1. Consider the abstract class Politician and two concrete subclasses, Senator and President. All Politicians implement Corruptible:

```java
interface Corruptible {
    void acceptMoney(int amount); // bribe money
    void setQuote(String s); // quotes the Politician paid to say
    String getQuote(); // given to Reporters
}
```

All Politicians accept Visitors, including subclasses: Lobbyist and Reporter.

When a Lobbyist visits a Senator, the Senator accepts bribe money and is told what to say to subsequent Reporters. When a Lobbyist visits a President the two are caught on camera (see methods next page for details).

A Reporter visiting either a Senator or President just gets a quote which is subsequently printed.

For instance:

```java
Politician ted = new Senator("Ted");
Politician george = new President("George");
Lobbyist jack = new Lobbyist("Jack");
Reporter bob = new Reporter("Bob");

jack.setBribe(10000); // give lobbyist money to pass to politician
jack.setQuote("we must drill in Anwar"); // tell politicians what to say
ted.acceptVisitor(jack); // send the lobbyist to Senator Ted
ted.acceptVisitor(bob); // Reporter Bob visits, gets quote from Ted
bob.printQuote(); // Ted’s words then get printed, and
george.acceptVisitor(jack); // George caught photographed with lobbyist
```

results in the output:

An unnamed source said we must drill in Anwar
George photographed with Jack, Click!

The code is provided in part on the next two pages. You are to complete the code so it would compile and run correctly.
1a) [25%] Complete this code for the Politician hierarchy (fill in ALL missing code, some requiring that you complete a line of Java). Note that Politicians do nothing between visits (they have no other methods).

```java
abstract class Politician implements Corruptible {
    private String quote;
    private int funds;
    private String name;

    Politician(String name) { this.name = name; }

    public String getName() { return name; }

    public String bePhotographedWith(Visitor v) {
        System.err.println(name + " photographed with " + v.getName() + ",
        Click!");
    }

    public void acceptMoney(int amount) { funds += amount; }
    public void setQuote(String s) { quote = s; }
    public String getQuote() { return quote; }

    abstract void acceptVisitor(Visitor v);
}

class Senator extends Politician {
    Senator(String name) { super(name); }
    void acceptVisitor(Visitor v) { v.visit(this); }
}

class President extends Politician {
    President(String name) { super(name); }
    void acceptVisitor(Visitor v) { v.visit(this); }
}
```
1b) [25%] And regarding the Visitor hierarchy:

```java
abstract class Visitor {
    private String name;

    Visitor(String name) { this.name = name; }
    public String getName() { return name; }
    abstract void visit(Senator s);
    abstract void visit(President p);
}

class Lobbyist extends Visitor {
    private int bribe = 0;
    private String quote = "";

    Lobbyist(String name) { super(name); }
    void setBribe(int x) { bribe = x; }
    void setQuote(String s) { quote = s; }

    void visit(Senator s) {
        s.acceptMoney(bribe);
        s.setQuote(quote);
    }
    void visit(President p) { p.bePhotographedWith(this); }
}

class Reporter extends Visitor {
    private String quote = "";

    Reporter(String name) { super(name); }
    void printQuote() {
        System.err.println("An unnamed source said "+ quote);
    }

    void visit(Senator s) { quote = s.getQuote(); }
    void visit(President p) { quote = p.getQuote(); }
}
```
2) [10%] Draw a UML class diagram for the three classes of the Visitor hierarchy, being sure to indicate abstract classes and methods.

3) [10%] Draw a UML sequence diagram to show the flow of control for the following (from main). Assume jack and ted are valid instances of Lobbyist and Senator.

```java
jack.setBribe(10000);  // give lobbyist money to pass to politician
jack.setQuote("we must drill in Anwar"); // tell politicians what to say
ted.acceptVisitor(jack); // send the lobbyist to Senator Ted
```

main

jack:Lobbyist

TED:Senator
4) [30%] Given:

interface I {
    public method m1();
}

abstract class A implements I {
    public method m1() { System.out.println("m1 in A"); }
}

class B extends A {
    public method m1() { System.out.println("m1 in B"); }
    public method m2() { System.out.println("m2 in B"); }
}

B b = new B()
A a = b;
I i = b;

a) [5%] Write an expression to call method m2 using variable a.

((B)a).m2();

b) [5%] Write an expression to call method m2 using variable i.

((B)i).m2();

c) [5%] What information is lost when one assigns an instance of a B to a variable of type interface I, and then assigns it back to a different variable of type B? Example:

B b1 = new B()
I i = (I)b1;
B b2 = (B)i;

none

d) [10%] What are two basic uses of interfaces?

any two of:
1) to narrow the visibility of a class to just a specific subset of methods.
2) to provide a requirement for classes to implement certain methods (with signatures).
3) to provide a convenient means to define constants

e) [5%] What are two reasons for making a class abstract?

1) to provide the basic framework for subsequent subclasses to conform to
2) to allow subclasses to be cast to base class and have the abstract methods available as concrete implementations by the particular subclasses.