Question 1: (2 marks)

Do not pay attention to real meaning of objects, variables and their values in the questions below.
Write a class **Trapezium** (*in the default package of the NetBean*) with the following information:

Trapezium
-firstEdge:float -secondEdge:float -height:float
+**Trapezium**() +**Trapezium**(firstEdge:float, secondEdge:float, height:float) +getAcreage():float +getInfo():String +setFirstEdge(fe:float):void +setSecondEdge(se:float):void

Where:

- **Trapezium**() - default constructor.
- **Trapezium**(firstEdge:float, secondEdge:float, height:float) - constructor, which sets values to first edge, second edge and height of the trapezium.
- **getAcreage**():float – Returns the area of the trapezium with 2 decimal places.
- **getInfo**(): String – Returns information about whether the trapezium is a "**Regular trapezium**" (*firstEdge is different secondEdge*) or "**Isosceles trapezium**" (*first edge equals to second edge*)
- **setFirstEdge**(fe:float): void – update value to firstEdge. Valid values are greater than 0**.** Other else, print out "**Invalid value**"
- **setSecondEdge**(se:float):void – update value to secondEdge. Valid values are greater than 0**.** Other else, print out "**Invalid value**"

Do not format the result.

The program output might look something like:

Enter first edge: 12.5 Enter second edge: 17 Enter height: 8.3 1. Test getAcreage () 2. Test getInfo() 3. Test setFirstEdge() 4. Test setSecondEdge() Enter TC (1, 2, 3 or 4): 1 OUTPUT: **122.43**	Enter first edge: 12.5 Enter second edge: 17 Enter height: 8.3 1. Test getAcreage () 2. Test getInfo() 3. Test setFirstEdge() 4. Test setSecondEdge() Enter TC (1, 2, 3 or 4): 2 OUTPUT: **Regular trapezium**
Enter first edge: 12.5 Enter second edge: 17 Enter height: 8.3 1. Test getAcreage () 2. Test getInfo() 3. Test setFirstEdge() 4. Test setSecondEdge() Enter TC (1, 2, 3 or 4): 3 0 OUTPUT: **Invalid value**	Enter first edge: 18 Enter second edge: 18 Enter height: 7 1. Test getAcreage () 2. Test getInfo() 3. Test setFirstEdge() 4. Test setSecondEdge() Enter TC (1, 2, 3 or 4): 2 OUTPUT: **Isosceles trapezium**

Question 2: (3 marks)

Define a **Student** class and a **StudentInIT** class extending from **Student** (i.e. **Student** is an abstract class and it is **superclass**, StudentInIT is a **subclass**) with the following information:

Student
-rollNumber:String -name:String -phone:String
+Student() +Student(rollNumber:String, name:String, phone:String) +getRollNumber():String +setRollNumber(color:String):void +getName():String +setName(name:String):void +getPhone():String +setPhone(name:String):void +mediumScore():double +classification():double

Where:

- **Shape**(): Default constructor to set color is black
- **Shape**(color:String): Constructor, which set value to color
- **getRollNumber**() – **getName**() – **getPhone**() are the getters of the student class.
- **setRollNumber**() – **setName**() – **setPhone**() – are the setters of the student class.
- mediumScore():double – an abstract method.
- classification():double – an abstract method.

StudentInIT
-basic, specialized, internship :double
+ **StudentInIT**() + **StudentInIT**(basic:double, specialized:double, internship:double) +getBasic():double +setBasic(b:double):void +getSpecialized():double +setSpecialized(s:double):void +getInternship():double +setInternship(i:double):void +mediumScore():double +classification():String

Where:

- **StudentInIT**() - default constructor to set all of the edges to 0
- **StudentInIT**(*basic*:double, *specialized*:double, *internship*:double) - constructor, which sets values to basic, specialized and internship subjects of the StudentInIT.
- **Setters** for subjects are needed to set values in range 0…10. In case the value is out of range, print out "**Incorrect**"
- **mediumScore**():double – Returns the calculated average of 3 subjects.
- **classification**():String – Returns a string value, evaluated against the following conventionally calculated average
  - 0…<5 : Weak student
  - 5…<7 : Average academic
  - 7…<8.5: Good academic
  - 8.5…10: Excellent Student

The program output might look something like:

Enter basic subject: 7 Enter specialized subject: 9 Enter internship subject: 9 1. Test mediumScore() 2. Test classification() 3. Test setBasic() 4. Test setSpecialized() 5. Test setInternship() Enter TC: 1	Enter basic subject: 7 Enter specialized subject: 9 Enter internship subject: 9 1. Test mediumScore() 2. Test classification() 3. Test setBasic() 4. Test setSpecialized() 5. Test setInternship() Enter TC: 2	Enter basic subject: 7 Enter specialized subject: 9 Enter internship subject: 9 1. Test mediumScore() 2. Test classification() 3. Test setBasic() 4. Test setSpecialized() 5. Test setInternship() Enter TC: 3

OUTPUT: 8.33	OUTPUT: **Good academic**	17 OUTPUT: **Incorrect**

Question 3 (3 marks)

Define a **Electronic** class with the following information:

Electronic
-code:String -name:String -brand:String -price:long
+**Electronic**() +**Electronic**(code:String, name:String, brand:String, price:long) +getCode():String +getName():String +getBrand():String +getPrice():long +setCode(code:String):void +setName (name:String):void +setBrand(brand:String):void +setPrice(price:long):void

Where:

- getCode():String – return code.
- getName():String – return name.
- getBrand():String – return brand.
- getPrice():long – return price.
- setCode(code:String): void – update code.
- setName(name:String): void – update name.
- setBrand(brand:String): void – update brand.
- setPrice(price:long): void – update price.

Create a **ListOfElectronic** class extending from **HashMap**<Key, Value> class in java collection. (*in which: Key: will be used to store code of electronic and Values store **Electronic** objects*).

ListOfElectronic
+toList():List< **Electronic**> +addElement(**Electronic** e):void +filterByPrice(long min, long max):List<Electronic> +countByBrand(String prefix):int

Where:

- ***toList***():List<Motorbike> – return a list of Motorbike objects.
- ***addElement***(Electronic e):void – add one Electronic object in the values list.
- ***filterByPrice***(long min, long mx): List<Electronic> – prints all of electronic objects which has price in range min and max value depend on arguments of this method
- ***countByBrand***(String prefix):int – count all electronic products of the brand with the filter condition based on the first characters of the brand's electronic.

When running, the program will add some data to the list. Sample output might look something like:

Added some Electronic info 1- Filter by Price 2- Count by Brand 1 Enter min range: 5000000	Added some Electronic info 1- Filter by Price 2- Count by Brand 2 Enter Brand name: Sony	Added some Electronic info 1- Filter by Price 2- Count by Brand 2 Enter Brand name: pana

Enter max range: 10000000 OUTPUT: Loa Bluetooth-Sony-8500000 Loa Bluetooth-Samsung-9200000	OUTPUT: 3	OUTPUT: 6

Question 4: (2 marks)

In this case, you are given a **Electronic** class and a **BusLogic** interface thus **you can use it without creating these files**.

Electronic	<<Interface>> **BusLogic**
-code:String -name:String -brand:String -price:long	***sortByCode***():void ***filterByName***(String prefix): List ***countByPriceRange***(long min, long max): int
+**Electronic**() +**Electronic**(code:String, name:String, brand:String, price:long) +**Getters** / **Setters**	

Define a class named **BusElectronic**, which implements the **BusLogic** interface:

BusElectronic
-items: List<Electronic>
+BusElectronic() +getItems():List<Electronic> +add(Electronic x):void +sortByCode():void +filterByName(String prefix): List<Electronic> +countByPriceRange(long min, long max):int

Where:

- **items**:List<Electronic> – to store list of Electronic objects.
- **getItems**():List<Electronic> - returns the items
- **BusElectronic**() – Initialize the items by ArrayList object.
- ***add***(Electronic x):void – add one Electronic object to items
- ***sortByCode***():void – Sort the list of items in ascending order for the code attribute of the Electronic object
- ***filterByName***(String prefix): List<Electronic> - Returns a list of Electronic objects with the filter condition based on the first characters of the electronic name
- ***countByPriceRange***(long min, long max): int Returns the value of the number of products in the requested price range.

You can change (*Add methods or some things*) into **Electronic** class, if it is needed

The program output might look something like:

Added some Electronic info 1- Filter by Name 2- Count by price range 3- Sort by code Enter TC (1,2 or 3): 1	Added some Electronic info 1- Filter by Name 2- Count by price range 3- Sort by code	Added some Electronic info 1- Filter by Name 2- Count by price range 3- Sort by code Enter TC (1,2 or 3): 3

Enter prefix name: tivi OUTPUT: PRD001-Tivi 65 inches-Sony-25000000 PRD005-Tivi 75 inches-Panasonic-37000000 PRD010-Tivi 55 inches-Sharp-12000000 PRD015-Tivi 85 inches-Samsung-52000000 PRD019-Tivi 75 inches-LG-32500000	Enter TC (1,2 or 3): 2 Enter min: 10000000 Enter max: 20000000 OUTPUT: 7	OUTPUT: PRD001-25000000 PRD002-17000000 PRD003-17000000 PRD004-12000000 PRD005-37000000 PRD006-8500000 PRD007-16000000 PRD008-38000000 PRD009-4500000 PRD010-12000000 PRD011-26000000 PRD012-3200000 PRD013-68000000 PRD014-4250000 PRD015-52000000 PRD016-11000000 PRD017-9200000 PRD018-27500000 PRD019-32500000 PRD020-18000000

PRO192 PE INSTRUCTIONS

Read the instructions below carefully before start coding.

Students are ONLY allowed to use:

- Materials on his/her computer (including JDK, NetBeans, Window explorer, Winrar, Winzip).
- For distance learning: Google Meet, Hangout (for Exam Monitoring Purpose).

Follow the steps below to complete PE:

1. Create a folder to save given projects, e.g. PRO_given (1). Down load given materials to (1).
2. Steps to do question 1 (do the same for other questions): Open NetBeans, open the given Q1 project, then complete it according to the requirements in the exam. (Do not: delete given files, or create java file with the same name as given files).
3. Before submission: Run the function **"Clean and Build Project"** (Shift+F11), then rename the folder dist to RUN (or run). (If the folder RUN already exists, delete it before renaming).
4. **Submission:** to submit the project Q1, at first you must select Question No = 1, browse and select the project folder (e.g. 1, Q1 or Q1X,...) then click the **Submit** button. Do the same for other questions. **Do not submit** the un-edited given project.
5. **Do not use accented Vietnamese** when writing comments in programs.
6. Software tools must be used: **NetBeans IDE** and **Java JDK 1.8**.
   If at least one of the above requirements is not followed, the exam will get ZERO.

Trouble shooting:

If the given project (e.g. Q1) runs with error, you need to run "Clean and Build Project" (Shift+F11). If still error, try to rename the project, e.g. from Q1 to Q1X or Q1Y,...

If the size of the project is too large for submission, try to delete the folder "build".
