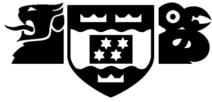


Student ID: ....

EXAMINATIONS — 2009

MID-YEAR

ENGR202 // COMP205
Software Design and Engineering

Time Allowed: 3 Hours

Instructions: There are 180 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.	Java Fundamentals I	30
2.	Principles of Software Design I	30
3.	Practices of Software Engineering I	30
4.	Java Fundamentals II	30
5.	Practices of Software Engineering II	30
6.	Principles of Software Design II	30
	Total	180

Student ID: ....

Question 1. Java Fundamentals I [30 marks]

Consider the following Java code, which compiles without error.

```
interface Shape {
  double GetArea();  // returns the area of this shape
}

class X // has x and y coordinates and a name
{
  private int x, y;
  private String name = "default";

  public X(int x, int y) { this.x = x; this.y = y; }
  public int one() { return x; }
  public int two() { return y; }
  public void changeNameTo(String foo) { name = foo; }
  public int hashCode() { int result = 31 * (31 + x) + y;
    result = 31 * result + (name == null ? 0 : name.hashCode());
    return result; // return result
  }
}

class Circle implements Shape
{
  static final double pi_ = Math.PI;
  private X p1_ = null;
  private int RADIUS;

  public Circle(X point1, int r) {
    X x; if ((x = point1) != null) p1_ = x;
    RADIUS = r;
}

    public double GetArea() {
        // return pi times radius times radius
        return pi_ * RADIUS * RADIUS; }}

class Rectangle extends X implements Shape {
    private int w,h; // width and height
    public Rectangle (int x, int y, int w, int h) {
        super(x, y);
        this.h = h; this.w = w;
    }

    public double GetArea() { return w*h; }
}
```

(a) [10 marks] Circle and number five separate problems of style with the code on the previous page. For each problem, write a brief (i.e. one or two line) description of the problem in the corresponding box below.

1)

2)

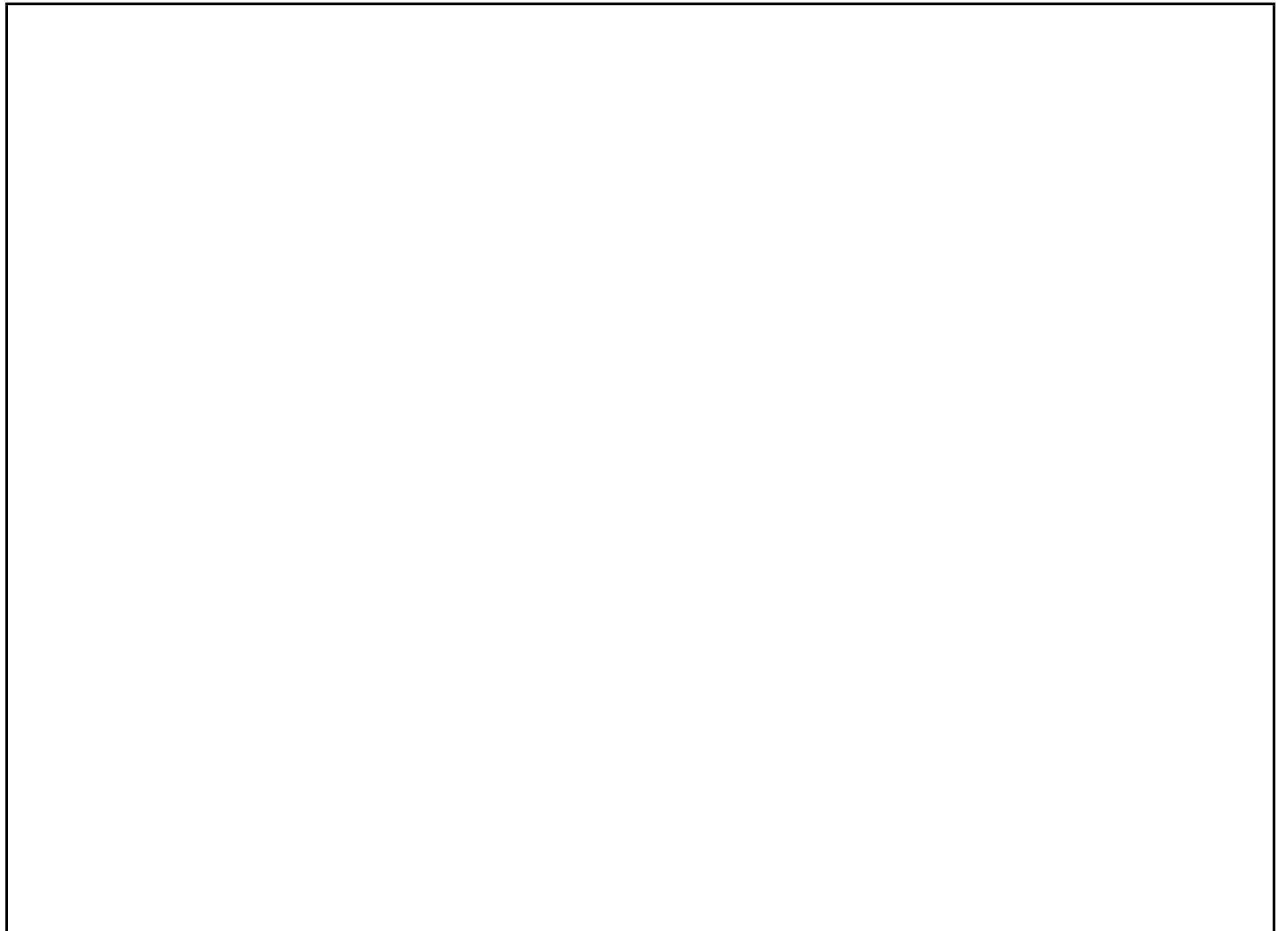
3)

4)

5)

(b) [8 marks] In the box below, draw a *UML Class Diagram* for the classes and interfaces shown on the previous page. (Show relationships and multiplicity, but no attributes and methods).

(c) [12 marks] Write an `equals` method for class `X` that is consistent with the `hashCode` method provided.



Student ID: . . . .

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. Principles of Software Design I [30 marks]

Consider the following Java code which is part of a data structure library. It compiles and runs correctly.

```
class Stack {

  private LinkedList<Object> elements =
                                  new LinkedList<Object>();

  public void push(Object o) {elements.addFirst(o);}

  public Object pop() {return elements.removeFirst();}
}

class Queue {

  private LinkedList<Object> elements =
                                  new LinkedList<Object>();

  public void push(Object o) {elements.addLast(o);}

  public Object pop() {return elements.removeFirst();}
}

class PeekQueue {

  private LinkedList<Object> elements =
                                  new LinkedList<Object>();

  public void push(Object o) {elements.addLast(o);}

  public Object pop() {return elements.removeFirst();}

  public Object peek() {return elements.getFirst();}
}

class Repeater {

  private Object element;

  public void push(Object o) {element = o;}

  public Object pop() {return element;}
}
```

(a) [5 marks] Refactor the declaration of the `PeekQueue` class by writing carefully on the code so that it inherits from `Queue`. Cross out any unnecessary code from `PeekQueue`. Did you have to alter `Queue` in any way? If so, explain why; if not, explain why not.

(b) [8 marks] Write a new class called `Sequence` that can be the superclass of both `Queue` and `Stack`. Carefully change the code of `Queue` and `Stack` to use that superclass.

(c) [2 marks] Did you make `Sequence` an **`abstract`** class? If so explain why; if not, explain why not.

(d) [3 marks] In the box below, define an **`interface`**, called `Sequenceable`, which all the classes (`Stack`, `Queue`, `PeekQueue`, `Repeater`, and `Sequence`) can implement.

(e) [2 marks] Would it have been better to make `Sequenceable` an **`interface`**, or an **`abstract`** class? Explain why?

(f) [5 marks] Explain what *behavioural subtyping* is, and why it is important, using these classes as examples.

(g) [5 marks] Imagine you've been working as a professional software engineer or software developer for several years. Some students working on their summer jobs ask you "why is it important that we refactor our code to use inheritance?" What would you tell them?

Question 3. Practices of Software Engineering I [30 marks]

(a) Consider the following Java class which compiles correctly, and the outline of a JUnit class for testing it. The `Stackulator` is an implementation of a "reverse notation" calculator: to compute $1+2$ you enter "1", then enter "2", then press "+", and the calculator displays the result.

```
class Stackulator {
    protected int a;
    protected int b;

    public int enter(int i) {b = a; a = i; return a;}

    public int plus() {a = a + b; return a;}
    public int times() {a = a * b; return a;}
    public int minus() {a = a - b; return a;}
    public int divide() {a = b / a ; return a;}
}

public class StackulatorTest {
    public StackulatorTest() {}

    private Stackulator skl;

    @Before public void setUp() {
        skl = new Stackulator();
    }

   ... // test cases go here
 }
```

(i) [10 marks] Each of the following JUnit test methods are part of the `StackulatorTest` class. For each, state whether it passes or fails — if it fails, explain why.

1)

```
    @Test public void addition() {
        // 3 + 4 = 7
        skl.enter(3);
        skl.enter(4);
        assertEquals(7, skl.plus());
    }
```

2)

```
@Test public void multiplication() {
    // 2 * 3 * 4 = 24
    skl.enter(2);
    skl.enter(3);
    skl.times();
    skl.enter(4);
    assertEquals(24, skl.times());
}
```

3)

```
@Test public void subtraction() {
    // 21 - 7 = 14
    skl.enter(21);
    skl.enter(7);
    assertEquals(14, skl.minus());
}
```

4)

```
@Test public void division() {
    // 21 / 7 = 3
    skl.enter(21);
    skl.enter(7);
    assertEquals(3, skl.divide());
}
```

5)

```
@Test public void expression() {
    // 5 + (10 * 3) = 35
    skl.enter(5);
    skl.enter(10);
    skl.enter(3);
    skl.times();
    assertEquals(35, skl.plus());
}
```

(ii) [3 marks] Consider this next test case. Does it pass or fail? If it passes, explain why; if it fails, explain why?

```
@Test(expected = NullPointerException.class)
    public void crash() {
    skl.divide();
}
```

(iii) [2 marks] Explain why JUnit supports the `expected` parameter to the `@Test` annotation.

Student ID: . . . .

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) [10 marks] Consider the following Java class, which compiles and runs correctly, and the outline of a JUnit class for testing it given below:

```
class Ratagotchi {
    private int happiness = 1;
    private int health = 1;
    private int age = 10; // age times 10

    public int getHappiness() {return happiness;}
    public int getHealth() {return health;}
    public int getAge() {return age / 10;}

    public void feed() {++health;}
    public void play() {++happiness;}
    public void tick() { if ((++age % 10) == 0) {health--;}}
}

public class RatagotchiTest {
    public RatagotchiTest() {}

    private Ratagotchi rat;

    @Before public void setUp() {rat = new Ratagotchi();}

   ... // test cases go here
}
```

Write JUnit test methods that could be added into this class to test that each of the following holds for a new Ratagotchi:

1. It has 1 happiness, 1 health, and 1 age.
2. Feeding it makes its health 2.
3. Doing three ticks does not change anything.
4. Doing ten ticks increases age to 2 and decreases health to zero.

1)

2)

3)

4)

(c) [5 marks] Explain why professional software engineers use unit tests.

Question 4. Java Fundamentals II

(a) [10 marks] The following class contains a list of `Nameable` objects. It can print out the name of each object in the list. Carefully and neatly change this code to make a generic `NameList` class.

```
public class NameList {

   private List items = new LinkedList();

   void add(Nameable i) {items.add(i);}

   void printOut() {

    for (Object i : items) {

         Nameable n = (Nameable) i;

         System.out.println(n.name());

    }

   }

   public Iterator iterator() {

    return items.iterator();

   }
}


// you do not need to change this interface
interface Nameable {
 String name();
}
```

(b) [5 marks] Does the `iterator` method on `NameList` breach encapsulation? Why or why not?

(c) [5 marks] Two programmers, Ben and Terri, need to write code that iterates through two `NameLists` in parallel.

Ben writes:

```
Iterator<Nameable> it3 = list1.iterator();
Iterator<Nameable> it4 = list2.iterator();

while (true) {
    try {System.out.println(it3.next() + " " + it4.next());}
    catch (NoSuchElementException e) {break;}
 }
```

while Terri writes:

```
Iterator<Nameable> it1 = list1.iterator();
Iterator<Nameable> it2 = list2.iterator();

while (it1.hasNext() && it2.hasNext())
    System.out.println(it1.next() + " " + it2.next());
```

Whose code is better, Ben's or Terri's? Explain why.

(d) [5 marks] Does the answer to your question change if you know that the "`toString`" method on some of the objects in the `NameLists` might throw a `NullPointerException`? Explain your answer.

(e) [5 marks] Explain the following type declaration in English:

```
static <T> void sort(List<T> list, Comparator<? super T> c)
```

You may find this interface definition useful:

```
interface Comparator<T> {
   // Compares its two arguments for order.  Returns a negative integer,
   // zero, or a positive integer as the first argument is less than, equal
   // to, or greater than the second.
   int compare(T o1, T o2);
}
```

Question 5. Practices of Software Engineering II [30 marks]

You are part of a software team developing an email client. Your job is to develop the email client's address book. A rough description of the requirements follows:

> "Users can organise their address book into folders, groups, and aliases. The folders are used to generate a hierarchical organisation and contain the actual addresses. Groups are used to group together addresses that might live in separate folders. Aliases are nicknames that can be used in place of an address or group. When defining a group, a user can use aliases rather than actual email addresses, so that a change in a person's email address can be corrected in just one place, even if it is used in many groups."

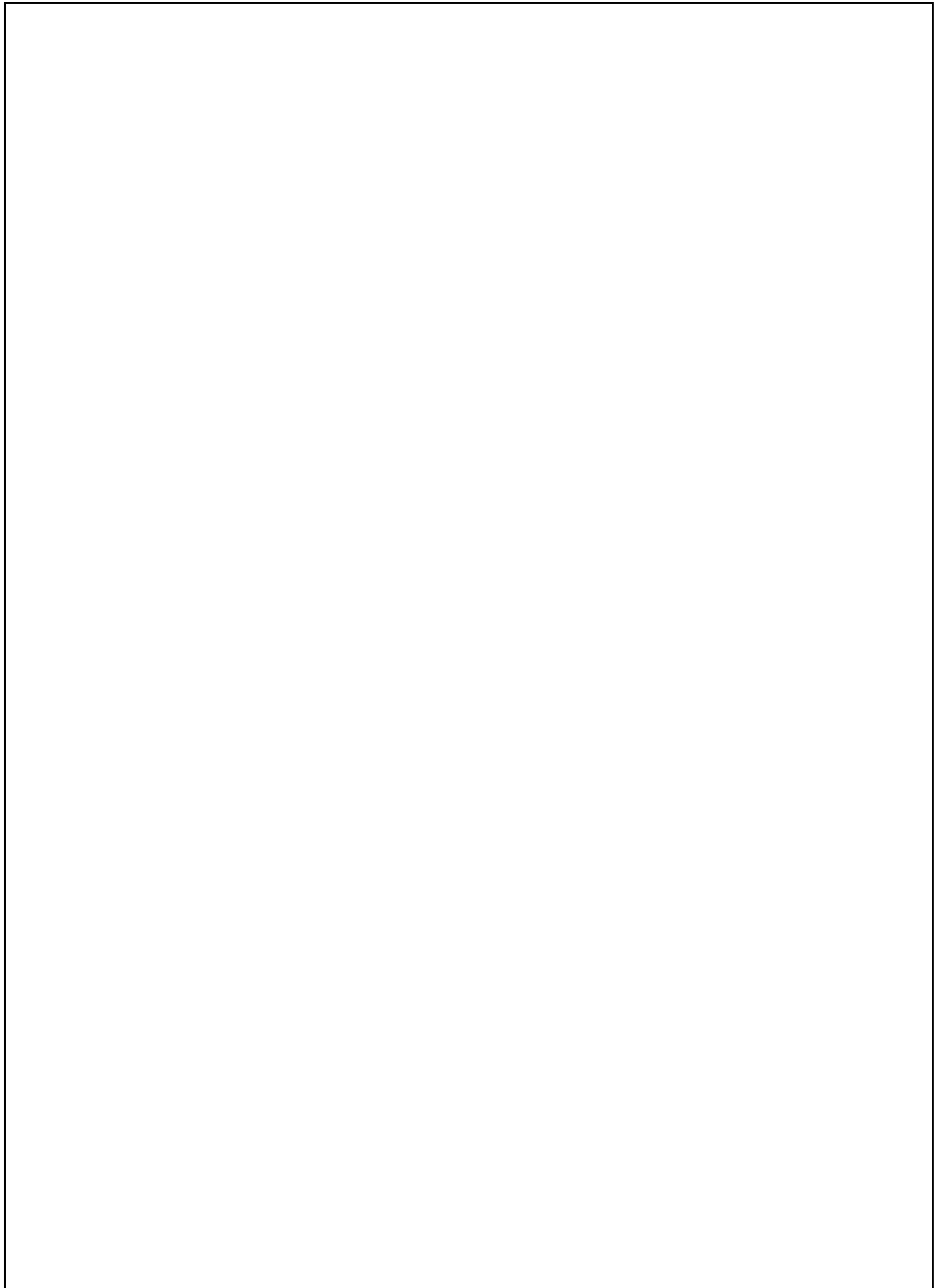
(a) [6 marks] Give the name of a design pattern that you are intending to use in the design of the address book. Explain its intent, and discuss consequences of using this pattern for your design.

Student ID: . . . .

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) [14 marks] In the box below, draw a *UML Class Diagram* for the email client's address book. Show relationships, multiplicity and class attributes in your diagram, but not methods.



(c) [10 marks] In the box below, give Java code for the classes in your design that represent *groups* and *aliases*. Provide methods that return a collection of all email addresses referred to by a group or alias. In doing this, you should follow good software engineering practice.

Student ID: . . . .

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 6. Principles of Software Design II

[30 marks]

The following code implements yet another stack, which uses a fixed-size array for storing the elements in the stack, and a `size` field to store the number of elements currently in the stack.

```
/** A stack of non-null Objects. */
public class Stack {
  protected Object[] data;
  protected int size = 0;

  public Stack(int maxSize) {
    data = new Object[maxSize];
  }

  public void push(Object o) {
    assert o != null;
    data[size] = o;
    size++;
  }

  public Object pop() {
    if (size > 0) size--;
    return data[size];
  }

  public int size() {
    return size;
  }

  public int maxSize() {
    return data.length;
  }
}
```

(a) [2 marks] Give an example of an invalid state for a `Stack` object.

(b) [2 marks] What happens if a `pop()` operation is called on an empty stack?

(c) [2 marks] What is a *class invariant*?

(d) [2 marks] The methods of `Stack` must *preserve the class invariant*. What does this mean?

(e) [4 marks] Using a simple logic notation, give a suitable class invariant for the `Stack` class.

(f) [2 marks] In Programming by Contract, when should the *precondition* of an operation hold? Who's responsibility is it to ensure that it holds?

(g) [2 marks] In Programming by Contract, when should the *postcondition* of an operation hold? Who's responsibility is it to ensure that it holds?

(h) [8 marks] Give suitable *preconditions and postconditions* for the `push()` and `pop()` operations.

(i) [2 marks] Assume you implement a subtype of `Stack`. What are the requirements on the *class invariant* for this *subtype*?

(j) [4 marks] Show how a subclass of `Stack` can break the class invariant for `Stack` and explain how `Stack` could be modified to prevent subclasses from breaking `Stack`'s class invariant.

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.

Student ID: . . . .

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.

****