

Padrões de Entrada/Saída

Lista 09

Instruções de Entrega

Nome do projeto java: ListaX-NUSP1-NUSP2. Sendo X o número da lista para ser realizado em duplas; NUSP1 e NUSP2 são os números USP dos participantes.

Entregar o projeto zipado, **apenas um projeto por dupla**. *Entrega normal no e-disciplinas*

Instruções Gerais

Vale a mesma observação da aula anterior, o uso de ferramentas de busca como o Google, a documentação do Java, sites sobre programação como o StackOverflow, ou mesmo a Wikipedia será essencial, pois a ideia é vocês pesquisarem como resolver o problema. Por isso, antes de tentarem resolver algum exercício, procurem um pouco e vejam como resolvê-lo de maneira correta.

Exercício 1 (3,0)

Usando uma **interface DAO**, que deve ser instanciada de duas formas diferentes, crie um programa que compara o desempenho entre imprimir um texto num arquivo e imprimir o mesmo texto na saída do terminal. Qual é mais rápido? Explique a diferença de desempenho.

Nota: O texto de entrada deve ser compostos pelas linhas em um arquivo de entrada. A saída deverá ser idêntica, tanto no arquivo de saída quanto na tela. Recomendamos usar alguma versão da bíblia (em português ou espanhol) disponível livremente na internet em txt para este exercício. Pode usar um fragmento da bíblia, se a coisa ficar muito demorada. Entregar o arquivo de entrada junto com o seu projeto (dentro dele)¹.

Exercício 2 (5,0)

Considere os seguintes códigos²:

Primeiro, a classe Student:

¹Seja razoável, **não** mande a bíblia inteira.

²http://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

```

public class Student {
    private String name;
    private int rollNo;

    Student(String name, int rollNo){
        this.name = name;
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }

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

    public int getRollNo() {
        return rollNo;
    }

    public void setRollNo(int rollNo) {
        this.rollNo = rollNo;
    }
}

```

Em seguida, uma interface DAO (Data Access Object).

```

import java.util.List;

public interface StudentDao {
    public List<Student> getAllStudents();
    public Student getStudent(int rollNo);
    public void updateStudent(Student student);
    public void deleteStudent(Student student);
}

```

Considere então uma implementação concreta desta interface, usando uma lista como banco de dados.

```

import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {
    //list is working as a database
    List<Student> students;
}

```

```

public StudentDaoImpl(){
    students = new ArrayList<Student>();
    Student student1 = new Student("Robert",0);
    Student student2 = new Student("John",1);
    students.add(student1);
    students.add(student2);
}
@Override
public void deleteStudent(Student student) {
    students.remove(student.getRollNo());
    System.out.println(
        "Student: Roll No " +
        student.getRollNo() +
        ", deleted from database"
    );
}

//retrive list of students from the database
@Override
public List<Student> getAllStudents() {
    return students;
}

@Override
public Student getStudent(int rollNo) {
    return students.get(rollNo);
}

@Override
public void updateStudent(Student student) {
    students
        .get(student.getRollNo())
        .setName(student.getName());
    System.out.println(
        "Student: Roll No " +
        student.getRollNo() +
        ", updated in the database"
    );
}
}
}

```

Finalmente, um exemplo de main usando DAO.

```

public class DaoPatternDemo {
    public static void main(String[] args) {
        StudentDao studentDao = new StudentDaoImpl();
    }
}

```

```

//print all students
for (Student student : studentDao.getAllStudents()) {
    System.out.println(
        "Student: [RollNo : " +
        student.getRollNo() +
        ", Name : " +
        student.getName() +
        "]"
    );
}

//update student
Student student = studentDao.getAllStudents().get(0);
student.setName("Michael");
studentDao.updateStudent(student);

//get the student
studentDao.getStudent(0);
System.out.println(
    "Student: [RollNo : " +
    student.getRollNo() +
    ", Name : " +
    student.getName() +
    "]"
);
}
}

```

Agora, implemente uma nova classe que implementa a interface `StudentDao` para ler e salvar os dados dos estudantes a partir de um arquivo de texto. Teste a classe criada (e somente ela) usando o JUnit³.

Dica 1: para facilitar, você pode escrever no arquivo usando o formato `.csv` (comma-separated values). Isso facilitará a análise posterior dos dados. Você pode usar classes pré-existentes que manipulam arquivos CSV em Java. Cuidado, existem DOIS formatos CSV (por quê?), e se seu manipulador de planilhas estiver em português, isso pode gerar um problema de visualização de dados.

Dica 2: É melhor ter um arquivo de entrada e outro arquivo de saída, assim você nunca perde os dados de entrada.

³Não é necessário testar se o arquivo é criado de fato, mas é necessário testar se você está lendo do arquivo corretamente. Se for necessário, pode incluir um arquivo de exemplo com o código do programa de vocês.

Exercício 3 (2,0)

Crie um programa que mostre a diferença entre `PrintWriter` e `OutputStreams`. Por que esta diferença ocorre?

Dica: uma possível maneira de mostrar essa diferença seria usando um gerador de bytes aleatórios como o `Random.nextBytes()` e usar a sua saída nos métodos de escrita das classes. Obviamente, para uma comparação adequada é necessário usar os mesmos bytes gerados como entrada para as duas classes.