**COMP 132 - Homework # 7**

**Interfaces and Polymorphism**

1. Define an interface named Flies to be implemented by any object that can fly. Include at least 2 relevant methods in your interface. You do not need to provide Javadoc for this interface. Copy and paste the code for your interface as the solution to this problem.

2. Define a class Airplane that implements both the Flies interface and the MakesSound interface. (The MakesSound interface can be found on the class meeting resources page.) You do not need to provide Javadoc for this class. Copy and paste the code for your Airplane class as the solution to this problem.

3. Consider the two interfaces defined below:

|  |  |
| --- | --- |
| **public** **interface** SmellsBad { **public** **void** stink();} | **public** **interface** LooksBad { **public** **void** yuck();} |

Now imagine two classes:

 Class Foo implements the SmellsBad interface.

 Class Bar implements both the SmellsBad and the LooksBad interfaces.

An instance of each of these classes can be created as follows:

Foo f1 = **new** Foo();

Bar b1 = **new** Bar();

Note: You may not compile or run any of the given code when answering this question.

Hint: Drawing object diagrams will help with this problem!

A. Given the above objects, indicate which of the following statements are *legal* (i.e. will compile as written) and which are *illegal* (i.e. will generate compiler errors):

i. f1.stink();

ii. f1.yuck();

iii. b1.stink();

iv. b1.yuck();

B. Given the above objects, indicate which of the following assignments are *legal* (i.e. will compile as written) and which are *illegal* (i.e. will generate a compiler error):

i. SmellsBad sb1 = f1;

ii. LooksBad lb1 = f1;

iii. SmellsBad sb2 = b1;

iv. LooksBad lb2 = b1;

C. Assuming that the legal assignments in Part B have been executed, indicate which of the following statements are *legal* (i.e. will compile as written) and which are *illegal* (i.e. will generate compiler errors):

i. sb2.stink();

ii. sb2.yuck();

iii. lb2.stink();

iv. lb2.yuck();

D. Assuming that the legal assignments in Part B have been executed, indicate if each of the following type casts are *legal* (i.e. will compile as written), will generate a *compiler error* or will generate a *runtime error*:

i. Foo f2 = (Foo) b1;

ii. Foo f3 = (Foo) sb1;

iii. Bar b2 = (Bar) sb2;

iv. SmellsBad sb3 = (SmellsBad) lb2;

v. LooksBad lb3 = (LooksBad) sb2;

vi. LooksBad lb4 = (LooksBad) sb1;

E. Assuming that the legal assignments in Part B have been executed, indicate which of the following type casts are *necessary* and which are *unnecessary*:

 i. Bar b3 = (Bar) lb2;

 ii. LooksBad lb5 = (LooksBad) b1;

 iii. SmellsBad sb4 = (SmellsBad) sb2;

 iv. SmellsBad sb5 = (SmellsBad) lb2;

 v. Foo f4 = (Foo) f1;

F. Assuming that the legal assignments in Part B have been executed indicate the value (true or false) of each of the following Boolean expressions:

i. b1 **instanceof** SmellsBad

ii. sb1 **instanceof** Bar

iii. sb2 **instanceof** Bar

iv. lb2 **instanceof** SmellsBad;

v. sb1 **instanceof** LooksBad;

4. Consider the following interface definition:

**public** **interface** Mystery {

 **public** **void** abc();

 **public** **int** def();

}

The following two classes both implement the Mystery interface:

|  |  |
| --- | --- |
| **public class** Strange  **implements** Mystery { **private** **int** x;  **public** Strange() { x = 12; }  **public** **void** abc() { x = x - 3; }  **public** **int** def() {  **return** 2\*x;  }} | **public class** Unknown  **implements** Mystery { **private** **int** x;  **public** Unknown() { x = 5; }  **public** **void** abc() { x = x + 9; }  **public** **int** def() {  **return** x + 6; }} |

Now assume that the following method is available to be invoked:

**public** **static** **void** doIt(Mystery my) {

my.abc();

System.*out*.println(my.def());

}

Given all of the above, what output would the following lines of code generate?

Strange s1 = **new** Strange();

Unknown u1 = **new** Unknown();

*doIt*(s1);

*doIt*(u1);

Mystery m1 = s1;

*doIt*(m1);

m1 = u1;

*doIt*(m1);

5. Consider the following snippet of code that creates an ArrayList of objects that implement the MakesSound interface.

ArrayList<MakesSound> soundMakers = **new** ArrayList<MakesSound>();

soundMakers.add(**new** Duck("Mallard"));

*… additional statements omitted …*

soundMakers.add(**new** Car("Volvo", 3));

A. Give a snippet of code that displays the sound that is made by each object in the ArrayList.

B. Give a snippet of code that invokes the swim() method on every object in the ArrayList that implements the Swims interface.