School of Computer Science and Information Technology
RMIT University

COSC1095/COSC1295
Programming Principles 2J/Java for Programmers

First Semester 2006 - Mock Exam

Student's Number: _____

Student's Family Name: _____

Student's Given Name: _____

Date: TBC

Time: TBC

Duration: 3 hours

Reading Time: 15 minutes

Number of Pages (including this page): 21

INSTRUCTIONS - PLEASE READ CAREFULLY

- Contribution of this examination to assessment: 60%.
- Total marks for this examination is 100.
- You have to achieve at least 50 marks on this examination to pass the course.
- This is a closed book examination. It consists of:
    - Part A: 20 multiple-choice questions: 20 x 1 = 20 marks
    - Part B: 3 short questions, 3 x 10 = 30 marks
    - Part C: 2 medium-long programming questions 15 + 35 = 50 marks
- As a rough guide, you should spend approximately 1.8 minutes per mark
- Use blue or black ink pen to answer all questions.
- All answers should be written on this examination paper. You can use the back of the sheets if you need more spaces for your answer. Clearly state the question number. Cross out any mistakes made.
- No books or reference materials are allowed.
- Electronic devices are not allowed.
- Selected AWT components and methods are in the Appendix of this examination paper (page 21)

Part A: Multiple choice (20 x 1 =20 marks)

Circle the appropriate answer in the question when answering the multiple-choice questions. If you make a mistake or change your mind, clearly cross out your choice and circle the new choice.

1. In a do-while loop

```
do  {
   statement
}
while(condition);
```

which one of the following statements is false:

A. statement is executed at least once
B. if condition is false the loop terminates
C. the loop may be skipped completely
D. if condition is true, statement is executed again

2. How many '*' will the following program output to the screen:

```
int n = 5;
for (int i = 1; i <= n; i++)  {
   System.out.print(" * ");
   for (int j = 1; j <= n; j++)  {
      if (j > 2) continue; {
         System.out.println(" * ");
      }
   }
}
```

A. 5
B. 10
C. 15
D. 20
E. 25
F. 30

3. Select the false statement from the following: a class may have more than one:

A. instance variable
B. constructor
C. accessor
D. mutator
E. superclass

F. subclass

4. Which one of the following statements about a constructor is false:

A. A default constructor will be used if the class has no constructor
B. A constructor can initialize the instance variables of any object only once
C. A constructor can call another constructor of the same class
D. A constructor can return a value

5. Select the false statements from the following: within a class:

A. A private method may call another private method
B. A private method may call a public method
C. A public method may call a private method
D. A public method may call another public method
E. A static method may call a non-static method
F. A non-static method may call a static method

6. *StringBuffer* is a class similar to *String* but its content and length can be changed dynamically using the *append()* method. What will be the output of the program below?

```
import java.io.*;
public class PrintNames{
   public static void main (String[] args) {
      StringBuffer names[] = new StringBuffer[3];
      names[0] = new StringBuffer("Tom");
      names[1] = names[0];
      names[2] = names[0];
      for (int i=0; i<3; i++)
         names[i].append("aa");
      for (int i=0; i<3; i++)
         System.out.print(" " + names[i]);
   }
}
```

A. Tomaa Tomaaaa Tomaaaaaa
B. Tomaaaaaa Tomaaaaaa Tomaaaaaa
C. Tomaaaa Tomaaaa Tomaaaa
D. Tomaa Tomaa Tomaa

7. Any exceptions that are thrown in a method and NOT caught:

A. must be ignored
B. must be declared as part of the signature of the main method
C. must never occur
D. must be declared as part of the method's signature
E. none of the above.

8. What will be the output of the program below.

```
public class PassingIntArray{
   public static void main (String[] args){
      int n[] = {10,20,30};
      addOne(n);
      System.out.println("Values are "+ n[0] +
      ","+n[1]+","+n[2]);
   }
   public static void addOne(int x[]) {
      x[0]++;
      x[1]++;
      x[2]++;
   }
}
```

A. Values are 10,20,30
B. Values are 11,21,31
C. Values are 11,22,33
D. Values are 11,11,11

9. If class *B* extends class *A*, and class *C* extends class *B,* then which one of the following statements is false?

A. an instance of class A may use all public methods of class A
B. an instance of class A may use all public methods of class B
C. an instance of class C may use all public methods of class A
D. an instance of class C may use all public methods of class B

10. Which of the constructors below cannot exist together in the same class with the following constructor

```
public Account(String accountName, double amount) { ...}
```

A. `public Account(double amount, String accountName)  { ...}`

B. `public Account(double amount)  { ... }`
C. `public Account(String accountName)  { ... }`
D. `public Account(String ID, double amount) { ... }`

11. Which one of the following methods will not be legal in the subclass?

```
import java.io.*;
public class A {
   public double method1(double x, double y) {
      return 1.0;
   }
}
class B extends A {
   //subclass method to be placed after this line
}
```

A. `public int method1(double x, double y) { ... }`
B. `public double method1(int x, int y) { .... }`
C. `public double method1(double x, double y) { ... }`
D. `public int method1(double x, double y, double z) { ... }`

12. Assume that *Salesman* is a subclass of *Employee* (both with default constructors). In the code below, *Employee* reference *e* is referring to a *Salesman* object. Which of the statements below attempting to assign *e* to a *Salesman* references are valid?

```
Employee e = new Salesman();
Salesman s;
.....  // statement to assign e to s
```

A. `s = e;`
B. `s = (Salesman) e;`
C. `s = (Employee) e;`

13. Assume that *CAccount* and *SAccount* are subclasses of *Account*, both of which have overridden the *deposit()* and *withdraw()* methods. Which one or more of the statements below is true regarding the transfer method?

```
public boolean transfer(Account from, Account to,
double amount) {
   if (from.withdraw(amount) == true) ) {
      to.deposit(amount);
      return true;
   }
   return false;
```

```
}
```

A. It will always call the *withdraw()* and *deposit()* methods from the *Account* class
B. It will always call the *withdraw()* and *deposit()* methods of one of the subclasses of *Account*
C. It will always call the *deposit()* of *Account* and *withdraw()* of *CAccount*
D. The actual method called cannot be determined at compile-time as it depends on the type of objects being passed

14. In the program below, class *A* has two methods *meth1()* and *meth2()*. The method *meth2()* is overridden in the subclass *B*. The subclass *B* has another method *meth3()*. Which one of the statements in the main method of the Test class will result in a compilation error?

```
public class Test  {
   public static void main(String[] args) {
      A a = new B();
      a.meth1();
      a.meth2();
      a.meth3();
   }
}
class A {
   public void meth1() {    }
   public void meth2() {    }
}

class B extends A {
   public void meth2() {    }
   public void meth3() {    }
}
```

A. a.meth1();
B. a.meth2();
C. a.meth3();
D. None of the above

15. Which one of the statements below attempting to override *method1()* in subclass *B* is not valid ?

```
class A {
   void method1(int amount) {
   }
}
class B extends A {
   // method1 to be overridden here
}
```

A. `private void method1(int amount) { }`
B. `void method1(int amount) { }`
C. `protected void method1(int amount) { }`
D. `public void method1(int amount) { }`

16. Which one of the following method is not a static method?

A. *readLine()* of *BufferedReader* class
B. *sqrt()* method of *Math* class
C. *parseInt()* method of *Integer* class
D. *parseDouble()* method of *Double* class

17. Which one of the following statements in the *addOne* method will give a compilation error?

```
public class Test  {
   public static void main(String[] args) {
      A a = new A();
      a.addOne(5);
   }
}
class A {
   public static void addOne(int x)      {
      int y=0;
      x++;
      y++;
      i++;
      j++;
   }
   private static int i;
   private int j;
}
```

A. x++;
B. y++;
C. i++;
D. j++;

18. The primary interface/interfaces for a collection class is/are:

A. Collection<T>
B. Set<T>
C. List<T>

D. All of the above

19. A _____ is an object that is a simple container class that groups other objects.

A. JPanel
B. JMenu
C. JButton
D. JTextArea

20. Threads execute in a _____ fashion.

A. parallel
B. serial
C. complex
D. none of the above

Part B: Short Questions (30 marks)

1. (10 marks)

Assume that you are hired as a consultant during your vacation to help implement the GST computation for a small computer shop. Currently there are two classes, one for *Product* and one for *Service*. These classes compute the charges taking into account various discounts offered for each transaction. However they lack the feature to compute the GST.

Assume that the *Product* class has a method *getTotalPrice()* and the *Service* class has a method *getServiceCharge()*. To avoid changing or adding any code to these classes, we want to make use of an interface and use inheritance to calculate the GST. Your job is to complete the following classes to help calculate the GST due for both products and services.

2. (10 marks = 7.5 + 2.5)

JDBC:

A. Draw a diagram and explain how JDBC establishes a connection between your code and a database.
B. What is a connection string? Give an example.

3. (10 marks)

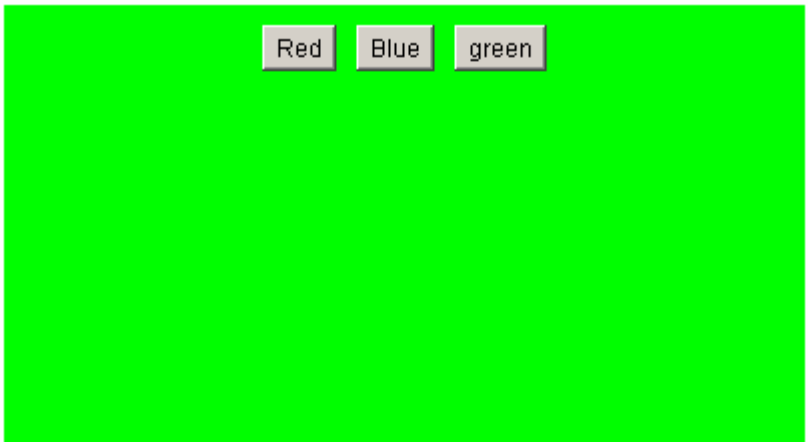
The *LinkedList* class shown below is a simple implementation of a generic linked list. This class can be applied to a list of bank accounts containing a unique account ID. The *Identifiable* interface contains the *getID()* method, which is to be implemented in subclasses (eg *SAccount* class) of the *Account* superclass covered in the lectures.

Complete the *find (String id)* method in the LinkedList class. This method can be used to find a node with a given ID in a list of bank accounts. It will return *true* if found and null if the ID is not in the list.

```
public interface Identifiable { public abstract String getID(); }

class Node {  //A list node
   Identifiable data;
   Node next ;
   public Node (Identifiable o) { // constructor
      data = o;
      next = null; // initially points nowhere
   }
}

public class LinkedList {
    Node head ;
    public LinkedList() { // constructor
        head = null;
    }
    //other methods from another type of list can be found here

    public Node find (String id) { // to be completed
```

Part C: Complete Programs (15 + 35 = 50 marks)

1. (5 + 10 = 15 marks)



Complete the classes below to produce the applet as shown above. When the user clicks on one of the buttons the background colour of the applet changes accordingly. Override the FlowLayout to place the buttons at the center of the applet.

Selected AWT components and Applet API are listed in the Appendix of this examination paper.

(a) Complete the *HandleButton* class

```
import java.awt.*;
import java.awt.event.*;
public class HandleButton implements ActionListener {
      //to be completed
```

(b) Complete the *ButtonTest* class

```
import java.applet.Applet;
import java.awt.*;
public class ButtonTest extends Applet {
      //to be completed
```

2. A simple program to manage Vehicles

Overview

Write a program to manage the rental vehicles owned by **BestCars**. The cost is based on the number of days rented. For premium vehicles, there is an additional charge based on distance covered (reading taken from odometer, which measures the number of kilometres travelled).

In the first part write a class named **Vehicle** with methods for renting, returning and storing vehicle details in a file. When renting a vehicle, the name and license number of the customer as well as the time of rental must be recorded. When the vehicle is returned the charges must be computed based on the daily rate and the rental period. To assist you with computations involving date/time you may make use of the **DateTime** class below. Assume that all vehicles are identified by a unique registration number. In the second part you are required to handle premium vehicles with the additional charge based on distance by deriving the class **PremiumVehicle**. In the last part create a simple menu driven program to rent and return vehicles using the **Vehicle** and **PremiumVehicle** classes. On exiting the menu the state of these objects must be written to the specified text file in the format shown in part (C).

The main intention of this question is to allow you to demonstrate your understanding of the concepts: encapsulation, inheritance, polymorphism, data structures, exception handling and file writing. In view of the limited time available most of the class design is done for you and you are not required to read from files. In case where the specification is subject to interpretation, you may make any reasonable assumptions but you are required to justify them in the form of comments.

Helper class DateTime

The **DateTime** class has two constructors. The first one which has no arguments, initialises the object to current date/time. The second one takes a String argument in the form "day-month-year-hour-min", with hours ranging from 0 to 23 (12-23 being PM). It provides an accessor to return date/time as a String in the same format as above. It has another method daysSince() to compute the number of days between two DateTime objects. Note this method returns 0 if the two date/times are less than 24 hours apart.

```
class DateTime
{  public DateTime()  { … }   // set it to current data and time
   public DateTime(String s) {…} // s in the form "day-month-year-hr-min"
   public String getDateTime(){…}// returned as "day-month-year-hour-min"
   public int daysSince(DateTime dueDate){…} //returns difference in days
   …
}

// using DateTime class
DateTime d1 = new DateTime();
// d1 represents the current date and time say 29th May, 9.00 pm
DateTime d2 = new Date("27-4-2006-20-15");
// represents 27th April 2006, 8.15 pm
int days = d1.daysSince(d2);
// days set to 2 as d1 and d2 are between 48 and 72 hours apart
```

(a) Superclass Vehicle Requirements (15 marks)

Write a class named **Vehicle**, to model the behaviour of vehicles being rented. Each vehicle has a unique **ID** (registration number) and a **daily rental rate**. A vehicle can be rented if the current **status** of vehicle is available (**'a'**) and returned back if the status is rented out (**'o'**). The person hiring the vehicle must supply his **name** and a valid **license number**, which is then stored in the vehicle object being rented together with the **date** and **time** of **rental**. Thus the data to be maintained for all vehicles includes **ID** (String), **rate** (double), **status** (either **'a'** or **' o'**), **rental date/time** (a **DateTime** object) **name** of renter (String) and **license number** of renter (String). Two constructors should be provided, one for the vehicles currently available, and one for those on loan. Methods should be provided for renting a vehicle, returning a vehicle, writing the details to a text file as well as appropriate accessors.

(i) Declare the instance variables identified above. You may use a **DateTime** object for storing **date/time** of **rental**. Please refer to the **DateTime** class in the previous page. **(2 marks)**

(ii) Provide appropriate accessors for **ID** and **status**. **(2 marks)**

(iii) Provide two constructors, first one for vehicles currently available and second one for vehicle currently rented out. Both constructors must take as arguments, vehicle **ID**, **daily rental rate** and **status**. The second constructor must take 3 additional arguments, **renters name**, **license number** and the **date/time of rental**. The date/time should be passed as a String in the format "day-mon-year-hr-min", from which a **DateTime** object can be constructed. **(2 marks)**

(iv) Complete the method below to rent a vehicle, passing name and license number of the renter. If an attempt is made to rent a vehicle that is already rented out an **Exception** must be thrown with the error message **"Cannot rent – vehicle with ID xxxx – already on loan"**. Note the current date/time can be obtained using the DateTime default constructor (constructor with no arguments). If successfully rented out the status must be changed to **' o'**.

```
public void rent(String name, String license)throws Exception{ …
```

(3 marks)

(v) Complete the method below to return back a rented vehicle. If an attempt is made to return a vehicle that is not rented out an **Exception** must be thrown with the error message **"Cannot return – vehicle with ID xxxx is not on loan"**, otherwise this method should reset the status of this object (to **'a'**) and compute and return the cost based on the number of days elapsed since **rental date** and the **daily rental-rate**. **DateTime** class provides a convenient method **int daysSince(DateTime)** to find the difference in days.

```
public double returnBack()throws Exception { …
```

(3 marks)

(vi) Complete the method below to write the vehicle object details to the **PrintWriter** object (associated with a text file). Separate all the fields with a separator colon (":"). For items on loan name, license number of the renter and date of rental must also be written. Refer to section (c) part(iii) for the format of the output file.

```
public void write(PrintWriter pw) { …
```

(3 marks)

(b) Subclass PremiumVehicle Requirements (8 marks)

In this part you are required to extend the class **Vehicle** to **PremiumVehicle** representing premium vehicles for which there is an additional distance based charge (distance derived from odometer readings before and after rental). Hence this class must have an additional instance variable for storing **odometer reading**, which must be updated whenever a vehicle is returned after rental. You may assume these vehicles are not used for any other purpose (such as private use). Two constructors should be provided (as for the **Vehicle** class) one for the premium vehicles currently available, and one for those on loan. Both constructors must take the odometer reading as one of the arguments. This class should provide a **returnBack**() method taking one argument, the new **odometer reading** (overloading the **returnBack()** method of the superclass), thus allowing the additional charge to be computed based on 0.10 dollars per kilometre. For example if the car has done 4500 kilometres during the rental period there must be an additional charge of $450 (4500 * 0.10).

(i) Derive a new class named **PremiumVehicle** with one additional instance variable for odometer reading. **(2 marks)**

(ii) Provide two constructors for this class identical to the **Vehicle** constructors except for the additional argument for odometer reading. **(2 marks)**

(iii) Complete the **returnBack()** method below with one int argument for odometer reading. The charges for premium vehicles must be based on the distance covered (rate of $0.10/km) as well as the number of days rented out.

```
public double returnBack(int currentOdometer)throws Exception {
```

(2 marks)

(iv) Override the **write** method to write the value for odometer reading followed by a delimiter ":", before calling the superclass **write** method to write the remaining data values. Hence the format of the text file should be as shown in section (C). **(2 marks)**

```
class PremiumVehicle { …
```

(c) Writing a Simple Menu Driven Application (12 marks)

This part requires you to create a few **Vehicle** and **PremiumVehicle** objects, manipulate them using a menu, and write their final details to a text file in the format shown below.

(i) Create an array of 4 **Vehicle** references and make these references refer to the following objects, using appropriate constructors.

First: **PremiumVehicle** with ID="SAT345",daily rate=50.0, status = 'a',

odometer reading = 12,000

Second: **Vehicle** with ID="QKO162",daily rate=30.0, status = 'o'

name of renter = "Charles Finney", License Number = "056186145"

date Rented = 12th March 2004, 11.15 AM

(Date must be passed as String in the form "day-month-year-hour-min")

Third: **Vehicle** with ID="PRT234" daily rate=35.0, status 'a',

Fourth: **PremiumVehicle** with ID="SYT234" daily rate=55.0, status 'a',

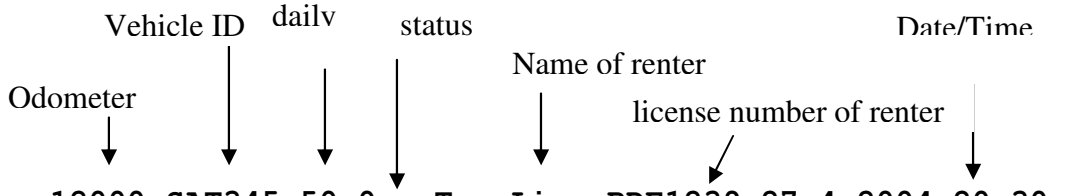
odometer reading = 15,000 **(2 marks)**

(ii) Use a simple menu with the following options to rent and return these **Vehicle** and **PremiumVehicle** objects by specifying their ID and other relevant details. Note when renting an item the user must supply the name and license number, and when returning a **PremiumVehicle** user must supply the new odometer reading. When an item is returned the total charge must be displayed. The code must catch the Exception object raised by **rent()** and **returnBack()**, displaying those error messages.

Rent **1**
Return **2**
Exit **3** **(8 marks)**

(iii) Create a PrintWriter object allowing us to write to the text file named **"Vehicles.dat"**, writing all the object details to this file in the format shown below. Note that, only premium vehicles have odometer reading as the first data item.

(2 marks)




```
12000:SAT345:50.0:o:Teo Ling:BDE1239:27-4-2004-20-30
QKO162:30.0:o:Charles Finney:056186145:12-3-2004-11-15
PRT234:35.0:a
15000:SYT234:55.0:a
```

Format of the text file Vehicles.dat

Appendix: Selected AWT component methods

Component *(An abstract class)*	
Method	**Description**
`String getName()`	Returns the name of the `Component`.
`void setVisible(boolean)`	Shows or hides the `Component` depending on the value of the input parameter.
`boolean isVisible()`	Returns `true` if the `Component` is visible, otherwise returns `false`.
`void setForeground(Color)`	Sets the foreground colour of the `Component` to the value of the input parameter.
`void setBackground(Color)`	Sets the background colour of the `Component` to the value of the input parameter.
`void setSize(int, int)`	Sets the width of the `Component` to the value of the first parameter and the height to that of the second.
`void paint(Graphics)`	Paints the `Component`; called when the `Component` first becomes visible.
`void repaint()`	Repaints the `Component`.
`void addMouseListener(MouseListener)`	Accepts a `MouseListener` as a parameter and adds this to the `Component`.
`void addMouseMotionListener( MouseMotionListener)`	Accepts a MouseMotionListener as a parameter and adds this to the `Component`.

Button *(Inherits all the methods of `Component`)*	
Method	**Description**
`Button()`	Constructs a `Button` with no label.
`Button(String)`	Constructs a `Button` with a label specified by the input parameter.
`void setLabel(String)`	Sets the `Button`'s label to the value of the input parameter.
`String getLabel(void)`	Returns the `Button`'s label.
`void addActionListener(ActionListener)`	Accepts an `ActionListener` as a parameter and adds this to the `Button`.

Applet (*Inherits all the methods of `Panel` (and `Container` and `Component`))*	
Method	**Description**
`Applet()`	Constructs a new `Applet`.
`void resize(int, int)`	Resizes the applet, assigning a width and height specified be the first and second parameters respectively.
`void init()`	Invoked the first time the applet is loaded (or reloaded) by a browser. The implementation provided does nothing; a subclass of `Applet` must provide the code as required.
`void start()`	Invoked after `init` when the applet is first loaded (or reloaded) and then invoked each time the applet is made visible again by returning to the page. The implementation provided does nothing; a subclass of `Applet` must provide the code as required.
`void stop()`	Invoked when the applet is hidden (by pointing the browser at a different page). The implementation provided does nothing; a subclass of `Applet` must provide the code as required.
`void destroy()`	Invoked after `stop` when the applet is abandoned (by closing the browser). The implementation provided does nothing; a subclass of `Applet` must provide the code as required.