TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2019

TRIMESTER 1

SWEN 221

SOFTWARE DEVELOPMENT

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
1.	Code Comprehension	30
2.	Testing & Object Contracts	30
3.	Java Generics	30
4.	Java Masterclass	30
	Total	120

1. Code Comprehension **(30 marks)**

(a) **(5 marks)** The questions below refer to the following piece of code:

```
1  abstract class Vehicle {
2   abstract void drive(int d);
3  }
4  class Car extends Vehicle {
5   private int colour;
6   void drive(int d) { ... }
7  }
8  Vehicle v1 = new Car();
9  Vehicle v2 = new Car();
10 v1.drive(1);
```

For each of the following questions, three statements (labelled A-C) have been provided. In each case, indicate which statement is correct by circling only one of the three choices.

(i)
A) `Car` is a *super-class* of `Vehicle`.
B) `Car` is a *sub-class* of `Vehicle`.
C) `Car` is *not a class* of `Vehicle`.

(ii)
A) `Car` *was a* `Vehicle`.
B) `Car` *has a* `Vehicle`.
C) `Car` *is a* `Vehicle`.

(iii)
A) Reflection ensures `v1.drive(1)` calls `Car`'s `drive method`.
B) Polymorphism ensures `v1.drive(1)` calls `Car`'s `drive method`.
C) The statement `v1.drive(1)` does not call `Car`'s `drive method`.

(iv)
A) A `Car` object is an instance of the `Car` class.
B) A `Car` class is an instance of the `Car` object.
C) A `Car` class is an instance of the `Vehicle` object.

(v)
A) There are many `Car` classes, but there is only one `Car` object.
B) There are many `Car` objects and many `Car` classes.
C) There are many `Car` objects, but there is only one `Car` class.

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.

(b) Consider the following implementation of a fixed-length vector class:

```
1  class FixedVector {
2   public static int count;
3   private Object data[];
4   private int length = 0;
5
6   public FixedVector(int size) {
7    count++;
8    data = new Object[size];
9   }
10
11  public void add(Object ob) {
12   if(length < data.length) { data[length++] = ob; }
13   else { throw new InsufficientSpaceException(); }
14  }
15
16  public Object get(int index) {
17   if(index < length) { return data[index]; }
18   else { throw new IndexOutOfBoundsException(); }
19  }
20
21  public Object clone() {
22   FixedVector c = new FixedVector(data.length);
23   for(int i=0;i<length;++i) {
24    c.data[i] = data[i];
25   }
26   return c;
27  }
28
29  public static void main(String argv[]) {
30   FixedVector n = new FixedVector(10);
31   try {
32    n.get(1);
33   } catch(InsufficientSpaceException ie) {
34    System.out.println("ERROR 1");
35   } catch(IndexOutOfBoundsException oe) {
36    System.out.println("ERROR 2");
37  }}}
```

i. **(1 mark)** In the implementation of `FixedVector`, how does the `add()` method signal an error has occurred?

ii. **(1 mark)** Is anything printed when the `main` method is executed? If so, what?

iii. **(2 marks)** What is being counted by the `count` field?

iv. **(2 marks)** What would happen if the `count` field was not `static`?

v. **(2 marks)** There are two standard ways of implementing a `clone()` method: *shallow copy* and *deep copy*. Which is used in `FixedVector`?

vi. **(2 marks)** Rewrite one line of `FixedVector` so that it uses the other type of clone.

vii. **(5 marks)** The `FixedVector` class does not use Java generics. Briefly discuss the pros and cons of making `FixedVector` use generics.

(c) **(5 marks)**

In Java, it is often possible to use an *interface* instead of an *abstract class*. Briefly describe which of these two mechanisms should be generally preferred and why.

(d) **(5 marks)** Java supports *reflection*. Briefly, discuss what this means.

2. Testing and Object Contracts **(30 marks)**

(a) **(5 marks)** For each of the following questions, three statements (labelled A-C) have been provided. In each case, indicate which statement is correct by circling only one of the three choices.

(i)
A) The JUnit test bar turns green if *at least one* test passes, red otherwise.
B) The JUnit test bar turns green if *most* tests pass, red otherwise.
C) The JUnit test bar turns green if *all* tests pass, red otherwise.

(ii)
A) Code coverage measures how many of your tests cover your program.
B) Code coverage measures how much of a program is covered by your tests.
C) Code coverage measures how much of the problem your program solves.

(iii)
A) Branch coverage measures conditionals with *at least one* branch taken.
B) Branch coverage measures conditionals with *both* branches taken.
C) Branch coverage measures conditionals with *no* branches taken.

(iv)
A) A white box test tests with knowledge of the implementation.
B) A black box test tests with knowledge of the implementation.
C) A green box test tests with knowledge of the implementation.

(v)
A) Boundary conditions are inputs at edges of the domain.
B) Boundary conditions are inputs causing integer overflows.
C) Boundary conditions are inputs causing loops to execute.

(b) Consider the following Java code, it compiles without error, but is of poor quality,

```
1  class Polygon {
2    public Point[] points;
3
4    public Polygon(List<Point> ps) {
5      points = new Point[ps.size()];
6      for(int i=0;i!=ps.size();++i) { points[i] = ps.get(i); }
7    }
8
9    public boolean equals(Object o) {
10     if(o instanceof Polygon) {
11       return ((Polygon)o).points == points;
12     } else {
13       return false;
14     }
15 } }
```

i. **(2 marks)** Describe a simple way to improve the *encapsulation* of this class.

ii. **(4 marks)** The `equals` method given above is incorrect. The problem is that two different polygons are *never* considered equal. Briefly, discuss why this is happening.

iii. **(5 marks)** Give a correct implementation of the `equals` method.

iv. **(2 marks)** The `Polygon` class does not work correctly with the `HashMap` class. State how it breaks the required object contract.

v. **(6 marks)** Write three sensible JUnit tests for the `equals` method. You should include at least one test for an edge case.

Test 1

Test 2

Test 3

vi. **(6 marks)** The object contract for `equals()` states that it should be *reflexive*, *symmetric* and *transitive*. Briefly, discuss what each means giving an example for each to illustrate.

Reflexive.

Symmetric.

Transitive.

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. Java Generics **(30 marks)**

(a) **(5 marks)** The questions below refer to the following piece of code:

```
1 class Box<T> {
2   private T item;
3
4   public Box(T item) { this.item = item; }
5
6   public T get() { return item; }
7 }
```

For each of the following questions, three statements (labelled A-C) have been provided. In each case, indicate which statement is correct by circling only one of the three choices.

(i)
A) `Box` is a *generic* class.
B) `Box` is a *final* class.
C) `Box` is a *sealed* class.

(iii)
A) `Box<String>` *may* hold `String` objects.
B) `Box<String>` *must* hold `String` objects.
C) `Box<String>` *never* holds `String` objects.

(ii)
A) `Box<Object>` *is* a subtype of `Box<String>`.
B) `Box<Object>` *is* a supertype of `Box<String>`.
C) `Box<Object>` is *neither* a subtype nor supertype of `Box<String>`.

(iv)
A) `Box<?>` uses a *joker* type.
B) `Box<?>` uses a *question* type.
C) `Box<?>` uses a *wildcard* type.

(v)
A) `Box<? extends String>` uses a *lower* bound.
B) `Box<? extends String>` uses an *upper* bound.
C) `Box<? extends String>` uses an *equal* bound.

(b) The `Tree` class, shown below, implements a *binary tree*.

i. **(6 marks)** By writing neatly on the box below turn `Tree` into a generic version, `Tree<T>`, where `T` specifies the type of data held in the tree.

```
1  public class Tree {
2    private Tree left;
3    private Tree right;
4    private Object data;
5
6    public Tree(Tree left, Tree right, Object data) {
7      this.left = left;
8      this.right = right;
9      this.data = data;
10   }
11
12   public Tree left() { return left; }
13
14   public Tree right() { return right; }
15
16   public Object data() { return data; }
17
18   public static void flatten(Tree tree, List list) {
19     if(tree.left != null) { flatten(tree.left,list); }
20     list.add(tree.data);
21     if(tree.right != null) { flatten(tree.right,list); }
22 } }
```

ii. **(4 marks)** In the box below, provide code which creates an instance of a generic `Tree` which holds `String`s. Your tree should contain at least three nodes.

iii. **(2 marks)** Briefly, discuss why `Tree<T>` is preferable to the non-generic version.

iv. **(3 marks)** Suppose you wanted a generic version of `Tree` which ensured every data object had a `compareTo()` method. Briefly, discuss how you would do this.

(c) **(6 marks)** `Tree<String>` is **not** a subtype of `Tree<Object>`. Briefly, discuss why this is not permitted, using example code to illustrate.

(d) **(4 marks)** Briefly, explain why `Tree<String>` **is** a subtype of `Tree<?>`.

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. Java Masterclass **(30 marks)**

As for the web assessment tool, for each of the following questions, provide in the answer box the code that should replace `[???]` such that `main(String[])` terminates normally.

(a) **(6 marks)**

```
1  //The answer must have balanced parentheses
2  class Vehicle{
3    public Vehicle(String plate) {this.plate=plate;}
4    public final String getPlate() {return this.plate;}
5    private String plate;
6    public int getWheels() {return 4;}
7  }
8  [???]
9  public class Question1 {
10   public static void main(String[]args) {
11     Vehicle t1=new Truck("BB99",6);
12     Vehicle t2=new Truck("CD88",8);
13     assert t1.getPlate().equals("BB99");
14     assert t2.getPlate().equals("CD88");
15     assert t1.getWheels()==6;
16     assert t2.getWheels()==8;
17 } }
```

(b) **(6 marks)**

```
1  //The answer must have balanced parentheses
2  public class Question2 {
3    public static void main(String[] args) {
4       try {assert false;}
5       finally {[???]}
6  } }
```

(c) **(6 marks)**

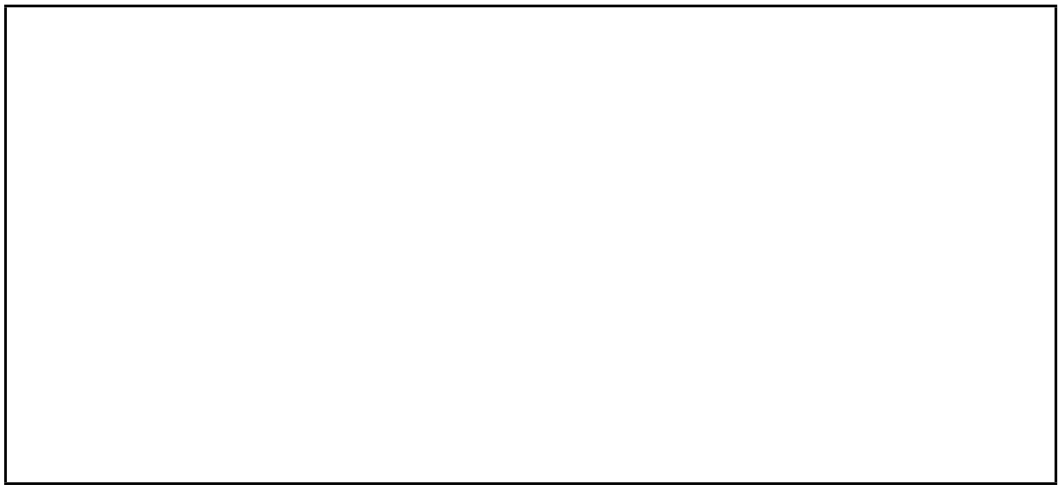
```
1  //The answer must have balanced parentheses
2  class Point{
3    public final int x;
4    public final int y;
5    Point(int x,int y){ this.x=x; this.y=y; }
6  }
7  [???]
8  public class Question3 {
9    public static void main(String[] args) {
10     Point p=new ColorPoint(1,2,"blue");
11     assert p.toString().equals("P(1,2,blue)");
12     assert p.toString().equals("P(1,2,BLACK)");
13   } }
```

(d) **(6 marks)**

```
1  //The answer must be digits only
2  class A{
3    int m(A p){return 1;}
4    private int m(B p){return 2;}
5  }
6  class B extends A{
7    int m(A p){return 3*100+super.m(p);}
8    int m(B p){return 4*100+super.m(p);}
9    int m(Object p){return 8;}
10 }
11 public class Question4 {
12   public static void main(String[] args) {
13     Object o=new B();
14     assert new B().m((B)o)==[???]; // only digits allowed
15 } }
```

(e) **(6 marks)**

```
1  //The answer must have balanced parentheses
2  interface A{
3    int m();
4    static int maker(Function<Integer,A> f){
5      return f.apply(1).m()*100+f.apply(3).m();
6    }
7  }
8  public class Question6 {
9    public static void main(String[]args){
10     assert A.maker([???])==103;
11   }
12 }
```



* * * * * * * * * * * * * * *

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.