Zadanie 1:

Klasa Student:

public class Student {  
 private final int nrI;  
 private final String surN;  
 private final String name;  
 private float mark;  
  
 public Student(int nrI, String surN, String name, float mark) {  
 this.nrI = nrI;  
 this.surN = surN;  
 this.name = name;  
 this.mark = mark;  
 }  
  
 public int getNrI() { return nrI; }  
 public String getName() { return name; }  
 public float getMark() { return mark; }  
 public String getSurN() { return surN; }  
  
 public void setMark(float mark) { this.mark = mark;}  
  
  
 @Override  
 public String toString() {  
 return nrI + " " + surN + " " + name + " " + mark;  
 }  
}

IteratorA:

import java.util.Iterator;  
import java.util.NoSuchElementException;  
  
public class IteratorA<T> implements Iterator<T> {  
  
 private final T[] students;  
 private int position;  
  
 public IteratorA(T[] array) {  
 this.students = array;  
 }  
  
 @Override  
 public boolean hasNext() {  
 return position < students.length;  
 }  
  
 @Override  
 public T next() {  
 if (hasNext())  
 return students[position++];  
 else  
 throw new NoSuchElementException();  
 }  
}

IteratorA filtrujący:

import java.util.Iterator;  
import java.util.function.Predicate;  
  
public class FilterIterator<T> implements Iterator<T> {  
  
 Iterator <T> iterator;  
 Predicate <T> predicate;  
  
 public FilterIterator(Iterator <T> iterator, Predicate <T> predicate) {  
 this.iterator = iterator;  
 this.predicate = predicate;  
 }  
  
 @Override  
 public boolean hasNext() {  
 return iterator.hasNext();  
 }  
  
 @Override  
 public T next() {  
 T element;  
 if (hasNext()) {  
 element = iterator.next();  
 if (predicate.test(element))  
 return element;  
 }  
 return null;  
 }  
}

iterator B I filtrujący:

public interface Iterator1B<T> {  
 void first();  
 void next();  
 boolean isDone();  
 T currentItem();  
}

mport java.util.NoSuchElementException;  
  
public class IteratorB<T> implements Iterator1B<T>{  
  
 private final T[] students;  
 private int position;  
  
 public IteratorB(T[] array) {  
 this.students = array;  
 }  
  
 @Override  
 public void first() {  
 position = 0;  
 }  
  
 @Override  
 public void next() {  
 if (!isDone())  
 position++;  
 else  
 throw new NoSuchElementException();  
 }  
  
 @Override  
 public boolean isDone() {  
 return position <= students.length;  
 }  
  
 @Override  
 public T currentItem() {  
 if (!isDone())  
 return students[position];  
 else  
 throw new NoSuchElementException();  
 }  
}

import java.util.function.Predicate;  
  
public class FilterB<T> implements Iterator1B<T>{  
  
 private final Iterator1B <T> iterator;  
 private final Predicate <T> predicate;  
  
 public FilterB(Iterator1B <T> iterator, Predicate <T> predicate) {  
 this.iterator = iterator;  
 this.predicate = predicate;  
 }  
  
 @Override  
 public void first() {  
 iterator.first();  
 }  
  
 @Override  
 public void next() {  
 iterator.next();  
 }  
  
 @Override  
 public boolean isDone() {  
 return iterator.isDone();  
 }  
  
 @Override  
 public T currentItem() {  
 T element;  
 if (!isDone()) {  
 element = iterator.currentItem();  
 if (predicate.test(element))  
 return element;  
 }  
 return null;  
 }  
}

Main:

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.Iterator;  
  
public class Main {  
  
 public static final int *amountOfStudents* = 5;  
 public static final Student[] *students* = new Student[*amountOfStudents*];  
  
 public static void Load(String path) {  
 String line;  
 String[] tokens;  
 int i = 0;  
  
 try(BufferedReader bufferedReader = new BufferedReader(new FileReader(path))) {  
 line = bufferedReader.readLine(); // pominięcie wiersza tytułowego  
 while ( (line = bufferedReader.readLine()) != null ) {  
 tokens = line.split(";");  
 Student st = new Student(Integer.*parseInt*(tokens[0]), tokens[1], tokens[2], Float.*parseFloat*(tokens[3]));  
 *students*[i] = st;  
 i++;  
 }  
 }  
 catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static <T> void Show(Iterator<T> iterator) { // 1  
 while(iterator.hasNext()) {  
 Student st = (Student) iterator.next();  
 if (st != null)  
 System.*out*.println(st.toString());  
 }  
 }  
  
 public static <T> void ChangeMark(Iterator<T> iterator, int nr, float mark) { // 2  
 while(iterator.hasNext()) {  
 Student st = (Student) iterator.next();  
 if (st != null && st.getNrI() == nr)  
 st.setMark(mark);  
 }  
 }  
  
 public static <T> float ShowAVR(Iterator<T> iterator) { // 3  
 float w = 0;  
 int size = 0;  
  
 while(iterator.hasNext()) {  
 Student st = (Student) iterator.next();  
 if (st != null) {  
 w += st.getMark();  
 size++;  
 }  
  
 }  
 return (w/size);  
 }  
  
 public static <T> Student[] Copy(Iterator <T> iterator, int size) {  
 Student[] students1 = new Student[size];  
 int i = 0;  
  
 while(iterator.hasNext()) {  
 Student s1 = (Student) iterator.next();  
 if (s1 != null) {  
 students1[i] = s1;  
 i++;  
 }  
  
 }  
 return students1;  
 }  
  
 public static void main(String[] args) {  
  
 System.*out*.println("============1==============");  
 *Load*("C:\\Users\\remig\\OneDrive - Politechnika Wroclawska\\Semestr2\\LaboUBiele\\Z2\\studenci.txt");  
 *Show*(new IteratorA<>(*students*));  
 System.*out*.println();  
  
 System.*out*.println("============2==============");  
 *ChangeMark*(new IteratorA<>(*students*), 5, 3.0f);  
 *Show*(new IteratorA<>(*students*));  
 System.*out*.println();  
  
 System.*out*.println("============3==============");  
 System.*out*.println(*ShowAVR*(new FilterIterator<>(  
 new IteratorA<>(*students*),  
 student -> student.getMark()>=3  
 )));  
 System.*out*.println();  
  
 System.*out*.println("============4==============");  
 *Show*(new FilterIterator<>(  
 new IteratorA<>(*students*),  
 student -> student.getMark()<3  
 ));  
 System.*out*.println();  
  
 System.*out*.println("============5==============");  
 Student[] goodStudents = *Copy*(new FilterIterator<>(  
 new IteratorA<>(*students*),  
 student -> student.getMark() >= 3  
 ), 4);  
 *Show*(new IteratorA<>(goodStudents));  
 System.*out*.println();  
  
 Student[] badStudents = *Copy*(new FilterIterator<>(  
 new IteratorA<>(*students*),  
 student -> student.getMark() < 3  
 ), 1);  
 *Show*(new IteratorA<>(badStudents));  
 System.*out*.println();  
  
 }  
}