BIRKBECK

(University of London)

BSc EXAMINATION FOR INTERNAL STUDENTS

Department of Computer Science
and Information Systems

Software and Programming III

BUCI056H6

CREDIT VALUE: 15 credits

DATE OF EXAMINATION: Monday, 24th May 2021
TIME OF EXAMINATION: 10:00 am
DURATION OF PAPER: Three hours

RUBRIC

1. Open-book and online examination.
2. Candidates should attempt ALL 10 questions on this paper.
3. The number of marks varies from question to question.
4. You should `add/commit/push` to your `GitHub Classroom` repository in the usual way.
5. Should you become "detached" from the Internet you can still `add/commit` locally and `push` at a later point.
6. Each question should be submitted as a file, or series of files (e.g., for diagrams, code fragments), within the given question folder (e.g., `Q01`).
7. You must answer each question using the Java programming language unless specified otherwise.
8. You are advised to look through the entire examination paper before getting started to plan your strategy.
9. Simplicity and clarity of expression in your answers is important.

Question:	1	2	3	4	5	6	7	8	9	10	Total
Marks:	12	13	10	8	6	5	10	12	6	18	100

Question 1 .... *12 marks*

What are the four main tenets of object-oriented programming?
Explain the advantages these features provide, illustrating your answer with appropriate JAVA language features.

Question 2 .... *13 marks*

This question refers to the JAVA Stream language feature (`java.util.stream`).

You are given the following (outline) declarations:

```
public record Employee(int id, String name, double salary){
    public double salaryIncrement(double number){ ... }
    public Employee findById(int id) { ... }
}

Employee[] arrayOfEmps = {
    new Employee(1, "Jeff Bezos", 100000.0),
    new Employee(2, "Bill Gates", 200000.0),
    new Employee(3, "Mark Zuckerberg", 300000.0)
};
```

and

```
List<Employee> empList = Arrays.asList(arrayOfEmps);
```

Using the functionality of `java.util.stream` answer the following:

(a) Write an expression which increments the salary for each employee in `empList` by `10.0`. [2 marks]

(b) Write an expression which converts a stream of `empIds`, which are of type `Integer`, into a stream of `Employees`. [3 marks]

(c) Write a method `filterEmployees` which takes a salary threshold and an array of employee ids, as its parameters. It first filters out `null` references for invalid employee ids and then, applies a filter to only keep employees with salaries over a certain threshold. The method should return the result as a `List<Employee>`. [5 marks]

(d) Suppose we have the following: [3 marks]

```
List<List<String>> namesNested = Arrays.asList(
    Arrays.asList("Jeff", "Bezos"),
    Arrays.asList("Bill", "Gates"),
    Arrays.asList("Mark", "Zuckerberg"));
```

which is a list of lists.

Write an expression which will enable the processing of the data as a single list.

Question 3 .... *10 marks*

Suppose `b` is:

- a `String` array, or
- an object of class `java.util.ArrayList<String>`, or
- `java.util.Set<String>`.

Then, one can write a foreach loop that processes each element of `b` as follows:

```
for (String s : b) {
    // Process s
}
```

With each iteration of the loop, another element of `b` is stored in `s` and then "processed" — whatever that means.

The use of such a *foreach* loop is made possible by the use of two interfaces:

- `Iterator`, and
- `Iterable`

which provide an implementation of the ITERATOR design pattern.

Write a class that implements the `Iterator` interface. An object of the class should be able to be used to enumerate the even values of an `Integer` array.

Question 4 . . . . *8 marks*

(a) How does dependency injection assist with *loose coupling*? Provide appropriate examples to illustrate your answer. 4 marks

(b) Using one sentence for each, state the two key limitations of constructors in Java. 4 marks

Question 5 . . . . *6 marks*

Consider the definition of the method `installDoor()`, written in Java-like pseudocode, from a `Contractor` class:

```
class Contractor {
   // any necessary instance variables
   // and methods defined here
  installDoor() {
    subcontractor = YellowPages.getSubcontractor();
    carpenter = subcontractor.getCarpenter();
    doorHandle = carpenter.getDoorHandle();
    doorBody = carpenter.getDoorBody();
    screws = carpenter.getScrews();
    door = carpenter.assemble(doorHandle, doorBody, screws);
    securityExpert = subcontractor.getSecurityExpert();
    securityExpert.installDoorSensors(door);
  }
}
```

What major design flaw is evident from the definition of the method?
Which design principle(s) are violated?
You may assume that any errors in method calls inside `installDoor()` are handled properly.

Question 6 . . . . *5 marks*

State, in words, what the following method does. Be precise enough that someone could use the method correctly.

```
import org.jetbrains.annotations.NotNull;

import java.util.List;

public class MysteryScript {
  public static Object mystery(@NotNull List list) {
    if (list.equals(List.of())) return "empty";
    if (list.size() == 1) return list.get(0);
    return mystery(list.subList(1, list.size()));
  }
}
```

Question 7 . . . . *10 marks*

(a) To implement the singleton pattern often (but not always) requires using what other pattern? Provide an appropriate example to support your answer. $\boxed{\text{4 marks}}$

(b) Consider a wrapper object whose implementation logs each call that is made to it. Briefly explain $\boxed{\text{6 marks}}$

- *when* and *why* the wrapper should be considered a *decorator*, and
- *when* and *why* that same wrapper should be considered a *proxy.*

Question 8 . . . . *12 marks*

Computer Science data structures known as *Trees* are naturally defined recursively. For example, we can define a binary tree as either

1. empty, or
2. a value together with a left binary tree and a right binary tree.

A more general tree can be defined as:

> "A tree is a value (the root value) together with a set of trees, called its children."

Such recursive definitions lend themselves naturally to recursive methods that process trees in some fashion.

(a) Assume that a node of a tree contains a value of type `int` and a set of children. The set is empty if the node is a leaf. $\boxed{\text{6 marks}}$

```
package recursion;

import java.util.Set;

public record Node(int value, Set<Node> children) {}
```

Write a method that returns the number of leaves of the tree.

The signature of the method should be:

```
    public int leaves(Node t)
```

(b) Write a method that returns a node of the tree with value `v`. If no node contains `v`, then the method returns `null`. $\boxed{\text{6 marks}}$

The signature of the method should be:

```
public Node getNode(Node t, int v)
```

Question 9 .... *6 marks*

An *Algebraic data type* is a composite type made from putting other types together.

Briefly describe two common classes of algebraic types. You should provide appropriate examples to illustrate your answer.

Question 10 .... *18 marks*

You are required to write an *undo-redo* mechanism for a simple system which manipulates "squares".

The program reads its data as successive lines from standard input. A line of input has one of the following forms, where `i`, `j`, and `k` are non-negative integers:

```
C i j
M i j k
S i j
U
R
P
```

Your program may assume that the input follows this syntax and that error handling is not required. Also, no graphical display is needed.

All information about squares will be output textually (see the `P` command).

- The `C` command (Create) creates square number `i` with side of length `j`, centred at the origin. It replaces any previous square with number `i`.
- The `M` command (Move) moves square number `i` by `j` pixels to the right and `k` pixels upwards. If there is no square numbered `i`, the command has no effect.
- The `S` command (Scale) scales square numbered `i` by a factor of `j`. If there is no square numbered `i`, the command has no effect.
- The `U` command (Undo) cancels the last not-yet-undone `C`, `M`, or `S` command. If none remains to be undone, `U` has no effect.
- The `R` command (Redo) is applicable only if the last executed command was `U` or `R`, otherwise it does nothing. It re-executes the most recent `C`, `M`, or `S` command undone and not yet redone.
- The `P` command (Print) prints on standard output details of all squares in the system. Squares are printed in ascending order with respect to number, one per line, where each line has the format

  ```
  i j k l
  ```

  where `i` is the square number, `j` its horizontal coordinate, `k` its vertical coordinate, and `l`, the length of its side.

  The values should be separated by single spaces.

You are provided with some outline code on the repository which is also reproduced here in Appendix A.

You should utilise the COMMAND design pattern in your solution.

Appendices

A Provided code

```
package command;

import java.util.Objects;

public class Square {
  private int number; // Square number
  private float sides; // Side length
  private float x; // Ordinate
  private float y; // Abscissa

  public Square(int number, int sides) {
    assert (number > 0);
    assert (sides > 0);
    this.number = number;
    this.sides = sides;
  }

  public float getX() {
    return x;
  }

  public void setX(float x) {
    this.x = x;
  }

  public float getY() {
    return y;
  }

  public void setY(float y) {
    this.y = y;
  }

  public int getNumber() {
    return number;
  }

  public void setNumber(int number) {
    this.number = number;
  }

  public float getSides() {
    return sides;
  }

  public void setSides(float sides) {
    this.sides = sides;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Square)) return false;
    Square square = (Square) o;
    return getNumber() == square.getNumber();
  }

  @Override
  public int hashCode() {
    return Objects.hash(getNumber());
  }

  // Move square by x horizontally and y vertically.
  public void move(float x, float y) {
    setX(getX() + x);
    setY(getY() + y);
  }

  // Scale current square by factor.
  public void scale(float factor) {
    setSides(getSides() * factor);
  }
}
```

and

```
package command;

public interface Command {
  boolean isDone(); // Has the current command been executed?

  void execute(); // Execute the current command

  void undo(); // Undo the current command on the square

  // Your code here?
}

// Modify and augment this code
class CreationCommand implements Command {
  @Override
  public boolean isDone() {
    // TODO
    return false;
  }

  @Override
  public void execute() {
    // TODO
  }

  @Override
  public void undo() {
    // TODO
  }
}
```

```
// Modify and augment this code
class MovementCommand implements Command {
  @Override
  public boolean isDone() {
    // TODO
    return false;
  }

  @Override
  public void execute() {
    // TODO
  }

  @Override
  public void undo() {
    // TODO
  }
}

// Modify and augment this code
class ScalingCommand implements Command {
  @Override
  public boolean isDone() {
    // TODO
    return false;
  }

  @Override
  public void execute() {
    // TODO
  }

  @Override
  public void undo() {
    // TODO
  }
}
```