**COMP 132 - Homework # 8**

**Inheritance and Polymorphism**

1. This question builds upon the TextMessage example from class and contained in the sample code provided on the Detailed Schedule and Resources page. You will define a new class named PaymentMessage that represents a new type of text message that can be sent to a vendor to make a payment. This new type of message has text but also has a cash value.

a. Give an implementation of the PaymentMessage class. Because a PaymentMessage is everything a TextMessage is, plus some other stuff, it should be a subclass of TextMessage. In addition to being a subclass of TextMessage, the PaymentMessage must:

i. have a field to keep track of the amount of the payment

ii. have an appropriate constructor

iii. have an accessor for the amount of the payment.

iv. not implement any other methods (yet).

b. List all of the methods that can be invoked on a PaymentMessage object as defined in part a. Remember a subclass inherits methods from its superclass (which also inherits from its superclass!).

c. Consider the following snippet of code that uses a PaymentMessage object as defined in part a:

PaymentMessage pm = **new** PaymentMessage(

7173456789L, 71798765432L, "Here ya go", 22.75);

System.*out*.println("pm.getMessageType(): " +

pm.getMessageType());

When executed this code would produce the output:

pm.getMessageType(): Text Message

Which is clearly not the desired output. Explain as clearly and fully as you can why this snippet generates this output.

d. Give the code that you would add to the PaymentMessage class to cause the snippet of code in part c produce the output:

pm.getMessageType(): Payment Message

e. Currently the length of a PaymentMessage would simply be the length of the text that it contains. However, storing the payment requires some space as well. Let us assume that the size of a PaymentMessage should be the length of its text plus 8 units of storage for the payment information. (The usual encoding of a double-precision floating point number requires eight bytes, but we do not study those details here.) Give the code that you would add to the PaymentMessage class so that it computes the correct length.

2. The questions below make use of the following three classes:

**class** Ecks {

**private** **int** x;

**public** Ecks(**int** a) {

x = a;

}

**public** **int** bar() {

**return** x + 1;

}

**public** **int** foo() {

**int** b = bar();

**return** x\*b;

}

}

**class** Why **extends** Ecks {

**private** **int** y;

**public** Why(**int** b) {

**super**(7);

y = b;

}

**public** **int** bar() {

**int** c=**super**.bar();

**return** c + y;

}

**public** **int** qux() {

**return** y + 3;

}

}

**class** Zee **extends** Why {

**private** **int** z;

**public** Zee() {

**super**(5);

z = 3;

}

**public** **int** bar() {

**int** d = qux();

**return** z \* d;

}

}

Assume that the following statements are executed before each of the questions below (i.e. the parts below do not build upon each other, each is started with a fresh set of objects and references).

Ecks xx = **new** Ecks(2);

Why yy = **new** Why(3);

Zee zz = **new** Zee();

Ecks xx2 = yy;

Ecks xx3 = zz;

Note: Try to answer the following questions without compiling or running the code.

a. For each of the following statements, indicate if it is *legal* (i.e. will compile as written) or *illegal* (will generate a compiler error):

i. Ecks x1 = yy;

ii. Why y1 = xx;

iii. Why y2 = zz;

iv. Zee z2 = yy;

b. For each of the following statements, indicate if it is *legal* (i.e. will compile as written) or will generate a *compile time error* or a *runtime error*.

i. Why y2 = xx3;

ii. Why y3 = (Why)xx3;

iii. Zee z1 = (Zee)xx3;

iv. Zee z2 = (Zee)xx2;

v. Why y4 = (Zee)xx3;

c. Give the output that would be produced by the following lines of code:

System.*out*.println("xx.bar() = " + xx.bar());

System.*out*.println("yy.bar() = " + yy.bar());

d. Give the output that would be produced by the following lines of code:

System.*out*.println("xx.foo() = " + xx.foo());

System.*out*.println("yy.foo() = " + yy.foo());

e. Give the output that would be produced by the following lines of code:

System.*out*.println("zz.bar() = " + zz.bar());

System.*out*.println("zz.foo() = " + zz.foo());

3. Imagine that a cell carrier wants to begin charging for sending multimedia messages. The cost of sending a multimedia message is 4 cents for each character of text (e.g. 0.04 times the length of the text) plus 1.5 cents for each byte in the multimedia file (e.g. 0.015 times the size of the file). Give a method computeCost as it would appear in the MultimediaMessage class that computes and returns the cost of sending the message.

4. This question uses the TextMessageList and MultimediaMessage classes from the sample code on the Detailed Schedule and Resources page.

a. Consider the following snippet of code:

MultimediaMessage mm1 =

**new** MultimediaMessage(3517654321L, 7171234567L, "WDYT?");

mm1.attachFile("skiTrip.jpg", 2000);

System.*out*.println(mm1);

This code displays the following output:

(From 3517654321 to 7171234567): WDYT?

A MultimediaMessage clearly has more information contained in it than was displayed (e.g. the file name and file size). Explain as clearly and as fully as you can why only the above information was displayed.

b. Modify the MultimediaMessage class so that when the snippet of code from part a is run the output is:

(From 3517654321 to 7171234567): WDYT?

 File: skiTrip.jpg (2000)

Give only the code that you added to the MultimediaMessage class as your answer to this question. Hint: Including escaped characters in a String can affect the way it is printed. Including a “\n” results in a new line and a “\t” results in a tab.

c. Add a method named listMessagesFrom to the TextMessageList class that accepts a single parameter indicating a phone number and displays the basic information about each message that was received from that number. The basic information to be displayed about each object should be obtained by invoking its toString method.

5. Consider the following snippet of code:

ArrayList<TextMessage> tmList = **new** ArrayList<TextMessage>();

TextMessage tm1 =

**new** TextMessage(7171234567L, 3517654321L, "Yo Joe!");

TextMessage tm2 =

**new** TextMessage(7171234567L, 2159876543L, "Hi Kim!");

tmList.add(tm1);

tmList.add(tm2);

MultimediaMessage mm1 =

**new** MultimediaMessage(7171234567L, 3517654321L, "Yo Joe!");

mm1.attachFile("joe.jpg", 2000);

System.*out*.println("contains mm1: " + tmList.contains(mm1));

Using the implementations of TextMessage and MultimediaMessage given in class, the last line above will print out:

contains mm1: true

This is clearly not the desired behavior. This happens because the contains method passes each object in the list to the equals method of mm1 and mm1 inherited its equals method from the TextMessage class. Thus, when tm1 is passed to mm1’s equals method it returns true and the ArrayList concludes that mm1 is contained in the list. To obtain the desired behavior we must override the equals method in the MultimediaMessage class.

Give an implementation of equals for the MultimediaMessage class that overrides the one inherited from TextMessage and leads to the expected behavior. Keep in mind that the objects passed to the equals method can be any type of object that could be in tmList. Hints: 1. A MultimediaMessage object can only be equal to another MultimediaMessage object. 2. The instanceof operator can be used.