TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Student ID: .....

EXAMINATIONS — 2012

MID-YEAR

SWEN221

Software Development

Time Allowed: 2 Hours

Instructions: There are 120 possible marks on the exam.
Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.
Non-electronic Foreign language dictionaries are allowed.
Calculators ARE NOT ALLOWED.
No reference material is allowed.

Question	Topic	Marks
1.	Debugging and Exceptions	20
2.	Encapsulation and Object Contracts	20
3.	Testing	20
4.	Inheritance and Polymorphism	20
5.	Java Generics	20
6.	Threading	20
	Total	120

Question 1. Debugging and Exceptions [20 marks]

Consider the following implementation of a Character Buffer, which compiles without error:

```
1  public class CharBuffer {
2   private char[] buffer;
3   private int length = 0;
4
5   public CharBuffer(int max) { buffer = new char[max]; }
6
7   public CharBuffer(char[] buffer) {
8    this.buffer = buffer;
9    this.length = buffer.length;
10  }
11
12  public void append(char c) {
13   if(length == buffer.length) {
14    // not enough space in buffer!
15    char[] nbuffer = new char[buffer.length * 2];
16    // copy elements from old buffer to new buffer
17    System.arraycopy(buffer,0,nbuffer,0,buffer.length);
18    // activate new buffer
19    buffer = nbuffer;
20   }
21   buffer[length] = c;
22   length = length + 1;
23  }
24
25  public char charAt(int index) {
26   if(index < 0 || index >= length) {
27    throw new IndexOutOfBoundsException();
28   }
29   return buffer[index];
30  }
31
32  // set the character at a given index
33  public void set(int index, char c) {
34    buffer[index] = c;
35  }
36
37  // Return size of buffer's active portion
38  public int length() { return length; }
39
40  // Construct string from active portion of buffer.
41  public String toString() {
42   return new String(buffer,0,length);
43  }
44 }
```

(a) The `charAt(int)` method returns the character at a given index in a `CharBuffer`.

(i) [2 marks] Under what circumstance is it impossible to call `charAt(int)` on a `CharBuffer` without raising an exception?

(ii) [2 marks] The `charAt(int)` method throws an exception when an error occurs. Rewrite this method to use an `assert` statement instead.

(b) Another important method in `CharBuffer` is `append(char)`.

(i) [4 marks] In your own words, describe what `append(char)` does.

(ii) [2 marks] Under what circumstances does `append(char)` fail to work correctly?

(iii) [2 marks] Rewrite `CharBuffer(int)` to ensure `append(char)` will always work correctly.

(c) Jim wrote the following code:

```
1  public static void main(String[] args){
2    char[] array = {'H','e','l','l','o'};
3    CharBuffer left = new CharBuffer(array);
4    CharBuffer right = new CharBuffer(array);
5    right.set(0,'h');
6    System.out.println(left.toString() + "␣=>␣" + right.toString());
7  }
```

Jim expected his code to print `"Hello␣=>␣hello"` when executed. However, he was surprised because it did not.

(i) [2 marks] What output was printed when Jim ran his code?

(ii) [4 marks] In your own words, describe what happened. You may use diagrams to support your explanation.

(iii) [2 marks] Suggest how `CharBuffer` could be improved to protect against this unexpected behaviour.

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.

Question 2. Encapsulation and Object Contracts [20 marks]

Consider the following implementation for *arithmetic expressions*, which compiles without error:

```
1  public abstract class Expression implements Cloneable {
2    public abstract int evaluate(Map<String,Integer> env);
3    public abstract Expression clone();
4  }
5
6  public final class Constant extends Expression {
7    private final int constant;
8
9    public Constant(int constant) { this.constant = constant; }
10   public int evaluate(Map<String,Integer> env) { return constant; }
11   public Constant clone() { return this; }
12 }
13
14 public final class Variable extends Expression {
15   private final String name;
16
17   public Variable(String name) { this.name = name; }
18
19   public int evaluate(Map<String,Integer> env) {
20     return env.get(name);
21   }
22
23   public Variable clone() { return this; }
24 }
25
26 public final class Sum extends Expression {
27   public final Expression[] operands;
28
29   public Sum(Expression[] ops) {
30     this.operands = new Expression[ops.length];
31     int i = 0 ;
32     for(Expression e : ops) { this.operands[i++] = e; }
33   }
34
35   public int evaluate(Map<String,Integer> env) {
36     int r = 0;
37     for(Expression e : operands) { r = r + e.evaluate(env); }
38     return r;
39   }
40
41   public Sum clone() { return new Sum(operands); }
42 }
```

(a) [2 marks] The code above represents arithmetic expressions. Illustrate how to create an instance of `Expression` corresponding to `2+x`.

(b) [4 marks] The `evaluate(Map<String,Integer> env)` method is required for all instances of `Expression`. Briefly, discuss the different implementations of this method.

(c) The `Expression` class provides a `clone()` method.

(i) [2 marks] There are two standard ways of implementing a clone method. Which kind of clone is implemented by the subclasses of `Expression` given on page 6?

(ii) [4 marks] Briefly, discuss how you would modify the subclasses of `Expression` given on page 6 to implement the other type of clone.

(iii) [4 marks] The `Constant` and `Variable` classes provide `clone()` methods that return **`this`**. Briefly, discuss if and why this is a sensible approach in these cases.

(d) [4 marks] The `operands` field of `Sum` is public. Using this as an example, discuss why public fields are considered bad practice.

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.

Question 3. Testing

[20 marks]

Consider the following code representing a 2-dimensional point:

```
1  public class Point {
2    private int x, y;
3    public Point(int x, int y) { this.x = x; this.y = y; }
4
5    public boolean equals(Object o) {
6      if(o instanceof Point) {
7        Point c = (Point) o;
8          if(c.x != x) { return false; }
9          if(c.y != y) { return false; }
10         return true;
11     }
12     return false;
13   }
14
15   public int compareTo(Point p) {
16     if(x > p.x) { return -1; }
17     if(x < p.x) { return 1; }
18     if(y < p.y) { return -1; }
19     if(y > p.y) { return 1; }
20     return 0;
21   }
22 }
23
24 public class PointTests {
25   @Test void testEquals() {
26     assertTrue(new Point(1,2).equals(new Point(1,2)));
27   }
28
29   @Test void testEquals() {
30     assertFalse(new Point(1,2).equals(new Point(2,2)));
31   }
32
33   @Test void testCompare() {
34     assertTrue(new Point(2,3).compareTo(new Point(2,1)) > 0);
35   }
36
37   @Test void testCompare() {
38     assertTrue(new Point(2,1).compareTo(new Point(2,3)) < 0);
39   }
40 }
```

(a) [4 marks] Draw the *control-flow graph* for the `Point.compareTo(Point)` method:

(b) A common way to measure the effectiveness of a test suite is to calculate *coverage*.

(i) [2 marks] State the *statement coverage* criterion.

(ii) [2 marks] Give the total *statement coverage* of `Point` obtained with `PointTests`.

(iii) [2 marks] State the *branch coverage* criterion.

(iv) [2 marks] Give the total *branch coverage* of `Point` obtained with `PointTests`.

(v) [4 marks] Briefly, discuss why *branch coverage* is superior to *statement coverage*.

(c) The *path coverage* criterion counts the proportion of all possible execution paths which are tested.

(i) [2 marks] Why is path coverage impossible to measure in general?

(ii) [2 marks] State what the *simple path coverage* criterion is.

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.

Question 4. Inheritance and Polymorphism [20 marks]

Consider the following Java classes.

```
1  public class A{
2    public static int x=1;
3    public int m(A a, B b) {
4      return a.m(b,b)+x;
5    }
6  }
7
8  public class B extends A {
9    public int m(A a1, B a2){
10     if(a1==a2)return x;
11     return super.m(a1,a2);
12   }
13 }
```

(a) Given the above declarations, state whether the following classes compile without error. For any which do not compile, briefly describe the problem.

(i) [2 marks]

```
1  public class C extends B {
2    public float m(A a1, B a2){
3      return 1.2f;
4    }
5  }
```

(ii) [2 marks]

```
1  public class D extends B {
2    public D(int f){
3      super();
4      x=3;
5    }
6  }
```

(b) Given classes `A` and `B` on page 14, state the output from the following code snippets.

(i) [2 marks]

```
1 public class Main {
2   public static void main(String[] args) {
3     B b1=new B();
4     A a1=b1;
5     A a2=b1;
6     System.out.println(a1.m(a2,b1));
7 } }
```

(ii) [2 marks]

```
1 public class Main {
2   public static void main(String[] args) {
3     A a1=new A();
4     A a2=new A();
5     B b1=new B();
6     System.out.println(a1.m(a2,b1));
7 } }
```

(iii) [3 marks]

```
1 public class Main {
2   public static void main(String[] args) {
3     A.x=2;
4     A a1=new A();
5     B b1=new B();
6     System.out.println(b1.m(a1,b1));
7 } }
```

(iv) [3 marks]

```
1 public class Main{
2   public static void main(String[] args){
3     A a1=new A();
4     A a2=new A(){ int m(A a,B b){return 10;}};
5     B b1=new B();
6     System.out.println(a1.m(a2,b1));
7   }
8 }
```

(v) [3 marks]

```
1 public class Main{
2   public static void main(String[] args){
3     A a1=new A();
4     A a2=new A();
5     B b1=(B)a1;
6     System.out.println(a1.m(a2,b1));
7   }
8 }
```

(vi) [3 marks]

```
1 public class Main{
2   public static void main(String[] args){
3     B b1=new B(){int m(A a,A b){return 10;}};
4     System.out.println(b1.m(b1,b1));
5   }
6 }
```

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.

Question 5. Java Generics [20 marks]

(a) The `Box` class, shown below, implements a generic container for `Items`. The only thing relevant for a general item is the value.

(i) [10 marks] By writing neatly on the box below turn `Box` into a generic version, `Box<T extends Item>`, where `T` specifies the type of item held in the box.

```
1  public abstract class Item { abstract int value(); }
2
3  public class Box {
4
5    private Item item;
6
7    public Box(Item item) { this.item=item; }
8
9    public Item getItem() { return this.item; }
10
11   public void setItem(Item item) { this.item=item; }
12
13   public static void swap(Box b1, Box b2) {
14
15     Item aux=b1.item;
16     b1.item=b2.item;
17     b2.item=aux;
18   }
19
20   public static List boxAll(List items) {
21
22     List result=new ArrayList();
23
24     for(Object i : items) { result.add(new Box((Item)i)); }
25     return result;
26   }
27
28   public static void sort(List boxes) {
29
30     Collections.sort(boxes,new Comparator() {
31
32       public int compare(Object b1, Object b2) {
33
34         return ((Box)b1).item.value()-((Box)b2).item.value();
35       }
36     });
37   }
38 }
```

(ii) [3 marks] In the box below, provide code which define a minimal concrete `Toy` class and create an instance of a generic `Box` which holds `Toys`.

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

(b) [5 marks] In Java, `Box<Toy>` is **not** a supertype of `Box<Item>`. Discuss why this is not permitted, using example code to illustrate.

Question 6. Threading

[20 marks]

Consider the following implementation of a *parallel sum*, which compiles without error:

```
1  public class ParSum {
2    private int sum;
3    private int[] data;
4
5    public ParSum(int[] data) {
6      this.data = data;
7    }
8
9    public long go(int numProcessors) {
10     // determine job size based on number of available processors
11     int jobSize = data.length / numProcessors;
12
13     // create and start each job
14     int index = 0;
15     SumJob[] jobs = new SumJob[numProcessors];
16     for (int i = 0; i != numProcessors; ++i) {
17       jobs[i] = new SumJob(index, index + jobSize);
18       jobs[i].start();
19       index = index + jobSize;
20     }
21
22     // return the result
23     return sum;
24   }
25
26   // SumJob is a non-static inner class
27   private class SumJob extends Thread {
28     private int start;
29     private int end;
30
31     public SumJob(int start, int end) {
32       this.start = start;
33       this.end = end;
34     }
35
36     public void run() {
37       for(int i=start;i!=end;++i) {
38         sum = sum + data[i];
39       }
40     }
41   }
42 }
```

(a) The code shown on page 20 contains several problems. For each problem identified below, briefly discuss what the issue is and how the code can be modified to fix the problem.

(i) [5 marks] *Reads and writes to the field* `sum` *are not synchronised.*

(ii) [5 marks] *The result from* `go()` *is returned before jobs finish.*

(iii) [5 marks] *Elements of* `data` *are sometimes ignored completely.*

(b) [5 marks] In multi-threaded code, synchronisation should be *minimised* to increase performance. Outline how you could redesign `ParSum` to improve performance.

****

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.