Medii si instrumente de programare

Utilizarea laboratoarelor de java in

Proiect TODO List

Dajboc Tudor-Gabriel

Introducere

Aplicatia TODO List a fost dezvoltata folosind cunostintele dobandite pe parcursul laboratorul de java. TODO List incearca sa rezolve problema gestionarii activitatilor de zi cu zi pe care trebuie sa le facem.

Introducerea in JAVA

Am folosit diverse tipuri de date si metode de output si input. Printre care ,in cazul scrierii in fisiere am folosit clasa FileWriter pentru a simplifica procesul:

```
try(FileWriter fWriter = new FileWriter( fileName: "export.txt")) {

    fWriter.write( str: this.lists.size() + "!");
    for(List list : this.lists)
    {
        numOfTasks+= list.getTasks().size();
        fWriter.write(list.toString());
    }
    fWriter.write( str: numOfTasks + "!");
```

Iar pentru citirea din fisier am folosit un simplu scanner caruia i-am dat un delimitator:

```
File file = new File( pathname: "export.txt");

Scanner sc = new Scanner(file);
sc.useDelimiter( pattern: "!");
```

Colectii java

In toate cazurile in care am avut nevoie, am folosit ArrayList<> deoarece a fost solutia cea mai simpla pentru utilizarea vectorilor:

```
private ArrayList<Task> tasks;
```

```
Task task = new Task(id, this.id, name, description);
this.tasks.add(task);
```

Clase JAVA

In tot proiectul am folosit 3 clase java (facand exceptie de clasa main): App, List, Task. Fiecare clasa contine parametri si metode specifice problemelor pe care le rezolva. Clasa App se ocupa de rularea aplicatiei si a interactiunii cu utilizatorul, iar clasele List si Task se ocupa cu abstractizarea datelor folosite in aplicatie.

```
public class App implements IApp {  2 usages

    ArrayList<List> lists;  17 usages
    int currentScreen;  12 usages
    int input;  14 usages
    Scanner in;  25 usages
    int currentListId;  7 usages


    public App() {  1 usage
        lists = new ArrayList<>();
        currentScreen = 0;
        input = 1;
        currentListId = 0;
        in = new Scanner(System.in);
    }
```

Interfete in JAVA

Am creat interfete pentru fiecare clasa in parte pentru a simplifica interactiunea cu clasele:

```
package interfaces;

public interface IApp {  4 usages  1 implementation
    public void startApp();  1 usage  1 implementation
    public void closeApp();  1 usage  1 implementation
}
```

```
public interface ITask {  2 usages  1 implementation
    int getId();  1 usage  1 implementation
    void setId(int id);  1 usage  1 implementation
    String getName();  1 implementation
    void setName(String name);  no usages  1 implementation
    String getDescription();  1 usage  1 implementation
    void setDescription(String description);  no usages  1 implementation
    boolean isDone();  1 usage  1 implementation
    void setDone(boolean done);  2 usages  1 implementation
}
```

Teste pentru fiecare metoda

Pentru a demonstra si verifica calitatea datelor ce rezulta din aplicatie, am creat pentru toate metodele cate un unit test:

```
void testCreateList() {
    App app = new App();
    app.createList( name: "List1");
    app.createList( name: "List2");
    assertEquals(app.lists.get(0).getId(), actual: 1);
    assertEquals(app.lists.get(1).getId(), actual: 2);
    assertEquals(app.lists.get(0).getName(), actual: "List1");
    assertEquals(app.lists.get(1).getName(), actual: "List2");
}

@Test  Dajboc Tudor
void testDeleteList() {
    App app = new App();
    app.createList( name: "List1");
    app.createList( name: "List2");
    app.deleteList( id: 2);
    assertEquals(app.lists.size(), actual: 1);
    assertEquals(app.lists.get(0).getId(), actual: 1);
    assertEquals(app.lists.get(0).getName(), actual: "List1");
}

@Test  Dajboc Tudor
void exportToFile() throws FileNotFoundException {
    App app = new App();
    app.createList( name: "List1");
    app.createList( name: "List2");
    app.lists.get(0).createTask( name: "task1", description: "this is task1");
    app.exportToFile();
    File file = new File( pathname: "export.txt");

    Scanner sc = new Scanner(file);

    String result = sc.nextLine();
    assertEquals(result, actual: "2!1!List1!2!List2!1!0!task1!this is task1!false!1!");
}
```

Persistenta datelor

Pentru a asigura persistenta datelor intre sesiuni, am dezvoltat 2 metode ce se ocupa cu export-ul datelor si cu import-ul acestora la deschiderea aplicatiei:

```
public void exportToFile()  2 usages  Dajboc Tudor
{

    int numOfTasks = 0;

    try(FileWriter fWriter = new FileWriter( fileName: "export.txt")) {

        fWriter.write( str: this.lists.size() + "!");
        for(List list : this.lists)
        {
            numOfTasks+= list.getTasks().size();
            fWriter.write(list.toString());
        }
        fWriter.write( str: numOfTasks + "!");
        for(List list : this.lists)
        {
            for(Task task : list.getTasks())
            {
                fWriter.write(task.toString());
            }
        }
    }
    catch (IOException e)
    {

        // Print the exception
        System.out.print(e.getMessage());
    }
}
```

```java
public void importFromFile()  1 usage  Dajboc Tudor
{
    try
    {
        File file = new File( pathname: "export.txt");

        Scanner sc = new Scanner(file);
        sc.useDelimiter( pattern: "!");

        int numOfLists = sc.nextInt();
        System.out.println(numOfLists);
        System.out.println("test");
        System.out.println("testint");
        int listId;
        int taskId;
        boolean done;
        String description;
        String taskName;
        String listName;

        System.out.println("test1");
        for(int i = 0; i < numOfLists; i++)
        {
            listId = sc.nextInt();
            listName = sc.next();
            this.createList(listName);

        }
```