Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 1 (Compulsory)

(a) In a Java technology program, briefly describe the following:

(i) the package statement [1]
(ii) the import statement [1]
(iii) classes, methods, and attributes [3]
(iv) constructors [1]

(i) specifies the package declaration at the beginning of the source file

(ii) specifies the class to which you want access
or
used to bring classes in other packages into the current namespace

(iii) class = a template or type definition for creating object method = a functional or behavioral element of an object

(iv) attribute = a data element of an object a special method used to initialize a new object upon its creation

- **award 1 mark for correct explanation on the package statement**
- **award 1 mark for correct explanation on the import statement**
- **award 1 mark each for correct explanation on classes, methods, and attributes**
- **award 1 mark for correct explanation on the constructors**

Accept alternative correct explanation.

(b) One form of polymorphism in Java is overloading.

(i) Explain, using an appropriate example, the term method overloading. [3]

(ii) State another form of polymorphism found in Java. [1]

(i) Overloading is the use of the same method name [1] for arguments of different types [1]

Example:
```
public void println( int a )
public void println( float f )
public void println( String s, double d )
```

- **award 1 mark if candidates mentioned that the methods have the same name**
- **award 1 mark if candidates mentioned that these methods take in different types or different number of arguments**
- **award 1 mark for an appropriate example**

(ii) Overriding

- **award 1 mark for correct identification of the overriding as another form of polymorphism in Java**

Accept alternative correct explanation, and example for part (i).

(c) The class Employee is defined as follows:

```
public class Employee {
    private String name;
    private Date birthDate;
    private String profile;

    public String getDetails() {
      //
    }
}
```

In this class,

(i) include a constructor that takes in two arguments to initialize the name and the date of birth of an employee. The employee's profile is initialized by invoking the getDetails() method. [3]

```
public Employee ( String n, Date dob ) {
    name = n;
    birthDate = dob;
    profile = getDetails();
}
```

- **award 1 mark for correct constructor's signature**
- **award 1 mark for correct statements to initialize the name and birthdate**
- **award 1 mark for correct statement to initialize the profile**

Accept alternative correct answer.
Do not deduct mark for trivial syntactical error.

(ii) Overload the constructor so that it takes in only one argument, the name of the employee, to initialize the Employee's object by calling the constructor in part (i). [2]

```
public Employee ( String n ) {
        this ( n, null );
}
```

- **award 1 mark for correct constructor's signature**
- **award 1 mark for using this(), with appropriate arguments, to call the constructor in part (i).**

Accept alternative correct answer.
Do not deduct mark for trivial syntactical error.

(iii) implement the getDetails() method that returns the name and date of birth of an employee. [1]

```
public String getDetails() {
        return “Name : ” + name + “\nDate of Birth : ” + birthDate;
}
```

Accept alternative correct answer.
Do not deduct mark for trivial syntactical error.

(iv) Override the equals() method, with the signature given below, that tests against the employee’s name and date of birth. If the object passed into this method is not null and an object of type Employee and the objects’ contents are equal, the method returns true; otherwise, it returns false.

```
public boolean equals ( Object obj ) {
}
```

[6]

```
public boolean equals ( Object obj ) {
boolean result = false;
if (( obj != null ) && ( obj instanceof Employee )) {
Employee emp = ( Employee ) obj;
if ( name.equals(emp.name) && birthDate.equals(emp.birthDate))
result = true;
}
return result;
}
```

- **award 1 mark for correct statements to initialize result to false**
- **award 1 mark for using the instanceof operator to check whether the object passed is an Employee,**
- **award 1 mark for checking that the object is not null**
- **award 1 mark for casting the object to the Employee’s object**
- **award 1 mark for using the equals() to compare the name and date of birth**
- **award 1 mark for correct statement to return the result**

Accept alternative correct answer.
Do not deduct mark for trivial syntactical error.

(d) A secretary is an employee. Using the Employee class definition in part (c), define the class Secretary that has an instance variable called department. Include in this class a constructor to initialize an object of type Secretary. [5]

```
public class Secretary extends Employee {
    private String department;

    public Secretary( String n, Date dob, String dept ) {
        super(n, dob);
        department = dept;
    }
}
```

- **award 1 mark for using the keyword extends to inherit the Employee class**
- **award 1 mark for correct declaration of the instance variable**
- **award 1 mark for correct constructor's signature**
- **award 1 mark for using super() to call the parent class's constructor**
- **award 1 mark for correct statement to initialize the department**

Accept alternative correct answer.
Do not deduct mark for trivial syntactical error.

(e) Briefly explain why you mark individual methods as final. [3]

- **methods marked final cannot be overridden [1]**
- **this is done for security reasons [1]**
- **you make the method final if the implementation of the method should not be changed and is critical to the consistent state of the object [1]**

Accept alternative correct answer.
Do not deduct mark for trivial syntactical error.

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 2

(a) What is the default layout manager for a Frame? [1]

Sample answer: Border Layout [1]

(b) Define a class called *PhotoFrame* which implements a frame by extending the Frame superclass, and includes the ActionListener interface. Your class should contain a constructor that creates the frame with a named title bar. In addition to this, include in your declaration a protected button called *btnModel* and a protected text field called *tfModel*. [5]

```
public class PhotoFrame extends Frame implements ActionListener{ [1]
    public PhotoFrame ( String title ) {  [1]
        super( title );  [1]
    }
    protected Button btnModel; [1]
    protected TextField tfModel; [1]
}
```

- **award 1 mark for correct class heading**
- **award 1 mark for correct constructor's signature**
- **award 1 mark for correct statement calling the superclass constructor to initialize the frame**
- **award 1 mark for correct statement to declare the button**
- **award 1 mark for correct statement to declare the textfield**

Do not deduct mark for trivial syntactical error.

(c) Write a method called *myPhoto* that accepts a string argument *name*, creates the button with title *name* and the text field of size 30. The method should also add the appropriate listener to the button component and place the button and the text field in the frame. [5]

```
public void myPhoto ( String name ) {  [1]
    btnModel = new Button(name); [1]
    tfModel = new TextField (30); [1]
    btnModel.addActionListener(this); [1]
    this.add(btnModel);
    this.add ( tfModel); [1]
}
```

- **award 1 mark for correct method's signature**
- **award 1 mark for correct statement to create the button with appropriate name**
- **award 1 mark for correct statement to create the textfield with appropriate size**
- **award 1 mark to register the correct listener to the button**
- **award 1 mark for correct statements to add the components on the frame**

Do not deduct mark for trivial syntactical error.

(d) Implement an appropriate event handler routine that defines the action listener interface such that it will write the value of the button to the text field when the button is pressed. [2]

```
public void actionPerformed ( ActionEvent ae ) {  [1]
    tfModel.setText(ae.getActionCommand());  [1]
}
```

- **award 1 mark for correct method's signature**
- **award 1 mark for correct statement to write the value of the button to the textfield**

Do not deduct mark for trivial syntactical error.

(e) Give the statements necessary to ensure that the AWT packages would be correctly made available to the class defined above. [2]

```
import java.awt.*; [1]
import java.awt.event.*; [1]
```

- **award 1 mark for each correct statement**

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 3

(a) Create a user-defined exception called *DialUpException* that has as data member a String called *reason*. Ensure that access to this data member is restricted to only class members. [2]

```
public class DialUpException extends Exception {
        private String reason;
}
```

- **Award 1 mark for correct class signature**
- **Award 1 mark for correctly including the data member**

(b) Include a constructor that accepts one argument that initializes the data member of the class accordingly. [2]

```
public DialUpException(String reason) {
        this.reason= reason;
}
```

- **award 1 mark for correct constructor's signature**
- **award 1 mark for correct statement to initialize the data member**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

(c) Include an accessory method called *getReason* to allow access to the data member of the class. [1]

```
public String  getReason(){
        return reason;
}
```

- **award 1 mark for correct implementation of accessor method**

Do not deduct mark for trivial syntactical error

(d) Write a method called *makeConnection* that takes in a String parameter called *phoneNum* and an integer parameter called *numOfTimes* indicating the number of attempts made. The method should attempt to make a connection to the dial up number calling the *connects* method, whose signature appears below. If the method *connects* returns a –1, then it is deemed that the connection is unsuccessful and a DialUpException should be thrown with the message "Line is engaged" if it is the first attempt, and the message "No Connections available" if it is the second. If the connection was successful, then a message indicating a connection to the line number is displayed.

int connects(String telephoneNum); [5]

```
public void makeConnection(String phoneNum, int numOfTimes)
                                                    throws DialUpException{
        int success = connects(phoneNum);
        if (success = -1 ){
                if (numOfTimes == 1)
                        throw new DialUpException("Line is engaged");
                else
                        throw new DialUpException("No connections available");
        }else
                System.out.println("Connected to " + phoneNum);
}
```

- **award 1 mark for correct method signature**
- **award 1 mark for calling the connects method with the correct number of arguments**
- **award 1 mark for checking for the unsuccessful connection condition and for including the condition for the number of attempts made**
- **award 1 mark for throwing the DialUpException correctly in both attempts**
- **award 1 mark for displaying message indicating a successful connection**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

(e) What is a *try* block? [1]

- **a try block is a set of statements that might generate an exception. [1]**

Accept alternative correct explanation

(f) Does an exception have to be caught in the same place where the try block created the exception? [2]

No [1], it is possible to catch an exception anywhere in the call stack [1]

Accept alternative correct explanation

(g) If there are two *catch* statements, one for base and one for derived, which should come first? [2]

- **catch statements are examined in the order they appear in the source code [1]**
- **the first catch statement whose signature matches the exception is used [1]**

Accept alternative correct explanation

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 4

Suppose that you want to display the string "Hello World!" in an applet panel wherever the user clicks. Assume that the class Point has been defined.

(a) Define a class called PaintModel that inherits the Applet class. Include in the class,

(i) a private instance variable named lastClick of type Point. This variable is accessible only within the PaintModel class and is initialized to null. [3]

```
public class PaintModel extends Applet {
        private Point lastClick = null ;
}
```

- **award 1 mark for correct class heading**
- **award 1 mark for correct usage of private keyword**
- **award 1 mark for correct statement to assign the null value to lastClick**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

(ii) a private inner class called MyModelRecorder which extends the MouseAdapter class. Implement the mousePressed() method that takes a MouseEvent object as its argument, initializes lastClick to the object's value by calling the getPoint() method, and calls the repaint() method. [4]

```
private class MyModelRecorder extends MouseAdapter {
    public class mousePressed( MouseEvent me ) {
        lastClick = me.getPoint();
        repaint();
    }
}
```

- **award 1 mark for correct class header**
- **award 1 mark for correct method's signature**
- **award 1 mark for correct statement to assign the value**
- **award 1 mark for calling the repaint() method**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

(iii) a public init() method that registers the MouseListener to the applet panel. The add listener method takes in the MyModelRecorder object as its argument. [3]

```
public void init () {
    addMouseListener ( new MyModelRecorder() );
}
```

- **award 1 mark for correct method's signature**
- **award 1 mark for registering the listener to the Applet's panel**
- **award 1 mark for passing the MyModelRecorder object into the add method**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

(b) Override the update() method so that the screen is not cleared before calling the paint() method. [2]

```
public void update ( Graphics g ) {
    paint(g);
}
```

- **award 1 mark for correct method's signature**
- **award 1 mark for calling the paint() method correctly**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

(c) Write an HTML tag that specifies the code that the appletviewer loads for the program in part (a). [3]

```
<applet code=”PaintModel.class” width=200 height=200>
</applet>
```

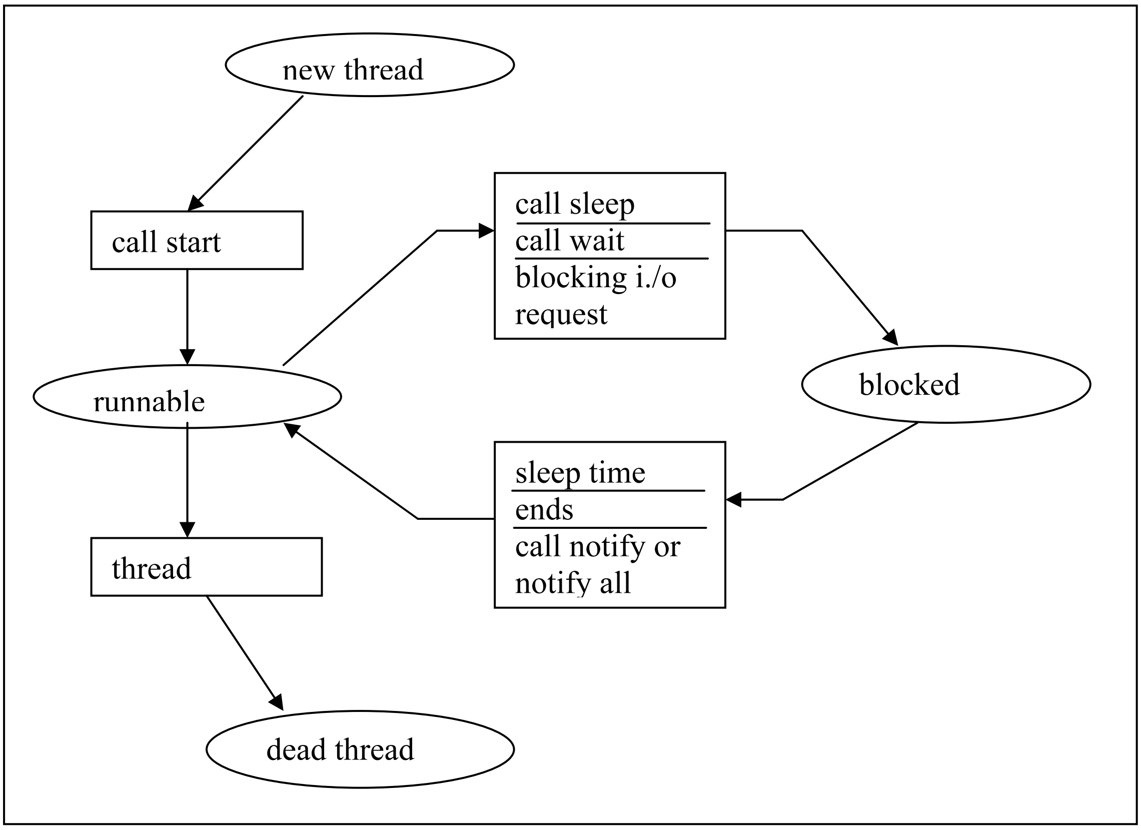
- **award 1 mark for correct applet code**
- **award 1 mark for correct statement in setting the applet's size**
- **award 1 mark for correct applet tag**

Do not deduct mark for trivial syntactical error
Accept alternative correct answer

Do not award half marks.
In all cases give credit for appropriate alternative answers.

Question 5

The diagram below shows how a thread can change state during its life cycle:



(a) Based on this diagram, after you created a thread, you make it active by calling the start method. The JVM then passes control to the appropriate run method. After a thread starts, it can be in a runnable or blocked state.

(i) What do you understand about a thread in a runnable state? [2]

- **it is either executing [1] or ready to execute as soon as the JVM gives it control [1]**
- **award 1 mark for each situation above**

(ii) Under what situation(s) a thread is in a blocked state? [2]

- **the thread can be blocked as a result of a call to sleep or wait [1] or because it is waiting for an I/O operation [1]**
- **award 1 mark for each situation above**

(b) You can take two approaches in creating and running a thread, depending on whether you choose to implement the Runnable interface or extend the Thread class. Briefly describe the steps to create and run an object that is Runnable, but not a Thread. [5]

4 steps:

(i) define a class that implements Runnable
(ii) implement the method run for your class
(iii) instantiate the class to create a Runnable object
(iv) create a Thread object, passing the Runnable object as an argument to the constructor Thread
(v) call the start method for the Thread object. Do not call the run method

- **award 1 mark for each correct step identified above**

(c) Identify three methods of the Thread class that are deprecated in the Java 2 platform. [3]

stop, suspend, resume

- **award 1 mark for each correct identification**

(d) Why are the methods in part (d) deprecated? [3]

- **stop & suspend are inherently unsafe [1] because they can make the stopped thread leave data in an inconsistent state [1]**
- **resume is deprecated because without suspend, you never need to call it**
-
- **award 2 mark for explaining the reason for stop & suspend being deprecated**
- **award 1 mark for explaining the reason for resume being deprecated**

Accept relevant correct explanation.

- END OF PAPER -