Classes

1. We want to realize a program to handle a document archive. For each document the following information is stored: name of the author, title of the document, and number of available copies.

Part 1: Write a class Document that implements the following methods:

* Document(String a, String t) : costructor that, given author and title, constructs a document for which the number of available copies is 0;
* String toString() : that returns a string with all the information about the document;
* void createCopies(int n) : that increments by n the number of available copies;
* void sellCopies(int n) : that decrements by n the number of available copies; (if n is greater than the number of available copies, then such number should become 0);
* String getAuthor() : that returns the name of the author of the document;
* String getTitle() : that returns the title of the document;
* int getCopies() : that returns the number of available copies of the document.

Document.java

public class Document {

private String author, title;

private int copies;

// constructor without parameters that, given author and title, constructs a

// document for which the number of available copies is 0

public Document(String a, String t) {

author = a; title = t; copies = 0;

}

// returns the information about the document

public String toString( ) {

return "Doc of " + author + " \"" + title + "\" Copies: " + copies;

}

// increments the number of available copies

public void createCopies(int n) {

copies += n;

}

// decrements the number of available copies

public void sellCopies(int n) {

if (n <= copies)

copies -= n;

else

copies = 0;

}

// returns the author of the document

public String getAuthor() {

return author;

}

// returns the title of the document

public String getTitle() {

return title;

}

// returns the number of copies of the document

public int getCopies() {

return copies;

}

}

TestDocument.java

public class TestDocument {

public static void main(String[] args) {

Document d1 = new Document("Mario Rossi", "My first document");

System.out.println(d1);

d1.createCopies(10);

System.out.println(d1);

d1.sellCopies(5);

System.out.println(d1);

System.out.println("Information on the document:");

System.out.println("Author: " + d1.getAuthor());

System.out.println("Title: " + d1.getTitle());

System.out.println("Copies: " + d1.getCopies());

}

}

Part 2:  Write a class UseDocument, client of Document, that contains the following public static methods:

* static String authorMaxCopies(Document d1, Document d2) : that, given two documents d1 and d2, returns the author of the document among d1 and d2 for which more copies are available. If for both documents the same number of copies is available, the method should return the concatenation of the names of the two authors, separated by " and ".
* static int countCopiesAuthor(Document[] docs, String auth) : that, given an archive docs of documents, rappresented simply as an array of documents, and an author auth, returns the total number of copies of all documents with author auth present in docs.

UseDocument.java

public class UseDocument {

// given two documents d1 and d2, returns the author of the document among d1

// and d2 for which more copies are available. If for both documents the

// same number of copies is available, the method should return the

// concatenation of the names of the two authors, separated by a " and "

public static String authorMaxCopies(Document d1, Document d2) {

String res;

if (d1.getCopies() > d2.getCopies())

res = d1.getAuthor();

else if (d1.getCopies() < d2.getCopies())

res = d2.getAuthor();

else

res = d1.getAuthor() + " and " + d2.getAuthor();

return res;

}

// given an archive of documents, rappresented simply as an array of

// documents, and an author a, returns the total number of copies of all

// documents with author a

public static int countCopiesAuthor(Document[] d, String a) {

int c = 0;

for (int i = 0; i < d.length; i++) {

if (d[i].getAuthor().equals(a))

c += d[i].getCopies();

}

return c;

}

}

TestUseDocument.java

public class TestUseDocument {

public static void main(String[] args) {

Document[] d = new Document[5];

d[0] = new Document("Mario Rossi", "My first document");

d[1] = new Document("Mario Rossi", "My second document");

d[2] = new Document("Giovanna Verdi", "My first document");

d[3] = new Document("Marco Neri", "My first document");

d[4] = new Document("Mario Rossi", "My third document");

d[0].createCopies(15);

d[1].createCopies(10);

d[2].createCopies(10);

d[3].createCopies(5);

System.out.println("Author of max sold copies between ");

System.out.println(" " + d[1]);

System.out.println(" and " + d[3]);

System.out.println(" is " + UseDocument.authorMaxCopies(d[1], d[3]));

System.out.println();

System.out.println("Author of max sold copies between ");

System.out.println(" " + d[1]);

System.out.println(" and " + d[2]);

System.out.println(" is " + UseDocument.authorMaxCopies(d[1], d[2]));

System.out.println();

int c1 = UseDocument.countCopiesAuthor(d, "Mario Rossi");

int c2 = UseDocument.countCopiesAuthor(d, "Marco Bruni");

System.out.println("Mario Rossi has " + c1 + " total copies");

System.out.println("Marco Bruni has " + c2 + " total copies");

}

}

2. We want to realize a program to handle an archive of records. For each record the following information is stored: name of the author, title of the record, and price.

Part 1: Write a class Record that implements the following methods:

* Record(String a, String t) : constructor that, given author and title, constructs a record whose price is initially 0.0 (the record is not being sold yet);
* String toString() : that returns a string with all the information about the record;
* void incrementPrice(double p) : that increments the price by the amount p;
* String getAuthor() : that returns the author of the record;
* String getTitle() : that returns the title of the record;
* double getPrice() : that returns the price of the record.

Record.jav

public class Record {

private String author, title;

private double price;

// constructor without parameters that, given author and title, constructs a

// record for which the price is 0

public Record(String a, String t) {

author = a;

title = t;

price = 0.0;

}

// returns the information about the record

public String toString() {

return "Record: " + author + ", " + title + ", " + price;

}

// increments the price by p;

public void incrementPrice(double p) {

price += p;

}

// returns the author of the record

public String getAuthor() {

return author;

}

// returns the title of the record

public String getTitle() {

return title;

}

// returns the price of the record

public double getPrice() {

return price;

}

}

TestRecord.java

public class TestRecord {

public static void main(String[] args) {

Record d1 = new Record("Pink Floyd", "The wall");

System.out.println(d1);

d1.incrementPrice(10);

d1.incrementPrice(d1.getPrice() \* 0.20);

System.out.println(d1);

System.out.println("Information about the record:");

System.out.println("Author: " + d1.getAuthor());

System.out.println("Title: " + d1.getTitle());

System.out.println("Price: " + d1.getPrice());

}

}

Part 2: Write a class UseRecord, client of Record, that contains the following public static methods:

* static String titleLowerPrice(Record r1, Record r2) : that, given two records r1 and r2, returns the title of the record among r1 and r2 with the lowest price. If both records have the same price, the method should return the concatenation of the titles of the two records, separated by a ",".
* static double averagePriceAuthor(Record[] arch, String auth) : that, given an archive arch of records, rappresented simply as an array of records, and an author auth, returns the average price of all records of the author auth that are being sold (i.e., whose price is greater than 0). If there are no records sold by the author, the method should return 0.

UseRecord.java

public class UseRecord {

// given two records r1 and r2, returns the title of the record among r1

// and r2 with the lowest price. If both records have the same price, the

// method should return the concatenation of the titles of the two records,

// separated by a ",".

public static String titleLowerPrice(Record r1, Record r2) {

String res;

if (r1.getPrice() < r2.getPrice())

res = r1.getTitle();

else if (r1.getPrice() > r2.getPrice())

res = r2.getTitle();

else

res = r1.getTitle() + ", " + r2.getTitle();

return res;

}

// given an archive archof records, rappresented simply as an array of

// records, and an author auth, returns the average price of all records of

// the author auth that are being sold (i.e., whose price is greater than 0).

// If there are no records sold by the author, the method should return 0.

public static double averagePriceAuthor(Record[] arch, String aut) {

int s = 0;

int n = 0;

for (int i = 0; i < arch.length; i++) {

if (arch[i].getAuthor().equals(aut) && arch[i].getPrice() > 0) {

s += arch[i].getPrice();

n++;

}

}

double average;

if (n > 0)

average = (double)s / n;

else

average = 0;

return average;

}

}

TestUseRecord.java

public class TestUseRecord {

public static void main(String[] args) {

Record[] r = new Record[6];

r[0] = new Record("Pink Floyd", "The wall");

r[1] = new Record("Pink Floyd", "The dark side of the moon");

r[2] = new Record("Police", "Zenyatta Mondatta");

r[3] = new Record("Mario Rossi", "My first record");

r[4] = new Record("Mario Rossi", "My second record");

r[5] = new Record("Pink Floyd", "Wish you were here");

r[0].incrementPrice(15);

r[1].incrementPrice(10);

r[2].incrementPrice(10);

r[3].incrementPrice(5);

System.out.println("Title of lower price between ");

System.out.println(" " + r[0]);

System.out.println(" and " + r[1]);

System.out.println(" is " + UseRecord.titleLowerPrice(r[0], r[1]));

System.out.println();

System.out.println("Title of lower price between ");

System.out.println(" " + r[1]);

System.out.println(" and " + r[2]);

System.out.println(" is " + UseRecord.titleLowerPrice(r[1], r[2]));

System.out.println();

double m1 = UseRecord.averagePriceAuthor(r, "Pink Floyd");

double m2 = UseRecord.averagePriceAuthor(r, "Carlo Neri");

System.out.println("The average price of Pink Floyd records is " + m1);

System.out.println("The average price of Carlo Neri records is " + m2);

}

}

3. We want to realize a program to handle books rented at a library. For each book the following information is stored: author and title of the book, status of the book (rented or not), name of the person who rented the book.

Part 1: Write a class Book that implements the following methods:

* Book(String a, String t) : constuctor that creates a book with author a and title t that is initially not rented;
* String toString() : that returns a string with all the information about the book;
* void rentTo(String n) : that modifies the information about the book by specifying that the book has been rented by the person whose name is n;
* void returned() : that modifies the information about the book by specifying that it has been returned to the library;
* boolean isRented() : that verifies whether a book is currently being rented;
* String whoHasBook() : that returns the name of the person that currently rents the book, or null if the book is currently not rented;
* String getAuthor() : that returns the author of the book;
* String getTitle() : that returns the title of the book;

Book.java

public class Book {

private String author;

private String title;

private String name;

// constructor that creates a book with author a and title t that is not

// rented

public Book(String a, String t) {

author = a;

title = t;

name = null;

}

// returns a string with all information about the book

public String toString() {

String r = "Book '" + title + "' of " + author;

if (name == null)

r += " is not rented";

else

r += " is rented to " + name;

return r;

}

// modifies the information about the book by specifying that it has been

// rented by the person whose name is n

public void rentTo(String n) {

name = n;

}

// modifies the information about the book by specifying that it has been

// returned to the library

public void returned() {

name = null;

}

// verifies whether a book is currently being rented

public boolean isRented() {

return name != null;

}

// returns the name of the person that currently rents the book,

// or null if the book is currently not rented

public String whoHasBook() {

return name;

}

// returns the author of the book

public String getAuthor() {

return author;

}

// returns the title of the book;

public String getTitle() {

return title;

}

}

TestBook.java

public class TestBook {

public static void main (String[] args) {

Book book = new Book("Carlo Verdi", "My first book");

System.out.println(book);

book.rentTo("Mario Rossi");

System.out.println(book);

book.returned();

System.out.println(book);

System.out.println("Information about the book");

System.out.println("Author: " + book.getAuthor());

System.out.println("Title: " + book.getTitle());

if (book.isRented())

System.out.println("The book is rented to " + book.whoHasBook());

else

System.out.println("The book is not rented");

}

}

Part 2: Write a class UseBook, client of Book, that contains the following public static methods:

* static int countAvailableBooks(Book b1, Book b2, Book b3) : that, given three books b1, b2, and b3, returns the number of books among b1, b2, and b3 that are currently not rented.
* static int countRentedBooks(Book[] b, String a) : that, given an archive arch of books, rappresented simply as an array of books, and the name auth of an author, returns the number of books of the author auth in arch that are currently rented.

UseBook.java

public class UseBook {

// given three books b1, b2, and b3, returns the number of books among b1,

// b2, and b3 that are currently not rented

public static int countAvailableBooks(Book b1, Book b2, Book b3) {

int count = 0;

if (!b1.isRented())

count++;

if (!b2.isRented())

count++;

if (!b3.isRented())

count++;

return count;

}

// given an archive arch of books, rappresented simply as an array of books,

// and the name auth of an author, returns the number of books of the author

// auth in arch that are currently rented

public static int countRentedBooks(Book[] d, String a) {

int count = 0;

for (int i = 0; i < d.length; i++)

if (d[i].getAuthor().equals(a) && d[i].isRented())

count++;

return count;

}

}

TestUseBook.java

public class TestUseBook {

public static void main (String[] args) {

Book[] books = new Book[5];

books[0] = new Book("Carlo Verdi", "My first book");

books[0].rentTo("Mario Rossi");

books[1] = new Book("Carlo Verdi", "My second book");

books[1].rentTo("Maria Rossi");

books[2] = new Book("Carlo Verdone", "My first book");

books[2].rentTo("Mario Rossi");

books[3] = new Book("Carlo Verdone", "My second book");

books[4] = new Book("Carlo Verdi", "My third book");

int c = UseBook.countAvailableBooks(books[3], books[0], books[4]);

System.out.println("Books currently not rented = " + c);

c = UseBook.countRentedBooks(books, "Carlo Verdi");

System.out.println("Books of Carlo Verdi currently being rented = " + c);

}

}

4. We want to realize a program to be used in a car rental office. For each car, the following information is stored: plate number (a string), category (a character), name of person who rented the car (a string), and number of days for which the car is rented (a non-negative integer). If the number of days is 0, this means that the car is not rented.

Part1: Write a class Car that implements the following methods:

* Car(String pn, char c) : costructor that, given plate number and category, constructs a car that is initially not rented;
* String getPlate() : that returns the plate number of the car;
* char getCategory() : that returns the category of the car;
* int getDays() : that returns the number of days for which the car is rented;
* String getPerson() : that returns the name of the person who rented the car, if the car is rented, and null otherwise;
* void rentCar(int d, String n) : that rents the car to person with name n for d days, if it is not already rented, and does nothing otherwise;
* void giveBackCar() : that sets the status of the car to being not rented (this means also that there is no person who rents the car);
* String toString() : that returns a string with all the information about the car.

Car.java

public class Car {

private String plate;

private char category;

private int days;

private String person;

public Car(String pn, char c) {

plate = pn;

category = c;

days = 0;

person = null;

}

public String getPlate() {

return plate;

}

public char getCategory() {

return category;

}

public String getPerson() {

return person;

}

public int getDays() {

return days;

}

public void rentCar(int d, String n) {

if (days == 0) {

days = d;

person = n;

}

}

public void giveBackCar() {

days = 0;

person = null;

}

public String toString() {

return "plate: " + plate +

", category: " + category +

", days: " + days +

", person: " + person;

}

}

TestCar.java

public class TestCar {

public static void main(String[] args) {

Car car = new Car("AB12", 'A');

System.out.println(car);

car.rentCar(5, "Mario Rossi");

System.out.println("After renting the car");

System.out.println(car);

String plate = car.getPlate();

char cat = car.getCategory();

int days = car.getDays();

String person = car.getPerson();

System.out.println("Information on the car:");

System.out.println("Plate: " + plate);

System.out.println("Category: " + cat);

System.out.println("Days: " + days);

System.out.println("Person: " + person);

car.rentCar(10, "Pippo");

System.out.println("Renting again the car should have no effect");

System.out.println(car);

car.giveBackCar();

System.out.println("After returning the car");

System.out.println(car);

}

}

Part 2: Write a class UseCar, client of Car, that contains the following public static method:

* static String longestRental(Car c1, Car c2): that, given two cars c1 and c2, returns the name of the person who rented for the longest period the car among c1 and c2. If both c1 and c2 are rented for the same number of days, the method should return the name of the person who rented c1. If both c1 and c2 are not rented, the method should return null.

UseCar.java

public class UseCar {

public static String longestRental(Car c1, Car c2) {

if (c1.getDays() == 0 && c2.getDays() == 0)

return null;

else if (c1.getDays() >= c2.getDays())

return c1.getPerson();

else

return c2.getPerson();

}

}

TestUseCar.java

public class TestUseCar {

public static void main(String[] args) {

Car c1 = new Car("AB12", 'A');

Car c2 = new Car("CD34", 'B');

System.out.println(c1);

System.out.println(c2);

System.out.println("Longest rental: " + UseCar.longestRental(c1,c2));

System.out.println();

c1.rentCar(5, "Mario Rossi");

System.out.println(c1);

System.out.println(c2);

System.out.println("Longest rental: " + UseCar.longestRental(c1,c2));

System.out.println();

c2.rentCar(5, "Carlo Bruni");

System.out.println(c1);

System.out.println(c2);

System.out.println("Longest rental: " + UseCar.longestRental(c1,c2));

System.out.println();

c2.giveBackCar();

c2.rentCar(10, "Carlo Bruni");

System.out.println(c1);

System.out.println(c2);

System.out.println("Longest rental: " + UseCar.longestRental(c1,c2));

}

}

5. We want to realize a program to be used in a hotel to manage booking of hotel rooms. For each room, the following information is stored: room number (a positive integer), room category (a string), name of guest who occupies the room (a string), and number of days for which the guest occupies the room (a non-negative integer). If the name of the guest is missing, this means that the room is not occupied.

**Part 1.** Write a class Room that implements the following methods:

* Room(int num, String cat) : constructor that, given room number and category, constructs a room that is initially not occupied;
* int getNumber() : that returns the room number;
* String getCategory() : that returns the category of the room;
* String getGuest() : that returns the name of the guest who occupies the room, if the room is occupied, and null otherwise;
* int getDays() : that returns the number of days for which the room will be occupied;
* void occupyRoom(String g, int d) : that occupies the room with the guest with name g for d days (with d a positive integer), if the room is not already occupied, and does nothing otherwise;
* void freeRoom() : that sets the status of the room to being not occupied (i.e., there is no guest who occupies the room); the method should also set to 0 the number of days for which the room will be occupied.
* String toString() : that returns a string with all the information about the room.

Room.java

public class Room {

private int number;

private String category;

private String guest;

private int days;

public Room(int num, String cat) {

number = num;

category = cat;

guest = null;

days = 0;

}

public int getNumber() {

return number;

}

public String getCategory() {

return category;

}

public String getGuest() {

return guest;

}

public int getDays() {

return days;

}

public void occupyRoom(String g, int d) {

if (guest == null) {

guest = g;

days = d;

}

}

public void freeRoom() {

guest = null;

days = 0;

}

public String toString() {

return "number: " + number +

", category: " + category +

", guest: " + guest +

", days: " + days;

}

}

TestRoom.java

public class TestRoom {

public static void main(String[] args) {

Room room = new Room(120, "Suite");

System.out.println(room);

room.occupyRoom("Mario Rossi", 4);

System.out.println("After occupying the room");

System.out.println(room);

int number = room.getNumber();

String cat = room.getCategory();

String guest = room.getGuest();

int days = room.getDays();

System.out.println("Information on the room:");

System.out.println("Number: " + number);

System.out.println("Category: " + cat);

System.out.println("Guest: " + guest);

System.out.println("Days: " + days);

room.occupyRoom("Pippo", 10);

System.out.println("Occupying again the room should have no effect");

System.out.println(room);

room.freeRoom();

System.out.println("After freeing the room");

System.out.println(room);

}

}

**Part 2.** Write a class UseRoom, client of Room, that contains the following public static method:

* static int shortestOccupation(Room r1, Room r2): that, given two rooms r1 and r2, returns the number of the room among r1 and r2 that will be occupied for the least number of days . If both r1 and r2 will be occupied for the same number of days, the method should return the number of room r1. If both r1 and r2 are not occupied, the method should return -1.

UseRoom.java

public class UseRoom {

public static int shortestOccupation(Room r1, Room r2) {

if (r1.getGuest() == null && r2.getGuest() == null)

return -1;

else if (r1.getDays() >= r2.getDays())

return r1.getNumber();

else

return r2.getNumber();

}

}

TestUseRoom.java

public class TestUseRoom {

public static void main(String[] args) {

Room r1 = new Room(120, "Suite");

Room r2 = new Room(230, "Double");

System.out.println(r1);

System.out.println(r2);

System.out.println("Shortest occupation: room number " +

UseRoom.shortestOccupation(r1,r2));

System.out.println();

r1.occupyRoom("Mario Rossi", 5);

System.out.println(r1);

System.out.println(r2);

System.out.println("Shortest occupation: room number " +

UseRoom.shortestOccupation(r1,r2));

System.out.println();

r2.occupyRoom("Carlo Bruni", 5);

System.out.println(r1);

System.out.println(r2);

System.out.println("Shortest occupation: room number " +

UseRoom.shortestOccupation(r1,r2));

System.out.println();

r2.freeRoom();

r2.occupyRoom("Carlo Bruni", 10);

System.out.println(r1);

System.out.println(r2);

System.out.println("Shortest occupation: room number " +

UseRoom.shortestOccupation(r1,r2));

}

}

6. We want to realize a program to be used in a book store to manage the book archive. For each book, the following information is stored: title (a string), author (a string), number of copies that are available (if the number of copies is 0, this means that the book is not available), and price (a double). If the book is not available, the price information is not significant (e.g., you may assume that it is 0).

**Part 1.** Write a class Book that implements the following methods:

* Book(String tit, String aut) : constructor that, given title, and author, constructs a book that is initially not available;
* String getTitle() : that returns the title of the book;
* String getAuthor() : that returns the author of the book;
* int getCopies() : that returns the number of copies of the book that are available;
* double getPrice() : that returns the price of the book, if the book is available, and 0 otherwise;
* void buyCopies(int n, double p) : that increments the number of available copies by n (a positive integer), and sets the price of the book to p (the previous price, even if significant, is ignored);
* void sellCopy() : that decrements the number of copies by 1, if the book is available, and does nothing otherwise;
* String toString() : that returns a string with all the information about the book.

Book.java

public class Book {

private String title;

private String author;

private int copies;

private double price;

public Book(String tit, String aut) {

title = tit;

author = aut;

copies = 0;

price = 0.0;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public int getCopies() {

return copies;

}

public double getPrice() {

if (copies > 0)

return price;

else

return 0.0;

}

public void buyCopies(int n, double p) {

copies += n;

price = p;

}

public void sellCopy() {

if (copies > 0)

copies--;

if (copies == 0)

price = 0.0;

}

public String toString() {

return "title: " + title +

", author: " + author +

", copies: " + copies +

", price: " + price;

}

}

TestBook.java

public class TestBook {

public static void main(String[] args) {

Book book = new Book("Das Glasperlenspiel", "Hesse");

System.out.println(book);

book.buyCopies(2, 20.5);

System.out.println("After buying copies of the book");

System.out.println(book);

String title = book.getTitle();

String author = book.getAuthor();

int copies = book.getCopies();

double price = book.getPrice();

System.out.println("Information on the book:");

System.out.println("Title: " + title);

System.out.println("Author: " + author);

System.out.println("Copies: " + copies);

System.out.println("Price: " + price);

book.buyCopies(1, 30.7);

System.out.println("Buying additional copies should change the price");

System.out.println(book);

book.sellCopy();

book.sellCopy();

book.sellCopy();

System.out.println("After selling all copies");

System.out.println(book);

System.out.println("Price: " + book.getPrice() + " (should be 0.0)");

book.sellCopy();

System.out.println("Selling another copy should have no effect");

System.out.println(book);

}

}

**Part 2.** Write a class UseBook, client of Book, that contains the following public static method:

* static String expensiveBook(Book b1, Book b2): that, given two books b1 and b2, returns the title of the more expensive book among b1 and b2. If b1 and b2 have the same price, the method should return the title of b1. If one of b1 or b2 is not available, the method should return the title of the available book. If neither b1 nor b2 are available, the method should return null.

UseBook.java

public class UseBook {

public static String expensiveBook(Book b1, Book b2) {

if (b1.getCopies() == 0 && b2.getCopies() == 0)

return null;

else if (b1.getPrice() >= b2.getPrice())

return b1.getTitle();

else

return b2.getTitle();

}

}

TestUseBook.java

public class TestUseBook {

public static void main(String[] args) {

Book b1 = new Book("Das Glasperlenspiel", "Hesse");

Book b2 = new Book("Stranger in a strange land", "Heinlein");

System.out.println(b1);

System.out.println(b2);

System.out.println("Most expensive book is: " +

UseBook.expensiveBook(b1, b2));

System.out.println();

b1.buyCopies(1, 20.5);

System.out.println(b1);

System.out.println(b2);

System.out.println("Most expensive book is: " +

UseBook.expensiveBook(b1, b2));

System.out.println();

b2.buyCopies(2, 20.5);

System.out.println(b1);

System.out.println(b2);

System.out.println("Most expensive book is: " +

UseBook.expensiveBook(b1, b2));

System.out.println();

b2.buyCopies(3, 30.7);

System.out.println(b1);

System.out.println(b2);

System.out.println("Most expensive book is: " +

UseBook.expensiveBook(b1, b2));

}

}

7. We want to realize a program for collecting statistical data about apartments. For each apartment, the following information is stored:

* address (a string),
* size (in square meters, a double),
* status (free or occupied),
* name of the person who lives in the aparment, in case the apartment is occupied (a string).

**Part 1.** Write a class Apartament that implements the following methods:

* Apartament(String a, double s) : constructor that creates an object representing an appartment whose address is a, whose size is s, and which is initially free;
* String getAddress() : that returns the address of the apartment;
* double getSize() : that returns the size of the apartment;
* String getPerson() : that returns the name of the person living in the apartment, if the apartment is occupied, and null otherwise;
* boolean isFree() : that checks whether the aparment is free;
* void occupy(String p) : that changes the status of the apartment to occupied and sets the name of the person living in the apartment to p; if the apartment is already occupied, the method should do nothing;
* void free() : that changes the status of the apartment to free; note that when the apartment is free, the name of the person living in it is not significant;
* String toString() : that returns a string with all the information about the apartment.

Apartment.java

public class Apartment {

private String address;

private double size;

private String person; // we represent a free apartment with person equal

// to null, and an occupied one with person

// different from null

public Apartment(String a, double s) {

address = a;

size = s;

person = null;

}

public String getAddress() {

return address;

}

public double getSize() {

return size;

}

public String getPerson() {

return person;

}

public boolean isFree() {

return person == null;

}

public void occupy(String p) {

if (person == null)

person = p;

}

public void free() {

person = null;

}

public String toString() {

return "address: " + address +

", size: " + size +

", occupied: " + (!isFree()) +

(!isFree()? " by " + person : "");

}

}

TestApartment.java

**Part 2.** Write a class UseApartment, client of Apartment, that contains the following public static method:

* static double averageSizeFree(Apartment[] arc) : that, given an archive arc of apartaments, represented as an array of apartments, calculates the average size of the apartments that are free. If there are no free apartments in the archive, the method should return 0.

UseApartment.java

public class UseApartment {

public static double averageSizeFree(Apartment[] arc) {

int count = 0;

double sum = 0;

for (int i = 0; i < arc.length; i++)

if (arc[i].isFree()) {

count++;

sum = sum + arc[i].getSize();

}

if (count == 0)

return 0;

else

return sum/count;

}

}

TestUseApartment.java

public class TestUseApartment {

public static void main(String[] args) {

Apartment[] aps = new Apartment[5];

aps[0] = new Apartment("Via Firenze", 90);

aps[1] = new Apartment("Via Milano", 100);

aps[1].occupy("Carlo Verdi");

aps[2] = new Apartment("Via Roma", 110);

aps[3] = new Apartment("Via Pisa", 80);

aps[3].occupy("Antonio Neri");

aps[4] = new Apartment("Via Torino", 40);

System.out.println("The next value should be 80.0.");

double s = UseApartment.averageSizeFree(aps);

System.out.println("Average size of free apartments = " + s);

Apartment[] aps2 = new Apartment[3];

aps2[0] = new Apartment("Via Firenze", 90);

aps2[0].occupy("Mario Rossi");

aps2[1] = new Apartment("Via Milano", 100);

aps2[1].occupy("Carlo Verdi");

aps2[2] = new Apartment("Via Pisa", 80);

aps2[2].occupy("Antonio Neri");

System.out.println("The next value should be 0.0");

s = UseApartment.averageSizeFree(aps2);

System.out.println("Average size of free apartments = " + s);

}

}

8. We want to realize a program for collecting statistical data about car sales. For each car, the following information is stored:

* brand (a string),
* horse power (an integer),
* price (a double),
* name of the owner of the car (a string); if the car has not been sold, the name of the owner is null.

**Part 1.** Write a class Car that implements the following methods:

* Car(String b, int hp) : constructor that creates an object representing an car whose brand is b, whose horse power is hp, and which has not been sold yet;
* String getBrand() : that returns the brand of the car;
* int getHP() : that returns the horse power of the car;
* double getPrice() : that returns the price of the car, if the car has been sold, and 0 otherwise;
* String getOwner() : that returns the name of the owner of the car, if the car has been sold, and null otherwise;
* boolean isSold() : that checks whether the car has been sold;
* void sell(String n, double p) : that changes the status of the car to sold, by setting the name of the owner of the car to n and the price to p; if the car had already been sold, the method should do nothing;
* String toString() : that returns a string with all the information about the car.

TestCar.java

public class TestCar {

public static void main(String[] args) {

Car car = new Car("Ferrari", 400);

System.out.println(car);

car.sell("Mario Rossi", 1.0E6);

System.out.println("After selling the car");

System.out.println(car);

String brand = car.getBrand();

int power = car.getHP();

double price = car.getPrice();

String owner = car.getOwner();

boolean sold = car.isSold();

System.out.println("Information on the car:");

System.out.println("Brand: " + brand);

System.out.println("Power: " + power);

System.out.println("Price: " + price);

System.out.println("Owner: " + owner);

System.out.println("Sold: " + sold);

car.sell("Carlo Bruni", 100);

System.out.println("Selling again the car should have no effect");

System.out.println(car);

}

}

**Part 2.** Write a class UseCar, client of Car, that contains the following public static method:

* static double averagePriceSold(Car[] ar) : that, given an archive ar of cars, represented as an array of cars, calculates the average price of the cars that have been sold. If there are no cars that have been sold in the archive, the method should return 0.

TestUseCar.java

public class TestUseCar {

public static void main(String[] args) {

Car[] cars = new Car[5];

cars[0] = new Car("Ferrari", 400);

cars[1] = new Car("Lancia", 150);

cars[1].sell("Carlo Verdi", 4.0E4);

cars[2] = new Car("BMW", 120);

cars[3] = new Car("Alfa", 100);

cars[3].sell("Antonio Neri", 3.0E4);

cars[4] = new Car("Fiat", 110);

System.out.println("The next value should be 35000.0.");

double p = UseCar.averagePriceSold(cars);

System.out.println("Average price of sold cars = " + p);

Car[] cars2 = new Car[4];

cars2[0] = new Car("Ferrari", 400);

cars2[1] = new Car("Lancia", 150);

cars2[2] = new Car("BMW", 120);

cars2[3] = new Car("Alfa", 100);

System.out.println();

System.out.println("The next value should be 0.0");

p = UseCar.averagePriceSold(cars2);

System.out.println("Average price of sold cars = " + p);

}

}

9. We want to realize a program to be used by a DVD rental shop to manage DVDs to be rented. For each DVD, the following information is stored:

* title (a string),
* age for which the DVD is suited (an integer),
* number of days that the DVD is rented (a non-negative integer), and
* name of the person that rented the DVD (a string).

If the number of days that the DVD is rented is 0, this means that the DVD is not rented.

**Part 1.** Write a class DVD that implements the following methods:

* DVD(String title, int age) : costructor that, given title and age, constructs a DVD that is initially not rented;
* String getTitle() : that returns the title of the DVD;
* int getAge() : that returns the age for which the DVD is suited;
* int getDays() : that returns the number of days that the video is rented;
* String getName() : that returns the name of the person that rented the DVD, if the DVD is rented, and null otherwise;
* void rentDVD(String n, int d) : that rents the DVD to the person with name n for d days, if it is not already rented, and otherwise does nothing;
* void giveBackDVD() : that sets the status of the DVD to being not rented (this means also that there is no person who rents the DVD);
* String toString() : that returns a string with all the information about the DVD.

DVD.java

public class DVD {

private String title;

private int age;

private int days;

private String person;

public DVD(String t, int a) {

title = t;

age = a;

days = 0;

person = null;

}

public String getTitle() {

return title;

}

public int getAge() {

return age;

}

public int getDays() {

return days;

}

public String getName() {

return person;

}

public void rentDVD(String n, int d) {

if (days == 0) {

days = d;

person = n;

}

}

public void giveBackDVD() {

days = 0;

person = null;

}

public String toString() {

return "title: " + title +

", age: " + age +

", days: " + days +

", person: " + person;

}

}

TestDVD.java

public class TestDVD {

public static void main(String[] args) {

DVD dvd = new DVD("Harry Potter", 8);

System.out.println(dvd);

dvd.rentDVD("Luca Bianchi", 5);

System.out.println("After renting the DVD");

System.out.println(dvd);

String title = dvd.getTitle();

int age = dvd.getAge();

int days = dvd.getDays();

String name = dvd.getName();

System.out.println("Information on the dvd:");

System.out.println("Title: " + title);

System.out.println("Age: " + age);

System.out.println("Days: " + days);

System.out.println("Name: " + name);

dvd.rentDVD("Pippo", 10);

System.out.println("Renting again the DVD should have no effect");

System.out.println(dvd);

dvd.giveBackDVD();

System.out.println("After returning the DVD");

System.out.println(dvd);

}

}

**Part 2.**

Write a class UseDVD, client of DVD, that contains the following public static method:

* static String shortestRental(DVD d1, DVD d2): that, given two DVDs d1 and d2, returns the name of the person who rented for the shortest period the DVDs among d1 and d2. If both d1 and d2 are rented for the same number of days, the method should return the name of the person who rented d1. If only one of d1 or d2 is rented, the method should return the name of the person who rented that DVD. If both d1 and d2 are not rented, the method should return null.

UseDVD.java

public class UseDVD {

public static String shortestRental(DVD d1, DVD d2) {

if (d1.getDays() == 0 && d2.getDays() == 0)

return null;

else if (d1.getDays() == 0)

return d2.getName();

else if (d2.getDays() == 0)

return d1.getName();

else if (d1.getDays() <= d2.getDays())

return d1.getName();

else

return d2.getName();

}

}

TestUseDVD.java

public class TestUseDVD {

public static void main(String[] args) {

DVD d1 = new DVD("Harry Potter", 8);

DVD d2 = new DVD("Incredible 4", 10);

System.out.println(d1);

System.out.println(d2);

System.out.println("Shortest rental: " + UseDVD.shortestRental(d1,d2));

System.out.println();

d1.rentDVD("Mario Rossi", 5);

System.out.println(d1);

System.out.println(d2);

System.out.println("Shortest rental: " + UseDVD.shortestRental(d1,d2));

System.out.println();

d1.giveBackDVD();

d2.rentDVD("Carlo Bruni", 10);

System.out.println(d1);

System.out.println(d2);

System.out.println("Shortest rental: " + UseDVD.shortestRental(d1,d2));

System.out.println();

d1.rentDVD("Anna Verdi", 10);

System.out.println(d1);

System.out.println(d2);

System.out.println("Shortest rental: " + UseDVD.shortestRental(d1,d2));

System.out.println();

d2.giveBackDVD();

d2.rentDVD("Marco Neri", 5);

System.out.println(d1);

System.out.println(d2);

System.out.println("Shortest rental: " + UseDVD.shortestRental(d1,d2));

}

}

10. We want to realize a program for managing the accounts of clients of a bank. For each account, the following information is stored:

* name of owner (a string),
* penalty (a double),
* balance (a double),
* number of operations performed on the account (an integer).

**Part 1.** Write a class Account that implements the following methods:

* Account(String o, double p) : constructor that creates an object representing a bank account whose owner is o, whose penalty is p, whose initial balance is 0, and whose initial number of performed operations is 0.
* String getOwner() : that returns the name of the owner of the bank account;
* double getPenalty() : that returns the penalty of the bank account;
* double getBalance() : that returns the balance of the bank account;
* int getNumOp() : that returns the number of operations performed on the bank account;
* void deposit(double a) : that deposits the amount of money a on the bank account, and increments the number of operations by 1;
* void withdraw(double a) : that withdraws the amount of money a from the bank account, and increments the number of operations by 1; if after the withdrawal the balance is negative (either because it becomes negative, or because it already was negative), an additional amount equal to the penalty is subtracted from the balance;
* String toString() : that returns a string with all the information about the bank account.

Account.java

public class Account {

private String owner;

private double penalty;

private double balance;

private int numOp;

public Account(String o, double p) {

owner = o;

penalty = p;

balance = 0;

numOp = 0;

}

public String getOwner() {

return owner;

}

public double getPenalty() {

return penalty;

}

public double getBalance() {

return balance;

}

public int getNumOp() {

return numOp;

}

public void deposit(double a) {

balance = balance + a;

numOp++;

}

public void withdraw(double a) {

numOp++;

balance = balance - a;

if (balance < 0)

balance = balance - penalty;

}

public String toString() {

return "owner: " + owner +

", penalty: " + penalty +

", balance: " + balance +

", number of operations: " + numOp;

}

}

TestAccount.java

public class TestAccount {

public static void main(String[] args) {

Account acc = new Account("Mario Rossi", 10);

System.out.println(acc);

acc.deposit(100);

System.out.println("After depositing 100");

System.out.println(acc);

String owner = acc.getOwner();

double penalty = acc.getPenalty();

double balance = acc.getBalance();

int numOp = acc.getNumOp();

System.out.println("Information on the account:");

System.out.println("Owner: " + owner);

System.out.println("Penalty: " + penalty);

System.out.println("Balance: " + balance);

System.out.println("Num op: " + numOp);

acc.withdraw(50);

System.out.println("After withdrawing 50");

System.out.println(acc);

acc.withdraw(50);

System.out.println("After withdrawing other 50, balance should be 0");

System.out.println(acc);

acc.withdraw(20);

System.out.println(

"After withdrawing other 20, penalty of 10 should be applied");

System.out.println(acc);

}

}

**Part 2.** Write a class UseAccount, client of Account, that contains the following public static method:

* static double getMinPositiveBalance(Account a1, Account a2) : that, given two accounts a1 and a2, returns the minimum positive balance of a1 and a2. More precisely,
  + if both accounts have a positive balance, the minimum of the two balances is returned;
  + if only one of the two accounts has a positive balance, that balance is returned;
  + if both accounts have a balance that is negative or 0, then 0 is returned.

UseAccount.java

public class UseAccount {

public static double getMinPositiveBalance(Account a1, Account a2) {

double b1 = a1.getBalance();

double b2 = a2.getBalance();

if (b1 < 0 && b2 <0)

return 0;

else if (b1 < 0)

return b2;

else if (b2 < 0)

return b1;

else if (b1 < b2)

return b1;

else

return b2;

}

}

TestUseAccount.java

public class TestUseAccount {

public static void main(String[] args) {

Account a1 = new Account("Mario Rossi", 10);

Account a2 = new Account("Carlo Bruni", 20);

a1.deposit(20);

a2.deposit(10);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseAccount.getMinPositiveBalance(a1, a2));

System.out.println();

a1.withdraw(30);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseAccount.getMinPositiveBalance(a1, a2));

System.out.println();

a1.deposit(40);

a2.withdraw(30);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseAccount.getMinPositiveBalance(a1, a2));

System.out.println();

a1.withdraw(40);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseAccount.getMinPositiveBalance(a1, a2));

}

}

11. We want to realize a program for managing the fund-raising activities of volunteers working in a no-profit organization. For each volunteer, the following information is stored:

* name (a string),
* bonus (in percent) granted by the Province (a double),
* amount of money that the volunteer has collected (a double),

**Part 1.** Write a class Volunteer that implements the following methods:

* Volunteer(String name) : constructor that creates an object representing a volunteer whose name is name, whose initial bonus is 0, and whose initial amount of money is 0.
* String getName() : that returns the name of the volunteer;
* double getBonus() : that returns the bonus of the volunteer.
* double getAmount() : that returns the amount of money collected by the volunteer;
* void collect(double m) : that increases the amount of money collected by the volunteer by m; if m is greater than or equal to 1000, then the bonus is increased by 1 (percent).
* void spend(double m) : that decreases the amount of money collected by the volunteer by m, if m is smaller or equal than the amount of money collected, and otherwise does nothing.
* void useBonus() : that increases the amount of money collected by the volunteer by the percentage specified by the bonus, and sets then the bonus to 0.
* String toString() : that returns a string with all the information about the volunteer.

Volunteer.java

public class Volunteer {

private String name;

private double bonus;

private double amount;

public Volunteer(String n) {

name = n;

bonus = 0;

amount = 0;

}

public String getName() {

return name;

}

public double getBonus() {

return bonus;

}

public double getAmount() {

return amount;

}

public void collect(double m) {

amount = amount + m;

if (amount >= 1000)

bonus++;

}

public void spend(double m) {

if (amount >= m)

amount = amount - m;

}

public void useBonus() {

amount = amount \* (1 + bonus/100);

bonus = 0;

}

public String toString() {

return "name: " + name +

", bonus: " + bonus +

", amount: " + amount;

}

}

TestVolunteer.java

public class TestVolunteer {

public static void main(String[] args) {

Volunteer vol = new Volunteer("Mario Rossi");

System.out.println(vol);

vol.collect(100);

System.out.println("After collecting 100");

System.out.println(vol);

String name = vol.getName();

double bonus = vol.getBonus();

double amount = vol.getAmount();

System.out.println("Information on the volunteer:");

System.out.println("Name: " + name);

System.out.println("Bonus: " + bonus);

System.out.println("Amount: " + amount);

vol.spend(50);

System.out.println("After spending 50");

System.out.println(vol);

vol.spend(60);

System.out.println("After spending other 60, nothing should change");

System.out.println(vol);

vol.collect(1000);

System.out.println(

"After collecting other 1000, the bonus should have increased");

System.out.println(vol);

vol.collect(1950);

System.out.println(

"After collecting other 1950, the bonus should have increased");

System.out.println(vol);

vol.useBonus();

System.out.println(

"After using the bonus");

System.out.println(vol);

}

}

**Part 2.** Write a class UseVolunteer, client of Volunteer, that contains the following public static method:

* static double totalAmount(Volunteer v1, Volunteer v2) : that, given two volunteers v1 and v2, calculates and returns the sum of the amount of money raised by the two volunteers, considering also the bonuses.
  + If both volunteers have a positive bonus, the maximum bonus among those of v1 and v2 is applied to the sum of the money raised by v1 and v2.
  + If only one of the two volunteers has a positive bonus (and the other one has bonus 0), then the positive bonus is applied only to the money of that volunteer.
  + If both volunteers have a bonus of 0, then no bonus is applied.

UseVolunteer.java

public class UseVolunteer {

public static double totalAmount(Volunteer v1, Volunteer v2) {

double b1 = v1.getBonus();

double b2 = v2.getBonus();

double m1 = v1.getAmount();

double m2 = v2.getAmount();

if (b1 > 0 && b2 > 0) {

double mb = b1 > b2? b1 : b2;

return (m1 + m2) \* (1 + mb/100);

} else if (b1 > 0)

return m1 \* (1 + b1/100) + m2;

else

return m1 + m2 \* (1 + b2/100);

}

}

TestUseVolunteer.java

public class TestUseVolunteer {

public static void main(String[] args) {

Volunteer v1 = new Volunteer("Mario Rossi");

Volunteer v2 = new Volunteer("Carlo Bruni");

v1.collect(300);

v2.collect(100);

System.out.println(v1);

System.out.println(v2);

System.out.println("No bonus is applied");

System.out.println(UseVolunteer.totalAmount(v1, v2));

System.out.println();

v1.collect(1700);

System.out.println(v1);

System.out.println(v2);

System.out.println("Bonus is applied only to first volunteer");

System.out.println(UseVolunteer.totalAmount(v1, v2));

System.out.println();

v1.collect(1000);

System.out.println(v1);

System.out.println(v2);

System.out.println("Bonus is applied only to first volunteer");

System.out.println(UseVolunteer.totalAmount(v1, v2));

System.out.println();

v2.collect(1900);

System.out.println(v1);

System.out.println(v2);

System.out.println("Maximum bonus is applied to both volunteers");

System.out.println(UseVolunteer.totalAmount(v1, v2));

}

}

12. We want to realize a program for collecting data about medical visits. For each visit, the following information is stored:

* name of a person (a string),
* price of the visit (a double),
* result of the visit (a string); if the visit has not been made, the result is null.
* duration in minutes (an integer); if the visit has not been made, the duration is -1.

**Part 1.** Write a class Visit that implements the following methods:

* Visit(String n, double p) : constructor that creates an object representing a visit whose name of person is n, whose price is p, and which has not been made yet;
* String getName() : that returns the name of the person;
* double getPrice() : that returns the price of the visit;
* String getResult() : that returns the result of the visit, if it has taken place, and null otherwise;
* int getDuration() : that returns the duration of the visit, if it has taken place, and -1 otherwise;
* boolean hasTakenPlace() : that checks whether the visit has already taken place;
* void visit(String res, int dur) : that changes the status of the visit so that it has taken place, by setting the result of the visit to res and the duration to dur; if the visit had already taken place, the method should do nothing;
* String toString() : that returns a string with all the information about the visit.

Visit.java

public class Visit {

private String name;

private double price;

private String result; // we represent a visit that has not taken place with

// result equal to null, and one that has taken place

// with result different from null

private int duration;

public Visit(String n, double p) {

name = n;

price = p;

result = null;

duration = -1;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public String getResult() {

return result;

}

public int getDuration() {

return duration;

}

public boolean hasTakenPlace() {

return result != null;

}

public void visit(String res, int dur) {

if (result == null) {

result = res;

duration = dur;

}

}

public String toString() {

return "name: " + name +

", price: " + price +

", taken place: " + (hasTakenPlace()? "with duration " + duration + " - Result: " + result : "false");

}

}

TestVisit.java

public class TestVisit {

public static void main(String[] args) {

Visit vis = new Visit("John", 40.0);

System.out.println(vis);

vis.visit("All OK", 35);

System.out.println("After the visit has taken place");

System.out.println(vis);

String name = vis.getName();

double price = vis.getPrice();

String result = vis.getResult();

int duration = vis.getDuration();

boolean done = vis.hasTakenPlace();

System.out.println("Information on the visit:");

System.out.println("Name: " + name);

System.out.println("Price: " + price);

System.out.println("Result: " + result);

System.out.println("Duration: " + duration);

System.out.println("Done: " + done);

vis.visit("Carlo Bruni", 50);

System.out.println("Doing again the visit should have no effect");

System.out.println(vis);

}

}

**Part 2.** Write a class UseVisit, client of Visit, that contains the following public static method:

* static double averageDuration(Visit[] ar) : that, given an archive ar of visits, represented as an array of visits, calculates the average duration of the visits that have taken place. If there are no visits that have taken place in the archive, the method should return 0.

UseVist.java

public class UseVisit {

public static double averageDuration(Visit[] ar) {

int count = 0;

double sum = 0;

for (int i = 0; i < ar.length; i++)

if (ar[i].hasTakenPlace()) {

count++;

sum = sum + ar[i].getDuration();

}

if (count == 0)

return 0;

else

return sum/(double)count;

}

}

TestUseVisit.java

public class TestUseVisit {

public static void main(String[] args) {

Visit[] visits = new Visit[5];

visits[0] = new Visit("John", 40.0);

visits[1] = new Visit("Mary", 15.0);

visits[1].visit("All OK", 40);

visits[2] = new Visit("Peter", 12.0);

visits[3] = new Visit("Ann", 10.0);

visits[3].visit("Some problems", 30);

visits[4] = new Visit("Susan", 11.0);

System.out.println("The next value should be 35.0.");

double p = UseVisit.averageDuration(visits);

System.out.println("Average duration of visits that have taken place = "

+ p);

Visit[] visits2 = new Visit[4];

visits2[0] = new Visit("John", 40.0);

visits2[1] = new Visit("Mary", 15.0);

visits2[2] = new Visit("Peter", 12.0);

visits2[3] = new Visit("Ann", 10.0);

System.out.println();

System.out.println("The next value should be 0.0");

p = UseVisit.averageDuration(visits2);

System.out.println("Average duration of visits that have taken place = "

+ p);

}

}

13. We want to realize a program for managing credit card accounts. For each account, the following information is stored:

* name of owner (a string),
* credit card number (a string),
* maximum allowed expenses (a double),
* current expenses in Dollars (a double).

**Part 1.** Write a class CCAccount that implements the following methods:

* CCAccount(String owner, String ccn, double me) : constructor that creates an object representing a credit card account whose owner is owner, whose credit card number is ccn, whose maximum allowed expenses is me, and whose initial expenses are 0.
* String getOwner() : that returns the name of the owner of the account;
* String getNumber() : that returns the credit card number for the account;
* double getMaxExpenses() : that returns the maximum allowed expenses for the credit card account;
* double getExpenses() : that returns the current expenses for the credit card account;
* double spend(double a) : if the amount of Euro a does not exceed the remaining allowed expenses (which are calculated as the difference between the maximum allowed expenses and the current expenses), then a is added to the current expenses, and the new remaining allowed expenses are returned. Otherwise the credit card account is not changed, and -1 is returned;
* double payBack() : that sets the current expenses to 0, and returns the current expenses (before they were set to 0);
* String toString() : that returns a string with all the information about the credit card account.

CCAccount.java

public class CCAccount {

private String owner;

private String number;

private double maxExpenses;

private double expenses;

public CCAccount(String o, String ccn, double me) {

owner = o;

number = ccn;

maxExpenses = me;

expenses = 0;

}

public String getOwner() {

return owner;

}

public String getNumber() {

return number;

}

public double getMaxExpenses() {

return maxExpenses;

}

public double getExpenses() {

return expenses;

}

public double spend(double a) {

if (a <= maxExpenses - expenses) {

expenses = expenses + a;

return maxExpenses - expenses;

} else

return -1;

}

public double payBack() {

double aux = expenses;

expenses = 0;

return aux;

}

public String toString() {

return "owner: " + owner +

", CC number: " + number +

", max expenses: " + maxExpenses +

", current expenses: " + expenses;

}

}

TestCCAccount.java

public class TestCCAccount {

public static void main(String[] args) {

CCAccount acc = new CCAccount("Mario Rossi", "ABCD0001", 1000);

System.out.println(acc);

double remainder;

remainder = acc.spend(100);

System.out.println("After spending 100, the remaining expenses are " +

remainder);

System.out.println(acc);

String owner = acc.getOwner();

String number = acc.getNumber();

double maxExpenses = acc.getMaxExpenses();

double expenses = acc.getExpenses();

System.out.println("Information on the account:");

System.out.println("Owner: " + owner);

System.out.println("CC number: " + number);

System.out.println("Max expenses: " + maxExpenses);

System.out.println("Expenses: " + expenses);

remainder = acc.spend(500);

System.out.println("After spending 500 more, the remaining expenses are " +

remainder);

System.out.println(acc);

remainder = acc.spend(500);

System.out.println("Spending 500 more should not be possible");

System.out.println("Remainder: " + remainder + " (should be -1.0)");

System.out.println("The account should not change:");

System.out.println(acc);

double amount = acc.payBack();

System.out.println("Paid back " + amount);

System.out.println(acc);

}

}

**Part 2.** Write a class UseCCAccount, client of CCAccount, that contains the following public static method:

* static double getMinRemainingExpenses(CCAccount a1, CCAccount a2) : that, given two credit card accounts a1 and a2, returns the minimum among the remaining expenses of a1 and a2 that is different from 0. More precisely,
  + if for both accounts the remaining expenses are positive, the minimum of the two remaining expenses is returned;
  + if for only one of the two accounts the remaining expenses are positive, that amount is returned;
  + if for both accounts the remaining expenses are 0, then 0 is returned.

UseCCAccount.java

public class UseCCAccount {

public static double getMinRemainingExpenses(CCAccount a1, CCAccount a2) {

double r1 = a1.getMaxExpenses() - a1.getExpenses();

double r2 = a2.getMaxExpenses() - a2.getExpenses();

if (r1 == 0 && r2 == 0)

return 0;

else if (r1 == 0)

return r2;

else if (r2 == 0)

return r1;

else if (r1 < r2)

return r1;

else

return r2;

}

}

TestUseCCAccount.java

public class TestUseCCAccount {

public static void main(String[] args) {

CCAccount a1 = new CCAccount("Mario Rossi", "ABCD0001", 100);

CCAccount a2 = new CCAccount("Carlo Bruni", "EFGH0001", 100);

a1.spend(10);

a2.spend(20);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseCCAccount.getMinRemainingExpenses(a1, a2));

System.out.println();

a1.spend(30);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseCCAccount.getMinRemainingExpenses(a1, a2));

System.out.println();

a1.spend(60);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseCCAccount.getMinRemainingExpenses(a1, a2));

System.out.println();

a2.spend(80);

System.out.println(a1);

System.out.println(a2);

System.out.println(UseCCAccount.getMinRemainingExpenses(a1, a2));

}

}

14. Design and implement a class ExchangeMachine. The class should model an exchange cash dispenser that accepts Dollar bills and provides Euro bills applying pre-set exchange rates (dependent on the amount of money to be changed).

We're interested in the following features:

* a **constructor** should create an ExchangeMachine with an initial cash amount of 0 Euros;
* a get method should return the **current cash amount** in Euros stored in the machine;
* an add method should **add a given amount of Euros** to the current cash amount in the machine;
* an **exchange** method should accept an amount of dollars as a parameter and it should return the corresponding amount of Euros decreasing the current amount of cash.  
  Note, however, that an error message should be printed out and 0 should be returned if the current cash amount is not enough.

The Euro/Dollar exchange rate is defined as the amount of Euros corresponding to one Dollar:  
0.5 if the exchanged amount is less than 10 dollars,  
0.9 if the exchanged amount is between 10 dollars and 100 dollars and  
1.0 if the exchanged amount is more than 100 dollars.

public class ExchangeMachine {

// attributes:

private double currentCash; // euros stored in the exchange machine

// constructor

public ExchangeMachine() {

currentCash = 0;

}

// get current cash and add to current cash:

public double getCurrentCash() {

return currentCash;

}

public void addToCurrentCash(double s) {

currentCash = currentCash + s;

}

public double exchange(double dollars) {

double rate = 0.0;

if (dollars < 10.0) {

rate = 0.5;

} else if (dollars >= 10.0 && dollars <= 100.0) {

rate = 0.9;

} else {

rate = 1.0;

}

double euros = dollars \* rate;

if (euros <= currentCash) {

currentCash = currentCash - euros;

return euros;

} else {

System.out.println("not enough euro cash stored");

return 0;

}

}

}

15. Define a class CD that handles data known about a CD, such as composer, title, publisher and owner.

The following methods are of interest:

* *CD* (constructor) creates a new CD object, given the composer, title and publisher;
* *printCD* prints out the title and publisher of a CD on two lines;
* *printCDOwner* prints out the owner of the CD;
* *getComposer* returns the composer of the CD;
* *getTitle* returns the title of the CD;
* *getPublisher* returns the publisher of the CD;
* *getOwner* returns the owner of the CD;
* *setOwner* sets the owner of the CD to the string passed as parameter

public class CD {

// data fields

private String author;

private String title;

private String publisher;

private String owner;

//constructor

public CD (String aut, String tit, String pub) {

author = aut;

title = tit;

publisher = pub;

owner = null;

}

public void printCD() {

System.out.println("Author: " + author);

System.out.println("Title: " + title);

System.out.println("Publisher: " + publisher);

}

public void printOwnerCD() {

System.out.println("Name of the owner: " + owner);

}

public String getAuthor() {

return author;

}

public String getTitle() {

return title;

}

public String getPublisher() {

return publisher;

}

public String getOwner() {

return owner;

}

public void setOwner(String own) {

owner = own;

}

}

16. Write a Java program that uses the class CD from the last exercise.  
The class' main method should perform the following operations:

* prompt for the title, the composer and the publisher of a CD and create a CD object using this data;
* repeat this step for a second CD;
* show the first CD;
* show the second CD;
* prompt for the owner of the first CD and update the CD object accordingly;
* show the stored data about the first CD including it's owner.

public class ClientCD {

public static void main(String[] args) {

String author = JOptionPane.showInputDialog("Insert author first CD");

String title = JOptionPane.showInputDialog("Insert title first CD");

String publisher = JOptionPane.showInputDialog("Insert publisher first CD");

CD firstCD = new CD(author, title, publisher);

author = JOptionPane.showInputDialog("Insert author second CD");

title = JOptionPane.showInputDialog("Insert title second CD");

publisher = JOptionPane.showInputDialog("Insert publisher second CD");

CD secondCD = new CD(author, title, publisher);

System.out.println("Data first CD");

firstCD.printCD();

System.out.println();

System.out.println("Data second CD");

secondCD.printCD();

String owner = JOptionPane.showInputDialog("Insert owner first CD");

firstCD.setOwner(owner);

System.out.println();

System.out.println("Data first CD");

firstCD.printCD();

firstCD.printOwnerCD();

System.exit(0);

}

}

17. Define a class for representing journey tickets. Each ticket should contain the following information:

* The first name and the last name of the passenger (two strings)
* The name of the city where the journey begins (a string)
* The name of the country where the journey begins (a string)
* The name of the city where the journey ends (a string)
* The name of the country where the journey ends (a string)
* The net fare (without VAT) for the ticket (a double)

The Ticket class should have the following methods:

* A constructor to create *Ticket* objects, with three parameters: one for the first name of the passenger, one for the last name of the passenger and one for the net fare (without VAT).
* *setDeparturePoint()*, which takes two parameters of type String (representing the name of a city and of a country) and sets the name of the city and of the country where the journey begins.
* *setArrivalPoint()*, which takes two parameters of type String (representing the name of a city and of a country) and sets the name of the city and of the country where the journey ends.
* *getFare()*, which takes one prameter of type double (representing the current VAT rate - for example: 0.20) and returns the ticket fare including the VAT.
* *printFullForm()*, which prints the ticket's information in a full form (in which names are not truncated and the ticket fare, including 20% VAT is shown). For example: ''Johnny English: Milan (Italy) - Paris (France), 300.00 EUR''
* *printCompactForm()*, which prints the ticket's information in a compact form, in which the first name is replaced with its first letter and a dot, while the name of a country is abbreviated with its first two letters in upper case. For example: ''J. English: Milan (IT) - Paris (FR)''.

For the design of the class, follow the methodology presented in the lectures.

public class Ticket

{

private String firstName;

private String lastName;

private String fromCity;

private String fromCountry;

private String toCity;

private String toCountry;

private double netFare;

public Ticket(String fn, String ln, double f)

{

firstName = fn;

lastName = ln;

netFare = f;

}

public void setDeparturePoint(String city, String country)

{

fromCity = city;

fromCountry = country;

}

public void setArrivalPoint(String city, String country)

{

toCity = city;

toCountry = country;

}

public void printFullForm()

{

StringBuffer buf = new StringBuffer();

buf.append(firstName + " " + lastName + ": ");

buf.append(fromCity + " (" + fromCountry + ")");

buf.append(" - ");

buf.append(toCity + " (" + toCountry + ")");

buf.append(", " + getFare(0.20) + " EUR");

System.out.println(buf);

}

public void printCompactForm()

{

StringBuffer buf = new StringBuffer();

buf.append(firstName.substring(0, 1) + ". " + lastName + ": ");

buf.append(fromCity + " (" + fromCountry.substring(0, 2).toUpperCase() + ")");

buf.append(" - ");

buf.append(toCity + " (" + toCountry.substring(0, 2).toUpperCase() + ")");

System.out.println(buf);

}

public double getFare(double vat)

{

return netFare \* (1.0 + vat);

}

}

18. Define a Java class implementing an example client for the class Ticket of Exercise 17. The example class should perform the following operations:

* Read from the keyboard the first and the last name of a passenger, and create a Ticket object.
* Read from the keyboard the names of a city and of a country, and use them to update the departure point of the Ticket object.
* Read from the keyboard the names of a city and of a country, and use them to update the arrival point of the Ticket object.
* Display the ticket's information in full form.
* Display the ticket's information in compact form.

public class TicketTest

{

public static void main(String[] a)

{

String ln, fn, city, country;

double f;

Scanner sc = new Scanner(System.in);

System.out.print("first name: ");

fn = sc.next();

System.out.print("last name: ");

ln = sc.next();

System.out.print("fare: ");

f = sc.nextDouble();

Ticket t = new Ticket(fn, ln, f);

System.out.print("departure city: ");

city = sc.next();

System.out.print("departure country: ");

country = sc.next();

t.setDeparturePoint(city, country);

System.out.print("arrival city: ");

city = sc.next();

System.out.print("arrival country: ");

country = sc.next();

t.setArrivalPoint(city, country);

t.printFullForm();

t.printCompactForm();

}

}

19. Implement a class Bookstore for representing bookstores having books in two languages: Italian and German. Each Bookstore object should maintain the following information:

* the number of books in Italian language;
* the number of books in German language.

The Bookstore class should export to its clients the following methods:

* A constructor to create a Bookstore object with a given initial number of books. The constructor has two parameters: one indicates the initial number of books in Italian language, another the initial number of books in German language.
* addBookI, which takes one parameter of type int and increases the number of books in Italian language by its value.
* addBookG, which takes one parameter of type int and increases the number of books in German language by its value.
* getOverallBooksNumber, which returns the total number of books (both in Italian and German language) stored in a bookstore.
* getPercentageOfBooksI, which returns the percentage of books in Italian language. The required percentage is a real number (i.e., float) and is equal to:

*100.0 \* number\_of\_books\_in\_italian\_language / overall\_number\_of\_books*

The method should work correctly also in the case where the overall number of books is 0, and in this case it should return 0 as percentage.

* getPercentageOfBooksG, which returns the percentage of books in German language. The required percentage is a real number (i.e., float) and is equal to:

*100.0 \* number\_of\_books\_in\_german\_language / overall\_number\_of\_books*

The method should work correctly also in the case where the overall number of books is 0, and in this case it should return 0 as percentage.

Finally, implement a client class which:

* Reads from the keyboard the initial number of books in Italian language, and the number of books in German language
* Creates a Bookstore object with a given initial number of books.
* Reads from the keyboard an additional number of books in Italian language and updates the object with its value.
* Reads from the keyboard an additional number of books in German language and updates the object with its value.
* Prints the percentage of books in Italian language.
* Prints the percentage of books in German language.
* Depending on the percentage of books in each language, prints one of the messages:
  + "The majority of books is in Italian"
  + "The majority of books is in German"
  + "The numbers of books in Italian and German are equal"

public class Bookstore {

private int germanBooks;

private int italianBooks;

public Bookstore(int itBooks, int gerBooks) {

italianBooks= itBooks;

germanBooks = gerBooks;

}

public void addBookI(int count) {

italianBooks += count;

}

public void addBookG(int count) {

germanBooks += count;

}

public int getOverallBooksNumber() {

return italianBooks + germanBooks;

}

public float getPercentageOfBooksI() {

if (getOverallBooksNumber() == 0) return 0f;

return (float)(100.0 \* italianBooks / getOverallBooksNumber());

}

public float getPercentageOfBooksG() {

if (getOverallBooksNumber() == 0) return 0f;

return (float)(100.0 \* germanBooks / getOverallBooksNumber());

}

}

Client.java

public class Client {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of italian books: ");

int num\_of\_it\_books = scanner.nextInt();

System.out.println("Enter the number of german books: ");

int num\_of\_ger\_books = scanner.nextInt();

Bookstore my\_bookstore = new Bookstore(num\_of\_it\_books, num\_of\_ger\_books);

System.out.println("Enter the value to increase the number of italian books: ");

int count = scanner.nextInt();

my\_bookstore.addBookI(count);

System.out.println("Enter the value to increase the number of german books: ");

count = scanner.nextInt();

my\_bookstore.addBookG(count);

float pi = my\_bookstore.getPercentageOfBooksI();

float pg = my\_bookstore.getPercentageOfBooksG();

System.out.println("Percentage of books in italian language: " + pi);

System.out.println("Percentage of books in german language: " + pg);

if (pi > pg) System.out.println("The majority of books is in Italian");

else if (pg > pi) System.out.println("The majority of books is in German");

else System.out.println("The numbers of books in Italian and German are equal");

}

}

20. Plan and implement a class Car with the following properties:

* licensePlate is a string holding the car's license number;
* mileage is an integer holding the car's total covered distance in km;
* acquisitionYear is an integer holding the year the car was first bought;
* basePrice is a double holding the car's base price when it was first bought;
* owner is the current car's owner.

The class Car should define the following methods:

* the constructor: it takes the license number, the acquisition year and the base price as parameters and sets mileage to 0 and owner to null;
* so called *"getters"*: i.e. methods for all attributes that return the field's value;
* a method addToMileage that increments mileage by an integer value, passed as parameter;
* a method setOwner that sets the owner field to a string value, passed as parameter.

Furthermore, design and implement a client class with a *static* method getCurrentValue that computes the car's current value, given a car object and the current year. Use the equation:

*currentValue = basePrice \* 0.8 ^ (currentYear - acquisitionYear)*

where "*a ^ b*" means "*a to the power b*". Use the pow method from the java.lang.Math package.

public class Car {

private String licensePlate;

private int mileage;

private short acquisitionYear;

private float basePrice;

private String owner;

public Car(String t, short ai, float pb) {

licensePlate = t;

acquisitionYear = ai;

basePrice = pb;

mileage = 0;

owner = null;

}

public String getLicensePlate() {

return licensePlate;

}

public int getMileage() {

return mileage;

}

public short getAcquisitionYear() {

return acquisitionYear;

}

public float getBasePrice() {

return basePrice;

}

public String getOwner() {

return owner;

}

public void addToMileage(int km) {

mileage += km;

}

public void setOwner(String p) {

owner = p;

}

}

**CarClient.java**

public class CarClient {

static double getCurrentPrice(int currentYear, Car mycar) {

int diff = currentYear - mycar.getAcquisitionYear();

return mycar.getBasePrice() \* Math.pow(0.8, diff);

}

}

21. Design and realize a class Experiment that stores data obtained from a temperature measurement. The data values are delivered as a string of space-separated integers (at least one) and are stored in an array of integers.  
The class Experiment should export the following functionalities:

* creation of an instance of an experiment, given a string with data values;
* computation of the number of stored values;
* computation of the sum of all stored values;
* computation of the average of the stored values (use a double!);
* computation of the smallest value;
* computation of the largest value.

Hint: use the StringTokenizer class once more to implement the constructor. Use its countTokens methods to count the number of integers in the input string.

Extend the class Experiment with the following functionalities:

* verification that two experiments are *equal* - that is, they hold the same values in the same order;
* verification that two experiments are *similar* - that is, they hold the same values, regardless the of order and regardless of how often each value appears;
* computing the *merge* two experiments - that is, a new experiment that contains the values from both experiments;
* checking whether a given value is present in an experiment's data set
* counting how often a given value is present in an experiment's data set
* finding the first value in an experiment's data set that appears at least twice;
* plotting an experiment's data set using text output: each text line should contain a number of stars (\*) proportional to each data value up to a maximum of 80; in case of negative values just a single minus sign (-) should be printed;

Finally, implement a method toString that returns a string representation of the experiment's data values.

import java.util.StringTokenizer;

public class Experiment {

private int[] data;

public Experiment(String s) {

StringTokenizer temp = new StringTokenizer(s);

data = new int[temp.countTokens()];

int i = 0;

while (temp.hasMoreTokens()) {

String t = temp.nextToken();

data[i] = Integer.parseInt(t);

i++;

}

}

/\* alternative implementation:

public Experiment(String s) {

StringTokenizer temp = new StringTokenizer(s);

data = new int[temp.countTokens()];

for (int i = 0; i < data.length; i++)

data[i] = Integer.parseInt(temp.nextToken());

}

\*/

public int getNumber() {

return data.length;

}

public int sum() {

int su = 0;

for (int i = 0; i < data.length; i++)

su = su + data[i];

return su;

}

public double average(){

return ((double)sum()) / getNumber();

}

public int minimum(){

int min = data[0];

for (int i = 1; i < data.length; i++)

if (data[i] < min) min = data[i];

return min;

}

public int maximum(){

int max = data[0];

for (int i = 1; i < data.length; i++)

if (data[i] > max) max = data[i];

return max;

}

public boolean equalTo(Experiment e) {

if (data.length != e.data.length) return false;

for (int i = 0; i < data.length; i++)

if (data[i] != e.data[i]) return false;

return true;

}

/\* alternative implementation that does not use intermediate return statements

public boolean equals(Experiment e) {

boolean res;

if (data.length != e.data.length)

res = false;

else {

res = true;

for (int i = 0; i < data.length; i++)

res = res && data[i] == e.data[i];

}

return res;

}

\*/

public boolean similar(Experiment e) {

for (int i = 0; i < data.length; i++)

if (!pcontains(data[i], e.data)) return false;

for (int i = 0; i < e.data.length; i++)

if (!pcontains(e.data[i], data)) return false;

return true;

}

private static boolean pcontains(int d, int[] a) {

for (int i = 0; i < a.length; i++)

if (d == a[i]) return true;

return false;

}

/\* alternative implementation with nested loops:

public boolean similar(Experiment e) {

boolean res = true;

for (int i = 0; i < data.length; i++) {

boolean present = false;

for (int j = 0; j < e.data.length; j++)

present = present || data[i] == e.data[j];

res = res && present;

}

for (int i = 0; i < e.data.length; i++) {

boolean present = false;

for (int j = 0; j < data.length; j++)

present = present || e.data[i] == data[j];

res = res && present;

}

return res;

}

\*/

public Experiment merge(Experiment e) {

int[] res = new int[data.length + e.data.length];

for(int i = 0; i < data.length; i++)

res[i] = data[i];

for(int i = 0; i < e.data.length; i++)

res[data.length+i] = e.data[i];

return new Experiment(res);

}

// private constructor

private Experiment(int[] a) {

data = a;

}

public boolean contains(int d) {

return pcontains(d, data);

}

/\* alternative implementation:

public boolean contains(int d) {

for (int i = 0; i < data.length; i++)

if (d == data[i]) return true;

return false;

}

\*/

public int count(int d) {

int counter = 0;

for (int i = 0; i < data.length; i++)

if (data[i] == d) counter++;

return counter;

}

public boolean hasRepeatedValues() {

for (int i = 0; i < data.length; i++)

for (int j = i+1; j < data.length; j++)

if (data[i] == data[j]) return true;

return false;

}

public void plot() {

double scale = 80.0/maximum();

for (int i = 0; i < data.length; i++) {

for (int j = 0; j < Math.round(data[i]\*scale); j++)

System.out.print("\*");

System.out.println();

}

}

public String toString() {

String res = "";

for (int i = 0; i < data.length; i++)

res = res + data[i] + " ";

return res;

}

}

22. Write a client class for the class Experiment that does the following:

1. create two experiments from data inserted by the user;
2. print the number of values, the sum as well as the average and minimum values of the two experiments;
3. check whether the two experiments are equal and print a message accordingly;
4. if they're not equal, check whether they are similar and print a message accordingly;
5. create an experiment that contains the data of both experiments;
6. have the user input an integer value and print out how often it is present in the newly merged experiment;
7. check if there are repeated values in the merged experiment;
8. have the merged experiment plotted;

Hint: to have the user input a data set, implement a static method that:

1. repeatedly calls JOptionPane.showInputDialog() (or the nextLine() or next() method of the Scanner class) until the user presses Cancel,
2. builds a space-separated string from the so-inserted integer values,
3. creates an Experiment object by passing the string to the constructor of Experiment, and
4. returns such an object.

public class Client {

private static String readExperimentData() {

String s = JOptionPane.showInputDialog(

"Insert value (Press Cancel to stop)");

String insData = "";

while (s!= null){

insData += s + " ";

s = JOptionPane.showInputDialog(

"Insert value (Press Cancel to stop)");

}

return insData;

}

private static void printExperiment(Experiment es) {

int num = es.getNumber();

int sum = es.sum();

double average = es.average();

int min = es.minimum();

int max = es.maximum();

System.out.println(

"Experiment: " + es +

"\nNumber of values stored: "+ num +

"\nSum: " + sum +

"\nAverage:" + average +

"\nMinimum: "+ min +

"\nMaximum: "+ max

);

}

public static void main (String[] args){

System.out.println("Insert data for experiment 1");

Experiment es1 = new Experiment(readExperimentData());

printExperiment(es1);

System.out.println();

System.out.println("Insert data for experiment 2");

Experiment es2 = new Experiment(readExperimentData());

printExperiment(es2);

if (es1.equalTo(es2))

System.out.println("\nThe two experiments are equal");

else {

System.out.println("\nThe two experiments are not equal");

if (es1.similar(es2))

System.out.println("The two experiments are similar");

else

System.out.println("The two experiments are not similar");

}

Experiment es3 = es1.merge(es2);

printExperiment(es3);

int d = Integer.parseInt(

JOptionPane.showInputDialog("Insert value"));

if (es3.contains(d))

System.out.println(d+ " is present " + es3.count(d) + " times");

else

System.out.println(d + " is not present");

if (es3.hasRepeatedValues())

System.out.println("There are repeated values in the two experiments");

else

System.out.println("There are no repeated values in the two experiments");

System.out.println("\nPlot of the two merged experiments");

es3.plot();

System.exit(0);

}

}

23. mplement a class Animal. It should define a variable holding the animal's name and the following methods:

* the constructor Animal creates an Animal object with the given name passed as parameter;
* makeSound prints out the sound of the generic animal (just print out some bogus string like "@#%&").

The dog, the cat and the parrot are special types of animals. Implement a specific class for each of them! Each specific class has a constructor and overrides makeSound such as to print out the sound the specific animal makes.

Furthermore, the parrot class should implement a method repeat that prints out the string parameter passed into it twice.

Finally implement two specific dog classes representing the Chiwawa and the Rottweiler breeds. Their makeSound methods are expected to print out "bau!" and "BAU!" respectively.

Animal.java

public class Animal {

private String name;

public Animal(String n) {

name = n;

}

public String makeSound() {

return "@#%&";

}

}

Cat.java

public class Cat extends Animal {

public Cat(String n) {

super(n);

}

public String makeSound() {

return "Miao";

}

}

Dog.java

public class Dog extends Animal {

public Dog(String n) {

super(n);

}

public String makeSound() {

return "Bau";

}

}

Parrot.java

public class Parrot extends Animal {

public Parrot(String n) {

super(n);

}

public String makeSound() {

return "Cra";

}

public String repeat(String s) {

return s + s;

}

}

Rottweiler.java

public class Rottweiler extends Dog {

public Rottweiler(String n) {

super(n);

}

public String makeSound() {

return "BAU";

}

}

Chiwawa.java

public class Chiwawa extends Dog {

public Chiwawa(String n) {

super(n);

}

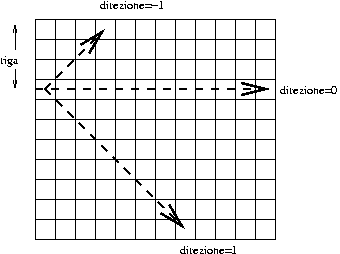
public String makeSound() {

return "bau";

}

}

24. The purpose of this exercise is to implement a game of battleship. The game is played on a grid of 10x10 cells.  
The computer-player is your enemy. It can randomly deploy a number of ships on the grid. Each ship occupies one cell. The computer-player has no weapons.  
The human-player tries to destroy all ships using as few turns as possible. At each turn he or she can shoot two different weapons: cannons and torpedos. The cannon can be shot at a cell at position *(row, column)* which is hit. The torpedo can be shot from a cell at position *(row, column)* in a direction given by the values -1, 0 and 1 as shown in the figure. The effect is that all cells on the torpedo's trajectory are hit.



**Part 1**

Implement a class Cell representing a cell on the game's 10x10 grid. It should store the following attributes:

* *hit*: true if the cell was hit at least once by a weapon;
* *occupied*: true if the computer-player has put a ship on it.

Cell should furthermore have the following methods:

* a constructor that initialises the cell as beeing not hit and not occupied;
* *getter*-methods for all attributes;
* a method to mark a cell as hit;
* a method to mark a cell as occupied;
* a method toString that returns:
  + a space ' ' if the cell has not been hit and it is not occupied;
  + a letter 'X' if the cell was occupied and has been hit (that is, a ship was destroyed);
  + a hash sign '#' if the cell was empty and has been hit (that is, a ship was missed).

public class Cell

{

private boolean occupied;

private boolean hit;

public Cell()

{

occupied = false;

hit = false;

}

public boolean hit()

{

return hit;

}

public boolean occupied()

{

return occupied;

}

public void setHit()

{

hit = true;

}

public void setOccupied()

{

occupied = true;

}

public void set(boolean occupied, boolean hit)

{

this.occupied = occupied;

this.hit = hit;

}

public String toString()

{

if (!hit)

return " "; // don't give information about wether the cell is occupied or not

if (occupied)

return "X"; // cell was occupied

else

return "#"; // cell was empty

}

}

**Part 2**

Implement a class BattleShip that holds 10x10 cells. BattleShip should have the following methods:

* a constructor that occupies 20% of all cells randomly (*Hint:* for each cell, call the static method random from java.lang.Math - if the resulting random value is less than 0.2 the cell is marked as *occupied*, otherwise it is marked as *not occupied*).
* getNumberOfShips returns the number of occupied and not hit cells;
* shootCannon is passed two integers *row* and *column*, and marks the cell at that position as hit;
* launchTorpedo is passed two integers *row* and *column* as well as a *direction* (-1, 0 or 1), and marks all cells on the torpedo's trajectory as hit.
* toString returns the string that represents the whole grid (10 lines of 10 character each), calling each cell's toString method.

import java.io.\*;

public class BattleShip

{

private Cell[][] cells;

private static final int DIM = 10;

private static final String FILE\_NAME = "battleship.sav";

public BattleShip()

{

cells = new Cell[DIM][DIM];

for (int r = 0; r < DIM; r++)

for (int c = 0; c < DIM; c++) {

cells[r][c] = new Cell();

if (Math.random() < 0.2)

cells[r][c].setOccupied();

}

}

public int getNumberOfShips()

{

int navi = 0;

for (int r = 0; r < DIM; r++)

for (int c = 0; c < DIM; c++)

if (cells[r][c].occupied() && !cells[r][c].hit()) navi++;

return navi;

}

public void shootCannon(int r, int c)

{

if (r >= 0 && r < DIM && c >= 0 && c < DIM)

cells[r][c].setHit();

}

public void launchTorpedo(int r, int c, int direzione)

{

while (c >= 0 && r >= 0 && c < DIM && r < DIM) {

cells[r][c].setHit();

c++;

r += direzione;

}

}

public String toString()

{

String s = " ";

for (int i = 0; i < DIM; i++)

s = s + (i % 10);

s = s + "\n +";

for (int i = 0; i < DIM; i++)

s = s + "-";

s = s + "+\n";

for (int r = 0; r < DIM; r++) {

s = s + (r % 10) + "|";

for (int c = 0; c < DIM; c++)

s = s + cells[r][c];

s = s + "|\n";

}

s = s + " +";

for (int i = 0; i < DIM; i++)

s = s + "-";

s = s + "+\n";

return s;

}

public void open() throws IOException

{

BufferedReader input = new BufferedReader(new FileReader(FILE\_NAME));

for (int r = 0; r < DIM; r++) {

// read row r

for (int c = 0; c < DIM; c++) {

int ch = input.read();

switch (ch) {

case '0':

cells[r][c].set(false, false);

break;

case '1':

cells[r][c].set(false, true);

break;

case '2':

cells[r][c].set(true, false);

break;

case '3':

cells[r][c].set(true, true);

break;

default:

throw new IOException("Syntax error in file " + FILE\_NAME);

}

}

// read \n at the end of line

int ch = input.read();

if (ch != '\n') {

throw new IOException("Syntax error in file " + FILE\_NAME);

}

} // next row

input.close();

}

public void save() throws IOException

{

BufferedWriter output = new BufferedWriter(new FileWriter(FILE\_NAME));

for (int r = 0; r < DIM; r++) {

// save row r

for (int c = 0; c < DIM; c++) {

boolean occupied = cells[r][c].occupied();

boolean hit = cells[r][c].hit();

if (!occupied && !hit) {

output.write('0');

} else if (!occupied && hit) {

output.write('1');

} else if (occupied && !hit) {

output.write('2');

} else {

output.write('3');

}

}

// save \n at the end of line

output.write('\n');

} // next row

output.close();

}

}

**Part 3**

Implement a client class for BattleShip. It should:

* create an instance of BattleShip;
* ask the human-player to insert brief commands repeatedly reading them from System.in. The commands are:
  + shoot a cannon on a cell;
  + launch a torpedo from a cell;
  + show the ships left to hit;
  + quit the game.

The command shoud be executed and the possibly modified game grid should be shown again in text mode;  
the game should end when the human-player decides to quit or when there are no ships left to hit.

import java.io.\*;

public class Game {

private static final String menu =

"commands are:\n" +

" s - shoot cannon\n" +

" t - launch torpedo\n" +

" n - show number of ships to hit left\n" +

" r - read a saved game from file battleship.sav\n" +

" w - write a game to battleship.sav\n" +

" q - quit";

public static void main (String[] args) throws IOException {

BattleShip bn = new BattleShip();

int row, column, direction;

String line, msg;

char choice;

boolean update = false;

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

System.out.println("\nWelcome to battleship! Insert h to see available commands.");

System.out.println("There are " + bn.getNumberOfShips() + " ships to hit.");

msg = "";

do {

if (update) {

System.out.println("\n" + bn);

update = false;

}

System.out.println(msg + "\n");

if (bn.getNumberOfShips() == 0) {

System.out.println("Congratulations! All ships have been hit.");

break;

}

System.out.print("command: ");

line = input.readLine();

if (line == null)

choice = 'q';

else if (line.length() == 0)

choice = ' ';

else

choice = line.charAt(0);

switch (choice) {

case 'h':

msg = menu;

break;

case 's':

System.out.print("row: ");

row = Integer.parseInt(input.readLine());

System.out.print("column:");

column = Integer.parseInt(input.readLine());

bn.shootCannon(row, column);

msg = "Cell hit.";

update = true;

break;

case 't':

System.out.print("row: ");

row = Integer.parseInt(input.readLine());

System.out.print("column: ");

column = Integer.parseInt(input.readLine());

System.out.print("direction [-1, 0, 1]: ");

direction = Integer.parseInt(input.readLine());

bn.launchTorpedo(row, column, direction);

msg = "Torpedo launched.";

update = true;

break;

case 'n':

msg = "Ships to hit left: " + bn.getNumberOfShips() + ".";

break;

case 'r':

try {

bn.open();

update = true;

msg = "Game read from file.";

} catch(IOException e) {

System.out.println("Error in reading the game from file.");

System.out.println(e.getMessage());

}

break;

case 'w':

try {

bn.save();

msg = "Game written to file.";

} catch(IOException e) {

System.out.println("Error in writing the game to file.");

System.out.println(e.getMessage());

}

break;

case 'q':

break;

default:

msg = "Unknown command - insert h to see available commands.";

}

} while (choice != 'q');

System.out.println("Bye.\n");

System.exit(0);

}

}

25. Write a class like below

public class Document {

private String author, title;

private int copies;

// constructor without parameters that, given author and title, constructs a

// document for which the number of available copies is 0

public Document(String a, String t) {

author = a; title = t; copies = 0;

}

// returns the information about the document

public String toString( ) {

return "Doc of " + author + " \"" + title + "\" Copies: " + copies;

}

// increments the number of available copies

public void createCopies(int n) {

copies += n;

}

// decrements the number of available copies

public void sellCopies(int n) {

if (n <= copies)

copies -= n;

else

copies = 0;

}

// returns the author of the document

public String getAuthor() {

return author;

}

// returns the title of the document

public String getTitle() {

return title;

}

// returns the number of copies of the document

public int getCopies() {

return copies;

}

}

Extend the class Document of shown above with a static method read(). The method should take a parameter *br* of type BufferedReader, read data from *br*, and create and return a new instance of the class Document. More precisely, the method read() should do the following:

1. It reads a single line of text from *br* into a string variable. The line contains the author's name, the document's title, and the number of copies, separated by ';'. The following is an example of a line formatted in such a way:

Bruce Tate;Beyond Java;53

1. It extracts the information from the string using the split() method of the class String.
2. It constructs a new Document object using the information extracted from the line of text, and returns it.

import java.io.\*;

public class Document {

private String author, title;

private int copies;

// constructor without parameters that, given author and title, constructs a

// document for which the number of available copies is 0

public Document(String a, String t) {

author = a; title = t; copies = 0;

}

// returns the information about the document

public String toString( ) {

return "Doc of " + author + " \"" + title + "\" Copies: " + copies;

}

// increments the number of available copies

public void createCopies(int n) {

copies += n;

}

// decrements the number of available copies

public void sellCopies(int n) {

if (n <= copies)

copies -= n;

else

copies = 0;

}

// returns the author of the document

public String getAuthor() {

return author;

}

// returns the title of the document

public String getTitle() {

return title;

}

// returns the number of copies of the document

public int getCopies() {

return copies;

}

//reads an object from the given input channel

public static Document read(BufferedReader reader) throws IOException {

String info[] = reader.readLine().split(";");

Document document = new Document(info[0], info[1]);

document.copies = Integer.parseInt(info[2]);

return document;

}

}

Implement a client of the class Document that reads a set of Document objects from a given input file, computes the average number of copies, and writes to and output file all those documents whose number of copies is greater than the average.

More precisely, the client application should do the following:

1. It reads from the command line the name of an input file and the name of an output file. The input file should contain in the first line an integer *numDocs*, and in the next *numDocs* lines the information about documents, one document per line, formatted according to part **A** of the exercise.
2. It reads from the first line of the input file the number *numDocs* of documents, and creates an array of Document objects of size *numDocs*.
3. Using the method read(), it reads from the remaining lines of the input file the Document objects and stores them into the array.
4. Based on the documents stored in the array, it computes the average number of copies.
5. It writes to the output file all documents whose number of copies is greater than the average. The first line of the file should again contain the number of documents stored in the file, and all remaining lines of the file should be formatted in the same way as for the input file.

The file [data.txt](https://www.inf.unibz.it/~calvanese/teaching/06-07-ip/labs/lab-09/data.txt) is an example input file for the exercise.

Data.txt

15

Kathy Sierra, Bert Bates;Head First Java, 2nd Edition;45

Joshua Bloch;Effective Java Programming Language Guide;44

Kathy Sierra, Bert Bates;Head First Java, 1nd Edition;33

David Flanagan;Java In A Nutshell, 5th Edition;0

Barry Burd, Barry Burd;Beginning Programming with Java for Dummies;120

Andrew Davison;Killer Game Programming in Java;32

Bruce Tate;Beyond Java;53

Pat Niemeyer, Jonathan Knudsen;Learning Java;61

Bruce Eckel;Thinking in Java (3rd Edition);4

Ian Darwin;Java Cookbook, Second Edition;10

David Flanagan;Java Examples in a Nutshell, 3rd Edition;72

Aaron E. Walsh;Java for Dummies;54

Robert Lafore;Data Structures and Algorithms in Java (2nd Edition);90

Elliotte Harold, Elliotte Rusty Harold;Java Network Programming, Third Edition;40

Scott Oaks, Henry Wong;Java Threads;31

DocumentClient.java

import java.util.Scanner;

import java.io.\*;

public class DocumentClient {

public static void main(String[] args) throws IOException{

//reading names of the input and output files from a command line.

Scanner sc = new Scanner(System.in);

System.out.print("Specify the input file: ");

String inputFile = sc.nextLine();

System.out.print("Specify the output file: ");

String outputFile = sc.nextLine();

//opening input file for reading

BufferedReader br = new BufferedReader(new FileReader(inputFile));

//reading the number of objects and creating the array

int numberOfDocuments = Integer.parseInt(br.readLine());

Document[] documents = new Document[numberOfDocuments];

//reading documents from the input file;

for (int i = 0; i < numberOfDocuments; i++)

documents[i] = Document.read(br);

//computing the average number of copies

float avgCount = 0;

for (int i = 0; i < numberOfDocuments; i++)

avgCount += documents[i].getCopies();

avgCount /= numberOfDocuments;

//closing input file

br.close();

//opening output file for writing

PrintWriter pw = new PrintWriter(new FileWriter(outputFile));

//counting the number of output lines

int count = 0;

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i].getCopies() > avgCount)

count++;

//writing the data

pw.println(count);

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i].getCopies() > avgCount)

pw.println(documents[i].getAuthor() + ";" +

documents[i].getTitle() + ";" +

documents[i].getCopies());

//closing output file

pw.close();

}

}

Adapt the client application to catch the following exceptions:

* FileNotFoundException. The message "The file does not exist" should be displayed.
* IOException. The message "The file cannot be read" should be displayed.
* NumberFormatException. Occurs when an input file is not formatted correctly, i.e. the first line does not contain an integer representing the number of documents, or an integer representing the number of copies of a document is not present in the required position in a line containing the info for a document. The message "Incorrect formatting: the number of documents or the number of copies is missing" should be displayed

The application should terminate if any of the above mentioned exceptions occurs.

import java.util.Scanner;

import java.io.\*;

public class DocumentClient2 {

public static void main(String[] args) throws IOException{

//reading names of the input and output files from a command line.

Scanner sc = new Scanner(System.in);

System.out.print("Specify the input file: ");

String inputFile = sc.nextLine();

System.out.print("Specify the output file: ");

String outputFile = sc.nextLine();

//opening input file for reading

try {

BufferedReader br = new BufferedReader(new FileReader(inputFile));

//reading the number of objects and creating the array

int numberOfDocuments = Integer.parseInt(br.readLine());

Document[] documents = new Document[numberOfDocuments];

//reading documents from the input file;

for (int i = 0; i < numberOfDocuments; i++)

documents[i] = Document.read(br);

//computing the average number of copies

float avgCount = 0;

for (int i = 0; i < numberOfDocuments; i++)

avgCount += documents[i].getCopies();

avgCount /= numberOfDocuments;

//closing input file

br.close();

//opening output file for writing

PrintWriter pw = new PrintWriter(new FileWriter(outputFile));

//counting the number of output lines

int count = 0;

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i].getCopies() > avgCount)

count++;

//writing the data

pw.println(count);

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i].getCopies() > avgCount)

pw.println(documents[i].getAuthor() + ";" +

documents[i].getTitle() + ";" +

documents[i].getCopies());

//closing output file

pw.close();

} catch(FileNotFoundException e1) {

System.out.println("The file does not exist.");

} catch(IOException e2) {

System.out.println("The file cannot be read.");

} catch(NumberFormatException e3) {

System.out.println("Incorrect formatting: the number of documents or the number of copies is missing");

}

}

}

26. Define a class DocumentFormatException to represent document formatting exceptions. Extend the method read() of the class Document so that it throws this exception in the following two cases:

* The name of the author, the title of the document, or the number of copies is missing. This can be checked using the size of the array obtained by executing the split method. The correct size of the array is 3.
* The number of copies in the input line is not an integer. This can be checked by catching NumberFormatException while converting a string into an integer.

Also, modify the client application to catch DocumentFormatException. If the exception occurs the client application should do the following:

1. It displays the notification message including the index of the badly formatted line.
2. It sets the corresponding element of the array of Documents to null. Afterwards, while calculating the average number of copies and writing to the output file, it skips all elements of the array whose value is equal to null.

The file [data\_errors.txt](https://www.inf.unibz.it/~calvanese/teaching/06-07-ip/labs/lab-09/data_errors.txt) is an example input file for the exercise, containing various types of errors.

import java.io.\*;

public class Document {

private String author, title;

private int copies;

// constructor without parameters that, given author and title, constructs a

// document for which the number of available copies is 0

public Document(String a, String t) {

author = a; title = t; copies = 0;

}

// returns the information about the document

public String toString( ) {

return "Doc of " + author + " \"" + title + "\" Copies: " + copies;

}

// increments the number of available copies

public void createCopies(int n) {

copies += n;

}

// decrements the number of available copies

public void sellCopies(int n) {

if (n <= copies)

copies -= n;

else

copies = 0;

}

// returns the author of the document

public String getAuthor() {

return author;

}

// returns the title of the document

public String getTitle() {

return title;

}

// returns the number of copies of the document

public int getCopies() {

return copies;

}

//reads an object from the given input channel

public static Document read(BufferedReader reader)

throws DocumentFormatException, IOException {

String info[] = reader.readLine().split(";");

if (info.length != 3) throw new DocumentFormatException();

Document document = new Document(info[0], info[1]);

try {

document.copies = Integer.parseInt(info[2]);

} catch(NumberFormatException e) {

throw new DocumentFormatException();

}

return document;

}

}

public class DocumentFormatException extends Exception {

public DocumentFormatException() {

super("The input line is badly formated");

}

}

import java.util.Scanner;

import java.io.\*;

public class DocumentClient3 {

public static void main(String[] args) throws IOException{

//reading names of the input and output files from a command line.

Scanner sc = new Scanner(System.in);

System.out.print("Specify the input file: ");

String inputFile = sc.nextLine();

System.out.print("Specify the output file: ");

String outputFile = sc.nextLine();

//opening input file for reading

try {

BufferedReader br = new BufferedReader(new FileReader(inputFile));

//reading the number of objects and creating the array

int numberOfDocuments = Integer.parseInt(br.readLine());

Document[] documents = new Document[numberOfDocuments];

int badlyFormatedLines = 0;

//reading documents from the input file;

for (int i = 0; i < numberOfDocuments; i++) {

try {

documents[i] = Document.read(br);

} catch(DocumentFormatException e) {

System.out.println("Line " + i + " is badly formated");

badlyFormatedLines++;

documents[i] = null;

}

}

//computing the average number of copies

float avgCount = 0;

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i] != null)

avgCount += documents[i].getCopies();

avgCount /= (numberOfDocuments - badlyFormatedLines);

//closing input file

br.close();

//opening output file for writing

PrintWriter pw = new PrintWriter(new FileWriter(outputFile));

//counting the number of output lines

int count = 0;

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i] != null && documents[i].getCopies() > avgCount)

count++;

//writing the data

pw.println(count);

for (int i = 0; i < numberOfDocuments; i++)

if (documents[i] != null && documents[i].getCopies() > avgCount)

pw.println(documents[i].getAuthor() + ";" +

documents[i].getTitle() + ";" +

documents[i].getCopies());

//closing output file

pw.close();

} catch(FileNotFoundException e1) {

System.out.println("The file does not exist.");

} catch(IOException e2){

System.out.println("The file cannot be read.");

} catch(NumberFormatException e3) {

System.out.println("Incorrect formatting: the number of documents is missing");

}

}

}

27. A monochromatic image can be seen as a matrix of elements that are **0** or **1**. If the image is shown on a screen, a **1** means a picture element (called *pixel*) is on and **0** means a *pixel* is off. A matrix of this kind is called a *bitmap*.  
Implement a class Map that represents a monochromatic image as a *bitmap*. To keep things simple use int as the type for the matrix elements.  
Besides the constructor, Map should have the following two methods:

* toString returns a string that is a textual representation of the stored *bitmap* - you might show **0** as space (" ") and **1** as hash sign ("#");
* a method floodfill that takes the coordinates of the seed point (row r and column c) as parameters and fills the connected area of **0**-pixels the seed point is in with **1**-pixels - remember to check the boundaries of the bitmap as well to avoid overflow.

Write a class Main that creates a Map object given the following example matrix:

int[][] image =

{

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

};

Print out the Map object before and after calling the floodfill method for row 6 and column 6.  
This is how the result should look:

|  |  |
| --- | --- |
| ####  ### ## #  # ### #  # #  # #  # #  # #  # #  #### #  # #  # #  ## #  #####  before method call | ####  ### #######  #############  ##############  ###############  ###############  ###############  ###############  ##############  #########  #########  ########  #####  after method call |

public class Map

{

private int[][] bitmap;

public Map (int[][] m)

{

bitmap = m;

}

public String toString()

{

int i, j;

StringBuffer buf = new StringBuffer();

for (i = 0; i < bitmap.length; i++) {

for (j = 0; j < bitmap[i].length; j++) {

if (bitmap[i][j] == 0) {

buf.append(' ');

} else {

buf.append('#');

}

}

buf.append('\n');

}

return buf.toString();

}

public void floodfill(int row, int col)

{

if (!inRange(row, col) || bitmap[row][col] == 1)

return;

bitmap[row][col] = 1;

if (inRange(row - 1, col))

floodfill(row - 1, col);

if (inRange(row + 1, col))

floodfill(row + 1, col);

if (inRange(row, col - 1))

floodfill(row, col - 1);

if (inRange(row, col + 1))

floodfill(row, col + 1);

}

private boolean inRange(int row, int col)

{

if (row >= 0 && row < bitmap.length && col >= 0 && col < bitmap[row].length)

return true;

else

return false;

}

}

public class Main

{

public static void main(String args[])

{

int[][] image =

{

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

};

Map myMap = new Map(image);

System.out.println(myMap);

myMap.floodfill(6, 6);

System.out.println(myMap);

}

}

28. The municipality needs a system to handle the queues at their offices. At the counter, persons in need of documents from the offices can get a unique, progressive number that defines their position in the queue. The requests are handled according to this number.

Implement a class Request with the following attributes:

* the name of the person that issued the request (a string);
* the requests's progressive number (an integer).

The class should implement the following features:

* a constructor to create a new request, given a person's name and a progressive number;
* a method getPerson that returns the person's name;
* a method getNumber that returns the progressive number;
* a method toString that allows one to get a String representation of a request object.

import java.io.\*;

public class Request

{

private String person;

private int number;

public Request(String p, int n)

{

person = p;

number = n;

}

public String getPerson()

{

return person;

}

public int getNumber()

{

return number;

}

public String toString()

{

return number + ": " + person;

}

}

29. Implement a class Counter that stores a queue of all requests issued at the counter.

The class should implement the following features:

* a constructor to create an empty counter, that is a counter with an empty queue;
* a method addRequest that, given a person's name appends a new request to the counter's queue and returns the progressive number associated to this request;
* a method handleRequest that gets the next request from the queue; if there are no requests left an exception should be thrown;
* a method giveUp that, given a progressive number eliminates the corresponding request from the queue; if there is no such request, an exception is thrown;
* a method waitingTime that, given a progressive number computes the number of requests that are queued before that request; if there is no such request, throw an exception;
* a method toString that returns a string representation of the request objects in the queue.

Hint: implement the queue using a dynamic array:

* The array's size is initially set to 10.
* If the array is full and addRequest is called to append one more request, create a new array double the size of the old one and copy the old array's content into the new one;
* If the array is less than one fourth full after calling handleRequest, create a new array half the size of the old one and copy the old array's content into the new one;

public class Counter

{

private static final int DIM = 10; // initial array dimension

private Request[] queue;

private int queueSize;

private int progressiveNumber;

public Counter ()

{

queue = new Request[DIM] ;

progressiveNumber = 0;

queueSize = 0;

}

public int addRequest(String nome)

{

// if necessary, grow array by allocating a bigger one and copying the elements

if (queueSize == queue.length) {

Request[] aux = new Request[queue.length \* 2];

for(int i = 0; i < queueSize; i++) {

aux[i] = queue[i];

}

queue = aux;

}

// add new request

queue[queueSize] = new Request(nome, progressiveNumber);

queueSize++;

progressiveNumber++;

return progressiveNumber - 1;

}

public Request handleRequest() throws CounterException

{

if (queueSize == 0) {

throw new CounterException("error: queue is empty");

} else {

Request p = queue[0];

// shift array, eliminating queue[0]

for(int i = 0; i < queueSize - 1; i++) {

queue[i] = queue[i + 1];

}

queue[queueSize - 1] = null;

queueSize--;

// if necessary, shrink array

if (queueSize < queue.length / 4) {

int newDIM = (DIM > queueSize / 2)? DIM : queueSize / 2;

Request[] aux = new Request[newDIM];

for(int i = 0; i < queueSize; i++) {

aux[i] = queue[i];

}

queue = aux;

}

return p;

}

}

public void giveUp(int number) throws CounterException

{

// find element queue[k] having queue[k].getNumber() == number

int k = 0;

while (k < queueSize && queue[k].getNumber() != number) {

k++;

}

if (k < queueSize && queue[k].getNumber() == number) {

// shift all elements starting from the k-th element one position down

for(int i = k; i < queueSize - 1; i++) {

queue[i] = queue[i + 1];

}

queue[queueSize - 1] = null;

queueSize--;

// if necessary, shrink array

if (queueSize < queue.length / 4) {

int newDIM = (DIM > queueSize / 2)? DIM : queueSize / 2;

Request[] aux = new Request[newDIM];

for(int i = 0; i < queueSize; i++) {

aux[i] = queue[i];

}

queue = aux;

}

} else {

throw new CounterException("error: progressive number not found");

}

}

public int waitingTime(int number) throws CounterException

{

// find element queue[k] having queue[k].number == number

int k = 0;

while (k < queueSize && queue[k].getNumber() != number) {

k++;

}

if (k < queueSize && queue[k].getNumber() == number) {

return k;

} else {

throw new CounterException("error: progressive number not found");

}

}

public String toString()

{

String ris = "";

for (int i = 0; i < queueSize; i++)

ris = ris + queue[i] + "\n";

return ris;

}

}

public class CounterException extends RuntimeException

{

public CounterException (String msg) {

super(msg);

}

}

Implement a client class for Counter that acts as follows:

1. create an initially empty Counter;
2. have the user input the name of a text file containing person's names (one name for each line); for each person add a request to the counter's queue;
3. ask the user repeatedly which of the following actions should be taken and perform them:
   * handle the next request from the queue;
   * give up a request (progressive number read from user input);
   * add a new request (person's name read from user input);
   * print the number of requests left (waiting time) before the given request (progressive number read from user input);
   * quit the program;

after each operation print out the queue.

The client should catch the exceptions thrown by Counter printing a warning message, but without terminating the program.

Here is an example input file (it contains the italian names of Disney comic book characters): disney.txt

**Note:** to open a file specify its **absolute path** (for example C:\java-projects\lab-11\disney.txt) since BlueJ does not set the current directory path.

disney.txt

Paperino

Paperina

Zio Paperone

Paperoga

Paperinik

Qui

Quo

Qua

Brigitta

Nonna Papera

Clarabella

Gastone

Archimede Pitagorico

Eta Beta

Pippo

Topolino

Minni

Super Pippo

Basettoni

Bassotto 0

Bassotto 1

Bassotto 2

Bassotto 3

Bassotto 4

Bassotto 5

Bassotto 6

Bassotto 7

Bassotto 8

Bassotto 9

Pluto

import java.io.\*;

public class Client

{

private static final String menu =

"Choose an action:\n" +

" n: handle next request\n" +

" r: give up a request\n" +

" a: add a new request\n" +

" t: print number of requests left before a given request\n" +

" l: print list of requests\n" +

" q: quit\n";

public static void main (String[] args) throws IOException

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Insert filename (absolute path): ");

String namefile = br.readLine();

BufferedReader brf = new BufferedReader(new FileReader(namefile));

Counter sp = new Counter();

String name = brf.readLine();

while (name != null) {

sp.addRequest(name);

name = brf.readLine();

}

brf.close();

System.out.println(sp);

Request p;

System.out.println(menu);

char choice = br.readLine().charAt(0);

while (choice != 'q') {

try {

switch (choice) {

case 'n':

System.out.println(sp.handleRequest());

break;

case 'r':

System.out.print("give up which request (progressive number)? ");

sp.giveUp(Integer.parseInt(br.readLine()));

break;

case 'a':

System.out.print("name of person? ");

name = br.readLine();

System.out.println(name + "'s request has been added with progressive number " + sp.addRequest(name));

break;

case 't':

System.out.print("show waiting time for which request (progressive number)? ");

System.out.println("waiting time: " + sp.waitingTime(Integer.parseInt(br.readLine())));

break;

case 'l':

System.out.println("Current queue:\n" + sp);

break;

} // switch (choice)

}

catch (CounterException e) {

System.out.println(e.getMessage());

}

catch (NumberFormatException e) {

System.out.println("error: integer wanted");

}

System.out.println(menu);

choice = br.readLine().charAt(0);

} // while (choice != 'q')

}

}

30. Implement a class Tape. An instance of Tape represents a video tape that can hold recorded television programs. For each program recorded on a tape, the following properties are of interest:

* the name of the recorded program (a string);
* the length of recorded program in minutes (an integer).

Each tape has a certain capacity (in minutes) and each recorded program starts at a certain position on the tape. The positions are numbered as 0, 1, 2, 3, ... The class Tape should implement the following features:

* the constructor:  
  given an integer parameter *k*, the constructor creates an initially empty tape with a capacity of *k* minutes;
* the method getResidualCapacity():  
  returns the residual capacity of the tape in minutes;
* the method getNumberOfPrograms():  
  returns the number of recorded programs;
* the method record():  
  given as parameters the name and the length of a program, record() appends the program to the tape - if there's not enough space left to contain the program a RuntimeException should be thrown;
* the method getProgram():  
  given the position *n* on the tape, getProgram() returns the program recorded at that position - if there is no program recorded at that position a RuntimeException should be thrown;
* the method getLength():  
  given the position *n* on the tape, getLength() returns the length in minutes of the program recorded at that position - if there is no program recorded at that position a RuntimeException should be thrown;

Implement a class Tape using a linked list to store informations about the programs on the tape.

**representation of the objects**

class Node {

String name;

int length;

Node next;

}

public class Tape {

// representation of the objects

private int residualCapacity; // space left on time in minutes

private int numberOfPrograms; // how many programs are stored on the tape

private Node programListing; // reference to linked list of nodes

// public methods

...

}

**skeleton of the class**

class Node {

String name;

int length;

Node next;

}

public class Tape {

// representation of the objects

private int residualCapacity; // space left on time in minutes

private int numberOfPrograms; // how many programs are stored on the tape

private Node programListing; // reference to linked list of nodes

// public methods

public Tape (int k) { }

public int getResidualCapacity() { }

public int getNumberOfPrograms() { }

public void record(String name, int length) { }

public String getProgram(int n) { }

public int getLength(int n) { }

}

**Tape.java**

class Node {

String name;

int length;

Node next;

}

public class Tape {

// representation of the objects

private int residualCapacity; // space left on time in minutes

private int numberOfPrograms; // how many programs are stored on the tape

private Node programListing; // reference to linked list of nodes

// public methods

public Tape (int k) {

residualCapacity = k;

numberOfPrograms = 0;

programListing = null;

}

public int getResidualCapacity() {

return residualCapacity;

}

public int getNumberOfPrograms() {

return numberOfPrograms;

}

public void record(String name, int length) {

if (residualCapacity < length) {

throw new RuntimeException(

"cannot record: not enough space left on tape");

} else {

// insert program to top of linked list

// (we store the programs in inverse order for simplicity)

Node aux = new Node();

aux.name = name;

aux.length = length;

aux.next = programListing;

programListing = aux;

numberOfPrograms++;

residualCapacity = residualCapacity - length;

}

}

public String getProgram(int n) {

return findNode(programListing, numberOfPrograms - 1 - n).name;

}

public int getLength(int n) {

return findNode(programListing, numberOfPrograms - 1 - n).length;

}

// helper method

private static Node findNode(Node lis, int pos) {

if (pos < 0 || lis == null)

throw new RuntimeException("could not find program");

else if (pos == 0)

return lis;

else

return findNode(lis.next, pos - 1);

}

}

Implement a client class for Tape. It should have a static method that, given a Tape instance and a file name as parameters, writes the program listing (name and length of each recorded program) to the file. The static method should also write the total length of all recorded programs and the residual capacity of the tape to the same file.

import java.io.\*;

public class TapeClient {

public static void writeListing(Tape t, String filename) throws IOException {

PrintWriter out = new PrintWriter(new FileWriter(filename));

int totalLength = 0;

for (int i = 0; i < t.getNumberOfPrograms(); i++) {

int length = t.getLength(i);

totalLength = totalLength + length;

out.println(i + ": " + t.getProgram(i) + " " + length);

}

out.println("Total length: " + totalLength);

out.println("Residual capacity: " + t.getResidualCapacity());

out.close();

}

}

Implement a test program that creates a Tape, records a few programs on it and then calls the static method from part 2 to write the tape's information to a file.

import java.io.\*;

public class Main {

public static void main (String[] args) throws IOException {

Tape t = new Tape(30);

t.record("Goofy", 5);

t.record("Pluto", 10);

t.record("Donald", 6);

t.record("Mickey", 6);

TapeClient.writeListing(t, "cartoon.txt");

}

}

/\*

After this program is run, the file "cartoon.txt" should contain:

0: Goofy 5

1: Pluto 10

2: Donald 6

3: Mickey 6

Total length: 27

Residual capacity: 3

\*/

31.

32.

33.