Final Examination: Semester 1, 2020, La Trobe University

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

Total pages: 15 (including this page)

There are a total of 7 questions. You are required to answer all of them. Write your answers on a word document and upload them using the designated link on LMS before the above time expires.

Question 1 (18 marks)

Consider the class below:

```
public class Homework
{
   private int number;
   private String topic;

   public Homework(int number, String topic)
   {
      this.number = number;
      this.topic = topic;
   }

   public int getNumber()
   {
      return number;
   }

   public String getTopic()
   {
      return topic;
   }

   public String getDetails()
   {
      return "number: " + number +", topic: " + topic;
   }

   public String toString()
   {
      return getClass().getName() + "[" + getDetails() + "]";
   }
}
```

Define a class `GradeHomework`, a subclass of `Homework`, which has an additional attribute called `weight` of type int (which is the percentage the grade homework contributes toward the final mark).

Include the following constructors and methods:

- A constructor with signature `GradeHomework(int number, String topic, int weight)`
- A constructor with signature `GradeHomework(int number, String topic)`
  This constructor creates a `GradeHomework` instance where `weight` is set to 0 (temporarily). This constructor must use the `this` statement.
- Method to get the weight.
- Method `getDetails` which returns a string showing the grade homework details: id, name and weight. For each attribute, the string shows the attribute name and attribute value, separated by a colon. The attributes are separated by commas.

Question 2 (15 marks)

In writing a piece of software that records information about the books sold by a company, we need to ensure that the book IDs and prices are valid.

A book ID is valid if it is a String with at least 6 characters. A book price is valid if it is a positive `double` value.

Suppose we have defined two exception classes

- `InvalidBookIDException`
- `InvalidBookPriceException`,

both of which are *checked* exceptions and are defined as follows:

```
public class InvalidBookIDException extends Exception
{
   public InvalidBookIDException ()
   {
      super();
   }
   public InvalidBookIDException(String message)
   {
      super(message);
   }
}
```

```
public class InvalidBookPriceException extends Exception
{
   public InvalidBookPriceException ()
   {
      super();
   }
   public InvalidBookPriceException(String message)
   {
      super(message);
   }
}
```

Write the Java code for the class `Book`. You are required to include only:

- The attributes (to store the book ID and the price),
- A constructor (to initialize the attributes), and
- A method to set the price (this method takes a price as argument and sets the book's price to this value).

The constructor must check that the book ID and the price are valid, and it must throw the appropriate exception if either of them is invalid. Similarly, the set method should also check if the price is valid (i.e., it is positive) and throw the appropriate exception when necessary.

Question 3 (18 marks)

Suppose we have a text file named `persons.txt` whose first 4 lines are shown below:

```
P01; Jane Smith; drawing, playing piano
P02; Tom Brown; gardening
P03; Bob Carter; drinking, dancing, running
P04; Frank Dawson; playing electronic games, soccer
...
```

The file has the general format:

```
<id> ; <name> ; <hobbies separated by commas>
```

Consider the incomplete method below:

```
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, ";");
      String id = tokenizer.nextToken().trim();
      String name = tokenizer.nextToken().trim();
      String listOfHobbies = tokenizer.nextToken().trim();

      // TO DO
      /*You can also use the split() method of the String class
      for the rest of the program if you want*/
   }

   infile.close();
}
```

Do not worry about any imports and it is safe to assume a person will have at least one hobby.

Complete the method (only write from TO DO) so that it reads the text file and displays on the screen the persons' details as shown below (for the first 4 persons):

```
Jane Smith (P01) has 2 hobbies
Tom Brown (P02) has 1 hobby
Bob Carter (P03) has 3 hobbies
Frank Dawson (P04) has 2 hobbies
...
```

That is, for each person, the method displays: The name of the person, followed by the ID in brackets, followed by the message:

- 'has ⟨n⟩ hobby' (if the value of n is 1), or,
- 'has ⟨n⟩ hobbies' (if the value of n is greater than 1)

Question 4 (15 marks)

Given the directory structure below, where the dots `(...)` that appears at a place means that we can have zero or more files or directories at that place:

```
mydir
      demo
            lab1.txt
            lab2.txt
            sample_programs
                  Demo1.java
                  Demo2.java
            ...

      programs
            ProcessFiles.java
            ...
```

Suppose you are working with class `ProcessFiles.java` in directory `programs`.

(a) Construct a `File` object to represent directory `demo`.

(3 marks)

(b) Write a code segment to display the names of all the files and subdirectories in directory `demo`.

Only the direct children of the directory `demo` are required to be listed.

(12 marks)

Question 5 (16 marks)

(a) Complete the *recursive method* shown below.

```
public static void recursiveCountUp(int low, int high)
// 1 <= low and low <= high
{
   ...
}
```

The method displays numbers from `low` to `high` on the screen. For example, the method call

```
recursiveCountUp(2, 4)
```

will display

```
2
3
4
```

Any non-recursive implementation will receive a mark of zero.

[12 marks]

(b) Complete the method shown below.

```
public static void countUp(int n) // n >= 1
{
   ...
}
```

The method displays numbers from 1 to `n` on the screen. For example, the method call

```
countUp(4)
```

will display

```
1
2
3
4
```

Hint: Use the method in part (a).

[4 marks]

Question 6 (16 marks)

Consider the methods below:

```
public static int maximum(int n1, int n2)
{
   if (n1 >= n2)
   {
      return n1;
   }
   else
   {
      return n2;
   }
}

public static int maximum(int n1, int n2, int n3)
{
   int max = maximum(n1, n2);
   max = maximum(max, n3);
   return max;
}
```

The two methods above calculate the maximum of two and three integers respectively.

Write two generic methods:

- one to determine the maximum of two objects and
- one to determine the maximum of three objects.

The objects are of the same type which implements the Java `Comparable` interface.

Question 7 (22 marks)

Consider the Interface below:

```
public interface Commissioned
{
   double getCommission();
}
```

a) Write a Java class named `Insurance` that implements the `Commissioned` interface.

Give the `Insurance` class the following attributes:

- `id` (a String): an ID number to identify the insurance that is sold
- `premium` (a double): the amount of money paid by the customer for the insurance coverage
- `commissionRate` (a double): the rate of commission.

Give the `Insurance` class the following methods:

- Constructor that takes three parameters, one for each attribute.
- The `getCommission` method that returns the commission on the sale of the insurance. The commission is calculated as,

```
commissionRate * premium
```

(7 marks)

b) There can be other classes such as Car, House, etc. that implements the Commissioned interface. Now, consider the method with the heading

```
public static void insuranceCommisionTotal(ArrayList<Commissioned>
    commisions)
```

Define the method so that it displays on the screen the total commission value for all the insurances in the list. The list can contain objects other than Insurance objects.

(8 marks)

c) Consider the following StringKeyed Interface:

```
public interface StringKeyed
{
   public abstract String getStringKey(); // note the String return type
}
```

Now, consider the following sorter class that sorts a collection of type `StringKeyed`.

```
public class Sorter
{
   public static void sortByString(StringKeyed[] collection)
   {
      for (int i = collection.length - 1; i > 0; --i)
      {
         int indexOfLargest = 0;
         for (int j = 1; j <= i; ++j)
         {
            if (collection[j].getStringKey().compareTo(collection[
                indexOfLargest].getStringKey())>0)
            {
               indexOfLargest = j;
            }
         }
         StringKeyed temp = collection[i];
         collection[i] = collection[indexOfLargest];
         collection[indexOfLargest] = temp;
      }
   }
}
```

If we want to re-use this above `sortByString()` method to sort a collection of `Insurance` objects (based on their ids), what changes do we have to make in the `Insurance` class?

You do NOT need to re-write the entire `Insurance` class. Just write the code that needs to be added to the class.

(7 marks)

Appendix – Selected Methods Reference (you might not need all of them for this exam)

PrintWriter

`PrintWriter(File)`	`PrintWriter outfile = new PrintWriter(`
	`  new File‘‘Test.txt’’));`
	Can throw a `FileNotFoundException`

`PrintWriter(FileWriter)`	`PrintWriter outfile = new PrintWriter(`
	`  new FileWriter("Test.txt", true));`
	‘true’ means appending new text to existing text
	Can throw a `IOException`

`PrintWriter(PrintStream)`	`PrintWriter out = new PrintWriter(`
	`  System.out);`
	To output to the screen

`print()`	Prints the argument
`println()`	Prints the argument, if any, then moves to the next line
`printf()`	Prints the arguments according to the format specifier
`close()`	Closes the writer

printf

Formatting strings:

`% [alignment] [min width] [.max width] s`

Alignment: minus sign means left alignment; the default is right alignment

If the string is longer than max-width, only the first max-width characters are displayed.

Formatting integers:

`% [alignment] [group separation] [min width] d`

Alignment: same as for strings

Group separation: a comma means that thousand groups are separated by commas

Formatting real numbers:

`% [alignment] [group separation] [min width][.decimal points] f`

Alignment: same as for strings

Group separation: same as for integers

Decimal points: For display, the number is rounded off if necessary to fit the number of decimal points

Scanner

`Scanner(InputStream)`	Often used with System.in to read from the keyboard: new Scanner(System.in)
`Scanner(File)`	Create a Scanner object to read from a text file Throws FileNotFoundException
`nextLine()`	Reads until the end of the line. Consumes the end-of-line character. Can throw various unchecked exceptions
`nextInt()`	Reads the next int. Does not consume delimiter character after the number
`nextDouble()`	Reads the next double
`nextBoolean()`	Reads the next boolean
`hasNext()`	Returns true if the scanner has another token
`hasNextLine()`	Returns true if the scanner has another line (or part of a line)
`hasNextInt()`	Returns true if the scanner has another int
`hasNextDouble()`	Returns true if the scanner has another double
`hasNextBoolean()`	Returns true if the scanner has another boolean
`close()`	Closes the scanner

BufferedReader

`BufferedReader(Reader)`	`BufferedReader in = new BufferedReader(`
	`new FileReader("sample.txt"));`
	Can throw `FileNotFoundException`

`readLine()`	Reads and returns the next line of text (as a `String`) Returns null if end of file has been reached
`read()`	Reads the next character and returns its numeric value Returns -1 if the end of file has been reached
`close()`	Closes the buffered reader

StringTokenizer

`StringTokenizer(String s)`	Creates a tokenizer for string s using whitespace characters as delimiters
`StringTokenizer(String s, String delimiters)`	Creates a tokenizer for string s using the characters in the second parameter as delimiters
`boolean hasMoreTokens()`	Returns true if there are remaining tokens, false otherwise
`int countTokens()`	Returns the number of remaining tokens The return value changes as we get tokens from the StringTokenizer object
`String nextToken()`	Returns the next token Throws NoSuchElementException if there are no more tokens
`String nextToken( String delimiters)`	Returns the next token using the characters in the parameter as delimiters Throws NoSuchElementException

Converting String tokens into other data types

`Integer.parseInt(String s)`	Returns the int value that is represented by the string s. Throws a NumberFormatException (unchecked) if the String argument cannot be converted to an int value
`Double.parseDouble(String s)`	Returns a double value
`Boolean.parseBoolean(String s)`	Returns a boolean value

The split method of String class

`String [] split(String regex)`	Takes a string argument which is treated as a regular expression, and splits the receiver string. Delimiters are strings that match the regular expression `s.split(''12'')` ⇒ delimiter is "12" `s.split(''1|2'')` ⇒ delimiters are "1" or "2" `s.split(''12|34'')` ⇒ delimiters are "12" or "34" `s.split(''\\s")` ⇒ delimiters are whitespace characters

File

`File(String pathName)`	Creates a File object with the specified pathname
`boolean exists()`	Returns true if there exists a file or directory with the associated pathname
`boolean isFile()`	Returns true if there exists a file with the associated pathname
`boolean isDirectory()`	Returns true if there exists a directory with the associated pathname
`File[] listFiles()`	Returns an array of File objects representing the files and directories in the directory
`String getName()`	Returns the simple filename
`String getPath()`	Returns the relative pathname
`String getAbsolutePath()`	Returns the absolute pathname
`long length()`	Returns the length of the associated file. The length of a directory is unspecified – usually it is 0
`boolean createNewFile()`	Creates a new empty file and returns true if (i) there is no file or directory with the same pathname, (ii) the path leading to the new file exists
`boolean mkdir()`	Creates a new empty directory and returns true if (i) there is no file/directory with the same pathname, (ii) the path leading to the new directory exists
`mkdirs`	Creates a new directory and returns true if there is no file/directory with the same pathname
`boolean delete()`	Deletes the denoted object and returns true if it is a file or an empty directory

ObjectOutputStream

`ObjectOutputStream(OutputStream out)`	Often used with `FileOutputStream(String filename)`
`void writeInt(int n)`	Can throw IOException
`void writeLong(long n)`	Can throw IOException
`void writeDouble(double x)`	Can throw IOException
`void writeFloat(float x)`	Can throw IOException
`void writeChar(int n)`	Can throw IOException
`void writeBoolean(boolean b)`	Can throw IOException
`void writeUTF(String aString)`	Can throw IOException (UTF stands for Unicode Transformation Format)
`void writeObject(Object anObject)`	throws IOException, NotSerializableException, InvalidClassException
`void close()`	Can throw IOException
`void flush()`	Can throw IOException. To clear the buffer

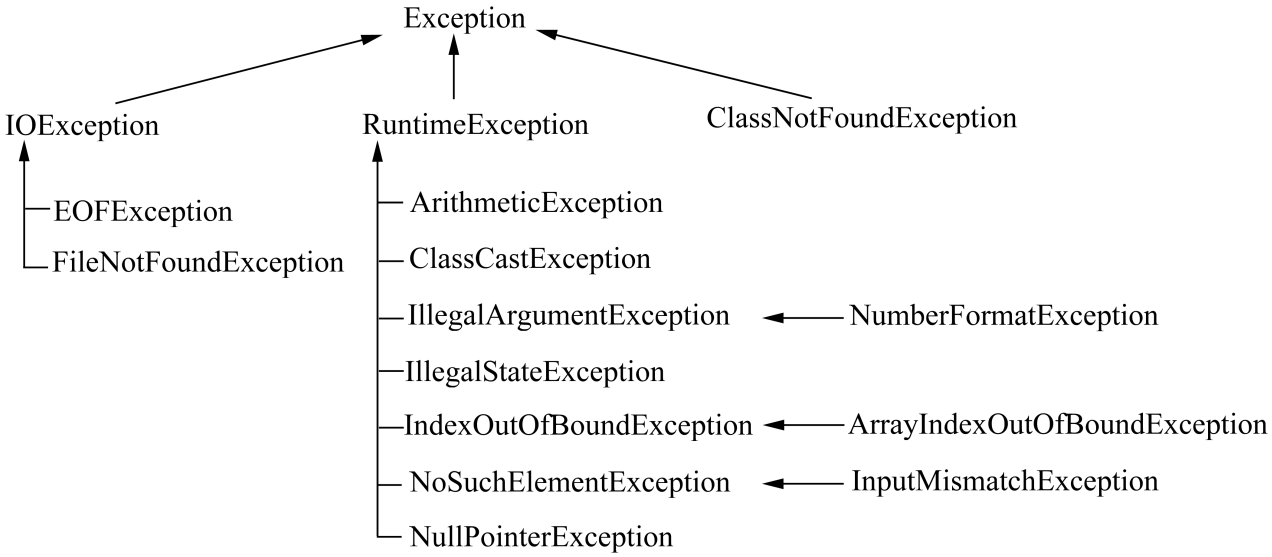
ObjectInputStream

`ObjectInputStream(InputStream in)`	Often used with `FileInputStream(String filename)`
`int readInt()`	Can throw IOException, EOFException
`long readLong()`	Can throw IOException, EOFException
`double readDouble()`	Can throw IOException, EOFException
`float readFloat()`	Can throw IOException, EOFException
`char readChar()`	Can throw IOException, EOFException
`boolean readBoolean()`	Can throw IOException, EOFException
`String readUTF()`	Can throw IOException, EOFException
`Object readObject()`	Can throw IOException, ClassNotFoundException, InvalidClassException, OptionalDataException, StreamCorruptedException
`void close()`	Can throw IOException
While reading a binary file, if a read method encounters the end of the file, it will throw an EOFException	

ArrayList (of Java Class Library)

`ArrayList()`	Creates a ArrayList object with no elements
`ArrayList(Collection< ? extends E> c)`	Creates a ArrayList object with the elements in the collection c. c is a collection of base type E or a subtype of E. Throws NullPointerException if the specified collection is null
`int size()`	Returns the number of elements in the list
`boolean isEmpty()`	Returns true if the size is 0
`boolean add (E e)`	Adds the element e to the end of the list
`void add(int index, E e)`	Adds the element e at the specified index. Throws IndexOutOfBoundsException if index is out of range
`E remove(int index)`	Retrieves and deletes the element at the specified index. Throws IndexOutOfBoundException if index is out of range
`E set(int index, E element)`	Replaces the element at index with the specified element. Returns the element previously at index. Throws IndexOutOfBoundException if index is out of range
`E get(int index)`	Retrieves the element at the specified index. Throws IndexOutOfBoundException if index is out of range
`void clear()`	Deletes all of the elements from the list
`boolean contains(Object o)`	Determines whether object o is in the list. Uses the `equals` method for comparison
`int indexOf(Object o)`	Returns the index where o first occurs in the list. (Returns -1 if object o is not found)
`int lastIndexOf(Object o)`	Returns the index where o last occurs in the list (Returns -1 if object o is not found)
`boolean remove(Object o)`	Removes the first occurrence of element o from the list. Returns true if the list has the specified element, false otherwise
`boolean addAll(Collection<? extends E> c)`	Adds each element from the collection c to the end of the ArrayList
`boolean removeAll(Collection<?> c)`	Deletes any element that is also in collection c
`boolean retainAll(Collection<?> c)`	Retains only the elements that are also in collection c

Exception Classes



■