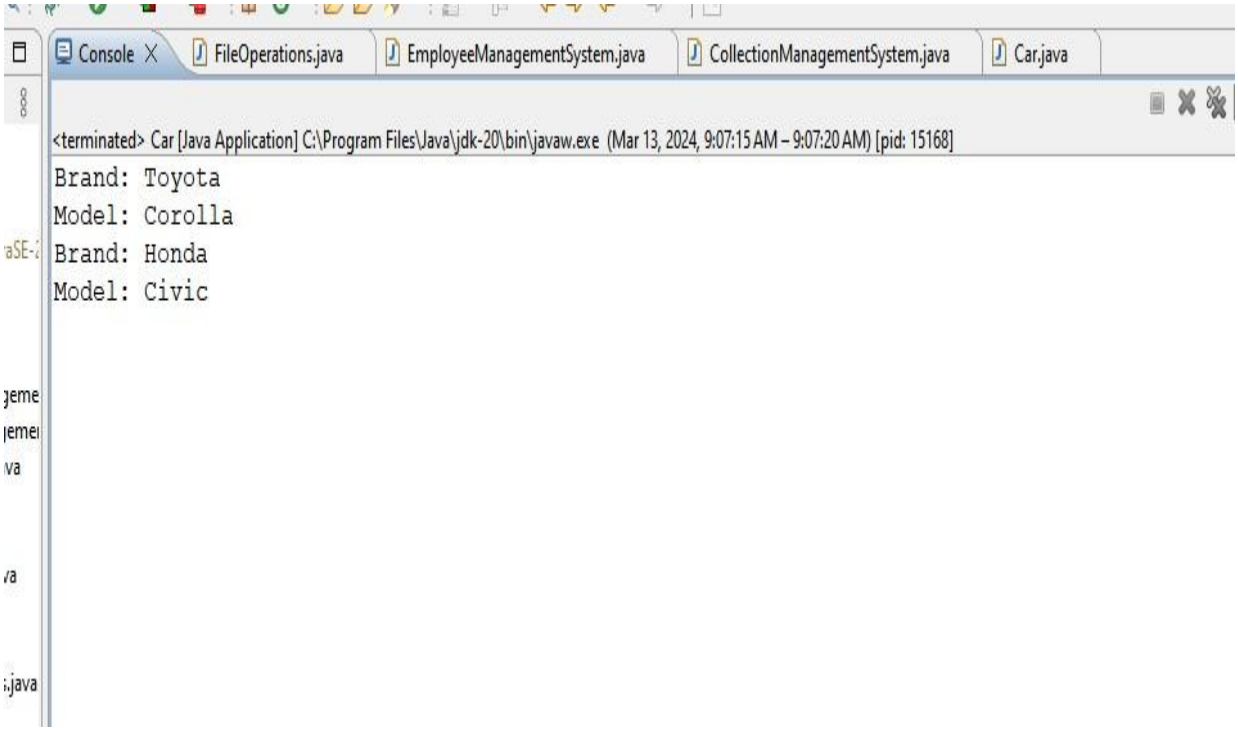
h) Abstract class

```
package info;

// Abstract class
abstract class Animal {
abstract void sound();
}

class Dog extends Animal {
@Override
void sound() {
System.out.println("Dog barks");
}
}

public class Main {
public static void main(String[] args) {
Dog dog = new Dog();
dog.sound();
}
}
```

```
Dog barks
```

i) Abstraction and Encapsulation

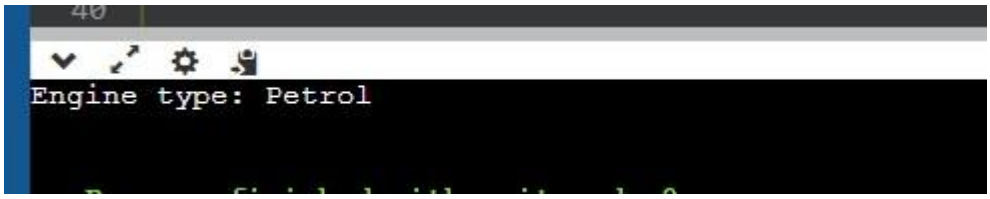
```java
package info;

// Abstraction and Encapsulation
class Car {
private String brand;
private String model;

public void setBrand(String brand) {
this.brand = brand;
}

public String getBrand() {
return brand;
}

public void setModel(String model) {
this.model = model;
}

public String getModel() {
return model;
}

public void displayInfo() {
System.out.println("Brand: " + brand);
System.out.println("Model: " + model);
}
}

public class Main {
public static void main(String[] args) {
Car car = new Car();
car.setBrand("Toyota");
car.setModel("Corolla");
car.displayInfo();
}
}
```



j) Composition and Aggregation

```
package info;
// Composition and Aggregation
class Engine {
private String type;

public Engine(String type) {
this.type = type;
}

public String getType() {
return type;
}
}

class Car {
private Engine engine;

public Car(Engine engine) {
this.engine = engine;
}

public void displayInfo() {
System.out.println("Engine type: " + engine.getType());
```

```
}
}

public class Main {
public static void main(String[] args) {
Engine engine = new Engine("Petrol");
Car car = new Car(engine);
car.displayInfo();
}
}
```



k) Generalization and Specialization

```
package info;

// Generalization and Specialization
class Vehicle {
public void move() {
System.out.println("Vehicle is moving");
}
}

class Car extends Vehicle {
public void honk() {
System.out.println("Car is honking");
}
}

public class Main {
public static void main(String[] args) {
Car car = new Car();
car.move();
car.honk();
}
}
```

```
Vehicle is moving
Car is honking
```

2) **Design a Java program that performs various string operations and uses control statements for user input validation. The program should allow the user to perform the following operations:**

   a) **Concatenate Strings: The user can enter two strings and the program should concatenate them.**
   b) **Find Length of a String: The user can enter a string, and the program should display its length.**
   c) **Convert to Uppercase and Lowercase: The user can enter a string, and the program should display it in both uppercase and lowercase.**
   d) **Extract Substring: The user can enter a string and specify the starting and ending index, and the program should extract and display the substring.**
   e) **Split a Sentence: The user can enter a sentence, and the program should split it into words and display them.**
   f) **Reverse a String: The user can enter a string, and the program should reverse and display it.**
   g) **Requirements:**
      i) **Use control statements (if-else, switch, loops) for input validation and handling possible errors.**
      ii) **Implement a user-friendly console interface for the user to interact with the program.**
      iii) **Cover all string concepts, such as concatenation, length, uppercase and lowercase conversion, substring extraction, splitting, and reversal.**

Program:

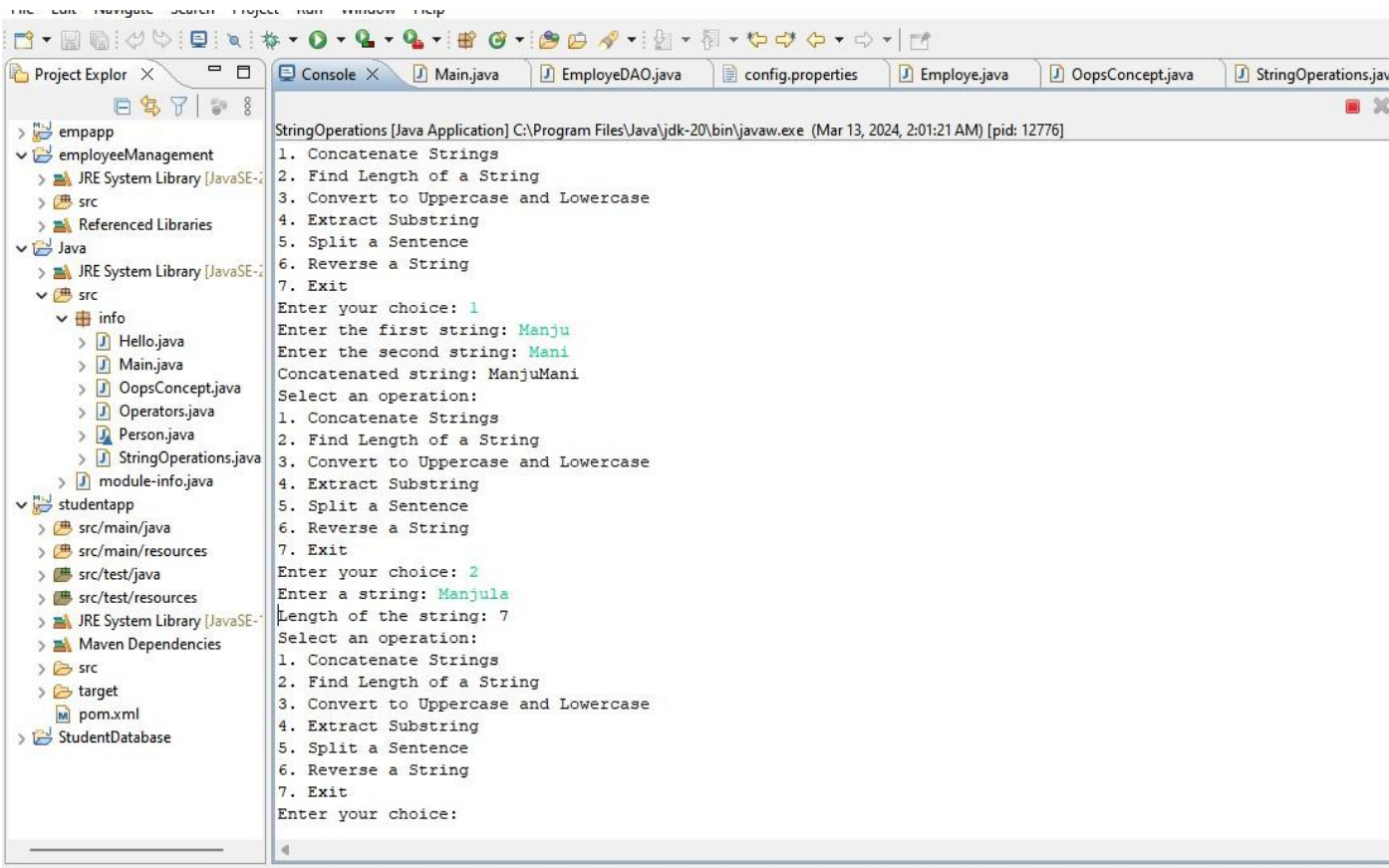
```java
package info;
import java.util.Scanner;

public class StringOperations {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
int choice;

do {
System.out.println("Select an operation:");
System.out.println("1. Concatenate Strings");
System.out.println("2. Find Length of a String");
```

```
System.out.println("3. Convert to Uppercase and Lowercase");
System.out.println("4. Extract Substring");
System.out.println("5. Split a Sentence");
System.out.println("6. Reverse a String");
System.out.println("7. Exit");
System.out.print("Enter your choice: ");
// Input validation
while (!scanner.hasNextInt()) {
System.out.println("Invalid input! Please enter a number.");
scanner.next();
}
choice = scanner.nextInt();

switch (choice) {
case 1:
concatenateStrings(scanner);
break;
case 2:
findStringLength(scanner);
break;
case 3:
convertToUppercaseLowercase(scanner);
break;
case 4:
extractSubstring(scanner);
break;
case 5:
splitSentence(scanner);
break;
case 6:
reverseString(scanner);
break;
case 7:
System.out.println("Exiting program...");
break;
default:
System.out.println("Invalid choice! Please enter a number
between 1 and 7.");
}
} while (choice != 7);

scanner.close();
}

private static void concatenateStrings(Scanner scanner) {
System.out.print("Enter the first string: ");
String str1 = scanner.next();
```

```
System.out.print("Enter the second string: ");
String str2 = scanner.next();
System.out.println("Concatenated string: " +
str1.concat(str2));
}

private static void findStringLength(Scanner scanner) {
System.out.print("Enter a string: ");
String str = scanner.next();
System.out.println("Length of the string: " + str.length());
}

private static void convertToUppercaseLowercase(Scanner
scanner) {
System.out.print("Enter a string: ");
String str = scanner.next();
System.out.println("Uppercase: " + str.toUpperCase());
System.out.println("Lowercase: " + str.toLowerCase());
}

private static void extractSubstring(Scanner scanner) {
System.out.print("Enter a string: ");
String str = scanner.next();
System.out.print("Enter starting index: ");
int startIndex = scanner.nextInt();
System.out.print("Enter ending index: ");
int endIndex = scanner.nextInt();
if (startIndex < 0 || endIndex >= str.length() ||
startIndex > endIndex) {
System.out.println("Invalid indices!");
} else {
System.out.println("Substring: " + str.substring(startIndex,
endIndex + 1));
}
}

private static void splitSentence(Scanner scanner) {
System.out.print("Enter a sentence: ");
scanner.nextLine(); // Consume newline character left from
previous input
String sentence = scanner.nextLine();
String[] words = sentence.split(" ");
System.out.println("Words in the sentence:");
for (String word : words) {
System.out.println(word);
}
}
```

```
private static void reverseString(Scanner scanner) {
System.out.print("Enter a string: ");
String str = scanner.next();
StringBuilder reversed = new StringBuilder(str).reverse();
System.out.println("Reversed string: " +
reversed.toString());
}
}
```

Output:



3) **Design a Java program to cover all File related topics, demonstrating various File operations in Java. The program should allow users to perform the following tasks:**

a) **Create a new directory.**
b) **Create a new text file and write content to it.**
c) **Read the content from an existing text file.**
d) **Append new content to an existing text file.**
e) **Copy the content from one text file to another.**
f) **Delete a text file.**
g) **List all files and directories in a given directory.**
h) **Search for a specific file in a directory and its subdirectories.**
i) **Rename a file.**
j) **Get information about a file (e.g., file size, last modified time).**
k) **Requirements:**
    i) **Use File Input and Output streams for reading and writing text files.**
    ii) **Implement exception handling to handle possible errors during file operations.**
    iii) **Provide a user-friendly console interface for the user to interact with the program.**

Program:

```
package info;

import java.io.*;
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
int choice;

do {
System.out.println("Select an operation:");
System.out.println("1. Create a new directory");
System.out.println("2. Create a new text file and write
content to it");
System.out.println("3. Read the content from an existing
text file");
System.out.println("4. Append new content to an existing
text file");
System.out.println("5. Copy the content from one text file
to another");
System.out.println("6. Delete a text file");
System.out.println("7. List all files and directories in a
given directory");
System.out.println("8. Search for a specific file in a
directory and its subdirectories");
System.out.println("9. Rename a file");
```

```java
System.out.println("10. Get information about a file");
System.out.println("11. Exit");
System.out.print("Enter your choice: ");

// Input validation
while (!scanner.hasNextInt()) {
System.out.println("Invalid input! Please enter a number.");
scanner.next();
}
choice = scanner.nextInt();

switch (choice) {
case 1:
createDirectory(scanner);
break;
case 2:
createTextFile(scanner);
break;
case 3:
readTextFile(scanner);
break;
case 4:
appendTextToFile(scanner);
break;
case 5:
copyTextFile(scanner);
break;
case 6:
deleteTextFile(scanner);
break;
case 7:
listFilesInDirectory(scanner);
break;
case 8:
searchFile(scanner);
break;
case 9:
renameFile(scanner);
break;
case 10:
getFileInfo(scanner);
break;
case 11:
System.out.println("Exiting program...");
break;
default:
```

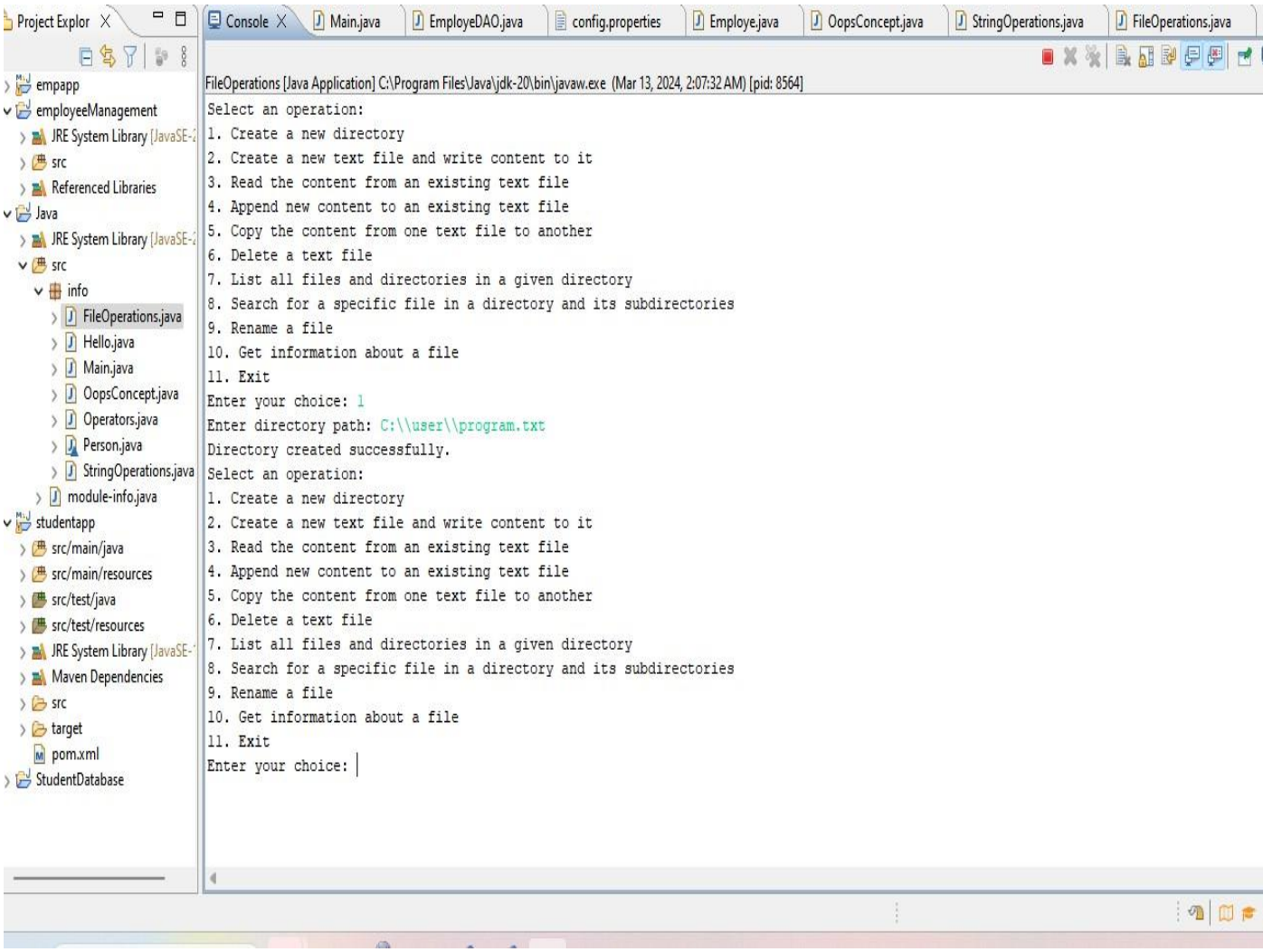
```
System.out.println("Invalid choice! Please enter a number
between 1 and 11.");
}
} while (choice != 11);

scanner.close();
}

private static void createDirectory(Scanner scanner) {
System.out.print("Enter directory path: ");
String dirPath = scanner.next();
File directory = new File(dirPath);
if (directory.exists()) {
System.out.println("Directory already exists.");
} else {
if (directory.mkdirs()) {
System.out.println("Directory created successfully.");
} else {
System.out.println("Failed to create directory.");
}
}
}

private static void createTextFile(Scanner scanner) {
System.out.print("Enter file path: ");
String filePath = scanner.next();
scanner.nextLine(); // Consume newline character left from
previous input
System.out.print("Enter content to write: ");
String content = scanner.nextLine();
try (PrintWriter writer = new PrintWriter(filePath)) {
writer.println(content);
System.out.println("Content written to file successfully.");
} catch (IOException e) {
System.out.println("Failed to write to file: " +
e.getMessage());
}
}

private static void readTextFile(Scanner scanner) {
System.out.print("Enter file path: ");
String filePath = scanner.next();
try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
String line;
while ((line = reader.readLine()) != null) {
System.out.println(line);
```

```
}
} catch (IOException e) {
System.out.println("Failed to read file: " +
e.getMessage());
}
}

private static void appendTextToFile(Scanner scanner) {
System.out.print("Enter file path: ");
String filePath = scanner.next();
scanner.nextLine(); // Consume newline character left from
previous input
System.out.print("Enter content to append: ");
String content = scanner.nextLine();
try (FileWriter writer = new FileWriter(filePath, true)) {
writer.write(content);
System.out.println("Content appended to file
successfully.");
} catch (IOException e) {
System.out.println("Failed to append to file: " +
e.getMessage());
}
}

private static void copyTextFile(Scanner scanner) {
System.out.print("Enter source file path: ");
String sourcePath = scanner.next();
System.out.print("Enter destination file path: ");
String destPath = scanner.next();
try {
Files.copy(Paths.get(sourcePath), Paths.get(destPath),
StandardCopyOption.REPLACE_EXISTING);
System.out.println("File copied successfully.");
} catch (IOException e) {
System.out.println("Failed to copy file: " +
e.getMessage());
}
}

private static void deleteTextFile(Scanner scanner) {
System.out.print("Enter file path: ");
String filePath = scanner.next();
File file = new File(filePath);
if (file.exists() && file.isFile()) {
if (file.delete()) {
System.out.println("File deleted successfully.");
} else {
```

```
System.out.println("Failed to delete file.");
}
} else {
System.out.println("File does not exist.");
}
}

private static void listFilesInDirectory(Scanner scanner) {
System.out.print("Enter directory path: ");
String dirPath = scanner.next();
File directory = new File(dirPath);
if (directory.exists() && directory.isDirectory()) {
File[] files = directory.listFiles();
System.out.println("Files and directories in " + dirPath +
":");
for (File file : files) {
System.out.println(file.getName());
}
} else {
System.out.println("Directory does not exist.");
}
}

private static void searchFile(Scanner scanner) {
System.out.print("Enter directory path to search: ");
String dirPath = scanner.next();
scanner.nextLine(); // Consume newline character left from
previous input
System.out.print("Enter file name to search: ");
String fileName = scanner.nextLine();
File directory = new File(dirPath);
if (directory.exists() && directory.isDirectory()) {
File[] files = directory.listFiles();
boolean found = false;
for (File file : files) {
if (file.getName().equals(fileName)) {
found = true;
System.out.println("File found at: " +
file.getAbsolutePath());
break;
}
}
if (!found) {
System.out.println("File not found in directory and its
subdirectories.");
}
} else {
```

```
System.out.println("Directory does not exist.");
}
}

private static void renameFile(Scanner scanner) {
System.out.print("Enter current file path: ");
String currentFilePath = scanner.next();
System.out.print("Enter new file path: ");
String newFilePath = scanner.next();
File currentFile = new File(currentFilePath);
File newFile = new File(newFilePath);
if (currentFile.exists() && currentFile.isFile()) {
if (currentFile.renameTo(newFile)) {
System.out.println("File renamed successfully.");
} else {
System.out.println("Failed to rename file.");
}
} else {
System.out.println("File does not exist.");
}
}

private static void getFileInfo(Scanner scanner) {
System.out.print("Enter file path: ");
String filePath = scanner.next();
File file = new File(filePath);
if (file.exists() && file.isFile()) {
System.out.println("File size: " + file.length() + "
bytes");
System.out.println("Last modified time: " +
file.lastModified());
} else {
System.out.println("File does not exist.");
}
}
}
```

Output:



4) **Design a Java program to implement a Collection Management System that manages different types of collections such as lists, sets, and maps. The program should allow users to perform the following operations for each type of collection:**

   a) **Lists:**
      i) **Add an element: The user can add an element to the list.**
      ii) **Remove an element: The user can remove an element from the list.**
      iii) **Display all elements: The user can view all elements in the list.**
   b) **Sets:**
      i) **Add an element: The user can add an element to the set.**

ii) Remove an element: The user can remove an element from the set.
iii) Display all elements: The user can view all elements in the set.

c) Maps:
i) Add a key-value pair: The user can add a key-value pair to the map.
ii) Remove a key-value pair: The user can remove a key-value pair from the map.
iii) Display all key-value pairs: The user can view all key-value pairs in the map.

d) Requirements:
i) Implement separate classes for each type of collection (ListManager, SetManager, MapManager).
ii) Use appropriate collection classes (e.g., ArrayList, LinkedList, HashSet, TreeMap) to store the elements and key-value pairs.
iii) Use inheritance and polymorphism to manage different types of collections.
iv) Implement exception handling to handle possible errors (e.g., element not found in the list/set, duplicate keys in the map).
v) Provide a user-friendly console interface for the user to interact with the Collection Management System.

e)Cover all Java collections topics, including Lists, Sets, and Maps

Program:

```
package info;
import java.util.*;

// Interface for common operations
interface CollectionManager<T> {
void add(T element);
void remove(T element);
void displayAll();
}

// ListManager class
class ListManager<T> implements CollectionManager<T> {
private List<T> list;

public ListManager() {
this.list = new ArrayList<>();
}

@Override
public void add(T element) {
list.add(element);
System.out.println("Element added to the list: " + element);
}

@Override
public void remove(T element) {
if (list.remove(element)) {
System.out.println("Element removed from the list: " +
element);
} else {
System.out.println("Element not found in the list.");
}
}

@Override
public void displayAll() {
System.out.println("Elements in the list:");
for (T element : list) {
System.out.println(element);
}
}
}

// SetManager class
class SetManager<T> implements CollectionManager<T> {
```

```java
private Set<T> set;

public SetManager() {
this.set = new HashSet<>();
}

@Override
public void add(T element) {
if (set.add(element)) {
System.out.println("Element added to the set: " + element);
} else {
System.out.println("Element already exists in the set.");
}
}

@Override
public void remove(T element) {
if (set.remove(element)) {
System.out.println("Element removed from the set: " +
element);
} else {
System.out.println("Element not found in the set.");
}
}

@Override
public void displayAll() {
System.out.println("Elements in the set:");
for (T element : set) {
System.out.println(element);
}
}
}

// MapManager class
class MapManager<K, V> implements
CollectionManager<Map.Entry<K, V>> {
private Map<K, V> map;

public MapManager() {
this.map = new TreeMap<>();
}

@Override
public void add(Map.Entry<K, V> entry) {
if (map.put(entry.getKey(), entry.getValue()) == null) {
```

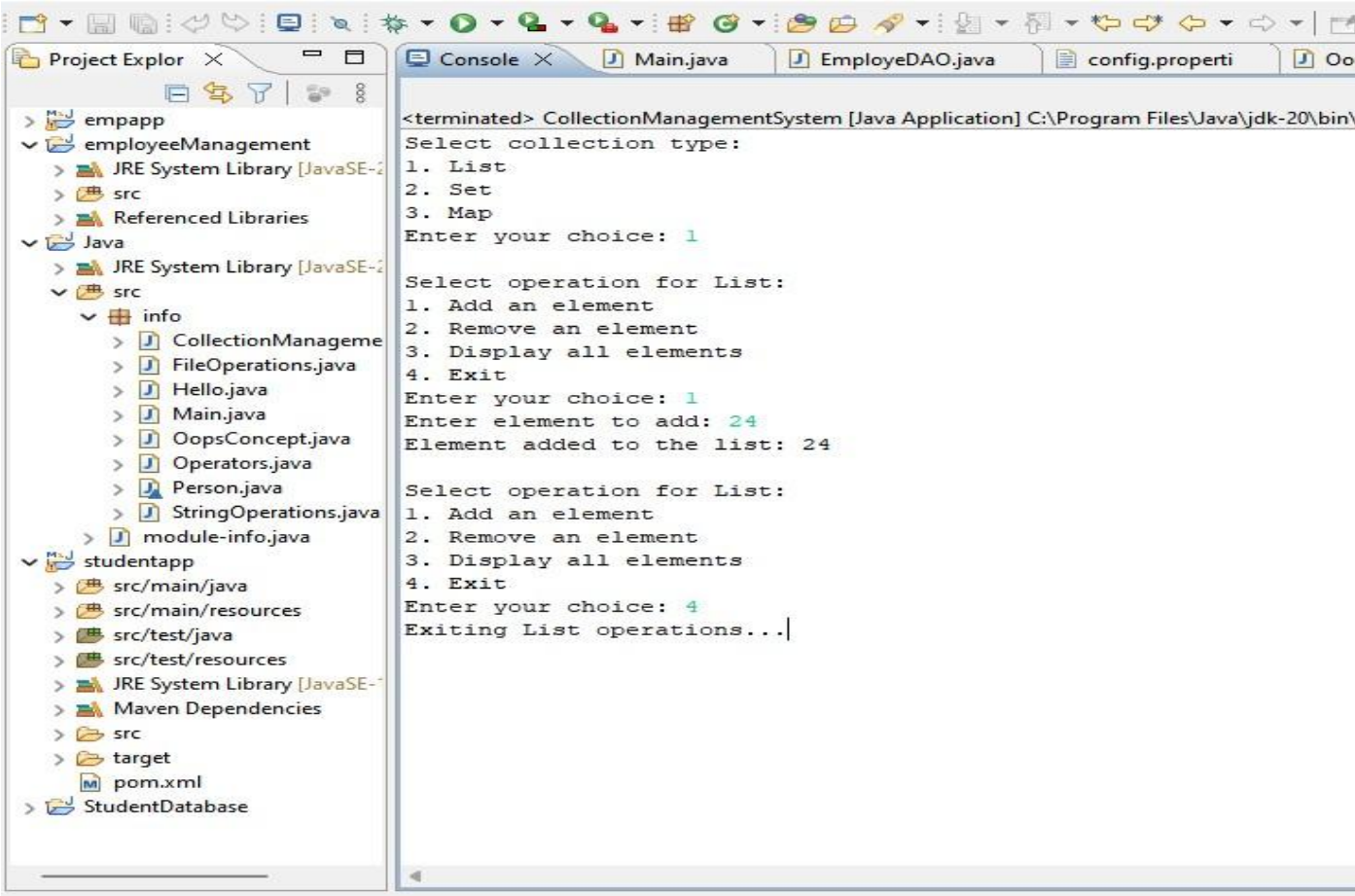
```
System.out.println("Key-value pair added to the map: " +
entry.getKey() + " -> " + entry.getValue());
} else {
System.out.println("Key already exists in the map. Use
update operation for key: " + entry.getKey());
}
}

@Override
public void remove(Map.Entry<K, V> entry) {
if (map.remove(entry.getKey()) != null) {
System.out.println("Key-value pair removed from the map: "
+ entry.getKey() + " -> " + entry.getValue());
} else {
System.out.println("Key-value pair not found in the map.");
}
}

@Override
public void displayAll() {
System.out.println("Key-value pairs in the map:");
for (Map.Entry<K, V> entry : map.entrySet()) {
System.out.println(entry.getKey() + " -> " +
entry.getValue());
}
}
}

// Main class for user interaction
public class CollectionManagementSystem {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

// Menu for selecting collection type
System.out.println("Select collection type:");
System.out.println("1. List");
System.out.println("2. Set");
System.out.println("3. Map");
System.out.print("Enter your choice: ");
int choice = scanner.nextInt();

switch (choice) {
case 1:
operateList(scanner);
break;
case 2:
operateSet(scanner);
```

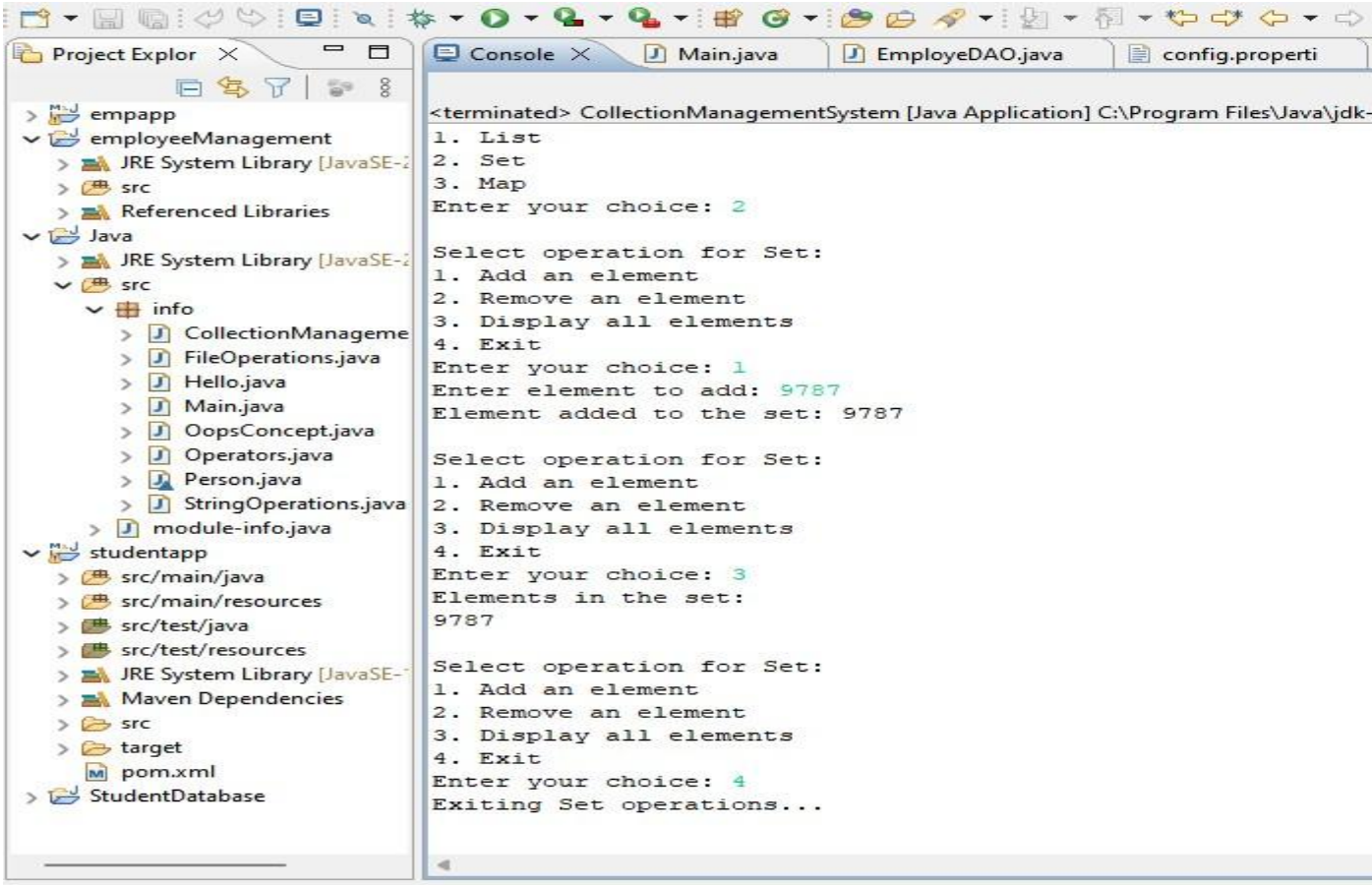
```
break;
case 3:
operateMap(scanner);
break;
default:
System.out.println("Invalid choice!");
}

scanner.close();
}

private static void operateList(Scanner scanner) {
ListManager<String> listManager = new ListManager<>();
int operation;

do {
System.out.println("\nSelect operation for List:");
System.out.println("1. Add an element");
System.out.println("2. Remove an element");
System.out.println("3. Display all elements");
System.out.println("4. Exit");
System.out.print("Enter your choice: ");
operation = scanner.nextInt();

switch (operation) {
case 1:
System.out.print("Enter element to add: ");
scanner.nextLine(); // Consume newline character left from
previous input
String element = scanner.nextLine();
listManager.add(element);
break;
case 2:
System.out.print("Enter element to remove: ");
scanner.nextLine(); // Consume newline character left from
previous input
String removeElement = scanner.nextLine();
listManager.remove(removeElement);
break;
case 3:
listManager.displayAll();
break;
case 4:
System.out.println("Exiting List operations...");
break;
default:
System.out.println("Invalid operation!");
```

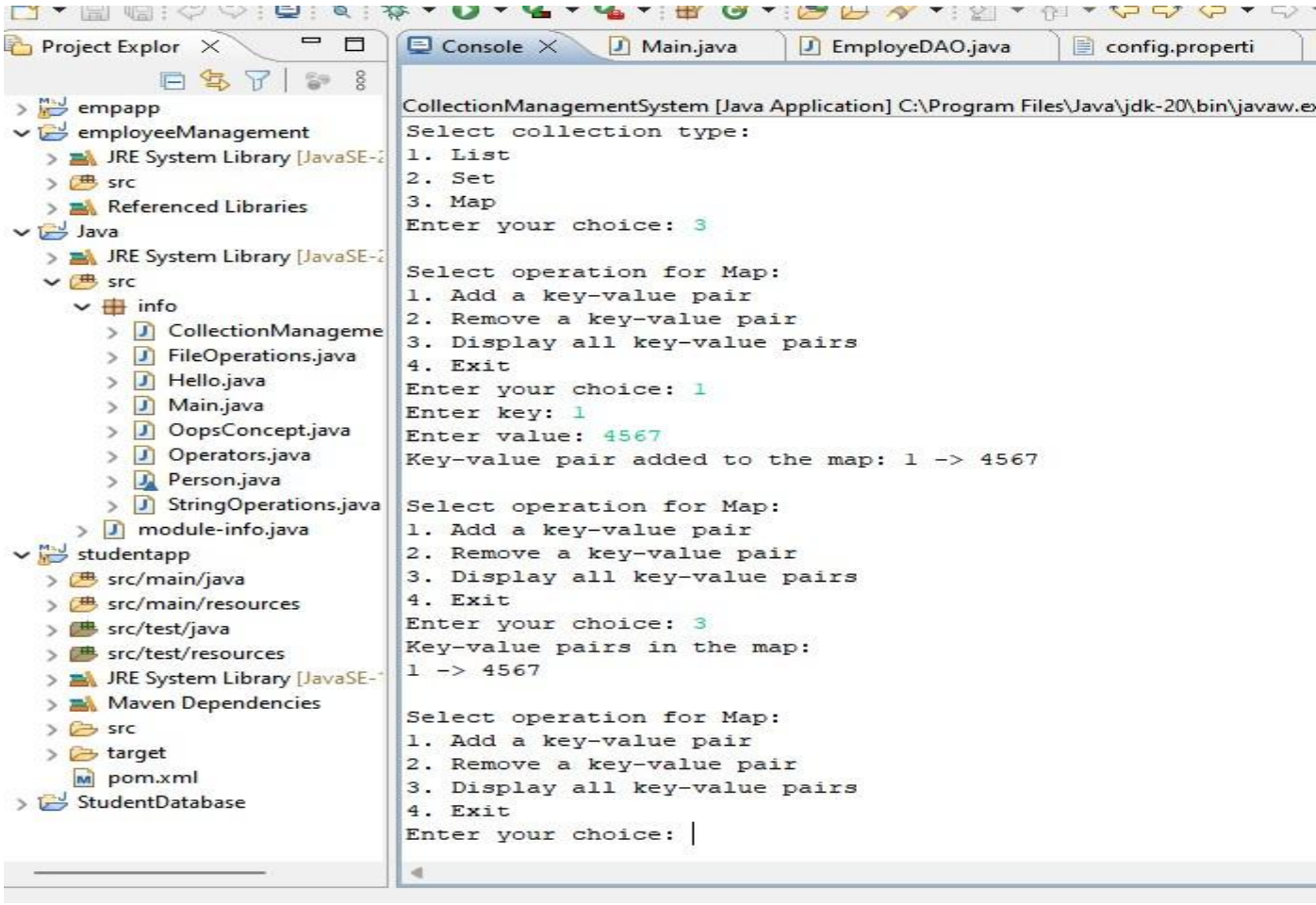
```
}
} while (operation != 4);
}

private static void operateSet(Scanner scanner) {
SetManager<String> setManager = new SetManager<>();
int operation;

do {
System.out.println("\nSelect operation for Set:");
System.out.println("1. Add an element");
System.out.println("2. Remove an element");
System.out.println("3. Display all elements");
System.out.println("4. Exit");
System.out.print("Enter your choice: ");
operation = scanner.nextInt();

switch (operation) {
case 1:
System.out.print("Enter element to add: ");
setManager.add(scanner.next());
break;
case 2:
System.out.print("Enter element to remove: ");
setManager.remove(scanner.next());
break;
case 3:
setManager.displayAll();
break;
case 4:
System.out.println("Exiting Set operations...");
break;
default:
System.out.println("Invalid operation!");
}
} while (operation != 4);
}

private static void operateMap(Scanner scanner) {
MapManager<String, Integer> mapManager = new MapManager<>();
int operation;

do {
System.out.println("\nSelect operation for Map:");
System.out.println("1. Add a key-value pair");
System.out.println("2. Remove a key-value pair");
System.out.println("3. Display all key-value pairs");
```

```
System.out.println("4. Exit");
System.out.print("Enter your choice: ");
operation = scanner.nextInt();

switch (operation) {
case 1:
System.out.print("Enter key: ");
String key = scanner.next();
System.out.print("Enter value: ");
int value = scanner.nextInt();
mapManager.add(new AbstractMap.SimpleEntry<>(key, value));
break;
case 2:
System.out.print("Enter key to remove: ");
mapManager.remove(new
AbstractMap.SimpleEntry<>(scanner.next(), null));
break;
case 3:
mapManager.displayAll();
break;
case 4:
System.out.println("Exiting Map operations...");
break;
default:
System.out.println("Invalid operation!");
}
} while (operation != 4);
}
}
```

Output:


Project Explor
empapp
employeeManagement
JRE System Library [JavaSE-
src
Referenced Libraries
Java
JRE System Library [JavaSE-
src
info
CollectionManageme
FileOperations.java
Hello.java
Main.java
OopsConcept.java
Operators.java
Person.java
StringOperations.java
module-info.java
studentapp
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-
Maven Dependencies
src
target
pom.xml
StudentDatabase
Console
Main.java
EmployeDAO.java
config.properti
<terminated> CollectionManagementSystem [Java Application] C:\Program Files\Java\jdk-20\bin\
Select collection type:
1. List
2. Set
3. Map
Enter your choice: 1
Select operation for List:
1. Add an element
2. Remove an element
3. Display all elements
4. Exit
Enter your choice: 1
Enter element to add: 24
Element added to the list: 24
Select operation for List:
1. Add an element
2. Remove an element
3. Display all elements
4. Exit
Enter your choice: 4
Exiting List operations...


Project Explor
empapp
employeeManagement
JRE System Library [JavaSE-2
src
Referenced Libraries
Java
JRE System Library [JavaSE-2
src
info
CollectionManageme
FileOperations.java
Hello.java
Main.java
OopsConcept.java
Operators.java
Person.java
StringOperations.java
module-info.java
studentapp
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1
Maven Dependencies
src
target
pom.xml
StudentDatabase
Console
Main.java
EmployeDAO.java
config.properti
<terminated> CollectionManagementSystem [Java Application] C:\Program Files\Java\jdk-
1. List
2. Set
3. Map
Enter your choice: 2
Select operation for Set:
1. Add an element
2. Remove an element
3. Display all elements
4. Exit
Enter your choice: 1
Enter element to add: 9787
Element added to the set: 9787
Select operation for Set:
1. Add an element
2. Remove an element
3. Display all elements
4. Exit
Enter your choice: 3
Elements in the set:
9787
Select operation for Set:
1. Add an element
2. Remove an element
3. Display all elements
4. Exit
Enter your choice: 4
Exiting Set operations...



5) **Add new employees: The user can add details like employee ID, name, department, and salary.**
    a) **Update employee details: The user can update the name, department, or salary of an existing employee based on their employee ID.**
    b) **Delete an employee: The user can delete an employee from the system based on their employee ID.**
    c) **Display all employees: The user can view a list of all employees and their details.**
    d) **Search for an employee: The user can search for an employee by their employee ID and view their details.**
    e) **Requirements:**
        i) **Use Object-Oriented Programming (OOP) principles and create an Employee class with appropriate attributes and methods.**

ii) **Use appropriate data structures (e.g., ArrayList, HashMap) to store the employee data.**
iii) **Implement exception handling to handle possible errors (e.g., invalid employee ID, input validation).**
iv) **Provide a user-friendly console interface for the user to interact with the Employee Management System.**

Program:

```
package info;

import java.util.*;

class Employee {
private int id;
private String name;
private String department;
private double salary;

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

public int getId() {
return id;
}

public String getName() {
return name;
}

public String getDepartment() {
return department;
}

public double getSalary() {
return salary;
}

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

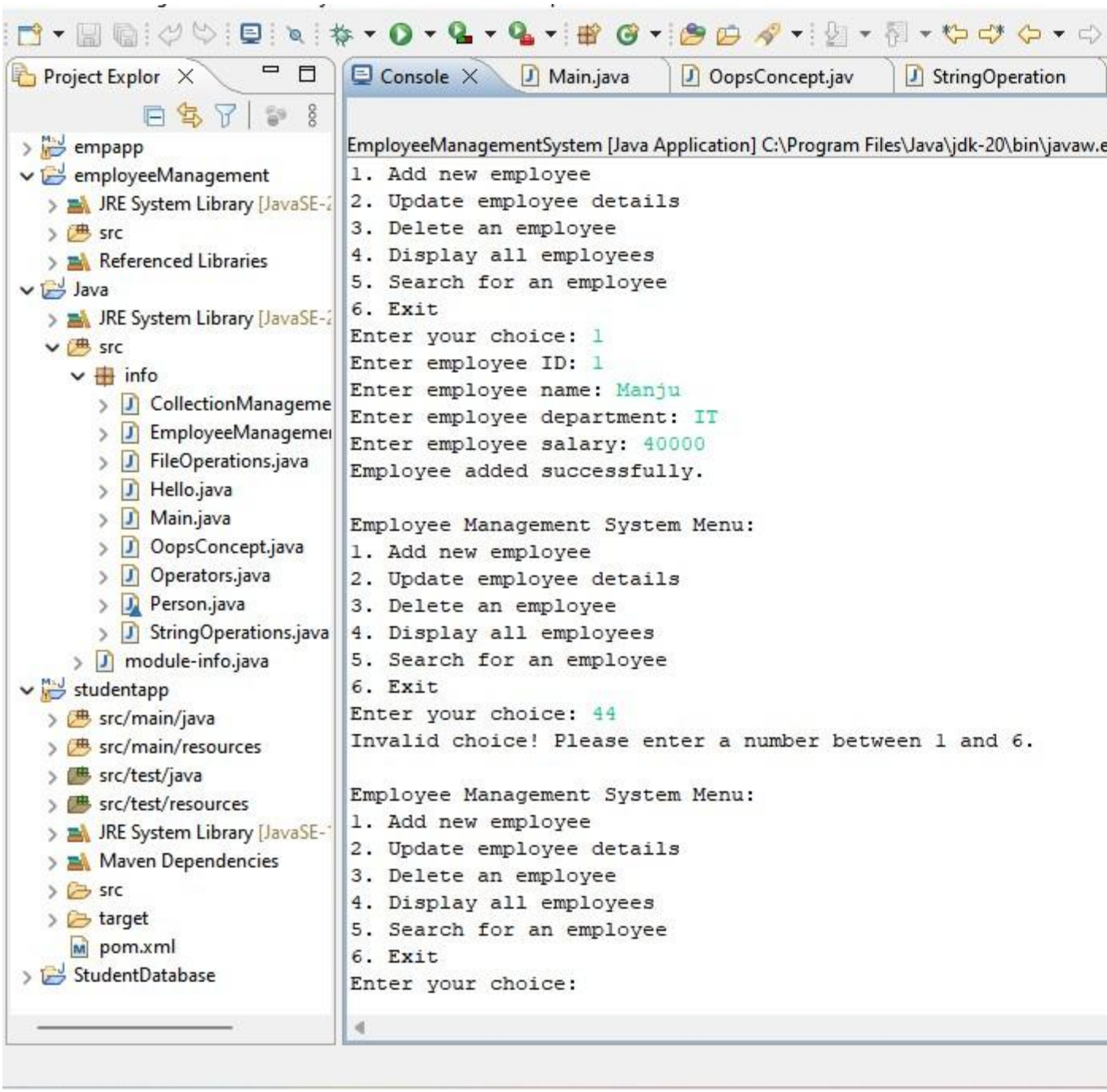
```java
public void setDepartment(String department) {
this.department = department;
}

public void setSalary(double salary) {
this.salary = salary;
}

@Override
public String toString() {
return "Employee ID: " + id + ", Name: " + name + ",
Department: " + department + ", Salary: " + salary;
}
}

public class EmployeeManagementSystem {
private List<Employee> employees;

public EmployeeManagementSystem() {
this.employees = new ArrayList<>();
}

public void addEmployee(int id, String name, String
department, double salary) {
Employee employee = new Employee(id, name, department,
salary);
employees.add(employee);
System.out.println("Employee added successfully.");
}

public void updateEmployee(int id, String name, String
department, double salary) {
for (Employee employee : employees) {
if (employee.getId() == id) {
employee.setName(name);
employee.setDepartment(department);
employee.setSalary(salary);
System.out.println("Employee details updated
successfully.");
return;
}
}
System.out.println("Employee not found with ID: " + id);
}

public void deleteEmployee(int id) {
```

```
Iterator<Employee> iterator = employees.iterator();
while (iterator.hasNext()) {
Employee employee = iterator.next();
if (employee.getId() == id) {
iterator.remove();
System.out.println("Employee deleted successfully.");
return;
}
}
System.out.println("Employee not found with ID: " + id);
}

public void displayAllEmployees() {
if (employees.isEmpty()) {
System.out.println("No employees found.");
} else {
System.out.println("List of all employees:");
for (Employee employee : employees) {
System.out.println(employee);
}
}
}

public void searchEmployee(int id) {
for (Employee employee : employees) {
if (employee.getId() == id) {
System.out.println("Employee details:");
System.out.println(employee);
return;
}
}
System.out.println("Employee not found with ID: " + id);
}

public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
EmployeeManagementSystem system = new
EmployeeManagementSystem();

int choice;
do {
System.out.println("\nEmployee Management System Menu:");
System.out.println("1. Add new employee");
System.out.println("2. Update employee details");
System.out.println("3. Delete an employee");
System.out.println("4. Display all employees");
System.out.println("5. Search for an employee");
```

```
System.out.println("6. Exit");
System.out.print("Enter your choice: ");

while (!scanner.hasNextInt()) {
System.out.println("Invalid input! Please enter a number.");
scanner.next();
}
choice = scanner.nextInt();

switch (choice) {
case 1:
System.out.print("Enter employee ID: ");
int id = scanner.nextInt();
scanner.nextLine(); // Consume newline character left from
previous input
System.out.print("Enter employee name: ");
String name = scanner.nextLine();
System.out.print("Enter employee department: ");
String department = scanner.nextLine();
System.out.print("Enter employee salary: ");
double salary = scanner.nextDouble();
system.addEmployee(id, name, department, salary);
break;
case 2:
System.out.print("Enter employee ID to update: ");
int updateId = scanner.nextInt();
scanner.nextLine(); // Consume newline character left from
previous input
System.out.print("Enter new name: ");
String newName = scanner.nextLine();
System.out.print("Enter new department: ");
String newDept = scanner.nextLine();
System.out.print("Enter new salary: ");
double newSalary = scanner.nextDouble();
system.updateEmployee(updateId, newName, newDept,
newSalary);
break;
case 3:
System.out.print("Enter employee ID to delete: ");
int deleteId = scanner.nextInt();
system.deleteEmployee(deleteId);
break;
case 4:
system.displayAllEmployees();
break;
case 5:
System.out.print("Enter employee ID to search: ");
```

```
int searchId = scanner.nextInt();
system.searchEmployee(searchId);
break;
case 6:
System.out.println("Exiting program...");
break;
default:
System.out.println("Invalid choice! Please enter a number
between 1 and 6.");
}
} while (choice != 6);

scanner.close();
}
}
```

Output:


Project Explor
empapp
employeeManagement
JRE System Library [JavaSE-2
src
Referenced Libraries
Java
JRE System Library [JavaSE-2
src
info
CollectionManageme
EmployeeManageme
FileOperations.java
Hello.java
Main.java
OopsConcept.java
Operators.java
Person.java
StringOperations.java
module-info.java
studentapp
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-
Maven Dependencies
src
target
pom.xml
StudentDatabase
Console
Main.java
OopsConcept.jav
StringOperation
EmployeeManagementSystem [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.e
1. Add new employee
2. Update employee details
3. Delete an employee
4. Display all employees
5. Search for an employee
6. Exit
Enter your choice: 1
Enter employee ID: 1
Enter employee name: Manju
Enter employee department: IT
Enter employee salary: 40000
Employee added successfully.
Employee Management System Menu:
1. Add new employee
2. Update employee details
3. Delete an employee
4. Display all employees
5. Search for an employee
6. Exit
Enter your choice: 44
Invalid choice! Please enter a number between 1 and 6.
Employee Management System Menu:
1. Add new employee
2. Update employee details
3. Delete an employee
4. Display all employees
5. Search for an employee
6. Exit
Enter your choice: