CSE1IOO/CSE4IOO Sample Written Exam

Reading time: 15 minutes

Writing time: 120 minutes

Total marks: 120

Allowable Materials:

Number 18	Students from non-English speaking backgrounds can bring unmarked, non-electronic translation dictionaries into the examination

Question 1 (30 marks)

An experimental cheese farm has commissioned you to write a Java program to help it determine how much feed it needs in its barns over winter.

Each of the animals on the farm has an ID number and an experimental code which indicates which experimental program it is in (B for body fat, G for growth and M for milk production). These values are not changed.

There are two different types of animals on the farm: cows and sheep.

- The weight of a cow is recorded as a double (in kg).

  A cow in the body fat or growth program requires 2% of its body weight in feed per day. For example, if a cow is 500 kg it will require an amount of 10 kg of feed per day.

  A cow in the milk production program requires 5% of their body weight in feed per day.

- The age of a sheep in months is recorded as an integer.

  A sheep require 1 kg of feed a day when less than 8 months old, 1.5 kg of feed per day from 8 months to less than 16 months, and 2 kg of feed per day when 16 months and over.

Fully define the following classes in Java. Include all the necessary attributes and methods. Pay attention to whether any methods or classes should be abstract.

(a) Class `Animal` (the superclass of classes `Cow` and `Sheep`)

(8 marks)

(b) Class `Cow` (to represent a cow)

(6 marks)

(c) Class `Sheep` (to represent a sheep)

(6 marks)

(d) The details of the cows and sheep are stored in a text file called `CowsSheep.txt`. The details of each animal is written on one line. Examples of the records for a cow and a sheep is shown below:

```
34207 Cow G 620.5
34527 Sheep B 15
```

For a cow, the details are the ID, the word "Cow", the experimental code (B, G or M), and the weight (separated by space characters).

For a sheep, the details are the ID, the word "Sheep", the experimental code, and the age in months (separated by space characters).

Complete the method with the header shown below. The method reads the data file and returns an `ArrayList` of `Cow` and `Sheep` objects:

```
public static ArrayList<Animal> readData() throws Exception
```

(10 marks)

Question 2 (12 marks)

(a) Define a checked exception class called **`InvalidDataException`**.

(4 marks)

(b) Write a complete definition of the method with the heading given below.

```
public static double calculateInsurancePremium(
   double carValue, int driverAge) throws Exception
```

The method calculates the insurance premium for a car. It takes two parameters:

- **`carValue`**, which must be at least 10,000 (dollars)
- **`driverAge`**, which must be between 18 and 90, inclusive

The insurance premium is 5% of the value of the car if the driver is 21 years old or older. Otherwise, the premium is increased by 10% of the value of the car.

The method is required to throw an **`InvalidDataException`**, with appropriate error message, if the parameters are outside the valid range.

(8 marks)

Question 3 (14 marks)

Write a method with the declaration

```
public static void makeListing(String dirName)
```

where **`dirName`** is meant to be the name of a *directory*.

Take into account the following situations and related requirements:

- If **`dirName`** is the name of a normal file (not a directory), output an appropriate error message and terminate the method.
- If **`dirName`** is *not* the name of an existing directory, output an appropriate error message and terminate the method.
- Otherwise (i.e. the directory exists), write the *names of all the files and subdirectories* of the directory to a text file called **`listing.txt`**.

  The **`listing.txt`** file is to be stored in the current directory.

  If there is already an existing file named **`listing.txt`** in the current directory, overwrite it.

  You can assume that there is no subdirectory named **`listing.txt`** in the current directory.

Question 4 (24 marks)

(a) Write a Java interface named **`HazardRating`**. The interface has one method named **`getRating`** which takes no arguments and returns a double value. (The return value represents the rating of a hazard.)

(2 marks)

(b) Write a Java class, named **`Chemical`** that implements the **`HazardRating`** interface. The **`Chemical`** objects must also be able to be written to and read from a *binary file*.

Give the **`Chemical`** class the following attributes:

- **`temperature`** (an int): the normal storage temperature of the particular chemical
- **`volume`** (a double): the amount of chemical that is stored.

Give the **`Chemical`** class the following methods:

- a constructor that takes two parameters, one for each of the attributes
- a **`toString`** method that returns the name of the class and the values of the attributes, as a String
- the **`getRating`** method, which calculates the hazard rating by dividing the temperature by the volume.

(10 marks)

(c) Assume there is another class, **`Fuel`**, that also implements the **`HazardRating`** interface and that **`Fuel`** objects are able to be written to and read from a *binary file*.

Define the method with the heading

```
public static void displayChemicals(ArrayList<HazardRating> list)
```

which takes a list of **`Chemical`** and **`Fuel`** objects and displays all the **`Chemical`** objects in the list.

(6 marks)

(d) Define the method with the heading

```
public static void save(ArrayList<HazardRating>, ObjectOutputStream
    out)
```

which takes a list of **`Chemical`** and **`Fuel`** objects (the first argument) and write them to a binary file (the second argument).

(6 marks)

Question 5 (16 marks)

(a) Complete the recursive method with the heading given below. The method takes a string and returns true if it is a palindrome and false otherwise.

```
public static boolean isPalindrome(String s)
```

Hint: `s.substring(start, end)` returns the substring of `s` from index `start` to index `end - 1`, inclusive.

(10 marks)

(b) Using a diagram, trace the execution of the method call

```
isPalindrome("RADAR")
```

(6 marks)

Question 6 (14 marks)

SpriteLights is a new product that has just been developed by your company. It is a set of party lights that can be plugged into each other to form a line. You are asked to model the product in code as a linked list.

The class **Light** has already been written. It models a single light as a node in a linked list, and is defined as follows:

```
public class Light
{
   private String colour;
   private boolean lit;
   private Light nextLight;

   public Light(String c)
   {
      colour = c;
      lit = false;
      nextLight = null;
   }

   public void switchOn()
   {
      lit = true;
   }

   public void switchOff()
   {
      lit = false;
   }

   public String getColour()
   {
      return colour;
   }

   public boolean isLit()
   {
      return lit;
   }

   public Light getNextLight()
   {
      return nextLight;
   }

   public void setNextLight(Light light)
   {
      nextLight = light;
   }
}
```

A skeleton of class **SpriteLights**, which models a string of lights as a linked list is shown below:

```
public class SpriteLights
{
   private Light head;

   public SpriteLights()
   {
      head = null;
   }

   // Operations to be defined
}
```

(a) Define a method **addLight** which adds a new light at the *starting end* of the string of lights (i.e. at the start of the linked list).

This method takes a **Light** object as an argument.

(6 marks)

(b) Define a method **turnOnLightsOfColour** which takes a String argument which is the name of a colour, and turns on all lights in the string that are of that colour.

(8 marks)

Question 7 (10 marks)

Write a generic method with the header

```
public static <E extends Comparable<E>>
          ArrayList<E> getRange(List<E> list)
```

The method

- takes a list of objects, and
- returns an **`ArrayList`** that has two elements. The first element is the *smallest* object in the list and the second is the *largest*, according to the **`compareTo`** method.

Selected Methods Reference

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

Format specifier (to specify how a data item is to be displayed):

```
% [flags] [width] [.precision] conversion-character
```

Conversion-Characters:

d	decimal integer [byte, short, int, long]
f	floating-point number [float, double]
c	character. Capital C will uppercase the character
s	String. Capital S will uppercase all the letters in the string

Width and Precision:

`width`	Specifies the minimum number of characters to be written to the output
`precision`	Restricts the output field length depending on the conversion. For a real number, it specifies the number of decimal digits. For a string, it specifies the maximum length of the substring extracted for output

Flags:

`-`	left-justify (default is to right-justify)
`+`	output a plus (+) or minus (-) sign for a numerical value
`0`	force numerical values to be zero-padded (default is blank padding)
`,`	insert comma grouping separator (for numbers > 1000)
	space will display a minus sign if the number is negative or a space if it is positive

Scanner

`Scanner(InputStream)`	Often used with System.in to read from the keyboard (System.in is a BufferedInputStream object)
`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 Can throw `IOEXception`
`read()`	Reads the next character and returns its numeric value Returns -1 if the end of file has been reached Can throw `IOEXception`
`close()`	Can throw `IOEXception`

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 tokens are removed 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("[12]")` ⇒ delimiters are “1” or “2” `s.split("\\s")` ⇒ delimiters are whitespace characters

File

`File(String fileName)`	Creates a File object with the specified name. Should use a name permitted by the target system
`boolean exists()`	Returns true iff there exists a file or directory with the name associated with the file object
`boolean isFile()`	Returns true iff there exists a file with the same name
`boolean isDirectory()`	Returns true iff there exists a directory with the same name
`File[] listFiles()`	Returns an array of File objects representing the files and directories in the directory
`String getName()`	Returns the simple name associated with the File object (with no information about the path leading to it)
`String getPath()`	Returns the pathname associated with the File object. It is the pathname that was used to create the File object. This name is usually a 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 iff a file or directory with the same name does not exist. If a new file is created, returns true, otherwise returns false

`boolean mkdir()`
`boolean mkdirs()`
	Creates a new empty directory iff a file or a directory with the same name does not yet exist. `mkdir` requires that the parent directory exists, `mkdirs` does not. If a new directory is created, returns true, otherwise returns false

`boolean delete()`	Deletes the file or the directory iff the file exists or the directory exists and is empty. If a file or an empty directory is deleted, return true Returns false if (a) the file or directory does not exist, or (b) the directory is not empty

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

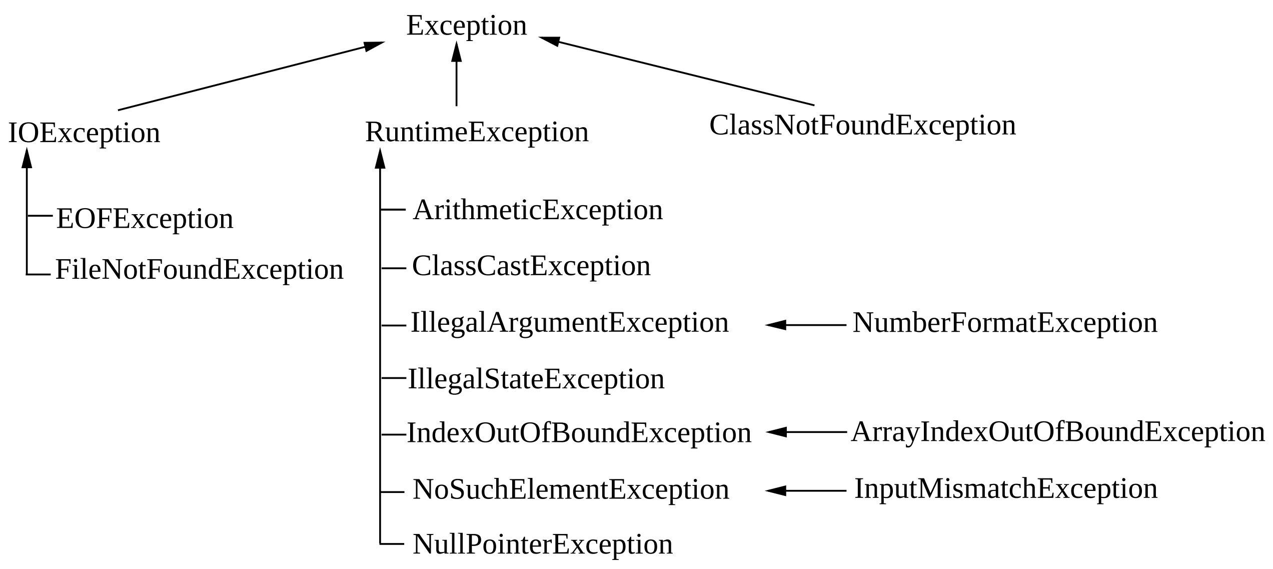
ObjectInputStream

`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



■