COMP132 – Computer Science II
Fall 2015

Homework #1
Java Review
Solutions

1. Write a program (i.e. a class with a `main` method) that takes the user's first and last names as command line arguments and displays a welcome message incorporating the user's first and last name. For example, if the program is run as:

```
java MyProgram Bob Smith
```

Then the output would be: `Hello Bob Smith.`

But if the program is run as:

```
java MyProgram Jane Jones
```

Then the output would be: `Hello Jane Jones.`

For this question you will need to create a new project in Eclipse, create a new class within that project and then create a `main` method in that class. You can find information about creating new projects and new classes in the How-To document linked to from the course home page. Copy and paste your `main` method as your solution to this question.

```
public static void main(String[] args) {
   String first = args[0];
   String last = args[1];

   System.out.println("Hello " + first + " " + last + ".");
}
```

2. What happens when your program from question 1 is run and the user only enters
their first name? For example:

```
java MyProgram Sue
```

Briefly explain why this happens.

If the user only enters their first name then there will be only one command line argument stored in `args[0]`. Further, `args` will be an array with fixed length of 1. Thus, there is no element at index 1 and when the program attempts to put `args[1]` into last, an `IndexOutOfBoundsException` will occur.

3. State which Java primitive data type you would use to hold each of the following values (You should do some research to find the size of these values!):

a. The population of the world.
b. The number of students enrolled at Dickinson College.
c. The number of atoms in the universe.
d. The national debt of the United States in dollars and cents (e.g. 123.45).
e. Daily high temperature in whole degrees Celsius in New York City.

a. `long`
b. `short`
c. `double`
Explanation: The number of atoms in the universe is a whole number but it is too large to be held by a `long`. A `double` can represent whole numbers that are large enough, as well as decimals. Thus, `double` is the right choice here.
d. `float`
e. `byte`

4. Consider the following variable declarations (the `Candidate` class can be found in the `comp132.examples.review` package of the `132SampleCode` project):

```
Candidate c1 = new Candidate("Bob", Candidate.REPUBLICAN);
Candidate c2 = new Candidate("Sam", Candidate.DEMOCRAT);
Candidate c3 = new Candidate("Jane", Candidate.INDEPENDENT);
```

a. Given the above variable declarations, and without executing any code on a computer, give the output that would be produced by the following lines of code.

```
c1.increaseVotes();
c2.increaseVotes(2);
c3.increaseVotes(3);

System.out.println("c1: " + c1.getName() + " : " + c1.getParty() +
    " : " + c1.getVotes());
System.out.println("c2: " + c2.getName() + " : " + c2.getParty() +
    " : " + c2.getVotes());
System.out.println("c3: " + c3.getName() + " : " + c3.getParty() +
    " : " + c3.getVotes());
```

```
c1: Bob : Republican : 1
c2: Sam : Democrat : 2
c3: Jane : Independent : 3
```

b. Assuming the variables given above are recreated (i.e. do not continue from part a), and without executing any code on a computer, give the output that would be produced by the following lines of code.

```
c1 = c3;

c1.increaseVotes();
c2.increaseVotes(2);
c3.increaseVotes(3);

System.out.println("c1: " + c1.getName() + " : " + c1.getParty() +
" : " + c1.getVotes());
System.out.println("c2: " + c2.getName() + " : " + c2.getParty() +
" : " + c2.getVotes());
System.out.println("c3: " + c3.getName() + " : " + c3.getParty() +
" : " + c3.getVotes());
```

```
c1: Jane : Independent : 4
c2: Sam : Democrat : 2
c3: Jane : Independent : 4
```

Explanation: After the statement `c1 = c3;` both `c1` and `c3` refer to the `Candidate` object for Jane. Thus, both `c1.increaseVotes()` and `c3.increaseVotes(3)` increase Jane's votes. Similarly, the line that prints `c1` and the line that prints `c3` both print out Jane's information. Drawing object diagrams can be very useful for understanding the operation of programs such as this.

c. Assuming the variables given above are recreated (i.e. do not continue from part a or part b), and without executing any code on a computer, give the output that would be produced by the following lines of code.

```
c2 = c3;
c3 = c1;

c1.increaseVotes();
c2.increaseVotes(2);
c3.increaseVotes(3);

c2.setParty(Candidate.LIBERTARIAN);

System.out.println("c1: " + c1.getName() + " : " + c1.getParty() +
   " : " + c1.getVotes());
System.out.println("c2: " + c2.getName() + " : " + c2.getParty() +
   " : " + c2.getVotes());
System.out.println("c3: " + c3.getName() + " : " + c3.getParty() +
   " : " + c3.getVotes());
```

```
c1: Bob : Republican : 4
c2: Jane : Libertarian : 2
c3: Bob : Republican : 4
```

Explanation: After statement `c2 = c3;` both `c2` and `c3` refer to the `Candidate` object for Jane. Then after the statement `c3 = c1;` both `c1` and `c3` refer to the `Candidate` object for Bob. Thus, both `c1.increaseVotes()` and `c3.increaseVotes(3)` increase Bob's votes and `c2.increaseVotes(2)` increases Jane's votes. Finally, the line that prints `c1` and the line that prints `c3` both print out Bob's information and the line that prints `c2` prints Jane's information. Again, drawing object diagrams can be very useful for understanding the operation of programs such as this.

5. Consider the following class definition, which contains both overloaded constructors and overloaded methods:

```
public class OverloadingHW {

    private int x;
    private int y;

    public OverloadingHW() {
        this(3);
    }

    public OverloadingHW(int x) {
        this(x, 2);
    }

    public OverloadingHW(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void foo() {
        x = 2;
        y = 4;
    }

    public void foo(int a) {
        x = x + a;
        y = 7;
    }

    public void foo(int a, int b) {
        this.foo(a+b);
    }
}
```

a. With the above class definition, and without executing any code on a computer, give the output that would be produced by the following lines of code.

```
OverloadingHW ohw = new OverloadingHW(3,5);
System.out.println("x=" + ohw.getX() + " y=" + ohw.getY());
ohw.foo();
System.out.println("x=" + ohw.getX() + " y=" + ohw.getY());
```

```
x=3 y=5
x=2 y=4
```

Explanation: The constructor for `OverloadingHW` is overloaded (i.e. there are multiple constructors.) Here the call to the constructor `Overloading(3,5)` provides 2 arguments of type `int`. Thus, the constructor with 2 parameters of type `int` is called:

```
public OverloadingHW(int x, int y) {
    this.x = x;
    this.y = y;
}
```

This constructor sets the values of the fields `x` and `y` to the values of the arguments provided (i.e. `3` and `5`). Thus, those are the values (3 and 5) that are displayed by the first `println` statement.

The method name `foo` is also overloaded in the `OverloadingHW` class. Here the call to `ohw.foo()` contains no arguments. Thus, the definition of `foo` with no parameters is called:

```
public void foo() {
    x = 2;
    y = 4;
}
```

This method changes the values of the fields `x` and `y` to `2` and `4` respectively, which are the values displayed by the second `println` statement.

b. With the above class definition, and without executing any code on a computer, give the output that would be produced by the following lines of code.

```
OverloadingHW ohw = new OverloadingHW(4);
System.out.println("x=" + ohw.getX() + " y=" + ohw.getY());
ohw.foo(2);
System.out.println("x=" + ohw.getX() + " y=" + ohw.getY());
```

```
x=4 y=2
x=6 y=7
```

Explanation: The constructor for `OverloadingHW` is overloaded (i.e. there are multiple constructors.) Here the call to the constructor `Overloading(4)` provides 1 argument of type `int`. Thus, the constructor with 1 parameters of type `int` is called:

```
    public OverloadingHW(int x) {
        this(x, 2);
    }
```

This constructor uses `this(x,2)` to invoke the constructor with 2 `int` parameters to do its work, passing `x` and `2` as the arguments. The values of the fields `x` and `y` are set by that constructor to values of the arguments provided (i.e. `4` and `2`), as happened in part a above. Thus, those are the values (`4` and `2`) that are displayed by the first `println` statement.

The method name `foo` is also overloaded in the `OverloadingHW` class. Here the call to `ohw.foo(2)` contains one argument of type `int`. Thus, the definition of `foo` with one parameter of type `int` is called:

```
    public void foo(int a) {
        x = x + a;
        y = 7;
    }
```

That method sets `x` to its current value plus the value of the parameter `a` (i.e. `2`) and sets `y` to `7`, which are the values displayed by the second `println` statement.

c. With the above class definition, and without executing any code on a computer, give the output that would be produced by the following lines of code.

```
OverloadingHW ohw = new OverloadingHW();
System.out.println("x=" + ohw.getX() + " y=" + ohw.getY());
ohw.foo(2,6);
System.out.println("x=" + ohw.getX() + " y=" + ohw.getY());
```

```
x=3 y=2
x=11 y=7
```

Explanation: The constructor for `OverloadingHW` is overloaded (i.e. there are multiple constructors.) Here the call to the constructor `Overloading()` provides no arguments. Thus, the constructor with no parameters is called:

```
    public OverloadingHW() {
        this(3);
    }
```

This constructor uses `this(3)` to invoke the constructor with 1 `int` parameter to do its work, passing 3 as the argument. The 1 argument constructor then uses `this` to invoke the 2 argument constructor as was described in part b above. Ultimately, the values of the fields `x` and `y` are set by that the two argument constructor to 3 and 2. Thus, those are the values (3 and 2) that are displayed by the first `println` statement.

The method name `foo` is also overloaded in the `OverloadingHW` class. Here the call to `ohw.foo(2,6)` contains two arguments of type `int`. Thus, the definition of `foo` with two parameters of type `int` is called:

```
    public void foo(int a, int b) {
        this.foo(a+b);
    }
```

That method uses the implicit parameter `this` to invoke the definition of `foo` with one parameter of type `int`, passing `a+b` (i.e. 8) as the argument. The definition of `foo` with one `int` parameter then sets the values of the fields `x` and `y` as discussed in part b above. The result is that `x` is `11` (`8+3`) and `y` is `7`, which are the values displayed by the second `println` statement.