Access Modifiers:

1. **Access Modifiers:** Private, default, protected, public: from most to least restrictive order. They are optional to use in methods and variables.
2. **Optional Specifiers:** static, abstract, final, synchronized, native, strictfp
3. Here are the rules for a method signature
   1. Return type should always come before the method name
   2. Optional specifiers can come before or after the access modifiers, i.e. **static public** or **public static** is the same.
4. Varargs: They look like String… or Int… and they are very cool to use, they have the following rules:
   * 1. You can only declare varargs as method parameters. They cannot be used as reference variables.
     2. While being passed as method parameter, varargs must be the last element to be passed
     3. Compiler will generate error if you have two overloaded methods with the same signature except one is varargs and the other is an array as both methods are same they accept arrays of a type.
     4. When calling a method with varargs parameter, you can pass in an array, you can list the elements of the array like (1,2,3,4) or you can even completely omit the parameters and just send ().
5. Out of the 4 access modifiers, protected is the trickiest one and it has some special rules we need to be careful about. The protected rules apply under two scenarios:
   1. A member is used without referring to a variable. In this case we are taking advantage of inheritance and protected access is allowed.

Ex:

Package pond.shore;

Public class Bird{

Protected String text =”floating”;  
 protected void floatInWater(){

System.out.println(text);

}

}

Package pond.goose;

Import pond.shore.Bird;

Public class Gosling extends Bird{

Public void swim(){

**floatInWater(); 🡪 Protected access to the method of parent Bird**

System.out.println**(text); 🡪 Protected access variable of parent Bird**

}

}

* 1. A member is used through a variable. In this case, the rules for the reference type of the variables are what matters. If it is a subclass, protected access is allowed. This works for references to the same class or a subclass.   
     Ex: Continuing the example above

Package pond.goose;

Import pond.shore.Bird;

Public class Gosling extends Bird{

Public void swim(){

Bird bird = new Bird();

bird.**floatInWater()**; -> **Compiler error, For reference Bird, because Bird is being used outside of pond.shore package and Bird does not extend Bird(Weird)**

bird.**text**;

// But the reference to Gosling works

Gosling gosling = new Gosling();

gosling.**floatInWater**(); -> **This is OK because the reference of Gosling is being used and Gosling extends Bird.**

gosling.**text**;

}

}