Questions

For many of the questions below, I did not give you "rules" for answering them. I would like you to answer such questions by answering how you think it should be, and checking this is consistent with how it works.

1. Suppose you are defining a class A, and you want to define an equals( A a) method:

   (a) What should x.equals(x) return ?
   (b) What should be the relationship, if any, between x.equals(y) and y.equals(x) ?
   (c) if x.equals(y) and y.equals(z) both return true then what can we say, if anything, about x.equals(z) ?
   (d) if x is not null, then what if anything can we say about x.equals(null) ?

2. Suppose class B extends class A. What is the relationship, if any between ArrayList<A> and ArrayList<B> ?

3. What is the output when you run the Test class below ?

```java
class A {
    String s;

    A(){
        s = "default string";
    }
    public String toString(){
        return "toString() from A returns: " + s;
    }
}

class B extends A {
    public String toString(){
        return "toString() from B returns: " + s;
    }
}

class Test {
    public static void main(String[] args){
        B b = new B();
        A a = b;
        System.out.println( a.toString() );
        System.out.println( b.toString() );
    }
}
```
4. (a) What is the output of the code below, when you run class B?

*Hint:* This is a nasty question. To answer it correctly, notice the variables s in A and B have nothing to do with each other. In particular, the output would not change if you were to rename either (or both) the s variables within classes A or B.

(b) Would the compiler complain if the variable b in main were declared as type A?

(c) Would the compiler complain the variable b in main were declared as type I?

```java
public interface I {
    public String m();
}

public abstract class A implements I {
    private String s = "A: ";
    public String m(String s_arg){
        return s + s_arg;
    }
}

public class B extends A{

    private String s = "B: ";

    public String m() {
        return s;
    }

    public static void main(String[] args){
        B b = new B();
        System.out.println( b.m( "test" ) );
    }
}

Note that A is indeed an abstract class since it doesn’t implement the m() method from the interface.
```
5. Which of the instructions in the code below does the compiler allow or not allow?

```java
public interface I{
    public void m(I other);
}

public class A implements I {
    public void m(I a){
        A myA = (A) a;
    }
}

public class B implements I {
    public void m(I x) {
        B myB = (B) x;
        A myA = (A) x;
        A a = myB;
        myA = (A) myB;
    }
}
```
6. Which (if any) of the instructions (c)-(g) of Test generate compiler errors? [There are no instructions (a), (b) because I wanted to avoid confusing with variables a, b.]

What is the output of the program after removing any lines that cause compiler errors?

```java
public class A {
    public int n = 3;

    A() { // constructor
        System.out.println("A");
    }

    public void foo() { System.out.println(n); }
}

public class B extends A {
    public int n;

    B() {} // constructor
    B(int n){ // constructor
        System.out.println( "B" );
        this.n = n;
    }
    int foo(int n) {
        System.out.println(this.n);
        return n;
    }
}

public class Test {

    public static void main(String[] args) {
        int n = 2;
        A a = new A(); // (c)
        a.foo(); // (d)
        a.foo( 7 ); // (e)
        B b = new B( 11 ); // (f)
        n = b.foo(4 ); // (g)
    }
}
```
7. (a) Exactly one of the lines (1)–(4) in the Test class has a compiler error. Which one and why?

(b) Assuming you have commented out the error line identified in (a), what is the output when you run the Test class? See lines (1)–(4). Briefly explain why, using the terms “override” or “overload” when appropriate.

```java
public class Test {
    public static void main(String[] args) {
        Vehicle v = new Vehicle();
        v.driver = "Lady Gaga";
        Movable m = new Car();
        System.out.println( v.move("Eminem") ); // (1)
        System.out.println( v.move() ); // (2)
        System.out.println( m.move("Snoop Dogg") ); // (3)
        System.out.println( m.move() ); // (4)
    }
}

class Vehicle implements Movable{
    public String driver = "Rihanna";
    Vehicle(){
        Vehicle(String driver){ this.driver = driver; }
    }
    public String move() { return "vehicle driven by " + driver; }
    public String move(String driver) { return "vehicle driven by " + driver; }
}

class Car extends Vehicle implements Movable{
    public String move(){ return "car driven by " + driver; }
}
```

8. Suppose the classes below are all in the same package. See questions on next page.

```java
public class Sharer{
    int sum;
    String ID;
    Sharer other;

    public Sharer(String ID, int sum){ this.ID = ID; this.sum = sum; }

    // give half, keep half
    void share(int n){ other.sum += n/2; this.sum += n - n/2; }

    public String toString(){ return ID + " " + sum + " "; }
}

class Giver extends Sharer{
    public Giver(String ID, int sum) { super(ID, sum); }

    // give to other
    void share(int n) { other.sum += n; this.sum -= n; }
}

class Taker extends Sharer{
    public Taker(String ID, int sum) { super(ID, sum); }

    // take from other
    void share(int n) { other.sum -= n; this.sum += n; }
}

public class Test {

    public static void main(String[] args) {
        Sharer a = new Giver("Geoff", 10);
        Sharer b = new Taker("Tina", 7);
        Sharer c = new Taker("Ted", 15);

        a.other = b; b.other = c; c.other = a;
        a.share(2); b.share(4); c.share(7);

        System.out.println( a.toString() + b.toString() + c.toString());
    }
}
```
(a) What is the output when the Test class is run?

(b) Suppose we were re-define the visibility of the three fields of Sharer to be private so that the Giver and Taker no longer have access to these fields. Rewrite the methods of Sharer, Giver, and Taker to be consistent with this new definition. You will need to add getters and setters.
Solutions

1. (a) returns true  
   (b) they have the same truth value (both true or both false)  
   (c) x.equals(z) should return true  
   (d) it should return false

2. There is no relationship between them. In particular, \texttt{ArrayList<B>} does not extend \texttt{ArrayList<A>}.

3. The output is

   \texttt{toString from B: default string}  
   \texttt{toString from B: default string}

   Why? The variable \texttt{a} references a class \texttt{B} object, and so \texttt{a.toString()} calls the method \texttt{toString()} from class \texttt{B}.

4. The output is:

   (a) \texttt{A: test}  
   Why? The \texttt{m()} method from class \texttt{A} must be used here since the method \texttt{m} in \texttt{main} is passed a \texttt{String} argument and no such method signature exists in \texttt{B}. Thus, \texttt{B} must inherit this method from its superclass \texttt{A}.
   Without the hint, you might have been confused as to whether or not inheritance implies that the \texttt{m(String ..)} code from \texttt{A} also belongs to class \texttt{B}. With the hint, you should see that the code does \textit{not} belong to \texttt{B}. Thus, class \texttt{A}’s \texttt{m} method is used, and this refers to the string \texttt{s} from class \texttt{A}.

   (b) The object referenced by variable \texttt{b} invokes an \texttt{m} method with \texttt{String} argument. This method is found in \texttt{A} so there would be no problem in declaring \texttt{b} to be type \texttt{A}.

   (c) There is only one \texttt{m} method in interface \texttt{I} and this does not have a \texttt{String} argument. Thus, the compiler would give an error if you declared \texttt{b} to be of type \texttt{I}.
5. public interface I{
    public void m(I other);
}

public class A implements I{
    public void m(I a){
        A myA = (A) a; // Programmer expects type A argument
        // (or descendent of A).
        : // Compiler allows it, since A implements I.
    }
}

public class B implements I {
    public void m(I b) {
        B myB = (B) b; // Compiler allows it, since B implements I.
        A myA = (A) b; // Compiler allows it, since A implements I.
        A a = myB; // compiler error (cannot convert from type B to A)
        myA = (A) myB; // compiler error (cannot cast from B to A)
        :
    }
}

6. The compilation error occurs in (e). The problem is that the A class has no foo method with an argument, and a is declared to be of class A.

Output:
A from (c)
3 from (d)
A from (f) <--- easy to miss this one (every constructor makes an implicit call to its super class constructor)
B from (f)
11 from (g)
7. (a) The compiler error is in line (3). m is of declared type Movable, and the move() method of the Movable interface doesn’t have a String argument.

If you think that m is of type Car, then you are confusing the type of the variable m with the type of the object that m is referencing. The variable m is declared to be of type Movable and so the compiler only allows m to invoke methods that belong to the interface Movable. It doesn’t matter that m has been initialized to reference some object whose class (Car) implements Movable and who has a move(String) method.

I suggest you think of the instruction

Movable m = new Car();

as two separate instructions (and see comments for the instructions do):

Movable m; // Declare type of variable m.
m = new Car(); // Variable m temporarily references a new object.

As the program proceeds, m might later reference other Movable objects and there may be other reference variables the reference that Car object.

(b) (1) prints “vehicle driven by Eminem”, since “Eminem” is the parameter passed to the overloaded Vehicle.move(String)

(2) prints “vehicle driven by Lady Gaga”, since Vehicle.move() uses the Vehicle.driver field, which has value “Lady Gaga”.

(4) prints “car driven by Rihanna”, since Car objects inherit the Vehicle.driver field, which is initialized to “Rihanna”. Car.move() overrides Vehicle.move().

8. (a) Geoff 1 Tina 13 Ted 18

(b) // in the Sharer class

Sharer getOther(){ return this.other; }
void setOther(Sharer other){ this.other = other; }
int getSum(){ return this.sum; }
void setSum(int n){ this.sum = n; }

// in the Giver class

void share(int n) {
    this.getOther().setSum( this.getOther().getSum() + n );
    this.setSum( this.getSum() - n );
}

// in the Taker class

void share(int n) {
    this.getOther().setSum( this.getOther().getSum() - n);
    this.setSum( this.getSum() + n );
}