Signature _____ **CSE 8B** **Name** _____

Quiz 3

cs8b ____ **Winter 2015** **Student ID** _____

This quiz is to be taken **by yourself** with closed books, closed notes, no calculators.

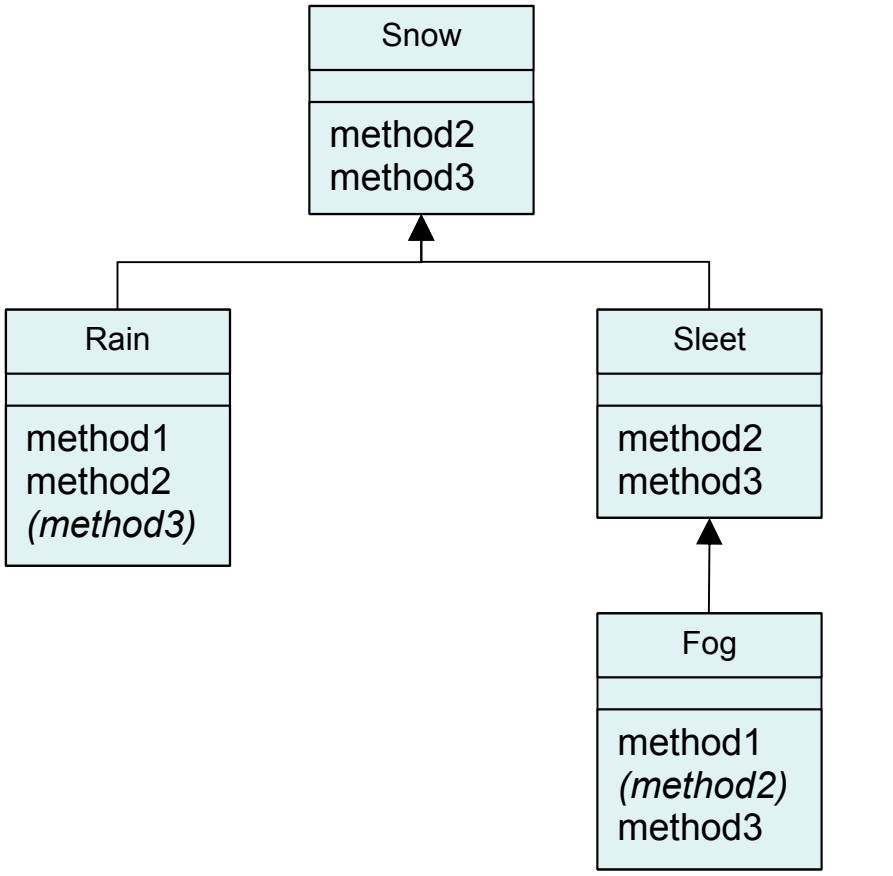
Given the following class definitions:

```
public class Snow {
  public void method2() {
    System.out.println("Snow 2");
  }
  public void method3() {
    System.out.println("Snow 3");
    method2();
  }
}

public class Rain extends Snow {
  public void method1() {
    System.out.println("Rain 1");
  }
  public void method2() {
    System.out.println("Rain 2");
    super.method2();
    method1();
  }
}

public class Sleet extends Snow {
  public void method2() {
    super.method2();
    System.out.println("Sleet 2");
  }
  public void method3() {
    System.out.println("Sleet 3");
  }
}

public class Fog extends Sleet {
  public void method1() {
    System.out.println("Fog 1");
  }
  public void method3() {
    method2();
    super.method3();
    System.out.println("Fog 3");
  }
}
```



What is the output given the following code:

```
Snow var1 = new Fog();
var1.method3();
```

Put your answer here:

snow 2
sleet 2
sleet 3
fog 3

class Snow (above) implicitly extends object.

When using the term static, think compile time while using the term dynamic, think run time.

Class Rain (above) has no constructor defined. Write the full code of what the compiler will automatically add:

Given the following class definitions for class Foo, class Fubar, and class FubarTest:

```
public class Foo {

  public Foo() {
    System.out.println( "Foo ctor #1" );
  }

  public Foo( int x, int y ) {
    this();
    System.out.println( "Foo ctor #2" );
  }

  public String toString() {
    System.out.println( "Foo.toString" );
    return "Foo";
  }
}
```

```
public class Fubar extends Foo {

  public Fubar() {
    this( 42, 420 );
    System.out.println( "Fubar ctor #1" );
  }

  public Fubar( int x, int y ) {
    System.out.println( "Fubar ctor #2" );
  }

  public String toString() {
    String s = super.toString() + " " + " +
               "Fubar.toString";

    System.out.println( s );
    return "Fubar";
  }
}
```

```
public class FubarTest {
  public static void main( String[] args ) {

    Foo ref = new Fubar();

    System.out.println( "-----" );

    System.out.println( ref.toString() );
  }
}
```

What is the output when we run FubarTest as in
java FubarTest

foo ctor #1
foo ctor #2
fubar ctor #1
— — —
foo.tostring
foo + fubar.tostring
fubar

The initGrid() method (below) should initialize a grid of ints as follows (remember indexing is [row][col]): if the row and col indices are the same, set grid[row][col] to 0. If the row and col indices are not the same, set grid[row][col] to the value (row + col).

```
public static void initGrid( int[][] grid ) {

  for ( int row = _______ ; row < ____________________________________; ++row ) {

    for ( int col = _______; col < ____________________________________; ++col ) {

      if ( _____________ == _____________ ) {

        grid[row][col] = _____________________________;

      } else {

        grid[row][col] = _____________________________;

      } // end if-else

    } // end inner for

  } // end outer for

} // end initGrid()
```

Examples:

If the grid passed in is 3 x 3 the grid should be initialized as:

0 1 2
1 0 3
2 3 0

If the grid passed in is 4 x 2 the grid should be initialized as:

0 1
1 0
2 3
3 4

If the grid passed in is 4 x 6 the grid should be initialized as:

0 1 2 3 4 5
1 0 3 4 5 6
2 3 0 5 6 7
3 4 5 0 7 8