Subject Code: CSE1IOO/CSE4IOO

Subject Name: Intermediate Object-Oriented Programming

Total marks: 120

Reading time: 15 mins

Writing time: 240 mins (4 hours)

Exam start time: 2:00 PM

Exam end time: 6:15 PM

**Question 1**

*GradeHomework.java* Class:

**public** **class** GradeHomework **extends** Homework{

**private** **int** weight;

GradeHomework(**int** number, String topic, **int** weight) {

**super**(number, topic);

**this**.weight = weight;

}

GradeHomework(**int** number, String topic) {

**super**(number, topic);

**this**.weight = 0;

}

**public** **int** getWeight() {

**return** **this**.weight;

}

**public** String getDetails() {

**return** "ID: " + **this**.getNumber() +", Name: " + **this**.getTopic() +", Weight: " + **this**.getWeight();

}

}

**Question 2**

*Book.java* Class:

**public** **class** Book {

**private** String BookID;

**private** **double** price;

Book(String BookID, **double** price) **throws** Exception{

**if**(BookID.length() >= 6) {

**this**.BookID = BookID;

}**else** {

**throw** **new** InvalidBookIDException("Book ID must be at least six characters");

}

**if**(price >= 0) {

**this**.price = price;

}**else** {

**throw** **new** InvalidBookPriceException("Book price must be a positive double value");

}

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

}

**Question 3**

The Complete Method:

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

Scanner infile = **new** Scanner(**new** File("persons.txt"));

**while**(infile.hasNext()) {

String line = infile.nextLine();

StringTokenizer tokenizer = **new** StringTokenizer(line, ";");

**if**(tokenizer.countTokens() == 3) {

String id = tokenizer.nextToken().trim();

String name = tokenizer.nextToken().trim();

String listOfHobbies = tokenizer.nextToken().trim();

String hobbies = " hobby";

**int** totalHobbies = listOfHobbies.split(", ").length;

**if**(totalHobbies > 1) {

hobbies = " hobbies";

}

System.***out***.println(name + " (" + id + ") has " + totalHobbies + hobbies) ;

}**else** {

System.***out***.println(line);

}

}

infile.close();

}

**Question 4**

A: Construct a File object to represent directory demo

File file = **new** File("../demo");

B: Write a code segment to display the names of all the ﬁles and subdirectories in directory demo.

**if**(file.exists() && file.isDirectory()) {

File[] files = file.listFiles();

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

System.***out***.println(files[i].getName());

}

}**else** {

System.***out***.println("folder directory doesnt exist ");

}

**Question 5**

1. Complete the recursive method:

**public** **static** **void** recursiveCountUp(**int** low, **int** high) {

**if**(low > high) **return**;

System.***out***.println(low);

*recursiveCountUp*(low + 1 , high);

}

(b) Complete the method shown below

**public** **static** **void** countUp(**int** n) {

*recursiveCountUp*(1 , n);

}

**Question 6**

**public** **static** numberObject maximumObject(numberObject num1, numberObject num2) {

**int** maxNumber = num1.compareTo(num2);

numberObject num\_obj = **new** numberObject(0);

**if**(maxNumber == num1.value) {

**return** num1;

}

**else** **if**(maxNumber == num2.value) {

**return** num2;

}

**return** num\_obj;

}

**public** **static** numberObject maximumObject(numberObject num1, numberObject num2, numberObject num3) {

numberObject num\_obj = **new** numberObject(0);

numberObject max = *maximumObject*(num1, num2);

**int** maxNumber = max.compareTo(num3);

**if**(maxNumber == max.value) {

**return** max;

}

**else** **if**(maxNumber == num3.value) {

**return** num3;

}

**return** num\_obj;

}

**public** **static** **class** numberObject **implements** Comparable<numberObject> {

**int** value;

numberObject(**int** value){

**this**.value = value;

}

**public** **int** getValue() {

**return** **this**.value;

}

@Override

**public** **int** compareTo(numberObject number) {

**if** (**this**.value >= number.value) {

**return** **this**.value;

} **else** {

**return** number.value;

}

}

}

**Question 7**

1. Insurance class:

**class** Insurance **implements** Commissioned{

String id;

**double** premium;

**double** commissionRate;

**public** Insurance(String id, **double** premium, **double** commissionRate){

**this**.id = id;

**this**.premium = premium;

**this**.commissionRate = commissionRate;

}

**public** **double** getCommission() {

**return** commissionRate \* premium;

}

}

1. Get insurance commission total:

**public** **static** **void** insuranceCommisionTotal(ArrayList<Commissioned> commisions) {

**double** totalCommission = 0.0;

**for**(Commissioned commission: commisions) {

totalCommission += commission.getCommission();

};

System.***out***.println(totalCommission);

}

1. To re-use this above sortByString() method to sort a collection of Insurance objects:

**class** Insurance **implements** Commissioned, StringKeyed

@Override

**public** String getStringKey() {

**return** id;

}