



**1. True/False [ 15 pts ]**

Answer the following questions True or False. Be sure to write out True and False, NOT just a T or F.

- 3** (a) Java provides polymorphism either through inheritance or through the use of interfaces.
- 3** (b) If a subclass object is used where a superclass object is expected, it will behave exactly as if it were a superclass object.
- 3** (c) The `NullPointerException` is a checked exception.
- 3** (d) The finally clause of a try-catch block is only executed if none of the catch blocks execute.
- 3** (e) Infinite recursion in Java (e.g., `void foo(){ foo(); }`) will make a program run forever.

## 2. Polymorphism and Dynamic Binding [15 pts ]

Consider the classes below.

```
public abstract class Money {
    public abstract int value();
}

public abstract class Coin extends Money {
    public abstract int value();
}

public class Nickel extends Coin {
    public int value() { return 5; }
}

public class Penny extends Coin {
    public int value() { return 1; }
}

... // definitions of Dime, Quarter

public abstract class Bill extends Money {
    public abstract int value();
}

public class Dollar extends Bill {
    int value() { return 100; }
}

... // definitions of TwoDollar, FiveDollar, TenDollar, etc.
```

On the next page, write a class `Change` that will contain an array of `Bills` and an array of `Coins`. A `Change` object should also be a `Money` object. A `Change` object's value should be the sum of the values of all coins and bills.

Create class Change below.

### 3. Polymorphism and Dynamic Binding [ 26 pts ]

Study the following code carefully. Note the different types of classes and interfaces.

```
public interface Vehicle {
    public void transport();
}

public abstract class Car {
    protected double mpg;
    public abstract void drive();
    public void repair() { System.out.println("Car's repair"); }
}

public class Toyota extends Car {
    public void drive() { System.out.println("Toyota's drive"); }
}

public class Avalon extends Car {
    public void drive() { System.out.println("Avalon's drive"); }
    public void zoom() { System.out.println("Avalon's zoom"); }
}

public class Camry extends Toyota implements Vehicle {
    public void transport() { System.out.println("Camry's transport"); }
    public void washMe() { System.out.println("Camry's washMe"); }
}
```

For each of the following code segments in parts (a) through (m), decide whether they will compile and run without error. You should answer in one of three ways:

- If the line(s) will compile and run without error, then write OK and print out what will be printed (or “No output”) when the code runs.
- If the line(s) will fail to compile, write COMPILE ERROR and give a brief, one-sentence reason why.
- If the line(s) will compile without error but will generate a run-time error, write RUNTIME ERROR and give a brief reason why.

That is, for each of the three options, you need to do two things. In order to get the two points of credit per item, you need to have both things right. Only getting the result right (e.g., OK, COMPILE ERROR, RUNTIME ERROR) will receive 0 points. You must also have the code output or error explanation correct to receive credit.

- [2] (a) `Car c1 = new Camry();`
- [2] (b) `Avalon a1 = new Car();`
- [2] (c) `Car c2 = new Car();`
- [2] (d) `Car c3 = new Camry();`  
`c3.drive();`
- [2] (e) `Toyota t1 = new Toyota();`  
`t1.repair();`
- [2] (f) `Object o1 = new Avalon();`  
`o1.zoom();`
- [2] (g) `Object o2 = new Avalon();`  
`((Avalon)o2).zoom();`

- 2 (h) Object o3 = new Camry();  
((Car)o3).repair();
- 2 (i) Car c4 = new Camry();  
((Avalon)c4).zoom();
- 2 (j) Vehicle v1 = new Vehicle();  
v1.transport();
- 2 (k) Vehicle v2 = new Avalon();  
v2.transport();
- 2 (l) Vehicle v3 = new Camry();  
v3.transport();
- 2 (m) Vehicle v4 = new Camry();  
v4.washMe();

**4. Handling Exceptions [ 12 pts ]**

Add code to the method below so that it handles exceptions potentially occurring because of division by zero or by accessing an invalid array index. The two exception classes are, respectively, `ArithmeticException` and `IndexOutOfBoundsException`. To handle each, just print out the type of exception that is occurring. Do not change any of the existing code.

```
public void worker(int index) {
```

```
    int[] arr = { 0, 10, 20, 30, 40 };  
    int val = arr[index];  
    val = 100 / index;
```

```
}
```

## 5. GUIs [ 20 pts ]

The class `ReboundPanel`, on the next page, should display a panel with a ball bouncing around inside. The ball image is in a file “ball.gif” (size 20 x 20). Assume that `ReboundPanel` is used in a correct program and can be found in a file with all the right imports. Add appropriate code to the constructor and `paintComponent` methods and the `ReboundListener` class and `actionPerformed` method so that the ball begins moving and keeps moving until it reaches a side of the panel and then rebounds, reversing its X and Y direction. You should use a timer to assist and you will need to add appropriate instance variables as necessary. Every time the timer fires, you should update the ball’s position. (Just choose some default velocity.) You can assume the panel size will always be 300x100. Don’t worry about the size of the image—just check that its upper left corner is inside the panel.

The following java API calls may assist you:

```
class Timer
    public Timer(int delay, ActionListener al)
    public void start()
    public void stop()

class ImageIcon
    public ImageIcon(String filename)
    public void paintIcon(Component c, Graphics g, int x, int y)
    // Note that you can use null for the Component parameter

class JPanel
    public void setPreferredSize(Dimension d)
    public void paintComponent(Graphics g)
    public void repaint();

interface ActionListener
    public void actionPerformed(ActionEvent e)
```

```
public class ReboundPanel extends JPanel
{

    public ReboundPanel()
    {

    } // end constructor ReboundPanel

    public void paintComponent (Graphics page)
    {

    } // end method paintComponent

    private class ReboundListener implements ActionListener
    {

        public void actionPerformed (ActionEvent event)
        {

        } // end actionPerformed
    } // end class ReboundListener
} // end class ReboundPanel
```

**6. Recursion [10 pts ]**

Write a method, using recursion, called `fib3` that is a 3-value Fibonacci sequence. By this we mean that, for each element in the sequence from the fourth element on, the element's value is the sum of the three previous values in the sequence. The values of the first second and third elements are all 1. Thus, the sequence begins

1, 1, 1, 3, 5, 9, 17, 31, 57, 105, ...

That is, `fib3(1) = 1`, `fib3(2) = 1`, and so on. Your method should take an integer parameter that is the sequence index position being calculated. It returns the integer value of that element in the sequence. Whenever your method receives a parameter (index) of zero or below, the method should return the value 0. You must use recursion to receive credit.

**7. EXTRA CREDIT [5 pts ]**

Consider the two classes below.

```
public class Parent {
    public int num;

    public Parent() {
        this(4);
        System.out.println("Parent-default");
    }

    public Parent(int n) {
        num = n;
        System.out.println("Parent-int");
    }

    public void doIt(String s) {
        System.out.println(s + num);
    }
} // end of class Parent

//-----

public class Child extends Parent {
    public Child(int n) {
        System.out.println("Child-int");
    }

    public void doIt(String s) {
        System.out.println("Child " + s + " " + num);
    }

    public static void main(String[] args) {
        Parent ar[] = new Parent[2];
        ar[0] = new Child(2);
        ar[1] = new Parent(1);
        for (int i = 0; i < 2; i++ ) {
            ar[i].doIt("index " + i + ": ");
        }
    }
} // end of class Child
```

What is the output when the program above is run?