CSE1/4IOO Sample Exam

Semester 2, 2020

This is a sample exam only. This sample exam is exactly similar to the final exam in the following aspects:

- total number of main questions (6 questions)
- areas of concerns for each question (see question scope at the beginning of each question)
- total marks for each question

Moreover, this sample exam should be used as a revision material. Please note, that the actual questions in the final exam **will be different**. The number of sub-questions and the difficulty level of each question may also vary in the final exam.

Question 1 (28 marks)

- Question Scope: Inheritance (topics covered in lectures 1-5)

(a) Consider the class `Vehicle` below:

```
public class Vehicle
{
   private String regNr;      // registration number
   private int year;          // year of production

   Vehicle(String regNr, int year)
   {
      this.regNr = regNr;
      this.year = year;
   }
   public String getRegNr()
   {
      return regNr;
   }
   public int getYear()
   {
      return year;
   }
   public String getDetails()
   {
      return "regNr: " + regNr + ", year: " + year;
   }
   public String toString()
   {
      return getClass().getName() + "[" + getDetails() + "]";
   }
}
```

Define a class `Truck`, a subclass of `Vehicle`, which has an additional attribute called `tonnage` of type double (e.g. tonnage = 2.5 for a truck with tonnage capacity of 2.5 tonnes).

Include the following constructors and methods:

- A constructor with signature `Truck(String regNr, int year, double tonnage)`
- A constructor with signature `Truck(String regNr, int year)`
  This constructor creates a `Truck` instance where `tonnage` is set to `-1` (i.e. -1 stands for the missing value). This constructor must use the `this` statement.
- Methods to get and set the value of attribute `tonnage`.
- Method `getDetails`, which contains the attribute name and attribute value for each of the three attributes of a truck. This method must call the method `getDetails` in the superclass.

(b) Define the method with the header given below (you do not have to write any code for a class, just have to complete the method definition):

`public static double averageTonnage(ArrayList<Vehicle> vehicles)`

The list vehicles may contain instances of both Vehicle and Truck. The method returns the average tonnage capacity of the trucks in the list.

Question 2 (15 marks)

- Question Scope: Exceptions (topics covered in lectures 6-8)

(a) Does the following class successfully compile? Explain your answer.

```
public class MyClass
{
   public static void main(String [] args)
   {
      if(Integer.parseInt(args[0]) < 0)
      {
         throw new RuntimeException();
      }
   }
}
```

If the class does compile, describe what will happen when we run it with command:

```
java MyClass -10
```

(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 a *checked* exception if any of the parameters is outside the valid range.

Question 3 (25 marks)

- Question Scope: Text Files, Binary Files, Managing Files and Directories (topics covered in lectures 9-11)

(a) Suppose that in the current directory, there is a text file called `persons.txt`. This file contains the details of a number of persons. Each person's record occupies 4 lines:

- The given name (on the first line of the record, consisting of one word)
- The surname (on the second line, consisting of one word)
- The email (on the third line)
- A dot (on the fourth line, indicating the end of the record)

Write a program called `DisplayContacts.java`.

This program reads the file `persons.txt` and displays the details on the screen.

Each person is displayed on a line. The given name, surname and email are separated by forward slashes. For example:

```
John/Smith/jsmith.somewhere.au
```

You are not required to catch any exception. You can also ignore the import statements.

(b) 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
        ...

    lectures
        lecture1.txt
        lecture2.txt
            extras
            CheatSheet.txt
        ...

    programs
        ProcessFiles.java
        ...
```

Suppose you are working with class `ProcessFiles.java` in directory `lectures`. Write a code segment to check if a file or directory named `lecture3.txt` exists in directory `lectures`. If it does exist, display on the screen the message `''Lecture3.txt already exists.''`. Otherwise, create it as a text file, and write to it the message: `''This lecture will be updated soon.''`

Question 4 (16 marks)

- Question Scope: Recursion (topics covered in lectures 13-14)

(a) Define a recursive method with the header given below

```
public static int recursiveSum(int[] a, int start, int end)
```

The method calculates the sum of the elements of the array `a` from index `start` to index `end` inclusive.

Assume that `a` is a completely-filled array, `start` and `end` are valid indexes, and `start <= end`.

Your solution must be a recursive solution. Non-recursive solution will receive no mark.

(12 marks)

(b) Use the solution of part (a) to define the method with the header shown below

```
public static int sum(int[] a)
```

The method returns the sum of all the elements of the array. Assume that the array is a completely filled array.

(4 marks)

Question 5 (22 marks)

- Question Scope: Dynamic data structures: LinkedList, ArrayList (topics covered in lectures 17-18, 20)

(a) Consider the Node class below:

```
public class Node
{
   private String data;
   private Node next;

   public Node(String data)
   {
      this.data = data;
      this.next = null;
   }

   public String getData() { return data;}

   public Node getNext() {return next;}

   public String getDetails()
   {
      return "data: " + data;
   }
}
```

The NodeList class below maintains a LinkedList of Node objects.

```
public class NodeList
{
   private Node head = null;
   public void insertAtEnd(Node node)
   {
      //to do (i)
   }
   public void displayNodes()
   {
      //to do (ii)
   }
}
```

i Complete the implementation of the method `insertAtEnd` (signature must be same as the code above), which inserts the node at the end of the list.

ii Complete the implementation of the method `displayNodes` (signature must be same as the code above), which iterates through the list and displays the data at each node.

(b) Consider the main method below:

```
public class Tester
{
   public static void main(String[] args)
   {
      //to do
   }
}
```

Write a code segment (to replace the to do part above) to perform the following sequential tasks:

- Create an `ArrayList<String>` with the following names: `"Bob", "David","Edward", "Marry", "Jane"`.
- Add `"Tom"` to the end of the list.
- Add `"Ann"` to the start of the list.
- Print the index of `"Edward"`.
- Print the name of the sixth person.
- Remove `"Jane"`.
- Remove the third person.

Question 6 (14 marks)

- Question Scope: Interfaces, Generics (topics covered in lectures 15-16, 19, 21)

(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.

(b) Write a Java class, named `Chemical` that implements the `HazardRating` interface.

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:

- constructor that takes two parameters, one for each of the attributes
- the `getRating` method which calculates the hazard rating by dividing the temperature by the volume

(c) Complete the definition of the following generic method that returns the minimum value in a list:

```
public static <E extends Comparable<E>> E minimum(ArrayList<E> list)
{

}
```

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

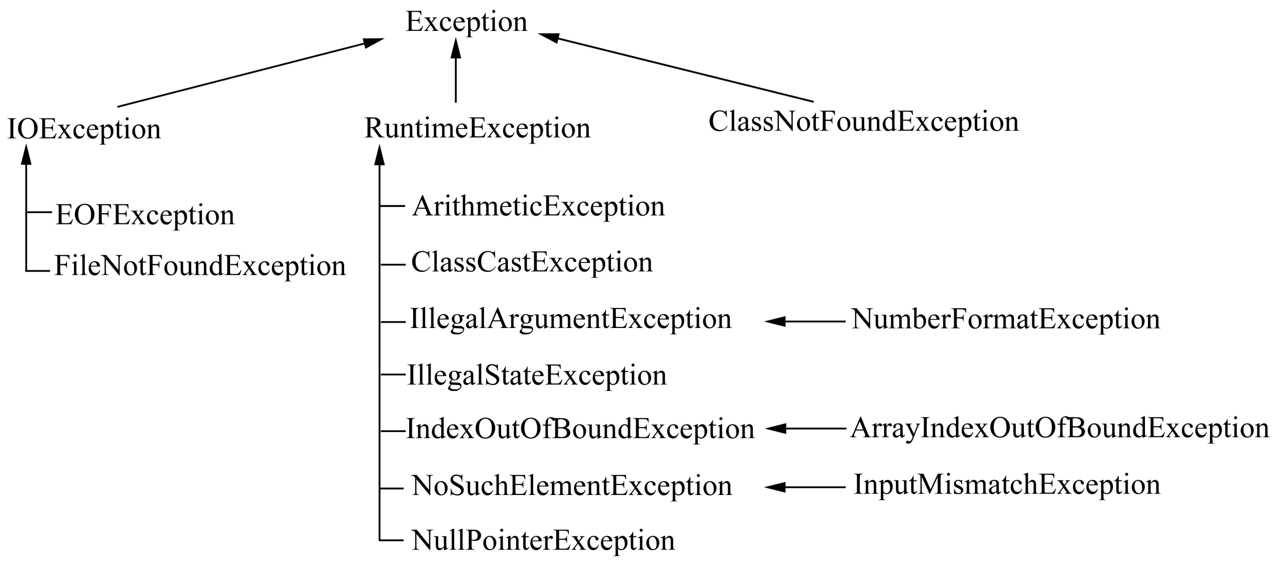
`ObjectInputStream(InputStream in)`	Often used with `FileInputStream(String filename)`
`int readInt()`	Can throw IOException, EOFException
`long readLong()`	Can throw IOException, EOFException
`double readDouble()`	Can throw IOException, EOFException
`float readFloat()`	Can throw IOException, EOFException
`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



■