OOAD

LAB - 6

Name : Sanmat Sanjayakumar Payagoudar

SRN : PES1UG20CS385

Section : G

Serialization

Serialization is the process of converting an object into a stream of bytes, which can be easily transmitted over the network or stored on disk. The serialized data can be later deserialized back into the object. In Java, the `java.io.Serializable` interface is used to mark classes that can be serialized.

Deserialization

Deserialization is the process of reconstructing an object from its serialized form. When an object is deserialized, a new instance of the object is created in memory and the serialized data is used to initialize its state. In Java, the `java.io.ObjectInputStream` class is used to deserialize objects.

HashMap

HashMap is a data structure in Java that stores key-value pairs. It provides constant-time performance for basic operations such as get() and put(). The key-value pairs in a HashMap are stored in an array of linked lists, where each element in the array is a bucket that can hold multiple entries. The key is used to calculate the index of the bucket, and the value is stored in a node in the linked list.

HashMap is useful for storing and retrieving configuration values, as it allows quick and efficient access to values based on their keys. By using a HashMap to manage configuration values, we can easily insert, update, and retrieve values in a key-value pair format.

Applications of Serialization :

1. Saving and restoring application state: Serialization is commonly used to save and restore the state of an application. By serializing an object's state, we can save it to a file or database and restore it later, even after the application has been closed and reopened.
2. Network communication: Serialization is also used to transfer data between applications over a network. By serializing an object and sending it over the network, we can send complex data structures without having to worry about the details of the network communication.
3. Caching: Serialization can also be used to cache frequently accessed data. By serializing the data and storing it in memory or on disk, we can avoid the overhead of repeatedly generating the data.
4. Object persistence: Serialization is commonly used to persist objects to a database or file system. By serializing an object and saving it to a file or database, we can ensure that the object can be restored later with its full state intact.
5. Remote procedure calls: Serialization is used in remote procedure calls (RPCs) to pass data between the client and server. By serializing the data and passing it over the network, we can make it easier for applications to communicate with each other.

CODE :

```
import java.io.*;
import java.util.*;

public class ConfigurationHandler {
   void saveConfig(int isNew, Map<String, String> config) {
      try {
         if (isNew == 1) {
            Map<String, String> defaultConfig = new HashMap<>();
            defaultConfig.put("path", null);
            defaultConfig.put("version", null);
            defaultConfig.put("sysName", null);
            FileOutputStream file = new FileOutputStream("config.cfg");
            ObjectOutputStream out = new ObjectOutputStream(file);
            out.writeObject(defaultConfig);
            out.close();
            file.close();
         } else {
            FileOutputStream file = new FileOutputStream("config.cfg");
            ObjectOutputStream out = new ObjectOutputStream(file);
            out.writeObject(config);
            out.close();
            file.close();
         }
      } catch (IOException e) {
         System.out.println(e);
      }
   }

   void readConfig() {
```

```
    try {
        Map<String, String> config = null;
        FileInputStream file = new FileInputStream("config.cfg");
        ObjectInputStream in = new ObjectInputStream(file);
        config = (Map<String, String>) in.readObject();
        System.out.println(config);
        in.close();
        file.close();
    } catch (IOException | ClassNotFoundException e) {
        System.out.println(e);
    }
}

void updateConfig(String key, String value) {
    try {
        Map<String, String> config = null;
        FileInputStream file = new FileInputStream("config.cfg");
        ObjectInputStream in = new ObjectInputStream(file);
        config = (Map<String, String>) in.readObject();
        if (config.containsKey("Jim")) {
            config.remove(key);
            config.put(key, value);
        } else {
            config.put(key, value);
        }
        saveConfig(0, config);
        in.close();
        file.close();
        System.out.println("Updated.");
    } catch (IOException | ClassNotFoundException e) {
        System.out.println(e);
```

```
        }
    }

    public static void main(String[] args) {
        // Check if configuration file exists
        File configFile = new File("config.cfg");
        Scanner scanner = new Scanner(System.in);
        ConfigurationHandler configHandler = new ConfigurationHandler();
        while (true) {
            if (configFile.exists()) {
                System.out.println("Configuration file exists.");
                configHandler.readConfig();
                System.out.print("Enter key: ");
                String key = scanner.next();
                if (key.equals("exit"))
                    break;
                System.out.print("Enter value: ");
                String value = scanner.next();
                configHandler.updateConfig(key, value);
                configHandler.readConfig();
            } else {
                System.out.println("Configuration file does not exist.");
                configHandler.saveConfig(1, null);
                System.out.println("Configuration file created.");
            }
        }
        scanner.close();
    }
}
```

OUT PUT:

```
Enter key: SRN
Enter value: PES1UG20CS385
Updated
{path=null, sysName=null, version=null, srn=8, SRN=PES1UG20CS385}
Configuration file exists
{path=null, sysName=null, version=null, srn=8, SRN=PES1UG20CS385}
```