



VICTORIA UNIVERSITY OF  
**WELLINGTON**  
 TE HERENGA WAKA

EXAMINATIONS – 2022

TRIMESTER 2

<p>SWEN 225</p> <p>SOFTWARE DESIGN</p>
--

**Time Allowed:** TWO HOURS

**CLOSED BOOK**

**Permitted materials:** No calculators permitted.  
 Non-electronic Foreign language to English dictionaries are allowed.

**Instructions:** Answer all questions

You may answer the questions in any order. Make sure you clearly identify the question you are answering.

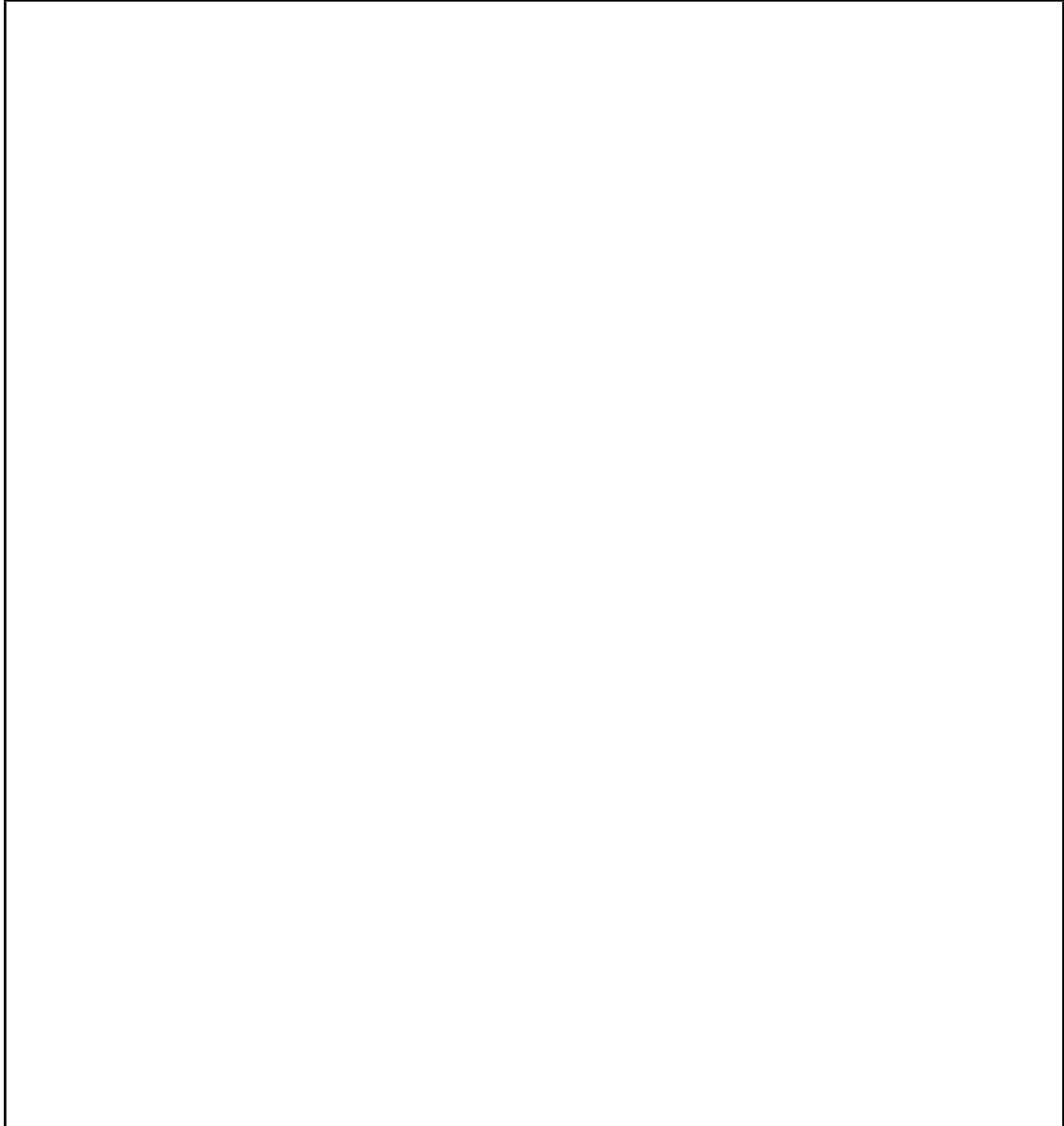
Question	Topic	Marks	Examiners Use Only
1.	UML and Git	30	<input type="text"/>
2.	The composite pattern	30	<input type="text"/>
3.	Code comprehension and Mocking	30	<input type="text"/>
4.	Contracts and invariants	30	<input type="text"/>
<b>Total</b>		120	

1. UML and Git

**(30 marks)**

**(a) Git (15 Marks)**

- i. **(3 marks)** What is the difference between Git, GitLab, GitHub, and Github Desktop?



- ii. **(3 marks)** What Git command is used when you want to download an existing git repository to your local computer? Write an example Git command.

- iii. **(3 marks)** What Git command is used for displaying the current status of your working directory? What Git command is used for switching current working directory to a specified branch? Write example Git commands.

- iv. **(3 marks)** What does the Git push command do and what is the purpose of the command? Write an example Git command.

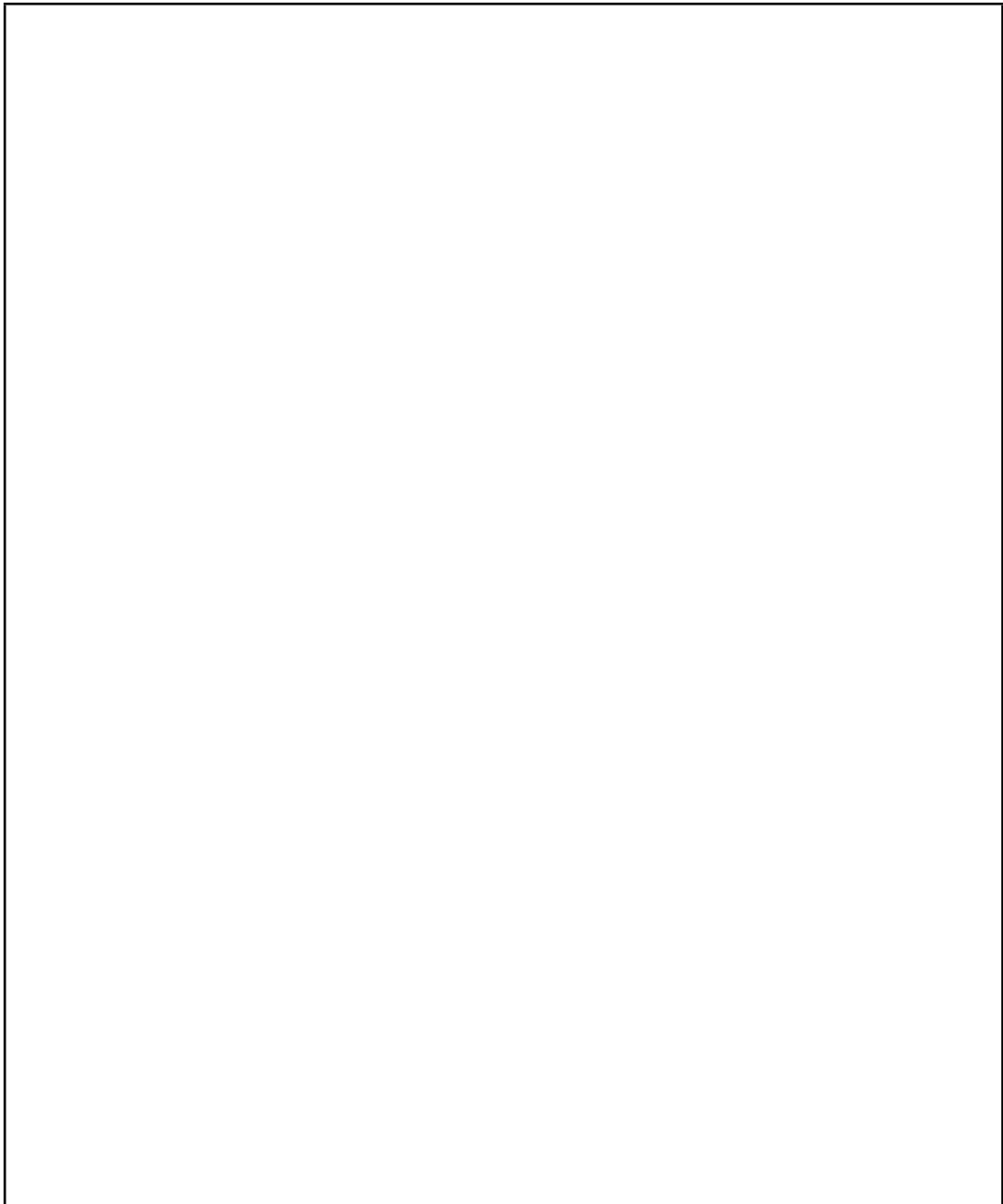
- v. **(3 marks)** What Git command is used to join a specified branch into your current branch (the one you are on currently)? Write an example Git command.

## (b) UML (15 Marks)

## i. (15 marks)

Uber is ride-sharing service. Uber has Drivers, Riders, Rides, and Vehicles. A driver can drive only one vehicle at a time. A vehicle can take up to four riders at a time. A ride can be classified as different kinds (e.g. UberX, Comfort). A rider takes rides in a vehicle and can take only one ride at a time. Riders can provide rating feedback on drivers on their service. Riders can make payments for rides via credit card only. Drivers can receive payments and also rate feedback on riders too.

Draw a UML class diagram using classes, associations, multiplicities, and inheritance that models the above Uber system. Make sure to use appropriate names for classes, association labels, and attribute names. Include attributes and operations where appropriate. For labelling associations use either labels for the whole association or role names at the association ends.



**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.

## 2. The composite pattern

**(30 marks)**

Follow closely the instruction below to implement a composite pattern for a sub set of html nodes. The nodes will offer the operation 'depth' in the way operations are implemented following the composite.

**Note:**

To get maximum marks you should avoid if/for/whiles and use streams when possible, but a correct answer using a traditional 'for' can still get most marks. To get maximum marks your code should be able to compile without warnings, but of course you can still get most marks if you miss minor details, like a ';'. However, mistakes about exceptions and generics are not minor details.

If you can not remember the name of a specific method, just make up a name and add a note describing the expected behaviour. You will not get full marks but you may get most marks.

**(a) (5 marks) Node**

Declare an interface 'Node' with the operation 'depth' returning an 'int'. Note: We will later add a static method to 'Node', but for now just declare the interface and the method 'depth'

```
interface Node{ int depth(); }
```

**(b) (5 marks) Paragraph**

Declare a record 'P' subtype of 'Node' so that the operation 'depth' returns '1'. The record will have a single field of type 'String'.

```
record P(String text) implements Node{  
    public int depth(){ return 1; } }
```

**(c) (5 marks) List item**

Declare a record 'Li' subtype of 'Node' and containing exactly one other 'Node' as a child. The operation 'depth' returns '1' + the depth of the child.

```
record Li(Node node)implements Node{  
  public int depth(){ return 1+node.depth(); } }
```

**(d) (5 marks) Unordered list**

Declare a record 'U1' subtype of 'Node' so that:

- the operation 'depth' returns '1' + the maximum depth of all the children.
- The record will have a single field to store a list of list items of type 'Li'; that is, only nodes of type 'Li' can be stored as children of an 'U1' node.

```
record U1(List<Li>lis) implements Node{  
  public int depth(){ return ns.stream()  
    .mapToInt(n->n.depth())  
    .max().orElse(0);  
  }  
}
```



**(e) (10 marks) Unordered list, Ordered list and Div**

Consider now the cases for 'Ol' and 'Div'. The children of an 'Ol' can only be nodes of type 'Li', while the children of a 'Div' can be any kind of node. That is, both 'Ol' and 'Div' have a single field, but the type will be different.

A naive implementation may repeat the code logic for computing the maximum depth three times: One in 'Ol', one in 'Ul' and one in 'Div'. Instead, we will edit the code of 'Node' to add a static method doing this task, and just call this functionality 3 times. Write the new complete code for 'Node', 'Ol', 'Ul' and 'Div'.

```
interface Node{
    int depth();
    static int maxDepth(List<? extends Node> ns){
        return ns.stream()
            .mapToInt(n->n.depth()).max().orElse(0);
    }
}
record Div(List<Node>ns) implements Node{
    public int depth(){ return 1+Node.maxDepth(ns); }
}
record Ul(List<Li>lis) implements Node{
    public int depth(){ return 1+Node.maxDepth(lis); }
}
record Ol(List<Li>lis) implements Node{
    public int depth(){ return 1+Node.maxDepth(lis); }
}
```

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.

## 3. Code comprehension and Mocking

**(30 marks)**

Consider the following incomplete code, where the omitted method bodies of 'ConcreteDB' are correctly accessing a real Data base. Note: to answer this question no former knowledge of Data bases is required.

**(a) (10 marks) Complete the code**

Similarly as in the WAT questions, on the next page, write the missing part ([??]) of the code so that it can compile and run without errors.

```

1  class ConcreteDB implements DBAccess{
2      public void beginTransaction() {/*..*/}
3      public void commitTransaction() {/*..*/}
4      public void rollbackTransaction() {/*..*/}
5      public void doQuery(String query) {/*..*/}
6  }
7  class TransactionManager{
8      private final DBAccess db;
9      private final Query q;
10     public TransactionManager(DBAccess db){
11         this.db = db;
12         this.q = str->db.doQuery(str);
13     }
14     public void makeTransaction(DoQueries qs){
15         db.beginTransaction();
16         try{ qs.of(q); }
17         catch(Error|RuntimeException err){
18             db.rollbackTransaction();
19             throw err;
20         }
21         db.commitTransaction();
22     }
23 }
24 [??]
25 class User{
26     public static void main(String[]arg) {
27         var t = new TransactionManager(new ConcreteDB());
28         runProgram(t);
29     }
30     static void runProgram(TransactionManager t){
31         t.makeTransaction(q->{
32             q.query("DROP_TABLE_Marks;");
33             q.query("DROP_TABLE_Students;");
34         });
35     }
36 }
```

```
interface DBAccess{
    void beginTransaction();
    void commitTransaction();
    void rollbackTransaction();
    void doQuery(String query);
}
interface Query{ void query(String str); }
interface DoQueries{ void of(Query q); }
```

- (b) **(3 marks)** The code of the class 'User' is divided into two methods. Explain why this is good and how this makes the code of 'User' more testable.

Testing a main method directly is very challenging. By splitting the method in 2 we can make the 'runProgram' method take input that can be used to allow for testing.

- (c) **(4 marks)** Discuss the specific details of the implementation of 'makeTransaction'. How is it interacting with user code? How can user code interact with it?

MakeTransaction starts by beginning a transactions. MakeTransaction then interacts with user code by calling the lambda qs in a try catch. If anything has gone wrong, the code rolls back the transaction and propagate the error. Otherwise, the transaction is committed.  
User code can interact with it by providing a lambda that will receive in input a Query object, that can be used to query the database during the transaction.

- (d) **(3 marks)** Discuss the role of the two fields of 'TransactionManager'.

The field db stores an object able to access the database. This field is private and it is not accessible to the user of the class.  
The field q is initialized with a lambda, and it is used to allow the user code to query the database without even releasing the db object to the user.

- (e) **(10 marks)** Similarly as in the WAT questions, write the missing part ([???) of the code so that if this was a JUnit test suits, it could compile and run without errors. Complete the following code using a mock object instead of an instance of 'ConcreteDB.

```

1  class UserTest {
2      [???)
3      @Test void test1() {
4          log.clear();
5          User.runProgram(t);
6          assertEquals(log, List.of(
7              "beginTransaction",
8              "doQuery_DROP_TABLE_Marks;",
9              "doQuery_DROP_TABLE_Students;",
10             "commitTransaction"
11         ));
12     }
13 }
```

```
record MockAccess(List<String> log) implements DBAccess{  
    public void beginTransaction(){ log.add("beginTransaction"); }  
    public void commitTransaction(){ log.add("commitTransaction"); }  
    public void rollbackTransaction(){ log.add("rollBackTransaction"); }  
    public void doQuery(String query){ log.add("doQuery_"+query); }  
}  
List<String> log = new ArrayList<>();  
TransactionManager t = new TransactionManager(new MockAccess(log));
```

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.

## 4. Testing and Code contracts

**(30 marks)****Note:**

To get maximum marks you should avoid if/for/whiles and use streams when possible, but a correct answer using a traditional 'for' can still get most marks. To get maximum marks your code should be able to compile without warnings, but of course you can still get most marks if you miss minor details, like a ';' . However, mistakes about exceptions and generics are not minor details.

If you can not remember the name of a specific method, just make up a name and add a note describing the expected behaviour. You will not get full marks but you may get most marks.

- (a) **(6 marks)** Consider manual testing, automatic testing, unit testing and property based testing/fuzz testing. For each of those 4 kinds of testing, include at least one sentence defining what that kind of testing this is.

**Manual testing:**

Manual testing is manually running the program to check if it behaves as expected.

**Automatic testing:**

Automatic testing is writing (and running) a program to run the program and to automatically check if it behaves as expected.

**Unit testing:**

Unit Testing is to (automatically) test individual units of behaviour independently.

**Fuzz testing:**

Fuzz testing is to write a program that generates a large amount of potential input and then runs unit tests using those inputs.



- (b) **(2 marks)** As discussed in class, for a program with about 1000 lines of code encoding the desired behaviour, what is the minimum amount of lines of code that we should expect to see in the well designed tests for such program?

1000 lines

- (c) **(2 marks)** As discussed in class, in a program with about 1000 lines of code encoding the desired behaviour, how many lines of code we should expect inside well designed asserts/contracts and functions called only inside asserts/contracts?

1000 lines

- (d) (8 marks) Make a record `Person` with a `String` `name` and an `int` `age`. We need to ensure that all instances of `Person` have non null and non empty names and that their age is not a negative number. State what is the kind of contract that can enforce that, and write the full code for this `Person` record.

```
record Person(String name, int age){
    Person{
        assert name!=null && !name.isEmpty();
        assert age>=0;
    }
}
```

(e) (12 marks) Consider the code below:

```
1 record Person(String name, int age, List<String> qualifications){
2   Person doInterview(List<Person> candidates){
3     var alsoCs = candidates.stream()
4       .filter(c->c.qualifications().contains("C#")).findFirst();
5     if(alsoCs.isPresent()){ return alsoCs.get(); }
6     return candidates.get(0);
7   }
8 }
```

The code above has the following contract:

Preconditions: The method `doInterview` assumes that the age of all the candidates is 18 or over, and that their qualifications contains "Java". This method also assume many other conditions implicitly, and you have to discover them.

Postcondition: This method ensures that either the resulting candidate knows C#, or that none of the provided candidates knows C#.

On the following page rewrite the method `doInterview` to check at run time for the pre and post conditions using the pattern shown in the course. When rewriting such method, do not use the keyword `var` but show the type explicitly.

```
Person doInterview(List<Person> candidates) {
    assert candidates!=null;
    assert !candidates.isEmpty();
    assert candidates.stream().allMatch(c->c!=null && c.age()>=18);
    assert candidates.stream()
        .allMatch(c->c.qualifications().contains("Java"));
    Person result;
    try{
        Optional<Person> alsoCs = candidates.stream()
            .filter(c->c.qualifications().contains("C#")).findFirst();
        if(alsoCs.isPresent()){ return result=alsoCs.get(); }
        return result=candidates.get(0);
    }
    finally{
        assert result.qualifications().contains("C#") ||
            candidates.stream().noneMatch(c->c.qualifications().contains("C#"));
    }
}
```

\*\*\*\*\*

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.