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

EXAMINATIONS — 2006

MID-YEAR

COMP 205
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.	Object-Oriented Design	30
2.	Design Patterns	30
3.	The Java Language	30
4.	Principles of Object-Oriented Programming	30
5.	Practices of Software Engineering	30
6.	Class Invariants	30
	Total	180

Student ID: .....

Question 1. Object-Oriented Design [30 marks]

Consider the following narrative:

> Sam flips open his mobile phone and selects the "Address Book". The phone asks the Address Book object for a list of all the Contact objects for people stored in Sam's phone and displays them in alphabetical order. Since Sam has many contacts, only the first few can be shown on the small display.
>
> Sam enters the letter "S" and, immediately, the list of contacts being displayed narrows to those whose name begins with "S". Sam moves through the list of names using the down button and selects "Susan". The phone retrieves the stored Contact object that represents Susan and displays her details.
>
> Finally, Sam presses the green "dial" button to call Susan. The phone asks Susan's Contact object for her Phone Number object and passes it to the Telephone Dialer object.

(a) [5 marks] Write an **essential use case** for this narrative with at most 3 intentions and 3 responsibilities.

(b) [10 marks] Draw a simple **UML class diagram**, including the following classes, that could be used to implement your essential use case from part **(a)**. (You only need to show class names and their relationships; you do not need to show the attributes or methods of each class).

- Address Book
- Contact
- Phone Number
- Telephone Dialer
- Mobile Phone

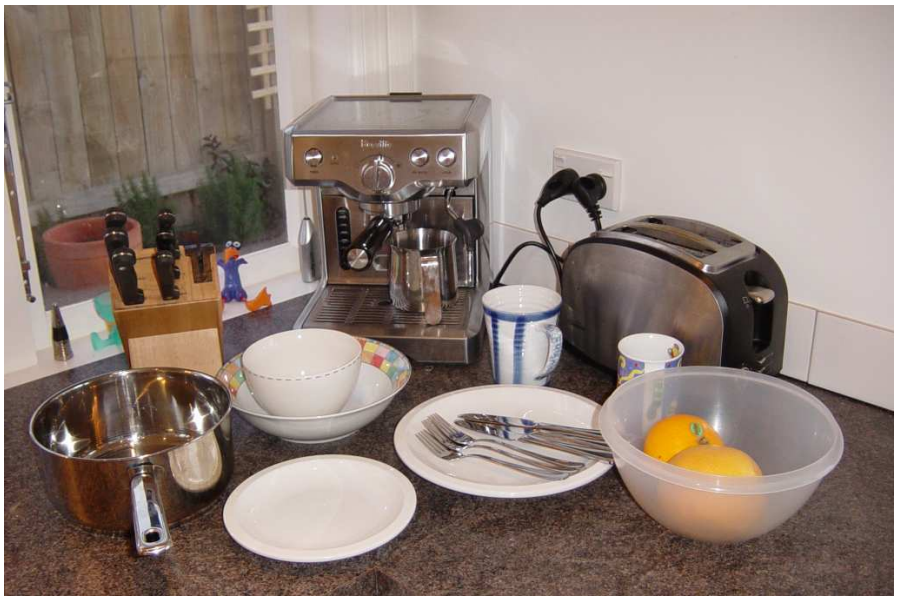
(c) [5 marks] Draw **CRC cards** for the following three classes: *Mobile Phone*, *Address Book* and *Contact*.

(d) [10 marks] Draw a simple **UML sequence diagram** illustrating how your essential use case works with the classes in your design

Student ID: .....

Question 2. Design Patterns [30 marks]

You have been contacted to design software for representing items found in a Kitchen. A photo of a Kitchen follows.



There are many items of different sizes that can be found in a kitchen. Plastic bowls are useful for storing things in the fridge and for mixing ingredients. Saucepans are used to cook food or hold smaller items. When not being used, Saucepans are often stored in a cupboard and may be placed inside other saucepans to save space. Plates are important! There are different kinds of plate, including side plates, main plates and serving plates. Bowls are similar to plates. They can be used to hold fruit and small bowls can be stacked inside large ones. Knives and Forks are small items that can be placed inside saucepans and bowls if needed. Chopping knives are for preparing food and are stored in a knife block. They can be blunt sometimes and may need sharpening. Drinks are served in cups, such as coffee cups and tea cups. Cups are similar to bowls, except they have handles.

(a) [10 marks] From the description above, identify ten household items that are found in a kitchen, including four containers to put items in.

(b) [15 marks] In the box below, draw a simple **UML class diagram** to represent these items. (You only need to show class names and their relationships; you do not need to show the attributes or methods of each class).

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 continued on next page)

(Question 2 continued)

(c) [5 marks] The system you are designing has a display of the current positions of the kitchen items. The display needs to be updated whenever a kitchen item is placed into another. Using the OBSERVER pattern, extend your class diagram so that the display can be automatically notified when a kitchen item is placed into another. Only include those classes from your previous diagram which are necessary to describe the extension.

Student ID: .....

Question 3. The Java Language [30 marks]

Consider the following piece of Java code for representing stationery items (e.g. pens, pencils, rulers etc) and stationery containers (e.g. pencil cases, pen holders, etc).

```
public abstract class StationeryItem {
 private StationeryHolder holder = null;

 public StationeryHolder getHolder() { return holder; }
 public void setHolder(StationeryHolder h) { holder = h; }
 public abstract String getDescription();
}

public class BallPointPen extends StationeryItem {
 public Color penColour;
 private float amountLeft;  // value between 0 and 1.0

 public BallPointPen(Color colour, float amount) {
  penColour = colour;
  amountLeft = amount;
 }

 public Color getColor() { return penColour; }
 public float getAmountLeft() { return amountLeft; }
 public String getDescription() { return penColour + " BallPointPen"; }
}

public class StationeryHolder extends StationeryItem {
 private List<StationeryItem> items = new ArrayList<StationeryItem>();

 public void addItem(StationeryItem item) {
  assert item.getHolder() == null;
  items.add(item);
  item.setHolder(this);
 }

 public void removeItem(StationeryItem item) {
  assert item.getHolder() == this;
  items.remove(item);
  item.setHolder(null);
 }

 public String getDescription() {
  return "Stationery holder";
}}
```

(a) [2 marks] Write Java code which will create one red ball-point pen, create one stationery holder and add the red pen to the stationery container. You may use the constant `Color.red` to denote the necessary `Color`. Assume the new pen is full.

(b) [2 marks] Is the `BallPointPen` class properly encapsulated? Justify your answer.

(c) [6 marks] In the space below, draw a **UML class diagram** for the `StationeryItem`, `BallPointPen` and `StationeryHolder` classes. You should include attributes, inheritance and associations as necessary; but, you do not need to show methods.

(d) [2 marks] Why is it impossible, without changing `StationeryHolder`, to write a method which accepts a `StationeryHolder` object and prints its contents?

(e) [2 marks] The `StationeryItem.getDescription()` method is *abstract*. What does this mean?

(f) [2 marks] Explain what happens if you attempt to remove a `StationeryItem` from a `StationeryHolder` that it is not in.

(g) [5 marks] Briefly outline the changes to the code which would be needed to turn `StationeryItem` into an *interface*, rather than a *class*.

(h) [5 marks] The `BallPointPen` class is not immutable. State three different ways of changing a `BallPointPen` object after it has been created.

(i) [4 marks] What constraint does the `StationaryItem.holder` field enable the code to enforce?

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 4. Principles of Object-Oriented Programming [30 marks]

(a) Consider the following specification for an alarm clock:

> "An alarm clock maintains the current time. A user can set the time as necessary and may also set a time for the alarm to go off. An alarm clock may be *electronic* or *mechanical* (e.g. wind-up). A *radio* alarm clock can play a radio station, instead of making an alarm sound. Identical alarm clocks may report different times."

(i) [2 marks] According to the specification, what *interface* does an alarm clock have?

(ii) [2 marks] According to the specification, what *implementations* are permitted for an alarm clock?

(iii) [2 marks] According to the specification, what constitutes the *state* of an alarm clock?

(iv) [2 marks] From the specification, how would you justify the claim that an alarm clock has *identity*?

(v) [2 marks] What *inheritance* relationship would hold between alarm clocks in the specification?

(vi) [5 marks] A key principle of any object-oriented system is *encapsulation*. Briefly discuss how this principle would apply to alarm clocks as described in the specification.

(b) Consider the following code for representing shapes in Java:

```
public class Rectangle {
 private int x, y;
 private int width, height;
 // accessors
 public int getX() { return x; }
 public int getY() { return x; }
 public int getWidth() { return width; }
 public int getHeight() { return height; }
 // mutators
 public void setPosition(int _x, int _y) { x=_x; y=_y; }
 public void setDimensions(int _width, int _height) {
  width=_width; height=_height;
}}

public class Square extends Rectangle {
 public void setDimensions(int _width, int _height) {
  if(_width != _height) { throw new RuntimeException(...); }
  super.setDimensions(_width,_height);
}}
```

(i) [4 marks] State the key idea of the *Liskov Substitution Principle*.

(ii) [6 marks] Explain why `Square` is not a *subtype* of `Rectangle`. You may use examples to aid your explanation where necessary.

(iii) [5 marks] Outline one way in which the code could be modified to make `Square` a subtype of `Rectangle`.

Question 5. Practices of Software Engineering [30 marks]

(a) Consider the following Book class written in Java:

```
public class Book { // This class represents a Book (1)
 private String x;
 private String y;

 public Book(String t, String a) {
  x = t;  // set title  (2)
  y = a;  // set author (3)
 }

 public String getAuthor() { return y; }   // Returns the Book's Author (4)
 public String getTitle() { return x; }    // Returns the Book's Title  (5)
}
```

(i) [2 marks] The fields "`Book.x`" and "`Book.y`" are poorly named. Suggest some better names for these fields.

(ii) [5 marks] Briefly discuss the problems caused by poor variable names, such as "`Book.x`" and "`Book.y`".

(iii) [5 marks] Good comments are an essential ingredient of good software. Consider the five numbered comments in the `Book` class. Which of these are good, and which are bad? Briefly explain your reasons.

(b) You have been assigned the job of *white-box testing* a vector multiplication method. The code for that method is provided below:

```
void multiply(Vector<Integer> v1, Vector<Integer> v2) {
 if(v2 == null) { return; }
 for(int i=0;i!=v1.size();++i) {
  int x = v1.get(i);
  int y = v2.get(i);
  v1.set(i,x*y);
}}
```

(i) [2 marks] Why is an *exhaustive* test of all inputs for this function impractical?

(ii) [5 marks] Briefly describe "White-Box Testing".

(iii) [5 marks] Identify five distinct classes of test case for the `multiply` method.

(iv) [6 marks] A desirable property of any set of test cases is that they are *path complete*. Discuss what this means.

Student ID: .....

Question 6. Class Invariants [30 marks]

(a) [6 marks] For each of the following questions, three statements (labelled A-C) have been provided. In each case, you should indicate the correct answer by circling only one of the three choices.

(i)
A) A class invariant should *never* be true for instances of that class.
B) A class invariant should *always* be true for instances of that class.
C) A class invariant should *sometimes* be true for instances of that class.

(ii)
A) The methods of a class should *never* preserve the class invariant.
B) The methods of a class should *sometimes* preserve the class invariant.
C) The methods of a class should *always* preserve the class invariant.

(iii)
A) The pre-condition for a method should *always* be true before it is called.
B) The pre-condition for a method should *never* be true before it is called.
C) The pre-condition for a method should *sometimes* be true before it is called.

(iv)
A) The post-condition for a method should *sometimes* be true after it is called.
B) The post-condition for a method should *never* be true after it is called.
C) The post-condition for a method should *always* be true after it is called.

(v)
A) The pre-condition for a method `C.f()` may be *stronger* in subtypes of `C`.
B) The pre-condition for a method `C.f()` may be *weaker* in subtypes of `C`.
C) The pre-condition for a method `C.f()` must be *identical* in subtypes of `C`.

(vi)
A) The post-condition for a method `C.f()` may be *stronger* in subtypes of `C`.
B) The post-condition for a method `C.f()` may be *weaker* in subtypes of `C`.
C) The post-condition for a method `C.f()` must be *identical* in subtypes of `C`.

(b) Your company is developing software for use inside *pedestrian crossings*. A pedestrian crossing has a *walk* light and a *don't walk* light:

Either the *walk* light is on, or the *don't walk* light is on. They cannot be both on or both off at the same time.

The following Java code implements a simple pedestrian crossing:

```
class PedestrianCrossing {
 private boolean walk;
 private boolean dontWalk;

 public PedestrianCrossing(boolean _walk, boolean _dontWalk) {
  walk=_walk; dontWalk=_dontWalk;
 }

 public void change() {
  if(walk) { walk=false; dontWalk=true; }
  else { walk=true; dontWalk=false; }
 }

 public void copy(PedestrianCrossing p) {
  walk = p.walk; dontWalk = p.dontWalk;
 }

 public void setWalk(boolean w) {
  if(w) { walk = true; dontWalk = false; }
  else { walk = false; dontWalk = true; }
 }

 public boolean walk() { return walk; }
}
```

(i) [3 marks] From the description of a pedestrian crossing, give the *class invariant* that the above code should maintain.

(ii) [5 marks] Why is the class invariant not properly enforced by `PedestrianCrossing`? Suggest a way to fix the problem.

(iii) [3 marks] Suggest an appropriate *pre-condition* for the `copy` method.

A *special* pedestrian crossing allows the *walk light* to flash to indicate the crossing's signal is about to change from *walk* to *don't walk*.

```
class SpecialCrossing extends PedestrianCrossing {
 // @invariant walk() || (!walk() && !flashing)
 private boolean flashing = false;

 SpecialCrossing(boolean walk) { super(walk,!walk); }

 public void setWalk(boolean walk) {
  super.setWalk(walk);
  if(!walk) { flashing = false; }
 }
 public void setFlashing() {
  if(walk()) { flashing=true; }
}}
```

(iv) [5 marks] Is the class invariant of `SpecialCrossing` properly maintained? Justify your answer.

(v) [4 marks] Methods can be divided into *mutators* and *accessors*. Suggest a general rule for dealing with inheritance which, if obeyed, will ensure a subclass's invariant is always true.

(vi) [4 marks] Discuss whether or not you believe `SpecialCrossing` is a proper subtype of `PedestrianCrossing`.

****

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.

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.